ЛКШ, ЛКШ.2018.Август B'
B', конспект лекции

Собрано 4 августа 2018 г. в 11:18

Содержание

Тема #1: Объединение прямоугольников

3 августа

1.1. Тупое решение

Будем работать в Декартовой системе координат.
Сделаем буловую табличку, по размеру достаточную, чтобы туда поместились все наши прямоугольники.
Перебираем все прямоугольники. Смотрим на очередной. Поставим в табличку единички в те клетки, которые лежат в каждом прямоугольнике.
Это работает за $\mathcal{O}(nS)$, где S — площадь таблички, так как мы n раз проходим по табличке площади S.
TODO: картинка

1.2. Сжатие координат

Наша проблема в том, что у нас могут быть огромные координаты, и тогда S очень большое. Но ведь мы можем не пользоваться клетками, в которых нет границ наших прямоугольников. Избавимся от них.
Завели вектор (массив) coords. Запихнули туда все x и y координаты. Сортируем. Удалим дубликаты.
В C++ это делается так:

```
1 sort(coords.begin(), coords.end());
2 coords.erase(unique(coords.begin(), coords.end()), coords.end());
```

Был массив **10 20 30 40 50 60**
Хотим заменить на **0 1 2 3 4 5**
Для этого нужно запустить lower_bound от необходимого числа в coords и найти индекс этого элемента (из результата вычесть coords.begin())
можно сделать функцию:

```
1 int get(int x) {
2     return lower_bound(coords.begin(), coords.end(), x) - coords.begin();
3 }
```

Мы научились переводить из индекса в координаты (надо просто обратиться к нужному элементу массива coord) и из координат в индексы.

```
1 struct Rect {
2     int x1, y1, x2, y2;
3 };
4 vector<Rect> rectangles;
5 get(x);
6 x1 = get(x1)
7 x2 = get(x2)
```

Теперь мы можем считать, что все прямоугольники лежатт на доске $m \times m$, где m — размер coords. Так как $m = \mathcal{O}(n)$, то получим, что $S = \mathcal{O}(n^2)$
Но эти клетки разных размеров.
Как посчитать площадь прямоугольника на разжатых координатах, если нам даны его координаты на сжатых? Очень просто:

```
int real_s(Rect a) {
    return (coord[a.x2] - coord[a.x1]) * (coord[a.y2] - coord[a.y1]);
}
```

Запустили предыдущее решение на сжатых координатах, получили табличку. Нужно по этой табличке восстановить площадь на разжатых координатах. Ну окей, мы уже умеем искать площадь прямоугольника. Клетка — тоже прямоугольник. Ответ — сумма площадей всех клеток, где стоят единички.
Это работает за $\mathcal{O}(n^3)$

1.3. Сканлайн

В этом способе решения нам не нужно сжатие координат.
Теперь мы будем смотреть только на отдельные вертикальные полосы, а не на всю таблицу.
Посмотри на очередную полосу. Посмотрим, какие прямоугольники ее пересекают. Все прямоугольники на полосе выписали за $\mathcal{O}(n)$ в какой-нибудь массив. Теперь хотим найти объединение этих отрезков. Это уже умеем делать сканлайном. Это работает за $\mathcal{O}(n \log n)$, потому что сортировка. По объединению отрезков мы можем найти площадь этого объединения.
Сканлайн работает за $\mathcal{O}(sort + n)$. Но зачем нам каждый раз сортировать все точки от нужных прямоугольников, если мы можем сделать это заранее? Изначально отсортируем минимальные и максимальные y всех прямоугольников как события в сканлайне. Если этот отрезок в сканлайне не нужен, то его y мы будем просто игнорировать.
Теперь сканлайн работает за $\mathcal{O}(n)$ и $\mathcal{O}(sort)$ предподсчета.
Осталось отсортировать все x координаты и перебрать столбцы, образованные соседними x_i и x_{i+1}.
Получили решение нашей задачи за $\mathcal{O}(n^2)$

1.4. Другое решение

Здесь мы сжимаем координаты.
Теперь будем хранить одну вертикальную полоску длины, равной высоте таблицы. Хранить будем, как массив интов.
Делаем сканалйн по x-ам. В этой полоске поддерживаем сколько из еще открытых прямоугольников имеют данную координату по x.
Открылся новый прямоугольник — ко всем нужным y в полоске прибавили единичку. Закрылся — вычитаем единичку.
TODO: здесь очень нужна картинка.
Чтобы найти площадь на текущем столбце, нужно сложить площади всех точек, в которых записан не ноль. Получили сложность $\mathcal{O}(n^2)$.
Это решение легко (ну или не совсем легко) переделывается в $\mathcal{O}(n \log n)$ с помощью дерева отрезков. А ещё с ДО можно не сжимать координаты и получить $\mathcal{O}(n \log U)$ (U — максимальная координата), но это уже совсем другая история...

Тема #2: Аудитории

3 августа

Задача. Есть k лекционных аудиторий. Вы — завуч. К вам приходят n препов и говорят «я хочу прочитать лекцию со стольки-то до стольки-то». Вам нужно удовлетворить как можно больше заявок. Препам без разницы, в какую аудиторию идти.

2.1. Решение

Давайте делать обычный сканлайн и считать баланс (сколько активных лекций есть сейчас). Лекция активна, если мы еще не отказали ей в проведении.
Пришла новая лекция. Если баланс нам все ещё позволяет, то делаем эту лекцию активной, увеличим баланс. Если баланс не позволяет, то мы должны отказать либо этой лекции, либо какой-то другой активной. Посмотрим, когда заканчиваются все активные лекции. Посмотрим на максимум. Если так случилось, что максимальная активная лекция заканчивается позже, чем наша, то давайте возьмем нашу, а ту выкинем (не забудем сделать неактивной).
TODO: картинка
Итак:
Храним текущие активные заявки. Их должно быть не больше k.
Мы должны уметь отвечать на запрос «посмотри (и при надобности удали) заявку с самым далеким правым концом», на запрос «добавь заявку в множество» и на запрос «удали заявку из множества по номеру». Нам нужна структура данных, которая умеет это все.
Если сделаем массив, то получим решение за $\mathcal{O}(n^2)$ (максимум ищем за линию).
В C++ есть структура данных set. Она умеет все это за $\mathcal{O}(\log n)$

```
set<pair<int, int>> active;
// вставляем пару из конца лекции и ее номера в сет активных лекций
active.insert({end_time[lection_id], lection_id});
// получаем максимальное время окончания среди всех активных лекций
max_lection_end = (*active.rbegin()).second;
// удаляем лекцию с максимальным временем окончания
active.erase(active.rbegin());
// удаляем лекцию по номеру
active.erase({end_time[lection_id], lection_id});
```

Как можно заметить, set сортирует элементы по возростанию, поэтому мы используем rbegin. Но это можно исправить.

```
set<pair<int, int>, greater<pair<int, int>>> active;
```

В питоне нет такого сета. Но там есть heapq.
Он не умеет удалять по номеру :-(
Будем делать ленивое удаление. Для каждого элемента помним, должны ли мы удалить его (например, массив булов). Когда ищем максимум, достаем элементы до тех пор, пока не встретим ещё не удаленный.

2.2. Другое решение

Сортируем отрезки по правому концу.
Возьмем первый отрезок (самый левый по правому концу). Пихаем его в какую-нибудь аудитория. Потом второй и так далее.

Пусть нам пришел очередной отрезок. Если не нашлось аудитории, в которую его можно пихнуть (для этого в этой аудитории последняя лекция должна заканчиваться не позже, чем время начала в этой заявке), то эту заявку мы обработать не можем. А если новую заявку можем куда-то запихнуть, то положим туда, где позже заканчивается лекция.
В какой-то структуре поддерживаем, во сколько освободится аудитория. Нужны запросы вида «обнови», «увеличь», «найди максимальное значение такое, что оно меньше либо равно t».
Последнее — это upper_bound
Сет все это умеет. Куча грустит :-(
Здесь мы получили рассадку в явном виде.

Тема #3: Прибавление на отрезке и сумма

3 августа

У нас есть большой (10^7) массив. Сначала приходят запросы вида «прибавь ко всем элементам на отрезке от L по R значение val», потом приходят запросы вида «скажи сумму элементов на отрезке от L по R».

3.1. Что если нет запросов первого типа?

Построили префикс-суммы и отвечаем на каждый запрос за $\mathcal{O}(1)$. Свели нашу задачу к «обработай запросы первого типа и выведи полученный массив».
Кстати, в C++ есть std::prefix_sum

3.2. Решение

Посчитаем все элементы массива. Тупо это работает за $\mathcal{O}(nq)$, где q — кол-во запросов.
Сделаем это за $\mathcal{O}(n+q)$
Для каждого запроса сделаем два события: «ко всем элементам на отрезке $L, +\infty$ прибавить x ко значению»; ко всем элементам на отрезке $R+1, +\infty$ прибавить $-x$ ко значению.
Сделаем сканлайн по этим событиям. Будем поддерживать, сколько нужно прибавить всем, начиная с текущего элемента. По окончании сканлайна мы будем знать ответ для каждого элемента.
Это работает за $\mathcal{O}(n + q \log q)$, если мы сортируем чем-то обычным и $\mathcal{O}(n+q)$, если подсчетом.

Тема #4: Два указателя

3 августа

Дан массив чисел. Нужно найти его наидлиннейший подотрезок, в котором все числа различны.

4.1. Не очень тупое решение $\mathcal{O}(n^2 \log n)$ или $\mathcal{O}(n^2)$

Перебираем все левые границы. Для каждой левой границы идем направо, пока не встретим элемент, который уже был.

4.2. Умное решение $\mathcal{O}(n)$

Берем прошлое решение. Где ответ для $L+1$? Не ближе, чем для L.

```
set<int> s;
j = 0;
for (i = 0; i < n; ++i) {
    while (j < n && !s.count(a[j])) {
        s.insert(a[j++]);
        relax(i, j);
    }
    s.erase(a[i]);
```

Функция relax обновляет ответ отрезком $[i, j]$ (если он лучше).
Это решение за $\mathcal{O}(n \log n)$. Чтобы сделать $\mathcal{O}(n)$, то нужно использовать unordered_set. Он все делает за $\mathcal{O}(1)$.

Тема #5: Объединение отрезков

3 августа

Это разбор задачи с практики. Он здесь нужен, чтобы было понятно, о чем идет речь в следующем пункте.

Задача. Даны n отрезков. Хотим научиться искать их объединение.

5.1. Решение

Ответ будем хранить в векторе пар ans.

Отсортировали события (отрезок открывается-закрывается) по координатам. Теперь идем сканлайном, поддерживаем максимальную покрытую правую границу ($maxr$). Изначально она $-\infty$.

Встречаем открывающийся отрезок. Есть два случая. Первый — его начало покрыто ($seg.l \leqslant maxr$). Тогда просто делаем $ans.back().second = max(ans.back().second, seg.r)$.

Иначе нужно добавить новый отрезок в ans. Ну сделаем это. Надо сделать push_back(seg).

Здесь никак не надо обрабатывать закрытие отрезка, поэтому эти события мы игнорируем (а лучше вообще не добавляем в вектор для сканлайна).

Время $\mathcal{O}(sort + n)$.

5.2. Другое решение

Так же отсортировали события.

Идём сканлайном. При появлении нового отрезка снова два таких же случая. Если наша левая граница не больше $maxr$, то ничего не делаем. Иначе делаем $ans.push\_back(seg.l, ??)$, где ?? — любое число.

Если встречаем конец отрезка, говорим $ans.back().r = seg.r$.

Те же $\mathcal{O}(sort + n)$.

Тема #6: Отрезки на окружности

3 августа

Нужно найти объединение отрезков на окружности. Проблема в том, что если мы хотим сканлайн, то не знаем, с какой точки начинать.

6.1. Решение

Выбрали какую-то точку на окружности. Скажем, что это ноль. «Разрежем» окружность по этому нулю, но перед этим посчитаем, какой баланс в нуле (сколькими отрезками он покрыт). Если там баланс не ноль, то добавим в ответ пару $\langle$что-то, ??$\rangle$. Что такое «что-то», мы определим потом.
Делаем обычный сканалйн на прямой (объединение отрезков было в контесте).
Если баланс в точке разреза был ноль, то мы победили.
Если не ноль, то скажем, что это «что-то» — начало последнего отрезка в объединении, полученном сканлайном. Еще надо не забыть удалить последний элемент в разбиении.