

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Оренбургский государственный университет»

Кафедра алгебры и дискретной математики

ЛАБОРАТОРНЫЕ РАБОТЫ ПО ТЕОРИИ КОНЕЧНЫХ ГРАФОВ

Методические указания

Рекомендовано к изданию редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по образовательным программам высшего образования по направлениям подготовки 02.03.01 Математика и компьютерные науки, 02.03.02 Фундаментальная информатика и информационные технологии и специальности 10.05.01 Компьютерная безопасность

Оренбург
2018

УДК 378.016:519.17(076.5)
ББК 22.176я 7+74.48я7
Л 12

Рецензент - кандидат физико-математических наук, доцент С.А.Герасименко
Авторы: О.А. Пихтилькова, Т.М. Отрыванкина, Л.Б. Усова, Д.У. Шакирова

Л 12 Лабораторные работы по теории конечных графов: методические указания / О.А. Пихтилькова, Т. М. Отрыванкина, Л. Б. Усова, Д. У. Шакирова; Оренбургский гос. ун-т. – Оренбург: ОГУ, 2018. – 76 с.

В методических указаниях содержатся теоретические сведения и задания лабораторных работ к дисциплине «Теория конечных графов» по темам: «Основные понятия, определения и способы задания графов», «Поиск минимального пути в орграфе. Расстояния в графе», «Унарные и бинарные операции над графами», «Связные графы. Компоненты связности», «Минимальные пути (маршруты) в нагруженных орграфах (графах). Алгоритм Форда-Беллмана. Алгоритм Дейкстры. Алгоритм Флойда-Уоршелла», «Эйлеровы графы», «Гамильтоновы графы. Алгоритм Литтла», «Деревья и остовы графов. Алгоритм Краскала. Алгоритм Прима», «Поток в сети. Алгоритм Форда-Фалкерсона».

Методические указания предназначены для обучающихся по направлениям подготовки 02.03.01 Математика и компьютерные науки, 02.03.02 Фундаментальная информатика и информационные технологии и специальности 10.05.01 Компьютерная безопасность.

УДК 378.016:519.17(076.5)
ББК 22.176я 7+74.48я7

© Пихтилькова О.А.,
Отрыванкина Т.М.,
Усова Л.Б.,
Шакирова Д.У, 2018
© ОГУ, 2018

Содержание

Введение	4
1 Лабораторная работа № 1	5
2 Лабораторная работа № 2	9
3 Лабораторная работа № 3	17
4 Лабораторная работа № 4	21
5 Лабораторная работа № 5	25
6 Лабораторная работа № 6	37
7 Лабораторная работа № 7	46
8 Лабораторная работа № 8	56
9 Лабораторная работа № 9	64
Список использованных источников	76

Введение

Лабораторные работы к дисциплине «Теория конечных графов» предназначены для обучающихся по направлениям подготовки «Математика и компьютерные науки», «Фундаментальная информатика и информационные технологии» и специальности «Компьютерная безопасность» факультета математики и информационных технологий ОГУ. Цель написания данных учебно-методических указаний – предоставить преподавателю краткое содержание лекционного и задачного материала для проведения лабораторных работ, а также помочь студентам в усвоении тем данной дисциплины. Методическая разработка будет содействовать усвоению и закреплению базовых понятий, а также составлению и разработки соответствующих программ для более глубокого усвоения дисциплины «Теория конечных графов». Данные указания содержат лабораторные работы по следующим темам:

- 1) «Основные понятия, определения и способы задания графов»,
- 2) «Поиск минимального пути в орграфе. Расстояния в графе»,
- 3) «Унарные и бинарные операции над графами»,
- 4) «Связные графы. Компоненты связности»,
- 5) «Минимальные пути (маршруты) в нагруженных орграфах (графах). Алгоритм Форда-Беллмана. Алгоритм Дейкстры. Алгоритм Флойда-Уоршелла»,
- 6) «Эйлеровы графы»,
- 7) «Гамильтоновы графы. Алгоритм Литтла»,
- 8) «Деревья и остовы графов. Алгоритм Краскала. Алгоритм Прима»,
- 9) «Поток в сети. Алгоритм Форда-Фалкерсона».

В лабораторных работах для лучшего усвоения темы и успешного написания программы содержатся разобранные примеры заданий по каждой теме.

1 Лабораторная работа № 1

Теоретические сведения по теме «Основные понятия, определения и способы задания графов»

Известны различные способы представления графов в памяти компьютера, которые различаются объемом занимаемой памяти и скорости выполнения операций над графами. Представление выбирается, исходя из потребностей конкретной задачи. В подавляющем большинстве случаев граф задается матрицей. Чаще всего графы представляют либо матрицей смежности, либо матрицей инцидентий.

Пусть дан граф $G=(V, E)$, $|V|=n$, $|E|=m$.

Определение. Матрица смежности вершин орграфа G , содержащего n вершин – это квадратная матрица $A = (a_{ij})$ n -го порядка, у которой строки и столбцы матрицы соответствуют вершинам орграфа. Элементы a_{ij} матрицы A равны числу дуг, направленных из i -й вершины в j -ю. Если орграф состоит из однократных дуг, то элементы матрицы равны либо 0, либо 1.

Определение. Матрицей смежности вершин неориентированного графа G , содержащего n вершин, называют квадратную матрицу $A = (a_{ij})$ n -го порядка, у которой строки и столбцы матрицы соответствуют вершинам неориентированного графа. Элементы a_{ij} матрицы A равны числу ребер, направленных из i -й вершины в j -ю. В случае неориентированного графа G ему вместе с ребром (v_i, v_j) принадлежит и ребро (v_j, v_i) , поэтому матрица смежности вершин $A = (a_{ij})$ будет симметрична относительно главной диагонали.

$$a_{ij} = \begin{cases} 1, & \text{если } (v_i, v_j) \in E, \\ 0, & \text{в противном случае.} \end{cases}$$

Замечание. Если G – мультиграф, то значение a_{ij} можно положить равным k , где k – кратность дуги (v_i, v_j) .

Определение. Матрицей инцидентности неорграфа G называется матрица B_G размера $n \times t$, в которой

$$b_{ij} = \begin{cases} 1, & \text{если вершина } v_i \text{ и ребро } e_j \text{ инцидентны,} \\ 0, & \text{в противном случае.} \end{cases}$$

Определение. Матрицей инцидентности орграфа G называется матрица B_G размера $n \times t$, в которой

$$b_{ij} = \begin{cases} 1, & \text{если ребро } e_j \text{ заходит в вершину } v_i, \\ -1, & \text{если ребро } e_j \text{ исходит из вершины } v_i, \\ 0, & \text{в противном случае.} \end{cases}$$

Из определения матрицы смежности следует, что сумма элементов i -ой строки равна степени вершины v_i , то есть $\sum_{j=1}^n a_{ij} = \deg v_i$. Для обыкновенных графов матрица смежности бинарна, то есть состоит только из 0 и 1, причем главная диагональ целиком состоит из нулей.

Матрица смежности полностью определяет структуру графа. Например, сумма всех элементов строки v_i матрицы дает полустепень исхода вершины v_i , а сумма элементов столбца v_i - полустепень захода вершины v_i .

Для одного и того же графа существует несколько матриц смежности, но они все подобны между собой, то есть могут быть получены друг из друга некоторой перестановкой строк и точно такой же перестановкой столбцов.

Списочное задание графа $G = (V, \rho)$ с конечным множеством вершин V осуществляется с помощью *структуры смежности*, представляющей собой список вершин, в котором для каждой вершины a указывается список ее последователей, т.е. таких вершин b , что $(a, b) \in \rho$.

Пример. Для графа $G = (V, \rho)$ с множеством вершин $V = \{a, b, c, d\}$ и бинарным отношением $\rho \in \{(a, c); (a, d); (b, c); (c, c); (d, a); (d, d)\}$ структура смежности имеет вид: $(a:(c,d)), (b:(c)), (c:(c)), (d:(a,d))$.

Определение. Обыкновенный граф - это граф без петель и кратных ребер.

Рассмотрим матрицу A_G^k и обозначим ее элементы $a_{ij}^{(k)}$.

Теорема. Элемент $a_{ij}^{(k)}$ матрицы A_G^k графа (орграфа) $G=(V,E)$ равен числу маршрутов (путей) в G длины k , соединяющих вершину v_i с вершиной v_j (из вершины v_i в вершину v_j).

Следствие.

а) В n -вершинном графе (орграфе) $G=(V,E)$ существует маршрут (путь), соединяющий вершину v_i с вершиной v_j (из вершины v_i в вершину v_j) \Leftrightarrow элемент c_{ij} матрицы $C = A_G + A_G^2 + A_G^3 + \dots + A_G^{n-1}$ отличен от нуля.

б) В n -вершинном графе (орграфе) $G=(V,E)$ существует замкнутый маршрут (путь), проходящий через вершину $v_i \Leftrightarrow$ элемент c_{ii} матрицы $C = A_G + A_G^2 + A_G^3 + \dots + A_G^n$ отличен от нуля.

Пусть G – произвольный граф. Упорядочим множество его вершин.

Матрицей Кирхгофа называется матрица $K = (k_{ij})_{n \times n}$, имеющая вид

$$K = \begin{pmatrix} \deg v_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \deg v_n \end{pmatrix} - A, \text{ где } A - \text{ матрица смежности графа } G.$$

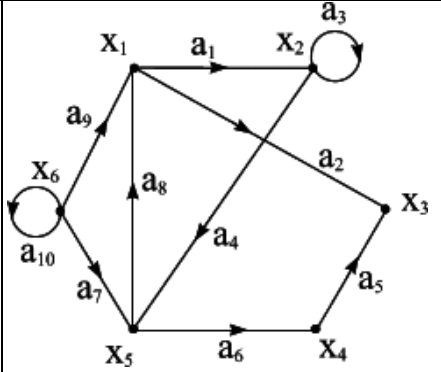
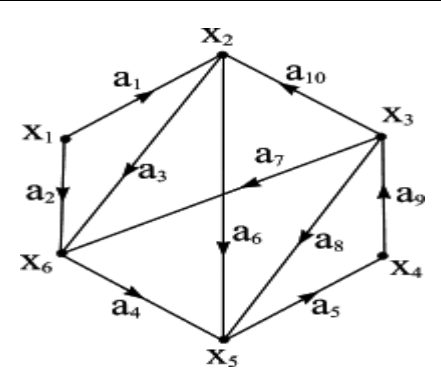
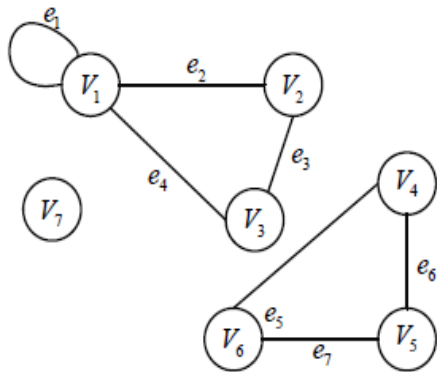
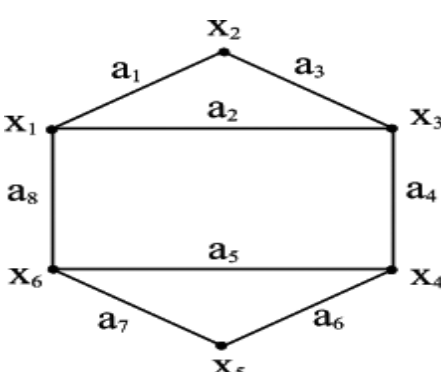
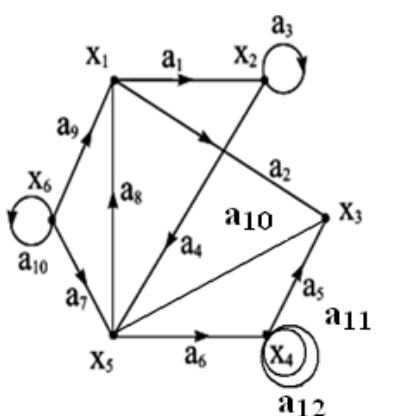
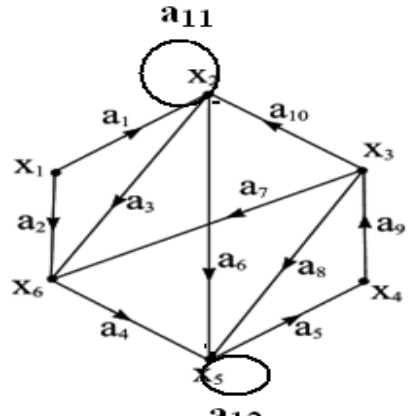
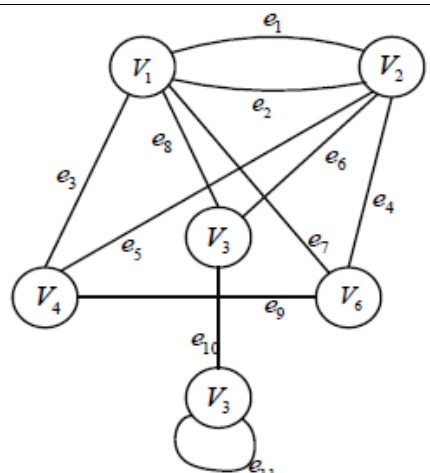
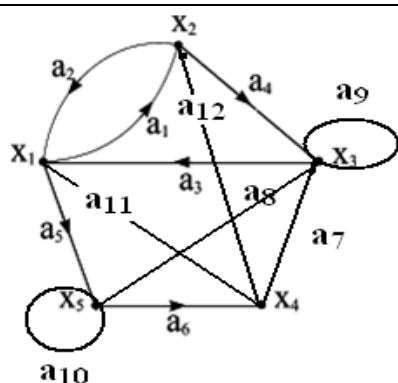
$$\text{Иными словами, } k_{ij} = \begin{cases} -1, & \text{если } i \neq j \text{ и } v_i \text{ смежна с } v_j, \\ 0, & \text{если } i \neq j \text{ и } v_i \text{ не смежна с } v_j, \\ \deg v_i, & \text{если } i = j. \end{cases}$$

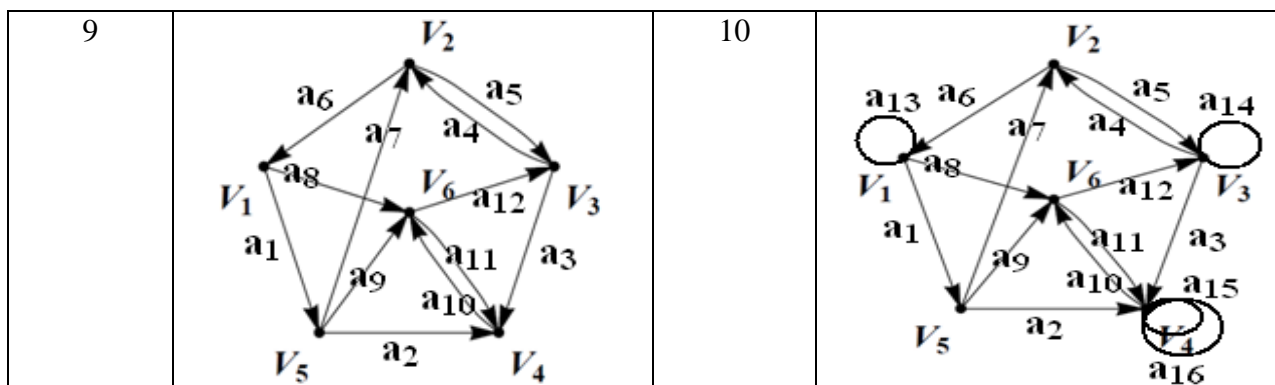
Лемма Алгебраические дополнения матрицы Кирхгофа равны между собой.

Задание:

Составить программу в консольном режиме, которая по известному графу:

1. Строит матрицу смежности графа.
2. Строит матрицу инцидентности графа.
3. Вычисляет степень каждой вершины графа, а для ориентированного еще и полустепень захода и полустепень исхода.
4. Вычисляет число маршрутов длины L из заданных вершин.

Вариант	Задание	Вариант	Задание
1		2	
3		4	
5		6	
7		8	



2 Лабораторная работа № 2

Теоретические сведения по теме «Поиск минимального пути из v в w в орграфе G . Расстояния в графе»

Алгоритм поиска минимального пути из v в w в орграфе G (алгоритм волны)

Введем следующие обозначения. Если $G=(V, E)$, $V_1 \subseteq V$, то

$G(v)=\{w \mid (v, w) \in E\}$ – образ вершины v ,

$G^{-1}(v)=\{w \mid (w, v) \in E\}$ – прообраз вершины v ,

$G(V_1)=\bigcup_{v \in V_1} G(v)$ – образ множества V_1 ,

$G^{-1}(V_1)=\bigcup_{v \in V_1} G^{-1}(v)$ – прообраз множества V_1 .

1. Отметим вершину v индексом 0. Вершины из $G(v)$ отметим индексом 1. Обозначим такие вершины $FW_1(v)$. $k:=1$.

2. Если $FW_k(v)=\emptyset$ или $k=n-1$ и $w \notin FW_k(v)$, то вершина w не достижима из v , и работа алгоритма заканчивается. В противном случае

3. Если $w \notin FW_k(v)$, то переходим к шагу 4. Иначе существует путь из v в w длины k , и он является минимальным. Искомый путь представляет собой последовательность вершин $vw_1w_2 \dots w_{k-1}w$, где

$$w_{k-1} \in FW_{k-1}(v) \cap G^{-1}(w),$$

$$w_{k-2} \in FW_{k-2}(v) \cap G^{-1}(w_{k-1}),$$

...

$$w_1 \in FW_1(v) \cap G^{-1}(w_2).$$

4. Помечаем индексом $k+1$ все непомеченные вершины, которые принадлежат образу множества вершин с индексом k , обозначая их $FW_{k+1}(v)$. $k:=k+1$. Переходим к шагу 2.

Замечание.

1) $FW_k(v)$ называется фронтом волны вершины v k -го уровня.

2) Вершины w_1, w_2, \dots, w_{k-1} могут быть выбраны неоднозначно.

Замечание. Если в алгоритме заменить $G(v)$, $G^{-1}(v)$ на $G(v)$, при изменении соответствующей терминологии он работает в неорграфе G .

Пусть $G=(V, E)$ – граф (в общем случае – псевдограф). Обозначим длину минимального маршрута, соединяющего v и w , $d(v, w)$. Для любой вершины $v \in V$ будем считать $d(v, v)=0$. Кроме того, $d(v, w)=+\infty$, если $v \neq w$ и не существует в G маршрута, соединяющего v и w .

Определение. Величину $d(v, w)$, $v, w \in V$, называют *расстоянием* между вершинами v и w .

Расстояние $d(v, w)$ удовлетворяет свойствам:

1) $d(v, w) \geq 0$, причем $d(v, w)=0 \Leftrightarrow v=w$.

2) $d(v, w)=d(w, v)$.

3) $d(v, w) \leq d(v, u)+d(u, w)$.

В связном графе

4) $d(v, w) < +\infty$.

Таким образом, выполняются все аксиомы метрики.

Определение. Пусть $G=(V, E)$ – связный граф. Величина $d(G)=\max_{v, w \in V} d(v, w)$ называется *диаметром* графа G .

Определение. Пусть $G=(V, E)$ – граф, $v \in V$. Величина $r(v)=\max_{w \in V} d(v, w)$ называется *максимальным удалением* в графе G от вершины v .

Определение. Величину $r(G)=\min_{v \in V} r(v)$ называют *радиусом* графа G .

Любая вершина $v \in V$ такая, что $r(v)=r(G)$, называется *центром* графа G .

Для связного графа $G = (V, E)$ симметрическая матрица $D(G)$ расстояний графа – это квадратная матрица $[d_{i,j}]_{i=1..n, j=1..n}$, в которой $d_{i,j} = \min_{p_{v_i-v_j}} |p_{v_i-v_j}|$ – длина кратчайшего пути $p_{v_i-v_j}^{\min}$ графа из вершины v_i в вершину v_j этого графа. При этом длина пути в не взвешенном графе (т.е. если не указана функция веса ребер $w: E \rightarrow R$) – это количество ребер в этом пути. *Эксцентриситетом* вершин графа называется функция $e: V \rightarrow N^+$ вида $e(v_i) = \max_{v_j \in V} D_{i,j}$, т.е. эксцентриситет $e(v_i)$ вершины $v_i \in V$ – это максимальное расстояние в графе от данной вершины до остальных вершин графа. *Центр* $C(G)$ графа – это подграф $G(V_c)$ данного графа, порожденный множеством $V_c = \{v \in G : e(v) = r(G)\}$ *центральных вершин* графа. Таким образом, центр $C(G)$ – это граф, включающий центральные вершины V_c данного графа и ребра исходного графа G , которые их соединяют. *Периферия* $Per(G)$ графа – это подграф $G(V_p)$ данного графа, порожденный множеством $V_p = \{v \in G : e(v) = d(G)\}$ периферических вершин графа. Две вершины $v, u \in V$ данного графа называются *антиподальными (антиподами)*, если $dist_G(v, u) = d(G)$, т.е. такие вершины находятся на расстоянии, равном диаметру графа и являются концами некоторого *диаметра графа*, т.е. некоторого пути графа, имеющего максимально возможную длину.

Пример.

Дан связный граф $G = (V, E) = ((1, 2, 3, 4, 5), \{\overline{12}, \overline{24}, \overline{45}, \overline{43}, \overline{35}\})$.

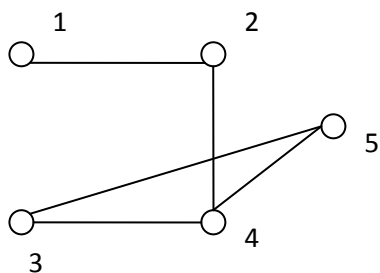


Рисунок 1

1. Изобразить данный граф.
2. Построить дистанционную матрицу графа.

Эту задачу решаем методом предварительного построения всех BFS (т.е. широтных) деревьев достижимости из всех вершин данного графа как начальных. Эта процедура дает следующий результат:

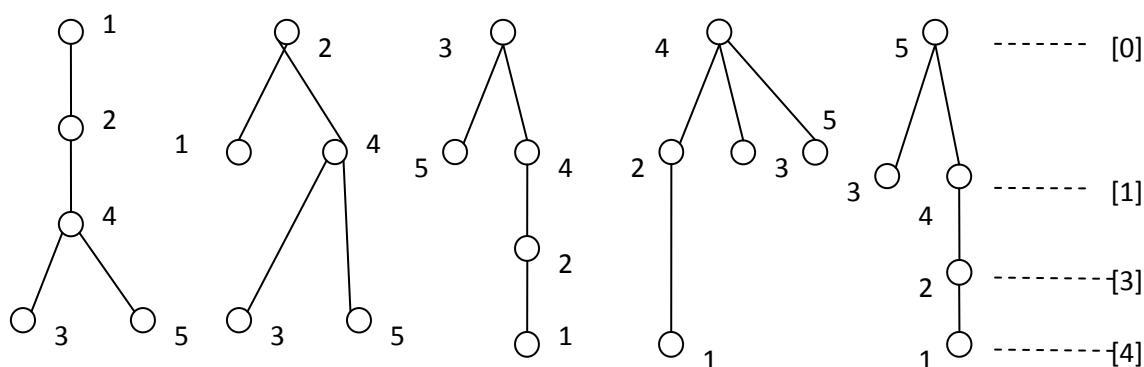


Рисунок 2

Используя уровни вершин в широтных деревьях достижимости, получаем дистанционную матрицу вида

$$D(G) = \begin{pmatrix} 0 & 1 & 3 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 3 & 2 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 1 & 1 & 0 \end{pmatrix}$$

3. Находим эксцентриситеты вершин. В соответствии с определением, для этой цели применяем следующую процедуру:

v	1	2	3	4	5	$e(v) = \max \rightarrow$	$C(G) = \min \downarrow$
1	0	1	3	2	3	3	-
2	1	0	2	1	2	2	2
3	3	2	0	1	1	3	-
4	2	1	1	0	1	2	2
5	3	2	1	1	0	3	-

Итак, эксцентриситеты вершин характеризуются таблицей

v	1	2	3	4	5
$e(v)$	3	2	3	2	3

Радиус графа $r(G) = \min\{3, 2, 3, 2, 3\} = 2$.

Множество центральных вершин, т.е. множество вершин, эксцентриситет которых, равен радиусу графа 2 $V_c = \{2, 4\}$.

Центр графа $C(G)$ - подграф $G(V_c) = (\{2, 4\}, \{\overline{24}\})$ графа G .

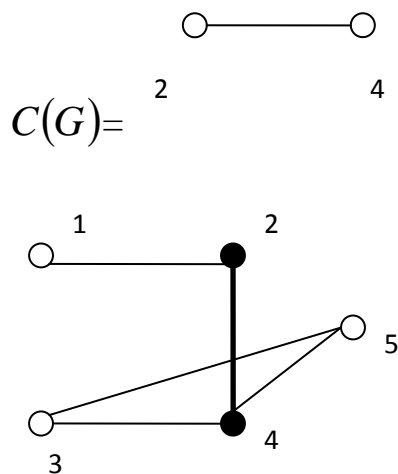


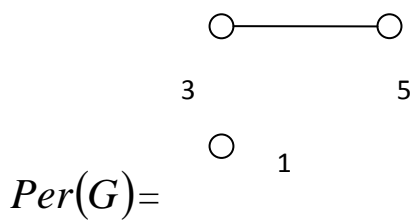
Рисунок 3

Покажем центр графа на исходном графе:

Диаметр графа $d(G) = \max\{3, 2, 3, 2, 3\} = 3$.

Множество периферических вершин, т.е. множество вершин, эксцентриситет которых, равен диаметру графа 3 $V_p = \{1, 3, 5\}$.

Периферия графа $Per(G)$ -подграф $Per(V_p) = (\{1, 3, 5\}, \{\overline{135}\})$ графа G .



Покажем периферию графа на исходном графе:

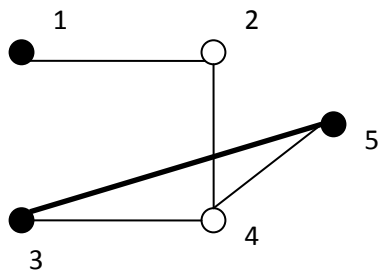


Рисунок 4

Пары антиподальных вершин и соответствующие диаметры графа:

$$\{1,3\} d_{13}^1 = 1 - 2 - 4 - 3;$$

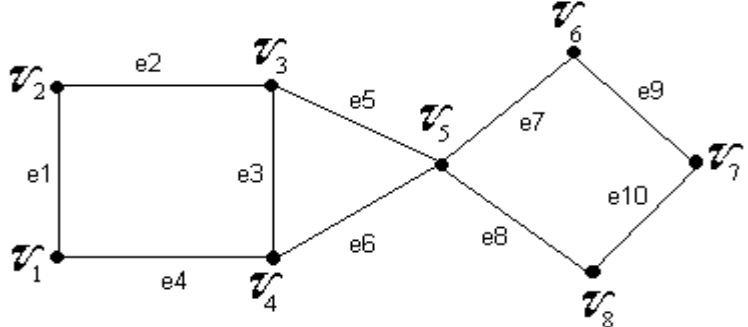
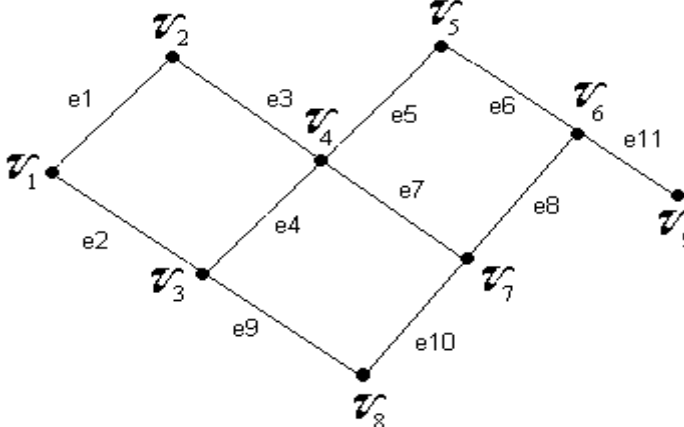
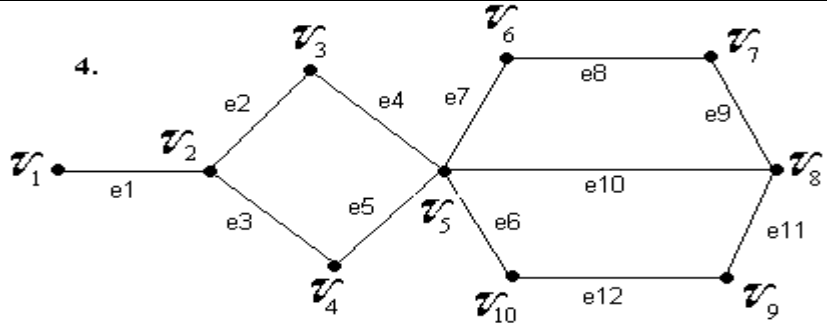
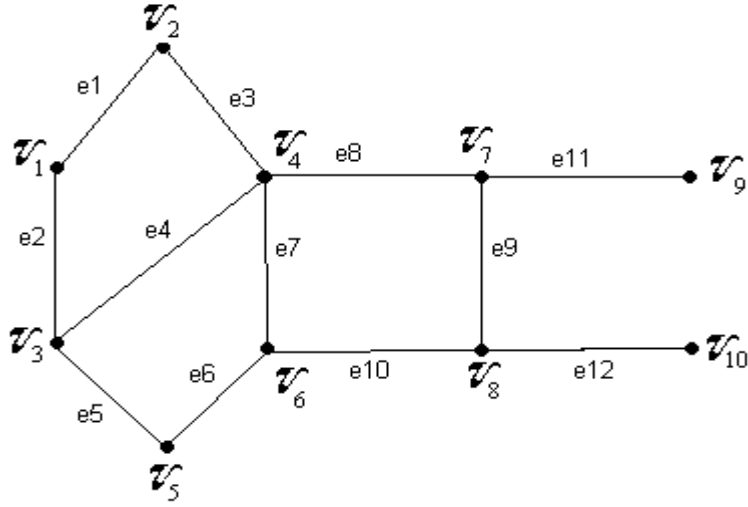
$$\{1,5\} d_{15}^1 = 1 - 2 - 4 - 5;$$

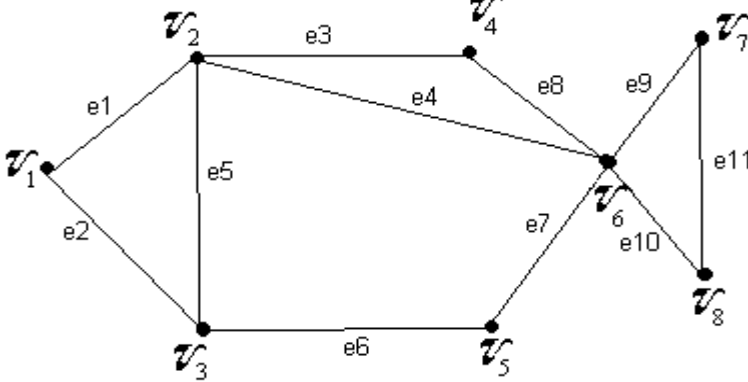
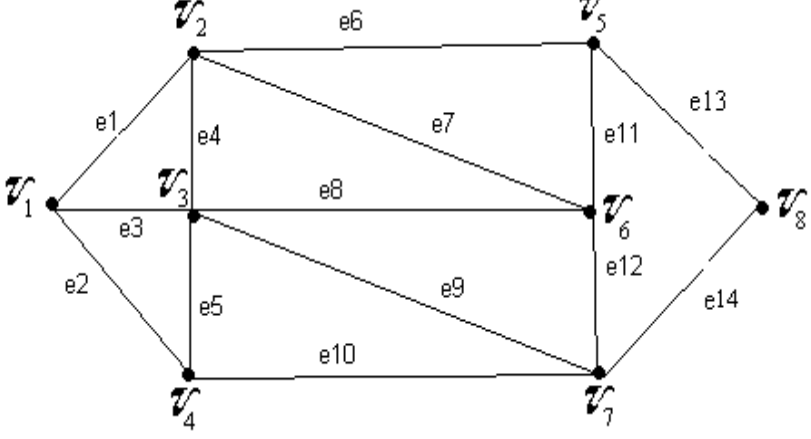
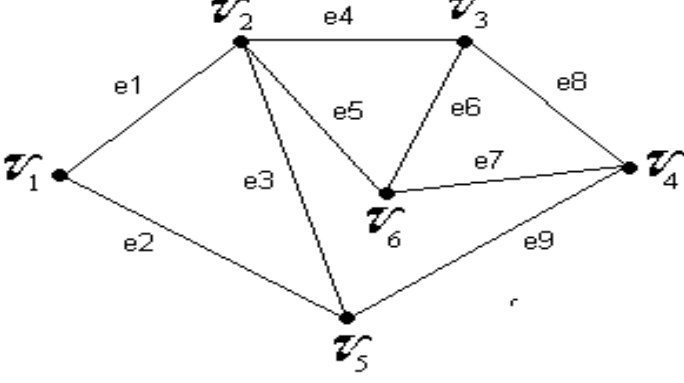
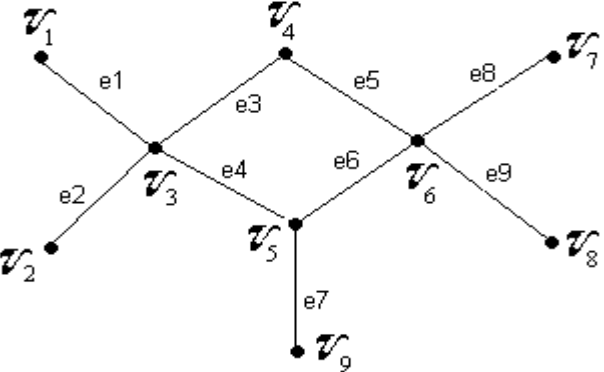
Задание:

Составить программу в консольном режиме, которая:

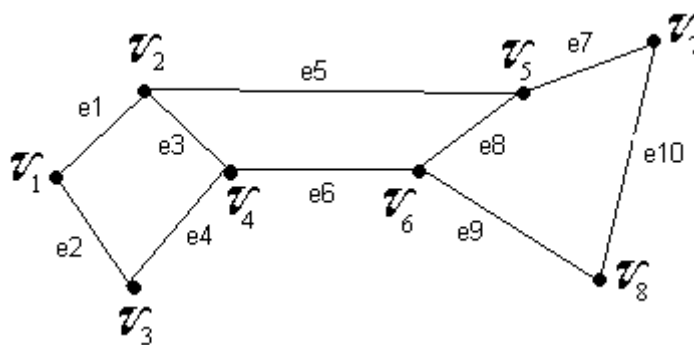
1. Находит матрицу смежности A .
2. С помощью алгоритма фронта волны находит расстояния из вершины v_1 в соответствующую вершину графа.
3. Находит матрицу расстояний.
4. Определяет и выводит диаметр; радиус, центральные и периферические вершины графа.

Вариант	Граф
1	<div style="text-align: center;"> </div> <p>Дан граф.</p> <p>Найти минимальный маршрут от v_1 до v_9.</p>

2	 <p>Дан граф G</p> <p>Найти минимальный маршрут от v_1 до v_7.</p>
3	 <p>Дан граф.</p> <p>Найти минимальный маршрут от v_1 до v_9.</p>
4	<p>4.</p>  <p>Дан граф G</p> <p>Найти минимальный маршрут от v_1 до v_9.</p>
5	 <p>Дан граф G</p> <p>Найти минимальный маршрут от v_1 до v_{10}.</p>

6	 <p>Дан граф G</p> <p>Найти минимальный маршрут от v_1 до v_8.</p>
7	 <p>Дан граф G</p> <p>Найти минимальный маршрут от v_1 до v_8.</p>
8	 <p>Дан граф G</p> <p>Найти минимальный маршрут от v_1 до v_4.</p>
9	 <p>Дан граф G</p> <p>Найти минимальный маршрут от v_1 до v_8.</p>

10



Дан граф G

Найти минимальный маршрут от v_1 до v_8 .

3 Лабораторная работа № 3

Теоретические сведения по теме «Унарные и бинарные операции над графами»

Пусть задан граф $G=(V, E)$.

Подграф G – граф $G_1=(V_1, E_1)$: $V_1 \subseteq V$, $V_1 \neq \emptyset$, $E_1 \subseteq E$. Обозначается: $G_1 \subseteq G$. (G_1 называют также *частью G*.)

Пусть $V_1 \subseteq V$, $V_1 \neq \emptyset$. *Подграфом G, порожденным множеством V_1* , называется граф $G_1=(V_1, E_1)$: E_1 состоит из тех и только тех элементов E , начала и концы которых лежат в V_1 .

Теорема. Пусть $G=(V, E)$ – некоторый граф, а $G_1=(V_1, E_1)$, $V_1 \subseteq V$, $V_1 \neq \emptyset$ – подграф G , порожденный множеством V_1 . Тогда A_{G_1} – подматрица матрицы A_G , находящаяся на пересечении строк и столбцов, соответствующих вершинам из V_1 .

Унарные операции

Добавление к $G=(V, E)$ вершины v : $G_1=(V \cup \{v\}, E)$

Добавление к $G=(V, E)$ дуги (v, w) : $G_1=(V \cup \{v, w\}, E \cup \{(v, w)\})$

Удаление из $G=(V, E)$ дуги (v, w) : $G_1=(V, E \setminus \{(v, w)\})$

Удаление из $G=(V, E)$ вершины v : $G_1=(V \setminus \{v\}, E \setminus \{(u, w) \mid u=v \text{ или } w=v\})$

Отождествление в $G=(V, E)$ вершин v и w (стягивание дуги (v, w))

Дополнение графа $G=(V, E)$: $G_1=(V, E')$

Бинарные операции

Пусть заданы графы $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$.

Пересечение графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$:

$$G_1 \cap G_2 = (V_1 \cap V_2; E_1 \cap E_2), \text{ где } V_1 \cap V \neq \emptyset.$$

Объединение графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$:

$$G_1 \cup G_2 = (V_1 \cup V_2; E_1 \cup E_2).$$

Кольцевая сумма графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$:

$$G_1 \oplus G_2 = (V_1 \cup V_2; E = E_1 \oplus E_2).$$

Соединение графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$:

$$G_1 + G_2 = (V_1 \cup V_2; E_1 \cup E_2 \cup \{\{u, v\} \mid u \in V_1, v \in V_2, u \neq v\}).$$

Произведение графов (декартово произведение графов) $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$: $G_1 \times G_2 = (V_1 \times V_2; E)$, где

$$E = \{ \{ (v_1; v_2), (u_1; u_2) \} \mid (v_1 = u_1 \text{ и } (v_2; u_2) \in E_2) \text{ или } (u_2 = v_2 \text{ и } (v_1; u_1) \in E_1) \}.$$

Композиция графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$: $G_1[G_2] = (V_1 \times V_2; E)$,

где $((v_1; v_2), (u_1; u_2)) \subseteq E$ тогда и только тогда, когда выполняется одно из условий:

1. $(v_1; v_2) \in E_1$;
2. $v_1 = v_2$, то $(u_1; u_2) \in E_2$.

Задание:

Составить программу в консольном режиме, которая:

1. Запрашивает число вершин и число дуг графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$.
2. Запрашивает матрицы смежности графов G_1 и G_2 .
3. Выполняет операцию объединение графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$:
 $G_1 \cup G_2 = (V_1 \cup V_2; E_1 \cup E_2)$.
4. Выполняет операцию пересечение графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$:
 $G_1 \cap G_2 = (V_1 \cap V_2; E_1 \cap E_2)$, где $V_1 \cap V \neq \emptyset$.

5. Выполняет операцию кольцевой суммы графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$: $G_1 \oplus G_2 = (V_1 \cup V_2; E = E_1 \oplus E_2)$.

6. Выполняет операцию соединения графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$:

$$G_1 + G_2 = (V_1 \cup V_2; E_1 \cup E_2 \cup \{\{u, v\} | u \in V_1, v \in V_2, u \neq v\}).$$

7. Выполняет операцию декартового произведения графов $G_1=(V_1, E_1)$ и $G_2=(V_2, E_2)$: $G_1 \times G_2 = (V_1 \times V_2; E)$, где

$$E = \{\{(v_1; v_2), (u_1; u_2)\} | (v_1 = u_1 \text{ и } (v_2; u_2) \in E_2) \text{ или } (u_2 = v_2 \text{ и } (v_1; u_1) \in E_1)\}.$$

8. Выводит их матрицы смежности.

Вариант	Графы
1	
2	
3	
4	

5	
6	
7	
8	
9	
10	

4 Лабораторная работа № 4

Теоретические сведения по теме «Связные графы. Компоненты связности»

Пусть $G=(V, E)$ – неориентированный граф.

Определение. Говорят, что вершина v_j достижима из вершины v_i в графе G , если существует некоторый (v_i, v_j) - маршрут или $v_i = v_j$.

Определение. Граф называется связным, если для любых двух его вершин существует связывающий их маршрут.

Определение. Если G – орграф, то вершина v_j достижима из вершины v_i в графе G , если существует некоторый (v_i, v_j) -путь или $v_i = v_j$.

Определение. Орграф называется сильно связным, если для любых двух его вершин существует путь из одной вершины в другую.

Определение. Псевдографом, ассоциированным с ориентированным псевдографом G , называется граф, множество ребер которого получается из множества E графа G заменой всех упорядоченных пар вершин на неупорядоченные.

Определение. Орграф называется слабосвязным, если связан ассоциированный с ним псевдограф.

Определение. Граф (орграф) G , который не является связным (слабосвязным), называется несвязным.

Определение. Компонентой связности (сильной связности) графа (орграфа) G называется его связный (сильносвязный) подграф, который не является собственным подграфом никакого другого связного (сильносвязного) подграфа графа (орграфа) G .

Утверждение. Если $G_1 = (V_1; E_1)$ – компонента связности орграфа G , то G_1 – подграф, порожденный множеством V_1 .

Замечание: Это предложение справедливо и для произвольных псевдографов.

Теорема. Граф связан тогда и только тогда, когда его нельзя представить в виде объединения двух непересекающихся графов.

Обозначим $p(G)$ количество компонент связности графа.

Определение. Вершину графа, удаление которой приводит к увеличению компонент связности графа, называют разделяющей (точкой сочленения).

Определение. Матрицей связности графа G называется матрица S_G , в которой: $s_{ij} = \begin{cases} 1, & \text{если } v_j \text{ достижима из } v_i, \\ 0, & \text{в противном случае.} \end{cases}$

Определение. Матрицей сильной связности орграфа G называется матрица S_G , в которой: $s_{ij} = \begin{cases} 1, & \text{если существуют } (v_i; v_j)\text{- путь и } (v_j; v_i)\text{- путь,} \\ 0, & \text{в противном случае.} \end{cases}$

Определение. Матрицей достижимости графа G называется матрица T_G , в которой: $t_{ij} = \begin{cases} 1, & \text{если существует } (v_i; v_j)\text{- путь или } i = j, \\ 0, & \text{в противном случае.} \end{cases}$

Определение. Матрицей контрдостижимости графа G называется матрица Q_G , в которой: $q_{ij} = \begin{cases} 1, & \text{если существует } (v_j; v_i)\text{- путь,} \\ 0, & \text{в противном случае.} \end{cases}$

$$Q_G = T_G^T. \text{ Легко установить, что } S_G = T_G * Q_G.$$

Пусть A_G – матрица смежности графа (орграфа) G .

Рассмотрим матрицу $K = A_G + A_G^2 + \dots + A_G^{n-1} + E$. В этой матрице содержится информация о наличии различных маршрутов (путей) в графе (орграфе) G .

С помощью матрицы K построим матрицу M : $m_{ij} = \begin{cases} 1, & \text{если } k_{ij} \neq 0, \\ 0, & \text{если } k_{ij} = 0. \end{cases}$

M – матрица достижимости (связности) орграфа (графа) G . Тогда матрица сильной связности $S_G = M * M^T$.

Предложение 1. Пусть G – орграф, количество его компонент сильной связности $p(G) \geq 2$: G_1, G_2, \dots, G_p . В результате удаления из G вершин, содержащихся в G_1 , получим орграф с $p(G) - 1$ компонентами связности.

Предложение 2. Пусть G' – компонента сильной связности орграфа G , $p(G) \geq 2$. А G'' – орграф, полученный из G удалением вершин, содержащихся в G' , тогда матрицы $A_{G''}$ и $S_{G''}$ получаются из A_G и S_G в результате удаления строк и столбцов, соответствующих вершинам графа G' .

Предложение 3. Единицы i -ой строки или i -го столбца матрицы S_G орграфа G с n вершинами соответствуют вершинам компоненты сильной связности орграфа G , содержащей вершину v_i .

Алгоритм подсчета $p(G)$ и выделения компонент сильной связности G_1, G_2, \dots, G_p

1. $p := 1$; $S_p := S_G$.
2. В множество V_p включаем вершины, соответствующие единицам в первой строке матрицы S_p . В качестве A_{G_p} берем подматрицу матрицы A_G , находящуюся на пересечении строк и столбцов, соответствующих вершинам из V_p .
3. Вычеркнем S_p из строки и столбцы, соответствующие вершинам из V_p . Если в результате не останется ни одной строки (соответственно, ни одного столбца), то p – количество компонент связности графа G , а $A_{G_1}, A_{G_2}, \dots, A_{G_p}$ – их матрицы смежности. В противном случае, оставшиеся после вычеркивания элементы матрицы обозначим S_{p+1} ; $p := p + 1$; и возвращаемся к шагу 2.

Замечание. Сформулированные выше предложения и сам алгоритм остаются справедливы и для произвольных псевдографов после изменений в обозначениях и терминологии.

Утверждение 1. Пусть ρ – отношение достижимости на множестве вершин псевдографа G . Тогда:

- 1) ρ – эквивалентность на V ;
- 2) $v \rho w$ тогда и только тогда, когда вершины v и w принадлежат одной компоненте связности псевдографа G ;
- 3) для любого класса эквивалентности $V_1 \in V/\rho$ псевдограф G_1 , порожденный множеством V_1 , является компонентой связности псевдографа G ;
- 4) для любой компоненты связности $G_1=(V_1, E_1)$ псевдографа G выполняется $V_1 \in V/\rho$.

Утверждение 2. Пусть ρ_1 – отношение достижимости на множестве вершин ориентированного псевдографа G , а $\rho_2 = \rho_1 \cap \rho_1^{-1}$ – отношение двусторонней достижимости на V . Тогда:

- 1) ρ_1 – рефлексивно и транзитивно;
- 2) ρ_2 – эквивалентность на V ;
- 3) $v \rho_2 w$ тогда и только тогда, когда вершины v и w принадлежат одной компоненте сильной связности ориентированного псевдографа G ;
- 4) для любого класса эквивалентности $V_1 \in V/\rho_2$ ориентированный псевдограф G_1 , порожденный множеством V_1 , является компонентой сильной связности ориентированного псевдографа G ;
- 5) для любой компоненты сильной связности $G_1=(V_1, E_1)$ ориентированного псевдографа G выполняется $V_1 \in V/\rho$.

Задание:

Составить программу в консольном режиме, которая по матрице смежности:

1. Восстанавливает ориентированный граф G ;
2. Находит матрицу инцидентности B , предварительно перенумеровав ребра;
3. Находит матрицу достижимости T ;
4. Находит матрицу сильной связности;
5. Находит компоненты сильной связности;

6. Находит матрицы смежности компонентов (сильной) связности.

1	2	3	4	5
$A = \begin{pmatrix} 01001 \\ 00100 \\ 10000 \\ 00101 \\ 00000 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 10001 \\ 00000 \\ 11000 \\ 00101 \\ 00100 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 00000 \\ 10010 \\ 01000 \\ 00100 \\ 10110 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 00001 \\ 10000 \\ 01010 \\ 00000 \\ 00110 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 00000 \\ 10001 \\ 11010 \\ 01000 \\ 00110 \end{pmatrix} \cdot$
6	7	8	9	10
$A = \begin{pmatrix} 00100 \\ 10000 \\ 01000 \\ 10100 \\ 11010 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 01001 \\ 00010 \\ 01000 \\ 10101 \\ 00000 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 00001 \\ 10000 \\ 01000 \\ 01101 \\ 00100 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 01010 \\ 00000 \\ 11000 \\ 00100 \\ 10010 \end{pmatrix} \cdot$	$A = \begin{pmatrix} 00011 \\ 10000 \\ 10000 \\ 00100 \\ 00110 \end{pmatrix} \cdot$

5 Лабораторная работа № 5

Теоретические сведения по теме «Минимальные пути (маршруты) в нагруженных орграфах (графах). Алгоритм Форда-Беллмана. Алгоритм Дейкстры. Алгоритм Флойда-Уоршелла»

Пусть $G=(V, E)$ – орграф. Рассмотрим функцию $f: E \rightarrow \mathbf{R}$, которая каждой дуге $e \in E$ орграфа ставит в соответствие некоторое действительное число $f(e)$. Функцию f называют *весовой функцией*, а орграф G , на дугах которого она определена, называют *нагруженным (взвешенным)*.

Значение $f(e)$ называют *длиной дуги e* . Если π – путь в G , то $f(\pi)$ в нагруженном графе будет обозначать сумму длин входящих в π дуг, причем каждая дуга учитывается столько раз, сколько раз она входит в путь. Величина $f(\pi)$ называется *длиной пути* в нагруженном орграфе.

Замечание. Если $f(e)=1$ для каждого $e \in E$, то $f(\pi)$ выражает длину пути в ненагруженном орграфе.

Определение. Путь (маршрут) в нагруженном орграфе (графе) G из вершины v в вершину w (соединяющий v и w), где $v \neq w$, называется

минимальным, если он имеет минимальную длину среди всех путей (маршрутов) орграфа (графа) G из v в w (соединяющих v и w).

Если в нагруженном орграфе имеются замкнутые пути отрицательной длины, то для заданных вершин v и w орграфа G , где $v \neq w$, минимального пути из v в w может не быть. Таким образом, имеет смысл лишь задача поиска минимальных путей (маршрутов), среди путей (маршрутов), число дуг (ребер) в которых ограничено сверху.

Перечислим некоторые свойства минимальных путей (маршрутов) в нагруженном орграфе (графе) $G=(V, E)$.

1) Если для любого $e \in E$ $f(e) > 0$, то любой минимальный путь (маршрут) является простой цепью.

2) Если $v_1 v_2 \dots v_k$ – минимальный путь (маршрут), то для номеров i, j таких, что $1 \leq i < j \leq k$, путь (маршрут) $v_i v_{i+1} \dots v_j$ также является минимальным.

3) Если $v \dots u w$ – минимальный путь (маршрут) среди путей (маршрутов) из v в w (соединяющих v и w), содержащих не более $k+1$ дуг (ребер), $v \dots u$ – минимальный среди путей из v в u (среди маршрутов, соединяющих v и u), содержащих не более k дуг (ребер).

Итак, рассмотрим задачу поиска минимального пути в нагруженном орграфе G (для нагруженного неорграфа рассуждения аналогичные).

Пусть $G=(V, E)$ – нагруженный орграф, $V=\{v_1, v_2, \dots, v_n\}$, $n \geq 2$.

Обозначим $\lambda_i^{(k)}$ – длину минимального пути из v_1 в v_i , содержащего не более k дуг. Если таких путей нет, то $\lambda_i^{(k)} = \infty$. Кроме того, если каждый $v_i \in V$ считать путем из v_i в v_i нулевой длины, то $\lambda_i^{(k)}$ можно ввести и для $k=0$: $\lambda_i^{(0)} = 0$, $\lambda_i^{(0)} = \infty$ для $i=2, \dots, n$.

Определение. Матрицей длин дуг нагруженного орграфа G называется квадратная матрица D порядка n , в которой $d_{ij} = \begin{cases} f((v_i, v_j)), & \text{если } (v_i, v_j) \in E, \\ \infty, & \text{в противном случае.} \end{cases}$

Значения $\lambda_i^{(k)}$ легко вычислить по формулам:

$$\lambda_i^{(k+1)} = \min_{1 \leq j \leq n} \{ \lambda_j^k + d_{ji} \}, i=2, \dots, n, k \geq 0;$$

$$\lambda_1^{(k+1)} = \min \{ 0, \min_{1 \leq j \leq n} \{ \lambda_j^k + d_{j1} \} \}.$$

Результат оформляется в виде таблицы, где значение $\lambda_i^{(k)}$ размещается в i -ой строке и $k+1$ столбце.

Алгоритм (Форда-Беллмана) нахождения минимального пути в нагруженном орграфе G из вершины v_1 в вершину v_{i_1} ($i_1 \neq 1$)

1. Составить таблицу значений $\lambda_i^{(k)}$, $i=1, \dots, n$, $k=0, \dots, n-1$.
2. Если $\lambda_{i_1}^{(n-1)} = \infty$, то вершина v_{i_1} недостижима из v_1 и алгоритм прекращает свою работу. Если $\lambda_{i_1}^{(n-1)} < \infty$, то минимальный путь из v_1 в v_{i_1} существует и его длина равна $\lambda_{i_1}^{(n-1)}$. Найдем наименьшее $k_1 \geq 1$, при котором $\lambda_{i_1}^{(k_1)} = \lambda_{i_1}^{(n-1)}$, тогда минимальный путь содержит ровно k_1 дуг.

3. Найдем номера $i_2, i_3, \dots, i_{k_1+1}$, такие что пары $(v_{i_2}, v_{i_1}), \dots, (v_{i_{k_1+1}}, v_{i_{k_1}}) \in E$. Кроме того, $f(v_{i_2}, v_{i_1}) = d_{i_2, i_1}; \dots f(v_{i_{k_1+1}}, v_{i_{k_1}}) = d_{i_{k_1+1}, i_{k_1}}$. Это получено с помощью следующих равенств:

$$\begin{aligned} \lambda_{i_2}^{(k_1-1)} + d_{i_2, i_1} &= \lambda_{i_1}^{(k_1)}; \\ \lambda_{i_3}^{(k_1-2)} + d_{i_3, i_2} &= \lambda_{i_2}^{(k_1-1)}; \\ &\dots \\ \lambda_{i_{k_1+1}}^{(0)} + d_{i_{k_1+1}, i_{k_1}} &= \lambda_{i_{k_1}}^{(1)} \end{aligned} \quad (*)$$

Действительно, сложив равенства (*) и учтя, что $\lambda_{i_{k_1+1}}^{(0)} = 0$, $i_{k_1+1} = 1$, $v_{i_{k_1+1}} = v_1$, получим, что $f(v_1, v_2, \dots, v_{i_{k_1}}, v_{i_1}) = \lambda_{i_1}^{(k_1)}$ является минимальной длиной пути.

Замечание.

- 1) Значения номеров i_2, i_3, \dots, i_{k_1} могут быть выделены неоднозначно.

2) Алгоритм Форда-Беллмана можно модифицировать с тем, чтобы определять минимальный путь из v_1 в заданную вершину среди путей, содержащих не более k_0 дуг ($k_0 \geq 1$). Тогда нет необходимости вычислять $\lambda_i^{(n-1)}$, а нужно остановиться на $\lambda_i^{(k_0)}$. При этом не нужно предположения об отсутствии простых контуров отрицательной длины в графе.

3) При соответствующих изменениях в терминологии и обозначениях алгоритм применим и для неориентированных графов. В этом случае условие отсутствия контуров отрицательной длины заменяется условием отсутствия ребер отрицательной длины.

Предложение. Пусть G –орграф с n вершинами и любая вершина v_i ($i = 2, 3, \dots, n$) достижима из v_1 . В орграфе G отсутствуют контуры отрицательной длины тогда и только тогда, когда $\lambda_i^{(n-1)} = \lambda_i^{(n)}$ ($i = 1, 2, \dots, n$).

Алгоритм Дейкстры

Вторым алгоритмом, который рассмотрим, является алгоритм, разработанный Эдсгером Дейкстрой в 1959 году. Этот алгоритм чрезвычайно популярен и используется во многих сферах программирования, таких как нахождение пути для искусственного интеллекта игровых персонажей или реальных роботов. Используется в протоколе Open Shortest Path First для вычисления маршрутов от подсети до подсети. Принцип работы алгоритма таков:

1. Каждой вершине графа, кроме начальной, присваивается метка ∞ , начальной вершине присваивается метка 0. (Аналогично алгоритму Форда-Беллмана)

2. Просматривается каждый сосед стартовой точки, метка соседа улучшается суммой метки начальной точки и ребра до соседа (в самом начале – просто вес ребра).

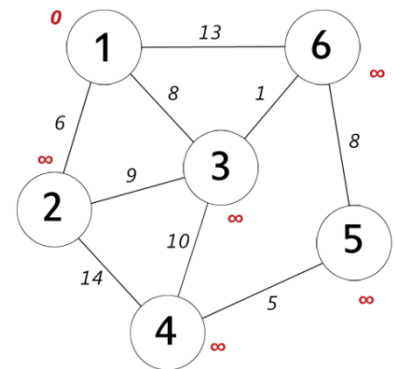
3. Начальная точка помечается как пройденная и больше не будет затрагиваться при последующих вычислениях.

4. Среди всех существующих на графе точек выбирается точка с наименьшей меткой, после чего она становится следующей «начальной» точкой. Те её соседи, помеченные как пройденные, рассматриваться не будут.

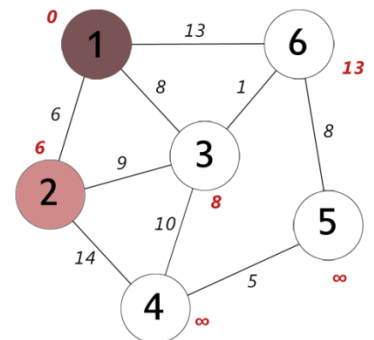
5. В конце действия алгоритма все точки, соединенные с начальной, будут помечены как пройденные, на остальных же будет оставаться бесконечная метка.

Для упрощения понимания алгоритма, рассмотрим его на *примере*:

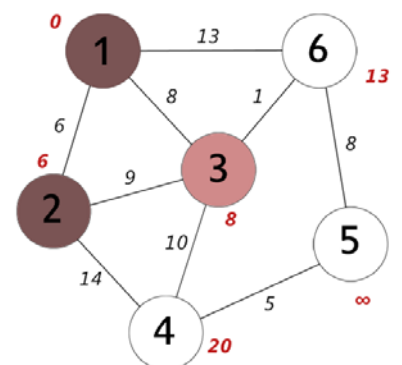
1. Допустим, нам надо найти путь до всех вершин из вершины 1. Для этого ставим метку 0 на 1, на все остальные ставим метку бесконечности.



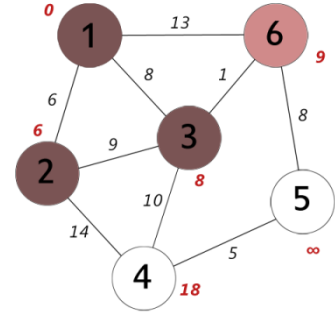
2. Обновляем метки для всех вершин, соседних с вершиной 1, после чего вычеркиваем ее и выбираем новую вершину с минимальной меткой – это будет вершина 2.



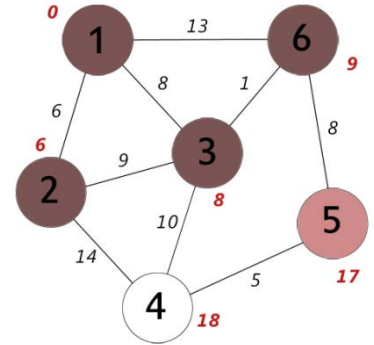
3. Улучшаем метки соседей вершины 2, игнорируя «вычеркнутую» вершину 1, и переходим к следующей вершине, которая будет вершиной 3.



4. Улучшаем метки соседних с вершиной 3 вершин 4 и 6, переходим к метке 6, имеющей минимальную метку в графе равную 9.

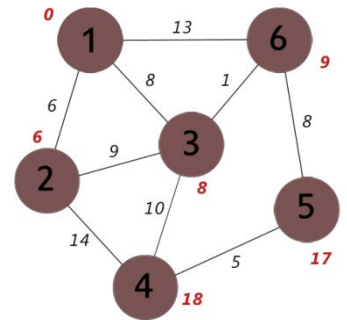


5. Теперь делаем те же шаги с вершиной 6 и переходим к вершине 5.



6. Пытаемся обновить метку вершины 4 из вершины 5, вычеркиваем вершину 5 и вершину 4, т.к. все ее соседи уже были пройдены. Работа алгоритма закончена.

Данный алгоритм будет работать верно только на графе без отрицательных граней.



Пример

Найдем кратчайший путь от V_1 до V_{10} , используя алгоритм Дейкстры, основанный на присвоении меток вершинам и пересчете меток; получаемые при этом постоянные метки и есть длины кратчайших путей.

I. Присвоим вершине V_1 (начальной) метку $\lambda(V_1) = 0$ и будем считать ее постоянной, а всем остальным вершинам — метки ∞ , их будем считать временными. Положим $p = V_1$, $\Gamma(p)$ — множеству вершин, смежных с p и имеющих временные метки.

II. Для всех вершин $V_i \in \Gamma(p)$ меняем метки по правилу:
 $\lambda(V_i) = \min(\lambda(V_i), \lambda(p) + l(p, V_i))$.

III. Среди **всех** вершин с временными метками находим V_i^* , метка которой минимальна и делаем ее постоянной; $p = V_i^*$.

IV. Возвращаемся к II до тех пор, пока вершина V_{10} (конечная) не получит постоянной метки. Постоянные метки вершин и дают длины кратчайших путей от V_1 до этих вершин.

V. Для построения самого пути движемся в обратном направлении от конечной вершины к начальной по убыванию меток так, чтобы разница между метками смежных вершин равнялась длине ребра.

На множестве вершин, смежных с V_{10} , найдем такую V_{P_1} , что

$$\lambda(V_{10}) = \lambda(V_{P_1}) + l(V_{P_1}, V_{10}). \quad (1)$$

Аналогично, на множестве вершин, смежных с V_{P_1} , найдем такую V_{P_2} , что $\lambda(V_{P_1}) = \lambda(V_{P_2}) + l(V_{P_2}, V_{P_1})$, и так далее.

После некоторого числа шагов вершина V_{P_k} совпадает с вершиной V_1 , путь $\mu = (V_1 - V_{P_{k-1}} - \dots - V_{P_1} - V_{10})$ — кратчайший, а его длина $l(\mu) = \lambda(V_{10})$.

Решим задачу по алгоритму Дейкстры (каждый шаг — присвоение одной постоянной метки).

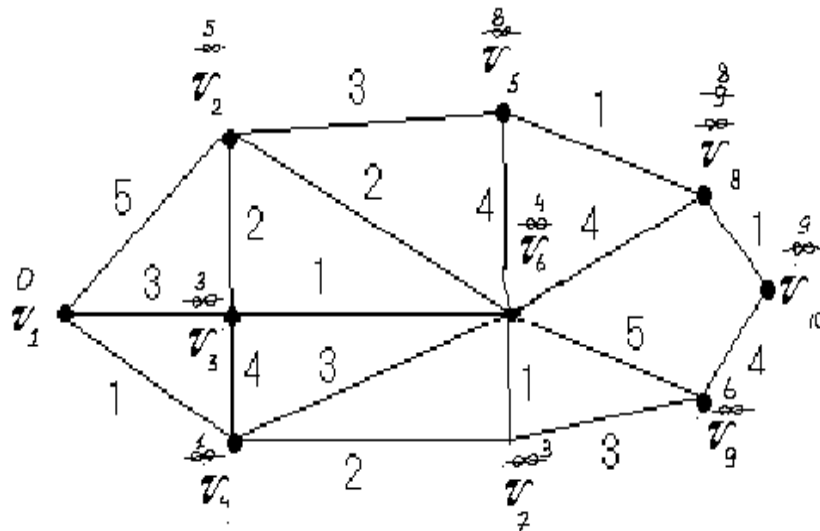


Рисунок 5

1 шаг. $\lambda(V_1) = 0$. $\lambda(V_1) = 0$ — постоянная метка. $\lambda(V_i) = \infty$ ($i = \overline{2,10}$) — временные метки. $p = V_1$, $\Gamma(p) = \{V_2, V_3, V_4\}$.

2 шаг. $\lambda(V_2) = \min(\infty, 0+5) = 5$, $\lambda(V_3) = \min(\infty, 0+3) = 3$,
 $\lambda(V_4) = \min(\infty, 0+1) = 1$. Метка $\lambda(V_4) = 1$ — наименьшая из **всех** временных меток, делаем ее постоянной. $\lambda(V_4) = 1$ — постоянная метка.

$$p = V_4, \quad \Gamma(p) = \{V_3, V_6, V_7\}.$$

3 шаг. $\lambda(V_3) = \min(3, 1+4) = 3$, $\lambda(V_6) = \min(\infty, 1+3) = 4$,
 $\lambda(V_7) = \min(\infty, 1+2) = 3$. Наименьшие из **всех** временных меток имеют вершины V_3 и V_7 . Выбираем, например, V_3 . $\lambda(V_3) = 3$ — постоянная метка.

$$p = V_3, \quad \Gamma(p) = \{V_2, V_6\}.$$

4 шаг. $\lambda(V_2) = \min(5, 3+2) = 5$, $\lambda(V_6) = \min(4, 3+1) = 4$. Метки не изменились, наименьшей из **всех** временных осталась метка 3, принадлежащая вершине V_7 . $\lambda(V_7) = 3$ — постоянная метка.

$$p = V_7, \quad \Gamma(p) = \{V_6, V_9\}.$$

5 шаг. $\lambda(V_6) = \min(4, 3+1) = 4$, $\lambda(V_9) = \min(\infty, 3+3) = 6$. $\lambda(V_6) = 4$ — постоянная метка.

$$p = V_6, \quad \Gamma(p) = \{V_2, V_5, V_8, V_9\}.$$

6 шаг. $\lambda(V_2) = \min(5, 4+2) = 5$, $\lambda(V_5) = \min(\infty, 4+4) = 8$,
 $\lambda(V_8) = \min(\infty, 4+4) = 8$, $\lambda(V_9) = \min(6, 4+5) = 6$. $\lambda(V_2) = 5$ — постоянная метка.

$$p = V_2, \quad \Gamma(p) = \{V_5\}.$$

7 шаг. $\lambda(V_5) = \min(8, 5+3) = 8$. $\lambda(V_9) = 6$ — постоянная метка.

$$p = V_9, \quad \Gamma(p) = \{V_{10}\}.$$

8 шаг. $\lambda(V_{10}) = \min(\infty, 6+4) = 10$. $\lambda(V_8) = 8$ — постоянная метка.

$$p = V_8, \quad \Gamma(p) = \{V_5, V_{10}\}.$$

9 шаг. $\lambda(V_5) = \min(8, 8+1) = 8$, $\lambda(V_{10}) = \min(10, 8+1) = 9$. $\lambda(V_5) = 8$ — постоянная метка.

$$p = V_5, \quad \Gamma(p) = \emptyset.$$

10 шаг. Последняя вершина V_{10} получает последнюю постоянную метку $\lambda(V_{10}) = 9$ — постоянная метка.

Строим путь, начиная с конечной вершины. Из вершин V_9 и V_8 , смежных с V_{10} , выбираем ту, для которой выполняется равенство (1):

для V_8 : $\lambda(V_{10}) = \lambda(V_8) + l(V_8, V_{10})$; $9 = 8 + 1$ верно;

для V_9 : $\lambda(V_{10}) = \lambda(V_9) + l(V_9, V_{10})$; $9 = 6 + 4$ неверно.

Значит, выбираем вершину V_8 .

Из вершин, смежных с V_8 выбираем ту, для которой выполняется равенство (1). Это будет вершина V_6 .

Из вершин, смежных с V_6 выбираем ту, для которой выполняется равенство (1). Это могут быть вершины V_7 и V_3 , оставим V_3 . Вершина V_1 смежна с V_3 и выполняется равенство (1). Значит, кратчайший путь от V_1 до V_{10} : $\mu = (V_1 - V_3 - V_6 - V_8 - V_{10})$; а длина его (вес) равна метке вершины V_{10} , то есть 9. $l(\mu) = 9$.

Алгоритм Флойда-Уоршелла

Данный алгоритм был разработан совместно Робертом Флойдом и Стивеном Уоршеллом в 1962 году. Несмотря на то, что алгоритм назван в их честь, он был опубликован еще в 1959 году Бернардом Ройем, но эта публикация осталась незамеченной.

***Важная особенность:** Для корректной работы алгоритма Флойда-Уоршелла граф не должен иметь петель.*

Поиск кратчайшего пути во взвешенном графе. Алгоритм Флойда-Уоршелла.

Дан ориентированный граф $G=(V,E)$ с матрицей кратчайших путей. Найти кратчайшее расстояние между всеми парами вершин.

В результате работы алгоритма мы должны получить матрицу кратчайших путей $D: \text{array}[1..n, 1..n]$, в которой $D[i,j]$ — кратчайшее расстояние от вершины с номером i до вершины с номером j .

Обозначим матрицу кратчайших путей, содержащих в качестве промежуточных вершины из подмножества $[1 \dots m]$ множества V через D^m . Тогда справедливо следующее:

$$D^{m+1}[i, j] = \min (D^m[i, j], \quad D^m[i, m+1] + D^m[m+1, j])$$

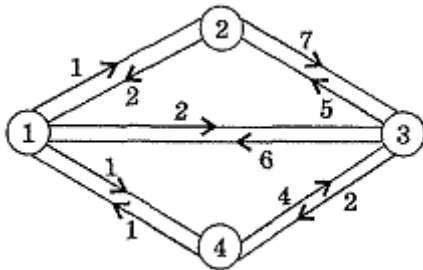
Действительно, если кратчайшее расстояние от i до j содержит вершину $m+1$, то его можно разбить на 2 части: от i до $(m+1)$ и $(m+1)$ до j , а дальше вычислять по рекуррентной формуле.

Отметим, что $D^0 = \text{mass}$, то есть изначально кратчайшие расстояния соответствуют матрице смежности.

Для вывода кратчайшего пути будем использовать тот же принцип «предков», только размерность массива «предков» будет соответствовать размерности D (что справедливо и для прошлых случаев). В целом техника запоминания предков остается та же.

Пример.

На рисунке приведен граф. Используя алгоритм Флойда-Уоршелла найти кратчайшие пути во взвешенном графе



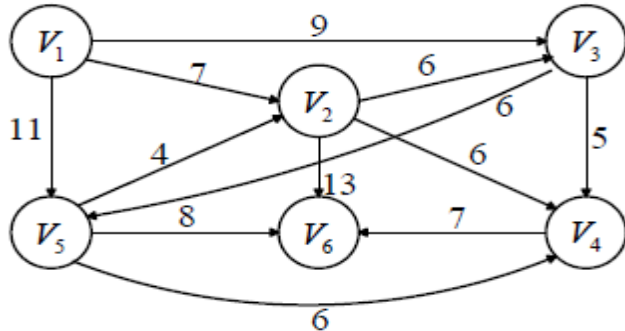
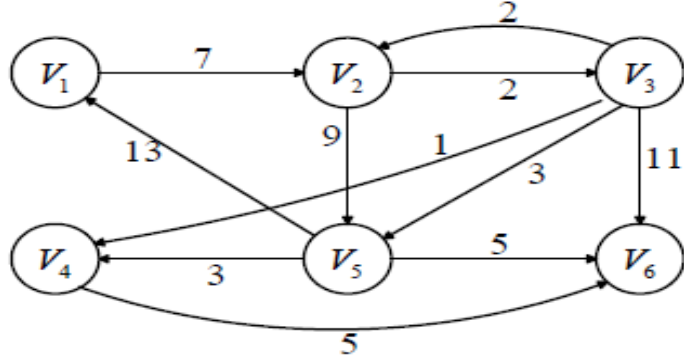
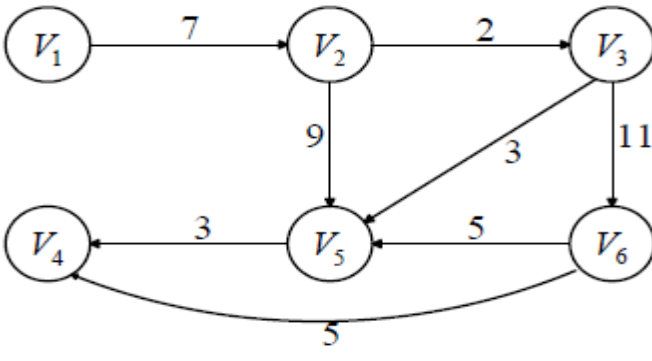
Запишем матрицу кратчайших путей D . По известной формуле найдем все матрицы.

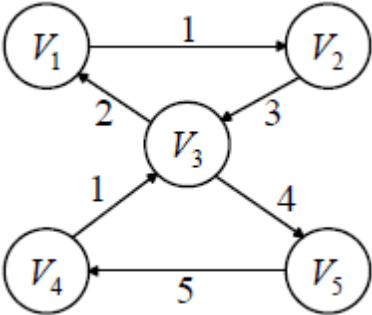
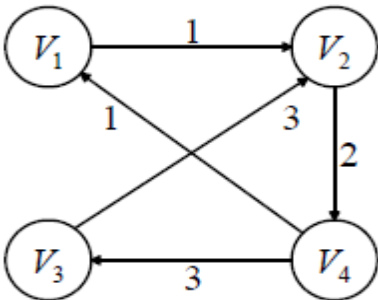
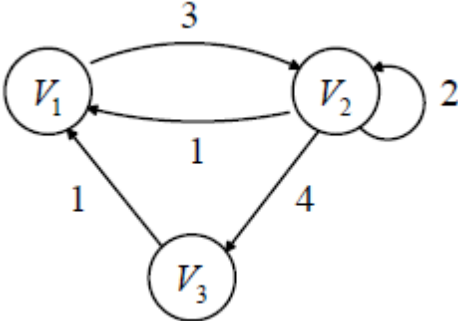
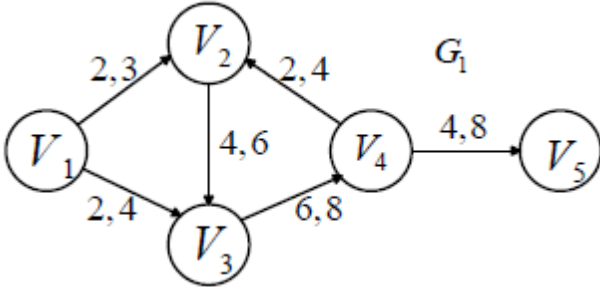
$$D^{m+1}[i, j] = \min (D^m[i, j], \quad D^m[i, m+1] + D^m[m+1, j])$$

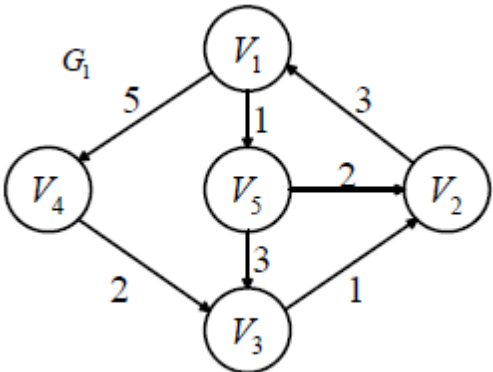
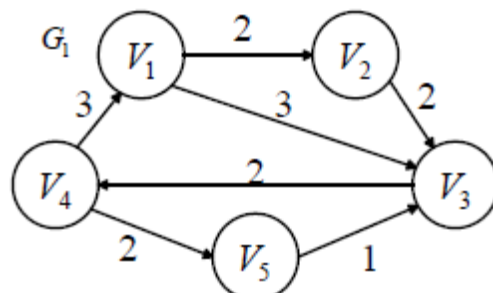
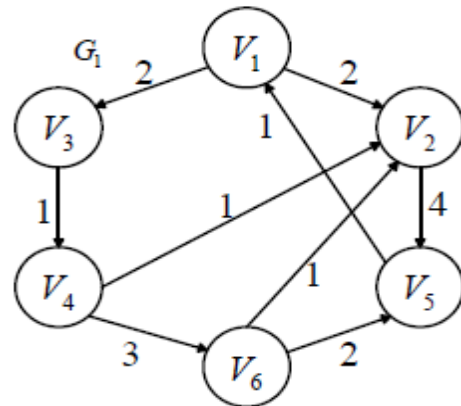
$$D^0 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 7 & \infty \\ 6 & 5 & 0 & 2 \\ 1 & \infty & 4 & 0 \end{pmatrix} D^1 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix}; D^1 = D^2; D^2 = D^3; D^4 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 3 & 4 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix}$$

Задание:

Составить программу в консольном режиме, для которой варианты задания представлены в таблице.

Вариант	Граф
1	<div></div> <p>Дан граф. Используя Алгоритм Дейкстры, найти минимальный путь и длину: 1) от 1 вершины до 4 вершины; 2) от 3 вершины до 6 вершины; 3) от 1 вершины до 6 вершины.</p>
2	<div></div> <p>Дан граф. Используя Алгоритм Дейкстры, найти минимальный путь и длину: 1) от 2 вершины до 4 вершины; 2) от 1 вершины до 5 вершины; 3) от 1 вершины до 4 вершины.</p>
3	<div></div> <p>Дан граф. Используя Алгоритм Дейкстры, найти минимальный путь и длину: 1) от 1 вершины до 4 вершины; 2) от 1 вершины до 6 вершины;</p>

	3) от 2 вершины до 6 вершины.
4	 <p>Дан граф G Используя алгоритм Уоршалла-Флойда, найти минимальное расстояние между всеми парами вершин.</p>
5	 <p>Дан граф G Используя алгоритм Уоршалла-Флойда, найти минимальное расстояние между всеми парами вершин.</p>
6	 <p>Дан граф G Используя алгоритм Уоршалла-Флойда, найти минимальное расстояние между всеми парами вершин.</p>
7	 <p>Дан граф G Используя алгоритм Уоршалла-Флойда, найти минимальное расстояние</p>

	между всеми парами вершин.
8	 <p>Дан граф G Используя алгоритм Уоршалла-Флойда, найти минимальное расстояние между всеми парами вершин.</p>
9	 <p>Дан граф G Используя алгоритм Уоршалла-Флойда, найти минимальное расстояние между всеми парами вершин.</p>
10	 <p>Дан граф G Используя алгоритм Уоршалла-Флойда, найти минимальное расстояние между всеми парами вершин.</p>

6 Лабораторная работа № 6

Теоретические сведения по теме «Эйлеровы графы»

Цикл (контур) и цепь (путь) называются эйлеровыми, если они содержат все ребра (дуги) графа по одному разу.

Признаком возможности построения такого цикла в неорграфе является четность степеней вершин графа. В орграфе полустепени захода и исхода должны быть равны в каждой вершине.

Признаком возможности построения эйлеровой цепи в неорграфе является наличие только двух вершин с нечетными степенями, а в орграфе наличие только двух вершин с разницей входящих и выходящих дуг, равной 1, причем в одной вершине должно быть больше выходящих дуг, а в другой входящих. Одна – будет началом, другая – концом цепи (пути).

Граф, который удовлетворяет условиям Эйлера, называется эйлеровым графом. Пример такого графа показан на рис. У этого графа каждая вершина имеет четную степень, равную 2. Цикл в этом графе $a-e_1-b-e_3-c-e_2-a$ называется эйлеровым, так как он проходит через все ребра по одному разу.

Рассмотрим алгоритмы построения эйлерового цикла и эйлеровой цепи, и их применение для нахождения покрытия графа непересекающимися по ребрам цепями.

1. Алгоритм построения эйлерова цикла

Предполагается, что граф связан и удовлетворяет условиям Эйлера. Все ребра графа не имеют индексов.

1. Положить $i = 1$. Построить простой цикл с началом в некоторой вершине w (в первый раз это будет вершина $v_{\text{нач}}$).

Примечание. В простом цикле вершины не повторяются.

2. Всем ребрам построенного цикла присвоить индекс i . Если ребер без индекса больше нет, то к п. 4, иначе к п. 3.

3. Положить $i=i+1$, среди ребер без индекса выбрать ребро, смежное одному из ребер с индексом. Построить простой цикл из еще не помеченных ребер, содержащий выбранное ребро, и перейти к п. 2.

4. Строим эйлеров цикл с началом в $v_{\text{нач}}$ последовательным обходом ребер графа. На каждом шаге выбираем ребро с наибольшим индексом. Если таких ребер несколько, то берем любое из них.

2. Алгоритм построения эйлеровой цепи

Если требуется построить эйлерову цепь, то

1. Сначала строится простая цепь между вершинами с нечетными степенями. Ее ребрам присваиваются индексы $i = 1$.
2. Если ребер без индекса больше нет, то к п. 4, иначе к п. 3.
3. Положить $i=i+1$, среди ребер без индекса выбрать ребро, смежное одному из ребер с индексом. Построить простой цикл из еще не помеченных ребер, содержащий выбранное ребро, присвоить ребрам этого цикла индекс i и перейти к п. 2.
4. Строим эйлеров цикл с началом в $v_{\text{нач}}$ последовательным обходом ребер графа. На каждом шаге выбираем ребро с наибольшим индексом. Если таких ребер несколько, то берем любое из них.

3. Модифицированный алгоритм построения эйлерова цикла

Практика построения эйлеровых циклов показывает, что в процессе получения простых циклов случайным блужданием при некоторых вершинах образуются частные циклы, которые приходится временно отбрасывать. Чтобы этого не делать, модифицируем алгоритм построения эйлерова цикла (цепи) следующим образом.

Циклы, которые мы пытаемся построить, назовем основными. Циклы, которые возникают при некоторых вершинах в процессе построения основных, назовем частными.

Введем две переменные i и j . В i будем подсчитывать общее количество циклов, в j число частных циклов, появляющихся при построении одного основного цикла.

1. Полагаем $i = 1$ и $j = 1$.
2. Случайным блужданием из ребер без индексов строим основной цикл с началом в некоторой вершине w (в первый раз это будет вершина $v_{\text{нач}}$).
3. Если в процессе построения основного цикла возникает частный цикл при какой-либо вершине, то ребрам этого цикла присваиваем индексы

$in = i + j$ и полагаем $j = j + 1$.

4. Продолжаем строить начатый основной цикл. Если частный цикл не образуется, то к п. 5, иначе к п. 3.

5. После завершения строительства основного цикла, т.е. когда мы вернемся в вершину w , его ребрам присвоим индексы $in = i$ и изменим значения i и j : $i = i + j$, $j = 1$.

6. Если ребер без индекса больше нет, то к п. 8, иначе к п. 7.

7. Среди оставшихся ребер без индексов выбираем ребро, инцидентное одной из вершин ранее построенного цикла (построенных циклов), пусть это будет вершина w , и к п. 2.

8. Строим эйлеров цикл с началом в $v_{\text{нач}}$ последовательным обходом ребер графа. На каждом шаге выбираем ребро с наибольшим индексом. Если таких ребер несколько, то берем любое из них.

Аналогично изменяется и алгоритм построения эйлеровой цепи.

4. Покрытие графа непересекающимися по ребрам цепями

Количество таких цепей равно $k/2$, где k – количество вершин с нечетными степенями, оно всегда четно.

Если $k = 2$, то получается эйлерова цепь.

Если граф не удовлетворяет условиям Эйлера, то для нахождения покрытия графа непересекающимися по ребрам цепями надо дополнить граф до эйлерова графа, построить эйлеров цикл, а затем удалить введенные ребра (и вершины). Если граф эйлеров, то просто строим эйлеров цикл или эйлерову цепь.

Для преобразования графа в эйлеров граф можно

- продублировать все ребра (дуги) графа, или
- добавить фиктивную вершину и соединить ее ребрами с вершинами, имеющими нечетную степень; таких вершин четное число, поэтому степень введенной вершины будет четной и, следовательно, все вершины будут удовлетворять условиям Эйлера, или

– разбить множество вершин с нечетными степенями на пары и соединить вершины каждой пары ребром или маршрутом кратчайшей длины.

Пример.

Пусть задан граф $G(8,13)$, показанный на рис.6. Требуется покрыть граф непересекающимися по ребрам цепями.

Решение:

Введем фиктивную вершину 9 и соединим ее фиктивными ребрами с вершинами 1,2,5,8, имеющими нечетные степени (рис.6). Граф стал эйлеровым.

Первая фаза

Пусть $v_{\text{нач}} = 9$, $i = 1$, $j = 1$ (i – формирователь индексов ребер основных циклов, j – формирователь индексов ребер частных циклов, возникающих при построении очередного основного цикла).

1. Случайно выбирая в каждой попавшейся в маршруте вершине очередное ребро (среди ребер, не имеющих индекса), строим первый основной цикл с началом в вершине 9. Пусть это будет цикл $9p2b3g6j8t9$. При строительстве этого цикла частных циклов не образовалось.

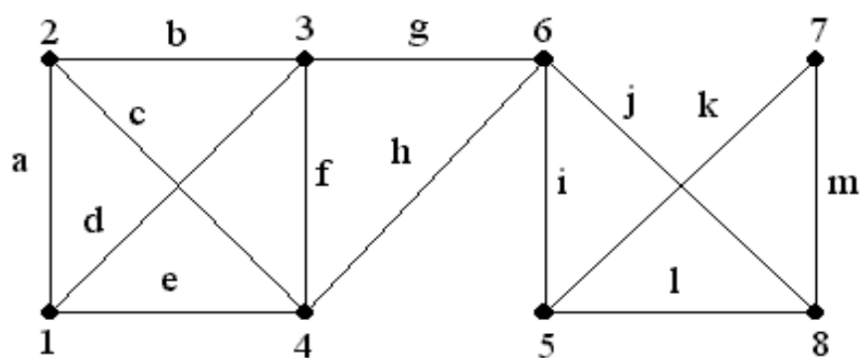


Рисунок 6

2. Ребрам построенного основного цикла присвоим индексы, равные значению $i = 1$ (см. рис. 7).

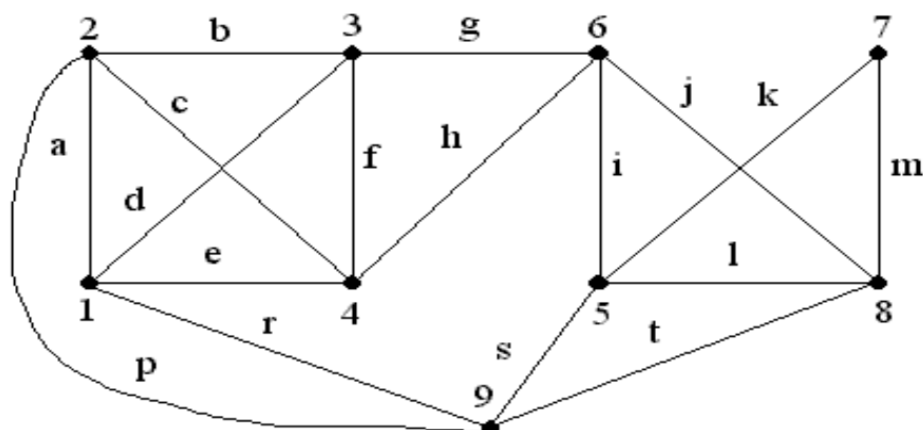


Рисунок 7

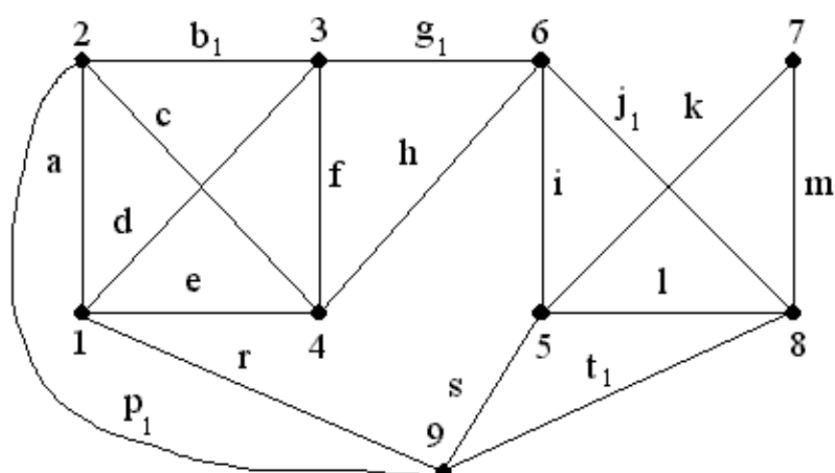


Рисунок 8

3. Выбираем вершину, принадлежащую построенному циклу и имеющую ребра без индексов. Пусть это будет опять вершина 9.

4. Полагаем $i = i + j = 1 + 1 = 2$, $j = 1$. Строим второй основной цикл с началом в вершине 9.

5. Пусть построили такой маршрут $9r1a2c4f3d1$. Замечаем, что при вершине 1 образовался частный цикл $1a2c4f3d1$. Ребрам этого цикла присвоим индекс $i + j = 2 + 1 = 3$ и исключаем их из основного цикла. Переменной j присваиваем новое значение $j = j + 1 = 1 + 1 = 2$.

6. Продолжаем строить основной цикл от вершины 1 – $9r1e4h6i5k7m8l5$ – замечаем, что образовался второй частный цикл при вершине 5 – $5k7m8l5$. Его ребрам присвоим индекс $i + j = 2 + 2 = 4$ и исключаем их из основного цикла, переменной j присвоим новое значение $j = j + 1 = 2 + 1 = 3$.

7. Продолжаем строить основной цикл от вершины 5. Получаем $9r_1e_4h_6i_5s_9$. Частных циклов больше не образовалось.

8. Ребрам полученного основного цикла присвоим индекс, равный $i=2$. Все ребра графа имеют индексы. Первая фаза закончена (см. рис. 9).

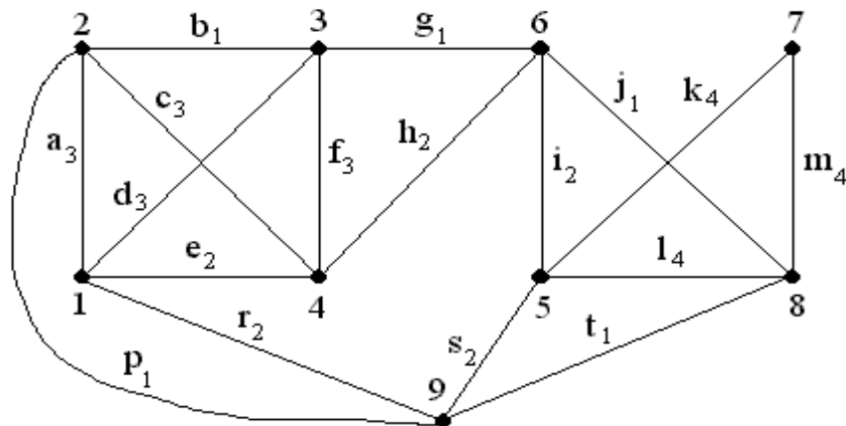


Рисунок 9

Вторая фаза

1. Строим эйлеров цикл с началом в вершине 9:

$9r_21a_32c_34f_33d_31e_24h_26i_25k_47m_48l_45s_29t_18j_16g_13b_12p_19$

2. Удалим введенную вершину 9 и инцидентные ей ребра. Получим

две покрывающие граф цепи, не имеющие общих ребер

$1a_32c_34f_33d_31e_24h_26i_25k_47m_48l_45$ и $8j_16g_13b_12$.

Пример.

Дан оргграф G .

1. Построить матрицу смежности вершин $A_1(G)$.

2. Построить матрицу инцидентности $A_2(G)$.

3. Проверить, является ли граф эйлеровым. Если да, построить эйлеров

ЦИКЛ

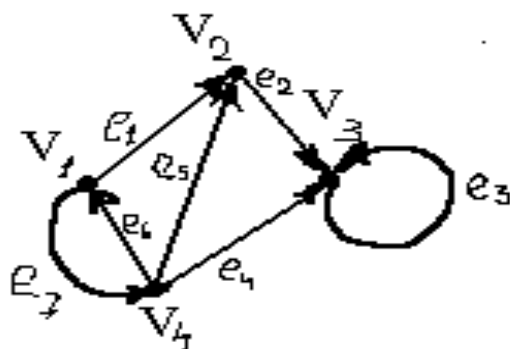


Рисунок 10

Решение. 1. В матрице смежности $A_1(G)$ для ориентированного графа элемент a_{ij} равен количеству дуг с началом в вершине v_i и концом в вершине v_j . В частности, для графа G $a_{ii} = 0$ для $i=1, 2, 4$, $a_{33} = 1$, так как в вершине v_3 имеется петля e_3 . Элементы $a_{14} = a_{41} = 1$, так как вершины v_1 и v_4 соединены двумя противоположно направленными дугами. В остальных случаях $a_{ij} \neq a_{ji}$.

$$A_1(G) = \begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{matrix}.$$

2. В матрице инцидентности $A_2(G)$ ориентированного графа G

$$a_{ij} = \begin{cases} -1, & \text{если } v_j - \text{начало дуги } e_i, \\ 1, & \text{если } v_j - \text{конец дуги } e_i, \\ 0, & \text{если } v_j \text{ не инцидентна дуге } e_i, \\ 2, & \text{если } e_i - \text{петля, инцидентная } v_j. \end{cases}$$

В частности для матрицы инцидентности $A_2(G)$ графа G $a_{33} = 2$, так как e_3 петля, инцидентная v_3 , $a_{13} = 0$, так как дуга e_1 не инцидентна v_3 , $a_{11} = -1$, так как v_1 - начало e_1 , а $a_{12} = 1$, так как v_2 — конец e_1 и так далее.

$$A_2(G) = \begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 \end{matrix} \\ \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{matrix} & \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 \end{pmatrix} \end{matrix}.$$

3. Необходимым и достаточным условием эйлеровости орграфа является его связность и равенство степеней $\rho_1(v)$ и $\rho_2(v)$ для каждой вершины v графа.

Здесь $\rho_1(v)$ - количество дуг инцидентных v , для которых v является началом, а $\rho_2(v)$ - количество дуг, инцидентных v , для которых v является концом. Для графа G $\rho_1(v_1)=2$, а $\rho_2(v_1)=1$, поэтому орграф не является эйлеровым.

Задание:

Построить эйлеров цикл в графе, начиная с первой вершины.

В-т	Задание	В-т	Задание
1		2	
3		4	

Задача коммивояжера (развозчика продукции или бродячего торговца).

Формулировка задачи: коммивояжер должен посетить один и только один раз каждый из n городов и вернуться в исходный пункт.

Пусть $C=[c_{ij}]$ – матрица расстояний между городами. Для составления математической модели задачи обозначим через x_{ij} факт переезда коммивояжером из города i в город j . Поскольку переезд из одного города в другой может осуществляться только один раз, то x_{ij} должны принимать только два значения: 1 или 0, т.е. *булевы* значения. Таким образом:

$$x_{ij} = \begin{cases} 1, & \text{если коммивояжер из города } i \text{ приезжает непосредственно в город } j; \\ 0, & \text{в противном случае.} \end{cases}$$

Математическая модель задачи примет следующий вид:

$$\min Z = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}; \quad (1)$$

$$\sum_{i=1}^n x_{ij} = 1 \quad (j = \overline{1, n}) \quad (2)$$

$$\sum_{j=1}^n x_{ij} = 1 \quad (i = \overline{1, n}) \quad (3)$$

Система ограничений (2) обеспечивает построение маршрута, при котором коммивояжер въезжает в каждый город только один раз, а система (3) – маршрута, когда он выезжает из каждого города только один раз. К сожалению, эти ограничения недостаточны, так как среди допустимых ими решений имеются маршруты, не образующие полный цикл, включающий все города. Устранение подциклов достигается при добавлении системы ограничений: $u_i - u_j + nx_{ij} \leq n-1 \quad (i = \overline{1, n}, j = \overline{1, n}, i \neq j), \quad (4)$

где переменные u могут принимать произвольные действительные значения.

Решение задачи коммивояжера методом ветвей и границ (алгоритм Литтла).

Если считать города вершинами графа, а коммуникации $(i; j)$ – его дугами, то нахождение минимального пути, проходящего один и только один

раз через каждый город с возвращением в исходную точку можно рассматривать как нахождение на графе гамильтонова цикла минимальной длины.

Рассмотрим алгоритм поиска гамильтонова цикла минимальной длины на графе с n вершинами (алгоритм Литтла). Если между вершинами i и j нет дуги, то ставится символ ∞ . Этот же символ ставится на главной диагонали, что означает запрет на возвращение в вершину, через которую уже проходил цикл. Основная идея метода состоит в том, что сначала строят нижнюю границу длин $\varphi_0(\Omega^0)$ множества гамильтоновых циклов Ω^0 . Затем множество циклов Ω^0 разбивается на два подмножества таким образом, что первое подмножество Ω_{ij}^1 состояло из гамильтоновых циклов, содержащих некоторую дугу $(i; j)$, а другое подмножество $\tilde{\Omega}_{ij}^1$ не содержало эту дугу. Для каждого из подмножеств определяются нижние границы по тому же правилу, что и для первоначального множества гамильтоновых циклов. Полученные нижние границы подмножеств Ω_{ij}^1 и $\tilde{\Omega}_{ij}^1$ оказываются не меньше нижней границы для всего множества гамильтоновых циклов, то есть

$$\varphi_0(\Omega^0) \leq \varphi(\Omega_{ij}^1) \equiv \varphi_{ij}^1, \quad \varphi_0(\Omega^0) \leq \varphi(\tilde{\Omega}_{ij}^1) \equiv \tilde{\varphi}_{ij}^1.$$

Сравнивая нижние границы φ_{ij}^1 и $\tilde{\varphi}_{ij}^1$ подмножеств Ω_{ij}^1 и $\tilde{\Omega}_{ij}^1$, можно выделить среди них то, которое с большей вероятностью содержит гамильтонов цикл минимальной длины. Затем одно из подмножеств Ω_{ij}^1 или $\tilde{\Omega}_{ij}^1$ по аналогичному правилу разбивается на два новых Ω_{ij}^2 и $\tilde{\Omega}_{ij}^2$. Для них снова отыскиваются нижние границы φ_{ij}^2 и $\tilde{\varphi}_{ij}^2$ и т.д.

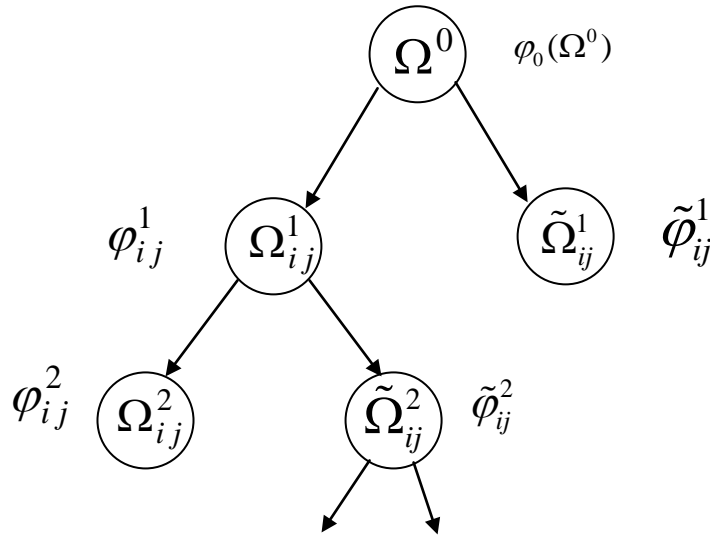


Рисунок 11

Процесс ветвления продолжается до тех пор, пока не отыщется единственный гамильтонов цикл. Его называют *первым рекордом*. Затем просматривают оборванные ветви. Если их нижние границы больше длины первого рекорда, то задача решена. Если же есть такие, для которых нижние границы меньше, чем длина первого рекорда, то подмножество с наименьшей нижней границей подвергают дальнейшему ветвлению, пока не убеждаются, что оно не содержит лучшего гамильтонова цикла. Если же такой найдется, то анализ оборванных ветвей продолжается относительно нового значения длины цикла. Его называют *вторым рекордом*. Процесс решения заканчивается тогда, когда будут проанализированы все подмножества.

Алгоритм Литтла

Пусть известна матрица весов некоторого орграфа, вершины которого занумеруем числами от 1 до n :

$$\Omega = \left(\begin{array}{c|cccc} & 1 & 2 & \dots & n \\ \hline 1 & \infty & \omega_{12} & \dots & \omega_{1n} \\ 2 & \omega_{21} & \infty & \dots & \omega_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n & \omega_{n1} & \omega_{n2} & \dots & \infty \end{array} \right),$$

где ω_{ij} – вес дуги, соединяющей вершины i и j , где $i = \overline{1, n}$, $j = \overline{1, n}$.

Символ ∞ ставится для блокировки дуг графа, которые явно не включаются в гамильтонов цикл, т.е. в расчет не берутся дуги l_{ii} и $\omega_{ii} = \infty$. Можно показать, что оптимальность решения не меняется от прибавления некоторого числа к строке или столбцу матрицы Ω .

Алгоритм Литтла разбивается на несколько шагов.

Шаг 1. Приведение исходной матрицы.

В каждом столбце и в каждой строке матрицы Ω нужно получить хотя бы один нуль. Для этого, например, в первом столбце выбираем минимальный элемент и вычитаем его из всех элементов первого столбца. Аналогично поступаем с остальными столбцами. Если при этом в некоторых строках не появляются нули, то для них осуществляем ту же процедуру.

После этого вычисляем константу приведения φ_0 – сумму минимальных элементов столбцов и строк, которые вычитались. Получаем приведенную матрицу Ω_0 с константой приведения φ_0 .

Шаг 2. Определение степеней нулей.

Находим степени каждого нуля – сумму минимальных элементов строки i_k и столбца j_m , $k = \overline{1, n}, m = \overline{1, n}$ в которых стоит этот нуль, без учета самого нуля. Нуль с максимальной степенью определяет дугу $l(i_1; j_1)$, которая вероятнее всего войдет в гамильтонов цикл. Например, если нуль с максимальной степенью находится на пересечении второй строки и третьего столбца, то дуга $l(2; 3)$, вероятнее всего, войдет в гамильтонов цикл.

Шаг 3. Ветвление.

На самом деле, дуга, соответствующая максимальной степени нуля, может, как входить в гамильтонов цикл, так и не входить в него. Поэтому дальше нужно рассмотреть сразу два случая.

Первый случай. Возможно, дуга $l(i_1; j_1)$ вошла в гамильтонов цикл. Блокируем ее, полагая $\omega_{j_1, i_1} = \infty$. Строку i_1 и столбец j_1 вычеркиваем. Если

требуется, приводим полученную матрицу меньшего порядка $\Omega_1(i_1; j_1)$ с константой приведения $\varphi_1 = \varphi_0 + \Delta_1$.

Второй случай. Дуга $l(i_1; j_1)$ не вошла в гамильтонов цикл. Полагаем $\omega_{i_1, j_1} = \infty$. Если требуется, приводим полученную матрицу $\tilde{\Omega}_1(i_1; j_1)$ с константой приведения $\tilde{\varphi}_1 = \varphi_0 + \tilde{\Delta}_1$.

Если $\varphi_1 < \tilde{\varphi}_1$, то шаги 2, 3 повторяем с матрицей $\Omega_1(i_1; j_1)$.

Если $\varphi_1 > \tilde{\varphi}_1$, то шаги 2, 3 повторяем с матрицей $\tilde{\Omega}_1(i_1; j_1)$. И так до тех пор, пока не дойдем до матрицы второго порядка, содержащей два нуля:

$$\Omega_{n-2}(i_{n-2}; j_{n-2}) = \left(\begin{array}{c|cc} & j_{n-1} & j_n \\ \hline i_{n-1} & 0 & A \\ i_n & B & 0 \end{array} \right),$$

где A и B – некоторые числа или ∞ . Эти нули соответствуют двум последним дугам гамильтонова цикла: $l(i_{n-1}; j_{n-1})$, $l(i_n; j_n)$. При этом $\varphi = \varphi_{n-2}$.

Если $\varphi_{n-2} < \tilde{\varphi}_k$, где $k = 1, 2, \dots, n-2$, то задача решена. Если же для некоторого k_0 получается $\varphi_{n-2} > \tilde{\varphi}_{k_0}$, то всю процедуру следует провести с матрицей $\tilde{\Omega}_{k_0}(i_{k_0}; j_{k_0})$. На последнем этапе получим новое значение функции φ : φ'_{n-2} . Это значение сравниваем с φ_{n-2} , то есть новый процесс продолжается до тех пор, пока новые $\varphi'_k < \varphi_{n-2}$.

Лемма. Если обыкновенный граф $G = (V, E)$ имеет не менее трех вершин и для любых его несмежных вершин a, b выполняется условие $d(a) + d(b) \geq |V|$, то граф G гамильтонов.

Следствие. Если обыкновенный граф $G = (V, E)$ имеет более трех вершин и для любой его вершины a выполняется условие $d(a) \geq \frac{1}{2}|V|$, то граф G гамильтонов.

Пусть $G = (V, E)$ – связный обыкновенный граф и $T = (V, E')$ – остов этого графа.

Фиксируем вершину $a \in V$ и называем ее корнем дерева T . Тогда множество вершин V дерева T разбивается на непересекающиеся уровни – подмножества $V_0, V_1, V_2, \dots, V_k \subset V$ такие что:

- 1) $V_0 = \{a\}$ – верхний уровень состоит из корня a ;
- 2) следующий ниже лежащий уровень V_1 состоит из вершин, смежных с единственной вершиной a верхнего уровня;
- 3) для последующих значений $i = 2, 3, \dots, k$ соответствующий ниже лежащий уровень V_i состоит из вершин, смежных с вершинами предыдущего уровня V_{i-1} .

В результате получаем $r(a) + 1$ уровней, где $r(a)$ – эксцентриситет вершины a . Такое разбиение множества вершин исходного графа G на уровни позволяет организовать упорядоченный перебор всех вершин этого графа следующими двумя способами:

1) *обход графа в глубину* заключается в выборе сначала корня a и затем последовательном просмотре вершин из ниже лежащих слоев, смежных с предыдущими вершинами; если очередная просматриваемая вершина является концевой, то возвращаемся назад до ближайшей вершины с инцидентными ей еще не рассмотренными ребрами и аналогично просматриваем вершины другого, еще не пройденного маршрута в том же порядке;

2) *обход графа в ширину* заключается в выборе сначала корня a и затем последовательном просмотре вершин строго по слоям, перемещаясь сверху вниз.

При обходе всего множества вершин эти обходы равносильны. Однако в случае поиска одной вершины с определенными свойствами, эффективность успешного завершения того или иного обхода вершин графа определяется структурой дерева: если дерево широкое и концевые вершины расположены на сравнительно близких уровнях, то целесообразней искать такую вершину поиском в глубину; если же дерево узкое и концевые

вершины достаточно далеко разбросаны по уровням, то целесообразней искать такую вершину поиском в ширину.

Если при решении задача A разбивается на подзадачи $A(1,1), A(1,2), \dots, A(1, n_1)$, каждая из которых $A(1, i)$, в свою очередь, разбивается на подзадачи $A(2, i, 1), A(2, i, 2), \dots, A(2, i, n_{2,i})$, и так далее, то в результате получается дерево решения задачи A , которое осуществляется путем обхода такого дерева поиском в глубину или в ширину. При этом обход дерева может существенно сокращаться с помощью специального *метода ветвей и границ* за счет искусственного отсечения лишних поддеревьев.

Методы обхода (просмотра) вершин графов.

Пример

В качестве примера возьмем граф на 10 вершинах, заданный следующим списком смежности: 1) 3,4,9; 2) 4,5,7; 3) 1,7; 4) 1,2,6,10; 5) 2,6; 6) 4,5,7,8; 7) 2,3,6,8; 8) 6,7,10; 9) 1,10; 10) 4,8,9.

Начинаем обход графа всегда с первой по номеру вершины. В процессе прохода будем составлять список S просмотренных вершин.

Поиск в глубину.

Шаг 0. Включаем в список просмотренных вершин первую вершину, поскольку мы в ней находимся: $S=\{1\}$.

Шаг 1. Берем из списка смежности последней просмотренной вершины первую, в которой еще не были, и включаем ее в список. Если все вершины включены в список, конец. Если не все вершины просмотрены, а в списке смежности последней просмотренной вершины не просмотренных нет, переходим к шагу 2.

Шаг 2. Возвращаемся назад через те вершины, по которым пришли в последнюю, пока в списке смежности очередной вершины не нашлась еще не просмотренная. Если нашлась, включаем ее в список S и переходим к шагу 1.

Для нашего примера поиск в глубину будет выглядеть так.

Выполняем шаг 1, пока это возможно: $S=\{1,3,7,2,5,6,4,10,9\}$.

Из вершины 9 перейти никуда нельзя, поскольку все вершины из ее списка смежности уже просмотрены. Поэтому переходим к шагу 2, то есть возвращаемся в вершину 10. Из вершины 10 переходим в не просмотренную вершину 8. Имеем $S=\{1,3,7,2,5,6,4,10,9,8\}$. Процесс просмотра завершен.

Поиск в ширину.

Шаг 0. Включаем в список просмотренных первую вершину, поскольку мы в ней находимся: $S=\{1\}$.

Шаг 1. Берем все не просмотренные вершины, смежные с последней включенной, и добавляем их в список: $S=\{1,3,4,9\}$.

Шаг 2. Для каждой вершины, включенной на предыдущем шаге, включаем в список все не включенные вершины из ее списка смежности.

Повторяем шаг 2, пока все вершины не будут включены в список.

Для нашего примера после первого выполнения шага 2 имеем: $S=\{1,3,4,9,7,2,6,10\}$. Поскольку остались не просмотренными вершины 5 и 8, повторяем шаг 2. В результате имеем: $S=\{1,3,4,9,7,2,6,10,8,5\}$.

В среднем поиск в глубину – более быстрый процесс, так как для включения еще не просмотренной вершины при поиске в ширину нам приходится перебирать все вершины, включенные на очередном шаге, а таких вершин может оказаться много. Изобразите самостоятельно схему просмотра вершин в нашем примере при поиске в глубину и поиске в ширину, и вы сможете в этом убедиться. Однако в целом выбор метода обхода зависит от типа графа и самой постановки задачи.

Задание (для вариантов 1-4):

Составить алгоритм и написать программу для графа заданного списком смежности:

- 1) обход графа методом поиска в глубину;
- 2) обход графа методом поиска в ширину;
- 3) выяснить какой способ обхода эффективнее для данного графа?

Задание (для вариантов 5-10):

Составить алгоритм и написать программу нахождения минимального гамильтонова цикла в графе, заданном матрицей весов.

Вариант	Список смежности
1	1) 2,3,4,5; 2) 1,3,5; 3) 1,2,7,8; 4) 1,6,7; 5) 1,2,6; 6) 4,5,11,12; 7) 3,4,8,11; 8) 3,7,9; 9) 8,10; 10) 9,11; 11) 6,7,10,12; 12) 6,11.
2	1) 2,5,6; 2) 1,3; 3) 2,4,5; 4) 3,8,9; 5) 1,3,6,8; 6) 1,5,7; 7) 6,8,12; 8) 4,5,7,9; 9) 4,8,10,12; 10) 9,11; 11) 10,12; 12) 7,9,11.
3	1) 2,3,9,10,12; 2) 1,3; 3) 1,2,4; 4) 3,5,10,11,12; 5) 4 6) 11; 7) 12; 8) 9,12; 9) 1,8,10; 10) 1,4,9,11; 11) 4,6,10; 12) 1,4,7,8.
4	1) 2,3,4,5; 2) 1,8; 3) 1,7; 4) 1,6,7; 5) 1,6; 6) 4,5,11; 7) 3,4,9; 8) 2,10; 9) 7,12; 10) 8,12; 11) 6,12; 12) 9,10,11.
5	$D_G = \begin{pmatrix} \infty & 6 & 4 & 5 & 2 \\ 6 & \infty & 3 & 3,5 & 4,5 \\ 4 & 3 & \infty & 5,5 & 5 \\ 5 & 3,5 & 5,5 & \infty & 2 \\ 2 & 4,5 & 5 & 2 & \infty \end{pmatrix}$
6	$D_G = \begin{pmatrix} \infty & 13 & 7 & 5 & 2 & 9 \\ 8 & \infty & 4 & 7 & 5 & \infty \\ 8 & 4 & \infty & 3 & 6 & 2 \\ 5 & 8 & 1 & \infty & 0 & 1 \\ \infty & 6 & 1 & 4 & \infty & 9 \\ 10 & 0 & 8 & 3 & 7 & \infty \end{pmatrix}$
7	$D_G = \begin{pmatrix} \infty & 5 & 2 & 3 & 3 & 3 \\ 2 & \infty & 1 & 0 & 1 & 3 \\ 2 & 1 & \infty & 3 & 4 & 3 \\ 3 & 1 & 3 & \infty & 3 & 3 \\ 0 & 2 & 6 & 3 & \infty & 1 \\ 3 & 2 & 4 & 3 & 1 & \infty \end{pmatrix}$
8	$D_G = \begin{pmatrix} \infty & 27 & 43 & 16 & 30 & 26 \\ 7 & \infty & 16 & 1 & 30 & 25 \\ 20 & 13 & \infty & 35 & 5 & 0 \\ 21 & 16 & 25 & \infty & 18 & 18 \\ 12 & 46 & 27 & 48 & \infty & 5 \\ 23 & 5 & 5 & 9 & 5 & \infty \end{pmatrix}$

9	$D_G = \begin{pmatrix} \infty & 2 & 3 & 18 & 0 & 1 & 5 \\ 0 & \infty & 1 & 1 & 3 & \infty & 2 \\ 7 & 5 & \infty & 5 & 8 & 5 & \infty \\ 13 & 19 & 11 & \infty & 25 & 20 & 16 \\ 4 & 8 & 5 & 9 & \infty & 0 & \infty \\ 36 & 21 & 49 & 39 & 50 & \infty & 73 \\ 4 & 5 & 10 & 6 & 5 & 9 & \infty \end{pmatrix}$
10	$D_G = \begin{pmatrix} \infty & \infty & 9 & \infty & \infty & 2 & 10 \\ 1 & \infty & \infty & \infty & 1 & 2 & 4 \\ 2 & 1 & \infty & \infty & 1 & \infty & 2 \\ \infty & 1 & 1 & \infty & \infty & 1 & \infty \\ 1 & 2 & \infty & 2 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 & \infty & 6 \\ \infty & 2 & 1 & \infty & 3 & 2 & \infty \end{pmatrix}$

8 Лабораторная работа № 8

Теоретические сведения по теме «Деревья и остовы графов. Алгоритм Краскала. Алгоритм Прима»

Определение. Деревом называется связный граф без циклов.

Теорема. Для связного графа $G = (V, E)$ следующие условия эквивалентны:

- 1) G – дерево;
- 2) любые две вершины графа G соединяются единственной цепью;
- 3) $|E| = |V| - 1$ т.е. число ребер графа G на единицу меньше, чем число его вершин;
- 4) любое ребро графа G является его мостом;
- 5) граф G не содержит циклов, но добавление к нему любого нового ребра приводит к образованию ровно одного простого цикла.

Определение. Граф, компоненты связности которого являются деревьями, называется *лесом*.

Лемма 1. Граф $G = (V, E)$ является лесом в том и только том случае, если он не содержит циклов. Такой лес удовлетворяет условию $|E| = |V| - C(G)$ где $C(G)$ – число компонент связности графа G .

Определение. *Остовом* (или *каркасом*) графа $G = (V, E)$ называется остовной подграф $G' = (V, E')$ этого графа, сохраняющий компоненты связности графа G . Другими словами, граф $G' = (V, E')$ является остовом графа G , если он имеет одинаковые с графом G компоненты связности и каждая такая компонента является деревом, т.е. ограничение графа G' на каждой компоненте связности графа G является деревом.

В частности, для связного графа G с единственной компонентой связности $K=V$ остовной подграф G' является деревом, которое называется *остовным деревом* графа G .

Лемма 2. Пусть граф $G = (V, E)$ имеет n вершин, m ребер и $C(G)$ компонент связности. Тогда для получения остова графа G необходимо удалить из исходного графа $m - n + C(G)$ ребер. Это число называется *циклическим рангом* графа и обозначается $\nu(G) = m - n + C(G)$.

Следствие. Граф в том и только том случае является лесом, если его циклический ранг $\nu(G) = 0$.

Пусть $G = (V, E, c)$ – взвешенный граф, в котором отображение $c: E \rightarrow R$ каждому ребру $e \in E$ ставит в соответствие действительное число $c(e)$ называемое весом этого ребра e . Тогда каждый остов $T = (V, E')$ графа G имеет вес $c(T) = \sum_{e \in E'} c(e)$

Определение. Остов T_0 взвешенного графа G называется *остовом минимального веса*, если выполняется условие $c(T_0) = \min\{c(T): T \text{ — остов графа } G\}$.

Алгоритм Краскала

В качестве примера возьмем неориентированный взвешенный граф со следующей матрицей весов. Диаграмма графа показана на рис.3.18.

$$D = \begin{pmatrix} 0 & 1 & \infty & \infty & 3 & \infty & \infty & 2 \\ 1 & 0 & 2 & 2 & 4 & \infty & \infty & \infty \\ \infty & 2 & 0 & 3 & \infty & 1 & 3 & 4 \\ \infty & 2 & 3 & 0 & 1 & \infty & 2 & 1 \\ 3 & 4 & \infty & 1 & 0 & 3 & \infty & \infty \\ \infty & \infty & 1 & \infty & 3 & 0 & 4 & \infty \\ \infty & \infty & 3 & 2 & \infty & 4 & 0 & 2 \\ 2 & \infty & 4 & 1 & \infty & \infty & 2 & 0 \end{pmatrix}$$

Шаг 1. Упорядочим все ребра по возрастанию весов. Имеем следующую последовательность: (1,2), (3,6), (4,5), (4,8), (1,8), (2,3), (2,4), (4,7), (7,8), (1,5), (3,4), (3,7), (5,6), (2,5), (3,8), (6,7).

Шаг 2. Берем первое слева (самое дешевое) ребро в списке и образуем первую компоненту связности: $G_1 = \{V_1 = \{1, 2\}, E_1 = \{(1, 2)\}\}$.

Шаг 3 (общий). Предположим, что на текущем шаге алгоритма у нас есть уже какое-то количество компонент связности G_1, \dots, G_k . Берем очередное ребро в списке и поступаем с ним следующим образом.

1. Если обе концевые вершины ребра принадлежат какой-то из компонент G_i , то пропускаем его и переходим к следующему.

2. Если одна концевая вершина ребра (u, v) , например, u принадлежит компоненте G_i , то включаем в эту компоненту ребро и новую вершину v – вторую концевую точку ребра.

3. Если обе концевые вершины не принадлежат ни одной из существующих k связных компонент, образуем компоненту G_{k+1} из этого ребра и его концевых вершин.

4. Если концевые вершины ребра принадлежат разным компонентам связности, например, G_s и G_t , то объединяем эти компоненты в одну и добавляем ребро, их соединяющее. Теперь имеем $k=k-1$ компоненту связности. Если $k=1$ и все вершины графа в нее включены, то процесс завершен. Продолжим теперь процесс построения для нашего графа.

В соответствии с условием 3 общего шага выбираем ребра (3,6), (4,5) и образуем две новые компоненты связности: $G_2=\{V_2=\{3,6\}, E_2=\{(3,6)\}\}$, $G_3=\{V_3=\{4,5\}, E_3=\{(4,5)\}\}$. (рис.12 а).

Следующее ребро (4,8) удовлетворяет условию 2, поэтому присоединяем его к третьей компоненте: $G_3=\{V_3=\{4,5,8\}, E_1=\{(4,5),(4,8)\}\}$ (рис. 12 б).

Ребро (1,8) удовлетворяет условию 4, т.е. объединяет две компоненты связности: $G_1=\{V_1=\{1,2,4,5,8\}, E_1=\{(1,2),(4,5),(4,8),(1,8)\}\}$ (рис.12 в).

Ребро (2,3) также удовлетворяет условию 4, в результате чего объединяются все уже построенные компоненты связности: $G_1=\{V_1=\{1,2,3,4,5,6,8\}, E_1=\{(1,2),(4,5),(4,8),(1,8),(3,6),(2,3)\}\}$ (рис.12 г).

Ребро (2,4) пропускаем согласно условию 1 общего шага. Следующее ребро (4,7) позволяет включить в остовное дерево последнюю вершину.

$G_1=\{V_1=\{1,2,3,4,5,6,7,8\}, E_1=\{(1,2),(4,5),(4,8),(1,8),(3,6),(2,3),(4,7)\}\}$ (рис.12 д). Процесс построения завершен. Вес полученного дерева $P=1+1+1+2+1+2+2=10$.

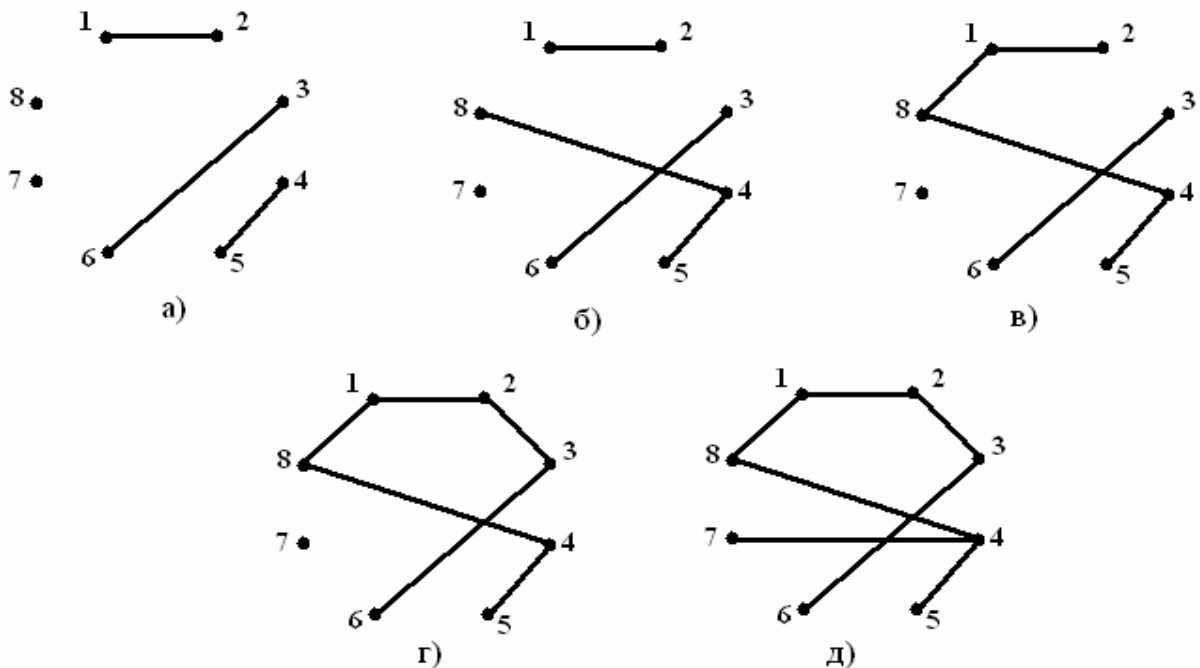


Рисунок 12. - Последовательность построения остовного дерева методом Краскала.

Алгоритм Прима

Некоторым больше нравится именно этот алгоритм. Компонента связности G здесь одна, но нам придется постоянно хранить список «запасных» ребер. Сама работа алгоритма напоминает «поиск в ширину».

Шаг 1. Берем первую вершину v_0 и все смежные с ней, то есть рассматриваем все инцидентные вершине v_0 ребра. Выбираем ребро минимального веса (v_0, v) включаем его в G и переходим в вершину v . Остальные ребра вносим в рабочий список E_0 . Можно сразу упорядочивать ребра в E_0 по возрастанию весов.

Шаг 2(общий).

2.1. Добавляем в список E_0 все ребра, инцидентные последней включенной в G вершине v .

2.2. Из полученного списка выбираем ребро минимального веса, содержащее хотя бы одну еще не включенную в G вершину u . Добавляем его в G вместе с вершиной u и переходим в эту вершину, то есть к шагу 2.1.

2.3. Если все вершины включены, то процесс завершен.

Давайте выполним такое построение для нашего графа с рис.3.18.

1. На первом шаге компонента связности та же: $G=\{V=\{1,2\},E=\{(1,2)\}\}$. Создаем рабочий список из оставшихся ребер, инцидентных вершине 1: $E_0=\{(1,8),(1,5)\}$. Будем упорядочивать ребра в рабочем списке по возрастанию весов слева направо.

2. Последней включенной вершиной является вершина 2. Добавляем инцидентные ей ребра в рабочий список, упорядочивая список по возрастанию весов: $E_0=\{(1,8),(2,3),(1,5),(2,5)\}$. Теперь из этого списка выбираем ребро минимального веса, содержащее еще не включенную вершину 8: $G=\{V=\{1,2,8\},E=\{(1,2),(1,8)\}\}$.

3. Исключаем ребро $(1,8)$ из списка, а в список добавляем ребра, инцидентные вершине 8: $E_0=\{(4,8),(2,3),(7,8),(1,5),(2,5),(3,8)\}$. Из него

выбираем очередное ребро, содержащее не включенную вершину 4:
 $G=\{V=\{1,2,4,8\}, E=\{(1,2), (1,8), (4,8)\}\}$. Продолжаем процесс по той же схеме.

4. $E_0=\{(4,5), (2,3), (7,8), (4,7), (1,5), (2,5), (3,8)\}$; $G=\{V=\{1,2,4,5,8\}, E=\{(1,2), (1,8), (4,8), (4,5)\}\}$.

5. $E_0=\{(2,3), (7,8), (4,7), (1,5), (5,6), (2,5), (3,8)\}$; $G=\{V=\{1,2,3,4,5,8\}, E=\{(1,2), (1,8), (4,8), (4,5), (2,3)\}\}$.

6. $E_0=\{(3,6), (7,8), (4,7), (1,5), (3,4), (5,6), (2,5), (3,8)\}$; $G=\{V=\{1,2,3,4,5,6,8\}, E=\{(1,2), (1,8), (4,8), (4,5), (2,3), (3,6)\}\}$.

7. $E_0=\{(7,8), (4,7), (1,5), (3,4), (5,6), (2,5), (3,8), (6,7)\}$; $G=\{V=\{1,2,3,4,5,6,7,8\}, E=\{(1,2), (1,8), (4,8), (4,5), (2,3), (3,6), (7,8)\}\}$.

Вес полученного дерева $P=1+2+1+1+2+1+2=10$, как и в предыдущем построении, хотя порядок включения ребер другой. И последнее ребро было выбрано другое, но аналогичного веса. Следовательно, полученные остовные деревья равносильны. В целом алгоритм Прима, хотя и кажется многим более понятным, в вычислительном плане менее эффективен, поскольку нам приходится просматривать список ребер по несколько раз.

Пример. Дан граф G

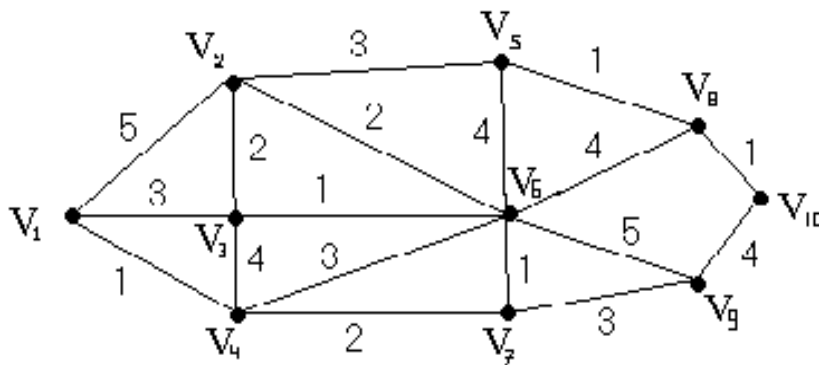


Рисунок 13

1. Построить минимальное соединение графа и найти его вес.
2. Используя алгоритм, найти кратчайший путь от V_1 до V_{10} .

Решение. 1. Для построения минимального соединения, то есть дерева, покрывающего граф и имеющего наименьший вес, используем правило экономичности или алгоритм Краскала.

I) Выбираем ребро с наименьшим весом, например: 1) $V_1 - V_4$.

II) Из оставшихся ребер выбираем ребро с наименьшим весом так, чтобы с уже отобранным оно не образовало цикл.

Выбираем ребра 2) $V_3 - V_6$; 3) $V_6 - V_7$; 4) $V_5 - V_8$; 5) $V_8 - V_{10}$.

Ребер с весом 1 больше нет. Выбираем ребра с весом 2 так, чтобы не получилось цикла: 6) $V_3 - V_2$; теперь взять $V_2 - V_6$ уже нельзя – получается цикл. 7) $V_4 - V_7$. Ребра с весом 2 также закончились. Выбираем ребра с весом 3 так, чтобы не получалось цикла. 8) $V_7 - V_9$; 9) $V_2 - V_5$.

III) Как только количество отобранных ребер должно быть на одно меньше числа вершин, отбор прекращается. Полученное дерево является минимальным соединением.

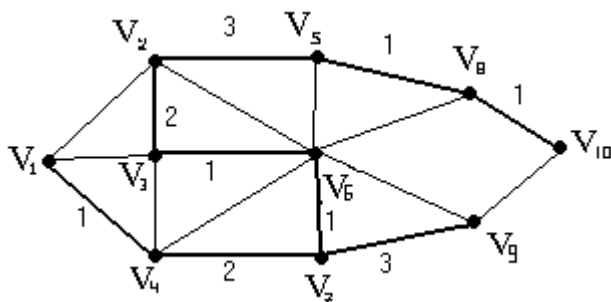


Рисунок 14

Вес минимального соединения графа G

$$l_{\min}(G) = 1 + 1 + 1 + 1 + 1 + 2 + 2 + 3 + 3 = 15.$$

Задание:

Составить алгоритм и написать программу нахождения минимального остовного дерева по алгоритму Краскала для неориентированного графа заданного в таблице, а также вес $w_{T_{\min}}$ полученного дерева.

Составить алгоритм и написать программу нахождения максимального остовного дерева по алгоритму Краскала для неориентированного графа заданного в таблице, а также вес $w_{T_{\max}}$ полученного дерева.

Составить алгоритм и написать программу нахождения минимального остовного дерева по алгоритму Прима для неориентированного графа заданного в таблице, а также вес $w_{T_{min}}$ полученного дерева.

Вариант	Задание	Вариант	Задание
1		2	
3		4	
5		6	
7		8	

9		10	
---	--	----	--

9 Лабораторная работа № 9

Теоретические сведения по теме «Поток в сети. Алгоритм Форда–Фалкерсона»

Рассмотрим сеть $G(V, U)$. Пусть каждой дуге $(i, j) = u_{ij} \in U$, идущей из i в j , поставлено в соответствие неотрицательное число d_{ij} , которое назовем *пропускной способностью* этой дуги (т.е. максимальное количество вещества d_{ij} , которое можно пропустить по дуге u_{ij} за единицу времени). Если вершины i и j соединены ребром (неориентированной дугой), то его заменяют парой противоположно ориентированных дуг u_{ij} и u_{ji} с одинаковой пропускной способностью $d_{ij} = d_{ji}$. Каждой дуге u_{ij} поставим в соответствие еще одно неотрицательное число x_{ij} , которое назовем *поток* по дуге u_{ij} .

Потоком по дуге u_{ij} называется количество вещества x_{ij} , проходящего через дугу u_{ij} в единицу времени. Из физического смысла грузопотока на сети следует неравенство: $0 \leq x_{ij} \leq d_{ij}$, $u_{ij} \in U$, т.е. поток по каждому ребру не может превышать его пропускной способности.

Потоком на сети из I в S называется множество X неотрицательных чисел x_{ij} , т.е. $X = \{x_{ij} \geq 0 \text{ для дуг } u_{ij} \in U\}$, удовлетворяющих следующим условиям: $\sum_i x_{iI} - \sum_j x_{Ij} = -z$; $\sum_i x_{iS} - \sum_j x_{Sj} = z$; $\sum_i x_{ik} - \sum_j x_{kj} = 0, k \in V, k \neq S, k \neq I$.

Данное равенство означает, что для любой вершины k сети, кроме источника

и стока, количество вещества, поступающего в данную вершину, равно количеству вещества, выходящего из нее. Эта связь называется условием сохранения потока, а именно: в промежуточных вершинах потоки не создаются и не исчезают. Следовательно, общее количество вещества, выходящего из источника I , совпадает с общим количеством вещества, поступающего в сток S , что и отражается в предыдущих условиях. Функция z называется величиной (*мощностью*) *потока* на сети и показывает общее количество вещества, которое может пройти по сети. Необходимо найти (построить) максимальный поток из источника I в сток S таким образом, чтобы величина потока x_{ij} по каждой дуге u_{ij} в сети не превышала пропускной способности этой дуги и выполнялось условие сохранения потока, т.е. найти $Z \rightarrow \max$ при предыдущих условиях.

Это типичная задача линейного программирования. Но так как задача о максимальном потоке имеет специфическую структуру, то для нее имеется более эффективный метод решения, чем симплексный.

Понятие разреза в сети

Каждой дуге ставятся в соответствие два числа (d_{ij}, x_{ij}) : первое – пропускная способность; второе – поток. Очевидно, если сеть является путем из I в S , то максимальный поток будет равен минимальной из пропускных способностей дуг, т.е. дуга с минимальной пропускной способностью является узким местом пути. Аналогом узкого места в сети является разрез.

Пусть множество вершин V в сети $G(V, U)$ разбито на два непересекающихся подмножества R, R' , причем объединение этих подмножеств дает множество V .

Разрезом (R, R') в сети $G(V, U)$ называется множество дуг u_{ij} , для которых вершины $v_i \in R$, а вершины $v_j \in R'$. Вообще говоря, разрез (R, R') не совпадает с разрезом (R', R) . Если $I \in R$, а $S \in R'$, то (R, R') будем называть

разрезом, отделяющим источник от стока. Пропускной способностью разреза (R, R') называется величина $C(R, R') = \sum_{u_{ij} \in (R, R')} d_{ij}$.

Рассмотрим сеть на рис. 15

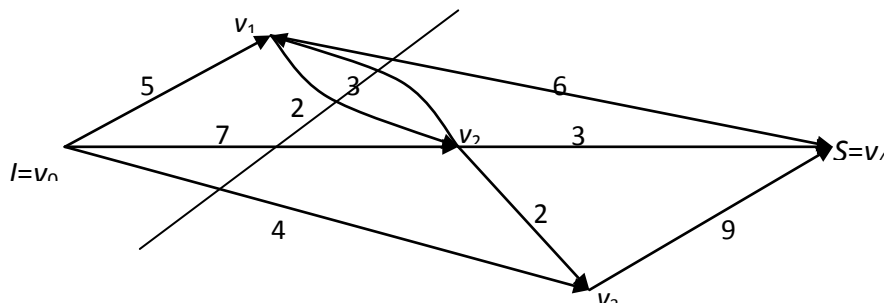


Рисунок 15

В данной сети можно построить 7 разрезов. Рассмотрим, например, разрез (R, R') , где $R = \{I, v_1\}$, $R' = \{v_2, v_3, S\}$. Тогда

$$(R, R') = \{u_{14}, u_{12}, u_{02}, u_{03}\},$$

$$C(R, R') = d_{14} + d_{12} + d_{02} + d_{03} = 6 + 2 + 7 + 4 = 19.$$

Разрез, отделяющий источник от стока и обладающий минимальной пропускной способностью, называется *минимальным разрезом*.

Алгоритм Форда-Фалкерсона построения максимального потока

Теорема Форда-Фалкерсона (о максимальном потоке и минимальном разрезе). Величина максимального потока в сети $G(V, U)$ из I в S равна минимальному разрезу, отделяющего источник от стока.

Если поток x_{ij} по дуге u_{ij} меньше его пропускной способности d_{ij} ($x_{ij} < d_{ij}$), то дуга называется *ненасыщенной*. Если же по дуге u_{ij} поток равен его пропускной способности ($x_{ij} = d_{ij}$), то такая дуга называется *насыщенной*.

Дуга, входящая в некоторый путь, называется *прямой*, если ее направление совпадает с направлением обхода вершин, и *обратной* – в противном случае. Путь из источника I в сток S называется *увеличивающим*

путем, если для прямых дуг выполняется условие $x_{ij} < d_{ij}$, а для обратных дуг выполняется условие $x_{ij} > 0$.

Алгоритм Форда-Фалкерсона построения максимального потока и нахождения минимального разреза заключается в следующем:

1. Находят увеличивающий путь методом расстановки меток.

1.1. Пусть в сети имеется допустимый поток $X = \{ x_{ij} \}$ (например, $\{ x_{ij} \} = 0$). Источник $I = v_0$ получает метку I^+ .

1.2. Просматривают все непомеченные вершины v_i , соседние с v_0 , и присваивают им метку v_0^+ , если u_{0i} – прямая дуга и $x_{0i} < d_{0i}$, и метку v_0^- , если u_{0i} – обратная дуга и $x_{i0} > 0$, и т.д.

1.k. Для каждой вершины v_k , помеченной на предыдущем $(k - 1)$ шаге, просматривают соседние с ней непомеченные вершины v_i . Каждая такая вершина v_i получает метку v_k^+ , если дуга u_{ki} прямая

и $x_{ki} < d_{ki}$, и метку v_k^- , если дуга u_{ik} обратная и $x_{ik} > 0$.

2. Если на k -м шаге не получается пометить сток S , то увеличивающего пути нет и поток в сети является максимальным. Построенный разрез (R, R') , где R – множество помеченных вершин в сети, а R' – множество непомеченных вершин, является минимальным разрезом, и алгоритм заканчивает свою работу. Иначе переходят к пункту 3.

3. Если на k -м шаге сток S оказался помеченным, то выписывают увеличивающий путь P , двигаясь от стока S к вершине, номер которой указан в ее метке; затем от нее к вершине, номер которой указан в ее метке; и в результате приходят к источнику.

4. Выписывают множество P^+ (прямых дуг) и множество P^- (обратных дуг) увеличивающего пути P .

5. Вычисляют для прямых дуг величину $\varepsilon_1 = \min_{u_{ij} \in P^+} (d_{ij} - x_{ij})$,

а для обратных – величину $\varepsilon_2 = \min_{u_{ij} \in P^-} x_{ij}$.

6. Вдоль дуг увеличивающего пути P изменяют поток $X = \{x_{ij}\}$ на величину $\varepsilon = \min \{\varepsilon_1, \varepsilon_2\}$ и получают новый поток $X' = \{x'_{ij}\}$, такой что

$$x'_{ij} = \begin{cases} x_{ij}, u_{ij} \notin P; \\ x_{ij} + \varepsilon, u_{ij} \in P^+; \\ x_{ij} - \varepsilon, u_{ij} \in P^-. \end{cases}$$

Переходят к пункту 1.

Пример

Хозяйственно-питьевой водопровод (сеть на рис.16) соединяет водонапорную башню (источник I) с фермой (стоком S).

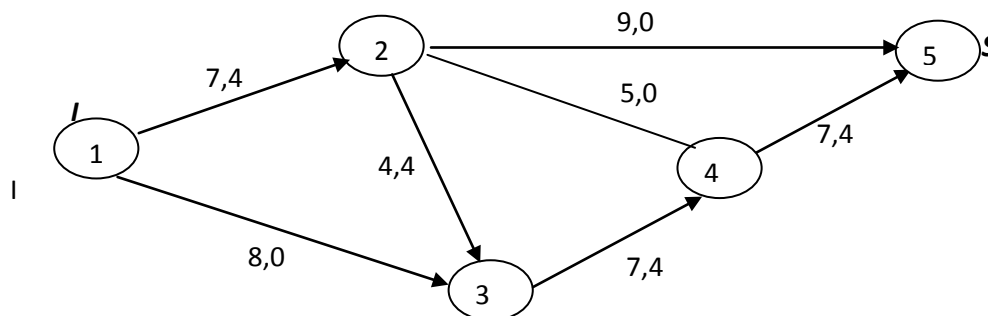


Рисунок 16

Имеется несколько путей, по которым можно доставлять воду из источника в сток. Вершины сети соответствуют пересечениям труб, а ребра и дуги – участкам труб между пересечениями. На сети указаны пропускные способности труб, т.е. максимальное количество воды в м^3 , которое можно пропустить по трубам за 1 ч. Также сформирован начальный поток с мощностью z_0 ($\text{м}^3/\text{ч}$). Какой поток воды максимальной мощности можно пропустить на данном трубопроводе? Указать «узкое место» сети и найти его пропускную способность.

Решение. А. Вершины 2 и 4 соединены ребром (неориентированной дугой), поэтому надо заменить его парой противоположно ориентированных дуг u_{24} и u_{42} с одинаковой пропускной способностью $d_{24} = d_{42} = 5$ ($\text{м}^3/\text{ч}$) (рис.17).

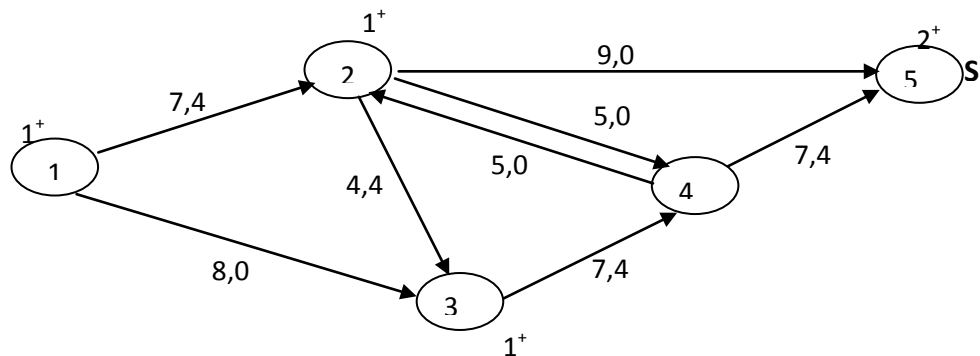


Рисунок 17

Применим алгоритм Форда–Фалкерсона для построения максимального потока и нахождения минимального разреза.

1. Найдем увеличивающий путь методом расстановки меток.

1.1. В сети имеется допустимый поток:

$$z_0 = - \sum_i x_{iI} + \sum_j x_{Ij} = -0 + (4 + 0) = 4 \text{ (м}^3/\text{ч)}.$$

(Сколько воды вытекло из источника I , столько и должно втечь в сток S). Источник $I = 1$ получает метку 1^+ .

1.2. Просматриваем все непомеченные вершины, соседние с 1. Это вершины 2 и 3. Присваиваем вершине 2 метку 1^+ , так как u_{12} – прямая дуга и $x_{12} < d_{12}$ ($4 < 7$). Вершине v_3 также присваиваем метку 1^+ , поскольку u_{13} – прямая дуга и $x_{13} < d_{13}$ ($0 < 8$).

1.3. Теперь просматриваем все непомеченные вершины, соседние с 2. Это вершины 4 и 5. Присваиваем вершине 5 метку 2^+ , так как u_{25} – прямая дуга и $x_{25} < d_{25}$ ($0 < 9$). Поскольку вершина 5 – это сток S , то на данном этапе вершину 4 можно не рассматривать.

2. Так как сток оказался помеченным, то выписываем увеличивающий путь P_1 , двигаясь от стока S к вершине 2, номер которой указан в ее метке; затем от нее к вершине 1, номер которой указан в метке. Таким образом, приходим к источнику I . P_1 : $1 - 2 - 5$.

3. Выписываем множество P^+ (прямых дуг) и множество P^- (обратных дуг): $P^+ = \{u_{12}, u_{25}\}$, а $P^- = \emptyset$ (пустое множество), поскольку обратные дуги в увеличивающий путь не входят.

4. Для прямых дуг вычисляем величину

$$\varepsilon_1 = \min_{u_{ij} \in P^+} (d_{ij} - x_{ij}) = \min\{d_{12} - x_{12}; d_{25} - x_{25}\} = \min\{7 - 4; 9 - 0\} = 3.$$

Так как обратных дуг нет, то величину ε_2 не рассматриваем.

5. Вдоль дуг увеличивающего пути P_1 (рис. 18) изменяем поток $z_0 = 4$ ($\text{м}^3/\text{ч}$) на величину $\varepsilon = \varepsilon_1 = 3$ и получаем новый поток

$$z_1 = z_0 + \varepsilon = 4 + 3 = 7 \text{ (м}^3/\text{ч)}, \text{ такой что}$$

$$x_{12} := x_{12} + \varepsilon = 4 + 3 = 7, u_{12} \in P^+;$$

$$x_{25} := x_{25} + \varepsilon = 0 + 3 = 3, u_{25} \in P^+.$$

Остальные x_{ij} остаются без изменений, так как не входят в увеличивающий путь P_1 .

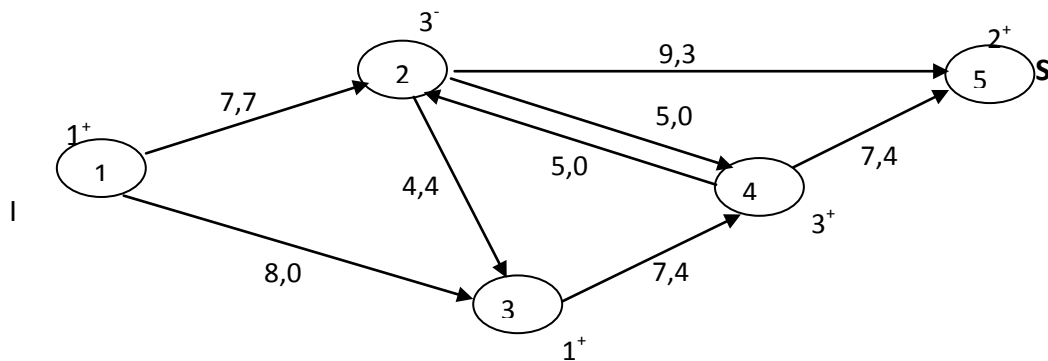


Рисунок 18

В. Опять применим алгоритм Форда–Фалкерсона для построения максимального потока и нахождения минимального разреза.

1. Найдем увеличивающий путь методом расстановки меток.

1.1. Источник $I = 1$ получает метку 1^+ .

1.2. Просматриваем все непомеченные вершины, соседние с 1. Это 2 и

3. Вершине 2 нельзя присвоить метку 1^+ , так как u_{12} – прямая дуга, но $x_{12} = d_{12}$ ($7 = 7$). А вершине 3 присваиваем метку 1^+ , поскольку u_{13} – прямая дуга и $x_{13} < d_{13}$ ($0 < 8$).

1.3. Просматриваем все непомеченные вершины, соседние с 3. Это 2 и

4. Вершина 2 получает метку 3^- , так как дуга u_{32} обратная и $x_{23} = 4 > 0$. А вершине 4 присваиваем метку 3^+ , поскольку u_{34} – прямая дуга и $x_{34} < d_{34}$ ($4 < 7$).

1.4. Потом просматриваем все непомеченные вершины, соседние с 2. Это 4 и 5. Присваиваем вершине 5 метку 2^+ , так как u_{25} – прямая дуга и $x_{25} < d_{25}$ ($3 < 9$). Поскольку вершина 5 – это сток S , то на данном этапе вершину 4 можно не рассматривать.

2. Так как сток оказался помеченным, то выписываем увеличивающий путь P_2 , двигаясь от S к вершине 2, номер которой указан в ее метке; затем от нее – к вершине 3, номер которой указан в метке, и т.д. пока не придем к источнику: $P_2: 1 - 3 - 2 - 5$.

3. Выписываем множество P^+ (прямых дуг) и множество P^- (обратных дуг): $P^+ = \{u_{13}, u_{25}\}$, $P^- = \{u_{32}\}$.

4. Вычисляем для прямых дуг:

$$\varepsilon_1 = \min_{u_{ij} \in P^+} (d_{ij} - x_{ij}) = \min\{d_{13} - x_{13}; d_{25} - x_{25}\} = \min\{8 - 0; 9 - 3\} = 6,$$

$$\text{для обратных: } \varepsilon_2 = \min_{u_{ij} \in P^-} x_{ij} = x_{23} = 4.$$

5. Вдоль дуг увеличивающего пути P_2 (рис. 19) изменяем поток

$z_1 = 7$ ($\text{м}^3/\text{ч}$) на величину $\varepsilon = \min\{\varepsilon_1, \varepsilon_2\} = \min\{6, 4\} = 4$ и получаем новый поток $z_2 = z_1 + \varepsilon = 7 + 4 = 11$ ($\text{м}^3/\text{ч}$), такой что:
 $x_{13} := x_{13} + \varepsilon = 0 + 4 = 4$, $u_{13} \in P^+$; $x_{25} := x_{25} + \varepsilon = 3 + 4 = 7$, $u_{25} \in P^+$;

$$x_{23} := x_{23} - \varepsilon = 4 - 4 = 0, u_{32} \in P^-.$$

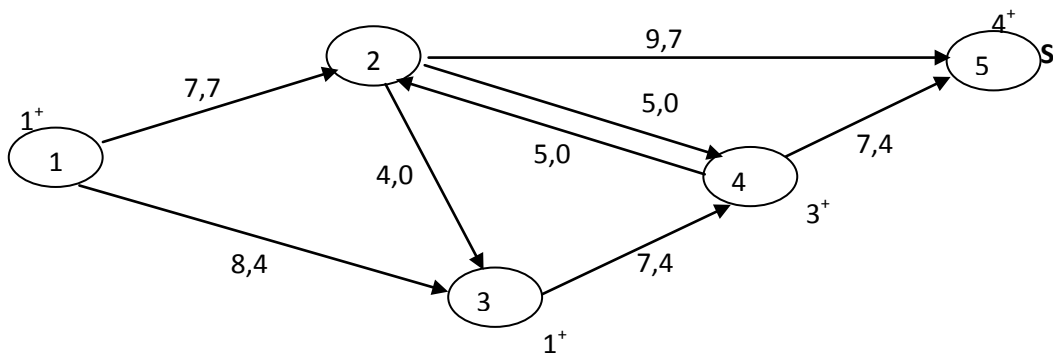


Рисунок 19

Остальные x_{ij} остаются без изменений, так как не входят в увеличивающий путь P_2 .

С. Рассуждая, как было написано выше, расставляем метки и выписываем третий увеличивающий путь: $P_3: 1 - 3 - 4 - 5$.

Записываем множества прямых и обратных дуг: $P^+ = \{u_{13}, u_{34}, u_{45}\}$, $P^- = \emptyset$.

Вычисляем для прямых дуг:

$$\begin{aligned} \varepsilon_1 &= \min_{u_{ij} \in P^+} (d_{ij} - x_{ij}) = \min\{d_{13} - x_{13}; d_{34} - x_{34}; d_{45} - x_{45}\} = \\ &= \min\{8 - 4; 7 - 4; 7 - 4\} = 3. \end{aligned}$$

Так как обратных дуг нет, то ε_2 не вычисляем.

Вдоль дуг увеличивающего пути P_3 (рис. 20) изменяем поток $z_2 = 11$ ($\text{м}^3/\text{ч}$) на величину $\varepsilon = \varepsilon_1 = 3$ и получаем новый поток $z_3 =$

$$\begin{aligned} x_{13} &:= x_{13} + \varepsilon = 4 + 3 = 7, \quad u_{13} \in P^+; \\ z_2 + \varepsilon &= 11 + 3 = 14 \text{ (м}^3/\text{ч)}, \text{ такой что } x_{34} := x_{34} + \varepsilon = 4 + 3 = 7, \quad u_{34} \in P^+; \\ x_{45} &:= x_{45} + \varepsilon = 4 + 3 = 7, \quad u_{45} \in P^+. \end{aligned}$$

Остальные x_{ij} остаются без изменений, поскольку не входят в увеличивающий путь P_3 .

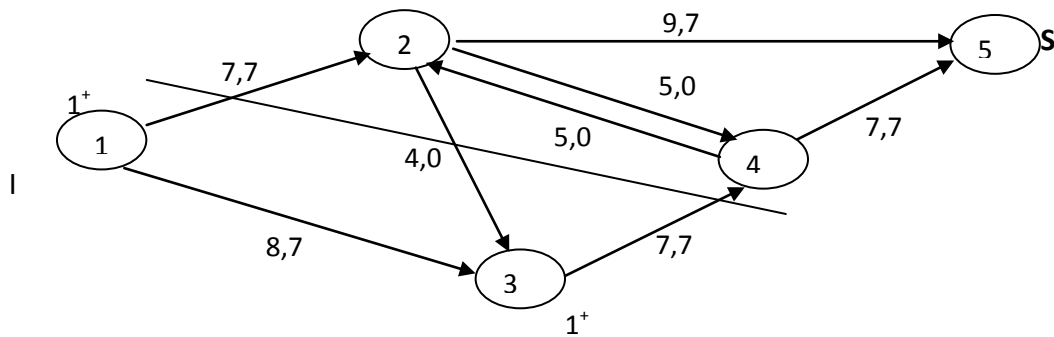


Рисунок 20

D. 1. Снова находим увеличивающий путь методом расстановки меток.

1.1. Источник $I = 1$ получает метку 1^+ .

1.2. Просматриваем все непомеченные вершины, соседние с 1. Это 2 и

3. Вершине 2 нельзя присвоить метку 1^+ , так как u_{12} — прямая насыщенная дуга, поскольку $x_{12} = 7 = d_{12}$. А вершине 3 присваиваем метку 1^+ , так как u_{13} — прямая дуга и $x_{13} < d_{13}$ ($7 < 8$).

1.3. Затем просматриваем все непомеченные вершины, соседние

с 3. Это 2 и 4. Вершине 2 нельзя присвоить метку 3^- , так как дуга u_{32} обратная, но $x_{23} = 0$. А вершине 4 нельзя присвоить метку 3^+ , поскольку u_{34} – прямая дуга, но $x_{34} = d_{34}$ ($7 = 7$).

2. Так как мы не можем пометить сток S , то увеличивающего пути нет и поток в сети является максимальным: $z_{\max} = z_3 = 14$ ($\text{м}^3/\text{ч}$).

Пусть R – множество помеченных вершин в сети, т.е. $R = \{1, 3\}$, а R' – множество непомеченных вершин, т.е. $R' = \{2, 4, 5\}$. Тогда построенный разрез $(R, R') = \{u_{12}, u_{34}\}$ является минимальным (дуга u_{23} в разрез не входит, так как ее начало – непомеченная вершина, а конец – помеченная). И алгоритм заканчивает свою работу.

Минимальный разрез (R, R') является «узким местом» сети. Найдем его пропускную способность: $C(R, R') = \sum_{u_{ij} \in (R, R')} d_{ij} = d_{12} + d_{34} = 7 + 7 = 14$ ($\text{м}^3/\text{ч}$),

что совпадает с величиной максимального потока воды в водопроводе.

Проведем анализ сети. Проверим условие сохранения потока на примере 2-й вершины. Известно, что в промежуточных вершинах пути потоки не создаются и не исчезают, т.е. $\sum_i x_{i2} - \sum_j x_{2j} = 0$.

Действительно, $(x_{12} + x_{42}) - (x_{23} + x_{24} + x_{25}) = (7 + 0) - (0 + 0 + 7) = 0$ ($\text{м}^3/\text{ч}$).

Покажем также, что общее количество воды, вытекающей из источника I (из водонапорной башни), совпадает с общим количеством воды, поступающей в сток S (на ферму), т.е. $-\sum_i x_{iI} + \sum_j x_{Ij} = \sum_i x_{iS} - \sum_j x_{Sj} = z_{\max} \Rightarrow$

$-0 + (x_{12} + x_{14}) = (x_{25} + x_{45}) - 0 \Rightarrow 7 + 7 = 7 + 7 = 14$ ($\text{м}^3/\text{ч}$).

Задание :

Постановка задачи. Хозяйственно-питьевой водопровод (сеть) соединяет источник I со стоком S . Имеется несколько путей, по которым можно доставлять воду из источника в сток. Вершины сети соответствуют пересечениям труб, а ребра и дуги – участкам труб между пересечениями. На сети указаны пропускные способности труб, т.е. максимальное количество

воды в м^3 , которое можно пропустить по трубам за 1 час. Также сформирован начальный поток с мощностью z_0 ($\text{м}^3/\text{ч}$). Какой поток воды максимальной мощности можно пропустить по данному трубопроводу?

Требуется:

- 1) посчитать мощность начального потока воды z_0 ($\text{м}^3/\text{ч}$);
- 2) построить на сети поток воды максимальной мощности z_{\max} ($\text{м}^3/\text{ч}$), направленный из источника I к стоку S;
- 3) указать «узкое место» сети и найти его пропускную способность.

Вариант	Граф
1	
2	
3	
4	

5	
6	
7	
8	
9	
10	

Список использованных источников

- 1 Горбатов, В.А. Дискретная математика: учебник для студентов вузов/ В.А. Горбатов, А.В. Горбатов, М.В. Горбатова - Москва: АСТ, Астрель, 2003. - 447 с.
- 2 Иванов, Б. Н. Дискретная математика. Алгоритмы и программы. Издательство: Физматлит, 2007. — 408 с. ISBN - 978-5-9221-0787-7
- 3 Кузнецов, О.П. Дискретная математика для инженера / О.П. Кузнецов. - 3-е изд., Санкт Петербург. - Лань, 2004. - 394 с.
- 4 Новиков, Ф.А. Дискретная математика для программистов: учебник для вузов / Ф.А. Новиков. – 3-е изд. – М.: Питер, 2009.
- 5 Судоплатов, С.В. Дискретная математика / С.В. Судоплатов, Е.В. Овчинникова. – М.: ИНФРА-М; Новосибирск: Изд-во НГТУ, 2005.
- 6 Хаггарт, Р. Дискретная математика для программистов / Р. Хаггарт. – М.: ТЕХНОСФЕРА, 2005. – 320 с.
- 7 Яблонский, С.В. Введение в дискретную математику: учеб. пособие для вузов/ С.В. Яблонский. - 3-е изд., стер. - Москва: Высшая школа, 2001. – 384 с.