Алгоритмы на графах. Мосты и точки сочленения

1. Введение.

Как уже упоминалось ранее, все ребра после обхода в глубину можно разделить на типы:

1) Ребро дерева, вошедшее в дерево обхода в глубину.

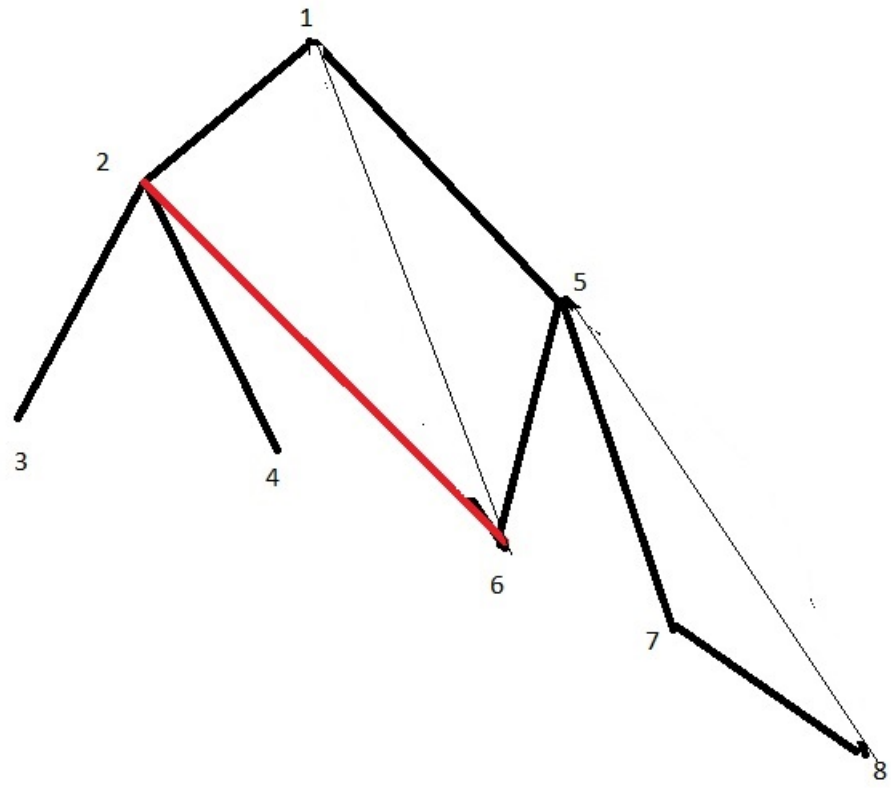
2) Обратное - то, которое смотрит в предка.

Также для ориентированных графов (ТОЛЬКО ДЛЯ НИХ):

3) Прямое - не вошедшее в дерево поиска, но смотрящее в предка

4) Перекрестное - все остальные ребра.

Еще раз: ПЕРЕКРЕСТНЫХ РЕБЕР В НЕОРИЕНТИРОВАННОМ ГРАФЕ НЕ БЫВАЕТ (см рисунок)



Поясненииние. Красное ребро - соединяет 2 разных поддерева поиска в глубину, т е перекрестное. КРАСНОГО РЕБРА В НЕОРИЕНТИРОВАННОМ ГРАФЕ БЫТЬ НЕ МОЖЕТ. Ребра (6;1) и (8;5) - обратные. Все остальные ребра на рисунке - прямые.

2. Метки времени. tin, tout, fup

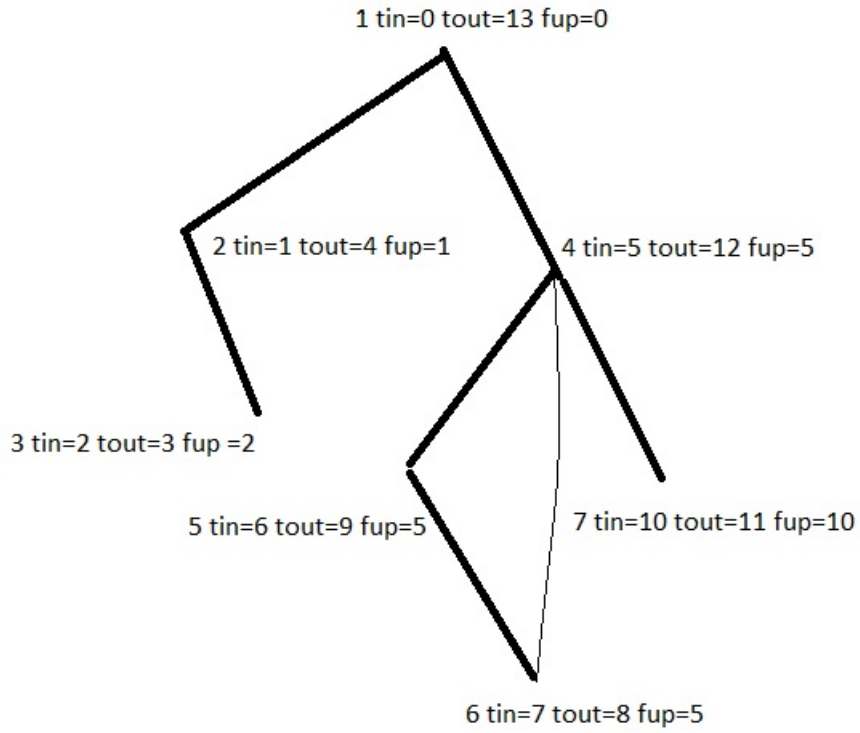
Рассмотрим стандартный поиск в глубину. Будем считать, что на каждую операцию входа в вершину требуется 1 секунда. Сохраним для каждой вершины время входа tin. Данная характеристика бывает часто полезна при решении задач.

Скажем также, что операция выхода из вершины в предка стоит 1 секунду (т е в тот момент, когда мы просмотрели всех детей вершины и выходим из нее). Сохраним для каждой вершины время выхода из нее tout.

Введем также понятие fup. Время fup[v] равно минимуму из времени захода в саму вершину tin[v], времён захода в каждую вершину p, являющуюся концом некоторого обратного ребра (v,p), а также из всех значений fup[to] для каждой вершины to, являющейся непосредственным сыном v в дереве поиска. По смыслу метка fup характеризует насколько высоко мы можем поднять вверх из данного поддерева.

Как считать fup? Пусть в DFS мы уже посчитали fup для всех детей данной вершины. Тогда мы можем посчитать fup для данной вершины как минимум из fup всех детей и tin вершины. Следует заметить что из некорневой вершины всегда есть одно особое ребро, не являющееся ни обратным ребром, ни ребром в детей - это ребро в родителя. Важно модифицировать DFS таким образом, чтобы он не пытался брать fup от родителя.

На изображении ниже приведен пример графа с посчитанными метками tin, tout и fup на нем.



Приведем код, который считает метки tin и tout и fup:

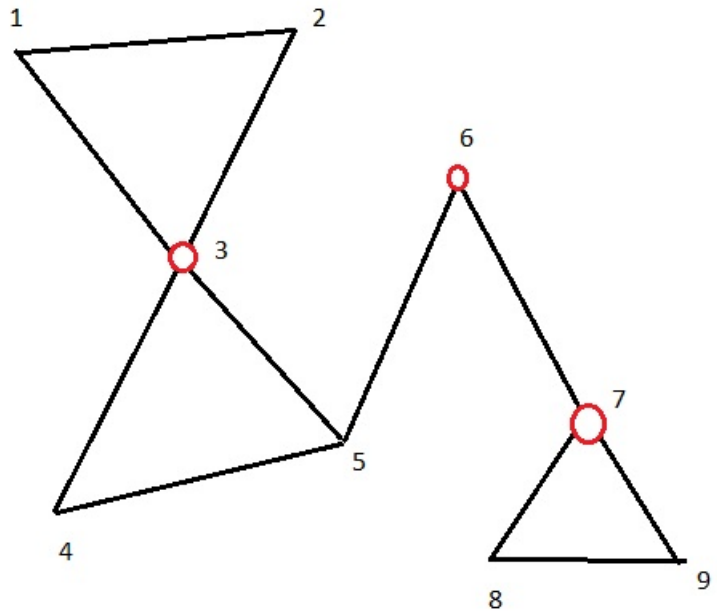
```
vector<int> tin;
vector<int> tout;
vector<int> fup;
vector<bool> used;
int t;
void DFS(int v, int p = -1)
{
    used[v] = 1;
    tin[v] = t;
    fup[v] = t;
    ++t;
    for (int i = 0; i < G[v].size(); ++i)
    {
        if (G[v][i] == p) //
            continue;
        if (used[G[v][i]] != 0)//
```

```
            fup[v] = min(fup[v], tin[G[v][i]];
        else
        {
            DFS(G[v][i], v);
            fup[v] = min(fup[v], fup[G[v][i]];
        }

    }
    tout[v] = t;
    ++t;
}
```

3. Мосты. Мостом в графе называется ребро, после удаления которого количество компонент связности увеличивается. Очевидно, что мост не входит ни в один цикл в графе.

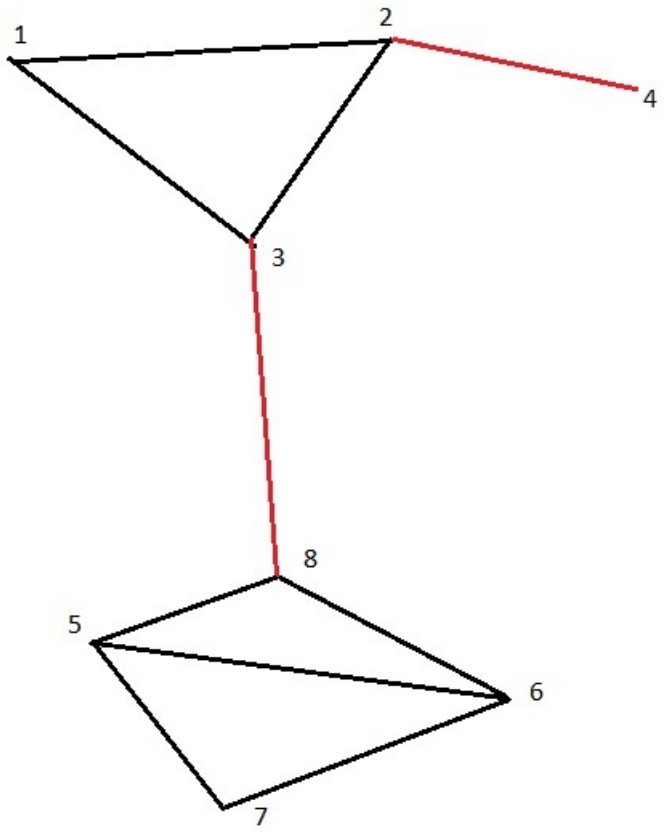
На рисунке ниже есть 2 моста - (3, 8) и (2, 4)



Несложно заметить, что ребро из u в v является мостом тогда и только тогда, когда во всем поддереве v нет обратного ребра, которое идет в u или выше. Это соответствует условию fup[v] == tin[v]

4. Точки сочленения. Точкой сочленения в графе называется вершина, после удаления которой количество компонент связности увеличивается.

На рисунке ниже есть 3 точки сочлеления - 3,6 и 7



При поиске точек сочленения надо учесть 2 случая:

1) Вершина является корнем дерева поиска в глубину. Тогда она является точкой сочленения если имеет 2 и более детей (не путать с количеством выходящих из нее ребер).

2) Все остальные вершины. Заметим, что если u явялется точкой сочленения, то она обязана разделить родителя p и одного из своих детей v на разные компоненты связность. Этому соответствует условие fup[v] >= tin[u].