SPb HSE, 1 курс, весна 2024/25
Практика по алгоритмам #24

Центроиды и приближения
9 апреля

Собрано 15 апреля 2025 г. в 17:38

Содержание

Центроиды и приближения

1. **Основы центроидов**

   Придумайте, как избавиться от функции `calc_size`. *Подсказка:* нам не нужен точный размер текущей компоненты, нам нужно чтобы по ходу рекурсии что-то убывало в 2 раза.

```
// Считаем размер текущей компоненты, которой принадлежит вершина v
int calc_size(int v, int parent = -1):
    int size = 1;
    for (int u: graph[v])
        if (u != parent && d[u] == -1)
            size += calc_size(u, v);
    return size;

// Ищем центроид, зная размер sz текущей компоненты. Возвращает пару (size, centroid):
// size - размер поддерева
// centroid - центроид всей компоненты (или −1, если ещё не найден)
pair<int, int> get_centroid(int v, int parent, int sz):
    int size = 1, centroid = -1;
    for (int u: graph[v])
        if (u != parent && d[u] == -1):
            auto [s, c] = get_centroid(u, v, sz);
            if (c != -1) centroid = c;
            size += s;
    if (centroid == -1 && size * 2 >= sz) centroid = v;
    return {size, centroid};

// Главная функция построения, v - любая вершина компоненты
void build(int v, int parentCentroid = -1, int depth = 0):
    int size = calc_size(v);
    int centroid = get_centroid(v, -1, size).second;
    d[centroid] = depth, p[centroid] = parentCentroid;
    calc_min(depth, centroid); // считаем минимумы на путях до centroid
    for (int u: graph[centroid]) if (d[u] == -1) build(u, centroid, depth + 1);
```

2. **RAM и ксс**

   Компоненты сильной связности за $\mathcal{O}(\frac{V^2}{w})$ в w-RAM и за $\mathcal{O}(V)$ в RAM. *Hint:* ускорьте dfs.

3. **RAM и параллелизм массивов**

   Сделайте предложенные операции с массивами за $\mathcal{O}(1)$.

   a) `a[] + b[]`
   b) `a[] == b[]` (посчитать, правда ли все равны)
   c) `a[] == b[]` (посчитать количество совпадений)

4. **RAM и философский камень**

   Решите 3-Vertex-Coloring за полином.

5. **Машина Тьюринга и +1**

   Пусть на ленте Машины Тьюринга написано $0, a_n, 0, a_{n-1}, 0, \ldots, a_0, 1$. Где a_i – цифры числа a в двоичной системе счисления. Младшая – a_0. Прибавьте к числу a единицу. Результат дайте в таком же виде. В конце вернитесь в ту же позицию, где начинали.

6. **bin-packing и Скрудж МакДак**

   Пусть даны $\frac{1}{3} \leqslant a_i \leqslant 1$. Решите задачу bin-packing за полином.

7. **Bin Packing: First fit**

   Задача Bin Packing: дано n предметов с размерами w_i и есть бесконечное количество коробок одинакового размера. Используя наименьшее количество коробок, нужно упаковать все предметы.

   Алгоритм First fit: перебираем предметы в *произвольном* порядке, кладём предмет в первую коробку, в которую он помещается. Если не помещается ни в одну из использованных, заводим новую коробку.

   Докажите, что это 2-приближение.

8. **Приближенное независимое множество**

   Выбрать в графе независимое множество размера $\lceil \frac{V}{D+1} \rceil$, где D – максимальная степень.

9. **Какмаркар-Карп и кардинальные улучшения**

   Оптимизация для Кармаркар-Карпа: остановимся при $n = 2 \log N$, где N – исходное число объектов. Что делать дальше? Как изменилась погрешность?

10. **Максимальный короткий путь в дереве**

    Дано дерево, каждая вершина имеет неотрицательный вес.
    Среди путей длины $\leqslant L$ найти путь max веса. $\mathcal{O}(n \log n)$.

11. **Максимальный путь в дереве**

    Дано дерево, каждая вершина имеет неотрицательный вес.

    a) За $\mathcal{O}(n \log n)$ среди всех путей дерева найти максимальный по весу.
    b) За $\mathcal{O}(\log n)$ отвечать на запрос «max по весу путь, проходящий через вершину v».

12. **Покраска близких соседей в дереве**

    Дано дерево, каждая вершина имеет цвет, изначально равный нулю. Запросы: «покрасить все вершины на расстоянии $\leqslant d$ от вершины v в цвет c», «вывести цвет вершины v».

    a) Во всех запросах покраски $v = 1$. $\mathcal{O}(\log n)$ на изменение, $\mathcal{O}(\log^2 n)$ на чтение.
    b) $\mathcal{O}(\log n)$ на изменение, $\mathcal{O}(\log^2 n)$ на чтение.
    c) Offline. Сперва идут все покраски, затем все запросы чтения. $\mathcal{O}((m+n) \log n)$.

13. **(*) Сумма сумм высот детей**

    Решим задачу «число путей в дереве длины ровно L» динамикой по дереву.

    $f[v, i]$ – число вершин в поддереве v на расстоянии i от v. При пересчёте динамики обычно используют естественную оптимизацию – приливать меньших (по высоте) детей к большему.

    Получается пересчёт в v за $\mathcal{O}(1) + \sum_{c \in g[v]} dep_c - \max_{c \in g[v]} dep_c$.
    Докажите, что полученное решение работает за $\mathcal{O}(n)$.

14. **(*) Машина Тьюринга и взрыв мозга**

    Напишите на Машине Тьюринга поиск битовой подстроки p в битовой строке s.
    Данные на ленте заданы следующим образом: $p_1 p_2 \ldots p_n \, 2 \, s_1 s_2 \ldots s_m \, 2$.

15. **(*) RAM и дары смерти.** Решите HAM-PATH за полином в модели RAM.

Разбор задач практики

1. **Основы центроидов**

   На самом деле, чтобы решать задачи, нам не нужна честная центроидная декомпозиция, нам нужно выбирать вершину `centroid` так, чтобы глубина «дерева декомпозиции» была $\leqslant \log_2 n$, для этого в функцию `get_centroid` будем передавать текущее ограничение на размер компоненты n, и будем следить, чтобы все результирующие были $\leqslant \frac{n}{2}$.

```
// size - оценка сверху на размер компоненты
void build(int v, int parentCentroid = -1, int depth = 0, int size = n):
    int centroid = get_centroid(v, -1, size).second;
    d[centroid] = depth, p[centroid] = parentCentroid;
    ...
    for (int u: graph[centroid])
        if (d[u] == -1)
            build(u, centroid, depth + 1, (size + 1) / 2);
```

2. **RAM и ксс**

   Давайте напишем dfs за $\mathcal{O}(n)$ операций с битовыми векторами.

```
void dfs(int v):
    used[v] = 1
    while ((t = FirstBit(~used & g[v])) != n)
        dfs(t)
```

   На w-RAM битовые вектора умею всё за $\mathcal{O}(\frac{n}{w})$, в RAM за $\mathcal{O}(1)$.

3. **RAM и параллелизм массивов**

   Запишем массив `a[]` длины n из чисел по w бит как одно $n(w+1)$-битное число A. Пусть число A уже есть в памяти.

   a) `a[] + b[]` : $A + B$
   b) `a[] == b[]` : $A \oplus B == 0$
   c) `a[] == b[]` : в числе $A \oplus B$ нужно найти количество кусков длины w, которые равны 0. Сперва не нули превратим в 1. Для одного `a[i]` битовыми операциями это делается так: `(a[i] & 1) | ((a[i]>>1) & 1) | ((a[i]>>2) & 1) ...`, тут $\mathcal{O}(w)$ битовых операций. Для чисел-массивов всё также, но 1 меняется на массив `[1,1,1]` = числу `000010000100001...` (в каждой группе w бит). Как построить число, соответствующее массиву `[x,x,x,x,....]`? По индукции увеличиваем длину массива в 2 раза битовыми операциями. Как теперь найти число единиц в массиве из нулей и единиц? За $\mathcal{O}(\log n)$: $A \to$ первая половина $A+$ вторая половина A.
   Пример: 001,001,000,001,000,000,001,001 $\to$ 001,001,001,002 $\to$ 002,003 $\to$ 005.

4. **RAM и философский камень**

   Как будем проверять конкретную раскраску y закодированную как $2 \cdot n$ бит?

```
int res = 1
for (auto [a, b, i] : edges)
    ca = (y >> (2*a)) & 3)
    cb = (y >> (2*b)) & 3)
```

```
    res &= (ca != cb)
for a=0..n-1:
    res &= (y >> (2*a)) != 0 // цвета только 1,2,3
return res
```

Теперь нужно все эти операции делать параллельно на массиве раскрасок `Y=[0,1,2,3,...]`. Как построить Y? По индукции за $\mathcal{O}(1)$ удваивать. Как делать описанные выше операции параллельно? Аккуратно использовать (a)(b)(c) из предыдущей задачи.

5. **Машина Тьюринга и +1**

   Стартовое состояние S. Чтобы сдвигаться вправо на 2 в поисках единицы нужно +2 состояния: A_0 – видели 0, нужно сделать ещё шажок и A_1, пришли в 1 Пришли в A_1, сместились влево $\Rightarrow$ перешли влево и попали в B. $S \to A_0 \to S \to A_0 \to \cdots \to S \to A_1 \to B$.

   Начинаем менять цифры, если под нами 0, то пишем 1, переходим $B \to F$ и это хорошо, если под нами 1, то пишем 0, переходим $B \to B_0 \to B$ (два шажка влево), и прибавляем 1 уже в следующем разряде. Когда мы в F, нужно вернуться на место. Для этого нужно идти до упора влево. Как понять, где упор? Изначально поставить там барьерную единицу: $S_0 \to A_0 \to S \to A_0 \to S \to \cdots \to A_1$, где S_0 такое же как S, но меняет под собой 0 на 1.

6. **bin-packing и Скрудж МакДак**

   Решение: $\frac{1}{3}+\frac{1}{3}+\frac{1}{3}$ в один, пока есть 3 по $\frac{1}{3}$ Далее в каждом рюкзак $\leqslant$2 предметов.
   Сортируем всех по возрастанию, далее 2 указателя: самому большому ищем самого жирного, кого можем к нему подкинуть и т.д.

7. **Bin Packing: First fit**

   Отмасштабируем всё так, чтобы все коробки стали единичного размера. Если наш ответ m, то $m-1$ корзина заполнена больше чем на половину. Итого $\sum a_i > \frac{1}{2}(m-1)$. С другой стороны $\mathrm{OPT} \geqslant \sum a_i \Rightarrow 2\mathrm{OPT} > m-1 \Rightarrow 2\mathrm{OPT} \geqslant m$.

8. **Приближенное независимое множество**

   Жадно берём. При каждом взятии из графа удаляется не более $D+1$ вершины.

9. **Какмаркар-Карп и кардинальные улучшения**

   Meet-in-the-middle! Мы умеем оптимально решить задачу за $2^{n/2}n$. Если бы делали как раньше, размер чисел уменьшился бы в $\frac{n}{2}\cdot\frac{n}{4}\cdot\frac{n}{8}\cdots$ раз. Если $n=2^k$, это $2^{k(k-1)/2}$. Мы же взяли из 2^n случайных множеств максимально близкое к 0 $\Rightarrow$ при предположении, что всё распределено равномерно, уменьшили ответ в $\approx 2^n = 2^{(2^k)}$. Вспоминаем, что $n = 2\log N \Rightarrow k = 1+\log\log N$, получаем было $\approx 2^{((\log\log N)^2)}$, а стало $N^2 \Rightarrow N^{-\log N} \to N^{-(2+\log N)}$.

10. **Максимальный короткий путь в дереве**

    Переберем центроид x, который будет на пути.
    За `dfs` по $C(x)$ считаем массив $w_x[d]$ – max вес пути до вершины на глубине d, $d \leqslant |C(x)|$.
    Снова `dfs` по $C(x)$. Вот `dfs` стоит в вершине v на глубине d с весом пути s. Обновляем ответ величиной $s + w_x[L-d]$.

    Но это может дать путь с самопересечением.
    Чтобы этого избежать, храним w_x не для всей $C(x)$, а для нескольких рассмотренных поддеревьев.

Обходя очередное поддерево x, только обновляем ответ.
Затем обходим поддерево второй раз, обновляя w_x.

Теперь хотим пути длины $\leqslant L$. Нужно поддерживать префиксные максимумы w_x.
Способ 1. Сортируем поддеревья x по возрастанию их размеров $C_1, C_2, \ldots$ Обходим их в таком порядке.
Тоже два обхода каждой C_i. На первом релаксруем ответ, на втором обновляем w_x.
После второго обхода надо обновить префиксные максимумы. Длина массива w_x сейчас станет $\leqslant |C_i|$, т.е. сделали $\mathcal{O}(|C_i|)$ операций.
Итого $\mathcal{O}(|C(x)|)$, в сумме $\mathcal{O}(n \log n)$.
Способ 2. Храним по два максимума в $w_x[d]$.
Второй max среди путей, ведущих не в то поддерево, где первый max.
Обходим всю $C(x)$ и считаем w_x.
Второй раз обходим всю $C(x)$ и обновляем ответ.
Если max путь пересекается в путем в текущую v, надо брать второй максимум.
Чтобы определить, есть ли пересечение, запомним в $w_x[d]$ не только вес пути, но и соседа x, с которого начался этот путь. То же самое помним и во втором `dfs`.

11. **Максимальный путь в дереве**

a) Строим centroid decomposition. Каждый путь проходит через какой-то центроид, в компоненте которого лежат оба конца пути.
Для каждого центроида ищем dfs-ом максимальные пути, проходящие через его детей (в исходном дереве) в его поддерево (в декомпозиции). Выбираем два максимума. Берём максимум по всем центроидам.
P.S. На самом деле это надо решать динамикой за $\mathcal{O}(n)$.

b) Считаем $\texttt{pathsum}[v, d]$. Найдем для каждого центроида x по два максимальных пути, идущих в $C(x)$ (через разных детей).
Сначала научимся искать макисмальный путь, для которого v – конец.
На любом пути есть самый высокий центроид $x\colon v \in C(x)$.
Перебираем всех x – предков v в декомпозиции. Если максимум из x идет в то же поддерево, где v, то смотрим на второй максимум.
Как определять, идет ли максимум в то же поддерево, где v? Можно запомнить ребенка (в исходном дереве), через которого прошел max путь из центроида.
Если max путь из x идет в соседа y, а $\texttt{up}[v, d_x - 1] = \texttt{up}[y, d_x - 1]$, то нужен второй максимум.
Теперь ищем пути, где v – не обязательно конец. Для этого ищем два максимальный пути, для которых v – конец, и объединяем.
Нужно следить, чтобы эти пути не пересекались. Для этого предподсчитаем величину $\texttt{next}[v, d]$ – первую после v вершину на пути из v в ее предка на уровне d.
Для каждого x – предка v, запомним вес найденного пути и $\texttt{next}[v, d_x]$.
В конце выберем из $\log n$ запомненных величин максимум и максимум среди тех, что проходят через другого соседа v.

12. **Покраска близких соседей в дереве**

c) Сначала решим offline-версию. Любой путь $v \rightsquigarrow u$ длины d проходит через некоторую $x\colon v, u \in C(x)$.
Лениво сохраним обновления в центроидах, в конце применим их все.

При i-м запросе $\langle v, d, c\rangle$ поднимаемся по x – предкам v и добавляем в список обновлений x запись $\langle d - \texttt{dist}[v, d_x], c, i\rangle$.
В конце для каждого центроида сортируем по времени все скопленные в нем обновления.
Полезно удалить лишние – те, которые перекрываются другими и по расстоянию, и по времени.
Если есть обновления $\langle d_1, c_1, t_1\rangle, \langle d_2, c_2, t_2\rangle : d_1 \leqslant d_2, t_1 \leqslant t_2$, то первое можно выкинуть.
Сложим обновления в стек, возрастающий по времени. При обработке $\langle d, c, t\rangle$ снимаем со стека все запросы с расстоянием $\leqslant d$.
Тогда стек будет возрастать по времени и убывать по расстоянию.
Теперь можно из каждого центроида x протолкнуть все изменения.
Обойдем $C(x)$ `bfs`-ом, поддерживая указатель в стеке обновлений x на текущее расстояние.
Крася вершину, запомним время покраски. Если из другого центроида придет покраска с меньшим временем, не применим ее.

b) Будем поддерживать стек, как в прошлом пункте, но сразу в процессе. Время $\mathcal{O}(\log n)$. Время, потраченное на уменьшение стека, самортизируется.
Чтобы узнать цвет вершины v, пройдемся по ее предкам-центроидам x. В стеке x бинпоиском найдем самую позднюю запись с расстоянием $\leqslant \texttt{dist}[v, d_x]$.
Среди выбранных по предкам записей выберем самую позднюю по времени.

13. **(*) Сумма сумм высот детей**

Нарисуем из каждого не листа сплошное ребро в самого глубокого ребёнка. Остальные рёбра тонкие. Сплошные рёбра покрывают все вершины дерева путями, пути не пересекаются.

Докажем, что время работы алгоритма – ровно суммарная длина этих путей, то есть, n. Высота вершины x равна длине пути, идущего из x вниз. Обрабатывая вершину x, мы учтём ровно все пути, начинающиеся в детях $v \Rightarrow$ каждый путь учтём ровно 1 раз.

Заметим, чтобы на практике прикинуть время, понять, что это $\mathcal{O}(n)$, достаточно посчитать время работы на трёх тестах: на «бамбуке», на полном бинарном дереве, на «солнышке». На бамбуке 0, на солнышке $n-2$, на полном бинарном дереве $\mathcal{O}(n)$.

P.S. Такое же утверждение не про глубины, а про размеры не верно. На полном бинарном дереве сумма размеров $\Theta(n \log n)$.

14. **(*) Машина Тьюринга и взрыв мозга**

?

15. **(*) RAM и дары смерти**

?

Домашнее задание

3.1. Обязательная часть

1. **(2) Число путей длины от L до R**

   Дано невзвешенное дерево. Посчитать количество простых путей в дереве длины от L до R.

2. **(2) Ближайшая чёрная вершина**

   Дано невзвешенное дерево. Изначально все вершины белого цвета.
   Научиться отвечать на запросы: покрасить белую вершину в чёрный; к заданной вершине найти ближайшую чёрную вершину. $\mathcal{O}(n \log n)$ на предподсчёт и $\mathcal{O}(\log n)$ на запрос в online.

   (+1) добавим запрос «покрасить чёрную в белый», $\mathcal{O}(\log^2 n)$ на запрос.

3. **(2) Bin Packing: Next Fit**

   Алгоритм Next Fit: перебираем предметы в *произвольном* порядке. Если предмет помещается в текущую коробку, кладём в неё, иначе закрываем эту коробку и заводим новую.

   Докажите, что это 2-приближение.

4. **(1) Был центроид. Сколько?**

   Сколько вершин дерева могут являться центроидами?

3.2. Дополнительная часть

1. **(3) Быстрый centroid decomposition**

   Дано дерево. Построить decomposition глубины не более $\log n$ за $o(n \log n)$.