Олимпиада школьников по программированию «ТехноКубок» 2021

Второй отборочный тур

Разбор задач

A. Побег из тюрьмы

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Задача эквивалентна нахождению расстояния до самой далекой клетки из (x, y). Легко видеть, что один из оптимальных путей из (i, j) в (x, y) имеет форму буквы L, и расстояние просто равно манхэттенскому расстоянию между двумя клетками.

Наибольшее расстояние, которое заключенный может пройти, меняя строчки, равно $max(x - 1, n - x)$, а меняя строчки — $(y - 1, m - y)$. Поэтому ответ просто равен $max(x - 1, n - x) + max(y - 1, m - y)$.

B. Перекраска улицы

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Если мы хотим покрасить все дома на улице в цвет x, то легко видеть, что мы должны перекрасить все дома цветов, отличных от x, и не обязательно перекрашивать дома, уже имеющие цвет x. Мы можем использовать следующий жадный алгоритм для минимизации числа дней.

Найдем самый левый дом, еще не покрашенный в цвет x. Пусть этот дом имеет номер i. Тогда покрасим все дома в отрезке $[i, i + k - 1]$ в цвет x. Повторим, пока все дома не станут покрашенными в цвет x. Почему это решение оптимально? Когда мы нашли самый левый дом, имеющий цвет x, понятно, что так или иначе мы должны его перекрасить. Так как он самый левый, то все дома левее уже имеют цвет x. Чтобы увеличить наши шансы сразу покрасить больше домов, которые нужно будет покрасить, мы позиционируем отрезок, который мы красим, максимально сдвинув его вправо. Такой жадный алгоритм может быть реализован одним линейным проходом.

Как же выбрать цвет x, в который мы покрасим все дома? Так как количество цветов ограничено, мы можем попробовать все варианты и выбрать минимальный ответ.

Итоговая сложность: $O(n \cdot max(c))$.

Использование памяти: $O(n)$.

C. Попрыгунчик

ограничение по времени на тест: 1 секунда
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Заметим, что вместо удаления первых клеток мы можем за y секунд увеличивать необходимое значение p на единицу, эти операции эквивалентны. Давайте переберем, каким окажется итоговое значение p после удаления клеток из начала, пусть это q ($p \le q \le n$). Тогда нужно добавить платформы в те из клеток q, $(q + k)$, $(q + 2k)$, и так далее, в которых они отсутствуют.

Вычислим массив c_i — количество клеток, в которых отсутствует платформа, среди клеток i, $(i + k)$, $(i + 2k)$, и так далее. Его можно вычислить методом динамического программирования, проводя вычисления в порядке от больших i к меньшим: $c_i = c_{i+k} + (1 - a_i)$.

Тогда время, необходимое для добавления платформ при заданном значении q равно $c_q \cdot x$, а время, необходимое для увеличения p до значения q равно $(q - p) \cdot y$. Итоговое время равно $c_q \cdot x + (q - p) \cdot y$. Осталось лишь выбрать минимум по всем допустимым значениям q.

D. XOR-пушка

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Сначала вычислим массив $b_1, b_2, \ldots, b_n$, где b_i равно номеру самого старшего бита, равного 1 в двоичной записи числа a_i. По условию $b_i \le b_{i+1}$. Эти величины можно вычислить, проводя деление исходных чисел на 2, пока они не станут нулем.

Заметим, что если для некоторого i выполняется равенство $b_{i-1} = b_i = b_{i+1} = t$, то можно применить одну операцию к a_i и a_{i+1}, и результирующее число будет строго меньше, чем a_{i-1}. Действительно, ведь в a_{i-1} старший единичный бит имеет номер t, а в $a_i \oplus a_{i+1}$ бит номер t, как и все более старшие, равны нулю. Значит, если существует такое i, что легко проверить одним проходом по массиву b, то ответ равен 1.

Далее заметим, что если такого i не существует, то размер массива n не превосходит $2 \cdot \left(\lfloor \log_2 10^9 \rfloor + 1\right) = 60$! Действительно, существует не более двух чисел с одинаковым старшим единичным битом. В таких ограничениях решение задачи существенно упрощается.

Пусть решение существует. Тогда в итоговом массиве, обозначим его c, для некоторого i выполняется $c_i > c_{i+1}$. Заметим, что каждый элемент итогового массива равен побитовому исключающему ИЛИ некоего подотрезка исходного массива, причем каждый элемент исходного массива входит ровно в один такой подотрезок. Пусть для c_i это подотрезок $a_l, a_{l+1}, \ldots, a_m$, а для c_{i+1} это подотрезок $a_{m+1}, a_{m+2}, \ldots, a_r$. Тогда ясно, что для нахождения оптимального решения нам достаточно лишь перебрать все возможные значения l, m и r и проверить, что побитовое исключающее ИЛИ всех элементов на левом отрезке больше, чем на правом. Если неравенство выполняется, то обновим ответ значением $r - l - 1$. Асимптотика этой части решения $O(n^3)$ или $O(n^4)$ в зависимости от реализации.

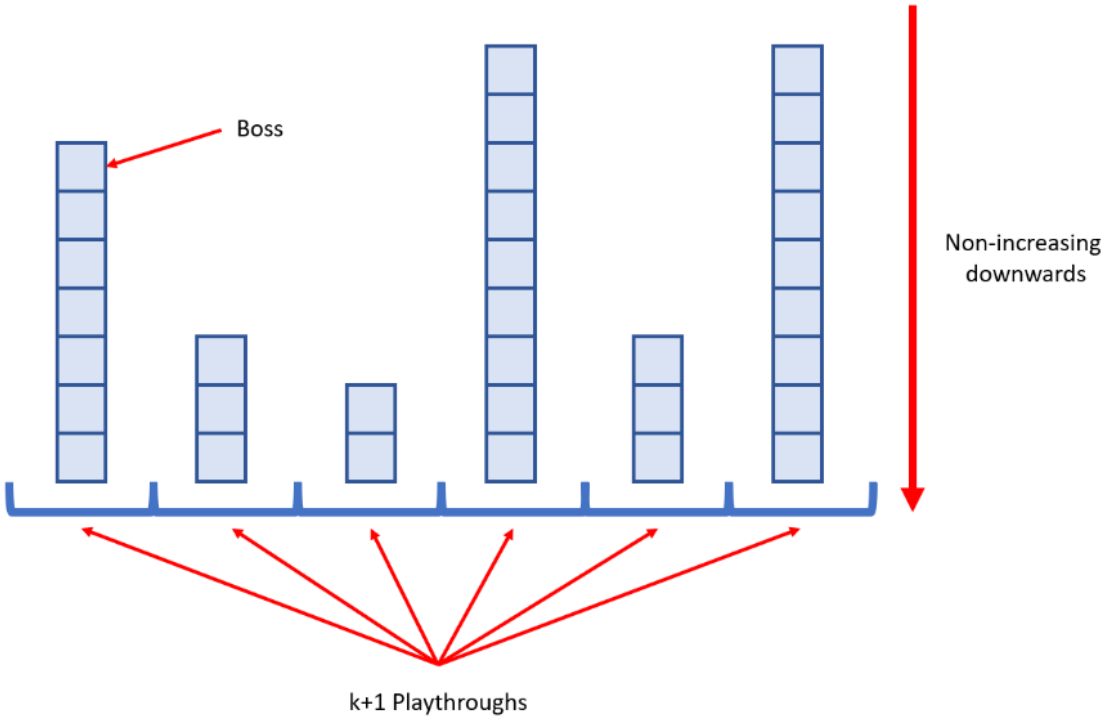
E. Новая игра плюс!

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Назовем победу над несколькими боссами между двух сбросов прохождением. Ясно, что прохождения не зависят друг от друга. Поэтому в задаче нужно разбить n боссов на $k+1$ прохождение.

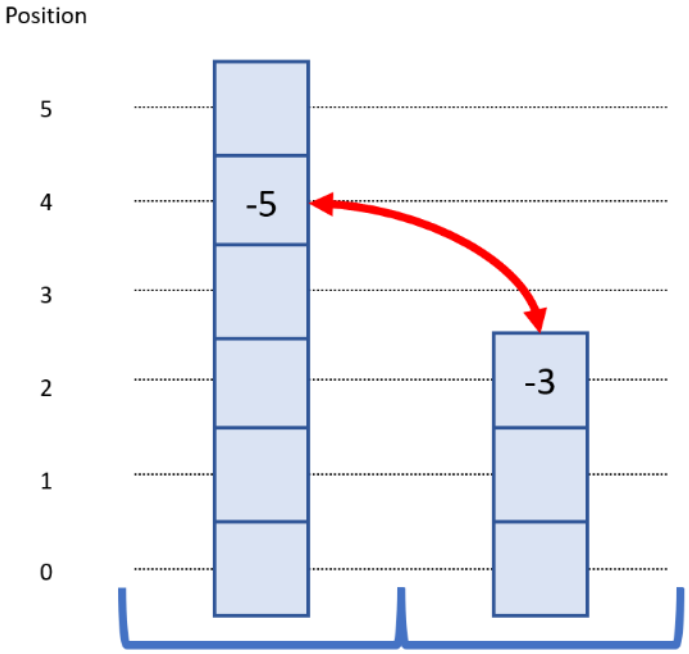
Рассмотрим прохождение с x боссами, влияние которых на комбо-бонус равно $a_1, a_2, \cdots, a_x$ в том порядке, в котором мы их проходим. Количество очков, которые мы получаем, равно $(x-1)a_1 + (x-2)a_2 + \ldots + (1)a_{x-1} + (0)a_x$. Легко показать, что внутри одного прохождения мы должны побеждать боссов в порядке невозрастания их влияния, то есть $a_1 \geq a_2 \geq \ldots \geq a_x$.

Мы можем представить решение как $k+1$ стек, каждый из которых представляет собой прохождение. Каждый стек содержит боссов, которые будут побеждены в этом прохождении. Стеки отсортированы по невозрастанию сверху вниз, что значит, что мы будем их побеждать в порядке сверху вниз.



Теперь легко понять, что случай $k = 0$ довольно прост: мы должны просто побеждать боссов в порядке невозрастания их влияния. Далее будем рассматривать только случай $k \geq 1$.

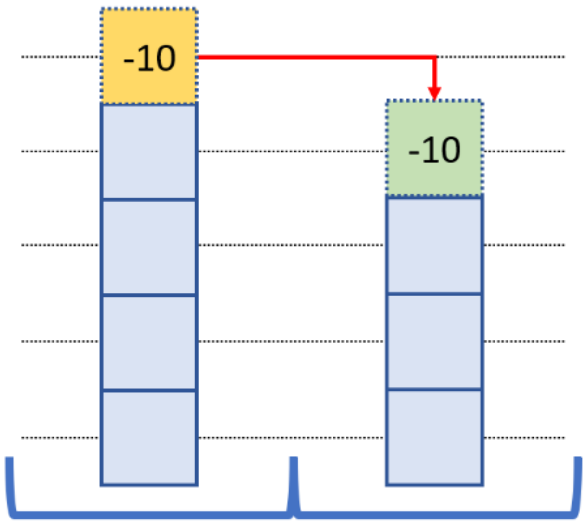
Для простоты скажем, что босс находится на месте p в некотором прохождении, если в стеке под ним находятся еще p боссов. Заметим, что если есть два босса в разных прохождениях с влияниями a и b на позициях i и j, то они дают нам $ia + jb$ очков. Поэтому ясно, что если $a \geq b$, то $i \geq j$, и наоборот. Например, данная ниже конфигурация не оптимальна, потому что если поменять местами -5 и -3, то получится ответ лучше.



Поэтому все боссы на позициях с меньшими номерами должны иметь меньшее влияние, чем все боссы на больших позициях. Это значит, что оптимальный ответ может быть получен добавлением боссов по одному в возрастающем порядке на верх стеков.

Назовем босса хорошим, если у него неотрицательное влияние, иначе назовем его плохим. Зафиксируем расположение плохих боссов и будем расставлять хороших боссов в порядке возрастания. Можно понять, что если мы поставим босса с влиянием a на стек высотой h, то к ответу добавится ah очков. Поэтому мы всегда выбираем стек с наибольшей высотой. Поэтому все хорошие боссы всегда попадут на один и тот же стек, и этот стек будет иметь наибольшую высоту. Назовем этот стек главным, а остальные k стеков побочными.

Если существуют два побочных стека, высота которых отличается не менее чем на 2, то мы можем переложить самого верхнего плохого босса с высокого стека на более короткий стек и уменьшить потерю очков. В примере ниже босса с влиянием -10 из левого стека (высоты 5) можно переложить на правый стек (высоты 3).



Поэтому высоты наименьшего и наибольшего из этих k побочных стеков отличаются не больше чем на 1.

Пусть наименьшая высота в побочных стеках равна h. Рассмотрим нижние h боссов во всех $k+1$ стеках, и вспомним, что все боссы с большим влиянием находятся выше чем боссы с меньшим влиянием. Поэтому рассмотренные $h(k+1)$ боссов должны иметь наименьшее влияние.

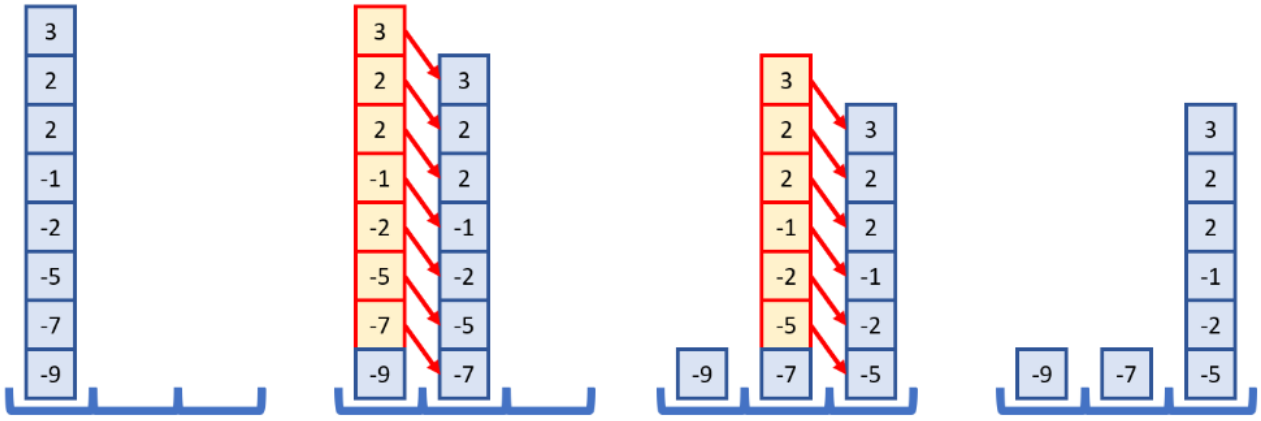
Простая жадность

Отсортируем боссов в порядке неубывания влияний. Для каждого возможного префикса P, который содержит только плохих боссов, рассмотрим их равномерное распределение в $k+1$ стек. Затем возьмем всех оставшихся боссов и положим их на самый высокий стек (если таких несколько, то на любой).

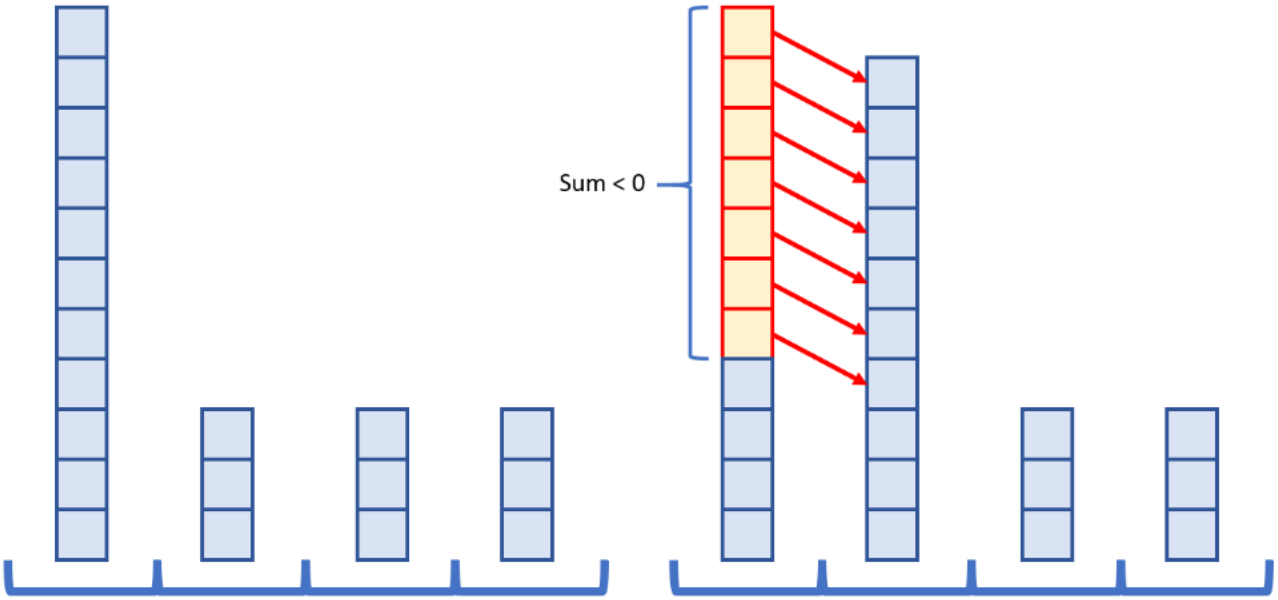
Наивное выполнение этих операций требует времени $O(n^2)$ но вычисления можно ускорить, предподсчитав взвешенные префиксные и суффиксные суммы, чтобы вычислить все ответы за $O(n)$. Итоговая асимптотика $O(n \log n)$ из-за сортировки.

Жадность посложнее

Возьмем все n влияний, сложим в один «скользящий» стек в неубывающем порядке снизу вверх. Далее будем перемещать этот стек по кругу по $k+1$ прохождениям и каждый раз будем оставлять самого нижнего босса на очередном стеке. Ниже показан пример с $n = 8$ и $k = 2$.



Каждый раз, когда мы перемещаем стек, мы уменьшаем ответ на сумму всех элементов, которые мы перемещаем вниз на одну позицию. Поэтому мы должны перемещать стек дальше, только если сумма этих величин меньше 0, и это даст нам оптимальный ответ.



Заметим, что влияния, которые мы перемещаем, всегда образуют суффикс влияний, отсортированных в порядке неубывания. Мы остановимся, когда сумма на очередном суффиксе станет неотрицательной. Поэтому мы можем найти правильную конфигурацию, сразу найдя правильный суффикс и расставив остальные элементы поровну.

F. Тортики для клонов

ограничение по времени на тест: 3 секунды
ограничение по памяти на тест: 256 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Пусть $mintime_i$ — это минимальное время, в которое мы можем прийти в координату x_i, и при этом все предыдущие тортики уже собраны, а последний созданный нами клон нам уже не нужен. Также пусть $dp_{i,j}$ — хранит в себе достижима ли ситуация, в которой мы только что собрали i-й тортик, а клон ждёт события j. Пусть мы сейчас находимся в i-м событии и последний созданный клон нам уже не нужен, тогда тортик появляющийся в этой точке всегда выгодно забрать клоном, а дальше есть два возможных варианта развития:

1. Мы идём к следующему событию, тогда надо просто попробовать обновить $mintime_{i+1}$, не забыв подождать пока клон заберет тортик в i-м событии.
2. Мы хотим оставить клона ждать какого-то j-го события, и сами забрать тортик в $i+1$-м событии, тогда если у нас хватает времени это сделать, то надо $dp_{i+1,j}$ сделать достижимым.

Если же мы находимся в событии, в котором мы только что собрали i-й тортик, а клон ждёт j-го, тогда если $i+1 \neq j$, то мы просто должны пойти и сами забрать $i+1$-й тортик, а иначе есть два возможных варианта развития:

1. $j+1$-е событие заберет клон, тогда надо просто попробовать обновить $mintime_{j+1}$
2. Мы хотим оставить клона ждать какого-то более позднего события k, и самостоятельно забрать $j+1$-й тортик, тогда если у нас хватает времени это сделать, то надо $dp_{j+1,k}$ сделать достижимым.

Собрать все тортики возможно, если $mintime_n \leq t_n$, либо $dp_{n-1,n}$ — достижимо.
Итоговая асимптотика $O(n^2)$, так как мы перебираем куда поставить следующего клона только если предыдущий клон уже не нужен, либо предыдущий клон ждёт следующего события.