

Prácticas Algorítmica – Curso 2015/2016

Francisco Javier Caracuel Beltrán

Índice

Práctica 1 – Eficiencia.....	2
Práctica 2 – Algoritmos Greedy.....	6
Práctica 3 – Programación dinámica.....	13

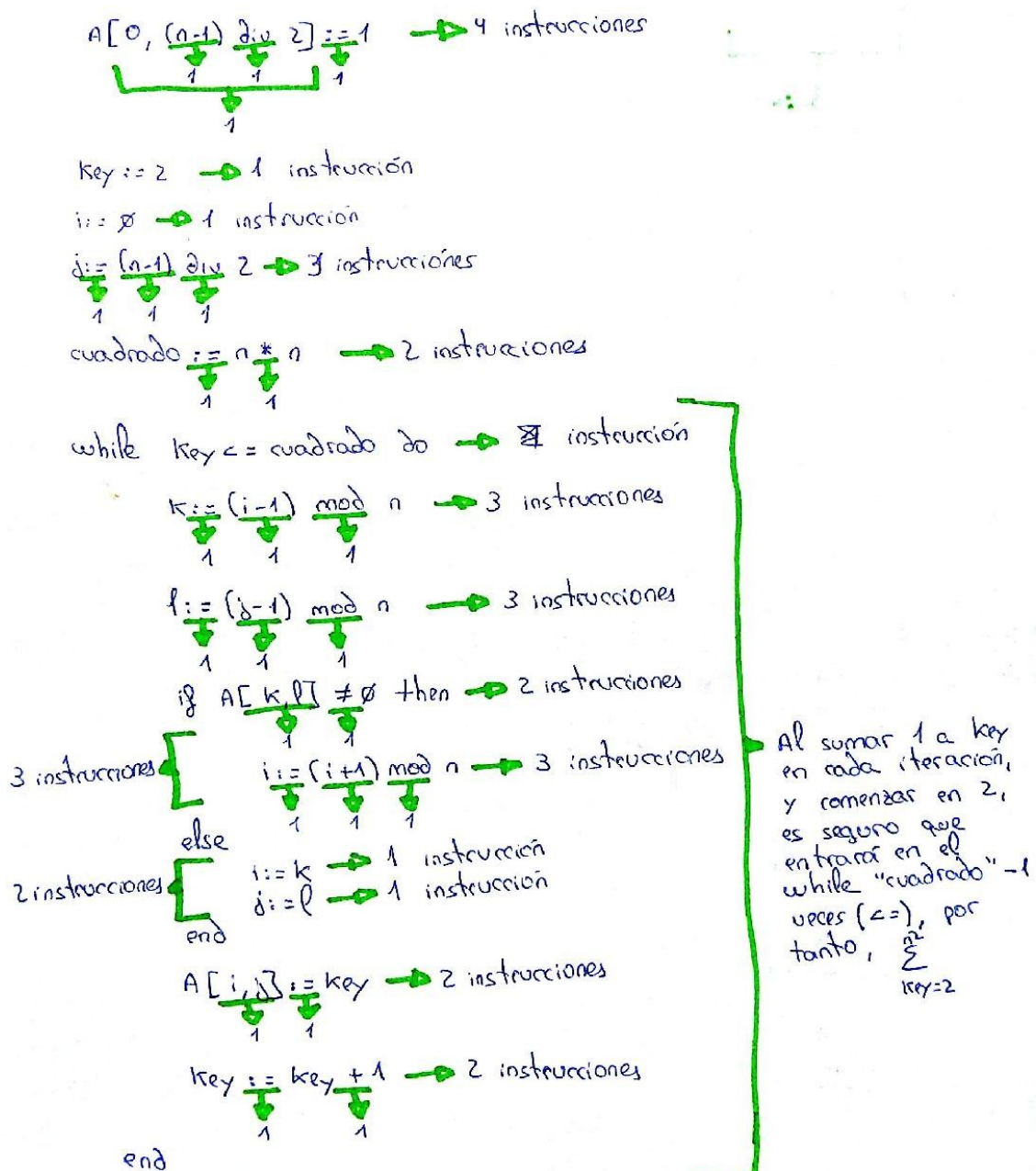
[CORREGIDAS](#)

Práctica 1 – Eficiencia

FRANCISCO JAVIER CARACUEL BELTRÁN

TEMA 1, PÁGINA 16.

a) Estudiar $f(n)$:



$$T(n) = 4 + 1 + 1 + 3 + 2 + \sum_{key=2}^{n^2} (1 + 3 + 3 + 2 + \max(\underbrace{T(18)}_3, \underbrace{T(else)}_2) + 2 + 2) =$$

$$= 11 + \sum_{key=2}^{n^2} (16) = 11 + 16 \sum_{key=2}^{n^2} 1 = 11 + 16 \cdot 1 (n^2 - 2 + 1) = 11 + 16(n^2 - 1)$$

Hay que conseguir una expresión de la forma $\sum_{i=a}^b k$,

para aplicar la fórmula: $\sum_{i=a}^b k = k(b - a + 1)$

$$T(n) = 11 + 16(n^2 - 1) = 11 + 16n^2 - 16 = \frac{16n^2 - 5}{1}$$

no tiene
importancia
en el cálculo

al no tener
importancia
se ignoran

$$T(n) \in O(n^2)$$

Se considera que
es un resultado
bueno porque da
soluciones en tiempo
real.

Práctica 2 - Algoritmos Greedy

2.1 Código fuente:

```
/*
 * File: main.cpp
 * Author: fran
 */

#include <cstdlib>
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// e -> vector con los elementos (peso, beneficio)
// n -> número de objetos disponibles
// M -> capacidad de la mochila
// s -> vector solución

////////////////////////////////////
// struct para guardar los datos de cada elemento
//
struct Element {

    int id;    // identificado
    double w; // peso
    double p; // beneficio

};

////////////////////////////////////
// struct para ordenar los elementos del vector. Se ordena por beneficio/peso
//
struct Compare {

    bool operator() (Element e1, Element e2) {
        return e1.p/e1.w > e2.p/e2.w;
    }
}
```

```

} comparator;

////////////////////////////////////
// Pide los datos del problema
//
void loadData(int &n, double &M, vector<Element> &e);

////////////////////////////////////
// Rellena el vector solución en base a un vector de elementos
//
void backpack(vector<double> &s, vector<Element> e, int n, double M);

////////////////////////////////////
// Calcula el resultado del algoritmo
//
double getProfit(vector<double> &s, vector<Element> e, int n);

////////////////////////////////////
// Muestra que elementos se coge y su proporción
//
void showRes(vector<double> &s, vector<Element> e, int n);

////////////////////////////////////
// main
//
int main(int argc, char** argv) {

    // Vectores con los pesos y el beneficio
    vector<Element> e;

    // Numero de elementos
    int n;

    // Capacidad de la mochila
    double M;

    // Vector solución
    vector<double> s;

```

```

// Resultado total
double total;

loadData(n, M, e);

// Se ordena el vector siguiendo el criterio de beneficio/peso mayor primero
sort(e.begin(), e.end(), comparator);

// Se llama al algoritmo greedy y s tendrá la solución
backpack(s, e, n, M);

total = getProfit(s, e, n);

cout << endl;

cout << "\nEl beneficio total es: " << total << endl;

showRes(s, e, n);

return 0;
}

void loadData(int &n, double &M, vector<Element> &e){

    double aux1, aux2;

    // Se pide la capacidad
    cout << "\nIntroduce la capacidad de la mochila: ";

    cin >> M;

    // Se pide el número de elementos
    cout << "\nIntroduce el número de elementos a introducir: ";

    cin >> n;

    cout << endl;

    // Se piden todos los pesos y sus beneficios

```

```

for(unsigned int i=0; i<n; i++){

    cout << "\nIntroduce el peso y beneficio del elemento " << i+1 << ": ";
    cin >> aux1 >> aux2;

    Element element;
    element.id = i+1;
    element.w = aux1;
    element.p = aux2;

    e.push_back(element);

}

cout << endl;

}

void backpack(vector<double> &s, vector<Element> e, int n, double M){

    // Se inicializa el vector solución a 0
    for(unsigned int i=0; i<n; i++){
        s.push_back(0);
    }

    // Se inicializa el peso actual de la mochila
    double weight = 0;

    // Contador para no sobrepasar el tamaño del vector
    unsigned int i = 0;

    // Se recorrerán todos los elementos del vector hasta que la mochila
    // esté completa
    while(weight < M){

        // Se guarda el mejor elemento restante
        double bestW = e[i].w;

        // Si el peso actual + peso del objeto es menor o igual a la capacidad
        // de la mochila se introduce al completo
    }
}

```

```

    if(weight + bestW <= M){
        s[i] = 1;
        // Se debe actualizar el peso actual
        weight += bestW;

        // Si no se puede coger al completo, se introduce la parte
        // correspondiente
    } else{

        // Se guarda el % de peso que se coge del último elemento
        s[i] = (M - weight) / bestW;

        // El peso actual es el total de la capacidad
        weight = M;

    }

    i++;
}

}

double getProfit(vector<double> &s, vector<Element> e, int n){

    double total = 0;

    for(unsigned int i=0; i<n; i++){

        if(s[i] != 0){

            total += s[i]*e[i].p;

        }

    }

    return total;

}

```



```

void showRes(vector<double> &s, vector<Element> e, int n){

    cout << "\nSe coge el elemento número:\n";

    for(unsigned int i=0; i<n; i++){

        if(s[i] != 0){

            cout << " " << e[i].id << " y el " << s[i]*100 << "%.\n";

        }

    }

    cout << endl;

}

```

2.2 Ejemplo 1:

Output

```

p2-greedy (Build, Run) x p2-greedy (Run) x
Introduce la capacidad de la mochila: 10
Introduce el número de elementos a introducir: 4
Introduce el peso y beneficio del elemento 1: 10 10
Introduce el peso y beneficio del elemento 2: 3 9
Introduce el peso y beneficio del elemento 3: 3 9
Introduce el peso y beneficio del elemento 4: 4 9

El beneficio total es: 27

Se coge el elemento número:
2 y el 100%.
3 y el 100%.
4 y el 100%.

RUN FINISHED; exit value 0; real time: 26s; user: 0ms; system: 0ms

```

2.3 Ejemplo 2:

```
Output
p2-greedy (Build, Run) x p2-greedy (Run) x
Introduce la capacidad de la mochila: 10 4
Introduce el número de elementos a introducir:
Introduce el peso y beneficio del elemento 1: 10 10
Introduce el peso y beneficio del elemento 2: 3 1
Introduce el peso y beneficio del elemento 3: 3 1
Introduce el peso y beneficio del elemento 4: 4 1

El beneficio total es: 10
Se coge el elemento número:
1 y el 100%.

RUN FINISHED; exit value 0; real time: 8s; user: 0ms; system: 0ms
```

Práctica 3 – Programación dinámica

3.1 Código fuente:

```
/*
 * File: main.cpp
 * Author: fran
 */

#include <cstdlib>
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

////////////////////////////////////
// struct para guardar los datos de cada elemento
//
struct Element {

    int id; // identificado
    int w; // peso
    int p; // beneficio

};

////////////////////////////////////
// Inicialización de la matriz resultado
//
void init(vector<vector<int> > &results, vector<Element> &e, vector<int> &x,
          int &M, int &n);

////////////////////////////////////
// Rellena la matriz con los subproblemas
//
void fillBackpack(vector<vector<int> > &V, vector<Element> e, int n, int M);

////////////////////////////////////
```

```

// Rellena el vector solución
//
void fillSolution(const vector<vector<int> > &V, const vector<Element> &e,
    vector<int> &x, int n, int M);

////////////////////////////////////
// main
//
int main(int argc, char** argv) {

    // Matriz con los resultados de los subproblemas
    vector<vector<int> > V;

    // Capacidad de la mochila
    int M;

    // Número de elementos
    int n;

    // Vector con los pesos y su beneficio
    vector<Element> e;

    // Vector con la solución
    vector<int> x;

    // Se piden los datos y se inicializa la matriz con los resultados
    init(V, e, x, M, n);

    fillBackpack(V, e, n, M);

    // Ver la matriz rellena
    /*for(unsigned int i=0; i<=n; i++){

        cout << endl;

        for(unsigned int j=0; j<=M; j++){

            cout << V[i][j] << " ";

        }
    }

```

```
*/
```

```
fillSolution(V, e, x, n, M);
```

```
cout << endl << "El beneficio máximo es " << V[n][M] << endl << endl;
```

```
cout << "Los elementos que se deben coger son el número: ";
```

```
for(int i=0; i<n; i++){
```

```
    if(x[i] == 1){
```

```
        cout << i+1;
```

```
        if(i!=n-1){
```

```
            cout << ", ";
```

```
        }
```

```
    }
```

```
}
```

```
cout << endl;
```

```
return 0;
```

```
}
```

```
void init(vector<vector<int> > &V, vector<Element> &e, vector<int> &x,  
    int &M, int &n){
```

```
    int aux1, aux2;
```

```
    // Se pide la capacidad
```

```
    cout << "\nIntroduce la capacidad de la mochila: ";
```

```
    cin >> M;
```

```
    // Se pide el número de elementos
```

```
cout << "\nIntroduce el número de elementos a introducir: ";

cin >> n;

cout << endl;

// Se piden todos los pesos y sus beneficios
for(unsigned int i=0; i<n; i++){

    cout << "\nIntroduce el peso y beneficio del elemento " << i+1 << ": ";
    cin >> aux1 >> aux2;

    Element element;
    element.id = i+1;
    element.w = aux1;
    element.p = aux2;

    e.push_back(element);

}

cout << endl;

// Se inicializa la matriz NxM con 0
for(unsigned int i=0; i<=n; i++){

    vector<int> row;

    for(unsigned int j=0; j<=M; j++){

        row.push_back(0);

    }

    V.push_back(row);

}

// Se inicializa a 0 el vector solución
for(int i=0; i<n; i++){
```

```

    x.push_back(0);
}

}

void fillBackpack(vector<vector<int> > &V, vector<Element> e, int n, int M){

    for(int i=1; i<=n; i++){

        for(int j=1; j<=M; j++){

            // El bucle for empieza en n, pero el vector de elementos
            // tiene tamaño n-1, por esto se le resta 1 a la posición del vector
            if(j-e[i-1].w < 0){
                V[i][j] = V[i-1][j];
            } else{
                V[i][j] = max(V[i-1][j], e[i-1].p + V[i-1][j-e[i-1].w]);
            }

        }

    }

}

void fillSolution(const vector<vector<int> > &V, const vector<Element> &e,
    vector<int> &x, int n, int M){

    // Se recoge el peso máximo de la mochila
    int j = M;

    // Se recorre la matriz para comprobar los resultados
    for(int i = n; i>=1; i--){

        if(V[i][j] != V[i-1][j]){

            // El bucle for empieza en n, pero el vector solución y de elementos
            // tiene tamaño n-1, por esto se le resta 1 a la posición del vector
            x[i-1] = 1;
            j = j-e[i-1].w;

        }

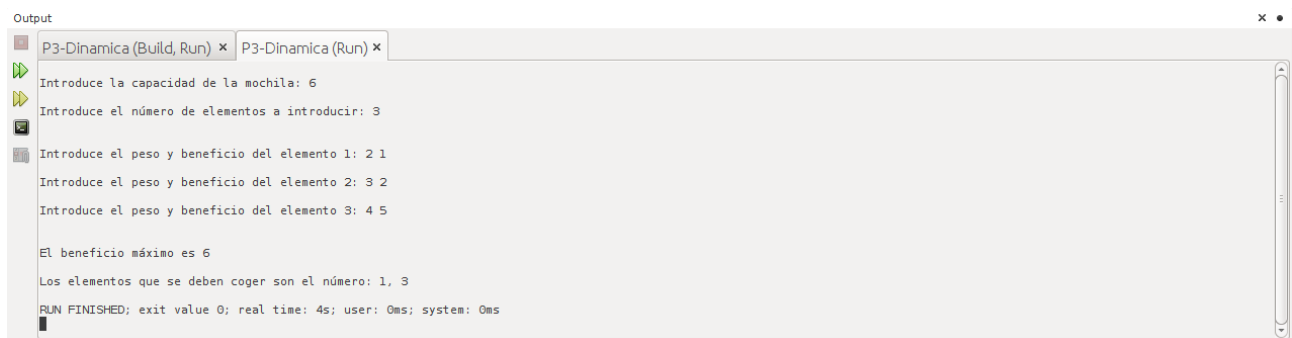
    }

}

```

```
}  
  
}  
  
}
```

3.1 Ejemplo 1:



Output

P3-Dinamica (Build, Run) x P3-Dinamica (Run) x

```
Introduce la capacidad de la mochila: 6  
Introduce el número de elementos a introducir: 3  
Introduce el peso y beneficio del elemento 1: 2 1  
Introduce el peso y beneficio del elemento 2: 3 2  
Introduce el peso y beneficio del elemento 3: 4 5  
  
El beneficio máximo es 6  
Los elementos que se deben coger son el número: 1, 3  
RUN FINISHED; exit value 0; real time: 4s; user: 0ms; system: 0ms
```

3.2 Ejemplo 2:



Output

P3-Dinamica (Build, Run) x P3-Dinamica (Run) x

```
Introduce la capacidad de la mochila: 7  
Introduce el número de elementos a introducir: 4  
Introduce el peso y beneficio del elemento 1: 1 2  
Introduce el peso y beneficio del elemento 2: 2 3  
Introduce el peso y beneficio del elemento 3: 3 4  
Introduce el peso y beneficio del elemento 4: 4 5  
  
El beneficio máximo es 10  
Los elementos que se deben coger son el número: 1, 2, 4  
RUN FINISHED; exit value 0; real time: 47s; user: 0ms; system: 0ms
```