Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez

En esta práctica vamos a analizar el uso de los algoritmos "voraces" o "greedy", algoritmos que seleccionan en cada momento lo mejor de entre un conjunto de candidatos, sin tener en cuenta lo ya hecho, para obtener una solución "rápida" al problema.

Vamos a tener dos problemas a los cuales vamos a aplicar esta manera de resolverlos y mediremos su eficiencia teórica.

Una vez diseñado el algoritmo, veremos los resultados de la ejecución y los compararemos con los resultados "óptimos", generados tras resolver el problema de la menor manera posible.

Recordemos que los algoritmos greedy no aseguran generar soluciones optimales siempre; esta desventaja es una ventaja en problemas en los que es muy difícil alcanzar la solución óptima, apliquemos el algoritmo que apliquemos, como el problema que se propone a continuación. No obstante, veremos que los resultados, a pesar de no ser los óptimos, son bastante eficientes, así como el tiempo de ejecución del algoritmo.

1. Problema común (Viajante de comercio)

Como hemos comentado anteriormente, aplicar un algoritmo que nos dé el resultado más óptimo para este problema es bastante complicado y su tiempo de ejecución se incrementaría bastante.

Es por eso por lo que el enfoque Greedy es una manera eficiente de solucionar este problema, generando un resultado que no es el óptimo pero se acerca a ello.

El problema se resume en encontrar un circuito hamiltoniano para una serie de puntos, en este caso ciudades, de manera que se recorran todas ellas sin volver a pasar por ninguna, de manera que la distancia total entre estas ciudades, es decir, del circuito, sea la mínima (y así minimizamos el recorrido).

1.1. Algoritmo basado en cercanía

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <vector>
5  #include <utility>
6  #include <cmath>
7  #include <limits>
8  #include "matriz.h"
9
10 using namespace std;
11 const double INF = numeric_limits<double>::max();
12
13
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
14 /*  FUNCION
15 *  Esta funcion recibe como parametros un vector, que son las distintas distancias
16 * a las distintas ciudades desde una misma ciudad y el vector candidatos para comprobar
17 * si ya se han visitado las respectivas ciudades.
18 *
19 * La funcion encuentra el menor elemento del vector v y devuelve la posicion dentro
20 * del vector de dicho elemento.
21 */
22 int BuscaMenor(vector<double> v, vector<int> & candidatos){
23     int menor;
24     double minimo = INF;
25     for(int i = 0;i< v.size(); ++i){
26         if( v[i] < minimo && candidatos[i] !=-1){
27             minimo = v[i];
28             menor = i;
29         }
30     }
31     return(menor);
32 }
33
34
35 int main(int argc, char **argv){
36
37     ifstream input_file(argv[1]);
38     string line;
39     int dimension;
40
41     int ciudad;
42     double coor_x, coor_y;
43
44     vector<pair<double, double> > v;
45     pair<double, double> p;
46
47     //Movemos el offset a la linea en la que se dice la dimension del grafo
48     for(int i=0; i<3; ++i){
49         getline(input_file, line);
50     }
51
52     getline(input_file, line);
53     line.erase(0, 11);
54     dimension = atoi(line.c_str());
55
56     //Reservamos espacio para el grafo
57     matriz <double> m(dimension, dimension, INF);
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
58
59      //Movemos el offset al comienzo de los datos
60      for(int i=0; i<3; ++i){
61          getline(input_file, line);
62      }
63
64      //Tomamos los datos
65      for (int i=0; i<dimension; ++i){
66          input_file >> ciudad;
67          input_file >> coor_x;
68          input_file >> coor_y;
69          p.first = coor_x;
70          p.second = coor_y;
71          v.push_back(p);
72      }
73
74      //Calculamos las distancias y las metemos en la matriz
           ↪ que representa
75      //el grafo
76      for(int i=0; i<dimension; ++i){
77          for (int j=i+1; j<dimension; ++j){
78              m[i][j]=sqrt(pow(v[j].first - v[i].first, 2) +
                   ↪ pow(v[j].second - v[i].second, 2));
79              m[j][i]=m[i][j];
80          }
81      }
82
83      //Algoritmo basado en cercanias. La idea es ir a la
           ↪ciudad mas cercana
84      //sin formar un ciclo. Exceptuando la ultima.
85
86      //Declaramos los vectores que albergaran los conjuntos
           ↪ Candidato y Solucion
87      vector<int> solucion;
88      vector<int> candidatos;
89
90      // Inicializamos el conjunto de candidatos, el rango
           ↪sera [0,15].
91      for(int i = 0; i<dimension;++i){
92          candidatos.push_back(i);
93      }
94
95      // Abergamos la primera ciudad en el conjunto solucion
           ↪.
96      int i = 0;
97      solucion.push_back(0);
98      candidatos[0] = -1;
99
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
100     // variable donde guardaremos el indice, es decir, la
           ↪ciudad a donde nos dirigimos.
101     int menor;
102
103     /*CUERPO DEL ALGORITMO:
104     * La idea es encontrar la ciudad mas cercana haciendo
           ↪uso de la matriz de
105     * distancias. Una vez encontrada la ciudad (indice) al
           ↪ que nos dirigimos,
106     * la posicion candidatos[indice] lo hacemos -1 para
           ↪mostrar que esa ciudad
107     * ya la hemos visitado y introducimos el indice en el
           ↪vector de soluciones.
108     *
109     * Para concluir, asignamos el valor del indice a la
           ↪variable i para empezar
110     * de nuevo todo el proceso
111     */
112     while(solucion.size()< dimension   ){
113         vector <double> c;
114         m.get_Fila(i,c);
115         menor =BuscaMenor(c, candidatos);
116         solucion.push_back(menor);
117         candidatos[menor] = -1;
118         i = menor;
119     }
120
121     // Imprimimos el vector solucion teniendo en cuenta
           ↪que para la implementacion
122     // La ciudad numero 1 ha sido el indice numero 0, por
           ↪lo tanto tenemos que sumar
123     // 1 a los valores del vector solucion.
124
125     for(int i = 0; i< solucion.size();i++){
126         cout << solucion [i]  + 1<< " --> ";
127     }
128     // Aniadimos la ciudad inicial para indicar que
           ↪completamos un ciclo.
129     cout << " 1 " << " FIN.";
130
131
132     // Volcamos la salida a un fichero para su
           ↪visualizacion con la ayuda de
133     // GNUPLOT.
134
135     ofstream ficherosalida("data/ulysses_camino.txt");
136     for (int i = 0; i < dimension; ++i){
137         int c = solucion[i];
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
138             ficherosalida << c+1 << " " << v[c].first << " "
                    ↪<< v[c].second << endl;
139         }
140
141
142         //FIN.
143         return (0);
144     }
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez

1.2. Algoritmo basado en inserción

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <utility>
#include <cmath>
#include <limits>
#include "matriz.h"

using namespace std;


/**
Este programa busca un ciclo que recorra todas las ciudades de un mapa mediante un algoritmo de tipo greedy. En esta version del algoritmo, dado un recorrido inicial que contiene tres nodos en cada paso se busca el nodo mas cercano al conjunto solucion que se encuentre en el conjunto de candidatos. El nodo mas cercano es insertado en la posicion del vector que menos aumente el recorrido.
*/
int main(int argc, char **argv){
    const double INF = numeric_limits<double>::max();

    if(argc < 2){
        cout << "ERROR. Faltan argumentos [archivo de datos]" << endl;
        exit(-1);
    }

    ifstream input_file(argv[1]);
    string line;
    int num_ciudades;

    int ciudad;
    double coor_x, coor_y;

    vector<pair<double, double> > v_coordenadas;
    pair<double, double> coordenadas;

    vector<int> candidatos;
    vector<int> solucion;

    //Movemos el offset a la linea en la que se dice la dimension del grafo
    for(int i=0; i<3; ++i){
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
39          getline(input_file, line);
40      }
41
42      getline(input_file, line);
43      line.erase(0, 11);
44      num_ciudades = atoi(line.c_str());
45
46      //Reservamos espacio para el grafo
47      matriz <double> distancias(num_ciudades, num_ciudades,
           ↪ 0);
48
49      //Movemos el offset al comienzo de los datos
50      for(int i=0; i<3; ++i){
51          getline(input_file, line);
52      }
53
54      //Tomamos los datos
55      for (int i=0; i<num_ciudades; ++i){
56          input_file >> ciudad;
57          input_file >> coor_x;
58          input_file >> coor_y;
59          coordenadas.first = coor_x;
60          coordenadas.second = coor_y;
61          v_coordenadas.push_back(coordenadas);
62      }
63
64      //Calculamos las distancias y las metemos en la matriz
           ↪ que representa
65      //el grafo
66      for(int i=0; i<num_ciudades; ++i){
67          for (int j=i+1; j<num_ciudades; ++j){
68              distancias[i][j]=sqrt(pow(v_coordenadas[j].
                   ↪first - v_coordenadas[i].first, 2) +
69              pow(v_coordenadas[j].second - v_coordenadas[i
                   ↪].second, 2));
70              distancias[j][i]=distancias[i][j];
71          }
72      }
73
74      distancias.draw();
75
76      //Generamos vector de candidatos
77      for(int i=0; i<num_ciudades; ++i)
78      candidatos.push_back(i);
79
80      //Elegimos el recorrido inicial
81      int E = 0, O = 0, N = 0;
82      double mas_al_E = v_coordenadas[0].first;
83      double mas_al_O = v_coordenadas[0].first;
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
84      double mas_al_N = v_coordenadas[0].second;
85
86      for(int i=1; i<num_ciudades; ++i){
87          if(v_coordenadas[i].second > mas_al_N){
88              mas_al_N = v_coordenadas[i].second;
89              N = i;
90          }
91          if(v_coordenadas[i].first > mas_al_E){
92              mas_al_E = v_coordenadas[i].first;
93              E = i;
94          }
95          if(v_coordenadas[i].first < mas_al_O){
96              mas_al_O = v_coordenadas[i].first;
97              O = i;
98          }
99      }
100
101     solucion.push_back(O); candidatos[O] = -1;
102     solucion.push_back(N); candidatos[N] = -1;
103     solucion.push_back(E); candidatos[E] = -1;
104
105     int tam_solucion = solucion.size(); //Tamanio del
        ↪conjunto solucion
106
107     //Comienzo del algoritmo
108     vector<int>::iterator sol_it, cand_it;  //Iteradores
        ↪de los vectores de candidatos y solucion
109     vector<int>::iterator ciudad_origen_it; //Iterador que
        ↪ almacenara la posicion de la ciudad del
110     //conjunto solucion, que tiene mas cerca a una ciudad
111     //del conjuno candidatos
112
113     while(tam_solucion < num_ciudades){ //Mientras que no
        ↪hayamos recorrido todas las ciudades
114         //Buscamos la ciudad mas cercana al conjunto
            ↪solucion
115         int ciudad_mas_cercana = 0;
116         double distancia_mas_cercana = INF;
117
118         for(sol_it=solucion.begin(); sol_it!=solucion.end
            ↪(); ++sol_it){
119             for (cand_it=candidatos.begin(); cand_it!=
                ↪candidatos.end(); ++cand_it){
120                 if ((distancias[*sol_it][*cand_it] <
                    ↪distancia_mas_cercana) && (*cand_it
                    ↪!= -1)){
121                     ciudad_origen_it = sol_it;
122                     ciudad_mas_cercana = *cand_it;
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
123                     distancia_mas_cercana = distancias[*
                           ↪sol_it][*cand_it];
124                 }
125             }
126         }
127
128         //Una vez encontrada vemos en que posicion del
               ↪conjunto solucion insertarla para minimizar
               ↪el trayecto
129         vector<int>::iterator ciudad_siguiente_it =
               ↪ciudad_origen_it;
130         vector<int>::iterator ciudad_anterior_it =
               ↪ciudad_origen_it;
131         vector<int>::iterator final_it = solucion.end();
132         final_it--;
133
134         //Puesto que el recorrido es un ciclo (cerrado)
               ↪hay que contemplar el caso de que la
135         //ciudad a insertar sea adyacente al primer o
               ↪ultimo elemento del conjunto solucion
136         if(ciudad_origen_it == solucion.begin()){
137             ++ciudad_siguiente_it;
138             ciudad_anterior_it = final_it;
139         }
140         else if(ciudad_origen_it == final_it){
141             ciudad_siguiente_it = solucion.begin();
142             --ciudad_anterior_it;
143         }
144         else{
145             ++ciudad_siguiente_it;
146             --ciudad_anterior_it;
147         }
148
149
150         if(distancias[ciudad_mas_cercana][*
               ↪ciudad_anterior_it] < distancias[
               ↪ciudad_mas_cercana][*ciudad_siguiente_it]){
151             solucion.insert(ciudad_origen_it,
                   ↪ciudad_mas_cercana);
152         }
153         else{
154             solucion.insert(ciudad_siguiente_it,
                   ↪ciudad_mas_cercana);
155         }
156
157         candidatos[ciudad_mas_cercana] = -1;
158         tam_solucion++;
159     }
160
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez


```
161     //Insertamos de nuevo el primer elemento del conjunto
            ↪solucion
162     // ya que es un caamino cerrado
163     solucion.push_back(*solucion.begin());
164
165     //Mostramos la solucion
166     cout << "Solucion: " << endl;
167
168     for(int i=0; i<tam_solucion; ++i){
169         cout << solucion[i]+1 << " ";
170     }
171     cout << endl;
172
173     //Calculo de la distancia total recorrida
174     double distancia_recorrida = 0.00;
175     for(int i=0; i<tam_solucion-1; ++i){
176         distancia_recorrida += distancias[i][i+1];
177     }
178
179     cout << "Distancia total recorrida: " <<
            ↪distancia_recorrida << endl;
180
181     //Salida de la solucion a fichero
182     ofstream output_file("data/ulysses16_insercion.txt");
183     for(int i=0; i<num_ciudades; ++i){
184         int c = solucion[i];
185         output_file << c+1 << " " << v_coordenadas[c].
                ↪first << " " << v_coordenadas[c].second <<
                ↪endl;
186     }
187
188     return (0);
189 }
```

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez

1.3. Algoritmo con otra estrategia

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez

1.4. Comparación de algoritmos

Adrián Carmona Lupiáñez
Ignacio Sánchez Herrera
Jacobo Casado de Gracia
Jesús José Mª Maldonado Arroyo
Juan Miguel Hernández Gómez

2. Problema específico - Ahorro de gasolina

El problema trata de partir de una ciudad y llegar a otra con un vehículo con cierta autonomía pasando por el menor número de gasolineras posibles.

Para entender el algoritmo lo podemos imaginar gráficamente. La autonomía del coche va a ser el radio de la circunferencia de centro la primera ciudad o gasolinera en donde nos encontremos.

Dentro de esa circunferencia se encontrarán las gasolineras a las que podemos llegar con la autonomía del vehículo. Solo nos queda elegir a cual de ellas. Muy facil, nos vamos a la gasolinera que este más cerca de la ciudad objetivo.

Así nos vamos moviendo de gasolinera en gasolinera hasta que dentro de nuestra circunferencia se encuentre a la ciudad objetivo.