Алгоритмы. Домашнее задание №7

Горбунов Егор Алексеевич

25 октября 2015 г.

Задача №1 (Сильная ориентация графа)

Нужно ориентировать рёбра данного неориентированного графа $G(V, E)$ за $\mathcal{O}(|V| + |E|)$ так, чтобы получившийся граф был сильно связным.

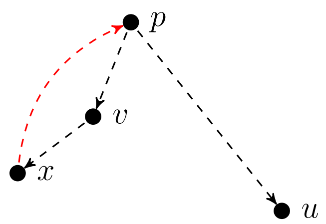
Из курса по дискретной математике нам известно, что неориентированный граф G может быть сильно ориентирован тогда и только тогда, когда он двусвязный, т.е. в G нет точек сочленения. Искать точки сочленения в графе G мы умеем за $\mathcal{O}(|V| + |E|)$ и первым делом запустим алгоритм поиска точек сочленения и в случае, если точки сочленения будут найдены, то сообщим о том, что данный граф G не допускает сильной ориентации.

Далее будем считать, что граф G не содержит точек сочленения, т.е. двусвязен, а значит допускает сильную ориентицаю рёбер. Построим такую ориентацию рёбер:

```
procedure DFS(v)
    isUsed[v] = true
    for u ∈ Γ(v) do
        orientEdgeFromTo(v, u)
        if isUsed[u] = false then
            DFS(u)
```

Тут $\Gamma(v)$ — множество вершин, смежных с v. Вызов «orientEdgeFromTo(v, u)» просто ориентирует ребро $\{v, u\}$ так: (v, u), т.е. $v \to u$. Таким образом мы просто ориентируем все рёбра в порядке обхода в глубину от предка к сыну, а если встречаем обратное ребро, которое может вести из вершины v к её предку, то ориентируем его от сына (v) к предку.

Этот алгоритм будет работать по следующим причинам: рассмотрим 2 любые вершины v и u графа G', который есть ориентированный граф G по процедуре, приведённой выше. Посмотрим на эти вершины в дереве, которое было построено поиском в глубину. Пускай p — это любой общий предок вершины v и u в этом дереве (он может совпадать с v или u). Заметим теперь, что т.к. G — двусвязный, то нам известно (из того же курса дискретной математики), что **любые 2 вершины G лежат на некотором цикле**, но это значит, что p и v лежат на некотором цикле. Ясно тогда, что найдётся такая вершина x, что в дереве, построенном обходом в глубину, v будет предком x (x может совпадать с v) и из x будет торчать обратное ребро в p: Но тогда мы видим, что из v можно пройти в u так: $v \to \ldots \to x \to p \to \ldots \to u$.



Таким образом мы показали, что для любых двух вершин v и u графа G', который есть ориентация графа G по приведённой процедуре, из v существует путь по ориентированным рёбрам в u. А это значит, что граф G' сильно связный. ■

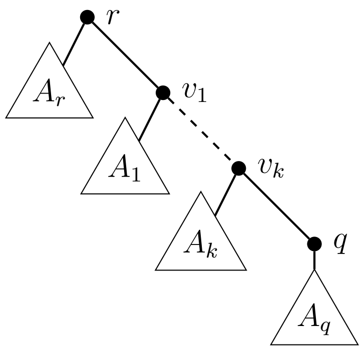
Задача №2 (Смена корня дерева за $\mathcal{O}(V)$)

Дерево T на V вершинах задано как массив $parent[1..V]$, где $parent[i]$ — индекс отца i-ой вершины, r — корень дерева T и $parent[r] = -1$. Нужно сменить корень дерева с r на q и перестроить массив $parent$ так, чтобы он задавал корневое дерево с корнем в q. Сложность: $\mathcal{O}(V)$

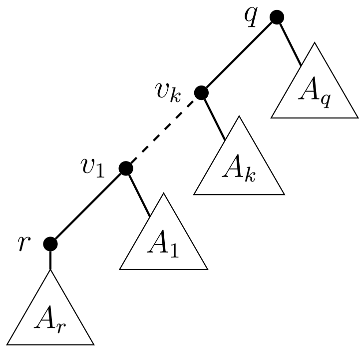
Будем обозначать корневое дерево T с корнем в r как T_r. Рассмотрим путь от вершины r до q в дереве T_r и этот же путь после перестроения в дереве T_q

Тут $A_1, \ldots, A_k, A_r, A_q$ — есть те подграфы дерева T_r, что не задействованы при спуске от r к q. Заметим, что в для любой вершины $v \in A_1 \cup \ldots \cup A_k \cup A_r \cup A_q$ значение $parent[v]$ для дерева T_r равно значению $parent[v]$ для дерева T_q, т.к. у такой вершины предок не изменился при перестроении. Предки изменились только лишь у вершин на пути от r до q: $r, v_1, \ldots, v_k, q$, причём так, что если v_i был предком v_{i+1} в T_r, то v_{i+1} стал предком v_i в T_q. Исходя из вышесказанного построим процедуру перестроения массива $parent$ из T_r в T_q:

procedure ChangeRoot($parent[1..V], r, q$)
    $prev \leftarrow -1$



(a) Исходное дерево T_r



(b) Перестроенное дерево: T_q

$cur \leftarrow q$
while $cur \neq -1$ **do**
  $next \leftarrow parent[cur]$
  $parent[cur] \leftarrow prev$
  $prev \leftarrow cur$
  $cur \leftarrow next$

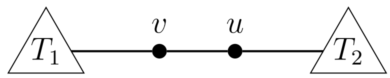
В силу того, что существует единственный путь из q в r в дереве, тело цикла отработает максимум V раз, а значит сложность алгоритма: $\mathcal{O}(V)$. ∎

Задача №3

Задача №4 (Число путей через ребро за $\mathcal{O}(V)$)

Дано дерево $T(V, E)$. Нужно для каждого ребра $e \in E$ вычислить, сколько простых путей через него проходит.

Ясно, что каждое ребро $e = \{v, u\} \in E$ есть мост (иначе, в T нашёлся бы цикл). А это значит, что при удалении e граф T распадётся на 2 компоненты связности C_1 и C_2. Пусть $v \in C_1$, а $u \in C_2$. Обозначи $T_1 = C_1 \setminus \{v\}$, $T_2 = C_2 \setminus \{u\}$. Тогда число путей, проходящий через $\{v, u\}$ равно числу путей, которые начинаются с v, проходят через u и уходят в T_2 плюс число путей, которые начинаются с u проходят через v и уходят в T_1 плюс всевозможные соединения ребром $\{v, u\}$ путей из v в T_1 и из u в T_2, ну и конечно нельзя забывать само ребро в качестве пути из 2 вершин.



Итого путей через $\{v, u\}$:

$$
\begin{aligned}
|T_1| + |T_2| + |T_1||T_2| + 1 &= \\
(|T_1| + 1)(|T_2 + 1|) &= \\
(|V| - |T_2| - 1)(|T_2| + 1)
\end{aligned}
$$

Тут воспользовались тем очевидным фактом, что $|V| = |T_1| + |T_2| + 2$. Теперь запустим поиск в глубину в дереве T из любой вершины, он подвесит нам дерево T за некоторую вершину. Рассмотрим вершину v в уже подвешенном дереве T и заметим, что поддерево T_v, с корнем в вершине v есть одна из компонент связности, на которую распадётся граф, если удалить ребро $\{v, parent(v)\}$, где $parent(v)$ — непосредственный родитель v в обходе в глубину. А значит, число путей, проходящих через ребро $\{v, parent(v)\}$, по вышеизложенным соображениям, равно: $size(T_v)(|V| - size(T_v))$, где $size(T_v)$ — число вершин в T_v включая саму v. Итого получим следующий алгоритм:

function DFS(v)
  isUsed[v] = $true$
  $size \leftarrow 0$
  for $e = (v, u) \in \Gamma(v)$ **do**
    if isUsed[u] = false **then**
      $childSize \leftarrow$ DFS(u, v)
      $pathCnt[e] \leftarrow childSize \cdot (|V| - childSize)$
      $size \leftarrow size + childSize$
  return $size$

Это поиск в глубину, он отработает за $\mathcal{O}(|V| + |E|) = \mathcal{O}(|V|)$, т.к. входной граф — дерево. ■

Задача №5