Problema 25

John David Ibañez Rodriguez.
Algorithm Design.
JON KLEINGBERG * ÉVA TARDOS

May 11, 2018

Enunciado.

Considere el problema al que se enfrenta un corredor de bolsa que trata de vender una gran cantidad de acciones en una compañía cuyo precio de acciones ha ido disminuyendo constantemente en valor. Siempre es difícil predecir el momento adecuado para vender acciones, pero poseer muchas acciones en una sola compañía agrega una complicación adicional: el solo hecho de vender muchas acciones en un solo día tendrá un efecto adverso en el precio.

Dado que los precios del mercado futuro, y el efecto de las grandes ventas en estos los precios son muy difíciles de predecir, las casas de bolsa usan modelos del mercado para ayudarlos a tomar tales decisiones. En este problema, consideraremos el siguiente modelo simple. Supongamos que tenemos que vender x acciones en una empresa, y supongamos que tenemos un modelo preciso del mercado:

Predice que el precio de las acciones tomará los valores $p_1, p_2, ..., p_n$ sobre el los próximos n días. Además, hay una función $f(\cdot)$ que predice el efecto de grandes ventas: si vendemos acciones y en un solo día, será permanente disminuir el precio en $f(y)$ a partir de ese día en adelante. Entonces, si vendemos acciones y_1 el día 1, obtenemos un precio por acción de p_1 - $f(y_1)$, para un ingreso total de $y_1 \cdot (p_1$ - $f(y_1))$. Habiendo vendido acciones y_1 en el día 1, podemos vender acciones y_2 el día 2 por un precio por acción de p_2-$f(y_1)$-$f(y_2)$; esto produce un ingreso adicional de $y_2 \cdot (p_2$-$f(y_1)$-$f(y_2))$. Este proceso continúa durante todos los n días. (Tenga en cuenta, como en nuestro cálculo para el día 2, que las disminuciones de días anteriores se absorben en los precios para todos los días posteriores).

Diseñe un algoritmo eficiente que tome los precios $p_1, ..., p_n$ y función $f(\cdot)$ (escrita como una lista de valores $f(1), f(2), ..., f(x))$ y determina la mejor manera de vender x acciones por día n. En otras palabras, encuentre los números naturales $y_1, y_2, ..., y_n$ de modo que $x = y_1 + ... + y_n$ y vendiendo acciones y_i el día i para $i = 1, 2, ..., n$ maximiza el ingreso total alcanzable. Debe suponer que el valor de acción p_i es monótono y $f(\cdot)$ es monótona en aumento; es decir, vender una mayor cantidad de acciones causa una caída más grande en el precio. El tiempo de ejecución de su algoritmo puede tener una dependencia polinómica en n (el número de días), x (el número de recursos compartidos) y p_1 (el precio máximo de las existencias).

Input

La entrada cosiste en tre lineas, la primera linea esta determinada por dos enteros x y n, tal que el entero x denota la cantidad de acciones y n que denota la cantidad de dias para vender x acciones en n dias, en la segunda linea contine $p_1, ..., p_n$ precios de las acciones. la tercera la linea contiene la funcion $f(1), ..., f(x)$

Output

Una sola linea con un entero n denotando del ingreso total maximo

Ejemplo Input

3 100.000
90 80 40

$$f(y) = \begin{cases} 1 & y \leq 40000 \\ 20 & y > 40000 \end{cases}$$

Ejemplo Output

7,100,000

Solution

Idea: En el i - ésimo día, las ventas no depende de cómo se realizó la venta en los días anteriores porque las restantes j acciones restantes incurrirá en la misma constante de penalizacion en ganancias debido a la valor acumulado de la fuction f de ventas previas. Entonces, la ganancia total simplemente disminuirá en j veces esa constante, Si se decide hacer la venta de y acciones en el día i-th, entonces las acciones j-th afectaran en la pérdida de ganancias de $f(y)$, independientemente de cuánto se vendan.

Diseño recurrente

Se define sub-problemas que involucran algunos de los últimos i días. por supuesto, tener $maxG(i)$ que denota la venta óptima de las acciones para los i a n dias es difícil, ya que no sabemos cuántas acciones están disponibles en el día i-ésimo.Implicando agregar un segundo parametro (i, j) que indique el beneficio de la venta óptima de j acciones a partir del día i.

El beneficio directo de vender y acciones con el precio p_i esta definido como $p_i * y$, adicionando la ganancia óptima para vender las acciones $j - y$ restantes a partir del día $i + 1$, es decir $maxG(i + 1, j - y)$,ahora para saber el ingreso total de de la venda $i - th$ esta definida por $y_i \cdot (p_i - f(y_i))$

Ahora toca tener en cuenta que el ingreso total de un dia $i-th$ se ve afectada por la venta y_{i-1} del $i-1$ dia anterior, entonces eso implica que tenemos que tenerlo encuenta en el calculo del ingreso máximo.

Análisis de contrucción del tercero factor para nuestro ingreso máximo.

En el primer día el ingreso total esta dado por

$$ingresoDia_1 = y_1 \cdot (p_1 - f(y_1))$$

En un segundo estaria dado por

$$ingresoDia_2 = ingresoDia_1 + y_2 \cdot (p_2 - f(y_1) - f(y_2))$$

Realizando el factor $y_2 \cdot (p_2 - f(y_1) - f(y_2))$

$$ingresoDia_2 = ingresoDia_1 + y_2 \cdot p_2 - y_2 \cdot f(y_1) - y_2 \cdot f(y_2))$$

Definicion del ingreso del dia 1

$$ingresoDia_2 = y_1 \cdot (p_1 - f(y_1)) + y_2 \cdot p_2 - y_2 \cdot f(y_1) - y_2 \cdot f(y_2))$$

Factorización - factor común

$$ingresoDia_2 = y_1 \cdot p_1 - y_1 \cdot f(y_1)) + y_2 \cdot p_2 - y_2 \cdot f(y_1) - y_2 \cdot f(y_2))$$
$$ingresoDia_2 = y_1 \cdot p_1 - f(y_1) \cdot (y_1 + y_2) + y_2 \cdot p_2 - y_2 \cdot f(y_2)$$

Dia 3 de ventas de las acciones, (se repite el proceso)

$$ingresoDia_3 = y_1 \cdot p_1 - f(y_1) \cdot (y_1 + y_2 + y_3) + y_2 \cdot p_2 - f(y_2) \cdot (y_2 + y_3) + y_3 \cdot p_3 - y_3 \cdot f(y_3)$$

Se puede decir que como premisa la $\sum_i y_i = x$, entonces nuestro tercer factor, seria

$$maxG(i,j) = p_i * y + maxG(i+1, j-y) - j * f(y))$$

Debido que nos piden el maxima Ganancia de vender y acciones, la $maxG$ recursiva estaria dada en la siguiente expresion: $maxG\,(i,j) = (\uparrow |y \leq j : p_i * y + maxG(i+1, j-y) - j * f(y))))$

Los casos bases detectados:

- Comprar más acciones de las existentes
- La cantidad de días disponibles para la ventas ya es la total
- Comprar la cantidad bastantes

Diseño iteractivo

Estructura de datos para el llenado de los parámetros de acumulación

Una tabla bidimensional M, cuyas filas representarían los días y las columnas de las acciones para vender, la matriz se llenaría utilizando la recurrencia anterior de la fila inferior a la superior, el resultado es un algoritmo es de complejidad $\Theta\left(n^3\right)$, el objetivo deseado es $maxG(1, x)$

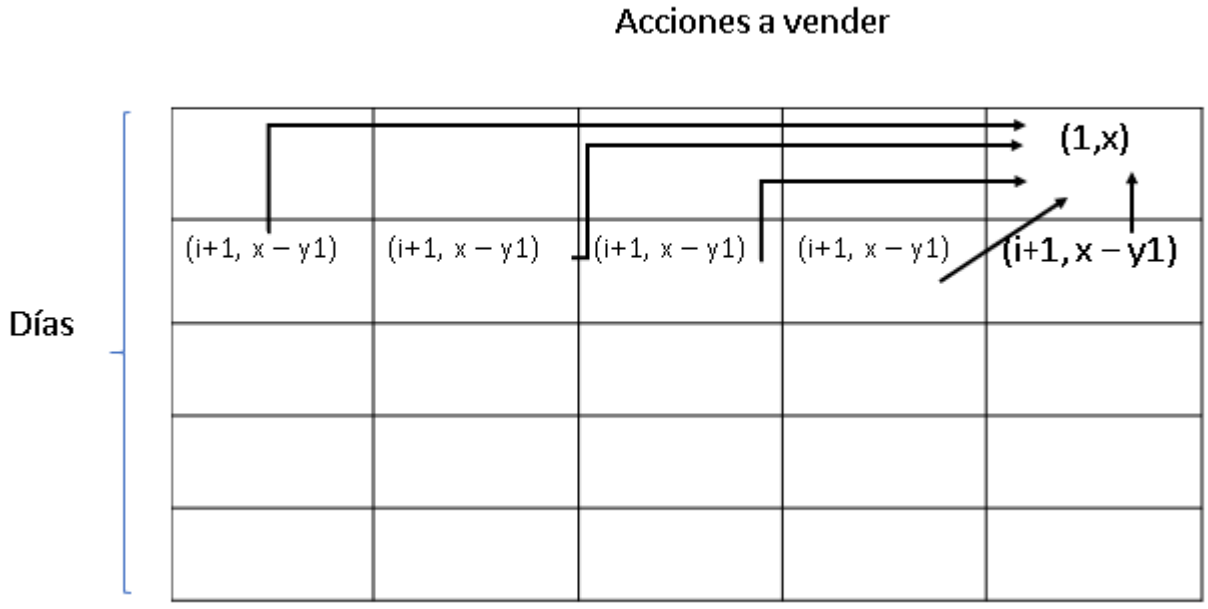

Figure 1: Diagrama de necesidades

```cpp
#include<bits/stdc++.h>
#define loop(i,a,b)  for(int i= a; i<b; i++)
#define FOR(i,b)  for(int i= 1; i<=b; i++)
#define LIM 80000

using namespace std;
typedef long long i64;
i64 F[LIM],P[LIM],x,n;

i64 maxG(i64 i,i64 j){
  /*
    Si la cantidad es mayor a n, implica que la cantidad de dia disponibles ya los tome todos.
    Si vendi mas acciones de las necesarias

    */
    if(i>n || j<0 )return INT_MIN;

    //acabe todas las acciones o todos los dias

    if(j==0 || i==n)return 0;
  i64 best = INT_MIN;
    // Tengo que encontrar una cantidad de acciones y tal que me de una ganancia maxima:
  for(int y = 1 ,y<=j;y++){
    if(j*f(y)>P[i]*y)continue;
    i64 Op =  P[i]*y - j*F[y] + maxG(i+1,j-y); // maxG(i+1/* Dia siguiente*/,j-y /*Vender j -y
    acciones compradas*/ )
    best = max(best,Op);
  }
    return best;
}

int main(){
  ios::sync_with_stdio(0);
  cout.precision(0);
    cin.tie(0);
  cout << fixed;
  cout << "Digite la cantidad de dias \n";
  cin >>n;
  cout << "Cuantas acciones desea \n";
  cin >>x;
  FOR(i,n)cin >> P[i];
```

```
41    FOR(i,x)cin >> F[i];
42      cout << maxG(0,x) << '\n';
43    return 0;
44  }
```