Студент: Маргарита Чудова
Группа: 2
Дата: 27 апреля 2022 г.

---

1. Воспользуемся поиском в глубину — будем запускать его, пока не обойдем каждую вершину в A, и на каждом шаге будем проверять, не попали ли мы в какую-нибудь вершину из B. Если мы обошли все вершины в A, но ни разу не попали в вершину из B, то пути нет.
   Псевдокод:

```
def dfs(v, path):
    used[v] = True
    for u in edges[v]: #массив соседей v
        if not used[u]:
            if u in B:
                path = True #если нашли соседа в B, то путь есть
                print("Путь в наличии!")
                break
            dfs(u)

def main():
    path = False #есть ли искомый путь
    used = [False]*V #зашли ли в вершину v
    for v in A: #пробегаемся по всем вершинам A, запускаем dfs
        if not used[v]:
            dfs(v)
```

3. **Старая идея**: преобразовать dfs и запустить его из нового корня q.
   Допустим, изначально вершины были пронумерованы в соответствии с видом дерева в корне r — то есть номер родителя вершины меньше номера самой вершины. Запустим dfs из q, есть два варианта, куда мы оттуда пойдем: первый — поддерево, которое изначально лежало ниже вершины q, в нем ничего менять не надо, второй — та часть дерева, которая изначально находилась выше q. В этой части могут быть два варианта — либо кусок дерева не лежит на пути от q до r, тогда в нем ничего не меняется, либо лежит, и тогда родителями станут вершины с большим номером.

   Пусть parents — название массива, в котором лежат значения родителей для каждой вершины. Таким образом, в dfs, запущенном из q, будем на каждом шаге для вершины v и ее детей u проверять, меньше ли значение parents[u], чем v. Если меньше, это значит, что рассматриваемое ребро лежит на пути в старый корень, значит нужно поменять значение parents[u] на v. Если нет, то порядок сохраняется. Поскольку в дереве $E = V - 1$, время работы $\mathcal{O}(V)$.

   Update: на самом деле, dfs здесь излиший, в вершины, где ничего менять не надо, можно не заходить. Идея остается той же самой: начинаем с вершины q, смотрим на то, что для нее указано в массиве parents (пусть это вершина u) и переходим к этой вершине, таким образом будем двигаться от q до r. По пути от q до r порядок нарушен и на самом деле родитель u — текущая вершина. Используем $\mathcal{O}(1)$ памяти в процессе и проходим так не более V вершин, значит время $\mathcal{O}(V)$

   Легче написать псевдокод, чем адекватно описать это текстом:

```
cur = q
parents[q] = -1
while cur != r: #пока не прошли путь от q до корня r
    nxt = parents[cur] #смотрим на "родителя" текущей вершины
    parents[nxt] = cur #его родитель теперь - текущая вершина
    cur = nxt #идем дальше
```

4. На лекции мы разбирали алгоритм топологической сортировки с очередью — сначала мы клали все истоки в очередь, потом по очереди их доставали, удаляли, смотрели на их соседей и уменьшали степень соседей на 1. Таким образом, вершины обрабатывались в том порядке, в котором хранились в массиве соседей edges, а нам нужен лексикографический. Для этого можем завести кучу вместо очереди и в качестве следующей вершины в топологической сортировке брать минимум из кучи. Псевдокод:

```
h = Heap()
for v in range(|V|):
    if deg[v] == 0: #входящая степень вершины
        h.add(v) #все истоки положили в кучу
while not h.empty():
    v = h.extractMin() #берем минимальный лексикографический исток
    ans.append(v) #исток добавили в ответ
    for u in edges[v]:
        deg[u] -= 1 #исток удалили, степень соседей понизилась
        if deg[u] == 0:
            h.add(u) #добавили в кучу новые истоки
```

5. В дереве каждое ребро является мостом (то есть только по нему можно попасть из одной части дерева в другую) и из каждой вершины в любую другую существует только один путь. Рассмотрим некоторое ребро между вершинами u и v:

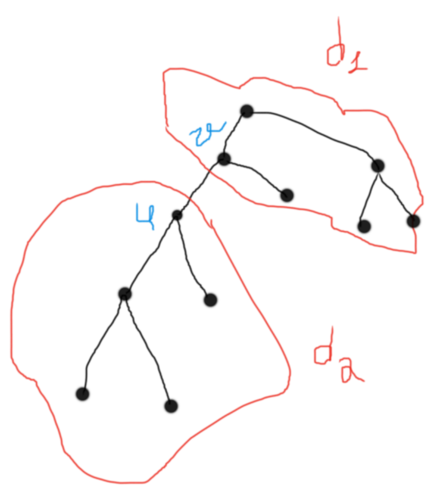

Рис. 1: Кривенький рисунок-пояснение

Оно делит граф на два поддерева: d_2 (дерево с корнем в u) и d_1 (дерево оставшихся вершин), пусть в них n_1 и n_2 вершин соответственно. Из каждой вершины d_1 можно добраться до каждой вершины d_2, причем одним способом, значит всего простых путей, проходящих через ребро (u, v), будет $n_1 n_2$. При этом несложно заметить, что $n_1 = V - n_2$.Таким образом, если мы насчитаем количество вершин для всевозможных деревьев типа d_2, мы сможем для каждого ребра определить количество простых путей $N = n_2(V - n_2)$.

Считать количество вершин для деревьев с корнем в некоторой вершине v будем с помощью динамики в DFS. Запустим DFS из корня, дойдем до листа z (то есть у вершины не будет непосещенных соседей) — тогда для z количество вершин в поддереве равно 1. Поднимемся на уровень вверх — для нелистовой вершины количество вершин будет суммой количества вершин для всех ее детей плюс 1. Таким образом сможем насчитать n_2 для всех вершин.

6. Можем представить это симметричное соотношение в виде графа. Пусть люди - это вершины, если они враги, то они соединены ребром, если друзья — не соединены.
   Пусть наша цель — раскрасить вершины на два цвета, синий и красный, в зависимости от того, в какой доле лежит вершина. Запустим dfs из любой вершины, причем этот dfs будет не просто отмечать вершины как посещенная/непосещенная, а красить их в синий/красный. Изначальным цветом пусть будет синий, а меняться цвет будет, когда смежных с текущей вершин, покрашенных в один цвет, будет 2.
   То есть на каком-то шаге мы как бы рассматриваем маленькое поддерево глубины 1, у которого не больше 3 вершин на втором уровне. Покрасили корень, спускаемся в вершины — первую красим в тот же цвет, вторую, если вернулись, уже в другой. Таким образом, поскольку вершин не более 3, сможем взять вместе с вершиной только одного врага, а других определить во вторую команду. Работает это все как и dfs за $\mathcal{O}(V + E)$