ESTRUCTURA DE DATOS 2
Código ST0247

Taller en Sala 7
Algoritmos Voraces en Grafos

Objetivos: **1.** Resolver problemas usando algoritmos de grafos, incluido el problema del camino más corto, y al menos un algoritmo del árbol de expansión de costo mínimo. **2.** Utilizar un algoritmo voraz para resolver un problema apropiado y determinar si la regla voraz conduce a solución óptima o no.

Consideraciones: Lean y verifiquen las consideraciones de entrega,

Trabajo en Parejas

Mañana, plazo de entrega

Docente entrega plantilla de código en GitHub

Sí .cpp, .py o .java

No .zip, .txt, html o .doc

Alumnos entregan código sin comprimir GitHub

En la carpeta Github del curso, hay un código iniciado y un código de pruebas (tests) que pueden explorar para solucionar los ejercicios

Estructura del documento: **a)** Datos de *vida real*, **b)** I*ntroducción* a un problema, **c)** Problema a resolver, **d)** Ayudas. Identifiquen esos elementos así:

a) **b)** **c)** **d)**

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

En la vida real, el algoritmo de Dijkstra se utiliza actualmente para definir protocolos de enrutamiento de redes de computadores como el *Open Shortest Path First (OSPF).* Más información en http://bit.ly/2uhRJw8

Ejercicios a resolver

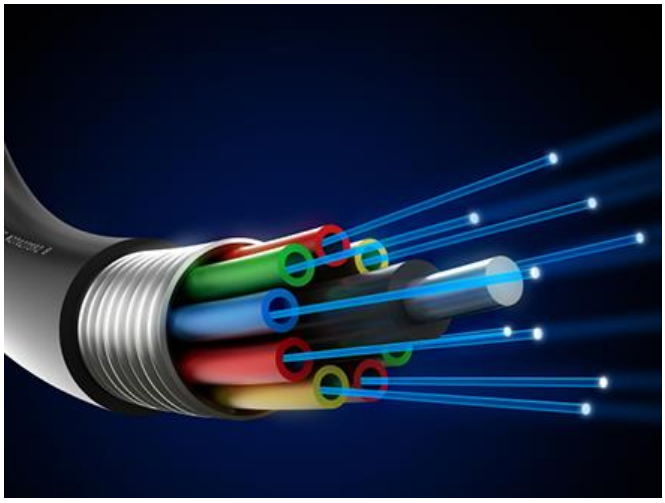
1 En el videojuego *League of Legends*, cuando uno solicita a un personaje desplazarse de un punto a otro, necesita que él tome la ruta más corta y la siga. Un problema que enfrenta *Riot Games* es que no es eficiente resolver este problema con un algoritmo de fuerza bruta o de *backtracking*.

Una aproximación eficiente al problema del camino más corto es el algoritmo de *Dijkstra*. De hecho, la mayoría de los videojuegos utilizan modificaciones del algoritmo de *Dijkstra*, como por ejemplo A*, para calcular las rutas

Implementen el algoritmo de *Dijkstra*, el cual dado un grafo con pesos *no* negativos y un vértice inicial *v*, halla el camino más corto desde *v* hasta todos los demás vértices en el grafo (o determina que no hay camino entre ellos).

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

2 Las empresas Claro y UNE están en proceso de aumentar su cobertura de fibra óptica en Medellín y el resto del área metropolitana. Para lograrlo enfrentan varios problemas. Por un lado, un cable de fibra óptica es muy costoso. Por otro, los costos de instalación del cableado varían.



En este problema, se pueden modelar las conexiones que se desean crear como un grafo y los pesos de las aristas el valor en pesos de cada segmento de cableado. Una vez modelado el problema como un grafo, la solución es encontrar un árbol de expansión con costo mínimo. Un algoritmo para lograr esto es el algoritmo de Prim

Dado un grafo conexo, no dirigido, con pesos, encuentren el costo total del subconjunto de aristas de costo mínimo que conservan el grafo conectado utilizando el algoritmo de Prim. ¿El algoritmo garantiza siempre la solución óptima?

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Ayudas para resolver los Ejercicios

Ejercicio 1

Nota 1: Se sugiere seguir los pasos 1.1 y 1.2 para resolver el ejercicio.

1.1 Implementen el algoritmo utilizando una tabla de distancias mínimas hasta cada uno de los vértices del grafo, y una tabla de padres para que pueda reconstruir el camino desde *v* hasta cada uno de los demás vértices.

```
public static Pair<int[], int[]> dijkstra(Digraph g, int v)
{
      // complete...
}
```

Tengan en cuenta lo siguiente:

- En caso de que no haya forma de llegar a un vértice *s* desde *v*, la distancia hasta este en la tabla debe ser -1
- El par que retorna debe contener el arreglo de distancias y el arreglo de padres en las posiciones *first* y *second*, respectivamente.
- Utilicen valores centinela para denotar distancias infinitas y nodos sin padre. Para este caso en específico, es conveniente considerar como infinita distancia el valor máximo que puede almacenar un entero: *Integer.MAX_VALUE.*
- Les recomendamos que implementen el algoritmo utilizando una cola de prioridad, y meta en esta los nodos en forma de pares de la siguiente manera: {peso, indice}.

Si deciden implementarlo de esta forma, puede utilizar la clase Comparador que le proveemos para que la cola de prioridad vaya soltando los nodos según los pesos (tienen mayor prioridad los que tienen menor peso). Para esto debe crear la cola de prioridad en su método de la siguiente manera:

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
PriorityQueue<Pair<Integer, Integer>> pq = new
PriorityQueue<>(new Comparador());
```

1.2 **Implementen un método que dada la tabla de padres generada por el método que implementó en el numeral 1.1, reconstruya el camino más corto desde un vértice *inicio* hasta un vértice *fin*.**

```
public static ArrayList<Integer> obtenerCamino(int inicio,
int fin, Pair<int[], int[]> par) {
     // complete...
}
```

Tenga en cuenta lo siguiente:

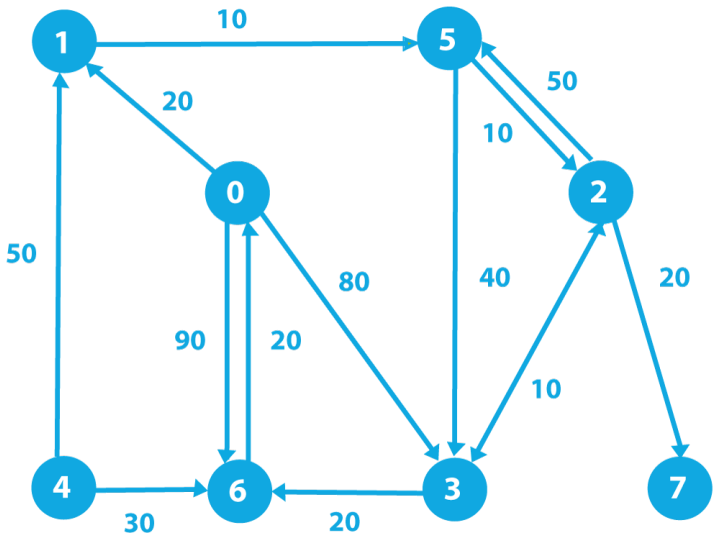
- Asuman que el par que recibe es el retornado luego de llamar el *Dijkstra* desde el vértice inicio.
- En caso de que inicio = fin, retornen null.
- En caso de que no exista camino entre inicio y fin, retornen null.

Nótese que este es el mismo problema del *Taller Nro. 4*, sólo que esta vez utilizamos un algoritmo mucho más eficiente y correcto. De hecho, este es el algoritmo para grafos en general con la menor complejidad asintótica, siempre y cuando el grafo *no* tenga aristas con peso negativo.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Ejercicio 1.2

Ejemplo 1: Consideren el siguiente grafo:



En este grafo, luego de correr el Dijkstra desde 0 la tabla de distancias es [0, 20, 40, 50, -1, 30, 70, 60], y la de padres [-1, 0, 5, 2, -1, 1, 3, 2].

Esto nos permite determinar que, por ejemplo, el camino más corto desde 0 hasta 6 tiene un costo total de 70, y que el camino como tal es 0, 1, 5, 2, 3, 6.

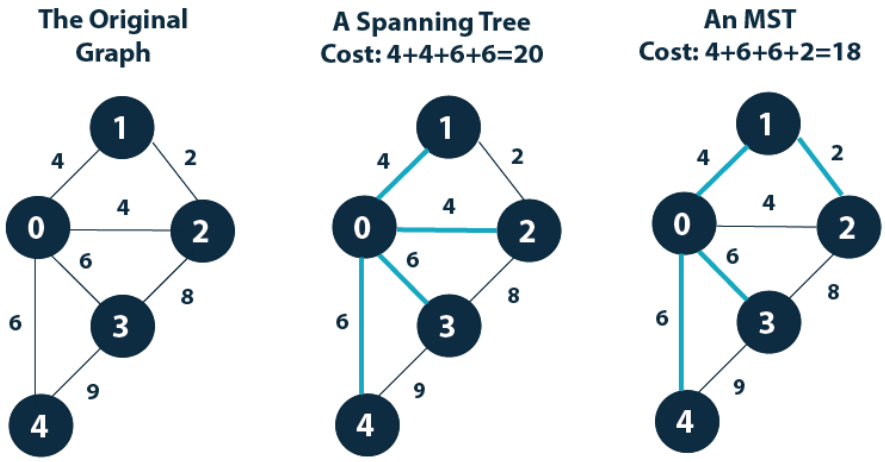
PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Ejercicio 2

Pista 1:

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

Este problema se conoce como el de hallar el Árbol de mínimo costo (*Minimum Spanning Tree*, *MST*), y difiere del problema de los talleres 4 y 6 (que por cierto aún no hemos solucionado de manera exacta) en que en este caso no es un recorrido lo que se busca (mucho menos uno cerrado), sino un árbol.



Nuevamente les recomendados utilizar una cola de prioridad para la implementación (se utilizaría exactamente igual que en el *Dijkstra)*

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

¿Alguna inquietud?

CONTACTO

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

Agenden una cita dando clic en la pestaña -*Semana*- de http://bit.ly/2gzVg10