Práctica 4
Programación Dinámica

Joaquín Sergio García Ibáñez
Juan Navarro Maldonado

Índice:

1.- Diseño de componentes de Programación Dinámica

Podemos observar que para que un problema pueda resolverse usando la técnica de Programación Dinámica debe de completar esta serie de requisitos:

- El problema a resolver debe ser un problema de optimización (maximización/minimización).
- El problema debe poder resolverse por etapas.
- El problema debe poder modelarse mediante una ecuación recurrente.
- Debe existir uno o varios casos base al problema.
- Debe cumplirse el Principio de Optimalidad de Bellman: Si una secuencia de pasos para resolver un problema es óptima, entonces cualquier subsecuencia de estos mismos pasos también es óptima.

Podemos observar que cumple con todos los requisitos nuestro problema de encontrar el máximo beneficio mediante la inversión en N acciones para la i empresa sin tener que gastarse todo el dinero posible, asi maximizando tambien el dinero final despues de comprar las acciones, y minimizando el dinero que se gaste para que no se quede con poco dinero.
Por lo tanto, nosotros devolveremos una tupla con la empresa y el beneficio máximo de esa empresa al comprar X acciones que han puesto a la venta.
El objetivo es conocer beneficios(i,N) = máximo beneficio mediante la inversión en N acciones para la i empresa Caso base: beneficios(i, 0)= 0 para conocer beneficios(i,N) = máximo beneficio mediante la inversion en N acciones para la i empresa sin tener que gastarse todo el dinero posible.

Diseño de la ecuación recurrente:
El objetivo es conocer $D_n[i][j]$ que es el posible beneficio después de comprar j acciones y K será el dinero final que tenemos después de comprar las acciones.
El caso general supone que para j acciones, tengamos un beneficio para cada empresa, por tanto la ecuación recurrente sería:

$$D_n[i][j] \ = \ max\{D_{n-1}[i][j],\ D_n[i][j] \ + K\}$$

Para el caso base tendremos cuando no se invierte en ninguna empresa:

$$D_o[i][0] \ = \ 0$$

2.- Diseño e implementación del algoritmo básico a partir de la ecuación recurrente de forma directa

Hemos diseñado e implementado 2 algoritmos básicos con diferentes heurísticas:
Algoritmo básico 1:
Comparamos cual es el número Beneficio máximo de cada empresa y las ordenamos
Hasta comprar el número máximo de acciones que podemos comprar con el dinero inicial sin tener en cuenta el dinero final que gastaremos todo el dinero posible.

Pasos:
1.- Calcular el posible máximo beneficio por Acción de cada empresa
2.- Ordenar las empresas por el beneficio máximo
3.- Comprar el máximo de acciones de cada empresa siguiendo el orden anterior
4.- Si no se puede comprar una acción, comprobar las empresas restantes hasta quedarse sin dinero

algoritmoBasico(empresas, dineroInicial):
devolver = []

Para cada empresa en empresas:
listaEmpresasOrdenada = ordenamos empresas e insertamos en listaEmpresas

iterador = inicio de listaEmpresasOrdenada
empresa1 = valor de iterador

Imprimir "Empresas ordenadas por el posible beneficio"
Para cada empresa en listaEmpresasOrdenada:
Imprimir "Nombre: " + nombre de la empresa
Imprimir "EstimacionBeneficio: " + beneficio estimado de la empresa

Para cada empresa en listaEmpresasOrdenada:
Imprimir "Empresa: " + nombre de la empresa
numAcciones = numeroMaximoAccionesPuedoComprar(dineroInicial) de la empresa
Imprimir "DInero antes de la compra: " + dineroInicial
comprarAcciones(dineroInicial, numAcciones) de la empresa
Imprimir "Dinero despues de la compra: " + dineroInicial
Imprimir "Numero de acciones de la empresa: " + acciones de la empresa
Agregar (empresa, numAcciones) a devolver

Devolver tupla devolver

Algoritmo basico2:
Comparamos cual es el numero Beneficio máximo de cada acción de cada empresa y las ordenamos
Hasta comprar el número máximo de acciones que nos darán el máximo beneficio posible por cada acción con el dinero inicial, maximizando el beneficio y minimizando el dinero gastado.

Pasos:
1.- Calcular el posible máximo beneficio por Acción de cada empresa
2.- Ordenar las empresas por el beneficio máximo
3.- Comprar el máximo de acciones de cada empresa siguiendo el orden anterior siempre y cuando el beneficio por acción de la empresa sea positivo
4.- Si no se puede comprar una acción, comprobar las empresas restantes hasta quedarse sin dinero

```
algoritmoBasico2(empresas, dineroInicial):
    devolver = []

    Para cada empresa en empresas:
      listaEmpresasOrdenada = ordenamos empresas e insertamos en listaEmpresas

    iterador = inicio de listaEmpresasOrdenada
    empresa1 = valor de iterador

   Imprimir "Empresas ordenadas por el posible beneficio"
   Para cada empresa en listaEmpresasOrdenada:
       Imprimir "Nombre: " + nombre de la empresa
       Imprimir "EstimacionBeneficio: " + beneficio estimado de la empresa

   Para cada empresa en listaEmpresasOrdenada:
       imprimir "Empresa: " + nombre de la empresa
       Si accionXbeneficioXcomision de la empresa es mayor que 0 entonces:
          numAcciones = numeroMaximoAccionesPuedoComprar(dineroInicial) de la empresa
          Imprimir "DInero antes de la compra: " + dineroInicial
          comprarAcciones(dineroInicial, numAcciones) de la empresa
          Imprimir "Dinero despues de la compra: " + dineroInicial
          Imprimir "Numero de acciones de la empresa: " + acciones de la empresa
          Agregar (empresa, numAcciones) a devolver
       Sino:
          Imprimir "De esta empresa no se compra: " + accionXbeneficioXcomision de la
empresa

   Devolver  tupla de devolver
```

3.- Diseño del algoritmo de Programación Dinámica de acuerdo a las componentes

```
Procedure MatrizAdy(empresas[0..N])
        Para i=0 hasta NumEmpresas, hacer:
                Para j=0 hasta NumAcciones, hacer:
                        MatrizAdy[i][j] = empresas[i]
        Devolver MatrizAdy

Procedimiento Invertir(MatrizAdy[0..N][0..N])
        Para j=1 hasta NumAcciones, hacer:
```

Para i=0 hasta NumEmpresas, **hacer**:
Si MatrizAdy[i][j] > max **Y no se repite en devolver**
max = MatrizAdy[i][j]
aux_i = i
aux_j = j
Si dinero != 0
añadir a solución: empresa[i] y j
Devolver solución

Procedure RecuperaSolucion(solucion[0..N])
Para i=0 **hasta** tamSolucion
solucion = sumatoria del numero total de acciones a comprar de cada empresa

Devolver Solución

4.- Implementación del algoritmo de Programación Dinámica

Hemos decidido por comodidad a la hora de programar incluir las tres funciones de programación dinámica dentro de una misma función, que corresponde con esta parte del código:

```
1. for(int i = 0; i < numEmpresas; i++){
2.     for(int j = 1; j <= NUM_ACCIONES; j++){
3.         beneficios[i][j] = empresas[i].getaccionXbeneficioXcomision()*j;
4.     }
5. }
```

Cuyo resultado sería el siguiente:

```
Matriz beneficios
Fruterias Loli  -0.05 -0.1 -0.15 -0.2 -0.25 -0.3 -0.35 -0.4 -0.45 -0.5
Bar Paco        0 0 0 0 0 0 0 0 0 0
Chuches Paqui   1.5 3 4.5 6 7.5 9 10.5 12 13.5 15
Car-Wash Asuncion       -0.4 -0.8 -1.2 -1.6 -2 -2.4 -2.8 -3.2 -3.6 -4
Ferreteria Manolo       4.95 9.9 14.85 19.8 24.75 29.7 34.65 39.6 44.55 49.5
Academia Los Patos      1.9 3.8 5.7 7.6 9.5 11.4 13.3 15.2 17.1 19
```

Para el procedimiento invertir, el codigo es el siguiente:

```
1. while(capital_aux > 0 and puedeComprar){
2.         //cout << "\nIteracion " << ++contador << endl;
3.         max = 0;
4.         for(int j = 1; j <= NUM_ACCIONES; j++){
5.             for(int i = 0; i < numEmpresas; i++){
```

```
                if(beneficios[i][j] > max and !findOnTuple(listaCompra, i, j, empresas)){
                    //cout << "Mejor empresa " << empresas[i].getNombre() << endl;
                    max = beneficios[i][j];
                    aux_i = i;
                    aux_j = j;
                 }
            }
            if(max != 0){
                capital_aux -= empresas[aux_i].getPrecio()+empresas[aux_i].getComision();
                //cout << "Capitalaux: " << capital_aux << endl;
                if(capital_aux < 0){
                    //cout << "Se acabo el dinero"<< endl;
                    puedeComprar = false;

                }
                //cout << "Compramos acciones de: " << empresas[aux_i].getNombre() << " Cantidad: " << aux_j << endl;
                else
                listaCompra.push_back(make_tuple(empresas[aux_i], aux_j));


            }

        }
    }
```

Y por ultimo la funcion de recuperar solución quedaría representada de esta forma:

```
for(const auto &it: listaCompra){
        //cout << "Devolver.size() = " << devolver.size() << endl;
        Empresa elemento1 = get<0>(it);
        int elemento2 = get<1>(it);
        //cout << "\nNombre empresa: " << elemento1.getNombre() << " == " << empresaAux.getNombre() << endl;

        if(empresaAux.getNombre() == elemento1.getNombre()){
            //cout << "if " << elemento2 << " > " << numAccionesAux << endl;
            if(numAccionesAux<elemento2){
                numAccionesAux = elemento2;
            }
        }else{
            if(primeraIteracion != 0)
                devolver.push_back(make_tuple(empresaAux, numAccionesAux));

            empresaAux = elemento1;
            numAccionesAux = elemento2;
            primeraIteracion++;
        }
    }
```

5.- Cálculo de la eficiencia

5.1.- Algoritmo Básico

Puesto que implementamos una sobrecarga del operador < con una eficiencia de $O(n * log(n))$, la eficiencia de los algoritmos básicos, se vería afectada por ese, como en las demas lines del codigo tenemos varios bucles for que recorremos el vector de empresas de tamaño N podemos decir que tienen una eficiencia de $O(n)$ y usando la regla de la suma nos que dariamos con el maximo que en este caso es O(n*log(n))

5.2.- Algoritmo P.D.

En este algoritmo hemos formado una matriz de N empresas por K acciones, por lo tanto al recorrer la matriz tenemos eficiencia $O(N * K)$ siendo K el numero de acciones puestas a la venta por las empresas.

6.- Aplicación de dos instancias de problema concretas

Para el algoritmo básico 1 podemos ver que la persona compra todas las acciones posibles sin tener en cuenta el beneficio que obtendrá tras las compras, es decir maximiza el número de acciones compradas por la persona.

Para el algoritmo de Programación Dinámica podemos ver que esta vez va a tener en cuenta el comprar el máximo número de acciones que le salgan rentables(que obtenga el máximo beneficio total tras la compra), es decir obtener el máximo beneficio comprando el número máximo de acciones de una empresa que le
salga rentable.

Ejecución 1

```
Matriz beneficios
Fruterias Loli   -0.05 -0.1 -0.15 -0.2 -0.25 -0.3 -0.35 -0.4 -0.45 -0.5
Bar Paco         0 0 0 0 0 0 0 0 0 0
Chuches Paqui    1.5 3 4.5 6 7.5 9 10.5 12 13.5 15
Car-Wash Asuncion        -0.4 -0.8 -1.2 -1.6 -2 -2.4 -2.8 -3.2 -3.6 -4
Ferreteria Manolo        4.95 9.9 14.85 19.8 24.75 29.7 34.65 39.6 44.55 49.5
Academia Los Patos       1.9 3.8 5.7 7.6 9.5 11.4 13.3 15.2 17.1 19
```

```
        ALGORITMOS DE PROGRAMACION DINAMICA RESULTADOS
Matriz beneficios
Fruterias Loli  -0.05 -0.1 -0.15 -0.2 -0.25 -0.3 -0.35 -0.4 -0.45 -0.5
Bar Paco        0 0 0 0 0 0 0 0 0 0
Chuches Paqui   1.5 3 4.5 6 7.5 9 10.5 12 13.5 15
Car-Wash Asuncion       -0.4 -0.8 -1.2 -1.6 -2 -2.4 -2.8 -3.2 -3.6 -4
Ferreteria Manolo       4.95 9.9 14.85 19.8 24.75 29.7 34.65 39.6 44.55 49.5
Academia Los Patos      1.9 3.8 5.7 7.6 9.5 11.4 13.3 15.2 17.1 19

Vector devolver

Nombre empresa: Ferreteria Manolo Acciones compradas: 10
Dinero antes de la compra: 100
Dinero despues de la compra 49.9
Numero de acciones de la empresa restantes: 0

Nombre empresa: Academia Los Patos Acciones compradas: 10
Dinero antes de la compra: 49.9
Dinero despues de la compra 38.9
Numero de acciones de la empresa restantes: 0

Nombre empresa: Chuches Paqui Acciones compradas: 3
Dinero antes de la compra: 38.9
Dinero despues de la compra 8.75
Numero de acciones de la empresa restantes: 7

Empresa: Ferreteria Manolo
Acciones Compradas: 10
Empresa, beneficio por accion: 2 * 10

Empresa: Academia Los Patos
Acciones Compradas: 10
Empresa, beneficio por accion: 3 * 10

Empresa: Chuches Paqui
Acciones Compradas: 3
Empresa, beneficio por accion: 1.2 * 3


Dinero final: 8.75
Posible Beneficio 166
Dinero total 174.75
```

```
        ALGORITMO BASICO RESULTADOS

        Empresas ordenadas por el posible beneficio

Nombre: Ferreteria Manolo
EstimacionBeneficio: 4.95

Nombre: Academia Los Patos
EstimacionBeneficio: 1.9

Nombre: Chuches Paqui
EstimacionBeneficio: 1.5

Nombre: Bar Paco
EstimacionBeneficio: 0

Nombre: Fruterias Loli
EstimacionBeneficio: -0.05

Nombre: Car-Wash Asuncion
EstimacionBeneficio: -0.4

Nombre empresa: Ferreteria Manolo
Dinero antes de la compra: 100
Dinero despues de la compra 49.9
Numero de acciones de la empresa: 0

Nombre empresa: Academia Los Patos
Dinero antes de la compra: 49.9
Dinero despues de la compra 38.9
Numero de acciones de la empresa: 0

Nombre empresa: Chuches Paqui
Dinero antes de la compra: 38.9
Dinero despues de la compra 8.75
Numero de acciones de la empresa: 7

Nombre empresa: Bar Paco
Dinero antes de la compra: 8.75
Dinero despues de la compra 8.75
Numero de acciones de la empresa: 10

Nombre empresa: Fruterias Loli
Dinero antes de la compra: 8.75
Dinero despues de la compra 1.05
Numero de acciones de la empresa: 3

Nombre empresa: Car-Wash Asuncion
Dinero antes de la compra: 1.05
Dinero despues de la compra 1.05
Numero de acciones de la empresa: 10

        Numero de acciones compradas:

Empresa: Ferreteria Manolo
Acciones Compradas: 10
Empresa, beneficio por accion: 2 * 10

Empresa: Academia Los Patos
Acciones Compradas: 10
Empresa, beneficio por accion: 3 * 10

Empresa: Chuches Paqui
```

```
        ALGORITMO BASICO RESULTADOS

        Empresas ordenadas por el posible beneficio

Nombre: Ferreteria2
EstimacionBeneficio: 1.96

Nombre: Polleria
EstimacionBeneficio: 1.5

Nombre: Car-Wash2
EstimacionBeneficio: 1

Nombre: Joyeria Isidoro
EstimacionBeneficio: 0.8

Nombre: Tejidos Manoli
EstimacionBeneficio: 0.75

Nombre: Academia2
EstimacionBeneficio: 0.15

Nombre empresa: Ferreteria2
Dinero antes de la compra: 100
Dinero despues de la compra 59.9
Numero de acciones de la empresa: 0

Nombre empresa: Polleria
Dinero antes de la compra: 59.9
Dinero despues de la compra 9.65
Numero de acciones de la empresa: 5

Nombre empresa: Car-Wash2
Dinero antes de la compra: 9.65
Dinero despues de la compra 9.65
Numero de acciones de la empresa: 10

Nombre empresa: Joyeria Isidoro
Dinero antes de la compra: 9.65
Dinero despues de la compra 9.65
Numero de acciones de la empresa: 10

Nombre empresa: Tejidos Manoli

        Numero de acciones compradas:

Empresa: Ferreteria2
Acciones Compradas: 10
Empresa, beneficio por accion: 1.5 * 10

Empresa: Polleria
Acciones Compradas: 5
Empresa, beneficio por accion: 1.2 * 5

Empresa: Academia2
Acciones Compradas: 8
Empresa, beneficio por accion: 1.2 * 8


Dinero final: 1.25
Posible Beneficio 129.6
Dinero total 130.85
```

```
        ALGORITMOS DE PROGRAMACION DINAMICA RESULTADOS
Matriz beneficios
Tejidos Manoli  0.75 1.5 2.25 3 3.75 4.5 5.25 6 6.75 7.5
Joyeria Isidoro         0.8 1.6 2.4 3.2 4 4.8 5.6 6.4 7.2 8
Polleria        1.5 3 4.5 6 7.5 9 10.5 12 13.5 15
Car-Wash2       1 2 3 4 5 6 7 8 9 10
Ferreteria2     1.96 3.92 5.88 7.84 9.8 11.76 13.72 15.68 17.64 19.6
Academia2       0.15 0.3 0.45 0.6 0.75 0.9 1.05 1.2 1.35 1.5

Vector devolver

Nombre empresa: Ferreteria2 Acciones compradas: 10
Dinero antes de la compra: 100
Dinero despues de la compra 59.9
Numero de acciones de la empresa restantes: 0

Nombre empresa: Polleria Acciones compradas: 5
Dinero antes de la compra: 59.9
Dinero despues de la compra 9.65
Numero de acciones de la empresa restantes: 5

Empresa: Ferreteria2
Acciones Compradas: 10
Empresa, beneficio por accion: 1.5 * 10

Empresa: Polleria
Acciones Compradas: 5
Empresa, beneficio por accion: 1.2 * 5


Dinero final: 9.65
Posible Beneficio 120
Dinero total 129.65
PS C:\Users\Wavin\Documents\GitHub\Algoritmica\Practica4>
```