Actividad 3: Grafos

Alejandro Ye, David Cuesta, Urtzi González, Zuhaitz Martínez

26 de noviembre de 2017

Índice general

Capítulo 1

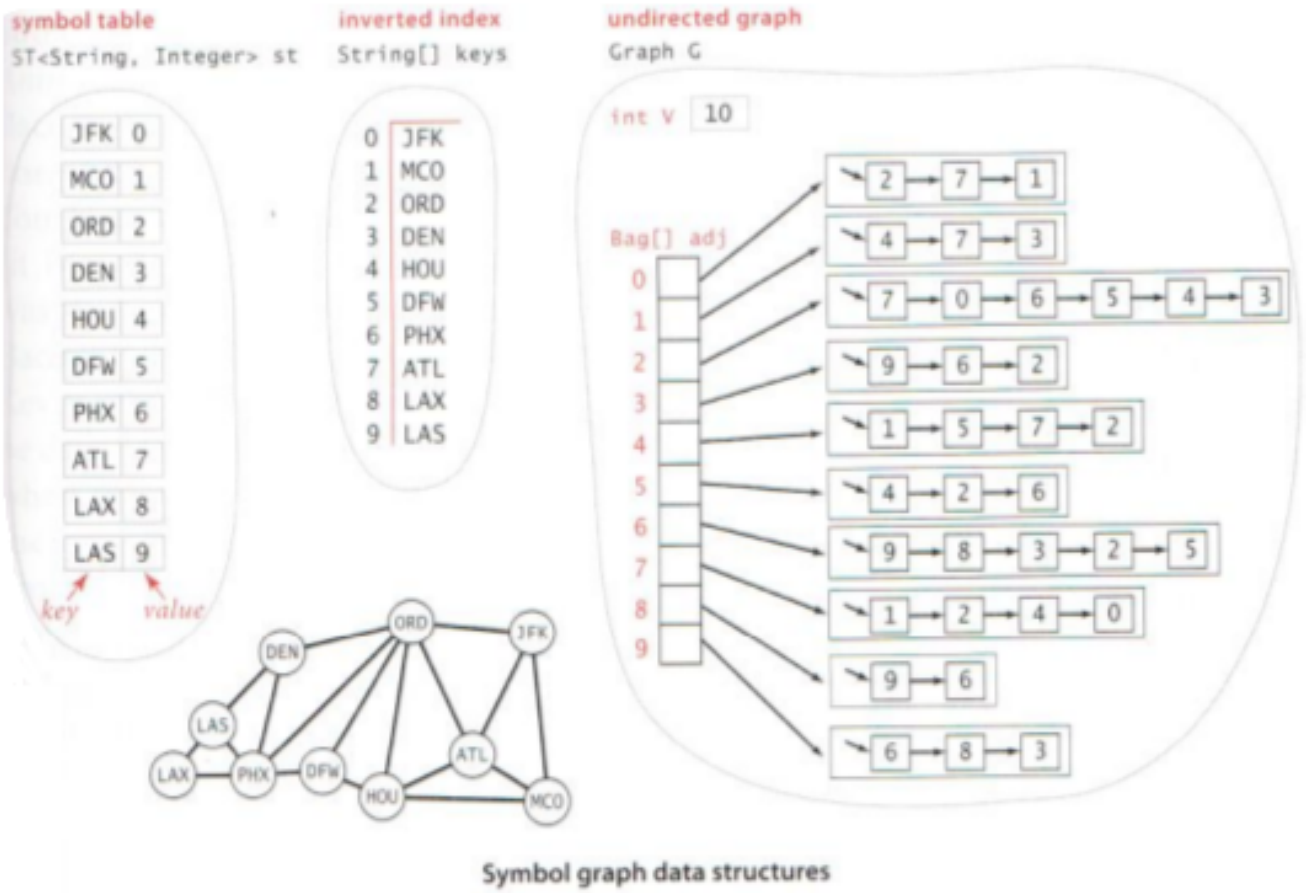
Introducción

El problema planteado es implementar una estructura de datos que conecte las paginas webs de la primera actividad ('El buscador de páginas web') y poder ver si hay una cadena de relaciones que une una web a otra. Además, como parte opcional se podrá detallar cuál es el camino recorrido .

Capítulo 2

Diseño de las clases

En esta sección se mencionarán las clases principales planteadas para la solución al problema.



Symbol graph data structures

Figura 2.1: Diagrama de clases.

Capítulo 3

Descripción de las estructuras de datos principales

Hemos utilizado las estructuras seleccionadas por los profesores (HashMap, array de strings y array de arraylists de integer).

El HashMap es utilizado debido a que es una estructura con un tiempo de inserción y búsqueda muy rápido incluso con strings.

El array de strings se usa debido a que todas las busquedas que vamos a realizar en él van a ser introduciendo el índice que queremos buscar y esa operación es de orden O(1) por lo que es muy eficiente para esta implementación.

El array que contiene arraylists se ha decidido utilizar ya que necesitamos almacenar las conexiones que existen entre los nodos. Posiblemente no sea la estructura más eficiente para esta función, pero es suficientemente eficiente como para realizar accesos a las posiciones del array y recorrer los elementos del arraylist, por lo que su uso es adecuado para esta implementación.

Se plantea el uso de una cola implementada mediante linkedlists para almacenar los posibles nodos a analizar cuando estamos realizando la comprobación de si hay conexión o no. El hecho de que sea una cola tiene sentido, ya que deseamos realizar la busqueda en el grafo en anchura en lugar de en profundidad, ya que realizar la busqueda en anchura nos asegura que se encontrará el camino más corto posible al posible nodo al que esté enlazada una pagina web.

Capítulo 4

Diseño e implementación de los métodos principales

4.1. Clase Graph

4.1.1. Método crearGrafo(Webs webs)

Postcondición: crea el grafo desde la lista de webs. Los nodos son nombres de webs.

Coste: O(n).

Definición: Dado la estructura de webs que hemos realizado en las anteriores actividades, irá creando el HashMap, array de strings y array de arraylists de integer, con su respectiva información.

4.1.2. Método estanConectados(String a1, String a2)

Precondición: se dan dos nombres de paginas web.

Postcondición: se devuelve un booleano que indica si esas dos webs estan conectadas.

Coste: O(n)

Definición: Dado los dos nombres de las webs, se comprueba si los dos son iguales en ese caso se habrá terminado, en caso contrario, se buscará la primera página en el HashMap para conseguir el id de la página y se obtendrá con ello la lista de enlaces de esa página. Se añadirán los elementos de esta lista a una cola de porExaminar para ir mirándolos en orden, y los que ya hayan sido revisados se marcaran con un booleano en un array de examinados. así hasta que se en cuentre la página o se vacíe la cola, que en este caso se devolverá False.

4.1.3. Método caminoConectado(String a1, String a2)

Precondición: se dan dos nombres de paginas web.

Postcondición: se devuelve una lista con los indices del recorrido.

Coste: O(n)

Definición: Dado los dos nombres de las webs, se comprueba si los dos son iguales en ese caso se habrá terminado, en caso contrario, se buscará la primera página en el HashMap para conseguir el id de la página y se obtendrá con ello la lista de enlaces de esa página. Se añadirán los elementos de esta lista a una cola de porExaminar para ir mirándolos en orden, y los que ya hayan sido revisados se marcaran con un booleano en un array de examinados. así hasta que se en cuentre la página o se vacíe la cola, cuando termine se comprobara el recorrido que ha hecho.

4.1.4. Método estanConectadosOpcional(String a1, String a2)

Precondición: se dan dos nombres de paginas web.

Postcondición: se devuelve una lista con las webs que conectan a1 con a2.

Coste: O(n + m) Siendo n el coste de la funcion caminoConectado(String a1, String a2) y m la longitud de la lista que devuelve.

Definición: Dado los dos nombres de las webs, se llama al método caminoConectado(String a1, String a2) que le devolverá la lista de id de las paginas que irá buscando en el array para añadir el nombre de la página a una array para luego devolverlo.

4.1.5. Método probar(int nVeces, int indMAX)

Definición: método que se usará para hacer numerosas pruebas de seguido, escogiendo ids aleatorios.

Casos de prueba (Véase el apartado 5.Código):

Para ello se ha utilizado el metodo probar(int nVeces, int indMAX).

Capítulo 5

Código

En este apartado se presentará el código de las clases, junto con los programas de prueba o Junits desarrollados.

5.1. Clase Graph

```
1  package eda;

3  import java.util.*;

5  public class Graph {

7      HashMap<String, Integer> th;
       String[] keys;
9      ArrayList<Integer>[] adjList;
       private static Graph miGraph = null;
11
       private Graph(){
13         this.th = new HashMap<String, Integer>();
       }
15
       public static Graph getGraph() {
17         if(miGraph == null) {
               miGraph = new Graph();
19         }
           return miGraph;
21     }

23     public void crearGrafo(Webs webs){
           // Post: crea el grafo desde la lista de webs
25         //       Los nodos son nombres de webs

27         // Paso 1: llenar th

29         int tamanoArbol = webs.tamano();
```

```
        int tamanoArray = tamanoArbol+1000;
31      PagWeb web;
        adjList = new ArrayList[tamanoArray]; //el array tiene espacio para
33
        //Inicializar las el array de arraylist
35      //Inicializar los arraylist del array
        for( int j=0; j < tamanoArray; j++ ){
37          adjList[j] = new ArrayList<Integer>();
        }
39
        Iterator<PagWeb> it = webs.getIterador();
41      while (it.hasNext()){
            web = it.next();
43          String nombre = web.getNombre();
            int id = web.getId();
45          th.put(nombre,id);
            // Paso 3: llenar adjList
47          adjList[id] = web.getListaRef(); //anade en el array de referen
        }
49
        // Paso 2: llenar keys en un array
51      keys = new String[th.size()];
        for (String k: th.keySet()) keys[th.get(k)] = k;
53
    }
55
    public void print(){
57     for (int i = 0; i < adjList.length; i++){
           System.out.print("Element: " + i + " " + keys[i] + " --> ");
59         for (int k: adjList[i])  System.out.print(keys[k] + " ### ");
           System.out.println();
61     }
    }
63
    public boolean estanConectados(String a1, String a2){
65      Queue<Integer> porExaminar = new LinkedList<Integer>();

67      int pos1 = th.get(a1);
        int pos2 = th.get(a2);
69      int actual = pos1;
        boolean enc = false;
71      boolean[] examinados = new boolean[th.size()];

73      // TODO: REVISAR
        if(a1.equals(a2)) enc = true;
75      else{
            porExaminar.add(pos1);
77          examinados[pos1] = false;
            while(!enc && !porExaminar.isEmpty()){
79              actual = porExaminar.poll();
```

```
                if (!examinados[actual]) {
81                  examinados[actual] = true;
                    if(actual == pos2) enc = true;
83                  else{
                        for (int i = 0; i<adjList[actual].size();i++){
85                          porExaminar.add(adjList[actual].get(i));
                        }
87                  }
                }
89          }
        }
91      return enc;
    }
93
    public LinkedList<Integer> caminoConectado(String a1, String a2){
95      //metodo que busca el camino y almacena los ids en un array
        Queue<Integer> porExaminar = new LinkedList<Integer>();
97      int[] conexiones = new int[th.size()];
        LinkedList<Integer> camino = new LinkedList<Integer>();
99      int pos1 = th.get(a1);
        int pos2 = th.get(a2);
101     int actual = pos1;
        int anterior = actual;
103     boolean enc = false;
        boolean[] examinados = new boolean[th.size()];
105
        // TODO: COMPLETAR
107     if (estanConectados(a1, a2)) {
            if(a1.equals(a2)) enc = true;
109         else{
                porExaminar.add(pos1);
111             examinados[pos1] = false;
                while(!enc && !porExaminar.isEmpty()){
113                 actual = porExaminar.poll();
                    if (!examinados[actual]) {
115                     //actual = porExaminar.poll();
                        conexiones[actual] = anterior;
117                     if(actual == pos2) enc = true;
                        else{
119                         for (int i = 0; i<adjList[actual].size();i++){
                                porExaminar.add(adjList[actual].get(i));
121                         }
                        }
123                     examinados[actual] = true;
                    }
125                 anterior = actual;

127             }
            }
129
```

```
            while(conexiones[actual] != conexiones[pos1]) {
                camino.add(conexiones[actual]);
                actual= conexiones[actual];
            }
            camino.add(conexiones[pos1]);
        }
        return camino;
    }

    public ArrayList<String> estanConectadosOpcional(String a1, String a2){
        //metodo opcional que dada la lista de ids del camino devuelve el n
        LinkedList<Integer> enlacesID =  caminoConectado(a1, a2);
        ArrayList<String> enlacesDeWebs = new ArrayList<String>();
        ListIterator<Integer> listIterator = enlacesID.listIterator();

        while(listIterator.hasNext()) {
                enlacesDeWebs.add(keys[enlacesID.pollLast()]);
        }
        return enlacesDeWebs;
    }

    public void printCamino(String a1, String a2){
        ArrayList<String> lista = new ArrayList<String>();
        lista = estanConectadosOpcional(a1, a2);
        ListIterator<String> listIterator = lista.listIterator();
        while(listIterator.hasNext()) {
            String s = listIterator.next();
            System.out.print(s);
            if(listIterator.hasNext())System.out.print(" --> ");
        }
        System.out.println("");
    }

    public double probar(int nVeces, int indMAX) {
        //post: el resultado es el tiempo(ms)
        Random rand = new Random();
        int cont = 0;
        double tTardado = 0;
        boolean conectado;

        StopWatch timer = new StopWatch();

        while (cont<nVeces){
            conectado = false;
            StopWatch timer2 = new StopWatch();
            int  n1 = rand.nextInt(indMAX) + 1;
            int  n2 = rand.nextInt(indMAX) + 1;
            if (n1 != n2) {
                String v1 = keys[n1];
                String v2 = keys[n2];
```

```
                    conectado = estanConectados(v1,v2);
181
                     //PARA PRUEBAS
183                 System.out.println("ID: "+n1 +" URL: "+v1+", "+"ID: "+n2+"
                    if (conectado) {
185                     System.out.print("SI estan conectadas.");
                    }
187                 else {
                        System.out.print("NO estan conectadas.");
189                 }

191                 cont++;
                }
193             System.out.println(" ha tardado en analizar la conexion "+timer
            }
195         System.out.println("Resultado: ");
            System.out.println("Ha tardado en analizar la conexion de " +cont+
197         tTardado = (timer.elapsedTime())/nVeces;
            return tTardado;
199     }

201 }
```

5.2. Casos de prueba Graph

Estos extractos de código se encuentran en MainMenu.java La tabla 5.1 en la siguiente página contiene los casos que se han probado. Las pruebas resultantes indican que cada búsqueda de conexiones tarda aproximadamente un milisegundo.

```
1       private void estanConectados() {
            //PRUEBA AUTOMATICA DE VARIAS WEBS
3           System.out.println("Introduzca el numero de conexiones que desea bu
            String string1 = miMenu.leer();
5           int nVeces = Integer.parseInt(string1);
            System.out.println("Introduzca un indice menor que el maximo numero
7           String string2 = miMenu.leer();
            int indMAX = Integer.parseInt(string2);
9           Graph grafo = Graph.getGraph();
            grafo.crearGrafo(Webs.getWebs());
11          double tiempoMedio = grafo.probar( nVeces,  indMAX);
            System.out.println("Ha tardado para cada web un tiempo medio de : "
13      }
```

```
1
        /*private void estanConectados() {
3           //PRUEBAS DE UNO EN UNO MANUAL
            System.out.println("Introduzca la primera pagina web: ");
5           String string1 = miMenu.leer();
            System.out.println("Introduzca la segunda pagina web: ");
7           String string2 = miMenu.leer();
            StopWatch timer = new StopWatch();
9           Webs webs = Webs.getWebs();
            Graph grafo = Graph.getGraph();
11          grafo.crearGrafo(webs);
            boolean conexion = grafo.estanConectados(string1,string2);
13          if (conexion) {
                System.out.println("Las dos webs introducidas si estan conectad
15          }
            else {
17              System.out.println("Las dos webs introducidas no estan conectad
            }
19          System.out.println("Ha tardado: "+timer.elapsedTime()+"s en encontr
            */
21      }
```

```
1       private void estanConectadosOpcional() {

3           System.out.println("Introduzca la primera pagina web: ");
            String string1 = miMenu.leer();
5           System.out.println("Introduzca la segunda pagina web: ");
            String string2 = miMenu.leer();
7           StopWatch timer = new StopWatch();
            Webs webs = Webs.getWebs();
9           Graph grafo = Graph.getGraph();
            grafo.crearGrafo(webs);
11          LinkedList<Integer> cam = new LinkedList<Integer>();
            cam = grafo.caminoConectado(string1,string2);
13          System.out.println(cam);
            System.out.println("Esto de arriba es una prueba para mi, debajo es
15          grafo.printCamino(string1,string2);
            System.out.println("Imprimira las conexiones, sino no imprimira nad
17
            System.out.println("Ha tardado: "+timer.elapsedTime()+"s en encontr
19      }
```

Casos de prueba		
Tamaño	**Hay conexión**	**No hay conexión**
Vacío	False	False
Un elemento	False	False
Dos elementos	True	False
Primer y último elemento	True	False
Elementos aleatorios	True	False

Cuadro 5.1: Los casos que se han probado con estos extractos de código son los mostrados.

Capítulo 6

Conclusiones

Después de finalizar la práctica hemos aprendido a cómo crear grafos y utilizar la clase colas. Asimismo, hemos visto una de las ventajas de esta estructura de datos colas y su eficiencia frente a, por ejemplo, las pilas. Ya que con las colas recorremos por niveles (circularmente) el grafo y así obtener el camino más corto, mientras que si hubiésemos utilizado pilas habríamos recorrido todos los caminos hasta el final y pararía una vez encontrada tal conexión pero ese camino y ese camino podría ser el más corto por casualidad pero lo más probable es que no lo fuera.

Además hemos llegado a la conclusión de que nos tenemos que comunicar mejor para trabajar conjuntamente y ayudarnos mutuamente en vez de realizar los trabajos tan individualmente.