PRÁCTICA 4
ALGORÍTMICA

Integrantes:

Raúl Granados López
Hossam El Amraoui Leghzali
Javier Montaño Rubio

Algoritmo de programación dinámica

Diseño de componentes

Cantidad de dinero disponible **X**

Número total de empresas **N**

Acciones disponibles de cada empresa $\mathbf{a_i}$

Precio de cada acción $\mathbf{p_i}$

Estimación del posible beneficio por acción comprada de una empresa $\mathbf{b_i}$

Comisión por pago de una comisión $\mathbf{c_i}$

Función objetivo: Maximizar $\mathbf{\Sigma(b_i * p_i) * a_i}$

Llamaremos PD(i,j) al beneficio de haber considerado comprar cada una de las acciones de las empresas 1 hasta i, sabiendo que nos queda una cantidad de j dinero.

El problema se puede resolver por etapas: En cada etapa **i** seleccionaremos comprar (o no) acciones de la empresa **i** y cuántas queremos comprar en caso afirmativo, considerando el dinero que nos queda. Si nos llevamos acciones, restamos su coste a **j**.

Objetivo: Conocer **PD(N,X)** = Beneficio máximo que se puede obtener invirtiendo en empresas **desde 1 hasta N**, considerando las restricciones de dinero (**X**), la comisión aplicada por cada acción comprada y el número de acciones de cada empresa.

Casos base:

- PD(i, 0) = 0 No nos queda dinero para invertir.
- PD(1, 1...p_1+c_1)= 0 Si solo hay una empresa y no tenemos dinero para ninguna acción (considerando la comisión).
- PD(1, p_1+c_1...X)= $b_1 * p_1 * (j / (p_1+c_1))$ Si solo hay una empresa y tenemos dinero para una acción o más (considerando la comisión), llevaremos el máximo número de acciones posible sin pasarnos de presupuesto.

Caso general

Para k en $\mathbf{a_i}$ siendo k el número de acciones escogido de la empresa i**, hacer:**

Coste = k * (p_i + c_i) siendo Coste el dinero necesario para comprar k acciones de la empresa i

Beneficio = 0

Si j es mayor que Coste, **entonces:**

Beneficio = k * p_i * b_i + PD[i-1][j – coste]

Fin- Si

Si k=0, **entonces:**

PD[i][j] = max { PD[i-1][j], Beneficio}

Si no, entonces:

PD[i][j] = max { PD[i][j], Beneficio}

Fin - Si

Fin – Para

El caso general supone que para una cantidad de dinero **j**, considerando comprar (o no) un determinado número de acciones **k** de las empresas **1 hasta i**, el máximo beneficio se conseguirá de las dos formas siguientes:

- En caso de considerar la compra de sólo la primera acción de la empresa i (**k=1**), no comprar la acción y considerar la compra de acciones desde las empresas **1 hasta i-1**: Primera parte **PD[i-1][j]**
- En caso de considerar la compra de más de una acción de la empresa i (**k>1**), no comprar esa cantidad de acciones y considerar la compra de una acción menos de esa misma empresa: Primera parte **PD[i][j]** con **k-1**
- Comprar las **k** acciones de la empresa **i** con la cantidad de dinero que tenemos (siempre y cuando tengamos dinero suficiente), restando el dinero más la comisión al dinero que tenemos **j-Coste** y aumentando el beneficio según el valor de k: k* p_i * b_i. Seguidamente, vemos si podemos comprar acciones de las empresas **1 hasta i-1** y cuántas podemos comprar con el dinero restante: Segunda parte **k*p_i*b_i + PD[i-1][j-Coste]**

Verificación del P.O.B.

PD[i][j] es óptimo si las decisiones tomadas anteriormente para **PD[i-1][j]** (en caso de tener en cuenta solo una acción) **o PD[i][j]**(en caso de tener en cuenta más de una acción) y **PD[i-1][j-Coste]** lo son.

- Para una cantidad de dinero **j** fija, PD[0][j] es óptimo ya que, da igual el valor de j, se selecciona comprar o no el máximo número de acciones posible con el máximo beneficio teniendo en cuenta el precio de cada acción y la respectiva comisión.
- Para PD[1][j] también es óptimo, ya que solo decidirá comprar k acciones de dicha empresa si la suma del precio de todas las acciones compradas(incluyendo las de la empresa anterior) es menor que el dinero disponible, o decidirá comprar menos acciones, sea de una o de otra empresa si el precio resulta ser superior.
- Se sigue por inducción hasta PD[i-1][j] y suponemos este óptimo. Por reducción al absurdo, si no lo fuera existiría otra decisión óptima distinta de PD[i-1][j] no considerada óptima, lo cual es imposible según lo especificado anteriormente.
- Demostrado óptimo PD[i-1][j], queda demostrado también para cualquier caso PD[i-1][j-Coste], como caso particular de PD[i-1][j].

Representación

Vamos a representar la solución al problema como una tabla PD(i,j):

- **Filas(i)**: Asociadas a las empresas, ordenadas ascendentemente por su valor $\mathbf{b_1/p_1 > b_2/p_2 > \dots > b_N/p_N}$.
- **Columnas(j)**: Cantidad de dinero disponible, desde 0 hasta **X**.
- **Celdas**: Cada celda PD(i,j) contendrá el máximo beneficio que se puede obtener para una cantidad de dinero **j** asumiendo la compra (o no) de acciones de las empresas **1 hasta i**.

Para saber qué acciones de qué empresas se han comprado, basta con comparar **PD[N][X]** con **PD[N-1][X]**. Si estos valores son iguales, entonces no se ha comprado ninguna acción de la empresa N; en caso contrario, sí se compró. Para saber cuántas acciones se compraron, basta con ir probando con cada valor entre 1 y el número de acciones de la empresa N (valor k) hasta que **PD[N][X]** coincida con **PD[N-1][X–Coste]**, siendo Coste el dinero necesario para comprar k acciones de dicha empresa, siendo k el número de acciones compradas en la empresa N.

Seguidamente, realizar el proceso recursivamente hasta llegar a la empresa 1 o cantidad de dinero 0 para conocer el resto de las acciones.

Algoritmo Básico (fuerza bruta)

Definición de variables

Cantidad de dinero disponible **X**

Vector con la información de cada empresa **empresas**

Cantidad de empresas **N**

Entero que indica la posición actual en el vector empresas **index**

Vector de enteros que almacena la solución óptima encontrada hasta el momento **sol**

Vector de enteros que almacena la combinación actual de acciones siendo probada **combination**

Máximo beneficio encontrado hasta el momento **maxBeneficio**

Diseño del algoritmo de fuerza bruta

Objetivo: Recorrer todas las posibilidades de compra de acciones de forma recursiva comprobando si para cada combinación de acciones compradas se pueden comprar y su beneficio es mayor que la mejor combinación encontrada.

Caso base:

- Index = N Ya hay una nueva combinación a comprobar

El caso base es el encargado de comprobar y actualizar la mejor solución al problema. Cuando **index** es igual a **N** significa que se han insertado tantos elementos en combination como empresas hay. Para actualizar la mejor solución, primero se comprueba si el beneficio total de dicha combinación es mayor que el mejor beneficio que se tenía hasta el momento. Si es verdadero, después se comprueba si se dispone del dinero suficiente para poder comprar las acciones, teniendo en cuenta su respectiva comisión. En el caso de que se cumplan ambas condiciones, se actualiza **sol** y **maxBeneficio** con los datos correspondientes a la combinación actual.

Caso general:

Para cada i entre 0 y número de acciones de la empresa index, **hacer**:

```
Insertar i en combination
resolverFuerzaBruta(X, empresas, index+1, sol, combination, maxBeneficio)
Eliminar última posición de combination
```

El caso general es el encargado de recorrer todas las posibles combinaciones que comprobará el caso base. Para ello, se inserta en **combination** la cantidad de acciones a comprar de la empresa index. Para seguir probando combinaciones, hay que eliminar de **combination** dicho número para insertar el siguiente y realizar su correspondiente comprobación.

Implementación del algoritmo básico

```
double calculaCoste(const vector<Empresa>& empresas, const vector<int>& combination) { O(e)
    double sum = 0.0;
    for (int i=0; i<empresas.size(); i++) { nº veces = empresas.size() → O(e); e = empresas.size()
        sum += (empresas[i].precio_accion + empresas[i].comision) * combination[i];
    }

    return sum;
}
```

```
double calculaBeneficio(const vector<Empresa>& empresas, const vector<int>& combination) { O(e)
    double sum = 0.0;
    for (int i=0; i<empresas.size(); i++) { nº veces = empresas.size() → O(e)
        sum += empresas[i].precio_accion * empresas[i].beneficio * combination[i];
    }

    return sum;
}
```

```
// Función parar resolver el problema utilizando fuerza bruta
void resolverFuerzaBruta(int X, const vector<Empresa>& empresas, int index, vector<int>& sol, vector<int>& combination,
                         double& maxBeneficio) {  O(a^e)  a = nº de acciones

    if (index == empresas.size()) {
        double sum = calculaBeneficio(empresas, combination); O(e)
        if (sum > maxBeneficio && calculaCoste(empresas, combination) <= X) { O(e)   ] O(e)   ] O(e)
            maxBeneficio = sum;
            sol = combination;
        }
    }
    else {
        // Recorrer todas las combinaciones posibles
        for (int i=0; i<=empresas[index].acciones_disponibles; i++) {
            // Insertar posible solución
            combination.push_back(i); O(1)

            resolverFuerzaBruta(X, empresas, index+1, sol, combination, maxBeneficio); T(e-1)   ] T(e) = a·(T(e-1)+1) → O(a^e)

            // Eliminar el último elemento del vector para seguir probando combinaciones
            combination.pop_back(); O(1)
        }
    }
}
```

```
// Función encargada de pasar los argumentos necesarios para la primera ejecución
// del algoritmo de fuerza bruta
vector<int> resolverFB(int& X, vector<Empresa>& empresas) { O(a^e)
    vector<int> sol, combination; O(1)
    double maxBeneficio = INT_MIN; O(1)

    resolverFuerzaBruta(X, empresas, 0, sol, combination, maxBeneficio); O(a^e)

    return sol;
}
```

Eficiencia del algoritmo básico

$$T(e) = a \cdot T(e-1) + a$$

$$T(e) - a \cdot T(e-1) = a$$

PH:

$$\begin{cases} x^e - a \cdot x^{e-1} = 0 \\ x \cdot x^{e-1} - a \cdot x^{e-1} = 0 \end{cases}$$

$$x^{e-1} \cdot (x-a) = 1$$

$$P_H(x) = (x-a)$$

$$P(x) = P_H(x) \cdot (x-b_1)^{d_1+1}$$

$$P(x) = (x-a) \cdot (x-1)^1$$

$r = 2$

$R_1 = a \quad m = 1$

$R_2 = 1 \quad m = 1$

$$a = b_1^m \cdot g_1(m) \begin{cases} b = 1 \\ g_1(m) = a \end{cases}$$

$$T(n) = c_{10} \cdot R_1^e \cdot e^m + c_{20} \cdot R_2^e \cdot e^m =$$

$$= a^e \cdot e^0 + 1^e \cdot e^0 = O(a^e)$$

Implementación y eficiencia del algoritmo de programación dinámica

```cpp
// Función auxiliar para resolver el problema utilizando programación dinámica
vector<int> resolverPD(int X, const vector<Empresa>& empresas) {
    int N = empresas.size();

    // Ordenar las empresas por la razón beneficio / precio en orden descendente
    vector<pair<double, int>> orden;
    for (int i = 0; i < N; i++) {

        double razon = (empresas[i].beneficio) / empresas[i].precio_accion;
        orden.push_back({razon, i});

    }

    sort(orden.begin(), orden.end());

    // Inicializar la matriz de programación dinámica
    vector<vector<double>> PD(N , vector<double>(X + 1, 0));

    // Rellenar casos base
    for (int i = 0; i < N; i++) {
        PD[i][0] = 0;
    }

    for (int j = 0; j <= X; j++) {
        int empresa = orden[0].second;
        PD[0][j] = 0;
        int num_acciones = j / (empresas[empresa].precio_accion + empresas[empresa].comision);
        if (num_acciones <= empresas[empresa].acciones_disponibles)
            PD[0][j] = num_acciones * empresas[empresa].precio_accion * empresas[empresa].beneficio;
        else
            PD[0][j] = empresas[empresa].acciones_disponibles * empresas[empresa].precio_accion * empresas[empresa].beneficio;
    }

    // Calcular el beneficio máximo
    for (int i=1 ; i<N; i++){
        int empresa = orden[i].second;
        for (int j=1; j<=X; j++){

            for (int k=1; k<=empresas[empresa].acciones_disponibles; k++){

                double coste = k*(empresas[empresa].precio_accion + empresas[empresa].comision);
                double beneficio = 0;

                if (j >= coste){
                    beneficio = k*empresas[empresa].precio_accion*empresas[empresa].beneficio;
                    beneficio += PD[i-1][j-coste];
                }

                if (k==1)
                    PD[i][j] = max(PD[i-1][j], beneficio);
                else
                    PD[i][j] = max(PD[i][j], beneficio);

            }
        }
    }

    // Obtener las acciones compradas
    vector<int> acciones_compradas(N, 0);
    int dinero = X;
    int i = N-1;

    while (i>=0){

        int empresa = orden[i].second;
        bool parar = false;

        if ( i>0 && PD[i][dinero] != PD[i-1][dinero] ){

            for (int k = 1; k <= empresas[empresa].acciones_disponibles && !parar; k++){
                if (PD[i][dinero] == k*empresas[empresa].precio_accion*empresas[empresa].beneficio +
                        PD[i-1][dinero-k*(empresas[empresa].precio_accion + empresas[empresa].comision)]){
                    acciones_compradas[empresa] = k;
                    dinero -= k*(empresas[empresa].precio_accion + empresas[empresa].comision);
                    i--;
                    parar = true;
                }
            }

        }
        else if (i==0 && PD[i][dinero] != 0){

            for (int k = 1; k <= empresas[empresa].acciones_disponibles && !parar; k++){
                if (PD[i][dinero] == k*empresas[empresa].precio_accion*empresas[empresa].beneficio){
                    acciones_compradas[empresa] = k;
                    dinero -= k*(empresas[empresa].precio_accion + empresas[empresa].comision);
                    parar = true;
                }
            }

        }
        else
            i--;
    }

    return acciones_compradas;
}
```

Ejecuciones de ejemplo

Para compilar el programa debemos introducir en la terminal, desde el directorio donde se encuentra el cpp y el txt:

g++ -o practica4 practica4.cpp

Para ejecutar el programa debemos introducir en la terminal:

./practica4 <fichero de texto con las empresas>

La estructura del fichero de empresas es la siguiente:

nº de empresas
nº acciones precio_acciones beneficio comision <---------- Empresa 1
nº acciones precio_acciones beneficio comision <---------- Empresa 2
nº acciones precio_acciones beneficio comision <---------- Empresa 3
. . . . .
. . . . .
. . . . .

Ejemplos proporcionados:

```
5
20 100 3 1
50 200 3.5 2
50 150 1 1
20 500 2 3
2 1000 8 4
```

empresas.txt

```
10
6 100 3 5
14 200 3.5 3
20 150 1 2
10 500 2 4
2 1000 12 6
23 150 3 8
7 333 3.5 4
22 666 1 7
15 445 2 2
1 2000 8 1
```

empresas2.txt

Ejecución con empresas.txt y 5000€

```
javier2003mr@javier2003mr-GF75-Thin-9SD:~/Descargas/Practicas-ALG-main/Practica4$ ./practica4 empresas.txt
Introduce la cantidad de Euros a gastar: 5000
Empresa 1: 20 acciones disponibles, 100 euros por acción, 3 % de beneficio, 1 euro(s) de comisión por cada acción.
Empresa 2: 50 acciones disponibles, 200 euros por acción, 3.5 % de beneficio, 2 euro(s) de comisión por cada acción.
Empresa 3: 50 acciones disponibles, 150 euros por acción, 1 % de beneficio, 1 euro(s) de comisión por cada acción.
Empresa 4: 20 acciones disponibles, 500 euros por acción, 2 % de beneficio, 3 euro(s) de comisión por cada acción.
Empresa 5: 2 acciones disponibles, 1000 euros por acción, 8 % de beneficio, 4 euro(s) de comisión por cada acción.

----------------------------------------
FUERZA BRUTA
Beneficio máximo: 261
Acciones a comprar:
    Empresa 1: 1
    Empresa 2: 14
    Empresa 3: 0
    Empresa 4: 0
    Empresa 5: 2

----------------------------------------
PROGRAMACIÓN DINÁMICA
Beneficio máximo: 261
Acciones a comprar:
    Empresa 1: 1
    Empresa 2: 14
    Empresa 3: 0
    Empresa 4: 0
    Empresa 5: 2
```

Ejecución con empresas2.txt y 10000€

```
javier2003mr@javier2003mr-GF75-Thin-9SD:~/Descargas/Practicas-ALG-main/Practica4$ ./practica4 empresas2.txt
Introduce la cantidad de Euros a gastar: 10000
Empresa 1: 6 acciones disponibles, 100 euros por acción, 3 % de beneficio, 5 euro(s) de comisión por cada acción.
Empresa 2: 14 acciones disponibles, 200 euros por acción, 3.5 % de beneficio, 3 euro(s) de comisión por cada acción.
Empresa 3: 20 acciones disponibles, 150 euros por acción, 1 % de beneficio, 2 euro(s) de comisión por cada acción.
Empresa 4: 10 acciones disponibles, 500 euros por acción, 2 % de beneficio, 4 euro(s) de comisión por cada acción.
Empresa 5: 2 acciones disponibles, 1000 euros por acción, 12 % de beneficio, 6 euro(s) de comisión por cada acción.
Empresa 6: 23 acciones disponibles, 150 euros por acción, 3 % de beneficio, 8 euro(s) de comisión por cada acción.
Empresa 7: 7 acciones disponibles, 333 euros por acción, 3.5 % de beneficio, 4 euro(s) de comisión por cada acción.
Empresa 8: 22 acciones disponibles, 666 euros por acción, 1 % de beneficio, 7 euro(s) de comisión por cada acción.
Empresa 9: 15 acciones disponibles, 445 euros por acción, 2 % de beneficio, 2 euro(s) de comisión por cada acción.
Empresa 10: 1 acciones disponibles, 2000 euros por acción, 8 % de beneficio, 1 euro(s) de comisión por cada acción.

----------------------------------------
FUERZA BRUTA
Beneficio máximo: 600.585
Acciones a comprar:
    Empresa 1: 1
    Empresa 2: 14
    Empresa 3: 0
    Empresa 4: 0
    Empresa 5: 2
    Empresa 6: 4
    Empresa 7: 7
    Empresa 8: 0
    Empresa 9: 0
    Empresa 10: 1

----------------------------------------
PROGRAMACIÓN DINÁMICA
Beneficio máximo: 600.585
Acciones a comprar:
    Empresa 1: 1
    Empresa 2: 14
    Empresa 3: 0
    Empresa 4: 0
    Empresa 5: 2
    Empresa 6: 4
    Empresa 7: 7
    Empresa 8: 0
    Empresa 9: 0
    Empresa 10: 1
```