О представлении графа в памяти компьютера и о «просмотре» его вершин

Граф – это пара множеств, обозначим как G=(V, E). Первое множество определяется как множество вершин V графа, а второе множество – множество ребер E. Ребра, соединяющие элементы множества V, могут быть ориентированными (их называют в этом случае дугами) или неориентированными и составлять соответственно *ориентированный граф* или *неориентированный граф*.

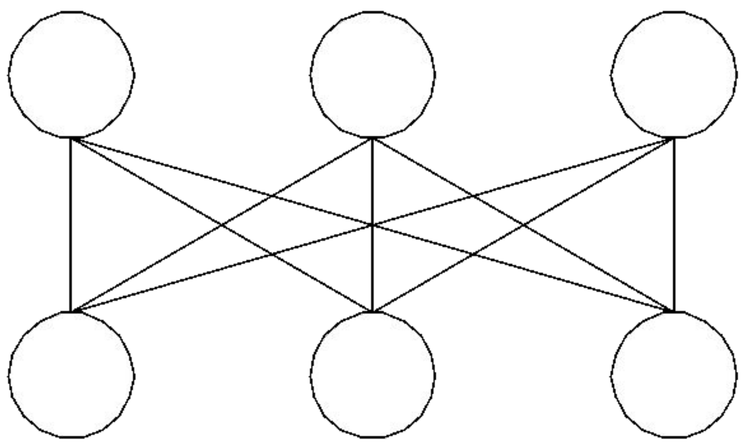

Рис. 1.1. Пример непомеченного и неориентированного графа

Граф называется *помеченным* (или пронумерованным), если его вершинам приписаны различные метки. Обычно в качестве меток используются целые положительные числа в интервале от 1 до n, где n равно количеству вершин графа – |V|. Количество ребер будем обозначать как m=|E|. На рис. 1.1 дан пример непомеченного, неориентированного графа (точки пересечения ребер не являются вершинами графа), а на рис. 1.2 – помеченного и ориентированного.

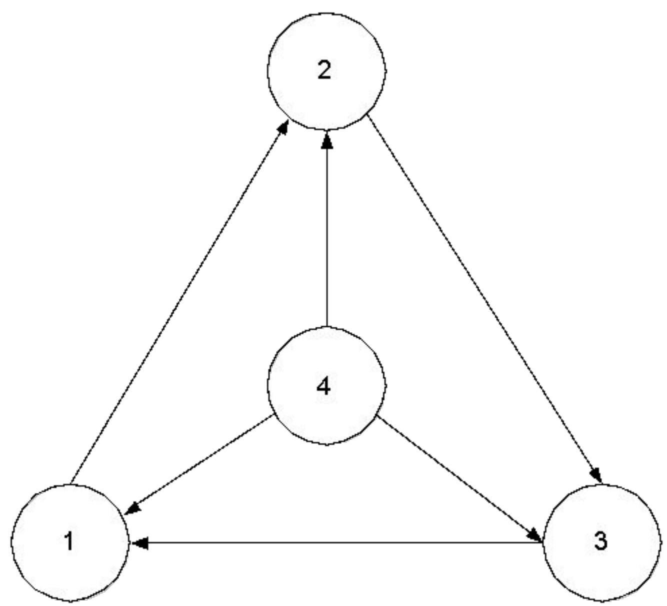

Рис. 1.2. Пример помеченного, ориентированного графа

Вершины, соединенные ребром, называются *смежными*. Ребра, имеющие общую вершину, также называются смежными. Ребро и любая из его двух вершин называются *инцидентными*. Ребро (i, j) инцидентно вершинам i и j. Каждый граф можно представить на плоскости множеством точек (кружков), соответствующих вершинам, которые соединены отрезками прямых линий, соответствующими ребрам.

Выбор соответствующей структуры данных для представления графа в памяти компьютера имеет принципиальное значение при разработке эффективных алгоритмов. Рассмотрим два способа.

Первый способ. Матрица смежности A – это двумерный массив размерности n·n.

$$A[i,j]=\begin{cases}1, \textit{если вершины с номерами } i \textit{ и } j \textit{ смежны}\\ 0, \textit{если вершины с номерами } i \textit{ и } j \textit{ не смежны}\end{cases}$$

Для неориентированного графа матрица смежности A симметрична относительно главной диагонали, для ориентированного графа – нет.

Пример. На рис. 1.3 приведен пример неориентированного графа.

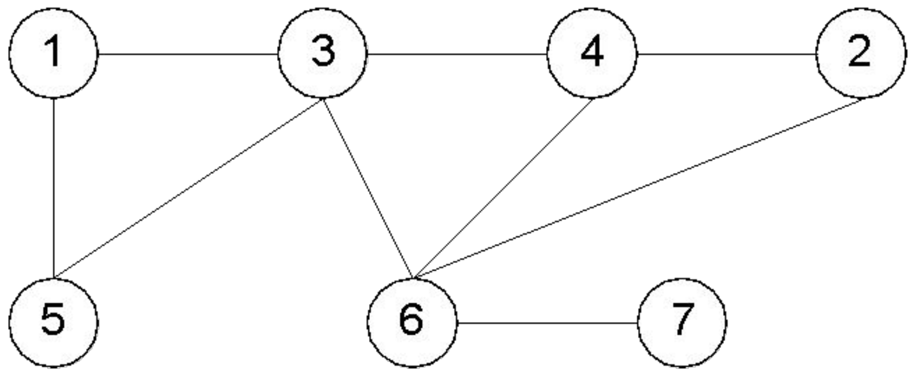

Рис. 1.3. Пример неориентированного графа

Его матрица смежности имеет вид: .

Формирование A не представляет особой сложности. Фрагмент реализации (предполагается, что первоначально все элементы A равны нулю):

```
For i := 1 To n-1 Do
 For j := i + 1 To n Do Begin
    Read(A[i,j]);
    A[j,i] := A[i,j];
  End;
```

Очевидно, что время, затрачиваемое на эти действия, есть $O(n^2)$ и все алгоритмы, требующие просмотра элементов A, будут иметь не меньшую временную сложность.

Для ориентированного графа приведенный фрагмент реализации выглядит несколько иначе.

```
For i := 1 To n Do
 For j := 1 To n Do
  Read(A[i,j]);
```

Второй способ. Он заключается в том, чтобы задать только перечень ребер графа. В этом случае возникает необходимость в присвоении меток ребрам, то есть их требуется перенумеровать. Самый простейший случай – это двумерный массив размерность m·2. Каждое ребро задается один раз. Так для примера на рис. 1.3 он, назовем его W, может иметь вид: $W = \begin{pmatrix} 1 & 3 \\ 1 & 5 \\ 2 & 4 \\ 2 & 6 \\ 3 & 4 \\ 3 & 5 \\ 3 & 6 \\ 4 & 6 \\ 6 & 7 \end{pmatrix}$. Мы неявно выполнили операцию присвоения ребрам номеров. Так ребро (1, 3) имеет номер один и так далее. Но не только. Если задать вопрос, а какой граф ориентированный или неориентированный представлен этим описанием, то ответ может быть и положительным, и отрицательным. Требуется некое соглашение (оно обязательно) типа: если граф неориентированный, то каждое ребро следует описывать дважды (количество строк в массиве удваивается). Например, ребро между вершинами 1 и 3 должно быть представлено и строкой (1, 3), и строкой (3, 1). Но даже в этом случае, это простое представление графа в памяти компьютера малопродуктивно. Так, даже простой просмотр вершин графа становится трудоемкой задачей, как по времени, так и по изяществу реализации.

Изменим принцип хранения списка ребер графа. То, что в этом случае необходимо пометить ребра (произвольным образом, только без повторений и разрывов в нумерации) очевидно. Пусть для графа на рис. 1.3 сделана разметка так, как показано на рис. 1.4. Граф неориентированный, поэтому каждое ребро помечается дважды.

Принцип. Для каждой вершины k (k=1..n) графа в массиве last (элемент last[k]) храним номер последнего ребра, связанного с ней. Номер ребра является индексом элемента в массиве t, а значением элемента t[last[k]] является номер вершины графа, с которой связана вершина k (ребро описано). В дополнительном массиве prev хранится номер предшествующего ребра графа, связанного с вершиной k.

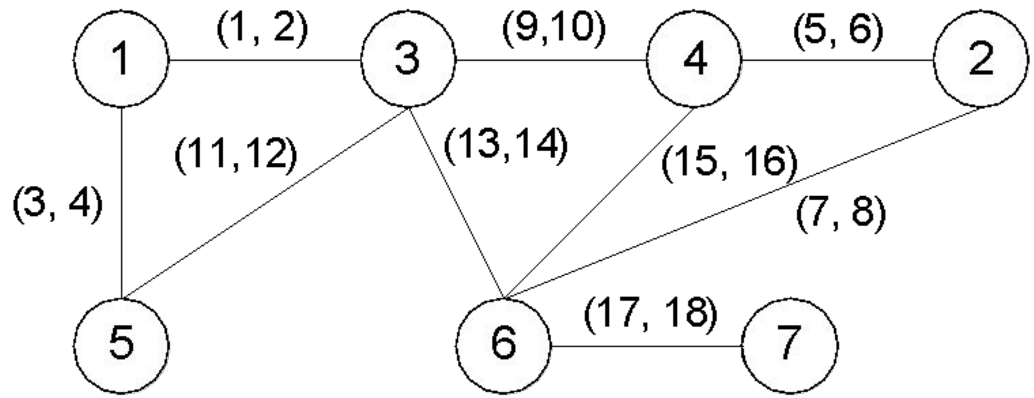

Рис. 1.4. Пример неориентированного графа с помеченными ребрами

Для рассматриваемого примера last[3]=13, t[13]=6, prev[13]=11 и затем – t[11]=5, prev[11]=9. Продолжим: t[9]=4, prev[9]=2; t[2]=1, prev[2]=0 – перечень ребер, выходящих их вершины с номером 3, перечислен. Для всех вершин графа он приведен в табл. 1.1 (i=last[k], k=1..n).

Таблица 1.1

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
t	3	1	5	1	4	2	6	2	4	3	5	3	6	3	6	4	7	6
prev	0	0	1	0	0	0	5	0	2	6	9	4	11	8	10	14	16	0
last	3	7	13	15	12	17	18											

Этот способ описания с помощью перечня ребер применим и для ориентированных графов. Изменим пример (рис. 1.5).

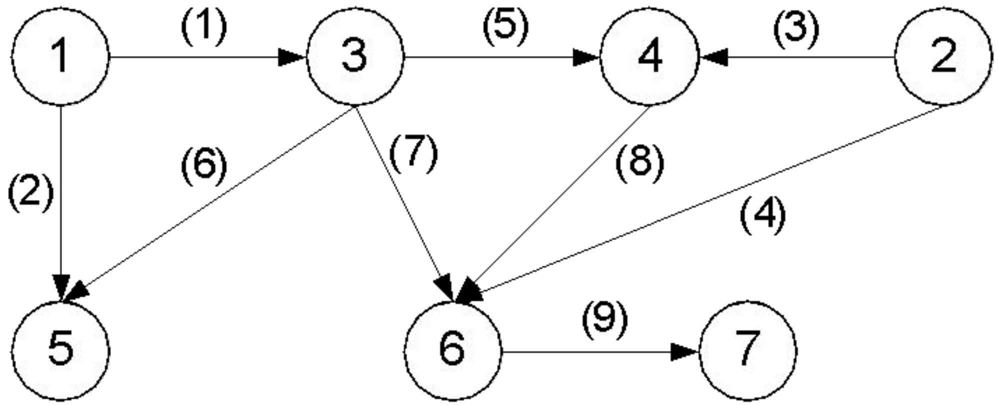

Рис. 1.5. Пример ориентированного графа с помеченными ребрами

Его описание в памяти компьютера будет иметь вид, приведенный в табл. 1.2.

Таблица 1.2

i	1	2	3	4	5	6	7	8	9
t	3	5	4	6	4	5	6	6	7
prev	0	1	0	3	0	5	6	0	0
last	2	4	7	8	0	9	0		

```
Const
  MAXV = ...;
  MAXE = ...;
Type
  graph_t = Record
      v, e : Integer;
      last : Array[1..MAXV] Of Integer;
      t, prev : Array[1..MAXE] Of Integer;
    End;
Procedure read_digraph(Var g : graph_t);
 Var i, u, v : Integer;
 Begin
  Read(g.v, g.e);
  For i := 1 To g.e Do Begin
    Read(u, v);
    g.t[i] := v;
    g.prev[i] := g.last[u];
    g.last[u] := i;
  End;
 End;
```

Поиск в глубину

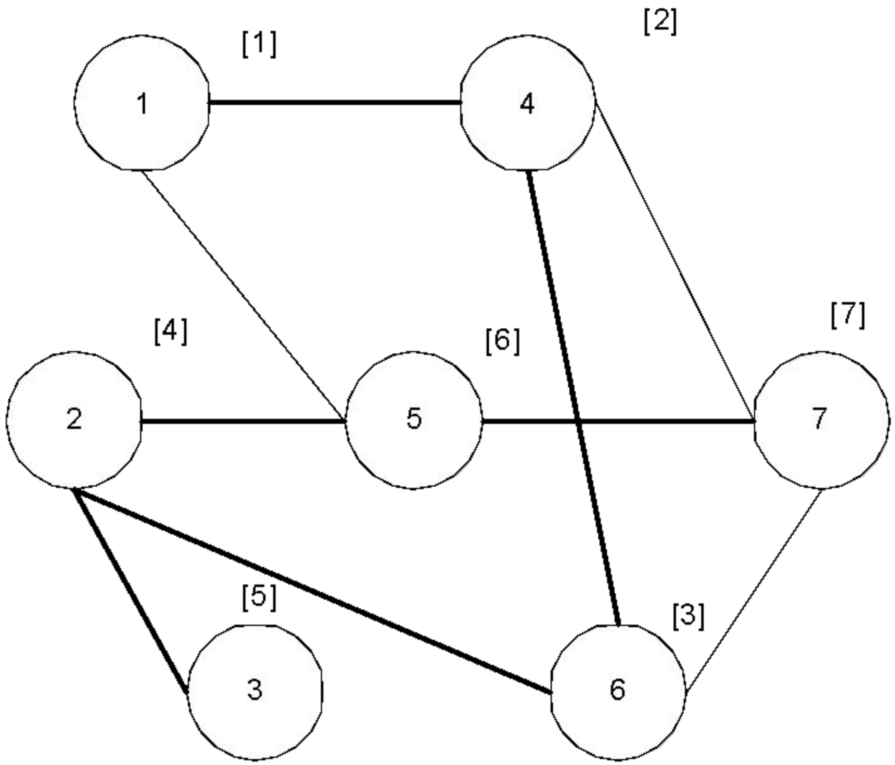

Рис. 1.6. Пример обхода вершин графа методом в глубину. Вершины графа просматриваются в очередности: 1, 4, 6, 2, 3, 5, 7. Очередность (порядок просмотра) приводится в квадратных скобках

```
Procedure dfs(v : Integer);
 Var j : Integer;
 Begin
  u[v] := True;
  Write(v:3);
  For j := 1 To n Do
   If (A[v,j] <> 0) And Not u[j] Then dfs(j);
 End;
```

Фрагмент основной логики.

```
For i := 1 To n Do u[i] := False;
For i := 1 To n Do
  If Not u[i] Then dfs(i);
```

Временная сложность метода $O(n^2)$ и её не улучшить при данном описании графа в памяти компьютера.

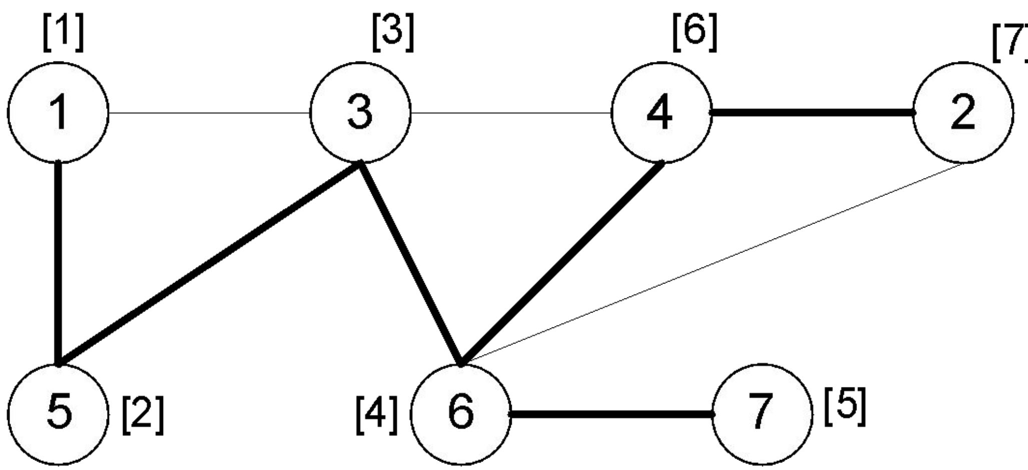

Рис. 1.7. Пример графа и его обхода в глубину при описании с помощью списка ребер

```
Procedure dfs(v : Integer);
 Var p : Integer;
 Begin
  Write(v,' ');
  u[v] := True;
  p := g.last[v];
  While p <> 0 Do Begin
    If Not u[g.t[p]] Then dfs(g.t[p]);
    p := g.prev[p];
   End;
 End;
```

Поиск в ширину

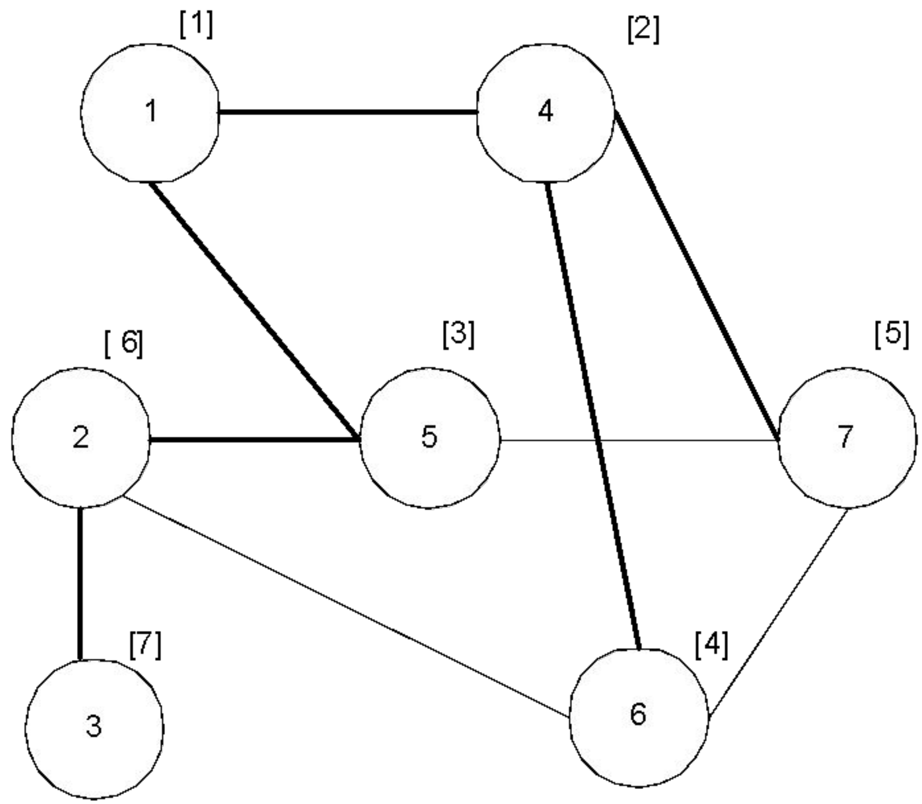

Рис. 1.8. Пример обхода вершин графа методом в ширину. Вершины графа просматриваются в очередности: 1, 4, 5, 6, 7, 2, 3. Очередность (порядок просмотра) приводится в квадратных скобках

```
Procedure breadth_search(v : Integer);
  Var turn : Array[1..n] Of Integer;{Очередь.}
      u : Array[1..n] Of Boolean;
      head, tail : Integer;{Указатели очереди.}
      j : Integer;
  Begin
   For j := 1 To n Do Begin
     turn[j] := 0;
     u[j] := False;
   End;
   head := 0; {Начальная инициализация.}
   tail := 1; {В очередь записываем вершину с номером v.}
   turn[tail] := v;
   u[v] := True;
   While head < tail Do Begin {Пока очередь не пуста.}
    head := head+1; {«Берем» элемент из очереди.}
    WriteLn(turn[head]);
    v := turn[head];
    For j := 1 To n Do {Просмотр всех вершин, связанных с
вершиной v.}
     If (A[v,j] <> 0) And Not u[j] Then Begin {Если
вершина ранее не просмотрена, то заносим её номер в
очередь.}
      tail := tail+1;
      turn[tail] := j;
      u[j] := True;
     End;
   End;
  End;
```

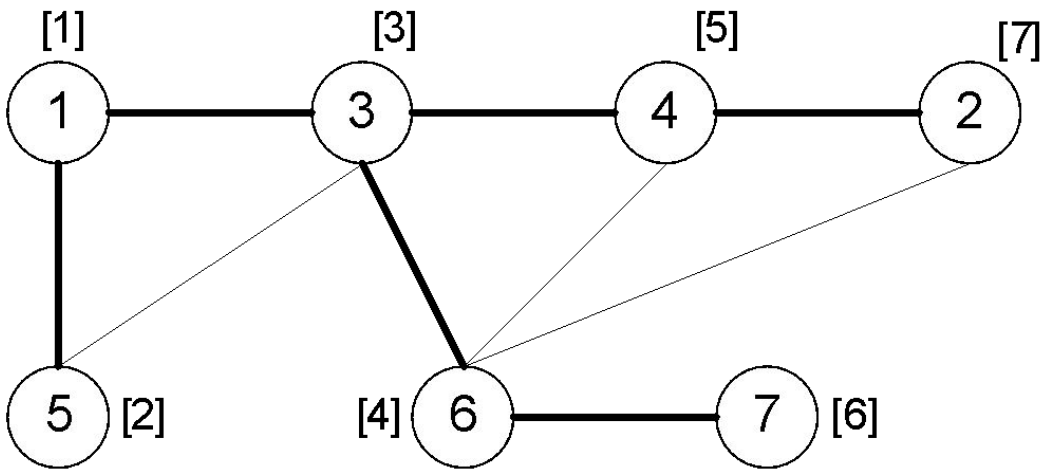

Рис. 1.9. Пример обхода вершин графа в ширину при его описании с помощью списка ребер. Выделены ребра, по которым осуществлялся переход