ЗКШ, февраль 2016
конспект лекций

Собрано 28 апреля 2017 г. в 19:17

Содержание

1. Определение, построение

Def 1.1. *Центр дерева (centroid) – вершина, при удалении которой, размеры компонент будут не более $\frac{n}{2}$, где n – количество вершин в дереве.*

Def 1.2. *Центроидная декомпозиция (centroid decomosition) – результат рекурсивного процесса «выделить центр дерева, удадить, запуститься от компонент». Заметим, каждая вершина станет центром ровно один раз. Результатом являются ссылки для каждой вершины-центра на центр-отца, то есть, новое дерево.*

Каждому центру v можно сопоставить в соответствие компоненту $G(v)$, которую этот центр разбил на более мелкие. Таким образом определяются *компоненты центроидной декомпозиции*.

Lm 1.3. Глубина центроидной декомпозиции не более $\log_2 n$.

Доказательство. При переходе на уровень ниже размер компоненты уменьшается не меньше чем в два раза. ■

Lm 1.4. Суммарный размер всех компонент центроидной декомпозиции не более $n \log_2 n$.

Доказательство. Ветки рекурсии непересекаются по вершинами. Поэтому на каждом уровне рекурсии каждая вершина присутствует не более одного раза. А уровней из предыдущей леммы всего $\log_2 n$ ■

- **Алгоритм построения.**

```
int level[N]; // уровень рекурсии, изначально -1
int parent[N]; // отец-центр в декомпозиции
vector<int> graph[N]; // соседи для каждой вершины
void build( int v, int depth, int last ) {
  int center = -1, size = dfs1(v); // только по x : level[x] == -1
  dfs2(v, size, center); // центр выбирается жадно
  level[center] = depth;
  parent[center] = last;
  for (int x : graph[v])
    if (level[x] == -1)
      build(x, depth + 1, center);
};
int dfs2( int v, int size, int &center, int p = -1 ) { // выбор центра
  int sum = 1;
  for (int x : graph[v])
    if (level[x] == -1 && x != p)
      sum += dfs2(v, size, center, v);
  if (center == -1 && 2 * sum >= size)
    center = v;
  return sum;
}
```

Поскольку `dfs1` и `dfs2` фактически делают потчи одно и то же, можно написать так:

```
  int center, tmp;
  dfs2(v, dfs2(v, size, tmp), center);
```

Ещё заметим, что нам не обязательно знать точный размер, достаточно знать оценку сверху и при переходе в рекурсии уменьшать её в два раза, получаем чуть другой алгоритм.

- **Алгоритм построения 2: короче и быстрее.**

```
int level[N]; // уровень рекурсии, изначально -1
int parent[N]; // отец-центр в декомпозиции
vector<int> graph[N]; // соседи для каждой вершины
void build( int v, int size, int depth, int last ) { // size - оценка сверху
  int center = -1;
  dfs(v, size, center); // центр выбирается жадно
  level[center] = depth;
  parent[center] = last;
  for (int x : graph[center])
    if (level[x] == -1)
      build(x, size / 2, depth + 1, center);
};
int dfs( int v, int size, int &center, int p = -1 ) { // выбор центра
  int sum = 1;
  for (int x : graph[v])
    if (level[x] == -1 && x != p)
      sum += dfs(x, size, center, v);
  if (center == -1 && (2 * sum >= size || p == -1)) // новый случай: дошли до верха
    center = v;
  return sum;
}
build(0, n, 0, -1); // первый параметр - любая вершина
```

Lm 1.5. Для любого пути $[a, b]$ на дереве, есть такой центр c, что $c \in path[a, b]$ и $a, b \in G(c)$.

Более того, этот центр просто найти. Для этого нужно найти `LCA` в дереве центоидной декомпозиции. Самый простой алгоритм работает за глубину дерева, то есть, в нашем случае за $\mathcal{O}(\log n)$.

```
int getCenter( int a, int b ) {
  while (level[a] > level[b]) a = parent[a];
  while (level[a] < level[b]) b = parent[b];
  while (a != b) a = parent[a], b = parent[b];
  return a;
}
```

2. Применение: минимум на пути

• **Решим задачу:** *дано дерево, нужно сделать предподсчёт и в режиме online отвечать на запросы "минимум на пути дерева"*. Когда предлагается задача вида "посчитать функцию на пути дерева", то или у вершин, или у рёбер есть веса (соответстующие им числа). В рамках этого текста будем везде предполагать, что **вес есть у рёбер**. Изменение в хранении графа: `vector<Edge> graph[N]`. Для случая вершинных весов почти все рассуждения легко переделываются.

• **Решение.** Построим центроидную декомпозицию. В процессе построения, как только нашли `center`, запустим предподсчёт `calc(center)`, который для всех вершин компоненты центра найдёт минимум на пути от центра до вершины. Чтобы обработать запрос "минимум на пути $[a, b]$", возьмём `getCenter(a,b)`, а от него две предподсчитанных величины.

```
int f[K][N]; // K = round_up(log2(N))
void calc( int v, int depth, int p = -1, int minimum = INT_MAX ) {
  f[depth][v] = minimum; // depth - уровень центра компоненты, которую мы сейчас обходим
  for (Edge e : graph[v])
    if (level[e.v] == -1 && e.v != p)
      calc(e.v, depth, v, min(minimum, e.weight));
}
calc(center, depth); // вызов из build

int get( int a, int b ) {
  int c = getCenter(a, b); // O(LCA), самая долгая часть
  return min(f[level[c]][a], f[level[c]][b]); // O(1)
}
```

Можно ускорить функцию `getCenter`. Есть много стандартных алгоритмов для поиска `LCA`, вот ещё один: сохраним `path[v]` – путь от корня дерева центроидной декомпозиции до v. Первый элемент `path[v]` – корень. Длина каждого пути не боле $\log_2 n$. `getCenter(a, b)` – бинпоиск, который ищет первую пару различных элементов `path[a], path[b]`.

• **Итог.**

Научились искать минимум на пути с предподсчётом $\mathcal{O}(n \log n)$, временем $\mathcal{O}(\log \log n)$.

3. Задачи с решениями на тему

3.1. Путь с заданным XOR

Дано дерево и число S. Найти любую пару вершин a, b: XOR$(path[a, b]) = S$.

Рассмотрим центр c, проверим, есть ли путь в $G()$, проходящий через c, с нужным XOR. Пусть соседи c в $G(c)$ — $x_1, x_2, \ldots, x_k$. За d_i обозначим множество всех возможных XOR на путях от c до вершины в поддереве x_i.

```
D = ∅
for (i = 1; i <= k; i++) {
    for (y in d_i)
        if y ^ S is in D then нашли путь
    D ∪= d_i
}
```

D здесь удобно хранить, как `unordered_map<int,int>`, а d_i как `vector<int>`. Чтобы код быстро работал, нужно на всю программу иметь одну большую хеш-таблицу с заранее выделенной памятью: `unordered_map<int,int> D(N)`. Асимптотика времени обработки центра c равна $\mathcal{O}(|G(c)|)$, поэтому суммарное время работы $\mathcal{O}(n \log n)$.

• **Простое решение той же задачи за $\mathcal{O}(n)$.**

Заметим, что поскольку $\forall a$: $a \,\hat{}\, a = 0$, то XOR$[a, b]$ равен XOR$[a, root]$ ^ XOR$[b, root]$. Поэтому для решения задачи достаточно подвесить дереву за любую вершину и посчитать `dfs`-ом по дереву для каждой a XOR на пути от корня. По ходу `dfs` каждый раз, получая новое значение y, проверяем, есть ли среди старых значений $y \,\hat{}\, S$ (опять `unordered_map`).

3.2. Путь с заданной суммой

Предыдущее решение за $\mathcal{O}(n \log n)$ легко обобщить на любую обратимую ассоциативную функцию. Например, сумму. Заметим, что решение за $\mathcal{O}(n)$ подходило только для XOR.

3.3. Количество простых путей длины от L до R

Решение. Рассмотрим центр c, посчитаем в $G(c)$ число путей, проходящих через c длины от L до R. Пусть соседи c в $G(c)$ — $x_1, x_2, \ldots, x_k$. За $d_i[l]$ обозначим количесто путей длины l, идущих из c в поддерево x_i. Через значения $d_i[l]$ можно выразить ответ на задачу. Главное – не посчитать непростые пути (оба конца лежат в поддереве одного и того же x_i). Воспользуемся принципом включения/исключения. $D[l] = \sum_i d_i[l]$. Обозначем за $s(z)$ частичные суммы массива z, тогда нас интересует величина $\sum_x D[x] \cdot s(D).get(L{-}x, min(x, R{-}x))$ минус непростые пути. Количествов непростых путей: $\sum_i \sum_x d_i[x] \cdot s(d_i).get(L{-}x, min(x, R{-}x))$. Мы пишем не просто $R{-}x$, а $min(x, R{-}x)$, чтобы x был длиной меньшей половины пути, чтобы каждый путь посчитать ровно один раз. Чтобы один раз учесть пути, заканчивающиеся в c, сделаем $d_i[0] = 0$, а $D[0] = 1$. Обе суммы считаются за $\mathcal{O}(|G(c)|)$, поэтому суммарное время работы $\mathcal{O}(n \log n)$.

3.4. Запрос покраски

Задача: дано дерево, нужно в online отвечать на два запроса

1. `paint(v, d, c)` – покрасить все вершины на расстоянии от v не более d в цвет c
2. `get(v)` – сказать цвет вершины v

Решение. Сперва сведём запрос вида `paint(v, d, c)` к запросу `paint2(v, d, c)`, который будет красить от центра v только в пределах $G(v)$.

Продолжение следует...