Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Taller 5: Rutas de costo mínimo en grafos, 2021-30

1 Objetivo

Utilizar la implementación de grafos vista en clase para modelar un problema. En particular, se desea evaluar las habilidades del estudiante en el uso de diferentes algoritmos que permitan responder preguntas particulares de la información estructurada.

2 Recordatorio: compilación con g++

La compilación con g++ (compilador estándar que será usado en este curso para evaluar y calificar las entregas) se realiza con los siguientes pasos:

1. **Compilación**: de todo el código fuente compilable (**ÚNICAMENTE LOS ARCHIVOS CON EXTENSIONES** `*.c`, `*.cpp`, `*.cxx`)
   `g++ -std=c++11 -c *.c *.cxx *.cpp`
2. **Encadenamiento**: de todo el código de bajo nivel en el archivo ejecutable
   `g++ -std=c++11 -o nombre_de_mi_programa *.o`

   Nota: Estos dos pasos (compilación y encadenamiento) pueden abreviarse en un sólo comando:
   `g++ -std=c++11 -o nombre_de_mi_programa *.c *.cxx *.cpp`
3. **Ejecución**: del programa ejecutable anteriormente generado
   `./nombre_de_mi_programa`

ATENCIÓN: Los archivos de encabezados (*.h, *.hpp, *.hxx) **NO SE COMPILAN**, se incluyen en otros archivos (encabezados o código). Así mismo, los archivos de código fuente (*.c, *.cpp, *.cxx) **NO SE INCLUYEN**, se compilan. Si el programa entregado como respuesta a este Taller no atiende estas recomendaciones, automáticamente se calificará la entrega sobre un 25% menos de la calificación máxima.

3 Descripción del problema

En la planeación para la construcción de un nuevo condominio residencial, una empresa de arquitectos utiliza una herramienta muy sencilla: un plano cartesiano. Inicialmente, las casas a construir se ubican dentro del plano cartesiano con puntos (x,y), y una vez están todas ubicadas, se conectan con algunos caminos que van a permitir la movilidad de los residentes dentro del condominio. Para cada camino se almacena su longitud, la cual se calcula sobre el plano como la distancia entre los dos puntos cartesianos que conecta. Dentro de la planeación, se incluye la portería como la primera "casa" del condominio. La Figura 1 representa un ejemplo de como se vería un plano cartesiano con un conjunto de casas ubicadas y conectadas por medio de algunos caminos, en este ejemplo el vértice A1 representa la portería o punto de entrada al condominio.

La información utilizada por los arquitectos para la planeación se almacena en un archivo de texto con la siguiente estructura:

```
N
X(0) Y(0)
X(1) Y(1)
...
X(N-1) Y(N-1)
E
A(0) B(0)
A(1) B(1)
...
A(E-1) B(E-1)
```

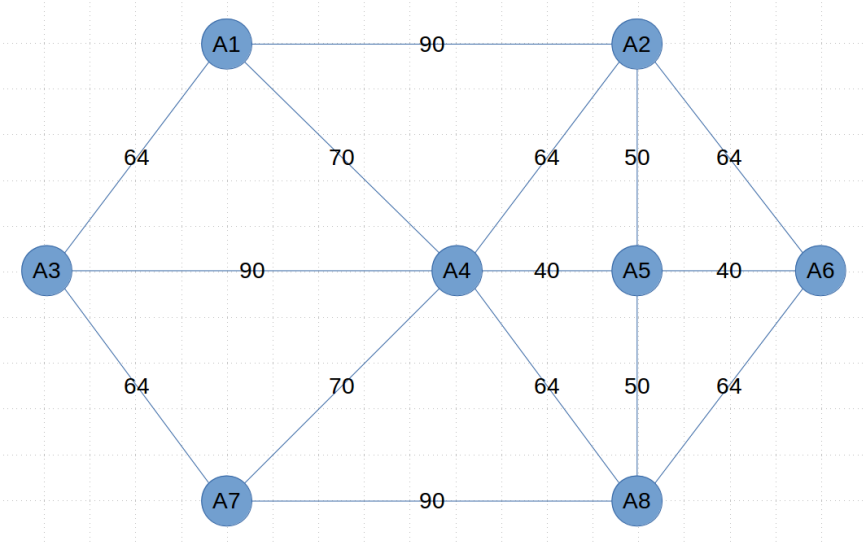

Figure 1: Ejemplo que muestra la ubicación de las casas y los caminos que las conectan.

Donde:

- `N` es un número entero positivo que representa la cantidad de casas a ubicar en el plano.
- `X(i) Y(i)` es la coordenada cartesiana de la i-ésima casa. Cada valor de la coordenada es real.
- `E` es un número entero positivo que representa la cantidad de caminos que conectan las casas en el plano.
- `A(j) B(j)` son los índices de las casas que definen (están conectadas por) el j-ésimo camino. Cada valor de esta tupla es un número entero que representa un índice de una casa en el plano.

Entre las revisiones que deben realizar los arquitectos al momento de la planeación del condominio, una de las más importantes es verificar qué tan separadas quedan las casas y qué tan largos podría llegar a ser los caminos entre ellas, como para tener una idea del tiempo que le tomaría a los residentes recorrer todo el condominio a partir de la portería. Para esto, requieren de un sistema que, sobre la información contenida en los archivos de texto, y asumiendo que la primera casa en el archivo representa la ubicación de la portería, calcule las distancias lineales entre la portería y las casas (en términos de su distancia euclidiana), y luego las compare con las distancias de los caminos que las conectan. De esta forma, el sistema debe generar un informe donde se indique, para cada casa en el condominio, la ruta más corta que la conecta a la portería a través de los caminos, la distancia total a recorrer, y la distancia lineal que existe entre la casa y la portería.

4 Descripción de la implementación

Para resolver el problema de las rutas más cortas entre las casas y la portería del condominio, se propone una implementación contenida en el archivo fuente "`taller_4_grafos.cxx`". El programa recibe como parámetro (por la línea de comandos) el nombre del archivo que contiene la información del condominio: la ubicación de las casas y la definición de los caminos entre ellas. Para identificar las rutas de menor costo, el programa compara los resultados obtenidos con 2 algoritmos diferentes: Prim y Dijkstra. Al ejecutarse, el programa muestra en pantalla las casas con sus coordenadas, la distancia lineal entre cada una y la portería, y para cada algoritmo, la secuencia de caminos interconectados que generan la ruta de menor costo junto con la sumatoria del costo total de la ruta.

5 Desarrollo del taller

El taller consistirá en implementar (o integrar) el código necesario para que el archivo fuente "`taller_4_grafos.cxx`" sirva para crear un programa que imprima por pantalla el informe con los datos relacionados a las rutas de menor costo entre las casas y la portería del condominio.

Para completar el taller, realice las siguientes actividades:

1. Estudie el archivo de código fuente entregado. Asegúrese de entender completamente los procesos allí descritos. Si encuentra alguna porción de código que no entienda, pregunte al profesor o al monitor de la clase.

2. Reemplace las secciones de código marcadas como ‘‘`TODO`’’ con las porciones de código adecuadas para el correcto funcionamiento del programa que se pide:

   - `TODO # 0`: incluir el archivo cabecera con la implementación del TAD Grafo.
   - `TODO # 1`: definir el tipo para un grafo cuyos vértices sean de tipo Punto y los costos sean valores reales.
   - `TODO # 2`: declarar el grafo a usar en el programa.
   - `TODO # 3`: insertar cada uno de los puntos del archivo como vértices del grafo.
   - `TODO # 4`: calcular el costo de cada arista entre los vértices (usando la distancia euclidiana), e insertar la arista con su costo como no dirigida dentro del grafo.
   - `TODO # 5`: calcular las distancias lineales (euclidianas) de cada vértice con respecto al vértice 0 (portería del condominio).
   - `TODO # 6`: aplicar los algoritmos de Prim y de Dijkstra para la búsqueda de rutas de costo mínimo desde el vértice 0 (portería) hasta los demás vértices del grafo.
   - `TODO # 7`: imprimir el informe con los resultados del algoritmo de Prim, que implica que para cada vértice del grafo se indica la secuencia de vértices que generan la ruta de menor costo junto con la sumatoria del costo total de la ruta.
   - `TODO # 8`: imprimir el informe con los resultados del algoritmo de Dijkstra, que implica que para cada vértice del grafo se indica la secuencia de vértices que generan la ruta de menor costo junto con la sumatoria del costo total de la ruta.

6 Evaluación

Como parte del desarrollo del taller, se debe entregar el código fuente modificado y funcional (que compile y ejecute correctamente). Al código fuente modificado del programa, debe adjuntarse la implementación propia del TAD Grafo, para garantizar su correcta ejecución.

La entrega se hará a través de la correspondiente asignación de BrightSpace antes de la medianoche del jueves 18 de noviembre. Se debe entregar un único archivo comprimido **(únicos formatos aceptados: .zip, .tar, .tar.gz, .tar.bz2, .tgz)** que contenga dentro de un mismo directorio **(sin estructura de carpetas interna)** todo el código fuente **(únicos formatos aceptados: .h, .hxx, .hpp, .c, .cxx, .cpp)**. Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

La evaluación del taller tendrá la siguiente escala para cada sección de código a completar:

- **Excelente** (**5.0/5.0**): El código compila, ejecuta y genera resultados correctos para los dos algoritmos (Prim y Dijkstra).
- **Bueno** (**3.0/5.0**): El código compila, ejecuta y genera resultados correctos sólo para uno de los algoritmos (Prim o Dijkstra).
- **No fue un trabajo formal de ingeniería** (**2.5/5.0**): El código compila y ejecuta, pero no genera resultados correctos.
- **Necesita mejoras sustanciales** (**2.0/5.0**): El código compila, pero genera errores de ejecución (violaciones de segmento, por ejemplo), o no puede ejecutarse adecuadamente.
- **Malo** (**1.0/5.0**) : El código entregado por el estudiante no compila en el compilador g++ (mínimo versión número 4.5).
- **No entregó** (**0.0/5.0**).