Заключительный этап
IX Открытой олимпиады
по программированию

Разбор задач первого дня

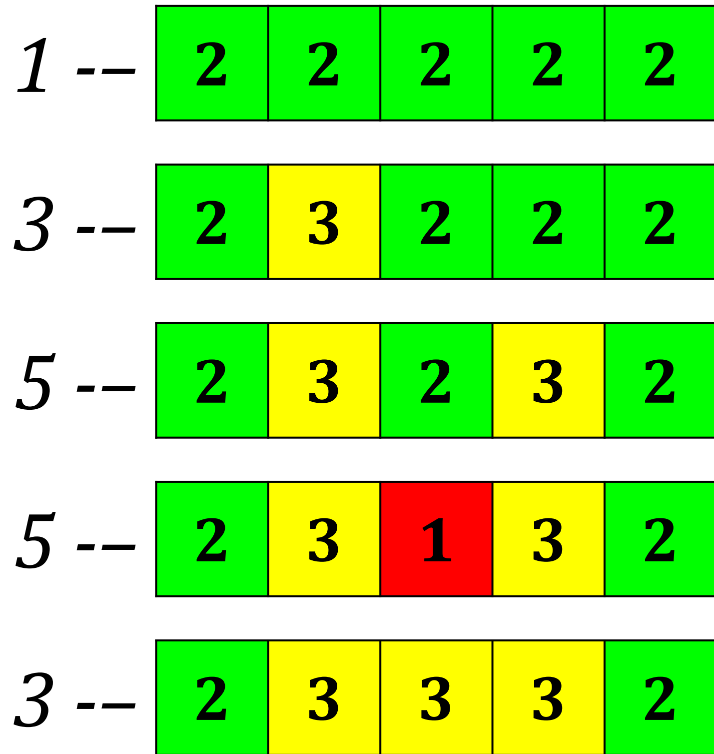
Задача «Ремонт асфальта»

Автор идеи: Михаил Пядёркин

Разработчик и автор разбора: Антон Полднев

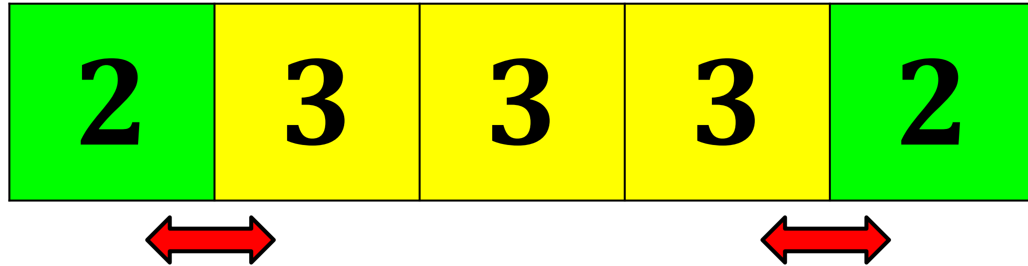
Формальная постановка

Есть полоска из N клеток, в каждой из которых написано число. На каждом из Q шагов меняется какое-то одно число. После каждого шага нужно вывести, на сколько одноцветных отрезков разбивается полоска.



Как найти количество отрезков?

Количество одноцветных отрезков — это количество пар соседних клеток разного цвета плюс один:



3 отрезка
2 пары

Как найти количество отрезков?

- Циклом for:
  ans := 1; for i=2..N: if a[i] ≠ a[i−1]: ans += 1;
- unique в C++:
  unique(a.begin(), a.end()) − a.begin()

50 баллов: O(NQ)

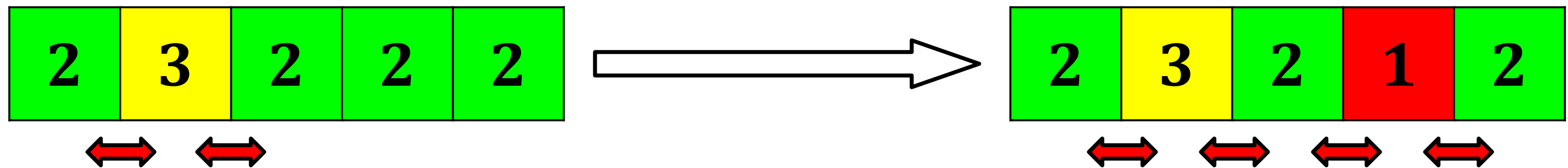
После каждого запроса обновим цвет соответствующей клетки и заново за O(N) найдём количество отрезков.

100 баллов: O(N+Q)

- Зачем после очередного запроса заново просматривать все клетки, если изменилась только одна?
- В самом начале один раз пройдёмся по полоске циклом и найдём количество отрезков.
- После каждого перекрашивания будем за O(1) определять, на сколько изменилось количество отрезков.
- Как?

100 баллов: O(N+Q)

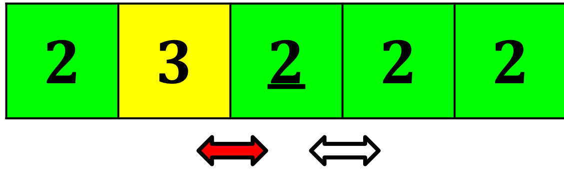
- Вспомним, что количество отрезков — это количество пар соседних клеток разного цвета плюс один.
- Значит, например, если после очередного перекрашивания отрезков стало на два больше, то и пар соседних клеток разного цвета стало на две больше:



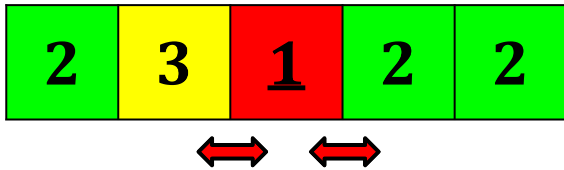
100 баллов: O(N+Q)

- Как быстро узнать, на сколько изменилось количество таких пар после перекрашивания одной клетки?
- Перекрашивание i-й клетки влияет только на две пары: (a[i−1], a[i]) и (a[i], a[i+1]).
- Найдём значения выражения $\text{int}(a[i-1] \neq a[i]) + \text{int}(a[i] \neq a[i+1])$ до и после перекрашивания. Разница между ними — то, на сколько изменилось количество отрезков.

100 баллов: O(N+Q)



До перекрашивания: клетка отличается только от **одного** своего соседа



После перекрашивания: клетка отличается от **двух** своих соседей

Значит, количество отрезков тоже увеличилось на 1.

100 баллов: O(N+Q)

Другая точка зрения на то же самое решение.

- При каждом перекрашивании будем интересоваться только перекрашиваемой клеткой и её соседями, если они есть.
- Для этих 3 клеток найдём количество отрезков до и после перекрашивания. Разницу прибавим к текущему количеству отрезков.

Задача «Выборы»

Автор идеи: обманутые налогоплательщики
Постановка задачи: Елена Андреева
Разработчик и автор разбора: Роман Андреев

Формальная постановка

Дано число N, нужно представить его в виде произведения K множителей ($N = a_1 a_2 \ldots a_K$) чтобы минимизировать $[(a_1+1)/2][(a_2+1)/2]\ldots[(a_k+1)/2]$.

Обозначение: [a] - целая часть числа a.

1 группа: $N \leq 1000$, $K \leq 2$

- $K = 1$

Ans = [(N + 1) / 2]

- $K = 2$

Переберем все делители d числа N:

Ans = min [(d + 1) / 2][((N / d) + 1) / 2]

Сложность - O(N)

2 группа: $N \leq 1000$

Воспользуемся динамическим программированием:

DP[k][n] - ответ на задачу

DP[1][n] уже знаем!

Пересчитываем также, как и для K = 2, перебираем делители d числа n:

DP[k][n] = min [(d + 1) / 2] DP[k - 1][n / d]

Сложность $O(N^2K)$

3 группа: $N \leq 10^6$

Заметим, что в прошлом решении нам не нужно рассматривать такие n, которые не являются делителями N. Для каждого d мы будем искать его делители за $O(d)$, а сумма по всем делителям числа N оказывается при заданных ограничениях $\approx O(N)$!

Сложность $O(NK)$.

Упражнение: оцените сумму точно и поймите, почему там $\approx$.

4 группа: $N \le 10^{11}$

Но раз мы не используем значения DP[k][d], где d не делитель N, давайте сожмем наш массив и получим DP’[k][i], где i - номер делителя числа N.

А сколько всего может быть делителей у числа N?

При $N \le 10^{15}$ максимум числа делителей:

Div(978 217 616 376 000) = 26 880. Совсем немного!

Находить все делители числа N мы умеем за $O(N^{0.5})$. Тогда мы получаем сложность $O(Div(N)^2 K + N^{0.5})$

5 группа: $N \leq 10^{13}$

А зачем мы каждый раз для всех k ищем все делители числа d? Мы же можем их предподсчитать!

Оказывается, что сумма по всем делителям числа делителей тоже небольшая!

Pairs(978 217 616 376 000) = 18 370 800

Сложность $O(Div(N)^2 + Pairs(N)K + N^{0.5})$

6 группа: $N \leq 10^{15}$ (Начало)

На самом деле, когда мы вычисляем все делители числа N, мы можем вместо этого разложить N на простые множители:

$N = p_1^{s_1} p_2^{s_2} \ldots p_t^{s_t}$

Тогда сгенерировать все делители мы можем перебрав все комбинации степеней простых за $(s_1+1)(s_2+1)\ldots(s_t+1)$. Причем при таком подходе мы сделаем это ровно за число делителей.

6 группа: $N \leq 10^{15}$ (Продолжение)

Абсолютно точно также можно сгенерировать и все делители делителей ровно за их количество: $(s_1+1)(s_1+2)/2 \ldots (s_t+1)(s_t+2)/2$.

Получаем решение за $O(Pairs(N)K + N^{0.5})$

PS Оптимизация в 2 раза: при пересчете динамики давайте рассматривать только делители $\leq d^{0.5}$, а их в два раза меньше!

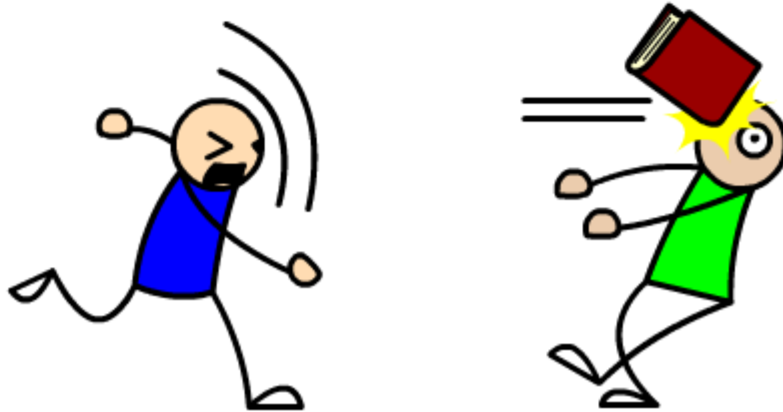
Бонус!

А можно ли быстрее?

7 группа: N ≤ ... (+INF баллов)

- Разложить N на множители можно за $O(N^{0.25} \log(N))$ ρ-методом Полларда, хотя наука умеет и сильно быстрее =)
- Заметим, что динамику по K можно вычислять с помощью быстрого возведения в степень!
- Оказывается, что двойки в разложении N в оптимальном ответе выгодно выделять в отдельные множители. Тогда от них можно избавиться с самого начала и Div(N) и Pairs (N) значительно уменьшаться!
- В итоге сложность $O(Pairs(N/2^{max})\log(K) + N^{0.25} \log(N))$

Задача «Британские учёные и Василий»



Автор идеи, разработчик и автор разбора:
Михаил Пядёркин

Формальная постановка

Будем называть *словарем* набор слов

Также будем говорить, что строка *порождается словарем*, если она представляется в виде конкатенации слов из словаря

Дана строка из букв a, b и c. Постройте словарь, порождающий данную строку, состоящий из минимального количество слов, каждое слово которого имеет длину, не превосходящую L

Решение на 10 баллов
“Я не хочу думать”

Необходимо представить строку в виде конкатенации слов из словаря.

Переберем все возможные разбиения строки на подстроки. Размер словаря есть количество различных среди них.

Выбираем оптимальное разбиение.

aba ab aba

Решение на 20 баллов "А строка-то из букв *a* и *b*"

Заметим, что так как строка состоит лишь из букв *a* и *b*, то ответ не превосходит 2.

abaabaaba

Осталось проверить, что не существует ответа 1.

Это бы означало, что строка периодична с периодом не больше L: переберем длину периода и проверим.

aba aba aba

Решение на 30 баллов
“А что толку от буквы *c*?”

Заметим, что так как строка состоит лишь из букв *a*, *b* и *c*, то ответ не превосходит 3.

abcabcababc

Проверять существования словаря из одного слова мы научились в прошлой серии. Теперь необходимо научится делать то же для двух слов.

abc abc ab abc

Откуда берутся... слова в словаре?

Отметим, что каждое из слов искомого словаря, конечно, должно быть подстрокой исходной строки

abcabcababcababc

Переберем подстроку, отвечающую первому слову.

Затем переберем вторую подстроку, отвечающую второму слову. Осталось научиться проверять, порождается ли наша строка этими двумя словами.

Как проверить? По-простому!

Очевидно, пустая строка порождается словарем из слов w1 и w2. Пустая строка - это префикс длины 0.

Если префикс длины *l* порождается, а далее за ним следует какое-либо из слов, то и больший префикс порождается.

abca bab bab|abca bababcabab

Будем идти слева направо, помечая порождаемые префиксы, и пытаться прикладывать каждое из слов.

Решение на 50 баллов
“Каждый с чего-то начинал”

Заметим, что первое вхождение первого слова обязательно находится в начале строки.

abc abc ab abc

Асимптотика решения в “лобовой” реализации *O(L * NL * NL)*, в реальности - сильно быстрее.

Первая идея

Пусть существует ответ, в котором одно слово является префиксом другого.

abc abc ab abc ab

Тогда “откусив” одно слово от другого, мы получим словарь из подстрок, который ничем не хуже.

ab c ab c ab ab c ab

Какой толк от беспрефиксности?

Заметим, что для того, чтобы проверить, что строка порождается беспрефиксным словарем, можно использовать “жадный” алгоритм: если слово можно прочитать с текущего момента, то это нужно делать.

abc abc abc cab abc abc cab

Где кончается реальность, и начинается… второе слово

Таким образом, если первое слово фиксировано, то его нужно прикладывать до тех пор, пока это возможно. Здесь и будет начинаться второе слово.

abc abc abc cab abcabccab

Прикладывать по-простому и понадеяться на лучшее. 60+ баллов.

Перербем первое слово. Приложим его до тех пор, пока возможно.

Переберем второе слово с места остановки.

Далее, с текущей позиции пытаемся продвинутся либо по первому слову, либо по второму.

Если не продвинулись - беда.

Если дошли до конца - победа!

Нет, это никакой не L^3N!

Пусть длина первого слова l_1, а длина второго слова l_2. Тогда количество прикладываний точно не больше, чем $n/min(l_1,l_2)$. Каждое прикладывание работает не более, чем за L. Суммируем:

$$\sum_{l_1=1}^{L}\sum_{l_2=1}^{L}\frac{NL}{\min(l_1,l_2)} \le \sum_{l_1=1}^{L}\sum_{l_2=1}^{L}\frac{NL}{l_1}+\frac{NL}{l_2} = 2NL^2\sum_{x=1}^{L}\frac{1}{x} \sim 2NL^2 \ln L$$

Сначала делать то, что легче!

На самом деле нетрудно заметить, что если сначала пытаться приложить более короткое слово, а лишь затем более длинное, то если мы выполнили сравнение x символов, то мы либо продвинемся на x символов, либо остановимся.

Это даст асимптотику $O(NL^2)$

Строковый алгоритм и 80+ баллов

В предыдущих решениях мы использовали лишь наивный метод сравнения строк.

Если же использовать, к примеру, хеширование, то прикладывание будет выполняться не за $O(L)$, а за O(1

Таким образом, из асимптотики исчезает одно L:

$O(NL \ln L)$

z-функция тоже работает

Посчитав один раз z-функцию для исходной строки, мы можем прыгать по первому слову за O(1): для этого достаточно сравнить значение z-функции текущей позиции с длиной слова (так как первое слово всегда начинается с первой позиции!).

abc abc ab abc abc ab

0 0 0 5 0 0 2 0 8 0 0 5 0 0 2 0

Второе слово съела корова!

Зафиксиров первое слово, мы можем опять вычислить z-функцию для оставшейся строки, что позволит нам аналогичным образом прыгать по любому второму слову (они начинаются все в одной и той же позиции!)

abc abc cbabcabccb

Магия строк и 100 баллов

Очевидно, что ни первое, ни второе слово не выгодно делать периодичными.

Оказывается, если перебирать лишь непериодичные, то это будет всегда работать очень быстро.

Однако, хотелось бы доказать подобное отсечение.

Написать легче, да и доказать можно

Вместо этого рассмотрим другое отсечение: пусть мы зафиксировали первое слово, и теперь перебираем различные вторые слова.

Пусть при прикладывании какого-либо второго слова мы пришли в позицию, в которой уже находились с тем же самым первым словом и каким-либо другим вторым.

Тогда процесс прикладывания **можно** остановить!

Что значит “можно”?

Пусть мы пришли в какую-либо позицию с помощью первого слова w_1 и двух разных вторых слов w_2 и w_3. Тогда либо словарь $\{w_1, w_3\}$ не порождает строку, либо она порождается и другим словарем из двух слов, суммарная длина слов которого строго меньше.

Таким образом, если первое слово перебирается в порядке возрастания длины, то при прикладывании второго никакую позицию не следует посещать дважды

Доказательство

Пусть мы пришли в какую-либо позицию с помощью первого слова w_1 и двух разных вторых слов w_2 и w_3. Тогда утверждается, что и w_1, и w_2, и w_3 порождаются некоторым словарем $\{t_1, t_2\}$.

Более общее утверждение: пусть некоторая строка порождается некоторым словарем из трех слов двумя различными способами. Тогда этот словарь “поглощается” некоторым словарем из двух слов.

Доказательство утверждения

Очевидно, что в таком случае словарь не беспрефиксный. Следовательно, одно слово является префиксом другого. Произведя “откусывание” перейдем к словарю с меньшей суммарной длиной слов.

Если же в какой-то момент разбиения совпали, то одно слово было конкатенацией двух других, а значит, его можно было выбросить.

Честный путь получения 100 баллов

- понять, что ответ не больше 3
- понять, что ответ можно выбрать беспрефиксным
- подсчет z-функции
- перебрать первое слово в порядке возрастания длины
- перебрать второе слово
- прикладывать второе слово, но при попадании в уже посещенную позицию останавливаться
- O(NL) с очень маленькой константой

Способ честный, но непонятный

- понять, что ответ не больше 3
- понять, что ответ можно выбрать беспрефиксным
- подсчет z-функции
- перебрать первое и второе слова
- не перебирать периодичные

Ограничения позволяли использовать другие, неасимптотические, оптимизации для получения 100 баллов

Нечестные способы

- не перебирать первую строку периодичной
- отсечение по времени: если ответ есть, то мы быстро его найдем
- хешировать по модулю 2^{64} (работает очень быстро, но неверно)

Быть может, мощь суффиксных структур позволяет решить эту задачу быстрее, чем за O(NL)? …

Задача «Пожар в НИИЧАВО»

Автор идеи, разработчик и автор разбора:
Максим Ахмедов

Формальная постановка

Дан граф, каждое ребро которого имеет свою температуру. Требуется ответить на несколько запросов вида “найти кратчайший путь из одной вершины в другую, если разрешается переходить с ребра на ребро только при условии не слишком большой разницы температур”.

Реализуем наивным образом

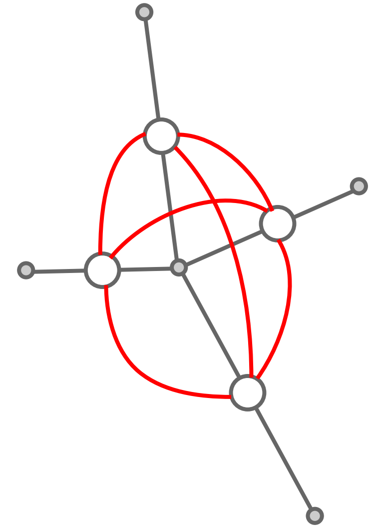
Из условия понятно, что состояние описывается ребром, на котором мы стоим, и направлением, куда мы идём.

Перейти из ребра (a, b, t_1) в ребро (b, c, t_2) можно, если $|t_1 — t_2| \le D$.

Делаем BFS. С каждого ребра можно перейти на O(N) рёбер, значит итоговая сложность — O(QMN).

Исследуем граф

Заметим, что мы делаем BFS на новом “рёберном” графе. Его вершинами являются рёбра исходного графа, при этом два ребра смежны, если в исходном графе они имеют общую вершину.

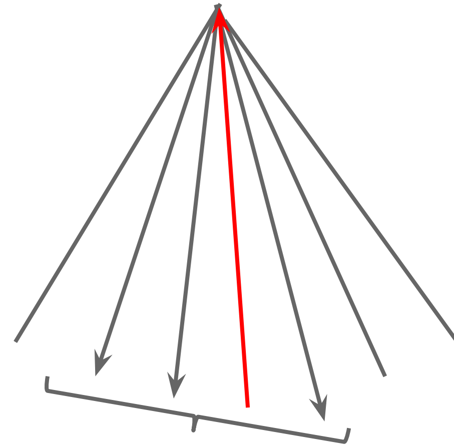


В случае вершины большой степени образуется полный подграф, в котором очень много рёбер — основное препятствие к быстрой работе BFS.

Улучшаем переход в BFS

Пусть мы идем по ребру (a, b) в направлении вершины b.

Рассмотрим исходящие рёбра в порядке возрастания температуры. Мы должны положить в очередь рёбра, находящиеся в некоторой окрестности нашего ребра. Вместо того, чтобы делать это наивно, можно воспользоваться какой-нибудь структурой данных.

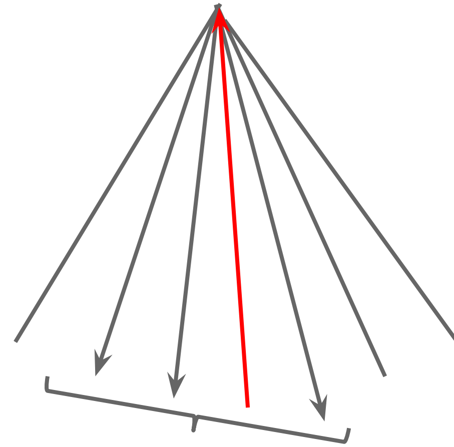


Требования к структуре данных

Эта структура данных должна уметь отвечать на запрос следующего вида.

Дано: ребро (a, b, t).

Требуется: добавить всё ребра в окрестности [t - d, t + d] в очередь, и удалить их из структуры (т. к. BFS не переходит дважды в одно и то же ребро).



Храним рёбра в set

Будем хранить неиспользованные рёбра, смежные вершине b, в сбалансированном двоичном дереве поиска по температуре (например, std::set на C++).

```
| while в очереди есть ребро (a, b, t):
| | выделяем в S[b] отрезок рёбер
| |    с температурами [t-d, t+d];
| | for (b, c, t2) in этот отрезок:
| | | добавляем в очередь ребро (b, c, t2);
| | | удаляем ребро (b, c, t2) из S[b];
```

Сложность — O(QM log N). К сожалению, с большой константой, набирает 60 — 70 баллов.

Используем дерево отрезков как set

Дерево отрезков очень часто может служить заменой сбалансированному двоичному дереву.

Храним в дереве отрезков единицы в позициях, соответствующих присутствующим рёбрам, в остальных позициях — нули.

Что выглядит как set и крякает как set?

```
| extract(node, left, right):
| if right < node.left or node.right < left
|     or count[node] == 0:
| | return;
| if node.left == node.right:
| | мы находимся в листе, соответствующем
| | неиспользованному ребру, обрабатываем его;
| extract(node.left_child, left, right);
| extract(node.right_child, left, right);
```

Сжатие путей

Поддерживаем для каждого ребра в последовательности два числа:

L[i] и R[i] — соответственно номера ближайшего слева неудалённого ребра и ближайшего справа неудалённого ребра.

Не будем пересчитывать эти величины после удаления — на помощь придёт эвристика сжатия путей!

Сжатие путей

```
get_left(int i):
| if del[L[i]]:
| | L[i] = get_left(L[i]);
| return L[i];

extract(i):
| p = L[i]
| while T[p] ≥ T[i] - d:
| | обрабатываем p
| | del[p] = false
| | p = get_left(p)
```

Переходим по ссылкам налево, сжимая пути, пока мы находимся в допустимой окрестности.

Затем действуем аналогично в направлении направо.

Получается решение за O (QM log N) с очень маленькой константой.

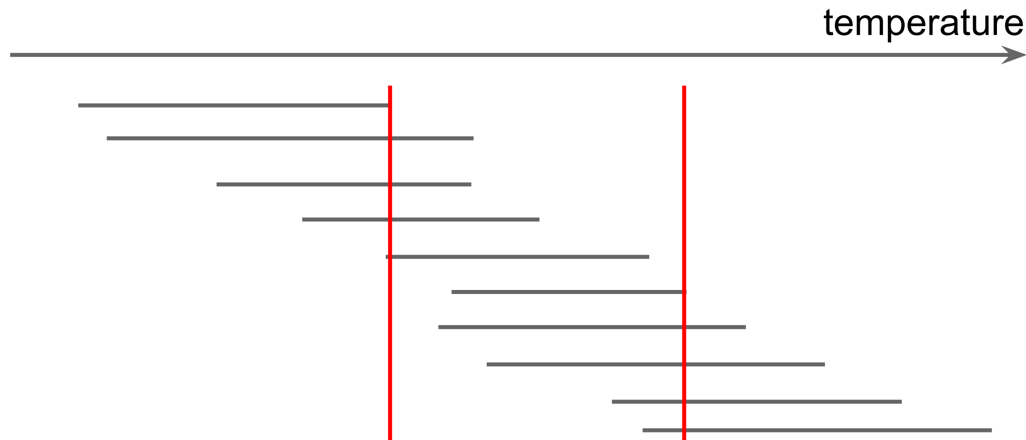
Линейное решение

Заметим, что допустимые окрестности образуют специфический набор отрезков.

Такой набор отрезков иногда называют “плывущее окно”. Никакие отрезок не лежит строго внутри другого отрезка.

Линейное решение

Такую систему отрезков всегда можно “прибить” каким-то количеством гвоздей таким образом, что каждый отрезок окажется прибитым ровно одним гвоздём.



Поддерживаем для каждого гвоздя самый левый и самый правый оставшийся элемент, относящиеся к нему.

Это даёт решение за O(QM).