Mario Casas Pérez, Ángela Maria Garrido Ruiz, Rocío Barragán Moreno, y
Marina Ruiz Palomino 2º D - GII

Práctica 4: Algoritmos Programación Dinámica

1. Diseño de componentes de Programación Dinámica.
El problema a resolver debe:

a) Ser un problema de optimización (maximización/minimización): Es un problema de maximización, porque busca obtener el máximo beneficio posible para una cantidad que va a gastarse.
b) Poder resolverse por etapas: Cada etapa representará la compra, o no, de las acciones de la empresa.
c) Poder modelarse mediante una ecuación recurrente:
   T(i,j) = Max{T(i-1,j), bi*pi+T(i-1,j-pi-ci)}
d) Existir uno o varios casos base al problema:
   1. No tenemos dinero, X=0
   2. La empresa no nos oferta acciones, A=0
   3. No tenemos suficiente dinero para comprar acciones de una empresa X < A[i].precio
e) Cumplir el principio de optimalidad de Bellman (P.O.B.): No existe una solución mejor ya que la ecuación en recurrencias siempre toma el máximo.
f) Poder ser diseñado mediante un algoritmo: Ya hemos obtenido la ecuación en recurrencias por lo que para diseñar el algoritmo vamos a ver cómo se diseña la memoria. En este caso para resolver el problema vamos a diseñar una matriz T en la que:
   i) Las filas representan las acciones que podemos comprar (0..n)
   ii) Las columnas representan el dinero que nos queda (0..X)

Por tanto la dimensión de la matriz es de n filas y X columnas y T(i,j) se completará con el resultado de la ecuación en recurrencias estudiada. Rellenaremos la matriz por filas.

2. Diseño e implementación del algoritmo básico (fuerza bruta) a partir de la ecuación recurrente de forma directa.
La plantilla (pseudocódigo del algoritmo es el siguiente:

```
invertirBolsa(A : conjunto de acciones, X : cantidad de dinero)
        //Casos base
        Si X == 0 //no tenemos dinero
                Devolver 0;
        Si A == 0 //no tenemos acciones que comprar
                Devolver 0;
        //En otro caso que no sean los casos base devolvemos el máximo de
        Para i = 1 hasta A.size()
                Para j=1 hasta X
```

valor1 = A[i].Precio*A[i].Beneficio;
valor2 = A[i].Precio*A[i].Beneficio+(j-A[i].Precio+A[i].Comision);
Si j < A[i].Precio
Devolver valor1
Si valor1 > valor2
Devolver valor1;
En otro caso
Devolver valor2
Fin-Para
Fin-Para
Fin

La implementación del algoritmo queda de la siguiente manera:

```
//Algoritmo de fuerza bruta
double invertirBolsa(vector<Accion> A, int X) {
	double valor=0.0;
	//Casos base
	if (X == 0) {
		cout << "Base 1" << endl;

		return 0;
	}
	else if (A.size() == 0) {
		cout << "Base 2" << endl;

		return 0;
	}

	else {
		for(int i = 1; i < A.size();i++){
			for(int j = 1; j < X; j++){

				double valor1=0, valor2 = 0;
				valor1 = A[i].Precio()*A[i].Beneficio();
				valor2 = A[i].Precio()*A[i].Beneficio() + (j-A[i].Precio()-A[i].Comision());

				if(j < A[i].Precio()) //En el caso de que no tenga dinero suficiente
					valor = valor1; //tomo el valor del beneficio de la accion anterior.

				if(valor1 > valor2)
					valor = valor1;
				else
					valor = valor2;
			}

		}

	}
	return valor;
}
```

3. Diseño de algoritmo de Programación Dinámica de acuerdo a las componentes diseñadas en el apartado 1.

```
invertirBolsa(A : conjunto de acciones, X : cantidad de dinero)
    T //matriz en la que haremos los cálculos
    //Casos base
    Para j= 0 hasta X //solo puedo comprar una acción
        Si (A[0].Precio+A[0].Comision > j)
                T[0][j] = 0;
        En otro caso
        T[i][j] = A[0].Precio*A[0].Beneficio-A[0].Comision;

    Para i = 0 hasta A.size() //no tengo dinero
        T[i][0] = 0

    //Caso general → relleno de la matriz mediante la recurrencia
    Para i = 1 hasta A.size()
        Para j = 0 hasta X Hacer
        double aux1, aux2;  //almacenan los valores de la recurrencia
        aux1 = T(i-1,j);
        aux 2 = T(i-1,j-A[i].precio-A[i].Comision) + A[i].Beneficio*A[i].Precio;

        //Si el valor de j sale negativo nos quedamos con la parte positiva
        if(j-A[i].Precio-A[i].Comision < 0)
            T[i][j] = aux1;
        //Si puedo pagar la acción
        if(A[i].Precio() + A[i].Comision() < j)
            T[i][j] = max(aux1,aux2);
        else
            T[i][j] = aux1;
    FinPara-j
    FinPara-i
    Devolver T[i][j] //mostramos la matriz de costes
```

4. Implementación del algoritmo de Programación Dinámica.

Este algoritmo lo hemos implementado como hemos expuesto anteriormente en el ejercicio 1, a través de una matriz como representación de la memoria.

Para gestionar las acciones hemos implementado una clase que contiene los valores necesarios para realizar los cálculos: precio de la acción, beneficio y la comisión que debemos pagar por la compra de la misma.

Al método le pasamos un vector de acciones posibles a compra y el dinero del que disponemos. A partir de esto vamos calculando la matriz de costes.

La implementación del código queda de la siguiente manera:

- Primero reservamos memoria y estudiamos los casos base

```cpp
void invertirBolsaPD(vector<Accion> A, int X) {
	double **T; //creacion de la matriz dinamica

	T = new double* [A.size()]; //reservo las filas
	for (int i = 0; i < A.size(); i++) {
		T[i] = new double[X + 1]; //reservo las columnas
	}

	//Relleno del caso base de la matriz -> solo puedo comprar una accion -> me quedo con el beneficio de esa accion
	for (int j = 0; j < X; j++) {
		if(A[0].Precio()+A[0].Comision() > j)
			T[0][j] = 0;
		else
			T[0][j] = A[0].Precio()*A[0].Beneficio()-A[0].Comision();
	}

	//Relleno del caso base de la matriz -> No tengo dinero
	for (int i = 0; i < A.size(); i++) {
		T[i][0] = 0;
	}
```

- Después estudiamos el caso general

```cpp
	//Caso general
	for (int i = 1; i < A.size(); i++) {
		for (int j = 1; j <= X; j++) {
			//Calculo las partes de la recurrencia
			double aux1=0, aux2=0;
			aux1 = T[i - 1][j];

			//calculo de la parte de la ecuacion recurrente en el caso de que se compre la accion i
			double a1 = (A[i].Beneficio() * A[i].Precio());
			int a2 = j - A[i].Comision() - A[i].Precio();
			aux2 = a1 + T[i - 1][a2];
			//Si la posicion de j es una a la que no puedo acceder la pongo la anterior
			if(a2 < 0){
				T[i][j] = aux1;
			}
			//Si puedo pagar la accion, me quedo con el maximo
			if(A[i].Precio() + A[i].Comision() <= j)
				T[i][j] = max(aux1,aux2);
			//Si no la puedo pagar
			else
				T[i][j] = aux1;
		}
	}
```

- Por último mostramos la matriz obtenida por pantalla y liberamos la memoria reservada

```cpp
	//Muestro por pantalla la matriz con los calculos realizados
	cout << "Matriz de beneficios:" << endl;
	for (int i = 0; i < A.size(); i++) {
		for (int j = 0; j < X; j++) {
			cout << T[i][j] << ' ';
		}
		cout << endl;
	}

	cout << "\nBeneficio Maximo con PD: " << T[A.size()-1][X-1] << endl;

	//Liberacion de la memoria
	for(int i = 0; i < A.size(); i++){
		delete [] T[i];
	}
	delete [] T;
}
```

5. Cálculo de eficiencia del algoritmo básico y de Programación Dinámica.

La eficiencia del algoritmo básico es:

EFICIENCIA ALGORITMO FUERZA BRUTA

- → Primer if : $\boxed{O(1)}$
- → Primer else if : $O(n)$ → ya que el "size()" tiene eficiencia $\boxed{O(n)}$
- → Primer else

SEGUNDO BUCLE FOR

$$\left.\begin{array}{ll} i(n) = O(1) & f(n) = O(1) \\ a(n) = O(1) & h(n) = O(n) \\ g(n) = O(1) & \end{array}\right\} \begin{array}{l} O(1+1+n(1+1+1)) = \\ = O(2+3n) = \boxed{O(n)} \end{array}$$

PRIMER BUCLE FOR

$$\left.\begin{array}{ll} i(n) = O(1) & f(n) = O(n) \\ a(n) = O(1) & h(n) = O(n) \\ g(n) = O(n) & \\ \downarrow & \\ \text{A.size()} & \end{array}\right\} \begin{array}{l} O(1+n+n(n+n+1)) = \\ = O(1+2n+n^2) = \boxed{O(n^2)} \end{array}$$

Eficiencia total del else → $\max(O(n), O(n^2)) = \boxed{O(n^2)}$

Por tanto, la eficiencia total del algoritmo de fuerza bruta es el $\max(O(1), O(n), O(n^2))$

$$\boxed{\text{EFICIENCIA TOTAL FUERZA BRUTA} \Rightarrow O(n^2)}$$

La eficiencia del algoritmo de Programación Dinámica es:

EFICIENCIA ALGORITMO PD : InvertirBolsaPD ( )

// Evaluamos el caso general

→ Segundo bucle

$i(n) = O(1)$

$a(n) = O(1)$

$g(n) = O(1)$

$f(n) = O(1)$

$h(n) = O(n)$

$O(1+1+n(1+1+1)) = O(2+3n) = \boxed{O(n)}$

Todos los if y else tienen eficiencia $O(1)$

→ Primer bucle

$i(n) = O(1)$

$a(n) = O(1)$

$g(n) = O(1)$

$f(n) = O(n)$

$h(n) = O(n)$

$O(1+1+n(1+n+1)) = O(2+2n+n^2) = \boxed{O(n^2)}$

Por tanto, la eficiencia total del algoritmo de PD invertirBolsaPD es $\max(O(n), O(n^2))$

$\boxed{\text{EFICIENCIA TOTAL INVERTIR BOLSA PD} \Rightarrow O(n^2)}$

Como se ve, a primera vista los dos algoritmos son igual de eficientes sin embargo hay que tener en cuenta que en el algoritmo de programación dinámica aprovecha la memoria disponible mejor que el algoritmo de fuerza bruta. Ahí es donde se ve la diferencia en términos de eficiencia computacional entre los dos códigos.

6. Aplicación a dos instancias de problema concretas, que se puedan leer desde fichero de texto.

La creación de los ficheros de texto ha sido de la siguiente manera:

```
2 0.5 0.42
3 0.6 0.63
5 0.8 1.05
```

```
2 0.3 0.42
3 0.5 0.63
4 0.05 0.84
8 0.7 1.68
```

Como vemos hemos ideado dos ficheros que representan una tabla en la que se indica el precio de la acción, el beneficio y el coste de la comisión correspondientemente.La comisión se ha calculado como el 21% del precio.
Cada uno de los dos ficheros corresponde a una empresa diferente.

La lectura del fichero desde el main se ha implementado de la siguiente manera:

```
int dinero = stoi(argv[2]);
vector <Accion> acciones;
string linea;
//abrimos el fichero
entrada.open(argv[1]);
if(entrada.is_open()){
    //mientras sigan habiendo líneas en el fichero, las va leyendo
    while(getline(entrada, linea)){
        //fracciona cada elemento de la línea que se está analizando
        stringstream fraccion(linea);
        //datos que obtendremos de la lectura de cada línea
        int precio, comision;
        double beneficio;
        //los lee
        fraccion >> precio >> beneficio >> comision;
        //va creando objetos de acciones y los va metiendo al vector de acciones.
        Accion a(precio, beneficio, comision);
        acciones.push_back(a);
    }

}

//Comprobación de que se guardan bien los datos en el vector de acciones
cout << "Contenido del fichero de texto: " << endl;
for (int i=0; i<acciones.size(); i++){
    cout << acciones[i].Precio() << " " << acciones[i].Beneficio() << " " << acciones[i].Comision() << endl;
}
```