Tarea 5
Implementación de Estructuras de Datos
Curso 2020

Índice

1. Introducción

El objetivo de esta tarea es la aplicación de los TADs desarrollados en las tareas anteriores.
En particular se implementará el TAD *Grafo* y algunos de sus algoritmos.
De manera informal un grafo es un conjunto de elementos, llamados vértices, tal que entre cada par de vértices puede haber o no un vínculo, llamado arista. Si entre dos vértices hay una arista se dice que ambos son adyacentes. Por ejemplo los vértices pueden representar alumnos y hay una arista entre un par de alumnos si son compañeros en algún curso. En otro ejemplo los vértices representan ciudades y hay una arista entre dos ciudades si hay un tramo de carretera que las une directamente (o sea, sin necesidad de pasar por una ciudad intermedia). Las aristas pueden tener un valor (costo, peso) y si es así se dice que el grafo es ponderado. En el ejemplo de las ciudades el valor de una arista puede ser la longitud del tramo de carretera de las ciudades que une.
En esta tarea el grafo tiene N vértices, identificados del 1 al N y hasta M aristas, siendo N y M parámetros pasados al crear el grafo. Las aristas tienen un valor de tipo `double`. Dado un vértice v al conjunto de sus vértices adyacentes se les llama los vecinos de v.

Nota A cada par de vértices (v_1, v_j) se le puede hacer corresponder un número que lo identifica. Una forma, entre otras, es el número

$$(\text{mín}(v_i, v_j) - 1) \times N + (\text{máx}(v_i, v_j) - 1).$$

Se debe notar que al par (v_i, v_j) le corresponde el mismo número que al par (v_j, v_i). En cambio a cualquier otro par en el que al menos uno de los componentes es distinto a v_i y a v_j le corresponde un número diferente.

2. Materiales

Los materiales para realizar esta tarea se extraen de *MaterialesTarea5.tar.gz*. Ver el procedimiento en la Sección **Materiales** de Funcionamiento y Reglamento del Laboratorio.

En esta tarea los archivos en el directorio `include` son los de la tarea anterior a los que se agrega **grafo.h**.

En el directorio `src` se incluyen ya implementados **utils.cpp** e **info.cpp**.

3. ¿Qué se pide?

Ver la Sección **Desarrollo** de Funcionamiento y Reglamento del Laboratorio.

En esta tarea se deben implementar los mismos archivos que en la tarea anterior y además `grafo.cpp`.
Se debe tener en cuenta que en `usoTads.h` se agregaron nuevas funciones.
En algunas operaciones se piden requerimientos de tiempo de ejecución. Ese tiempo es el del peor caso, a menos que se especifique otra cosa de manera explícita.

4. Descripción de los módulos y algunas funciones

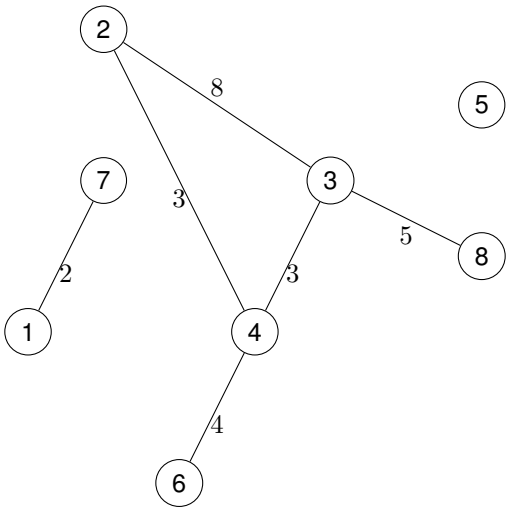

Figura 1: Ejemplo para `Accesibles` y `LongitudesCaminosMasCortos`

4.1. Accesibles

```
/*
  Devuelve un arreglo que indica a cuales vértices se puede llegar desde 'v'.
  El arreglo devuelto tiene tamaño N+1. La coordenada 0 no se tiene en cuenta.
  La coordenada 'i' (1 <= i <= N) es 'true' si y solo si el vértice 'i' es
  accesible desde 'v'.
  Precondición: 1 <= v <= N.

  Seudocódigo:
   - Inicializar cada coordenada del arreglo a devolver con 'false'
     (se considera cada vértice no accesible).
   - Pasar 'v' como parámetro a una función auxiliar. Esta función
     -- marca el parámetro como accesible desde 'v'.
     -- para cada vecino de su parámetro que todavía sea no accesible
        desde 'v' hace una llamada recursiva.
   - Devolver el arreglo.
 */
ArregloBools accesibles(nat v, TGrafo g);
```

- El resultado de invocar `Accesibles` desde **5** es:

False	False	False	False	True	False	False	False
1	2	3	4	5	6	7	8

  Al hacer la llamada con **5** como parámetro, la coordenada 5 del arreglo queda en true. Como el vértice **5** no tiene ningún vecino no se producen más cambios

- El resultado de invocar `Accesibles` desde **3** es:

False	True	True	True	False	True	False	True
1	2	3	4	5	6	7	8

  Se hace la llamada con **3** como parámetro por lo que la coordenada 3 queda en `True`. Los vecinos del vértice **3** son **2**, **4** y **8**.

Como el vértice **2** todavía se considera no accesible se hace la llamada con **2** como parámetro. La coordenada 2 queda en `True`. Los vecinos de **2** son **3**, que ya se sabe que es accesible, y **4**.

Se hace la llamada con **4** como parámetro. La coordenada 4 queda en `True`. Los vecinos de **4** son **2**, **3**, que ya se sabe que son accesibles, y **6**.

Se hace la llamada con **6** como parámetro. La coordenada 6 queda en `True`. El único vecino de **6** es **4** que ya se sabe que es accesible.

Ahora el vértice **4** ya es accesible.

El vértice **8** todavía no se considera accesible por lo que se hace la llamada con **8** como parámetro. La coordenada 8 queda en `True`. El único vecino de **8** es **3**, que ya se sabe que es accesible.

4.2. LongitudesCaminosMasCortos

```
/*
  Devuelve un arreglo con las longitudes de los caminos más cortos desde 'v'
  hasta cada vértice del grafo.
  El arreglo devuelto tiene tamaño N+1. La coordenada 0 no se tiene en cuenta.
  En la coordenada 'i' (1 <= i <= N) se devuelve la longitud del camino más
  corto desde 'v' hasta 'i'. Si 'i' no es accesible desde 'v' el valor de esa
  coordenada es DBL_MAX (definido en float.h)
  Precondición: 1 <= v <= N.

  Seudocódigo:
   - Crear colecciones 'C' y 'S', inicialmente vacías, de pares (u,du)
     donde 'u' representa un vértice y 'du' es la longitud del camino más
     corto desde 'v' hasta 'u'. El valor de 'du' en 'C' es provisorio
     mientras que en 'S' es definitivo.
   - Insertar (v,0) en 'C'.
   . Mientras 'C' no es vacía:
     -- se obtiene y remueve de 'C' el par (u, du) tal que 'du' es mínimo
        entre todos los pares de 'C'.
     -- se inserta ese par en 'S'.
     -- para cada vecino 'w' de 'u' que no está en S sea dw' = du + d(u,w),
        donde d(u,v) es la distancia entre 'u' y 'w'.
        Si 'w' no está en 'C' se inserta (w,dw') en 'C'.
        Si 'w' está en 'C' en el par (w,dw) y dw' < dw entoces se actualiza
        'C' con el par con (w,dw') en lugar de (w,dw).
   - Para cada vétice 'u' que no pertenece a 'S' se inserta en 'S' el par
     (u, infinito).
   - Devolver 'S'.

 */
ArregloDoubles longitudesCaminosMasCortos(nat v, TGrafo g);
```

El resultado de invocar `longitudesCaminosMasCortos` desde **3** es:

M	6	0	3	M	7	M	5
1	2	3	4	5	6	7	8

en donde M representa el máximo valor que se le puede asignar a un elemento de tipo `double`.

Inicialmente C y S están vacías.

Se agrega $(3, 0)$ a C (la distancia para llegar desde **3** hasta **3** es 0).

Se obtiene y extrae el único elemento de C, que es $(3, 0)$, y se lo incluye en S. Esto determina que en el arreglo resultado el valor de la coordenada 3 sea 0.

Los vecinos de **3** son **2**, **4** y **8** y ninguno de ellos está en C ni en S. Entonces la longitud del camino más corto conocida hasta este momento para llegar hasta **2** es la suma del valor de la arista que une **2** con **3**, que es 8, más la longitud del camino más corto para llegar a **3**, que es 0. Por lo tanto se incluye $(2, 8)$ en C. De manera análoga se incluyen $(4, 3)$ y $(8, 5)$ en C.

Entre los vértices que están en C el que tiene un camino de longitud más corta para llegar al vértice **3** es el **4**. Por lo tanto se obtiene y extrae $(4, 3)$ de C y se lo incluye en S, por lo que el valor de la coordenada 4 del arreglo es 3. Los vecinos de **4** son **2**, **3** y **6**.

> El vértice **3** ya está en S, ya se determinó la longitud del camino más corto, por lo que no se hace nada con él.
>
> El vértice **6** no está en C ni en S y ahora se conoce un camino para llegar a él. La longitud de ese camino es la suma del valor de la arista que une **4** con **6**, que es 4, más la longitud del camino más corto para llegar a **4**, que es 3. Por lo tanto se incluye $(6, 7)$ en C.
>
> El vértice **2** ya está en C con un posible camino de longitud igual a 8. Pero ahora se sabe que se puede llagar a **2** a través del vértice **4** por un camino con una longitud que es la suma del valor de la arista que une a **2** y **4**, que es 3, más la longitud del camino más corto para llegar a **4**, que es 3. Entonces se sabe que hay un camino desde **3** hasta **2** de longitud 6, que es menor a la que se conocía hasta el momento. Por lo tanto se actualiza $(2, 6)$ en C.

Se obtiene y extrae $(8, 5)$ de C y se lo incluye en S. Esto implica que el valor de la coordenada 8 sea 5. Como **3** es el único vecino de 8 y ya está en S no se hace más nada con este vértice.

Se obtiene y extrae $(2, 6)$ de C y se lo incluye en S por lo que el valor de la coordenada 2 es 6. Los vecinos de **2** ya están en S.

Se obtiene y extrae $(6, 7)$ de C y se lo incluye en S por lo que el valor de la coordenada 6 es 7. Los vecinos de **6** ya están en S.

La colección C quedó vacía. Esto implica que el valor de las coordenadas 1, 5 y 7 quede en M.

5. Entrega

Se mantienen las consideraciones reglamentarias y de procedimiento de las tareas anteriores.
La tarea otorga hasta 6 puntos. La adjudicación de los puntos de las tareas de laboratorio queda condicionada a la respuesta correcta de una pregunta sobre el laboratorio en el segundo parcial.

Se debe entregar el siguiente archivo, que contiene los módulos implementados `avl.cpp`, `binario.cpp`, `cadena.cpp`, `colaAvls.cpp`, `colaDePrioridad.cpp`, `conjunto.cpp`, `iterador.cpp`, `grafo.cpp`, `mapping.cpp`, `pila.cpp`, `usoTads.cpp`:

- **Entrega5.tar.gz**

Este archivo se obtiene al ejecutar la regla `entrega` del archivo *Makefile*:

```
$ make entrega
tar zcvf Entrega5.tar.gz -C src cadena.cpp usoTads.cpp binario.cpp iterador.cpp pila.cpp colaAvls.cp
cadena.cpp
usoTads.cpp
binario.cpp
iterador.cpp
pila.cpp
colaAvls.cpp
avl.cpp
```

```
conjunto.cpp
colaDePrioridad.cpp
mapping.cpp
grafo.cpp
```

Con esto se empaquetan los módulos implementados y se los comprime.
Nota: En la estructura del archivo de entrega los módulos implementados deben quedar en la raíz, NO en el directorio `src` (y por ese motivo se usa la opción `-C src` en el comando `tar`).

5.1. Plazos de entrega

El plazo para la entrega es el **miércoles 8 de julio a las 14 horas**.

5.2. Identificación de los archivos de las entregas

Cada uno de los archivos a entregar debe contener, en la primera línea del archivo, un comentario con el número de cédula del estudiante, **sin el guión y sin dígito de verificación**. Ejemplo:

```
/* 1234567  */
```

5.3. Individualidad

Ver la Sección **Individualidad** de Funcionamiento y Reglamento del Laboratorio.