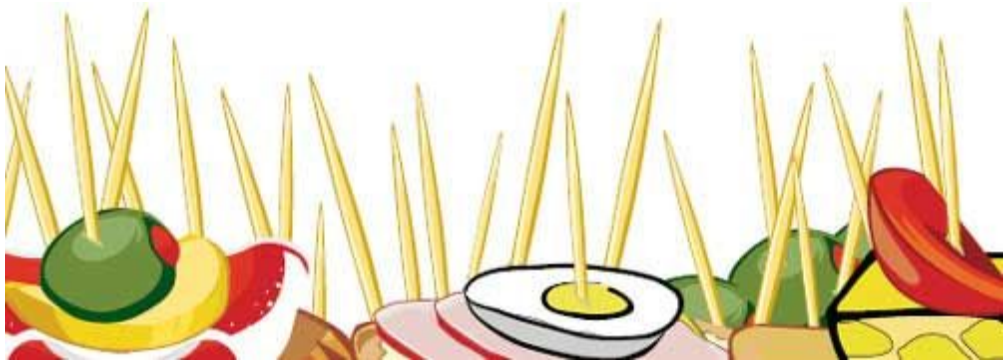
MEMORIA
Práctica 3: Algoritmos Voraces (Greedy)

Problema “Bar de tapas”



REALIZADO POR:

Baeza Álvarez, Jesús
García Moreno, Jorge
López Maldonado, David
Rodriguez Calvo, Jose Manuel
Sánchez Molina, Alejandro

Grupo A1

ÍNDICE

1. Introducción

El objetivo de esta práctica es diseñar un algoritmo Greedy que resuelva de forma óptima el problema del “bar de tapas” y demostrar que el algoritmo diseñado encuentra la solución óptima.

Como bien sabemos, un algoritmo Greedy, buscan siempre la mejor opción en cada momento con la esperanza de llegar a una solución general óptima.

2. Explicación del problema

El enunciado del problema a resolver es:

“Un restaurante de tapas ofrece un amplio surtido de n tapas, todas al mismo precio. De cada tapa ti se conoce el número de calorías (ci) que contiene. Se desea elegir como menú un subconjunto de tapas (sin repetir ninguna) que garantice un consumo de un mínimo de M calorías pagando el menor precio posible. Diseñad un algoritmo voraz para resolver el problema y demostrar su optimalidad.”

Para resolverlo y que sea óptimo, debemos coger las tapas que mayor calorías contengan para satisfacer así las M calorías solicitadas por el cliente y minimizar así el precio.

En los siguientes apartados, se explican los pasos seguidos para la implementación del algoritmo Greedy así como la demostración de la optimalidad del mismo.

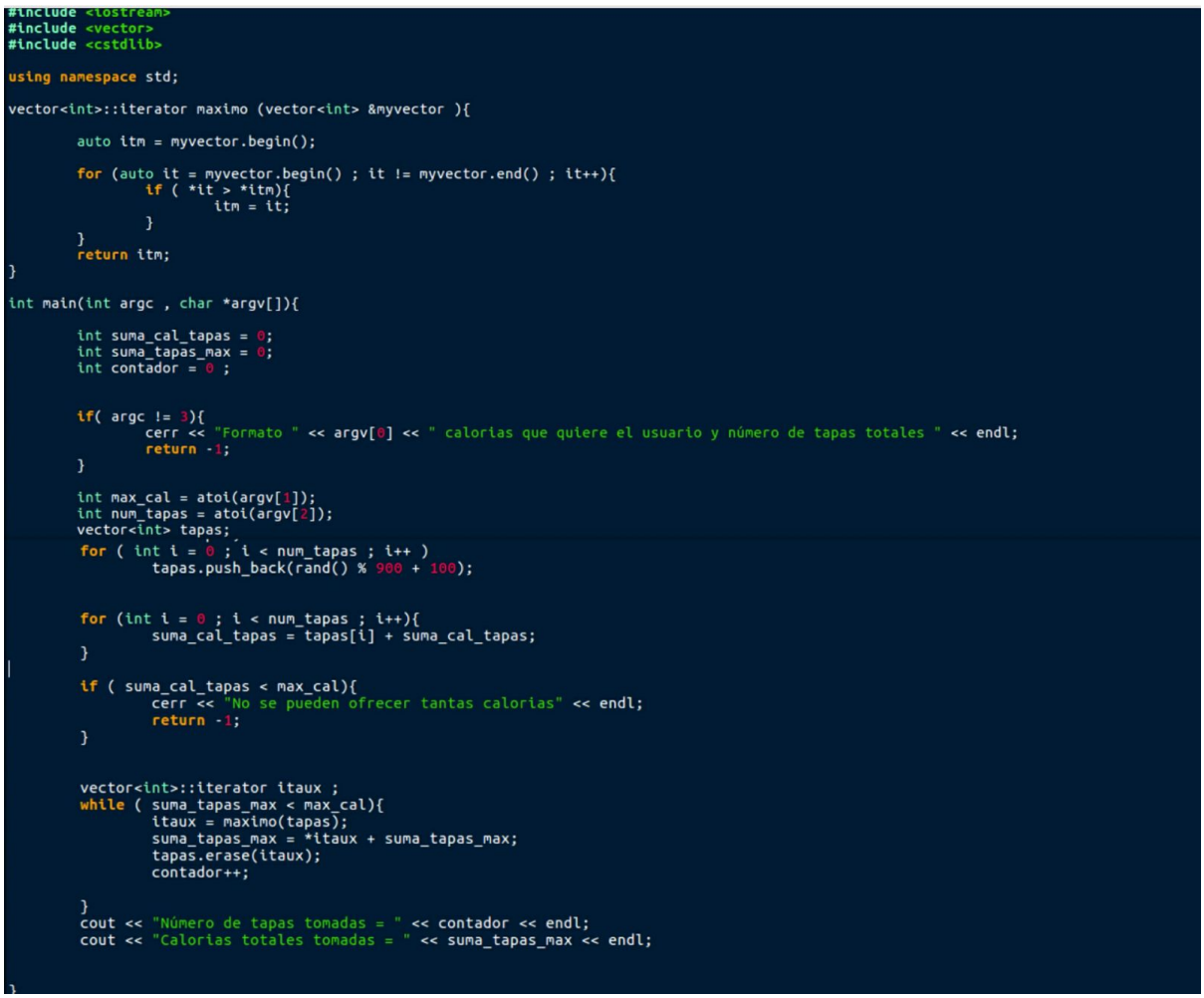
3. Explicación del algoritmo voraz

Hemos desarrollado un algoritmo que recoge dos valores, los cuales inicialmente son introducidos al ejecutarlo,dichos valores son: el número de tapas “n” que tendrá nuestro restaurante y el número de calorías que desea tomar el consumidor.

Para desarrollarlo hemos creado un vector de un tamaño igual al introducido como primer valor e inicializamos el número de calorías de cada tapa a un valor random.

Después, vamos llamando a un método que nos busca y elige la tapa con el mayor número de calorías, lo elimina del vector inicial para expresar que dicha tapa ya ha sido elegida y lo suma al resultado de calorías proporcionadas por las tapas.

Si el número de calorías proporcionado por la tapa no es mayor o igual al número deseado por el consumidor, nos vuelve a elegir la tapa mayor y la vuelve a sumar, hasta alcanzarlo o superarlo.



```
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

vector<int>::iterator maximo (vector<int> &myvector ){

        auto itm = myvector.begin();

        for (auto it = myvector.begin() ; it != myvector.end() ; it++){
                if ( *it > *itm){
                        itm = it;
                }
        }
        return itm;
}

int main(int argc , char *argv[]){

        int suma_cal_tapas = 0;
        int suma_tapas_max = 0;
        int contador = 0 ;


        if( argc != 3){
                cerr << "Formato " << argv[0] << " calorias que quiere el usuario y número de tapas totales " << endl;
                return -1;
        }

        int max_cal = atoi(argv[1]);
        int num_tapas = atoi(argv[2]);
        vector<int> tapas;
        for ( int i = 0 ; i < num_tapas ; i++ )
                tapas.push_back(rand() % 900 + 100);


        for (int i = 0 ; i < num_tapas ; i++){
                suma_cal_tapas = tapas[i] + suma_cal_tapas;
        }

        if ( suma_cal_tapas < max_cal){
                cerr << "No se pueden ofrecer tantas calorias" << endl;
                return -1;
        }


        vector<int>::iterator itaux ;
        while ( suma_tapas_max < max_cal){
                itaux = maximo(tapas);
                suma_tapas_max = *itaux + suma_tapas_max;
                tapas.erase(itaux);
                contador++;

        }
        cout << "Número de tapas tomadas = " << contador << endl;
        cout << "Calorias totales tomadas = " << suma_tapas_max << endl;


}
```

4. Demostración de la optimalidad

Vamos a denominar T al conjunto de tapas a seleccionar, cada una con sus calorías.
Ahora tenemos A={a1,a2,a3,a4}, que es un conjunto óptimo seleccionado por nuestro algoritmo.

Usando reducción al absurdo vamos a suponer que hay una solución mejor, esto supone elegir menos tapas, por tanto quitamos de la solución a3 y a4 y se queda una solución que no cumple el mínimo de calorías exigido.

Ahora suponemos que hay una tapa b1 que pertenece a T, tal que A={a1,a2,b1} y esto si satisface el mínimo de calorías exigido.

Esto último supondría que b1 > máx(a3,a4) y por tanto no tiene sentido ya que si b1 existiera, nuestro algoritmo lo habría cogido, ya que tiene más prioridad que a3 y a4, llegando a la conclusión de que es absurdo y deduciendo así que nuestro algoritmo es óptimo.