Proyecto Final

Estructuras de Datos

Departamento de Electrónica y Ciencias de la Computación

Pontificia Universidad Javeriana

Profesor Carlos Alberto Ramírez Restrepo
carlosalbertoramirez@javerianacali.edu.co

El presente proyecto tiene como objetivo enfrentar a los estudiantes del curso:

- Al diseño e implementación de Tipos Abstractos de Datos.
- Al análisis de la complejidad computacional de la implementación de operaciones primitivas de un Tipo Abstractos de Datos.
- Al análisis y comparación en términos de complejidad computacional de diferentes estrategias de representación para un mismo Tipo Abstracto de Datos.
- Al diseño e implementación de operaciones generales sobre un Tipo Abstracto de Datos.
- Al uso de la biblioteca STL del lenguaje de programación C++ y al análisis de sus ventajas en términos de eficiencia, practicidad y generalidad con respecto a estructuras y funciones definidas directamente.

1 Generalidades sobre Grafos

Un *grafo* es un tipo abstracto de datos que agrupa un conjunto de elementos llamados *vértices* o *nodos*. Estos elementos están unidos por lineas llamadas *arcos* o *aristas* las cuales permiten representar relaciones entre los vértices. De esta manera, los grafos en ciencias de la computación y matemáticas se utilizan para representar sistemas donde sus elementos se interrelacionan arbitrariamente. Son múltiples las áreas donde son utilizados los grafos, las cuales incluyen matemáticas discretas, optimización, procesamiento de lenguaje natural y computación teórica, entre otras.

Más formalmente, un grafo G es una pareja $\mathrm{G} = \mathrm{V}, \mathrm{E}$ donde:

- V es un conjunto finito de vértices y
- E es un conjunto finito de aristas que relacionan los elementos en V. De esta manera se tiene que:

$$\mathrm{E} = \{(a, b) \in \mathrm{E} \times \mathrm{E}\}$$

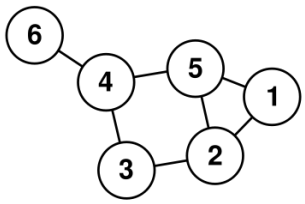

Figure 1: Ejemplo de Grafo

La Figura 1 muestra un grafo en el cuál se tiene que:

$$\text{V} = \{1, 2, 3, 4, 5, 6\}$$

$$\text{E} = \{(1, 2), (1, 5), (2, 3), (2, 5), (5, 4), (4, 3), (4, 6)\}$$

En adición a los vértices y aristas, en algunos grafos también se dispone de una *función de ponderación* que asigna un valor numérico (su *peso* o *longitud*) a cada arista. Considere por ejemplo, el grafo de la Figura 2, en el cual se tiene que:

$$\text{V} = \{a, b, c, d, e\}$$

$$\text{E} = \{(a, b), (a, c), (b, c), (b, d), (c, d), (d, e), (c, e)\}$$

Además, se tiene la siguiente función de ponderación:

$$f((a, b)) = 1$$
$$f((a, c)) = 3$$
$$f((b, c)) = 5$$
$$f((b, d)) = 10$$
$$f((c, d)) = 9$$
$$f((c, e)) = 7$$
$$f((d, e)) = 4$$

Cabe resaltar que la función de ponderación en un grafo puede asignar una arista cualquier valor incluyendo valores negativos.

Caminos en un Grafo. Un *camino* entre dos nodos a y b en un grafo corresponde a una secuencia de nodos $v_1, \ldots, v_m$ tales que $v_1 = a$, $v_m = b$ y existe un arista que conecta cada vértice v_i con el vértice v_{i+1}. De esta manera, para el grafo de la Figura 1 se tiene que la secuencia de nodos 1, 5, 4, 6 es un camino entre el nodo 1 y el nodo 6 ya

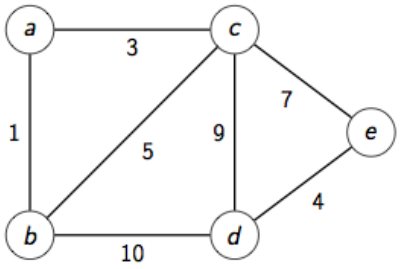

Figure 2: Ejemplo de Grafo Ponderado

que existen aristas que conectan los nodos 1 y 5, 5 y 4 y 4 y 6. En tanto que, la secuencia B, A, C, D no es un camino entre los nodos B y D puesto que no existe una arista entre el nodo B y el nodo A (si la hay entre A y B) ni entre los nodo C y D.

Para el caso de los grafos etiquetados o ponderados, se tiene la noción de *peso* o *costo* de un camino. El costo de un camino corresponde a la suma de las ponderaciones o costos de las aristas incluidas en el camino. Considere por ejemplo, el grafo ponderado de la Figura 3. El costo del camino A, B, C, E, D es 14 puesto que corresponde a la suma de los costos de las aristas que conectan al nodo A con el nodo B (costo 5), a B con C (costo 3), a C con E (costo 4) y a E con D (costo 2).

Grafos Dirigidos. Un *grafo dirigido* es un tipo especial de grafo en el cual cada arista tiene una dirección. De esta manera, la relación entre vértices es no simétrica. Dada una arista (a, b) se dice que a es el *nodo inicial* y b es el *nodo final* y dicha arista es diferente de la arista (b, a) a pesar de que relacione los mismos nodos.

Considere por ejemplo, el grafo dirigido de la Figura 3, en el cual se tiene que:

$$\text{V} = \{A, B, C, D, E\}$$

$$\text{E} = \{(A, B), (B, C), (C, A), (C, E), (B, D), (D, E), (E, D)\}$$

Además, se tiene la siguiente función de ponderación:

$$f((A, B)) = 5$$
$$f((B, C)) = 3$$
$$f((C, A)) = 3$$

$$f((C,E)) = 4$$
$$f((B,D)) = 1$$
$$f((D,E)) = 4$$
$$f((E,D)) = 2$$

Tenga en cuenta que dado que es un grafo dirigido el vértice A está directamente comunicado con el vértice B pero no viceversa. Algo similar ocurre con las demás parejas de vértices. Se dice que un nodo a es *adyacente* a un nodo b si hay una arista que comunica a a con b (en esa dirección).

2 Representaciones

Como se mencionó anteriormente, los grafos son ampliamente utilizados en computación. Por esta razón, es necesario representar computacionalmente los grafos. Existen algunas estrategias de representación de grafos, de las cuales las más usadas son: las *listas de adyacencia* y la *matriz de adyacencia*. En las siguientes secciones se describirán estos enfoques. Cada una de las representaciones será ilustrada utilizando como referencia los grafos de las Figuras 1, 2 y 3.

2.1 Listas de Adyacencia

Para representar un grafo G $= (\mathrm{V}, \mathrm{E})$ mediante listas de adyacencia se utiliza un vector, arreglo o lista de tamaño $n = |\mathrm{V}|$ donde cada elemento es una lista (o una referencia a una lista) que contiene los vértices o nodos adyacentes a cada vértice. De esta manera, un grafo como el de la Figura 1, se representaría como la lista `L` como sigue:

```
L = [[2, 5], [1, 3, 5], [2, 4], [3, 5, 6],
     [1, 2, 4], [4]]
```

De esta manera en la posición 0 de la lista `L` se tiene la lista `[2, 5]` que quiere decir que el nodo 1 está conectado a los vértices 2 y 5. Así mismo, `L[1] = [1, 3, 5]` puesto que el nodo 2 está conectado a los vértices 1, 3 y 5.

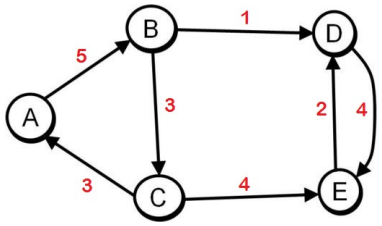

Figure 3: Ejemplo de Grafo Dirigido

Por otro lado, en el caso de los grafos ponderados o etiquetados, se tienen dos posibilidades equivalente de representación mediante listas de adyacencia. Por ejemplo, el grafo de la Figura 2 se representaría como una lista `M` como sigue:

```
L = [[(c, 3), (b, 1)],
     [(a, 1), (c, 5), (d, 10)],
     [(a, 3), (b, 5), (d, 9), (e, 7)],
     [(b, 10), (c, 9), (e, 4)],
     [(c, 7), (d, 4)]]
```

Así, se tiene por ejemplo que `L[0] = [(c, 3), (b, 1)]` quiere decir que el primer nodo (nodo a) está conectado al nodo c con una arista de peso 3, mientras que está conectado al nodo b con una arista de peso 1.

Otra posibilidad para representar el grafo de la Figura 2 con listas de adyacencia es utilizar dos listas `M1` y `M2` como sigue:

```
M1 = [[c, b], [a, c, d], [a, b, d, e],
      [b, c, e], [c, d]]
M2 = [[3, 1], [1, 5, 10], [3, 5, 9, 7],
      [10, 9, 4], [7, 4]]
```

En donde se tiene por ejemplo que `M1[2] = [a, b, d, e]` ya que el nodo c está conectado a los nodos a, b, d y e. Además, `M2[2] = [3, 5, 9, 7]` ya que los pesos de las conexiones del nodo c son 3, 5, 9 y 7 respectivamente.

Finalmente, el grafo de la Figura 3 se puede representar mediante las listas `T1` y `T2` como sigue:

```
T1 = [[B], [C, D], [A, E], [E], [D]]
```

```
T2 = [[5], [3, 1], [3, 4], [4], [2]]
```

Además, es importante resaltar que se podría representar como se hizo anteriormente para el primer grafo.

2.2 Matriz de Adyacencia

Otra estrategia para representar un grafo corresponde a la matriz de adyacencia. Dado un grafo G $= (\mathrm{V}, \mathrm{E})$, se utilizará una matriz de dimensiones $n \times n$ donde $n = |\mathrm{V}|$. De esta manera, el indice de cada fila representará el vértice desde donde sale una arista y el índice de cada columna el vértice al que llega una arista. En el caso de los grafos no ponderados, cada posición (i, j) en la matriz será 1 o 0 dependiendo de si hay o no una conexión entre los vértices i y j. Mientras que en el caso de los grafos ponderados, cada posición (i, j) en la matriz será el peso de la arista que conecta los vértices i y j en caso de existir o tendrá un valor `inf` en caso contrario.

Un grafo como el de la Figura 1 se representaría con una matriz `Mat1` como sigue:

```
Mat1 = [[0, 1, 0, 0, 1, 0],
        [1, 0, 1, 0, 1, 0],
        [0, 1, 0, 1, 0, 0],
        [0, 0, 1, 0, 1, 1],
        [1, 1, 0, 1, 0, 0],
        [0, 0, 0, 1, 0, 0]]
```

Como dicho grafo es no ponderado, la matriz `Mat1` solo tiene unos o ceros. De esta manera, por ejemplo `Mat1[0][1] = 1` puesto que el primer nodo (nodo 1 ubicado en el índice 0 de la matriz) está conectado con el segundo nodo (nodo 2 ubicado en el índice 1 de la matriz). En tanto que, `Mat1[5][2] = 0` porque el nodo 6 (ubicado en el índice 5 de la matriz) no está conectado con el nodo 3 (ubicado en el índice 2 de la matriz.

Finalmente, existen algunos enfoques en los cuales se representa un grafo mediante una combinación de los dos enfoques anteriores. Se utilizan listas de adyacencia para representar de una forma explícita las conexiones entre nodos y una matriz de adyacencia para representar los costos de cada arista.

3 Recorridos sobre Grafos

Cuando se recorre una estructura de datos lineal como una lista, un arreglo o un vector, se realiza un desplazamiento entre sus elementos con el fin de ejecutar alguna acción como imprimir o desarrollar alguna operación aritmética. Similarmente, para resolver problemas que involucren grafos es necesario recorrerlos, esto es, desplazarse por cada uno de los nodos para obtener su valor y realizar alguna operación. Dado que los grafos no se pueden recorrer secuencialmente, existen múltiples estrategias para recorrerlos entre ellas se encuentran la búsqueda en profundidad y la búsqueda en amplitud. Ambos recorridos serán considerados en el presente proyecto y serán descritos en las próximas secciones.

3.1 DFS - Depth First Search

La *búsqueda primero en profundidad* o DFS (del inglés *Depth First Search*) es un algoritmo para recorrer árboles y grafos en el cúal inicialmente se selecciona un nodo `raiz` de forma arbitraria o con algún tipo de criterio que puede depender del contexto mismo donde se utilice el grafo. Intuitivamente, el algoritmo de búsqueda en profundidad intenta desplazar por las aristas hasta encontrar un nodo que no tenga nodos adyacentes o hasta llegar a un nodo por el que ya había pasado. Cuando esto ocurre se intenta explorar un camino diferente al original siguiendo la misma estrategia.

Sea un grafo G $= (\mathrm{V}, \mathrm{E})$ y un nodo `raiz`, el algoritmo se puede expresar de la siguiente manera:

```
// inicialmente todos los nodos estan
// sin visitar
DFS(G, raiz):
   marcar raiz como visitado
   para cada nodo v adyacente a raiz:
      si v no esta visitado:
        hacer DFS(G, v)
```

Para el grafo de la Figura 1, si se selecciona el nodo 1 como la raíz, el algoritmo revisará los nodos en el siguiente orden: 1, 2, 3, 4, 6, 5. Mientras que si selecciona el nodo 6 como el nodo raíz, el algoritmo revisará los nodos en el siguiente orden: 6, 4, 3, 2, 1, 5. Así mismo, para el

grafo de la Figura 3, si se selecciona el nodo A como el nodo raíz, el algoritmo revisará los nodos en el siguiente orden: A, B, C, E, D. En tanto que, si la raíz es el nodo B, el recorrido será el siguiente: B, C, A, E, D.

Un aspecto importante a resaltar del algoritmo es que para evitar que hayan ciclos infinitos se utiliza algún tipo de estructura que permita determinar si un nodo ya ha sido visitado previamente.

3.2 BFS - Breadth First Search

La *búsqueda primero en amplitud* o BFS (del inglés *Breadth First Search*) es un algoritmo de recorridos de grafos o árboles en el cual también se selecciona inicialmente un nodo `raiz`. Intuitivamente, el algoritmo de búsqueda en amplitud primero visita el nodo `raiz` y luego visita de forma secuencial cada uno de los nodos adyacentes al nodo raiz. Este proceso es realizado para nodo del grafo y se utiliza una cola como estructura auxiliar.

Sea un grafo $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ y un nodo `raiz`, el algoritmo se puede expresar de la siguiente manera:

```
// inicialmente todos los nodos estan
// sin visitar
BFS(G, raiz):
   sea una cola Q
   colocar raiz en Q
   mientras la cola no este vacia:
     extraer el tope de Q en z
     marcar z como visitado
     para cada nodo v adyacente a z:
        si v no esta visitado:
          colocar v en Q
```

Para el grafo de la Figura 1, si la raíz es el nodo 1, luego la búsqueda en amplitud recorrerá los elementos así: 1, 2, 5, 3, 4 y 6. Mientras que si la raíz es el nodo 4, el recorrido será así: 4, 3, 5, 6, 2 y 1. De la misma manera, para el grafo de la Figura 3, si la raíz es el nodo A, luego el recorrido en la búsqueda por amplitud será: A, B, C, D y E. Además, si la raíz del grafo es B, luego el recorrido será: B, C, D, A y E.

Como en la búsqueda por profundidad, para evitar que hayan ciclos infinitos se utiliza algún tipo de estructura (lista, arreglo, vector, etc.) que permita determinar si un nodo ya ha sido visitado previamente.

4 Algoritmo de Djikstra

El algoritmo **Dijkstra** es un algoritmo que opera sobre grafos y que permite encontrar el camino más corto desde un nodo (origen) hacía múltiples nodos (destinos). Se sugiere revisar el siguiente enlace:

`https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra`

y la Sección 24.3 del texto *Introduction to Algorithms* de Cormen, Leiserson, Rivest y Stein para mayor información sobre este algoritmo. Además, el profesor destinará un espacio de clase para explicar las generalidades del funcionamiento de este algoritmo.

En esencia, este algoritmo realiza un recorrido sobre el grafo y utiliza una cola de prioridad en el proceso de calcular los caminos más cortos. La complejidad del algoritmo está asociada a la forma en que es implementada la cola de prioridad.

5 Operaciones

Si se plantea un grafo como un TAD se identifican las siguientes operaciones:

Crear Grafo Vacío: Esta operación permite crear una instancia de un grafo $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ que inicialmente no tiene vértices ni aristas, i.e., $\mathrm{V} = \emptyset$ y $\mathrm{E} = \emptyset$.

Crear Grafo No Vacío: Esta operación permite crear una instancia de un grafo $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ a partir de algunos nodos y aristas que son suministrados como parámetros.

Agregar Arista: Esta operacion permite agregar una nueva arista a un grafo $\mathrm{G} = (\mathrm{V}, \mathrm{E})$. Tenga en cuenta que esta operación se debe realizar con respecto a nodos a y b existentes en el grafo, i.e., $a, b \in \mathrm{V}$.

Agregar Nodo: Esta operacion permite agregar un nuevo nodo a un grafo $\mathrm{G} = (\mathrm{V}, \mathrm{E})$.

Modificar Arista: Esta operación permite modificar la ponderación de una arista de un grafo $G = (V, E)$. Tenga en cuenta que esta opercion solo aplica para grafos ponderados.

Eliminar Arista: Esta operación permite eliminar una arista de un grafo $G = (V, E)$.

Eliminar Nodo: Esta operación permite eliminar un nodo de un grafo $G = (V, E)$ siempre y cuando no haya ninguna arista asociada a dicho nodo.

Obtener Lista de Adyacencia de un Nodo: Esta operación permite obtener la lista de nodos adyacentes a un nodo dado como parámetro.

Obtener Número de Vertices: Esta operación permite obtener el número de vértices de un grafo $G = (V, E)$.

Obtener Número de Nodos: Esta operación permite obtener el número de nodos de un grafo $G = (V, E)$.

6 Entrega

En el presente proyecto se espera que usted implemente el TAD grafo como una clase en C++ utilizando y explotando las características de la POO. Además, usted debe crear una clase llamada `OperacionesGrafo` en la que hayan funciones que permitan hacer los recorridos en profundidad y en amplitud de un grafo y que implementen el algoritmo de Dijsktra. Más especificamente, cada grupo de trabajo debe:

1. Implementar el TAD grafo como una clase en C++ que utilice la representación como listas de adyacencia o como la matriz de adyacencia.
2. Incluir en cada implementación todas las operaciones descritas en este documento.
3. Analizar la complejidad de cada operación. No es necesario que se analice detalladamente cada linea de cada operación pero se debe explicar claramente la complejidad que se reporte para cada una de ellas.
4. Implementar la clase `OperacionesGrafo` con las operaciones mencionadas y que utilicen apropiadamente estructuras como listas, pilas, colas y colas de prioridad.
5. Explicar claramente las decisiones de implementación realizadas y probar cada una de las operaciones implementadas.

7 Informe

El informe debe entregarse en un archivo **pdf** y debe contener las explicaciones de las diferentes decisiones que se tomen en la implementación y las conclusiones del proyecto. Más especificamente, el informe debe contener:

- Detalles principales de las implementaciones del TAD grafo y de la clase `OperacionesGrafo`.
- Análisis de la complejidad de estas implementaciones.
- Conclusiones y aspectos a mejorar. Siendo esta una de las partes más interesantes del trabajo, usted debe **analizar los resultados obtenidos** y **justificar** cada una de sus afirmaciones.

Aclaraciones

1. El proyecto se debe realizar en **grupos de 3** personas. Cualquier indicio de copia causará la anulación del proyecto y la ejecución del proceso disciplinario que corresponda de acuerdo a la normativa de la universidad.
2. Para la entrega debe generar un archivo `.zip` o `.tar.gz` con el contenido del proyecto (código fuente, documentación del proyecto y ayudas) el cual debe seguir la convención *Apellido1Apellido2Apellido3ProyectoEstr17.zip*. Deben incluir un archivo llamado `informe.pdf`, correspondiente al informe del proyecto.

   El proyecto debe ser realizado en LaTeXy tener una estructura apropiada y buena redacción. Estos elementos serán evaluados.
3. Los criterios de evaluación del proyecto son los siguientes:
   - Implementación TAD Grafo: 25%

- Implementación Búsqueda en Profundidad: 13%
- Implementación Búsqueda en Amplitud: 14%
- Implementación Algoritmo de Dijkstra: 18%
- Correcta Utilización POO: 5%
- Informe y sustentación: 25%

4. El proyecto tendrá dos entregas. La primera entrega debe contener la implementación del TAD Grafo y debe ser enviada a través de la plataforma *moodle* a más tardar el día **11 de Noviembre a las 10:00pm**.

   La segunda entrega debe contener todos los elementos adicionales además de posibles mejoras que haya identificado sobre la primera entrega. La entrega final debe hacerla a través de la plataforma *moodle* a más tardar el día **30 de Noviembre a las 9:00am**. Dependiendo de la disposición de todo el grupo, es posible dejar la entrega final para el día **1 de Diciembre a las 2:00pm**. El mismo día de la entrega se realizará la sustentación del proyecto.

   La nota final del proyecto será individual y dependerá de la sustentación de cada estudiante. Cada estudiante, después de la sustentación tendrá asignado un número real (el factor de multiplicación) entre 0 y 1, correspondiente al grado de calidad de su sustentación. Su nota definitiva será la nota del proyecto, multiplicada por ese valor. Si su asignación es 1, su nota será la del proyecto. Pero si su asignación es 0.9, su nota será 0.9 por la nota del proyecto. La no asistencia a la sustentación tendrá como resultado una asignación de un factor de 0.

   Tenga muy en cuenta está aclaración. El propósito de la sustentación es que usted demuestre su participación en el proyecto. Por esta razón trabaje a conciencia y prepare muy bien su sustentación.

5. Recuerde, en caso de dudas y aclaraciones puede preguntar al inicio de las clases, asistir al horario de monitoria o enviar un correo con su consulta al profesor.