Calcul scientifique à haute performance

Cours SIM203 — ENSTA Paris — Année 2020-2021

Arbre couvrant de poids minimal

(Sur base des slides de Sourour Elloumi)

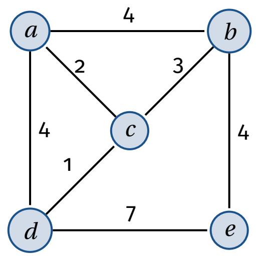
Axel Modave – 02/04/2021

Arbre couvrant de poids minimal (ACPM)

Définition du problème

Soit un graphe $G_E = (V, E)$ connexe non orienté dont les arrêtes sont pondérées. L'idée est de sélectionner les arêtes qui forment un arbre couvrant $G_T = (V, T)$ dont la somme des poids est minimale.

Exemple

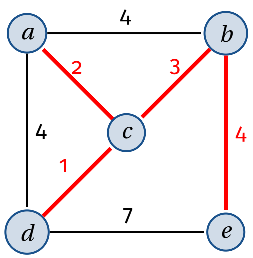


Arbre couvrant de poids minimal (ACPM)

Définition du problème

Dans un graphe $G_E = (V, E)$ non orienté, pondéré et connexe, sélectionner les arêtes qui forment un arbre $G_T = (V, T)$ et dont la somme des poids est minimale.

Exemple



Quelques applications : applications dans les réseaux *(transports, électrique, informatique, télécommunications)*, stockage minimal de données répétitives *(exemple en génomique)*, clustering, …

Algorithme de Kruskal

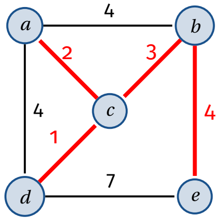
Principe

- Trier les arêtes de E par ordre des poids croissants.
- Dans cet ordre, sélectionner itérativement une arête à T si elle ne forme pas un cycle avec les arêtes déjà sélectionnées.

Pseudo-code

```
T ← ∅       // Liste des arêtes choisies
k ← 0       // Nombre d'arêtes choisies
Trier les arêtes de E par ordre de poids croissant
Tant que k < |V|-1      // |V| est le nombre de sommets
  Sélectionner la première arête ‘a’ de la liste triée ...
    ... qui ne forme pas de cycle avec celles déjà sélectionnées
  T ← T⋃{a}
  k ← k+1
Fin
```

Quelques propriétés de l'ACPM [1/4]



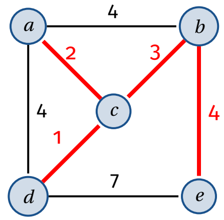
Soient | $G_E = (V, E)$ un graphe,
$G_T = (V, T)$ un arbre couvrant de G_E,

on a : $|T| = |V| - 1$

Pour toute arête (k, l) dans E mais pas dans T :

- G_T contient un chemin unique de k à l ;
- si on ajoute (k, l) à T, on obtient un cycle unique.

Quelques propriétés de l'ACPM [2/4]

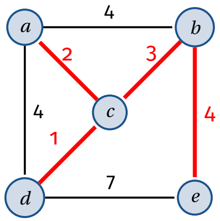


Si on enlève un arc (i, j) de T, le graphe résultant partitionne l'ensemble des sommets V en deux sous-ensembles :

- l'ensemble S des sommets reliés à i,
- l'ensemble $\bar{S}$ des sommets reliés à j.

Les arêtes entre S et $\bar{S}$ constituent la coupe associée à l'arête (i, j) de T.

Quelques propriétés de l'ACPM [3/4]



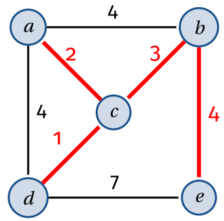
Théorème 1 - CNS d'optimalité [coupe]

Un arbre couvrant $G_T = (V, T)$ est un ACPM de $G_E = (V, E)$

si et seulement si

$\forall (i, j) \in T$, pour toute arête (k, l) de la coupe associée à l'arête (i, j), on a $w_{ij} \leq w_{kl}$

Quelques propriétés de l'ACPM [4/4]



Théorème 2 - CNS d'optimalité [chemin]

Un arbre couvrant $G_T = (V, T)$ est un ACPM de $G_E = (V, E)$

si et seulement si

$\forall (k, l) \notin T$, pour toute arête (i, j) du chemin de G_T reliant k et l, on a $w_{ij} \leq w_{kl}$

Retour sur l'algorithme de Kruskal

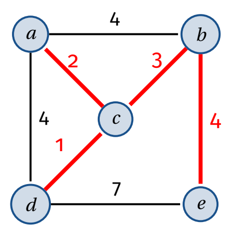
Pseudo-code

```
T ← ∅       // Liste des arêtes choisies
k ← 0       // Nombre d'arêtes choisies
Trier les arêtes de E par ordre de poids croissant
Tant que k < |V|-1      // |V| est le nombre de sommets
  Sélectionner la première arête 'a' de la liste triée ...
    ... qui ne forme pas de cycle avec celles déjà sélectionnées
  T ← T⋃{a}
  k ← k+1
Fin
```

Comment détecter si une arête forme un cycle avec des arêtes sélectionnées ?

- Au départ, aucune arête n'est sélectionnée. On considère une partition du graphe en $|V|$ sous-ensembles, chacun correspondant à un sommet *(chaque sous-ensemble est un singleton)*.
- Chaque fois qu'une arête est choisie, on fusionne les sous-ensembles auxquels appartiennent les deux extrémités de l'arête choisie.
- Au moment du choix d'une arête, on vérifie qu'elle relie des sommets appartenant à des ensembles différents. On est ainsi sûr que la nouvelle arête choisie ne formera pas de cycle.

Algorithme de Prim



Principe

- Les sommets de l'arbre en construction sont contenus dans un ensemble S, qui est initialisé avec un sommet quelconque.
- De manière itérative, trouver une arête de poids minimal joignant un sommet déjà sélectionné à un sommet non encore sélectionné.

Algorithme de Prim

Pseudo-code [Implémentation naïve]

Pour chaque sommet `i` :

- `dansS[i]` indique si le sommet `i` est déjà dans S. (`true` or `false`)
- `voisin[i]` indique un sommet de S qui est le plus proche de `i`.

```
// Initialisation
dansS[1] ← true          // Départ du sommet '1'
k ← 1                    // Nombre de sommets dans S
Pour tout i ≠ 1
  dansS[i] ← false       // Tableau d'appartenance à S
  voisin[i] ← 1          // Tableau des voisins
Fin
// Itérations
Tant que k < |V|-1
  Trouver i et j tels que (dansS[i] = true) ...
    ... et (dansS[j] = false) et (w[i,j] est minimal)
  dansS[j] ← true
  voisin[j] ← i
  k ← k+1
Fin
```

À la fin de l'algorithme, les arêtes (`i`, `voisin[i]`) constituent l'ACPM.

Algorithme de Prim

Vers une implémentation plus efficace ...

Pour chaque sommet `i` :

- `dansS[i]` indique si le sommet `i` est déjà dans S. (`true` or `false`)
- `voisin[i]` indique un sommet de S qui est le plus proche de `i`.
- `dist[i]` indique la distance min. entre `i` et un des sommets déjà dans S. (`dist[i]` $=\infty$ si aucune arête ne relie `i` à un sommet de S.)

À la fin de l'algorithme :

- Les arêtes (`i`, `voisin[i]`) constituent l'ACPM.
- La somme des éléments de `dist` correspond au poids de l'ACPM.

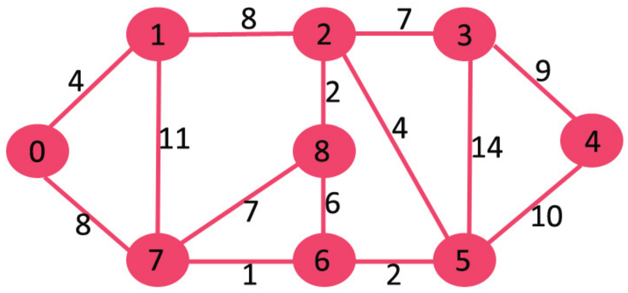
Algorithme de Prim

Pseudo-code [Implémentation plus efficace]

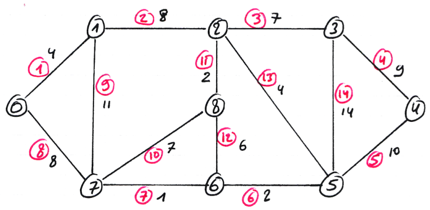
```
// Initialisation
dansS[1] ← true        // Départ du sommet '1'
dist[1] ← 0
k ← 1                  // Nombre de sommets dans S
Pour tout i ≠ 1
  dansS[i] ← false     // Tableau d'appartenance à S
  dist[i] ← w[1,i]     // Tableau des distances   (= ∞ si (1, i) n'existe pas)
  voisin[i] ← 1        // Tableau des voisins
Fin
// Itérations
Tant que k < |V|-1
  Trouver i tels que (dansS[i] = false) et (dist[i] est minimal)
  dansS[i] ← true
  Pour tout j (voisin de i) tel que (dansS[j]=false)
    Si (dist[j] > w[i,j])
      dist[j] ← w[i,j]
      voisin[j] ← i
    Fin
  Fin
  k ← k+1
Fin
```


Exercice

Appliquer les algorithmes de Kruskal et Prim au graphe ci-dessous :



Solution avec l'algorithme de Kruskal



$|V| = 9$
$|E| = 14$
$|T| = |V| - 1 = 8$

1) Trier les arêtes de E par ordre de poids croissant :

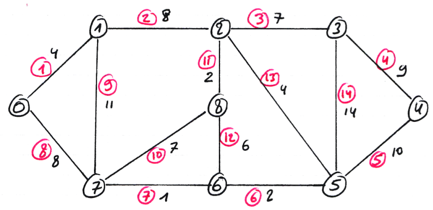
7	6	11	1	13	12	3	10	2	8	4	5	9	14
1	2	2	4	4	6	7	7	8	8	9	10	11	14

2) On selectionne "dans l'ordre" des arêtes qui ne forment pas un cycle avec une arête déjà sélectionnée :

7 6 11 1 13 ~~3~~ ~~2~~ 4 | STOP car on a 8 arêtes.

arêtes non-sélectionn. car elles forment des cycles.

Solution avec l'algorithme de Prim



$|V| = 9$
$|E| = 14$
$|T| = |V| - 1 = 8$

ETAPE INIT :

	dansS	dist	vois
0	T	0	■
1	F	4	0
2	F	∞	0
3	F	∞	0
4	F	∞	0
5	F	∞	0
6	F	∞	0
7	F	8	0
8	F	∞	0

On démarre avec "0", qui est lié à "1" et "7".

ensuite on doit ajouter "1" dans S.

ETAPE 1 :

	dansS	dist	vois
0	T	0	■
1	T	4	0
2	F	8	1
3	F	∞	0
4	F	∞	0
5	F	∞	0
6	F	∞	0
7	F	8	0
8	F	∞	0

On ajoute "1" dans S et on met à jour dist [2] et vois [2]

ensuite, on doit ajouter "2" ou "7" dans S.

ETAPE 2 :

	dansS	dist	vois
0	T	0	■
1	T	4	0
2	F	8	1
3	F	∞	0
4	F	∞	0
5	F	∞	0
6	F	1	7
7	T	8	0
8	F	7	7

On a choisi d'ajouter "7" dans S. et on met à jour dist [6] vois [6] dist [8] vois [8]

etc.