Exercici 2: Arbre de recobriment de cost mínim

- Partim d'un graf connex, ponderat i no dirigit $G = (V, A, p)$ d'arcs no negatius, i volem trobar l'arbre de recobriment de G de cost mínim.
- Per arbre de recobriment d'un graf G entenem un subgraf sense cicles que continga tots els seus vèrtexs. En cas d'haver-hi diversos arbres de cost mínim, ens quedarem d'entre ells amb el que posseïsca menys arcs.
- Hi ha almenys dos algorismes molt coneguts que resolen aquest problema, com són el de Prim i el de Kruskal. En tots dos es va construint l'arbre per etapes.
- La manera en què es realitza aquesta elecció és el que distingeix un algorisme d'altre.
- Es pot usar per determinar com utilitzar el mínim de conductor per connectar $|V|$ punts elèctricament equivalents en un circuit integrat (el pes de cada arc és la distància entre punts).

Siga $p(i, j)$ el pes de l'arc (i, j). El que volem és construir un arbre T, és a dir, un subconjunt $T \subseteq E$, tal que

$$P(T) = \sum_{(i,j)\in T} p(i, j)$$

siga mínim, és a dir, busquem

$$T = \arg\min_{T\subseteq E} P(T) = \arg\min_{T\subseteq E} \sum_{(i,j)\in T} p(i, j)$$

Algorisme de Prim

En l'algorisme de Prim,[1] la solució `solucio` en construcció és sempre un únic arbre (podria ser un *bosc* o conjunt d'arbres). Comença amb un vèrtex qualsevol, busca l'arc de menor pes que ix d'aquest vèrtex, i hi va afegint a cada pas un arc lleuger que connecta un node de l'arbre actual amb un node que encara no hi ha estat afegit fins que es cobreixen tots els vèrtexs. Per a això, ha de portar compte de quins vèrtexs hi ha ja en l'arbre.

L'algorisme 1 mostra una versió similar a la que expliquen Ferri et al. (1998). No demostrarem aquí que l'algorisme de Prim troba un arbre de recobriment de pes mínim (es pot demostrar fàcilment).

Tal com està escrit, el cost temporal de l'algorisme està dominat pel bucle implícit en la línia 8 (de cost $O(|V|^2)$). Una manera de reduir el cost és anar guardant per a cada vèrtex de $V - B$ el vèrtex més pròxim en B (per exemple, després d'actualitzar B), per no haver de calcular el vèrtex de pes mínim ($\arg\min$).

[1] Realment anterior: V. Jarník (1930) "O jistém problému minimálním" [Sobre un cert problema mínim], *Práce Moravské Přírodovědecké Společnosti*, **6**:57–63.

Algorithm 1 Algorisme de Prim bàsic

```
1: Dades: G = (V, E, p)                          ▷ Graf no dirigit, connex i amb pesos
2: Resultats: T ⊆ E                              ▷ L'arbre de recobriment mínim
3: Auxiliar: B ⊆ V                               ▷ Conjunt de vèrtexs usats
4: T ← ∅                                         ▷ Inicialitza arbre buit
5: v ← elementAleatori(V)                        ▷ Agarrem un vèrtex qualsevol
6: B ← {v}                                       ▷ Comencem amb aquest vèrtex
7: while B ≠ V do                                ▷ Mentre no s'hagen usat tots els vèrtexs
8:     (u, v) ← arg min_{(u',v')∈(V−B)×B} p(u, v) ▷ Atenció al cost d'aquest bucle
9:     T ← T ∪ {(u, v)}                          ▷ Afegim l'arc més lleuger
10:    B ← B ∪ {u}                               ▷ I el nou vèrtex u als usats
11: end while
```

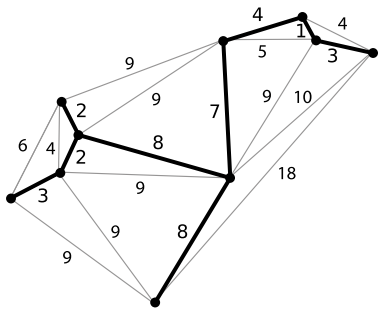

Figura 1: Exemple d'arbre de recobriment d'un graf no dirigit

Algorisme de Kruskal

L'algorisme de Kruskal (algorisme 2) va construint, arc a arc, un *bosc* (un conjunt d'arbres) que finalment es fusiona en un únic arbre.

Algorithm 2 Algorisme de Kruskal (bàsic)

```
1:  Dades: G = (V, E, p)                          ▷ Graf ponderat, no dirigit amb pesos
2:  Resultats: T ⊆ E                              ▷ L'arbre de recobriment mínim
3:  Auxiliar: F ⊆ E                               ▷ Conjunt dels arcs candidats
4:  T ← ∅                                         ▷ Comencem amb un bosc buit
5:  E' ← ordenarPesosCreixent(E)                  ▷ Ordenem els vèrtexs
6:  while nombredarcs(T) < |V| − 1 ∧ E ≠ ∅ do
7:                                                ▷ |V| vèrtexs → |V| − 1 arcs
8:      (u, v) = primer(E')                       ▷ El millor vèrtex
9:      E' ← E' − {(u, v)}                        ▷ Se n'elimina
10:     if nocreacicle((u, v), T) then            ▷ Ha de connectar dos arbres existents
11:         T ← T ∪ {(u, v)}                      ▷ L'hi afegim
12:     end if
13: end while
14: if |T| = |V| − 1 then                         ▷ Existeix arbre de recobriment
15:     return T
16: else
17:     return ∅
18: end if
```

Com diuen Horowitz i Sahni (1978, p. 181), que descriuen una versió similar de l'algorisme, "per a poder realitzar eficientment el pas [10], els vèrtexs de G s'han d'agrupar de manera que un puga determinar fàcilment si els vèrtexs u i v estan ja connectats per algun dels arcs seleccionats anteriorment. Una possibilitat és col·locar tots els vèrtexs de cada component connectat de T en un conjunt (tots els components connectats de T seran també arbres)". Hi podeu trobar una demostració en les pp. 182 i 183.

De fet, hi ha una estructura de dades molt útil anomenada *conjunts disjunts*[2] que inspira una implementació molt més eficient de l'algorisme (algorisme 3). Ací, com que les arestes ja estan ordenades (línia 5, cost $O(|E| \log |E|)$, de fet $O(|E| \log |V|)$, ja que $|E| \leq |V|^2$), quan es junten dos arbres del bosc es fa sempre usant l'aresta de menor pes.

[2] Aquesta estructura de dades té dues operacions bàsiques, *find*, que diu en quin subconjunt disjunt és un element del conjunt gran, i *union*, que uneix dos subconjunts disjunts en un.

Algorithm 3 Algorisme de Kruskal (millorat)

```
1: Dades: G = (V, E, p)                          ▷ Graf ponderat, no dirigit amb pesos
2: Resultats: T ⊆ E                              ▷ L'arbre de recobriment mínim
3: Auxiliar: F ⊆ E                               ▷ Conjunt dels arcs candidats
4: T ← ∅                                         ▷ Comencem amb un bosc buit
5: E' ← ordenarPesosCreixent(E)                  ▷ Ordenem els vèrtexs
6: for v ∈ V do
7:     creaconjunt(v)                            ▷ Crea un conjunt per a cada vèrtex
8: end for
9: for (u, v) ∈ E' do                            ▷ En ordre creixent de p(u, v)
10:    if trobaconjunt(u) ≠ trobaconjunt(v) then ▷ Si els dos vèrtexs estan
11:                                              ▷ en conjunts diferents
12:        T ← T ∪ {(u, v)}                      ▷ Afegim l'aresta a l'arbre
13:        unioconjunts(u, v)                    ▷ Unim els conjunts (arbres) on estan u i v.
14:    end if
15: end for
16: if |T| = |V| − 1 then                        ▷ Existeix arbre de recobriment
17:    return T
18: else
19:    return ∅
20: end if
```