Лекция 11 от 16.02.2016

Алгоритмы на графах

Графы бывают ориентированными и неориентированными, при этом неориентированные — частный случай ориентированных. Сегодня мы будем говорить исключительно о неориентированных.

Достижимость

Вход: $G(V, E); s, t \in V$. Вопрос: есть ли путь из s в t в G?

Небольшое отступление: ещё бывают взвешенные графы, где каждому ребру сопоставлен его вес (например, длина).

А можно поставить задачу поиска *кратчайшего пути*: Вход: $G(V, E); s, t \in V, W_i$. Выход: длина кратчайшего пути из s в t.

Но о взвешенных графах мы тоде говорит сегодня не будем.

Вообще говоря, многие задачи на графах не сразу бросаются в глаза как задачи, собственно, на графах — например, та же задача о расстоянии редактирования; если рассмотреть слова как вершины графа, а рёбра провести между теми вершинами, которые можно перевести одну в другую одой операцией; а хотим мы найти кратчайшее расстояние между dthibyjq s и t. Понятно, что такое решение не очень эффективно просто потому, что граф получается бесконечный, но тем не менее, как решение вполне годится.

Итак, задача достижимости. Что можно сделать в самом начале? Посмотреть на соседей s. Если среди них есть t, то мы выиграли; если нет, то посмотрим на соседей этих соседей и так далее. Запишем алгоритм, который обойдёт всевершины, достижимые из s и пометит их:

Пусть Q — множество тех, кого мы уже нашли, но чьих соседей ещё не проверили;

```
Explore(G, s)
    Q := {s}
    while Q != {} do
        extract u from Q
        mark u
        for v such that (u, v) in E do
            if v is not marked then
             add v to Q
```

Однако, алгоритм ещё не идеален; мы не уточнили, какую именно вершину мы извлекаем из Q; проверим, что будет, если Q работает как стек.

(тут неплохо бы как-то это описать)

Заметим, что такой алгоритм будет "углубляться" в граф на каждой итерации, двигаясь по неокоторому ациклическому пути пока ему есть кда идти; как только идти стало некуда, он начнёт "отступать", пока не окажется в вершине, из которой можно попасть в некоторую ещё не исследованную. То есть, попав в вершину f, мы не вернёмся назад, пока не обойдём всех соседей f. Такой алгоритм называется *поиск в глубину* (*depth-first search*).

Можно записать его рекурсивно:

```
DFS(G, S)
    mark s
    for v such that (s, v) in E do
        if v is not marked
            DSF(G, v)
```

А теперь попробуем сделать Q очередью. Пройдясь по графу вручную, становится понятно, что для улучшения алгоритма стоит помечать вершины перед их добавлением в очередь. Порядок обхода это не меняет, впрочем; наш алгоритм обходит алгоритм как бы "по слоям":

Пусть $L_0 = \{s\}$. определим слои рекуррентно: $L_{j+1} = \{v \mid \forall i \leqslant j : v \notin L_i, \exists u \in L_j : (u, v) \in S\}$. Заметим, что номер слоя, в который входит вершина, есть длина кратчайшего пути из s в неё, то есть *поиск в ширину* (*breadth-first search*) можно использовать для поиска кратчайшего пути. Перепишем его для этого:

Пусть d — список расстояний.

```
BFS(G, s)
    for v in V do
        d[v] := infinity
    d[s] := 0
    Q := [s]
    while Q is not empty do
         u := dequeue(Q)
         for v such that (u, v) in E do
         if d[v] = infinity then
            enqueue(v)
            d[v] = d[u]+1
```

<++>