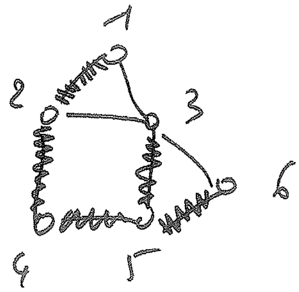
9 ⌊ Retourner A.

∘ Ex :



$E = \{24, 35, 56, 36, 45, 32, 12, 13\}$

$A = \{12, 24, 45, 35, 56\}$.

∘ Analyse de l'algo :

① Terminaison : for + test → OK.

② Correction : On prouve, par récurrence sur le nombre d'arêtes traitées, que les propriétés suivantes sont vraies tout au long de l'algo :

" - Pour toute valeur i de la fonction c, A forme un arbre couvrant sur les sommets x tels que $c(x) = i$.

- Si $c(x) \neq c(y)$ alors A ne contient pas l'arête xy. "

Au début $A = \emptyset$, c'est vrai

lorsqu'on ajoute une arête xy :

. si $c(x) = c(y)$, elle change pas, A non plus les p^tés sont encore vraies.

. si $c(x) \neq c(y)$, on fusionne les composantes et on voit que les 2 propriétés sont conservées.

③ Complexité : la même que COMPOSANTE → $O(m^2)$.

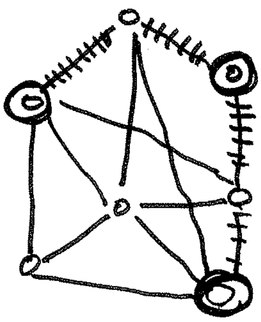
◦ **Pb du jour n° 2** :

∗ __Pbl__ (__STEINER TREE__):

| ENTRÉE : $G=(V,E)$ connexe et $S \subseteq V$ un ensemble de sommets.
| SORTIE : $T=(V',A)$ un sous arbre de G contenant S avec $|A|$ min.

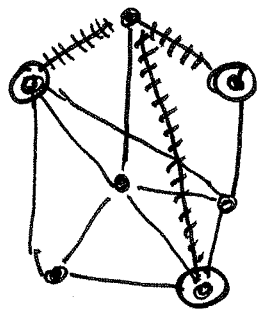
"Connecter certains points (S) à coût minimum."

__Ex__ :



◎ → S.

coût 4.



coût 3.

Le problème est __NP-difficile__ : c'est comme les pbls NP-complets, on ne connaît pas d'algo. polynomiaux pour les résoudre (seul^t exponentiel). On dit NP-complet pour les pbls de décision (la réponse est OUI/NON) et NP-difficile pour les pbls d'optimisation (trouver la meilleure valeur/structure telle que...). Le million de \$ vaut pour n'importe quel pbl NP-complet ou NP-difficile.

◦ __Rem__ : . Si $S=V$, on sait faire, c'est l'arbre couvrant !

. Si $|S|=2$, on sait faire aussi, $S=\{x,y\}$, on cherche un plus court chemin de x à y ! (cf + tard).

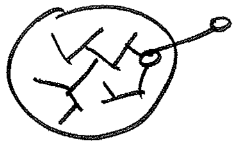
IV). Quelques propriétés des arbres.

- **Prop 1** : Un arbre sur n sommets contient $n-1$ arêtes.

Pr : Induction (récurrence)

- Vrai si $n=1$ (pas d'arête !)
- Supposons que la propriété soit vraie sur tous les arbres à n sommets et considérons T un arbre sur $n+1$ sommets.

T possède une feuille f, par hypothèse $T \setminus f$ est un arbre sur n sommets donc avec $n-1$ arêtes.



$T \setminus f$

Ainsi T a n arêtes $(n-1+1)$. ∎

- Une **forêt** est une union disjointe d'arbres.

- **Prop 2** : Un graphe est une forêt ssi il ne possède pas de cycle.

Pr : $\Rightarrow$ Immédiat, une forêt n'a pas de cycle.

$\Leftarrow$ Si G est sans cycle, chacune de ses composantes connexes est sans cycle et connexe donc un arbre ∎

- **Prop 3** : Une forêt ayant n sommets et c composantes connexes a $n-c$ arêtes.

Pr : On note $n_1, n_2, \dots, n_c$ les tailles des composantes connexes de la forêt : $n_1+n_2+\dots+n_c = n$ et $m = (n_1-1)+\dots+(n_c-1) = n-c$.

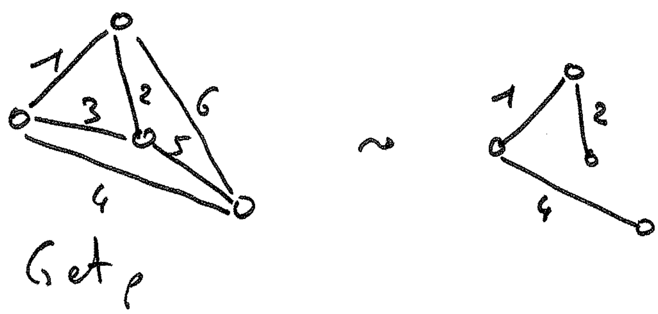
V] Arbre couvrant de coût minimum

· **ENTRÉE** : $G=(V,E)$ un graphe connexe et $p: E \to \mathbb{R}^+$ une fonction de ~~coût~~ poids

· **SORTIE** : Un arbre couvrant $T=(V,A)$ de G tel que $p(A)=\sum_{e\in A} p(e)$ soit minimum.

"On veut relier tous les sommets de G à coût minimum".

· **Exemple** :

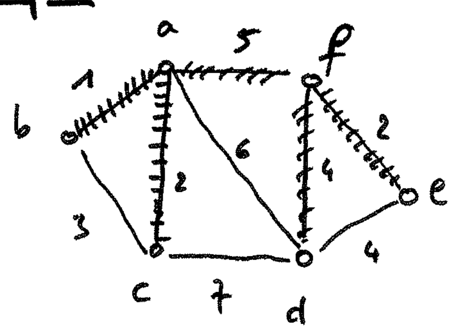


A de coût min
$p(A)=7$.

· ALGORITHME DE KRUSKAL : On ajoute la ligne suivante à l'algo ARBRE-COUVRANT :

o [Trier les arêtes de E par ordre croissant selon p.

o **Exemple** :



Etrié = {ab, ac, fe, ~~fe~~, fd, ~~de~~, af, ~~ad~~, ~~cd~~}
A = {ab, ac, fe, fd, af}.
$p(A)=14$. $\left[p(A)=\sum_{xy\in A} p(xy)\right]$.

o Rem : · Sol. non unique (voir TD), par ex., on peut remplacer fd par ed (ça dépend de l'ordre des arêtes de poids =).

· L'algo utilisé fait partie de la famille des algorithmes gloutons : la solution partielle déjà calculée est étendue dès qu'on peut le faire (pas de retour en arrière, pas de "backtrack").