Langages formels (11X003) - Automne 2025

2. Algorithmes gloutons (II)

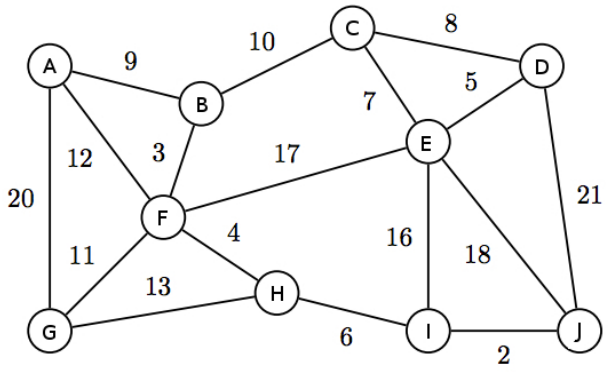
Enseignant: Arnaud Casteigts (et F. Raynaud) *Assistant: Brian Pulfer*
Moniteurs: T. von Düring & A. Maendly

Ces notes correspondent au deuxième cours sur le sujet.

2.3 Arbre couvrant de poids minimum

Le problème de l'arbre couvrant de poids minimum (MST, pour minimum spanning tree, en anglais) est très utilisé, notamment pour faire du routage dans les réseaux. Étant donné un graphe pondéré (les arêtes ont un poids), l'objectif est de trouver un arbre couvrant de ce graphe (c.à.d. un sous-graphe sans cycle qui connecte tous les sommets) dont la somme des poids est la plus petite possible.

Voici un exemple d'instance pour ce problème :



Il existe deux algorithmes gloutons bien connus pour ce problème : l'algorithme de Kruskal et l'algorithme de Prim. Les deux algorithmes sont très faciles à décrire. En revanche, le fait qu'ils trouvent l'optimum systématiquement n'est pas évident (mais c'est bien le cas).

Algorithme de Kruskal

1. Démarrer avec un arbre vide,
2. Ajouter à l'arbre la plus petite arête qui ne crée pas de cycle,

3. Recommencer l'étape 2 jusqu'à ce que l'arbre ait $n - 1$ arêtes.

Notez qu'en cours d'exécution, la solution partiellement calculée jusqu'à présent n'est pas encore un arbre couvrant, mais une **forêt** couvrante composée de petits arbres (comme illustré à la Figure 1). En revanche, quand l'algorithme termine, il n'y a bien qu'un seul arbre, et magie, le poids total de cet arbre est minimum parmi tous les arbres possibles. Nous verrons pourquoi un peu plus bas.

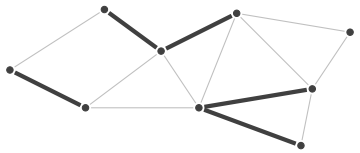


FIGURE 1 – Une forêt couvrante de 4 arbres (dont un n'ayant qu'un sommet).

En attendant, quelle est la complexité de cet algorithme ? Soit n le nombre de sommets et m le nombre d'arêtes. Pour implémenter concrètement cet algorithme, on peut commencer par trier les arêtes par ordre croissant, ce calcul a une complexité en $O(m \log m)$ ($= O(m \log n)$, pourquoi ?). Puis les parcourir une seule fois. La difficulté principale est de détecter si l'ajout d'une arête candidate crée un cycle dans l'arbre (afin de la rejeter). On peut faire cela en maintenant à jour une liste des composantes connexes de l'arbre, ce qui est un peu plus coûteux, mais également faisable en temps $O(m \log n)$ (en utilisant une structure de donnée `union-find`). L'algorithme coûte donc $O(m \log n) + O(m \log n) = O(m \log n)$ (pourquoi ?).

Algorithme de Prim

1. Choisir un sommet de départ (arbitraire) et le marquer comme appartenant à l'arbre,
2. Trouver la plus petite arête dont une extrémité est marquée et l'autre non,
3. Ajouter cette arête et marquer le sommet correspondant,
4. Recommencer les étapes 2 et 3 jusqu'à ce que tous les sommets soient marqués.

Cet algorithme est similaire à celui de Kruskal, mais il a une propriété supplémentaire très pratique : à tout moment, la solution partiellement calculée est connexe et nous n'avons pas besoin de tester les cycles. Il est donc plus facile à implémenter. Par ailleurs, sa complexité est légèrement meilleure : $O(m + n \log n)$ (dans le O, le $+$ est un max) en utilisant des tas de Fibonacci pour trouver l'arête la plus petite à chaque étape.

2.4 Matroïdes & co

Les algorithmes de Kruskal et de Prim sont gloutons : ils sélectionnent les meilleurs éléments à chaque étape sans se soucier du résultat final. Il n'est pas évident que ces choix

"locaux" doivent conduire à un optimum "global". Et pourtant, c'est bien le cas, mais pourquoi ? Avant d'approfondir cette question, examinons de plus près l'algorithme de Kruskal et prouvons déjà qu'il construit bien une solution optimale.

Preuve que l'arbre construit par Kruskal est optimal

Soit G le graphe d'entrée et soit $T \subseteq G$ l'arbre construit par l'algorithme de Kruskal en sélectionnant les arêtes $e_1, e_2, \cdots, e_{n-1}$ dans cet ordre.

Theorème 2.1. *Le poids total de T est bien minimum.*

Démonstration. Par contradiction, supposons que T n'est pas minimum. Il existe alors une arête e_i qui a été le "premier mauvais choix" de l'algo, dans le sens où les arêtes sélectionnées avant, à savoir $e_1, \cdots, e_{i-1}$, pouvaient encore être étendues à un arbre optimal (appelons-le T') mais $e_1, \cdots, e_i$ ne peut plus l'être. Puisque T' est un arbre couvrant qui ne contient pas e_i, le graphe $T' + e_i$ doit contenir un cycle impliquant e_i. Réfléchissons à ce cycle. Si l'une des autres arêtes du cycle a un poids plus grand que e_i, alors on peut remplacer cette arête par e_i, ce qui contredit l'optimalité de T' ; donc chacune de ces arêtes a un poids inférieur ou égal à e_i. Par ailleurs, au moins l'une d'elles, disons f, n'appartient pas à T (puisque T n'a pas de cycle). On a deux cas possibles : soit $w(f) < w(e_i)$, soit $w(f) = w(e_i)$. Dans le premier cas, f aurait été sélectionnée par l'algorithme à la place de e_i (contredisant le fait qu'on a exécuté Kruskal), et dans l'autre cas $T' - f + e_i$ est une solution optimale, contredisant le fait que choisir e_i était une erreur. □

Qu'est-ce qu'un matroïde ?

La preuve ci-dessus est en réalité un cas particulier d'un argument plus abstrait, impliquant une structure appelée un matroïde. La littérature sur les matroïdes est vaste, nous n'en dirons que quelques mots.

Étant donné un ensemble E quelconque, on peut s'intéresser à certains sous-ensembles de E qui satisfont une certaine propriété. Ce cadre est général, on pourrait parler de n'importe quel type d'ensemble E, appelé ici ensemble de base. Quant à l'ensemble des sous-ensembles qui satisfont la propriété choisie, on le note généralement $\mathcal{I}$ et on appelle $(E, \mathcal{I})$ une famille d'ensembles.

Un matroïde est un cas particulier de famille d'ensemble qui satisfait deux axiomes supplémentaires :

1. (Hérédité.) Soit $A \subseteq E$ tel que $A \in \mathcal{I}$. Alors pour tout $B \subseteq A$, on a aussi $B \in \mathcal{I}$. Autrement dit, si un certain sous-ensemble de E a la propriété, alors n'importe quel sous-ensemble de ce sous-ensemble l'a aussi.

2. (Échange.) Si $A \in \mathcal{I}$, $B \in \mathcal{I}$, et $|A| > |B|$, alors il existe un élément $e \in A \setminus B$ tel que $B \cup e \in \mathcal{I}$. Autrement dit, on peut trouver un élément de A transférable vers B sans contredire la propriété.

Revenons au problème de l'arbre couvrant et interrogeons-nous : est-ce que l'ensemble des forêts d'un graphe forment un matroïde ? Autrement dit, si on considère l'ensemble E des arêtes du graphe et l'ensemble $\mathcal{I}$ des sous-ensembles de E qui forment une forêt (= n'ont pas de cycles), la famille $(E, \mathcal{I})$ vérifie-t-elle les deux axiomes désirés ?

1. (Hérédité.) Si A est une forêt et que l'on prend $B \subseteq A$, alors B est une forêt.
2. (Échange.) Soient A et B deux forêts telles que $|A| > |B|$. Alors A a moins de composantes connexes que B. Il existe donc un arbre dans la forêt A qui relie des sommets appartenant à deux composantes différentes de B, et plus précisément, il existe au moins une arête de cet arbre dont les deux extrémités sont dans des composantes différentes de B. On peut alors ajouter cette arête à B sans créer de cycle.

Conclusion : L'ensemble des forêts d'un graphe donné forme donc un matroïde.

Pourquoi tous ces efforts ? Parce que dans n'importe quel matroïde, l'algorithme glouton est garanti de trouver le solution optimale (la preuve est analogue, bien que plus abstraite, à celle de Kruskal). Plus précisément, soit $(E, \mathcal{I})$ un matroïde, et soit $w : E \to \mathbb{R}^+$ une fonction de poids. Alors l'algorithme suivant trouve toujours l'optimum :

1. Initialiser la solution S à $\emptyset$
2. Trouver l'élément e le plus petit (ou le plus grand, si l'on veut maximiser) tel que $S + e$ est dans $\mathcal{I}$, et l'ajouter à S.
3. Répéter l'étape 2 jusqu'à ce qu'aucun élément supplémentaire ne puisse être ajouté.

Observons au passage que c'est exactement l'algorithme de Kruskal, mais formulé de manière plus générale.

En résumé, si l'on soupçonne qu'un problème possède une structure de matroïde, la seule chose que l'on a à faire est de montrer que la famille d'ensemble correspondante satisfait les axiomes d'hérédité et d'échange, et cela garantit immédiatement que l'algorithme glouton trouvera la solution optimale. Il n'y a rien d'autre à démontrer.

Autres structures similaires

Qu'en est-il de l'algorithme de Prim ? Il se trouve que l'algorithme de Prim est également glouton, bien que ses solutions ne forment pas un matroïde. Elles satisfont cependant les axiomes d'une structure plus générale appelée greedoïde, où l'algorithme glouton est également garanti de trouver l'optimum. Il existe de nombreux types de familles d'ensembles, avec divers avantages algorithmiques, les matroïdes et les greedoïdes n'étant que les plus célèbres d'entre eux.