

Objectifs

- À la fin de cette séance, vous devriez être capable de :
- reconnaître un algorithme glouton
 - déterminer s'il est ou non optimal
 - calculer sa complexité

Nota Bene. Commencez par terminer les exercices du TD précédent si nécessaire, ils sont plus abordables que le problème présenté ici.

Exercice 1 : Algorithme de Kruskal

Rappels On se place dans un graphe non orienté dont les arêtes sont pondérées (autrement dit on dispose d'une fonction $poids : R \rightarrow \mathbb{R}$).

Un **arbre** est un graphe connexe et sans cycle. On dispose de diverses caractérisations équivalentes à cette définition, en termes de nombre d'arêtes et/ou de graphe « connexe minimal » ou « acyclique maximal » (voir le cours si besoin).

On appelle **forêt** un graphe sans cycle, autrement dit un graphe dont chaque composante connexe est un arbre sur l'ensemble de ses sommets.

On appelle **arbre couvrant** de G un sous-graphe de G qui est un arbre, et dont l'ensemble des sommets est X . L'objectif de ce TD est d'étudier un algorithme de construction d'un **arbre couvrant de poids minimum** (ACPM) de G , c'est-à-dire un arbre couvrant de G dont la somme des poids des arêtes soit minimal.

L'algorithme de Kruskal

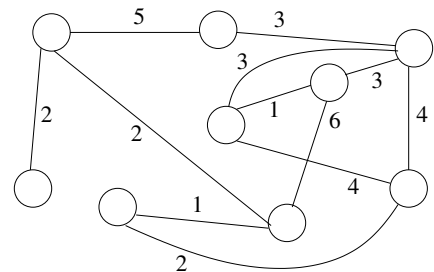
Voici le principe général de l'algorithme de Kruskal :

Initialiser le graphe Sol à (X, \emptyset) .

En parcourant les arêtes (x, y) de G par poids croissant :

Si x et y sont dans des composantes connexes différentes alors
Ajouter (x, y) au graphe Sol

1. Faire tourner cet algorithme sur le graphe ci-contre.
2. Déterminer un critère d'arrêt efficace pour cet algorithme.
3. *Si vous avez déjà fait le TD2 cette semaine :* comment pouvez-vous utiliser la structure de données **Union-Find** pour répondre efficacement à la question : x et y sont-ils dans la même composante connexe ?
Écrire un algorithme détaillé utilisant notamment cette structure.



Correction de l'algorithme

1. Que contient Sol durant l'exécution de l'algorithme ?
2. Démontrer qu'à toute étape de l'algorithme, Sol est un sous-graphe d'un arbre couvrant de poids minimal.

Complexité Pour simplifier les calculs, on considère dans cette partie que le test « x et y sont dans des composantes connexes différentes » a une complexité en $\mathcal{O}(1)$ (grâce à Union-Find vue en TD2).

1. Évaluer le coût des différentes étapes de l'algorithme de Kruskal, en n'oubliant pas les étapes « préliminaires » qui pourraient être coûteuses.

2. En déduire la complexité de cet algorithme.

Correction de l'exercice 1

L'algorithme de Kruskal

1. L'algorithme commence par ajouter successivement au graphe Sol toutes les arêtes de poids 1, 2 et 3.

Ensuite, deux des deux arêtes de poids 4 sont ajoutées (en fonction de l'ordre retenu lors du tri par poids croissant). En revanche, la troisième est rejetée car elle créerait un cycle (avec les deux arêtes de poids 2 et une autre arête de poids 4).

Enfin on ajoute dans Sol les deux arêtes de poids 5, et comme ce graphe est alors connexe plus aucune autre arête n'est ajoutée.
2. Un arbre à n sommets comporte toujours $n - 1$ arêtes, il faut donc s'arrêter dès qu'on a ajouté $n - 1$ arêtes à Sol (ce qui ne demande que de compter les arêtes, pas de tester la connexité du graphe).
3. On utilise la structure Union-Find vue en TD2.

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 $\text{Find}(x) \neq \text{Find}(y)$ alors
 Ajouter (x, y) au graphe Sol
 Union(x, y).

Correction

1. Sol est une forêt (i.e. un graphe acyclique ou une réunion d'arbres disjoints).
2. Cet algorithme construit une solution à partir du graphe (X, \emptyset) en ajoutant peu à peu des arêtes (sans jamais en enlever). Les arêtes sont examinées selon deux critères simples : par poids croissant et en testant si elles relient des sommets de composantes connexes différentes.

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 même 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.

$(Sol \text{ est sous-graphe d'un ACPM}) \text{ et } (Sol \text{ est connexe}) \Rightarrow Sol \text{ est un ACPM.}$

Complexité

1. La première heuristique (qui consiste à attacher le plus petit arbre sous le plus grand lors d'une union) garantit un coût au pire logarithmique (dans le nombre d'éléments de chaque composante connexe) pour les deux opérations Union et Find.

(Si en plus on compresse les chemins lors de chaque opération Find, alors on peut espérer un coût quasi constant.)

2. Notons m le nombre d'arêtes et n le nombre de sommets.

La boucle qui teste les arêtes du graphe et les insère ou non dans Sol s'exécute au plus m fois, et effectue au pire deux Find et une Union. Son coût est donc au pire de $m \log n$ (voire m avec la version la plus efficace de Union-Find).

L'essentiel du « travail » de l'algorithme est donc le tri des arêtes, qui est fait préalablement à cette boucle.

L'algorithme de Kruskal est donc en $m \log(m)$.