**Лекция 3. ПОИСК В ШИРИНУ В ГРАФЕ**

При поиске в глубинупросмотренные, но еще не использованные вершины скапливались в стеке. Чем позднее была просмотрена вершина, тем раньше ее удаляли из стека. Заменим стек очередью.

Механизм подключения элемента к началу очереди полностью совпадает с вталкиванием в стек. Механизм выталкивания из очереди – первым из очереди удаляется первый пришедший элемент.

Основная идея поиска в ширину(***breadth first search***)– у вершины *v* просматриваются и помещаются в очередь **сразу все** ее новые «соседи», после чего *v* становится использованной и удаляется из очереди (рис. 3.1).

D:\google\Учеба\Дисциплины\Алгоритмы на сетях и графах\Пособие на доработку\Картинки\3.1.tif

Рис. 3.1. Поиск в ширину из вершины 1

Поиск в ширину можно использовать для нахождения кратчайшего пути между парой фиксированных вершин *s* и *t*. Под **кратчайшим путем** мы понимаем путь с минимальным количеством ребер, так как пока все ребра графа имеют одинаковую длину (можно считать, что все ребра имеют единичную длину).

Поскольку у текущей вершины поиск находит сразу все новые соседние вершины, он, как волна от источника, равномерно распространяется во все стороны. Поэтому в очереди сначала находятся все вершины, удаленные от источника на расстояние 1, затем – все вершины, удаленные на расстояние 2 и т.д. Понятно, что путь, найденный таким образом, будет кратчайшим.

***Вопрос* 1**. Можно ли использовать поиск в ширину для разделения несвязного графа на компоненты связности?

Чтобы найти кратчайший путь, будем сохранять историю поиска в одномерном массиве ПРЕД. Для каждой вершины будем запоминать предыдущую вершину. Если в вершину *u2* попали из *u*1, то *u*1 была предыдущей для *u*2, поэтому ПРЕД[u2]:= u1. Первоначальная инициализация массива: ПРЕД[u]:= u.

С помощью массива ПРЕД можно будет восстановить кратчайший путь, двигаясь от конца к началу. Рассмотрим последовательность вершин, где u1 = ПРЕД[u2]. В *t* мы попадем из *uk*, в *uk* из *uk*-1 и т.д., пока не встретим *s*. Эта последовательность – путь из *s* в *t*.

У несвязного графа начало и конец пути могут находиться в разных компонентах связности, пути между ними не существует. Добавим логическую переменную Flag, чтобы запомнить, был ли достигнут конец пути.

Как только конец пути *t* попадёт в очередь, поиск следует прекратить. Поэтому поиск в ширину выполняется до тех пор, пока не найден конец пути и очередь не пуста.

**АЛГОРИТМ 3.1** {*Поиск кратчайшего пути из s в t*}

Данные: Неориентированный граф *G=<V, E>,* заданный списками инцидентности ЗАПИСЬ[*v*], *v*  *V*, две выделенные вершины источник *s* и сток *t*.

Результаты: Печать кратчайшего пути из *s* в *t*.

Глобальные переменные: НОВЫЙ, ПРЕД, Flag.

1 procedure BFS(v); {поиск в ширину из вершины v}

2 begin

3 ОЧЕРЕДЬ:= NIL; ОЧЕРЕДЬ <= v; НОВЫЙ[v]:= false; {v помещается

в очередь}

4 while (ОЧЕРЕДЬ<>NIL) AND NOT(Flag) do

5 begin

6 tail <= ОЧЕРЕДЬ;{выталкиваем нижний элемента}

7 for u  ЗАПИСЬ[tail] do {ищем всех новых соседей вершины tail}

8 if НОВЫЙ[u] then

9 begin

10 ОЧЕРЕДЬ <= u; НОВЫЙ[u]:= false;

11 ПРЕД[u]:=tail;

12 if u=t then Flag:= true;

13 end

14 end {while ОЧЕРЕДЬ <> NIL}

15 end;

1 begin {основная программа}

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

3 for v  V do ПРЕД[v]:= v; {инициализация}

4 Flag:= false; {конец пути не найден}

5 BFS(s);

6 if Flag then {путь существует}

7 begin

8 v:= t; {текущий конец пути}

9 while v<>s do

10 begin

11 write(v,’-’); v:= ПРЕД[v]

12 end;

13 writeln(s)

14 end

15 else writeln(’Пути нет’)

16 end.

**Вычислительная сложность алгоритма**

Как и для поиска в глубину, легко показать, что каждая вершина просматривается ровно один раз и вычислительная сложность равна O(*n* + *m*).

***Вопрос* 2**. Как будет заполнен массив ПРЕД при поиске в ширину на рис. 3.1 в момент помещения в очередь вершины 5?

**Алгоритм поиска в ширину переносится на ориентированные графы.**

***Ответ* 1.** Да.

***Ответ* 2.** ПРЕД[1] = 1; ПРЕД[2] = ПРЕД[3] = 1; ПРЕД[4] =3; ПРЕД[5] = 2.