Всероссийская олимпиада школьников по информатике

Казань

2–8 апреля 2016 года

Задача «Крепость»

Задача «Крепость»

Задача «Крепость»

Задача «Крепость»

- Идея задачи — Глеб Евстропов
- Подготовка тестов — Павел Кунявский
- Разбор задачи — Александр Кленин

Постановка задачи

- Стена крепости из n участков, всего s защитников.
- На участок i нападает a_i солдат, один защитник может отбить k_i нападающих.
- Сквозь x_i защитников прорывается $max(a_i - x_i \cdot k_i, 0)$ нападающих.
- Минимизировать количество прорвавшихся.

Частичные группы: $k_i = 1$

- Распределение защитников не имеет значения.
- Прорывается $max(s - \sum a_i, 0)$ нападающих.
- **17 баллов**.

Частичные группы: $k_i \leq 2$

- Сначала распределяем по $\lfloor a_i/2 \rfloor$ защитников на стены с $k = 2$.
- Затем произвольно распределяем оставшихся.
- **+21 балл**.

Частичные группы: $n \leq 100$, $s \leq 10\,000$

- Необходимо максимально эффективно использовать каждого защитника.
- Жадный алгоритм. Повторять, пока $s > 0$:
  1. Поставить одного защитника на стену с максимальным значением $d = min(a_i, k_i)$.
  2. Уменьшить a_i на d_i.
  3. Уменьшить s на 1.
- Сложность $O(n \cdot s)$. **+23 балла**.

Оптимизация

- Заметим, что значение d в предыдущем алгоритме равно k_i в течение $\lfloor a_i/k_i \rfloor$ шагов, затем $(a_i \mod k_i)$ на последнем шаге.
- Вычислим все возможные значения d_i заранее.
- Для каждого i получаем $\lfloor a_i/k_i \rfloor$ потенциальных защитников с эффективностью k_i и одного с эффективностью $(a_i \mod k_i)$.

Полное решение

- Будем подсчитывать в количество защитников с эффективностью d в элементе массива $cnt[d]$, затем пройдём по массиву cnt с конца.
- $O(n+s)$ времени, $O(s)$ памяти. **+0 баллов**.
- Создадим массив пар <эффективность, количество>, отсортируем по убыванию эффективности, затем пройдём по массиву.
- Сложность $O(n \cdot \log n)$. **+39 баллов**.

Вопросы?

Задача «Экспериментальная робототехника»

Задача «Экспериментальная робототехника»

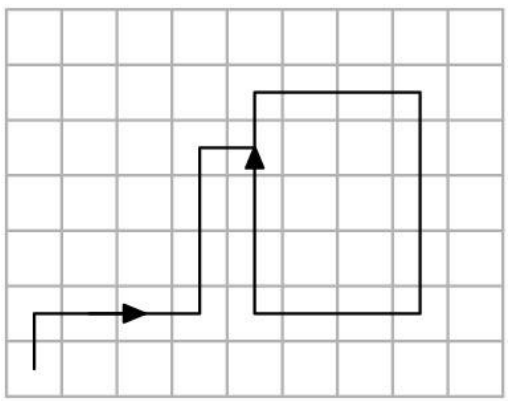
- Идея задачи — Михаил Мирзаянов
- Подготовка тестов — Михаил Пядеркин
- Разбор задачи — Елена Андреева

Постановка задачи

- На прямоугольном поле расставлены роботы и заданы направления движения из каждой клетки.
- Активированные роботы движутся согласно заданным направлениям.
- Надо активировать как можно больше роботов, чтобы они никогда не столкнулись.

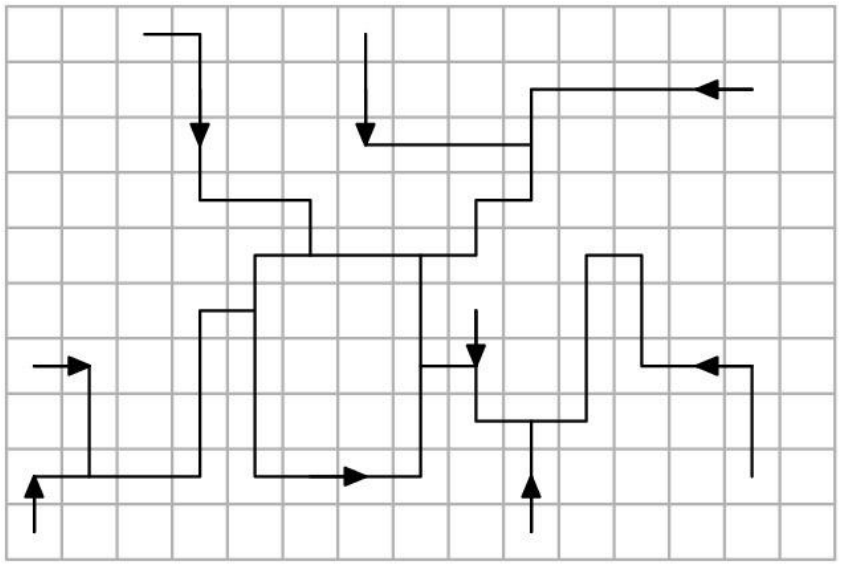
Решение

- Рассмотрим траекторию одного робота.
- Заметим, что если робот попадает в клетку, в которой он уже когда-либо был, то он будет дальше ходить по циклу.



Решение

- Если два робота в разные моменты попадали в какую-то одну и ту же точку, то дальше они будут двигаться одинаково.



Решение

- Это значит, что либо два робота будут ходить по одному и тому же циклу, либо их пути не пересекаются.
- Это наблюдение дает оценку сверху на ответ — нельзя запустить роботов больше, чем суммарная длина всех циклов.

В каждой клетке находится робот

- В случае, когда робот находится в каждой клетке, достаточно выделить циклы.
- После этого достаточно запустить роботов, которые сразу стоят на цикле, в момент времени 1.
- За это решение можно было получить 11 или 24 балла, в зависимости от эффективности алгоритма поиска циклов в графе с помощью поиска в глубину.

Общий случай

- Для того, чтобы найти ответ в общем случае, кроме выделения циклов, необходимо посчитать количество роботов которые придут на каждый цикл.
- Оказывается, что можно запустить роботов так, чтобы на каждом цикле все роботы, которые туда придут, могут двигаться сколь угодно долго, если их количество не превосходит длины цикла.

Общий случай

- Таким образом, чтобы посчитать ответ в общем случае, необходимо просуммировать по всем циклам минимум из длины цикла и количества роботов, которые на него придут.
- В зависимости от эффективности реализации такие решения получали 37 или 54 балла.

Восстановление ответа

- Решим задачу независимо для каждом цикла.
- На каждом цикле выберем одну точку.
- Заметим, что достаточно, чтобы в выделенной точке никакие два робота не оказались одновременно.
- Выберем любых роботов которых будем активировать так, чтобы на каждом цикле оказалось максимально возможное число роботов.

Восстановление ответа

- Найдем расстояние от выбранных роботов до выделенной точки.
- Зная расстояние до точки, мы можем выбрать момент, когда робот в нее придет, и по нему вычислить, когда запустить робота.

Восстановление ответа

- Пронумеруем роботов приходящих в один цикл от 1 до длины цикла или их количества.
- Запустим робота с номером i в момент времени $n \times m + i - d_i$, где d_i расстояние от него до выделенной точки.
- Тогда робот с номером i придет в выделенную точку в момент времени $n \times m + i$, и никакие два их них не окажутся в ней одновременно.

Вопросы?

Задача «Ловить или не ловить»

Задача
«Ловить или не ловить»

Задача «Ловить или не ловить»

Задача «Ловить или не ловить»

Идея задачи	Глеб Евстропов
Подготовка тестов	Нияз Нигматуллин Сергей Копелиович
Разбор задачи	Сергей Копелиович

Постановка задачи

- Дана река, в некоторых точках реки можно ловить рыбу, в некоторых продавать.
- Чтобы плыть вверх по течению, нужно тратить топливо, оно стоит денег.
- Получить максимальную прибыль!

Общие идеи

- Жадность!
- Спускаемся до x, ловим рыбу
- Поднимаемся, продаём рыбу
- Расход на топливо пропорционален x
- Нет смысла плавать туда-сюда
- Задача в том, чтобы выбрать x и посчитать стоимость

Общие идеи

- Жадность!
- Спускаемся до x, ловим рыбу
- Поднимаемся, продаём рыбу
- Расход на топливо пропорционален x
- Нет смысла плавать туда-сюда
- Задача в том, чтобы выбрать x и посчитать стоимость

Общие идеи

- Жадность!
- Спускаемся до x, ловим рыбу
- Поднимаемся, продаём рыбу
- Расход на топливо пропорционален x
- Нет смысла плавать туда-сюда
- Задача в том, чтобы выбрать x и посчитать стоимость

Общие идеи

- Жадность!
- Спускаемся до x, ловим рыбу
- Поднимаемся, продаём рыбу
- Расход на топливо пропорционален x
- Нет смысла плавать туда-сюда
- Задача в том, чтобы выбрать x и посчитать стоимость

Общие идеи

- Жадность!
- Спускаемся до x, ловим рыбу
- Поднимаемся, продаём рыбу
- Расход на топливо пропорционален x
- Нет смысла плавать туда-сюда
- Задача в том, чтобы выбрать x и посчитать стоимость

Общие идеи

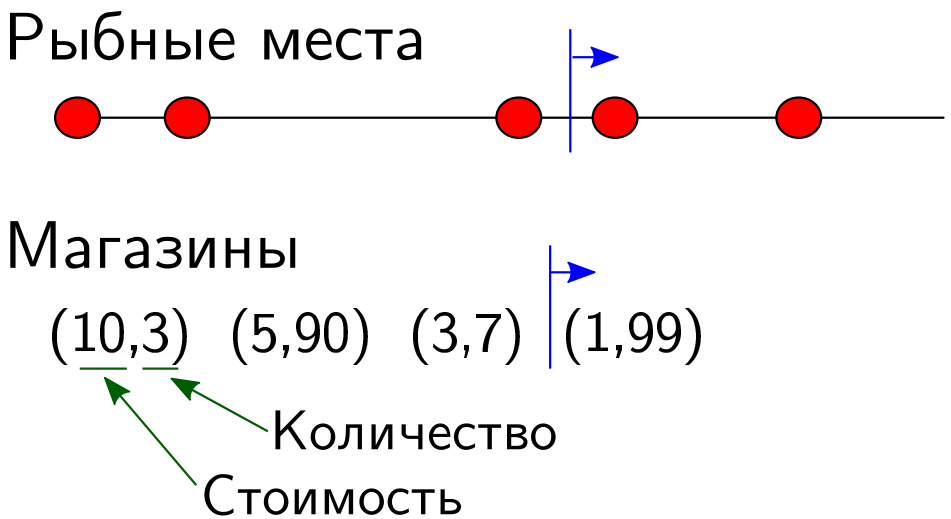
- Жадность!
- Спускаемся до x, ловим рыбу
- Поднимаемся, продаём рыбу
- Расход на топливо пропорционален x
- Нет смысла плавать туда-сюда
- Задача в том, чтобы выбрать x и посчитать стоимость

Бесплатное топливо

- Выгодно подняться до самого верха (бесплатно же!) и выловить всю рыбу
- Продаём тем, кто готов купить дороже
- Сортируем места продажи по убыванию цены

Сперва магазины, потом рыба

- Сортируем все места продажи по убыванию цены
- Двумя указателями по рыбным местам и магазинам поддерживаем суммарную выручку от продажи рыбы.



Сперва рыба, потом магазины

1. Выловим всю рыбу, пусть в сумме поймали A.
2. По очереди проплываем магазины, нужно уметь добавить новый магазин $\langle \texttt{cost}, \texttt{b} \rangle$ в множество:
   `B += b; COST += cost * b; "добавь";`
   `while B > A do "удали самый дешёвый";`
3. Достаточно создать `set<магазинов>`.



Общий случай

1. Идём от устья, обрабатываем события

1a. Рыбное место? `A += a`$_i$.

1b. Магазин $\langle$`cost, b`$\rangle$? Нужна структура данных.

2. Структура данных "сортированный массив": $\mathcal{O}(n)$.

3. Дерево отрезков, считающее две суммы.
   `count[cost] += b; profit[cost] += cost*b;`
   Ищем `max cost: sumCount [cost,+`∞`) `$\geqslant A$.

4. $\mathcal{O}(\log^2 n)$. Бинпоиск по `cost`, запрос к дереву.

5. $\mathcal{O}(\log n)$. Дерево отрезков со спуском.

Общий случай

1. Идём от устья, обрабатываем события

1a. Рыбное место? `A += a`$_i$.

1b. Магазин $\langle$`cost, b`$\rangle$? Нужна структура данных.

2. Структура данных "сортированный массив": $\mathcal{O}(n)$.

3. Дерево отрезков, считающее две суммы.
   `count[cost] += b; profit[cost] += cost*b;`
   Ищем `max cost: sumCount [cost,+`∞`) `$\geqslant A$.

4. $\mathcal{O}(\log^2 n)$. Бинпоиск по `cost`, запрос к дереву.

5. $\mathcal{O}(\log n)$. Дерево отрезков со спуском.

Общий случай

1. Идём от устья, обрабатываем события
1a. Рыбное место? `A` += a_i.
1b. Магазин $\langle$`cost`, `b`$\rangle$? Нужна структура данных.
2. Структура данных "сортированный массив": $\mathcal{O}(n)$.
3. Дерево отрезков, считающее две суммы.
   `count[cost] += b; profit[cost] += cost*b;`
   Ищем `max cost: sumCount [cost,`$+\infty$`)` $\geqslant A$.
4. $\mathcal{O}(\log^2 n)$. Бинпоиск по `cost`, запрос к дереву.
5. $\mathcal{O}(\log n)$. Дерево отрезков со спуском.

Общий случай

1. Идём от устья, обрабатываем события
1a. Рыбное место? `A` += a_i.
1b. Магазин $\langle$`cost`, `b`$\rangle$? Нужна структура данных.
2. Структура данных "сортированный массив": $\mathcal{O}(n)$.
3. Дерево отрезков, считающее две суммы.
   `count[cost] += b; profit[cost] += cost*b;`
   Ищем `max cost: sumCount [cost,+`∞`) >=` A.
4. $\mathcal{O}(\log^2 n)$. Бинпоиск по `cost`, запрос к дереву.
5. $\mathcal{O}(\log n)$. Дерево отрезков со спуском.

Общий случай

1. Идём от устья, обрабатываем события

1a. Рыбное место? `A += a`$_i$.

1b. Магазин $\langle$`cost, b`$\rangle$? Нужна структура данных.

2. Структура данных "сортированный массив": $\mathcal{O}(n)$.

3. Дерево отрезков, считающее две суммы.
   `count[cost] += b; profit[cost] += cost*b;`
   Ищем `max cost: sumCount [cost,+`∞`)` $\geqslant A$.

4. $\mathcal{O}(\log^2 n)$. Бинпоиск по `cost`, запрос к дереву.

5. $\mathcal{O}(\log n)$. Дерево отрезков со спуском.

Общий случай

1. Идём от устья, обрабатываем события
1a. Рыбное место? `A += a`$_i$.
1b. Магазин $\langle$`cost, b`$\rangle$? Нужна структура данных.
2. Структура данных "сортированный массив": $\mathcal{O}(n)$.
3. Дерево отрезков, считающее две суммы.
   `count[cost] += b; profit[cost] += cost*b;`
   Ищем `max cost: sumCount [cost,+`∞`) `$\geqslant A$.
4. $\mathcal{O}(\log^2 n)$. Бинпоиск по `cost`, запрос к дереву.
5. $\mathcal{O}(\log n)$. Дерево отрезков со спуском.

Общий случай

1. Идём от устья, обрабатываем события

1a. Рыбное место? `A += a`$_i$.

1b. Магазин $\langle$`cost, b`$\rangle$? Нужна структура данных.

2. Структура данных "сортированный массив": $\mathcal{O}(n)$.

3. Дерево отрезков, считающее две суммы.
`count[cost] += b; profit[cost] += cost*b;`
Ищем `max cost: sumCount [cost,+`∞`)` $\geqslant A$.

4. $\mathcal{O}(\log^2 n)$. Бинпоиск по `cost`, запрос к дереву.

5. $\mathcal{O}(\log n)$. Дерево отрезков со спуском.

Вопросы?

Задача «Обитаемые горы»

Задача «Обитаемые горы»

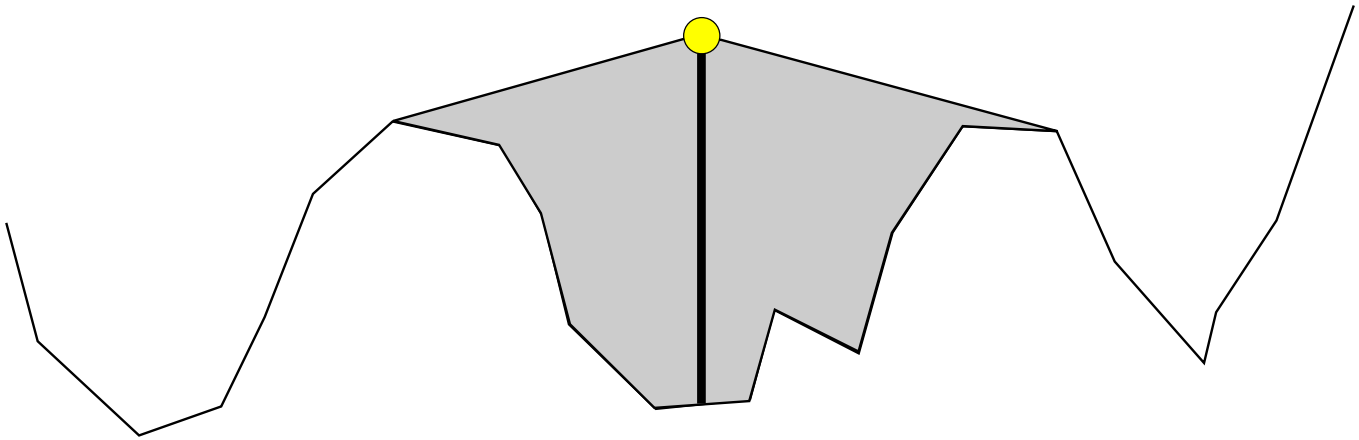
Задача «Обитаемые горы»

Задача «Обитаемые горы»

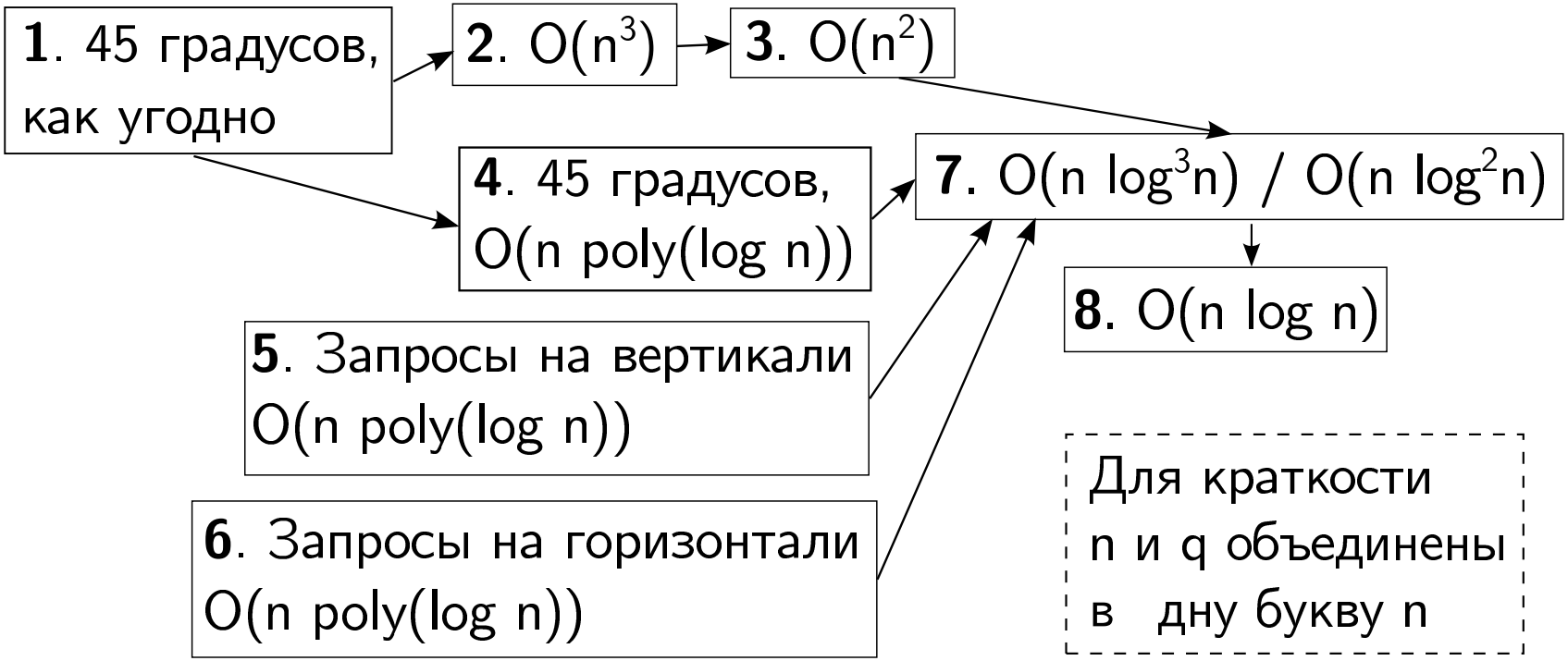
Идея задачи	Максим Ахмедов
Подготовка тестов	Максим Ахмедов Глеб Евстропов
Разбор задачи	Глеб Евстропов

Постановка задачи

- Есть ломаная, состоящая из n звеньев
- Есть q точек-запросов (u_j, v_j) над ломаной
- Для каждой точки-запроса требуется определить максимальный отрезок $[l_j, r_j]$ ломаной, содержащий u_j который она видит

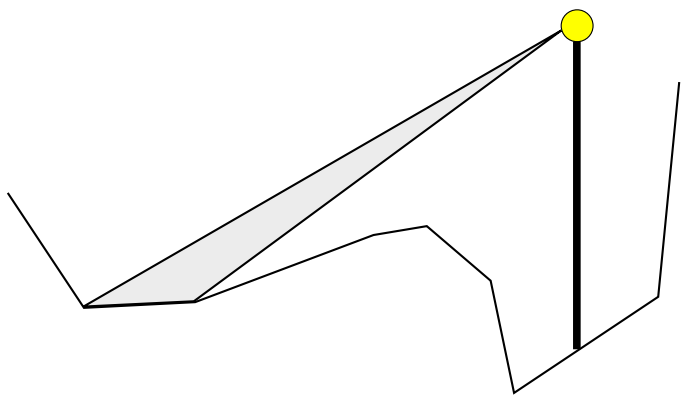


Подзадачи


1. 45 градусов, как угодно
2. O(n^3)
3. O(n^2)
4. 45 градусов, O(n poly(log n))
5. Запросы на вертикали O(n poly(log n))
6. Запросы на горизонтали O(n poly(log n))
7. O(n log^3 n) / O(n log^2 n)
8. O(n log n)
Для краткости n и q объединены в дну букву n

Подзадачи 1 и 2: $\mathcal{O}(n^3)$

- Границы видимого отрезка — всегда вершины ломаной
- Переберём справа налево первый отрезок, который мы не видим
- Отрезок никем не загораживается, если мы видим оба его конца

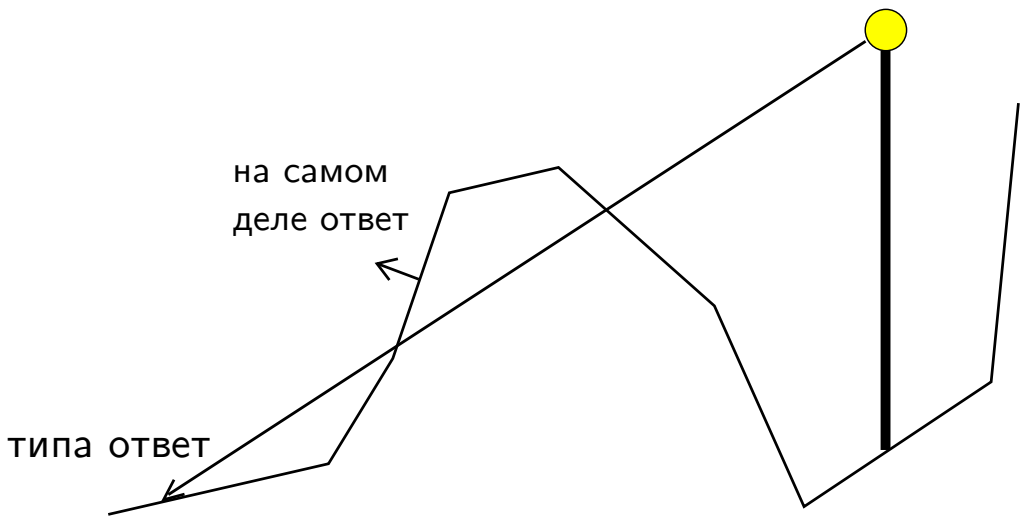


Подзадачи 1 и 2: $\mathcal{O}(n^3)$

- Перебираем запрос, перебираем отрезок-кандидат, перебираем, кто бы его загородил
- Сложность решения — $\mathcal{O}(n^3)$

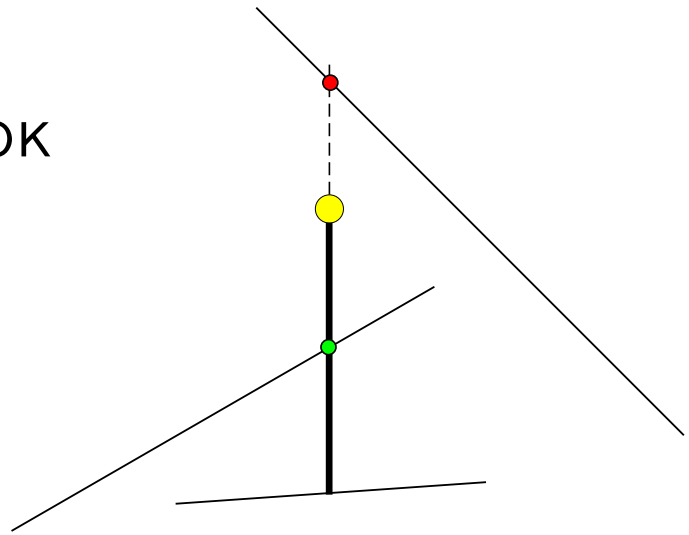
Исследуем задачу

- Отрезок точно не виден если точка-запрос находится под прямой, его содержащей, назовём такие отрезки *накрывающими*
- На самом деле для ближайшего слева не просматриваемого отрезка это критерий



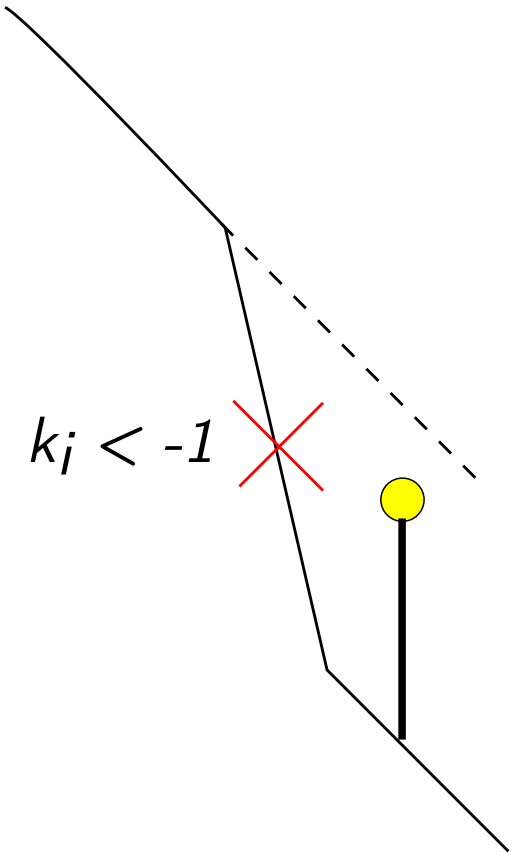
Подзадача 3: $\mathcal{O}(n^2)$

- Идём по звеньям ломаной от позиции нашего запроса налево и направо, находим первый накрывающий отрезок
- Можно делать простую целочисленную проверку: строим уравнение прямой, берём значение в точке u_j, сравниваем с v_j
- $v_j < y_i + k_i(u_j - x_i)$?



Подзадача 4: $\mathcal{O}(n)$ и $\mathcal{O}(n \log n)$

- Будем искать, например, левую границу ответа
- Отрезок с $k_i = -1$ не может быть накрывающим, иначе в координате u_j точка-запрос окажется под ломаной (ломаная не может слишком круто спускаться вниз)



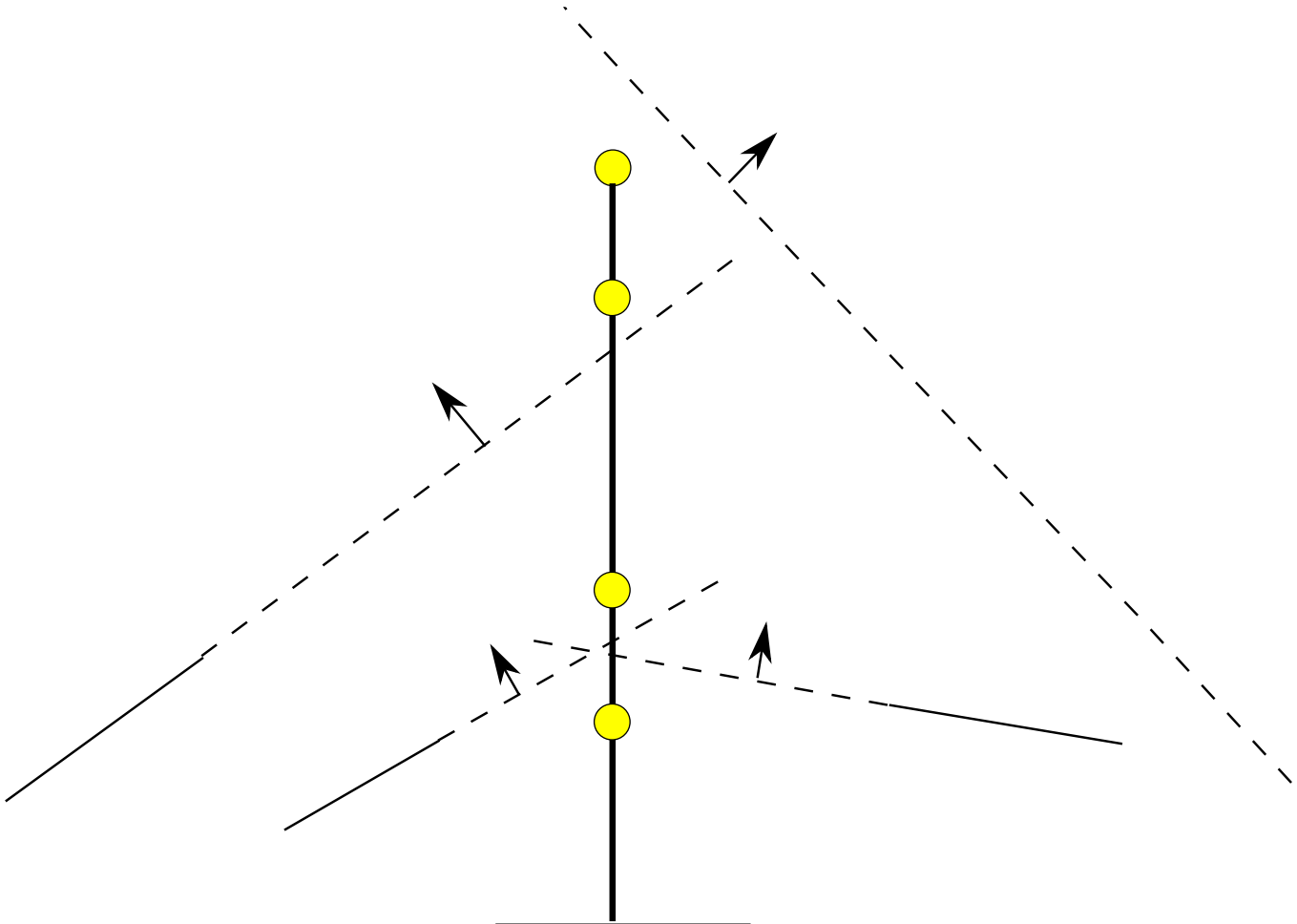
Подзадача 4: $\mathcal{O}(n)$ и $\mathcal{O}(n \log n)$

- $v_j < y_i + (u_j - x_i) \Leftrightarrow v_j - u_j < y_i - x_i$
- Будем проходить отрезки и запросы слева направо, поддерживая отрезки в некоторой структуре данных, используя $v_j - u_j$ в качестве ключа
- Когда мы встречаем запрос, нужно спросить самый большой индекс отрезка из имеющих ключ строго больше $v_j - u_j$
- Варианты структуры данных: дерево отрезков, декартово дерево, стек рекордов + бинарный поиск

Подзадача 5: $\mathcal{O}(n \log n)$

- Каждый отрезок высекает на вертикальной прямой, на которой живут запросы, некоторый луч, направленный вверх
- На протяжении всего этого луча отрезок является накрывающим
- Выписываем события двух видов: начала лучей и запросы
- Идём сверху вниз, поддерживаем ближайший слева и ближайший справа накрывающие отрезки
- Получаем решение за $\mathcal{O}(n \log n)$ (sort)

Подзадача 5: $\mathcal{O}(n \log n)$



Подзадача 6: $\mathcal{O}(n \log n)$

- Если все запросы на одной горизонтальной прямой, работают все те же самые рассуждения, что и в 5 подзадаче, только теперь бывают отрезки разных видов: дающие левый луч, правый луч, прямую или не дающие ничего
- Либо обрабатываем по отдельности каждый вид, либо сортируем события и отвечаем на запросы, поддерживая множество активных накрывающих отрезков в подходящей структуре (например, `std::set`)

Подзадача 7: $\mathcal{O}(n \log^2 n)$

- Переформулируем задачу так: нам нужно находить ближайший слева отрезок, такой что точка-запрос не лежит в верхней полуплоскости относительно него
- Мы умеем по множеству полуплоскостей строить структуру данных, позволяющую быстро ответить на запрос «лежит ли точка во всех полуплоскостях одновременно» — пересечение полуплоскостей.
- Запрос «лежит ли точка» обрабатывается за $\mathcal{O}(\log n)$ одним бинпоиском.
- Пересечение уже отсортированных по углу полуплоскостей можно построить за $\mathcal{O}(n)$ одним проходом со стеком.

Подзадача 8: $\mathcal{O}(n \log^2 n)$

- Построим дерево отрезков на массиве данных нам полуплоскостей.
- В вершине дерева отрезка пересечение полуплоскостей отрезка.
- Построение вершины: merge детей + $\mathcal{O}(n)$.
- Запрос к дереву «ближайшая слева полуплоскость, в которой наша точка не лежит».
- Спуск по дереву, в каждой вершине бинпоиск по пересечению.
- Итого online решение $\langle \mathcal{O}(n \log n), \mathcal{O}(\log^2 n) \rangle$.
- 100 баллов.

Решение за $\mathcal{O}((n+m)\log n)$

У жюри есть несколько решений за такое время

- Добавим к предыдущему решению технику fractional cascading, получим запрос за $\mathcal{O}(\log n + \log n)$ на запрос. Бинпоиск будем делать только в корне, в любой другой вершине дерева $\mathcal{O}(1)$.
- Наша задача offline. Можно её решать сканирующей прямой слева направо. Идея «поддерживать set прямых, пересекающих данную вертикальную, потенциально могущих быть ответом на запрос» приводит к решению за $\mathcal{O}((n+m)\log n)$.
- Так же есть другое решение сканирующей прямой...

Вопросы?

Задача «Управление видеонаблюдением»

Задача «Управление видеонаблюдением»

Задача «Управление видеонаблюдением»

Задача «Управление видеонаблюдением»

- Идея задачи — Максим Ахметов
- Подготовка тестов — Нияз Нигматуллин
- Разбор задачи — Александр Кленин

Постановка задачи

- Матрица a размером $n \times m$.
- Разрешены циклические сдвиги по горизонтали и вертикали.
- Максимизировать количество блоков (i, j), где $a_{i,j} = a_{i+1,j} = a_{i,j+1} = a_{i+1,j+1}$.

Частичные группы: $n, m \leq 50$

- Переберём все возможные сдвиги.
- Для каждого сдвига подсчитаем количество блоков.
- Сложность $O(n^4)$. **37 баллов**.

Частичные группы: $n, m \leq 300$

- Переберём все возможные сдвиги.
- Для каждого сдвига подсчитаем количество блоков, разрезанных границей.
- Аккуратно обработаем углы, чтобы избежать двойного подсчёта.
- Сложность $O(n^3)$. **+28 баллов**.

Полное решение

- Создадим матрицу A размером $2n \times 2m$ из 4 копий матрицы a.
- Каждый циклический сдвиг матрицы a соответствует подматрице матрицы A.
- Предварительно рассчитаем массив p, где $p_{i,j}$ — количество удобных блоков в подматрице $A_{0,0} - A_{i,j}$.
- Количество удобных блоков для сдвига (i, j) равно $p_{i-1,j-1} + p_{i+n,j+m} - p_{i-1,j+m} - p_{i+n,j-1}$.
- Сложность $O(n^2)$. **+35 баллов**.

Вопросы?

Задача Расшифровка ДНК

Задача Расшифровка ДНК

Задача Расшифровка ДНК

Расшифровка ДНК

Идея задачи	Максим Ахмедов
Подготовка тестов	Никита Сендерович
Разбор задачи	Павел Маврин

Постановка задачи

- Загадана последовательность длины n из k различных символов.
- Можно делать запросы $get(L, R)$: «Сколько различных символов на отрезке от L до R?».
- Восстановить последовательность с точностью до подстановки символов.

Общие идеи

- Будем восстанавливать строку слева направо.
- Пусть мы восстановили все символы до i-го.
- Если $get(j, i) = get(j, i - 1)$, значит символ i встречается на отрезке от j до $i - 1$.

Решение за n^2 запросов

- Будем делать запросы, увеличивая отрезок.
- Если $get(j, i) = get(j, i-1)$, и при этом $get(j+1, i) \neq get(j+1, i-1)$, значит $a[i] = a[j]$.
- Если для всех j: $get(j, i) \neq get(j, i-1)$, значит $a[i]$ ранее не встречался в строке.

Решение за $2n \log n$ запросов

- Будем делать то же самое, но двоичным поиском.
- Для проверки используем два запроса: $get(j, i) = get(j, i - 1)$.

Решение за $n \log n$ запросов

- Будем делать один запрос вместо двух.
- Действительно, ответ на запрос $get(j, i-1)$ мы и так знаем.

Решение за nk запросов

- Будем для каждого символа хранить его самое правое вхождение среди уже просмотренных.
- Достаточно делать запросы, у которых левый конец в одном из таких символов.

Решение за $n \log k$ запросов

- То же самое можно делать двоичным поиском, для этого нужно поддерживать последние вхождения в отсортированном порядке.

Не бойтесь интерактивов

Вопросы?

Задача «Курьерская служба»

Задача «Курьерская служба»

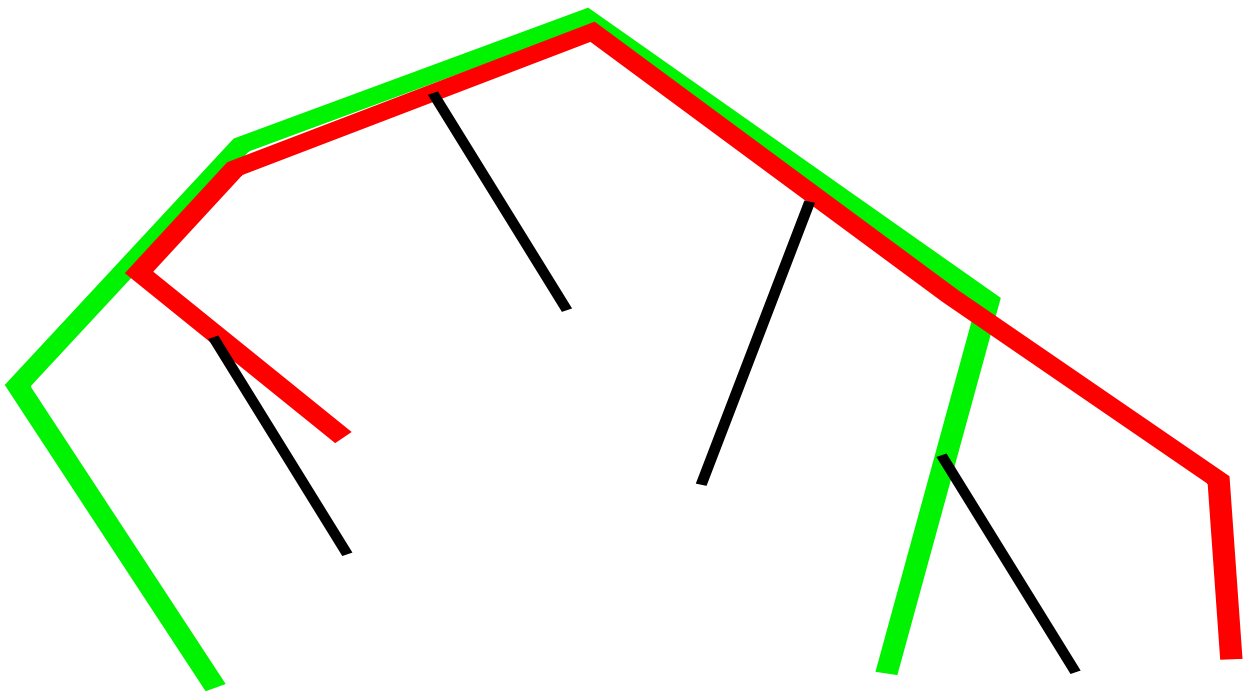
Задача «Курьерская служба»

Задача «Курьерская служба»

- Идея задачи — Михаил Пядёркин
- Подготовка тестов — Михаил Пядёркин, Сергей Копелиович
- Разбор задачи — Михаил Пядёркин

Математическая формулировка

- Дано дерево и несколько путей в нём
- Найти два пути с максимальным общим количеством рёбер



Простое решение на 29 баллов: $O(nk^2)$

- Рассмотрим каждую пару путей P_1, P_2.
- С помощью поиска в глубину найдем вершины, лежащие на первом пути
- С помощью того же поиска в глубину найдем вершины, лежащие на втором пути
- Длина пересечения путей равна количеству вершин, лежащих на обоих путях, минус один

Ускорение простого решения в 64 раза

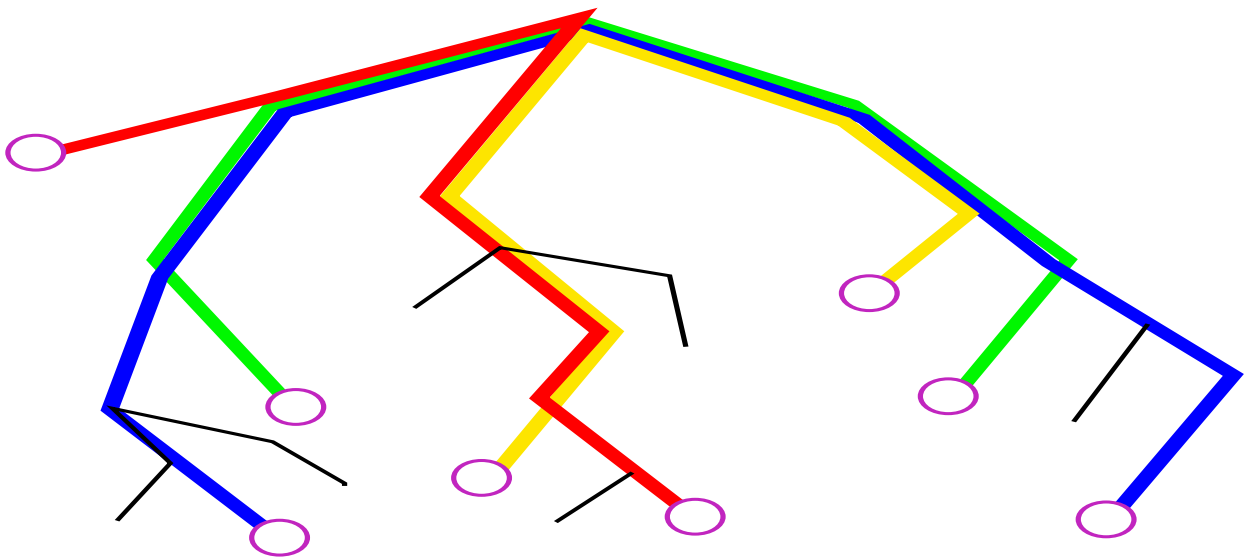
- Заранее для каждого из путей с помощью поиска в глубину найдем множество вершин, лежащих на этом пути
- Рассмотрим каждую пару путей P_1, P_2
- Необходимо найти размер пересечения двух множеств — это можно сделать с помощью битового сжатия
- Можно использовать `<bitset>`

Идейное ускорение простого решения

- Заранее для каждого из путей с помощью поиска в глубину найдем множество вершин, лежащих на этом пути
- С помощью этого сохраним *для каждой вершины* все пути, проходящие через эту вершину
- Ответ к задаче является путем, переберем один из его концов, пусть это вершина v

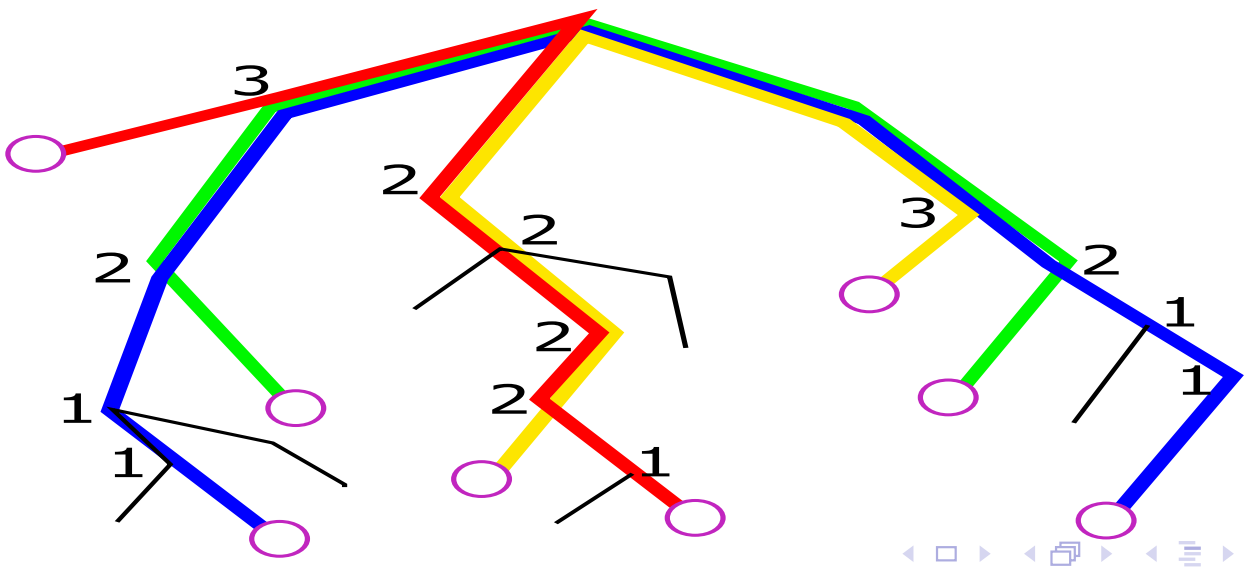
Идейное ускорение простого решения

- Подвесим дерево за выбранную вершину v
- Для каждого из путей, проходящих через v, отметим концы этого пути как особенные



Идейное ускорение простого решения

- Для каждого из поддеревьев вычислим количество особенных вершин в поддереве
- Требуется выбрать самую глубокую вершину, в поддереве которой имеется хотя бы две особенных вершины

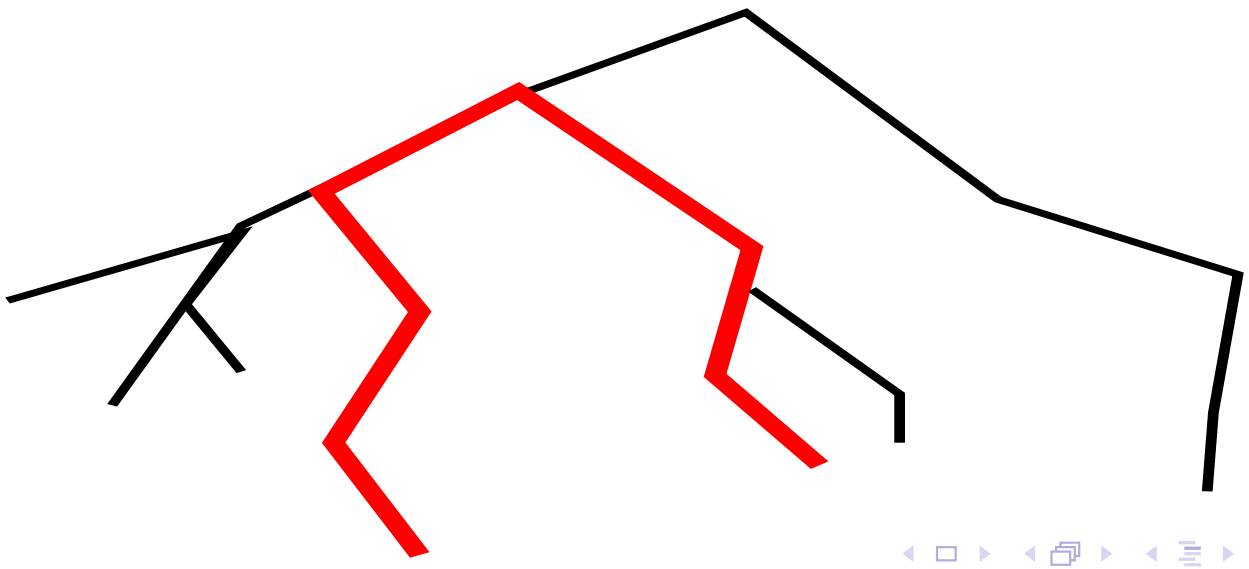


Идейное ускорение простого решения

- Таким образом, первая часть решения работает за $O(nm)$
- Вторая часть решения требует запуска обхода в глубину из каждой вершины, и поэтому работает за $O(n^2)$
- Итоговая асимптотика $O(n(n+m))$
- 41 балл

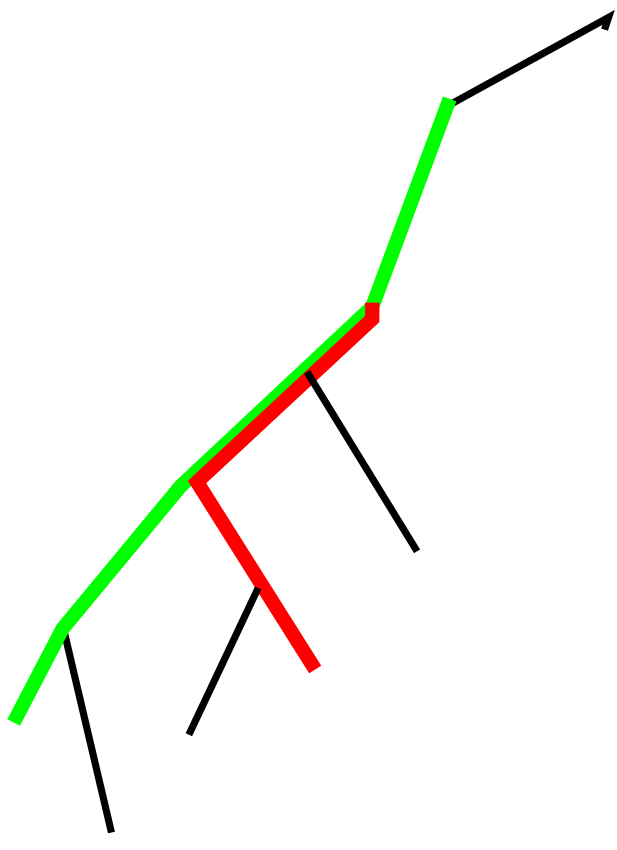
Быстрое пересечение путей

- Переберем пару путей, после этого быстро найдем длину пересечения
- Каждый путь декомпозируется на два вертикальных: для этого нужно уметь находить наименьшего общего предка



Пересечение вертикальных путей

- Найдем наименьшего общего предка для нижних концов, это будет являться нижней точкой пересечения, если они пересекаются
- Верхней точкой пересечения вертикальных путей будет та из верхних точек исходных путей, которая ниже



Оценка сложности

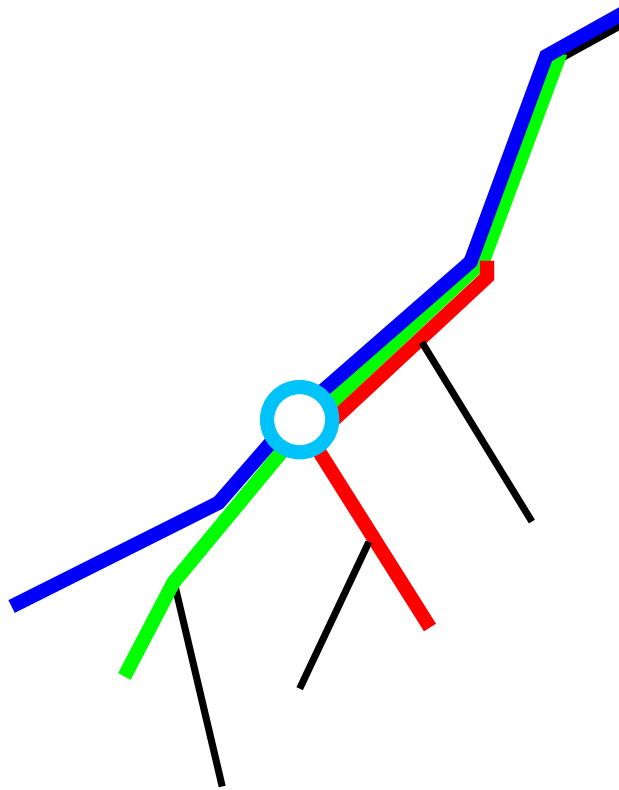
- Находить наименьшего общего предка можно за $O(\log n)$ с помощью двоичного подъема
- Итоговая сложность решения равна $O(n \log n + m^2 \log n)$
- 48 баллов

Поиск общего предка за $O(1)$

- Рассмотрим эйлеров обход дерева
- Тогда наименьшем общим предком двух вершин является вершина минимальной глубины на отрезке эйлерова обхода между двумя вершинами
- С помощью метода разреженных таблиц мы можем отвечать на запрос минимума за $O(1)$
- Сложность решения теперь $O(n \log n + m^2)$
- 56 баллов

Решение «снизу вверх»

- Пусть мы зафиксировали нижнюю точку пересечния
- Тогда из путей, проходящих через эту вершину, нас интересуют лишь два, заканчивающихся как можно выше
- Это можно поддерживать с помощью обычного обхода в глубину

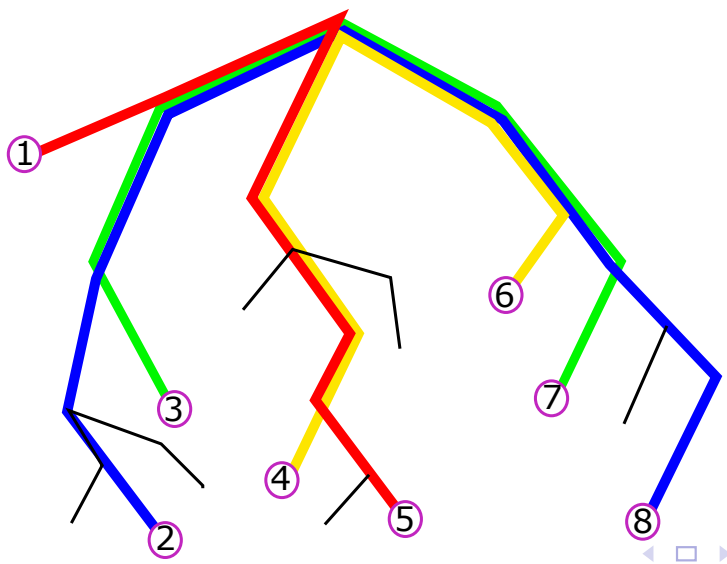


Решение за $O(mh \log n)$

- Для каждой вершины сохраним список путей, проходящих через эту вершину
- Переберем один из коцов пересечения путей, пусть это вершина v
- Упорядочим концы путей, проходящих через эту вершину, по времени входа

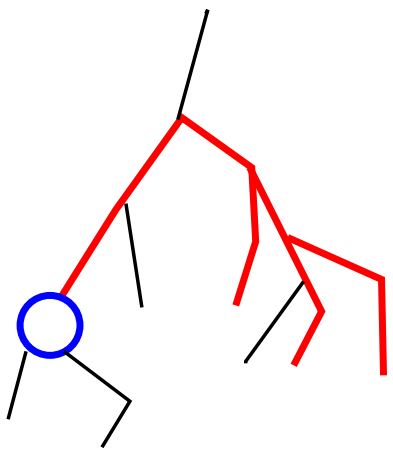
Решение за $O(mh \log n)$

- Переберем теперь один из путей, проходящих через вершину v, и рассмотрим его конец
- Для выбора второго пути заметим, что интересными являются два кандидата — концы путей, ближайшие по времени входа



Полное решение

- Запустим обход в глубину, который будет возвращать концы путей, которые начинаются в этом поддереве, а заканчиваются где-то вне этого поддерева
- При этом, концы путей будут упорядочены по времени входа: требуется `set`

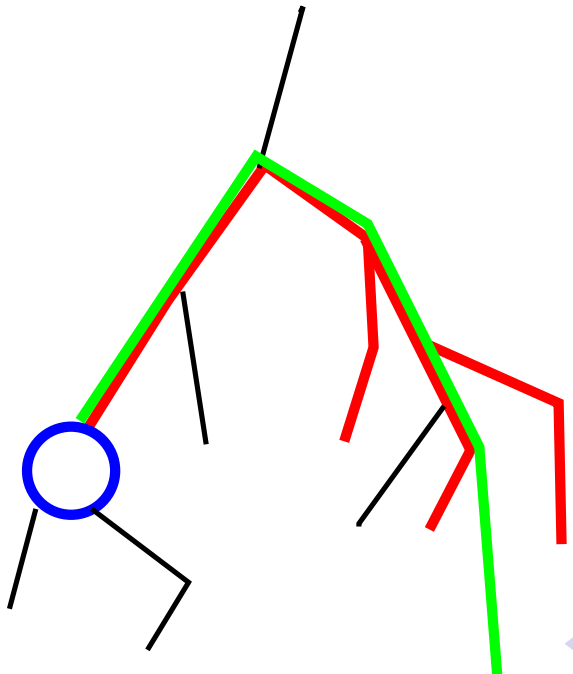


Полное решение

- Для реализации этого обхода в глубину необходимо объединять множества для сыновей
- При объединении двух множеств будем использовать метод перекидывания из меньшего множества в большее

Полное решение

- В процессе перекидывания одного элемента рассмотрим два ближайших по времени входа конца путей, уже лежащих в множестве, и попробуем обновить ответ



Все очень легко!

Вопросы?

Задача “Тренажёр «10_2-пальцевый набор»”

Задача “Тренажёр «10_2-пальцевый набор»”

- Идея задачи — Глеб Евстропов
- Подготовка тестов — Максим Ахмедов и Глеб Евстропов
- Разбор задачи — Глеб Евстропов

Постановка задачи

- Дано n строк суммарной длины L и текст T длины m.
- Одно использование суффикса или префикса строки i стоит c_i.
- Суффиксы и префиксы конкатенируются в произвольном порядке.
- Получить текст за минимальную стоимость.

Динамическое программирование

- $ans(i)$ — лучая стоимость для i-го суффикса.
- Пробуем приложить все суффиксы и префиксы.
- Наивно проверяем переходы.
- Сложность решения $O(mLl_{max})$.
- **20 баллов**.

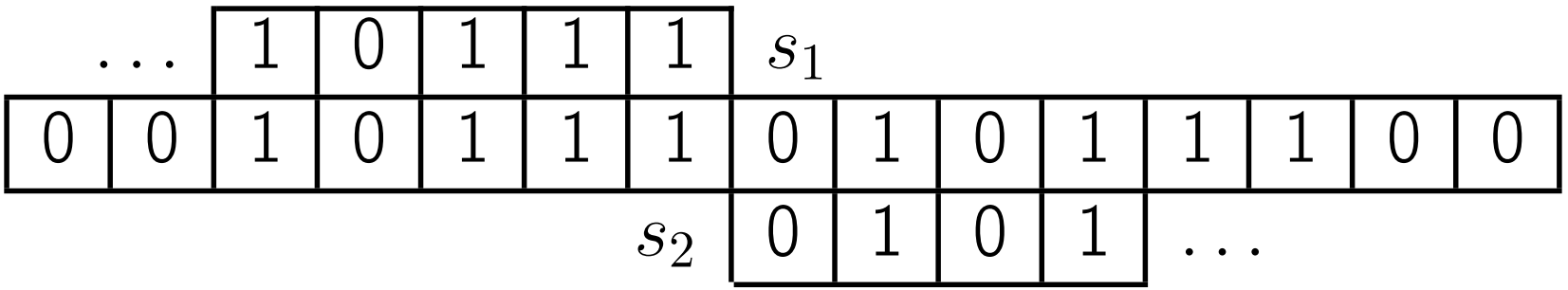
Хеш-функция

- Сравниваем строки с помощьюе хешей.
- Сложность улучшается до $O(mL)$.
- **30 баллов**.

Дополнительные наблюдения

- Не требуется прикладывать все строки.
- Добавим в хеш-таблицу хеши всех суффиксов и префиксов.
- По каждому значению хеша выберем оптимум.
- Асимптотика решения — $O(ml_{max} + L)$.
- **38 или 46 баллов**.

Динамика в две стороны



Динамика в две стороны (продолжение)

- Сохраним все строки в боре.
- Сохраним все развёрнутые строки в боре.
- Сложность $O(ml_{max} + L)$, но константа маленькая.
- **46 баллов**.

Маленькое n

- Переход вперёд — минимум на отрезке.
- Переход вперёд — релаксация минимума на отрезке.
- Итоговая сложность $O(mn \log m + L)$.
- **+10 или +15 баллов**.

Маленькое

- Делаем переходы по каждому c независимо.
- Бинпоиск + хеши дают самый длинный суффикс и префикс.
- Переход вперёд — минимум на отрезке.
- Переход вперёд — релаксация минимума на отрезке.
- Итоговая сложность $O(m \log m + L)$.
- **+9 баллов**.

Корневая эвристика

- Строк длины больше $\sqrt{L}$ не больше $\sqrt{L}$.
- Для коротких строк решаем аналогично группе 4.
- Для длинных строк решаем аналогично группе 6.
- Асимптотика $O(m\sqrt{L}\log m)$ или $O(m\sqrt{L\log m})$.
- **80+ баллов**.

Корневая эвристика возвращается

- $O(m)$ запросов и $O(m\sqrt{L})$ релаксаций.
- $O(m\sqrt{L})$ запросов и $O(m)$ релаксаций.
- Таким образом придумаем корневую эвристику.
- Чтобы делать корневую эвристику пока делаем корневую эвристику.
- Асимптотика $O(m\sqrt{L})$.
- **90+ баллов**.

0. Оптимизации

- Ахо-Корасик или автомат аналогичный z-функции.
- Суффиксные структуры вместо хешей.
- Аккуратный код.
- **100 баллов**.

Вопросы?