Práctica de Programación 2

Mikel Dalmau

13 de Abril de 2015

Resumen

En la siguiente práctica se realiza un estudio analítico y experimental de un problema de optimización que se resolverá utilizando programación dinámica.

1 Enunciado del problema

El gerente de una planta de producción debe planificar los productos que conviene fabricar en un día determinado. En la base de datos de la empresa cuentan con la información siguiente:

- una tabla T(1 . . n) en la que T(i) indica el tiempo (en minutos) necesario para fabricar un producto de clase i : 1 . . n.
- una tabla B(1 . . n) en la que B(i) indica el beneficio que se obtiene por cada producto de clase i : 1 . . n fabricado.
- una tabla C(1 . . n) en la que C(i) indica el número máximo de unidades del producto de clase i : 1 . . n que pueden fabricarse ese día, dadas las existencias de componentes básicos en la planta.

Determinar cuántos productos de cada clase conviene producir para conseguir el máximo beneficio en M minutos.

2 Estudio analítico

El problema expuesto anteriormente es idéntico al problema de la mochila sin repetición, pero en el contexto de una planta de producción. A diferencia del problema de la mochila con repetición, donde tenemos elementos ilimitados para utilizar aquí no sucede eso y por lo tanto tenemos que conocer de alguna manera que elementos hemos utilizado hasta el momento.

Así como en el problema de la mochila utilizábamos un array con todos los valores de w donde:

$K(w)$ = valor máximo alcanzable con una mochila de tamaño w

Año: 2014 - 2015
Profesor(a): Jesus Bermudez De Andres
Teléfono:
E-mail: mdalmau002@ikasle.ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

En este caso utilizaremos la siguiente estructura, una matriz de esta forma:

$K(w,j)$ = valor máximo alcanzable con una mochila de tamaño w utilizando los objetos 1 hasta j

El siguiente ejemplo ilustra mejor esta idea:

objeto	valor	peso
1	3	2
2	1	3
3	2	1
4	2	1
5	4	5

Para una mochila de tamaño 5 la matriz K tendría este aspecto.

obj	peso					
	ϕ	1	2	3	4	5
ϕ	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	3	3	4
3	0	2	3	5	5	5
4	0	2	4	4	7	7
5	0	2	4	4	7	7

De esta manera la solución se encontrará en la última posición de la matriz, volviendo a nuestro problema será el máximo beneficio alcanzable en M minutos con los n productos disponibles, $K(M,N)$.

La potencia y simplicidad de la programación dinámica reside en el concepto de utilizar la memoria y los valores ya calculados para determinar si el elemento j sería necesario para alcanzar el beneficio óptimo.

La función de optimización expresada recursivamente sería la siguiente:

$$K(w,j) = max\{\ K(w-w_j, j-1) + b_j,\ K(w, j-1)\}\ \ ,\ \ siempre\ \ y\ \ cuando\ \ w > w_j$$

El algoritmo por lo tanto consiste en ir rellenando la matriz K e ir marcando que casillas pertenecen a alguna de las soluciones aunque no necesariamente tienen que pertenecer a la solución óptima, luego recorrer la matriz desde el último elemento saltando entre las posiciones de la matriz e ir obteniendo la solución óptima, cuando un objeto es parte de la solución avanzaremos una fila hacia arriba y tantas columnas como peso tenga el objeto ya que habremos liberado a la mochila de ese peso, si no pertenece a la solución entonces tan solo avanzaremos una fila hacia arriba.

Pseudocódigo de Dasgupta [Das] adaptado al problema.

Rellenar la matriz K.

```
Inicializar todos K(0; j) = 0 y todos K(w; 0) = 0
j = 0
for i = 1 to n:
   for k = 1 to max unidades producibles:
   j++;
      for w = 1 to W:
         if w_j > w: K(w, j) = K(w, j - 1)
            K(w, j).pertenece = false;
         else:
            if K(w - w_j, j - 1) + v_j > K(w, j - 1)
               K(w, j) = K(w - w_j, j - 1) + v_j
               K(w, j).pertenece = true
            else:
               K(w, j) = K(w, j - 1)
               K(w, j).pertenece = false
```

Recorrer la matriz K.

```
w = W
j = N
while(w ≠ 0 and j ≠ 0)
   if K(w, j).pertenece
      añadir a la solución K(w, j)
      w = w - w_j
      j = j - 1
   else:
      j = j - 1
```

Coste del algoritmo:

Analizando el pseudo-código podemos ver que el algoritmo recorre cada posición de la matriz una sola vez y realiza una serie comparaciones y accesos cada vez, todas de estas, operaciones de complejidad constante. Por lo tanto el coste del algoritmo dependerá de W los minutos disponibles para producir y del número total de productos N, en nuestro problema esto será igual al número de productos multiplicado por el máximo producible de cada uno:

$$f(n) = \sum_{i=1}^{N} \sum_{j=1}^{W} 1 \equiv O(NW)$$

3 Estudio experimental

La siguiente gráfica muestra como aumenta el tiempo de computo en función tanto de N como de W y demuestra que el análisis del costo hecho previamente es se corresponde con la realidad.

Para no distorsionar los resultado se han tomado consideraciones como hacer que todos los productos tengan 100 elementos máximos producibles y no dejar este valor como aleatorio. También se han tomado cinco pruebas en cada caso y se ha realizado una media.

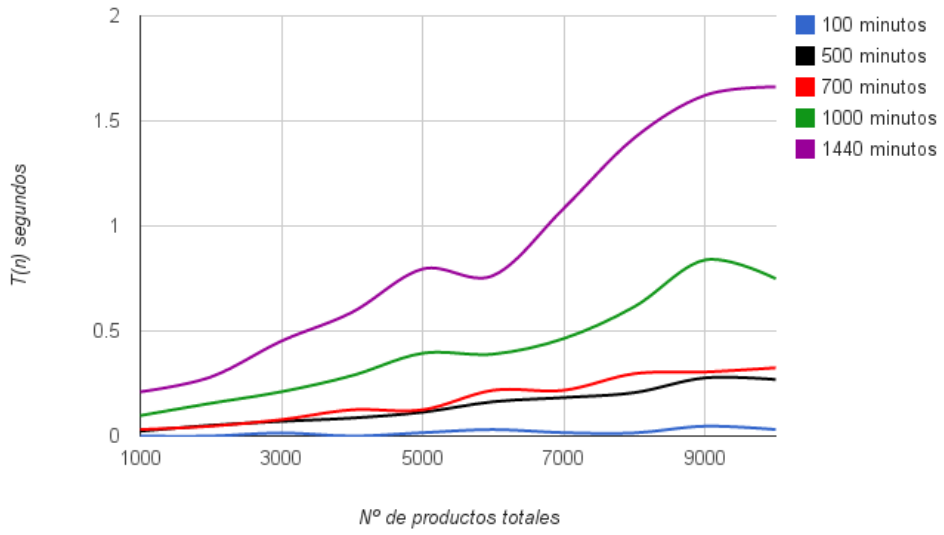

Figura 1: Representación gráfica de las las pruebas del problema de la planta de producción.

N	T(s) 100 M	T(s) 500 M	T(s) 700 M	T(s) 1000 M	T(s) 1440 M
1000	0	0.023	0.031	0.098	0.211
2000	0	0.051	0.047	0.156	0.281
3000	0.015	0.07	0.078	0.211	0.452
4000	0	0.086	0.125	0.288	0.589
5000	0.016	0.114	0.125	0.394	0.795
6000	0.031	0.164	0.218	0.39	0.764
7000	0.016	0.183	0.218	0.464	1.0845
8000	0.015	0.207	0.297	0.616	1.42
9000	0.047	0.277	0.305	0.838	1.622
10000	0.031	0.269	0.325	0.749	1.662
10000	1.499	1.92	1.419	1.81	1.662

Tabla 1: Tiempos medios de las pruebas.

Bibliografía

[cppRef] http://es.cppreference.com .

[NotasDA] https://egela.ehu.es/pluginfile.php/768559/mod_resource/content/1/NotasDeClase7–12.pdf.

[Das] Algorithms, S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani.

4 Anéxo

Comentarios:

Las siguientes páginas contienen todo el código utilizado, en lenguaje c++.

```
/*
* File:   main.cpp
* Author: mikel
*
* Created on 25 de marzo de 2015, 15:47
*/

#include <vector>
#include <cstdlib>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <cmath>
#include <time.h>

using namespace std;

vector<string> pruebas;

typedef struct {
    int tiempo;
    int beneficio;
    int unidadesMax;
    int unidadesSoluccion;
}producto;

typedef struct {
    int beneficioSuma;
    bool pertenece;
    int numProducto;
}valoresF;

int leerFichero(string fname, vector<producto> *pV);
void crearFicheroPrueba(int numProd, int numMinutos , string fname);
void imprimirTablaProduccion(vector<producto> *pV,int pMinutos);
void ejecutarBloquePruebas();
void crearBloquePruebas();
void imprimirSolucion(vector<producto> *pT);
void calcularBeneficioMaximo(string fname);
/*
*
*/
```

```
int main() {
    crearBloquePruebas();
    ejecutarBloquePruebas();
    return 0;
}

void calcularBeneficioMaximo(string fname){
    int Minutos,Ksize = 0, Tj,Bj;

    vector<producto> Tabla;
    Minutos = leerFichero(fname,&Tabla);

    for(int i = 0; i < Tabla.size(); i++){
        Ksize = Ksize + Tabla[i].unidadesMax;
    }

    //Inicializamos la matriz de optimizacion
    valoresF** K;

    K = new valoresF* [Minutos+1];
    for (int j = 0; j < Minutos+1; j++)
        K[j] = new valoresF [Ksize+1];

    //Inicializamos fila y columna 0
    for (int i = 0; i < Ksize+1; i++){
        K[0][i].beneficioSuma = 0;
        K[0][i].pertenece = false;
        K[0][i].numProducto = 0;
    }
    for(int i = 0; i < Minutos+1; i++){
        K[i][0].beneficioSuma = 0;
        K[i][0].pertenece = false;
        K[i][0].numProducto = 0;
    }
    int i = 0;
    for(int j = 0; j < Tabla.size(); j++){
        Tj = Tabla[j].tiempo;
        Bj = Tabla[j].beneficio;
        for(int k = 0; k < Tabla[j].unidadesMax; k++){
            i++;
            for(int t = 1; t < Minutos+1; t++){
                K[t][i].numProducto = j;
                if( Tj > t){
                    K[t][i].beneficioSuma = K[t][i-1].beneficioSuma;
                    K[t][i].pertenece = false;
                }else{
                    if(K[t][i-1].beneficioSuma < K[t-Tj][i-1].beneficioSuma + Bj){
                        K[t][i].beneficioSuma = K[t-Tj][i-1].beneficioSuma + Bj;
                        K[t][i].pertenece = true;
                    }else{
                        K[t][i].beneficioSuma = K[t][i-1].beneficioSuma;
                        K[t][i].pertenece = false;
                    }
                }
            }
        }
    }
    //Recorremos la matriz incluyendo los elementos que perteneces a la solucion
    i = Minutos;
    int j = Ksize;
    while((i != 0) and (j != 0)){
        if(K[i][j].pertenece){
            Tabla[K[i][j].numProducto].unidadesSoluccion++;
            i = i - Tabla[K[i][j].numProducto].tiempo;
            j--;
        }else{
```

```
            j--;
        }
    }
    imprimirSolucion(&Tabla);
};

void imprimirSolucion(vector<producto> *pT){
    for(int i = 0; i < pT->size(); i++){
        cout << pT->at(i).unidadesSoluccion << endl;
    }
};
/*
* Input: Nombre de fichero y apuntador al vector de productos
* Outpu: Rellena array de enteros con los datos del fichero y devuelve en numero de minutos de
     produccion
*/
int leerFichero(string fname, vector<producto> *pV){
    int i,lines,Minutos;
    ifstream fe(fname.c_str());

    fe >> lines;
    if(not(lines > 100 or lines < 1)){
        vector<producto> V(lines);
        for(i = 0; i < lines ; i++){
            fe >> V.at(i).tiempo;
            fe >> V.at(i).beneficio;
            fe >> V.at(i).unidadesMax;
            V.at(i).unidadesSoluccion = 0;
        }
        *pV = V;
        fe >> Minutos;
        fe.close();
        return Minutos;
    }
    fe.close();
    return -1;
};
/*
* Input: int size, tamano de la lista de elementos, fname nombre para el fichero
* Outpu: Construye un fichero con un entero positivo en la primera linea indicando el numero
    de lineas
*  que contiene el fichero
*/
void crearFicheroPrueba(int numProd, int numMinutos , string fname){
    int i,tiempo, benficio, unidadesMax;
    ofstream fs(fname.c_str());

    fs << numProd << endl;
    for (i = 0; i < numProd ; i++ ){
        tiempo = 1 + (rand() % 1440);        // Tiempo de produccion posible [ 1 - 1440]
        benficio = 1 + (rand() % 50);      // Beneficio posible [1 - 50]
        //unidadesMax = 1 + (rand() % 10);  // unidadesMaximas posibles [1 - 100]
        unidadesMax = 100;
        fs << tiempo << " " << benficio << " " << unidadesMax << endl;
    }
    fs << numMinutos << endl;
    pruebas.push_back(fname);
    fs.close();
};
/*
* Input: vector de productos
* Outpu: imprime por pantalla la tabla de produccion
*/
void imprimirTablaProduccion(vector<producto> *pT,int pMinutos){

    cout <<"Tabla de Produccion"<< endl;
```

```
        cout << endl;
        cout << "N\t" <<"Tiempo\t" << "Benef\t"<< "MaxUnid\t"<< "CantidadOPT\t" << endl;
        for(int i = 0; i < pT->size(); i++){
            cout << i+1 << "\t"<< pT->at(i).tiempo << "\t" << pT->at(i).beneficio << "\t" << pT->
                at(i).unidadesMax << "\t"<< pT->at(i).unidadesSoluccion << endl;
        }
        cout << endl;
        cout << "Tiempo para producir (min):  " << pMinutos << endl;
    };

    void ejecutarBloquePruebas(){

        clock_t tic,toc;
        for(int i =0; i< pruebas.size(); i++){
            tic = clock();
            calcularBeneficioMaximo(pruebas.at(i));
            toc = clock();
            //cout << pruebas.at(i) << " : " << ((double)(toc - tic) / CLOCKS_PER_SEC)<< "
                segundos" <<endl;
            cout << ((double)(toc - tic) / CLOCKS_PER_SEC) << endl;;
            if(i == 9 or i == 19 or i == 29){
                cout << endl;
            }

        }
    };
    void crearBloquePruebas(){

        crearFicheroPrueba(10, 100, "10prod500min.txt");
        crearFicheroPrueba(20, 100, "20prod500min.txt");
        crearFicheroPrueba(30, 100, "30prod500min.txt");
        crearFicheroPrueba(40, 100, "40prod500min.txt");
        crearFicheroPrueba(50, 100, "50prod500min.txt");
        crearFicheroPrueba(60, 100, "60prod500min.txt");
        crearFicheroPrueba(70, 100, "70prod500min.txt");
        crearFicheroPrueba(80, 100, "80prod500min.txt");
        crearFicheroPrueba(90, 100, "90prod500min.txt");
        crearFicheroPrueba(100, 100, "100prod500min.txt");

    };
```