День 01. Поиск в глубину

Теория

1. Методы хранения графов
    1. Матрица смежности.
    2. Списки смежности.
    3. Особенности хранения взвешенных графов, мультиграфов.
2. DFS
    1. Описание DFS. Сложность с матрицей смежности и списками смежности. Задача: проверка неориентированного графа на связность.
    2. Термины: дерево обхода, цвета вершин (белые/серые/черные), типы ребер (ребра дерева/прямые/обратные/перекрестные), времена входа/выхода. Дерево обхода неориентированного графа (отсутствие перекрестных ребер). Задача: поиск циклов в ориентированном графе, неориентированном графе (особенная обработка обратных ребер).
    3. Свойства времен входа/выхода. Задача: в подвешенном дереве проверять, является ли вершина v предком вершины u. Топологическая сортировка ациклического графа.
3. Компоненты сильной связности в ориентированном графе, конденсация. Поиск компонент сильной связности, построение конденсации.
4. Поиск мостов в неориентированном графе. Поиск точек сочленения.

Материалы

Конспект

День 02. Остовные деревья и СНМ

Теория

1. Определение минимального остовного дерева: набор ребер, с концами образуют дерево, суммарный вес минимален.
2. Алгоритм Прима за O(n2)O(n^2)O(n2). Получается напрямую из леммы, на каждом шаге ищем безопасное ребро из накопленного T. Алгоритм Прима с использованием set за O((n+m)log⁡n)O((n + m) \log n)O((n+m)logn) (priority_queue, O(mlog⁡m)O(m \log m)O(mlogm)). Восстановление.
3. Лемма о разрезе (безопасном ребре): если есть выбранные рёбра какого-то MST и разбиение графа на две части (разрез), такое, что выбранные рёбра не пересекают разрез то можно добавить минимальное по весу ребро, пересекающее разрез. От противного: добавим это ребро в MST, получится цикл, удалим в нем другое ребро, пересекающее разрез. Вес удалённого ребра не меньше, чем вес

добавленного. Использование леммы о разрезе для обоснования алгоритма Прима и Краскала.

4. Алгоритм Краскала. При сортировке очередное ребро безопасно для какой-то компоненты. Надо сливать компоненты, O(mlogm+n2)O(m \log m + n^2)O(mlogm+n2) в лоб, O(mlogm+nT(merge))O(m \log m + n T(merge))O(mlogm+nT(merge)).
5. Система непересекающихся множеств, постановка задачи.
    1. Реализация на vector<vector>. Весовая эвристика - вливаем меньший вектор в конец большего вектора.
    2. Реализация при помощи леса. Ранговая эвристика, оценка O(logn)O(\log n)O(logn) для ранговой эвристики.
    3. Сжатие путей, оценка сложности эвристики со сжатием путей.
    4. Говорим, что объединение ранговой эвристики и эвристики сжатия путей вместе работают очень быстро (обратная функция Аккермана, т.е. де-факто - константа), без доказательства.

Сложность эвристики со сжатием путей

АС: Это доказательство сложное для тех, кто не знаком с амортизационным анализом (большинство младших школьников) и плохо подходит для примера первого доказательства такого плана. Лучше опустить, если группа слабее или большинство школьников из 7-8-9 классов).

Теорема. В СНМ с эвристикой переподвешивания выполнение m операций на n исходных множествах требует времени O(mlogn)O(m \log n)O(mlogn).

Доказательство

Разделим ребра на три типа: рёбра, ведущие в корень, а также "лёгкие" и "тяжелые" ребра. Тяжелое ребро - это ребро, на котором висит как более половины вершин в поддереве, висящем на вершине этого ребра (верхней), а легкие - все остальные. Заметим, что в вершину не может входить более одного тяжелого ребра.

Посчитаем количество переходов по ребрам каждого типа отдельно. Пусть мы сделали суммарно m запросов. Запрос на объединение состоит из двух запросов нахождения корня и одного переподвешивания.

Количество переходов по ребрам корень составляет O(m)O(m)O(m).

Суммарное число переходов по лёгким ребрам на один запрос не более, чем O(logn)O(\log n)O(logn), так как при переходе по легкому ребру вниз размер поддерева, подвешенной к этой вершине сокращается не менее, чем в два раза.

С тяжелыми ребрами сложнее. Если мы переходим вверх по тяжелому ребру, то его нижний конец будет переподвешен в корень (ребро перейдет в категорию "ребра, ведущие в корень"), но вместо него может появиться другое тяжёлое ребро, подвешенное

вместо него вниз в той же вершине. При этом размер поддерева, подвешенного в этой вершине, уменьшится минимум в два раза, поэтому в одной вершине процедура "пройти по тяжёлому ребру, удалить тяжёлое ребро, вместо него появилось другое тяжёлое ребро" в одной вершине может быть выполнена не более, чем O(log⁡n)O(\log n)O(logn) раз.

Посчитаем число вершин, от которых вниз идёт тяжёлое ребро. Такая вершина может появиться только в результате переподвешивания, поэтому число таких вершин не более, чем O(m)O(m)O(m), и в каждой такой вершине проход по тяжёлому ребру может выполняться не более, чем O(log⁡n)O(\log n)O(logn) раз, итого общее число операций не более, чем O(mlog⁡n)O(m \log n)O(mlogn).

Доказательство сжатия пути (2)

Дима Саютин: доказательство выше корректное, но мне кажется переусложнённым с точки зрения изложения, предлагаю рассказывать это доказательство вот так:

Ребра бывают трёх типов:

1. Ребро в корень, которое ровно одно на пути. Его принципиально оценить отдельно, так как после прохода по нему не происходит переподвешивания.
2. Лёгкое ребро (на котором висит меньше половины поддерева). На любом пути на верх не более log(n) лёгких рёбер, так как размер поддерева при переходе наверх увеличивается в два или более раз.
3. Тяжёлое ребро. Переходы по таким рёбрам оценим в сумме.

Заметим, что после того момента, как вершина перестаёт быть представительской, её размер поддерева фиксируется константой <= n и после этого только уменьшается.

При переходе по тяжёлому ребру (и не ребру в корень) v->u, размер поддерева u уменьшается хотя бы в два раза.

Значит тяжёлых переходов в вершину u будет не более log(n).

Итого формальная оценка: если n вершин и m запросов, то количество проходов не более

1. m [на ребро в корень]
    - m log (n) [на лёгкие рёбра]
    - n log(n) [на тяжёлые рёбра]

Материалы

Конспект

День 03. Дерево отрезков

Теория

Основная реализация дерева отрезков - на массиве, индексация элементов массива - с нуля (потомки элемента i - 2*i+1 и 2*i+2), запросы - на полуинтервалах [l, r).

1. Задачи RMQ, RSQ. Наивные реализации RMQ и RSQ.
2. Описание структуры ДО.
3. Разбиение произвольного отрезка на O(log n) элементарных: на каждом уровне используется не более 4-х
4. Реализация дерева при помощи массива, структура массива. Полуинтервалы. Рекурсивная операция построения дерева на подотрезке (build_tree). Реализация запроса на отрезке (query_max).
5. Размер массива для хранения дерева
    - 2n недостаточно, достаточно 4n (привести пример, когда 2n недостаточно)
    - более точная оценка - 2 * 2^k для минимальной степени двойки, большей или равной n.
    - обратить внимание, что неиспользуемые элементы могут быть по дороге
6. Примеры различных операций и запросов:
    - сумма на отрезке
    - максимум на отрезке
    - количество максимумов на отрезке
    - индекс максимума на отрезке
    - k-я единица на отрезке (массив из 0 и 1)
    - максимальное число подряд идущих 1 на отрезке (массив из 0 и 1).
7. Изменение одного элемента (add_elem).
8. Групповые изменения типа "увеличить значение на отрезке". Поле add ("прибавь add на всем поддереве"), операция add_segment, операция push, правило "прежде чем идти в поддерево, сделай push".
    - есть два подхода к хранению данных в корне несогласованного поддерева. Можно хранить сразу модифицированную информацию, тогда модификация и push сложнее, но обновление детей и возвращение значения проще. Можно хранить немодифицированную информацию, тогда модификация корня происходит перед обнулением модификатора в push, но следует учитывать модификатор при возвращении значения.
9. Групповое изменение типа "присвоить новое значение на отрезке". Отличие - необходимость проталкивать значение (push) при модификации отрезка.

Эталонная реализация

tree - дерево отрезков (без групповой операции)

add - групповая операция на отрезке, увеличение всех элементов отрезка на величину delta.

Передаваемые параметры:

i - индекс текущей вершины

[l, r) - полуинтервал, за который отвечает вершина

[q_l, q_r) - полуинтервал, для которого выполняется запрос.

```
vector<int> a(n);
vector<int> tree(4 * n);
vector<int> add (4 * n);

const int INF = 1e9;

void build(int i, int l, int r)
{
    if (l + 1 == r)
        tree[i] = a[l];
    else
    {
        build(2 * i + 1, l, (l + r) / 2);
        build(2 * i + 2, (l + r) / 2, r);
        tree[i] = max(tree[2 * i + 1], tree[2 * i + 2]);
    }
}

int get_max(int i, int l, int r, int q_l, int q_r)
{
    if (q_l <= l && r <= q_r)
        return tree[i] + add[i];
    if (q_r <= l || q_l >= r)
        return -INF;
    return max(get_max(2 * i + 1, l, (l + r) / 2, q_l, q_r),
               get_max(2 * i + 2, (l + r) / 2, r, q_l, q_r))
               + add[i];
}

void add_elem(int i, int l, int r, int q_i, int delta)
{
    if (q_i < l || q_i >= r)
        return;
    if (l + 1 == r)
        tree[i] += delta;
    else
    {
        add_elem(2 * i + 1, l, (l + r) / 2, q_i, delta);
        add_elem(2 * i + 2, (l + r) / 2, r, q_i, delta);
```

```
        tree[i] = max(tree[2 * i + 1] + add[2 * i + 1],
                      tree[2 * i + 2] + add[2 * i + 2]);
    }
}

void add_segment(int i, int l, int r, int q_l, int q_r, int delta)
{
    if (q_r <= l || r <= q_l)
        return;
    if (q_l <= l && r <= q_r)
        add[i] += delta;
    else
    {
        add_segment(2 * i + 1, l, (l + r) / 2, q_l, q_r, delta);
        add_segment(2 * i + 2, (l + r) / 2, r, q_l, q_r, delta);
        tree[i] = max(tree[2 * i + 1] + add[2 * i + 1],
                      tree[2 * i + 2] + add[2 * i + 2]);
    }
}
```

День 04. Декартово дерево

Теория

1. Двоичное дерево поиска, поиск в нем. Добавление элементов в двоичное дерево поиска. Проблема балансировки.
2. Декартово дерево, реализация при помощи структур и указателей.
3. Случайные приоритеты, оценка высоты (без доказательства).
4. Операции merge и split, вставка и удаления из дерева при помощи 3-х операций merge и split.
5. Вставка при помощи одной операции split, удаление при промощи одной операции merge.
6. k-ая порядковая статистика за O(log) (спуск по дереву). Почему set в STL это не умеет?
7. Единственность декартова дерева для набора пар (при различных x и y). Построение дерева по отсортированному набору пар за O(n).

День 05. ДД по неявному ключу

1. Дорассказать то, что не успели.

2. In-order нумерация вершин дерева. Сохранение in-order порядка вершин при split и merge, интерпретация ДД как массива с операциями "склеить" и "разделить" (rope). Отказ от x-ов.
    1. Задача: дана строка, отвечать на запросы "циклически сдвинуть подстроку" и "найти k-ый символ".
3. Групповые операции в ДД, пометки add, операция push.
    1. Задача: вставка-добавление элемента, прибавление на отрезке, сумма на отрезке.
    2. Задача: добавить арифметическую прогрессию ко всем элементам отрезка.
    3. Задача: вставка-добавление элемента, реверс подотрезка, сумма на отрезке.

День 06. Фенвик, Sparse, корневая

Теория

1. Дерево Фенвика.
    - Реализация.
    - Ассимптотика операций.
    - Двумерный Фенвик, многомерный Фенвик.
2. Решение задачи RMQ за время O(n log n)/O(1): sparse table.
    - В каком порядке хранить массив в sparse table: sparse[t, i], чтобы попадало в кеш.
3. Корневая оптимизация для реализации структур данных.
    - Идея корневой декомпозиции. В зале кинотеатра нужно поставить K мест в виде прямоугольника n×m. Чтобы дойти до места нужно подняться снизу до нужного ряда и затем пройти вдоль всего ряда. Каким нужно взять n и m, чтобы минимизировать максимальный путь до места?
    - Задача: структура изменение одного элемента, максимум (сумма) на отрезке. Решение: разбиваем на блоки длиной sqrt(n).
    - Дополнение: групповое обновление (прибавление на отрезке). Решение: храним значение add в каждой вершине, проталкиваем вниз при необходимости.
4. Корневая оптимизация со структурой данных внутри.
    - Задача: увеличение на отрезке; запрос: количество элементов, которые <= x на данном отрезке. Решение: храним внутри каждого кусочка элементы упорядоченными.
5. Корневая оптимизация с перестроением структуры данных (нужна вставка и удаление элементов).
    - Задача: групповая операция (сумма), вставка и удаление элементов.
    - Решение номер 1: разбиваем на sqrt(n) векторов длины sqrt(n). Вставляем и удаляем в середину каждого вектора. Если длина вектора стала больше чем sqrt(n) - перестраиваем структуру.

- Решение номер 2: отрезок храним кусками, сам массив разбивается на куски и хранятся указатели на фрагменты первого массива. В самом начале один кусок размером [0, n). Для вставки - разрезаем кусок на две части, элемент добавляем в конец. Это приводит к увеличению количества кусков на 2. Когда кусков стало больше, чем sqrt(n) -- перестраиваем. Суммы на отрезке реализуем через префиксные суммы.
- Теперь научимся делать как с декартовым деревом - порезать на части, переставить части. В решении 1 - разрезание приводит к тому, что одна часть разрезается на 2, т.е. количество частей увеличивается. Если частей стало много - делаем перестроение. В решении 2 - всё уже реализовано (split, merge).

[Подробный план корневой декомпозиции на два дня]

Код Фенвика

```
// Запрос
int sum (int r) { // Сумма элементов на отрезке a[0..r]
    int result = 0;
    while (r >= 0) {
        result += tree[r];
        r = (r & (r + 1)) - 1;
    }
    return result;
}

// Обновлениесдел
int add(int i, int delta) { // Увеличение элемента a[i] на величину delta
    while (i < n) {
        tree[i] += delta;
        i = i | (i + 1);
    }
}

```

День 07. Корневая по запросам

Теория

1. Можно дорассказать то, что не успели вчера.
2. Декомпозиция по запросам.
    - Задача: дан set, нужно добавлять элементы и подсчитывать, сколько есть элементов от l до r. Решение - храним упорядоченный массив и список

добавленных элементов отдельно. Перестраиваем множество, когда накопилось много добавленных элементов.
    - Dynamic Connectivity Offline. Запросы вида: соединить две вершины ребром, удалить ребро, проверить, лежат ли две вершины в одной компоненте связности.
    - Винни-Пух и бочки мёда.
    - Задача на графе. Перекрашиваем вершину, запрос - для данной вершины сколько различных цветов среди её соседей.
3. Алгоритм Мо.
    - Постановка задачи.
    - Реализация.
    - Оценка сложности.
    - Примеры запросов (минимум, максимум, медиана, количество различных чисел). Как искать медиану без декартова дерева? (Ответ - два multiset).
    - Пусть запрос типа "медианный элемент". Давайте сделаем O(1) переход, и O(sqrt(n)) ответ на запрос при помощи корневой оптимизации. Для этого храним, какие элементы использованы при помощи подсчёта, а ответ на запрос - корневой оптимизацией.

[Все подробности в плане Димы Саютина]

День 08. Элементарная геометрия

Теория

1. Точка, вектор. Декартовы и полярные координаты. Переход между ними. Функция atan2.
2. Скалярное и векторное произведение, определения и формулы в координатах. Длина вектора, нормирование вектора. Взаимное расположение векторов. Вычисление ориентированного угла при помощи atan2.
3. Прямая, представление в виде уравнения, двух точек, точки и направляющего вектора, переходы между разными представлениями.
4. Вычисление расстояния от точки до прямой, от точки до отрезка. Расстояние между отрезками.
5. Проверка пересечения двух отрезков.

Материалы

[План лекции от Антона Тимофеева]

День 9. Строки

Теория

1. Задача поиска подстроки в строке. Наивный алгоритм.
2. Префикс-функция. Алгоритм КМП, доказательство сложности. Применения:
    - Поиск подстроки в строке.
    - Период строки.
    - Количество различных подстрок в строке.
3. Z-функция, доказательство сложности. Применения:
    - Поиск подстроки в строке.
    - Период строки.
    - Количество различных подстрок в строке.
4. Полиномиальные хэши по модулю. Формула для префиксных хэшей, вычисление хэша подстроки за O(1). Применение:
    - Поиск подстроки в строке за O(n + m).
    - Лексикографическое сравнение подстрок за log(длина подстроки).
    - Минимальный циклический сдвиг за O(длина строки)
    - Поиск наибольшей общей подстроки.
    - Поиск наибольшего подпалиндрома, подсчёт числа палиндромов.
    - Парадокс дней рождений, подбор модуля.

День 10. Окружности, многоугольники

Теория

1. Пересечение прямых (СЛУ от двух переменных).
2. Поворот вектора на угол (матрица поворота). Пересечение окружности и прямой, двух окружностей. Построение касательных из точки к окружности, общих касательных двух окружностей.
3. Многоугольник, площадь многоугольника. Проверка принадлежности точки невыпуклому многоугольнику за O(n) двумя способами: число точек пересечения с лучем, подсчет суммы углов.
4. Выпуклый многоугольник. Проверка многоугольника на выпуклость. Принадлежность точки выпуклому многоугольнику за O(log n).
5. Выпуклая оболочка множества точек. Построение выпуклой оболочки за O(n log n) (алгоритм Грэхема).

Материалы

[План лекции от Антона Тимофеева]

День 11. Динамическое программирование

1. ДП по префиксам/количеству: НВП, НОП. Решение НВП за O(n log n) с помощью ДО, с помощью стека и бинпоиска.

2. ДП по подотрезкам: количество подпалиндромов, максимальный подпалиндром, построение минимальной триангуляции многоугольника.
3. ДП по поддеревьям: кол-во связных подграфов дерева, максимальное независимое множество, максимальное взвешенное паросочетание.
    a. Динамиика в максимальном независимом множестве, паросочетании: данная вершина не включена, данная вершина обязательно включена
4. ДП по цифрам.
    a. Количество чисел от 1 до n с суммой цифр k. Состояние динамики - длина суффикса и сумма цифр на нём, то есть log(n)×k состояний, количество переходов - 10×log(n)×k. Восстановление ответа - 10×log(n) (идем от старшей цифры к младшей). (*) Запретим две повторяющиеся цифры, тогда состояний динамики будет 10×log(n)×k.
5. ДП по подмножествам
    a. Макс. паросочетание - динамика $O(2^n)$ состояний, $O(n \times 2^n)$ переходов.
    b. Гамильтонов путь в произвольном графе. Состояние динамики - использованные вершины и последняя вершина, $O(n \times 2^n)$ состояний. Переходов - $O(n^2 \times 2^n)$

Не успеем: Рюкзак - разбить n предметов на минимальное количество подмножеств веса не более W

- за $O(4^n)$ - состояний 2^n (для каждого подмножества на сколько рюкзаков оно разбивается), для каждого подмножества делаем следующее: а) если оно непусто и масса предметов меньше W, то ответ 1, иначе перебираем все возможные меньшие подмножества (тупо за 2^n), и разбиваем это множество на два подмножества, для которых ответ уже известен.
- $O(3^n)$ - перебор подмасок делаем умно за $O(3^n)$ суммарно (см. код внизу). Обоснование, почему это $O(3^n)$.
- (можно пропустить) O $(n \times 2^n)$ - значение динамики - это пара из минимального числа рюкзаков, в которые можно уложить предметы и массы последнего неполного рюкзака.