# **Донецкий ГОСУДАРСТВЕННЫЙ университет**

# **Физико-технический факультет**

# Кафедра компьютерных технологий

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ВЫПОЛНЕНИЮ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

ПО ДИСЦИПЛИНЕ

«ДИСКРЕТНАЯ МАТЕМАТИКА»

(Часть 2)

для студентов специальности 09.03.01

«Информатика и вычислительная техника»

очной и заочной формы обучения

**Донецк-2023**

# 1. ВВЕДЕНИЕ В ТЕОРИЮ НЕОРИЕНТИРОВАННЫХ ГРАФОВ

1736 г. считается годом возникновения теории графов: Л.Эйлер опубликовал результаты решения задачи о Кенигсбергских мостах и сформулировал критерий существования в графах специального маршрута (эйлерова цикла). Более чем 100 лет эти результаты оставались единственными и лишь в середине XIX в. инженер-электрик Г. Кирхгоф разработал теорию деревьев для исследования электрических цепей. Математик А. Кэли решил перечислительные задачи для определенных типов деревьев при исследовании строения углеводородов. К этому же периоду относится появление знаменитой проблемы 4 красок.

За последние десятилетия теория графов превратилась в один из некоторых бурно развивающихся разделов математики. В теоретико-графовых терминах формулируется большое число задач, связанных с дискретными объектами.

Термин “граф” был введен в 1936 г. венгерским математиком Д. Кенигом.

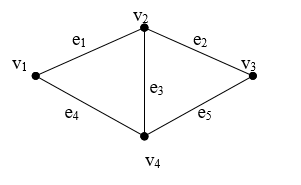
## 1.1. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Пусть имеется некоторое множество V≠∅ и пусть V(2) – неупорядоченное множество всех его двухэлементных подмножеств (V(2) ={(u,v): u,v∈V, неупорядоченная пара}). Тогда **неориентированным графом** G называется пара множеств (V,E), где . Обозначается: G=(V,E).

Множество V называется **множеством вершин графа**, а множество Е – **множеством ребер графа**.

Число |V| вершин графа G называется его **порядком**. Если , а , то граф G=(V, E) называют **p-графом,** или **(p,q)-графом**,или G p, q.

*Пример*: 4-граф, или (4,5)-граф, или G 4, 5.



Две **вершины** графа называются **смежными**, если они соединены ребром.

Два **ребра** графа называются **смежными**, если они имеют общую вершину.

Обозначим ребро графа: e=(u, v), где u и v – концевые вершины ребра. Ребро e **инцидентно** вершине v, если вершина v является одной из концевых вершин ребра e.

Заметим, что **смежность** есть отношение между однородными элементами графа, тогда как **инцидентность** является отношением между разнородными элементами.

Если множество V конечно, то граф называют **конечным**. Далее будем говорить только о конечных графах.

## 1.2 СПОСОБЫ ЗАДАНИЯ ГРАФОВ

Существуют следующие основные способы задания графов.

1. **Перечисление** множеств V (вершин) и E (ребер), задающих граф.

G=(V, E).

1. **Графический**: множество вершин V – это множество точек плоскости, множество ребер Е – множество отрезков прямой(кривой) на плоскости.
2. **Матричные** способы описания.

Пусть G=(V,E), , а , тогда:

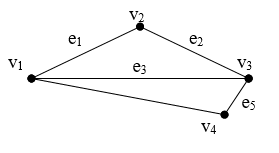
1. **матрица** **смежности** – квадратная матрица , , где



1. **матрица инцидентности –** прямоугольная матрица, , , где



*Пример*: задан граф G=(V, E),где V={v1,v2, v3, v4}, E**={**v1v2, v2v3, v1v3, v1v4, v3v4**}=**{e1, e2, e3, e4, e5}.



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A – матрица смежности: | | | | |  | B – матрица инцидентности: | | | | | |
| А | v1 | v2 | v3 | v4 |  | B | e1 | e2 | e3 | e4 | e5 |
| v1 | 0 | 1 | 1 | 1 |  | v1 | 1 | 0 | 1 | 1 | 0 |
| v2 | 1 | 0 | 1 | 0 |  | v2 | 1 | 1 | 0 | 0 | 0 |
| v3 | 1 | 1 | 0 | 1 |  | v3 | 0 | 1 | 1 | 0 | 1 |
| v4 | 1 | 0 | 1 | 0 |  | v4 | 0 | 0 | 0 | 1 | 1 |

## 1.3. СТЕПЕНИ ВЕРШИН ГРАФА

**Окружением вершины** v называется множество всех вершин графа G, смежных с ней; обозначается: N(v).

**Степенью** или **валентностью** вершины v (обозначается deg(v)) неориентированного графа G называется число ребер, инцидентных данной вершине (число вершин в ее окружении).



Число ΔG = max deg(v), v∈V называется **максимальной степенью** графа G, а число δ(G) = min deg(v), v∈V – **минимальной степенью** графа G.

*Пример*: задан граф G

|  |  |
| --- | --- |
| G |  |
|  | deg(v1)= deg(v3)=3; deg(v2)= deg(v4)=2.  ΔG =3, δ(G) = 2. |

Вершина v графа G называется **изолированной**, если ее степень равна нулю (deg(v)=0).

Вершина v графа G называется **висячей** или **концевой,** если степень этой вершины равна единице (deg (v)=1).

Вершина v графа G(p,q) называется **доминирующей**, если ее степень равна p-1 (deg (v)=p-1).

*Пример*: задан граф G

|  |  |
| --- | --- |
| G |  |
|  | доминирующей вершины нет;  висячая – v3;  изолированная – v5. |

**Лемма о рукопожатии**. Сумма степеней всех вершин графа четна и равна удвоенному числу ребер:

.

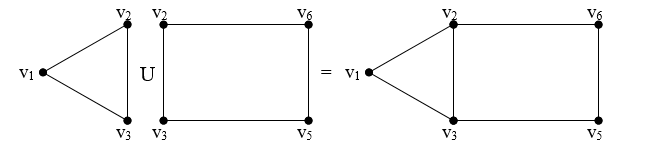
**Следствие.** В любом графе число вершин с нечетной степенью четно.

## 1.4. ОПЕРАЦИИ НАД ГРАФАМИ

Рассмотрим графы  и .

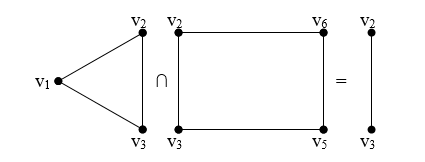
**Объединением двух графов**  и  называется граф G, множество вершин которого  и множество ребер .

*Пример*:



**Пересечение двух графов**  и  – это граф G, у которого множество вершин  и множество ребер .

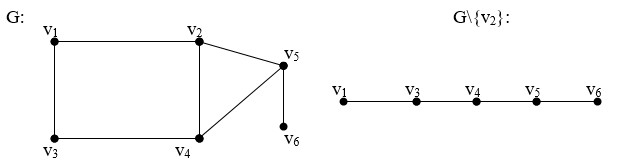
*Пример*:



**Непересекающимися** графами называются графы  и  такие что ∅ и  ∅ (нет общих ребер и нет общих вершин).

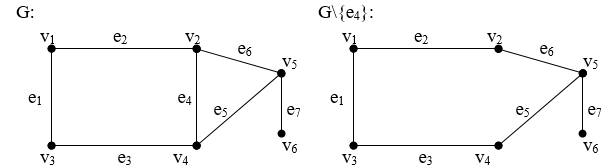
Пусть задан граф G=(V,E) и , тогда **удаление вершины** – это унарная операция над графом G, заключающаяся в построении порожденного подграфа G\{v} графа G на множестве вершин V\{v} (вершина удаляется из графа вместе с инцидентными ей ребрами).

*Пример*: v=v2



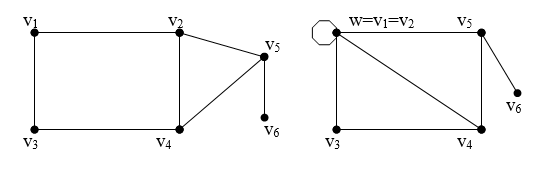
Пусть , тогда **удаление ребра** – это унарная операция над графом G, заключающаяся в построении порожденного подграфа G\{e} графа G с тем же множеством вершин и множеством ребер .

*Пример*: e=e4



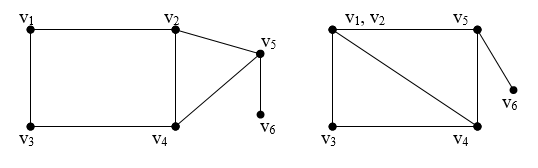
Говорят, что **пара вершин** vi и vj в графе G **замыкается (отождествляется)**, если они заменяются такой новой вершиной, что все ребра графа G, инцидентные вершинам vi и vj, становятся инцидентными этой новой вершине.

*Пример*: замыкание v1,v2



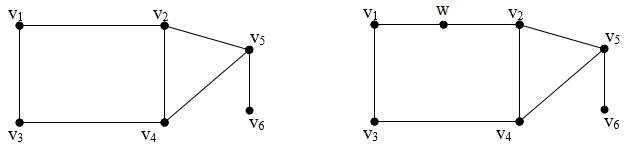
**Стягивание** заключается в удалении ребра e и отождествлении его концевых вершин.

*Пример*: стягивание v1, v2



**Подразбиением ребра** v1, v2 графа называется операция удаления ребра с добавлением новой вершины w и двух ребер v1, w и w, v2. Это означает, что добавляется новая вершина w на ребре v1, v2, которое, таким образом, разбивается на два ребра.

*Пример:* подразбиение ребра v1, v2



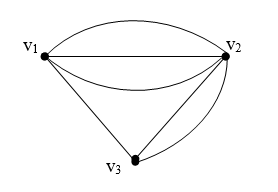
## 1.5. СПЕЦИАЛЬНЫЕ ГРАФЫ

В общем случае во множестве Е допускается более чем одно ребро с одинаковыми концевыми вершинами. Такие ребра называются **параллельными,** или **кратными**.

Ребро, соединяющее вершину v саму с собой, называется **петлей**.

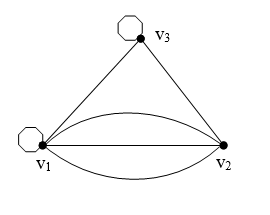
Граф, содержащий кратные ребра, но не содержащий петель, называется **мультиграфом**.

*Пример*:



Граф, содержащий и петли, и кратные ребра, называется **псевдографом**.

*Пример*:



Граф, не содержащий ни одного ребра, называется **пустым** **графом** и обозначается Оn, где n – количество вершин графа.

*Пример*: О3

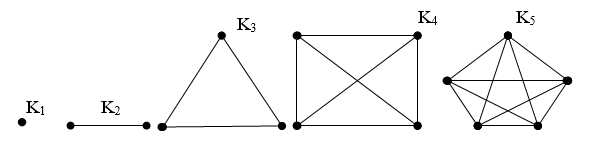


Граф, не содержащий ни одного ребра и ни одной вершины, называется  
**0-графом** (нуль-графом).

**Тривиальный граф** – это граф (1,0).

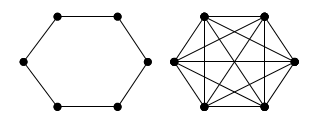
Граф, G называется **полным**, если у него все вершины смежные между собой. Каждая вершина полного графа является доминирующей. Обозначается: Kp, где p – количество вершин.

*Пример*:



**Однородным,** или **регулярным,** называется граф, у которого степени всех вершин равны. Все полные графы являются однородными.

*Пример*:



## 1.6. ПОДГРАФЫ

Неориентированный граф G=(V,E) называется **помеченным,** или **перенумерованным**, если каждой вершине графа поставлена в соответствие уникальная метка (число, символ); в противном случае граф называется **абстрактным**.

*Пример*:

|  |  |
| --- | --- |
| помеченный граф | абстрактный граф |
|  |  |

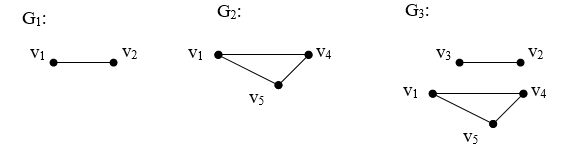
Граф  называется **дополнением** графа G, если множества вершин графов  и G совпадают, т. е. , а множество ребер . Следовательно, любые две вершины, смежные в графе G, не смежны в его дополнении .

*Пример*:

|  |  |
| --- | --- |
| G: | : |
|  |  |

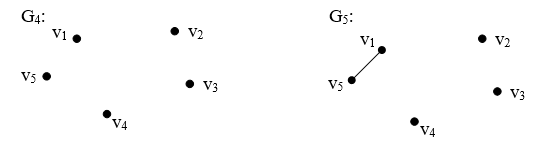
**Подграфом** графа G называется такой граф , у которого  

*Пример:* подграфы графа G



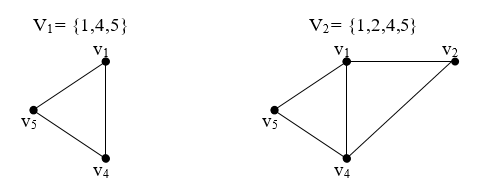
**Остовным подграфом** графа G называется такой подграф, у которого множество вершин равно множеству вершин исходного графа, а множество ребер может составлять часть множества ребер исходного графа ().

*Пример*: остовные подграфы графа G



**Порожденным подграфам** графа G (подграфом, порожденным множеством вершин ) называется подграф  такой что содержит все ребра, инцидентные вершинам из множества ( т.е. те вершины, которые соединены в исходном графе ребром, будут соединены ребром и в порожденном подграфе).

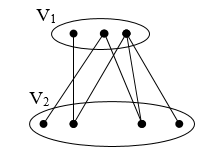
*Пример*: подграфы графа G, порожденные вершинами



## 1.7. ДВУДОЛЬНЫЕ ГРАФЫ

Граф, G=(V,E) называется **двудольным,** или **биграфом**, если множество его вершин V можно **разбить** на два подмножества V1 и V2, такие что любое ребро графа имеет одну концевую вершину во множестве V1 и другую во множестве V2. Таким образом, , V1∩V2=∅, V1∪V2=V.

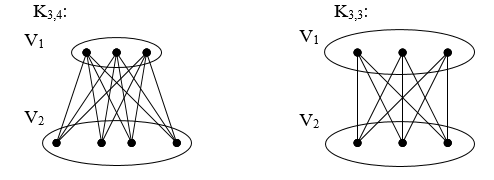
*Пример*: двудольный граф



Если в двудольном графе G с разбиением (V1,V2)  , то такой двудольный граф называется **полным двудольным графом**.

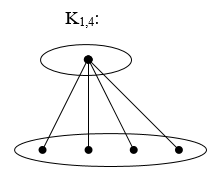
Обозначается: , где .

*Пример*:



Полный двудольный граф К1,n называется **звездой**.

*Пример*:



Граф G=(V,E) называется **k-дольным**, если множество вершин V можно разбить на k непересекающихся подмножеств таких что каждое ребро графа, имеющее концевую вершину , имеет другую концевую вершину , т.е.

(i≠j), i,j=, Vi=∅, Vi=V.

*Пример*:

## 1.8. ИЗОМОРФИЗМ И ГОМЕОМОРФИЗМ ГРАФОВ

Два графа G и Н называют **изоморфными (**), если существует взаимооднозначное соответствие (биекция) между множествами их вершин, сохраняющее отношение смежности.

Пусть G = (V, E), H = (V1, E1). Тогда , если



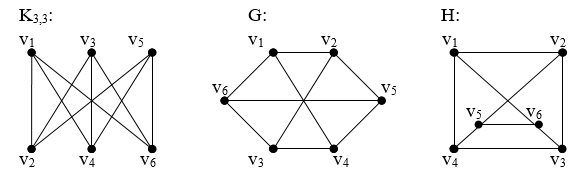
и наоборот.

*Пример*:

|  |  |
| --- | --- |
| G | H |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V | v1 | v2 | v3 | v4 |
| φ(v) | v2 | v1 | v3 | v4 |

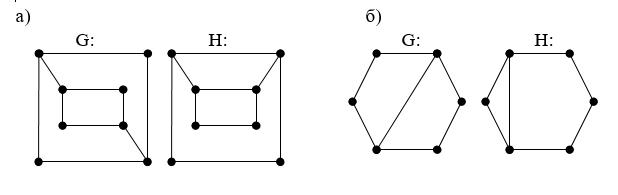
*Пример* изоморфных графов:



Условия изоморфизма графов:

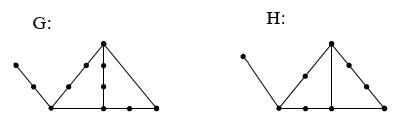
* совпадение количества вершин и ребер;
* соответствие распределения степеней вершин;
* биекция, сохраняющая отношение смежности между вершинами графов.

*Примеры* неизоморфных графов:



Графы G и Н называются **гомеоморфными**, если они могут быть получены друг из друга с помощью операций подразбиения ребер и стягивания вершин степени 2.

*Пример* гомеоморфных графов:



## 1.9. НЕЗАВИСИМОЕ, ДОМИНИРУЮЩЕЕ МНОЖЕСТВО ВЕРШИН, КЛИКА

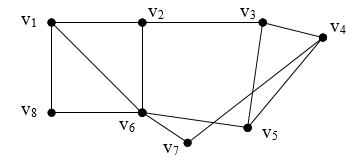
**Независимым множеством вершин** графа G, или **внутренним устойчивым** множеством, называется такое множество вершин S графа G, что любые две вершины этого множества не смежны.

Подграф, порожденный независимым множеством, является **пустым**.

Независимое множество называется **максимальным (Q)**, если оно не является собственным подмножеством другого независимого множества.

Независимое множество называется **наибольшим**, если оно наибольшее по мощности среди всех независимых множеств. Мощность наибольшего независимого множества называется **числом независимости**; обозначается: α(G).

*Пример*:



независимые множества:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S1={v1,v3} | S6={v2,v8} | S11={v1,v4} | S16={v7,v8} | S21={v3,v7,v8} |
| S2={v3,v6} | S7={v2,v7} | S12={v4,v6} | S17={v2,v4,v8} | S22={v5,v7,v8} |
| S3={v3,v8} | S8={v3,v7} | S13={v1,v5} | S18={v1,v3,v7} | S23={v2,v7,v5} |
| S4={v5,v8} | S9={v4,v8} | S14={v1,v7} | S19={v1,v5,v7} | S24={v2,v7,v8} |
| S5={v2,v4} | S10={v2,v5} | S15={v5,v7} | S20={v2,v8,v5} | S25={v2,v5,v7,v8} |

, наибольшее – , α(G) = 4.

**Доминирующим,** или **внешним устойчивым множеством** графа G, называется подмножество вершин исходного графа S такое что:

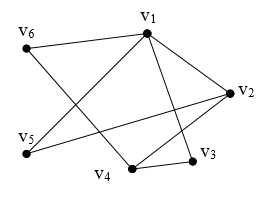
 такая что .

Доминирующее множество называется **минимальным**, если нет другого доминирующего множества, содержащегося в нем.

**Наименьшее** доминирующее множество – доминирующее множество с наименьшей мощностью.

Мощность наименьшего доминирующего множества называется **числом доминирования**; обозначается: β(G).

*Пример*:



S1={v3,v5,v6}, S2={v1,v4,v6}, S3={v1,v4}, S4={v2,v4},

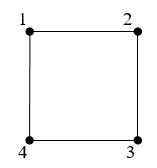
S3, S4 – минимальные, наименьшие множества, β(G) = 2.

**Ядром** графа называется множество вершин графа, которое является одновременно и независимым, и доминирующим множеством.

Не каждый граф содержит ядро, например, K3:

K3:

Граф K2,2 имеет два ядра: {1, 4} и {2, 3}.



**Кликой** графа G называется такое подмножество вершин исходного графа, что любые две его вершины смежны.

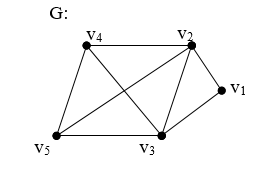
Подграф, порожденный кликой, является полным.

**Максимальной кликой** графа G называется клика, не являющаяся собственным подмножеством другой клики с большим числом вершин.

Клика графа G называется **наибольшей**, если число вершин в ней является максимальным среди всех других клик.

Мощность наибольшей клики называется **кликовым числом,** или **плотностью графа**; обозначается: φ(G).

*Пример*:



Клики:

S1={v1,v2}, S2={v1,v3}, S3={v2,v3}, S4={v2,v4}, S5={v2,v5}, S6={v3,v4},

S7={v3,v5}, S8={v4,v5}, S9={v1,v2,v3}, S10={v2,v3,v5}, S11={v2,v4,v5},

S12={v3,v4,v5},S13={v2,v3,v4}, S14={v2,v3,v4,v5},

Q={S9, S14}, наибольшая – S14, φ(G) = 4.

## 1.10. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определение неорграфа, подграфа, остовного и порожденного подграфов.
2. Дополнение к графу.
3. Операции над графами.
4. Изоморфизм графов.
5. Помеченные и абстрактные графы.
6. Клика. Максимальная и наибольшая клика. Кликовое число, или плотность графа.
7. Независимое множество. Максимальное и наибольшее независимое множество. Число независимости.
8. Полный граф. Число ребер в полном графе.
9. Доминирующее множество. Минимальное и наименьшее доминирующие множество. Число доминирования.
10. Ядро графа.

# 2. МАРШРУТЫ И СВЯЗНОСТЬ НЕОРИЕНТИРОВАННЫХ ГРАФОВ

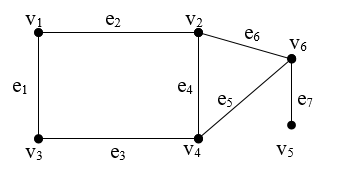
## 2.1. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Пусть дан граф G.

**Маршрутом** M=**(**v0,v1, …, vn) (или **(**v0 – vn)) в графе G=(V,E) называется чередующаяся конечная последовательность вершин и ребер такая, что:

* начинается и заканчивается вершиной;
* каждое ребро соединяет две вершины (предыдущую и последующую), т.е. (v0,e1,v1, …, en ,vn), где ei= (vi-1,vi), .

*Пример*:



М= **(**v1,e2,v2,e6,v6,e5,v4)= **(**v1,v2,v6,v4).

Вершины v0 и vn в маршруте М называются **конечными** или **терминальные**. Все остальные вершины маршрута называются **внутренними**.

Маршрут М называется **замкнутым**, если v0 = vn (т.е. начальная и конечная вершины совпадают), в противном случае маршрут М называется **открытым**.

Маршрут называется **цепью**, если все его ребра различны, и **простой цепью**, если его ребра и вершины различны.

Замкнутая цепь называется **циклом** (замкнутый маршрут, в котором не повторяется ни одно ребро).

**Простым циклом** называется замкнутая простая цепь (замкнутый маршрут, в котором не повторяются ни ребра, ни вершины), причем количество вершин n≥3.

|  |  |  |
| --- | --- | --- |
| исходный граф G | (v1,v2,v1) – замкнутый, но не цикл | цикл |
|  |  |  |

*Примеры маршрутов*:

M1=(v1,v2,v4,v3,v1,v2,v6,v5) – маршрут;

M2=(v1,v3,v4,v2,v6,v4, v3,v1) – замкнутый маршрут, не цикл;

M3=(v1,v3,v4,v2,v1) – простой цикл;

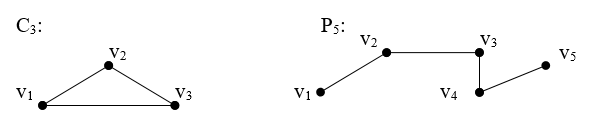
M4=(v5,v6,v4,v2, v6) – цепь;

M5=(v1,v3,v4,v2 ) – простая цепь.

Граф, состоящий из одного простого цикла, обозначается Сn, где n – количество вершин в цикле.

Граф, состоящий из одной простой цепи, обозначается Pn.

*Пример*:



**Примечание:** любые две простые цепи и любые два простых цикла являются гомеоморфными.

## 2.2. СВЯЗНОСТЬ, КОМПОНЕНТЫ СВЯЗНОСТИ

Граф G называется **связным**, если любая пара его вершин соединена маршрутом.

Две вершины vi и vj называются **связанными**, если в графе G существует маршрут между этими двумя вершинами. Вершина считается связанной сама с собой.

Максимальный связный подграф графа G называется **компонентой связности** графа G, или **компонентой**.

Связный граф состоит из одной компоненты связности.

*Пример*:

|  |  |  |
| --- | --- | --- |
| граф G1 связный | граф G2 не связный, имеет 4 компоненты | граф G3 не связный, имеет 2 компоненты |
|  |  |  |

**Утверждение 1.** Любой (u-v) – маршрут в неориентированном графе G содержит (u-v) – простую цепь.

**Утверждение 2.** Любой цикл неориентированного графа содержит в себе простой цикл.

В силу утверждений все определения связности можно давать в терминах простой цепи, например: две вершины называются **связанными**, если они соединены простой цепью.

**Теорема о связности**. Любой неориентированный граф является объединением своих компонент связности.

**Теорема о связности дополнения.** Любой неориентированный граф или его дополнение связны.

## 2.3. МЕТРИЧЕСКИЕ ХАРАКТЕРИСТИКИ ГРАФА

**Длиной маршрута (**v0,v1, …, vn) называется число ребер, содержащихся в маршруте, причем каждое ребро учитывается столько раз, сколько оно встречается в маршруте.

*Пример:*

Длина маршрута M=( v0,v1, …, vn) равна n.

Любому ребру приписывается единичная длина, если граф не является взвешенным.

**Взвешенным (нагруженным)** называется граф, у которого каждому ребру поставлено в соответствии некоторое число, называемое **весом** или длиной ребра.

Достаточно задать матрицу весов , где



**Обхватом графа** Gназывается длина наименьшего простого цикла графа G, если таковой имеется; обозначается: g(G).

**Окружением графа** Gназывается длина наибольшего простого цикла графа G, если таковой имеется; обозначается: C(G).

|  |  |  |
| --- | --- | --- |
| g(G) = C(G) =3 | g(G) = C(G) =0 | g(G) = 3; C(G) =6 |
|  |  |  |

**Расстоянием между двумя вершинами** u и vграфа G называется длина кратчайшей простой цепи, соединяющей эти вершины; обозначается: d(u,v).

Если вершины u и vграфа G не связаны, расстояние считается равным бесконечности: d(u,v)=∞.

В связном графе расстояние является метрикой, т.е. удовлетворяет следующим аксиомам.

∀ u,v,w ∈V:

1. d(u,v)≥0 (d(u,v)=0 <=> u=v);
2. d(u,v)=d(v,u);
3. d(u,v)+d(v,w)= d(u,w).

Кратчайшая простая цепь, соединяющая вершины u и v, называется **геодезической** и обозначается σ(u,v).

Волновой алгоритм нахождения кратчайших маршрутов.

Пусть задан неориентированный граф G=(V,E). Необходимо найти расстояние от одной заданной вершины до другой, или от одной заданной ко всем остальным вершинам графа G.

Итерации алгоритма:

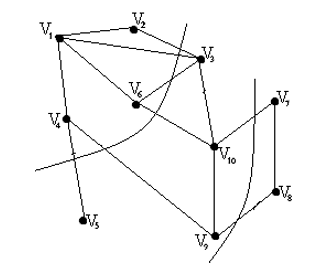
1. выбирается некоторая вершина графа G и определяются все “соседи” выбранной вершины («соседями» называются вершины, смежные с данной);
2. эти вершины помечаются;
3. определяются «соседи соседей» и также помечаются и т. д.

Алгоритм заканчивает работу, если:

* все вершины помечены (граф связен);
* построены все геодезические, связывающие данную вершину со всеми остальными;
* найдены расстояния от данной вершины ко всем остальным.

Если не все вершины в результате работы алгоритма помечены, это означает что граф G не связен.

*Пример:* задан граф



Итерации алгоритма сведены в таблицу:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| v1 | v2 | v4 | v6 | v3 | v5 | v9 | v10 | v7 | v8 |
| 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 |

**Эксцентриситетом** e(v) (или **отклоненностью)** вершины v графа G называется длина максимальной геодезической, исходящей из этой вершины:

e(v) = max d(v, u), v ∈ V, u ∈ V.

**Диаметром** графа G d(G) называется максимальный среди всех эксцентриситетов вершин графа:

d(G) = max e(u), u ∈ V.

**Периферией графа** G называется множество вершин графа G, для которых эксцентриситет равен диаметру, т.е. e(v) = d(G).

**Радиусом графа** G r(G) называется минимальный из эксцентриситетов вершин графа G:

r(G)=min e(u), u∈V.

**Центром графа** G называется множество вершин, для которых эксцентриситет равен радиусу графа. т.е. e(v) = r(G).

|  |  |  |
| --- | --- | --- |
| центр | центр | все вершины – центральные |
|  |  |  |

Определим для графа с количеством вершин p **матрицу расстояний** , где



Матрица расстояний несвязного графа может состоять из нескольких матриц (для каждой компоненты) или иметь блочный вид.

*Пример*: найти e(v) в заданном графе



Используем волновой алгоритм.

Первая компонента связности графа G:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **v** | **j** | **u** | **σ(u,v)** | **d(u,v)** |
| a | 0 | a | a | 0 |
|  | 1 | b | ab | 1 |
|  | 2 | d | abd | 2 |

Вторая компонента связности графа G:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **v** | **j** | **u** | **σ(u,v)** | **d(u,v)** |
| g | 0 | g | g | 0 |

Построим матрицу расстояний:



d(G)=3; r(G)=2; периферия: {a,c,e,f}; центр: {b,d} – для первой компоненты связности.

d(G)=1; r(G)=1; периферия: {g,h}; центр: {g,h} – для первой компоненты связности.

## 2.4. ВЕРШИННАЯ И РЕБЕРНАЯ СВЯЗНОСТЬ

**Вершинной связностью** графа G χ(G) называется наименьшее число вершин, удаление которых приводит к несвязному, или тривиальному, графу.

χ(Kn)=n-1; χ(Pn)=1; χ(Cn)=2.

**Реберной связностью** графа G λ(G) называется наименьшее число ребер, удаление которых приводит к несвязному графу.

λ(Kn)=n-1; λ(Pn)=1; λ(Cn)=2.

Для тривиального графа реберную связность полагают равной 0.

Вершина v графа G называется **точкой сочленения,** или **разделяющей вершиной**, если граф G\{v} имеет больше компонент связности, чем граф G.

Ребро е графа G называется **мостом,** если его удаление приводит к нарушению связности графа.

*Пример*:

|  |  |
| --- | --- |
| Граф G | χ(G), λ(G) |
|  | χ(G)=1, λ(G)=2. |
|  | χ(G)=1; λ(G)=1;  точки сочленения: v3, v4;  мост: (v3, v4). |
|  | χ(G)=1;  λ(G)=2;  точки сочленения: v2, v3. |

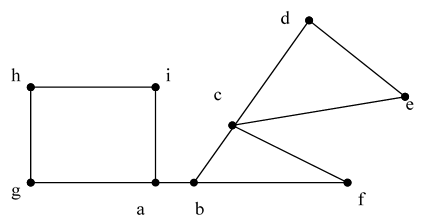
Связный непустой граф называется **неразделимым**, если в нем отсутствуют точки сочленения.

**Блок** – это максимальный неразделимый подграф исходного графа.

**Теорема Уитни.** Пусть σ( G) – минимальная степень вершин графа G, тогда справедливо следующее неравенство:

χ(G) ≤ λ(G) ≤ σ( G).

*Пример:* для заданного графа найти блоки



Граф не является неразделимым; блоки графа**:** {c,d,e},{c,b,f},{a,g,h,i},{a,b}.

## 2.5. КРАТЧАЙШИЕ МАРШРУТЫ В ГРАФАХ

Пусть задан граф G = (V, E), ребрам которого приписаны веса (стоимости), задаваемые матрицей . Задача состоит в нахождении кратчайшего пути от заданной начальной вершины s ∈ V до заданной конечной вершины t ∈ V при условии, что такой путь существует. Элементы матрицы весов могут быть положительными, отрицательными или нулевыми. Единственное ограничение состоит в том, чтобы граф не содержал циклов с суммарным отрицательным весом.

Задачи данного типа имеют следующие модификации:

* для заданной начальной вершины **s** найти кратчайшие пути от нее ко всем другим вершинам графа;
* найти кратчайшие пути между всеми парами вершин графа.

### 2.5.1. Алгоритм Дейкстры

Постановка задачи. Имеется произвольный взвешенный (*n*, *m*)-граф (в матрице весов нет отрицательных чисел), т.е.:



Требуется найти кратчайший маршрут от вершины s ко всем остальным вершинам графа.

В процессе выполнения этого алгоритма при переходе от вершины *i* к следующей вершине *j* используется специальная процедура метки вершин. Метки вершин в алгоритме Дейкстры могут быть двух типов: ***временные*** и ***постоянные***. Временная метка впоследствии может быть заменена на другую временную, если будет найден более короткий путь к данной вершине. Когда же станет очевидным, что не существует более короткого пути от исходной вершины *s* к данной, статус временной метки изменяется на постоянный. Таким образом, временная метка вершины дает верхнюю границу длины пути от *s* к данной вершине. Эти метки постепенно уменьшаются с помощью итерационной процедуры, и на каждом шаге итерации одна из временных меток становится постоянной. Последнее указывает на то, что метка уже не является верхней границей, а дает точную длину кратчайшего пути от *s* к рассматриваемой вершине.

Кратчайший маршрут между вершиной *s* и любой другой вершиной определяется, начиная с вершины назначения путем прохождения их в обратном направлении с помощью информации, представленной в постоянных метках.

**Алгоритм**.

Обозначим через *l*(*i*) метку вершины *i*, через *сij* – длину ребра (*i*, *j*).

Шаг 0. **Присвоение начальных значений.** Положить метку вершины *s* *l*(*s*)=0 и считать эту метку постоянной. Для всех вершин *i*, не смежных с *s*, положить их метки *l*(*i*)=∞. Присвоить меткам смежных с *s* вершин *l*(*i*)=*сis* и считать эти метки временными.

Шаг 1. **Поиск первой постоянной метки.** Найти вершину *p*, для которой ее временная метка минимальна:

*l*(*р*)=min[*l*(*i*)].

Сделать метку *l*(*р*) постоянной. Ее значение и будет являться кратчайшим расстоянием от *s* до *р*.

Шаг k. **Обновление меток.**

Для всех вершин *i*, имеющих временные метки и достижимых из *р*, метки обновляются в соответствии со следующим выражением:

*l*(*i*)=min[*l*(*i*), *l*(*p*)+*cpi*)].

Идея в следующем. На предыдущем шаге вычислено кратчайшее расстояние между вершинами *s* и *р* (метка *l*(*р*)), а также на предыдущих итерациях получена длина пути из *s* в *i* (*s* → *i*), не проходящего через *p*. Метка *l*(*i*) обновляется в том случае, если путь *s* → *i* целесообразно заменить на путь *s* → р → *i*.

Шаг k+1. **Превращение метки в постоянную.**

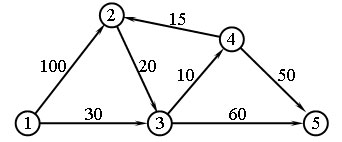
Среди всех вершин с временными метками найти такую, для которой

*l*(*i*\*)=min[*l*(*i*)].

Если таких меток несколько, то выбор произволен. Считать метку вершины *i*\* постоянной, положить *р*= *i*\*.

Если все вершины имеют постоянные метки, процесс вычислений заканчивается, иначе – переходим на шаг k.

*Пример*: имеется транспортная сеть, состоящая из пяти городов (расстояния между городами (в километрах) приведены возле соответствующих дуг сети). Необходимо найти кратчайшие расстояния от города 1 (вершина 1) до всех остальных четырех городов.



Шаг 0. Назначим вершине 1 постоянную метку *l*(1)=0. Из 1 можно достичь вершин 2 и 3. Вычисляем метки для этих вершин, в результате получаем следующую таблицу меток:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)=0 | **Постоянная** |
| 2 | *l*(2)=100 | Временная |
| 3 | *l*(3)=30 | <- Временная |
| 4 | *l*(4)=∞ | Временная |
| 5 | *l*(5)=∞ | Временная |

Шаг 1. Среди вершин 2 и 3 вершина 3 имеет наименьшее значение расстояния (*l*(3)=30). Поэтому статус метки этой вершины изменяется на «постоянная», *р*=3.

Шаг 2. Из вершины 3 (последней вершины с постоянной меткой) можно попасть в вершины 4 и 5. Получаем следующий список вершин и меток:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)=0 | **Постоянная** |
| 2 | *l*(2)=100 | Временная |
| 3 | *l*(3)=30 | **Постоянная** |
| 4 | *l*(4)=min[∞, 30+10)]=40 | <- Временная |
| 5 | *l*(5)=min[∞, 30+60)]=90 | Временная |

Временный статус метки вершины 4 заменяется на постоянный.

Шаг 3. Из вершины 4 можно достичь вершин 2 и 5. После обновления меток получим следующий их список:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)=0 | **Постоянная** |
| 2 | *l*(2)= min[100, 40+15]=55 | <- Временная |
| 3 | *l*(3)=30 | **Постоянная** |
| 4 | *l*(4)=40 | **Постоянная** |
| 5 | *l*(5)=min[90, 40+50)]=90 | Временная |

Временная метка *l*(2), полученная на втором шаге, изменена на 55. Это указывает на то, что найден более короткий путь к этой вершине (проходящий через вершину 4). На третьем шаге вершина 5 получает две метки с одинаковым значением расстояния *l*(5) = 90.

Шаг 4. Из вершины 2 можно перейти только в вершину 3, но она уже имеет постоянную метку, которую нельзя изменить. Поэтому на данном шаге получаем такой же список меток, как и на предыдущем шаге, но с единственным изменением: метка вершины 2 получает статус постоянной. С временной меткой остается только вершина 5, но так как из этой вершины нельзя попасть ни в какой другой, процесс вычислений заканчивается.

Определим кратчайший маршрут между вершиной 1 и 2 путем прохождения в обратном направлении с помощью информации, представленной в постоянных метках. Получаем такую обратную последовательность вершин:

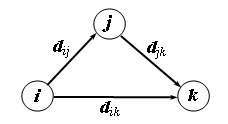
(2) → [55] → (4) → [40] → (3) → [30] → (1).

Таким образом, получаем путь 1→3→4→2 общей длиной 55 километров.

### 2.5.2. Алгоритм Флойда

Этот алгоритм более общий по сравнению с алгоритмом Дейкстры, так как он находит кратчайшие маршруты между любыми двумя вершинами графа. В этом алгоритме граф представлен в виде квадратной матрицы расстояний с *n* строками и *n* столбцами. Элемент (i, j) равен расстоянию *dij* от вершины *i* к вершине *j*, которое имеет конечное значение, если существует дуга (i, j), и равен бесконечности в противном случае.

Покажем сначала основную идею метода Флойда. Пусть есть три вершины *i*, *j* и *k* и заданы расстояния между ними.



Если выполняется неравенство *dij* + *djk* < *dik*, то целесообразно заменить маршрут *i* → *k* маршрутом *i* → *j* → *k*. Такая замена (далее ее будем условно называть ***треугольным оператором***) выполняется систематически в процессе выполнения алгоритма Флойда.

Алгоритм базируется на использовании последовательности из *n* преобразований (итераций) начальной матрицы весов. При этом на k-той итерации матрица представляет длины кратчайших маршрутов между каждой парой вершин. Кроме того, вводится матрица маршрутов S=[*sij*] – матрица, в которой элемент *sij* указывает вершину, непосредственно предшествующую вершине *j* в кратчайшем маршруте от *i* к *j*.

В конце алгоритма кратчайшие маршруты получаются непосредственно из заключительной матрицы S*n*.

**Алгоритм**.

Шаг 0. **Присвоение начальных значений.**

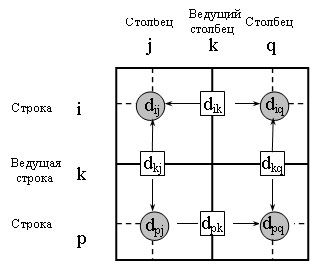
Определяем начальную матрицу расстояний D0 и матрицу маршрутов S0. Диагональные элементы обеих матриц помечаются знаком «0», показывающим, что эти элементы в вычислениях не участвуют. Матрице S0 присваиваются начальные значения *sij* = *i* для всех *i* и *j*.

Полагаем *k* = 1.

**Основной шаг k.** Задаем строку *k* и столбец *k* как ведущую строку и ведущий столбец. Рассматриваем возможность применения треугольного оператора ко всем элементам *dij* матрицы D*k*-1. Если выполняется неравенство *dik* + *dkj* < *dii*, (*i* <> *k*, *j* <> *k*, *i* <> *j*), тогда выполняем следующие действия:

* создаем матрицу D*k* путем замены в матрице D*k*-1 элемента *dij* на сумму *dik* + *dkj*,
* создаем матрицу S*k* путем замены в матрице S*k*-1 элемента *sij* на *k*. Полагаем *k* = *k* + 1 и повторяем шаг *k*.

Поясним действия, выполняемые на *k*-м шаге алгоритма, представив матрицу D*k*-1 так, как она показана на рисунке:

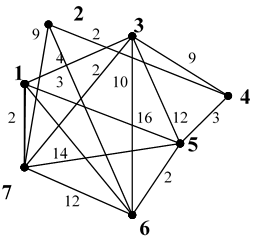


На этом рисунке строка *k* и столбец *k* являются ведущими. Строка *i* – любая строка с номером от 1 до *k*-1, а строка *p* – произвольная строка с номером от *k*+1 до *n*. Аналогично столбец *j* представляет любой столбец с номером от 1 до *k*-1, столбец *q* – произвольный столбец с номером от *k*+1 до *n*. Треугольный оператор выполняется следующим образом. Если сумма элементов ведущих строки и столбца (показанных в квадратах) меньше элементов, находящихся в пересечении столбца и строки (показанных в кружках), соответствующих рассматриваемым ведущим элементам, то расстояние (элемент в кружке) заменяется на сумму расстояний, представленных ведущими элементами.

После реализации *n* шагов алгоритма определение по матрицам D*n* и S*n* кратчайшего маршрута между узлами *i* и *j* выполняется по следующим правилам.

1. Расстояние между узлами *i* и *j* равно элементу *dij* в матрице D*n*.
2. Промежуточные узлы пути от узла *i* к узлу *j* определяем по матрице S*n*.
3. Пусть *sij* = *k*, тогда имеем путь *i* → *k* → *j*. Если далее *sik* = *k* и *skj* = *j*, тогда считаем, что весь маршрут определен, так как найдены все промежуточные узлы. В противном случае повторяем описанную процедуру для маршрутов от узла *i* к узлу *k* и от узла *k* к узлу *j*.

*Пример*: для заданного графа G построить кратчайшие маршруты алгоритмом Флойда.



Шаг 0. Построение начальной матрицы расстояний D0 непосредственно по заданному графу и матрицы маршрутов S0.

Полагаем *k* = 1.

Шаг 1. В матрице D0 выделены ведущие строка и столбец (*k* = 1).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | ∞ | 4 | ∞ | 16 | 14 | 2 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | 2 | ∞ | 3 | 9 |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 4 | ∞ | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | ∞ | 2 | 9 | 0 | 3 | ∞ | ∞ |  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | 16 | ∞ | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | ∞ | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | ∞ | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

В матрице D0 нет элементов, значения которых можно улучшить с помощью треугольного оператора.

Полагаем *k* = 2.

Шаг 2. В матрице D1 выделены ведущие строка и столбец (*k* = 2). Треугольный оператор применяется к элементам матрицы D1 и S1. Двойной рамкой представлены элементы *d*46 и *d*47, а также симметричные им *d*64, и *d*74, единственные среди элементов матрицы D1, значения которых можно улучшить с помощью треугольного оператора.

Матрицы D1 и S1 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | ∞ | 4 | ∞ | 16 | 14 | 2 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | 2 | ∞ | 3 | 9 |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 4 | ∞ | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | ∞ | 2 | 9 | 0 | 3 | ∞ | ∞ |  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | 16 | ∞ | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | ∞ | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | ∞ | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

Таким образом, чтобы на основе матриц D1 и S1 получить матрицы D2 и S2, выполняем следующие действия.

Заменяем *d*46 на *d*42 + *d*26 = 2 + 3 = 5 и устанавливаем *s*46 = 2.

Заменяем *d*47 на *d*42 + *d*27 = 2 + 9 = 11 и устанавливаем *s*47 = 2.

Заменяем *d*64 на *d*62 + *d*24 = 3 + 2 = 5 и устанавливаем *s*64 = 2.

Заменяем *d*74 на *d*72 + *d*24 = 9 + 2 = 11 и устанавливаем *s*74 = 2.

Полагаем *k* = 3.

Шаг 3. Выделяем в матрице D2 ведущую строку и столбец (*k* = 3). Двойной рамкой выделены элементы *d*14 и *d*41, к которым применяется треугольный оператор, жирным – элементы, значения которых изменены на предыдущем шаге. Матрицы D2 и S2 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | ∞ | 4 | ∞ | 16 | 14 | 2 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | 2 | ∞ | 3 | 9 |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 4 | ∞ | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | ∞ | 2 | 9 | 0 | 3 | **5** | **11** |  | 4 | 4 | 4 | 4 | 4 | 4 | **2** | **2** |
| 5 | 16 | ∞ | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | **5** | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | **2** | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | **11** | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | **2** | 7 | 7 | 7 |

Для получения матриц D3 и S3 выполняем следующие действия.

Заменяем *d*14 на *d*13 + *d*34 = 4 + 9 = 13 и устанавливаем *s*14 = 3.

Заменяем *d*41 на *d*43 + *d*31 = 9 + 4 = 13 и устанавливаем *s*41 = 3.

Полагаем *k* = 4.

Шаг 4. Выделяем в матрице D3 ведущую строку и столбец (*k* = 4). Элементами, значения которых может улучшить треугольный оператор, являются *d*12, *d*23, *d*25 и симметричные к ним (выделены двойной рамкой).

Матрицы D1 и S1 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | ∞ | 4 | **13** | 16 | 14 | 2 |  | 1 | 1 | 1 | 1 | **3** | 1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | 2 | ∞ | 3 | 9 |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 4 | ∞ | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | **13** | 2 | 9 | 0 | 3 | 5 | 11 |  | 4 | **3** | 4 | 4 | 4 | 4 | 2 | 2 |
| 5 | 16 | ∞ | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | 5 | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | 2 | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | 11 | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | 2 | 7 | 7 | 7 |

Получаем матрицы D4 и S4 с помощью следующих действий.

Заменяем *d*12 на *d*14 + *d*42 = 13 + 2 = 15 и устанавливаем *s*46 = 4.

Заменяем *d*23 на *d*24 + *d*43 = 2 + 9 = 11 и устанавливаем *s*23 = 4.

Заменяем *d*25 на *d*24 + *d*45 = 2 + 3 = 5 и устанавливаем *s*25 = 4.

Заменяем *d*21 на *d*24 + *d*12 = 2 + 13 = 15 и устанавливаем *s*46 = 4.

Заменяем *d*32 на *d*34 + *d*42 = 9 + 2 = 11 и устанавливаем *s*23 = 4.

Заменяем *d*52 на *d*54 + *d*42 = 3 + 2 = 5 и устанавливаем *s*25 = 4.

Полагаем *k* = 5.

Шаг 5. Выделяем в матрице D4 ведущую строку и столбец (*k* = 5).

Матрицы D4 и S4 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D4 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S4 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | **15** | 4 | 13 | 16 | 14 | 2 |  | 1 | 1 | **4** | 1 | 3 | 1 | 1 | 1 |
| 2 | **15** | 0 | **11** | 2 | **5** | 3 | 9 |  | 2 | **4** | 2 | **4** | 2 | **4** | 2 | 2 |
| 3 | 4 | **11** | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 4 | 3 | 3 | 3 | 3 | 3 |
| 4 | 13 | 2 | 9 | 0 | 3 | 5 | 11 |  | 4 | 3 | 4 | 4 | 4 | 4 | 2 | 2 |
| 5 | 16 | **5** | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | 5 | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | 2 | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | 11 | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | 2 | 7 | 7 | 7 |

В матрице D4 нет элементов, значения которых можно улучшить с помощью треугольного оператора.

Полагаем *k* = 6.

Шаг 6. Матрицы D5 и S5 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 15 | 4 | 13 | 16 | 14 | 2 |  | 1 | 1 | 4 | 1 | 3 | 1 | 1 | 1 |
| 2 | 15 | 0 | 11 | 2 | 5 | 3 | 9 |  | 2 | 4 | 2 | 4 | 2 | 4 | 2 | 2 |
| 3 | 4 | 11 | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 4 | 3 | 3 | 3 | 3 | 3 |
| 4 | 13 | 2 | 9 | 0 | 3 | 5 | 11 |  | 4 | 3 | 4 | 4 | 4 | 4 | 2 | 2 |
| 5 | 16 | 5 | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | 5 | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | 2 | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | 11 | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | 2 | 7 | 7 | 7 |

В матрице D5 нет элементов, значения которых можно улучшить с помощью треугольного оператора.

Полагаем *k* = 7.

Шаг 6. Выделяем в матрице D6 ведущую строку и столбец (*k* = 7). Матрицы D6 и S6 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D6 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S6 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 15 | 4 | 13 | 16 | 14 | 2 |  | 1 | 1 | 4 | 1 | 3 | 1 | 1 | 1 |
| 2 | 15 | 0 | 11 | 2 | 5 | 3 | 9 |  | 2 | 4 | 2 | 4 | 2 | 4 | 2 | 2 |
| 3 | 4 | 11 | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 4 | 3 | 3 | 3 | 3 | 3 |
| 4 | 13 | 2 | 9 | 0 | 3 | 5 | 11 |  | 4 | 3 | 4 | 4 | 4 | 4 | 2 | 2 |
| 5 | 16 | 5 | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | 5 | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | 2 | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | 11 | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | 2 | 7 | 7 | 7 |

В матрице D6 элементами, значения которых можно улучшить с помощью треугольного оператора, являются *d*12 и *d*21, получаем окончательную матрицу кратчайших расстояний D7 и кратчайших маршрутов S7, для чего выполняем следующие действия.

Заменяем *d*12 на *d*17 + *d*72 = 2 + 9 = 11 и устанавливаем *s*12 = 7.

Заменяем *d*21 на *d*27 + *d*72 = 9 + 2 = 17 и устанавливаем *s*21 = 7.

Матрица всех кратчайших маршрутов D7 и матрица кратчайших маршрутов S7 имеют вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | **11** | 4 | 13 | 16 | 14 | 2 |  | 1 | 1 | **7** | 1 | 3 | 1 | 1 | 1 |
| 2 | **11** | 0 | 11 | 2 | 5 | 3 | 9 |  | 2 | **7** | 2 | 4 | 2 | 4 | 2 | 2 |
| 3 | 4 | 11 | 0 | 9 | 12 | 10 | 2 |  | 3 | 3 | 4 | 3 | 3 | 3 | 3 | 3 |
| 4 | 13 | 2 | 9 | 0 | 3 | 5 | 11 |  | 4 | 3 | 4 | 4 | 4 | 4 | 2 | 2 |
| 5 | 16 | 5 | 12 | 3 | 0 | 2 | 14 |  | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 5 |
| 6 | 14 | 3 | 10 | 5 | 2 | 0 | 12 |  | 6 | 6 | 6 | 6 | 2 | 6 | 6 | 6 |
| 7 | 2 | 9 | 2 | 11 | 14 | 12 | 0 |  | 7 | 7 | 7 | 7 | 2 | 7 | 7 | 7 |

Распишем матрицу кратчайших маршрутов:

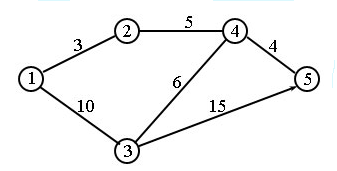
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 1,7,2 | 1,3 | 1,3,4 | 1,5 | 1,6 | 1,7 |
| 2 | 2,7,1 | 0 | 2,4,3 | 2,4 | 2,4,5 | 2,6 | 2,7 |
| 3 | 3,1 | 3,4,2 | 0 | 3,4 | 3,5 | 3,6 | 3,7 |
| 4 | 4,3,1 | 4,2 | 4,3 | 0 | 4,5 | 4,2,6 | 4,2,7 |
| 5 | 5,1 | 5,4,2 | 5,3 | 5,4 | 0 | 5,6 | 5,7 |
| 6 | 6,1 | 6,2 | 6,3 | 6,2,4 | 6,5 | 0 | 6,7 |
| 7 | 7,1 | 7,2 | 7,3 | 7,2,4 | 7,5 | 7,6 | 0 |

Кратчайшие маршруты от вершины 1:

|  |  |
| --- | --- |
|  | (1-2): (1,7,2)=11 |
|  | (1-3): (1,3)=4 |
|  | (1-4): (1,3,4)=(1,7,3,4)=(1,7,2,4)=13 |
|  | (1-5): (1,5)=16 |
|  | (1-6): (1,6)=14 |
|  | (1-7): (1,7)=2 |

Листинг программы, реализующей алгоритм Флойда, на С++ приведен в приложении А.

*Пример с частично ориентированным графом.* Найдем для графа, показанного на рисунке, кратчайшие пути между любыми двумя узлами. Расстояние между узлами этой сети проставлены на рисунке возле соответствующих ребер. Ребро (3, 5) ориентированно, поэтому не допускается движение от узла 5 к узлу 3. Все остальные ребра допускают движение в обе стороны.



Шаг 0. Начальные матрицы D0 и S0 строятся непосредственно по заданной схеме сети. Матрица D0 симметрична, за исключением пары элементов *d*35 и *d*53, где *d*53 равно бесконечности, поскольку невозможен переход от узла 5 к узлу 3:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0 | 1 | 2 | 3 | 4 | 5 |  | S0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 3 | 10 | ∞ | ∞ |  | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 0 | ∞ | 5 | ∞ |  | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 10 | ∞ | 0 | 6 | 15 |  | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | ∞ | 5 | 6 | 0 | 4 |  | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | ∞ | ∞ | ∞ | 4 | 0 |  | 5 | 5 | 5 | 5 | 5 | 5 |

Полагаем *k* = 1.

Шаг 1. В матрице D0 выделены ведущие строка и столбец (*k* = 1). Двойной рамкой представлены элементы *d*23 и *d*32, единственные среди элементов матрицы D0, значения которых можно улучшить с помощью треугольного оператора. Таким образом, чтобы на основе матриц D0 и S0 получить матрицы D1 и S1, выполняем следующие действия.

Заменяем *d*23 на *d*21 + *d*13 = 3 + 10 = 13 и устанавливаем *s*23 = 1.

Заменяем *d*32 на *d*31 + *d*12 = 10 + 3 = 13 и устанавливаем *s*32 = 1.

Матрицы D1 и S1 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1 | 1 | 2 | 3 | 4 | 5 |  | S1 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 3 | 10 | ∞ | ∞ |  | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 0 | **13** | 5 | ∞ |  | 2 | 2 | 2 | **1** | 2 | 2 |
| 3 | 10 | **13** | 0 | 6 | 15 |  | 3 | 3 | **1** | 3 | 3 | 3 |
| 4 | ∞ | 5 | 6 | 0 | 4 |  | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | ∞ | ∞ | ∞ | 4 | 0 |  | 5 | 5 | 5 | 5 | 5 | 5 |

Полагаем *k* = 2.

Шаг 2. В матрице D1 выделены ведущие строка и столбец. Треугольный оператор применяется к элементам матрицы D1 и S1, выделенным двойной рамкой. В результате получаем матрицы D2 и S2:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D2 | 1 | 2 | 3 | 4 | 5 |  | S2 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 3 | 10 | **8** | ∞ |  | 1 | 1 | 1 | 1 | **2** | 1 |
| 2 | 3 | 0 | 13 | 5 | ∞ |  | 2 | 2 | 2 | 1 | 2 | 2 |
| 3 | 10 | 13 | 0 | 6 | 15 |  | 3 | 3 | 1 | 3 | 3 | 3 |
| 4 | **8** | 5 | 6 | 0 | 4 |  | 4 | **2** | 4 | 4 | 4 | 4 |
| 5 | ∞ | ∞ | ∞ | 4 | 0 |  | 5 | 5 | 5 | 5 | 5 | 5 |

Полагаем *k* = 3.

Шаг 3. В матрице D2 выделены ведущие строка и столбец. Треугольный оператор применяется к элементам матрицы D2 и S2, выделенным двойной рамкой. В результате получаем матрицы D3 и S3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D3 | 1 | 2 | 3 | 4 | 5 |  | S3 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 3 | 10 | 8 | **25** |  | 1 | 1 | 1 | 1 | 2 | **3** |
| 2 | 3 | 0 | 13 | 5 | **28** |  | 2 | 2 | 2 | 1 | 2 | **3** |
| 3 | 10 | 13 | 0 | 6 | 15 |  | 3 | 3 | 1 | 3 | 3 | 3 |
| 4 | 8 | 5 | 6 | 0 | 4 |  | 4 | 2 | 4 | 4 | 4 | 4 |
| 5 | ∞ | ∞ | ∞ | 4 | 0 |  | 5 | 5 | 5 | 5 | 5 | 5 |

Полагаем *k* = 4.

Шаг 4. Ведущие строка и столбец в матрице D3 выделены. Получаем новые матрицы D4 и S4.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D4 | 1 | 2 | 3 | 4 | 5 |  | S4 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 3 | 10 | 8 | **12** |  | 1 | 1 | 1 | 1 | 2 | **4** |
| 2 | 3 | 0 | **11** | 5 | **9** |  | 2 | 2 | 2 | **4** | 2 | **4** |
| 3 | 10 | **11** | 0 | 6 | **10** |  | 3 | 3 | **4** | 3 | 3 | **4** |
| 4 | 8 | 5 | 6 | 0 | 4 |  | 4 | 2 | 4 | 4 | 4 | 4 |
| 5 | **12** | **9** | **10** | 4 | 0 |  | 5 | **4** | **4** | **4** | 5 | 5 |

Полагаем *k* = 5.

Шаг 5. Ведущие строка и столбец в матрице D4 выделены. Никаких действий на этом шаге не выполняем; вычисления закончены.

Конечные матрицы D4 и S4 содержат всю информацию, необходимую для определения кратчайших путей между любыми двумя узлами сети. Например, кратчайшее расстояние между узлами 1 и 5 равно *d*15 = 12.

Для нахождения соответствующих маршрутов напомним, что сегмент маршрута (*i*, *j*) состоит из ребра (*i*, *j*) только в том случае, когда *sij* = *j*. В противном случае узлы *i* и *j* связаны, по крайней мере, через один промежуточный узел. Например, поскольку *s*15 = 4 и *s*45 = 5, сначала кратчайший маршрут между узлами 1 и 5 будет иметь вид 1→4→5. Но так как *s*14 не равно 4, узлы 1 и 4 в определенном пути не связаны одним ребром (но в исходной сети они могут быть связаны непосредственно). Далее следует определить промежуточный узел (узлы) между первым и четвертым узлами. Имеем *s*14 = 2 и *s*24 = 4, поэтому маршрут 1→4 заменяем 1→2→4. Поскольку *s*12 = 2 и *s*24 = 4, других промежуточных узлов нет. Комбинируя определенные сегменты маршрута, окончательно получаем следующий кратчайший путь от узла 1 до узла 5: 1→2→4→5. Длина этого пути равна 12 километрам.

## 2.6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определение маршрута, цепи, цикла.
2. Какой граф называется связным? Что такое компонента связности?
3. Привести примеры графов, которые имеют все периферийные и все центральные вершины.
4. Что такое эксцентриситет?
5. Чем диаметр графа отличается от его радиуса (дайте их определение)?
6. Что такое число вершинной и реберной связности?
7. Дайте определения моста и точки сочленения.
8. Для произвольного графа постройте матрицу расстояний (вес любого ребра равен 1).
9. Привести пример графа, удовлетворяющего строгому неравенству теоремы Уитни.
10. Сформулируйте задачу нахождения кратчайших маршрутов в графе.

# 3. ДЕРЕВЬЯ И ОСТОВЫ

## 3.1. ОПРЕДЕЛЕНИЯ ДЕРЕВА

Дерево – это специфический вид графов. **Деревом** называется связный граф, не содержащий циклов. Любой граф, не содержащий циклов, называется **ациклическим,** или **лесом**. Компонентами леса являются деревья.

*Пример*:

Выше приведено основное определение дерева. Следующая теорема дает 5 различных определений дерева

**Теорема эквивалентности.** Для любого (p,q) – графа следующие утверждения эквивалентны:

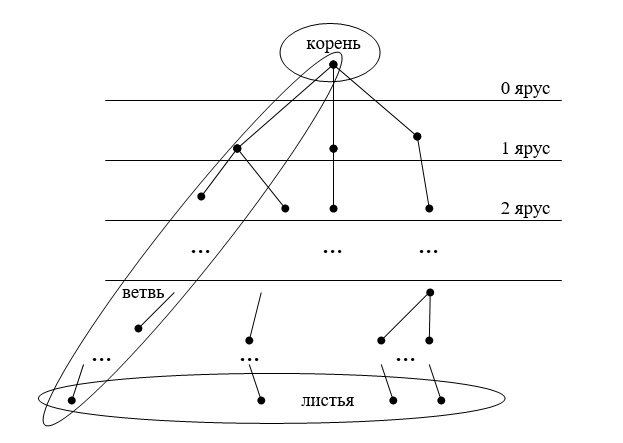
1. граф G – дерево: связный ацикличный граф;
2. любые две несовпадающие вершины графа G соединены единственной простой цепью;
3. граф G связен и q=p-1(количество ребер в нем на 1 меньше, чем количество вершин);
4. граф G ацикличен и q=p-1;
5. граф G ацикличен и, если любую пару его несовпадающих несмежных вершин соединить ребром е, то полученный граф G+e будет содержать в точности один цикл.

**Теорема о висячих вершинах дерева**. В любом нетривиальном () дереве имеются, по крайней мере, две висячие вершины.

**Теорема о центральных вершинах дерева**. Центр любого дерева состоит либо из одной, либо из двух (смежных) вершин.

## 3.2. ЯРУСНАЯ ФОРМА ПРЕДСТАВЛЕНИЯ И СПОСОБЫ ОБХОДА ДЕРЕВЬЕВ.

Выберем некоторую произвольную фиксированную вершину дерева и назовем ее **корнем**. Будем полагать, что корень дерева расположен на 0 ярусе. Все остальные вершины сориентируем относительно корня следующим образом:   
на i-й ярус поместим вершины с расстоянием от корня, равным i. Концевые висячие вершины будем называть **листами**, а геодезические от корня к листу – **ветвями**.



Существует два основных способа обхода деревьев, подразумевающих просмотр ветвей в его ярусном представлении.

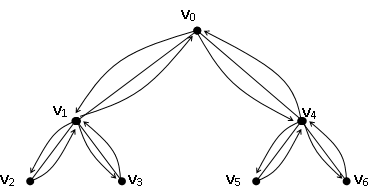
**1. Способ обхода (поиска) в глубину.**

Начинаем поиск с некоторой фиксированной вершины v0. Находим вершину u, смежную с v0, и повторяем процесс, начиная с вершины u:

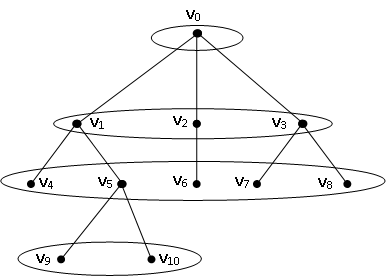
- если существует не просмотренная вершина, смежная с вершиной u, рассматриваем ее и, начиная с нее, продолжаем поиск;

- если не существует ни одной новой вершины, смежной с u, то говорят, что вершина u использована, и делается возврат в вершину, из которой мы попали в вершину u; продолжаем процесс;

- если на каком-то шаге u= v0, и нет ни одной не просмотренной вершины, смежной v0, то алгоритм заканчивает работу.



**2. Способ обхода (поиска) в ширину** подразумевает просмотр вершин по ярусам с перебором вершин одного яруса.



## 3.3. ОСТОВЫ И АЛГОРИТМЫ ИХ ПОСТРОЕНИЯ

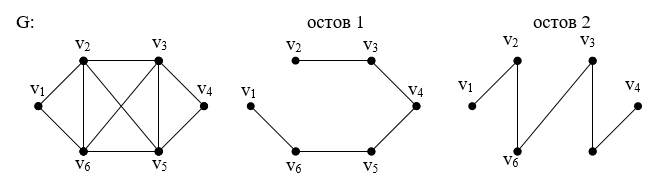
**Остов (каркас, скелет)** графаG – это остовный подграф графа G, задающий дерево на каждой компоненте связности графа G.

Для связного графа **остов** – это дерево, покрывающее все вершины исходного графа.

Очевидно, что в каждом графе существует остов, в общем случае, не один. Его можно получить, разрушая циклы в каждой компоненте связности.

Ребра остова Т некоторого графа G называются **ветвями**, а ребра графа G, не вошедшие в остов, называются **хордами**.

*Пример:*



В первом остове графа G:

(v2,v3) , (v3,v4) , (v4,v5) , (v5,v6) , (v1,v6) – ветви;

(v1,v2) , (v2,v6) , (v2,v5) , (v3,v6) , (v3,v5) – хорды.

### 3.3.1. Теорема Кирхгофа

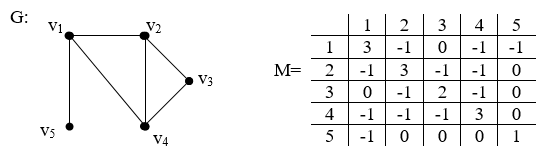
Число остовных деревьев в связном графе G определяется из теоремы Кирхгофа.

Матрица Кирхгофа – это квадратная матрица

, 

где  – соответствующий элемент матрицы смежности.

*Пример:*



Матрица Кирхгофа обладает следующим свойством: сумма элементов любой строки матрицы равна сумме элементов любого столбца, равна сумме всех элементов матрицы и равна нулю.

**Теорема** **Кирхгофа.** Число остовных деревьев в связном графе G порядка n (n ≥ 2) равно алгебраическому дополнению любого элемента матрицы Кирхгофа.

В связном помеченном графе все алгебраические дополнения матрицы Кирхгофа равны между собой и определяют общее число помеченных остовов графа.

***Следствие*** ***из теоремы Кирхгофа***. При n ≥ 2 (где n − количество вершин графа) число остовов полного графа Kn равно .

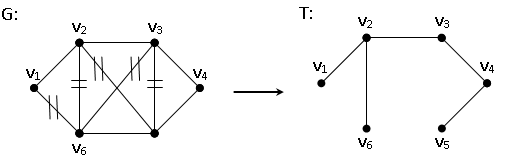
**Теорема А. Кэли**. Число помеченных деревьев порядка n равно .

### 3.3.2. Алгоритмы построения остова

Остов Т можно построить удалением ребер исходного графа, либо добавлением ребра к пустому графу.

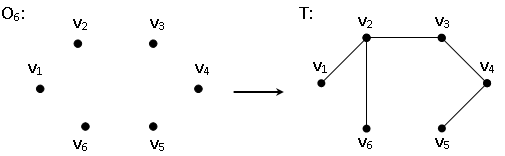
При **удалении ребер исходного графа** очередное удаляемое ребро не должно приводить к нарушению связности, но делать граф ацикличным.

*Пример:*



При **добавлении ребра пустому графу (**количество вершин в нем должно совпадать с количеством вершин исходного графа**)** очередное добавляемое ребро не должно приводить к образованию цикла.

*Пример:*



Алгоритмы построения **остовов кратчайших маршрутов** из произвольной вершины строят не просто дерево, а геодезические до каждой вершины графа.

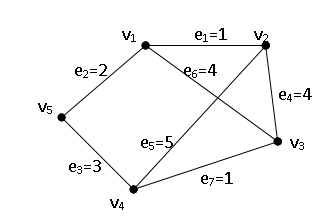
Постановка задачи.Имеется связный граф G = (V, E), заданный матрицей весов C = ||Cij||, i, j = . Требуется найти остов с минимальным суммарным весом ребер.

Для поиска остовов кратчайших маршрутов используются алгоритмы Прима и Краскала.

**Алгоритм Краскала** состоит в следующем. Ребра графа упорядочиваются в порядке не убывания их весов. На каждом шаге к пустому графу Ор, где р − количество вершин исходного графа, добавляется ребро с минимальным весом из списка. Добавляемое ребро не должно приводить к образованию цикла. Алгоритм заканчивает работу, если количество ребер в формируемом графе станет равным   
р – 1.

**Алгоритм Прима** отличается от алгоритма Краскала тем, что на каждом шаге строится не просто граф без циклов, но дерево. Упорядочиваются ребра исходного графа в порядке не убывания их весов. Строится пустой граф Ор, где р − количество вершин исходного графа. На каждом шаге в граф Ор добавляется ребро из списка ребер таким образом, чтобы не образовался цикл и не нарушалась связность. Для этого список ребер каждый раз просматривается сначала. Алгоритм строит остов кратчайших маршрутов посредством разрастания поддерева T за счет присоединения очередного ребра (xi, xj) с наименьшим весом.

*Пример*: для заданного графа построить остов кратчайших маршрутов.



Упорядочим ребра в порядке их возрастания.

|  |  |
| --- | --- |
| Ребра | Вес |
| e1 = (v1,v2) | 1 |
| e7 = (v3,v4) | 1 |
| e2 = (v1,v5) | 2 |
| e3 = (v4,v5) | 3 |
| e6 = (v1,v3) | 4 |
| e4 = (v2,v3) | 4 |
| e5 = (v2,v4) | 5 |

|  |  |  |
| --- | --- | --- |
| № итерации | алгоритм Краскала | алгоритм Прима |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |

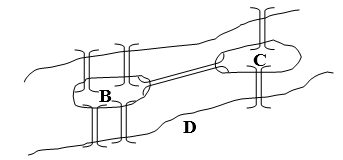
## 3.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определения дерева, леса.
2. Рисунки деревьев заданного порядка с указанием центральных вершин.
3. Сгенерировать все различные абстрактные, неизоморфные друг другу деревья порядка (4,7). Разделить множество деревьев на 2 подмножества с одной и двумя центральными вершинами.
4. Теорема о висячих вершинах дерева.
5. Ярусное представление деревьев.
6. Перечислить способы обходы деревьев. Найти обходы для заданного дерева.
7. Определение остова, ветвей, хорд. Алгоритмы построения остовных деревьев.
8. Остовы кратчайших маршрутов. Отличие алгоритмов Прима и Краскала.
9. Матричная теорема Кирхгофа и следствие из нее.
10. Теорема Кэли.

# 4. ЭЙЛЕРОВЫ И ГАМИЛЬТОНОВЫ ГРАФЫ. ПЛОСКИЕ И ПЛАНАРНЫЕ ГРАФЫ.РАСКРАСКА ГРАФОВ

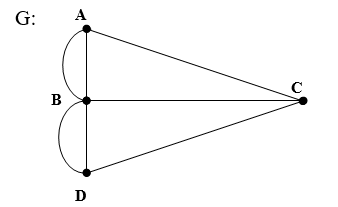
## 4.1. ЭЙЛЕРОВЫ ГРАФЫ И ЭЙЛЕРОВЫ ЦИКЛЫ

Начало теории графов, как отмечалось ранее, связывают с так называемой задачей о кенигсбергских мостах. Эта знаменитая в свое время задача состоит в следующем. Семь мостов г. Кенигсберга (ныне Калининград) были расположены на реке Прегель так, как изображено на рисунке.



Возможно ли, выйдя из дома, вернуться обратно, пройдя в точности один раз по каждому мосту?

Сопоставим рисунку граф G, вершины которого соответствуют четырем разделяемым рекой участкам суши A, B, C и D, а ребра – мостам.



Таким образом, данную задачу на языке теории графов можно сформулировать следующим образом: существует ли в мультиграфе G цикл, содержащий все его ребра ровно раз? В 1767 г. Л.Эйлер доказал, что данная задача не имеет решения. Он сформулировал и решил общую проблему теории графов: при каких условиях связный граф содердит цикл, проходящий через каждое его ребро?

**Цикл** в графе G называется **эйлеровым**, если он содержит все ребра исходного графа. Цепь, содержащая все ребра графа, называется **эйлеровой цепью**.

Связный граф называется **эйлеровым**, если он содержит эйлеровцикл.

*Пример:*

|  |  |
| --- | --- |
| не эйлеров граф | эйлеров граф |
|  |  |

**Теорема о существовании эйлерового цикла в графе.** Связный граф G является эйлеровым тогда и только тогда, когда степени всех его вершин четны.

***Следствие №1.*** Если граф G эйлеров, то множество его ребер можно разбить на простые циклы.

***Следствие №2.*** Для того чтобы связный граф G покрывался единственной эйлеровой цепью, необходимо и достаточно, чтобы он содержал ровно две вершины с нечетной степенью. Тогда цепь начинается в одной из этих вершин и заканчивается в другой.

Алгоритм построения эйлерового цикла, или алгоритм Флёри, сводится к нумерации ребер исходного графа от 1 до q так, чтобы номер ребра указывал, каким по счету это ребро войдет в эйлеров цикл.

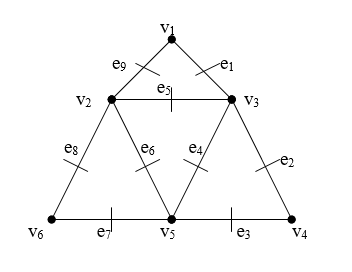
**Алгоритм Флёри.**

Начиная с любой вершины v, присваиваем ребру (v,u) номер 1. Вычеркиваем это ребро из списка ребер и переходим к вершине u.

Пусть w − вершина, в которую мы попали в результате выполнения предыдущего шага и k − номер, присвоенный очередному ребру на этом шаге. Выбираем произвольное ребро, инцидентное вершине w, причем мост и ребро (w,v) выбираем только в крайнем случае, если других возможностей выбора ребра не существует. Присваиваем этому ребру номер k + 1 и вычеркиваем его из списка ребер.

Алгоритм заканчивает работу, если мы попали в вершину v, и нет ни одного не пройденного ребра, инцидентного этой вершине.

*Пример*: эйлеров цикл: (v1,v3,v4,v5,v3,v2,v5,v6,v2,v1).



Обобщением задачи о поиске эйлерового цикла в графе является задача китайского почтальона.

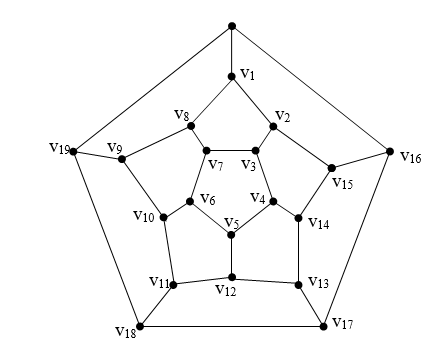
**Задача** **китайского почтальона:** построить маршрут (цикл, цепь) с минимальным суммарным весом, проходящий по каждому ребру графа.

## 4.2. ГАМИЛЬТОНОВЫ ГРАФЫ И ГАМИЛЬТОНОВЫ ЦИКЛЫ

Простой **цикл** называется **гамильтоновым**, если он содержит каждую вершину графа. **Гамильтонова цепь** – простая цепь, содержащая каждую вершину исходного графа.

Граф называется **гамильтоновым**, если он содержит гамильтонов цикл.

В 1859 г. ирландский математик У. Гамильтон предложил игру, которая называется «Кругосветное путешествие». Каждой из двадцати вершин додекаэдра приписано название одного крупного города мира. Требовалось, переходя от одного города к другому, посетить каждый город ровно один раз и вернуться в исходный город.



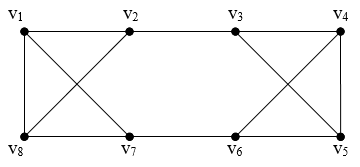
До сих пор не найдено эффективного критерия, отвечающего на вопрос, есть ли в графе гамильтонов цикл. Решается задача способом полного перебора (в ярусном представлении дерева). Поиск эффективных алгоритмов для решения данной задачи − одно из популярных направлений современной теории графов.

Достаточные условия существования Гамильтонова цикла в графе формулируются следующими теоремами.

**Теорема Г. Дирака.** Если число вершин графа p ≥ 3 и для любой вершины v выполняется условие: deg v ≥  , то граф G – гамильтонов.

**Теорема О. Оре (1960 г.).** Если число вершин графа *p* ≥ 3 и для любых двух несмежных вершин uи v выполняется неравенство: deg *u* + deg *v* ≥ *p*, то граф G – гамильтонов.

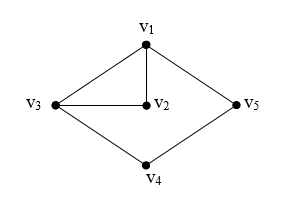
Следует отметить, что условия Оре-Дирака не являются необходимыми, т.е. существуют графы, в которые есть Гамильтонов цикл, но условия Оре-Дирака не выполняются. Например, следующий граф гамильтонов, в нем существует гамильтонов цикл M=(v1,v2,v3,v4,v5,v6,v7,v8,v1).



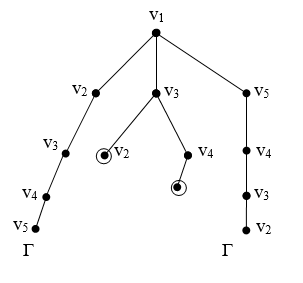
Однако, граф не удовлетворяет условиям Оре-Дирака, т.к.

p = 8; deg vi = 3; 3   = 4.

*Пример:* для графа, удовлетворяющего условиям Оре-Дирака, найти гамильтонов цикл.



Задачу решим способом полного перебора в ярусном представлении дерева, дерево полного перебора имеет вид:



(v1,v2,v3,v4,v5,v1), (v1,v5,v4,v3,v2,v1) – гамильтоновы циклы

Алгоритм Робертса – Флореса является алгоритмом поиска гамильтонова цикла полным перебором.

**Алгоритм.**

Выбирается некоторая начальная вершина графа v1. Эта вершина образует первый элемент множества S: S = {v1}. Множество S на каждом шаге будет хранить уже найденные вершины гамильтоновой цепи. К S добавляется первая вершина (пусть это вершина a) в столбце v1. Затем к S добавляется первая возможная вершина в столбце a. Пусть это вершина b, затем к S добавляется первая возможная вершина (например, с) в столбце b и т.д.

S = {v1, a, b, с, . . .}

Под «возможной» понимается вершина, которой нет в S.

Пусть множество S на шаге r имеет вид:

S = {v1, a, b, c, …, vr-1, vr}.

Существует **две причины**, препятствующих включению некоторой вершины во множество S.

1. В столбце vr нет возможной вершины (множество S нельзя расширить).
2. Цепь, определяемая последовательностью вершин в множестве S, имеет длину p-1, где p – количество вершин графа, т.е. она является гамильтоновой цепью.

Во втором случае тоже 2 варианта:

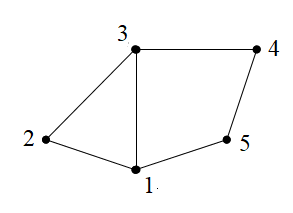
а) в графе G существует ребро (vr,v1), следовательно, найден гамильтонов цикл;

б) ребро (vr,v1) не существует, следовательно, гамильтонов цикл не может быть получен.

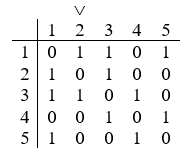
В случаях 1) и 2б) следует прибегнуть к возвращению, а в случае 2а) прекратить поиск (если необходимо найти только один гамильтонов цикл) или (если нужны все гамильтоновы циклы) прибегнуть к возвращению. Возвращение состоит в удалении последней включенной вершины из S, после чего множество S примет вид: S = {v1, a, …, vr-1} и добавлении к S первой возможной вершины, следующей за vr в столбце vr-1. Если не существует никакой возможной вершины, то делается следующий шаг возвращения и т. д.

Поиск заканчивается, когда множество S состоит из одной вершины v1 и не существует никакой возможной вершины, которую можно добавить в S, так что шаг возвращения делает S пустым. Это значит, что все гамильтоновы циклы найдены. Алгоритм заканчивает работу.

*Пример:* с помощью алгоритма Робертса – Флореса найти все гамильтоновы циклы в графе, представленном на рисунке



На этапе инициализации пусть множество S = {1}. Во множество S можно добавить вершины 2, 3 и 5. Рассмотрим все варианты, начиная с вершины 2, т.е. в столбце 1 матрицы смежности выберем вершину 2.



Итерации алгоритма следующие:

1. S = {1}
2. S = {1, 2} – добавление вершины 2
3. S = {1, 2, 3}
4. S = {1, 2, 3, 4}
5. S = {1, 2, 3, 4, 5} − гамильтонов цикл
6. S = {1, 2, 3, 4} – возращение
7. S = {1, 2, 3} – возращение
8. S = {1, 2} – возращение
9. S = {1} – возращение
10. S = {1,3} – добавление вершины 3
11. S = {1,3,2}
12. S = {1,3}
13. S = {1,3,4}
14. S = {1,3,4,5}
15. S = {1,3,4} – возращение
16. S = {1,3} – возращение
17. S = {1} – возращение
18. S = {1, 5} – добавление вершины 5
19. S = {1, 5, 4}
20. S = {1, 5, 4, 3}
21. S = {1, 5, 4, 3, 2} − гамильтонов цикл
22. S = {1, 5, 4, 3} – возращение
23. S = {1, 5, 4} – возращение
24. S = {1, 5} – возращение
25. S = {1} – возращение
26. S = ∅ – конец алгоритма

Обобщением задачи о поиске гамильтонова цикла в графе является задача коммивояжера.

**Задача коммивояжера:** в полном взвешенном графе G построить маршрут (цикл, цепь) минимального веса, проходящий по крайней мере один раз через каждую вершину исходного графа.

Если такой маршрут существует, то задача сводится к задаче поиска гамильтонова цикла.

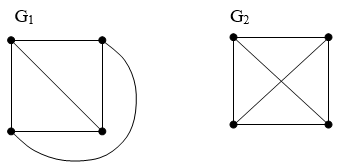
## 4.3. ПЛОСКИЕ И ПЛАНАРНЫЕ ГРАФЫ

**Плоским** называется такой граф G, у которого вершины − точки плоскости, а ребра − непрерывные плоские линии без пересечений и самопересечений, т.е. ребра соединяют вершины таким образом, что никакие два ребра не имеют общих точек, кроме инцидентных им обоим вершин. Примерами плоских графов являются простые циклы, деревья, лес.

*Примеры плоских графов:*

**Планарный** − это граф, который изоморфен плоскому.

*Пример планарного и плоского графа:*



Очевидно, что G1  G2, граф G1 − плоский, G2 − планарный.

О планарных графах говорят, что они имеют плоскую укладку.

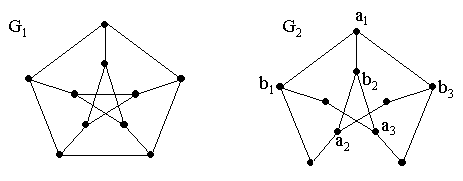
### 4.3.1. Критерии планарности графа. Теорема Эйлера.

Критерии планарности выражены двумя теоремами, приведенными ниже.

**Теорема Понтрягина-Куратовского**. Граф планарен тогда, когда он не содержит подграфов гомеоморфных K5 или K3,3.

**Теорема Вагнера.** Граф планарен тогда, когда он не содержит подграфов, стягиваемых K5 или K3,3 (путем применения операции стягивания).

*Пример:* граф Петерсена G1



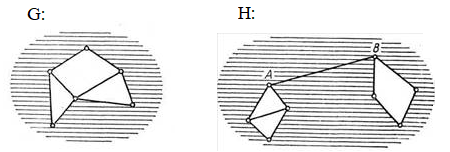
Граф G1 не планарный, т.к. он содержит подграф G2, гомеоморфный K3,3.

Отметим, что, стягивая любое ребро планарного графа, вновь получим планарный граф. Если же дан непланарный граф, то стянув одно или несколько ребер можно получить планарный граф. В качестве характеристики плоского представления графа вводится понятие грани.

**Гранью** в плоском представлении графа G называется часть плоскости, ограниченная простым циклом и не содержащая внутри других циклов. Грани могут быть внешними и внутренними.

**Внешняя грань** – это грань, которая окружает весь граф и не имеет ребер внутри себя, являясь границей всего графа. Она расположена вне плоского представления графа, ограничена изнутри простым циклом и не содержит в себе других циклов. Ее еще называют **бесконечной** гранью.

*Пример:*



Часть заштрихованной плоскости для графа G является внешней гранью, а для графа H не является, поскольку она не огра­ничена изнутри простым циклом.

Всегда имеется одна неограниченная внешняя грань, все остальные грани называются внутренними.

**Внутренние грани** – это грани, которые находятся внутри внешней грани и имеют ребра внутри себя. Они образуются в результате разбиения плоскости на несколько частей с помощью ребер графа.

*Пример:* найти грани в графе, изображенном на рисунке

1

4

3

5

2

6

7

Грани: (1, 7, 4, 1), (1, 3, 2, 4, 1), (1, 2, 3, 1), (2, 6, 5, 4, 2), (1, 2, 6, 5, 4, 7, 1).

**Примечание:** часть плоскости, ограниченная простым циклом (1, 2, 4, 1), не является гранью, поскольку она содержит внутри себя цикл (1, 2, 3, 1).

*Пример:* назвать все грани графа H.

1

3

2

H:

4

5

Гранями графа H являются (1, 2, 3, 4) и (1, 2, 3, 4, 5, 1).

**Примечание:** часть плоскости, ограниченная простым циклом (1, 2, 3, 4, 5, 1), является гранью, поскольку ребро (4, 5), расположенное внутри грани, не образует цикла.

*Пример:* плоский граф G имеет 3 грани, причем грань 1 – внешняя, а грани 2 и 3 – внутренние.

1

3

2

G:

**Внутренние области** – это области, которые находятся внутри внутренних граней. Они могут быть связными или несвязными, в зависимости от того, есть ли пути между вершинами внутри области.

Если в плоском графе нет циклов, то у него имеется только одна грань, т.е дерево и лес имеют одну бесконечную грань. Если же циклы есть, то граница каждой грани содержит цикл, но не обязательно является циклом.

**Теорема Эйлера для плоского графа.** Для любого связного графа G, являющегося плоским, справедливо соотношение:

p – q + f = 2,

где p и q – количество его вершин и ребер, f – количество его граней.

**Следствие 1.** Для связного плоского графа, содержащего p вершин и q ребер, при p > 3 выполняется неравенство q ≤ 3 p – 6.

**Следствие 2.** В любом планарном графе существует вершина, степень которой не больше 5.

Грани и внутренние области в планарном графе играют важную роль при анализе и визуализации графа. Они помогают определить структуру и связи между элементами графа, а также позволяют лучше понять его свойства и характеристики.

Приведем примеры практических задач использования теоремы Эйлера.

**Задача 1.** Мэрия решила построить в каждом квартале города, имеющего 155 перекрестков и 260 отрезков улиц между перекрестками, универсам. Сколько будет построено универсамов?

*Решение:* карту города можно считать плоским графом G, в котором перекрестки будут вершинами, а отрезки улиц – ребрами. Поскольку кварталы города соответствуют внутренним граням плоского графа G, то найдем число граней по формуле Эйлера. Граф G имеет 155 вершин и 260 ребер. Число граней в нем: f = q – p + 2 = 260 - 155 + 2 = 107.

*Ответ:* в городе нужно построить 106 универсамов.

**Задача 2.** Печатная плата представляет собой пластинку из изолирующего материала, в специально изготовленные гнезда которой устанавливают электронные приборы. В качестве проводников, соединяющих эти приборы, служат напыленные металлические дорожки. Поскольку проводники не изолируются, то дорожки не должны пересекаться. Если это может произойти, то одну из дорожек переносят на другую сторону платы. Конструктор Иванов придумал хорошую схему печатной платы, которая состоит из 12 приборов и 32 проводников, соединяющих их. Можно ли изготовить такую плату так, что все проводники будут расположены на одной ее стороне?

*Решение:* схему печатной платы можно представить в виде графа, вершины которого будут изображать приборы, а ребра – проводники, их соединяющие. Решение задачи сводится к ответу на вопрос: будет ли граф G, изображающий плату инженера Иванова, плоским?

Для графа G, описывающего плату инженера Иванова, p = 12, q = 32, и неравенство, являющееся следствием 1 из теоремы Эйлера, не выполняется.

*Ответ:* граф G не является плоским, следовательно, изготовить одностороннюю плату невозможно.

Зачастую критерии планарности очень трудно проверить на практике, поэтому, чтобы проверить планарность графа и произвести его плоскую укладку, удобно пользоваться гамма-алгоритмом.

### 4.3.2. Гамма-алгоритм плоской укладки графа

На вход алгоритму подаются графы со следующими свойствами:

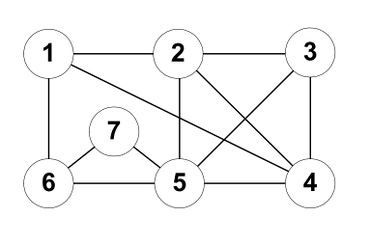
1. Граф связный.
2. Граф содержит хотя бы один цикл.
3. Граф не имеет мостов.

Если нарушено свойство 1, то граф нужно укладывать отдельно по компонентам связности. Если нарушено свойство 2, то граф – дерево и нарисовать его плоскую укладку тривиально.

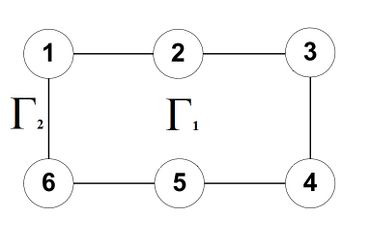
Более подробно рассмотрим случай, когда в графе G нарушено свойство 3. Сначала все мосты нужно убрать, далее произвести отдельную укладку всех компонент следующим образом: уложим одну компоненту связности, а следующую компоненту, связанную с первой в графе G мостом, будем рисовать в той грани, в которой лежит вершина, принадлежащая мосту. Иначе может сложиться ситуация, когда концевая вершина моста будет находиться внутри плоского графа, а следующая компонента – снаружи. Таким образом мы сможем соединить мостом нужные вершины. Далее будем так поступать с каждой новой компонентой.

Алгоритм укладки графа G представляет собой процесс последовательного присоединения к некоторому уложенному подграфу G0 графа G новой цепи, оба конца ко­торой принадлежат G0. Тем самым эта цепь разбивает одну из граней графа G0 на две. При этом в качестве на­чального плоского графа G0 выбирается любой простой цикл графа G. Процесс продолжается до тех пор, пока не будет построен плоский граф, изоморфный графу G, или присоединение некоторой цепи окажется невозможным. В последнем случае граф G не является планарным.

Рассмотрим работу алгоритма, параллельно разбирая на примере каждый шаг. Пусть дан граф G:



Первый этап – инициализация алгоритма. В графе G выбирается любой простой цикл и производится его укладка на плоскость. Пусть в примере это будет цикл (1,2,3,4,5,6). После его укладки получаем две грани: Γ1 и Γ2.

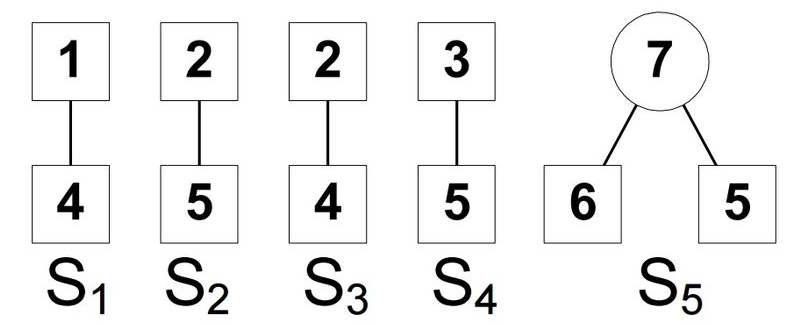


Уже уложенную во время работы алгоритма часть будем обозначать Gplane. В примере сейчас Gplane – выбранный цикл (1,2,3,4,5,6).

Второй этап – общий шаг. Построим множество сегментов. Каждый **сегмент** S относительно уже построенного Gplane представляет собой одно из двух:

1. ребро, оба конца которого принадлежат Gplane, но само оно не принадлежит Gplane;
2. связную компоненту графа G∖Gplane, дополненную всеми ребрами графа G такими, у которых один из концов принадлежит связной компоненте, а второй принадлежит графу Gplane.

Вершины, одновременно принадлежащие Gplane и какому-либо сегменту, назовем **контактными**. На рисунке ниже изображены сегменты для исходного графа G. Контактные вершины обведены в квадрат.

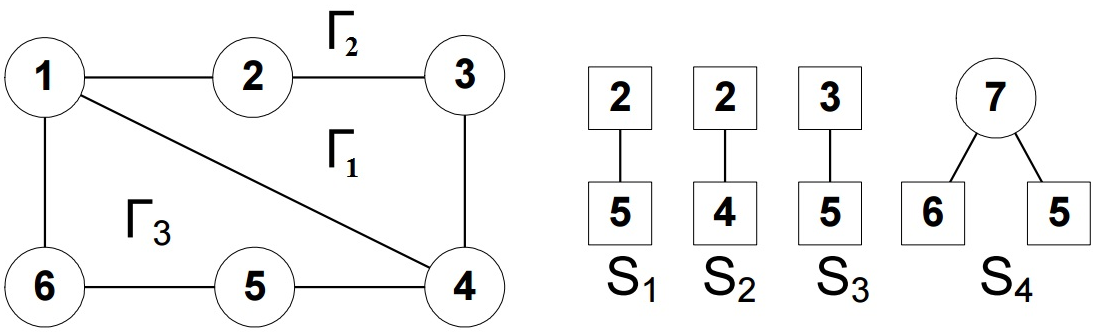


Пусть в каком-то сегменте нет ни одной контактной вершины. В таком случае граф до выделения Gplane был несвязным, что противоречит условию. Пусть контактная вершина в сегменте только одна. Это значит, что в графе был мост или точка сочленения, чего быть не может так же по условию. Таким образом, в каждом сегменте имеется не менее двух контактных вершин. Соответственно, в каждом сегменте есть цепь между любой парой контактных вершин.

Пусть грань Γ вмещает сегмент S, если номера всех контактных вершин S принадлежат этой грани, S⊂Γ. Очевидно, таких граней может быть несколько. Множество таких граней обозначим Γ(S), а их число – |Γ(S)|.

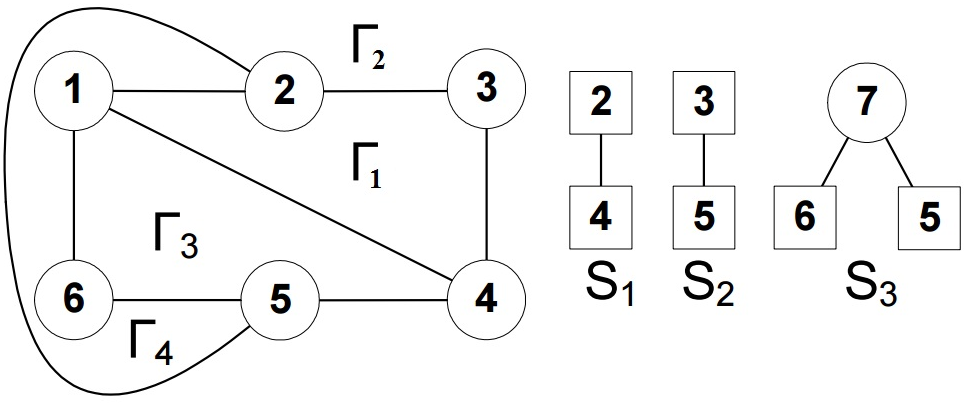
Итак, рассмотрим все сегменты Si и для каждого определим число |Γ(Si)|. Если найдется такой номер i, что |Γ(Si)|=0, то граф не планарен, алгоритм завершает работу. Иначе выбираем такой сегмент Si, для которого число |Γ(Si)| минимально. Если таких сегментов несколько, можно выбрать любой из них. Найдем в этом сегменте цепь между двумя контактными вершинами и уложим ее в любую грань из множества Γ(Si), совместив контактные вершины сегмента с соответствующими вершинами грани. Выбранная грань разобьется на две. Выбранный сегмент после того, как из него взяли цепь, либо исчезнет, либо распадется на меньшие части, в которых будут новые контактные вершины, ведущие к вершинам обновленного Gplane.

В нашем примере для любого i: Si⊂{Γ1,Γ2}, то есть |Γ(Si)|=2. Следовательно, можем выбрать любой Si. Пусть это будет сегмент S1. В нем есть цепь (1,4). Вставим эту цепь в грань Γ1, например, и этот сегмент исчезнет. После укладки цепи граф G и сегменты будут выглядеть так:

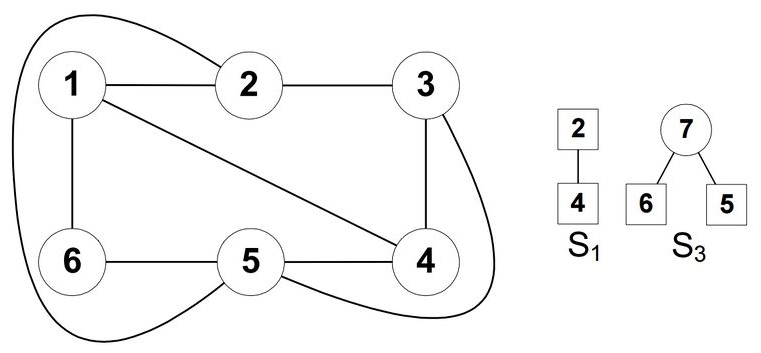


Третий и последующие этапы аналогичны второму. Повторять вышеуказанные действия нужно либо до тех пор, пока не будет получена плоская укладка, то есть множество сегментов не останется пустым, либо пока не будет получено, что граф не планарен.

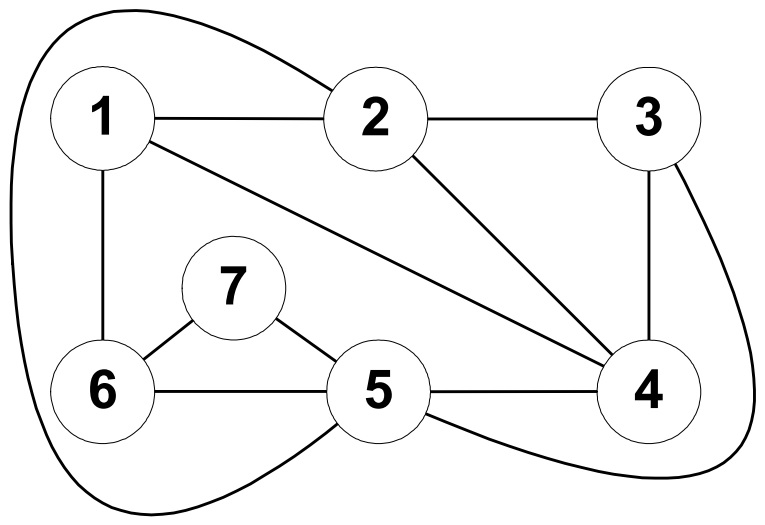
Разберем пример до конца. Повторим снова общий шаг. Теперь |Γ(S1)|=|Γ(S3)|=1, |Γ(S2)|=|Γ(S4)|=2. Возьмем S1. В ней цепь (2,5), которую мы уложим в грань Γ2, после чего этот сегмент исчезнет. Теперь картина будет следующая:



На следующем шаге |Γ(S1)|=|Γ(S3)|=2, |Γ(S2)|=1. Выбираем сегмент S2, содержащий цепь (3,5). Уложим ее в грань Γ2, после чего этот сегмент снова исчезнет:



Теперь |Γ(S1)|=1, а |Γ(S3)|=2. Уложим сначала цепь (2,4) из первого сегмента, он пропадет, потом уложим цепь (6,7,5) из третьего. В результате граф будет полностью уложен на плоскость, множество сегментов останется пустым:



Таким образом, мы получили плоскую укладку исходного графа G.

Опишем алгоритм коротко и формально:

1. Инициализация. Выбирается простой цикл в исходном графе и изображается на плоскости.
2. Общий шаг. Этот шаг повторяется до тех пор, пока граф не будет уложен или пока не будет получено, что граф не планарен.

* строится множество сегментов;
* для каждого сегмента вычисляется величина |Γ(S)|. Если существует i: |Γ(Si)|=0, то граф не планарен, алгоритм завершает работу;
* выбирается сегмент с минимальным числом |Γ(Si)|;
* в этом сегменте выбирается цепь между двумя контактными вершинами;
* эта цепь укладывается в любую грань, вмещающую данный сегмент.

1. Либо получена плоская укладка графа, либо граф оказался не планарен.

## 4.4. РАСКРАСКА ГРАФОВ

Раскраска элементов графа в k цветов, где *k* − натуральное число, представляет собой разбиение элементов графа на k классов. Рассматривают раскраски вершин, ребер и областей неориентированных графов. При раскраске элементам графа ставятся в соответствие метки с учетом определенных ограничений; эти метки традиционно называются «цветами». В простейшем случае такой способ окраски вершин графа, при котором любым двум смежным вершинам соответствуют разные цвета, называется раскраской вершин. Аналогично раскраска ребер присваивает цвет каждому ребру так, чтобы любые два смежных ребра имели разные цвета. Наконец, раскраска областей планарного графа назначает цвет каждой области, так, что каждые две области, имеющие общую границу, не могут иметь одинаковый цвет. В дальнейшем мы ограничимся сведениями о вершинных раскрасках.

### 4.4.1. Определение и терминология

Пусть имеется некоторый неориентированный граф G, тогда ***k*-раскраской** графа G называется произвольная функция *f*, отображающая множество вершин графа G в некоторое *k*-элементное множество:

*f*: *VG* → {*a*1, *a2*, …, *ak*} = *A*.

В качестве элементов множества *A* чаще всего используется отрезок натурального ряда {1, 2, …, *k*} либо {*a*, *b*, …, *n*} или краски типа {синий, красный, …, черный}.

Раскраска называется **правильной**, если *f*(*u*) ≠ *f*(*v*) для любых смежных вершин *u* и *v* графа G (или концевые вершины любого ребра окрашены в разные цвета).

Граф, для которого существует правильная *k*-раскраска, называется ***k*-раскрашиваемым**.

Задача раскраски заключается в нахождении минимального числа цветов, достаточного для правильной раскраски.

Задача о раскраске плоских карт – одна из самых старых проблем теории графов. Еще в конце XIX в. было доказано, что для раскраски любой плоской карты достаточно пяти красок. Высказанная тогда же «гипотеза четырех красок» (всегда достаточно четырех красок) была доказана в 70-х гг. XX в. американскими математиками Аппелем и Хакеном. Их существенно использовало компьютерные вычисления, поскольку было связано с построением и анализом около двух тысяч различных случаев.

**Хроматическое число** χ(G) графа G **−** это минимальное число красок, при котором граф имеет правильную раскраску. Граф называется ***k*-хроматическим**, если χ(G) = *k*, и бихроматическим, если χ(G) = 2.

Для полного графа Kn хроматическое число равно χ(Kn) = *n*, для цикла с четным числом вершин – χ(Cчетн.) = 2, с нечетным числом вершин – χ(C2*n + 1*) = 3.

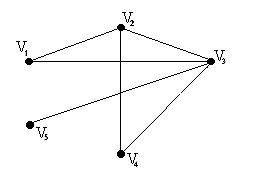
Для пустого графа хроматическое число равно χ(0*n*) = 1.

**Теорема Кёнига.** Непустой граф является бихроматическим тогда и только тогда, когда он не содержит циклов нечетной длины.

**Следствие 1.** Любое дерево бихроматично.

**Следствие 2.** Любой двудольный граф бихроматичен.

*Пример:* раскрасить правильно граф, приведенный на рисунке, найти его хроматическое число:



Для данного графа существует несколько вариантов правильной раскраски

|  |  |
| --- | --- |
|  | граф 5-раскрашиваемый |
|  | граф 3-раскрашиваемый |

Хроматическое число графа χ = 3.

Правильную *k*-раскраску графа G можно рассматривать как разбиение множества вершин графа G на не более чем, *k* непустых множеств, которые называются **цветными классами**:

*V* = *V1* ∪ … *Vk*.

Каждый цветной класс является независимым множеством.

### 4.4.2. Алгоритмы раскраски графов

Приведем формальное описание **последовательного метода раскраски**, основанного на упорядочивании множества вершин.

1. Упорядочить вершины по невозрастанию их степени.

2. Окрасить первую вершину в цвет 1.

3. Выбрать цвет окраски 1.

4. Пока не окрашены все вершины, повторять п.4.1.-4.2.:

4.1. Окрасить в выбранный цвет всякую вершину, которая не смежна с другой, уже окрашенной в этот цвет.

4.2. Выбрать следующий цвет.

В этом простейшем из методов вершины вначале располагаются в порядке невозрастания их степеней. Первая вершина окрашивается в цвет 1; затем список вершин просматривается сверху вниз (по невозрастанию степеней) и в цвет 1 окрашивается всякая вершина, которая не смежна с другой, уже окрашенной в этот цвет. Потом возвращаемся к первой в списке неокрашенных вершин, окрашиваем ее в цвет 2 и снова просматриваем список вершин сверху вниз, окрашивая в цвет 2 любую неокрашенную вершину, которая не соединена ребром с другой, уже окрашенной в цвет 2 вершиной. Аналогично действуем с цветами 3, 4 и т. д., пока не будут окрашены все вершины. Число использованных цветов будет приближенным значением хроматического числа графа.

Приведенный выше алгоритм можно модифицировать. Для этого после каждого шага нужно упорядочивать неокрашенные вершины. Простая модификация описанной выше эвристической процедуры состоит в переупорядочивании неокрашенных вершин по невозрастанию их относительных степеней. Под относительными степенями понимаются степени соответствующих вершин в неокрашенном подграфе исходного графа.

В данной модификации предполагалось, что если две вершины имеют одинаковые степени, то порядок таких вершин случаен. Такие вершины можно также упорядочить, но уже по двухшаговым степеням. Двухшаговую степень определим как сумму относительных степеней инцидентных вершин. Аналогично можно поступать и далее.

Для определения хроматического числа графа может быть использован достаточно эффективно простой метод неявного перебора.

**Алгоритм неявного перебора.**

1. Произвольной вершине графа G приписываем цвет 1.
2. Пусть раскрашены *i* вершин графа G в цвета от 1 до *l*, где *l* ≤ *i*. Произвольной неокрашенной вершине *vi+*1 приписываем минимальный цвет неиспользованной при раскраске смежных вершин.

Данный алгоритм можно ускорить, используя следующие замечания:

1. При любом упорядочении вершин допустимые цвета j для вершины xi удовлетворяют условию j≤i.
2. С вычислительной точки зрения выгодно размещать вершины так, чтобы первые вершины образовывали максимальную клику графа. Тогда все вершины, входящие в эту клику, будут окрашены в цвет 1 и алгоритм может закончить работу раньше.

Алгоритм последовательной раскраски зависит от способа перебора вершин.

*Пример:*

|  |  |
| --- | --- |
|  | последовательность раскраски:  (v1, v2, v6, v3, v5, v4) |
|  | последовательность раскраски:  (v1, v2,v4, v3, v5, v6) |

*Пример:*

|  |  |
| --- | --- |
|  | Наибольшее − первыми, последовательность раскраски:  (v4, v1, v2, v3, v5, v6) |
|  | Наименьшее − последними,последовательность раскраски:  (v4, v3, v5, v2, v1, v6) |

### 4.4.3. Практическое применение раскраски графов

Разнообразные задачи, возникающие при планировании производства, составлении графиков осмотра, хранении и транспортировке товаров и т. д., могут быть представлены часто как задачи теории графов, тесно связанные с задачей раскраски. Графы, рассматриваемые в этой части, являются неориентированными и не имеют петель.

**Планирование.** Раскраска вершин моделирует многие проблемы планирования. В своей простейшей постановке заданный набор работ должен быть распределен по временным отрезкам, каждая такая работа занимает один отрезок. Они могут быть выполнены в любом порядке, но две работы могут конфликтовать в том смысле, что не могут быть выполнены одновременно, так как, например, используют общие ресурсы. Соответствующий граф содержит вершину для каждой из работ и ребро для каждой конфликтующий пары. Хроматическое число построенного графа – это минимальное время выполнения всех работ без конфликтов.

Детали проблемы планирования определяют структуру графа. Например, когда идет распределение самолетов по рейсам, результирующий граф конфликтов является интервальным графом, так что проблема раскраски может быть решена эффективно. При распределении радиочастот получается граф единичных кругов конфликтов, и для такой задачи существует 3-аппроксимационный алгоритм.

**Теория расписаний.** В задачах теории расписаний осмотры представляются в виде временных интервалов. Каждому осмотру можно сопоставить вершину некоторого графа, причем две любые вершины графа будут соединены ребром лишь тогда, когда соответствующие им осмотры нельзя осуществлять одновременно. Требуется составить такой график осмотра, который связан с наименьшими временными затратами (с учетом приведенных выше ограничений на «совместимость» осмотров). Эта задача эквивалентна задаче о раскраске вершин графа с использованием наименьшего числа цветов. Хроматическое число графа как раз и соответствует осмотру, требующему наименьших временных затрат.

Распределение ресурсов. Пусть для выполнения каких-то n работ надо распределить m имеющихся в наличии ресурсов. Считаем, что каждая из работ выполняется за некоторый (одинаковый для всех работ) промежуток времени и что для выполнения i-й работы требуется подмножество ресурсов Si.

Построим граф G: каждой работе соответствует определенная вершина графа, а ребро (xi, xj) существует в графе тогда и только тогда, когда для выполнения i-й и j-й работ требуется хотя бы один общий ресурс, т. е. когда Si∩Sj≠Ø. Это означает, что i-я и j-я работы не могут выполняться одновременно. Раскраска графа G определяет тогда некоторое распределение ресурсов (по выполняемым работам), причем такое, что работы, соответствующие вершинам одного цвета, выполняются одновременно. Наилучшее использование ресурсов (т. е. выполнение всех nработ за наименьшее время) достигается при оптимальной раскраске вершин графа G.

**Распределение регистров.** Компилятор – это компьютерная программа, которая переводит один компьютерный язык в другой. Для улучшения времени выполнения результирующего кода одной из техник компиляторной оптимизации, является распределение регистров, в которой наиболее часто используемые переменные компилируемой программы хранятся в быстродействующих регистрах процессора. В идеальном случае переменные хранятся в регистрах так, что они все находятся в регистрах во время их использования.

Стандартный подход к этой задаче состоит в сведении ее к задаче раскраски графов. Компилятор строит интерференционный граф, где вершины соответствуют регистрам, а грань соединяет две из них, если они нужны в один и тот же момент времени. Если этот граф k-хроматический, то переменные могут храниться в k-регистрах.

**Цифровые водяные знаки.** Технология цифровых водяных знаков позволяет вместе с данными (будь то медиафайлы, исполняемые файлы и прочие) передать некое скрытое сообщение («водяной знак»). Такое скрытое сообщение может быть применено в защите авторских прав для идентификации владельца данных.

Это важно, например, для установления источника их распространения нелегальным образом. Или же для подтверждения прав на данные, например, программное обеспечение систем на кристалле (system-on-chip).

Сообщение можно закодировать в том числе и в способе распределения регистров. Одну из таких техник предложили Qu и Potkonjak (поэтому ее иногда называют QP-алгоритмом).

Состоит она вот в чем: пусть G – граф несовместимости (интерференционный граф) распределения регистров процессора, о котором говорилось выше. Его раскраска используется в программе, причем, используется так, что допустимо поменять содержимое графа с небольшим увеличением его хроматического числа. Оказывается возможно закодировать послание в программном продукте с помощью раскраски графа G, то есть распределения регистров. Извлечь это сообщение можно путем сравнения распределения регистров с исходной раскраской.

Задача раскраски графов была поставлена во многих других областях применения, включая сопоставление с образцом.

Решение головоломки Судоку можно рассматривать как завершение раскраски 9 цветами заданного графа из 81 вершины.

Естественно, что круг применения не ограничен приведенными примерами.

## 4.5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определение эйлерова цикла. Привести пример эйлерова графа.
2. Алгоритм Флери построения эйлерова цикла.
3. Определение гамильтонова цикла. Привести пример графа, удовлетворяющего достаточным условиям гамильтоновости.
4. Идея алгоритма Робертса-Флореса поиска гамильтонова цикла.
5. Задача коммивояжера и задача китайского почтальона.
6. Какой граф называется плоским, планарным?
7. Сформулировать теорему Эйлера для плоского графа, критерии планарности.
8. Дать определение жордановой кривой, теоремы Жордана, грани плоского графа.
9. В чем состоит задача раскраски графа?
10. Что такое правильная раскраска графа?
11. Сформулировать определение хроматического числа, хроматического индекса.
12. Сформулировать теорему Кенига и следствия из нее.

# 5. ОРИЕНТИРОВАННЫЕ ГРАФЫ

Пусть имеется непустое множество V (V ≠ ∅) и пусть V2 − его декартов квадрат, тогда **орграфом** называется пара множеств G = (V, A), где A ⊆ V2.

Элементы множества V называются **вершинами**, элементы множества A− **дугами** орграфа.

**Дуга** (***u,v***) − это упорядоченная пара вершин ***u*** и ***v***, где ***u*** − начало дуги, ***v*** − конец дуги:

## 5.1. ОСНОВНАЯ ТЕРМИНОЛОГИЯ И СПОСОБЫ ЗАДАНИЯ ОРГРАФА

Матричные способы задания орграфа.

Пусть задан орграф G = (V, A), |V| = p, |A| = q.

1. **Матрица смежности** (непосредственной достижимости):

A = ||aij||, i,j = ,

где



Матрица не является симметричной.

1. **Матрица** **инцидентности**:

B = ||bij||, i = , j = ,

где



Неориентированный граф, полученный из орграфа G в результате снятия ориентации дуг орграфа, называется **основанием орграфа**.

**Обратный граф ** − это граф, заданный на том же множестве вершин, у которого все дуги переориентированы, т. е.: ∃(i,j)∈A (j, i) ∈.

*Пример*:

|  |  |  |
| --- | --- | --- |
| Орграф | Основание | Обратный граф |
|  |  |  |

Орграф G1 называется **симметричным**, если для любой дуги (i,j)∈A существует дуга (j,i)∈A.

**Турниром** называется орграф, основание которого − полный граф.

**Полустепень исхода** вершины v орграфа G есть число дуг, исходящих из этой вершины:

deg  v = v = |{(u, v)|, v, u ∈ V, (v, u) ∈ A}|.

**Полустепень захода** вершины v орграфа G есть число дуг, входящих в данную вершину.

deg  v = v = |{u, v)|v, u ∈ V, (u, v) ∈ A}|.

**Степенью** вершины v орграфа G называется:

deg v = v + v.

**Аналог леммы о рукопожатии для орграфа:** сумма полустепеней исхода всех вершины орграфа G равна сумме полустепеней захода всех его вершин и равна числу дуг орграфа G:

.

## 5.2. МАРШРУТЫ, СВЯЗНОСТЬ И ОБХОДЫ В ОРГРАФАХ

**Ориентированным маршрутом** М = (v0, а1, v1, а2, … , аn, vn), где ai = (vi – 1, vi), i,j = называется конечная чередующаяся последовательность вершин и дуг орграфа G такая, что каждая дуга исходит из предыдущей вершины маршрута и заходит в последующую:

**Цепь** − это ориентированный маршрут без повторяющихся дуг.

**Путь** − это цепь без повторяющихся вершин (аналог простой цепи для неорграфа).

**Циклический маршрут** **–** этозамкнутая цепь.

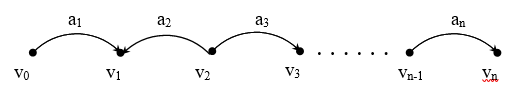
**Контур** − это замкнутый путь.

**Длина маршрута** − количество дуг, образующих данный маршрут, причем каждая дуга считается столько раз, сколько раз она входит в маршрут.

Если существует (u, v)-маршрут, то вершина v − **достижима** из вершины u, и вершина u контрдостижима для вершины v, если существует (v, u)−маршрут.

Если вершины u и v достижимы друг для друга, то они называются **взаимодостижимыми.**

**Полумаршрутом** в орграфе G называется маршрут его основания:



аi = (vi – 1, vi), или аi = (vi, vi – 1) i,j = .

Орграф называется **сильным (сильно связанным)**, если любые две его вершины достижимы друг для друга.

Орграф называется **односторонним (односторонне связанным)**, если для каждой пары его вершин, по крайней мере, одна достижима из другой.

Орграф называется **слабым (слабо связанным)**, если любые две его вершины соединены полумаршрутом.

*Пример:*

|  |  |  |
| --- | --- | --- |
| Сильный граф | Односторонний граф | Слабый граф |
|  |  |  |

Орграф называется **несвязным**, если его основание не связно.

Замечание. Определения подграфов и порожденных подграфов для ориентированных графов аналогичны соответствующим определениям для графов неориентированных.

Необходимые и достаточные условия связности орграфа приводятся в следующих теоремах.

**Теорема 1.** Орграф является **сильносвязным**, тогда и только тогда, когда в нем существует остовный циклический маршрут **(остовным маршрутом** называется маршрут, проходящий через все вершины графа).

**Теорема 2.** Орграф является односторонним тогда и только тогда, когда в нем существует остовный маршрут.

**Теорема 3.** Орграф является слабым тогда и только тогда, когда в нем существует остовный полумаршрут.

Определения эйлеровых и гамильтоновых циклов орграфа сходны с аналогичными определениями для неориентированного графа.

Орцепь, содержащая каждую дугу орграфа, называется **эйлеровой цепью.**

Связный орграф называется **эйлеровым графом**, если он содержит замкнутую эйлерову цепь.

**Теорема.** Для связного орграфа G следующие утверждения эквивалентны:

1. орграф G - эйлеров;
2. для любой вершины v орграфа G справедливо следующее равенство:

v = v.

1. орграф G является объединением контуров, попарно не имеющих общих дуг.

Контур или замкнутый ормаршрут является **гамильтоновым**, если он содержит все вершины орграфа, и сам граф является гамильтоновым, если он содержит гамильтонов контур.

**Теорема (достаточное условие гамильтоновости).**

Пусть G – сильный орграф порядка *p* > 1 (без петель и кратных дуг). Тогда, если для любой пары его смежных вершин выполняется равенство

deg u + deg v ≤ 2p – 1,

то в графе G есть гамильтонов контур.

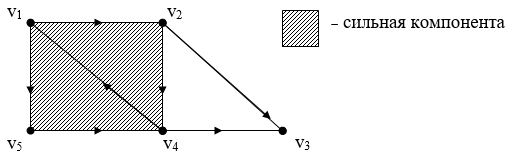
## 5.3. ТИПЫ КОМПОНЕНТ СВЯЗНОСТИ. КОНДЕНСАЦИЯ, БАЗА И АНТИБАЗА

**Сильная компонента** − это максимальный (по включению) сильно связанный подграф исходного графа.

**Односторонняя компонента** − это максимальный (по включению) односторонне связанный подграф исходного графа.

**Слабая компонента** – это максимальный (по включению) слабый подграф исходного графа.

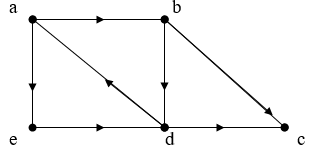
*Пример*:



Поскольку отношение взаимной достижимости двух вершин орграфа является отношением эквивалентности (рефлексивно, симметрично и транзитивно), то оно разбивает множество всех вершин на классы эквивалентности, т. е. такие подмножества вершин орграфа, что подграфы, порожденные этими подмножествами вершин, являются сильными компонентами.

**Конденсацией** называется такой орграф G\*, вершины которого соответствуют сильным компонентам S1,S2,…, Sk орграфа G, и пара (Si, Sj) в G\* является дугой тогда и только тогда, когда в орграфе G существует хотя бы одна дуга, начало которой принадлежит компоненте Si, конец - Sj.

*Пример*: построить конденсацию для орграфа G:



Сильные компоненты графа G: S1 = {a, e, d, b}, S2 = {c}, конденсация G\*:



Для построения конденсации существует следующий алгоритм.

**Алгоритм построения конденсации.**

1. Построить матрицу достижимости R = ||rij||, *i*, *j* = ,

где



2. Построить матрицу контрдостижимости Q = ||*qij*||, *i*, *j* = ,

где



Замечание. Матрица контрдостижимости Q может быть получена с помощью транспонирования матрицы R: Q = RТ.

3. Построить матрицу взаимной достижимости S = ||sij||, i, j = ,

где



Матрица S получается путем поэлементного умножения матриц R и Q, т.е.

S = R \* Q,

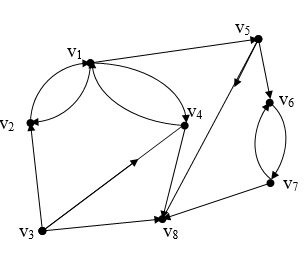
где «\*» − оператор поэлементного умножения матриц R и Q:

sij = rij \* qij.

4. Определить сильные компоненты: привести матрицу S к блочно-диагональному виду (перестановкой ее строк и столбцов). Вершины, составляющие каждый блок - сильные компоненты.

5. Каждой сильной компоненте поставить в соответствие одну вершину конденсации. Построить множество дуг (по определению конденсации). Полученный орграф – конденсация исходного графа.

*Пример:* построить конденсацию для графа G



1. Матрица достижимости R имеет вид

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

2. Получим матрицу контрдостижимости Q транспонированием матрицы R

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

3. После поэлементного умножения матриц R и Q получим матрицу S

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

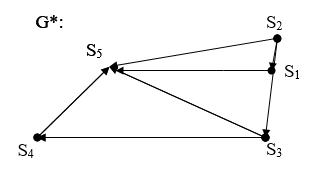
4. Приводим матрицу S к блочно-диагональному виду перестановкой строк и столбцов 3 и 4

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | 1 | 2 | 4 | 3 | 5 | 6 | 7 | 8 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Вершины, составляющие каждый блок S – сильные компоненты. Таким образом сильными компонентами S являются

S1 = {1, 2, 4}, S2 = {3}, S3 = {5}, S4 = {6,7}, S5 = {8}.

5. Каждая полученная сильная компонента соответствует одной вершине конденсации, после построения множества дуг для исходного графа G получим конденсацию G\*



**База** орграфа G − наименьшее (относительно включения) подмножество вершин B (B ⊂ V), удовлетворяющее условию: любая вершина v ∈ V\B достижима из какой-либо вершины w ∈ B.

**Антибаза** орграфа G − наименьшее (относительно включения) подмножество вершин B' (B'⊂ V) такое, что любая вершина v∈ B' достижима из какой- либо вершины w ∈ V\ B'.

В орграфе всегда существует база. Вершины, полустепени захода которых равны нулю, принадлежат базе.

База бесконтурного графа состоит только из вершин, полустепени захода которых равны нулю.

Сильные компоненты орграфа, в которые не входят дуги из других сильных компонент, соответствуют вершинам с нулевыми полустепенями захода в орграфе G\* и являются **базовыми компонентами**.

**Алгоритм построения базы**.

1. Построить конденсацию G\* орграфа G.
2. Выделить в ней базовые компоненты; им в конденсации соответствует вершины с нулевой полустепенью захода.
3. Из каждой базовой компоненты произвольно выбрать по одной вершине.

Замечание. База определяется не единственным образом, исключая ациклический граф.

В рассмотренном выше примере база: B={v3}.

**Алгоритм построения антибазы**.

1. Построить конденсацию G\* орграфа G.
2. Выделить в конденсации вершины с нулевой полустепенью исхода.
3. Из соответствующих сильных компонент выбрать произвольно по одной вершине.

В рассмотренном выше примере антибаза: B'={v8}.

## 5.4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дать определение матрицы смежности и матрицы инцидентности орграфа.
2. Обратный орграф, основание, турнир.
3. Степени вершин орграфа.
4. Дать определение следующим понятиям: ормаршрут, цепь, путь, полумаршрут, полуцепь, полупуть, замкнутый маршрут, цикл, контур.
5. Матрицы достижимости, контрдостижимости и взаимодостижимости.
6. Типы связности орграфа, сильные компоненты.
7. Конденсация, база, антибаза. Алгоритмы построения.
8. Обходы орграфа.

# ПРИЛОЖЕНИЕ А

Листинг программы, реализующей алгоритм Флойда, на С++

**#include** <iostream.h>

**#define** TRUE 1

**#define** FALSE 0

**#define** MaxNodes 5 **//Количество вершин.**

**//Описание типа узла стека.**

**typedef** **struct** Zveno \*svqz;

**typedef** **struct** Zveno

{

**int** Element;

svqz Sled;

};

**class** Spisok

{

**private**:

**int** Mas[MaxNodes][MaxNodes]; **//Матрица весов дуг.**

**int** DD[MaxNodes][MaxNodes]; **//Матрица расстояний.**

**int** SS[MaxNodes][MaxNodes]; **//Матрица последовательных узлов.**

svqz Stack; **//Указатель на рабочий стек.**

**void** UDALENIE (svqz \*, **int** \*);

**void** W\_S (svqz \*, **int**);

**void** Small\_Put (int,int);

**public**:

Spisok() {Stack = **NULL**;}

**void** Vvod\_Ves();

**void** Reshenie ();

};

**void** main()

{

Spisok A;

A.Vvod\_Ves();

A.Reshenie();

}

**void** Spisok::Small\_Put (**int** one, **int** two)

**//Нахождение кратчайшего пути.**

{

svqz St=NULL; **//Указатель на вспомогательный стек.**

svqz UkZv;

**int** Flag=FALSE; **//Флаг построения кратчайшего пути.**

**int** elem1,elem2,k;

**//Помещение в стек конечной и начальной вершин.**

W\_S (&Stack,two);

W\_S (&Stack,one);

**while** (!Flag)

{

**//Извлекли верхних два элемента.**

UDALENIE(&Stack,&elem1);

UDALENIE(&Stack,&elem2);

**if** (SS[elem1][elem2]==elem2) **//Если есть путь...**

**if** (elem2==two) **//и это конечный узел...**

{

Flag = TRUE; **//то кратчайший путь найден.**

W\_S (&St,elem1);

W\_S (&St,elem2);

}

**else** **//и это не конечный узел...**

{

W\_S (&St,elem1); **//В вспомогательный стек.**

W\_S (&Stack,elem2); **//Обратно в рабочий стек.**

}

**else** **//Если пути нет.**

{

W\_S (&Stack,elem2); **//Обратно в рабочий стек.**

k = SS[elem1][elem2];

W\_S (&Stack,k); **//Запомнить промежуточную вершину.**

W\_S (&Stack,elem1); **//Обратно в рабочий стек.**

}

}

UkZv = St;

**while** ( UkZv != **NULL** )

{ cout << (UkZv->Element+1) << " ";

UkZv = UkZv->Sled; }

cout << endl;

}

**void** Spisok::W\_S (svqz \*stk, **int** Elem)

**//Помещение Elem в стек stk.**

{

svqz q=new (Zveno);

(\*q).Element = Elem;

(\*q).Sled = \*stk; \*stk = q;

}

**void** Spisok::UDALENIE (svqz \*stk, **int** \*Klad)

**//Удаление звена из стека, заданного указателем \*stk.**

**//Значение информационного поля удаляемого звена сохраня-**

**//ется в параметре Klad.**

{

svqz q;

**if** (\*stk==NULL) cout<<"Попытка выбора из пустого стека!\n";

**else**

{ \*Klad = (\*\*stk).Element;

q = \*stk; \*stk = (\*\*stk).Sled; **delete** q; }

}

**void** Spisok::Vvod\_Ves()

**//Ввод матрицы весов дуг заданного графа.**

{

cout << "Вводите элементы матрицы весов дуг по строкам:\n";

**for** (**int** i=0;i<MaxNodes;i++)

**for** (**int** j=0;j<MaxNodes;j++)

{

cout << "Введите Mas[" << (i+1) << "," << (j+1) << "]: ";

cin >> Mas[i][j];

}

}

**void** Spisok::Reshenie()

{

**int** one,two;

**int** i,j;

**//Инициализация.**

**for** (i=0;i<MaxNodes;i++)

**for** (j=0;j<MaxNodes;j++)

{

**if** (Mas[i][j]>0) SS[i][j]=j;

**else** SS[i][j]=0;

DD[i][j]=Mas[i][j];

}

cout << "\nНачальная вершина: ";

cin >> one; one--;

cout << "Конечная вершина: ";

cin >> two; two--;

**int** ved=0;

**while** (ved<MaxNodes)

{

**for** (i=0;i<MaxNodes;i++)

**for** (j=0;j<MaxNodes;j++)

**if** (i!=j && i!=ved && j!=ved &&

DD[i][ved]>0 && DD[ved][j]>0)

**if** (DD[i][ved]+DD[ved][j]<DD[i][j] || DD[i][j]==0)

{

DD[i][j]=DD[i][ved]+DD[ved][j];

SS[i][j]=ved;

}

ved++;

}

i=one;

**if** (SS[i][two]!=two && SS[i][two]!=0)

**while** (SS[i][two]!=two)

{

j=SS[i][two];

**while** (SS[i][j]!=j) j=SS[i][j];

i=j;

}

cout << "\nКратчайший путь (в обратном порядке): ";

Small\_Put (one, two);

cout << "Длина минимального пути между этими вершинами: " << DD[one][two];

}

# ПРИЛОЖЕНИЕ Б

Алгоритмы генерации вариантов графов

Входные данные:

* статический локальный параметр – множество S

*S* = <фамилия> <имя> <отчество>;

* таблично заданная функция *n*(*char*), которая ставит в соответствие букве русского алфавита *сhar* ее порядковый номер.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| а | б | в | г | д | е | ё | ж | з | и | й | к | л | м | н | о | п |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |  |
| р | с | т | у | ф | х | ц | ч | ш | щ | ъ | ы | ь | э | ю | я |  |

Параметры алгоритма:

* число *p*, задающее количество вершин в генерируемом графе;
* множество натуральных чисел *Х* для построения матрицы смежности *А*.

**Алгоритм генерации неориентированного графа GV(p, X).**

1. Вычеркнуть из *S* все повторные вхождения букв.
2. Построить *Y* = ║*yij*║, *i*, *j* = 1, 2, …, *p*, где *yij* = │*n*(*Si*)-*n*(*Sj*)│.
3. Построить *A* = ║*aij*║, *i*, *j* = 1, 2, …, *p*:

если *i* = *j*, то *aij* = 0, иначе:



1. Для каждой изолированной (доминирующей) вершины добавить (удалить) одно ребро. Добавляемое (удаляемое) ребро связывает текущую вершину со следующей (по номеру). Для последней вершины следующей считать первую вершину.
2. Ребра графа взвесить соответствующими элементами матрицы Y.

**Алгоритм генерации орграфа GV1(p, X).**

1. Вычеркнуть из *S* все повторные вхождения букв.
2. Построить *Y* = ║*yij*║, *i*, *j* = 1, 2, …, *p*, где верхний треугольник матрицы заполняется по формуле *yij* = │*n*(*Si*)-*n*(*Sj*)│, а нижний – *yij* = *n*(*Si*)+*n*(*Sj*).
3. Построить *A* = ║*aij*║, *i*, *j* = 1, 2, …, *p*:

если *i* = *j*, то *aij* = 0, иначе:



***Пример 1:*** используя алгоритм генерации варианта GV(5,{2,3}), построить неориентированный граф G по множеству *S*, которое получено по ФИО

<Юфа> <Елена> <Яковлевна>.

Получим по ФИО множество S=<ю,ф,а,е,л,н,я,к,о,в>.

Берем первые 5 его элементов и найдем их порядковые номера в алфавите:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **i** | **1** | **2** | **3** | **4** | **5** |
| **Si** | ю | ф | а | е | л |
| **n(Si)** | 32 | 22 | 1 | 6 | 13 |

Построим вспомогательную матрицу Y, вычислив │n(Si)-n(Sj)│:

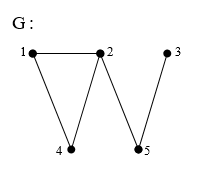
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Y | 32 | 22 | 1 | 6 | 13 |
| 32 | 0 | 10 | 31 | 26 | 29 |
| 22 | 10 | 0 | 21 | 16 | 9 |
| 1 | 31 | 21 | 0 | 5 | 12 |
| 6 | 26 | 16 | 5 | 0 | 7 |
| 13 | 29 | 9 | 12 | 7 | 0 |

Для получения матрицы смежности A обнуляем те элементы *yij*, которые не делятся на 2 или 3, остальные приравниваем к 1. В итоге А имеет вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A1 | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 0 | 1 |
| 4 | 1 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 1 | 0 | 0 |

Так как вершина 2 доминирующая, то ребро (2,3) удалено.

Используя матрицу смежности A, отобразим граф G графически:



***Пример 2:*** используя алгоритм генерации варианта GV(7,{2,3}), построить неориентированный граф G по множеству *S*, которое получено по ФИО

<Юфа> <Елена> <Яковлевна>.

Получим по ФИО множество S=<ю,ф,а,е,л,н,я,к,о,в>.

Берем первые 7 его элементов и найдем их порядковые номера в алфавите:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **Si** | ю | ф | а | е | л | н | я |
| **n(Si)** | 32 | 22 | 1 | 6 | 13 | 15 | 33 |

Построим вспомогательную матрицу Y, вычислив │n(Si)-n(Sj)│:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Y | 32 | 22 | 1 | 6 | 13 | 15 | 33 |
| 32 | 0 | 10 | 31 | 26 | 29 | 17 | 1 |
| 22 | 10 | 0 | 21 | 16 | 9 | 7 | 11 |
| 1 | 31 | 21 | 0 | 5 | 12 | 14 | 32 |
| 6 | 26 | 16 | 5 | 0 | 7 | 9 | 27 |
| 13 | 29 | 9 | 12 | 7 | 0 | 2 | 20 |
| 15 | 17 | 7 | 14 | 9 | 2 | 0 | 18 |
| 33 | 1 | 11 | 32 | 27 | 20 | 18 | 0 |

Для получения матрицы смежности A обнуляем те элементы *yij*, которые не делятся на 2 или 3, остальные приравниваем к 1. В итоге А имеет вид:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 4 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 7 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

Используя матрицу смежности A, отобразим граф G графически:

|  |  |
| --- | --- |
| Граф G без учета веса ребер | Граф G со взвешенными ребрами |
|  |  |

***Пример 3:*** используя алгоритм генерации варианта GV(13,{6,7}), построить неориентированный граф G по множеству S=<ю,ф,а,е,л,н,я,к,о,в>.

Поскольку мощность множества S меньше требуемого количества вершин в графе, то дополним его до 13 произвольными символами, не входящими в S, например: {ы, р, м}.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| **Si** | ю | ф | а | е | л | н | я | к | о | в | ы | р | м |
| **n(Si)** | 32 | 22 | 1 | 6 | 13 | 15 | 33 | 12 | 16 | 3 | 29 | 18 | 14 |

Строим вспомогательную матрицу Y:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Y** | 32 | 22 | 1 | 6 | 13 | 15 | 33 | 12 | 16 | 3 | 29 | 18 | 14 |
| 32 | 0 | 10 | 31 | 26 | 19 | 17 | 1 | 20 | 16 | 29 | 3 | 14 | 18 |
| 22 | 10 | 0 | 21 | 16 | 9 | 7 | 11 | 10 | 6 | 19 | 7 | 4 | 8 |
| 1 | 31 | 21 | 0 | 5 | 12 | 14 | 32 | 11 | 15 | 2 | 28 | 17 | 13 |
| 6 | 26 | 16 | 5 | 0 | 7 | 9 | 27 | 6 | 10 | 3 | 23 | 12 | 8 |
| 13 | 19 | 9 | 12 | 7 | 0 | 2 | 20 | 1 | 3 | 10 | 16 | 5 | 1 |
| 15 | 17 | 7 | 14 | 9 | 2 | 0 | 18 | 2 | 1 | 12 | 14 | 3 | 1 |
| 33 | 1 | 11 | 32 | 27 | 20 | 18 | 0 | 21 | 17 | 30 | 4 | 15 | 19 |
| 12 | 20 | 10 | 11 | 6 | 1 | 2 | 21 | 0 | 4 | 9 | 17 | 6 | 2 |
| 16 | 16 | 6 | 15 | 10 | 3 | 1 | 17 | 4 | 0 | 13 | 13 | 2 | 2 |
| 3 | 29 | 19 | 2 | 3 | 10 | 12 | 30 | 9 | 13 | 0 | 26 | 15 | 11 |
| 29 | 3 | 7 | 28 | 23 | 16 | 14 | 4 | 17 | 13 | 26 | 0 | 11 | 15 |
| 18 | 14 | 4 | 17 | 12 | 5 | 3 | 15 | 6 | 2 | 15 | 11 | 0 | 4 |
| 14 | 18 | 8 | 13 | 8 | 1 | 1 | 19 | 2 | 2 | 11 | 15 | 4 | 0 |

Для получения матрицы смежности A обнуляем те элементы *yij*, которые не делятся на 6 или 7, остальные приравниваем к 1. В итоге А имеет вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Используя матрицу смежности A, отобразим граф графически:

|  |  |
| --- | --- |
| Граф G без учета веса ребер | Граф G со взвешенными ребрами |
|  |  |

***Пример 4:*** используя алгоритм генерации варианта GV(9,{6,7}), построить ориентированный граф G по множеству S=<ю,ф,а,е,л,н,я,к,о,в>.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Si** | ю | ф | а | е | л | н | я | к | о |
| **n(Si)** | 32 | 22 | 1 | 6 | 13 | 15 | 33 | 12 | 16 |

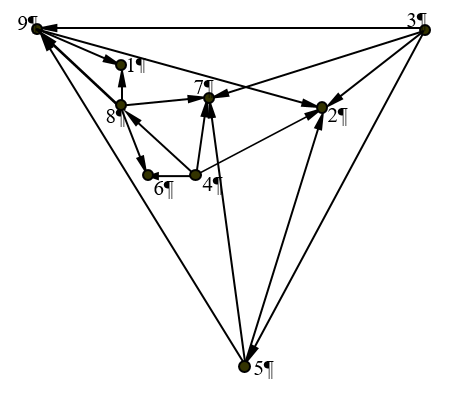
Строим вспомогательную матрицу Y, верхний треугольник матрицы заполняется по формуле *yij* = │*n*(*Si*)-*n*(*Sj*)│, а нижний – *yij* = *n*(*Si*)+*n*(*Sj*):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Y** | 32 | 22 | 1 | 6 | 13 | 15 | 33 | 12 | 16 |
| 32 | 0 | 10 | 31 | 26 | 19 | 17 | 1 | 20 | 16 |
| 22 | 54 | 0 | 21 | 16 | 9 | 7 | 11 | 10 | 6 |
| 1 | 33 | 23 | 0 | 5 | 12 | 14 | 32 | 11 | 15 |
| 6 | 38 | 28 | 7 | 0 | 7 | 9 | 27 | 6 | 10 |
| 13 | 45 | 35 | 14 | 19 | 0 | 2 | 20 | 1 | 3 |
| 15 | 47 | 37 | 16 | 21 | 28 | 0 | 18 | 3 | 1 |
| 33 | 65 | 55 | 34 | 39 | 46 | 48 | 0 | 21 | 17 |
| 12 | 44 | 34 | 13 | 18 | 25 | 27 | 45 | 0 | 4 |
| 16 | 48 | 38 | 17 | 22 | 29 | 31 | 49 | 28 | 0 |

Для получения матрицы смежности A обнуляем те элементы *yij*, которые не делятся на 6 или 7, остальные приравниваем к 1. В итоге А имеет вид:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 9 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Используя матрицу смежности A, отобразим граф графически:



# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ №1

***Тема:* «**Подграфы. Изоморфизм».

***Цель работы****:* изучение основных понятий теории графов, приобретение практических навыков определения изоморфизмов графа; построение подграфов, независимых множеств и клик.

***Содержание работы:***

1. Используя алгоритм генерации варианта GV, построить неориентированный граф G: GV(7,{2,3}).
2. Описать граф матрицей смежности, матрицей инцидентности. Изобразить графически граф G и его дополнение G̅. Построить произвольный остовный подграф и подграф, порожденный множеством вершин {1,2,5,6,7}.
3. Построить все помеченные 5-графы, изоморфно вложимые в граф G. Среди них определить классы изоморфных графов, построив биекцию их вершин. Для каждого класса изоморфных графов привести рисунок абстрактного графа.
4. Найти все максимальные и наибольшие клики данного графа. Определить плотность графа G.
5. Найти все максимальные и наибольшие независимые множества исходного графа. Определить число независимости.
6. Найти полный двудольный подграф Kp,q, изоморфно вложимый в граф G, с максимальным количеством вершин p+q (p≠1).
7. Найти звезду K1,n, изоморфно вложимую в граф G, с максимальным значением n.

ПРИМЕР ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 1

**1. *Используя алгоритм генерации варианта* GV*, построить неориентированный граф* G*:* GV(7,{2,3}) .**

Построение графа G по S=<ю,ф,а,е,л,н,я,к,о,в> приведено в примере 2 приложения Б. Опишем полученный неориентированный граф G=(V,E), используя способ перечисления.

V={1, 2, 3, 4, 5, 6, 7} – множество вершин графа;

E={(1,2), (1,4), (2,3), (2,4), (2,5), (3,5), (3,6), (3,7), (4,6), (4,7), (5,6), (5,7), (6,7)} – множество ребер графа.

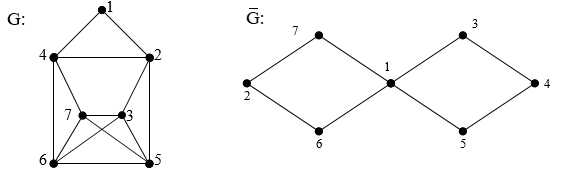
**2. *Описать граф матрицей смежности, матрицей инцидентности. Изобразить графически граф G и его дополнение G̅. Построить произвольный остовный подграф и подграф, порожденный вершинами* {1,2,5,6,7}.**

Матрица смежности А получена с помощью алгоритма GV(7,{2,3}) и имеет вид:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 4 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 7 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

Матрица инцидентности

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | (1,2) | (1,4) | (2,3) | (2,4) | (2,5) | (3,5) | (3,6) | (3,7) | (4,6) | (4,7) | (5,6) | (5,7) | (6,7) |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |



|  |  |
| --- | --- |
|  | Остовный подграф G1=(V,E1), где E1={(1,2), (1,4), (2,5), (3,5), (5,6), (5,7), (6,7)} |
|  | Подграф G2, порожденный вершинами {1, 2, 5, 6, 7} |

**3. *Построить все помеченные 5-графы, изоморфно вложимые в граф* G*. Среди них определить классы изоморфных графов, построив биекцию их вершин. Для каждого класса изоморфных графов привести рисунок абстрактного графа.***

Количество способов, которыми можно выбрать 5 вершин из 7 вершин без учета порядка их следования, определяется сочетаниями

.

5-графы, изоморфно вложимые в граф G, представлены в следующей таблице:

|  |  |  |
| --- | --- | --- |
| Gi | Множество вершин Vi \* | Графическое представление Gi |
| G1 | V1 = {1, 2, 3, 4, 5, 6, 7} |  |
| G2 | V2 = {1, 2, 3, 4, 5, 6, 7} |  |
| G3 | V3 = {1, 2, 3, 4, 5, 6, 7} |  |
| G4 | V4 = {1, 2, 3, 4, 5, 6, 7} |  |
| G5 | V5 = {1, 2, 3, 4, 5, 6, 7} |  |
| G6 | V6 = {1, 2, 3, 4, 5, 6, 7} |  |
| G7 | V7 = {1, 2, 3, 4, 5, 6, 7} |  |
| G8 | V8 = {1, 2, 3, 4, 5, 6, 7} |  |
| G9 | V9 = {1, 2, 3, 4, 5, 6, 7} |  |
| G10 | V10 = {1, 2, 3, 4, 5, 6, 7} |  |
| G11 | V11 = {1, 2, 3, 4, 5, 6, 7} |  |
| G12 | V12 = {1, 2, 3, 4, 5, 6, 7} |  |
| G13 | V13 = {1, 2, 3, 4, 5, 6, 7} |  |
| G14 | V14 = {1, 2, 3, 4, 5, 6, 7} |  |
| G15 | V15 = {1, 2, 3, 4, 5, 6, 7} |  |
| G16 | V16 = {1, 2, 3, 4, 5, 6, 7} |  |
| G17 | V17 = {1, 2, 3, 4, 5, 6, 7} |  |
| G18 | V18 = {1, 2, 3, 4, 5, 6, 7} |  |
| G19 | V19 = {1, 2, 3, 4, 5, 6, 7} |  |
| G20 | V20 = {1, 2, 3, 4, 5, 6, 7} |  |
| G21 | V22 = {1, 2, 3, 4, 5, 6, 7} |  |

\* – желтым выделены номера вершин, не входящие в Vi.

Найдем классы изоморфных графов. Абстрактные графы у изоморфных совпадают. Разобьём всё множество 5-графов на подмножества, содержащие графы с 5, 6, 7 и 8 ребрами.

1) Графы с пятью ребрами: G5, G10, G12, G16.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Биекция | | | | | | Рисунок абстрактного графа |
| **Gi** | **v1** | **v2** | **v3** | **v4** | **v5** |  |
| G5 | 1 | 4 | 6 | 3 | 5 |
| G10 | 1 | 4 | 7 | 3 | 5 |
| G12 | 1 | 2 | 3 | 7 | 6 |
| G16 | 1 | 2 | 5 | 7 | 6 |

2) Графы с шестью ребрами: G1, G2, G3, G4, G7, G8, G9, G13, G14, G17, G19.

Это множество графов содержит 4 подмножества изоморфных друг другу графов: {G1, G13}, {G2, G4, G7, G9}, {G3, G8, G14, G19}, {G17}.

2.1)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Биекция | | | | | | Рисунок абстрактного графа |
| **Gi** | **v1** | **v2** | **v3** | **v4** | **v5** |  |
| G1 | 1 | 2 | 3 | 5 | 4 |
| G13 | 1 | 4 | 7 | 6 | 2 |

2.2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Биекция | | | | | | Рисунок абстрактного графа |
| **Gi** | **v1** | **v2** | **v3** | **v4** | **v5** |  |
| G2 | 1 | 2 | 3 | 6 | 4 |
| G4 | 1 | 2 | 5 | 6 | 4 |
| G7 | 1 | 2 | 3 | 7 | 4 |
| G9 | 1 | 2 | 5 | 7 | 4 |

2.3)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Биекция | | | | | | Рисунок абстрактного графа |
| **Gi** | **v1** | **v2** | **v3** | **v4** | **v5** |  |
| G3 | 1 | 2 | 3 | 6 | 5 |
| G8 | 1 | 2 | 3 | 7 | 5 |
| G14 | 1 | 4 | 6 | 3 | 7 |
| G19 | 1 | 4 | 6 | 5 | 7 |

2.4) Граф G17 не изоморфен ни одному из 5-графов.

3) Графы с семью ребрами: G6, G11, G15, G20.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Биекция | | | | | | Рисунок абстрактного графа |
| **Gi** | **v1** | **v2** | **v3** | **v4** | **v5** |  |
| G6 | 2 | 3 | 5 | 6 | 4 |
| G11 | 2 | 3 | 5 | 7 | 4 |
| G15 | 4 | 6 | 7 | 3 | 2 |
| G20 | 5 | 7 | 6 | 4 | 2 |

4) Графы с восемью ребрами: G18, G21.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Биекция | | | | | | Рисунок абстрактного графа |
| **Gi** | **v1** | **v2** | **v3** | **v4** | **v5** |  |
| G18 | 2 | 5 | 6 | 7 | 3 |
| G21 | 4 | 7 | 3 | 5 | 6 |

**4. *Найти все максимальные и наибольшие клики данного графа. Определить плотность графа* G.**

Выпишем все клики данного графа.

1) Клики двумя вершинами:

K1={1, 2}; K2={1, 4}; K3={2, 3}; K4={2, 4}; K5={2, 5}; K6={3, 5}; K7={3, 6}; K8={3, 7}; K9={4, 6}; K10={4, 7}; K11={5, 6}; K12 ={5, 7}; K13={6, 7}.

2) Клики с тремя вершинами:

K14={1, 2, 4}; K15={2, 3, 5}; K16={3, 5, 6}; K17={3, 6, 7}; K18={4, 6, 7}; K19={5, 6, 7}; K20={3, 5, 7}.

3) Клики с четырьмя вершинами: K21={3, 5, 6, 7}.

Для данного графа клики:

* K14, K15, K18, K21 – максимальные;
* K21 – наибольшая.

Плотность графа β(G)=⎪K21⎪=4.

**5. *Найти все максимальные и наибольшие независимые множества исходного графа. Определить число независимости.***

Независимые множества графа представляют собой клики его дополнения. Очевидно, что G̅ не содержит клики, мощность которых больше 2. Анализируя дополнение G̅, выпишем все независимые множества графа G.

S1={1, 3}; S2={1, 5}; S3={1, 6}; S4={1, 7}; S5={2, 6}; S6={2, 7}; S7={3, 4}; S8={4, 5}.

Для данного графа все его независимые множества являются и максимальными, и наибольшими.

Число независимости графа α(G)=⎪S1⎪=2.

**6. *Найти полный двудольный подграф* Kp,q*, изоморфно вложимый в граф* G, *с максимальным количеством вершин* p+q *(*p≠1).**

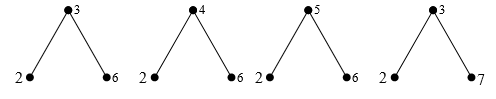
Полные двудольные подграфы, изоморфно вложимые в G:

Kp,q=K2,2; p+q=2+2=4.



**7. *Найти звезду* K1,n*, изоморфно вложимую в граф* G, *с максимальным значением* n.**

Максимальное значение n звезд K1,n, изоморфно вложимых в G, не превышает 2, т.е. K1,n=K1,2; n=2. Отобразим некоторые из них графически:



# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ №2

***Тема:* «**Маршруты и связность».

***Цель работы:*** изучение понятий связных графов и маршрутов, приобретение практических навыков построения матрицы расстояний и нахождения матрицы кратчайших маршрутов.

***Содержание работы:***

1. Используя алгоритм генерации варианта GV, построить неориентированный граф G1: GV(13,{6,7}) и граф G2: GV(7,{2,3}). Ребра графа G2 взвешены соответствующими элементами матрицы Y.
2. Определить, является ли граф G1 связным.
3. Для максимальной компоненты графа G1 выделить:
   1. открытый маршрут, не являющийся цепью;
   2. замкнутый маршрут, не являющийся циклом;
   3. цепь, не являющуюся простой цепью;
   4. простую цепь;
   5. цикл, не являющийся простым циклом;
   6. простой цикл;
   7. определить обхват и окружение;
   8. найти вершинную и реберную связность.
4. Для каждой компоненты графа G1:
   1. построить матрицу расстояний;
   2. определить эксцентриситеты вершин;
   3. найти радиус и диаметр;
   4. найти центр и периферию;
   5. выделить блоки;
   6. найти точки сочленения и мосты.
5. В графе G2:
   1. построить кратчайшие маршруты от произвольной вершины ко всем остальным при помощи алгоритма Дейкстры;
   2. построить кратчайшие маршруты при помощи алгоритма Флойда. При построении вести две матрицы – матрицу маршрутов и матрицу расстояний.

ПРИМЕР ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 2

***1. Используя алгоритм генерации варианта GV, построить неориентированный граф G1: GV(13,{6,7}) и граф G2: GV(7,{2,3}). Ребра графа G2 взвешены соответствующими элементами матрицы Y.***

Построение графа G1 по множеству S=<ю,ф,а,е,л,н,я,к,о,в> приведено в примере 3 приложения Б, взвешенного графа G2 – в примере 2 приложения Б. Опишем полученные графы G1=(V1,E1) и G2=(V2,E2), используя способ перечисления:

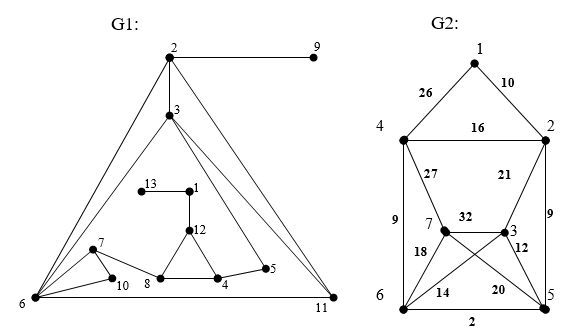
V1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13} – множество вершин графа G1;

E1={(1,12), (1,13), (2,3), (2,6), (2,9), (2,11), (3,5), (3,6), (3,11), (4,5), (4,8), (4,12), (6,7), (6,10), (6,11), (7,8), (7,10), (8,12)} – множество ребер графа G1;

V2={1, 2, 3, 4, 5, 6, 7} – множество вершин графа G2;

E2={(1,2), (1,4), (2,3), (2,4), (2,5), (3,5), (3,6), (3,7), (4,6), (4,7), (5,6), (5,7), (6,7)} – множество ребер графа G2.

Приведем графические изображения графов G1 и G2:



**2. *Определить, является ли граф* G1 *связным.***

Граф G1 является связным, поскольку между любыми двумя его вершинами можно построить маршрут. Он содержит только одну компоненту связности, которая совпадает с исходным графом.

**3. *Для максимальной компоненты графа* G1 *выделить:***

* 1. ***открытый маршрут, не являющийся цепью;***
  2. ***замкнутый маршрут, не являющийся циклом;***
  3. ***цепь;***
  4. ***простую цепь;***
  5. ***цикл;***
  6. ***простой цикл;***
  7. ***определить обхват и окружение;***
  8. ***найти вершинную и реберную связность.***

В графе G1:

1. выделим открытый маршрут, не являющийся цепью:

M1={2,**3,5,3**,6,7}=M(2,7);

1. выделим замкнутый маршрут, не являющийся циклом:

M2={**1,12**,4,5,3,6,7,8,**12,1**}=M(1,1);

1. выделим цепь:

M3={11,**2**,6,3,**2**,9}=M(11,9);

1. выделим простую цепь:

M4={3,6,7,8,12,1,13}=M(3,13);

1. выделим цикл:

M5={10,7,**6**,3,2,**6**,10}=M(10,10);

1. выделим простой цикл:

M6={2,3,5,4,8,7,6.11,2}=M(2,2);

1. определим обхват и окружение:

g(G)=|M3|-1=3; С(G)=|M13|-1=10;

1. найдем вершинную и реберную связность:

- если удалить вершину 2, или вершину 12, или вершину 1, то граф станет не связным, следовательно, вершинная связность данного графа χ(G)=1;

- если удалить ребро (2,9), или ребро (1,12), или ребро (1,13), то граф станет не связным, следовательно, реберная связность данного графа λ(G)=1.

**4. *Для каждой компоненты графа* G1:**

1. ***построить матрицу расстояний;***
2. ***определить эксцентриситеты вершин;***
3. ***найти радиус и диаметр;***
4. ***найти центр и периферию;***
5. ***выделить блоки;***
6. ***найти точки сочленения и мосты.***

Так как граф G1 связный, то он имеет только одну компоненту связности, совпадающую с самим графом.

1. Для нахождения метрических характеристик графа G1 построим для него матрицу расстояний D, т.е. для каждой пары его вершин vi и vj найдем длину кратчайшей простой цепи, соединяющей эти вершины.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | e(vi) |
| 1 | 0 | 5 | 4 | 2 | 3 | 4 | 3 | 2 | 6 | 4 | 5 | 1 | 1 | 6 |
| 2 | 5 | 0 | 1 | 3 | 2 | 1 | 2 | 3 | 1 | 2 | 1 | 4 | 6 | 6 |
| 3 | 4 | 1 | 0 | 2 | 1 | 1 | 2 | 3 | 2 | 2 | 1 | 3 | 5 | 5 |
| 4 | 2 | 3 | 2 | 0 | 1 | 3 | 2 | 1 | 4 | 3 | 3 | 1 | 3 | 4 |
| 5 | 3 | 2 | 1 | 1 | 0 | 2 | 3 | 2 | 3 | 3 | 2 | 2 | 4 | 4 |
| 6 | 4 | 1 | 1 | 3 | 2 | 0 | 1 | 2 | 2 | 1 | 1 | 3 | 5 | 5 |
| 7 | 3 | 2 | 2 | 2 | 3 | 1 | 0 | 1 | 3 | 1 | 2 | 2 | 4 | 4 |
| 8 | 2 | 3 | 3 | 1 | 2 | 2 | 1 | 0 | 4 | 2 | 3 | 1 | 3 | 4 |
| 9 | 6 | 1 | 2 | 4 | 3 | 2 | 3 | 4 | 0 | 3 | 2 | 5 | 7 | 7 |
| 10 | 4 | 2 | 2 | 3 | 3 | 1 | 1 | 2 | 3 | 0 | 2 | 3 | 5 | 5 |
| 11 | 5 | 1 | 1 | 3 | 2 | 1 | 2 | 3 | 2 | 2 | 0 | 4 | 6 | 6 |
| 12 | 1 | 4 | 3 | 1 | 2 | 3 | 2 | 1 | 5 | 3 | 4 | 0 | 2 | 5 |
| 13 | 1 | 6 | 5 | 3 | 4 | 5 | 4 | 3 | 7 | 5 | 6 | 2 | 0 | 7 |

1. i-тый элемент последнего столбца e(vi) матрицы расстояний D является длиной максимальной геодезической, исходящей из вершины vi, т.е. ее эксцентриситетом:

e(1)=6; e(2)=6; e(3)=5; e(4)=4; e(5)=4; e(6)=5; e(7)=4; e(8)=4; e(9)=7; e(10)=5; e(11)=6; e(12)=5; e(13)=7;

1. найдем радиус и диаметр:

* радиус, т.е. минимальный среди всех эксцентриситетов, равен

r(G1)=e(4)=4;

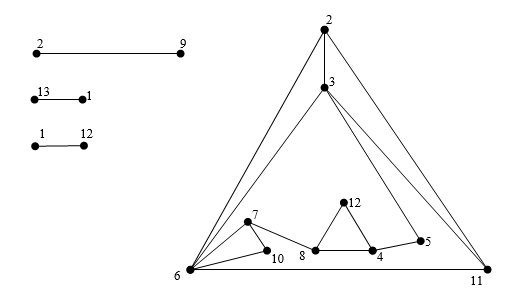
* диаметр, т.е. максимальный среди всех эксцентриситетов, равен

d(G1)=e(13)=7;

1. найдем центр и периферию:

* центром графа G1 является множество {4,5,7,8}, поскольку эксцентриситет этих вершин равен радиусу, т.е. имеет значение 4;
* периферией графа G1 является множество вершин {9,13}, поскольку эксцентриситет этих вершин равен диаметру, т.е. имеет значение 7;

1. выделим блоки:



1. найдем точки сочленения и мосты:

* если удалить вершину 2, или вершину 1, или вершину 12, то компонент связности в графе станет больше, следовательно, точками сочленения данного графа являются вершины 1, 2, 12;
* если удалить ребро (2,9), или ребро (1,12), или ребро (1,13), то граф станет не связным, следовательно, мостами данного графа являются ребра (2,9), (1,12), (1,13).

**5. *В графе* G2:**

1. ***построить кратчайшие маршруты от произвольной вершины ко всем остальным при помощи алгоритма Дейкстры;***
2. ***построить кратчайшие маршруты при помощи алгоритма Флойда. При построении вести две матрицы – матрицу маршрутов и матрицу расстояний.***

Построим кратчайшие маршруты от вершины 5 ко всем остальным с помощью **алгоритма Дейкстры**.

Матрица весов:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | ∞ | 26 | ∞ | ∞ | ∞ |
| 2 | 10 | 0 | 21 | 16 | 9 | ∞ | ∞ |
| 3 | ∞ | 21 | 0 | ∞ | 12 | 14 | 32 |
| 4 | 26 | 16 | ∞ | 0 | ∞ | 9 | 27 |
| 5 | ∞ | 9 | 12 | ∞ | 0 | 2 | 20 |
| 6 | ∞ | ∞ | 14 | 9 | 2 | 0 | 18 |
| 7 | ∞ | ∞ | 32 | 27 | 20 | 18 | 0 |

Шаг 0. Присвоение начальных значений. Назначим вершине 5 постоянную метку *l*(5)=0:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)= ∞ | Временная |
| 2 | *l*(2)=9 | Временная |
| 3 | *l*(3)=12 | Временная |
| 4 | *l*(4)=∞ | Временная |
| 5 | *l*(5)=0 | **Постоянная** |
| 6 | *l*(6)=2 | Временная |
| 7 | *l*(7)=20 | Временная |

Шаг 1. Среди смежных с 5 вершин вершина 6 имеет наименьшее значение расстояния (*l*(6)=2). Поэтому статус метки этой вершины изменяется на «постоянная», *р*=6.

Шаг 2. Из вершины 3 можно попасть в вершины 3, 4 и 7. Получаем следующий список меток вершин:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)= ∞ | Временная |
| 2 | *l*(2)=9 | <- Временная |
| 3 | *l*(3)= min[12, 2+14)]=12 | Временная |
| 4 | *l*(4)= min[∞, 2+9)]=11 | Временная |
| 5 | *l*(5)=0 | **Постоянная** |
| 6 | *l*(6)=2 | **Постоянная** |
| 7 | *l*(7)= min[20, 2+18]=20 | Временная |

Временный статус метки вершины 2 заменяется на постоянный. На втором шаге вершина 7 получает две метки с одинаковым значением расстояния *l*(7) = 20, т.е. эта вершина достигается непосредственно из 5 или через вершину 6 с равным расстоянием.

Шаг 3. Из вершины 2 можно достичь вершин 1, 3 и 4. После обновления меток получим следующий их список:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)= min[∞, 9+10)]=19 | Временная |
| 2 | *l*(2)=9 | **Постоянная** |
| 3 | *l*(3)= min[12, 9+21)]=12 | Временная |
| 4 | *l*(4)= min[11, 9+16)]=11 | <- Временная |
| 5 | *l*(5)=0 | **Постоянная** |
| 6 | *l*(6)=2 | **Постоянная** |
| 7 | *l*(7)=20 | Временная |

Временный статус метки вершины 4 заменяется на постоянный.

Шаг 4. Из вершины 4 можно перейти в вершины 1, 2, 6, 7. Но вершины 2 и 6 имеют уже постоянные метки, временные метки у вершин 1 и 7. После обновления меток получим следующее:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)= min[19, 11+26)]=19 | Временная |
| 2 | *l*(2)=9 | **Постоянная** |
| 3 | *l*(3)=12 | <- Временная |
| 4 | *l*(4)=11 | **Постоянная** |
| 5 | *l*(5)=0 | **Постоянная** |
| 6 | *l*(6)=2 | **Постоянная** |
| 7 | *l*(7)= min[20, 11+27)]=20 | Временная |

Временный статус метки вершины 3 заменяется на постоянный.

Шаг 4. Из вершины 3 можно попасть в вершины 2, 6 и 7, из них только вершина 7 имеет временную метку. В итоге, после обновления, получаем следующий список меток:

|  |  |  |
| --- | --- | --- |
| Вершина | Метка *l*(*i*) | Статус метки |
| 1 | *l*(1)=19 | <- Временная |
| 2 | *l*(2)=9 | **Постоянная** |
| 3 | *l*(3)=12 | **Постоянная** |
| 4 | *l*(4)=11 | **Постоянная** |
| 5 | *l*(5)=0 | **Постоянная** |
| 6 | *l*(6)=2 | **Постоянная** |
| 7 | *l*(7)=20 | Временная |

Временный статус метки вершины 1 заменяется на постоянный. Со временной метой остается только вершина 7, но из 1 в нее попасть нельзя, поэтому процесс вычислений заканчивается.

Итак, кратчайшие маршруты:

M5,1={5,2,1}=19; M5,2={5,2}=9; M5,3={5,3}=12;

M5,4={5,6,4}=11; M5,6={5,6}=2; M5,7={5,7}={5,6,7}=20.

Построим кратчайшие маршруты при помощи **алгоритма Флойда**.

Шаг 0. Строим начальную матрицу расстояний D0 и матрицу маршрутов S0. Полагаем *k* = 1 и в матрице D0 выделяем ведущие строку и столбец. Здесь и далее ведущие строка и столбец отображены серым фоном.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | ∞ | 26 | ∞ | ∞ | ∞ |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 10 | 0 | 21 | 16 | 9 | ∞ | ∞ |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | ∞ | 21 | 0 | ∞ | 12 | 14 | 32 |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 26 | 16 | ∞ | 0 | ∞ | 9 | 27 |  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | ∞ | 9 | 12 | ∞ | 0 | 2 | 20 |  | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | ∞ | ∞ | 14 | 9 | 2 | 0 | 18 |  | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | ∞ | ∞ | 32 | 27 | 20 | 18 | 0 |  | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

Шаг 1. В матрице D0 нет элементов, значения которых можно улучшить с помощью треугольного оператора. Полагаем *k* = 2 и выделяем соответствующие ведущие строку и столбец в D1. Матрицы D1 и S1 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | **∞** | 26 | **∞** | ∞ | ∞ |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 10 | 0 | 21 | 16 | 9 | ∞ | ∞ |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | **∞** | 21 | 0 | **∞** | 12 | 14 | 32 |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 26 | 16 | **∞** | 0 | **∞** | 9 | 27 |  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 5 | **∞** | 9 | 12 | **∞** | 0 | 2 | 20 |  | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 6 | ∞ | ∞ | 14 | 9 | 2 | 0 | 18 |  | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | ∞ | ∞ | 32 | 27 | 20 | 18 | 0 |  | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

Здесь и далее значения элементов текущей матрицы Di, которые можно улучшить с помощью треугольного оператора, выделены штриховкой. В данном случае в матрице D1 таковыми являются элементы *d*13, *d*15, *d*34, *d*45, а также симметричные им *d*31, *d*41, *d*51, *d*43 и *d*54. Выполняем следующие действия:

заменяем *d*13 на *d*12 + *d*23 = 10 + 21 = 31 и устанавливаем *s*13 = 2;

заменяем *d*15 на *d*12 + *d*25 = 10 + 9 = 19 и устанавливаем *s*15 = 2;

заменяем *d*34 на *d*32 + *d*24 = 21 + 16 = 37 и устанавливаем *s*34 = 2;

заменяем *d*45 на *d*42 + *d*25 = 16 + 9 = 25 и устанавливаем *s*45 = 2;

заменяем *d*31 на *d*32 + *d*21 = 21 + 10 = 31 и устанавливаем *s*31 = 2;

заменяем *d*51 на *d*52 + *d*25 = 9 + 10 = 19 и устанавливаем *s*51 = 2;

заменяем *d*43 на *d*42 + *d*23 = 16+ 21 = 37 и устанавливаем *s*43 = 2;

заменяем *d*54 на *d*52 + *d*24 = 9 + 16 = 25 и устанавливаем *s*54 = 2.

Полагаем *k* = 3.

Шаг 2. Матрицы D2 и S2 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | **31** | 26 | **19** | **∞** | **∞** |  | 1 | 1 | 1 | **2** | 1 | **2** | 1 | 1 |
| 2 | 10 | 0 | 21 | 16 | 9 | **∞** | **∞** |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | **31** | 21 | 0 | **37** | 12 | 14 | 32 |  | 3 | **2** | 3 | 3 | **2** | 3 | 3 | 3 |
| 4 | 26 | 16 | **37** | 0 | **25** | 9 | 27 |  | 4 | 4 | 4 | **2** | 4 | **2** | 4 | 4 |
| 5 | **19** | 9 | 12 | **25** | 0 | 2 | 20 |  | 5 | **2** | 5 | 5 | **2** | 5 | 5 | 5 |
| 6 | **∞** | **∞** | 14 | 9 | 2 | 0 | 18 |  | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | **∞** | **∞** | 32 | 27 | 20 | 18 | 0 |  | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

Изменения, внесенные в матрицы D1 и S1 на предыдущем шаге, выделены жирным. Применяем треугольный оператор:

заменяем *d*16 на *d*13 + *d*36 = 31+ 14 = 45 и устанавливаем *s*16 = 3;

заменяем *d*17 на *d*13 + *d*37 = 31 + 32 = 63 и устанавливаем *s*17 = 3;

заменяем *d*61 на *d*63 + *d*31 = 14 + 31 = 45 и устанавливаем *s*61 = 3;

заменяем *d*71 на *d*73 + *d*31 = 32 + 31 = 63 и устанавливаем *s*71 = 3;

заменяем *d*26 на *d*23 + *d*36 = 21+ 14 = 35 и устанавливаем *s*26 = 3;

заменяем *d*27 на *d*23 + *d*37 = 21 + 32 = 53 и устанавливаем *s*27 = 3;

заменяем *d*62 на *d*63 + *d*32 = 14 + 21 = 35 и устанавливаем *s*62 = 3;

заменяем *d*72 на *d*73 + *d*32 = 32 + 21 = 53 и устанавливаем *s*72 = 3.

Полагаем *k* = 4.

Шаг 3. Матрицы D3 и S3 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | 31 | 26 | 19 | **45** | **63** |  | 1 | 1 | 1 | 2 | 1 | 2 | **3** | **3** |
| 2 | 10 | 0 | 21 | 16 | 9 | **35** | **53** |  | 2 | 2 | 2 | 2 | 2 | 2 | **3** | **3** |
| 3 | 31 | 21 | 0 | 37 | 12 | 14 | 32 |  | 3 | 2 | 3 | 3 | 2 | 3 | 3 | 3 |
| 4 | 26 | 16 | 37 | 0 | 25 | 9 | 27 |  | 4 | 4 | 4 | 2 | 4 | 2 | 4 | 4 |
| 5 | 19 | 9 | 12 | 25 | 0 | 2 | 20 |  | 5 | 2 | 5 | 5 | 2 | 5 | 5 | 5 |
| 6 | **45** | **35** | 14 | 9 | 2 | 0 | 18 |  | 6 | **3** | **3** | 6 | 6 | 6 | 6 | 6 |
| 7 | **63** | **53** | 32 | 27 | 20 | 18 | 0 |  | 7 | **3** | **3** | 7 | 7 | 7 | 7 | 7 |

Элементы, к которым применяется треугольный оператор, заштрихованы.

Заменяем *d*16 на *d*14 + *d*46 = 26+ 9 = 35 и устанавливаем *s*16 = 4.

Заменяем *d*17 на *d*14 + *d*47 = 26 + 27 = 53 и устанавливаем *s*17 = 4.

Заменяем *d*61 на *d*64 + *d*41 = 9 + 26 = 35 и устанавливаем *s*61 = 4.

Заменяем *d*71 на *d*74 + *d*41 = 27 + 26 = 53 и устанавливаем *s*71 = 4.

Заменяем *d*26 на *d*24 + *d*46 = 16+ 9 = 25 и устанавливаем *s*26 = 4.

Заменяем *d*27 на *d*24 + *d*47 = 16 + 27 = 43 и устанавливаем *s*27 = 4.

Заменяем *d*62 на *d*64 + *d*42 = 9 + 16 = 25 и устанавливаем *s*62 = 4.

Заменяем *d*72 на *d*74 + *d*42 = 27 + 16 = 43 и устанавливаем *s*72 = 4.

Полагаем *k* = 5.

Шаг 4. Матрицы D5 и S5 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | 31 | 26 | 19 | **35** | **53** |  | 1 | 1 | 1 | 2 | 1 | 2 | **4** | **4** |
| 2 | 10 | 0 | 21 | 16 | 9 | **25** | **43** |  | 2 | 2 | 2 | 2 | 2 | 2 | **4** | **4** |
| 3 | 31 | 21 | 0 | 37 | 12 | 14 | 32 |  | 3 | 2 | 3 | 3 | 2 | 3 | 3 | 3 |
| 4 | 26 | 16 | 37 | 0 | 25 | 9 | 27 |  | 4 | 4 | 4 | 2 | 4 | 2 | 4 | 4 |
| 5 | 19 | 9 | 12 | 25 | 0 | 2 | 20 |  | 5 | 2 | 5 | 5 | 2 | 5 | 5 | 5 |
| 6 | **35** | **25** | 14 | 9 | 2 | 0 | 18 |  | 6 | **4** | **4** | 6 | 6 | 6 | 6 | 6 |
| 7 | **53** | **43** | 32 | 27 | 20 | 18 | 0 |  | 7 | **4** | **4** | 7 | 7 | 7 | 7 | 7 |

Элементы, к которым применяется треугольный оператор, заштрихованы.

Заменяем *d*16 на *d*15 + *d*56 = 19+ 2 = 21 и устанавливаем *s*16 = 5.

Заменяем *d*17 на *d*15 + *d*57 = 19 + 20 = 39 и устанавливаем *s*17 = 5.

Заменяем *d*61 на *d*65 + *d*51 = 2 + 19 = 21 и устанавливаем *s*61 = 5.

Заменяем *d*71 на *d*75 + *d*51 = 20 + 19 = 39 и устанавливаем *s*71 = 5.

Заменяем *d*26 на *d*25 + *d*56 = 9+ 2 = 11 и устанавливаем *s*26 = 5.

Заменяем *d*27 на *d*25 + *d*57 = 9 + 20 = 29 и устанавливаем *s*27 = 5.

Заменяем *d*62 на *d*65 + *d*52 = 2 + 9 = 11 и устанавливаем *s*62 = 5.

Заменяем *d*72 на *d*75 + *d*52 = 20 + 9 = 29 и устанавливаем *s*72 = 5.

Полагаем *k* = 6.

Шаг 5. Матрицы D6 и S6 имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D6 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S6 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | 31 | 26 | 19 | **21** | **39** |  | 1 | 1 | 1 | 2 | 1 | 2 | **5** | **5** |
| 2 | 10 | 0 | 21 | 16 | 9 | **11** | **29** |  | 2 | 2 | 2 | 2 | 2 | 2 | **5** | **5** |
| 3 | 31 | 21 | 0 | **37** | 12 | 14 | 32 |  | 3 | 2 | 3 | 3 | 2 | 3 | 3 | 3 |
| 4 | 26 | 16 | **37** | 0 | **25** | 9 | 27 |  | 4 | 4 | 4 | 2 | 4 | 2 | 4 | 4 |
| 5 | 19 | 9 | 12 | **25** | 0 | 2 | 20 |  | 5 | 2 | 5 | 5 | 2 | 5 | 5 | 5 |
| 6 | **21** | **11** | 14 | 9 | 2 | 0 | 18 |  | 6 | **5** | **5** | 6 | 6 | 6 | 6 | 6 |
| 7 | **39** | **29** | 32 | 27 | 20 | 18 | 0 |  | 7 | **5** | **5** | 7 | 7 | 7 | 7 | 7 |

В матрице D6 элементами, значения которых можно улучшить с помощью треугольного оператора, являются *d*34 и *d*45, а также симметричные им *d*43 и *d*54.

Заменяем *d*34 на *d*36 + *d*64 = 14 + 9 = 13 и устанавливаем *s*34 = 6.

Заменяем *d*45 на *d*46 + *d*65 = 9+ 2 = 11 и устанавливаем *s*45 = 6.

Заменяем *d*43 на *d*46 + *d*63 = 9+ 14 = 13 и устанавливаем *s*43 = 6.

Заменяем *d*54 на *d*56 + *d*64 = 2+ 9 = 11 и устанавливаем *s*54 = 6.

Полагаем *k* = 7.

Шаг 7. После замены значений элементов *d*34 на *d*43 в матрице D6, получаем окончательную матрицу кратчайших расстояний D7 и кратчайших маршрутов S7:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | S7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 10 | 31 | 26 | 19 | 21 | 39 |  | 1 | 1 | 1 | 2 | 1 | 2 | 5 | 5 |
| 2 | 10 | 0 | 21 | 16 | 9 | 11 | 29 |  | 2 | 2 | 2 | 2 | 2 | 2 | 5 | 5 |
| 3 | 31 | 21 | 0 | **13** | 12 | 14 | 32 |  | 3 | 2 | 3 | 3 | **6** | 3 | 3 | 3 |
| 4 | 26 | 16 | **13** | 0 | **11** | 9 | 27 |  | 4 | 4 | 4 | **6** | 4 | **6** | 4 | 4 |
| 5 | 19 | 9 | 12 | **11** | 0 | 2 | 20 |  | 5 | 2 | 5 | 5 | **6** | 5 | 5 | 5 |
| 6 | 21 | 11 | 14 | 9 | 2 | 0 | 18 |  | 6 | 5 | 5 | 6 | 6 | 6 | 6 | 6 |
| 7 | 39 | 29 | 32 | 27 | 20 | 18 | 0 |  | 7 | 5 | 5 | 7 | 7 | 7 | 7 | 7 |

Распишем матрицу кратчайших маршрутов:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1** | 0 | 1,2 | 1,2,3 | 1,4 | 1,2,5 | 1,2,5,6 | 1,2,5,7 |
| **2** | 2,1 | 0 | 2,3 | 2,4 | 2,5 | 2,5,6 | 2,5,7 |
| **3** | 3,2,1 | 3,2 | 0 | 3,6,4 | 3,5 | 3,6 | 3,7 |
| **4** | 4,1 | 4,2 | 4,6,3 | 0 | 4,6,5 | 4,6 | 4,7 |
| **5** | 5,1 | 5,4,2 | 5,3 | 5,4 | 0 | 5,6 | 5,7 |
| **6** | 6,5,2,1 | 6,5,2 | 6,3 | 6,4 | 6,5 | 0 | 6,7 |
| **7** | 7,5,2,1 | 7,5,2 | 7,3 | 7,4 | 7,5 | 7,6 | 0 |

# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ №3

***Тема:* «**Деревья и остовы. Циклы и обходы».

***Цель работы****:* изучение понятий деревьев и остовов, а также гамильтоновых и эйлеровых циклов, приобретение практических навыков построения матрицы Кирхгофа и вычисления количества помеченных остовов.

***Содержание работы:***

1. Используя алгоритм генерации варианта GV, построить неориентированный граф G1: GV(5,{2,3}) и граф G2: GV(13,{6,7}). Ребра графа G2 взвешены соответствующими элементами матрицы Y.
2. Для графа G1 составить матрицу Кирхгофа и посчитать количество помеченных остовов.
3. Для графа G2:
4. построить дерево обхода вершин графа в ширину и в глубину;
5. решить задачу построения остовов кратчайших маршрутов, используя алгоритмы Прима и Краскала (в качестве весов ребер использовать элементы матрицы Y);
6. Определить, являются ли графы G1 и G2 эйлеровыми, построить эйлеровы циклы по алгоритму Флёри, эйлеровы цепи. Если граф не эйлеров, добавить минимальное число ребер, делающих его эйлеровым.
7. Определить, является ли граф G2 гамильтоновым, построить гамильтонов цикл, используя алгоритм Робертса-Флореса. Если граф не является гамильтоновым, то добавить минимальное число ребер, делающих его гамильтоновым.

ПРИМЕР ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 3

***1. Используя алгоритм генерации варианта GV, построить неориентированный граф G1: GV(5,{2,3}) и граф G2: GV(13,{6,7}). Ребра графа G2 взвешены соответствующими элементами матрицы Y.***

Построение графа G1 по множеству S=<ю,ф,а,е,л,н,я,к,о,в> приведено в примере 1 приложения Б, взвешенного графа G2 – в примере 3 приложения Б. Опишем полученные графы G1=(V1,E1) и G2=(V2,E2), используя способ перечисления:

V1={1,2,3,4,5} – множество вершин графа G1;

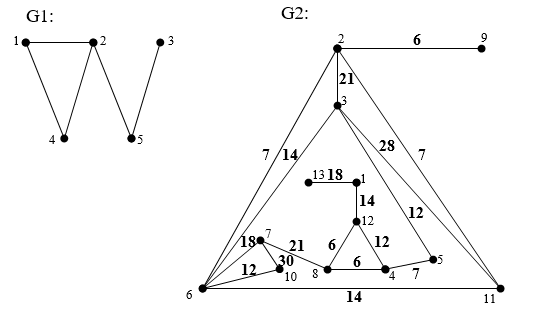
E1={(1,2),(1,4),(2,4),(2,5),(3,5)} – множество ребер графа G1;

V2={1,2,3,4,5,6,7,8,9,10,11,12,13} – множество вершин графа G2;

E2={(1,12),(1,13),(2,3),(2,6),(2,9),(2,11),(3,5),(3,6),(3,11),(4,5),

(4,8),(4,12),(6,7),(6,10),(6,11),(7,8),(7,10),(8,12)} – множество ребер графа G2.

Приведем графические изображения графа G1 и взвешенного графа G2:



**2. *Для графа* G1 *составить матрицу Кирхгофа и посчитать количество помеченных остовов*.**

Матрица Кирхгофа для графа G1 имеет вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| M | 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | -1 | 0 | -1 | 0 |
| 2 | -1 | 3 | 0 | -1 | -1 |
| 3 | 0 | 0 | 1 | 0 | -1 |
| 4 | -1 | -1 | 0 | 2 | 0 |
| 5 | 0 | -1 | -1 | 0 | 2 |

Количество помеченных остовов k графа G равно алгебраическому дополнению любого элемента матрицы Кирхгофа:



Итак, в графе G1 содержится 3 помеченных остова:

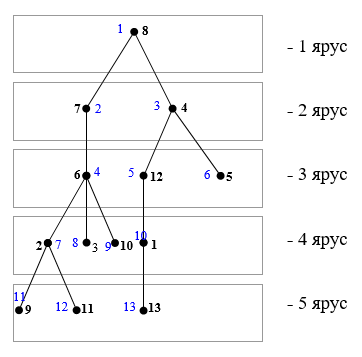


**3. *Для графа G2***

***a) построить дерево обхода вершин графа в ширину и в глубину.***

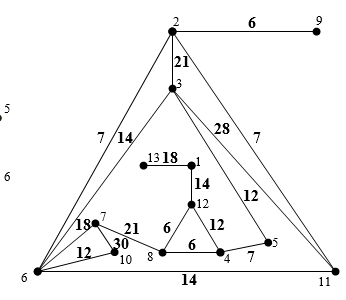
|  |  |
| --- | --- |
| Граф G2 | Остов графа G2 и обход в глубину |
|  |  |

Ярусная форма представления остова и обход в ширину:



***b) решить задачу построения остовов кратчайших маршрутов, используя алгоритмы Прима и Краскала (в качестве весов ребер использовать элементы матрицы Y).***

Взвешенный граф G2 приведен в примере 3 приложения Б и имеет вид:



**Алгоритм** **Краскала:** упорядочим ребра по неубыванию их веса. К пустому графу Op (p=13) добавляется ребро с минимальным весом без образования цикла. Количество итераций равно p-1=12.

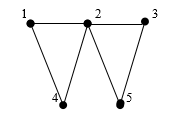
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Порядок добавления ребра | Вес | Ребро |  | Остов графа по алгоритму Краскала |
| 1 | 6 | (2,9) |  |  |
| 2 | (4,8) |  |
| 3 | (8,12) |  |
| 4 | 7 | (2,6) |  |
| 5 | (2,11) |  |
| 6 | (4,5) |  |
| 7 | 12 | (3,5) |  |
|  | (4,12) | хорда |
| 8 | (6,10) |  |
| 9 | 14 | (1,12) |  |
| 10 | (3,6) |  |
|  | (6,11) | хорда |
| 11 | 18 | (1,13) |  |
| 12 | (6,7) |  |
|  | 21 | (2,3) | хорда |
|  | (7,8) | хорда |
|  | 28 | (3,11) | хорда |
|  | 30 | (7,10) | хорда |

**Алгоритм Прима:** упорядочим ребра по неубыванию их веса. К пустому графу Op (p=13) добавляется ребро с минимальным весом без образования цикла и нарушения связности. Количество итераций равно p-1=12.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Порядок добавления ребра | Вес | Ребро |  | Остов графа по алгоритму Краскала |
| 1 | 6 | (2,9) |  |  |
| 2 | 7 | (2,6) |  |
| 3 | 7 | (2,11) |  |
| 4 | 12 | (6,10) |  |
| 5 | 14 | (3,6) |  |
| 6 | 12 | (3,5) |  |
| 7 | 7 | (5,4) |  |
| 8 | 6 | (4,8) |  |
| 9 | 6 | (8,12) |  |
| 10 | 14 | (1,12) |  |
| 11 | 18 | (1,13) |  |
| 12 | 18 | (6,7) |  |
|  | 12 | (4,12) | хорда |
|  | 14 | (6,11) | хорда |
|  | 21 | (2,3) | хорда |
|  | 21 | (7,8) | хорда |
|  | 28 | (3,11) | хорда |
|  | 30 | (7,10) | хорда |

***4.******Определить, являются ли графы G1 и G2 эйлеровыми, построить эйлеровы циклы по алгоритму Флёри, эйлеровы цепи. Если граф не эйлеров, добавить минимальное число ребер, делающих его эйлеровым.***

1) Рассмотрим граф G1: в нем все вершины имеют четную степень, следовательно, граф G1 имеет эйлеров цикл, то есть является эйлеровым графом.



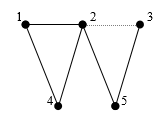
Построим в графе эйлеров цикл с помощью **алгоритма Флёри**. Задача сводится к поиску способа нумерации всех ребер в полученном графе таким образом, чтобы номер каждого ребра указывал порядок вхождения ребра в цикл.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| V | (1,2) | (1,4) | (2,3) | (2,4) | (2,5) | (3,5) |
| № | 1 | 2 | 4 | 3 | 6 | 5 |
| № | 4 | 5 | 3 | 6 | 1 | 2 |

Эйлеровы циклы:

С1={2,1,4,2,3,5,2}; C2={2,5,3,2,1,4,2}.

Построим в графе эйлеровы цепи. Для этого удалим ребро (2,3), тогда только две вершины – 2 и 3 – будут иметь нечетную степень, а это является необходимым и достаточным условием того, что граф G1 покрывается эйлеровой цепью.

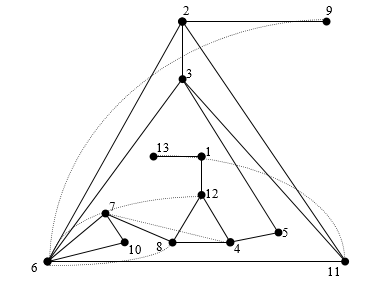


Эйлеровы цепи:

P1={2,1,4,2,5,3}; P2={3,5,2,4,1,2}.

2) Рассмотрим граф G2: так как в нем вершины 4, 6, 7, 8, 10, 11, 12, 13 имеют нечетную степень, то граф G2 не имеет эйлерова цикла и, следовательно, не является эйлеровым графом.

Добавим ребра (6,9), (6,12), (6,8), (4,7), (11,13). Получим граф, все вершины которого будут иметь четную степень. Это является достаточным условием для того, чтобы граф стал эйлеровым.



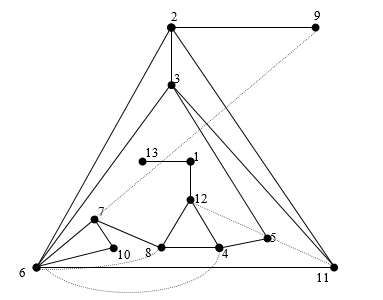
Построим в графе эйлеров цикл с помощью **алгоритма Флёри**.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| V | (1,12) | (1,13) | (2,3) | (2,6) | (2,9) | (2,11) | (3,5) | (3,6) |
| № | 9 | 8 | 5 | 23 | 1 | 4 | 12 | 13 |
| V | (3,11) | (4,5) | (4,7) | (4,8) | (4,12) | (6,7) | (6,8) | (6,9) |
| № | 6 | 11 | 17 | 16 | 10 | 18 | 22 | 2 |
| V | (6,11) | (6,12) | (7,8) | (7,10) | (8,12) | (11,13) | (6,10) |  |
| № | 3 | 14 | 21 | 20 | 15 | 7 | 19 |  |

Эйлеров цикл:

С={2,9,6,11,2,3,11,13,1,12,4,5,3,6,12,8,4,7,6,10,7,8,6,2}.

Построим в графе эйлерову цепь. Добавим ребра (1,6), (3,2), тогда только две вершины – 7 и 13 – будут иметь нечетную степень, а это является необходимым и достаточным условием того, чтобы граф G2 покрывался эйлеровой цепью.

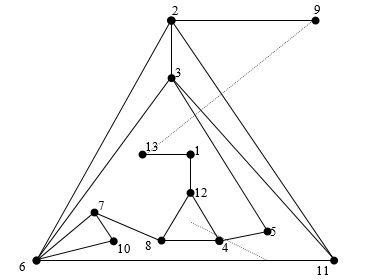


Эйлерова цепь:

P2={13,1,12,11,2,9,7,8,12,4,3,11,6,7,10,6,8,4,6,2,3,6}.

***6. Определить, является ли граф G2 гамильтоновым, построить гамильтонов цикл, используя алгоритм Робертса-Флореса. Если граф не является гамильтоновым, добавить минимальное число ребер, делающих его гамильтоновым.***

Граф G2 не является гамильтоновым, так как в графе есть 2 висячие вершины: 9, 13, следовательно, нельзя выделить гамильтонов цикл, то есть такой простой цикл, который содержит каждую вершину графа.



Добавив ребро (9,13), можно выделить гамильтонов цикл:

Cg = {9,2,3,11,6,10,7,8,4,5,12,1,13,9}.

Следовательно, полученный граф – гамильтонов.

**Алгоритм Робертса-Флореса.**

Выберем вершину 10 в качестве начальной.

|  |  |
| --- | --- |
| 1. S={}; 2. S={10}; 3. S={10,7}; 4. S={10,7,8}; 5. S={10,7,8,4}; 6. S={10,7,8,4,5}; 7. S={10,7,8,4,5,12}; 8. S={10,7,8,4,5,12,1}; 9. S={10,7,8,4,5,12,1,13}; 10. S={10,7,8,4,5,12,1,13,9}; 11. S={10,7,8,4,5,12,1,13,9,2}; 12. S={10,7,8,4,5,12,1,13,9,2,3}; 13. S={10,7,8,4,5,12,1,13,9,2,3,11}; 14. S={10,7,8,4,5,12,1,13,9,2,3,11,6} | V={1,2,3,4,5,6,7,8,9,10,11,12,13};  V={1,2,3,4,5,6,7,8,9,11,12,13};  V={1,2,3,4,5,6,8,9,11,12,13};  V={1,2,3,4,5,6,9,11,12,13};  V={1,2,3,5,6,9,11,12,13};  V={1,2,3,6,9,11,12,13};  V={1,2,3,6,9,11,13};  V={2,3,6,9,11,13};  V={2,3,6,9,11};  V={2,3,6,11};  V={3,6,11};  V={6,11};  V={6};  V={}; |

Так как цепь, проходящая через вершины множества S, имеет длину   
p-1=13-1=12, и в графе G2 существует ребро, соединяющее последнюю и первую вершины цепи, замыкая ее, то в графе существует гамильтонов цикл и множество S={10,7,8,4,5,12,1,13,9,2,3,11,6,10} – множество вершин этого цикла.

# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ №4

***Тема:*** «Ориентированные графы».

***Цель работы:*** изучение основных понятий ориентированных графов, приобретение практических навыков построения матрицы смежности, матрицы инцидентности; определение типов связности орграфа; построение конденсации орграфа.

***Содержание работы:***

1. Используя алгоритм генерации варианта GV1, построить ориентированный граф G: GV1(9,{6,7}).
2. Построить матрицу смежности и матрицу инцидентности заданного орграфа.
3. Построить основание и обратный граф. Определить, является ли граф симметричным.
4. Построить ормаршрут, цепь, путь, полумаршрут, полуцепь, полупуть, замкнутый маршрут, цикл и контур.
5. Построить матрицу достижимости, контрдостижимости, взаимной достижимости. Представить ограниченно достижимую матрицу для числа достижимости, равного 2.
6. Определить тип связности орграфа, выделить сильные компоненты.
7. Построить конденсацию. Определить базы и антибазы.

ПРИМЕР ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 4

***1. Используя алгоритм генерации варианта GV, построить ориентированный граф G: GV(9,{6,7}).***

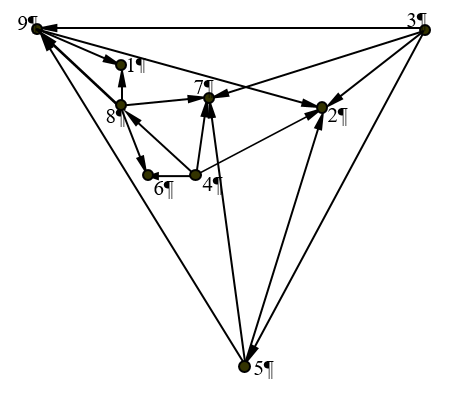
Построение графа G по множеству S=<ю,ф,а,е,л,н,я,к,о,в> приведено в примере 4 приложения Б. Опишем полученные графы G=(V, А), используя способ перечисления:

V={1,2,3,4,5,6,7,8,9} – множество вершин графа;

A={(3,2),(3,5),(3,7),(3,9),(4,2),(4,6),(4,7),(4,8),(5,2),(5,7),(5,9),(8,1),(8,6),

(8,7),(8,9),(9,1),(9,2)} – множество дуг графа.

Построим графическое изображение графа G:



**2. *Построить матрицу смежности и матрицу инцидентности заданного графа.***

Матрица смежности

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 9 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Матрица инцидентности

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B | (3,2) | (3,5) | (3,7) | (3,9) | (4,2) | (4,6) | (4,7) | (4,8) | (5,2) | (5,7) | (5,9) | (8,1) | (8,6) | (8,7) | (8,9) | (9,1) | (9,2) | deg+ | deg - | deg |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | 2 | 2 |
| 2 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 4 | 4 |
| 3 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 4 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 4 |
| 5 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 4 |
| 6 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| 7 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 4 | 4 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 4 | 1 | 5 |
| 9 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | 1 | 1 | 2 | 3 | 5 |

**3. *Построить основание и обратный граф. Определить, является ли граф симметричным.***

Основание орграфа – это неориентированный граф, полученный из орграфа в результате снятия ориентации дуг.

Обратный граф – это граф, дуги которого переориентированы по отношению к исходному орграфу.

|  |  |
| --- | --- |
| Основание | Обратный граф |
|  |  |

***4. Построить ормаршрут, цепь, путь, полумаршрут, полуцепь, полупуть, замкнутый маршрут, цикл и контур.***

М1={4,8,9,2} – ормаршрут: чередующаяся последовательность вершин и дуг графа такая, что каждая дуга исходит из предыдущей вершины и заходит в последующую вершину;

М2={4,8,9,1} – цепь: ориентированный маршрут без повторяющихся дуг;

M3={3,5,9,2} – путь: цепь без повторяющихся вершин;

M4={3,2,5,9,1,7,4,2,5,9} – полумаршрут: маршрут основания орграфа;

M5={2,3,5,2,4,7,8,6} – полуцепь: цепь в основании графа;

M6={1,8,7,3,2,4} – полупуть: простой цикл в основании графа.

Замкнутый маршрут – это маршрут, который начинается и заканчивается в одной и той же вершине.

Цикл – это замкнутая цепь.

Контур – это замкнутый путь.

Замкнутый маршрут в графе выделить невозможно, следовательно, нельзя также выделить также цикл и контур.

***5. Построить матрицу достижимости, контрдостижимости, взаимной достижимости. Представить ограниченно достижимую матрицу для числа достижимости, равного 2.***

Матрица достижимости

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 8 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Матрица контрдостижимости

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

Матрица взаимной достижимости

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Ограниченно достижимая матрица для числа достижимости, равного 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 8 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**6. *Определить тип связности орграфа, выделить сильные компоненты.***

Так как в графе любые две вершины соединены полумаршрутом, то граф G слабосвязанный.

Так как в матрице взаимодостижимости единицы стоят только на главной диагонали, то сильными компонентами графа G являются вершины графа 1, 2, 3, 4, 5, 6, 7, 8, 9.

**7. *Построить конденсацию. Определить базы и антибазы.***

Так как сильными компонентами графа G1 являются его вершины, то конденсацией является исходный граф G1.

Так как в графе нет контуров, то базу образуют вершины, которые имеют полустепень захода равную нулю, то есть вершины {3,4}, а антибазу – вершины, имеющие полустепень исхода равную нулю, то есть вершины {1,2,6,7}.