Algo L3 Info
Travaux dirigés, séance 12.1

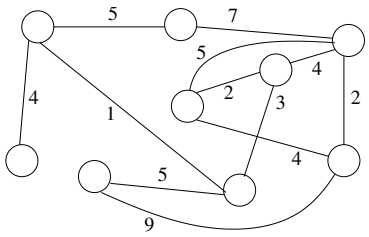
UFR IM²AG
UNIVERSITÉ Grenoble Alpes

Arbre couvrant de poids minimum(ACPM)
suite

Soit G un graphe connexe dont les arêtes sont pondérées par des valeurs positives.

Algorithme de Kruskal

C'est un algorithme glouton. Il construit une solution à partir du graphe $(X, \emptyset)$ en ajoutant des arêtes reliant des sommets de composantes connexes différentes. les arêtes sont examinées par poids croissant. Le graphe en construction est une forêt, et la preuve de correction de l'algo consiste à montrer que cette forêt est un sous-graphe d'un ACPM de G.



Principe de l'algorithme :
Initialiser le graphe Sol à $(X, \emptyset)$.
En parcourant les arêtes (x, y) par poids croissant :
Si x et y sont dans des composantes connexes différentes de Sol alors
Ajouter (x, y) au graphe Sol

Preuve : illustrez la preuve suivante à l'aide de dessins.
Montrons donc que la propriété "Sol est sous-graphe d'un ACPM de G" est un invariant de la boucle de l'algorithme.
Supposons que Sol soit sous-graphe d'un ACPM T, puis que l'on ajoute l'arête (x, y) à Sol.

- si (x, y) est une arête de T, alors Sol reste sous-graphe de T.

- si (x, y) n'est pas une arête de T, alors dans T, il y a un chemin (unique) entre x et y, qui forme un cycle avec l'arête (x, y).
  Le long de ce chemin, puisque x et y ne sont pas (encore) dans la même composante connexe de Sol, il y a des arêtes qui ne sont pas dans Sol. Soit (u, v) l'une de ces arêtes.
  Puisque Sol est sous-graphe de l'arbre T, u et v ne sont pas dans la même composante connexe de Sol : il y aurait sinon un chemin entre u et v dans Sol, qui formerait un cycle avec (u, v), et ce cycle serait sous-graphe de T.
  D'autre part, le poids de (u, v) ne saurait être strictement inférieur à celui de (x, y) : en effet, l'algorithme aurait choisi d'insérer cette arête dans Sol précédemment, sinon. Donc le poids de (x, y) est inférieur ou égal à celui de (u, v).
  Considérons alors l'arbre T' obtenu en remplaçant dans T l'arête (u, v) par l'arête (x, y). (c'est bien un arbre, connexe et $n - 1$ arêtes). Le poids de T' est inférieur ou égal à celui de T, et il est couvrant.
  Donc, c'est un arbre de mêne poids que T, c'est un ACPM dont la solution en construction est sous-graphe.

D'autre part, cet invariant est vrai avant le début de la boucle.
Par conséquent, à la sortie de la boucle, Sol est toujours un sous-graphe d'un ACPM. De plus, il n'y a plus d'arête reliant des sommets dans des composantes connexes différentes, donc Sol est connexe.
(Sol est sous-graphe d'un ACPM) et (Sol est connexe) $\Rightarrow$ Sol est un ACPM.

Mise en oeuvre
Remarque : on peut s'arrêter dès que $n-1$ arêtes ont été ajoutées (si n est le nombre de sommets).
Comment répondre efficacement à la question : x et y sont-ils dans la même composante connexe ? Il n'est pas nécessaire de construire explicitement le graphe Sol autrement que sous la forme d'une liste d'arêtes. Il suffit d'utiliser une structure Union-Find performante. L'essentiel du "travail" de l'algorithme est donc le tri des arêtes. Et donc l'algo est en $m\log(m)$ (m étant le nombre d'arêtes).

Réécriture de l'algorithme
Tri des arêtes par poids croissant.
Initialiser le graphe Sol à $(X, \emptyset)$.
Initialiser la partition des sommets par les singletons.
En parcourant les arêtes (x, y) par poids croissant et tant que Sol comporte moins de $n-1$ arêtes :
    Si Find(x) $\neq$ Find(y) alors
        Ajouter (x, y) au graphe Sol
        Union(x, y).

Algorithme de Prim

C'est aussi un algorithme glouton. Il construit une solution à partir du graphe $(\{x_0\}, \emptyset)$ où x_0 est un sommet quelconque de X. En notant S l'ensemble des sommets de la solution en construction, l'algorithme ajoute au graphe en construction une arête (x, y) avec $x \in S$ et $y \in X - S$ qui minimise le poids des arêtes reliant S à $X - S$, et bien sûr ajoute y à S.

Principe de l'algorithme
Initialiser le graphe $Sol = (S, E)$: $S = \{x_0\}$ et $E = \emptyset$.
Répéter n-1 fois :
    Soit (x, y) de poids minimum parmi les arêtes (u, v) telles que $u \in S$ et $v \in X - S$
    Ajouter y à S et (x, y) à E.

Preuve
En vous inspirant de la preuve de l'algorithme de Kruskal, rédigez la preuve de correction de cet algorithme en montrant que Sol est toujours un sous-arbre d'un ACPM.

Mise en oeuvre :
Proposez une solution pour mettre en oeuvre cet algorithme :
- Quelles sont les informations à maintenir au cours de l'exécution de l'algorithme ?
- Quelle(s) structure(s) de données est(sont) adaptée(s) ?
- Quel est le coût de l'algorithme ?
- Réécrivez l'algorithme de Prim en faisant apparaître le schéma de mise en oeuvre.