Universidad Tecnológica Nacional – Facultad Córdoba
Carrera: Ingeniería en Sistemas de Información
Cátedra: DLC

Parcial Único 2021 – Versión Completa

Consignas: Diseñar e implementar mejoras en el Algoritmo de Prim

[PREVIEW]
Hemos visto que el Algoritmo de Prim permite calcular un Árbol de Expansión Mínimo (AEM) para un grafo no dirigido, ponderado y conexo. Pero si este algoritmo se aplica sobre un grafo no conexo solo calculará el AEM para la componente conexa en la que se haya posicionado al inicio del proceso.

Por otra parte, en la implementación del algoritmo que hemos visto en clases existen algunos elementos que podrían ser mejorados desde el punto de vista del tiempo de ejecución (aunque quizás empleando algo más de memoria).

Su tarea consiste en diseñar e implementar variantes y mejoras para el Algoritmo de Prim, en base al modelo que se entregó en clases. Para ello se sugiere implementar, en la clase UnDirectedGraph, un nuevo método:

```
public long getMSTValue_Prim_NEW()
```

tal que contenga todas las variantes que se solicita diseñar e implementar. Los estudiantes podrán, además, introducir las variantes que crean necesarias en las clases del modelo, pero siempre garantizando que esos cambios sean contemplados también en el resto de los métodos del modelo (en otras palabras: si se hace un cambio en algún atributo/método de alguna clase, o se agrega algún atributo/método, los métodos de esa clase deben seguir funcionando y no es válido simplemente eliminar los métodos que se vean afectados por el cambio).

Los aspectos del Algoritmo de Prim que deberá rediseñar son los siguientes:

1. El algoritmo modificado debe poder calcular un AEM para cada componente conexa del grafo incluso si el grafo no es conexo (al estilo de lo que efectivamente hace el Algoritmo de Kruskal). El método debe seguir retornando la suma de los pesos del AEM calculado, al igual que el método original.
2. En el modelo original visto en clases, se usa una lista *x* para ir almacenando los vértices que ya están en el AEM parcial. Y en la línea 108 del código fuente se aplica una validación para verificar que los vértices *n1* y *n2* no estén ambos ya contenidos en *x*. Pero el control de pertenencia a *x* se hace con el método *contains()* de la clase *LinkedList*, que realiza una búsqueda secuencial con su consecuente rendimiento lineal. Diseñe una mejora que reduzca este rendimiento lineal al hacer esta validación. La mejora puede (obviamente) incluir la idea de que *x* no sea una *LinkedList* sino otra estructura a elección del estudiante, o cualquier otra idea que se considere adecuada para bajar el tiempo de comprobación.
3. Por supuesto, en cualquier lugar del algoritmo original en el que se use *contains()* para controlar si un vértice pertenece a *x* (o cualquier otro proceso que implique el recorrido de *x*), debe ser válida la mejora propuesta en el punto anterior.

Cada alumno debe estar presente a través de la sala de clase a distancia el día estipulado para el parcial, para completar el programa. En ese momento se plantearán nuevos requerimientos para que sean programados en forma individual.

Al terminar, cada alumno debe subir individualmente su proyecto comprimido, empleando el link que figura para esta misma actividad.

Consignas a resolver con ingreso a la sala de clases a distancia, en el día y hora del parcial:

La consigna a resolver en forma individual (SIN compartir nada de lo que hagan desde aquí y en adelante, y SIN trabajar en grupos) es la siguiente:

a. Modifique el método `get_MSTValue_Prim_NEW()` que se le pidió implementar en el preview, pero de forma tal que ahora *no retorne* un *long* con la suma de los pesos del árbol de expansión mínimo (AEM) calculado, sino que retorne el *UndirectedGraph* constituido por el AEM que calculó. No se maree: el AEM de un grafo *g1* es OTRO grafo *g2*, que contiene todos los vértices de *g1* y solamente la mínima cantidad de arcos de *g1* que se necesitan para que *g2* siga conexo pero sin formar ciclos (o en su defecto, para que cada componente conexa de *g1* siga siendo conexa en *g2* y no forme ciclos), pero tal que la suma de esos arcos sea mínima. Su tarea, esencialmente, es armar y retornar el grafo *g2* a partir del AEM que el método **ya calculó** para *g1* (que a los efectos prácticos, es el objeto *this*…), en lugar de retornar la suma de sus pesos.

b. Una vez hecha esa modificación, diseñe un *main()* de prueba en el que se creen un par de grafos con vértices y arcos fijos (no los cargue por teclado, simplemente asígnelos) y se pruebe el método, mostrando la conversión a String de cada uno de estos grafos originales y de los AEM calculados. Asegúrese de que uno de los grafos testeados sea conexo, y de que el segundo no lo sea. Asegúrese también de incluir al menos 8 vértices en cada uno de los grafos orinales, y al menos 12 arcos en cada uno.