Алгоритмы и Структуры Данных. Лекция 8

24.04.2024

__scarleteagle imkochelorov

Дерево — ациклический связный (неориентированный) граф
Подвешенное дерево — дерево, одну вершину в котором назвали корнем
Лист — вершина дерева, не имеющая потомков

Предок вершины:
Вершина `u` является предком вершины `v`, если по пути от `v` до корня, будет достигнута вершина `u`

LCA

(Lowest Common Ancestor)

Задача: даны две вершины, необходимо найти их наименьшего *(самого низкого)* общего предка

Хранение дерева:

- Вершины пронумерованы от 1 до n
- У каждой вершины `v` хранится список её детей — `children[v]`
  А также непосредственный предок *(предок, связанный с вершиной ребром)* — `p[v]`

Наивный алгоритм

Построение пути от `u` и `v` до корня:

```
pu = []
pv = []
while u != None:
  pu.append(u)
  u = p[u]
while v != None:
  pv.append(v)
  v = p[v]
```

Вывод ответа:

```
for i in range(min(len(pu), len(pv)) + 1):
  if i == min(len(pu), len(pv)):
    print(pu[-i])
  elif pu[-i] != pv[-i]:
    print(pu[-i + 1])
    break
```

Краевые случаи:

`lca(v, v) = v`
`lca(v, u) = v` (если `v` — предок `u`)

Перед следующим алгоритмом ответим на вопрос:

Как проверять, что одна вершина — предок другой?

Обойдём дерево DFS'ом, сохраняя в каждой вершине временем входа и выхода из конкретной вершины:

```
timer = 0
def dfs(v):
  global timer
  timer += 1
  tin[v] = timer # время входа
  for child in v.children:
    dfs(child)
  timer += 1
  tout[v] = timer # время выхода
```

Теперь `u` — предок `v`, если время входа и выхода для `v` лежит между временами входа и выхода `u`:

```
def isParent(u, v):
  return tin[u] < tin[v] < tout[v] < tout[u]
```

Двоичные подъёмы

`t` — вершина на пути от `v` до корня
Правда ли, что `t` — предок `u`?

Это монотонный предикат: результат сначала всегда `False`, затем всегда `True`
Поэтому мы можем сделать по нему бинпоиск. А как?

`up[i][v]` — предок вершины `v` на расстоянии 2^i

Как считать `up`?
Воспользуемся динамикой:

$i = 0 \ldots \log_2 n$

```
p[root] = root
up[0][v] = p[v]
up[i][v] = up[i - 1][up[i - 1][v]]
```

Как теперь использовать бинпоиск?
Попытаемся найти ребёнка `lca(u, v)` (последний 0 предиката)

```
for i in range(log2(n), -1, -1):
  pv = up[i][v]
  if !isParent(pv, u):
    v = pv
return p[v]
```

Возможно, этот код отработает некорректно, когда одна вершина является предком другой.
Этот случай необходимо за`if`-ать отдельно

Теперь научимся считать `d[v]` — глубину вершины `v` в дереве:

```
d[root] = 0
d[v] = d[p[v]] + 1
```

Двоичные подъёмы 2.0

Посмотрим, кто глубже, `u` или `v`?

Пусть v глубже.

Поднимем `v` до глубины `u` бинпоиском.

```
if d[u] > d[v]:
  u, v = v, u
for i in range(log2(n), -1, -1):
  pv = up[i][v]
  if d[pv] >= d[u]:
    v = pv
```

Затем будем поднимать `u` и `v` одновременно

```
if u == v:
  return v
for i in range(log2(n), -1, -1):
  pv = up[i][v]
  pu = up[i][u]
  if pu != pv:
    u = pu
    v = pv
return p[v]
```

LCA → RMQ

Сведём задачу по нахождению LCA в задачу по нахождению RMQ

Применим Эйлеров обход (*Euler tour*):

```
def dfsOrdering(v):
  order.append(v)
  for child in v.children:
    dfsOrdering(child)
    order.append(v)
```

Построение `order` — $O(n)$, `len(order)` порядка `2n`
Запишем вместе с `order` глубину вершин
Тогда если `order[i] = u`, `order[j] = v`, то ответ — вершина с индексом `min(d[i:j])`

Проверка корректности:
1. Путь $u \rightsquigarrow v$ содержится на отрезке
2. `u` и `v` лежат в поддереве `lca(u, v)`, если мы увидели `p[lca(u, v)]`, то мы окончательно вышли из `lca(u, v)` до вхождения в `v`, противоречие

Алгоритм Фарах-Колтона и Бендера

Вернёмся к применению Эйлерова обхода. В прошлом алгоритме на этом моменте мы решили строить над массивом какую-то структуру данных. Однако можно сделать лучше:

1. Разделим `order` на блоки размера $B \approx \log_2 n$
2. В каждом блоке ищем `min`
3. Строим ST над массивом минимумов $\left(T = O\left(\frac{n}{B} \log \frac{n}{B}\right) = O\left(\frac{n}{\log(n)} \log \frac{n}{\log(n)}\right) = O(n)\right)$
4. ...

Заметим, что соседние глубины в массиве отличаются на 1

Вычтем из каждого блока первый элемент.
Блоки, ставшие одинаковыми назовём классами эквивалентности

Количество классов эквивалентности $\le 2^{B-1} = \frac{n}{2}$
Так как фактически каждый блок можно представить как последовательность $+1$ и -1

Переберём все классы эквивалентности:
для каждого префикса запишем минимум, для каждого суффикса запишем минимум

$O(2^B \cdot B) = O(n \log n)$ 😫
Пусть $B = \cancel{\log_2 n}\ \frac{\log_2 n}{2}$
$O(2^B \cdot B) = O(\sqrt{n} \cdot \log n) = O(n)$

Печалька: если никакой блок не помещается в запрос, то не работает
Храним для каждого класса эквивалентности, для каждого отрезка минимум

$O(2^B \cdot B^2) = O(\sqrt{n} \cdot \log^2 n) = O(n)$

Рассмотренные алгоритмы:

Алгоритм	Препроцессинг	Запрос
Наивный	$O(n)$	$O(n)$
Двоичные подъёмы	$O(n \cdot \log n)$	$O(\log n)$
Эйлеров обход + Дерево отрезков	$O(n)$	$O(\log n)$
Эйлеров обход + Sparse Table	$O(n \cdot \log n)$	$O(1)$
Фарах-Колтон, Бендер	$O(n)$	$O(1)$