Лабораторная работа № 7

ГРАФЫ В ПРИКЛАДНЫХ АЛГОРИТМАХ

Цель работы: изучение и анализ структур данных, применяемых для описания графов в прикладных программах; анализ и программная реализация алгоритмов систематического обхода вершин графа; разработка и исследование прикладных программ, в которых применяются неориентированные и ориентированные графы.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Машинное представление графов

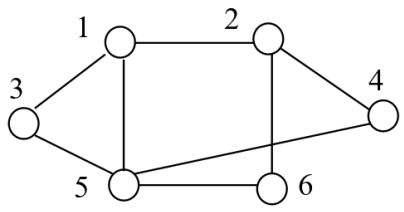
В математике граф $G = (V, E)$ определяется как множество вершин $V = \{v_1, v_2, v_n\}$ и множество ребер $E = \{e_1, e_2, ..., e_m\}$. Каждое ребро e_k задает пару вершин v_i и v_j, которые оно соединяет.

Если пара вершин каждого ребра неупорядочена, что записывается как $e_k = [v_i, v_j]$ или $e_k = [v_j, v_i]$, то граф называется *неориентированным*.

Если пара вершин каждого ребра упорядочена, что записывается как $e_k = (v_i, v_j)$, то граф называется *ориентированным*, а ребра с заданным направлением называются *дугами*.

Матричные представления. Наиболее известным способом представления графа является *матрица смежности* $S = [s_{ij}]_{n\times n}$ с элементами $s_{ij} = 1$, если $[v_j, v_i] \in E$ или $(v_i, v_j) \in E$ и $s_{ij} = 0$ в противном случае.

Примеры матриц смежности для неориентированного и ориентированного графов показаны на рис. 1,б и рис. 2,б соответственно.



	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	0	1	0	1
3	1	0	0	0	1	0
4	0	1	0	0	1	0
5	1	0	1	1	0	1
6	0	1	0	0	1	0

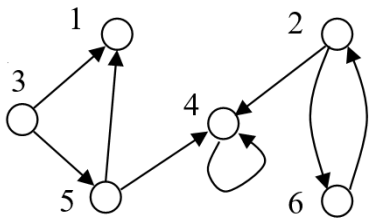
1	2	3	5	0
2	1	4	6	0
3	1	5	0	0
4	2	5	0	0
5	1	3	4	6
6	2	5	0	0

а б в

Рис. 1. Матричное представление неориентированного графа:
а - граф; б – матрица смежности; в – векторы смежности

Основным преимуществом матрицы смежности является удобство обращения к данным. В частности:

- наличие ребра (дуги) устанавливается выбором одного элемента;
- все вершины, смежные v_i, определяются просмотром i-й строки;
- вершины, для которых смежной является вершина v_j, находятся просмотром j-го столбца и т.п.



	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	1	0	1
3	1	0	0	0	1	0
4	0	0	0	1	0	0
5	1	0	0	1	0	0
6	0	1	0	0	0	0

1	0	0
2	4	6
3	1	5
4	4	0
5	1	4
6	2	0

а б в

Рис. 2. Матричное представление ориентированного графа:
а - граф; б – матрица смежности; в – векторы смежности

Недостатком использования матрицы смежности является тот факт, что независимо от числа ребер объем необходимой памяти составляет n^2 ячеек. Поэтому для чтения матрицы или нахождения в ней необходимого элемента требуется время порядка $O(n^2)$, что не позволяет создавать алгоритмы с временем $O(n)$ даже для работы с графами, имеющими порядка $O(n)$ ребер или дуг. Кроме того, проверка большинства свойств графа (наличие циклов, связность и др.) на матрице смежности также требует порядка $O(n^2)$ элементарных действий.

Векторы смежности образуют более экономичное матричное представление графа. Каждая строка матрицы, называемая вектором смежности соответствующей вершины, содержит номера смежных ей вершин. Количество столбцов такой матрицы для неориентированного графа определяется максимальной степенью вершины (рис. 1, в), а для ориентированного графа - максимальной полустепенью исхода (рис. 2, в).

Списочные представления. Во многих случаях лучшим представлением графов являются *списки смежности*, которые представляют собой структуру данных, включающую массив заголовков `Head` и собственно списки смежности, где элемент `Head[i]` содержит указатель на список смежности вершины v_i. Заметим, что каждый такой список обычно является связным и однонаправленным. Примеры организации списков смежности показаны на рис. 3 и 4.

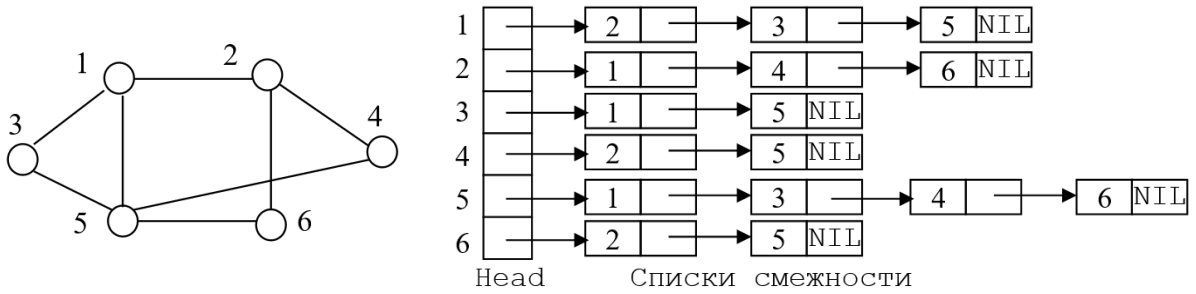


Рис. 3

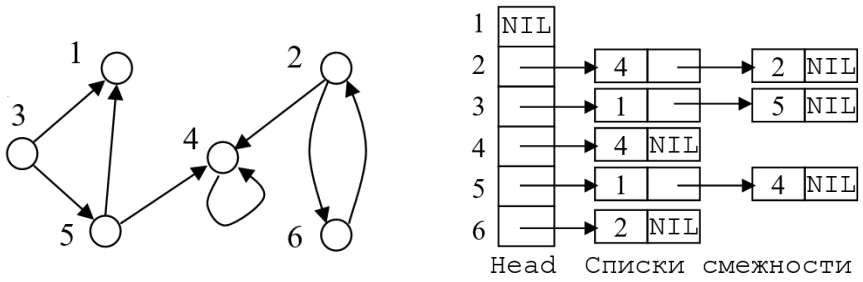


Рис. 4

Списки смежности достаточно легко позволяют модифицировать структуру графа путем удаления или добавления ребер. Время поиска определенного ребра пропорционально их количеству, поскольку они все могут входить в один список смежности некоторой вершины. Для представления списков смежности требуется $n+2m$ ячеек памяти для неориентированного графа и $n+m$ ячеек – для ориентированного.

Списки смежности на основе массивов. Если известно, что структура графа не будет изменяться в процессе решения задачи, то списки смежности можно представить двумя одномерными массивами `Node` и `Next` с числом элементов $n+2m$ и $n+m$ для неориентированных и ориентированных графов соответственно. Первый массив содержит индексы вершин графа, входящих в списки смежности, а второй массив показывает порядок их следования в списках, т.е. его элементы определяют индексы ячеек первого массива.

Чтобы определить вершины, смежные i-й вершине, требуется просматривать элементы этих массивов с использованием значений массива `Next`, начиная от `Next[i]`. Конец списка смежности определяется нулевым значением в массиве `Next`.

Например, рассмотренный ранее неориентированный граф описывается следующим образом.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Node							2	3	5	1	4	6	1	5	2	5	1	3	4	6	2	5
Next	7	10	13	15	17	21	8	9	0	11	12	0	14	0	16	0	18	19	20	0	22	0

Тогда, чтобы определить вершины, смежные второй вершине, выбирается значение `Next[2]=10`. Это означает, что список смежности второй вершины начинается с элемента `Node[10]=1`, следующий элемент списка находится в ячейке массива `Node` с индексом `Next[10]=11`, т.е. это вершина `Node[11]=4`, далее читается `Next[11]=12` и просматривается `Node[12]=6`. Значение `Next[12]=0` означает конец списка. В результате просмотра массивов получаем, что в список смежности второй вершины входят узлы с индексами 1, 4 и 6.

Ориентированный граф, показанный выше, описывается так:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Node							4	6	1	5	4	1	4	2
Next	0	7	9	11	12	14	9	0	10	0	0	13	0	0

Заметим, что первые *n* ячеек массива `Node` являются пустыми и могут использоваться для хранения некоторой числовой информации о вершинах графа. В частности, это могут быть метки вершин в задачах построения путей на графе.

1.2. Алгоритмы обхода вершин графа

Существует много алгоритмов, основанных на систематическом переборе вершин графа, при котором каждая вершина просматривается в точности один раз. Наиболее известными и широко используемыми для этого методами являются поиск в глубину и поиск в ширину.

Поиск в глубину (Depth-First Search) на графе $G = (V, E)$ осуществляется следующим образом. Из некоторой вершины $v_i \in V$ производится переход по ребру (дуге) к любой вершине v_j, смежной с v_i, т.е. $v_j \in \Gamma(v_i)$. Если v_j уже просматривалась ранее, то выполняется возврат к вершине v_i и выбирается следующее инцидентное ребро (дуга) и т.д. Если вершина v_j еще не просматривалась, то процедура *рекурсивно* применяется к этой вершине.

Процесс продолжается до окончания просмотра всех ребер (дуг), выходящих из вершины v_i. Далее происходит возврат к вершине $v_k \in V$, из которой ранее был выполнен переход к вершине v_i по ребру $[v_k, v_i] \in E$ или дуге $(v_k, v_i) \in E$. Конец процедуры поиска в глубину определяется условием, требующим выполнить возврат из вершины, с которой был начат просмотр вершин.

Рассмотрим формализованное описание процедуры `DFS` поиска в глубину. Пусть граф представлен списками смежности, где `L(i)` - список смежности *i*-й вершины. Массив `mark[1..n]` показывает состояние вершин графа: `mark[i]=true`, если *i*-я вершина еще не просмотрена и `mark[i]=false`, если уже просмотрена.

Сначала все вершины помечаются как непросмотренные, а затем производится поиск в глубину, начиная из непросмотренной вершины. При этом в зависимости от структуры графа и нумерации его вершин возможно несколько обращений в процедуре `DFS`.

```
for i:= 1 to n do
  mark[i] := true;
for i:= 1 to n do
  if mark[i] then DFS(i);
```

Процедура `DFS(i)` поиска в глубину начинает работу из *i*-й вершины и печатает номера вершин в порядке их обхода.

```
procedure DFS(i: integer);
var j: integer;
begin
  mark[i]:=false; write(i);
   for j∈L(i) do
      if mark[j] then DFS[j];
end;
```

В представленном описании процедуры `DFS` оператор цикла, используемый для выбора индексов вершин `j` из списка смежности `L(i)`, требует детализации с учетом конкретной реализации списков смежности.

Оценим вычислительную сложность процедуры поиска в глубину. Для каждой вершины, которая просматривается впервые, производится строго одно обращение к `DFS`. При каждом обращении число выполняемых действий пропорционально количеству ребер (дуг), инцидентных рассматриваемой вершине. Поэтому время поиска в глубину оценивается как $O(n+m)$ действий.

Пример 1. На рис. 5 показан неориентированный граф, списки смежности и порядок прохождения ребер при поиске в глубину.

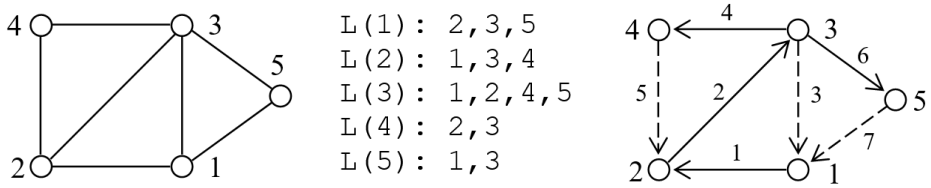


Рис. 5

Просмотр начинается с первой вершины, причем ребра, ведущие в уже просмотренные вершины, показаны штриховой линией. В результате работы алгоритма будет напечатано: 1, 2, 3, 4, 5.

Ребра (дуги) исходного графа, приводящие в новые, еще не просмотренные вершины, показываются сплошной линией и образуют *ориентированное связывающее дерево*. Такое дерево, получаемое при поиске в глубину, называется *DFS-деревом*.

DFS-дерево имеет несколько важных свойств. Если исходный граф является несвязным, то DFS-дерево представляется лесом. Заметим, что для связного ориентированного графа в зависимости от его структуры и нумерации вершин, DFS-дерево может быть как связным, так и несвязным (лесом). Для его построения процедура поиска в глубину может вызываться многократно, поскольку после первого обращения могут остаться непросмотренные вершины. При этом для связного неориентированного графа DFS-дерево всегда включает одну компоненту связности.

Если в DFS-дереве имеется путь из вершины v_i в вершину v_j, то v_i является предком для v_j, а v_j - потомком для v_i. Ребра (дуги) исходного графа, не принадлежащие DFS-дереву, называются *обратными ребрами* (*дугами*). Обратное ребро или обратная дуга всегда направлены от потомка к предку. Доказано, что если в орграфе есть цикл (контур), то обратная дуга обязательно встретится при обходе вершин этого графа методом поиска в глубину. Указанные свойства DFS-деревьев используются в прикладных алгоритмах на графах, некоторые из которых далее будут рассмотрены.

Поиск в ширину (Breadth-First Search) является еще одним методом систематического обхода вершин графа. Он получил свое название из-за того, что при достижении во время обхода любой вершины v_i далее просматриваются все вершины $v_j \in \Gamma(v_i)$, смежные вершине v_i. Далее вершина v_i исключается из рассмотрения, а процесс продолжается, начиная с любой вершины $v_j \in \Gamma(v_i)$.

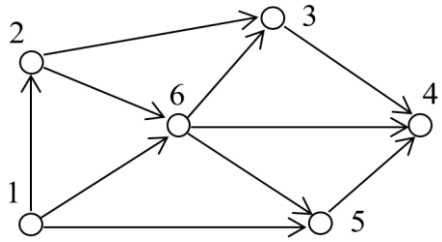
Поскольку во время обхода возврат к ранее просмотренным вершинам не производится (как при поиске в глубину) рекурсия не требуется, а порядок просмотра вершин определяется формируемой очередью. При поиске в глубину такой порядок сохраняется в стеке при рекурсивных вызовах процедуры `DFS`.

Пусть граф представлен списками смежности `L(i)`, формируемую очередь обозначим переменной `Q`, просмотренные вершины будем отмечать в массиве `mark[1..n]`. Тогда формализованное описание алгоритма можно представить в следующем виде, где `BFS(i)` - процедура поиска в ширину, которая начинает работу с *i*-й вершины.

```
for i:= 1 to n do mark[i]:=true;
for i:= 1 to n do
  if mark[i] then BFS(i);

procedure BFS(i: integer);
var
  Q: QUEUE;
  j,k: integer;
begin
  MAKENULL(Q); mark[i]:=false; BACK(i,Q);
  while not EMPTY(Q) do
    begin
      j:=FRONT(Q); write(i);
      for k∈L(j) do
        if mark[k] then
          begin mark[k]:=false; BACK(k,Q); end
    end;
end;
```

Пример 2. На рис. 6 приведен ориентированный граф, списки смежности его вершин и порядок выбора дуг при поиске в ширину. Работа процедуры BFS начинается с первой вершины. Дуги, показанные штриховой линией, приводят в ранее просмотренные вершины. При работе алгоритма будет напечатана следующая последовательность индексов вершин графа, соответствующая поступлению в формируемую очередь: 1, 2, 5, 6, 3, 4.




```
L(1): 2,5,6
L(2): 3,6
L(3): 4
L(4): нет
L(5): 4
L(6): 3,4,5
```

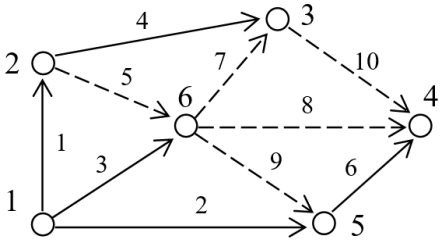


Рис. 6

Вызов процедуры `BFS(i)` приводит к просмотру всех вершин связной компоненты графа, содержащей вершину v_i, причем каждая вершина просматривается в точности один раз. Если граф имеет несколько компонент связности, то процедура `BFS` должна вызываться для вершин каждой компоненты связности. Вычислительная сложность поиска в ширину, также как и для поиска в глубину, составляет порядка $O(n+m)$ действий.

Поиск в ширину используется в различных алгоритмах оптимизации на графах, например, при построения кратчайших путей, для нахождения связных компонент графа и при решении других задач.

1.3. Прикладные алгоритмы на графах

Нахождение связных компонент. Простейшим случаем применения процедуры поиска в глубину является нахождение связных компонент неориентированного графа. Алгоритм заполняет массив `comp[1..n]`, где `comp[i]` - номер компоненты, которой принадлежит *i*-я вершина. Значение `comp[i]` определяется номером `num` компоненты связности порождаемого DFS-дерева.

```
for i:= 1 to n do
  begin mark[i]:=true; comp[i]:=0; end;
num:=0;
for i:= 1 to n do
  if mark[i] then
    begin
      num:=num+1; write('num=',num,':'); DFS1(i);
    end;
```

Модифицированная процедура `DFS1` поиска в глубину имеет следующий вид.

```
procedure DFS1(i: integer);
var j: integer;
begin
   mark[i]:=false; write(i); comp[i]:=num;
   for j ∈ L(i) do
      if mark[j] then DFS1[j];
end;
```

Пример 3. На рис. 7 показаны исходный граф, его списки смежности, полученное DFS-дерево и результаты решения задачи в виде выводимых данных. Конечное состояние массива `comp` имеет следующий вид:

`comp =(1,1,2,2,2,2,1).`

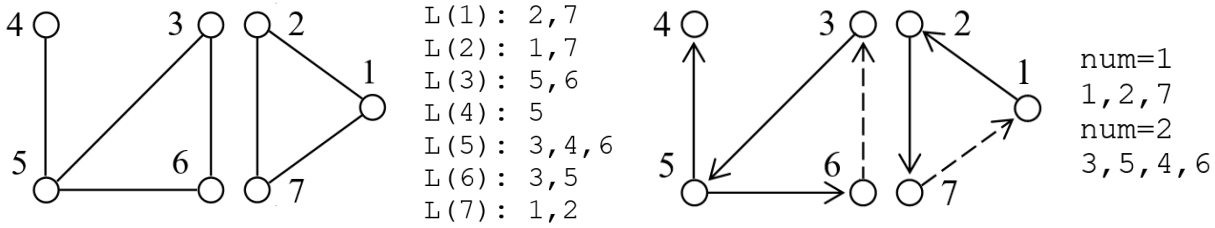


Рис. 7

Рассмотренную задачу также можно решить на основе модифицированной процедуры поиска в ширину `BFS1`.

```
for i:= 1 to n do
  begin mark[i]:= true; comp[i]:= 0; end;
num:= 0;
for i:= 1 to n do
  if mark[i] then
    begin
      num:= num+1; write('num=',num); BFS1(i);
    end;

procedure BFS1(i: integer);
var
  Q: QUEUE;
  j,k: integer;
begin
  MAKENULL(Q);
  mark[i]:= false; BACK(i,Q);
   while not EMPTY(Q) do
     begin
       j:= FRONT(Q); write(i); comp[i]:= num;
       for k∈L(j) do
```

```
        if mark[k] then
          begin mark[k]:= false; BACK(k,Q); end
    end;
```

Применение алгоритма поиска в ширину для нахождения компонент связности графа из примера 3 (рис. 7), дает такое же решение, но несколько изменяется порядок просмотра вершин второй компоненты связности (`num=2`): 3, 5, 6, 4.

Топологическая сортировка. Топологическая сортировка вершин ориентированного графа $G = (V, E)$ заключается в присваивании его вершинам номеров 1, 2, ..., n таким образом, чтобы для любой дуги $(v_i, v_j) \in E$ выполнялось условие $i < j$. Заметим, что такая нумерация вершин называется правильной.

Топологическая сортировка может рассматриваться как процесс отыскания на множестве вершин линейного порядка, в который может быть вложен частичный порядок, определяемый множеством дуг. Она возможна только тогда, когда ориентированный граф является бесконтурным.

Топологическая сортировка начинается с нахождения вершины, из которой не выходят дуги. Доказано, что такая вершина существует, если граф не имеет контуров. Этой вершине присваивается наибольший номер n и она удаляется из графа вместе с заходящими в нее дугами. Поскольку оставшийся граф также является бесконтурным, процесс повторяется и новой вершине, из которой не выходят дуги, присваивается наибольший номер n-1 и т.д.

Пример 4. На рис. 8 показан граф, вершины которого топологически не отсортированы. Если последовательно удалять (вместе с заходящими дугами) вершины с индексами 3, 2, 4, 6, 1, 5 и заменять их на 6, 5, 4, 3, 2, 1, то получим правильную нумерацию вершин этого графа (рис. 9). Заметим, что возможен и другой порядок удаления вершин - 3, 4, 2, 6, 1, 5 (с заменой их номеров на 6, 5, 4, 3, 2, 1).

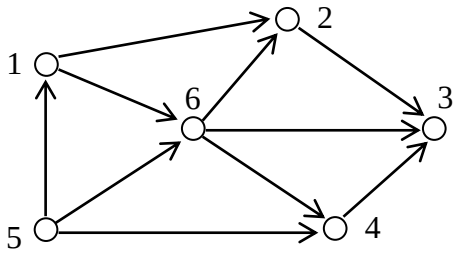


Рис. 8

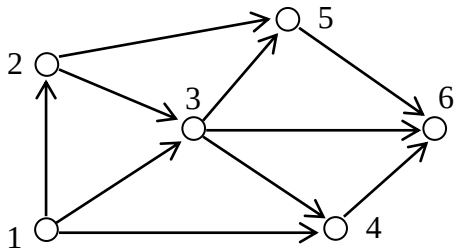


Рис. 9

Рассмотренный метод нетрудно реализовать на матрице смежности, однако при этом требуется порядка $O(n^2)$ операций.

Трудоемкость топологической сортировки можно уменьшить, если исключить повторяющиеся действия по нахождению вершин, из которых не выходят дуги. Это достигается применением поиска в глубину, который выполняется только один раз для заданного ациклического графа и обеспечивает выбор вершин, не имеющих выходящих дуг. Для сохранения новых индексов будем использовать массив меток вершин `label[1..n]`, с помощью которого также моделируется удаление вершин из графа.

Если *i*-я вершина еще присутствует в графе, то `label[i]=0`. В противном случае, т.е. после «удаления», получаем `label[i]>0`, где для любой дуги $(v_i, v_j) \in E$ выполняется условие `label[i]<label[j]`.

```
for i:= 1 to n do
  begin mark[i]:= true; label[i]:= 0; end;
num:= n+1;
for i:= 1 to n do
  if mark[i] then DFS2(i);

procedure DFS2(i: integer);
var j: integer;
begin
  mark[i]:= false;
  for j∈L(i) do
    if mark[j] then DFS2(j);
    num:= num-1; label[i]:= num; write(i);
end;
```

Пример 5. Рассмотрим граф из примера 4, показанный на рис. 8. Его списки смежности и порядок работы модифицированной процедуры `DFS2` поиска в глубину, начиная из первой вершины, даны на рис. 10, где шаги поиска указаны номерами дуг, в квадратных скобках около вершин показаны значения меток `label[i]`.

```
L(1): 2,6
L(2): 3
L(3): нет
L(4): 3
L(5): 1,4,6
L(6): 2,3,4
```

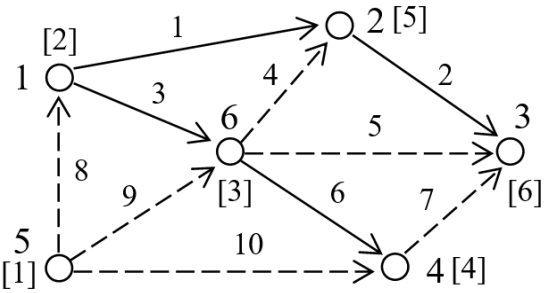


Рис.10

В результате будет напечатана последовательность индексов вершин 3,2,4,6,1,5 и получен массив label=(2,5,6,4,1,3).

Обнаружение контуров. Если ориентированный граф не является бесконтурным, то топологическая сортировка вершин становится невозможной, но рассмотренный алгоритм (с небольшой модификацией) можно использовать для обнаружения контура. Для этого используется факт нахождения обратной дуги, направленной от потомка к предку, при поиске в глубину. В частности:

- вершина v_i становится просмотренной, т.е. mark[i]=false, сразу же после вызова процедуры DFS2(i);

- метку label[i]>0 эта вершина получает только после выполнения поиска в глубину из всех смежных вершин $v_k \in \Gamma(v_i)$.

- если в процессе поиска в глубину из некоторой вершины $v_k \in \Gamma(v_i)$ найдена вершина v_j, уже просмотренная ранее (mark[j]=false), но не имеющая метки label[j]>0, то это означает, что найдена обратная дуга (v_k, v_j);

- дуга (v_k, v_j) направлена в вершину v_j, из которой ранее начат, но не завершен поиск в глубину, т.е. вершина v_k является потомком вершины v_j и дуга (v_k, v_j) входит в контур.

Вариант процедуры DFS2, обеспечивающий, если это возможно, топологическую сортировку вершин ориентированного графа, и обнаружение контура в противном случае, имеет следующий вид.

```
procedure DFS2(i: integer);
var j: integer;
begin
  mark[i]:= false;
  for j∈L(i) do
    if mark[j] then DFS2(j);
    else
      if label(j)= 0 then
        Error('Контур через вершины',i,'и',j);
  num:= num-1; label[i]:= num; write(i);
end;
```

Подпрограмма Error выполняет вывод сообщения об ошибке, т.е. об обнаружении контура в графе, и принудительно завершает работу алгоритма.

Пример 6. На рис. 11 показан ориентированный граф, который не является бесконтурным, его списки смежности, и результаты работы последнего варианта процедуры DFS2.

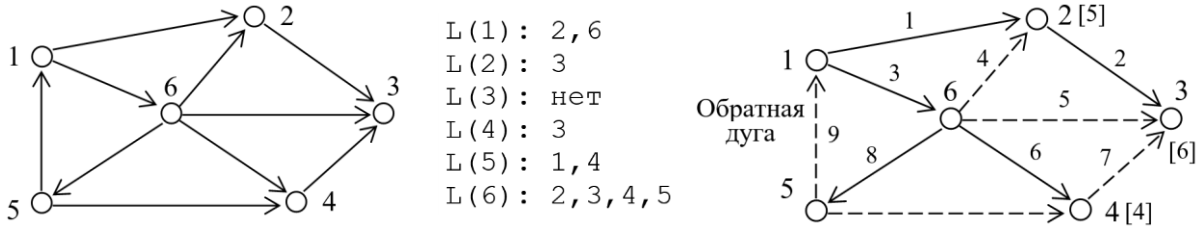


Рис. 11

Процесс поиска в глубину начинается из первой вершины. Обратная дуга (5, 1) будет обнаружена после вызова `DFS2(5)`, где 5-я вершина является потомком 1-й вершины в DFS-дереве, поскольку обращению `DFS2(5)` предшествуют вызовы `DFS2(6)`, ..., `DFS2(1)`. В результате топологическая сортировка не завершится, но будет выведено сообщение: `Контур через вершины 5 и 1.`

2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Реализовать на языке программирования алгоритмы поиска в глубину и поиска в ширину на графе. Для описания графа использовать матрицу смежности, векторы смежности, списки смежности на основе указателей или на основе массивов в соответствии с полученным заданием.

2.2. Решить контрольные примеры вручную, включая описание исходного графа заданным способом, и сравнить с результатами машинного решения.

2.3. Разработать алгоритм и программу нахождения связных компонент графа. Использовать согласованный способ описания графа и процедуру поиска в глубину или поиска в ширину в соответствии с полученным заданием.

2.4. Решить контрольный пример вручную и сравнить с машинным решением.

2.5. Разработать алгоритм и программу топологической сортировки вершин ориентированного бесконтурного графа с использованием процедуры поиска в глубину.

2.6. Решить контрольный пример вручную (допускается решение на основе матрицы смежности) и сравнить с машинным решением.

2.7. Модифицировать программу топологической сортировки вершин ориентированного графа таким образом, чтобы для произвольного графа при наличии контура (т.е. при невозможности сортировки) выполнялось его обнаружение с указанием вершин, через которые он проходит.

2.8. Решить контрольный пример вручную и сравнить с машинным решением.

3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы.
2. Тексты программных модулей, реализующих алгоритмы поиска в глубину и поиска в ширину для заданного варианта машинного представления графа.
3. Результаты решения контрольных примеров по реализации алгоритмов поиска в глубину и поиска в ширину, включая описание графа заданным способом.
4. Алгоритм и текст программы нахождения связных компонент неориентированного графа.
5. Решение контрольного примера (ручное и машинное) для задачи нахождения связных компонент неориентированного графа.
6. Алгоритм и текст программы топологической сортировки вершин ориентированного бесконтурного графа.
7. Решение контрольного примера (ручное и машинное) для задачи топологической сортировки вершин графа.
8. Модифицированный текст программы топологической сортировки вершин произвольного ориентированного графа.
9. Решение контрольного примера при наличии контура в ориентированном графе.
10. Выводы.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какое машинное представление графа является наиболее удобным для реализации поиска в глубину?
2. Перечислите и поясните основные свойства DFS-дерева.
3. Какое машинное представление графа является наиболее удобным для реализации поиска в ширину?
4. Для чего в алгоритме поиска в ширину используется очередь?
5. Поясните задачу нахождения связных компонент неориентированного графа.
6. Какие подходы возможны к решению задачи нахождения связных компонент неориентированного графа?
7. Почему задача нахождения связных компонент ориентированного графа является более сложной по сравнению со случаем неориентированного графа?
8. Предложите подход к решению задачи нахождения связных компонент ориентированного графа.
10. Почему задача топологической сортировки вершин имеет решение только для бесконтурного ориентированного графа?

11. Какие алгоритмы можно использовать для топологической сортировки вершин бесконтурного ориентированного графа?

12. Какие свойства DFS-дерева позволяют решить задачу обнаружения контуров в ориентированном графе?