

Тренировочный тур № 2

Разборы задач A–D

А. Будильники

Пусть $M = \sum_i m_i$ — общее число будильников. Если зафиксировать открытый интервал (L, R) , он может быть без звука тогда и только тогда, когда *внутри* него звонят не более $N - K$ различных будильников. Все эти будильники придётся не выбирать; остальные могут войти в искомые K . будильников ровно в моменты L и R не мешают.

Границы оптимального интервала можно взять среди $0, T$ и моментов будильников. Действительно, пока граница не проходит через следующий будильник, её можно сдвигать наружу, не меняя множество звонящих внутри. Поэтому отсортируем все пары (t, i) , где t — момент будильника, i — номер будильника, и объединим равные моменты времени. Добавим в отсортированный список граничные моменты 0 и T .

Дальше применим два указателя к этому списку. При фиксированной правой границе R храним частоты звонков каждого будильника *строго внутри* (L, R) и число d будильников с положительной частотой. Сдвигая R вправо, добавляем будильники из бывшей правой границы; пока $d > N - K$, сдвигаем L вправо и удаляем будильники из новой левой границы. После этого обновляем ответ величиной $R - L$. Все звонки одного момента добавляются и удаляются одной группой: это сохраняет открытость интервала и корректно обрабатывает совпадающие времена.

Лемма. Для каждой правой границы алгоритм находит самую левую допустимую левую границу.

Доказательство. Если расширить интервал влево, множество будильников, звонящих внутри, может только увеличиться. Значит, допустимые левые границы образуют суффикс списка. Когда правая граница движется вправо, самая левая допустимая граница не может сдвинуться влево. Поэтому удалять будильники слева, пока снова не станет $d \leq N - K$, достаточно. $\square$

При каждой правой границе самая левая допустимая левая граница даёт самый длинный допустимый интервал. Следовательно, максимум по всем правым границам — искомый ответ. Сортировка занимает $\mathcal{O}(M \log M)$, проход двумя указателями — $\mathcal{O}(M)$, память — $\mathcal{O}(N + M)$.

В. Ловля яблок

После яблока (t_1, x_1) робот успеет к яблоку (t_2, x_2) тогда и только тогда, когда

$$t_2 - t_1 \geq |x_2 - x_1|.$$

Раскрыв модуль, получаем два условия: $t_2 - x_2 \geq t_1 - x_1$ и $t_2 + x_2 \geq t_1 + x_1$. Поэтому удобно заменить координаты яблока на

$$u = t - x, \quad v = t + x.$$

Робот может поймать второе яблоко после первого ровно тогда, когда обе новые координаты не уменьшаются. Отсортируем яблоки по (u, v) , сохраняя их исходные номера. После такой сортировки координата u больше не мешает переходам; при равных u яблоки идут по возрастанию v . Каждому роботу достаточно хранить v последнего пойманного яблока.

Подзадача 3: $N \leq 500$

При $N \leq 500$ задачу можно решить и через паросочетание. Построим ориентированный граф яблок: ребро $i \rightarrow j$ есть, если робот успеет от i к j . Время вдоль каждого ребра строго возрастает, поэтому циклов нет. Разделим каждую вершину на левую и правую копии и для каждого перехода $i \rightarrow j$ проведём ребро $i_L j_R$ в двудольном графе. Найдём максимальное паросочетание алгоритмом Куна.

Рёбра паросочетания задают переходы роботов: у каждого яблока будет не более одного предшественника и одного следующего яблока. Цикл возникнуть не может, поэтому получатся непересекающиеся пути. Если в паросочетании m рёбер, путей ровно $N - m$. Наоборот, любое распределение яблок между K роботами содержит $N - K$ переходов, которые образуют паросочетание в построенном графе. Следовательно, минимум роботов равен $N - m_{\max}$. Для восстановления назначения начинаем в каждой вершине без предшественника и идём по выбранным рёбрам, помечая все вершины пути одним номером робота. Построение графа требует $\mathcal{O}(N^2)$ памяти и времени, алгоритм Куна в худшем случае — $\mathcal{O}(N^3)$ времени.

Подзадача 5: ответ без распределения

Если план ловли выводить не нужно, есть ещё одно решение. После сортировки по (u, v) маршрут каждого робота соответствует неубывающей подпоследовательности значений v . Яблоки из строго убывающей подпоследовательности не могут оказаться у одного робота, так что её длина не превосходит число роботов. Из теоремы о разбиении последовательности на неубывающие подпоследовательности следует и обратное: минимальное число таких подпоследовательностей равно длине наибольшей строго убывающей. Её можно найти за $\mathcal{O}(N \log N)$, например посчитав длину строго возрастающей подпоследовательности для чисел $-v$. Строгость здесь важна: одинаковые v одному роботу поручать разрешено.

Полное решение

Очередное яблоко с координатой v_a отдаём роботу, у которого последнее значение v максимально среди всех $v \leq v_a$. Если подходящего робота нет, создаём нового. Равенство допустимо: в условии достижимости стоят нестрогие неравенства.

Объясним, почему выгодно выбирать именно наибольшее подходящее v . Пусть оптимальное распределение уже совпадает с нашим до текущего яблока, но отдаёт его роботу с последним значением v_2 , тогда как мы выбрали робота с $v_1 \geq v_2$. Отдадим яблоко выбранному роботу, а дальнейшие маршруты этих двух роботов поменяем местами. Продолжение маршрута после v_1 можно поручить роботу с v_2 , поскольку $v_2 \leq v_1$; продолжение после текущего яблока остаётся допустимым. Число роботов не меняется. Если оптимальное распределение

создало для яблока нового робота, хотя подходящий уже был, его будущий маршрут также можно передать подходящему роботу, а прежнее продолжение этого робота — новому. Значит, жадный выбор можно делать на каждом шаге, не теряя оптимальности.

Чтобы получить требуемое распределение, вместе с последним v храним номер робота и записываем этот номер для каждого яблока в исходном порядке. Значения v удобно держать по убыванию. Бинарным поиском находим первое значение, не превосходящее v_a , и заменяем его на v_a ; порядок после замены сохраняется. Если такого значения нет, добавляем в конец массива нового робота. Сортировка и все поиски занимают $\mathcal{O}(N \log N)$ времени, память — $\mathcal{O}(N)$.

С. Закулисье

Подзадачи 1–2: малые координаты

Если координаты запросов не превышают 500 и шаблон мал, можно обойтись без работы с бесконечной сеткой. Построим её фрагмент с координатами от $-RC$ до $500 + RC$ по каждой оси и найдём компоненты связности обычным поиском в ширину. После этого каждый запрос сводится к сравнению номеров компонент двух клеток. Запас вокруг прямоугольника запросов нужен потому, что соединяющий путь может сначала выйти за его пределы. Благодаря периодичности обход, уходящий дальше этого запаса, можно заменить обходом через более близкие копии шаблона. При $R, C \leq 5$ получается поле порядка 550×550 клеток, поэтому такой способ подходит для групп с маленькими координатами.

Подзадача 3: A^*

Для подгруппы с $R, C, Q \leq 5$ и координатами до 10^5 полезен поиск A^* . Состояние — свободная клетка (x, y) бесконечной сетки, $g(x, y)$ — число шагов от начальной клетки, а оценка оставшегося расстояния равна

$$h(x, y) = |x - g_x| + |y - g_y|.$$

В очередь с приоритетом кладём состояния по значению $f = g + h$, храним лучшую найденную длину пути до каждой посещённой клетки и при извлечении состояния рассматриваем его свободных соседей. Один шаг меняет манхэттенское расстояние не более чем на единицу, поэтому h не завышает истинное расстояние и является согласованной эвристикой. Если цель извлечена из очереди, путь найден; для решения задачи сама его длина не нужна.

У этого подхода есть важная оговорка. Если цель недостижима, обычный A^* на бесконечной компоненте может искать бесконечно долго. Манхэттенская эвристика сама по себе не даёт условия, когда можно вывести No. Для строгого решения нужно дополнительно ограничить область поиска с доказательством достаточности этого ограничения либо отдельно проверить компоненты связности. Фиксированный лимит на число извлечённых клеток лишь превращает A^* в практическое частичное решение; на одних тестах он проходит, на других может ошибиться. Ниже приведён способ отвечать на оба вида запросов без такого лимита.

Полное решение

Периодичность позволяет свести задачу к конечному графу. Пусть базовый шаблон имеет R строк и C столбцов. Любую клетку (x, y) однозначно представим двумя частями:

$$a = (x \bmod C, y \bmod R), \quad T = \left(\left\lfloor \frac{x}{C} \right\rfloor, \left\lfloor \frac{y}{R} \right\rfloor \right).$$

Здесь a — клетка внутри базового шаблона, а T — номер его копии: первая координата копии растёт вправо, вторая — вверх. Все координаты из условия неотрицательны, поэтому остаток и частное вычисляются обычным целочисленным делением.

Создадим вершину для каждой свободной клетки шаблона. Соседние свободные клетки соединим рёбрами, *включая переходы через границу шаблона*. Например, если можно шагнуть вправо из правого столбца в левый столбец следующей копии, соответствующие вершины соединены ребром. Аналогично соединяем верхнюю и нижнюю границы. Так получается конечный граф на торе, содержащий не более RC вершин и не более $4RC$ ориентированных рёбер.

Каждому ориентированному ребру $a \rightarrow b$ приписываем вектор $w(a, b)$ изменения номера копии. Внутри шаблона он равен $(0, 0)$; переход вправо через границу даёт $(1, 0)$, влево — $(-1, 0)$, вверх — $(0, 1)$, вниз — $(0, -1)$. Обратное ребро имеет противоположную метку. При $R = 1$ или $C = 1$ могут возникать петли; их тоже необходимо учитывать.

Если две клетки шаблона лежат в разных компонентах этого графа, соответствующие клетки бесконечной сетки заведомо не связаны. Но принадлежности одной компоненте недостаточно: нужно ещё попасть в правильную *копию* конечной клетки. Именно эту информацию несут метки рёбер.

Для каждой компоненты конечного графа выберем корень и построим остов обходом в глубину или ширину. Для каждой вершины a вычислим $h(a)$ — сумму меток на остовном пути от корня до a . Если начать в нулевой копии корня и идти только по остову, мы придём в копию $h(a)$ клетки a .

Теперь рассмотрим любое ориентированное ребро $a \rightarrow b$ и вектор

$$z(a, b) = h(a) + w(a, b) - h(b).$$

Это смещение замкнутого обхода: сначала идём по остову от корня к a , затем по ребру к b , затем возвращаемся по остову от b к корню. Последний путь даёт $-h(b)$, поэтому знак в формуле именно *минус*. Для остовного ребра $z = (0, 0)$. Неостовные рёбра могут давать ненулевые векторы. Обратный обход даёт $-z$, так что любой такой вектор разрешено брать с целым коэффициентом, в том числе отрицательным.

Возьмём произвольный путь в конечном графе от a до b и сложим его метки. Если на пути встречается ребро $p \rightarrow q$, заменим его мысленно остовным маршрутом $p \rightarrow$ корень $\rightarrow q$. Разница между старым и новым смещением равна $z(p, q)$. Повторив замену для всех неостовных рёбер, получим базовое смещение $h(b) - h(a)$ плюс целочисленную сумму векторов z . В обратную сторону каждую такую сумму можно реализовать: дойти до корня, выполнить нужные замкнутые обходы в прямом или обратном направлении и затем пойти к b . Следовательно, множество всех возможных смещений между копиями клеток a и b имеет вид

$$h(b) - h(a) + H, \quad H = \left\{ \sum_e k_e z_e : k_e \in \mathbb{Z} \right\}.$$

Группа H общая для всей компоненты конечного графа.

Пусть начальная клетка принадлежит вершине a и копии S , а конечная — вершине b и копии G . Сначала сравниваем номера компонент a и b . Если они разные, выводим **№**. Иначе нужен путь со смещением $G - S$. По предыдущему выводу он существует тогда и только тогда, когда

$$q = G - S - h(b) + h(a) \in H.$$

Например, если все замкнутые обходы имеют нулевое смещение, то условие превращается в $S - h(a) = G - h(b)$. Это означает, что обе клетки принадлежат одной и той же конечной копии компоненты.

Остаётся определить, какие векторы входят в H . Если все $z_e = (0, 0)$, никакой замкнутый обход не переводит нас в другую копию корня. Каждая поднятая на плоскость компонента конечна, и проверка запроса сводится к $q = (0, 0)$.

Если все ненулевые z_e параллельны, соответствующая компонента тянется вдоль одного направления. Возьмём один из них $z_0 = (p, r)$, положим $m = \gcd(|p|, |r|)$ и выберем примитивное направление

$$d = (p/m, r/m).$$

Любой порождающий вектор единственным образом записывается как $z_e = k_e d$ с целым k_e . Обозначим $g = \gcd_e |k_e|$, где нулевые векторы можно пропустить. По тождеству Безу целочисленные комбинации чисел k_e — в точности кратные g . Поэтому

$$H = \{kgd : k \in \mathbb{Z}\}.$$

Для проверки $q = (q_x, q_y)$ сначала убеждаемся, что он коллинеарен d : $q_x d_y = q_y d_x$. Затем находим его целый коэффициент k при d по любой ненулевой координате и проверяем $k \equiv 0 \pmod{g}$. Если соответствующая координата d равна нулю, координата q также должна быть нулевой. Например, при $d = (1, 1)$ и $g = 2$ достижимы смещения $(\dots, (-2, -2), (0, 0), (2, 2), \dots)$, но не $(1, 1)$.

Простейший пример такого случая — шаблон из одной строки `.#`. Свободны только клетки с чётной координатой x , а вдоль оси y можно ходить сколько угодно. Замкнутые обходы в графе шаблона дают смещения $(0, 1)$ и $(0, -1)$ между его копиями. Поэтому две свободные клетки связаны ровно тогда, когда их координаты x совпадают, хотя остатков $x \bmod 2$ для проверки недостаточно.

Если среди z_e есть два неколлинеарных вектора, для этой задачи получаем $H = \mathbb{Z}^2$. Тогда любые две клетки, относящиеся к одной компоненте конечного графа, достижимы друг из друга независимо от номеров копий, и на запрос отвечаем **Yes**.

Здесь существенно, что граф действительно получен сворачиванием *плоской квадратной решётки*. Для произвольного графа с векторными метками два независимых вектора могут порождать лишь подгруппу конечного индекса, например $(2, 0)$ и $(0, 2)$ не дают $(1, 0)$. В нашем случае связный граф вложен в тор. Простой замкнутый путь, который не стягивается в точку, задаёт примитивное направление на торе. Если есть второй независимый путь, после разрезания тора вдоль первого между последовательными пересечениями со вторым найдётся обход, пересекающий первый один раз. Эти два обхода образуют базис смещений $(1, 0), (0, 1)$ с точностью до целочисленной замены координат. Поэтому индекс равен единице.

Для каждой свободной клетки храним номер компоненты и $h(a)$. Для каждой компоненты поддерживаем один ненулевой вектор, пока не обнаружили независимое направление. Если ненулевых векторов нет, достаточно проверки $q = (0, 0)$. Если найден неколлинеарный вектор, для этой компоненты на любой запрос отвечаем **Yes**. Иначе вычисляем примитивное направление d и НОД коэффициентов k_e .

Удобно сначала обойти весь граф и заполнить h , а затем одним проходом по ориентированным рёбрам вычислить все $z(a, b)$. Рёбро можно учитывать дважды: обратное даёт $-z$, а тип и НОД от этого не меняются. При миллионе вершин рекурсивный DFS способен переполнить стек, поэтому лучше пользоваться очередью или собственным стеком.

В запросе получаем остатки координат по C и R , находим соответствующие вершины и копии S, G , после чего выполняем описанные проверки. Гарантия условия позволяет не проверять, свободны ли запрошенные клетки. Координаты достигают 10^{12} , поэтому для копий, векторов h, z, q и произведений при проверке коллинеарности используем 64-битные целые числа.

Граф содержит $\mathcal{O}(RC)$ вершин и рёбер. Его обход и обработка всех векторов занимают $\mathcal{O}(RC)$ арифметических операций; каждый из Q запросов — $\mathcal{O}(1)$ таких операций. Итого $\mathcal{O}(RC + Q)$ времени и $\mathcal{O}(RC)$ памяти. НОД вычисляется алгоритмом Евклида; при учёте битовой сложности к соответствующим операциям добавляется логарифмический множитель.

Д. Кварковая микроскопия

Обозначим ответ на запрос в точке x через $(r(x), m(x))$: $r(x)$ — расстояние до второго ближайшего кварка, а $m(x)$ — сколько кварков находится ровно на этом расстоянии. В частности, при $m(x) = 2$ обе точки $x - r(x)$ и $x + r(x)$ заняты кварками.

$$Q \leq 15\,000$$

Можно искать кварки слева направо. Зная два соседних кварка $a < b$, запросим b . Если полученное расстояние не равно $b - a$ или кратность равна двум, следующий кварк стоит в $b + r(b)$. Иначе вторым ближайшим пока остаётся a . Будем искать бинарным поиском первую точку x , где перестанет выполняться $x - r(x) = a$; в этот момент следующий кварк находится в $x + r(x)$. На каждый кварк уходит около 60 запросов, всего порядка $60N$. Для первой группы даже не нужно отдельно восстанавливать самый левый кварк, потому что он уже известен.

$$Q \leq 5\,600$$

Можно сначала идти вправо шагами длины примерно $10^{18}/N$, пока вторым ближайшим остаётся a . Когда это перестанет быть верным, искомая граница заключена в одном таком шаге; там и запускаем бинарный поиск. Его длина уменьшается примерно до $\log_2(10^{18}/N)$, то есть до 54 запросов при $N = 100$. За всё решение полных шагов вправо будет порядка N : пройденное ими расстояние не превосходит длину всей прямой. Получается около $54N$ запросов.

$$Q \leq 3\,500$$

Полезно учесть длину уже известного промежутка $b - a$. Пусть l — последняя точка, где вторым ближайшим ещё является a , а h — первая проверенная точка, где это уже не так. Вместо обычной середины можно спрашивать около геометрического среднего расстояний от b до границ, то есть в точке

$$b + \left\lfloor \sqrt{(l + 1 - b)(h - b)} \right\rfloor.$$

Так мы быстрее подстраиваемся к масштабу следующего промежутка. Кроме того, не требуется сужать (l, h) до двух соседних целых точек. Достаточно получить $h - l \leq (b - a)/2$: переход ко второму ближайшему кварку происходит примерно посередине между a и следующим кварком c , а от этого места до c больше $(b - a)/2$. Значит, точка h ещё левее c , и ответ уже равен $c = h + r(h)$. Получается около $31N$ запросов.

Полное решение

Сначала найдём крайние кварки. В точке 1 все кварки лежат справа, поэтому второй слева находится в $1 + r(1)$. Зная его координату b , можно восстановить первый кварк запросами в b и $b - 1$. Пусть ответ в b равен (t, k) , а расстояние в $b - 1$ равно s . Если $k = 2$, первый кварк находится в $b - t$; если в $b - 1$ кратность равна двум, он находится в $b - 1 - s$. При кратности один в обеих точках остаются случаи:

s	координата первого кварка
$t - 1$	$b - t$
$t \quad (t = 1)$	$b - t$
$t \quad (t > 1)$	$b - t - 1$

Если $s = t + 1$, в точках b и $b - 1$ вторым ближайшим остаётся третий кварк $c = b + t$. Двигаясь влево, найдём бинарным поиском границу, где вторым ближайшим вместо c становится первый кварк. Пока вторым остаётся c , выполняется $x + r(x) = c$; в точке по другую сторону границы первый кварк равен $x - r(x)$. При равенстве расстояний его также сразу выдаёт кратность два.

Такие же запросы от правого конца отрезка дают два крайних кварка справа. Удобно мысленно отразить координаты заменой $x \mapsto 10^{18} + 1 - x$: тогда для правого края действует ровно тот же способ. Теперь известны самый левый и самый правый кварк.

Пусть известны кварки в точках $a < z$. Нам нужно найти кварк внутри (a, z) либо понять, что внутри нет ни одного или есть не более одного. Запросим середину $x = \lfloor (a + z)/2 \rfloor$ и положим $L = x - a$. Кварки вне $[a, z]$ не могут оказаться ближе к x , чем соответствующий крайний кварк, поэтому на ответ влияют только кварки этого отрезка.

Если $r(x) > L$, внутри кварков нет. Если $r(x) = L$ и кратность равна одному, внутри может быть не более одного кварка. Если кратность равна двум, посмотрим на точки $x - r(x)$ и $x + r(x)$: они обе заняты. Если одна из них лежит строго внутри (a, z) , мы нашли новый кварк. Если это ровно a и z , внутри может оставаться только один кварк. Последний случай возникает, когда длина отрезка чётна.

Остаётся $r(x) < L$ и $m(x) = 1$. Тогда один из двух кандидатов $x - r(x)$ и $x + r(x)$ обязательно является внутренним кварком. Чтобы выбрать нужного, запросим $x + 1$; пусть там расстояние равно s .

- Если кратность в $x + 1$ равна двум, внутренний кварк находится в $x + 1 - s$.
- Если $s = r(x) + 1$, подходит левый кандидат $x - r(x)$; если $s = r(x) - 1$, правый $x + r(x)$.
- Если $s = r(x)$, запросим $x + r(x)$. При кратности два ответ этого запроса сразу указывает внутренний кварк слева от точки запроса. Иначе сделаем последний запрос в $x - 1$. Если его радиус равен $r(x) - 1$, кварк стоит в $x - r(x)$, в противном случае — в $x - r(x) + 1$.

Когда $r(x) = 1$ и кратность равна одному, середина x сама занята кварком: иначе по обе стороны от неё должны были бы стоять кварки на расстоянии один и кратность была бы равна двум. Поэтому сразу возвращаем x как найденный внутренний кварк. Различать кандидатов $x - 1$ и $x + 1$ здесь не требуется, и запросы в соседние точки можно пропустить.

На один отрезок уходит не больше четырёх запросов. Первый делаем в середине; если он не определил внутренний кварк, спрашиваем $x + 1$. Только когда радиус там остался прежним и кратность не помогла, могут понадобиться ещё запросы в $x + r(x)$ и $x - 1$. Сравнение ответов работает потому, что при сдвиге точки наблюдения на единицу каждое расстояние, а значит и расстояние до второго ближайшего кварка, меняется не более чем на один. Если кратность равна двум, обе точки на данном расстоянии заняты, и выбирать между ними уже не нужно.

Когда нашёлся внутренний кварк p , повторим ту же процедуру для (a, p) и (p, z) . Если внутренних кварков нет, закончим с отрезком. Если ясно лишь, что их не больше одного, пока запомним этот отрезок: позже уже будут известны его соседи снаружи. Так рекурсия перечислит найденные кварки по возрастанию и оставит между ними только промежутки с нулём или одним ещё не найденным кварком.

Разберём такой промежуток (b, c) . Пусть a — уже известный кварк непосредственно слева от b , если он существует. В точке b запрос даёт (r, k) . Если $b + r \geq c$, промежуток пуст. Если $b + r < c$ и либо $k = 2$, либо a нет, либо $a < b - r$, единственный внутренний кварк стоит в $b + r$. В оставшемся случае $a = b - r$: расстояние r относится к левому кварку, а искомый, если он есть, лежит правее $b + r$. Если $b + r \geq c - 1$, его уже негде разместить.

Запрос в c часто снимает эту неопределённость. Пусть там получено расстояние s . Если $c - s \leq b + r$, внутреннего кварка нет: иначе он лежал бы между $b + r$ и c и был бы ближе к c , чем s . Если кратность в c равна двум, кварк стоит в $c - s$. Иначе запросим $c - 1$ и обозначим его ответ через (t, k) . При $t = 1$ искомый кварк находится в $c - 1$, если $k = 1$, и в $c - 2$, если $k = 2$. При $t = s - 1$ он находится в $c - s$. Остальные случаи разберём бинарным поиском.

Если и эти запросы не определили ответ, используем бинарный поиск между b и c . Пока вторым ближайшим остаётся a , выполняется $x - r(x) = a$. При движении вправо впервые нарушиться это равенство может только тогда, когда роль второго ближайшего перейдёт к неизвестному кварку; его координата будет $x + r(x)$. Если дошли до c , так и не встретив внутренний кварк, промежуток пуст. Случай равных расстояний виден по $m(x) = 2$ и позволяет закончить поиск сразу.

Почему этого достаточно? В каждом рекурсивном вызове мы либо находим новый кварк,

либо доказываем, что внутри осталось не более одного. Поздняя проверка каждого такого промежутка либо находит этот единственный кварк, либо доказывает, что его нет. Крайние кварки уже известны, следовательно, после обработки всех промежутков найдены все N координат.

Число рекурсивных отрезков линейно по N : найденный внутренний кварк делит отрезок на два, а каждый конечный отрезок проверяется один раз. На каждый из них уходит не больше четырёх запросов. Дополнительные бинарные поиски нужны только для тех одно-точечных промежутков, которые не разрешились запросами у концов. Ответы на повторные запросы мемоизируются.