Taller en Sala Nro. 4
Backtracking para Grafos - Básico

Los algoritmos para encontrar caminos se usan en aplicaciones de mapas. Vean: http://kevanahlquist.com/osm_pathfinding/

Importante:

Los algoritmos que desarrollarán en este taller deben funcionar al menos para los siguientes tipos de grafo:

- ☑ Grafos con ciclos
- ☑ Grafos con islas (donde dados dos vértices de este estos pueden no estar conectados de ninguna manera
- ☑ Grafos donde dados dos vértices de este hay varios caminos entre ellos

Trabajo en Parejas

Hoy, plazo máximo de entrega

Docente entrega código suelto en GitHub

Sí .cpp, .py o .java

No .zip, .txt, html o .doc

Alumnos entregan código suelto por GitHub

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

Ejercicios a resolver

1. Implementen un método que dado un grafo (*Digraph g*) y el identificador de un nodo en el grafo (*int start*), realice un recorrido en profundidad desde *start*, y retorne una lista en la que almacena los indicadores de los vértices a medida que los visita.

2. Implementen un método que dado un grafo (*Digraph g*) y el identificador de un nodo en el grafo (*int start*), realice un recorrido en anchura desde *start*, y retorne una lista en la que almacena los indicadores de los vértices a medida que los visita.

3. Usando DFS, escriban una implementación que retorne si hay camino o no entre un nodo *i* y un nodo *j* en un digrafo *g*.

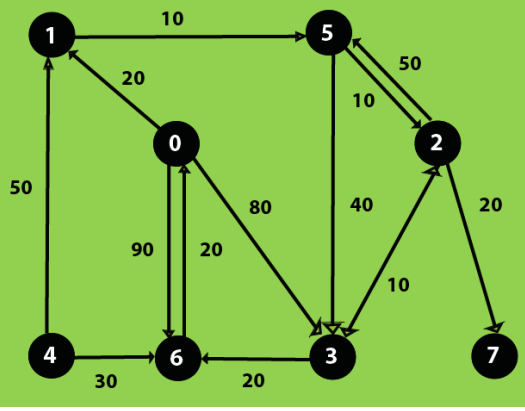
4. Usando BFS, escriban una implementación que retorne si hay camino o no entre un vértice *i* y un vértice *j* en un digrafo *g*.

5. Dado un grafo dirigido y el índice de dos de sus vértices *inicio* y *fin*, hallen el costo del camino de menor costo (valga la redundancia) que inicia en *inicio* y llega a *fin* **utilizando backtracking.**

Asuman que los vértices dados sin válidos y que para ir del vértice *k* a sí mismo el costo es 0. Si no hay camino retorne -1.

```
public static int costoMinimo(Digraph g, int inicio, int fin) {
   // complete...
}
```

Por ejemplo, en el siguiente grafo el camino más corto para ir de 0 a 6 tiene costo total 70: 0, 1, 5, 2, 3, 6.



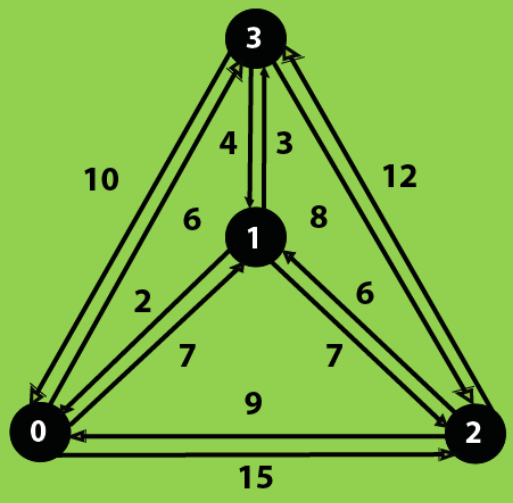
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Nótese que hay otros caminos como 0, 6 y 0, 3, 6; pero tienen un costo mayor (90 y 100 respectivamente).

6. Dado un grafo dirigido completo, hallen el costo mínimo del recorrido que pasa por todos los vértices exactamente una vez y vuelve al inicial utilizando *backtracking*.

```
public static int recorrido(Digraph g) {
     // complete...
}
```

Por ejemplo, en el siguiente grafo el costo total del recorrido de costo mínimo que pasa por los cuatro vértices (iniciando y volviendo a 0) de 22:



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

Ayudas para resolver los ejercicios

Ayudas para el Ejercicio 1

Pista: En caso de poder visitar más de un hijo en el recorrido, visítelos en base a su identificador de menor a mayor.

☑ En caso de que el nodo *start* sea un nodo aislado (es decir, no se conecta a ningún otro nodo) retorne *null*. Por ejemplo, en el grafo de la imagen los nodos 0, 1, 2, 4, 6, 9 y 10 son nodos aislados.

Como un ejemplo, así es un recorrido en profundidad desde 7 en el grafo de la imagen:

7 → 7, 8, 9, 11, 2, 10

```
public static ArrayList<Integer> dfs(Digraph g, int start) {

}
```

Pista: implemente el método de forma recursiva y cree una función auxiliar similar a esta:

```
private static void dfs(Digraph g, int nodo, boolean[] visitados,
ArrayList<Integer> list) {

}
```

Error Común 1: Olvida hacer el arreglo de visitados

```
public void dfsMalo(Graph g, int v) {
```

```
   dfsAux(Graph g, int v, visitados);
}
```

Error Común 2: No utiliza el arreglo de visitados

```
public void dfsAuxMalo(Graph g, int v, boolean[] vi){
   vi[v] = true;
   System.out.println(v);
   ArrayList<Integer> vecinos = g.successors(v);
   for (Integer vecino: vecinos) {
      dfsAux(g, vecino, vi);
   }
}
```

Ayudas para el Ejercicio 2

Pista: En caso de poder visitar más de un hijo en el recorrido, visítelos en base a su identificador de menor a mayor.

☑ En caso de que el nodo start sea un nodo aislado (es decir, no se conecta a ningún otro nodo) retorne null. Por ejemplo, en el grafo de la imagen los nodos 0, 1, 2, 4, 6, 9 y 10 son nodos aislados.

Como un ejemplo, así es un recorrido en anchura desde 7 en el grafo de la imagen:

7 → 7, 8, 11, 9, 2, 10

```
public static ArrayList<Integer> bfs(Digraph g, int start) {

}
```

Ayudas para el Ejercicio 3

Pista: Consideren el siguiente código:

```
public static boolean hayCaminoDFS(Digraph g, int i, int j) {
}
```

Ayudas para el Ejercicio 4

Pista: Consideren el siguiente código:

```
public static boolean hayCaminoBFS(Digraph g, int i, int j) {
}
```

Ayudas para resolver el Ejercicio 5

Pista: Utilicen DFS y en vez de un arreglo de visitados usen uno de distancias, y vayan mejorando las distancias paso a paso.

```
private static void dfs(Digraph g, int v, int[] costo) {
   // complete...
}
```

Recomendación: Si bien esta es probablemente la forma menos eficiente de encontrar el camino más corto entre dos vértices en un grafo, el siguiente problema puede ser resuelto de esta forma: **http://codeforces.com/problemset/problem/601/A**

Se recomienda que solucionen este problema ya que le dará una mejor comprensión del algoritmo y probará con mayor rigurosidad la exactitud de su solución.

Ayudas para resolver el Ejercicio 6

Pista 1: Nótese que puede asumir que inicia en cualquier vértice, como por ejemplo 0.

Pista 2: Calculen todas las posibles permutaciones (*sin* repetición) y retorne la que tenga menor costo total. Para esto es posible que quieran crear un método que dado un arreglo de enteros *a* elimine el elemento en la posición *k*, al igual que una función auxiliar que solo tenga en cuenta lo vértices sin visitar.

```
private static int[] removeAt(int k, int a[]) {
 // complete...
}

private static int recorrido(Digraph g, int v, int[]
unvisited) {
 // complete...
}
```

.

Pista 3: Recuerden tener en cuenta el costo para volver al vértice inicial.

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

¿Alguna inquietud?

CONTACTO

Docente Mauricio Toro Bermúdez
Teléfono: (+57) (4) 261 95 00 **Ext.** 9473
Correo: mtorobe@eafit.edu.co
Oficina: 19- 627

Agende una cita con él a través de **http://bit.ly/2gzVg10** , en la pestaña *Semana. Si no da clic en esta pestaña, parecerá que toda la agenda estará ocupada.*