Конспект по вычислительной геометрии (Ковалёв)

Volhov M., Mukhutdinov D., Belyy A.

7 января 2016 г.

Содержание

1 Как пользоваться этим документом

1. Писать билетики параллельно
2. Все формулы оформлять в латехе.
   - `http://www.math.uiuc.edu/~hildebr/tex/course/intro2.html`
   - `https://en.wikibooks.org/wiki/LaTeX/Mathematics`
   - `https://en.wikibooks.org/wiki/LaTeX/Advanced_Mathematics`
   - Рекомендуется смотреть конспект Сугака тут.
3. Можно пользоваться `(org-toggle-pretty-entities)` чтобы latex отображался юникодом в emacs'е.

4. **TODO** обозначают степень написанности материала. Положительное значение интерпретируется как процент. X – метка для "тут нифига не ясно и проблемы".

5. Смотреть превью формул с помощью `C-c C-x C-l`.

6. Экспортить с помощью `C-c C-e l p`. Если не работает – сначала в ‘tex‘, а потом уже экспортировать ‘pdflatex‘’ом. Экспорт игнорирует ошибки, поэтому сделать это руками и пофиксить их – не так плохо.

7. Если что-то не собирается, писать `@volhovm`.

8. Синтаксис теха смотреть хз где, но есть `http://detexify.kirelabs.org/classify.html` для непонятных символов.

9. Синтаксис орга – это тут:

   - `http://orgmode.org/manual/Emphasis-and-monospace.html`
   - `http://orgmode.org/manual/LaTeX-and-PDF-export.html#LaTeX-and-PDF-export`
   - `http://orgmode.org/tmp/worg/org-tutorials/org-latex-export.html`
   - Остальное (списки, хедеры, блабла) очевидно.

2 Tickets

4	1	Принадлежность точки выпуклому и невыпуклому многоугольникам
21	1	Триангуляция Делоне. Алгоритм и доказательство его корректности.
5	2	Статические выпуклые оболочки на плоскости. Джарвис, Грэм, Эндрю, Чен, QuickHull. Оболочка многоугольника, оболочка полилинии.
20	2	Триангуляция Делоне. - существование; - приводимость любой триангуляции флипами к ТД; - эквивалентность критерия Делоне для треугольников критерию для ребер.
8	3	Триангуляция многоугольника. Существование, ушная триангуляция.
17	3	Сумма Минковского (определение, вычисление)
12	4	ППЛГ и РСДС (PSLG и DCEL): определение, построение РСДС множества прямых
15	4	Трапецоидная карта.
10	5	Пересечение полуплоскостей, связь с выпуклыми оболочками
13	5	Пересечение многоугольников (PSLG overlaying)
7	6	Выпуклая оболочка в n-мерном пространстве. Quick-hull и вероятностный алгоритм.
11	6	Пересечение множества отрезков.
9	7	Триангуляция многоугольника заметающей прямой
14	7	Локализация в ППЛГ. - методом полос (персистентные деревья); - Киркпатрик.
6	8	Динамическая выпуклая оболочка (достаточно $\log^2$ на добавление/удаление)
19	8	Граф видимости и планирование движения. - построение графа видимости заметающим лучом; - сокращение графа видимости; - построение навигационного графа на трапецоидной карте; - планирование маршрута невыпуклого тела с вращением (без суммы Минковского).
3	9	Пересечение отрезков и поворот: определение, свойства, вычисление
22	9	Диаграмма Вороного. - определение и свойства; - диаграмма Вороного высших порядков, построение; - связь с подразбиением Делоне (ближайший и дальнейший); - алгоритм построения ДВ.
1	10	Skip quadtree: определение, время работы
18	10	Минимальная охватывающая окружность множества точек. Вероятностный алгоритм.
2	11	Пересечение прямоугольника с множеством прямоугольников и непересекающихся отрезков: - range tree + fractional cascading; - interval tree; - segment tree; - priority search tree; - k-d tree. van Kreveld, de Berg, Overmars, Cheong
16	11	Диаметр множества точек (вращающиеся калиперы)

3 2 1: Skip quadtree

Сперва поймем, что такое квадродерево и сжатое квадродерево.

3.1 Определение (квадродерево)

Дерево, каждая внутренняя (не листовая) вершина которого содержит 4 ребёнка. Построение квадродерева по множеству точек P - пусть дан квадрат S, содержащий все точки P. Если $|P|=1$, то квадродерево состоит из одного листа, соответствующего квадрату S. Если $|P|>1$,

то поделим S на 4 маленьких квадрата и рекурсивно запускаемся от подмножеств P, соответствующих разным четвертям. Картинка.

3.2 Определение (интересный квадрат)

Квадрат, в котором содержится хотя бы одна точка из P. Пример.

3.3 Определение (сжатое квадродерево)

Квадродерево, внутренним вершинам которого соответствуют только интересные квадраты. Построение сжатого квадродерева - строим по обычному квадродереву следующим образом: у внутренней вершины заводим по 4 указателя для 4 четвертей. Если в четверти 2 и более точки P - указатель ссылается на наибольший интересный квадрат этих точек, если одна - ссылается на неё саму, если 0 - указатель NULL.

3.4 Время работы операций в сжатом квадродереве

O(n) на локализацию, вставку и удаление. Могли бы просто завести список точек, в общем. Самое время узнать про skip quadtree.

3.5 Определение (randomized skip quadtree)

Последовательность сжатых квадродеревьев над подмножествами точек P: $\{P_0, P_1, \ldots, P_k\}$, где $P_0 = P$ - исходное множество точек, $P_i \in P_{(i-1)}$ и каждый элемент $P_{(i-1)}$ входит в P_i с вероятностью $p \in (0; 1)$. Skip quadree - это последовательность "уровней"$\{Q_i\}$, где "уровень"Q_i - сжатое квадродерево над точками P_i.

3.6 Время работы операций в skip quadtree

O(log(n)). Сначала опишем, как проходят операции, а потом докажем их время работы.

3.6.1 Общая операция 'подъем'

По вершине с уровня i нужно получить эту же вершину на уровне i - 1. За O(1). Как сделать? Проще всего как в skip list: "прошить"ссылками на вершину уровня выше каждую внутреннюю вершину каждого квадродерева.

3.6.2 Локализация

Локализуемся на уровне k, далее сделаем подъем, окажемся в квадродереве уровня ниже и локализуемся в нем, но уже не от корня, а с того квадрата, который нашли на предыдущем уровне. Повторим это k раз. В результате локализуемся на нулевом уровне.

3.6.3 Вставка

Локализуемся на всех уровнях, запоминая ссылки. Сделаем вставку в квадродерево нулевого уровня, далее с вероятностью p сделаем вставку на 1 уровне и так далее до первого недобавления. Количество уровней при этом увеличится максимум на 1 (с вероятностью p^k).

3.6.4 Удаление

Локализуемся на всех уровнях, удалим квадрат везде и обновим ссылки. Если уровень стал пустым - удалим его.

3.6.5 Лемма (о количестве шагов на одном уровне)

На каждом уровне в среднем совершается O(1) шагов для поиска точки x.

Доказательство

3.6.6 Лемма (о количестве уровней)

Математическое ожидание количества уровней составляет O(log(n)).

Доказательство

3.6.7 Теорема (о времени работы)

Локализация, вставка и удаление работают в среднем за O(log(n)).

Доказательство: следует из двух предыдущих лемм.

4 0 2: Пересечение многоугольника с множеством полигонов/отрезков

5 1 3: Пересечение отрезков и поворот

Рассмотрим задачу проверить пересечение отрезков.

Вот есть у нас $S_1 = (p_{11}, p_{12}), S_2 = (p_{21}, p_{22})$.

В общем случае с Евклидовым пространством возникакуют какие-то проблемы, поэтому рассмотрим следующее определение Афинного пространства:

A – аффинное пространство, если A – такой набор точек, что:

1. В пространстве существует хотя бы одна точка.
2. $A, B, \leftrightarrow v = \overrightarrow{AB}$, причем $B = A + v$.
3. Точка + вектор = точка.
4. ... и еще 40 аксиом векторного пространства

Аффинное пространство отличается от стандартного евклидового тем, что в нем все точки равноправны, то есть ноль не зафиксирован. Типа у нас в этом пространстве есть точки, а векторы строятся из них.

Рассмотрим гиперплоскость в n-мерном аффинном пространстве. Она, очевидно, задается $n-1$ вектором, или как минимум n точками.

Рассмотрим произвольную точку A и набор векторов: $AP_1 \cdots AP_n$. Тогда если точка A принадлежит гиперплоскости, то такой набор, очевидно, линейно зависим.

Возьмем другую случайную точку B и посмотрим, как меняются координаты при переходе из системы координат, связанной с A в систему, связанную с B (очевидно, что такой набор векторов может задавать базис, если он ЛНЗ).

Теорема 1 (О повороте)**.**
Тут должно быть какое-то утверждение о повороте.

Доказательство. Рассмотрим точку X в базисах из векторов $\{\overrightarrow{AP_i}\}_i$ и $\{\overrightarrow{BP_i}\}_i$. Тут точки P_i задают гиперплоскость, то есть принадлежат ей и не линейно зависимы друг относительно друга в ней.

$$X = X_A^1\overrightarrow{AP_1} + X_A^2\overrightarrow{AP_2} + \cdots + X_A^n\overrightarrow{AP_n} = X_B^1\overrightarrow{BP_1} + X_B^2\overrightarrow{BP_2} + \cdots + X_B^n\overrightarrow{BP_n}$$

Для каждого вектора $\overrightarrow{AP_i}$ выразим его в базисе векторов $\overrightarrow{BP_i}$.

$$\overrightarrow{AP_1} = \alpha_1^1 \overrightarrow{BP_1} + \cdots + \alpha_1^n \overrightarrow{BP_n}$$
$$\cdots$$
$$\overrightarrow{AP_n} = \alpha_n^1 \overrightarrow{BP_1} + \cdots + \alpha_n^n \overrightarrow{BP_n}$$

Подставим выраженные AP_i в первое уравнение.

$$\begin{aligned} X &= X_A^1 \left(\sum \alpha_1^i \overrightarrow{BP_i} \right) + X_A^2 \left(\sum \alpha_2^i \overrightarrow{BP_i} \right) + \cdots + X_A^n \left(\sum \alpha_n^i \overrightarrow{BP_i} \right) \\ &= \overrightarrow{BP_1} \left(\sum \alpha_i^1 X_A^i \right) + \overrightarrow{BP_2} \left(\sum \alpha_i^2 X_A^i \right) + \cdots + \overrightarrow{BP_n} \left(\sum \alpha_i^n X_A^i \right) \end{aligned}$$

Сопоставив это с X, выраженным через $\{\overrightarrow{BP_i}\}_i$, получим следующую зависимость:

$$\left(X_B^1, X_B^2, \cdots, X_B^n\right) = \left(X_A^1, X_A^2, \cdots, X_A^n\right) \times \begin{pmatrix} \alpha_1^1 & \cdots & \alpha_1^n \\ \vdots & \ddots & \vdots \\ \alpha_n^1 & \cdots & \alpha_n^n \end{pmatrix} + \left(\overrightarrow{BA}^1, \cdots, \overrightarrow{BA}^n\right)$$

Последнее — вектор перехода из точки B в A. Пусть дана точка O, которая воспринимается как ноль координат. Пусть также дана точка O', которая выражается через O. Тогда матрица A записывается следующим образом:

$$A = \begin{pmatrix} P_1 - O' \\ P_2 - O' \\ \cdots \\ P_n - O' \end{pmatrix}$$

Тут P_i и O' – это точки, координаты которых записаны отнсительно базиса $O\{e_1, \cdots, e_n\}$.

Заметим, что мы можем разбить все пространство на три класса согласно того, какой знак перехода из O в O'. A – матрица перехода от O к O',

Ориентация – свойство точки относительно базиса $O\{e_1, \cdots, e_n\}$ и гиперплоскости, заданной точками $\{P_i\}_{i=1}^n$.

Известный факт из линейной алгебры:

$$\begin{vmatrix} \overrightarrow{P_1} & 1 \\ \overrightarrow{P_2} & 1 \\ \vdots & \vdots \\ \overrightarrow{P_n} & 1 \\ \overrightarrow{A} & 1 \end{vmatrix} = \begin{vmatrix} P_1 - A \\ P_2 - A \\ \vdots \\ P_n - A \end{vmatrix}$$

Покажем, что знак детерминанта матрицы A действительно зависит от положения точки относительно гиперплоскости. Возьмем A, B, рассмотрим множество точек $\{\overrightarrow{A}t + \overrightarrow{B}(1-t)\}$.

тут какая-то магия, TODO

□

6 З 4: Локализация в многоугольнике

Есть многоугольник P и вершина q. Задача локализации qрешается по-разному в зависимости от вида P.

6.1 P - выпуклый

Время работы O(log(n)). Зафиксируем направление обхода точек P. Если q лежит левее грани [p_0, p_1] или правее грани [p_0, p_n-1], точка снаружи. Иначе бинпоиском найдем ребро [p_i, p_i+1] такое, что повороты [p_0, p_i, q] и [p_0, p_i+1, q] имеют разный знак. Проверим поворот [p_i, p_i+1, q]. Если левый - точка внутри, если правый - снаружи.

6.2 P - невыпуклый

Время работы O(n). Пустим луч из точки куда-нибудь (например, по иксу), посчитаем количество пересечений с границей. Если луч пересекается по точке P, будем учитывать только верхнюю точку. Если луч пересекается по прямой, забьем на такое пересечение. А если точки целочисленные, можно просто пускать косой луч.

7 2 5: Статические выпуклые оболочки в $\mathbb{R}$

7.1 3 Джарвис (заворачивание подарка)

1. Берем самую нижнюю левую точку p_0.
2. За $O(n)$ перебираем все точки, берем минимальную точку по углу относительно p_0.

   Пояснение: Пусть мы хотим сравнить по этому параметру точки p_i и p_j. Тогда $p_i < p_j \Leftrightarrow turn(p_0, p_i, p_j) < 0$.
3. Добавляем выбранную точку в оболочку, проделываем то же самое с ней и т. д.

Общее время работы, очевидно, $O(n^2)$
Доказательство корректности
Пусть после завершения Джарвиса осталась точка P, не лежащая внутри полученной оболочки. Это значит, что она лежит справа от некоторого ребра AB (считаем, что ребра оболочки упорядочены против часовой стрелки, так что все внутренние точки лежат слева от них).

Но тогда P меньше по повороту относительно A чем B. Значит, мы должны были выбрать ее, а не B, для построения очередного ребра оболочки, когда мы рассматривали точку A. **Противоречие**. Следовательно, такой точки P не существует.

7.2 3 Грэм

Возьмем самую левую нижнюю точку p. Отсортируем все остальные точки по повороту, который они образуют с этой каким-нибудь нормальным алгоритмом (за $O(n \log n)$). Если все три точки лежат на одной прямой, то меньшей считается та точка, которая ближе к p.

Положим в стек точку p и первую точку из отсортированного списка остальных. Далее идем по всем точкам из списка и делаем следующее:

1. Обозначим рассматриваемую точку как a, а последние 2 точки, лежащие на стеке - как b и c.
2. Если $turn(c, b, a) \geq 0()$, то скидываем со стека точку b и возвращаемся к пункту 1
3. Иначе кладем a на стек и рассматриваем следующую вершину по списку.

В конце в стеке будут лежать вершины выпуклой оболочки.
Корректность
Докажем корректность алгоритма по индукции.

- **База** На третьем шаге алгоритм, очевидно, построит корректную выпуклую оболочку для первых 3 точек (просто потому, что невыпуклую построить нельзя))))

- **Переход** Пусть на k - 1 шаге построена корректная выпуклая оболочка для первых k - 1 точек. Докажем, что на k-ом шаге будет построена корректная выпуклая оболочка для k точек.

    1. В силу отсортированности точек по повороту, точки $p_1..p_{k-1}$ лежат слева от ребра p_kp_0 (возможно, p_{k-1} лежит на ребре)
    2. На шаге 2 алгоритма из прошлой оболочки будут выброшены все вершины, видные из p_k, то есть, ни с каким из оставшихся в оболочке ребер p_k не будет образовывать правый поворот.
    3. Следовательно, все ребра новой оболочки будут образовывать со всеми остальными вершинами левый (или нулевой) поворот, что нам и нужно.

    Асимптотика

    Сортировка точек за $O(n \log n)$. Проход по точкам за $O(n)$, так как каждая точка может 1 раз быть добавлена в стек и 1 раз из него удалена, всего точек n. Итого $O(n \log n)$.

7.3 3 Эндрю

Эндрю - это почти в точности Грэм.

1. Возьмем самую левую и самую правую точки - p_0 и p_n
2. Разделим все множество точек на "верхние"и "нижние выше прямой p_0p_n и ниже ее, соответственно.
3. Для "верхних"и "нижних"точек построим верхнюю и нижнюю оболочку соответственно. Строить будет Грэмом, но представляя, что точка p_0 лежит в inf и $-\inf$ соответственно. Тогда мы можем сказать, что обычная сортировка точек по координате x эквивалентна сортировке по повороту относительно бесконечно удаленной точки. Значит, отсортируем на самом деле точки каждой из половин по x-координате и запустим Грэма.
4. Объединим верхнюю и нижнюю оболочки.

Корректность

Грэм корректен, а значит, верхняя и нижняя оболочки будут корректны. Тогда и вся оболочка корректна.

Асимптотика

Ровно такая же как у Грэма. Но на практике Эндрю чуть быстрее лишь потому, что сортировка идет по x-координате, а не по повороту, и это быстрее.

7.4 3 Чен

Чен - это продукт классической методики улучшения каких-то алгоритмов: возьмем 2 известных алгоритма - один просто хороший, а другой - обладающий неким нужным свойством. Разобьем задачу на подзадачи, подзадачи решим одним алгоритмом, а объединим решения другим. Останется подобрать константу посерединке.

Так и здесь - Чен объединяет просто хороший алгоритм Грэма с output-sensitive алгоритмом Джарвиса, получая хороший output-sensitive алгоритм с временем работы $O(n \log k)$, где k - количество вершин выпуклой оболочки.

Алгоритм

Разобьем все точки на произвольные группы по m (или меньше) штук в каждой. Тогда всего групп будет $r = \frac{n}{m}$

1. Для каждой группы в отдельности найдем ее выпуклую оболочку Грэмом за $O(m \log m)$. Значит, всего на этот шаг уйдет $O(r) \cdot O(m \log m) = O(\frac{n}{m}) \cdot O(m \log m) = O(n \log m)$ времени.

2. Теперь запустим на всех точках Джарвиса. Однако заметим, что среди точек, входящих в одну группу, мы можем выбрать самую левую по повороту бинпоиском - так как для группы построена выпуклая оболочка. (Бинпоиск - это вот эта прекольная тема с вложенными выпуклыми оболочками, например)

   Значит, на одном шаге Джарвисанам нужно перебрать все группы, среди которых подходящую точку мы ищем за $O(\log m)$. Итого - $O(r \log m) = O(\frac{n}{m} \log m)$. Всю выпуклую оболочку мы найдем за $O(\frac{kn}{m} \log m)$.

Сложив асимптотики двух шагов, видим, что полное время работы - $O(n(1 + \frac{k}{m}) \log m)$. Из этого получится желанная асимптотика $O(n \log k)$, если мы с самого начала выберем $m = k$. Но как нам это сделать?

Давайте просто перебирать m, начиная с маленького. Если вдруг во время выполнения на m + 1 шаге Джарвис еще не построил выпуклую оболочку, значит, $m < k$ и нам надо взять его побольше.

Но как перебрать m достаточно быстро, и при этом не переборщить на последнем шаге? Давайте возьмем начальный $m = 2$ и на каждом шаге перебора будем возводить его в квадрат. Иными словами, $m = 2^{2^t}$, и t перебирается от 0 до $\lceil \log \log k \rceil$

Докажем, что такой перебор не замедлит общее время работы:

$$\sum t = 0^{\lceil \log \log k \rceil} O\left(n \log(2^{2^t})\right) = O(n) \sum_{t=0}^{\lceil \log \log k \rceil} O(2^t) = O\left(n \cdot 2^{1+\lceil \log \log k \rceil}\right) = O(n \log k)$$

Итак, мы получили алгоритм с гарантированным временем работы $O(n \log k)$.

7.5 2 QuickHull

Как QuickSort, только QuickHull.

1. Возьмем крайние по иксу точки (они точно войдут в оболочку), обозначим их как p_0 и p_1
2. Разобьем множество на точки, лежащие ниже и выше прямой p_0p_1 (посвопаем 2 указателями, как в квиксорте)
3. Для верхнего множества найдем самую удаленную от p_0p_1 точку - q_1
4. Выкинем все точки, лежащие внутри треугольника $p_0p_1q_1$
5. Разделим оставшиеся точки на S_1 - лежащие выше p_0q_1, и S_2 - лежащие выше q_1p_1.
6. Рекурсивно повторим пункт 3 для S_1 и S_2.
7. Повторим пункт 3 для нижнего множества.
8. Объединим верхнюю и нижнюю оболочки

Утверждается, что для случайного набора точек этот алгоритм отработает за $O(n \log n)$ Понятно, что в худшем случае алгоритм отработает за $O(n^2)$ - мы можем построить такой выпуклый многоугольник, что на шаге 4 никогда ничего не будет выкинуто, а на шаге 5 в S_1 будут входить все оставшиеся точки.

Докажем, что для случайно разбросанных точек алгоритм отработает за $O(n \log n)$

WARNING: ЭТО ГОВНО Я ПРИДУМЫВАЛ САМ (почти)

Пусть время, необходимое для нахождения оболочки над некой прямой и множеством точек S есть $T(S)$ Тогда $T(S) = O(|S|) + T(S_1 \in S) + T(S_2 \in S)$, где S_1 и S_2 из пункта 5.

За $O(|S|)$ мы находим самую удаленную от прямой точку q_1. Заметим, что вообще все рассматриваемые точки находятся в прямоугольнике, ограниченном прямой p_0p_1 снизу, и вершиной q_1 сверху. Заметим также, что треугольник $p_0q_1p_1$ занимает половину площади этого прямоугольника. Это значит, что при равномерном распределении точек внутрь треугольника

попадет примерно половина всех точек. А значит, количество рассматриваемых точек на следующем шаге рекурсии будет меньше в 2 раза. Значит, всего шагов рекурсии будет $O(\log n)$, что в итоге дает оценку $O(n \log n)$.

7.6 0 Оболочка многоугольника

7.7 0 Оболочка полилинии

8 2 6: Динамическая выпуклая оболочка

(CH_DYN_1)

Начнем с подзадачи: пусть у нас есть две каких-то верхних оболочки в $\mathbb{R}^2$, разделенных по иксу. Мы хотим объединить эти верхних оболочки, проведя касательную сверху. Как такую касательную построить? (inb4 такая существует, потому что "палка сверху падает на холмики"). Как искать такую касательную за логарифм?

Очевидно, что касательная не проходит по экстремальным точкам (нарисуем большой холмик и рядом маленький).

Если мы хотим за логарифм, то че делать?

(CH_DYN_2)

Предположим, что есть пара точек на холмах. Будем типа пользоваться некоторым подобием бинпоиска на двух холмах сразу – четыре границы одновременно. Ну, два массивчика – это два множества точек для двух оболочек, отсортированных по иксу.

(CH_DYN_3) описывает классификацию всех попаданий касательной к кускам выпуклой оболочки для левой и правой кучи. Эта классификация важна, так как по ней мы будем определять текущее состояние. Как эти состояния отличать, понятно – считаем повороты. Случаи с двумя точками по одну сторону классифицируются поворотом.

(CH_DYN_4)

Рассмотрим случай A в CH_DYN_2. Рассмотрим прямую l и какую-то касательную к левой куче. Утверждается, что если мы будем поворачивать касательную вокруг точки касания, поворачивать вниз, то пересечение касательной и l как точка, будет опускаться вниз. Короче случай A распознается так: это случай слева а), а справа г). Тогда мы можем отрезать нижние куски выпуклых оболочек.

Проверка на два случая делается за $2 \times 2 = 4$ поворота.

Рассмотрим остальные случаи, например B в CH_DYN_2. В этом случае мы можем откинуть нижнюю часть правой оболочки. Симметричный случай тоже очевиден.

Случай с двумя касательными тоже распознается однозначно и есть ответом.

(CH_DYN_5)

Пусть на правом холме у нас касательная, а на левом точка из случая а) – CH_DYN_5 A. Тогда на левом холме мы можем откусить нижний кусок, а на правом – левый нижний от касательной. Симметрично тоже.

CH_DYN_5 B тоже так решается, то есть можно слева откусить нижний, а справа нижний левее точки касания.

(CH_DYN_6)

Теперь рассмотрим самый нетривиальный случай: пусть слева б), а справа д). Рассмотрим пересечение прямых l_1 и l_2. Прямые проведем через текущие вершины и следующие выше. Проверим точку L пересечения l_2 и l_2. Тогда если прямая L лежит полностью в интервале между холмами, то можем выкинуть и у левого и у правого нижние куски. Если точка L лежит в левом холме (левее самой правой точки левого холма), то мы выкидываем весь нижний кусок только левого холма вместе с этой точкой. Аналогично с правым холмом.

Теперь мы умеем решать задачу найти касательную двух верхних полуоболочек.

Тут Славик рассказал способ найти касательную точки и многоугольника с помощью подразбиения многоугольника на подмногоугольники (каждый вложенный берет точки предыдущего через одну). Потом он типа ищет для самого вложенного треугольника касательную, а потом передвигается к более богатым многоугольникам, сдвигая касательную влево или вправо

на одну вершину. Тоже алгоритм за log(n). Типа на каждом шаге есть step, мы рассматриваем текущего кандидата на касательную + step и -step. Выбираем лучшего, переходим к нему и делим шаг на два.

А как найти все четыре касательные для двух выпуклых множеств? Можно разбить на несколько и сведем к предыдущей задачи. Без этого? Нетривиальненько.

Теперь мы хотим честного итеративного построения. Можно хранить оболочки skip-листом и вместо бинпоиска просто спускаться на нижний уровень и ходить там. Вот мы идем по какому-то уровню, берем вершинку. Вдруг мы поняли, что нужно отрезать левую часть листа. Пойдем вправо. Спускаемся вниз, если нужно пойти в какую-то сторону, а та вершина уже "отрезана".

(`CH_DYN_7`)

Пусть есть оболочка, являющаяся общей частью двух оболочек. Типа дана оболочка, есть указатель на точку, по которой нужно разделиться. Причем у нас есть синяя и красная (карандашом) часть. Тогда мы можем фактически сделать две оболочки – это за 2 * logn для объединения двух скиплистов.

А как вообще все хранить, чтобы было итеративно? Будем хранить дерево, в котором листья – наши точки, а другие узлы – это верхняя оболочка сыновей. Это n*logn памяти, а хотим меньше. Причем неочевидно, как делать удаление. Как добавить? Прокинуть вершину вниз и перестроить все оболочки вверх во время просеивания. Если дерево нужно балансировать, то тоже нормально – перестроим что-нибудь.

Можно, формально, хранить немного не так: в самом верхнем узле будет храниться честная выпуклая оболочка всех точек. А в не верхнем, будем хранить только ту часть выпуклой оболочки, которая не является общей с родителем. Ну, типа, как раз синяя или красная часть. Тогда при продавливании точки вниз все проще: разбиваем текущую выпуклую оболочку (сначала корневую), объединяем за $\log n$ с чилдами. Определяем, куда кидать точку – влево или вправо. На одну часть забиваем. Так проходим вниз и добавляем вершинку. Заметим, что теперь уже не нужно хранить ничего в листах, так как два соседних листа однозначно определяются оболочкой в их паренте. Дальше строим оболочку и просеиваем вверх. Типа двух братьев берем, объединяем, отдаем паренту оболочку, себе оставляем только те части, которые не входят в парента.

Итого мы умеем удалять и добавлять вершинки за $\log^2 n$

Антон решил пояснить за то, как нужно делать мердж skip-листов. Лист мы держим сверху за вершину самого высокого уровня. Сплит: дали нам вершинку, нашли ее в самом нижнем уровне. Удаляем, обрезаем. Идем влево, пока не можем подняться наверх, поднимаемся, делаем вершинку терминальной, и так до верхнего уровня. Аналогично идем вправо и делаем ее первой. Мердж делается так же, про асимптотику думать не нужно.

9 0 7: Трехмерные выпуклые оболочки (CHN)

Немножко модифицируем quickhull на плоскости, чтобы можно было очевидно его перенести в n-мерное пространство. Quickhull не работает хорошо с детерменированной прямой.

Давайте выберем прямую L_1L_2. Зафиксируем в надмножестве случайную точку A. Все точки, которые попали в L_1AL_2 выкидываем. Рассмотрим все точки, которые не попали внутрь. Подразобьем их лучами L_1A и L_2A. Типа будем выбирать случайные точки вверху и продолжать выпуклую оболочку.

Для каждого разбиения мы перебираем все точки и для каждой мы запоминаем грани, которые видно.

Че делать в n-мерном пространстве? Возьмем произвольный тетраэдр. На самом деле лучше брать максимально большой тетраэдр. Потом для каждой новой случайной точки мы понимаем, к какой гране он принадлежит, какие грани эта точка видит.

10 З 8: Триангуляция (существование и ушная триангуляция)

Читать на вики.

Определение (триангуляция)**.** *Разбиение многоугольника на множество треугольников, внутренние области которых попарно не пересекаются.*

Определение (простой многоугольник)**.** *Многоугольник без самопересечений.*

Теорема 2 (О существовании триангуляции многоугольника)**.**
У любого простого многоугольника P с n вершинами всегда существует триангуляция, причем количество треугольников в ней равно $n-2$.

Доказательство. По индукции. Для $n = 3$ все понятно. Для больших n берем самую левую вершину v. Тогда либо ребро между ее соседями, либо между ней самой и самой дальней вершины от соседей – диагональ. Она поделит исходный n-угольник на два меньшего размера ($|P_1| + |P_2| = n + 2$), у которых по индукции существует триангуляция. По индукции P_1 и P_2 поделятся на $m_1 - 2$ и $m_2 - 2$ треугольников соответственно, так что в исходном n-угольнике будет $(m_1 - 2) + (m_2 - 2) = n - 2$ треугольника. □

Алгоритм примитивной триангуляция за $O(n^4)$: переберем $O(n^2)$ возможных диагоналей, за $O(n)$ проверим, пересекает ли она внутренние ребра. Повторим это $n-3$ раза. Итого $O(n^4)$.

Определение (ухо)**.** *Вершина многоульника v_i называется ухом, если диагональ $v_{i-1}v_{i+1}$ лежит строго во внутренней области многоугольника.*

Теорема 3 (о существовании двух ушей в многоугольнике)**.**
У любого простого многоугольника P с n вершинами всегда существует два не пересекающихся между собой уха.

Доказательство. Индукции. Для $n = 4$ все понятно. Для больших n возьмем произвольную вершину v. Два случая:

- v – ухо. Отрежем его, получим $n-1$-угольник, в котором, по индукции, есть два непересекающихся уха. Они также являются ушами исходного n-угольника, поэтому теорема верна.
- v – не ухо. Значит, треугольник $prev(v); v; next(v)$ содержит вершины P. Как и в теореме о существовании триангуляции, выберем наиболее ближнюю к v вершину, поделим P на P_1 и P_2 по диагонали, у P_1 и P_2 по индукции есть два уха – все хорошо.

□

Алгоритм (ушная триангуляция за $O(n^2)$): как в лабе писали короче: пройдемся по всем вершинам и за $O(n)$ проверим их на уховость. Если ухо - отрежем. На уховость проверяем за $O(n)$ по определению. Итого $O(n^2)$.

11 1 9: Триангуляция с заметающей прямой

Также известен как монотонный метод. Читать на вики.

11.1 Определение (монотонный многоугольник)

Многоугольник P называется монотонным относительно прямой l, если любая l' $\perp$ l пересекает стороны P не более двух раз.

11.2 Определение (y-монотонный многоугольник)

Многоугольник, монотонный относительно оси Y.

11.3 Определение (start, end, split, merge и regular-вершины)

Пусть ϕ - внутренний угол при вершине. Тогда назовем вершину: Start - если два ее соседа лежат ниже ее самой и $\phi < \pi$ Split - если два ее соседа лежат ниже ее самой и $\phi > \pi$ End - если два ее соседа лежат выше ее самой и $\phi < \pi$ Merge - если два ее соседа лежат выше ее самой и $\phi > \pi$ Regular - если один сосед лежит выше, а другой ниже ее самой

11.4 Лемма (достаточное условие y-монотонности)

Если в многоугольнике нет split- и merge-вершин, то он y-монотонен.

Доказательство: контрапозиция. Покажем, что не y-монотонный многоугольник содержит либо merge, либо split вершину. Дальше на викиконспектах все понятно.

11.5 Алгоритм (разбиение на монотонные части)

Будем избавляться от split- и merge-вершин, проводя из них диагонали. Пойдем горизонтальной заметающей прямой сверху вниз и, встречая split/merge-вершину, будем проводить диагонали до ближайшей от прямой вершины. TODO : разобраться подробнее + корректность

11.6 Алгоритм (триангуляция монотонного многоугольника)

KW : стек нетриангулированных вершин, свойство перевернутой воронки Разобраться в остальном.

12 0 10: Полуплоскости и выпуклые оболочки

13 0 11: Пересечение множества отрезков

14 0 12: PSLG и DCEL

15 0 13: PSLG overlaying

16 0 14: Локализация в PSLG

17 0 15: Трапецоидная карта

18 0 16: Вращающиеся калиперы

19 0 17: Сумма Минковского

20 2 18: Вероятностный алгоритм мин. охва. окружности множества точек

Рассмотрим задачу: У нас есть множество $P = \{p_1, ..., p_n\}$ точек на плоскости. Нужно построить окружность такую, чтобы все точки из P лежали бы внутри нее или на границе, причем из таких окружностей надо выбрать минимальную.

Идея Строим окружность итеративно, рассматривая точки по одной. В этом нам очень поможет следующая

Лемма 1 (О добавлении точки в минимальную окружность)**.** *Определим $P_i = \{p_1, ..., p_i\}$, а $D_i% - мин. охват. окружность для P_i. Рассмотрим точку p_i. Верно следующее:*

1. *Если $p_i \in D_{i-1}$, то $D_i = D_{i-1}$*
2. *Иначе p_i лежит на границе D_i*

Доказательство. Докажем эту лемму, как следствие следующей (по сути, следующая - это переформулировка этой) □

Лемма 2 (О точках, лежащих внутри и на границе)**.** *Пусть P - множество точек на плоскости, R - тоже (возможно, пустое) Обозначим как $md(P, R)$ наименьшую окружность, охватывающую P и имеющую все точки R на границе. Пусть $p \in P$. Тогда:*

1. *Если $md(P, R)$ существует, то он единственен.*
2. *Если $p \in md(P \setminus \{p\}, R)$, то $md(P, R) = md(P \setminus \{p\}, R)$*
3. *Если $p \notin md(P \setminus \{p\}, R)$, то $md(P, R) = md(P \setminus \{p\}, R \cup \{p\})$*

Доказательство.

1. Если $|R| > 2$, то это очевидно невозможно - потому что по 3 точкам окружность строится единственным образом. Пусть тогда $|R| > 2$ и существуют 2 минимальные окружности D_0 и D_1 с радиусом r и центрами x_0 и x_1 соответственно.

   Тогда $P \subset D_0 \cap D_1$, q_0 и q_1 - точки пересечения D_0 и D_1, и $R \subset \{q_0, q_1\}$. Но если мы построим окружность с центром точно посередине q_0 и q_1, она будет включать в себя $D_0 \cap D_1$ и на ее границе будет лежать R И по построению ее радиус будет меньше, чем r. Значит, D_0 и D_1 не являются минимальными охватывающими окружностями.

2. Очевидно.

3. Обозначим $D_0 = md(P \setminus \{p\}, R)$ и $D_1 = md(P, R)$. Это две окружности, очевидно, они гомотопически эквивалентны. Обозначим их центры и радиусы как x_0, r_0, x_1 и r_1 соответственно.

   Построим между ними кратчайшую гомотопию следующим образом: $D(\lambda) = \{x(\lambda), r(\lambda)\}$, где $x(\lambda) = (1 - \lambda)x_0 + \lambda x_1$, $r(\lambda) = \|z - x(\lambda)\|$, где z - одна из точек пересечения D_0 с D_1

   (Замечание: точки пересечения всегда есть, когда R непусто, а если оно пусто, то они должны быть из соображений минимальности)

   Очевидно, что $\forall \lambda \in [0, 1] : P \subset D(\lambda), R \subset \partial D(\lambda)$

   - ведь это верно для пересечения D_0 и D_1, которое по построению в себя включает каждая из $D(\lambda)$

   Тогда существует некая $\lambda*$, $0 < \lambda* \leqslant 1$, такая, что $p \in \partial D(\lambda)$ Но по построению $r(\lambda) \leqslant r_1$, и если $\lambda* < 1$, то D_1 не является $md(P, R)$, так как ей является $D(\lambda)$. Противоречие! Значит, $\lambda* = 1$, из чего следует, что $p \in \partial D_1$, что и требовалось доказать.

□

Алгоритм Напишем 3 функции. Первая функцию - `make0` - будет делать то, что нужно:

21 0 19: Граф видимости и планирование движения

Задача поставлена следующим образом: есть объект, точечный или нет, нужно провести его через полигональные препятствия (все в $\mathbb{R}^2$). Известность карты – тоже входной параметр.

Решим задачу для точечных объектов. Пусть у нас есть поле, точки A и B. Нужно попасть из первой во вторую, оптимально.

Первая тривиальная идея, которая приходит в голову – это построить граф, в котором узлы – это вершины полигонов, составляющих карту, а ребра между двумя вершинами u, v строим в том случае, если uv не пересекается ни с одним полигоном из данных. Такой граф называется картой видимости. Можно его обойти дейкстрой. Получаем $O(n^2)$ и памяти и времени на запрос (если использовать дейкстру без кучи). Предподсчет будет занимать втупую $O(n^3)$, то есть для каждой пары точек проверить пересечение со всеми отрезками полигонов (их n штук).

Подумаем, что с этим можно сделать:

- Не хранить ребра, а создавать их только когда мы пришли в вершину.
- Оптимальный путь – ломаная (доказательство от противного, пусть есть какая-то кривая, огибающая препятствие, тогда спрямим ее, получим прямую меньшей длины).
- Если рассмотреть вершину полигона P, то путь из двух ребер (входящее в нее aP и исходящее Pb) неоптимален, если угол $aPb < 180$. Доказательство простое – рассмотрим такой угол. Тогда возьмем две любые точки $c \in aP$, $d \in Pb$, получим по неравенству треугольника что путь $acdb$ короче чем aPb. Такие ребра в общем можно не добавлять. Алсо такая оптимизация не понижает асимптотику, а только уменьшает константу.
- Препроцессинг можно уменьшить с $O(n^3)$ до $O(n^? \log n)$ с помощью алгоритма Бентли-Оттмана (заметающая прямая). Мы будем использовать следующую модификацию:

  Для очередной точки P найдем, какие отрезки из нее исходят вправо (предполагаем, что все отрезки влево уже были добавлены на предыдущем шаге). Для этого рассмотрим все ребра, которые пересекают прямые, начиная от P и вниз и вправо против часовой стрелки вверх на 180. Типа рассмотрели зону видимости "вправо". Формально мы все отрезки, которые заканчиваются правее нашей точки берем, раскладываем на события (стандартные Б-О ивенты типа начало отрезка, пересечение, конец отрезка) и сортим по углу поворота относительно P. Дальше перебираем их всех против часовой стрелки и храним стейт всех отрезков, которые пересекает наша прямая. Первый отрезок в стейте будет отрезком, который "виден"из P – будем по ходу дела добавлять концы видимых отрезков в ответ.

  Сам Б-О работает за $O((n+k)\log n)$, где n – количество отрезков, а k – количество пересечений. Таким образом для всех точек оцениваем сверху препроцессинг до $O(n^2 \log n)$.
- Препроцессинг на самом деле уменьшается до $O(n^2)$, но это древняя магия (есть какие-то статьи).
- Динамически отвечать на запросы, чтобы снизить память. Я так понимаю, это когда мы минимум выбираем (за $O(n)$), то ищем вершины, потом релаксируем, храним предка чтобы знать путь. Вот короче динамически так будем строить граф.
- Есть много других интересных подходов, в том числе алгоритм Митчелла.

22 0 20: Триангуляция Делоне

23 0 21: Доказательство (алгоритм + корректность)

24 2 22: Диаграмма Вороного

Статья на викиконспектах

24.1 Алгоритм и асимптотика

Антону больше нравится инкрементальный алгоритм построения диаграммы Вороного, так как он похож на Делоне.

Типа вот есть бакеты, мы там чето меняем, проводим $O(1)$ времени на каждом уровне, суммарно получается $O(n)$.

У нас есть $O(log(n))$ уровней, где есть какие-то сабсеты, для каждого мы можем построить за $O(1)$ новую диаграмму.

Как локализоваться в диаграмме Вороного, где точек $O(1)$? Тупо найти ближайшую точку, посчитав метрику.

`VOR_0` Как с помощью $n+1$ уровня найти ближайшую точку на n-м уровне? X — ближайшая точка на $n+1$ уровне. A — точка, которую мы хотим вернуть, то есть ближайшая к q

на \$n\$-м уровне. Проведем отрезок XA и проверим все соседние грани точки X, выберем ту, которую пересекает XA. XA также может пересекать какую-то точку триангуляции. Тогда нужно перебрать все соседние прямые, исходящие из этой точки и выбрать такие две, между которыми проходит XA.

Как достроить диаграмму Вороного, если мы уже локализовались? Построим между q и A серединный перпендикуляр, пересечь его с фейсом вершины A. Будем дальше идти по соседним DCEL'ам и заворачивать, строя серединные перпендикуляры, прямые вокруг q. Таким образом, построим грань для вершины q.

Асимптотика (inb4 можно это делать, строя двойственную триангуляцию):

- Вставка: посчитаем среднюю степень, проведем регрессионный анализ, как в алгоритме Делоне.
- Локализация: пересечем $O(1)$ ребер. Это доказательство тоже копируется с Делоне. Можно сказать, что мы пройдем по количеству DCEL'ов которые не добавились на более высокий уровень. Поскольку слои диаграммы — это множество Бернулли, то на каждом шаге мы добавим не больше чем сколько-то точек, а они экспоненциально убывают.

24.2 Удаление из диаграммы Вороного

`VOR_2`

Возьмем сайт, его фейс. Будем строить типа straight skeleton, двигая стороны внутрь по серединным перепендикулярам. Тогда в какой-то момент схлопнется.

24.3 Построение из триангуляции диаграмму

`VOR_1`

Как построить из триангуляции Делоне диаграмму Вороного? Возьмем диаграмму, выделим какую-то точку A. Построим серединные перпендикуляры для каждого ребра, исходяшего из A, пересечем их всех. Поймем, что получившееся пересечение сер. перпендикуляров образует ячейку Вороного.

Покажем, что такая ячейка конечна. Рассмотрим треугольник ABC. По определению, этот треугольник — треугольник Делоне, поэтому точка пересечение серединных перпендикуляров лежит внутри, и расстояние от S до точек прямоугольника минимально, если точка есть пересечение серединных перпендикуляров. Более того, по свойству Делоне, в окружности не лежит никаких других точек.

Любой отрезок ячейки Вороного принадлежит ей, потому что ячейки диаграммы Вороного выпуклые. Отсюда, поскольку точки отрезка лежат в ячейке, отрезок тоже лежит. Типа сама точка A лежит ближе всего к себе. Точка пересечения сер. перпендикуляров тоже лежит в ячейке, тогда для каждых двух соседних точек прямая между ними тоже лежит, т.к. ячейка вороного — выпуклый многугольник.

24.4 Построение из диаграммы триангуляции

Возьмем диаграмму Вороного и построим **разбиение** Делоне — то есть могут получиться не треугольники. В этом случае можно показать, что любой такой многоугольник можно триангулировать любым образом, при этом свойство Делоне останется.

Почему в общем случае граф, в котором мы соединили сайты соседних граней, получится разбиением Делоне? Ну типа, возьмем в DCEL'е узел, который соединяет три грани. Берем их сайты, соединяем. Это получится треугольник. Прогоним обратное следствие в нужную сторону.

24.5 Высшие порядки

Диаграмма Вороного второго порядка VD^2 это: $P_1, P_2 \in V_{q_1 q_2} \Leftrightarrow d(p_1, q_1) = min_1, d(p_1, q_2) = min_2$.

Аналогично строим VD^k — диаграмму Вороного k-го порядка.

Диаграмма Вороного $n-1$-го порядка — это набор таких сайтов, что для каждого есть $n-1$ точка, и для всех точек от них есть какая-то одна самая далекая.

Как строить инеркментально? Нужно проводить алгоритм удаления точек, но не удаляя прямые, которые мы двигаем, до самого конца.

Для каждого фейса мы делаем это за: $klogk$, но

$$\sum k \log k \leq \sum k \log n = \log n \sum k = O(n \log n)$$

Сколько будет вершин в диаграмме вороного второго порядка? Столько же, сколько и ребер, вернее удвоенное количество.

ЧИТАТЬ НА ВИКИ

24.6 Диаграмма минус первого порядка

Граф Делоне двойственный диаграмме -1 порядка — это верхняя крышка проекции диаграммы на параболоид.

Типа возьмем треугольник, тогда в окружности должны находиться все точки множества. Количество вершин в такой диаграмме Вороного — это количество вершин выпуклой оболочки.

25 Бонусные задачи

- Из множества прямых произвольных восстановить DCEL. Можно делать инкрементально. Для трех понятно, как строить. Дальше кидаем прямую. Берем первое пересечение, локализуем точку на прямой за $O(n)$, дальше обходим соседние фейсы DCEL'а пока не найдем точку пересечения нашей прямой с какой-то другой. И так пока все не пересечем. Можно показать, что асимптотика будет норм – $O(n^2)$.

  Рассмотрим `BON-O`. l – наша прямая. Утверждается, что от пересечения нашей прямой с какой-то другой до следующего пересечения нужно пробежать не более чем $O(n)$ ребер.

  В среднем заметим, что у нас $O(n^2)$ ребер и $O(n^2)$ фейсов. Тем не менее, из этого не следует, что на каждый фейс приходится $O(1)$ ребер, может быть так, что какие-то фейсы жирные, а какие-то нет.

  Мы всегда знаем направление, в котором мы будем двигаться от точки.

  Рассмотрим все Покрасим точки с положительным наклоном относительно прямой синим цветом, а серым – с отрицательным. Будем считать только те ребра, которые лежат в DCEL'ах, которые пересекает наша прямая l.

  Посчитаем, сколько таких цветных ребер есть (кстати, синих и серых ребер будет одинаковое количество).

  Прямая l прйдет через O(n) ребер. Выкинем самую правую прямую, которая имеет синий отрезок. Тут типа индукция. Тогда есть $\exists c$, что прямая пересекает не более $c * (n-1)$. Попробуем ее заново добавить. Пусть прямая имеет серый наклон вправо. Тогда такая прямая подразобьет не более чем $O(1)$ ребер.