Código de C++ apliciones

Joaquin Velarde

Abril 2022

1 Lectura de instancias

Esta sección tiene los objetos ciudad e información de las instancias que vamos a leer.

Listing 1: Instancia.h

```
#pragma once
#include <vector>
#include <string>
using namespace std;

class Ciudad
{
  public:
    int Identificador;
    int X_i;
    int Y_i;
    Ciudad(int P_Identificador, int P_X_i, int P_Y_i)
    {
      Identificador = P_Identificador;
      X_i = P_X_i;
      Y_i = P_Y_i;
    }
};
class Instancia_Info
{
  public:
    string Name;
    string Comment;
    string Type;
    string Dimension;
    string Edge_Type;
    vector<Ciudad> Nodos;
    Instancia_Info(string P_Name, string P_Comment, string P_Type, string P_Dimension,
    string P_Edge_Type, vector<Ciudad> PNodos)
    {
       Name   = P_Name;
       Comment    = P_Comment;
       Type   = P_Type;
       Dimension  = P_Dimension;
       Edge_Type  = P_Edge_Type;
       Nodos    = PNodos;
    }
};
```

En Listing 1, podemos ver que creamos 2 objetos tanto como ciudad e Instancia Info, la clase ciudad es la que tendrá la información individual de cada nodo en la instancia, como el identificador, y las coordenadas X y Y.

Listing 2: Setup.cpp

```
#include "Instancia_Info.h"
#include <fstream>
#include <sstream>
#include <iostream>
using namespace std;

```

```
7     string LTrim(const string& data)
8     {
9         string aux(data);
10        for (int i = 0; (aux.length() > 0) && !iswgraph(aux.substr(i, 1).operator [](0)); )
11            aux.erase(i, 1);
12        size_t doubleSpace = aux.find("  ");
13        while (doubleSpace != string::npos)
14        {
15            aux.erase(doubleSpace, 1);
16            doubleSpace = aux.find("  ");
17        }
18        return aux;
19    }
20
21    int main()
22    {
23        Instancia_Info Instancia = LecturaTSP("Instancias/a280.txt");
24    }
```

En listing 2 tenemos las librerias que necesitamos incluir, como la cabecera "Instancia.h", el método LTrim espara eliminar espacios dobles o cualquier otro problema con la lectura de la instancia.

Listing 3: Main.cpp

```
1
2     // LecturaTSP = Mi objeto de datos "Instancia_Info" de la instancia dado que mi nombre
      de archivo es "Parchivo".
3  Instancia_Info LecturaTSP(string PArchivo)
4  {
5     string Name;
6     string Comment;
7     string Type;
8     string Dimension;
9     string Edge_Type;
10    vector<Ciudad> Nodos;
11
12    string      line;
13    ifstream    Archivo(PArchivo);
14    char        split_double  = ':';
15    char        split_space   = ' ';
16    bool      Datos       = false;
17
18    if (Archivo.is_open())
19    {
20        while (getline(Archivo, line) && line != "")
21        {
22            string token;
23            vector<string>  VectorLinea;
24            stringstream    ss(line);
25            if (line.find("NAME")       != string::npos)
26            {
27                while (getline(ss, token, split_double))
28                    VectorLinea.push_back(token);
29                Name = VectorLinea[1];
30            }
31            else if (line.find("EDGE_WEIGHT_TYPE") != string::npos)
32            {
33                while (getline(ss, token, split_double))
34                    VectorLinea.push_back(token);
35                Edge_Type = VectorLinea[1];
36            }
37            else if (line.find("COMMENT") != string::npos)
38            {
39                while (getline(ss, token, split_double))
40                    VectorLinea.push_back(token);
41                Comment = VectorLinea[1];
42            }
43            else if (line.find("TYPE") != string::npos)
44            {
45                while (getline(ss, token, split_double))
46                    VectorLinea.push_back(token);
```

```
47                 Type = VectorLinea[1];
48             }
49             else if (line.find("DIMENSION")  != string::npos)
50             {
51                 while (getline(ss, token, split_double))
52                     VectorLinea.push_back(token);
53                 Dimension = VectorLinea[1];
54             }
55 
56             else if (line.find("NODE_COORD_SECTION")  != string::npos)
57                 continue;
58             else if (line.find("EOF") != string::npos)
59                 break;
60             else
61             {
62                 vector<string> Coordenadas;
63                 stringstream ss(LTrim(line));
64                 while (getline(ss, token, split_space))
65                     Coor.push_back(token);
66                 Ciudad Nodo  = Ciudad(stoi(Coor[0]), stoi(Coor[1]), stoi(Coor[2]));
67                 Nodos.push_back(Nodo);
68             }
69             }
70         }
71         Instancia_Info Info = Instancia_Info(Name,
72                                              Comment,
73                                              Type,
74                                              Dimension,
75                                              Edge_Type,
76                                              Nodos);
77         return Info;
78     }
```

Listing 3 es el más pesado y complejo y consiste de la función LecturaTSP, la cual lee un archivo con su nombre en el parámetro PArchivo y los guarda en las clases creadas de Listing 1.

2 Matriz de distancias

En esta sección empezaremos con el calculo de nuestra matriz de distancia en nuestra instancia.

Listing 4: MetodosAuxiliares.h

```
1 #pragma once
2 #include "Instancia_Info.h"
3 
4 
5 float  EuclideanDistance(Ciudad Node_1, Ciudad Node_2)
6 {
7     float x = Node_1.X_i - Node_2.X_i;
8     float y = Node_1.Y_i - Node_2.Y_i;
9     float dist;
10    dist = pow(x, 2) + pow(y, 2);
11    dist = sqrt(dist);
12    return dist;
13 }
14 
15 vector<vector<float>> Matrix(Instancia_Info Instancia)
16 {
17     vector<vector<float>> M;
18     for (int i = 0; i < Instancia.Nodos.size(); i++)
19     {
20         vector<float> D;
21         for (int j = 0; j < Instancia.Nodos.size(); j++)
22         {
23             if (i == j)
24                 D.push_back(0);
25             else
26                 D.push_back(EuclideanDistance(Instancia.Nodos[i], Instancia.Nodos[j]));
27         }
28         M.push_back(D);
```

```
    }
    return M;
}
```

En listing 4 podemos ver dos métodos de interés el cual es la distancia euclidiana y nuestro método que calcula la distancia.

La distancia euclidiana es calculada de la siguiente manera

$$Distancia(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (x_1 - x_2)^2}$$

nuestro algorimo 1, contiene además el pseudo-algoritmo para el calculo de la matriz de distancia.

Algorithm 1: Generador de matriz de distancias

```
M = []
for each arc nodo i ∈ G do
    D = []
    for each arc nodo i ∈ G do
        if i == j then
            D[i] = 0;
        else
            D[i] = EuclideanDistance(i,j);
        end
    end
    M.add(D);
end
return M;
```

Por lo tanto nuestro main se ve de esta manera (Listing 5).

Listing 5: Main

```
int main()
{
    Instancia_Info Instancia = LecturaTSP("Instancias/a280.txt");

    vector<vector<float>> MatrizDistancias = Matrix(Instancia);
}
```

3 Fase de inicialización

En esta fase iniciamos algunos parámetros como por ejemplo la matriz de feromonas.

3.1 Matriz de feromonas

Estas son las feromonas puestas para cada arco al inicio del programa, utilizaremos varios valores, en nuestra variable τ_0, la primera sera la más simple $\tau_0 = 1$.

En Listing 6 muestro la adición del algoritmo que inicializa la matriz de feromonas.

Listing 6: MetodosAuxiliares.h

```


vector<vector<float>> MFeromonas(Instancia_Info Instancia)
{
    vector<vector<float>> M;
    for (int i = 0; i < Instancia.Nodos.size(); i++)
    {
        vector<float> D;
        for (int j = 0; j < Instancia.Nodos.size(); j++)
        {
            if (i == j)
                D.push_back(0);
            else
                D.push_back(1);
```

```
        }
        M.push_back(D);
    }
    return M;
}
```

Por lo tanto nuestro main se ve de esta manera (Listing 7).

Listing 7: Main

```
int main()
{
    Instancia_Info Instancia = LecturaTSP("Instancias/a280.txt");

    vector<vector<float>> MatrizDistancias = Matrix(Instancia);

    //Fase de inicializacin

    vector<vector<float>> MatrizFeromonas = MFeromonas(Instancia);
}
```

3.2 Inicialización de hormigas

En nuestro algoritmo las hormigas son objetos con diversos atributos o variables que utilizaremos para nuestro algoritmo. En listing 8 se declara el objeto hormiga así como sus atributos y su constructor.

Listing 8: Instancia.h

```


class Hormiga
{
public:
  int       Identificador;
  Ciudad      Inicial;
  Ciudad      Actual;
  vector<Ciudad>  Solucion;
  vector<Ciudad>  Visitar;
  float     L;
  float     Delta_L;

  Hormiga(int P_Identificador, Ciudad PInicial, Ciudad PActual, vector<Ciudad> PSolucion,
    vector<Ciudad> PVisitar)
  {
    Identificador = P_Identificador;
    Inicial = PInicial;
    Actual = PActual;
    Solucion = PSolucion;
    Visitar = PVisitar;
    L = 0;
    Delta_L = 0;
  }
};
```

También se agrego un método que nos ayudara a obtener un número aleatorio de una un rango proporcionado en listing 9 se muestra este método.

Listing 9: MetodosAuxiliares.h

```


int NumeroAleatorio(int max)
{
    srand(time(NULL));
    return rand() % max;
}
```

Después inicializo a todas las hormigas haciendo un método auxiliar llamado IniciarHormigas, en listing 10 se inicializan.

Listing 10: MetodosAuxiliares.h

```
1
2  vector<Hormiga> IniciarHormigas(int M, Instancia_Info Instancia)
3  {
4      vector<Hormiga> Hormigas;
5      for (int k = 0; k < M; k++)
6      {
7          vector<Ciudad>   Solucion;
8          Ciudad Inicial = Instancia.Nodos[NumeroAleatorio(Instancia.Nodos.size())];
9          Solucion.push_back(Inicial);
10         vector<Ciudad>  Visitar;
11         for (int i = 0; i < Instancia.Nodos.size(); i++)
12         {
13             if (Instancia.Nodos[i].Identificador != Inicial.Identificador)
14                 Visitar.push_back(Instancia.Nodos[i]);
15         }
16         Hormiga H = Hormiga(k, Inicial, Solucion, Visitar);
17         Hormigas.push_back(H);
18     }
19     return Hormigas;
20 }
```

Ahora ahora nuestro main se ve así (Listing 11). Por ultimo podemos agregar inicializar las dos variables que controlaran la información heurística y el grado de feromonas.

Listing 11: MetodosAuxiliares.h

```
1  int main()
2  {
3      Instancia_Info Instancia = LecturaTSP("Instancias/a280.txt");
4
5      vector<vector<float>> MatrizDistancias = Matrix(Instancia);
6
7      //Fase de inicializacin
8
9      vector<vector<float>> MatrizFeromonas = MFeromonas(Instancia);
10
11     int M = <Numero Hormigas>;
12     vector<Hormiga> Hormigas = IniciarHormigas(M, Instancia);
13
14     int Beta = <valor>;
15     int Alfa = <valor>;
16 }
```

4 Fase de construcción

En esta fase empezaremos a construir nuestra solución para cada hormiga

4.1 Métodos de apoyo

En listing 12, escribimos la estructura de nuestro código para poder construir la solución de cada hormiga.

Listing 12: Main.cpp

```
1  int main()
2  {
3      //Fase de construccin
4      int n = Instancia.Nodos.size();
5      for (int i = 0; i < n; i++)
6      {
7          if (i < (n - 1))
8          {
9              for (int k = 0; k < M; k++)
10             {
11                 Hormigas[k] = EleccionPseudoAleatoria(Hormigas[k], Instancia, Alfa, Beta,
       MatrizFeromonas, MatrizDistancias);
12             }
13         }
```

```
        else
        {
            for (int k = 0; k < M; k++)
            {
                Hormigas[k].Actual = Hormigas[k].Inicial;
                Hormigas[k].Solucion.push_back(Hormigas[k].Inicial);
                Hormigas[k].L = Distancia(Hormigas[k].Solucion, MatrizDistancias);
                Hormigas[k].Delta_L = 1 / Hormigas[k].L;
            }
        }
    }
}
```

A continuación crearemos un clase que nos ayudara a seleccionar al siguiente destino de cada hormiga, este tendrá un identificador y la probabilidad de ser escogido. (listing 13)

Listing 13: Instancia.h

```
class ProCiudad
{
public:
  int   Identificador;
  float P_i;
  ProCiudad(int P_Identificador, float P)
  {
    Identificador = P_Identificador;
    P_i = P;
  }
};
```

Para poder seleccionar a nuestro siguiente vecino necesitamos 3 métodos de apoyo que nos ayudaran a lograrlo, estos son, NumeroAleatorio(), SumatoriaCumulativa(), SelectionSort() y RuletaRusa(). En listing 14 vemos a estos tres métodos.

Listing 14: MetodosAuxiliares.h

```
int NumeroAleatorio(int max)
{
    srand(time(NULL));
    return rand() % max;
}

vector<float> SCumulativa(vector<ProCiudad> OrderP_i)
{
    vector<float> SumaCumulativa;
    for (int i = 0; i < OrderP_i.size(); i++)
    {

        float CumulativoLocal = 0;
        if (SumaCumulativa.size() == 0)
            SumaCumulativa.push_back(OrderP_i[0].P_i);
        else
        {
            for (int j = 0; j <= SumaCumulativa.size(); j++)
            {
                CumulativoLocal = CumulativoLocal + OrderP_i[j].P_i;
            }
            SumaCumulativa.push_back(CumulativoLocal);
        }
    }
    return SumaCumulativa;
}

vector<ProCiudad> SelectionSort(int n, vector<ProCiudad> P_i)
{
    vector<ProCiudad> OrderP_i;
    for (int j = 0; j < n; j++)
    {
        float Smallest = P_i[0].P_i;
        int   Smallest_i = 0;
        for (int i = 0; i < P_i.size(); i++)
```

```
        {
            if (P_i[i].P_i < Smallest)
            {
                Smallest = P_i[i].P_i;
                Smallest_i = i;
            }
        }
        OrderP_i.push_back(P_i[Smallest_i]);
        P_i.erase(P_i.begin() + Smallest_i);
    }
    return OrderP_i;
}

int RuletaRusa(vector<float> SumaCumulativa)
{
    int     Index = 0;
    float RandomNumber = (float)rand() / RAND_MAX;
    for (int i = 0; i < SumaCumulativa.size(); i++)
    {
        if (i == 0)
        {
            if (0 <= RandomNumber && RandomNumber <= SumaCumulativa[i])
                return Index = i;
        }
        else if (i == SumaCumulativa.size() - 1)
        {
            if (SumaCumulativa[i - 1] <= RandomNumber && RandomNumber <= 1)
                return Index = i;
        }
        else
        {
            if (SumaCumulativa[i - 1] < RandomNumber && RandomNumber <= SumaCumulativa[i])
                return Index = i;
        }
    }
}
```

4.2 Elección PseudoAleatoria

A cada hormiga iremos agregándole un nodo, esto lo haremos de acuerdo a la regla de Elección Pseudo Aleatoria, en lsting 15 declaramos el método.

Listing 15: MetodosAuxiliares.h

```
Hormiga EleccionPseudoAleatoria(Hormiga H, Instancia_Info Instancia, float Alfa, float Beta,
     vector<vector<float>> MF, vector<vector<float>> MD)
{
    vector<ProCiudad> P_i;
    float Sumatoria = 0;
    for (int j = 0; j < H.Visitar.size(); j++)
    {
        float Tau_ij = MF[H.Actual.Identificador - 1][H.Visitar[j].Identificador - 1];
        float Eta_ij = 1 / (MD[H.Actual.Identificador - 1][H.Visitar[j].Identificador - 1]);
        float P = pow(Tau_ij, Alfa) * pow(Eta_ij, Beta);
        Sumatoria = Sumatoria + P;
    }
    for (int j = 0; j < H.Visitar.size(); j++)
    {
        float Tau_ij = MF[H.Actual.Identificador - 1][H.Visitar[j].Identificador - 1];
        float Eta_ij = 1 / (MD[H.Actual.Identificador - 1][H.Visitar[j].Identificador - 1]);
        float P = (pow(Tau_ij, Alfa) * pow(Eta_ij, Beta)) / Sumatoria;
        ProCiudad PCiudad = ProCiudad(H.Visitar[j].Identificador, P);
        P_i.push_back(PCiudad);
    }

    int n = P_i.size();
    vector<ProCiudad>   OrderP_i = SelectionSort(n, P_i);

    vector<float>       SumaCumulativa = SCumulativa(OrderP_i);

```

```
26      int Index = RuletaRusa(SumaCumulativa);
27
28      H.Actual = Instancia.Nodos[OrderP_i[Index].Identificador - 1];
29      H.Solucion.push_back(H.Actual);
30      for (int i = 0; i < H.Visitar.size(); i++)
31      {
32          if (H.Actual.Identificador == H.Visitar[i].Identificador)
33              Index = i;
34      }
35      H.Visitar.erase(H.Visitar.begin() + Index);
36      H.L = Distancia(H.Solucion, MD);
37      H.Delta_L = 1 / H.L;
38
39      return H;
40  }
```

4.3 Evaporación local

Algorithm 2: RecibirAnt(Ant)

```
AjustarAtributos(Ant);
Enviar(Ant,MejorVecino(Ant));
```

Algorithm 3: AjustarAtributos(Ant)

```
if Nido(Ant) then
    if Retornando(Ant) then
        Inicializar(Ant);
    else
        AjustarBanderaRetorno(Ant);
        CalcularFeromona(Ant);
    end
end
ActualizarMejorRuta();
```

Algorithm 4: MejorVecino(Ant)

```
if Retornando(Ant) then
    DepositarFeromona();
    return UltimoNodoVisitado(Ant);
end
J = EleccionAleatoria();
ActualizarRuta(J);
EvaporacionLocal();
AñadirVisitados(J,Ant);
return J;
```