Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Омский государственный технический университет»

# ИСПОЛЬЗОВАНИЕ MATLAB ПРИ РЕШЕНИИ ОПТИМИЗАЦИОННЫХ ЗАДАЧ НА ГРАФАХ

Методические указания к практическим работам по дисциплине «Системный анализ, управление и обработка информации»

Омск 2021

Составитель Л.А. Денисова, д. т. н., профессор кафедры «Автоматизированные системы обработки информации и управления»

Приводятся теоретические сведения об алгоритмах оптимизации на графах (задачи о минимальном остовном дереве и кратчайшем пути) и методах их решения в среде инженерных и научных расчетов MATLAB, расширение Graph Theory Toolbox. Для рассмотренных задач приведены примеры программирования, задания для самостоятельной работы и контрольные вопросы.

Для аспирантов направления подготовки 09.06.01 – «Информатика и вычислительная техника», направленности 05.13.01 – «Системный анализ, управление и обработка информации»

# ПРАКТИЧЕСКАЯ РАБОТА №1

# ПОСТРОЕНИЕ ОРГРАФА С ПОМОЩЬЮ ПАКЕТА MATLAB

## 1. Основные определения теории графов

Дадим некоторые определения теории графов с точки зрения теории множеств. Это наиболее общий подход к изучению графов и их свойств [1–3].

**Определение 1**. *Граф G* – это совокупность двух множеств *V и E:*

$$G = (V, E), \tag{1}$$

где $V$ – основное множество, $E$ – множество двухэлементных подмножеств множества $V$.

**Определение 2.** Элементы основного множества $V$ называются *вершинами.*

Вершины графа обозначаются $v_1, v_2 \ldots v_n$ (от англ. vertex – вершина).

**Определение 3.** Количество вершин графа $G$ (мощность множества $V$) называется *размером* графа.

Размер графа обозначается буквой $n$: $|V| = n$. Здесь функция $|...|$ – количество элементов множества.

**Определение 4.** Элементы множества $E$ называются *ребрами.*

Ребра обозначают буквами $e_1, e_2, ..., e_m$ (от англ. *edge* – ребро).

**Определение 5**. Количество ребер графа $G$ (мощность множества $E$) называется *мощностью* графа.

В графе по определению 1 не может быть кратных ребер и петель, так как по основным аксиомам теории множеств одинаковые элементы множества $E$ считаются одним элементом и в каждом ребре должны быть две различные вершины.

Обобщением графа (1) являются *мультиграф* и *псевдограф*.

В мультиграфе разрешены кратные ребра, то есть может быть несколько ребер, соединяющих одну и ту же пару вершин. Но каждое ребро должно соединять две разные вершины – петель нет.

В псевдографе допускаются и кратные ребра, и петли, то есть ребро может соединять вершину саму с собой.

**Определение 6**. Ребра называются *инцидентными* друг другу, если они имеют хотя бы одну общую вершину. Ребро называется *инцидентным* вершине, если вершина является концом ребра.

**Определение 7**. Если каждой вершине (или ребру) поставлено в соответствие действительное число $b_k$ (или $c_k$), то граф называется *графом со взвешенными вершинами* (или *ребрами*), а числа $b_k$ (или $c_k$) – *весами вершин* (или *ребер*).

**Определение 8**. Граф размера $n$ называется *полным* (*кликой*), если в нем каждая вершина соединена ребром с любой другой. Клика обозначается $K_n$. Количество ребер в $K_n$ равно

$$m = n(n-1)/2. \tag{2}$$

**Определение 9**. Если каждому ребру графа приписывается направление (указывается, какая вершина является первой, а какая – второй), то граф называется *ориентированным графом*, или *орграфом*.

Ребра ориентированного графа называются *дугами*, или *стрелками*.

## 2. Как задать граф для решения задач в MATLAB

Система MATLAB, как язык программирования высокого уровня для научных и технических вычислений, имеет ряд пакетов расширения, реализующих различные возможности компьютерной математики [4,5].

Для решения задач теории графов используется инструментарий GrTheory Toolbox, разработанный С.П. Иглиным [3] и доступный для свободного копирования по адресу [6]. Для рисования графов применяется функция P1otGraph.

Для задания исходных данных для графов необходимо определиться с входными и выходными параметрами. Во-первых, нужно задать координаты вершин – это 2-мерный массив размером $n \times 2$, который обозначается идентификатором *V*.

Граф (орграф) может быть со взвешенными вершинами. Если задать *V* размером не $n \times 2$, а $n \times 3$, то в 3-м столбце будут задаваться веса вершин. Веса задают, чтобы их изобразить при рисовании. Поэтому, если не задать веса, то в каждом кружке, изображающем вершину, напишется ее номер, а если задать – то вес.

Далее нужно задать структуру графа (орграфа) – список его ребер (дуг) в виде массива размером $m \times 2$. Он обозначается идентификатором *E*. Если ребра (дуги) взвешены, нужно также задать и их веса. Это удобно сделать в 3-м столбце массива *E*. Как и для вершин, если задать массив *E* размером $m \times 2$, то возле каждого ребра (дуги) будет писаться его номер, а если $m \times 3$, то вес. Если в графе вообще нет ребер (одни вершины), то можно или вообще не задавать второй аргумент, или задать его в виде пустого массива.

И, наконец, нужно указать, что необходимо: граф или орграф. Здесь достаточно одного символа (идентификатор p). Если это символ 'o', то рисуется орграф, во всех других случаях (т. е. по умолчанию) рисуется граф. Простые ребра рисуются прямыми, кратные – полуэллипсами.

В качестве выходного параметра (идентификатор h) MATLAB возвращает дескриптор созданной фигуры с нарисованным на ней графом.

## 3. Описание процедуры PlotGraph

Процедура P1otGraph предназначена для рисования графов (в том числе мультиграфов и псевдографов) и орграфов.

Процедура P1otGraph имеет следующий порядок вызова, входные и выходные аргументы:

```
Function h=P1otGraph(V,E,p)
% Функция h=P1otGraph(V,E,p) рисует граф (орграф).
```

```
% Входные параметры:
% V(n,2) или (n,3) – координаты вершин
% (1-й столбец – x,  2-й – y)  и, может быть,  3-й – веса;
% n – количество вершин;
% если V(n,2), рисуются номера вершин,
% если V(n,3) , рисуются веса вершин.
% E(m,2)  или  (m,3)  – ребра графа  (дуги орграфа)  и их веса;
% 1-й и 2-й элементы каждой строки – это номера вершин;
% 3-й элемент каждой строки – это вес дуги;
% m – количество дуг.
% если E(m,2) , рисуются номера ребер  (дуг);
% если E (т,3) , рисуются веса ребер (дуг);
%   для графа без ребер задаем E=[] ;
% p = 'g'  (рисовать граф)  или  'o'   (рисовать орграф);
%(необязательный параметр,  по умолчанию 'g') .
% Выходной параметр:
% h – дескриптор фигуры.
```

Номера всех вершин в массиве *E* должны быть целыми положительными числами. Размер массива *V* должен быть совместим с номерами вершин в массиве *E*.

## 4. Выполнение заданий

**Задание 1.** Рассмотрите пример обращения к процедуре PlotGraph.

Работа этой функции проиллюстрирована на примере рисования орграфа с 25 вершинами и 46 дугами. Рисуется орграф с невзвешенными вершинами и невзвешенными дугами. Установлен шрифт и подписан заголовок.

```
clear all
V=[[0 4];[1 4];[2 4];[3 4];[4 4];[0 3];[1 3];[2 3];[3 3];[4 3];...
   [0 2];[1 2];[2 2];[3 2];[4 2];[0 1];[1 1];[2 1];[3 1];[4 1];...
   [0 0];[1 0];[2 0];[3 0];[4 0]];
E=[[ 1  2];[ 3  2];[ 4  3];[ 5  4];[ 6  1];...
   [ 2  7];[ 8  2];[ 3  8];[ 9  4];[ 9  5];...
   [10  5];[ 7  6];[ 8  7];[ 8  9];[10  9];...
   [11  6];[ 7 12];[13  8];[14  9];[15 10];...
   [12 11];[13 12];[13 14];[14 13];[15 14];...
   [16 11];[12 17];[13 18];[20 15];[17 16];...
   [17 18];[18 17];[19 18];[19 20];[21 16];...
   [17 22];[18 22];[22 18];[18 23];[19 24];...
   [20 25];[21 22];[22 21];[23 24];[24 23];[24 25]];
PlotGraph(V,E,'o'); % рисуем орграф
set(get(gcf,'CurrentAxes'),...
  'FontName','Times New Roman Cyr','FontSize',14)
title('\bfПример рисования орграфа ')
```

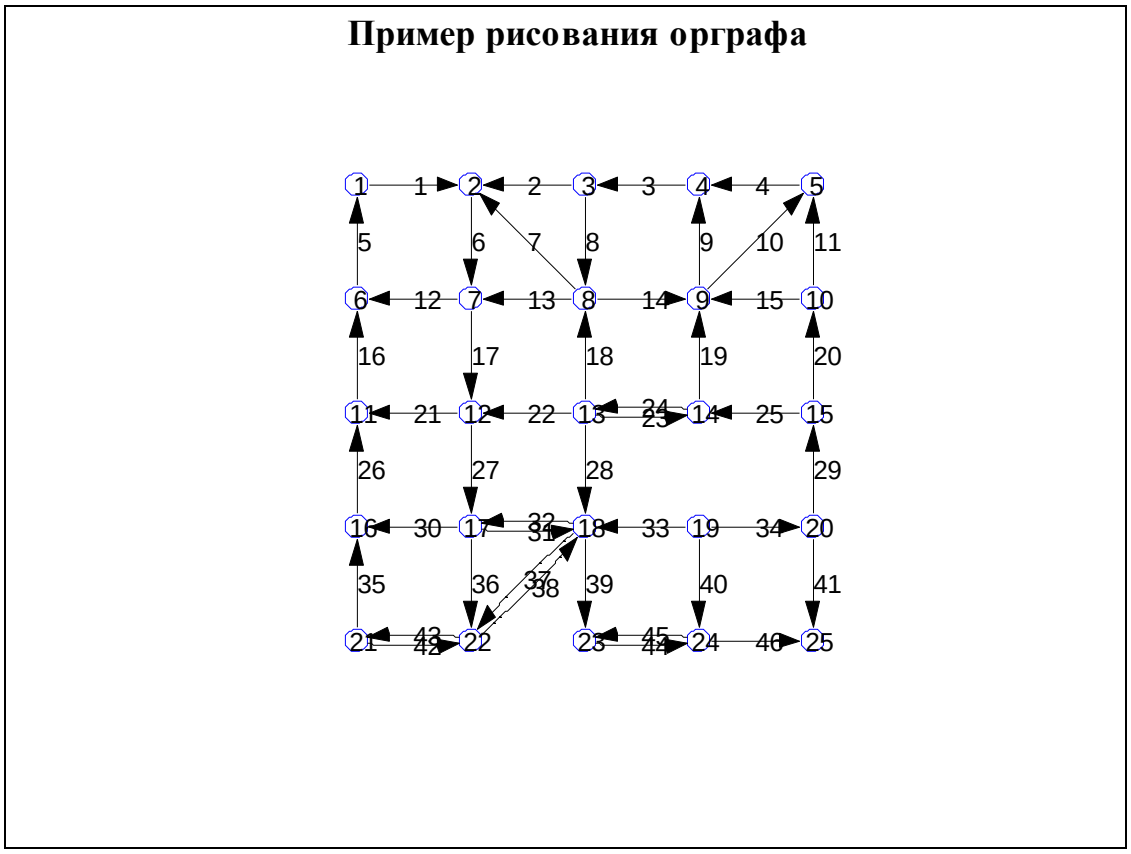


Рис. 1. Орграф, нарисованный с помощью MATLAB

Нарисуйте приведенный орграф. Откорректируйте пример. Нарисуйте его в виде неориентированного графа с невзвешенными вершинами и взвешенными ребрами. Веса ребер задайте самостоятельно. Измените формат надписи рисунка.

**Задание 2.** Нарисуйте ориентированный граф с 11 вершинами и 22 ребрами. Введите веса дуг.

```
clear all
V=[[0 0];[1 1];[1 0];[1 -1];...
   [2 1];[2 0];[2 -1];[3 1];...
   [3 0];[3 -1];[4 0]]; % координаты вершин
E=[[1 2 5];[1 3 5];[1 4 5];[2 3 2];[3 4 2];[2 5 3];...
   [2 6 2];[3 6 5];[3 7 2];[4 7 3];[6 5 1];[6 7 1];...
   [5 8 5];[6 8 2];[6 9 3];[7 9 2];[7 10 3];[8 9 2];...
   [9 10 2];[8 11 5];[9 11 4];[10 11 4]]; % ребра и их веса
PlotGraph(V,E,'o'); % рисуем орграф
set(get(gcf,'CurrentAxes'),...
  'FontName','Times New Roman Cyr','FontSize',14)
title('\bfОрграф со взвешенными дугами')
```

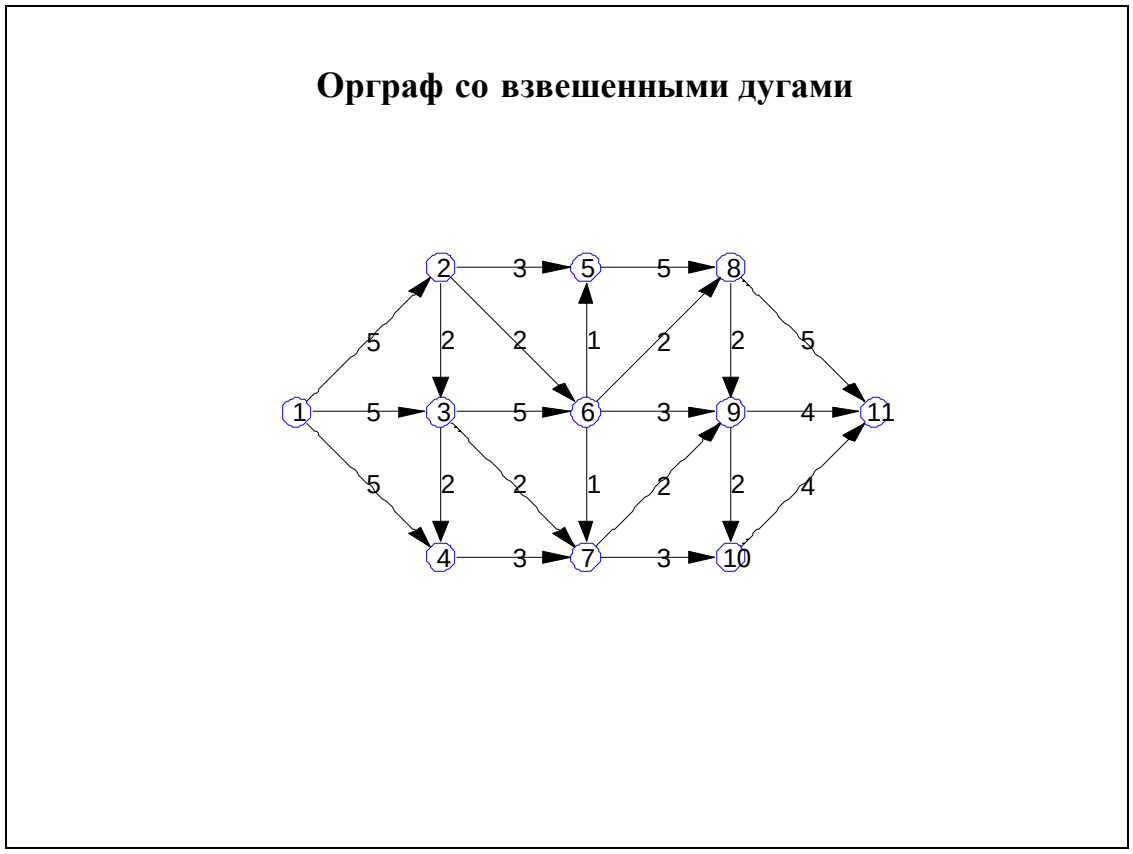


Рис. 2. Орграф со взвешенными дугами, нарисованный с помощью MATLAB

Откорректируйте пример. Нарисуйте его в виде неориентированного графа со взвешенными вершинами и невзвешенными ребрами. Веса вершин задайте самостоятельно.

**Задание 3.** Нарисуйте полный граф (клику) $K_n$ с количеством вершин $n$=4 и $n$=5. Введите веса дуг.

**Контрольные вопросы**

1. Сформулируйте определение графа и его обобщений.
2. Объясните понятие инцидентности.
3. Какое количество дуг будет в ориентированной клике без петель и кратных дуг? Приведите примеры.
4. Какой порядок вызова, входные и выходные аргументы имеет функция P1otGraph?

## ПРАКТИЧЕСКАЯ РАБОТА №2

## ПОСТРОЕНИЕ ОСТОВНОГО ДЕРЕВА МИНИМАЛЬНОГО ВЕСА

Обычно в приложениях ставится задача нахождения остовного дерева минимального веса (в дальнейшем - МОД, минимальное остовное дерево). Такая задача возникает, например, при проектировании линий электропередач: для прокладки линий выбирают участки с минимальной стоимостью работ, обеспечивающие связность сети. В работе проведем построение минимального остовного дерева связного графа с помощью жадного алгоритма.

### 1. Понятие жадного алгоритма.

Рассмотрите пример. Требуется найти максимум линейной функции $z = (c, x) \to \max$ на перестановке $P_n$ значений ее аргументов *x*. Для численного примера

$$z = 4x_1 - 3x_2 - 6x_3 + 2x_4 \to \max, \qquad (1)$$

где вектор аргументов $(x_1, x_2, x_3, x_4)$ – одна из перестановок чисел (2, 3, 4, 5).

На какие места в векторе *x* нужно поставить числа 2, 3, 4 и 5, чтобы максимизировать функцию (1)?

Для решения задачи применяется такой алгоритм. Коэффициент при $x_1$ максимальный: он равен 4. Поэтому переменной $x_1$ присваивается максимальное из имеющихся значений: 5. Следующий по величине коэффициент – это 2 при $x_4$, поэтому переменной $x_4$ присваивается максимальное из оставшихся значений 4. Далее переменной $x_2$ присваивается значение 3, а $x_3$ – 2.

Этот процесс показан на рис. 3. Получаем *z* = 7. Можно перебрать все 24 перестановки и убедиться, что действительно найдено максимальное значение. Алгоритм, который применен для решения этого примера, называется *жадным*. Его общий принцип так же прост, как человеческая жадность: бери как можно больше из того, что осталось.

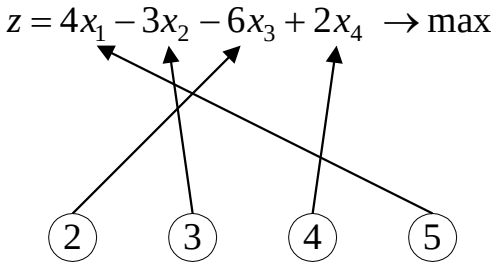


Рис. 3. Максимизация линейной функции
на множестве перестановок значений ее аргументов

## 2. Остовное дерево минимального веса

**Определение 1**. *Путь (маршрут)* в графе $G = (V,E)$ – это последовательность вершин и ребер вида $v_1e_1v_2e_2...v_k$, в которой соседние элементы инцидентны.

**Определение 2.** Маршрут называется *простым,* если каждая вершина встречается в нем только один раз.

В простом маршруте нет пересечений (повторяющихся вершин) и, следовательно, нет повторяющихся ребер.

**Определение 3.** Граф $G = (V,E)$ называется *связным,* если существует путь из любой его вершины в любую другую.

В несвязных графах можно выделить отдельные связные компоненты. В крайнем случае, когда у графа вообще нет ребер: $E=\varnothing$, у него $n$ компонент – по одной вершине в каждой. Если из связного графа (или мультиграфа) $G$ удалить некоторые ребра, он останется связным или распадется на отдельные компоненты.

Если задан граф со взвешенными ребрами, то ставится задача: как по максимуму удалить ребра максимального веса, чтобы остался связный граф с общим минимальным весом ребер?

**Теорема 1.** В связном графе $G = (V,E)$ выполняется: $m \geq n-1$, т. е. минимально возможное число ребер связного графа равно $n$-1.

*Доказательство.* При $n=1$ говорить о связности вообще нет смысла: одну вершину не с чем связывать (одна вершина связана с собой нулевым количеством ребер). Две вершины можно соединить одним ребром – граф станет связным. Третью вершину можно подсоединить с помощью второго ребра. По индукции: пусть теорема верна для $n = k$, то есть граф с $k$ вершинами и $k$-1 ребрами связный. Очередную, ($k$+1)-ю вершину можно подсоединить к нему с помощью $k$-го ребра. Полученный граф с ($k$+1) вершинами и $k$ ребрами тоже будет связным. По индукции теорема доказана.

**Определение 4.** Связный граф $G = (V,E)$ с $n$ вершинами и $n$-1 ребрами называется *остовным деревом*.

На рис. 4 показан граф *(а)* и некоторые из его остовных деревьев *(б, в)*.

Если связный граф $G = (V,E)$ не является остовным деревом, то из него можно удалить некоторые ребра так, что оставшиеся ребра $E_1 \subseteq E$ вместе с множеством вершин $V$ образуют остовное дерево $G_1 = (V,E_1)$.

**Определение 5.** Путь $v_1e_1v_2e_2..v_k$ называется *циклом,* если в нем начальная и конечная вершины совпадают.

**Определение 6.** Цикл называется *простым,* если в определяющем его пути каждая промежуточная вершина встречается только один раз.

Простой цикл – это цикл без самопересечений. Если первую и последнюю вершины считать за одну, то в нем нет повторяющихся вершин и нет повторяющихся ребер. Из любого непростого цикла можно выделить, по крайней мере, два разных простых цикла, а из непростого маршрута – по крайней мере один простой цикл.

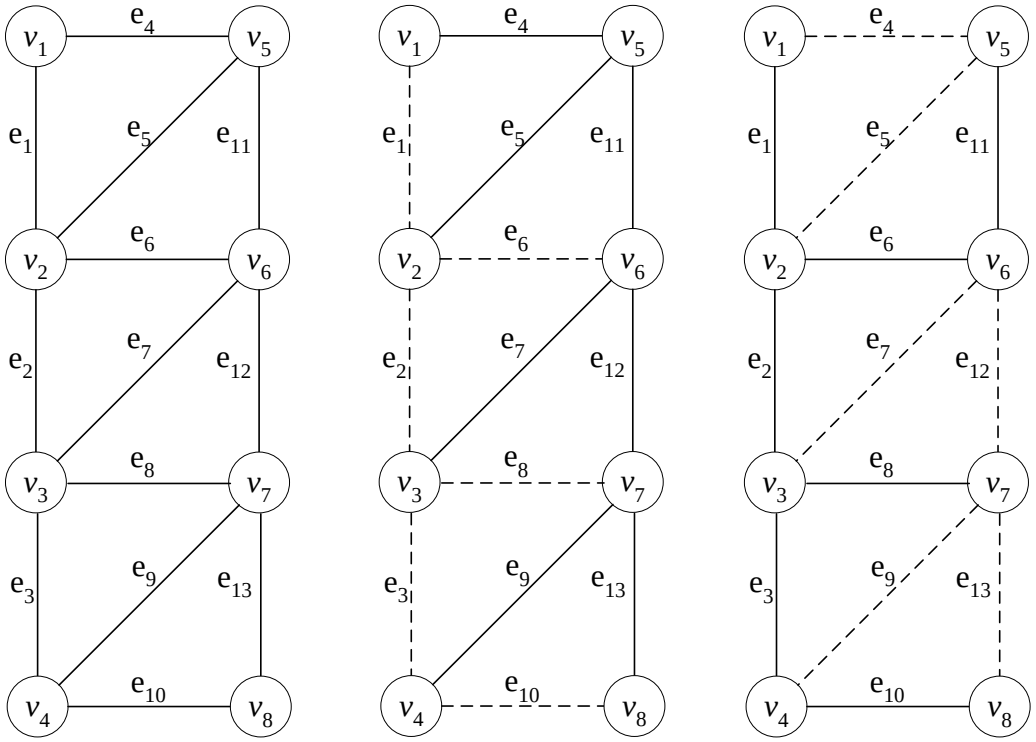


Рис. 4. *а)* – граф*; б), в)* – его остовные деревья.

**ТЕОРЕМА 2.** В остовном дереве $G_1 = (V,E_1)$ связного графа $G = (V,E)$ нет циклов.

*Доказательство.* От противного: если в $G_1$ есть хотя бы один цикл, то можно без ущерба для связности удалить одно из ребер этого цикла: путь из любой вершины в любую другую можно организовать по оставшейся части цикла. Значит, в $G_1$ больше, чем $n$-1 ребер: удаление из него одного ребра не нарушает связности. Следовательно, $G_1$ не является остовным деревом.

**ТЕОРЕМА 3.** В остовном дереве $G_1$ связного графа существует единственный путь из каждой вершины в каждую другую.

*Доказательство.* От противного: если бы из некоторой вершины $v_i$ существовало два различных пути в $v_j$, то существовал бы цикл $v_i... v_j,.. v_i$, что противоречит теореме 2.

Рассмотрите возможности применения жадного алгоритма для нахождения МОД. Вначале рассмотрите две схемы алгоритма, а затем доказательство, что обе они приводят к одному и тому же правильному результату. Основной принцип построения МОД: надо не *удалять* ребра из исходного графа, а *добавлять* их в строящееся МОД.

**Первая схема** показана на рис. 5. Здесь в вершинах проставлены их номера, а возле ребер – веса. Будем обозначать каждое ребро буквой *e* с двумя индексами – номерами соединяемых вершин. В рассматриваемом графе *n=9, m=16,* поэтому в построенном МОД должно остаться только 8 ребер из 16. Выберем первое ребро минимального веса. Минимальный вес ребер 1, и первым в списке ока-

залось $e_{24}$, поэтому начинаем построение МОД с него *(а)*. Далее просматриваем все ребра, инцидентные уже построенному фрагменту МОД, то есть к $e_{24}$. Всего таких ребер шесть (по три в вершинах $v_2$ и $v_4$). Выбираем из них ребро минимального веса. Минимальный вес среди этих шести ребер – 2, он есть у $e_{12}$ и $e_{25}$. Первым в списке идет $e_{12}$. Его и подсоединяем к строящемуся МОД (б).

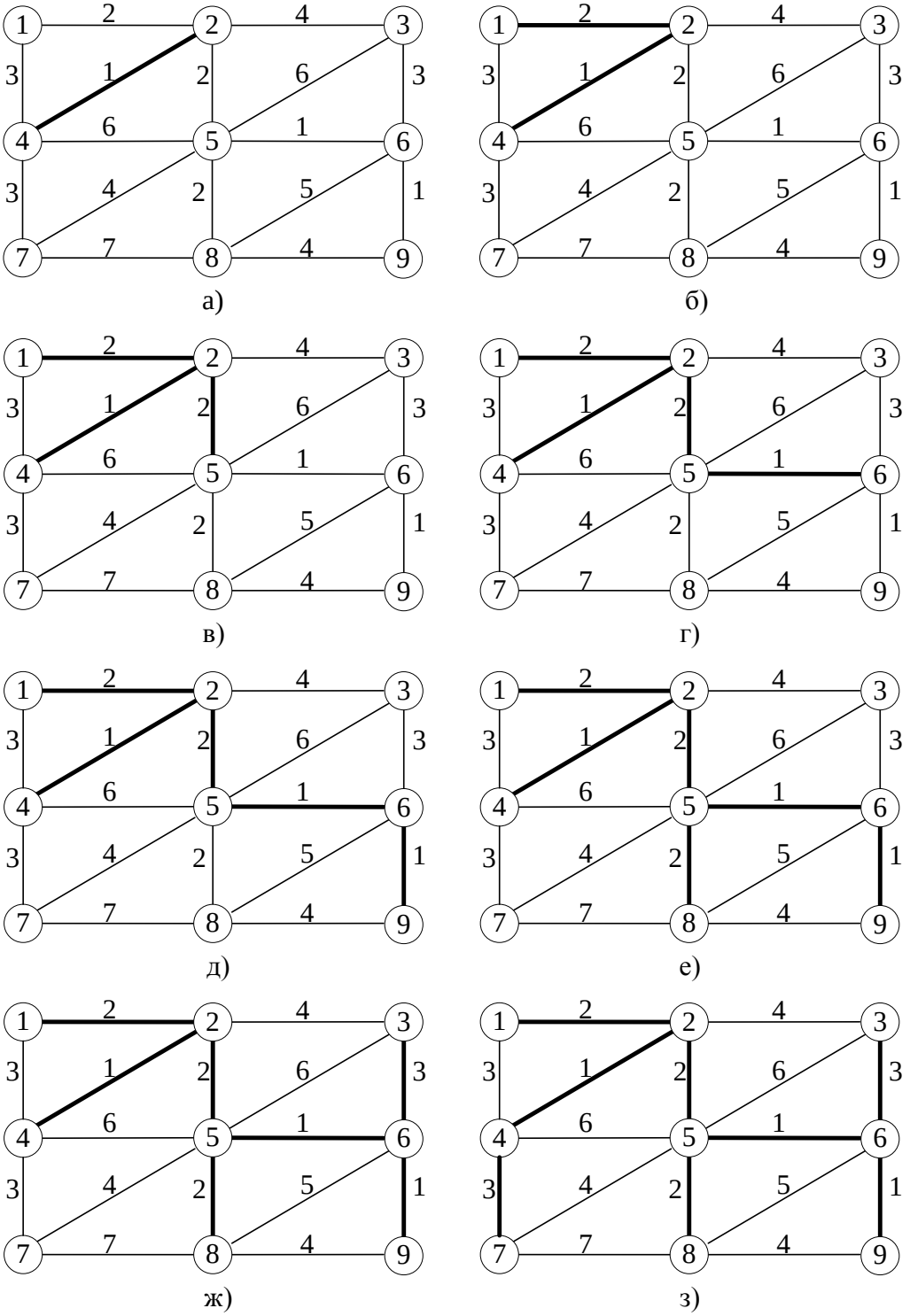


Рис. 5. Первая схема построения МОД при помощи жадного алгоритма

Продолжаем просматривать ребра, инцидентные к уже построенному фрагменту МОД и не образующие в нем циклов. Берем все ребра, инцидентные к $v_1$, $v_2$, и $v_4$ (кроме $v_{14}$ – его нельзя брать, так как образуется цикл) и выбираем из них

ребро минимального веса. Минимальный вес 2 у ребра $e_{25}$, его и присоединяем *(в)*. Теперь у нас появилась возможность присоединить к МОД ребро $e_{56}$ веса 1 (*г*), а затем - $e_{69}$ *(д)*. Среди оставшихся ребер минимальный вес 2 – у ребра $e_{58}$, его присоединяем к МОД *(е)*. И, наконец, присоединяем ребра $e_{36}$ и $e_{47}$ веса 3 *(ж, з)*. Получили 8 ребер - МОД построено. Его вес 15.

**Вторая схема** показана на рис. 6. Основное ее отличие от первой схемы в том, что добавляются в строящееся МОД не обязательно инцидентные ребра. Главное, чтобы не образовывалось циклов.

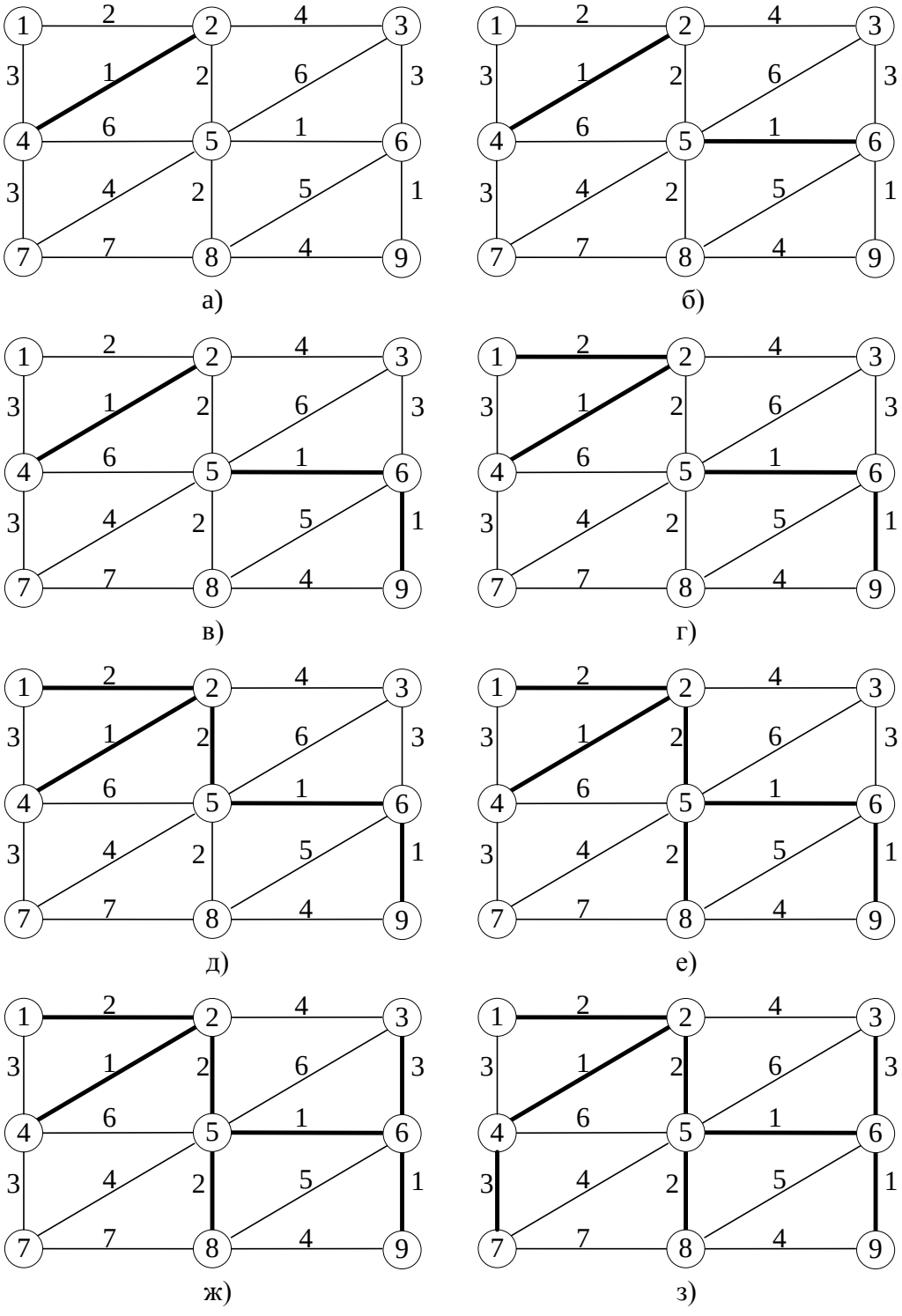


Рис. 6. Вторая схема построения МОД при помощи жадного алгоритма

Просматриваем все ребра минимального веса 1. Начинаем с $e_{24}$ – оно встретилось первым. Следующее ребро веса 1 – это $e_{56}$, и оно не образует циклов – добавляем его (б). Затем проверяем – $e_{69}$*:* циклов нет, присоединяем (*в)*. Все ребра веса 1 исчерпаны. Переходим к оставшимся ребрам минимального веса 2. Ребро $e_{12}$ можно присоединить (*г*), $e_{25}$ – тоже *(д),* $e_{58}$ подходит (*е*). Далее – ребра веса 3. Ребро $e_{14}$ не годится (образуется цикл), $e_{36}$ подходит (*ж*), и $e_{47}$ тоже *(з).* В результате получили то же самое МОД веса 15, что и по схеме 1.

Схемы 1 и 2 отличаются порядком присоединения ребер. В 1-й схеме все время присоединяем инцидентные ребра, т.е. наращиваем МОД, оставляя его связным. Во 2-й схеме строятся отдельные куски МОД, которые затем соединяются. Каждый из этих кусков называется *деревом,* а их совокупность – *лесом.*

**Определение 7.** *Деревом* называется остовное дерево, построенное на некотором подмножестве вершин $V_1 \subseteq V$ и инцидентным ко всем им ребрам.

**Определение 8.** *Лесом* называется множество непересекающихся деревьев. Например, лес на рисунке 6, *г* состоит из деревьев, построенных на подмножествах вершин $\{v_1,v_2,v_4\}$, $\{v_3\}$, $\{v_5,v_6,v_9\}$, $\{v_7\}$, $\{v_8\}$. В подмножествах из одной вершины никаких ребер нет, и соответствующее дерево состоит только из вершины.

Ниже сформулирована и доказана теорема, которая устанавливает применимость жадного алгоритма к построению МОД. Обратите внимание, что МОД может быть не единственным. Если в графе много ребер одинакового малого веса, то, возможно, существует несколько МОД. В частности, если все ребра имеют одинаковый вес, то любое остовное дерево будет минимальным. Поэтому надо говорить не о МОД, а об одном из МОД.

**Теорема 4.** Пусть на непересекающихся подмножествах вершин $V_1$, $V_2$, ..., $V_k$ построены деревья минимального веса $G_1= (V_1,E_1)$, $G_2= (V_2,E_2)$, ..., $G_k= (V_k,E_k)$. Пусть также $e_j$ - ребро минимального веса, один конец которого входит в $G_1$ (т. е. инцидентен какой-либо вершине из $V_1$), а другой – в какое-либо другое дерево: $G_2$, $G_3$, ..., $G_k$. Тогда среди всех деревьев минимального веса, построенных на объединении ребер $\cup E_i$, существует дерево минимального веса, содержащее ребро $e_j$.

Поясним формулировку теоремы на примере. Возьмем один из этапов построения МОД, показанный на рис. 6, *в*. Изобразим его отдельно (рис. 7, *а).* Здесь построено несколько минимальных деревьев.

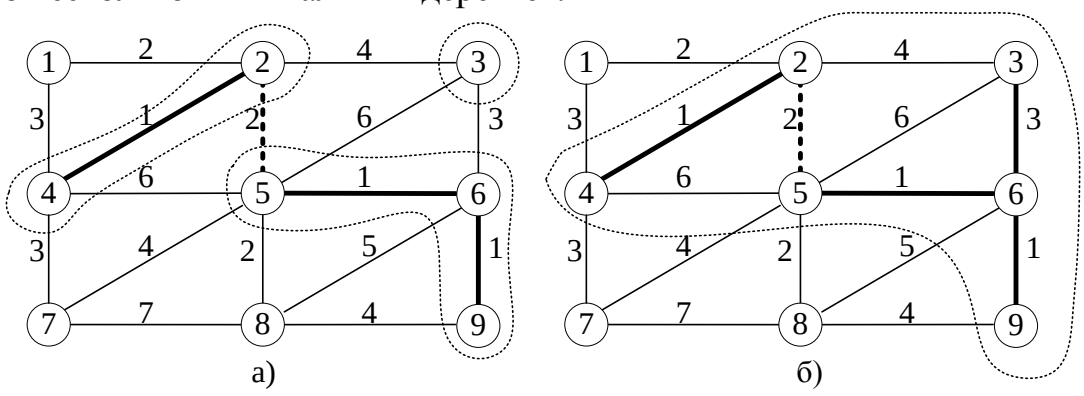


Рис. 7. Пояснение к теореме 4

Рассмотрим случай $k=3$. Пусть, например, $V_1=\{v_2,v_4\}$, $V_2=\{v_5,v_6,v_9\}$, $V_3=\{v_3\}$, и на каждом подмножестве вершин построено минимальное дерево. Они обведены штриховыми контурами. Имеем $E_1=\{e_{24}\}$, $E_2=\{e_{56}, e_{69}\}$, $E_3=\varnothing$. Найдем ребро минимального веса, выходящее из $G_1$ в сторону $G_2$ или $G_3$. В сторону $G_2$ из $G_1$ выходят ребра $e_{25}$ и $e_{45}$, а в сторону $G_3$ – $e_{23}$. Из этих трех ребер минимальный вес 2 – у $e_{25}$. В формулировке теоремы это $e_i$. Оно обозначено на рисунке штриховой линией.

Теорема утверждает следующее. Если построить минимальное дерево на множестве вершин $\{v_2,v_4,v_5,v_6,v_9,v_3\}$, то среди всех минимальных деревьев, в которых есть ребра $e_{24}$, $e_{56}$, $e_{69}$ наверняка найдется хотя бы одно минимальное дерево, содержащее $e_{25}$ (рис. 7, *б)*.

*Доказательство.* От противного: предположим, что существует дерево $G_0=(V_0,E_0)$, построенное на объединении вершин $V_0=\cup V_i$, включающее в себя объединение ребер $E_0 \supseteq \cup E_i$, не включающее в себя ребро $e_j$ и имеющее вес меньше, чем вес любого минимального дерева, включающего $e_j$. В иллюстрации на рис. 7 такое случилось бы, если бы вместо $e_{25}$ мы попытались бы соединить $G_1$ с $G_2$ ребром $e_{45}$.

Докажем, что такое невозможно. Для этого добавим в $G_0$ ребро $e_j$. После этого $G_0$ перестанет быть деревом: в нем образуется один цикл. Этот цикл будет заходить в $G_1$, но не будет лежать в нем полностью: он будет заходить и в какое-либо другое множество вершин из $V_2$, $V_3$, ..., $V_k$. Действительно, один из концов $e_j$ инцидентен вершине из $G_1$, а другой – какой-либо другой вершине из $G_2$, $G_3$, ..., $G_k$. Так, в графе на рис. 7 образовался бы цикл $v_2e_{25}v_5e_{45}v_4e_{24}v_2$. Этот цикл заходит в $G_1$, но захватывает еще и $G_2$.

Удалим из этого цикла какое-либо другое ребро, одним концом входящее в $G_1$, а другим – в другое $G_i$. Такое ребро наверняка существует, иначе не было бы цикла, охватывающего $G_1$ и какое-либо другое $G_i$. И вес удаляемого ребра не меньше (а, может быть, и больше), чем присоединенного $e_j$. В примере с графом на рис. 7 мы удаляем $e_{45}$. Связность всех вершин в $V_0$ при этом сохранится: мы можем перейти от $G_1$ к другому $G_i$, по $e_j$ вместо удаленного ребра. Значит, после такой замены ребер у нас останется дерево, и вес его – не меньше, чем у исходного дерева. А это противоречит предположению о том, что $G_0$ имеет вес меньше, чем у дерева с $e_j$.

На основании этой теоремы можно утверждать, что обе рассмотренные выше схемы дают МОД (хотя, может быть, и разные). В первой схеме вначале каждое $G_i$ состоит из одной вершины без ребер. Вначале выбираем ребро минимального веса. Оно соединяет два $G_i$, одно из которых можно обозначить $G_1$. Значит, оно войдет в одно из МОД. Далее обозначаем через $G_1$ минимальное дерево, образованное двумя вершинами и соединяющим их ребром, и находим ребро минимального веса; инцидентное $G_1$ одним своим концом. В теореме 4 это $e_j$. Согласно этой теореме можно присоединить его к строящемуся дереву, при этом не теряется возможность построить одно из МОД. Вторая схема приведена как иллюстрация к доказательству теоремы.

## 3. Описание процедуры MinSpanTree

Для построения остовных деревьев в MATLAB применяется процедура MinSpanTree. При реализации на MATLAB используется схема 1, так как она программируется проще, чем схема 2: для одного строящегося дерева (не для нескольких) осуществляется проверка ребер на инцидентность и на отсутствие циклов.

Входным параметром в процедуре построения МОД является список ребер размером $m \times 2$ или $m \times 3$, который обозначен идентификатором E. Если задано два столбца, то решается невзвешенная задача, а если три – то взвешенная. Выходным параметром является список номеров ребер, входящих в МОД.

Заголовок функции MinSpanTree и справочная информация выглядят так:

```
   function nMST=MinSpanTree(E)
% Функция nMST=MinSpanTree(E) решает задачу о минимальном
% остовном дереве для связного графа.
% Входной параметр
% E(m,2) или (m,3) – ребра графа и их веса;
% 1-й и 2-й элементы каждой строки – это номера вершин;
% 3-й элемент каждой строки – это вес ребра;
% m – количество ребер.
% Если задан массив E(m,2) , то все веса равны 1.
% Выходной параметр:
%  nMST – список номеров ребер, включенных в минимальное
% (взвешенное) остовное дерево в порядке включения.
%  Используется жадный алгоритм.
```

При проверке исходных данных проверяется наличие данных, размерность и количество столбцов входного массива, положительность и целочисленность первых двух столбцов.

## 4. Выполнение заданий

**Задание 1.** Рассмотрите пример обращения к процедуре MinSpanTree.

Работа функции MinSpanTree проиллюстрирована на графе. Вводятся исходные данные: координаты вершин, список ребер и их веса.

Нарисуйте исходный граф со взвешенными ребрами

```
clear all
V=[[0 4];[1 4];[2 4];[3 4];[4 4];...
   [0 3];[1 3];[2 3];[3 3];[4 3];...
   [0 2];[1 2];[2 2];[3 2];[4 2];...
   [0 1];[1 1];[2 1];[3 1];[4 1];...
   [0 0];[1 0];[2 0];[3 0];[4 0]]; % координаты вершин
E=[[ 1  2 12];[ 3  2 17];[ 4  3 10];[ 5  4  8];[ 6  1  4];...
  [ 2  7 11];[ 8  2 10];[ 3  8  9];[ 9  4  4];[ 9  5  2];...
  [10  5 14];[ 7  6 16];[ 8  7 15];[ 8  9 13];[10  9 11];...
```

```
   [11  6 10];[ 7 12 11];[13  8 13];[14  9  1];[15 10  2];...
   [12 11  3];[13 12  3];[13 14  4];[14 13 12];[15 14  5];...
   [16 11  6];[12 17  7];[13 18  8];[20 15 10];[17 16  9];...
   [17 18 10];[18 17 11];[19 18 12];[19 20 13];[21 16 14];...
   [17 22 12];[18 22 19];[22 18 18];[18 23 17];[19 24 16];...
   [20 25 15];[21 22 14];[22 21 13];[23 24 12];[24 23 11];[24 25 10]];
PlotGraph(V(:,1:2),E); % рисуем граф
set(get(gcf,'CurrentAxes'),...
  'FontName','Times New Roman Cyr','FontSize',14)
title('\bfИсходный граф со взвешенными ребрами ')
```


Рис. 8. Исходный граф, нарисованный с помощью MATLAB

Для данного исходного графа решается две задачи: невзвешенная и взвешенная. В первом случае с помощью функции MinSpanTree строится первое встретившееся остовное дерево, а во втором − МОД. Печатается количество ребер и их суммарный вес в обоих вариантах. Рисуются оба решения: показываются все вершины, но только те ребра, которые входят в найденное остовное дерево.

```
nMST=MinSpanTree(E(:,1:2)); % невзвешенная задача
fprintf('Количество ребер в остовном дереве = %d\n',length(nMST));
fprintf('Общий вес = %d\n',sum(E(nMST,3)));
PlotGraph(V(:,1:2),E(nMST,:),'g');
set(get(gcf,'CurrentAxes'),...
```

```
  'FontName','Times New Roman Cyr','FontSize',14)
title('\bfОстовное дерево')
nMST=MinSpanTree(E); % взвешенная задача
fprintf('Количество ребер в МОД = %d\n',length(nMST));
fprintf('Общий вес = %d\n',sum(E(nMST,3)));
PlotGraph(V(:,1:2),E(nMST,:),'g');
set(get(gcf,'CurrentAxes'),...
  'FontName','Times New Roman Cyr','FontSize',14)
title('\bfМинимальное остовное дерево')
```

Количество ребер в остовном дереве = 24
Общий вес = 244
Количество ребер в МОД = 24
Общий вес = 182

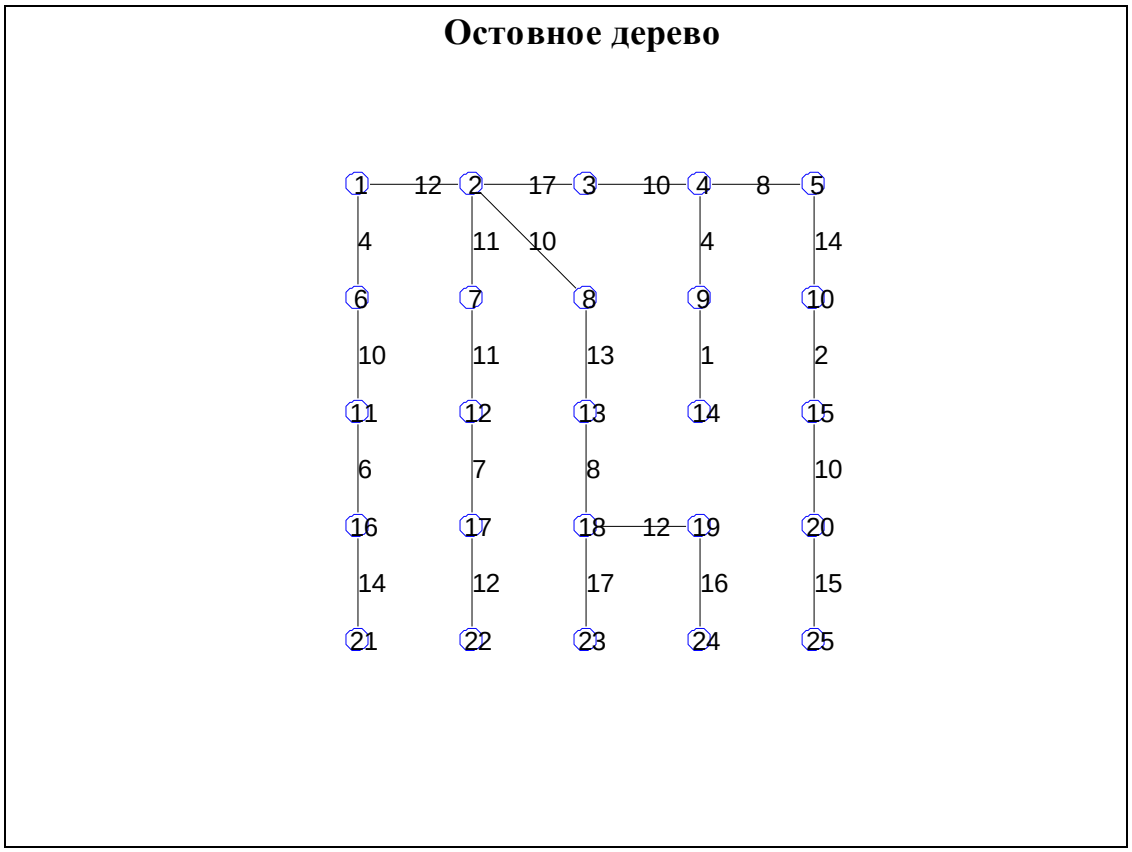


Рис. 9. Остовное дерево, нарисованное с помощью MATLAB

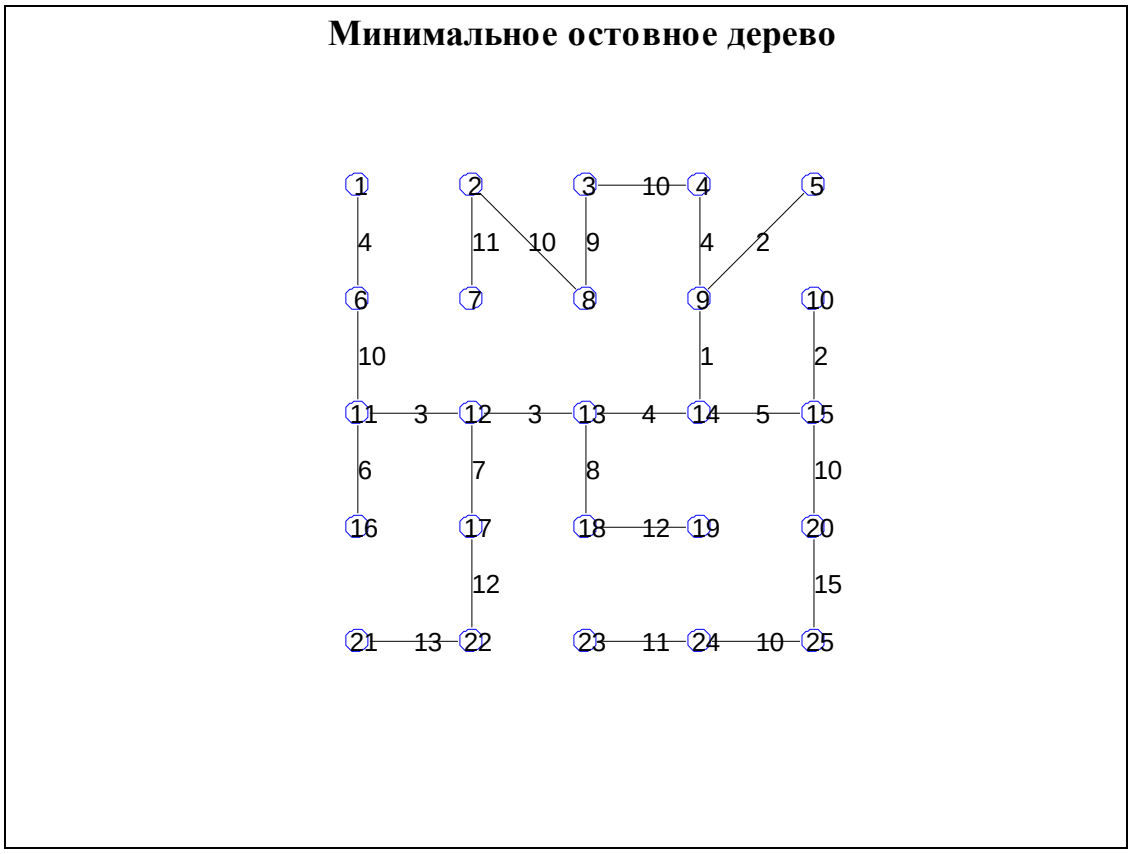


Рис. 10. МОД, нарисованное с помощью MATLAB

**Задание 2**. Нарисуйте неориентированный граф, приведенный на рис. 11. Введите исходные данные: координаты вершин, список ребер и их веса.

Нарисуйте исходный граф со взвешенными ребрами. Решите невзвешенную и взвешенную задачи. С помощью функции MinSpanTree постройте остовное дерево и МОД. Напечатайте количество ребер и их суммарный вес в обоих вариантах. Нарисуйте оба решения.

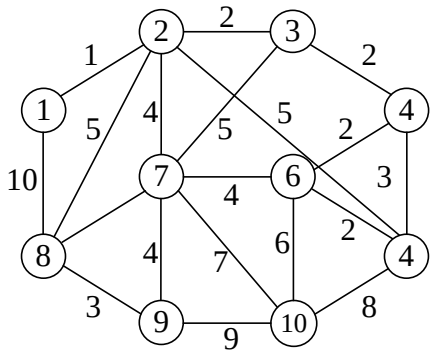


Рис. 11. Граф к заданию 2

**Контрольные вопросы.**

1. Что называется путем в графе? Простым путем в графе?
2. Какой граф называется связным? Что называется остовным деревом?

3. Что называется циклом? Простым циклом?
4. Что называется деревом? Что называется лесом?
5. Сколько всего существует остовных деревьев у графа $G$=($V$,$E$)?
6. Приведите доказательства теорем 1 – 4.
7. Покажите эквивалентность следующих утверждений:
   - остовное дерево – это связный граф с $|E| = |V| - 1$;
   - остовное дерево – это связный граф без циклов;
   - остовное дерево – это связный граф, в котором существует единственный путь из любой вершины в любую другую.
8. Объясните работу алгоритмов построения минимального остовного дерева.

## ПРАКТИЧЕСКАЯ РАБОТА №3
## ПОСТРОЕНИЕ КРАТЧАЙШЕГО ПУТИ В ГРАФЕ

### 1. Постановка задачи о кратчайшем пути

Пусть задан орграф $G = (V,E)$. Рассмотрим различные постановки задачи о кратчайшем пути [1, 3]. Можно ли из вершины $v_i$ попасть в $v_j$, двигаясь только по стрелкам? Иными словами: существует ли путь из $v_i$ в $v_j$? Если таких путей несколько, то какой из них кратчайший (имеет минимальное количество дуг)?

Для орграфа со взвешенными дугами можно сформулировать задачу так: если существует несколько путей из $v_i$ в $v_j$, то какой из них имеет минимальный общий вес?

При определении кратчайшего пути невзвешенный орграф можно рассматривать как частный случай взвешенного, если всем дугам приписать веса, равные 1. Можно обобщить и взвешенную задачу: дополнить орграф до взвешенной клики недостающими дугами, приписав им бесконечные веса. Тогда формально можно определять кратчайший путь от любой вершины до любой другой. Если результатом будет бесконечность, то это значит, что реальный путь отсутствует.

Предполагается, что все веса дуг в задаче неотрицательные. Это существенное ограничение: если в орграфе существует цикл с общим отрицательным весом, то задача минимизации потеряет смысл: можно кружить по данному циклу, уменьшая общий вес до минус бесконечности.

Рассмотрим два наиболее часто используемых алгоритма решения задачи о кратчайшем пути. В одном из них, алгоритме Дейкстры (E.W. Dijkstra), находятся кратчайшие пути из заданной вершины во все остальные. Другой алгоритм – Флойда-Уоршелла, (R.W.Floyd, S.Warshall), решает более общую задачу: строит матрицу кратчайших путей из любой вершины в любую другую. Этот алгоритм реализован на MATLAB.

## 2. Алгоритм Дейкстры

Задана конкретная вершина $v_s$, и находятся кратчайшие пути из нее во все остальные вершины. Алгоритм основан на постепенном разрастании путей от $v_s$ до всех других вершин по дугам минимального веса.

Рассмотрим шаги этого алгоритма. Входным параметром является матрица весов дуг между любой парой вершин. Если какая-либо дуга отсутствует, полагаем вес бесконечным. На выходе получаем вектор *M* длины *n,* в координатах которого – кратчайшие пути из заданной вершины $v_s$ до всех остальных.

**Шаг 1**. Вначале *s*-й элемент вектора кратчайших путей M полагаем равным 0, а остальные элементы – это веса дуг из $v_s$ во все вершины. Тем самым мы определяем минимальную длину пути за один переход по дуге. Если для какой-то вершины $v_s$ соответствующий элемент вектора *M* равен бесконечности, то это значит, что мы не можем перейти из $v_s$ в $v_i$ за один переход по дуге, так как соответствующей дуги нет. Обозначим через *W* подмножество вершин, в которое включим пока только одну исходную вершину $v_s$ На следующих шагах мы будем пополнять *W* другими вершинами и искать кратчайший путь из $v_s$ в другие вершины за два перехода по дугам, затем за три и т. д.

**Шаг 2**. Из всех вершин, не входящих в подмножество *W*, выберем вершину с кратчайшим путем (элементом вектора *M*). Если таких вершин несколько, берем первую попавшуюся. Полученную вершину (обозначим ее $v_i$) добавляем в подмножество *W* и *i*-й элемент вектора *M* оставляем без изменения. Для остальных вершин, не входящих в *W,* пересчитываем длину кратчайшего пути (координату вектора *M*). Делаем это так: для каждой *j*-й вершины находим сумму 1-го элемента вектора *M* и веса дуги $e_{ij}$. Если эта сумма меньше *j*-го элемента вектора *M*, ставим эту сумму на его место.

**Шаг 3**. Если в *W* осталось включить только последнюю вершину, то выходим, а если больше одной – идем на шаг 2.

На рис. 12, *а* показан исходный орграф, возле дуг обозначены их веса. Требуется найти кратчайшие пути от $v_1$ до остальных вершин. Проводим 1-й шаг. Расстояния от $v_1$ до остальных вершин заполняем в соответствии с весами дуг. Если дуги нет, пишем бесконечность. В подмножество *W* заносим одну вершину $v_1$. Этот этап показан на рис. 12, *б.* Здесь вместо номеров вершин написаны расстояния от $v_1$ до них на 1-м шаге, т. е. текущее состояние вектора *M*. Подмножество *W* выделено штриховой линией.

Выполняем шаг 2. На рис. 12, *б* среди вершин, не входящих в *W* минимальное значение пути, равное 1, – у $v_3$, поэтому присоединяем ее к *W* (рис. 12, *в*). Пересчитываем пути для вершин, не входящих в *W.* Здесь далее через |...| обозначены пути до вершин и веса ребер. Проверяем вершину $v_2$: $|v_2| = 2$; $|v_1| + |e_{12}| = \infty$. Уменьшения нет – оставляем $|v_2|=2$ без изменения. Следующая в списке $v_4$: $|v_4|=\infty$; $|v_1| + |e_{14}| = \infty$. Также нет уменьшения. Далее проверяем $v_5$: $|v_5|=\infty$; $|v_1| + |e_{15}| = \infty$. Есть уменьшение, поэтому теперь будет $|v_5|=2$. И, наконец, $v_6$: $|v_6|=\infty$; $|v_1| + |e_{16}| = \infty$. Оставляем без изменений.

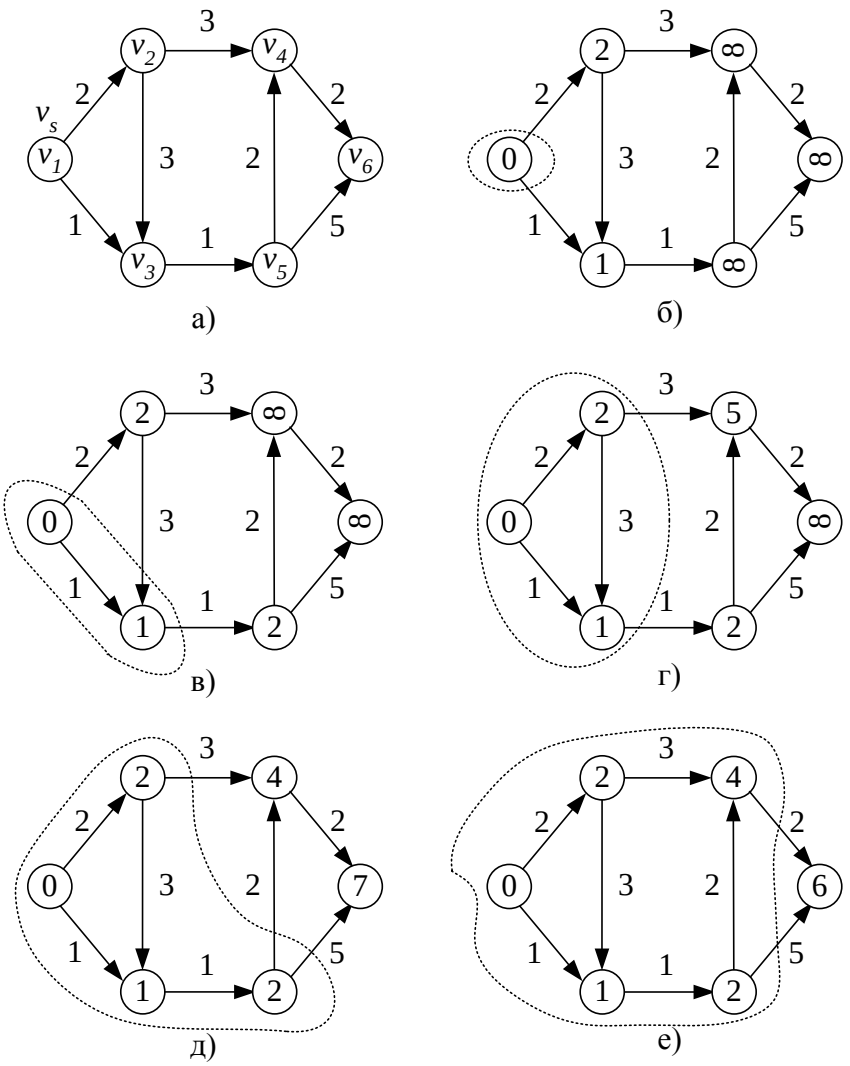


Рис. 12. Алгоритм Дейкстры решения задачи о кратчайшем пути

Выполняем шаг 3: проверяем, сколько вершин не попало в *W*. Осталось более одной вершины – снова идем на шаг 2. Теперь минимальный путь у вершин $v_2$ и $v_4$: по 2. Первой в списке идет $v_2$ – ее и добавляем в *W*. Пересчитываем пути до остальных вершин (рис. 12, *г)*. Для $v_4$: $|v_4| = \infty$; $|v_2|+|e_{24}|=5$. Есть уменьшение, поэтому полагаем $|v_4| = 5$. Для $v_5$: $|v_5| = 2$; $|v_2| + |e_{25}| = \infty$ – оставляем без изменения. И для $v_6$: $|v_6|=\infty$; $|v_2| + |e_{26}|=\infty$ – также оставляем без изменения. Проверка на шаге 3 показывает, что осталось более одной вершины, не включенной в *W*, поэтому продолжаем процесс.

На этом этапе (это снова шаг 2) минимальный путь у $v_5$: $|v_5| = 2$. Добавляем эту вершину в *W*. Остались две вершины, не включенные в *W:* $v_4$, $v_6$ (рис. 12, *д)*. Пересчитываем их пути: $|v_4| = 5$; $|v_5| +|e_{54}|=4$ – есть уменьшение, полагаем $|v_4|= 4$. Далее: $|v_6| =\infty$; $|v_5| + |e_{56}|= 7$ – тоже есть уменьшение, поэтому теперь $|v_6| =7$. Шаг 3: пока осталось две вершины, поэтому снова идем на шаг 2.

Из двух оставшихся вершин минимальный путь у $v_4$: $|v_4| = 4$. Ее и добавляем в *W* (рис. 12, *е)*. Проверяем $v_6$: $|v_6|= 7$; $|v_4| + |e_{46}| = 6$ – уменьшаем $|v_6|$ до 6. Шаг 3: осталась единственная вершина $v_6$, поэтому заканчиваем алгоритм. В каждой вершине – кратчайший путь от $v_1$ до нее.

Правильность алгоритма Дейкстры доказывается по индукции, так как на каждой итерации путь до новой вершины находится таким образом, что получаются кратчайшие пути для всех вершин из *W*.

### 3. Алгоритм Флойда-Уоршелла

В этом алгоритме строится матрица кратчайших путей между всеми парами вершин орграфа. На входе нужно задать матрицу *D* размера *n*×*n,* в каждом элементе которой *d* находится вес дуги – длина пути из $v_i$ в $v_j$.. При этом диагональные элементы должны быть равны бесконечности: $\forall d_{ii} = \infty$. Весь алгоритм умещается в тройной цикл. Вот как он выглядит в синтаксисе MATLAB:

```
for j=1:n do,
  for i=setdiff([1:n],j) do,
     for k=setdiff([1:n],j) do,
     D(I,k)=min(D(I,k),D(i,j)+D(j,k));
     end
   end
end
```

Смысл этого тройного цикла показан на рис. 13: для каждой промежуточной вершины $v_1$ мы проверяем все возможные начальные вершины $v_i$ и конечные $v_k$, не совпадающие с $v_1$, но, возможно, совпадающие между собой. Если при каждой такой проверке сумма весов дуг $d_{ij} + d_{jk}$ окажется меньше $d_{ik}$, мы заменяем более длинный путь $d_{ik}$ более коротким $d_{ij} + d_{jk}$. Эта операция называется операцией треугольника.

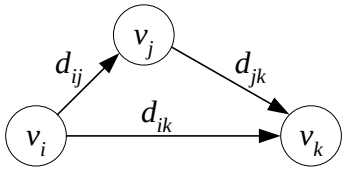


Рис.13. Операция треугольника

**Определение 1.** *Операцией треугольника* для фиксированной вершины $v_j$ называется операция:

$$d_{ik} = \min(d_{ik}, d_{ij} + d_{jk});\ i \in [1,n];\ i \neq j;\ k \in [1,n];\ k \neq j, \qquad (1)$$

где допускается *i*=*k*.

Таким образом, алгоритм Флойда-Уоршелла заключается в проведении операции треугольника для каждой вершины.

**Теорема 1.** Если все веса дуг неотрицательные, то после однократного проведения операции треугольника для всех вершин матрица D становится матрицей кратчайших путей.

*Доказательство.* Докажем по индукции, что после шага номер *s* матрица *D* станет матрицей кратчайших путей из любой вершины в любую по вершинам с

номерами $j \le s$. Начнем с $s$=1. Если провести операцию (1) только для $j$= 1, то пути из любой вершины в любую через $v_1$ будут кратчайшими. Значит, для $s$ =1 теорема имеет место.

Предположим, что теорема верна для $s$=$r$-1*:* пути из любой вершины в любую через вершины с номерами от 1 до $r$-1 минимальны по длине.

Проведем операцию треугольника еще и для промежуточной вершины номер $r$: $d_{ik} = \min(d_{ik}, d_{ir} + d_{rk})$. Докажем, что в этом случае путь из любой вершины в любую другую, проходящий через вершины $v_1$,$v_2$, ...,$v_r$, будет кратчайшим. Если реальный кратчайший путь на самом деле не проходит через новую вершину $v_r$, то минимальным будет первое число $d_{ik}$, и в этом случае теорема верна, так как добавление новой промежуточной вершины $v_r$ ничего не меняет. Если же реальный кратчайший путь пройдет через новую вершину $v_r$, то $d_{ik}$ будет заменено меньшей величиной $d_{ir}+d_{rk}$. Но каждое из слагаемых $d_{ir}$ и $d_{rk}$ является кратчайшим путем через промежуточные вершины $v_1$,$v_2$, ...,$v_{r-1}$, поэтому $d_{ir} + d_{rk}$ действительно будет кратчайшим путем из $v_i$ в $v_k$ через промежуточные вершины $v_1$,$v_2$, ...,$v_r$. Таким образом, по индукции теорема доказана.

## 4. Описание процедуры ShortPath

Рассмотрим реализацию алгоритма Флойда-Уоршелла на MATLAB. На вход подается список дуг размером $m \times 2$ или $m \times 3$ (идентификатор E). Если задано два столбца, то решается невзвешенная задача и ищется путь из минимального количества дуг, а если три – то взвешенная, и в этом случае ищутся пути минимального общего веса. На выходе процедура возвращает матрицу кратчайших путей размера $n \times n$.

Заголовок процедуры ShortPath и справочная информация к ней имеют вид:

```
function dSP=ShortPath(E)
% Функция dSP=ShortPath(E) решает задачу о кратчайшем пути
% между всеми вершинами орграфа.
% Входной параметр
% E(m,2) или (m,3)  – дуги орграфа и их веса;
% 1-й и 2-й элементы каждой строки – это номера вершин;
% 3-й элемент каждой строки – это вес дуги;
% m – количество дуг.
% Если задан массив E(m,2) ,  то все веса равны 1.
% Выходной параметр:
% dSP(n,n) – матрица кратчайших путей
% Каждый элемент dSP(i,j) – это кратчайший путь
%  из вершины i  в вершину j (может быть inf,
%  если вершина j не достижима из вершины i).
% Используется алгоритм Флойда-Уоршелла.
```

Процедура ShortPath проверяет исходные данные: наличие данных, размерность и количество столбцов входного массива, положительность и целочис-

ленность первых двух столбцов. Для запуска алгоритма Флойда-Уоршелла нужно задать матрицу весов дуг размером $n \times n$. Если какой-либо дуги нет, в соответствующем элементе записывается бесконечное число.

## 5. Выполнение заданий

**Задание 1.** Рассмотрите пример обращения к процедуре ShortPath.

Для иллюстрации работы функции ShortPath в качестве исходного приведен ориентированный граф с 11 вершинами и 22 ребрами. Введены веса дуг.

Нарисуйте данный орграф.

```
clear all
V=[[0 0];[1 1];[1 0];[1 -1];...
  [2 1];[2 0];[2 -1];[3 1];...
  [3 0];[3 -1];[4 0]]; % координаты вершин
E=[[1 2 5];[1 3 5];[1 4 5];[2 3 2];[3 4 2];[2 5 3];...
  [2 6 2];[3 6 5];[3 7 2];[4 7 3];[6 5 1];[6 7 1];...
  [5 8 5];[6 8 2];[6 9 3];[7 9 2];[7 10 3];[8 9 2];...
  [9 10 2];[8 11 5];[9 11 4];[10 11 4]]; % ребра и их веса
PlotGraph(V,E,'o'); % рисуем орграф
set(get(gcf,'CurrentAxes'),...
 'FontName','Times New Roman Cyr','FontSize',14)
title('\bfИсходный орграф со взвешенными дугами')
```

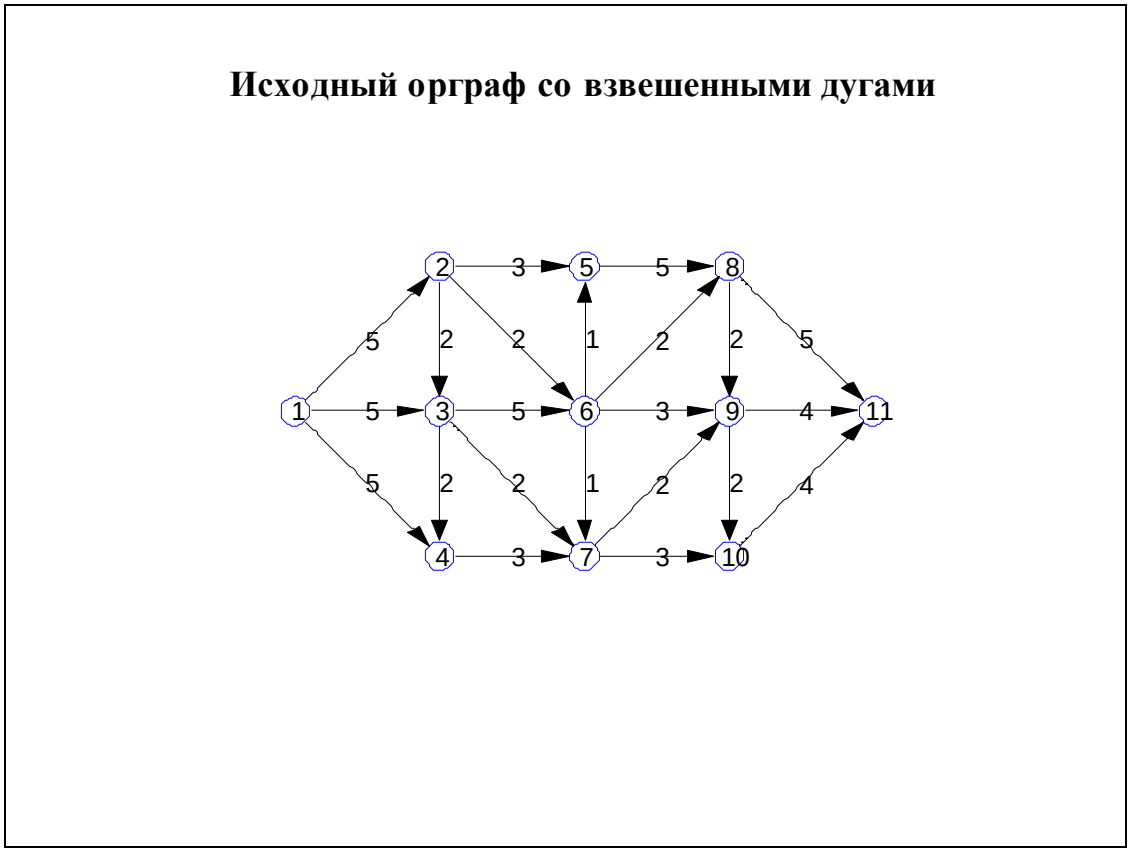


Рис. 14. Исходный ограф, нарисованный с помощью MATLAB

Найдем матрицу кратчайших путей между любой парой вершин и напечатаем ее.

```
dSP=ShortPath(E);
disp('Кратчайшие пути между всеми вершинами:');
fprintf('    %2.0f',1:size(dSP,2));
fprintf('\n');
for k1=1:size(dSP,1),
  fprintf('%2.0f',k1)
  fprintf('%6.2f',dSP(k1,:))
  fprintf('\n')
end
```

Кратчайшие пути между всеми вершинами:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Inf | 5.00 | 5.00 | 5.00 | 8.00 | 7.00 | 7.00 | 9.00 | 9.00 | 10.00 | 13.00 |
| 2 | Inf | Inf | 2.00 | 4.00 | 3.00 | 2.00 | 3.00 | 4.00 | 5.00 | 6.00 | 9.00 |
| 3 | Inf | Inf | Inf | 2.00 | 6.00 | 5.00 | 2.00 | 7.00 | 4.00 | 5.00 | 8.00 |
| 4 | Inf | Inf | Inf | Inf | Inf | Inf | 3.00 | Inf | 5.00 | 6.00 | 9.00 |
| 5 | Inf | Inf | Inf | Inf | Inf | Inf | Inf | 5.00 | 7.00 | 9.00 | 10.00 |
| 6 | Inf | Inf | Inf | Inf | 1.00 | Inf | 1.00 | 2.00 | 3.00 | 4.00 | 7.00 |
| 7 | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | 2.00 | 3.00 | 6.00 |
| 8 | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | 2.00 | 4.00 | 5.00 |
| 9 | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | 2.00 | 4.00 |
| 10 | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | 4.00 |
| 11 | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf | Inf |

**Задание 2.** Рассмотрите свой пример ориентированного графа. Введите веса дуг. Нарисуйте орграф с помощью MATLAB.

Найдите матрицу кратчайших путей между любой парой вершин и напечатайте ее.

**Контрольные вопросы**

1. Сформулируйте постановку задачи о кратчайшем пути.
2. Объясните сущность алгоритмов Дейкстры и Флойда-Уоршелла.
3. Какие исходные данные необходимо задать для вызова процедуры ShortPath?

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Белов, В.В. Теория графов: учеб. пособие для втузов/ В.В. Белов, Е.М. Воробьёв, В.Е. Шаталов. − М.: Высш. шк., 1976.

2. Горбатов, В.А. Основы дискретной математики: учеб. пособие для студентов вузов/ В.А. Горбатов. − М.: Высш. шк., 1986.

3. Иглин, С.П. Математические расчеты на базе MATLAB/ С.П. Иглин. − СПб.: БВХ-Петербург, 2005.

4. Дьяконов, В.П. MATLAB 6/6.1/6.5 + Simulink 4/5. Основы применения: полное руководство пользователя/ В.П. Дьяконов. − М.: Солон-Пресс, 2002.

5. Потемкин, В.Г. Система MATLAB: справ. пособие/ В.Г. Потемкин. − М.: Диалог-МИФИ, 1998.

6. Mathwork File Exchange Central.– http://www. Mathworks.com/mathcentral/fileexchange/loadCategory.do.

## СОДЕРЖАНИЕ