Laboratorio Nro. 3: Vuelta atrás (*Backtracking*)

Kevin Arley Parra Henao
Universidad EAFIT
Medellín, Colombia
kaparrah@eafit.edu.co

Daniel Alejandro Mesa Arango
Universidad EAFIT
Medellín, Colombia
damesaa@eafit.edu.co

1.8) Escriban una explicación entre 3 y 6 líneas de texto del código del ejercicio en línea del numeral 1.6 Digan cómo funciona, cómo está implementado y destaquen las estructuras de datos y algoritmos usados

Este código es una versión modificada de DFS en el cual se tiene un parámetro fin, que indica hasta qué punto irá el recorrido, si fin es igual a -1, se recorrerán todos los vértices del grafo, Así este código sirve para el ejercicio 1.6 y 2.1. Este código usa además dos listas de enteros, una para guardar los vértices que conforman el menor camino y la otra para guardar los pesos, al final se suman los pesos y se sabe el costo mínimo del trayecto.

3) Simulacro de preguntas de sustentación de Proyectos

1. **Para resolver el problema del camino más corto en un grafo, fuera de fuerza bruta y backtracking, ¿qué otras técnicas computacionales existen?**

   - Algoritmo de Dijkstra, resuelve el problema de los caminos más cortos desde un único vértice origen hasta todos los otros vértices del grafo.
   - Algoritmo de Bellman - Ford, resuelve el problema de los caminos más cortos desde un origen si la ponderación de las aristas es negativa.
   - Algoritmo de Búsqueda A*, resuelve el problema de los caminos más cortos entre un par de vértices usando la heurística para intentar agilizar la búsqueda.
   - Algoritmo de Floyd - Warshall, resuelve el problema de los caminos más cortos entre todos los vértices.
   - Algoritmo de Johnson, resuelve el problema de los caminos más cortos entre todos los vértices y puede ser más rápido que el de Floyd-Warshall en grafos de baja densidad.
   - Algoritmo de Viterbi, resuelve el problema del camino estocástico más corto con un peso probabilístico adicional en cada vértice.

2. **Teniendo en cuenta lo anterior, tomen los tiempos de ejecución del programa realizado en el numeral 1.1 y en el laboratorio anterior con la solución de fuerza bruta de las n reinas. Completen la siguiente tabla.**

Valor de N	Tiempo de ejecución
4	0 milisegundos
5	0 milisegundos
6	1 milisegundos
7	1 milisegundos
8	18 milisegundos
9	133 milisegundos
10	2602 milisegundos
11	57044 milisegundos
12	Mas de 20 minutos
13	Mas de 20 minutos
14	Mas de 20 minutos
15	Mas de 20 minutos
16	Mas de 20 minutos
17	Mas de 20 minutos
18	Mas de 20 minutos
19	Mas de 20 minutos
20	Mas de 20 minutos
21	Mas de 20 minutos
22	Mas de 20 minutos
23	Mas de 20 minutos
24	Mas de 20 minutos
25	Mas de 20 minutos
26	Mas de 20 minutos
27	Mas de 20 minutos
28	Mas de 20 minutos
29	Mas de 20 minutos
30	Mas de 20 minutos
31	Mas de 20 minutos
32	Mas de 20 minutos
N	**O ( )**

3. **Para recorrer grafos, ¿en qué casos conviene usar *DFS*? ¿En qué casos *BFS*?**

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Para encontrar el camino mas corto por ejemplo DFS suele ser más rápido al igual que para detectar si hay un ciclo o no en el grafo, para Facebook por ejemplo BFS es más rápido encontrando amigos de amigos ya que busca en amplitud y no tiene que ir hasta el fondo, como un ejemplo más en un árbol genealógico si los familiares más antiguos están en la cima y los mas recientes en las puntas será más fácil para DFS llegar a las puntas mientras que buscar ancestros le tomara menos tiempo a BFS

4. **¿Qué otros algoritmos de búsqueda existen para grafos? Basta con explicarlos, no hay que escribir los algoritmos ni programarlos.**

   Algoritmo de Bloques
   El algoritmo de bloques de Jungnickel (Blockcut en el original) es una adaptación del algoritmo DFS de Robert Tarjan, enunciado en el año 72, y que a su vez es una variación ligeramente más eficiente del algoritmo de Kosaraju. Permite hacer una partición de un grafo en los bloques de que está formado. Además, encuentra los puntos de corte del grafo original.

   Algoritmo de Dijkstra
   El algoritmo de Dijkstra debe su nombre a su creador, Edsger Dijkstra, quien lo concibió en 1956 (fue posteriormente publicado en 1959). Se trata de un algoritmo de búsqueda sobre grafos que soluciona el problema del camino mínimo (de ahí que también sea conocido como algoritmo de caminos mínimos) en grafos de aristas con pesos no negativos.

5. **Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y 2.2 [Ejercicio Opcional] y digan cómo funciona el programa.**

   Este código es una versión modificada de DFS en el cual se tiene un parámetro fin, que indica hasta qué punto irá el recorrido, si fin es igual a -1, se recorrerán todos los vértices del grafo, Así este código sirve para el ejercicio 1.6 y 2.1. Este código usa además dos listas de enteros, una para guardar los vértices que conforman el menor camino y la otra para guardar los pesos, al final se suman los pesos y se sabe el costo mínimo del trayecto.

6. **Calculen la complejidad de los ejercicios en línea del numeral 2.1 y 2.2 [Ejercicio Opcional] y agréguenla al informe PDF**

   Al ser un algoritmo DFS modificado, su complejidad sigue siendo O(n+m), ya que las modificaciones no alcanzan a superar esta complejidad ni a minimizar en el peor de los casos.

7. **Expliquen con sus palabras las variables (*qué es 'n', qué es 'm', etc.*) del cálculo de complejidad del numeral 3.6**

   Descripción de las variables: n es el número de vértices y m el número de aristas

4) Simulacro de Parcial

1. A. solucionar(n-a, a, b, c)+1;
   B. Math.max(res, solucionar(n-b,a,b,c)+1);
   C. Math.max(res, solucionar(n-c, a,b,c)+1);
2. A. pos==graph.length
   B. sePuede(v, graph,path, pos+1)
   C. cicloHamilAux(graph, path, pos+1)

3. A. 0-3-7-4-2-6
1-2-4-6
3-7
2-4-6
4-2-6
7
6-2-4
5
B. 7
6-2-4
5
4-2-6
3-7
2-4-6
1-2-5-4-6
0-3-4-7-2-6

4.

```
public static ArrayList<Integer> recorrido(Digraph g, int star, int final) {
      int tamano = g.size();
      boolean [] visitados =  new boolean[tamano];
      ArrayList<Integer> recorridos = new ArrayList<>();
      recorrido(g, star, final visitados, recorridos);
      return recorridos;
   }

   private static boolean recorrido(Digraph g, int pos, int final, boolean[] unvisited, ArrayList<Integer>
recorridos)
   {
      unvisited[pos] = true;
      recorridos.add(pos);
      if(pos == final)
      {
        return true;
      }
      recorridos.add(pos);
      ArrayList<Integer> sucesores = g.getSuccessors(pos);
      if(sucesores != null)
      for(Integer sucesor : sucesores)
         {
            if(!unvisited[sucesor])
               {
                  recorrido(g, sucesor, unvisited, recorridos);
               }
            //return;
         }

         return false;


   }
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

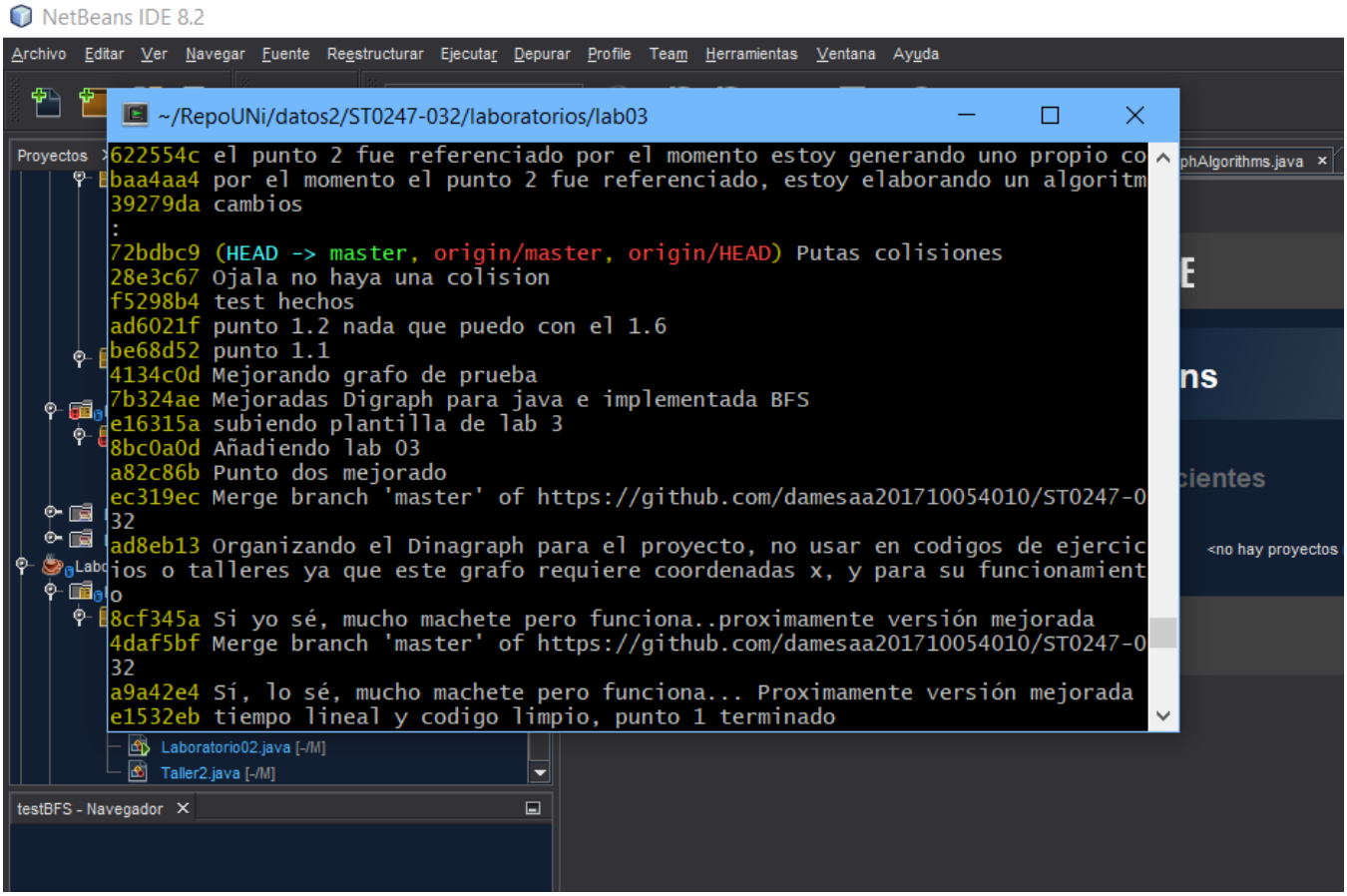
5. *A. 1*
B. ni, nj
C. 2^n

6) Trabajo en Equipo y Progreso Gradual (Opcional)

a) Actas de reunión

Integrante	Fecha	Hecho	Haciendo	Por hacer
Kevin Parra	9/03/2018	Subir plantilla a drive	Punto 1 del laboratorio	Subir código
Daniel Mesa	10/03/2018	Punto 1.1 y 1.2	Punto 3.1	tiempos
Kevin Parra	11/03/2018	Subir carpetas al repositorio	Punto 1 y 2	Poner acta en el informe y subir el código final
Daniel Mesa	11/03/2018	punto 3.1, 3.2, 3.4	simulacro	punto 1.4
Daniel Mesa	11/03/2018	punto 1.4 y simulacro	Revisar	Entregar

b) El reporte de cambios en el código



c) El reporte de cambios del informe de laboratorio