ALGO1 – Un algorithme glouton : l'algorithme de Kruskal

François Schwarzentruber

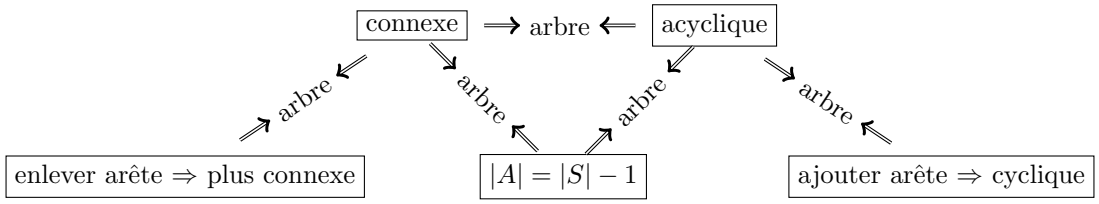
4 novembre 2020

1 Caractérisation des arbres

Définition 1 (arbre) *Un arbre est un graphe non orienté connexe et acyclique.*

Proposition 2 (démonstration en exo) *Soit $G = (S, A)$ un graphe non orienté. Sont équivalentes :*

1) G est un arbre.
2) G est connexe et $|A| = |S| - 1$.
3) G est connexe mais, si l'on enlève une arête quelconque à A, le graphe ne l'est plus
3') Deux sommets quelconques de G sont reliés par une chaîne élémentaire unique.
4) G est acyclique et $|A| = |S| - 1$.
5) G est acyclique, mais si l'on ajoute une arête quelconque à A, il contient un cycle.

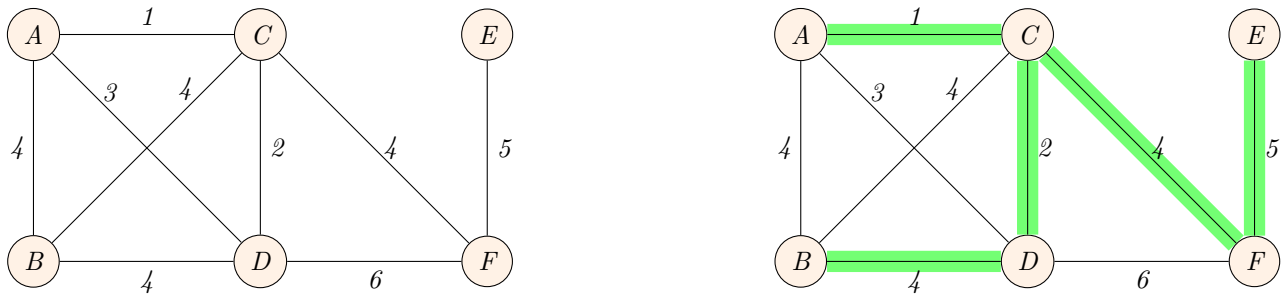


2 Arbre couvrant de poids minimal

Applications : connecter des sommets avec coût minimal.

	Sommets	Arêtes	Coût d'une arête
Circuits électroniques	Broches	Chemins où on peut faire couler de l'étain	Quantité d'étain
Pistes cyclables	Maisons	Routes	Longueur d'une route

Exemple 3



Définition 4 (arbre couvrant de poids minimal) *Soit un graphe $G = (S, A, poids)$ non orienté pondéré. Un arbre couvrant de poids minimum de G est un sous-ensemble d'arête $E \subseteq A$ avec (S, E) qui est un arbre et telle que $\sum_{(a,b)\in E} poids(a, b)$ est minimal.*

Définition 5 *Le problème de l'arbre couvrant de poids minimum est le problème suivant :*

- *entrée : un graphe non orienté connexe pondéré $G = (S, A, poids)$;*
- *sortie : un arbre couvrant de poids minimal $E \subseteq A$.*

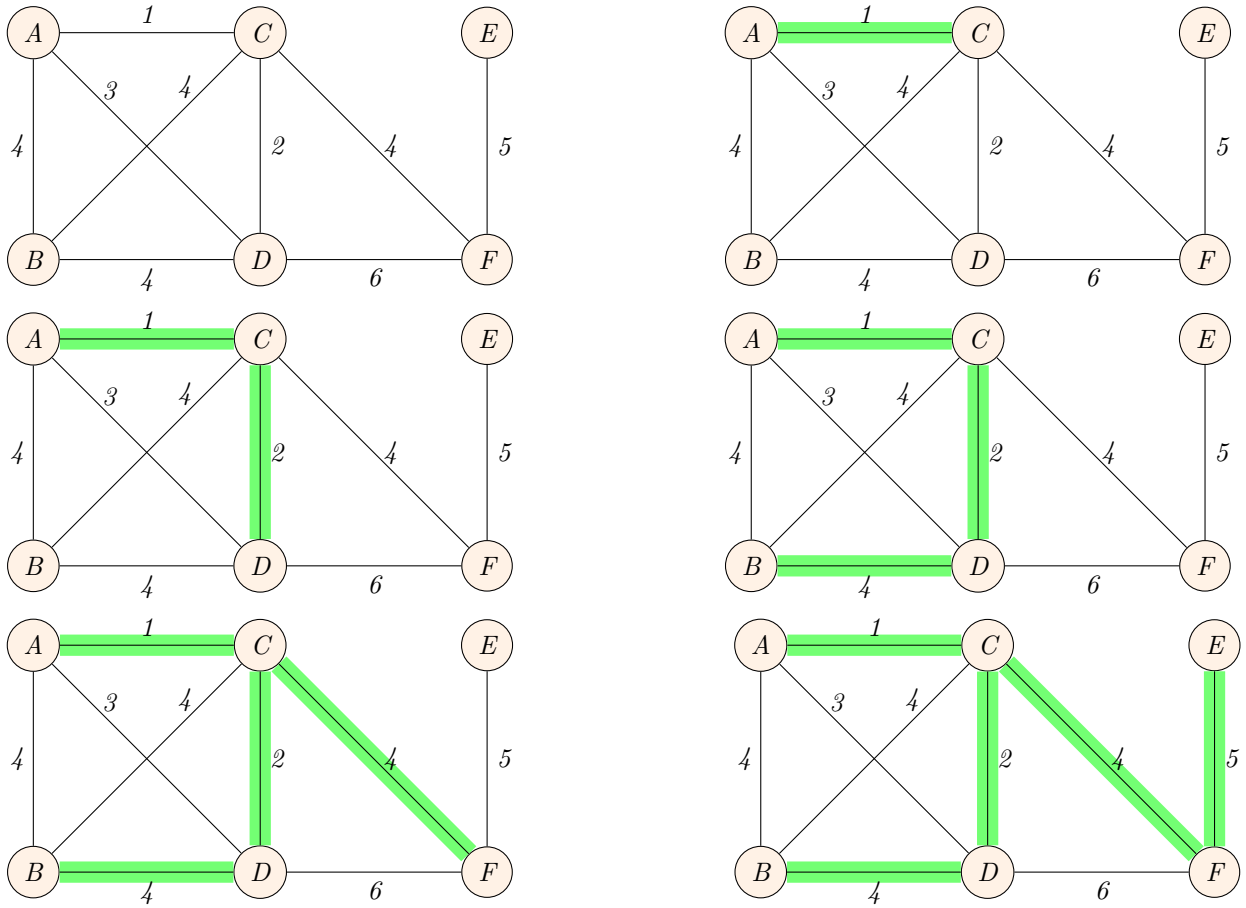
Remarque 6 *Les poids sont des nombres réels et peuvent être négatifs.*

3 Algorithme de Kruskal

L'algorithme date de 1956. Kruskal (1928-2010) a aussi travaillé dans d'autres domaines : statistique et linguistique notamment. Même si, il est surtout célèbre pour son algorithme de calcul d'un arbre couvrant minimal... un classique des cours d'algorithmique !

pre : Un graphe $G = (S, A, poids)$ pondéré non orienté
post : Un ACM (arbre couvrant minimal) de G
fonction kruskal(G)
    $E := \varnothing$
    $L :=$ liste triée des arcs de A, par ordre croissant des poids.
    pour $\{x, y\} \in L$ dans l'ordre croissant **faire**
        si x et y ne sont pas reliés dans le graphe (S, E) **alors**
            $E := E \cup \{\{x, y\}\}$
    retourner E

Exemple 7



Théorème 8 *kruskal(G) est un ACM de G.*

Démonstration.

Point méthodologie : on compare la solution construite E à un ACM T

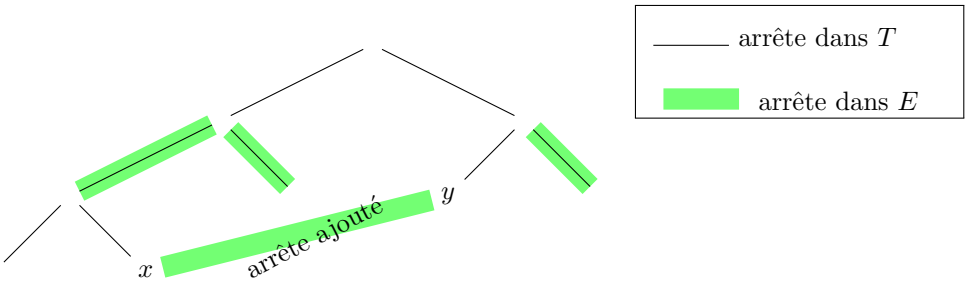
On démontre que la propriété suivante est un invariant :

$\mathcal{I}$: il existe T ACM tel que $E \subseteq T$

Initialisation. Au début, $E = \varnothing$, et il existe bien un ACM (qui trivialement contient les arêtes de E) !
Conservation. Montrons que l'invariant est conservée par un tour de boucle. On considère l'exécution d'un tour de boucle qui vient d'ajouter une arête $\{x, y\}$.

$$E \setminus \{x, y\} \xrightarrow{\text{tour de boucle}} E.$$

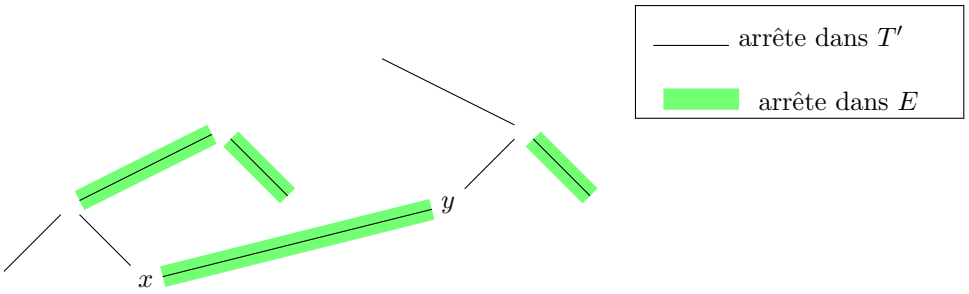
Supposons que l'invariant est vraie au début de la boucle, i.e. il existe T ACM tel que $E \setminus \{x, y\} \subseteq T$. Montrons qu'il existe bien un T' ACM tel que $E \subseteq T'$. Si $\{x, y\} \in T$, on a de la chance et il suffit de prendre $T' = T$. Regardons le cas $\{x, y\} \notin T$ et montrons comment construire un bon T'. Voici la situation actuelle qu'il faut réparer :



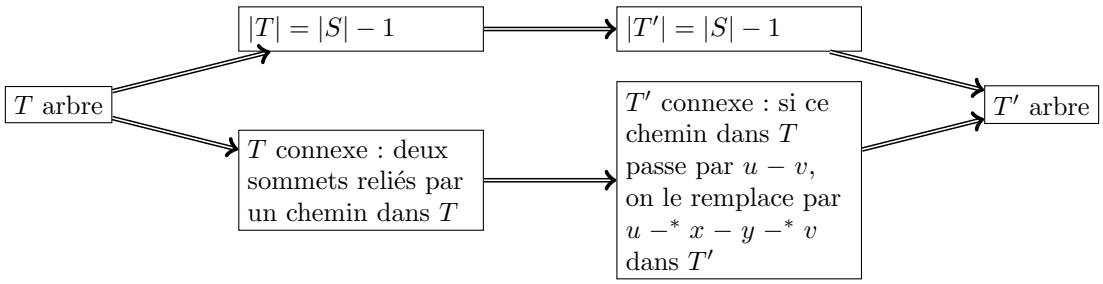
Comme T est un arbre, il est connexe et x et y sont reliés par un chemin c dans T, mais comme l'arête $\{x, y\}$ est ajoutée, ce chemin c ne contient pas que des arêtes de $E \setminus \{x, y\}$. Bref, une des arêtes du chemin c n'est pas dans E. Soit $\{u, v\}$ une telle arête de $T \setminus E$. On pose :

$$T' = T \cup \{\{x, y\}\} \setminus \{\{u, v\}\}.$$

Voici la situation avec ce T' où on a $E \subseteq T'$:



Montrons que T' est un arbre couvrant de poids minimal. D'une part, pour montrer que c'est un arbre, on s'appuie sur la caractérisation arbre $\Leftrightarrow$ connexe et $|S| - 1$ arêtes :



D'autre part, $poids(\{x, y\}) \leq poids(\{u, v\})$.

En effet, l'arête $\{u, v\}$ n'a pas été considéré avant le tour de boucle. Si cela avait été le cas, soit elle aurait été ajouté et $\{u, v\}$ serait dans E mais ce n'est pas le cas ! Soit, elle n'aurait pas été ajoutée car il y avait un chemin de u à v dans le E de l'époque et donc dans $E \setminus \{x, y\}$, et donc dans T. Mais comme $\{u, v\}$ est aussi dans T, on aurait un cycle dans T ce qui n'est pas possible.

Ainsi, $\{x, y\}$ et $\{u, v\}$ ne sont pas encore considéré avant le tour de boucle. Comme $\{x, y\}$ a été choisie, c'est elle de poids minimal, car les arêtes sont considérés par poids croissants.

Ainsi : $poids(T') = poids(T) + poids(\{x, y\}) - poids(\{u, v\}) \leqslant poids(T)$. Donc le poids de T' est minimal.

Conclusion. Montrons que E est un arbre couvrant minimal. L'invariant nous dit qu'il existe T ACM tel que $E \subseteq T$. Ainsi, si on montre que (S, E) est un arbre, c'est gagné.

— D'une part, (S, E) est acyclique car pas de cycle dans (S, T).

— D'une part (S, E) est connexe. Cela résulte de la connexité de G et du traitement systématique de toute les arêtes de G. Comme G est connexe, si $x, y \in S$, on sait que x et y sont reliés dans G :

$$x = x_1 - x_2 - \cdots - x_n = y.$$

On construit alors un chemin de x à y avec des arêtes de E de la manière suivante. Pour tout i, si $\{x_i, x_{i+1}\} \in E$, on prend cette arête là pour le chemin. Sinon, si $\{x_i, x_{i+1}\} \notin E$, lorsque l'algorithme a traité $\{x_i, x_{i+1}\}$, $\{x_i, x_{i+1}\}$ n'a pas été ajouté à E. On avait un chemin de x_i à x_{i+1} dans E (il est toujours intégralement composé d'arêtes de E puisque E est croissant lors de l'algorithme). Donc on prend ce chemin pour aller de x_i à x_{i+1}. On a ainsi construit un chemin de x à y avec des arêtes de E.

■

4 Implémentation

Le test naïf pour savoir si x et y sont reliés dans le graphe (S, E) est couteux. Nous allons utiliser un type abstrait *union-find* pour la relation d'équivalence

$$x \text{ et } y \text{ sont reliés dans le graphe } (S, E)$$

qui possède les opérations suivantes :
— créer_union_find : créer la partition $\cup_{s \in S}\{\{s\}\}$:
— `find` : trouver la composante (ou un représentant) de x
— `union` : fusionne les composantes de x et y.

```
pre : Un graphe G = (S, A) pondéré non orienté
post : Un ACM de G
fonction kruskal(G)
    CC := créer_union_find(S)
    E := ∅
    L := liste triée des arcs de A, par ordre croissant des poids.
    pour {x, y} ∈ L, dans l'ordre croissant faire
        si CC.find(x) ≠ CC.find(y) alors
            E := E ∪ {{x, y}}
            CC.union(x, y)
    retourner E
```

Théorème 9 *L'algorithme de Kruskal s'exécute en* $|creer\_union\_find| + O(A \log(A)) + O(A(|find| + |union|))$.

Démonstration. Il y a la création de la structure union find, puis le tri, puis une boucle, dont chaque passage de boucle appelle $find$ et $union$. ■

	tableau	union-find	union-find avec compression (amorti)
créer	$O(S)$	$O(S)$	$O(S)$
find	$O(1)$	$O(\log S)$	$O(\log^* S)$
union	$O(S)$	$O(\log S)$	$O(\log^* S)$
Kruskal sans le tri	$O(A \times S)$	$O(A \log S)$	$O(A \log^* S)$