Código de Java aplicado a lectura de instancias

Joaquin Velarde

Abril 2022

1 Introducción

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

Listing 1: Instancia.java

```
1  import java.util.ArrayList;
2
3  public class Instancia
4  {
5    public String Name;
6    public String Comment;
7    public String Type;
8    public String Dimension;
9    public String Edge_Type;
10
11   ArrayList<Ciudad> Ciudades = new ArrayList<Ciudad>();
12 }
```

Listing 2: Ciudad.java

```
1
2  public class Ciudad
3  {
4    public Ciudad(int PIdentificador, int PX, int PY)
5    {
6      Identificador   = PIdentificador;
7      X         = PX;
8      Y         = PY;
9    }
10   public int Identificador;
11   public int X;
12   public int Y;
13 }
```

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

Listing 3: Principal.java

```
1  import java.util.ArrayList;
2  import java.io.File;  // Import the File class
3  import java.io.FileNotFoundException;  // Import this class to handle errors
4  import java.util.Scanner;
5
6  public class Principal
7  {
8
9    public static void main(String[] args)
10   {
11     Instancia TSP = LecturaTSP("a280.txt");
12     System.out.println("An error occurred.");
13   }
14
15   public static Instancia LecturaTSP(String PArchivo)
16   {
```

```
17      Instancia InfoInstancia   = new Instancia();
18        try
19        {
20          ArrayList<Ciudad> Ciudades  = new ArrayList<Ciudad>();
21            File Archivo      = new File(PArchivo);
22            Scanner myReader    = new Scanner(Archivo);
23            while (myReader.hasNextLine())
24            {
25              String Linea = myReader.nextLine();
26              if(Linea.contains("NAME"))
27              {
28                String[] Arreglo   = Linea.split(":", 2);
29                InfoInstancia.Name = Arreglo[1];
30              }
31              else if(Linea.contains("COMMENT"))
32              {
33                String[] Arreglo      = Linea.split(":", 2);
34                InfoInstancia.Comment = Arreglo[1];
35              }
36              else if(Linea.contains("TYPE"))
37              {
38                String[] Arreglo   = Linea.split(":", 2);
39                InfoInstancia.Type = Arreglo[1];
40              }
41              else if(Linea.contains("DIMENSION"))
42              {
43                String[] Arreglo        = Linea.split(":", 2);
44                InfoInstancia.Dimension = Arreglo[1];
45              }
46              else if(Linea.contains("EDGE_WEIGHT_TYPE"))
47              {
48                String[] Arreglo        = Linea.split(":", 2);
49                InfoInstancia.Edge_Type = Arreglo[1];
50              }
51              else if(Linea.contains("NODE_COORD_SECTION"))
52              {
53                continue;
54              }
55              else if(Linea.contains("EOF"))
56                break;
57              else
58              {
59                Linea = Linea.trim().replaceAll(" +", " ");
60                String[] Arreglo = Linea.split(" ", 3);
61                Ciudad Nodo = new Ciudad(
62                        Integer.parseInt(Arreglo[0]),
63                        Integer.parseInt(Arreglo[1]),
64                        Integer.parseInt(Arreglo[2]));
65                Ciudades.add(Nodo);
66              }
67            }
68
69            InfoInstancia.Ciudades = Ciudades;
70
71            myReader.close();
72         }
73         catch (FileNotFoundException e)
74         {
75            System.out.println("An error occurred.");
76            e.printStackTrace();
77         }
78        return InfoInstancia;
79    }
80
81  }
```

En Listing 3, tenemos nuestro método para leer la instancia, consiste de varios IF.

2 Matriz de distancias

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

Listing 4: MetodosAuxiliares.java

```
import java.lang.Math;
import java.util.ArrayList;

public class MetodosAuxiliares
{
  public static double EuclidianDistance(Ciudad Node_1, Ciudad Node_2)
  {
    float x = Node_1.X - Node_2.X;
    float y = Node_1.Y - Node_2.Y;
    double dist;
    dist = Math.pow( x, 2) + Math.pow( y, 2);
    dist = Math.sqrt(dist);
    return dist;
  }

  public static ArrayList<ArrayList<Float>> Matrix(Instancia Info)
  {
    ArrayList<ArrayList<Float>> M = new ArrayList<ArrayList<Float>>();
    for(int i = 0; i < Info.Ciudades.size(); i++)
    {
      ArrayList<Float> D = new ArrayList<Float>();;
      for(int j = 0; j < Info.Ciudades.size(); j++)
      {
        if(i == j)
          D.add((float) 0);
        else
          D.add((float) EuclidianDistance(Info.Ciudades.get(i), Info.Ciudades.get(j)));
      }
      M.add(D);
    }
    return M;
  }
}
```

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

La distancia ecleudiana 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 pseudoalgoritmo 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

```
  public static void main(String[] args)
```

```
2   {
3     Instancia TSP = LecturaTSP("a280.txt");
4
5     ArrayList<ArrayList<Float>> M = MetodosAuxiliares.Matrix(TSP);
6
7     System.out.println("An error occurred.");
8   }
```

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.java

```
1
2   public static ArrayList<ArrayList<Float>> MFeromonas(Instancia Info)
3   {
4     ArrayList<ArrayList<Float>> M = new ArrayList<ArrayList<Float>>();
5     for(int i = 0; i < Info.Ciudades.size(); i++)
6     {
7       ArrayList<Float> D = new ArrayList<Float>();;
8       for(int j = 0; j < Info.Ciudades.size(); j++)
9       {
10        if(i == j)
11          D.add((float) 0);
12        else
13          D.add((float) 1);
14      }
15      M.add(D);
16    }
17    return M;
18  }
```

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

Listing 7: Main

```
1   public static void main(String[] args)
2   {
3     Instancia TSP = LecturaTSP("a280.txt");
4
5     ArrayList<ArrayList<Float>> M = MetodosAuxiliares.Matrix(TSP);
6
7     //Fase de inicializaci n
8
9     ArrayList<ArrayList<Float>> MFeromonas = MetodosAuxiliares.MFeromonas(TSP);
10
11  }
```

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: Hormiga.java

```
1
2 public class Hormiga
3 {
```

```
4   public Hormiga(int PId, Ciudad PInicial, Ciudad PActual, ArrayList<Ciudad> PSolucion,
      ArrayList<Ciudad> PVisitar)
5   {
6     Identificador   = PId;
7     Inicial  =  PInicial;
8     Actual   =  PInicial;
9           Solucion =  PSolucion;
10    Visitar  =  PVisitar;
11    L  =  (float) 0;
12    Delta_L  =  (float) 0;
13  }
14  public int    Identificador;
15  public Ciudad   Inicial;
16  public Ciudad   Actual;
17  public ArrayList<Ciudad> Solucion;
18  public ArrayList<Ciudad> Visitar;
19  Float L;
20  Float Delta_L;
21 }
```

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

Listing 9: MetodosAuxiliares.java

```
1 public static int NumeroAleatorio(int Max)
2 {
3    return (int)(Math.random()*Max);
4 }
5
```

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

Listing 10: MetodosAuxiliares.java

```
1   public static ArrayList<Hormiga> IniciarHormigas(int M, Instancia Info)
2   {
3     ArrayList<Hormiga> Hormigas = new ArrayList<Hormiga>();
4     for(int i = 0; i < M; i++)
5     {
6       ArrayList<Ciudad> Solucion = new ArrayList<Ciudad>();
7       Ciudad Inicial = Info.Ciudades.get(NumeroAleatorio(Info.Ciudades.size()));
8       Solucion.add(Inicial);
9       ArrayList<Ciudad> Visitar = new ArrayList<Ciudad>();
10      for(int j = 0; j < Info.Ciudades.size(); j++)
11      {
12        if(Info.Ciudades.get(j).Identificador != Inicial.Identificador)
13          Visitar.add(Info.Ciudades.get(j));
14      }
15      Hormiga H = new Hormiga( i, Inicial, Inicial, Solucion, Visitar);
16      Hormigas.add(H);
17    }
18    return Hormigas;
19  }
```

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: Main

```
1  public static void main(String[] args)
2  {
3    Instancia TSP = LecturaTSP("a280.txt");
4
5    ArrayList<ArrayList<Float>> M = MetodosAuxiliares.Matrix(TSP);
6
7    //Fase de inicializaci n
8
9    ArrayList<ArrayList<Float>> MFeromonas = MetodosAuxiliares.MFeromonas(TSP);
10
11   int M1 = 10;
```

```
12
13   ArrayList<Hormiga> Hormigas = MetodosAuxiliares.IniciarHormigas(M1,TSP);
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: Principal.java

```
1     //Fase de construcci n
2     int n = TSP.Ciudades.size();
3     for(int i = 0; i < n; i++)
4     {
5       if(i < (n-1))
6       {
7         for(int k = 0; k < M1; k++)
8         {
9           Hormigas.set(k, MetodosAuxiliares.EleccionPseudoAleatoria(Hormigas.get(k), TSP,
      Alfa, Beta, M, MFeromonas));
10        }
11      }
12      else
13      {
14        for(int k = 0; k < M1; k++)
15        {
16          Hormigas.get(k).Actual = Hormigas.get(k).Inicial;
17          Hormigas.get(k).Solucion.add(Hormigas.get(k).Inicial);
18          Hormigas.get(k).L = MetodosAuxiliares.Distancia(Hormigas.get(k).Solucion, M);
19          Hormigas.get(k).Delta_L = 1/ Hormigas.get(k).L;
20        }
21      }
22    }
```

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: ProCiudad.java

```
1 public class ProCiudad
2 {
3   public ProCiudad(int PIdentificador, float P)
4   {
5     Identificador   = PIdentificador;
6     P_i     = P;
7   }
8   public int   Identificador;
9   public float P_i;
10 }
```

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

Listing 14: MetodosAuxiliares.java

```
1   public static ArrayList<Float> SCumulativa(ArrayList<ProCiudad> OrderP_i)
2   {
3     ArrayList<Float> SumaCumulativa = new ArrayList<Float>();
4     for(int i = 0; i < OrderP_i.size(); i++)
5     {
6       float CumulativoLocal = 0;
7       if(SumaCumulativa.size() == 0)
```

```
        SumaCumulativa.add(OrderP_i.get(i).P_i);
      else
      {
        for(int j = 0; j <= SumaCumulativa.size(); j++)
        {
          CumulativoLocal = CumulativoLocal + OrderP_i.get(j).P_i;
        }
        SumaCumulativa.add(CumulativoLocal);
      }
    }
    return SumaCumulativa;
  }

  public static int RuletaRusa(ArrayList<Float> SumaCumulativa)
  {
    int   Index = 0;
    Float   RandomNumber = (float) Math.random();
    for(int i = 0; i < SumaCumulativa.size(); i++)
    {
      if(i == 0)
      {
        if(0 <= RandomNumber && RandomNumber <= SumaCumulativa.get(i))
          return Index = i;
      }
      else if(i == SumaCumulativa.size() - 1)
      {
        if(SumaCumulativa.get(i-1) <= RandomNumber && RandomNumber <= 1)
          return Index = i;
      }
      else
      {
        if(SumaCumulativa.get(i-1) < RandomNumber && RandomNumber <= SumaCumulativa.get(i))
          return Index = i;
      }
    }
    return Index;
  }

  public static ArrayList<ProCiudad> SelectionSort(int n, ArrayList<ProCiudad> P_i)
  {
    ArrayList<ProCiudad> OrderP_i = new ArrayList<ProCiudad>();
    for(int j = 0; j < n; j++)
    {
      float Smallest    = P_i.get(0).P_i;
      int   Smallest_i  = 0;
      for(int i = 0; i < P_i.size(); i++)
      {
        if(P_i.get(i).P_i < Smallest)
        {
          Smallest   = P_i.get(i).P_i;
          Smallest_i = i;
        }
      }
      OrderP_i.add(P_i.get(Smallest_i));
      P_i.remove(Smallest_i);
    }
    return OrderP_i;
  }

  public static float Distancia(ArrayList<Ciudad> Solucion, ArrayList<ArrayList<Float>> MD)
  {
    float L = 0;
    for(int i = 0; i < Solucion.size() - 1; i++)
    {
      L = L + MD.get(Solucion.get(i).Identificador - 1).get(Solucion.get(i+1).Identificador
    - 1);
    }
    return L;
  }

```

4.2 Elección PseudoAleatoria

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

Listing 15: MetodosAuxiliares.java

```
1  public static Hormiga EleccionPseudoAleatoria(Hormiga H, Instancia Info, float Alfa, float
       Beta, ArrayList<ArrayList<Float>> MD, ArrayList<ArrayList<Float>> MF)
2  {
3    ArrayList<ProCiudad> P_i = new ArrayList<ProCiudad>();
4    float Sumatoria = 0;
5    for(int j = 0; j < H.Visitar.size(); j++)
6    {
7      float Tau_ij = MF.get(H.Actual.Identificador - 1).get(H.Visitar.get(j).Identificador -
     1);
8      float Eta_ij = 1/(MD.get(H.Actual.Identificador - 1).get(H.Visitar.get(j).
    Identificador - 1));
9      float P = (float) (Math.pow(Tau_ij, Alfa) * Math.pow(Eta_ij, Beta));
10     Sumatoria = Sumatoria + P;
11   }
12   for(int j = 0; j < H.Visitar.size(); j++)
13   {
14     float Tau_ij = MF.get(H.Actual.Identificador - 1).get(H.Visitar.get(j).Identificador -
     1);
15     float Eta_ij = 1/(MD.get(H.Actual.Identificador - 1).get(H.Visitar.get(j).
    Identificador - 1));
16     float P = (float) (Math.pow(Tau_ij, Alfa) * Math.pow(Eta_ij, Beta)) / Sumatoria ;
17     ProCiudad PCiudad =  new ProCiudad(H.Visitar.get(j).Identificador, P);
18     P_i.add(PCiudad);
19   }
20   int n = P_i.size();
21   ArrayList<ProCiudad> OrderP_i = SelectionSort(n, P_i);
22   ArrayList<Float> SumaCumulativa = SCumulativa(OrderP_i);
23   int Index = RuletaRusa(SumaCumulativa);
24   H.Actual = Info.Ciudades.get(OrderP_i.get(Index).Identificador - 1);
25   H.Solucion.add(H.Actual);
26   for(int i = 0; i < H.Visitar.size(); i++)
27   {
28     if(H.Actual.Identificador == H.Visitar.get(i).Identificador)
29       Index = i;
30   }
31   H.Visitar.remove(Index);
32   H.L = Distancia(H.Solucion, MD);
33   H.Delta_L = 1 / H.L;
34   return H;
35 }
```