VERSIÓN 1.0.0

PREDA 20 / 01 / 2022

PRÁCTICA DE PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS – CURSO 2021-2022

COSTE MÍNIMO ENTRE NODOS CON ALGORITMO DE FLOYD

FERMÍN GORRAIZ GARCÍA -72806064R

fermin.gorraiz@gmail.com

646 612 485

CONTENIDO

COSTE MÍNIMO ENTRE NODOS CON ALGORITMO DE FLOYD

OBJETIVO PRINCIPAL

Se trata de calcular el camino de coste mínimo entre cada par de nodos de un grafo dirigido utilizando el algoritmo de Floyd, algoritmo que utiliza el esquema de Programación Dinámica. El resultado del algoritmo será, para cada par de nodos, la secuencia de nodos del camino más corto y su valor total o longitud.

La práctica constará de una memoria y de un programa en java original que resuelva el problema aplicando el esquema indicado.

ANÁLISIS DEL ALGORITMO

El algoritmo de Floyd es un algoritmo que usa la idea de programación dinámica para encontrar la ruta más corta entre múltiples fuentes en un gráfico ponderado dado, similar al algoritmo de Dijkstra.

En resumen, este algoritmo resuelve el camino más corto entre dos puntos.

Puede manejar correctamente el problema de la ruta más corta de un gráfico dirigido o un gráfico dirigido o peso negativo (Pero no puede haber un ciclo de peso negativo).

Al momento de analizar un algoritmo revisamos el conjunto de operaciones primitivas de alto nivel que son independientes del lenguaje de programación que se utilice, estas pueden ser identificadas en el pseudocódigo del mismo.

```
}
for (k = 0; k < N; k++) {
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            tmp = this.M[i][k] + this.M[k][j];
            if (tmp < this.M[i][j]) {
                this.M[i][j] = tmp;
                this.rutas[i][j] = k;
            }
        }
    }
}
```

Si queremos dar una caracterización O(·) del tiempo de ejecución en términos de n del algoritmo debemos llevar a cabo tres sencillos pasos.

CONTEO DE OPERACIONES PRIMITIVAS

Para esto, revisamos cada paso del algoritmo en el pseudocódigo y contamos las operaciones que se ejecutan. A continuación, se muestra el procedimiento llevado a cabo:

- +1 -> Asignación k = 1
- +4(n) -> Leer referencia k, leer referencia n, comparación k = n, comparación k < n
- +1(n) -> Asignación i = 1
- +4(n^2) -> Leer referencia i, leer referencia n, comparación i = n, comparación i < n
- +1(n^2) -> Asignación j = 1

- +4(n^3) -> Leer referencia j, leer referencia n, comparación j = n, comparación j < n
- +9(n^3) -> Leer referencias i, j, k, tmp, M, rutas, comparación mínima, asignación de tmp y rutas
- +3(n^3) -> Leer referencia j, suma j+1, asignación j = j+1
- +3(n^2) -> Leer referencia i, suma i+1, asignación i = i+1
- +3(n) -> Leer referencia k, suma k+1, asignación k = k+1

ESTIMACIÓN DEL TIEMPO DE EJECUCIÓN

Ahora, calculamos el tiempo de ejecución t(n) del algoritmo Floyd-Warshall sumando el costo obtenido de las operaciones primitivas:

$16n^3 + 8n^2 + 8n + 1$

CARACTERIZACIÓN BIG-O

Aplicando la definición de Big-O, obtenemos una constante c > 0 que pertenezca a los reales y nos da como resultado un $O(N^3)$.

CONCLUSIONES

- El coste temporal con los tres bucles anidados es de $O(N^3)$ mientras que el coste espacial está en $O(N^2)$.
- La complejidad respecto al algoritmo de Dijkstra es la misma pero las operaciones de Floyd son más simples.
- Para grafos poco densos Dijkstra puede optimizarse, por lo que es preferible en el caso de grafos dispersos. Así que Floyd es recomendable para los casos de grafos densos

NOTAS Y BIBLIOGRAFÍA

- La bibliografía que he utilizado es el libro de la asignatura y búsquedas en Google, principalmente https://es.stackoverflow.com/

ANEXO I

CÓDIGO DE LAS CLASES

SKYLINE

PACKAGE MAIN

FLOYD.JAVA

```
1.  package main;
2.
3.  import main.common.CommonApiDate;
4.  import main.common.CommonApiMenu;
5.
6.  public class FloydMain {
7.
8.      private static final String SHOW_TRACE = "-t";
9.      private static final String SHOW_HELP = "-h";
10.
11.     private static boolean showTrace = false;
12.
13.     public static void main(String[] args) {
14.         System.out.println(main.common.CommonApiDate.dateNowToString() + " || Launch
    Floyd\r\n");
15.         boolean isInput = true;
16.
17.         String inputFile = null;
18.         String outputFile = null;
19.
20.         if (args.length > 4) {
21.             System.out.println("Too much args, see [floyd -h] for help\r\n");
22.             return;
23.         }
24.
25.         for (String a : args) {
26.             switch (a) {
27.             case SHOW_TRACE:
28.                 showTrace = true;
29.                 continue;
30.             case SHOW_HELP:
31.                 System.out.println(CommonApiMenu.showHelp());
32.                 continue;
33.             }
34.
35.             if (isInput) {
36.                 inputFile = a;
37.                 isInput = false;
38.             } else {
39.                 outputFile = a;
40.             }
41.         }
42.
43.         (new Floyd()).run(inputFile, outputFile, showTrace);
44.         System.out.println(CommonApiDate.dateNowToString() + " || Close Floyd\r\n");
45.     }
46. }
47.
```

FLOYD.JAVA

```
1.  package main;
2.
3.  import static java.util.Objects.isNull;
4.
5.  import main.common.CommonApiFile;
6.  import main.common.CommonApiMenu;
7.
8.  public class Floyd {
9.
10.   public static final int INT_INF = 9999;
11.   public static final String STR_INF = "-";
12.
13.   public int[][] M = new int[0][0];
14.   public int[][] rutas = new int[0][0];
15.
16.   public void run(String inputFile, String outputFile, boolean showTrace) {
17.             int[][] A = null;
18.             if (!isNull(inputFile)) {
19.                       A = CommonApiFile.readFile(inputFile);
20.             } else {
21.                       A = CommonApiMenu.showInputMenu();
22.             }
23.
24.             int N = 0;
25.             for (int[] ints : A) {
26.                       N = ints.length;
27.             }
28.
29.             this.M = new int[N][N];
30.             this.rutas = new int[N][N];
31.
32.             int i, j, k, tmp;
33.             for (i = 0; i < N; i++) {
34.                       for (j = 0; j < N; j++) {
35.                                 this.M[i][j] = A[i][j];
36.                                 this.rutas[i][j] = 0;
37.                       }
38.             }
39.             for (k = 0; k < N; k++) {
40.                       for (i = 0; i < N; i++) {
41.                                 for (j = 0; j < N; j++) {
42.                                           tmp = this.M[i][k] + this.M[k][j];
43.                                           if (tmp < this.M[i][j]) {
44.                                                     this.M[i][j] = tmp;
45.                                                     this.rutas[i][j] = k;
46.                                           }
47.                                 }
48.                       }
49.             }
50.
51.             String result = VerRutas(A, N);
52.
53.             if (!isNull(outputFile)) {
54.                       CommonApiFile.createAndWriteFile(outputFile, result);
55.             } else {
56.                       System.out.println(result);
57.             }
58.   }
59.
60.   private String VerRutas(int[][] A, int N) {
61.             StringBuilder sb = new StringBuilder();
62.
63.             int i, j;
64.             for (i = 0; i < N; i++) {
```

```
65.                     for (j = 0; j < N; j++) {
66.                             if (this.rutas[i][j] != INT_INF) {
67.                                     sb.append("[" + (i + 1) + ", " + (j + 1) + "]:
    ");
68.                                     sb.append((i + 1) + ",");
69.                                     sb.append(ImprimeRutaRec(i, j));
70.                                     sb.append((j + 1) + ": ");
71.                                     sb.append(this.M[i][j]);
72.                                     sb.append("\r\n");
73.                             }
74.                     }
75.             }
76.
77.             return sb.toString();
78.     }
79.
80.     private String ImprimeRutaRec(int i, int j) {
81.             StringBuilder sb = new StringBuilder();
82.
83.             int k = this.rutas[i][j];
84.             if (k != 0) {
85.                     sb.append(ImprimeRutaRec(i, k));
86.                     sb.append((k + 1) + ",");
87.                     sb.append(ImprimeRutaRec(k, j));
88.             }
89.
90.             return sb.toString();
91.     }
92. }
93.
```

PACKAGE COMMON

COMMONAPIDATE.JAVA

```
1.  package main.common;
2.
3.  import java.text.DateFormat;
4.  import java.text.SimpleDateFormat;
5.  import java.util.Date;
6.
7.  public class CommonApiDate {
8.      public static String dateNowToString() {
9.          Date date = new Date();
10.         DateFormat dateFormat = new SimpleDateFormat("hh:mm:ss.SSS");
11.
12.         return dateFormat.format(date);
13.     }
14. }
15.
```

COMMONAPIFILE.JAVA

```
1.  package main.common;
2.
3.  import static java.util.Objects.isNull;
4.
5.  import java.io.File;
6.  import java.io.FileNotFoundException;
7.  import java.io.FileWriter;
8.  import java.io.IOException;
9.  import java.util.Scanner;
10.
11. import main.Floyd;
12.
```

```java
public class CommonApiFile {

    public static int[][] readFile(String filename) {
        int[][] A = null;

        try {
            File myObj = new File(filename);
            Scanner myReader = new Scanner(myObj);

            int i = 0;
            while (myReader.hasNextLine()) {
                int j = 0;
                String data = myReader.nextLine();
                String[] split = data.split(" ");

                if (isNull(A)) {
                    int l = split.length;
                    A = new int[l][l];
                }

                int value = 0;
                for (String s : split) {
                    switch (s) {
                    case Floyd.STR_INF:
                        value = Floyd.INT_INF;
                        break;
                    default:
                        value = Integer.valueOf(s);
                        break;
                    }

                    A[i][j] = value;
                    j++;
                }

                i++;
            }
            myReader.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred.\r\n");
            e.printStackTrace();
        }

        return A;
    }

    public static void createAndWriteFile(String filename, String text) {
        createFile(filename);
        writeToFile(filename, text);
    }

    public static void createFile(String filename) {
        try {
            File myObj = new File(filename);
            if (myObj.createNewFile()) {
                System.out.println("File created: " + myObj.getName() + "\r\n");
            }
        } catch (IOException e) {
            System.out.println("An error occurred.\r\n");
            e.printStackTrace();
        }
    }

    public static void writeToFile(String filename, String text) {
        try {
            FileWriter myWriter = new FileWriter(filename);
            myWriter.write(text);
            myWriter.close();
            System.out.println("Successfully wrote to the file: " + filename + "\r\n");
        } catch (IOException e) {
```

```
83.             System.out.println("An error occurred.\r\n");
84.             e.printStackTrace();
85.         }
86.     }
87. }
88.
```

COMMONAPIMENU.JAVA

```
1.  package main.common;
2.
3.  import java.util.Scanner;
4.
5.  import main.Floyd;
6.
7.  public class CommonApiMenu {
8.
9.    public static String showHelp() {
10.             StringBuilder sb = new StringBuilder();
11.
12.             sb.append("SINTAXIS: floyd [-t][-h] [fichero entrada] [fichero
    salida]\r\n");
13.             sb.append("-t Traza el algoritmo\r\n");
14.             sb.append("-h Muestra esta ayuda\r\n");
15.             sb.append("[fichero entrada] Matriz de adyacencia que representa el
    grafo\r\n");
16.             sb.append(
17.                                 "[fichero salida] Para cada par de nodos: la lista de
    nodos del camino más corto y su valor o longitud\r\n");
18.
19.             return sb.toString();
20.   }
21.
22.   public static String showFileNames(String inputFile, String outputFile) {
23.             StringBuilder sb = new StringBuilder();
24.
25.             sb.append("fichero entrada: " + inputFile + "\r\n");
26.             sb.append("fichero salida: " + outputFile + "\r\n");
27.
28.             return sb.toString();
29.   }
30.
31.   public static int[][] showInputMenu() {
32.             int N = 0;
33.             int[][] A = null;
34.
35.             Scanner in = new Scanner(System.in);
36.
37.             try {
38.                       System.out.println("Introduce el orden de la matriz: ");
39.                       N = Integer.valueOf(in.nextLine());
40.
41.                       A = new int[N][N];
42.
43.                       String msg = "";
44.                       for (int i = 0; i < N; i++) {
45.                                 for (int j = 0; j < N; j++) {
46.                                           msg = String.format("Introduce el valor
    [%d][%d] de la matriz: ", i + 1, j + 1);
47.                                           System.out.println(msg);
48.                                           String s = in.nextLine();
49.
50.                                           int value = 0;
51.                                           switch (s) {
52.                                           case Floyd.STR_INF:
```

```
53.                                                 value = Floyd.INT_INF;
54.                                                 break;
55.                                         default:
56.                                                 value = Integer.valueOf(s);
57.                                                 break;
58.                                         }
59.
60.                                         A[i][j] = value;
61.                                 }
62.                         }
63.             } catch (Exception e) {
64.                         System.out.println("!!No es un número válido¡¡");
65.             }
66.
67.             in.close();
68.             return A;
69.     }
70. }
71.
```