UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

LICENCE Informatique
Info0501 – 2018/2019

Travaux dirigés n° 5

Graphes : arbres couvrants de poids minimal

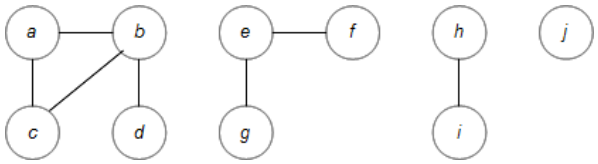
Exercice 1 (Préliminaire : gestion des composantes connexes d'un graphe)
L'algorithme de Kruskal, servant à déterminer un arbre couvrant de poids minimal dans un graphe, est basé sur la gestion des composantes connexes de ce graphe. Cette dernière peut se faire efficacement à l'aide d'une structure de données pour ensembles disjoints.

Une structure de données d'ensembles disjoints gère une collection $S = \{S_1, S_2, ..., S_k\}$ d'ensembles dynamiques disjoints. Chaque ensemble est identifié par un **représentant**, qui est un certain membre de l'ensemble. Chaque élément d'un ensemble étant représenté par un objet x, on souhaite disposer des opérations suivantes :

- `creer-ensemble(x)` : créé un nouvel ensemble dont le seul membre (et donc le représentant) est x. Comme les ensembles sont disjoints, il faut que x ne soit pas déjà membre d'un autre ensemble.
- `union(x,y)` : réunit les ensembles qui contiennent x et y, disons S_x et S_y, dans un nouvel ensemble qui est l'union de ces deux ensembles. Les deux ensembles sont supposés être disjoints avant l'opération. Le représentant de l'ensemble résultant est un membre quelconque de $S_x \cup S_y$. Après l'union, S_x et S_y n'existent plus dans la collection S.
- `trouver-ensemble(x)` : retourne un pointeur vers le représentant de l'ensemble (unique) contenant x.

Pour l'instant, vous pouvez supposer que ces trois opérations sont définies et les utiliser pour répondre aux questions suivantes.

L'une des nombreuses applications des structures de données d'ensembles disjoints apparaît lorsqu'il s'agit de déterminer les composantes connexes d'un graphe non orienté. Pour ce faire, une façon de procéder est de placer initialement chaque sommet v dans son propre ensemble. Ensuite, Pour chaque arête (u, v), on réunit les ensembles contenant u et v. Quand toutes les arêtes ont été traitées, deux sommets se trouvent dans la même composante connexe si et seulement si les objets correspondants se trouvent dans le même ensemble. Soit le graphe et le tableau ci-dessous :



Arêtes traitées	Collection d'ensembles disjoints
ensembles initiaux	{a} {b} {c} {d} {e} {f} {g} {h} {i} {j}
(b,d)	
(e,g)	
(a,c)	
(h,i)	
(a,b)	
(e,f)	
(b,c)	

1°) Complétez le tableau en donnant l'état de la collection d'ensembles disjoints après le traitement de chaque arête.

2°) Définissez les procédures suivantes :

a) `composantes-connexes` : utilise les opérations d'ensembles disjoints pour calculer les composantes connexes d'un graphe, tel que défini précédemment ;

b) `meme-composante` : indique, après un prétraitement du graphe par `composantes-connexes`, si deux sommets se trouvent dans la même composante connexe.

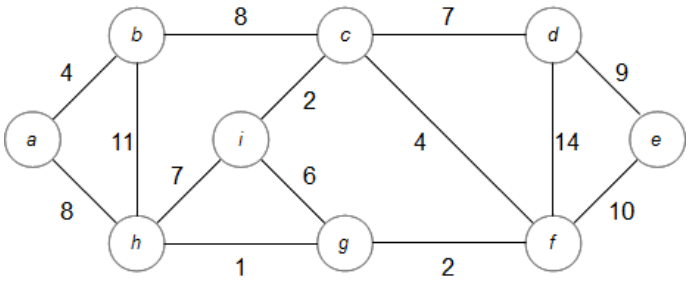
Une façon simple de mettre en oeuvre une structure de données d'ensemble disjoints est de représenter chaque ensemble par sa propre liste chaînée. L'objet associé à chaque ensemble a les attributs *tete*, pointant vers le premier objet de la liste, et *queue*, pointant vers le dernier objet. Chaque objet de la liste contient un membre de l'ensemble, un pointeur vers l'objet suivant dans la liste et un pointeur vers l'objet ensemble. Dans chaque liste chaînée, les objets peuvent figurer dans n'importe quel ordre. Le représentant est le membre de l'ensemble contenu dans le premier objet de la liste.

3°) Illustrez cette représentation avec les ensembles $S_1 = \{d, f, g\}$ avec f comme représentant, et $S_2 = \{b, c, e, h\}$ avec c comme représentant. Illustrez ensuite le résultat de `union(g,e)`.

4°) Définissez les procédures `creer-ensemble`, `union` et `trouver-ensemble` utilisant la représentation par listes chaînées. Donnez aussi leur temps d'exécution.

5°) Quand les arêtes du graphe sont "statiques" (jamais modifiées au cours du temps), les composantes connexes peuvent être calculées plus rapidement par une recherche "en profondeur d'abord". Montrez qu'un parcours en profondeur d'un graphe non orienté G peut servir à identifier les composantes connexes de G et que la forêt de parcours en profondeur contient autant d'arbres que le graphe a de composantes connexes. Plus précisément, montrez comment modifier la procédure `parcours-profondeur` vue au TD précédent de façon qu'il assigne à chaque sommet v une étiquette $v.cc$ comprise entre 1 et k, où k est le nombre de composantes connexes de G, telle que $u.cc = v.cc$ si et seulement si u et v appartiennet à la même composante connexe.

Exercice 2 (Algorithme de Kruskal)
Soit le graphe non orienté, connexe et pondéré suivant :



1°) Donnez un exemple d'arbre couvrant pour ce graphe.

L'algorithme de Kruskal est basé sur la gestion des composantes connexes d'un graphe. Initialement, tous les sommets font partie d'un ensemble E où il n'y a aucune arête. Autrement dit, chaque sommet constitue une composante connexe distincte. Ensuite, à chaque étape de l'algorithme, on trouve une arête à ajouter à la forêt en construction que représente E en cherchant, parmi toutes les arêtes reliant deux arbres quelconques de la forêt, une arête (u, v) de poids minimal. Une fois l'arête ajoutée, ces deux arbres sont combinés pour faire partie de la même composante connexe. L'algorithme se termine lorsque E forme un seul arbre et celui-ci est alors un arbre couvrant de poids minimum du graphe.

2°) Illustrez l'exécution de l'algorithme de Kruskal et donnez l'arbre couvrant de poids minimal obtenu.

3°) Définissez la procédure `acpm-kruskal` et donnez son temps d'exécution.

Exercice 3 (Algorithme de Prim)
L'algorithme de Prim a pour propriété que les arêtes de l'ensemble E constituent toujours un arbre. L'arbre démarre d'un sommet racine r arbitraire puis croît jusqu'à couvrir tous les sommets de S. Chaque étape ajoute une arête minimale à l'arbre E pour relier celui-ci à un sommet isolé (sommet auquel n'arrive aucune arête de E). Quand l'algorithme se termine, les arêtes de E forment un arbre couvrant de poids minimal.

1°) Illustrez l'exécution de l'algorithme de Prim et donnez l'arbre couvrant de poids minimal obtenu.

2°) Définissez la procédure `acpm-prim` et donnez son temps d'exécution.

Références

Cormen T. H., Leiserson C. E., Rivest R. L. et Stein C., "Algorithmique" 3e édition, Dunod, 2010.