# Лекция 2. ПОИСК В ГЛУБИНУ В ГРАФЕ

Существует много различных алгоритмов, в основе которых лежит систематический перебор вершин графа, при котором каждая вершина просматривается ровно один раз. Будем рассматривать такие методы обхода или поиска в графе, что:

а) алгоритм решения легко «погружается» в этот метод;

б) каждое ребро графа анализируется число раз, ограниченное константой.

Опишем классический метод поиска в неориентированном графе – метод **поиска в глубину** (***depth first search***) (рис. 2.1).

Общая идея следующая. Начинаем поиск с некоторой фиксированной вершины *k* (корня). Затем выбираем одну из смежных (соединенных ребрами) с ней вершин. Назовем эту вершину *u*. Далее процесс поиска повторяется от *u*.

Пусть на каком-то этапе мы находимся в вершине *v*. Если среди ее соседей есть хотя бы одна новая (еще непросмотренная) вершина *w*, то просматриваем ее. После этого *w* перестает быть новой, и дальнейший поиск продолжаем с *w*.

Если же у вершины *v* нет ни одной новой «соседки», то говорим, что *v* **использована**, и возвращаемся на шаг назад– в вершину, из которой попали в *v*. Просмотренные, но еще не использованные вершины накапливаются в стеке. Использованные вершины удаляются из стека. Когда стек опустеет, то у связного графа будут просмотрены все вершины, у несвязного – компонента связности, содержащая вершину *k*.

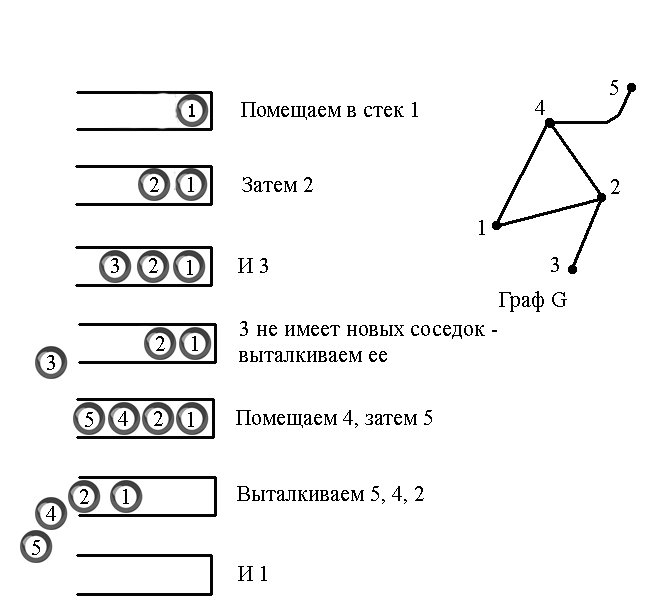


Рис. 2.1. Поиск в глубину из вершины 1

***Вопрос* 1**. Когда следует прервать работу алгоритма, если мы ищем путь от 1 до 5? Где находится путь?

**АЛГОРИТМ 2.1** {Поиск в глубину}

*Данные:* Неориентированный граф G=<V, E>, заданный списками инцидентности ЗАПИСЬ[v], v  V.

*Результаты:* Печать компонент связности графа.

*Глобальные переменные:* НОВЫЙ.

1 procedure DFS(v) {поиск в глубину из вершины v}

2 begin

3 НОВЫЙ[v]:= false; write(v,’ ’);

4 for u  ЗАПИСЬ[v] do {перебор всех соседей v}

5 if НОВЫЙ[u] then DFS[u]

6 end;

1 begin {основная программа}

2 for v  V do НОВЫЙ[v]:= true; {инициализация}

3 for v  V do {перебор всех вершин графа}

4 if НОВЫЙ[v] then

5 begin DFS(v); writeln end

6 end.

**Вычислительная сложность и корректность алгоритма**

Благодаря тому, что в основной программе поиск в глубину запускается из каждой новой вершины, будут просмотрены все компоненты связности графа.

Покажем теперь, что этот алгоритм просматривает каждую вершину в точности один раз и сложность его порядка O(*n + m*)*.*

Первый вызов DFS(*v*) для некоторой вершины *v* влечет за собой просмотр всех вершин компоненты связности графа, содержащей *v*. Это обеспечивает цикл 3 процедуры DFS: после просмотра *v* будет вызвана DFS для всех ее новых соседей.

Каждая вершина просматривается ровно один раз – просматриваются только новые вершины, сразу после просмотра НОВЫЙ[*v*] := false.

Гарантия того, что будут просмотрены все вершины – цикл 3 основного алгоритма.

Оценим теперь вычислительную сложность алгоритма.

Циклы 2 и 3 основного алгоритма содержат *n* шагов, не считая вызовы процедуры DFS. Строка 2 процедуры DFS для всех вызовов повлечет O(*n*) шагов. Полное число шагов цикла 3 процедуры DFS для всех рекурсивных вызовов будет порядка *m* – числа всех ребер, так как от вершины к ее соседям мы двигаемся по ребрам.

Суммарная сложность алгоритма О(*n + m*).

**Алгоритм поиска в глубину переносится на ориентированные графы.** Более подробно см. главу 6 «Поиск в глубину на ориентированном графе».

***Вопрос* 2**. Какие списки инцидентности следует использовать для поиска в глубину на ориентированном графе?

***Ответ* 1.** Когда 5 попадет в стек. В стеке.

***Ответ* 2.** Списки следующих вершин СЛЕД[v].