

Вычислительная геометрия, день 1

Борис Золотов Ольга Карманова

21 июня 2023 г.

Летняя школа МКН СПбГУ

Содержание

О-нотация

Задачи, выполнимые за $O(1)$

Algorithms to get a feeling

Триангуляция и картинная галерея

Алгоритм с заметающей прямой

DCEL (de Berg et al., p. 29+10)

О-нотация

Идея O

Пусть некоторая программа принимает на вход данные. Она, скорее всего, работает тем дольше, чем больше данных поступило на вход. Мы хотим оценить время работы программы, сравнивая его с известными функциями $\mathbb{N} \rightarrow \mathbb{N}$.

Определение из анализа

Определение

Функции $f, g : \mathbb{R} \rightarrow \mathbb{R}$. Говорят, что

$$f(x) = O(g(x)), \quad \text{если} \quad \exists C \forall x \quad |f(x)| \leq C \cdot |g(x)|.$$

Определение для нас

Определение

Время работы программы составляет $O(f(n))$, если существует C : для входа *размера* n количество операций составит не более $f(n) \cdot C$.

- Какие операции считаются в данном определении, обычно известно из контекста.
- Если вход состоит из *одного числа*, размер входа равен 1, вне зависимости от величины числа.

Поиск наибольшего элемента в массиве

Задача

Дан массив из n чисел, найти в нём наибольший элемент.

Отсмотрим все элементы по порядку, храня наибольший
отсмотренный элемент. Мы обращаемся к каждому элементу
по разу, поэтому $O(n)$. Быстрее, очевидно, нельзя.

[1 4 4 2 8 1 6 9 8]



max to date: 8

Наличие элемента в отсортированном массиве

Задача

Дан массив M из n чисел, они упорядочены по возрастанию.

Дано число k — проверить, есть ли оно в M .

Сравним k и элемент в середине M — назовём его m .

Если $m > k$, ищем в первой половине массива. Иначе во второй.

[1 2 3 5 8 13 21 34 55]

здесь? \leftarrow | \rightarrow или здесь?

Как оценить время работы

Пусть $T(n)$ — время работы на массиве из n элементов. Тогда

$$T(n) = 1 + T\left(\frac{n}{2}\right).$$

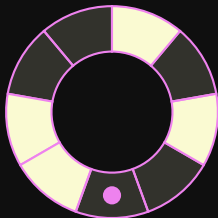
Отсюда

$$T(n) = O(\log_2(n)).$$

Количество комнат на космическом корабле

Задача

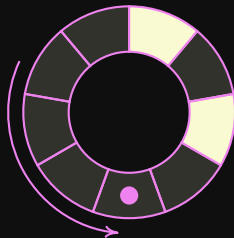
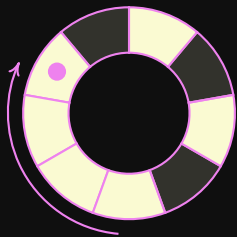
Космический корабль в форме кольца разделён на n одинаковых комнат, в которых горит или не горит свет. Можно переходить между соседними комнатами, включать или выключать свет. Найти n .



Простое решение

Процедура $\text{проход}(k)$ — пройти на k комнат вперёд, включая в каждой комнате свет (может быть что угодно).

Потом пройти на k комнат назад, выключая в каждой комнате свет (должен быть включен).



Оценка времени

Выполним проход(1), проход(2), проход(3), ...

В какой-то момент мы вернёмся, чтобы выключить свет в первой комнате, а он уже будет там выключен.

Значит, мы прошли полный круг.

$$2 \cdot (1 + 2 + \dots + n) = O(n^2) \text{ переходов.}$$

Решение за линейное время (*Doubling search*)

Пусть $2^{k-1} \leq n < 2^k$.

Выполним проход(1), проход(2), ..., проход(2^i), ...

Каждый раз длина прохода увеличивается в два раза.

В какой-то момент на обратном пути мы придём
в комнату, свет в которой уже выключен.

$$2 \cdot (1 + 2 + 4 + \dots + 2^k) = 2 \cdot (2^{k+1} - 1) \leq 8 \cdot n = O(n).$$

Алгоритм Беллмана — Форда

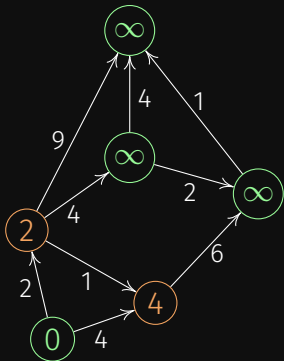
Задача

Дан граф $G = (V, E)$, на рёбрах расставлены неотрицательные веса. Выбрана $v \in V$, найти наименьшие веса путей от v до каждой из остальных вершин.

Изначально $d(v_i) = +\infty$ для всех v_i .

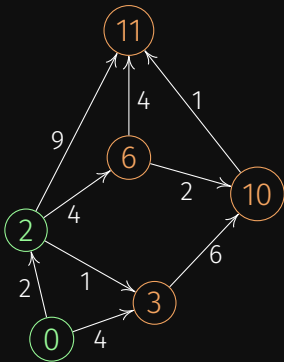
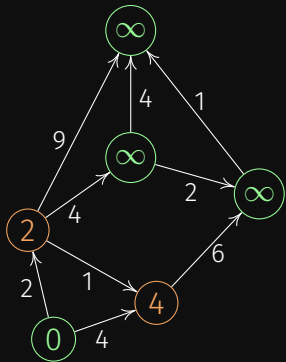
Идея

Считаем, что все $d(v_i)$ обновляются одновременно. В реальности дело чуть лучше.



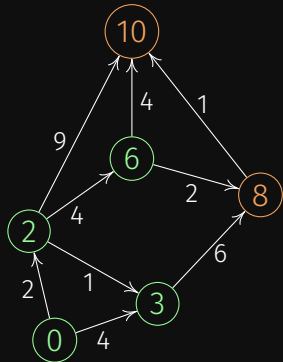
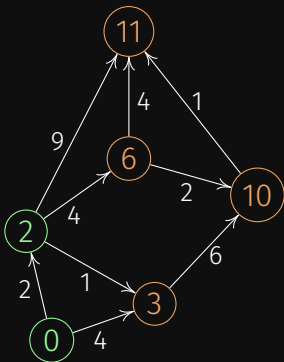
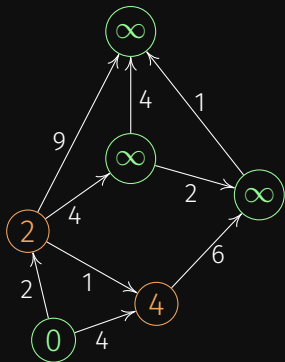
Идея

Считаем, что все $d(v_i)$ обновляются одновременно. В реальности дело чуть лучше.



Идея

Считаем, что все $d(v_i)$ обновляются одновременно. В реальности дело чуть лучше.



Улучшение $d(v_i)$

```
for  $i = 1$  to  $|V| - 1$  do
  for all  $e = (u_1, u_2) \in E$  do
    if  $d(u_1) + \text{weight}(e) < d(u_2)$  then
       $d(u_2) \leftarrow d(u_1) + \text{weight}(e)$     ▷ Пытаемся с помощью ребра  $e$ 
                                              улучшить путь в  $v_2$ 
    end if
  end for
end for
```

▷ Нашли оптимальные пути длины $\leq i$

Асимптотика с несколькими параметрами

Время работы оценивается как $O(|V| \cdot |E|)$.

Мы, конечно, знаем, что $|V| \leq |E| + 1$ и $|E| \leq |V|^2$,
но зачастую нам нужна точная оценка
относительно обоих параметров входа.

Замечания

- $\log_a n \ll n^k \ll a^n$

- $\log_a n \sim \log_b n$

- $\log_a (x^k) \sim \log_a x$

Задачи, выполнимые за $O(1)$

Некоторые договоренности

Для удобства мы будем считать, что точки **в общем положении**

- Никакие 3 из них не лежат на одной прямой.
- Никакие 4 из них не лежат на одной окружности.
- У них нет общих x и y координат.

В совокупности это верно почти всегда.

Задание фигур уравнениями

При работе с геом. объектами удобно задавать их уравнениями

- Прямая:

$$\ell: ax + by + c = 0 \text{ или } \ell: y = kx + b.$$

Прямая через точки $A(x_1, y_1), B(x_2, y_2)$:

$$\ell: \begin{cases} ax_1 + by_1 + c = 0 \\ ax_2 + by_2 + c = 0 \end{cases}$$

$$a = y_2 - y_1, \quad b = x_1 - x_2, \quad c = -(ax_1 + by_1)$$

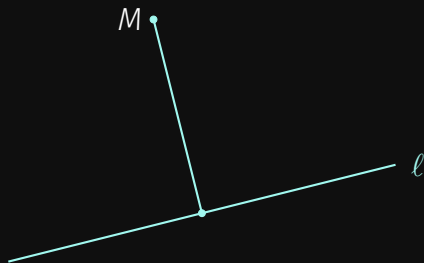
- Окружность:

$$(x - x_0)^2 + (y - y_0)^2 = R^2, \text{ где } (x_0, y_0) \text{ — центр, а } R \text{ — радиус}$$

Расстояние от точки до прямой

Если прямая задана как $\ell: ax + by + c = 0$, то расстояние от точки $M(x_0, y_0)$ до неё можно рассчитать как

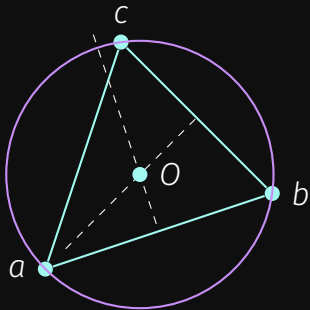
$$d(M, \ell) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$



Центр описанной окружности

Задача

Дана тройка точек a, b, c . Требуется найти центр описанной окружности $\triangle abc$.



Центр описанной окружности

1. Ищем серединные перпендикуляры к $[ac]$ и $[bc]$,
как прямые p_1 и p_2 .

Если $\ell_1: y = a_1x + b_1$, а $\ell_2: y = a_2x + b_2$, то $\ell_1 \perp \ell_2 \Leftrightarrow a_1 \cdot a_2 = -1$.

Находим серединный перпендикуляр к стороне как прямую,
проходящую через середину стороны и перпендикулярную ей.

2. Ищем их пересечение, как решение линейной системы:

$$p_1: y = \alpha_1x + \beta, \quad p_2: y = \alpha_2x + \beta_2.$$

$$\begin{cases} y = \alpha_1x + \beta_1 \\ y = \alpha_2x + \beta_2 \end{cases}$$

Смежные задачи

Ясно, что очень большое количество задач можно решить совершенно аналогичным образом, например, задачи нахождения

- инцентра.
- ортоцентра.
- барицентра.

Угол между векторами

Рассмотрим вектора $\vec{a} = (x_1, y_1)$ и $\vec{b} = (x_2, y_2)$.

$$\langle \vec{a}, \vec{b} \rangle = x_1 x_2 + y_1 y_2 = |\vec{a}| |\vec{b}| \cos \angle(\vec{a}, \vec{b}) = \sqrt{x_1^2 + y_1^2} \sqrt{x_2^2 + y_2^2} \cos \angle(\vec{a}, \vec{b})$$

Значит, мы знаем, как найти угол между векторами \vec{a} и \vec{b} :

$$\angle(\vec{a}, \vec{b}) = \arccos \left(\frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} \sqrt{x_2^2 + y_2^2}} \right)$$

Косое произведение

Определение

Косым произведением векторов $\vec{a} = (x_1, y_1)$ и $\vec{b} = (x_2, y_2)$ на плоскости будем называть

$$\vec{a} \wedge \vec{b} = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1$$

Покажем, что $\vec{a} \wedge \vec{b} = |\vec{a}| |\vec{b}| \sin(\angle(\vec{a}, \vec{b}))$, где $\angle(\vec{a}, \vec{b})$ — угол вращения против часовой стрелки от \vec{a} к \vec{b} .

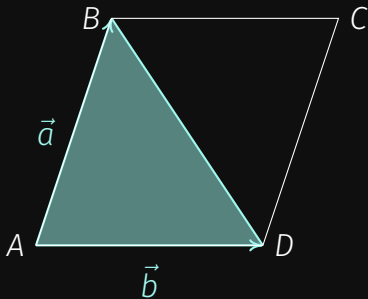
Эквивалентность определений

В самом деле, $\cos(\angle(a, b)) = x_1x_2 + y_1y_2/(\sqrt{x_1^2 + y_1^2}\sqrt{x_2^2 + y_2^2})$.

$$\begin{aligned}\sin(\angle(\vec{a}, \vec{b})) &= \sqrt{1 - \cos^2(\angle(\vec{a}, \vec{b}))} = \sqrt{1 - \frac{(x_1x_2 + y_1y_2)^2}{(x_1^2 + y_1^2)(x_2^2 + y_2^2)}} = \\ &= \sqrt{\frac{x_1^2x_2^2 + x_1^2y_2^2 + y_1^2x_2^2 + y_1^2y_2^2 - x_1^2x_2^2 - y_1^2y_2^2 - 2x_1x_2y_1y_2}{(x_1^2 + y_1^2)(x_2^2 + y_2^2)}} \\ &= \frac{x_1y_2 - y_1x_2}{\sqrt{(x_1^2 + y_1^2)}\sqrt{(x_2^2 + y_2^2)}} = \frac{\vec{a} \wedge \vec{b}}{|\vec{a}||\vec{b}|}\end{aligned}$$

Площадь треугольника

Теперь ясно, что $\vec{a} \wedge \vec{b} = 2S_{\triangle ABD}$, где площадь **ориентированная**.



Ориентация

Определение

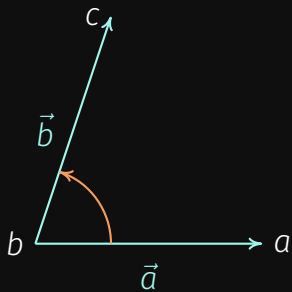
Будем говорить, что тройка точек (a, b, c) **положительно ориентирована** и писать $\text{sign}(a, b, c) > 0$, если поворот вектора \vec{ba} к вектору \vec{bc} осуществляется против часовой стрелки.

Замечание

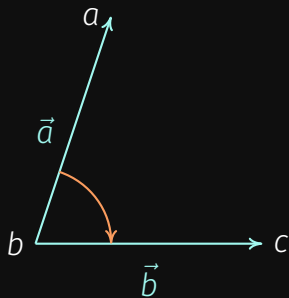
Ориентация тройки точек (a, b, c) совпадает со знаком кросс-произведения $\vec{ba} \wedge \vec{bc}$.

Ориентация

$$\text{sign}(a, b, c) > 0$$



$$\text{sign}(a, b, c) < 0$$



Пересечение отрезков

Задача

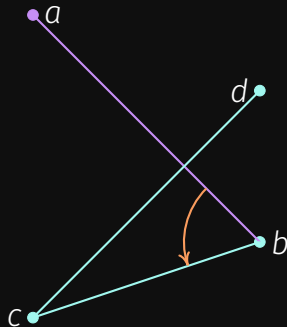
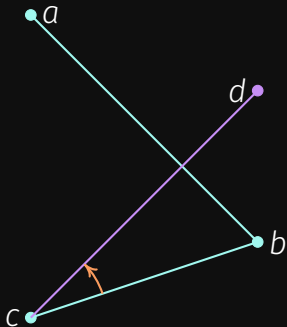
Дана четверка точек a, b, c, d . Требуется определить, пересекаются ли отрезки $[ab]$ и $[cd]$.

Заметим, что отрезки $[ab]$ и $[cd]$ пересекаются т. и т. т., когда

- Концы a, b лежат по разные стороны от прямой (cd) .
- Концы c, d лежат по разные стороны от прямой (ab)

Заметим, что точки a и b лежат по разные стороны от прямой (cd) т. и т.т., когда различны $\text{sign}(a, c, d)$ и $\text{sign}(b, c, d)$.

Пересечение отрезков



Пересечение отрезков

INTERSECT(a, b, c, d):

if $\text{sign}(a, c, d) = \text{sign}(b, c, d)$ then

 return FALSE

else

 if $\text{sign}(a, b, c) = \text{sign}(a, b, d)$ then

 return FALSE

 else

 return TRUE

 end if

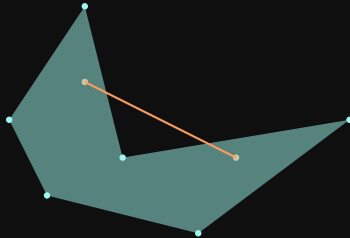
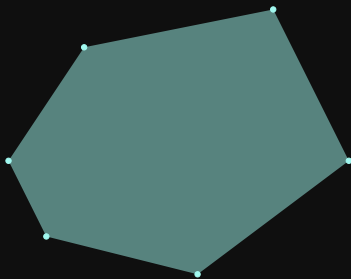
end if

Algorithms to get a feeling

Выпуклое множество

Определение

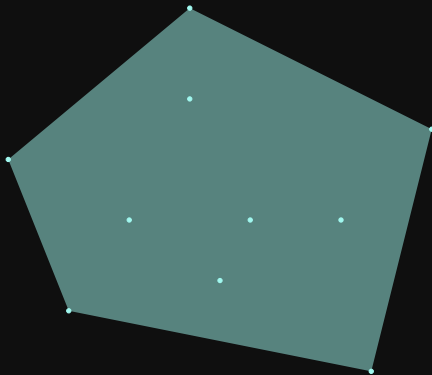
Множество S — **выпуклое**, если оно вместе с любыми двумя точками содержит отрезок между ними.



Выпуклая оболочка

Определение

Выпуклая оболочка $\mathcal{CH}(S)$ множества S — наименьшее выпуклое множество, содержащее S .



Вычисление выпуклой оболочки

Задача

Дано множество $S \subset \mathbb{R}^2$, $|S| = n$. Требуется найти координаты вершин его выпуклой оболочки $\mathcal{CH}(S)$ в порядке против часовой стрелки.

Вычисление выпуклой оболочки

Есть много алгоритмов вычисления выпуклой оболочки на плоскости. Большинство из них напоминают алгоритмы сортировок, к примеру

- Алгоритм Джарвиса — Selection Sort.
- Quick Hull — Quick Sort.
- Алгоритм «Разделяй и властвуй» — Merge Sort.

Алгоритмы «Разделяй и властвуй»

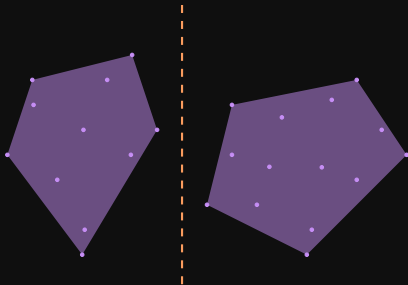
Из представленных выше мы рассмотрим алгоритм *Divide-and-conquer*.

Все алгоритмы «Divide-and-Conquer» имеют одну идею:

- Разбить задачу на подзадачи, от них вызываться рекурсивно.
- Научиться быстро сливать подзадачи.

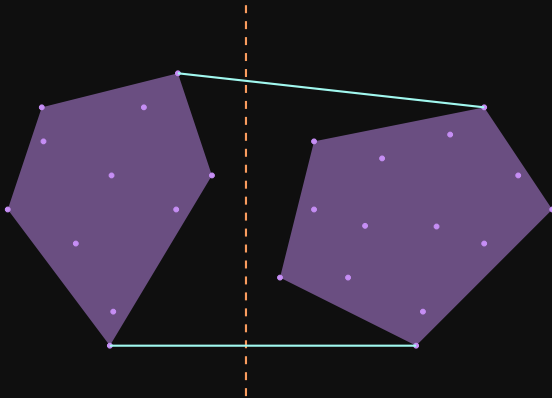
Алгоритм $D\&C$ для CH : описание

- $n \leq 3 \Rightarrow$ «brute force».
- $n \geq 4 \Rightarrow$ разбиваем S на два примерно равных подмножества по x -координате, вызываемся на них рекурсивно.



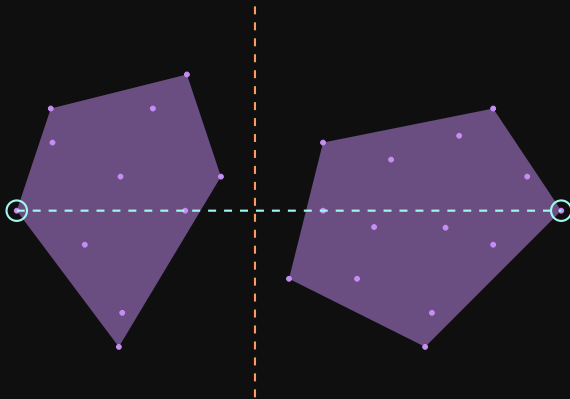
Алгоритм $D\&C$ для $СН$: слияние подзадач

Для слияния подзадач будем считать **верхнюю** и **нижнюю** касательные.



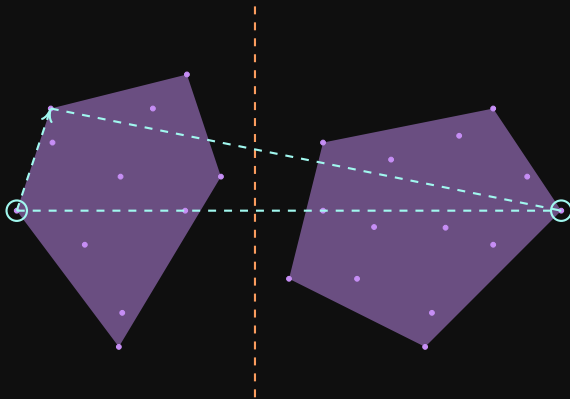
Алгоритм $D\&C$ для CH : верхняя и нижняя касательные

Идея вычисления: поднимаем тот конец отрезка, который можем поднять.



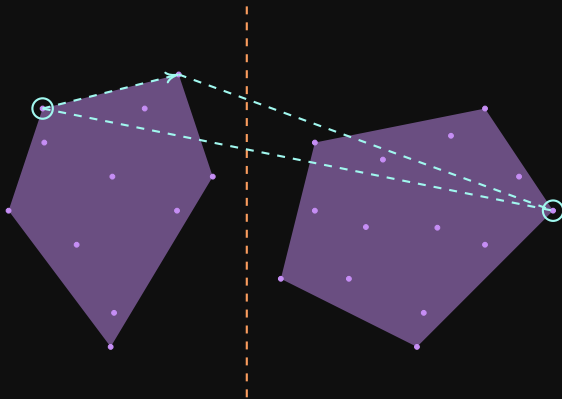
Алгоритм $D\&C$ для CH : верхняя и нижняя касательные

Идея вычисления: поднимаем тот конец отрезка, который можем поднять.



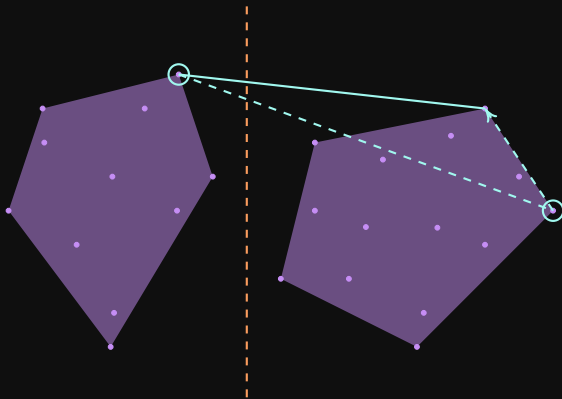
Алгоритм $D\&C$ для CH : верхняя и нижняя касательные

Идея вычисления: поднимаем тот конец отрезка, который можем поднять.



Алгоритм $D\&C$ для CH : верхняя и нижняя касательные

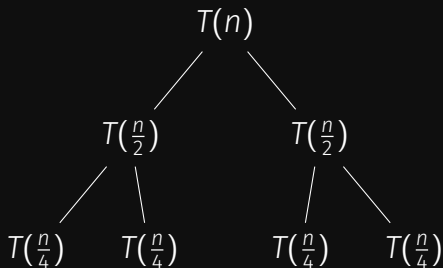
Идея вычисления: поднимаем тот конец отрезка, который можем поднять.



Алгоритм $D\&C$ для \mathcal{CH} : оценка времени работы

Количество точек в подзадаче сокращается хотя бы в два раза, на слияние мы тратим линейное время

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$



Алгоритм $D\&C$ для \mathcal{CH} : оценка времени работы

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Распишем это:

$$T(n) = O\left(n \cdot \left(\frac{2}{2} + \left(\frac{2}{2}\right)^2 + \dots + \left(\frac{2}{2}\right)^{\log_2(n)}\right)\right) = O\left(n \cdot \sum_{k=1}^{\log_2 n} 1\right) = O(n \log_2(n)).$$

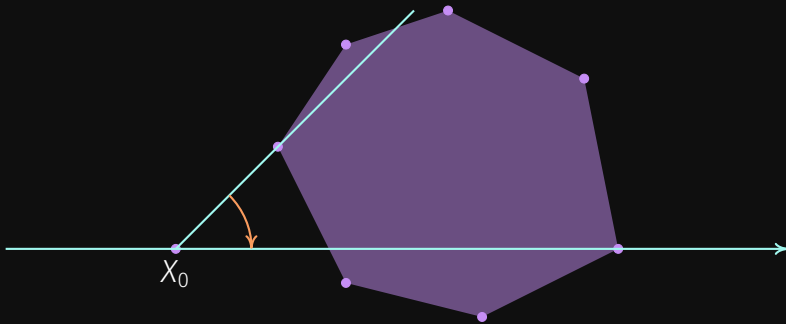
Касательная к выпуклому многоугольнику

Задача

Дан выпуклый многоугольник $\mathcal{P} \subset \mathbb{R}^2$ и точка $X_0(x_0, y_0) \in \mathbb{R}^2$. Найти касательную к \mathcal{P} из точки $X_0(x_0, y_0)$.

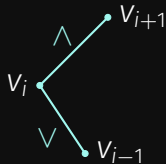
Касательная к выпуклому многоугольнику: алгоритм

Каждой вершине сопоставим *аргумент* — угол, под которым она видна из точки X_0 .

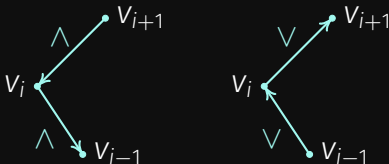


Касательная к выпуклому многоугольнику: алгоритм

У нужной вершины v_i аргументы будут расположены так:

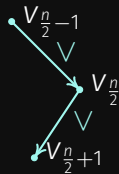
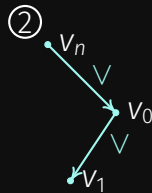
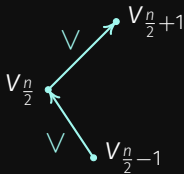
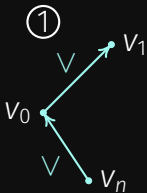


Как понять, с какой стороны нужная вершина:

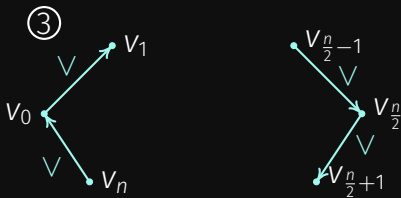


Касательная к выпуклому многоугольнику: алгоритм

- Выберем две вершины на противоположных сторонах — v_0 и $v_{\frac{n}{2}}$, посчитаем аргументы их и их соседей.
- Получим один из случаев:



Касательная к выпуклому многоугольнику: алгоритм



- Каждый раз отбрасываем половину вершин.
- После отбрасывания половины добавляем вершину посередине нового промежутка и продолжаем.

Кас. к вып. многоуг.: время работы

Каждый следующий запрос — $O(1)$ проверок ориентации. Всего запросов — $O(\log(n))$, а значит, время работы — $O(\log(n))$.

Упражнение.

Придумайте алгоритм выпуклой оболочки, использующий алгоритм построения касательной и алгоритм $D\&C$. Время работы алгоритма должно составлять $O(n \log(h))$, где h — количество вершин в выпуклой оболочке.

Триангуляция и картинная галерея

Лемма о триангуляции

Лемма (О триангуляции)

Всякий многоугольник можно диагоналями разбить на треугольники, причем полученный граф красится в 3 цвета.

Доказательство.

Индукция по числу вершин.

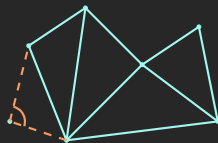
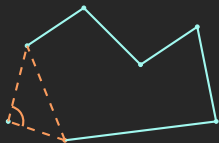
База: $n = 3$.

Переход: Находим вершину с углом $< 180^\circ$.

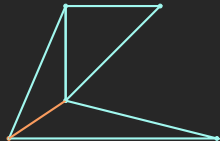
- Отрезок между соседними вершинами лежит в многоугольнике: Отрезаем вершину, красим по индукции, возвращаем, красим в свободный цвет.

Лемма о триангуляции: доказательство

Доказательство.



- Отрезок между соседними вершинами не лежит в многоугольнике:

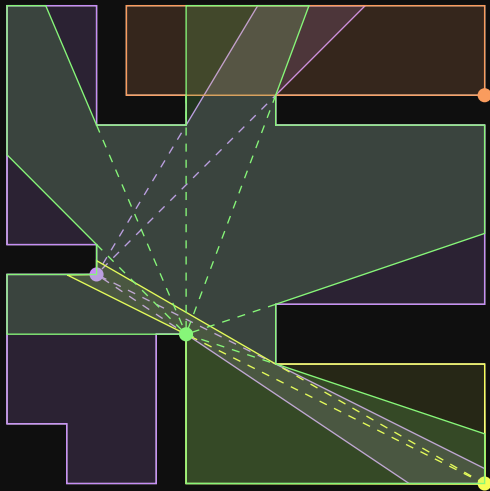


Задача о картинной галерее

Задача

Дана картинная галерея, план которой — многоугольник без самопересечений. Какое минимальное число охранников нужно поставить в точках галереи, чтоб они просматривали каждую точку?

Задача о картинной галерее: иллюстрация



Задача о картинной галерее

Теорема (Хватал)

Для произвольного n -угольника достаточно $\lfloor \frac{n}{3} \rfloor$ охранников, поставленных во внутренних точках, чтобы охранникам были видны все внутренние точки n -угольника.

Доказательство

По лемме строим разбиение галереи на треугольники, вершины раскрашиваются в три цвета. Из этих цветов выбираем тот, который встречается *не чаще других*. Ставим охранников в вершины, окрашенные этим цветом.

Алгоритм с заметающей прямой

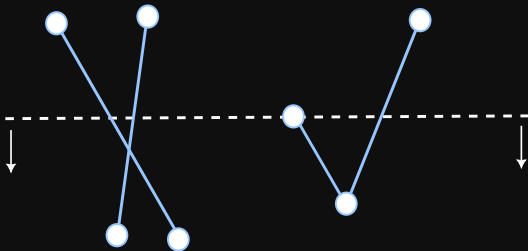
Постановка задачи

Дано множество отрезков на плоскости,
найти всевозможные их пересечения $\{x_i\}$
и вместе с каждой точкой x_i — множество отрезков,
пересекающихся в ней.

Заметающая прямая

Запустим горизонтальную *sweep line*, начинающую свое движение *над* всеми отрезками входа.

(Разумеется, мы не будем реализовывать *непрерывное* движение)



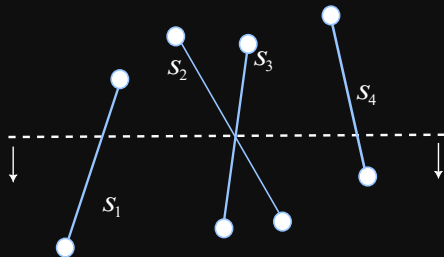
Event points

Вершины, соответствующие концам отрезков,
будем называть *крайними событиями*,
а их пересечениям — *внутренними событиями*.
Упорядочим события следующим образом:

$$p \prec q \iff \begin{cases} p_y > q_y \\ p_y = q_y, p_x < q_x \end{cases}$$

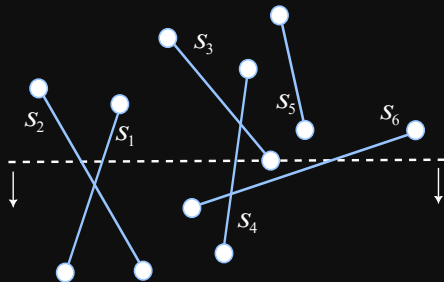
Статус

Статусом заметающей прямой будем называть последовательность отрезков, пересекающих её в данный момент.



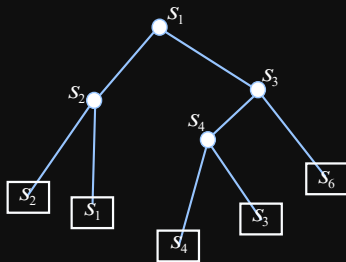
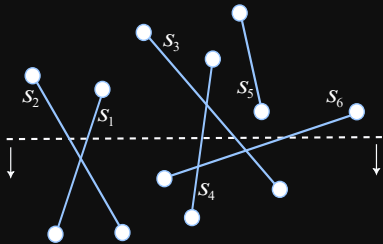
Статус

Статусом заметающей прямой будем называть последовательность отрезков, пересекающих её в данный момент.



Бинарное дерево статуса

Для хранения статуса используем бинарное дерево T . Оно понадобится нам для нахождения ближайших к событию отрезков.



Общий алгоритм

- создаем пустую очередь Q и заполняем ее крайними событиями (если событие — верхняя вершина отрезка s , то кладем s вместе с ней)
- создаем пустое дерево статуса
- пока $Q \neq \emptyset$, определяем, какое событие будет следующим, удаляем текущее и обрабатываем следующее

Обработка события

$U(p)$ — отрезки, верхняя вершина которых p

$L(p)$ — отрезки, нижняя вершина которых p

$C(p)$ — отрезки, содержащие p внутри

if $U(p) \cup L(p) \cup C(p) \neq \emptyset$ **then**

p — пересечение набора отрезков $U(p) \cup L(p) \cup C(p)$

end if

удаляем отрезки из $L(p) \cup C(p)$

добавляем отрезки из $U(p) \cup C(p)$

Обработка события

if $U(p) \cup C(p) = \emptyset$ then

находим соседей p с помощью $T - s_l, s_r$

FindNewEvent(s_l, s_r, p)

else

$s' :=$ крайний левый в $U(p) \cup C(p)$

$s_l :=$ слева от s'

FindNewEvent(s_l, s', p)

$s'' :=$ крайний правый в $U(p) \cup C(p)$

$s_r :=$ справа от s''

FindNewEvent(s'', s_r, p)

end if

СЛОЖНОСТЬ

$O(n \cdot \log n)$ — заполнение очереди

Пока $Q \neq \emptyset$ ($\leq 2n + 1$ раз)

изменение очереди — $O(\log n)$

изменение дерева статуса — $O(\log n \cdot |U(p) \cup L(p) \cup C(p)|)$

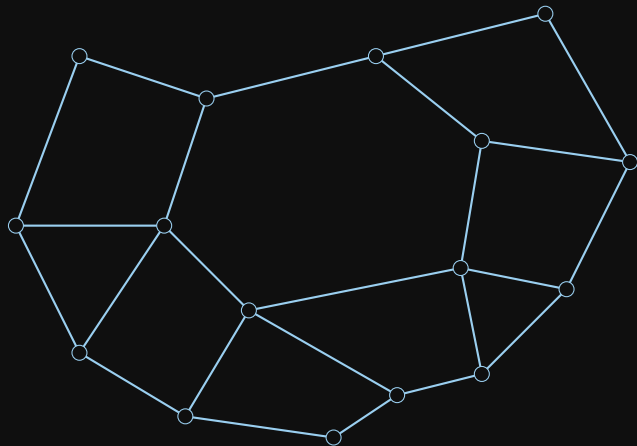
$$\sum_p |U(p) \cup L(p) \cup C(p)| = O(2n + 1) = O(n + 1)$$

$$\begin{aligned} O(n \cdot \log n) + O((2n + 1) \cdot \log n) + O\left(\log n \cdot \sum_p |U(p) \cup L(p) \cup C(p)|\right) = \\ = O((n + 1) \cdot \log n) \end{aligned}$$

DCEL (de Berg et al., p. 29+10)

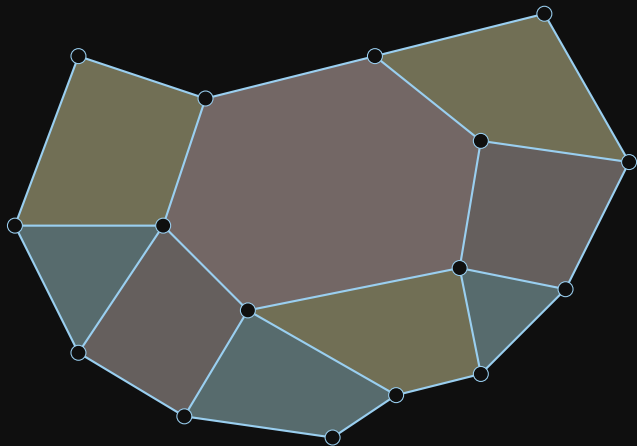
Карта на плоскости

Разбиение \mathbb{R}^2 на маркированные области.



Карта на плоскости

В вершинах и в областях могут храниться разные данные.

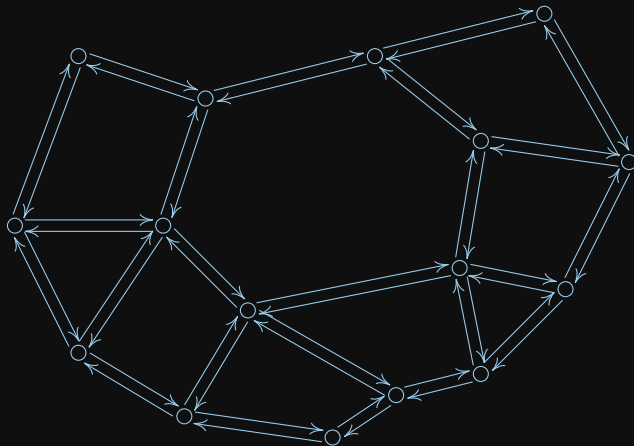


Поддерживаемые операции

- Обход области против часовой стрелки: даны клетка и ребро, перейти к следующему ребру.
- Обход вершины по часовой стрелке: даны вершина и ребро, перейти к следующему ребру.
- Переход между клетками: даны клетка и ребро, указать клетку «по ту сторону» ребра.

Полурёбра

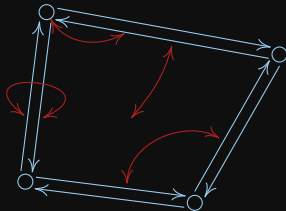
Рассмотрим вместо каждого ребра его два направленных варианта.



Структура данных

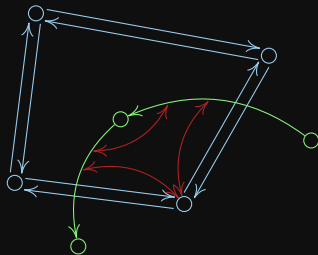
Запись для каждой вершины, ребра, области; и указатели:

- от каждого полуребра к предыдущему и следующему,
- между двумя полурёбрами одного ребра,
- между вершиной и смежными с ней рёбрами,
- между областью и её рёбрами.



Дополнительные операции

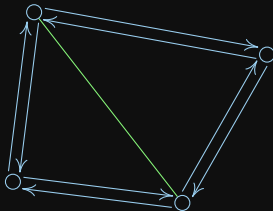
Имея DCEL, легко построить DCEL для *двойственного графа*.
Скопируем записи вершин, это будут новые грани.



Чтобы соединять указателями полурёбра, будем хранить, какие вершины мы уже обслужили.

Разрезание и склеивание

- $\text{split}(f, v_1, v_2)$ — разделить область f ребром v_1v_2 на две новых.
- $\text{merge}(f_1, f_2, e)$ — стереть ребро e , объединив между собой области f_1 и f_2 .



В худшем случае занимают время $O(n)$, потому что переделывать все указатели ребро \leftrightarrow область.

Ломти

Разделим области на **маленькие** и **большие**. Большие — те, у которых больше \sqrt{n} вершин.

Полурёбра маленьких областей будем хранить как есть, полурёбра больших областей объединим в **ломти** по примерно \sqrt{n} подряд идущих.

Картинка на доске

То же самое за polylog

Упражнение.

Убедитесь, что, если хранить полурёбра в структуре *link-cut tree*, можно делать операции *split* и *merge* в среднем за $O(\text{polylog}(n))$.

Спасибо за внимание!

О-нотация

Задачи, выполнимые за $O(1)$

Algorithms to get a feeling

Триангуляция и картинная галерея

Алгоритм с заметающей прямой

DCEL (de Berg et al., p. 29+10)