EDA (ETS d'Enginyeria Informàtica). Curs 2020-2021

Pràctica 6: Una aplicació de l'algorisme de Kruskal a la vida real: disseny de la línia elèctrica entre ciutats

Sesió 2: Obtenció d'un Arbre de Recobrimient Mínim

Departament de Sistemes Informàtics i Computació. Universitat Politècnica de València

1. Obectius

Per a complir amb els seus objectius generals, en finalitzar la segona sessió d'aquesta pràctica l'alumne haurà de ser capaç d'implementar l'algorisme de Kruskal de manera eficient, i.e. reutilitzant la Jerarquia Java `UFSet`.

2. Descripció del problema

Com es va indicar en la primera sessió de la pràctica, el conjunt d'arestes que defineix un Arbre de Recobriment d'un graf No Dirigit i Connex és només una solució factible al problema de connectar amb el menor cost possible els seus `N` vèrtexs mitjançant `N-1` arestes. La solució òptima a aquest problema passa per trobar un conjunt de `N-1` arestes del graf tal que la suma dels seus pesos siga mínima o, equivalentment, que definisquen un Arbre de Recobriment Mínim (ARM) del graf. Encara que poden existir diversos conjunts solució per a un mateix graf, l'algorisme de Kruskal garanteix obtindre un d'ells emprant una estratègia molt intuitiva :

> Processar en ordre creixent de pesos, una a una, les arestes del graf (arestes factibles), incloent en el conjunt solució cada aresta que no forme un cicle amb les ja incloses en ell -perquè un Arbre de Recobriment és Acíclic per definició.

El procés descrit acabarà, bé quan ja s'han seleccionat les `N-1` arestes que defineixen un ARM del graf, perquè és Connex, bé quan ja no queda cap aresta factible a processar i encara no s'han seleccionat `N-1` arestes, perquè el graf no és Connex.

El següent esquema algorítmic resumeix el procés d'obtenció "a la Kruskal" del conjunt d'arestes que defineixen un ARM d'un graf No Dirigit, si és que existeix. En ell s'introdueix la notació que s'emprarà en la resta de la secció; a saber: `E` denota el conjunt d'arestes del graf i `aristasFactibles` el de les arestes del graf encara per processar; amb el parell `(v, w)` es nota l'aresta que connecta els vèrtexs `v` i `w` del graf, amb `|E|` el número de les seues arestes i amb `E'` el conjunt d'arestes que defineixen un de les seues ARM, `null` si el graf no és Connex; finalment, s'empra el símbol ∅ per a denotar el conjunt buit.

```
E' = ∅; cardinalE' = 0;
arestesFatibles = E;

mentres (cardinalE'  <  N - 1 && arestesFatibles !=  ∅):
   (v, w) = eliminarMin(arestesFatibles);
   Si ((v, w) NO forma cicle amb les arestes d' E'):
       E' = E' UNIO (v, w);
       cardinalE'++;
   FinSi
FinMentres

Si (cardinalE' == N - 1): solució = E'; Sino: solució = null; FinS
```

Note's que una implementació eficient de l'algorisme presentat requereix que, en cada pas del procés iteratiu que descriu, siga possible. . .

- Obtindre i eliminar el mínim de `aristasFactibles` (`eliminarMin`) de la forma més eficient possible, i.e. en O(log`|E|`). Una manera d'aconseguir-ho és representar el conjunt `aristasFactibles` com una Cua de Prioritat implementada, per exemple, mitjançant un `MonticuloBinarioR0`, classe disponible en el paquet *jerarquicos* del projecte *BlueJ eda*.

- Comprovar si l'aresta `(v, w)` forma cicle amb les arestes de `E'` de la forma més eficient possible, i.e. en aproximadament O(1). Atés que una aresta NO forma cicle si els vèrtexs dels seus extrems estan en diferents components connexes, una manera d'aconseguir-ho és representar les components connexes del graf definit per `E'` mitjançant un *UF-Set* `cc` de talla `N`. En efecte:
  - Inicialment, `E'` és un conjunt buit d'arestes que defineix un graf de `N` vèrtexs aïllats; per tant, el *UF-Set* `cc` està compost per `N` components connexes o `N` vèrtexs aïllats, cadascun en la seua pròpia component connexa.
  - En cada iteració, per a comprovar si l'aresta `(v, w)` extreta de `aristasFactibles` forma cicle amb les de `E'` s'han de determinar primer, via operació `find` del *UF-Set*, les components connexes a les quals pertanyen `v` i `w`: `int ccV = cc.find(v); int ccW = cc.find(w);` Fet això, si `v` i `w` estan en diferents components, i.e. si `ccV != ccW`, s'inclou l'aresta `(v, w)` en el conjunt solució `E'` i s'actualitzen les components connexes de `E'` via operació `union` del *UF-Set* (`cc.union(ccV, ccW)`).
  - Finalment, si el graf definit per les arestes de `E'` és Connex, la iteració acaba quan la talla de `E'` és `N -1`.

Usant les EDAs indicades, la implementació de l'algorisme de Kruskal obté un ARM d'un graf en O(`|E|` log`|E|`), exactament el mateix cost asimptòtic que requereix processar en ordre creixent les 2`|E|` arestes que pot contindre la Cua de Prioritat `aristasFactibles` en el Pitjor dels Casos. Per a ser més concrets, el fet que el cost dominant siga el de les operacions de la Cua de Prioritat i no el de les del *UF-Set* es deu exclusivament a l'ús d'una implementació "en Bosc" eficient del *UF-Set*: en el Pitjor dels Casos, comprovar si 2`|E|` arestes formen cicle suposa realitzar en temps constant `N-1` operacions `union` i 2`|E|` operacions `find`, i.e. realitzar O(`N` + `|E|`) operacions.

3. Activitats

Abans de realitzar les activitats que es proposen en aquesta sessió, l'alumne ha d'actualitzar diversos paquets del seu projecte *BlueJ eda* seguint els passos que s'indiquen a continuació. Tots els fitxers esmentats en ells estan disponibles en *PoliformaT* i s'han de descarregar en les carpetes corresponents al paquet del mateix nom del seu projecte *eda*.

- Descarregar en la carpeta *modelos* el fitxer `UFSet.java`.
- Descarregar en la carpeta *jerarquicos* el fitxer `ForestUFSet.class`, que conté una implementació eficient de la interficie `UFSet`.
- Descarregar en la carpeta *grafos* el fitxer `TestKruskal.class`.
- Obrir el projecte *BlueJ eda* i compilar la classe `UFSet` del paquet *librerias.estructurasDeDatos.modelos*. En acabar, eixir de *BlueJ*.
- Invocar de nou al *BlueJ* i accedir al paquet *librerias.estructurasDeDatos.grafos*

3.1. Actualitzar la classe `Arista` i implementar el mètode `kruskal` de la classe `Grafo`

En aquesta activitat l'alumne ha de completar el codi del mètode `kruskal` de la classe `Graf` usant l'algorisme homònim descrit en l'apartat 2 d'aquest butlletí. En concret, per a tindre en compte les consideracions realitzades en aquest apartat sobre la implementació eficient del algortimo de Kruskal, l'alumne deu. . .

- Usar les classes `ColaPrioridad` i `MonticuloBinarioR0` per a implementar la cua de prioritat `aristasFactibles`.
- Usar les classes `UF-Set` i `ForestUFSet` per a implementar el *UF-Set* `cc`.
- Incloure les directives `import` que apareixen comentades en la classe `Grafo` per a poder reutilitzar els modeos i implementacions Java de cua de prioritat i *UF-Set*.
- Modificar el codi de la classe `Arista` per a que implemente la interficie `Comparable`, perquè `aristasFactibles` és una cua de prioritat de `Aristas`.

3.2. Validar el codi desenvolupat en la práctica

Per a comprovar la correcció del codi implementat durant la sessió l'alumne ha d'executar el programa `TestKruskal`.