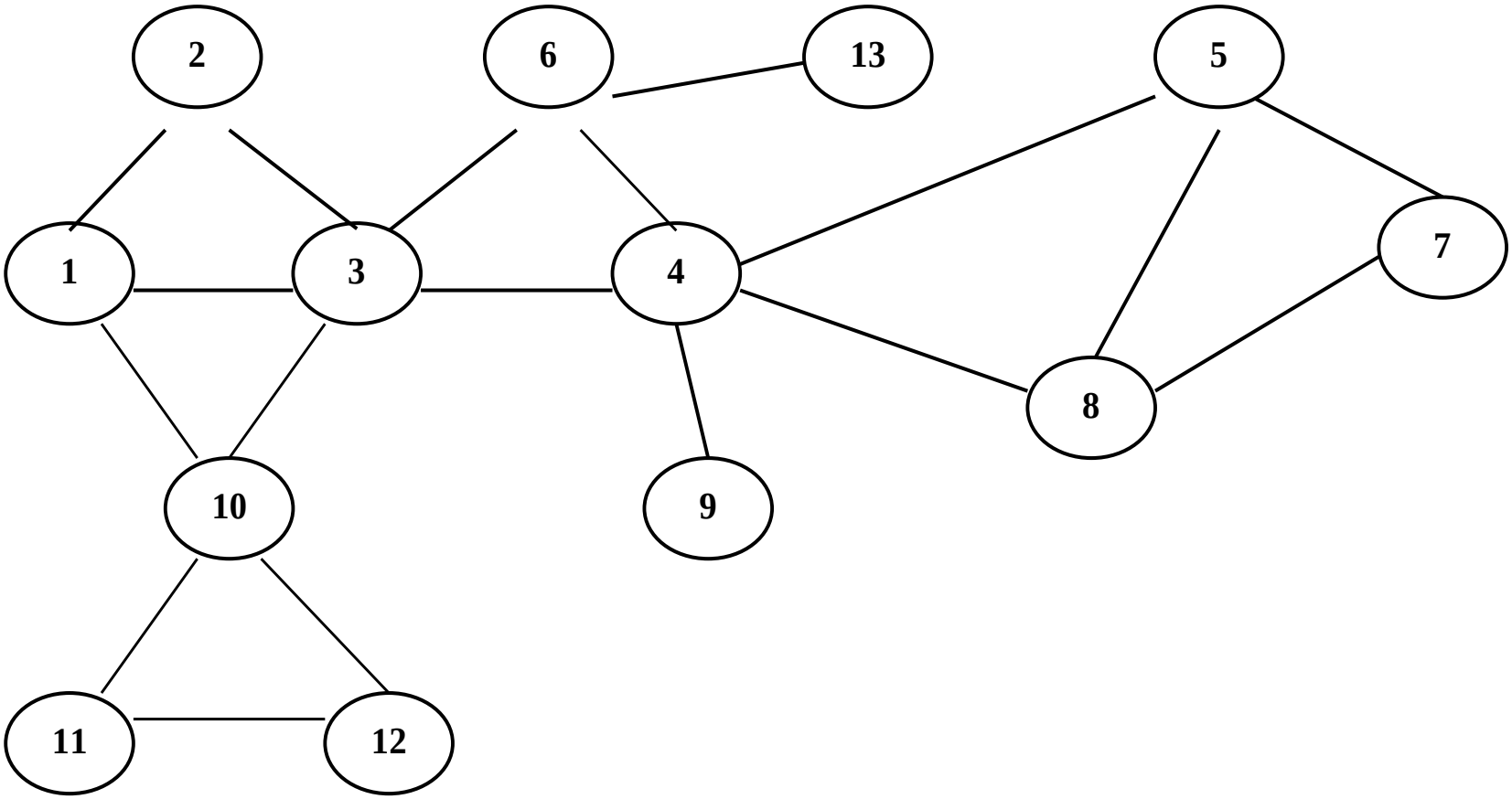
Поиски вершин в графах

- При поиске в графах, каждое ребро графа анализируется число раз, ограниченное константой (в пределе не более одного раза).
  - **ПОИСК В ГЛУБИНУ**
  - **(Depth First Search, DFS)**
- Поиск в глубину из вершины **v** основан на поиске в глубину из всех новых вершин, смежных с **v**.

поиск в глубину из вершины v

- **Procedure DFS_Graph(v);**
- // переменные **NEU**, **ZAP** - глобальные;
- //процедура **Look** - рассмотрение вершины.
- **Begin**
- **Look(v);** //рассмотрим вершину **v**
- **NEU[v] ←false;** //пометим ее как не новую
- **For u from ZAP[v] Do**
- //для каждой вершины, смежной с **v**
- **If NEU[u] Then DFS_Graph(u);**
- //если она новая, то идем в глубину от нее
- **End;** // вершина **v** использована

Пример графа

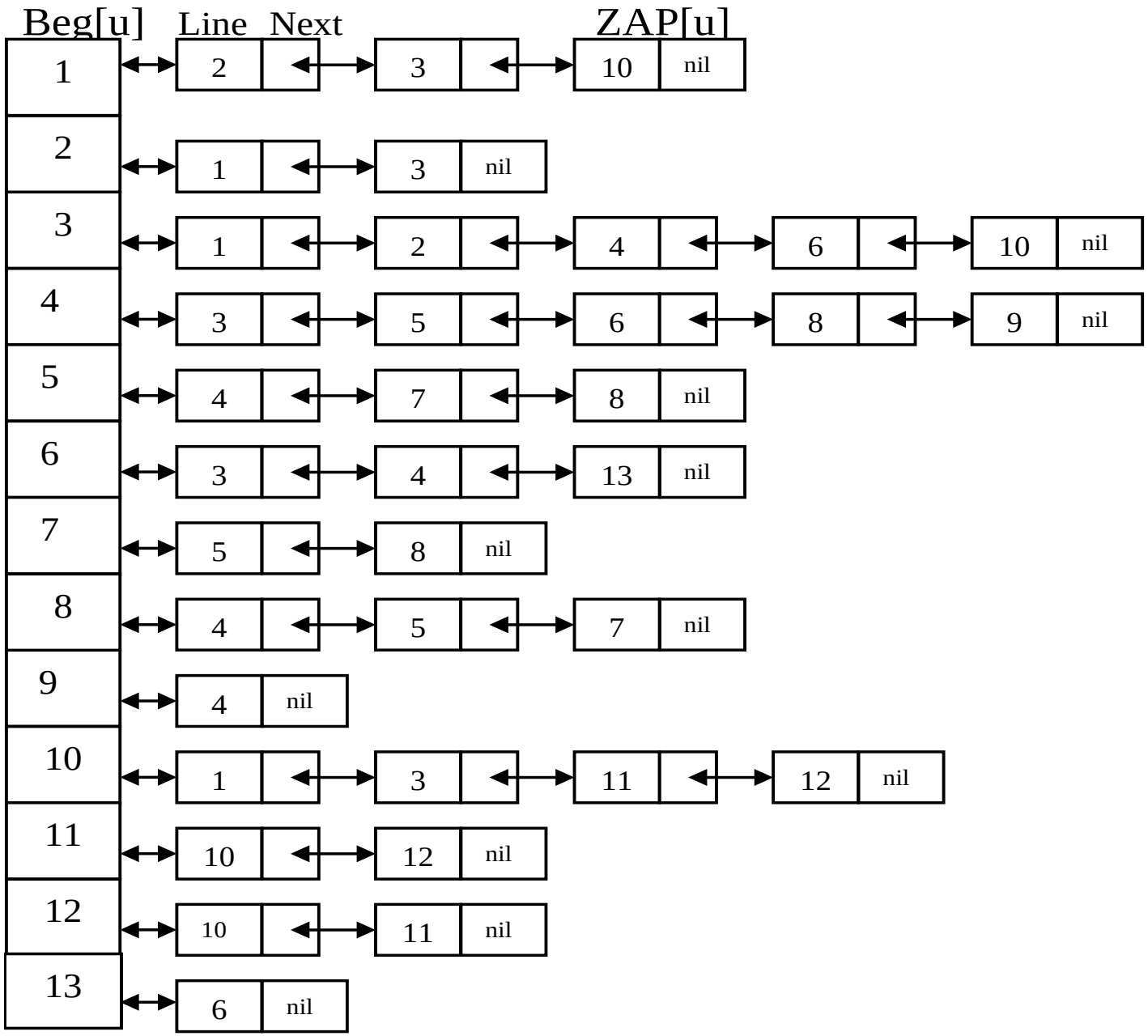


При поиске в глубину вершины данного графа просматриваются в следующем порядке:

1 – 2 – 3 – 4 – 5 – 7 – 8 – 6 – 13 – 9 – 10 – 11 – 12

отст.к 4 к 4 отст.к 3

Список инцидентности



- Поиск в глубину, необязательно в связном графе G=<V,E>, проводится по алгоритму:
-
- **Begin**
- **For v from V Do NEU[v] ←true;**
- **For v from V Do If NEU[v]**
- **Then DFS_Graph(v);**
- **End.**
    - **Сложность DFS – O(n+m) –**
        - т.е. - линейная

- Если каждую просмотренную вершину помещать в стек и удалять оттуда после использования, то процедура будет нерекурсивна:
- В начале поиска элемент глобального массива **Beg[u]** есть указатель на первую запись списка **ZAP[u]** для каждой вершины **u**.
- **Procedure DFS_Gr1(v);**
- // нерекурсивн. поиск в глуб. из вер-ны **v**
- **Begin** //массивы **Beg** и **Neu**- глобальные
- **stack ← 0;** // стек пустой – пустое множество
- **stack ← v;** //помещаем в стек **v**
- **Look(v);** //рассмотрим **v**
- **NEU[v] ←false;** //помечаем, что она не новая

- **While stack<>0 Do**
- **Begin**
- **t ← upper(stack);**
- //**t** - верхний элемент стека
- //поиск первой ***новой*** вершины в списке **ZAP[t]**
- **If Beg[t]=nil** //если список пустой
- **Then forw ← false**
- //то вершин больше нет
- **Else forw ←not NEU[Beg[t]^.LINE];**

//иначе ищем по строке новую вершину, для нее
//идем к ее списку

```
While forw Do
                                  //пока можно идти вперед
        Begin
           Beg[t] ← Beg[t]^.NEXT;
                                  //помещаем след. вершину
            If Beg[t]=nil
                                        //если она последняя
              Then forw ← false
                                    //то вперед идти нельзя
             Else forw← not NEU[ Beg[t]^.LINE ];
                                        //иначе - идем вперед
          End;
```

```
If Beg[t]<>nil    //найдена новая вершина в списке ZAP[t]
      Then
         Begin
             t←Beg[t]^.LINE;  stack ← t;
                    //помещаем ее в верхн.элемент стека
              Look(t);                    //рассматриваем ее
               NEU[t] ←false        //помечаем, как не новую
           End
       Else                          // вершина  t  использована
              t← stack
                 //удаление ее из верхнего элемента стека
     End          //While stack<>0
End;              //Procedure
```

- Путь, полученный поиском в глубину, не является, в общем случае, кратчайшим из **v** и **u**. Это общий недостаток поиска в глубину.

ПОИСК В ШИРИНУ (Breadth First Search, **BFS**).

- **Procedure BFS_GR(v);**
  - //поиск в ширину в графе с началом в вершине **v**
  - //переменные **NEU**, **ZAP**, **PREV** - глобальные
  - //процедура **use** - использование вершины.
- **Begin**
- **que ← 0; que ← v; NEU[v] ← false;**
- **While que<>0 Do**
- **Begin**
- **t ← que; use(t);** //посетить **t**
- **For u from ZAP[t] Do**
- **If NEU[u]**
- **Then**
- **Begin**
- **que ←u; NEU[u] ← false;**
- **PREV[u] ← t**
- //создания списка кратчайших путей из **v** в **u**
- **End**
- **End**
- **End;**
- **Сложность BFS : O(V+E) или O(n+m) – т.е. – линейная**

- Если добавить в процедуру **BFS_GR** вектор предыдущих вершин **PREV**, в нем для каждой просмотренной вершины **v** содержится вершина **PREV[u],** из которой и был переход в **v** - это кратчайший путь из **u** в **v**. Он определяется последовательностью вершин:
- **u=u(1), u(2), ..., u(k)=v,**
- где **u(k+1)=PREV[u(k)]** для 1 <= **k** <= i,
- **i** - первый индекс **k**, для которого **u(k)=v**.
- Последовательность появления в очереди и просмотра вершин при поиске в ширину
- 1 -> 2 -> 3 -> 10 -> 4 -> 6 -> 11 -> 12 -> 5 -> 8 -> 9 -> 13 -> 7
- из 1 из 3 из 10 из 4 из 6 из 5
- Содержимое вектора PREV (элемент PREV[1] не определен)

v	1	2	3	4	5	6	7	8	9	10	11	12	13
PREV[v]	1	1	3	4	3	5	4	4	1	10	10	6	

СТЯГИВАЮЩИЕ ДЕРЕВЬЯ (КАРКАСЫ) ГРАФА.

При использовании алгоритмов **DFS** и **BFS** мы получаем некоторые подграфы, которые имеют названия: каркасы, остовы или стягивающие деревья.

ДЕРЕВО - неориентированный связный граф без циклов. Для связного неорграфа **G=<V,E>** каркасом будет каждое дерево **<V,T>,** где **T** принадлежит **E**. Мы видим, что все вершины **V** исходного графа принадлежат каркасу.

Пусть **G = (V,E)** связный ациклический граф – не дерево, тогда следующие свойства равносильны (свойства деревьев):

- **G** является деревом (без выделенного корня);
- для любых 2-х вершин **G** существует единственный соединяющий их простой путь;
- граф **G** связан, но перестает быть связным, если удалить любое его ребро;
- граф **G** связан и **|E| = |V| -1**;
- граф **G** ациклический и **|E| = |V| -1;**
- граф **G** ациклический, но добавление любого ребра к нему порождает цикл.

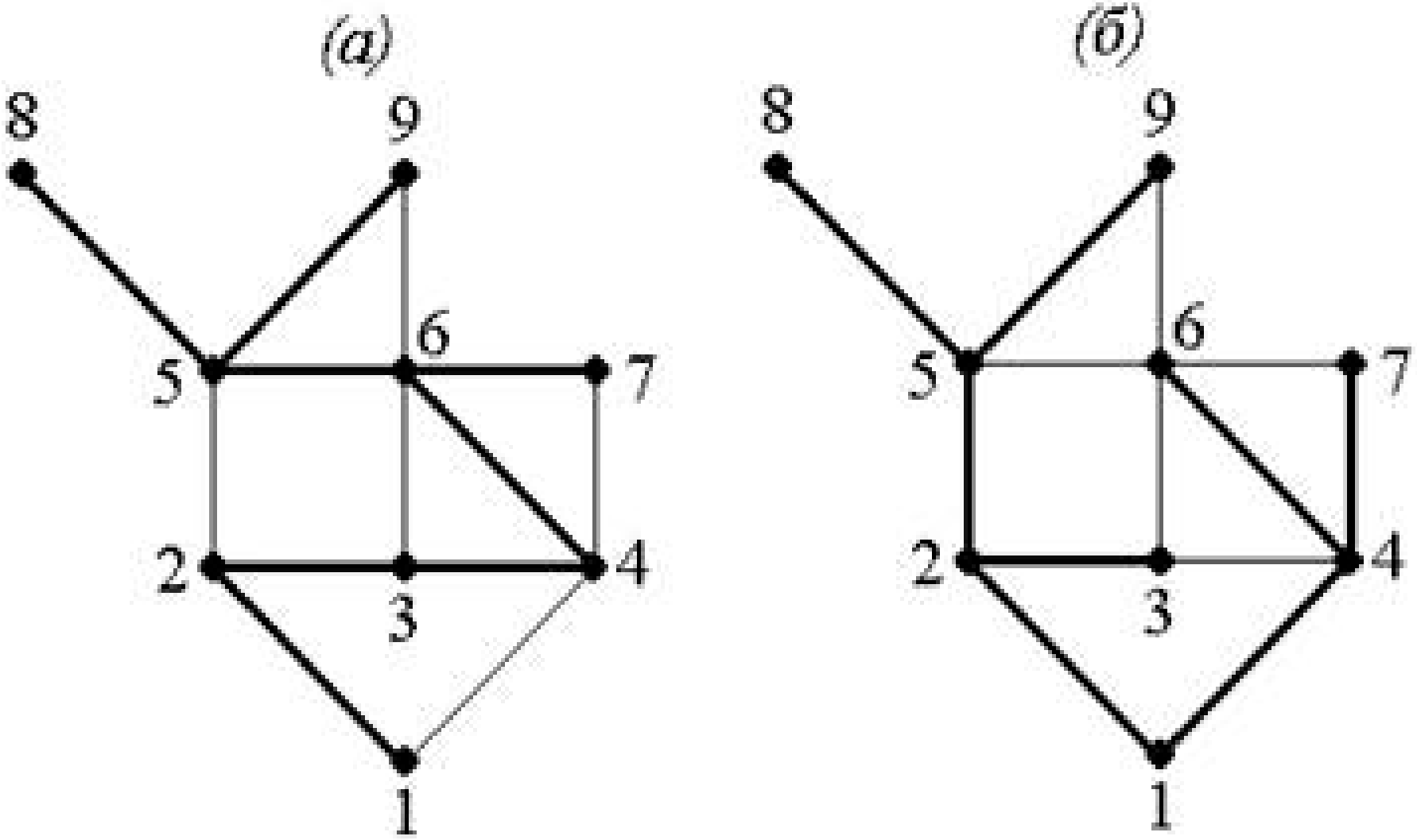
ПОИСК В ГЛУБИНУ

- **Procedure TREEDFS(v);**
  //поиск одного ребра из вершины **v**
- //переменные **NEU, ZAP,T** - глобальные
- **Begin**
- **NEU[v] ←false;** //помечаем вершину как не новую
- **For u from ZAP[v] Do**
- **If NEU[u] Then** //**{v,u}** - новая ветвь дерева
- **Begin**
- **T←T+{v,u};**
- **TREEDFS(u)**
- **End**
- **End;**
- **Begin** //основная программа
- **For u from V Do NEU[u] ← true;** //инициализация
- **T←0; TREEDFS(r)**
- **End.**

ПОИСК В ШИРИНУ

```
Procedure TREEBFS;
  Begin
    For u from V Do   NEU[u] ← true; T← 0;   //инициализация
    que← 0;  que  ←   r;  NEU[r] ← false;      //r –корень дерева
   While que<>0 Do
     Begin
       v ←  que;
       For u from ZAP[v] Do
         If NEU[u] Then                          // {v,u} - новая ветвь
           Begin
             que ←  u;    NEU[u] ←  false;     T ← T+{v,u}
           End
     End
 End.
```

- Алгоритмы сложностью O(n+m) строят стягивающее дерево графа **G**.
- Очевидно, что если стягивающее дерево построено при помощи алгоритма поиска в ширину, то путь в дереве от вершины **v** к корню **r** является кратчайшим путем из **V** в **R** и в исходном графе **G**.


(а)
8
9
6
5
7
2
4
3
1
(б)
8
9
6
5
7
2
4
3
1