Лабораторная работа № 9. Представление графов

**Цель работы:** изучить основные способы представления графов в компьютере.

### Краткие теоретические сведения

* 1. **Основные определения**

**Граф** – это совокупность двух конечных множеств: *множества* точек и *множества* линий, попарно соединяющих некоторые из этих точек.

Множество точек называется ***вершинами (узлами) графа***.

Множество линий, соединяющих *вершины графа*, называются ***ребрами (дугами) графа***.

На рис. 1изображен типичный граф, на котором кружки являются *вершинами*, а линии — *ребрами* графа. Вершины обычно каким-то образом помечаются — часто для пометки используются буквы. Каждое ребро ограничивается двумя вершинами, находящимися на его концах.



Рис.1. Типичный граф

***Инцидентность*** — понятие, используемое только в отношении ребра или дуги и вершины: если {\displaystyle v\_{1},v\_{2}}v1,v2 — вершины, а {\displaystyle e=(v\_{1},v\_{2})}e(v1,v2) — соединяющее их ребро, тогда вершина v1{\displaystyle v\_{1}} и ребро {\displaystyle e} e инцидентны, вершина {\displaystyle v\_{2}}v2 и ребро {\displaystyle e}e тоже инцидентны (соединены). Две вершины (или два ребра) инцидентными быть не могут. Для обозначения ближайших вершин (рёбер) используется понятие *смежности*.

***Смежность*** — понятие, используемое в отношении только двух *рёбер* либо только двух *вершин*: Два ребра, *инцидентные* одной вершине (соединенные с одной вершиной), называются **смежными**; две вершины, *инцидентные*  одному ребру (соединенные с одним ребром) , также называются **смежными**.

Так, на рис. 1 вершины I и G являются смежными, а вершины I и F — нет. Вершины, смежные с некоторой вершиной, называются ее *соседями*. Например, соседями G являются I, H и F.

**Виды графов**

***Ориентированный граф (орграф)*** – *граф*, у которого все *ребра* ориентированы, т.е.  *ребрам*  которого присвоено направление.

***Неориентированный граф (неорграф)*** – *граф*, у которого все *ребра* неориентированы, т.е.  *ребрам* которого не задано направление.

***Смешанный граф*** – *граф*, содержащий как ориентированные, так и неориентированные *ребра*.

Граф называется ***связным***, если от каждой вершины к любой другой вершине ведет хотя бы один путь (рис. 2, *а*). Но если из одной из вершин графа невозможно добраться до какой-нибудь другой вершины, то граф называется несвязным— пример показан на рис. 2, *б*.

Несвязный граф состоит из нескольких областей. На рис. 2, *б* одна из таких областей состоит из вершин A и B, а другая — из вершин C и D.



Рис. 2 Связный и несвязный графы

На некоторых графах ребрам присваиваются *веса* — числа, представляющие физическое расстояние между двумя вершинами, время перехода от вершины к вершине или затраты на такой переход (скажем, в примере с авиарейсами). Такие графы называются ***взвешенными.***

***Путь***представляет собой последовательность вершин и соединяющих их ребер. На рис. 1 можно выделить путь от вершины B к вершине J, проходящий через вершины A и E ;этот путь можно обозначить BAEJ. Между двумя вершинами может существовать более одного пути; например, от B к J также ведет путь BCDJ.

***Простой путь в неориентированном графе*** - это последовательность смежных ребер **(v1,v2), (v2,v3),..., (vk-1,vk)**, таких, что все **vi** кроме, быть может, **v1** и **vk** различны.

Число ребер, составляющих простой путь, называется ***длиной простого пути***.

Простой путь называется:

* ***в неориентированном графе - цепью***, а
* ***в ориентированном графе - путем***.

Простой путь, начинающийся и заканчивающийся в одной и той же вершине, называется:

* ***в неориентированном графе - циклом***, а
* ***в ориентированном графе - контуром***.

***Расстояние между вершинами*** **u** и **v** в связном неориентированном графе - это минимальная длина цепи между **u** и **v**.

***Петлей*** называется *ребро*, соединяющее вершину саму с собой.

*Ребра*, соединяющие одну и ту же пару вершин, называются ***кратными*.**

***Простой граф*** – это *граф*, в котором нет ни петель, ни кратных ребер.

***Мультиграф*** – это *граф*, у которого любые две вершины соединены более чем одним *ребром*.

***Помеченный граф*** (*Labeled graph) -* граф, вершинам которого приписаны метки, например номера *n* или символы из какого-нибудь алфавита.

***Маршрутом*** в графе называется конечная чередующаяся последовательность смежных вершин и ребер, соединяющих эти вершины.

*Маршрут* называется ***открытым***, если его начальная и конечная вершины различны, в противном случае он называется замкнутым.

*Маршрут* называется **цепью**, если все его *ребра* различны.

Открытая цепь называется **путем**, если все ее вершины различны.

Замкнутая цепь называется **циклом**, если различны все ее вершины, за исключением концевых.

**Гамильтонов путь -**путь без петель, проходящий через каждую вершину графа ровно один раз.

**Гамильтоновым циклом** называется маршрут, включающий ровно по одному разу каждую вершину графа.

**Эйлеров путь** (**эйлерова цепь**) в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.

**Эйлеров цикл** — эйлеров путь, являющийся циклом, то есть замкнутый путь, проходящий через каждое ребро графа ровно по одному разу.

*Граф* называется ***связным*,** если для любой пары вершин существует соединяющий их *путь*.

***Вес вершины*** – число (действительное, целое или рациональное), поставленное в соответствие данной вершине (интерпретируется как *стоимость*, *пропускная способность* и т. д.).

***Вес (длина) ребра*** – число или несколько чисел, которые интерпретируются по отношению к **ребру как** ***длина*,** ***пропускная способность* и т. д.**

***Взвешенный граф*** – *граф*, каждому ребру которого поставлено в соответствие некое *значение* (*вес* *ребра*).

Обозначения, используемые в теории графов:

* G=(V, E), здесь G – граф, V – его вершины, а E – ребра;
* |V| – порядок (число вершин);(vertex)
* |E| – размер графа (число рёбер).(edge)
  1. **Представление графа в программе**

**Вершины**

В абстрактной программе для работы с графом можно пронумеровать вершины от 0 до *N* – 1 (где *N* — количество вершин). На практике вершины обычно представляют реально существующие объекты, а объекты описываются полями данных. Например, если вершина представляет город в модели сети воздушного сообщения, в ней может храниться название города, высота над уровнем моря, координаты и т. д. Таким образом, для представления вершин обычно удобно использовать объекты соответствующего класса. В примерах программ для каждой вершины хранится только буквенная метка (например, A) и флаг, используемый поисковыми алгоритмами (об этом ниже). Определение класса вершины графа Vertex выглядит так:

***class Vertex***

***{***

***public char label; // Метка (например, ‘A’)***

***public boolean wasVisited;***

***public Vertex(char lab) // Конструктор***

***{***

***label = lab;***

***wasVisited = false;***

***}***

***} // Конец класса Vertex***

Объекты вершин можно хранить в массиве и обращаться к ним по индексу. В наших примерах они будут храниться в массиве с именем vertexList. Для хранения вершин также можно использовать список или другую структуру данных. Впрочем, структура для хранения вершин выбирается только для удобства. Она никак не связана со способом соединения вершин ребрами. Для моделирования ребер графа понадобится другой механизм.

**Ребра**

Граф обычно не имеет такой жесткой структуры, как дерево. В двоичном дереве каждый узел имеет не более двух потомков, а в графе каждая вершина может быть соединена с произвольным количеством других вершин. Например, на рис. 2, *а* вершина A соединена с тремя другими вершинами, а вершина C соединена только с одной.

Для моделирования подобной нежесткой структуры нужен другой способ представления ребер. При работе с графами обычно применяются ***матрица смежности,*  *список смежности, матрица инцидентности, список ребер*.**

**Матрица смежности**

***Матрицей смежности*** *(англ. Adjacency matrix)* ***A=|| ai,j ||*** взвешенного графа **G=(V, E)** называется матрица **A[V x V],** , в которой ***ai,j*** — вес ребра, соединяющего вершины **Vi**  и **Vj**. Для невзвешенного графа ***ai,j*** =1 при наличии ребра между вершинами ai и aj и ***ai,j*** =0 при отсутствии такого ребра

Если граф содержит *N* вершин, то матрица смежности представляет собой массив *N* × *N*. На рис. 3 представлен граф и его матрица смежности.

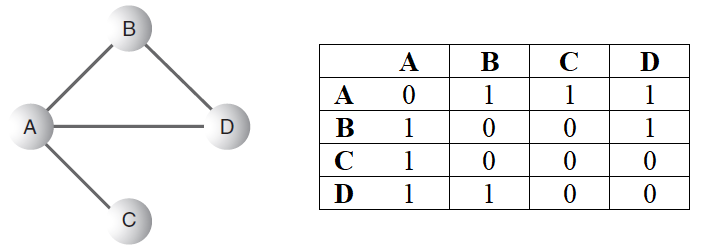


Рис. 3. Граф и его матрица смежности

Заголовками как строк, так и столбцов являются метки вершин. Ребро между двумя вершинами обозначается 1; при отсутствии ребра элемент матрицы равен 0.

Как видно из таблицы, вершина A является смежной со всеми тремя остальными вершинами, вершина B — с вершинами A и D, вершина C — только с A, а вершина D — с A и B. В этой матрице «соединение» вершины с самой собой обозначается 0, поэтому диагональ, идущая из левого верхнего в правый нижний угол (от A–A до D–D), заполнена нулями. Элементы диагонали не несут полезной информации, поэтому в них с таким же успехом можно записать 1, если это окажется удобнее в программе.

**Список смежности**

Вообще говоря, список смежности скорее является массивом списков (а иногда списком списков), содержащем информацию о том, какие вершины являются смежными по отношению к заданной. В таблице 2 приведены списки смежности для графа на рис. 3.

**Таблица 2.** Списки смежности

|  |  |
| --- | --- |
| **Вершина** | **Список смежных вершин** |
| A | B—>C—>D |
| B | A—>D |
| C | A |
| D | A—>B |

В этой таблице знаком —> обозначается связь в списке. Элементами списка являются вершины. Список смежности показывает, какие вершины являются смежными по отношению к заданной (то есть находятся от нее на расстоянии одного ребра)

**Матрица инцидентности**

***Матрица инцидентности графа*** — это матрица, значения элементов которой характеризуется инцидентностью соответствующих вершин графа (по горизонтали) и его рёбер (по вертикали). Для неориентированного графа элемент принимает значение веса ребра (или 1), если соответствующие ему вершина и ребро инцидентны. Для *ориентированного* графа элемент принимает значение веса ребра (или 1), если инцидентная вершина является началом ребра, и значение (-вес ребра или **-1)**, если инцидентная вершина является концом ребра; в остальных случаях (в том числе и для *петель*) значению элемента присваивается **0**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | a | b | c | d | | (a, b) | 1 | 1 | 0 | 0 | | (a, d) | 1 | 0 | 0 | 1 | | (а, c) | 1 | 0 | 1 | 0 | | (b, d) | 0 | 1 | 0 | 1 | |

Рис. 4.Матрица инцидентности графа

В программе матрица инцидентности задается, также как и матрица смежности, а именно при помощи двумерного массива. Его элементы могут быть инициализированы при объявлении, либо по мере выполнения программы.

**Список ребер**

***Список ребер*** – это таблица, содержащая обозначения пар вершин и вес соединяющих их ребер.

Список ребер можно представить таблицей, в каждой строке которой записаны две смежные вершины и вес, соединяющего их ребра.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | Начало | Конец | Вес | | А | B | 1 | | A | C | 1 | | A | D | 1 | | B | D | 1 | |

**Добавление вершин и ребер в граф**

Чтобы включить в граф новую вершину, нужно создать новый объект вершины оператором new и вставить его в массив вершин vertexList. В реальной программе объект вершины может содержать много полей данных, но для простоты далее используется только один символ. Таким образом, создание вершины выглядит примерно так:

***vertexList[nVerts++] = new Vertex('F');***

Команда вставляет в граф вершину F. В переменной nVerts хранится текущее количество вершин в графе. Способ добавления ребра зависит от выбранного представления (матрица смежности или списки смежности, матрица инцидентности или список ребер). Допустим, в программе используется матрица смежности, и в граф добавляется ребро между вершинами 1 и и 3. Эти числа соответствуют индексам массива vertexList, в котором хранятся вершины. При создании матрица смежности adjMat заполняется нулями. Чтобы вставить в нее ребро, выполните следующие команды:

***adjMat[1][3] = 1;***

***adjMat[3][1] = 1;***

В программе со списком смежности в список 3 добавляется вершина 1, а в список вершины 1 добавляется вершина 3.

**Класс Graph**

Класс Graph содержит методы для создания списка вершин и матрицы смежности, а также для добавления вершин и ребер в объект Graph:

***class Graph***

***{***

***private final int MAX\_VERTS = 20;***

***private Vertex vertexList[]; // Массив вершин***

***private int adjMat[][]; // Матрица смежности***

***private int nVerts; // Текущее количество вершин***

***// -------------------------------------------------------------***

***public Graph() // Конструктор***

***{***

***vertexList = new Vertex[MAX\_VERTS];***

***// Матрица смежности***

***adjMat = new int[MAX\_VERTS][MAX\_VERTS];***

***nVerts = 0;***

***for(int j=0; j<MAX\_VERTS; j++) // Матрица смежности***

***for(int k=0; k<MAX\_VERTS; k++) // заполняется нулями***

***adjMat[j][k] = 0;***

***}***

***// -------------------------------------------------------------***

***public void addVertex(char lab) // В аргументе передается метка***

***{***

***vertexList[nVerts++] = new Vertex(lab);***

***}***

***// -------------------------------------------------------------***

***public void addEdge(int start, int end)***

***{***

***adjMat[start][end] = 1;***

***adjMat[end][start] = 1;***

***}***

***// -------------------------------------------------------------***

***public void displayVertex(int v)***

***{***

***System.out.print(vertexList[v].label);***

***}***

***// -------------------------------------------------------------***

***} // Конец класса Graph***

В классе Graph вершины определяются своим индексом в списке vertexList.

Вывод вершины сводится к выводу ее символьной метки.

* 1. **Обход графа**

Одной из основных операций, выполняемых с графами, является определение всех вершин, достижимых от заданной вершины.

Существует два основных способа обхода графов: ***обход в глубину***и ***обход в ширину***. Оба способа в конечном итоге обеспечивают перебор всех соединенных вершин. Обход в глубину реализуется на базе стека, а обход в ширину реализуется на базе очереди. Это приводит к обходу вершин графа в разном порядке.

**Обход в глубину**

Посещается корень (выбранный узел), затем проход вдоль ребер графа до попадания в *тупик*. (Тупик - *вершина*, у которой все смежные с ней вершины посещены.) После попадания в *тупик* – возврат назад вдоль пройденного пути, пока не будет обнаружена *вершина не тупик,*  затем двигаться в направлении не посещенной вершины. Поиск оказывается завершенным при возвращении в начальную вершину, причем все смежные с ней вершины уже должны быть посещены.

Алгоритм обхода в глубину хранит в стеке информацию о том, куда следует вернуться при достижении «тупика».

**Пример**

Для объяснения принципов обхода в глубину будет использован граф на рис. 5.



Рис.5. Обход графа в глубину

Цифры на графе обозначают порядок посещения вершин.

Обход в глубину начинается с выбора отправной точки — в нашем примере это вершина A. Затем алгоритм выполняет три операции: посещает вершину, заносит ее в стек и помечает для предотвращения повторных посещений. Затем алгоритм переходит к любой вершине, смежной с A, которая еще не посещалась ранее. Будем считать, что вершины выбираются в алфавитном порядке; значит, это будет вершина B. Алгоритм посещает B, помечает эту вершину и заносит в стек.

Далее текущей вершиной является B, поэтому алгоритм делает то же, что и прежде: он переходит к смежной вершине, которая еще не посещалась ранее. В нашем примере это вершина F. Назовем этот процесс правилом 1.

**ПРАВИЛО 1**

Посетить смежную вершину, не посещавшуюся ранее, пометить ее и занести в стек.

Применение правила 1 снова приводит к вершине H. Однако на этот раз необходимо сделать что-то другое, потому что в графе не осталось непосещенных вершин, смежных с H. На помощь приходит правило 2.

**ПРАВИЛО 2**

Если выполнение правила 1 невозможно, извлечь вершину из стека.

В соответствии с этим правилом вершина H извлекается из стека, в результате чего алгоритм возвращается к F . У вершины F тоже нет непосещенных смежных вершин; она извлекается из стека. То же происходит с вершиной B. Теперь в стеке остается только вершина A. Однако у A имеются непосещенные смежные вершины, поэтому алгоритм посещает следующую вершину C. Но поскольку эта вершина тоже является тупиковой, она извлекается из стека, и алгоритм возвращается к A. Далее алгоритм посещает D, G и I, а затем извлекает их при достижении тупика в I . Происходит возврат к A. Алгоритм посещает вершину E и снова возвращается к A. Но на этот раз у вершины A не осталось непосещенных соседей, и она извлекается из стека. В стеке не остается элементов, и вступает в действие правило 3.

**ПРАВИЛО 3**

Если выполнение правил 1 и 2 невозможно, обход закончен.

В нашем примере вершины будут посещаться в порядке ABFHCDGIE.

Таким образом, алгоритм обхода в глубину стремится как можно быстрее удалиться от исходной вершины и возвращается к ней только при достижении тупика.

Если понимать под «глубиной» расстояние от исходной вершины, становится понятно, откуда взялось название алгоритма.

**Обход в ширину**

Алгоритм обхода в глубину старается как можно быстрее удалиться от исходной вершины на максимальное расстояние. При обходе в ширину, напротив, алгоритм стремится держаться как можно ближе к исходной вершине. Он посещает все вершины, смежные с исходной, и только после этого отходит дальше. Такая разновидность обхода реализуется на базе очереди (вместо стека).

**Пример**

На рис. 13.9 изображен тот же граф, что и на рис. 13.5, но на этот раз применяется обход в ширину. Как и прежде, цифрами обозначается порядок посещения вершин.

9

**Рис. 13.9.** Обход в ширину

Обход начинается с вершины A; алгоритм посещает ее и делает текущей. Далее применяются следующие правила.

**ПРАВИЛО 1**

Посетить следующую вершину, не посещавшуюся ранее, смежную с текущей вершиной,

пометить ее и занести в очередь.

**ПРАВИЛО 2**

Если выполнение правила 1 невозможно, извлечь вершину из очереди и сделать ее текущей вершиной.

**ПРАВИЛО 3**

Если выполнение правил 1 и 2 невозможно, обход закончен.

Таким образом, сначала посещаются все вершины, смежные с A. Алгоритм вставляет каждую вершину в очередь при посещении. Так посещаются вершины A, B, C, D и E. На этой стадии очередь (от начала к концу) содержит вершины BCDE. Других непосещенных и смежных с A вершин не осталось, поэтому алгоритм извлекает B из очереди и ищет вершины, смежные с B. Алгоритм находит вершину F и вставляет ее в очередь. Других непосещенных вершин, смежных с B, не осталось, поэтому вершина C извлекается из очереди. И снова непосещенных смежных вершин нет, алгоритм извлекает D из очереди и посещает G . У D не осталось других смежных непосещенных вершин, из очереди извлекается E. Теперь в очереди остались вершины FG. Алгоритм извлекает F и посещает H, затем извлекает G и посещает I.

В результате очередь содержит вершины HI . Но когда алгоритм извлекает каждую из них и не находит смежных непосещенных вершин, очередь остается пустой, и работа алгоритма завершается. Узлы посещаются в порядке ABCDEFGHI.

Обход в ширину обладает одним интересным свойством: он сначала находит все вершины, находящиеся на расстоянии одного ребра от начальной вершины, затем все вершины на расстоянии двух ребер и т. д. Это свойство может пригодиться при поиске кратчайшего пути от начальной вершины к заданной. Запустите обход в ширину, и при обнаружении заданной вершины вы точно знаете, что построенный путь является кратчайшим путем к вершине. Если бы более короткий путь существовал, то алгоритм обхода в ширину нашел бы его ранее.

1. Порядок выполнения работы
2. Выбрать по номеру студента в журнале свой вариант задания.
3. По выбранному списку ребер построить граф.
4. Построить для полученного графа матрицу смежности, список смежности, матрицу

инцидентности.

1. По желанию выполнить бонусные задания.

Содержание отчета о лабораторной работе

1. Титульный лист
2. Цель работы и задание для выполнения
3. Список ребер в соответствии с вариантом.
4. Граф, построенный на основе списка ребер.
5. Матрица смежности, список смежности, матрица инцидентности.
6. Выводы по лабораторной работе.

Контрольные вопросы

1. Что такое граф?
2. Какие существуют виды графов?
3. Что такое матрица смежности графа?
4. Что такое список смежности графа?
5. Что такое список инцидентности графа?
6. . Что такое список ребер графа графа?
7. Как можно в программе представить вершины графа?
8. Как можно в программе представить ребра графа?

Задания

Дан список дуг с указанием их длин. Составьте по нему рисунок ориентированного графа.

**1.** (0;1) – 3, (0;2) – 9, (1;2) – 5, (2;4) – 1, (1;3) – 8, (2;3) – 2, (3;5) – 4, (4;5) – 6.

**2.** (0;1) – 4, (0;2) – 5, (1;2) – 8, (2;4) – 3, (1;3) – 11, (2;3) – 5, (3;5) – 3, (4;5) – 6.

**3.** (0;1) – 3, (0;2) – 9, (1;2) – 12, (2;4) – 1, (1;3) – 2, (2;3) – 3, (3;5) – 10, (4;5) – 5.

**4.** (0;1) – 6, (0;2) – 2, (2;1) – 3, (2;4) – 6, (1;3) – 1, (2;3) – 5, (3;5) – 8, (4;5) – 7.

**5.** (0;1) – 6, (0;2) – 5, (1;2) – 1, (2;4) – 6, (1;3) – 7, (2;3) – 6, (3;5) – 8, (4;5) – 7.

**6.** (0;1) – 3, (0;2) – 2, (2;1) – 1, (2;5) – 3, (1;5) – 4, (5;4) – 8, (5;3) – 5, (3;4) – 3, (4;6) – 2, (3;6) – 4.

**7.** (0;1) – 10, (0;2) – 5, (2;1) – 4, (2;5) – 8, (1;5) – 3, (5;4) – 4, (5;3) – 2, (3;4) – 1, (4;6) – 5, (3;6) – 7.

**8.** (0;1) – 3, (0;2) – 2, (2;1) – 2, (2;5) – 12, (1;5) – 8, (5;4) – 2, (5;3) – 6, (3;4) – 1, (4;6) – 8, (3;6) – 3.

**9.** (0;1) – 2, (0;2) – 7, (2;1) – 1, (2;5) – 6, (1;5) – 12, (5;4) – 10, (5;3) – 5, (3;4) – 4, (4;6) – 2, (3;6) – 7.

**10.** (0;1) – 4, (0;2) – 2, (2;1) – 1, (2;5) – 7, (1;5) – 5, (5;4) – 4, (5;3) – 1, (3;4) – 4, (4;6) – 3, (3;6) – 7.

**11.** (0;2) – 2, (0;1) – 7, (2;1) – 4, (2;4) – 9, (1;3) – 3, (3;4) – 1, (4;6) – 2, (3;5) – 8, (6;5) – 4, (6;7) – 6, (5;7) – 5.

**12.** (0;2) –3, (0;1) – 5, (2;1) – 1, (2;4) – 4, (1;3) – 3, (3;4) – 5, (4;6) – 3, (3;5) – 7, (6;5) – 10, (6;7) – 5, (5;7) – 1.

**13.** (0;2) – 4, (0;1) – 6, (2;1) – 4, (2;4) – 6, (1;3) – 3, (3;4) – 2, (4;6) – 4, (3;5) – 7, (6;5) – 3, (6;7) – 8, (5;7) – 5.

**14.** (0;2) – 3, (0;1) – 1, (2;1) – 4, (2;4) – 2, (1;3) – 6, (3;4) – 4, (4;6) – 3, (3;5) – 6, (6;5) – 4, (6;7) – 9, (5;7) – 7.

**15.** (0;2) – 8, (0;1) – 8, (2;1) – 3, (2;4) – 6, (1;3) – 5, (3;4) – 4, (4;6) – 7, (3;5) – 4, (6;5) – 6, (6;7) – 3, (5;7) – 6.

**16.** (0;1) – 3, (0;2) – 3, (1;4) – 3, (2;5) – 3, (1;3) – 2, (0;3) – 4, (2;3) – 2, (3;4) – 2, (3;6) – 4, (3;5) – 2, (4;6) – 3.

**17.** (0;1) – 3, (1;2) – 2, (1;4) – 6, (2;5) – 5, (1;3) – 7, (0;3) – 1, (2;3) – 9, (3;4) – 5, (3;6) – 8, (3;5) – 3, (4;6) – 4.

**18.** (1;2) – 4, (0;2) – 5, (1;4) – 7, (2;5) – 7, (1;3) – 5, (0;3) – 8, (2;3) – 4, (3;4) – 2, (3;6) – 7, (3;5) – 1, (4;6) – 2.

**19.** (0;1) – 5, (1;2) – 5, (1;4) – 4, (2;5) – 5, (1;3) – 3, (0;3) – 1, (2;3) – 3, (3;4) – 3, (3; 6) – 4, (3;5) – 3, (4;5) – 7.

**20.** (0;1) – 6, (0;2) – 7, (1;4) – 8, (2;5) – 9, (1;3) – 8, (0;3) – 4, (2;3) – 6, (3;4) – 8, (3;6) – 7, (3;5) – 5, (5;6) – 5.

**21.** (0;1) – 2, (1;2) – 3, (0;2) – 6, (2;3) – 4, (3;4) – 2, (2;4) – 7, (4;5) – 5, (5;6) – 3, (4;6) – 9.

**22.** (0;1) – 3, (1;2) – 5, (0;2) – 7, (2;3) – 6, (3;4) – 5, (2;4) – 12, (4;5) – 3, (5;6) – 2, (4;6) – 4.

**23.** (0;1) – 5, (1;2) – 4, (0;2) – 10, (2;3) – 4, (3;4) – 5, (2;4) – 8, (4;5) – 7, (5;6) – 6, (4;6) – 14.

**24.** (0;1) – 2, (1;2) – 4, (0;2) – 5, (2;3) – 4, (3;4) – 5, (2;4) – 8, (4;5) – 3, (5;6) – 2, (4;6) – 4.

**25.** (0;1) – 3, (1;2) – 2, (0;2) – 5, (2;3) – 1, (3;4) – 1, (2;4) – 3, (4;5) – 2, (5;6) – 2, (4;6) – 3.

**26.** (0;1) – 10, (0;2) – 4, (1;4) – 8, (2;5) – 20, (3;1) – 5, (4;3) – 3, (2;3) – 4, (3;5) – 17, (4;6) – 7, (5;6) – 5.

**27.** (0;1) – 6, (0;2) – 2, (1;4) – 4, (2;5) – 15, (3;1) – 2, (4;3) – 3, (2;3) – 13, (3;5) – 2, (4;6) – 7, (5;6) – 1.

**28.** (0;1) – 2, (0;2) – 3, (1;4) – 1, (2;5) – 10, (3;1) – 1, (4;3) – 6, (2;3) – 4, (3;5) – 5, (4;6) – 20, (5;6) – 6.

**29.** (0;1) – 2, (0;2) – 3, (1;4) – 7, (2;5) – 6, (3;1) – 1, (4;3) – 2, (2;3) – 2, (3;5) – 3, (4;6) – 8, (5;6) – 10.

**30.** (0;1) – 3, (0;2) – 7, (1;4) – 8, (2;5) – 3, (3;1) – 1, (4;3) – 4, (2;3) – 2, (3;5) – 2, (4;6) – 7, (5;6) – 6.

**Бонусные задания (5 баллов за каждое)**

* 1. Создать программу для построения и вывода на экран изображения графа по одной из форм представления его ребер (по списку ребер, по матрице смежности, по списку смежности, по матрице инцидентности).
  2. Создать программу для построения по одной из форм представления графа оставшихся трех форм представления ребер.