Задача A. Работать или отдыхать?

Будем решать задачу с помощью жадного алгоритма.

Давайте перебирать дни последовательно от 1 до n-о, обозначим текущее количество денег как cur. Есть три варианта того что будет происходить в i-й день:

1. $cur \geqslant b_i$, в этом случае просто отдыхаем в этот день, и сохраняем информацию о том что мы отдыхали в этот день.

2. $cur < b_i$, но существует такой день отдыха $j(j < i)$ что выполняется $cur + a_i < cur + b_j + a_j - b_i$. То есть мы можем поставить работу в j-й день, а отдыхать в день i и в итоге получить большее количество денег. В этом случае выберем ту работу на которую выгоднее всего поменять.

3. $cur < b_i$ но не существует выгодной замены. В данном случае мы просто работаем в день i.

Рассмотрим второй вариант подробнее, на что выгодно заменять? Переместим a_i на другую сторону и у нас получится $cur < cur + a_j + b_j - (a_i + b_i)$. Давайте введём новое значение : $c_x = a_x + b_x$, теперь наше условие преобразовалось в $cur < cur + c_j - c_i$. Избавимся от cur, $c_i < c_j$. Теперь можно заметить что самая выгодная замена это замена на день j с максимальным c_j. Для того чтобы быстро находить такое j, добавлять элементы и удалять элементы достаточно поддерживать структуру данных set.

Время работы $O(NlogN)$.

Задача B. Друзья

Данная задача решается жадно.

Давайте найдём вершину с самой маленькой убедительностью которая является соседней для одной из активированных вершин. Если сумма убедительностей меньше чем эта убедительность, то мы больше не можем активировать вершины, иначе мы просто активируем вершину и повторяем действия из этого абзаца.

Вершину с самой маленькой убедительностью можно искать с помощью структуры данных set, каждая вершина добавляется ровно один раз. Тогда время работы будет $O(NlogN)$.

Задача C. Тройка

Нужно посчитать $S = \sum_{i=l}^{i\leqslant r}\sum_{j=l}^{j\leqslant r}\sum_{k=l}^{k\leqslant r}(max(a_i,a_j,a_k) - min(a_i,a_j,a_k))$.

$S = 6*\sum_{i=l}^{i\leqslant r}\sum_{j=i+1}^{j\leqslant r}\sum_{k=j+1}^{k\leqslant r}(max(a_i,a_j,a_k) - min(a_i,a_j,a_k)) + 6*\sum_{i=l}^{i\leqslant r}\sum_{j=i+1}^{j\leqslant r}(max(a_i,a_j) - min(a_i,a_j))$.

Закинем все элементы с отрезка в массив b и отсортируем его:

$sz = r - l$

$S = 6*\sum_{i=0}^{i\leqslant sz}\sum_{j=i+1}^{j\leqslant sz}\sum_{k=j+1}^{k\leqslant sz}(b_k - b_i) + 6*\sum_{i=0}^{i\leqslant sz}\sum_{j=i+1}^{j\leqslant sz}(b_j - b_i)$.

$S = 6*\sum_{i=0}^{i\leqslant sz}\sum_{k=i+1}^{k\leqslant sz}((b_k - b_i)*(k-i))$.

$S = 6*\sum_{i=0}^{i\leqslant sz}\sum_{k=i+1}^{k\leqslant sz}(b_k*k + b_i*i - b_k*i - b_i*k)$.

$S = 6*(\ \sum_{i=0}^{i\leqslant sz}\sum_{k=i+1}^{k\leqslant sz}(b_k*k + b_i*i)\ + \sum_{i=0}^{i\leqslant sz}\sum_{k=i+1}^{k\leqslant sz}(-b_k*i - b_i*k))$.

$S = 6*(\ \sum_{i=0}^{i\leqslant sz}(b_i*i*i + b_i*i*(sz-i))\ - \sum_{i=0}^{i\leqslant sz}\sum_{k=i+1}^{k\leqslant sz}(b_k*i + b_i*k))$.

$S = 6*(\ \sum_{i=0}^{i\leqslant sz}(b_i*i*sz)\ - \sum_{i=0}^{i\leqslant sz}(b_i*\sum_{j=0}^{j<i} j + b_i*\sum_{j=i+1}^{j\leqslant sz} j))$.

$S = 6*(\ \sum_{i=0}^{i\leqslant sz}(b_i*i*sz)\ - \sum_{i=0}^{i\leqslant sz}(b_i*(\sum_{j=0}^{j\leqslant sz} j - i)))$.

$S = 6*(\sum_{i=0}^{i \leqslant sz}(b_i * i * sz) - \sum_{i=0}^{i \leqslant sz}(b_i * (sz * (sz+1)/2 - i)))$.
$S = 6*(\sum_{i=0}^{i \leqslant sz}(b_i * i * sz - (b_i * sz * (sz+1)/2 - b_i * i)))$.
$S = 6*(\sum_{i=0}^{i \leqslant sz}(b_i * i * sz - b_i * sz * (sz+1)/2 + b_i * i))$.
$S = 6*(\sum_{i=0}^{i \leqslant sz}(b_i * i * (sz+1) - b_i * sz * (sz+1)/2))$.
$S = 6*(sz+1) * (\sum_{i=0}^{i \leqslant sz}(b_i * i - b_i * sz/2))$.
$S = 3*(sz+1) * (\sum_{i=0}^{i \leqslant sz}(2 * b_i * i - b_i * sz))$.
$cnst = 3 * (sz+1)$
$sum_2 = \sum_{i=0}^{i \leqslant sz} b_i * i$
$sum = \sum_{i=l}^{i \leqslant r} a_i$
$S = cnst * (2 * sum_2 - sum * sz)$.

Заметим что единственная проблемная вещь для подсчёт это sum_2, так как sum можно легко посчитать префикс суммами, а $sz = r - l$.

Давайте считать sum_2 с помощью алгоритма MO. Во-первых нужно понять как добавлять элементы и удалять их.Представим что у нас построен массив b и нам нужно вставить число z так чтобы массив b остался неубывающим. Найдём максимальную позицию j такую что $b_j < z$, и вставим элемент z после этой позиции j. После данного действия с sum_2 произойдёт два изменения. Первое изменение это $z * j$,это изменение связано с добавлением нового элемента. Второе изменение это $\sum_{i=j+1}^{i \leqslant sz} b_i$, это изменение связано с тем что все элементы после j сдвинутся на один вправо после добавление элемента z. Такие вычисления можно делать с помощью дерева Фенвика. Удаление происходит аналогично, найдём первое существующее число z и удалим его, попутно изменяя значение sum_2. Однако время работы такого решения $O((N+q) * \sqrt(N) * log(N))$

Улучшим наше решение применив технику sweepline mo . Если вкратце то пусть $f(l, r, z)$ это изменение sum_2 при добавлении элемента z к отрезку (l, r). Можно заметить что $f(l, r, z) = f(1, r, z) - f(1, l - 1, z)$, пусть $f_2(l, r, z)$ это изменение sum_2 при удалении элемента z из отрезка (l, r) где $a_r = z$, тогда $f_2(l, r, z) = f(l, r-1, z) = f(1, r-1, z) - f(1, l, z)$, пусть $f_3(l, r, z)$ это изменение sum_2 при удалении элемента z из отрезка (l, r) где $a_l = z$, тогда $f_3(l, r, z) = f(l+1, r, z) = f(1, r, z) - f(1, l, z)$. Если запомнить к какому запросу какие дельты относятся то можно заметить что их можно посчитать в офлайне, а в конце пройтись в нужном порядке добавляя нужные дельты. Теперь если мы за $O(1)$умеем подсчитывать количество элементов меньших чем z, и сумму элементов которые больше или равны z, и добавлять в эту структуру элемент за $O(\sqrt(N))$, то задача будет решена. К счастью существует корневая декомпозиция которая полностью подходит под это описание. Нам нужно сделать N добавлений, так как мы смогли изменить $f(l, r, z)$ так чтобы l всегда было равно 1, и $(N + Q)\sqrt(N)$ запросов вида посчитать кол-во чисел меньше z и посчитать сумму всех чисел $\geqslant z$. Следовательно наше решение будет работать $O((N + Q) * \sqrt(N))$, данное решение работает 950 мс на максимальном тесте даже при неаккуратном написании. Если сортировать запросы в порядке hilbert curve mo, то время работы на максимальном тесте будет 800 мс. Я уверен то что данное решение может работать и быстрее.

Для большего понимания решения рекомендую ознакомиться с статьёй **sweepline mo на codeforces.**

Автор : Сахмолдин Мухаммадариф