Разбор задачи «Скобки»

Автор задачи: Антон Банных
Тесты подготовил: Сергей Мельников
Разбор подготовил: Виталий Гольдштейн

Задана правильная скобочная последовательность.

Задача заключалась в том, чтобы найти количество позиций, в которые можно поставить две скобки, чтобы последовательность осталась правильной скобочной последовательностью.

Пару скобок в порядке (и) можно поставить в любое место скобочной последовательности. Поэтому ответ равен как минимум $(N + 1) * (N + 2) / 2$

Далее будем рассматривать позиции, в которые можно поставить скобки в порядке) и (.

O(n³) — 40 баллов

Переберем пары позиций, в которые будут вставлены скобки. После вставки проверим, является ли скобочная последовательность правильной.

O(n²) — 70 баллов

Вместо скобочной последовательности s рассмотрим последовательность чисел a:

- $a[i] = 1$, если $s[i] =$ (
- $a[i] = -1$, если $s[i] =$)

Балансом скобочной последовательности называется разность количества открывающих и закрывающих скобок.

Посчитаем последовательность b – баланс скобок для каждого префикса последовательности

- $b[0] = 0$
- $b[i] = b[i - 1] + a[i]$

Скобочная последовательность является правильной, если для любого префикса последовательности баланс неотрицательный, а так же баланс всей последовательности равен 0.

Скобку) можно поставить после любого i, что $b[i] > 0$.

Пусть закрывающая скобка поставлена после позиции i, тогда баланс новой скобочной последовательности на позициях больших i уменьшиться на 1. Так как баланс префиксов должен быть неотрицателен, то (необходимо поставить в любую позицию до 0. Таким образом, найдем в последовательности b позицию ближайшего справа к i нуля. Пусть это позиция равна r. Тогда к ответу необходимо прибавить $r - i$.

O(n) — 100 баллов

Найдем позиции всех нулей в последовательности b. Пару скобок) и (можно ставить в любые позиции между подряд идущими нулями. Если количество позиций между нулями равно n, то к ответу необходимо прибавить $n * (n + 1) / 2$

Разбор задачи
«Школа олимпийского резерва»

Автор задачи: Елена Владимировна Андреева
Тесты подготовили: Сергей Копелиович и Сергей Мельников
Разбор подготовили: Елена Владимировна Андреева и Сергей Копелиович

Задача заключалась в следующем: есть N человек 1994-1996 годов рождения. Все они набрали на тестировании различное число очков. Нужно выбрать M94, M95 и M96 соответственно человек каждого года рождения, чтобы M94 + M95 + M96 было равно M, величина |A – M94| + |B – M95| + |C – M96| была минимальной, и было выполнено неравенство S94 > S95 > S96 (S94 – балл последнего выбранного человека 1994-го года рождения, S95 и S96 – по аналогии). Также, если мы выбираем человека Y-го года рождения, мы должны брать всех людей того же года рождения, набравших больше очков.

Краткое описание правильного решения:
Быстрый перебор M94, M95, M96 и проверка удовлетворяющих всем условиям троек M94, M95, M96 на оптимальность.

Далее подробно описаны некоторые из возможных решений этой задачи.

$O(n^3)$ — 25 баллов

Создадим массивы T94, T95, T96 – баллы, набранные участниками соответствующего года рождения. Переберем тройки чисел M94, M95, M96. Для каждой проверим, что M94 + M95 + M96 = M, а T94[M94] > T95[M95] > T96[M96]. Из всех таких выберем тройку с оптимальной величиной |A – M94| + |B – M95| + |C – M96|.

$O(n^2)$ — 50 баллов

Изменим предыдущее решение: перебирать M96 не нужно. Т.к. M96 можно вычислить, зная M94 и M95, по формуле M96 = M – M94 – M95. Нужно не забыть проверить, что вычисленное таким образом M96 лежит в диапазоне от 1 до N96 (количество элементов в T96).

O(nlogn) или O(n) — 100 баллов

Пусть массивы T94, T95, T96 отсортированы по убыванию. Переберем M95. Так как T94[M94] > T95[M95], существует такое минимальное m94, что M94 ≤ m94. Такое m94 можно найти или бинарным поиском по массиву T94, или методом двух указателей. T96[M96] < T95[M95], поэтому существует такое максимальное m96, что M96 ≥ m96, его можно найти так же, как и m94.

Пересчет величин m94 и m96 методом двух указателей при увеличении M95 на 1:

Пока m94 + 1 ≤ N94 и T94[m94 + 1] > T95[M95] нужно увеличить m94 на 1

Пока m96 ≤ N96 и T96[m96] > T95[M95] нужно увеличить m96 на 1

Если M94 обозначить за X, то M96 = M – M95 – X и нам нужно минимизировать функцию F(X) = |(M – M95 – X) – C| + |X – A| + |T95[M95] – B|. При этом должно быть верно, что 1 ≤ X ≤ m94 и m96 ≤ M – M95 – X ≤ N96. Это равносильно следующему: M – M95 – N96 ≤ X ≤ M – M95 – m96. Т.е. X лежит в отрезке от L до R. Минимум такой функции обязательно достигается при одном из следующих X: L, R, A, M – M95 – C, 1, m94, N94.

Получилось следующее решение: перебираем M95, внутри цикла по M95 пересчитываем m94 и m96, далее проверяем указанные X на корректность (на вхождение в диапазон 1..N94), затем проверяем тройку $M94 = X$, M95, $M96 = M - M95 - X$ на оптимальность: вычисляем $|A - M94| + |B - M95| + |C - M96|$ и сравниваем с текущим минимумом.

Разбор задачи «Снова в космос»

Автор задачи: Елена Владимировна Андреева
Тесты подготовил: Юрий Петров
Разбор подготовил: Сергей Копелиович

Задача заключалась в следующем: дана прямоугольная матрица (прямоугольник) букв, требуется найти такую её минимальную подматрицу (подпрямоугольник), которой можно замостить плоскость так, чтобы из неё можно было вырезать исходную матрицу. Предлагалось два вида замощений – клетчатое и со сдвигом строк.

Решение за $O(n^4)$ для клетчатого разбиения:

Переберем W и H подпрямоугольника, которым замощается прямоугольник. Далее за $O(N^2)$ проверим корректность замощения.
Нужно проверять, что `s[x,y] = s[x+W,y] = s[x,y+H]`.

Решение за $O(n^5)$ для разбиения со сдвигом строк:

Кроме того, что описано выше, нужно перебирать сдвиг от 0 до W-1.

Решение за $O(n^2)$ для клетчатого разбиения:

Есть целых два красивых решения.

Оба они пользуются важной идеей: по X и по Y задачу можно решать независимо.

Сперва я расскажу вам более громоздкое.
Для каждой строки найдем за линейное время все такие T, что s[T+i] = s[i]. Если посчитать Z-функцию, то T нам подходит, тогда и только тогда, когда z[T] + T = n. Теперь найдем минимальное T, которое подходит всем строкам матрицы. Это и есть ширина подпрямоугольника, которым мы замощаем матрицу. Чтобы получить высоту, нужно решить аналогичную задачу для столбцов.

Теперь более короткое решение. Требует знание и понимание hash-ей.
Посчитаем массив hash-ей столбцов и найдем период этого массива. Это ширина. Соответственно, период массива hash-ей строк будет высотой.

Решение за $O(n^4)$ для разбиения со сдвигом строк:

По направлению X задача не изменилась, поэтому ширину можно найти за $O(n^2)$, так как описано выше. Теперь можно перебрать высоту и сдвиг и сделать проверку за $O(n^2)$. Далее, чтобы улучшить это решение научимся делать проверку быстрее.

Решение за $O(n^3)$ для разбиения со сдвигом строк:

Проверять можно сравнением строк hash-ами за O(1). Как это применить? Если мы считаем, что высота H, а горизонтальный сдвиг = S, то str[i] должна совпасть с str[i+H] со сдвигом S. Это сравнение мы и будем делать за O(1) hash-ами.

Решение за $O(n^2)$ для разбиения со сдвигом строк:

Не будем останавливаться на достигнутом и скажем, что если высота равна H, сдвиг равен S, то подпрямоугольник (0,0) – (N-S,N-H) равен подпрямоугольнику (S,H)-(N,N). Утверждается, что если это равенство выполнено, то пара H и S нам подходит. А сравнить подпрямоугольники можно теми же hash-ами за O(1).

Как считать hash-и для подпрямоугольников? Hash – это частичная сумма величин sum[i,j] = a[i,j] * P^i * Q^j, где P и Q – большие простые числа.

Hash[LX..RX, LY..RY] = (sum[RX,RY] - sum[LX-1,RY] -
- sum[RX,LY-1] + sum[LX,LY]) * P^{N-LX} * Q^{N-LY}.

Альтернативное решение за O(n^2logn):

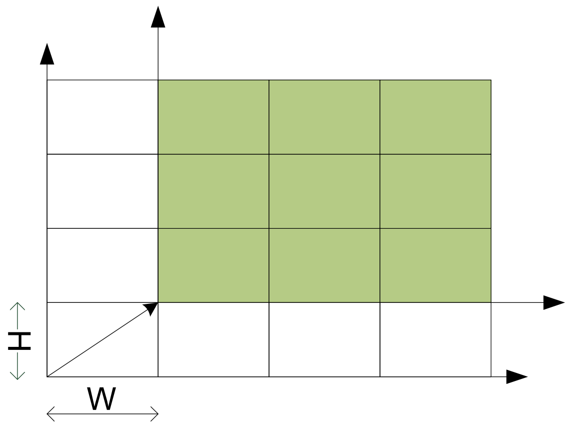
Вычислим W, зафиксируем H. Теперь сожмем за O(N^2) полоски ширины H в массив hash-ей столбцов. Теперь у нас есть задача размера N x N/H. Мы можем ее решить за время O(N^2/H). Для этого нужно для всех строк новой сжатой матрицы посчитать множество допустимых сдвигов. Общее время решения получается $\sum_{H=1..N} N^2 / H$.

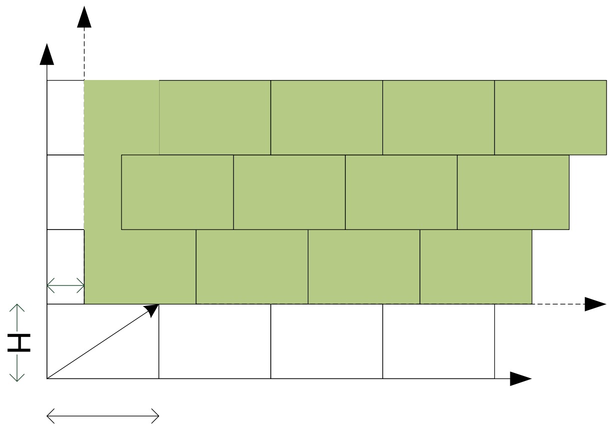
Что равно O(N^2logN).

Бонус этого решения в том, что оно нигде не использует то, что сдвиг всех строк одинаков. Оно может из полученных сдвигов потом попытаться выбрать одинаковый для всех строк, а может выбрать для каждой строки какой-то свой сдвиг.

КОНЕЦ

Вспомогательные картинки:





Разбор задачи «Почта»

Автор задачи: Георгий Корнеев
Тесты подготовили: Сергей Копелиович и Сергей Назаров
Разбор подготовил: Георгий Корнеев

Будем последовательно пробовать ставить логистический центр в первом, втором, третьем и так далее почтовых отделениях. После этого, задача сведется к решению N подзадач, в каждой из которой требуется найти время, затрачиваемое Е-мобилем при старте из соответствующего почтового отделения.

Будем решать каждую из полученных подзадач, непосредственно подсчитывая время проезда по каждому участку, начиная с заданного. В этом случае для каждой подзадачи потребуется сделать O(K) операций, где K – сумма E_i для всех участков. Таким образом, вся задача будет решена за O(NK) операций. Такое решение набирало 20 баллов.

Рассмотрим, как зависит время, затрачиваемое Е-мобилем для проезда по участку (t_{transit}) от времени въезда на него (t_{in}). В соответствии с условием задачи, скорость на каждом участке является кусочно-постоянной функцией от времени. Таким образом, пройденное расстояние, является кусочно-линейной функцией от времени. При этом точки излома будут соответствовать временам, в которые изменяется скорость (рис. 1).

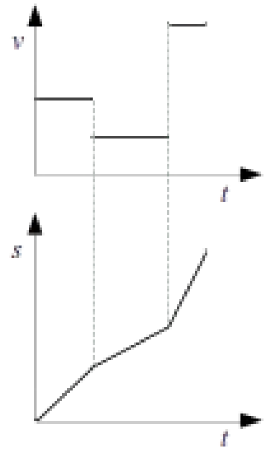


Рис. 1. Зависимость скорости и расстояния от времени

Для проезда всего участка Е-мобилю требуется проехать расстояние d (длину участка). Отметим, что в процессе проезда участка с разными временами въезда, но одинаковыми скоростями на начале, в середине (возможно, несколько разных скоростей) и конце участка, время выезда будет линейно зависеть от времени въезда (рис. 2).

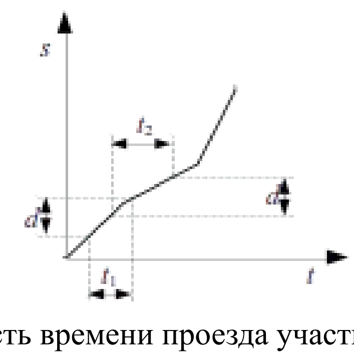


Рис. 2. Зависимость времени проезда участка от времени выезда

Если в процессе решения время въезда на участок только увеличивается, то мы можем локально пересчитывать зависимость времени проезда участка от времени въезда на него

($t_{transit}(t_{in})$). Для этого рассмотрим два случая:

- в процессе пересчета скорость на начале и конце участка не изменяется, тогда $t_{transit}(t_{in}+\Delta t) = t_{transit}(t_{in}) + \Delta t(v_{in}/v_{out})$, где Δt – изменение времени въезда на участок;
- если скорость в начале или конце участка в процессе пересчета изменяется (так как произошел переход на новый интервал скорости движения в начале или конце участка), то разобьем пересчет на части, для каждой из которых скорости не изменяются. Это сводит данный случай к предыдущему.

Таким образом, в случае увеличения t_{in}, для i-го участка мы можем в явной форме подсчитать $t_{transit}$ для требуемых t_{in} за $O(E_i)$ операций.

Что бы использовать эту возможность «удвоим» кольцевой маршрут, считая, что после N-го участка опять следует первый, за ним второй и так далее (рис. 3)

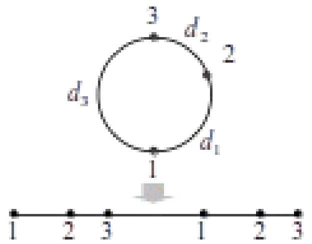


Рис. 3. «Удвоение» маршрута

Обозначим в этой последовательности отделения через U_i ($1 \le i \le 2N$). При этом исходная задача сведется к нахождению времени проезда от отделения U_i до отделения U_{i+N} для i от 1 до N. Если вести расчет в обратном порядке (по i от N до 1), то время въезда на каждый рассматриваемый участок будет только увеличиться. Таким образом, на рассмотрение всех участков потребуется $O(K)$ операций на пересчет скоростей и $O(N^2)$ операций на нахождение времен выезда с участков (N раз по N участков). При этом общее время работы программы будет $O(K + N^2)$. Такое решение набирало 40 баллов.

Рассмотрим функцию времени выезда с участка от времени въезда на него ($t_{out}(t_{in}) = t_{transit}(t_{in}) + t_{in}$). Данная зависимость будет кусочно-линейной функцией, точки излома которой соответствуют моментам времени, когда скорость движения меняется в начале или конце участка (рис. 2, 4).

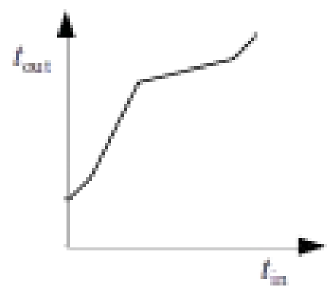


Рис. 4. Зависимость времени въезда на участок от времени выезда

Заметим, что зная $t_{out}(t_{in})$ для двух соседних отрезков мы можем построить $t_{out}(t_{in})$ для этой пары отрезков в целом. Построенная функция так же будет кусочно-линейной, при этом число точек излома будет суммой числа точек излома исходных участков. Это позволит, попарно объединяя соседние участки, построить дерево отрезков, которое позволит считать время проезда от отделения U_i до отделения U_{i+N} за $O(\log N)$ вычислений $t_{out}(t_{in})$ для некоторого подотрезка. При этом суммарное число точек излома на каждом уровне дерева будет $O(K)$. Построение такого дерева потребует $O(K \log N)$ времени и памяти.

Знание вида зависимости $t_{out}(t_{in})$ для некоторого подотрезка позволяет считать t_{out} для конкретного t_{in} за $O(\log K)$, найдя соответствующий «кусок» функции двоичным поиском. Таким образом, для вычисления время проезда от отделения U_i до отделения U_{i+N} за $O(\log^2 K)$

операций (log K запросов по O(log K) действий в каждом). Общее время решения составит O(K log $N + N \log^2 K$). Такое решение при правильной реализации набирало 94 балла, при этом в двух тестах решение не проходило по точности.

Для получения 100 баллов для данной задачи можно было рассмотреть зависимость скоростей от местоположения и времени (рис. 5). В каждой из полученных прямоугольных областей (некоторые из которых бесконечны по x или t) скорость постоянна, поэтому местоположение меняется линейно со временем (рис 6). Рассмотрим линии, соответствующие траекториям движения Е-мобиля для каждого почтового отделения. (рис. 7).

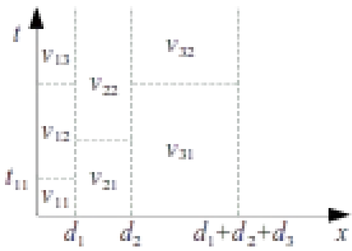


Рис. 5. Зависимость скоростей от местоположения и времени

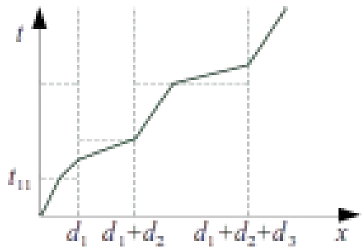


Рис. 6. Линейность времени в областях

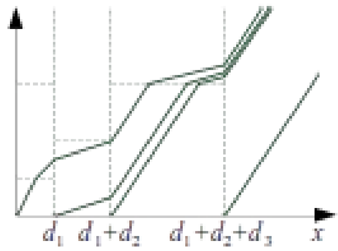


Рис. 7 Все траектории движения

Отметим, что в каждой области отрезки траекторий параллельны. Будем рассматривать области в порядке снизу-вверх, слева-направо. Для каждой области интересующие нас точки на траекториях (N точек) делятся на четыре класса (рис. 8):

- точки с $x = x_2$. Положение этих точек не изменяется;
- точки с $t = t_1$. Эти точки параллельным переносом переходят на правую или верхнюю части прямоугольника;
- точки с $x = x_1$ и $t < t_2$. Эти точки также параллельным переносом переходят на правую или верхнюю части прямоугольника;
- точки с $x = x_1$ и $t \geq t_2$. Положение этих точек не изменяется.

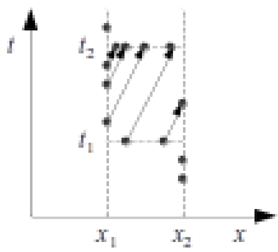


Рис. 8. Все траектории движения

Храня точки всех четырех типов в дереве отрезков с индексом по номеру траектории. На каждом этапе найдя в дереве отрезков указанные классы двоичным поиском, с помощью массовых операций на подотрезках можно получить дерево отрезков, Учитывающее рассмотренную область. Так как всего областей O(K) и для обработки каждой требуется O(log N) времени, то общее время работы составляет O(K log N). Такое решение набирало полный балл.

Разбор задачи «Парк аттракционов»

Автор задачи: Глеб Евстропов
Тесты подготовили: Сергей Поромов и Андрей Лопатин
Разбор подготовили: Андрей Лопатин и Елена Владимировна Андреева

Задача заключалась в следующем: есть N человек и $M \le N$ игровых автоматов. В каждый момент времени каждым игровым автоматом может пользоваться не более одного человека. Время игры на i-м автомате для любого человека равно t[i]. Процесс игры прерывать нельзя. Человек подходит к автомату и играет t[i] минут подряд. Нужно составить расписание игр так, чтобы каждый человек успел поиграть на каждом автомате и момент времени, когда все освободятся (поиграют на всех автоматах) был минимально возможным.

Предполагалось следующее решение этой задачи.

Сначала решим задачу для N=M (60 баллов).

Во-первых, заметим, что автомат с максимальным временем работы (обозначим это время T_{Max}) будет занят не менее чем N*TMax единиц времени. Таким образом, если нам удастся решить задачу за время N*T_{Max}, то это решение очевидно будет оптимальным.

Разобьём наш интервал времени длины $N * T_{Max}$ на N интервалов длины T_{Max}. В каждый из этих интервалов мы сопоставим одного участника одному автомату (занумеруем автоматы $i = 1, 2, 3, \ldots$) . Будем нумеровать интервалы j=0, 1, 2, …, N–1 – их время начала, соответственно, будет $T_{max}\times j$. Тогда решение задачи легко получить с помощью циклического сдвига перестановки 1, 2, …, N (напомним, что $M = N$).

Время	**Автомат 1**	**Автомат 2**	**Автомат 3**
0	1	2	3
T_{max}	3	1	2
$2T_{max}$	2	3	1

Легко видеть, что при таком подходе в каждом столбце и в каждой строке будут стоять различные числа – все участники от 1 до N также в порядке циклического сдвига, что и требуется для решения задачи.

Фактически i-й участник начинает с автомата номер i, переходя в очередной момент времени к автомату i +1, или автомату 1, если $i + 1 > M$.

Теперь решим задачу для N >= M (100 баллов)

Решение задачи получается из решения предыдущей добавлением “фиктивных” автоматов с большими номерами. Мы считаем, что участники, которые играют на автоматах $M + 1$, $M + 2$, …, N, на самом деле гуляют и отдыхают в это время. Мы можем на выбор – заполнить квадратную табличку и для вывода использовать только несколько столбцов, или заполнить только первые M столбцов.

	$i = 1$	$i = 2$	$i = 3$
$j = 0$	1	6	5
$j = 1$	2	1	6
$j = 2$	3	2	1
$j = 3$	4	3	2
$j = 4$	5	4	3
$j = 5$	6	5	4

Заметим, что можно было не заполнять таблицу в массиве, а сразу выводить ответ в выходной файл. Такое решение может выглядеть примерно так:

```
scanf ("%d%d", &n, &m);
int tmax = 0;
for (int i = 0; i < m; i++) {
  int t;
  scanf ("%d", &t);
  if (t > tmax) tmax = t;
}
printf ("%d\n", tmax * n);
for (int i = 0; i < n; i++) {
  puts ("");
  for (int j = n - i; j < m; j++)
    printf ("%d %d\n", j + 1, (j - n + i) * tmax);
  for (int j = 0; j < min (m, n - i); j++)
    printf ("%d %d\n", j + 1, (j + i) * tmax);
}
```

Разбор задачи «Велогонка»

Автор задачи: Сергей Волченков
Тесты подготовили: Сергей Назаров и Сергей Копелиович
Разбор подготовили: Сергей Назаров и Сергей Копелиович

В условии были указаны 4 подзадачи, рассмотрим решение каждой из подзадач отдельно.

Подзадача 1 (20 баллов)

$2 \le n \le 50$, $0 \le x_i \le 1000$, $0 \le v_i \le 1000$. Гарантируется, что существует ответ, в котором t – целое число, не превышающее 1000.

Для решения этой подзадачи достаточно моделировать движение велосипедистов. Будем последовательно получать положения велосипедистов в моменты времени $t = 0, 1, 2, \ldots$ В момент времени t i-й велосипедист находится в точке $x_i + v_i t$. Мы можем посчитать максимум и минимум величин $x_i + v_i t$ и посмотреть на их разность. Это расстояние между лидером и аутсайдером. Из таких разностей нужно выбрать минимум.

Подзадача 2 (20 баллов)

$2 \le n \le 200$.

Рассмотрим движение велосипедистов в течение некоторого небольшого промежутка времени $[t_1, t_2]$, в течение которого они не обгоняют друг друга. При этом не меняется ни лидер гонки, ни аутсайдер. В зависимости от их скоростей расстояние между ними либо увеличивается, либо уменьшается. Поэтому для того, чтобы узнать минимальное расстояние между лидерам и аутсайдером в течение этого промежутка времени, достаточно лишь подсчитать расстояние в момент времени t_1 и в момент времени t_2. Следовательно, нас интересуют только лишь те моменты времени, в которые велосипедисты обгоняют друг друга и начальный момент времени $t = 0$.

Решение для этой подзадачи следующее. Переберем все моменты времени, в которые велосипедисты обгоняют друг друга и в каждый такой момент времени вычислим расстояние от лидера до аутсайдера. Также подсчитаем это расстояние в начальный момент времени. Из всех этих расстояний выберем наименьшее.

Возможно $O(n^2)$ обгонов, для каждого обгона будем вычислять позицию лидера и позицию аутсайдера за $O(n)$, следовательно, получим решение за $O(n^3)$.

Подзадача 3 (30 баллов)

$2 \le n \le 2000$.

Заметим, что нас интересуют не все моменты времени, в которые происходят обгоны, а только лишь те моменты времени, в которых меняется лидер или аутсайдер. В самом деле, если лидер и аутсайдер не меняются на промежутке времени $[t_1, t_2]$, расстояние между ними меняется монотонно. Поэтому минимум этого расстояния достигается на одном из концов отрезка: либо в точке t_1, либо в точке t_2.

Будем моделировать движение велосипедистов, каждый раз получая их положения в моменты времени, когда меняется лидер или аутсайдер. Для каждого такого положения будем вычислять ответ. В процессе моделирования каждый велосипедист будет лидером и аутсайдером не более одного раза, поэтому мы рассмотрим $O(n)$ положений велосипедистов, следовательно, получим решение за $O(n^2)$.

Подзадача 4 (30 баллов)

$2 \le n \le 10^5$.

Расстояние между лидером и аутсайдером представляет собой некоторую функцию от времени *f(t)*, минимум которой необходимо найти. Если в течение некоторого момента времени скорость текущего лидера меньше скорости аутсайдера, то расстояние между ними уменьшается, следовательно и уменьшается значение функции *f(t)*. Если эти скорости равны, то *f(t)* остается постоянной. Если же скорость лидера больше скорости аутсайдера, то *f(t)* возрастает. Лидер и аутсайдер меняются с течением времени, но скорость лидера только лишь возрастает, а скорость аутсайдера лишь убывает. Поэтому, если в начальный момент скорость лидера меньше скорости аутсайдера, то *f(t)* сначала монотонно убывает до тех пор, пока скорость лидера меньше скорости аутсайдера, затем возможно остается неизменной, если скорость лидера равна скорости аутсайдера, а затем монотонно возрастает, если скорость лидера станет больше скорости аутсайдера. Если же в начальный момент времени скорость лидера больше или равна скорости аутсайдера, то минимум функции *f(t)* достигается в начальный момент времени $t = 0$.

Таким образом, минимум *f(t)* можно найти тернарным поиском.

Задачу можно также решать бинарным поиском. Рассмотрим функцию *v(t)*: разность скоростей между самым быстрым лидером (несколько велосипедистов могут быть лидерами одновременно) и самым медленным аутсайдером в момент времени *t*. Очевидно, что функция *v(t)* не убывает. При помощи бинарного поиска найдем наименьшее такое *t*, при котором $v(t) \ge 0$. В этот момент времени значение *f(t)* минимально.

Про погрешность.

Мысль первая:

Троичный поиск нельзя было писать как `while (R - L > 10^-9)`.

Если время вычислена с точностью $\pm 10^{-9}$, то f(t) вычислена с точностью $\pm 10^{-9} * f'(t) = \pm 10^{-2}$.

Правильный вид троичного поиска: `for i := 1 to 200 do`.

Мысль вторая:

Даже, если троичный поиск был написан правильно, в этой задаче могли возникнуть некоторые проблемы с погрешностью. Рассмотрим подробно то, как мы считаем f(t).

```
f(t) = max (x[i] + v[i] * t) - min (x[i] + v[i] * t).
```

Это выражение вида A – B, где A и B неотрицательные числа. Если величина A – B мала по сравнению с A и B, возможны проблемы с погрешностью.

Величины x[i] и v[i] не более 10^7. Величина t от 0 до 10^7. A и B от 0 до 10^{14}.
Рассмотрим тест:

2
0 10000000
10000000 9999999

$f(10^7) = 0$.
Тем не менее при $t = 10^7$, A и B равны $10^7 * (10^7-1) \sim= 10^{14}$.
Т.е. если мы их храним в вещественном типе double (15 знаков), посчитаны с точностью ±0.1.
Значит, величина A – B тоже посчитана с точностью ±0.1, что говорит о том, что погрешность (максимум из абсолютной и относительной) вычисления функции f = 0.1.
Чтобы побороться с погрешностью в данном случае будем вычислять
F = (x[i] + v[i] * t) – (x[j] + v[j] * t) другим способом: F = (x[i] – x[j]) + t * (v[i] – v[j]). Тогда пропадает вычитание величин порядка 10^{14}, получается разность величин порядка 10^7. Прием называется ``Вынесем t за скобки'' .

Мысль третья:
Можно было вместо того, чтобы минимизировать троичным поиском f(t), искать минимальное t, для которого $v_{max}(t) - v_{min}(t) \geq 0$ бинарным поиском. Основной бонус в том, что мы считаем разность величин порядка 107, эффект такой же, как и с выносом t за скобки.

Мысль четвертая:
Можно было на свою голову получить гораздо больше проблем с погрешностью, если поиск делать по промежутку не от 0 до 10^7 (10^7 объясняется тем, что $dx / dv \leq 10^7$), а от 0 до, например, 10^{18}. Тогда в соответствующей разности получались настолько большие числа, что помочь было уже не чем. Эффект проявлялся на тесте с *параллельными прямыми*. Чтобы понять, что есть ``параллельные прямые'' нужно нарисовать графики движения велосипедистов и увидеть на картинке множество прямых.

КОНЕЦ

Разбор задачи «Сад пермского периода»

Автор задачи: Павел Маврин
Тесты подготовил: Юрий Петров
Разбор подготовил: Павел Пономарев

Краткое условие

Задача заключалась в следующем: есть разбиение прямоугольника высотой H и шириной W на N квадратов. Нам даны координаты центров этих квадратов и координаты углов прямоугольников. Нужно восстановить размеры квадратов.

Основная идея.

Рассмотрим самый левый квадрат (с минимальным x) с координатами центра (x, y). Так как этот квадрат должен покрывать точку $(0, y)$, то сторона квадрата не может быть меньше $2x$, и не может быть больше $2x$, иначе он выйдет за границы прямоугольника. Значит можно покрывать прямоугольник квадратами слева направо, жадно выбирая размер квадрата. Для этого отсортируем центры квадратов в порядке возрастания x.

С целыми числами работать удобней, поэтому можно предварительно умножить все координаты центров, углов прямоугольника и размеры прямоугольника на 2.

Решения за O(NH) (40 баллов)

Для каждой целой координаты y_0 от 0 до H будем хранить, до какой координаты x покрыт прямоугольник на прямой $y=y_0$. Для этого заведем массив Rx размера $(H+1)$.

Рассмотрим i-ый квадрат в порядке сортировки квадратов по возрастанию координаты x его центра. Координаты его центра (x_0, y_0), а на прямой $y=y_0$, прямоугольник уже покрыт до $x=Rx[y_0]$. Тогда отрезок с концами $(Rx[y_0], y_0)$ и (x_0, y_0) может быть покрыт только этим квадратом, иначе все другие не рассмотренные еще квадраты покрывали бы (x_0, y_0), но сторона квадрата не может быть нулевой. Тогда размер i-ого квадрата это
$side_i=2(x_0-Rx[y_0])$. Теперь надо изменить массив Rx. Для этого для всех y от
$(y_0-side_i/2)$ до $(y_0+side_i/2)$ включительно обновим значение Rx,
$Rx=max(Rx, x_0+side_i/2)$.

Улучшение до O(NlogH) (80 баллов)

Можно воспользоваться деревом отрезков с операциями: присвоение значения на отрезке и взятия максимума в точке, чтобы делать все операции с Rx за $O(logH)$.

Улучшение до O(N²). (60 баллов)

Заметим, что нас интересуют значения Rx только в тех точках y, в которых есть центры. Заведем массив Sy, различных координат y точек центров всех квадратов, отсортированных по возрастанию. Тогда в $Rx[i]$ будем хранить до какого x покрыт прямоугольник на прямой $y=Sy[i]$. Бинарным поиском можно найти i, при котором $Sy[i]=y_0$, значит размер каждого квадрата можно найти за $O(LogN)$. Границы, в которых нужно изменить значения массива Rx, находятся с помощью двух бинарных поисков, и в худшем случае требуется изменить N значений массива Rx.

Использование обоих улучшений (100 баллов)

Можно также воспользоваться деревом отрезков с операциями: присвоение значения на отрезке и взятие максимума в точке для того, чтобы делать все операции с Rx за $O(logN)$. Тогда сложность алгоритма в этом случае получается $O(NlogN)$.

Разбор задачи «Распил бревен»

Автор задачи: Георгий Корнеев
Тесты подготовили: Сергей Мельников и Андрей Серовиков
Разбор подготовил: Сергей Мельников

Общие замечания

Заметим, что если внутри треугольника не лежит ни одной точки, то он не пересекает ни одного отрезка ломаной (так как есть точка с x = 0). Рассмотрим все точки всех сечений, необходимо построить треугольник так, чтобы все точки лежали вне или на его границе.

Подзадачи 2 и 4.

(Искомый треугольник прямоугольный)

Найдём для каждой точки максимальный прямоугольный треугольник, который не содержит эту точку. Пусть координата точки (x, y), тогда высота этого треугольника равна $|x|+y$, а площадь $(|x|+y)^2$.
Искомый треугольник содержит все эти точки, поэтому является минимальным из всех этих треугольников. Его площадь равна $(\min\{|x|+y\})^2$.

Подзадачи 1, 3 и 5.

(общий случай)

Отразим все точки из четвертой координатной четверти относительно оси Oy в первую четверть.
Рассмотрим теперь задачу: вписать в первую четверть треугольник одна сторона которого лежит на оси Ox, другая на оси Oy. Правильный ответ можно получить отразив треугольник относительно оси Oy.
На стороне оптимального треугольника лежит хотя бы две точки.

Подзадача 1. Решение за O(n^3)

- Перебираем пару точек
- Строим треугольник по этим двум точкам
- Проверяем что внутри треугольника нет других точек
- Выбираем треугольник с максимальной площадью

Подзадача 3. Решение за O(n^2)

- Выберем точку с максимальным углом в верхней полуплоскости,
  и с минимальным в нижней
- Остальные точки не могут образовать пару на стороне треугольника

Подзадача 3. Решение за O($n \log n$)

- Построим выпуклую оболочку
- Только точки, соседние на выпуклой оболочке, могут лежать на стороне треугольника