MTH6412B
Implémentation d'algorithmes de recherche opérationnelle

Arbres de recouvrement minimaux II

Dominique Orban
dominique.orban@polymtl.ca

Mathématiques et Génie Industriel
École Polytechnique de Montréal

Automne 2016

Deux heuristiques

Techniquement, l'union de deux composantes connexes pourrait créer de longues chaînes.

Afin d'accélérer les opérations sur les composantes connexes, on propose les deux heuristiques suivantes :

1. l'union via le rang ;
2. la compression des chemins.

Le but est de trouver la racine d'un arbre dans une forêt d'ensembles disjoints plus rapidement (en temps presque linéaire).

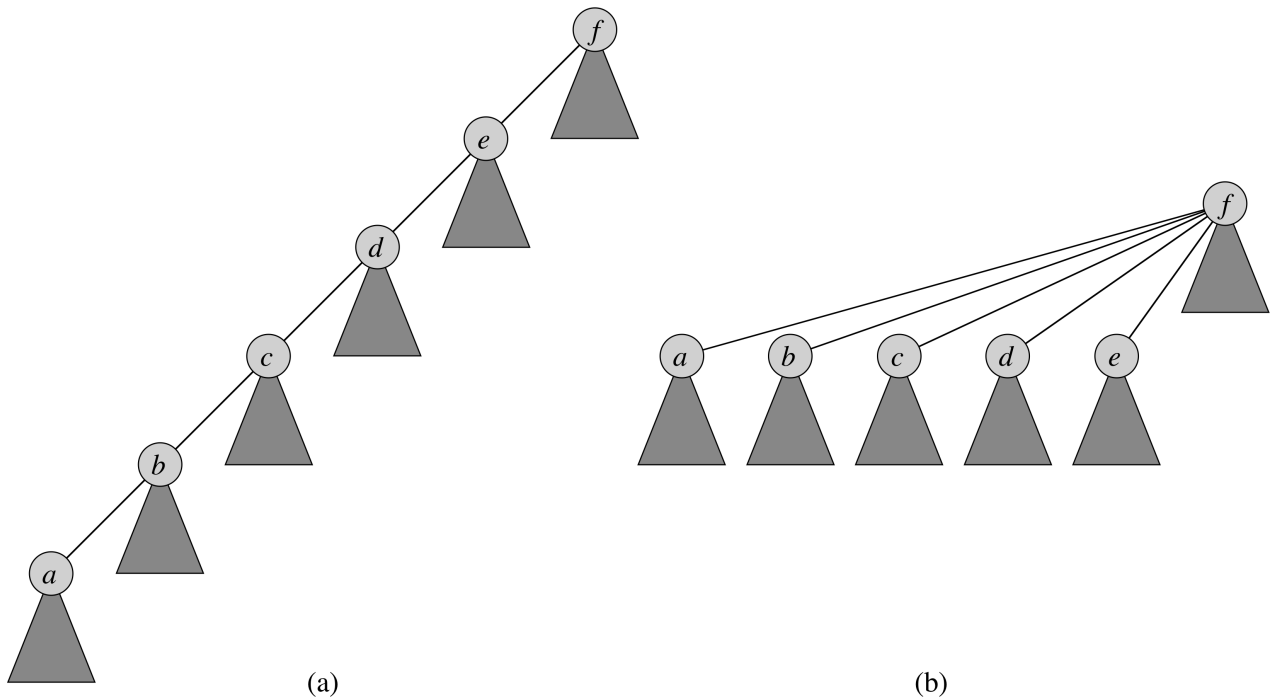
Heuristique 1 : union via le rang

Chaque nœud dans une forêt d'ensembles disjoints possède un attribut de rang dont le rôle est de donner une borne supérieure sur la hauteur de ce nœud dans son arbre.

1. Initialement, chaque nœud reçoit le rang 0 ;
2. lors de la réunion de deux ensembles, les rangs des racines sont comparés :
   - 2.1 si les rangs sont distincts, la racine de plus haut rang devient le parent de la racine de moindre rang et leurs rangs sont inchangés ;
   - 2.2 si les rangs sont égaux, l'une des deux racines devient le parent de l'autre et son rang augmente de 1.

Heuristique 2 : compression des chemins

Lors de la recherche de la racine d'une composante connexe, on passe d'un nœud à son parent. Les nœuds et les arcs ainsi empruntés forment le *chemin de recherche*.
La compression des chemins consistent à faire de la racine le parent direct de chacun des nœuds sur le chemin de recherche.



Seconde partie de ce laboratoire

1. Modifier la structure de données d'ensembles disjoints pour implémenter l'union via le rang ;
2. montrer que le rang d'un nœud sera toujours inférieur à $|S| - 1$. Montrer ensuite que ce rang sera en fait toujours inférieur à $\lfloor \log_2(|S|) \rfloor$;
3. modifier la procédure de remontée vers la racine pour implémenter la compression des chemins.

Algorithme de Prim

L'algorithme de Prim accumule des arêtes de $G = (S,A)$ dans un unique sous-arbre de recouvrement minimum de G et ce sous-arbre grandit au fil des itérations. Tout sommet ne faisant pas partie de ce sous-arbre est isolé.

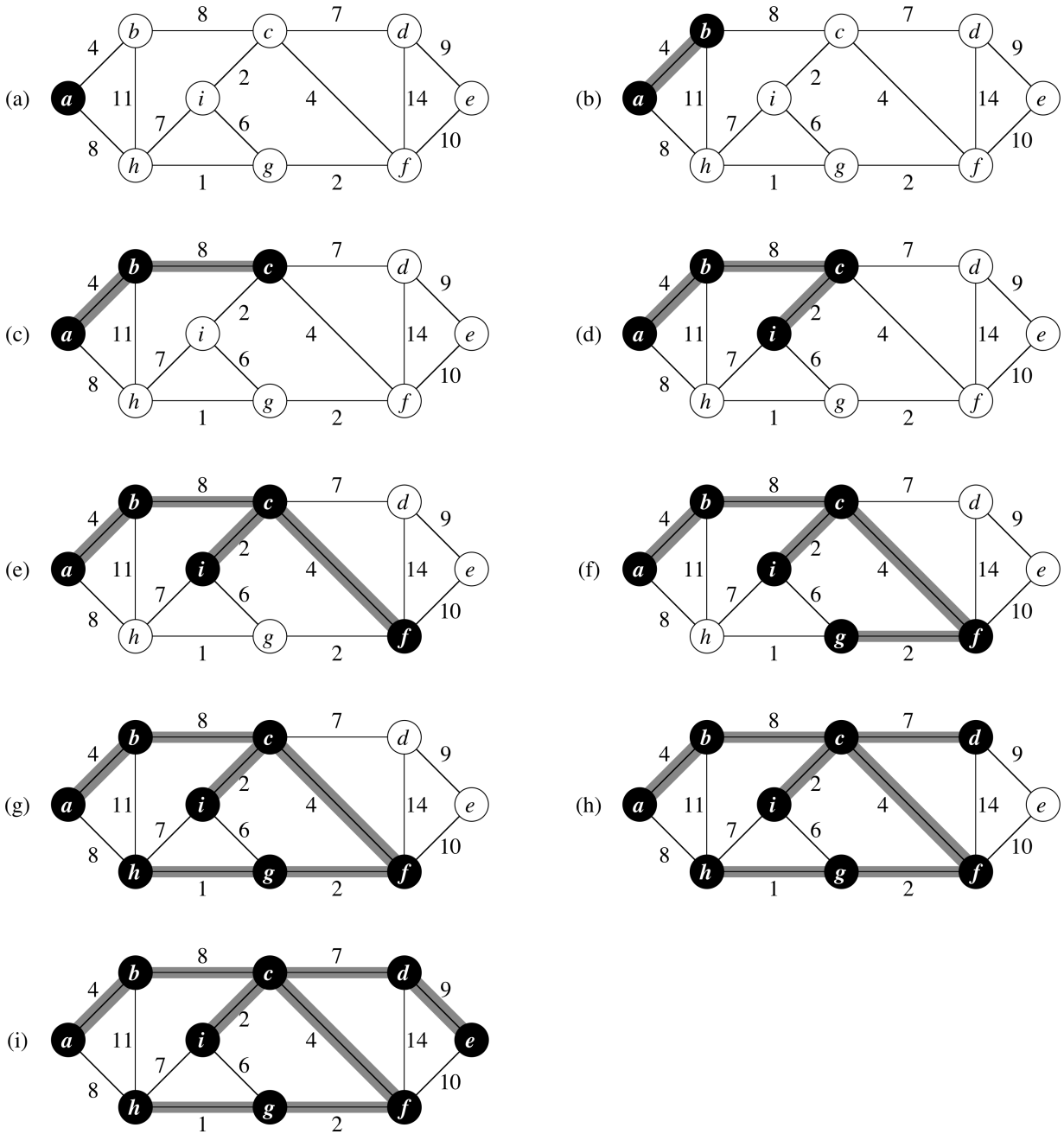
À chaque étape, on ajoute une arête légère entre ce sous-arbre et un sommet isolé. Par le corollaire du laboratoire 3, le sous-arbre ainsi obtenu est encore un sous-arbre de recouvrement minimum.

À chaque étape, les nœuds faisant partie du sous-arbre déterminent une coupe. L'algorithme choisit une arête légère qui traverse cette coupe.

Idées de l'implémentation

1. Chaque nœud a un attribut **`min_weight`** qui donne le poids de l'arête de poids minimal connectant ce nœud au sous-arbre. Initialement, **`min_weight = None`** (i.e., $+\infty$) ;
2. Initialement, le parent de chaque nœud est **`None`** ;
3. l'algorithme débute en un nœud source s choisi par l'utilisateur et l'attribut **`min_weight`** de s est 0 ;
4. une file de priorité contient tous les nœuds qui n'ont pas encore été ajoutés à l'arbre et **`min_weight`** donne la priorité ;
5. chaque fois qu'un nœud est connecté à l'arbre, les attributs **`min_weight`** et **`parent`** de ceux qui n'ont pas encore été connectés doivent être mis à jour.

Illustration de l'algorithme de Prim



Troisième partie de ce laboratoire

1. Implémenter l'algorithme de Prim et le tester sur l'exemple des notes de cours ;
2. Tester votre implémentation sur diverses instances du TSP symétrique.