Algorithmes gloutons (2)

CM 10 • Le problème de l'allocation d'une ressource • 23 novembre 2021

F. Laroussinie
M1 – Algo
2021 – 2022

Codage de Huffman

Le problème

Objectif : étant donné un texte sur un alphabet $\sum$,

le coder succinctement (brièvement) **en binaire.**

- Données : un alphabet $\sum$ et une fonction de "fréquence" $\mathrm{f}\colon \sum \rightarrow N$
- Résultat : un codage $\Phi : \sum \rightarrow \{0,1\}^*$ tel que $\sum_{a\,\epsilon\,\sum} f(a) \cdot |\Phi(a)|$ soit minimal.

Codage de Huffman

Exemple

$\sum$:	a	b	c	d	e	f
f :	45	13	12	16	9	5

- **Solution 1 :**

a	b	c	d	e	f
000	001	010	011	100	101

$$\sum f(a) \cdot |\Phi(a)| = f(a) \cdot 3 + f(b) \cdot 3 + f(c) \cdot 3 + f(d) \cdot 3 + f(e) \cdot 3 + f(f) \cdot 3$$
$$= 45 \cdot 3 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 3 + 5 \cdot 3 = 300$$

- **Solution 2 :**

a	b	c	d	e	f
0	101	100	111	1101	1100

$$\sum f(a) \cdot |\Phi(a)| = 45 \cdot 1 + 13 \cdot 3 \ \ldots = 224$$

Codage de Huffman

Codage…

Ici en limite aux codes préfixes, *ie* : aucun $\Phi(x)$ n'est préfixe d'un $\Phi(y)$

- Propriété : Il existe un code préfixe optimal.
- *Avantage* : Le décodage est très simple !

Codage de Huffman

Exemple de décodage

a	b	c	d	e	f
0	101	100	111	1101	1100

$$1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1$$

$$\underbrace{1\ 0\ 1}_{b}\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1$$

$$\underbrace{1\ 0\ 1}_{b}\ \underbrace{0}_{a}\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1$$

$$\underbrace{1\,0\,1}_{b}\ \underbrace{0}_{a}\ \underbrace{1\,0\,0}_{c}\ 1\,1\,1\,1\,1\,0\,1\,0\,1\,0\,1$$

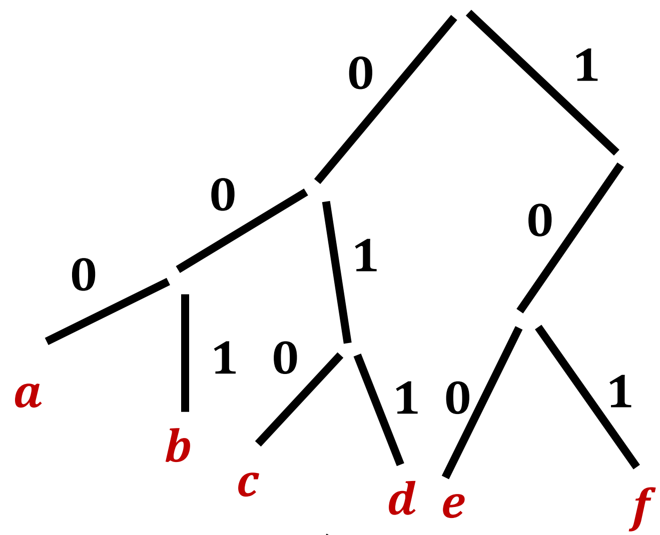
$$\underbrace{1\,0\,1}_{b}\ \underbrace{0}_{a}\ \underbrace{1\,0\,0}_{c}\ \underbrace{1\,1\,1}_{d}\ 1\,1\,0\,1\,0\,1\,0\,1$$

$$\underbrace{1\,0\,1}_{b}\ \underbrace{0}_{a}\ \underbrace{1\,0\,0}_{c}\ \underbrace{1\,1\,1}_{d}\ \underbrace{1\,1\,0\,1}_{e}\ \underbrace{0}_{a}\ \underbrace{1\,0\,1}_{b}$$

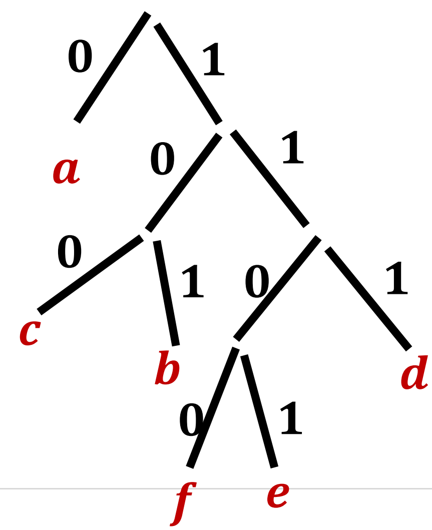
Codes préfixes et arbres binaires

Exemple de décodage

a	b	c	d	e	f
000	001	010	011	100	101



a	b	c	d	e	f
0	101	100	111	1101	1100



Codes préfixes et arbres binaires

Un code prefixe optimal est toujours représenté sous la forme d'un arbre binaire localement complet.

Cout d'un arbre T selon f :

$$\boldsymbol{B_f(T) = \sum_{a \in F(T)} f(a) \cdot prof_T(a)}$$

$\mathbf{F(T)}$: **feuilles de T**

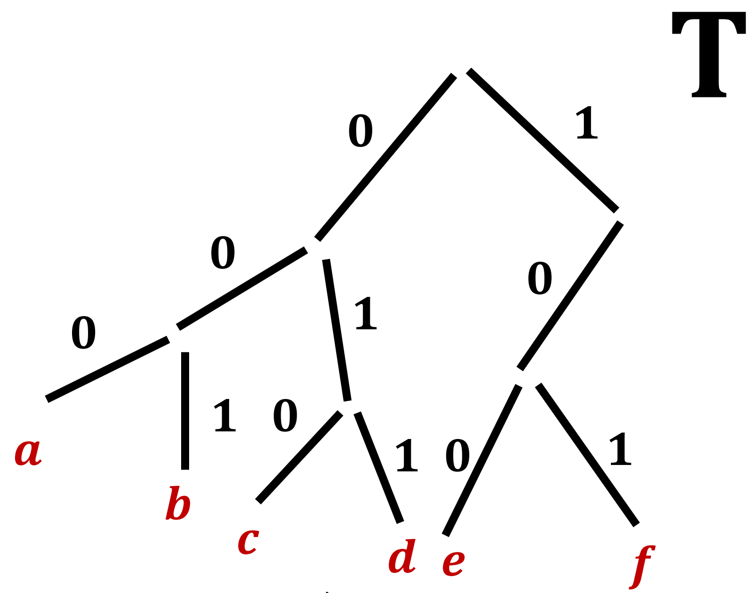
$\mathbf{prof_T(a)}$: **profondeur du noeud a dans T**

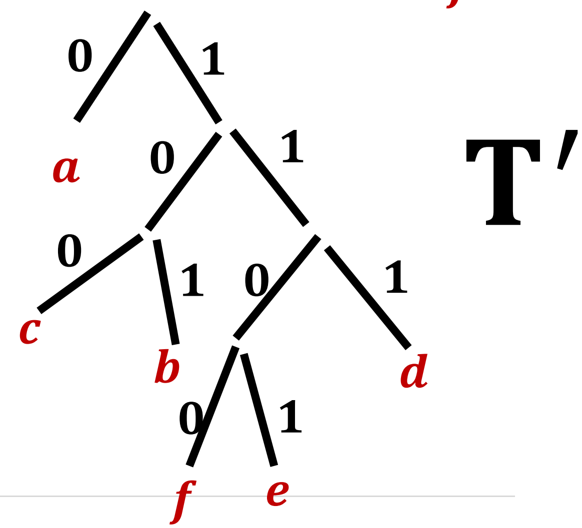
- On écrira B(T) lorsque f est fixée par le contexte.
- On étend B() aux codages : $B(\Phi) = \sum f(a) \cdot |\Phi(a)|$

Codes préfixes et arbres binaires

a	b	c	d	e	f
000	001	010	011	100	101

$$\mathbf{B(T) = (45 + 13 + 12 + 16 + 9 + 5) * 3 = 300}$$





a	b	c	d	e	f
0	101	100	111	1101	1100

$$\mathbf{B(T') = \; 45 \cdot 1 + 13 \cdot 3 \; + 12 * 3 + 16 * 3 \; + \; 9 * 4 \; + 5 * 4 = 224}$$

Arbres

On considere les primitives suivantes sur les arbres :

- **feuille(a) :** Crée une feuille étiquetée par "a"
- **arbre(t1,t2) :** Crée un arbre avec t1 come fils gauche et t2 comme fils droit
- **fg(t) :** Retourne le fils gauche
- **fd(t) :** Retourne le fils droit

File de priorité

On utilise une **file de priorité** pour stocker des **arbres** (correspondant à des codes pour des sous-ensembles de $\sum$) avec comme **clé** la somme des **fréquences** de ces lettres.

Algorithme d'Huffman

$n := |\Sigma|$

$FP := \text{FillePriorité}(\{ (\text{feuille}(a), f(a)) \mid a \in \Sigma \})$

$\text{Pour } i = 1 \text{ à } n - 1:$

$\quad (t1, f1) := \text{ExtraireMin}(FP)$

$\quad (t2, f2) := \text{ExtraireMin}(FP)$

$\quad \text{Ajouter}\big(FP, (\text{arbre}(t1, t2), f1 + f2)\big)$

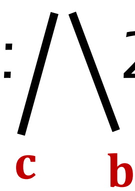
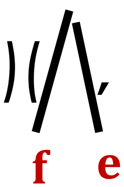
$(T, f) := \text{ExtraireMin}(FP)$

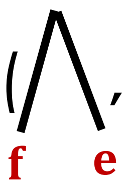
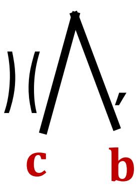
$\text{retourner } T$

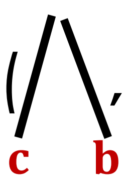
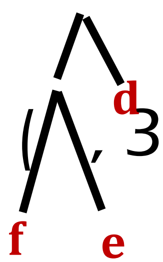
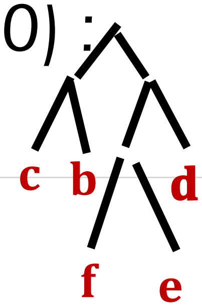
Exemple

$\sum$:	a	b	c	d	e	f
f :	45	13	12	16	9	5

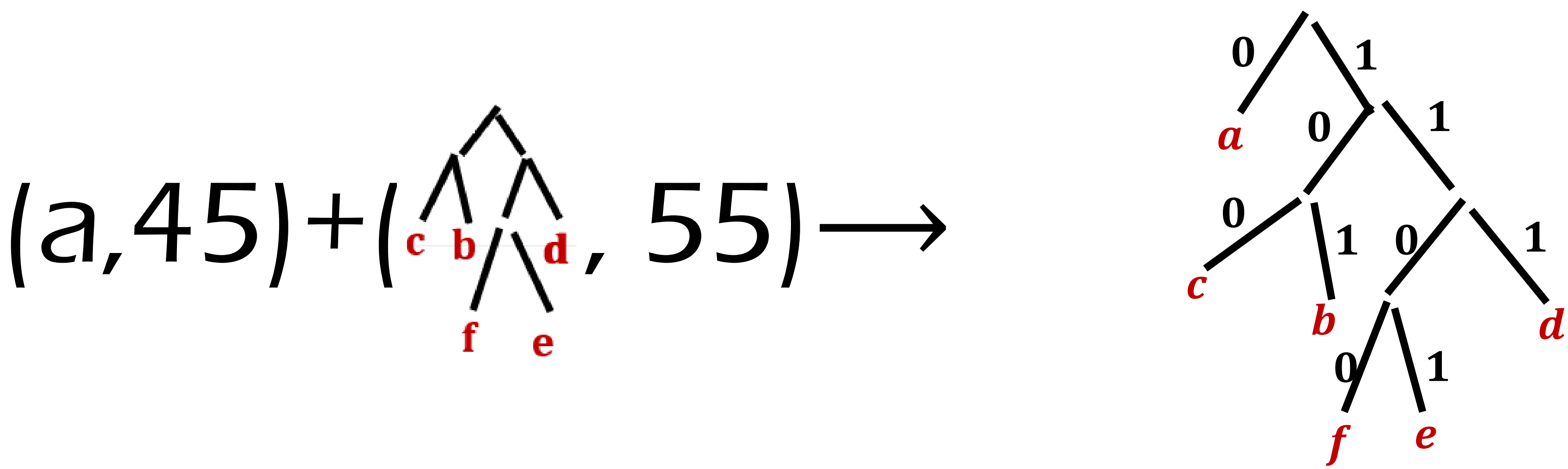
- (f,5) et (e,9) :  14 +(a,45)(b,13)(c,12)(d,16)

- (c,12) et (b,13) :  25 +(a,45)(d,16)(, 14)

- (, 14) et (d,16) :  30 +(a,45)(, 25)

- (, 25) et (, 30) :  55 +(a,45)

Exemple



Algorithme d'Huffman

Théorème : L'algorithme d'Huffman donne un code préfixe optimal.

Lemme 1 :

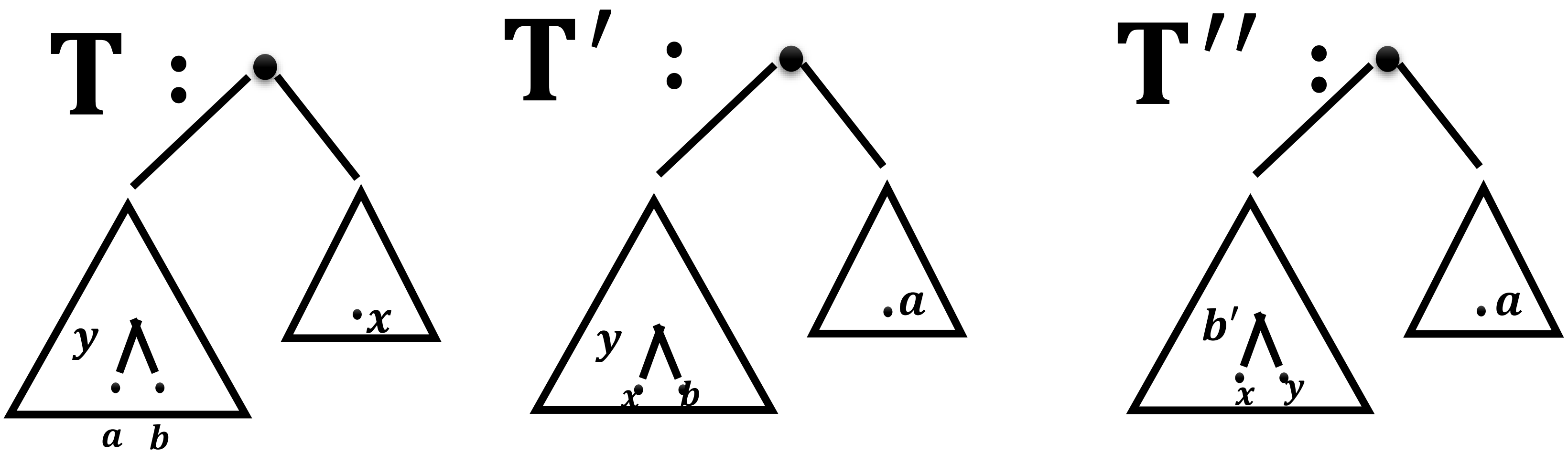
Etant donnes $(\sum, f)$ et $x, y \in \sum$ telles que x et y aient des fréquences minimales, alors il existe un code préfixe optimal Φ avec $\Phi(\mathrm{x}) = w \cdot 0$ et $\Phi(\mathrm{y}) = w \cdot 1$

NB : $\Phi(\mathrm{x})$ et $\Phi(\mathrm{y})$ ont la même longueur et, x et y ont le même père dans l'arbre binaire associé à Φ.

Algorithme d'Huffman

Lemme 1 – preuve

- Soit T l'arbre associé à un code optimal.
- Soit a,b deux feuilles de T, de même père et situées à la profondeur max dans T



Hypothèse :

$$f(x) \leq f(a) \qquad f(y) \leq f(b) \quad pf_T(a) < \; pf_T(x)$$

Algorithme d'Huffman

Lemme 1 – preuve

$$B(T') = B(T) - \boldsymbol{f(x) \cdot pf_T(x) - f(a) \cdot pf_T(a)} + f(x) \cdot pf_T(a) + f(a) \cdot pf_T(x)$$

$$B(T') = B(T) + f(x) \cdot \big(pf_T(a) - pf_T(x)\big) - f(a) \cdot (pf_T(a) - pf_T(x))$$

$$B(T') = B(T) + \big(f(x) - f(a)\big) \cdot \big(pf_T(a) - pf_T(x)\big)$$

$$\Rightarrow B(T') \leq B(T)$$

$$\boldsymbol{B(T'') \leq B(T') \leq B(T) : T'' \; optimal\,!}$$

Algorithme d'Huffman

Lemme 2 :

Soit $\mathbf{T}$ un arbre binaire représentant un code préfixe optimal pour $(\sum, \mathbf{f})$.

Soient $\mathbf{x}$ et $\mathbf{y}$ deux feuilles avec le même père $\mathbf{z}$ dans $\mathbf{T}$.

Soient $\mathbf{T}' = \mathbf{T} \backslash \{\mathbf{x}, \mathbf{y}\}$ et $\mathbf{f}' = \mathbf{f}_{|\sum \backslash \{\mathbf{x},\mathbf{y}\}}$ **et** $\mathbf{f}'(\mathbf{z_{new}}) = \mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{y})$

Alors $\mathbf{T}'$ représente un code préfixe optimal pour $(\sum', \mathbf{f}')$.

NB : $\sum' = \sum \backslash \{\mathbf{x}, \mathbf{y}\} \cup \{\mathbf{z_{new}}\}$

Algorithme d'Huffman

Lemme 2 – preuve

$B(T) = \sum f(a) \cdot pf_T(a)$

$\boldsymbol{pf(x) = pf(y) = pf(z_{new}) - 1}$

$B(T') = B(T) - \boldsymbol{f(x) \cdot pf(x)} - \boldsymbol{f(y) \cdot pf(y)} + (f(x) + f(y)) \cdot pf(z_{new})$

$B(T') = B(T) - (f(x) + f(y)) \cdot (pf(x) - pf(z_{new}))$

$B(T') = B(T) - f(x) - f(y)$

Si $\boldsymbol{T'}$ n'est pas optimal, il existe $\boldsymbol{T''}$ optimal pour $(\sum', \mathbf{f}')$
Et remplacer $\boldsymbol{z}$ par $(\boldsymbol{x}, \boldsymbol{y})$ dans $\boldsymbol{T''}$ donne $\boldsymbol{T'''}$ tq
$\boldsymbol{B(T''') = B(T''') + f(x) + f(y) < B(T)}$!

Algorithme d'Huffman

Théorème :
L'algorithme d'Huffman donne un code préfixe optimal.

Preuve : par induction sur $|\sum|$

- $|\sum|$ = 2 : ok
- $|\sum|$ = n + 1

Soit Φ_{algo} le code renvoyé par l'algo et Φ_{opt} un code optimal.

Soient x et y les deux lettres choisies par l'algo avec des priorités min.
Par le Lemme 1, on peut supposer $\boldsymbol{\Phi_{opt}(x) = w \cdot 0}$ et $\boldsymbol{\Phi_{opt}(y) = w \cdot 1}$

Par le Lemme 2, on sait que $\boldsymbol{\Phi'}$ définie par :

$$\boldsymbol{\Phi'(u) = \Phi_{opt}(u)} \text{ si } \boldsymbol{u} \in \sum\backslash\{\boldsymbol{x, y, z_{new}}\} \text{ ET } \boldsymbol{\Phi'(z_{new}) = w}$$

Est optimal pour $(\sum', \mathbf{f}')$[…] !
Et de plus : $\boldsymbol{B(\Phi') = B\left(\Phi_{opt}\right) - f(x) - f(y)}$

Algorithme d'Huffman

De son côté, l'algorithme calcule aussi un code $\boldsymbol{\Phi_{algo}}'$ pour $(\sum', \mathbf{f}')$ et par hypothèse d'induction, il est optimal, donc : $\boldsymbol{B}\left(\boldsymbol{\Phi_{algo}}'\right) = \boldsymbol{B}(\boldsymbol{\Phi}')$

Et on a : $\boldsymbol{B}\left(\boldsymbol{\Phi_{algo}}'\right) = \boldsymbol{B}\left(\boldsymbol{\Phi_{algo}}\right) - \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{y})$ (car $\boldsymbol{\Phi_{algo}}(\boldsymbol{x}) = \boldsymbol{w}' \cdot \mathbf{0}$ et $\boldsymbol{\Phi_{algo}}(\boldsymbol{y}) = \boldsymbol{w}' \cdot \mathbf{1}$)

D'où : $\boldsymbol{B}\left(\boldsymbol{\Phi_{algo}}\right) = \boldsymbol{B}\left(\boldsymbol{\Phi_{opt}}\right)$

Le sac à dos "le retour"

n objets :

w	...
v	

- Un sac de capacité **k**.
- **Objectif :** maximiser la valeur sans dépasser le poids **K**...
- **Cas particuliers :**
  - $v_i = 1 \ \forall \ i$
  - Objets fractionnables

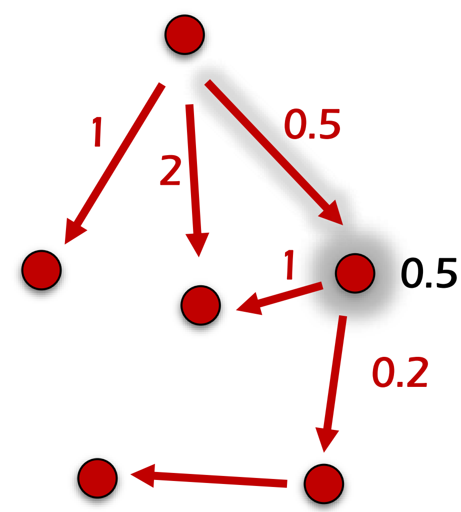
Algo. de Dijkstra (PCC)

$G = (S, A, w)$: orienté valué avec $w: A \to \mathbb{R}_+$
$\to$ calculer les PCC depuis une origine $s \in S$

$\mathbf{Procedure\ PCC - Dijkstra(G, s)}$

// $G = (S, A, w)$: un graphe orienté valué
// $s \in S$: un sommet origine.

begin
  pour chaque u ∈ S **faire**
    π[u] := nil
    $d[u] := \begin{cases} 0 & \text{si } u = s \\ \infty & \text{sinon} \end{cases}$
  F := FilePriorité(S,d)
  tant que F ≠ ∅ **faire**
    u := Extraire-Min(F)
    pour chaque (u,v) ∈ A **faire**
      si d[v] > d[u] + w(u,v) **alors**
        d[v] := d[u] + w(u,v)
        π[v] := u
        MaJ-F-Dijkstra(F,d,v)
  return (d, π)
end

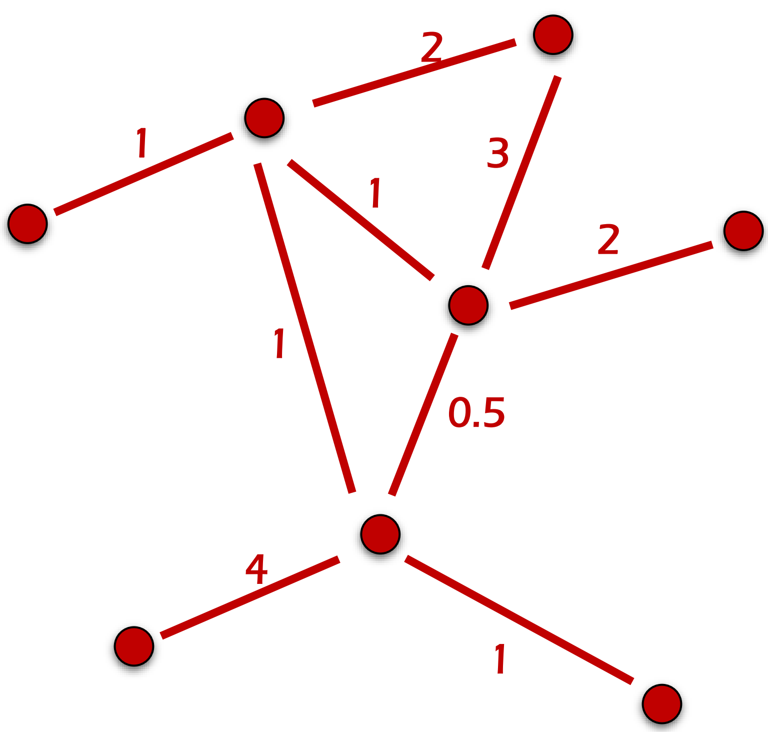


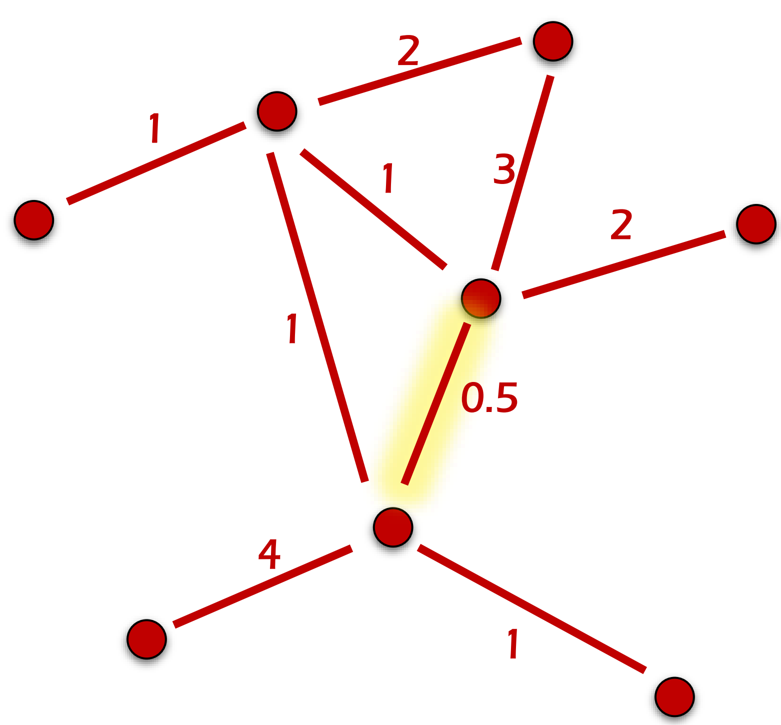
Arbres Couvrants Minimaux (ACM)

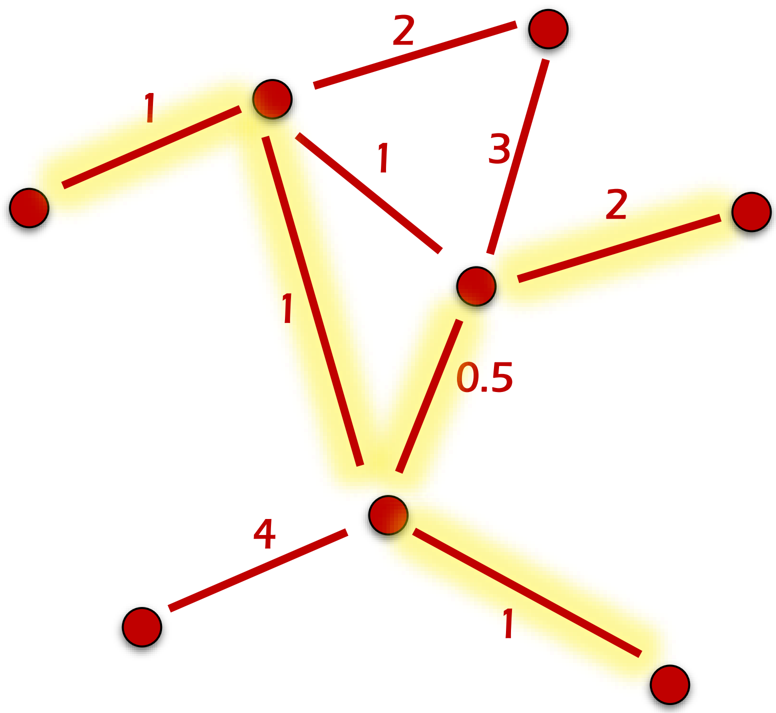
$\boldsymbol{G} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{w})$: non-orienté , connexe.

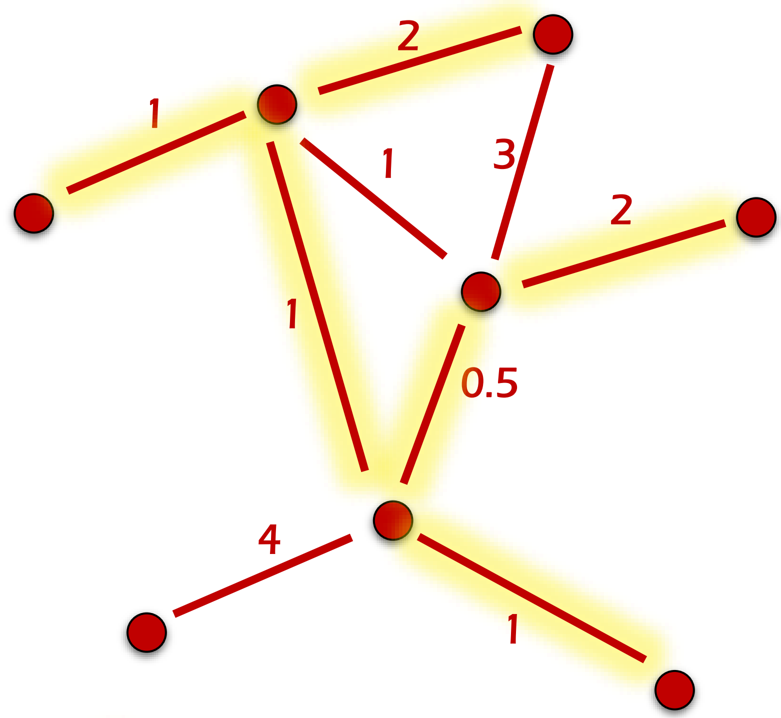
ACM $= A' \subseteq A$ t.q. (S, A') est connexe et acyclique — **AC**

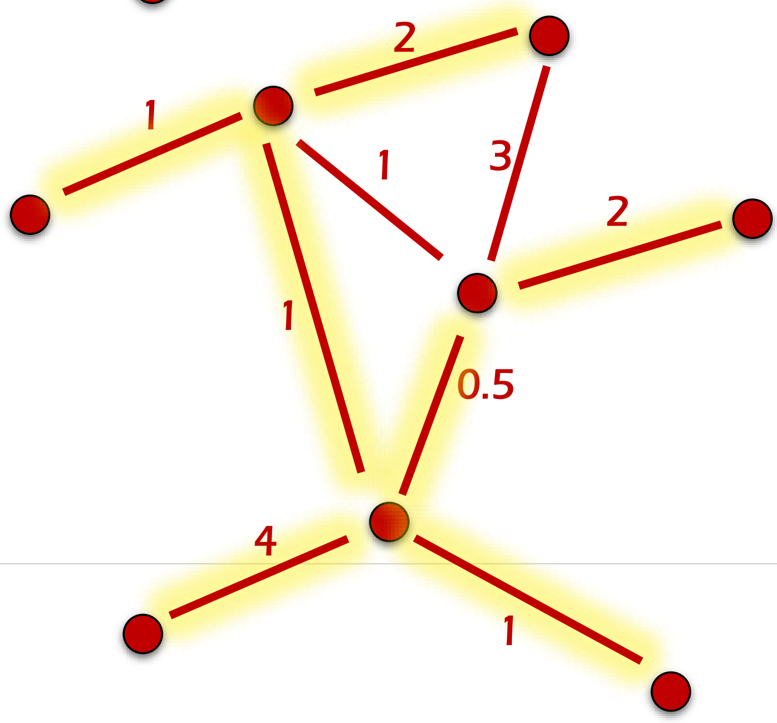
et $\boldsymbol{w}(\boldsymbol{A}') = \sum_{(x,y) \in A'} \boldsymbol{w}(\boldsymbol{x}, \boldsymbol{y})$ est minimal — **M**

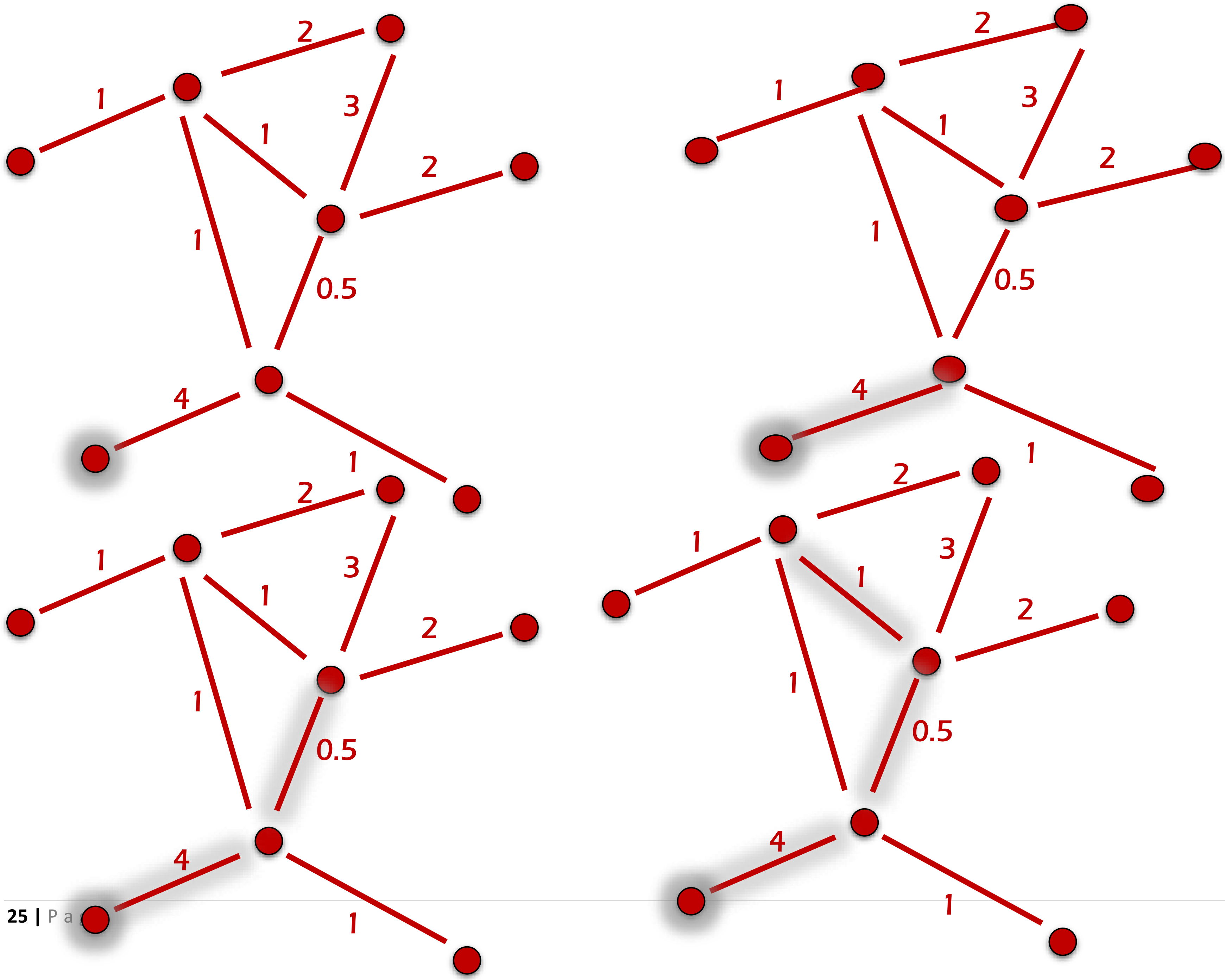


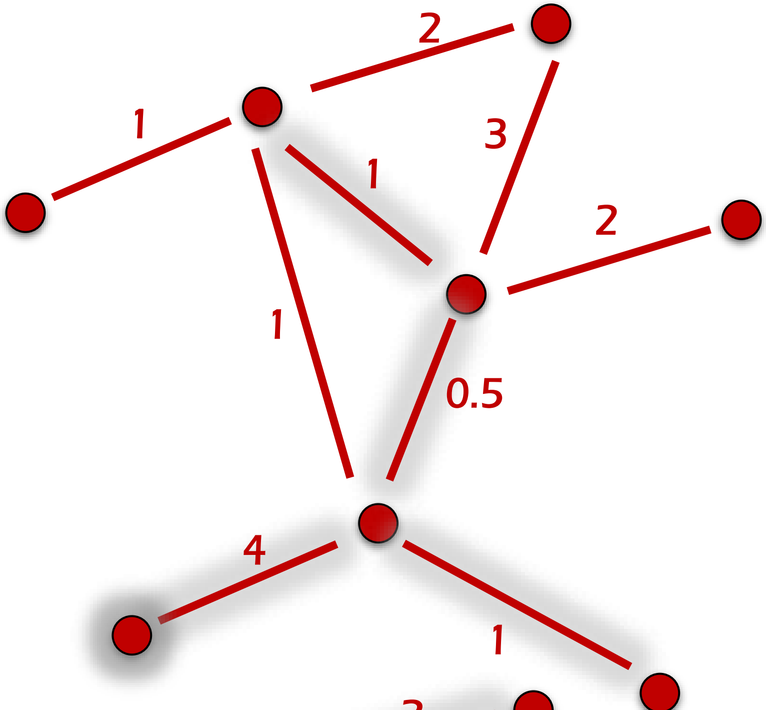


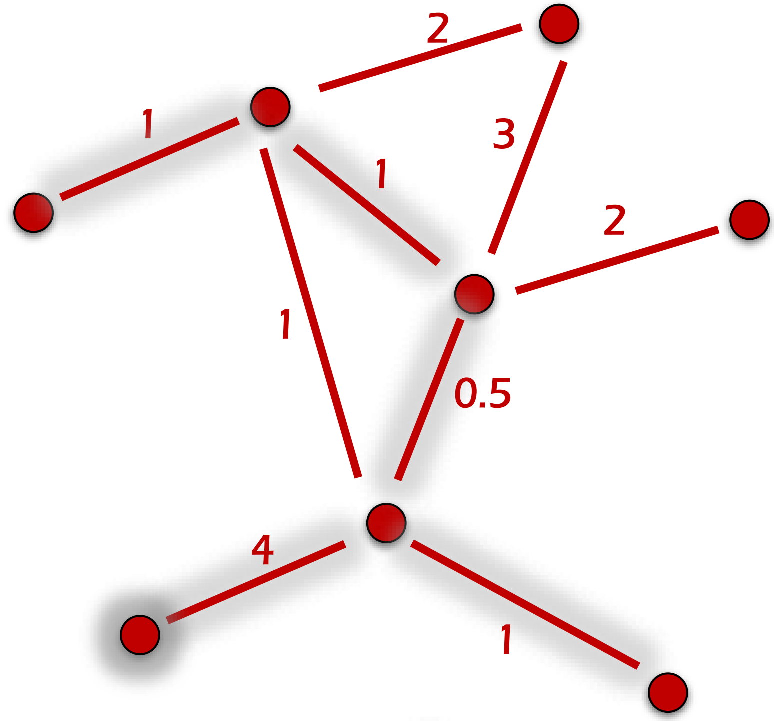

2
1
1
3
2
1
0.5
4
1

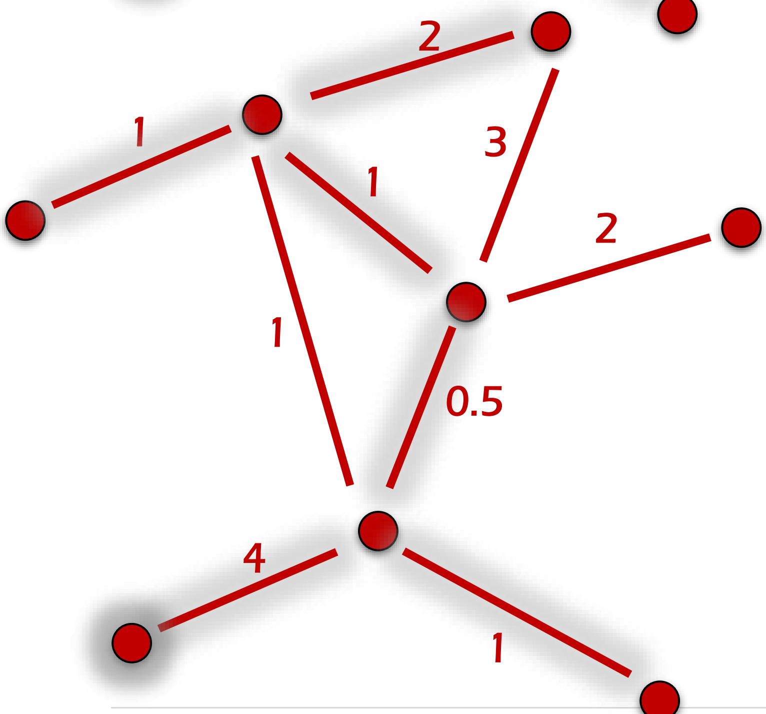

2
1
1
3
2
1
0.5
4
1

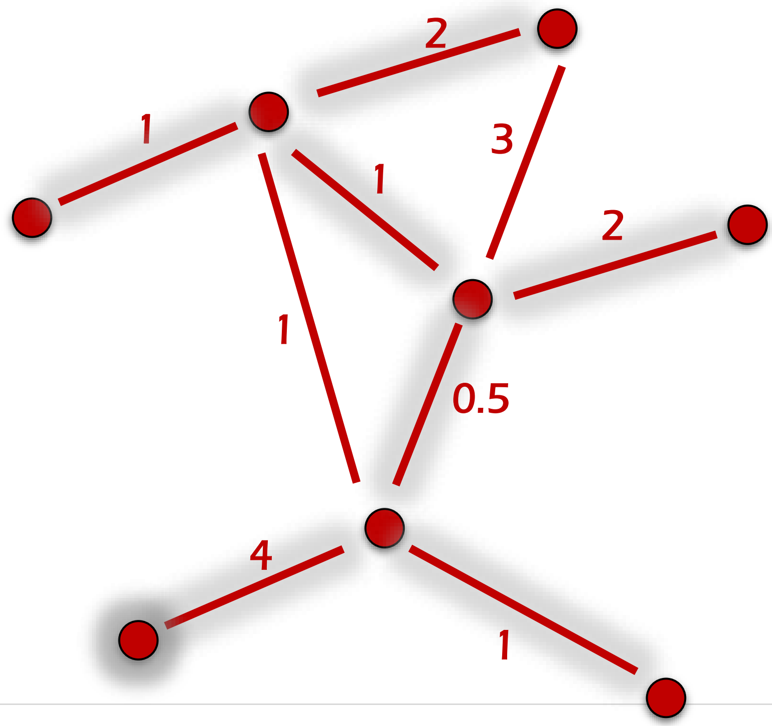

2
1
1
3
2
1
0.5
4
1


2
1
3
1
2
1
0.5
4
1
2
1
3
1
2
1
0.5
4
1
2
1
3
1
2
1
0.5
4
1
2
1
3
1
2
1
0.5
4
1


2
1
3
1
2
1
0.5
4
1


2
1
3
1
2
1
0.5
4
1


2
1
3
1
2
1
0.5
4
1


2
1
3
1
2
1
0.5
4
1

Algo générique

Procedure Recherche − ACM(G)

//G = (S, A, w) : un graphe non − orienté, valué et connexe.

begin

$A' := \emptyset$

tant que *A' n'est pas un arbre couvrant* **faire**

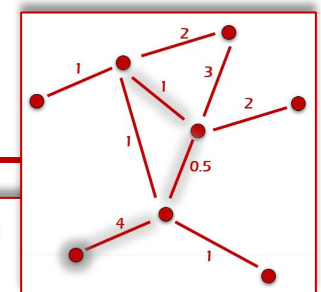
Choisir $(u, v) \in A$ t.q. (u, v) est <u>compatible</u> avec A'

$A' := A' \cup \{(u, v)\}$

retourner A'

end

t.q. $A' \cup \{(u, v)\}$ est <u>inclu</u> dans un ACM.



Théorème

<u>Si</u>

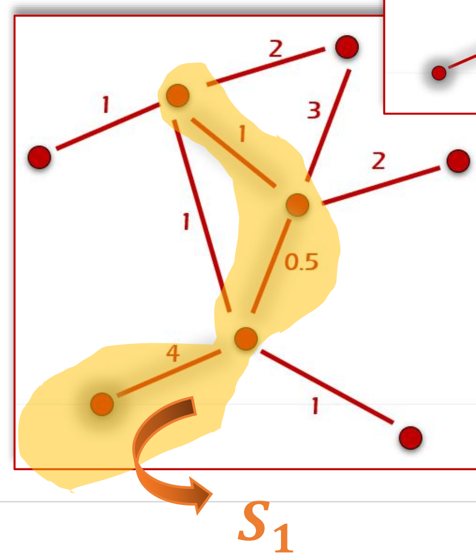
(1) $G = (S, A, w)$ non orienté, valué, connexe.

(2) $A' \subseteq A$ t.q. $\exists$ un ACM contenant A'

(3) (S_1, S_2) partition telle que $\forall (x, y) \in A'$, $(x, y) \in S_1 \cdot S_1$ *ou* $(x, y) \in S_2 \cdot S_2$

(4) $(u, v) \in A$ et $u \in S_1$, $v \in S_2$ et <u>minimale</u>.

Alors (u, v) est compatible avec A'



Algo. de Prim

$\mathbf{Procedure\ Recherche - ACM - Prim(G, s_0)}$

$//G = (S, A, w)$: un graphe non − orienté, valué et connexe.
$//\ s_0 \in S$: un sommet "point de depart".

```
begin
  pour chaque s ∈ S faire
      π[s] := nil
      d[s] := { 0  si s = s0
              { ∞  sinon
      DejaExtrait[s] := ⊥
  A' := ∅
  F := FilePriorité(S,d,IndiceDansF) // Construit F une file de     étendue.
  tant que F ≠ ∅ faire
      s := Extraire-Min(F)
      DejaExtrait[s] := T
      si s ≠ s0 alors A' := A' ∪ {( π[s] , s)}
      pour chaque (s,u) ∈ A faire
          si ( ¬ DejaExtrait[u] ∧ (w(s,u) < d[u] ) alors
              π[u] := s
              d[u] := w(s,u)
              MaJ-F-Prim(F,d,u)
  return A'
end
```

Algo. de Kruskal

$\mathbf{Procedure\ Recherche - ACM - Kruskal(G)}$

$//G = (S, A, w)$: un graphe non − orienté, valué et connexe.

begin

$A' := \emptyset$

pour chaque $s \in S$ **faire**

CréerEnsemble(s)

Trier A par poids w(u,v) croissants.

pour chaque (x,y) $\in A$ **faire**

//On énumère les aretes dans l'ordre du tri…

si Représentant-Ens(x) ≠ Représentant-Ens(y) **alors**

$A' := A' \cup \{(x, y)\}$

Fusion(x, y)

return A'

end

Résolution des formules de Horn

- **Le problème de la satisfiabilité des <u>formules propositionnelles</u>.**
- Variables propositionnelles : $\boldsymbol{x_1} \dots \boldsymbol{x_n}$
- $\land, \lor, \neg, \Rightarrow, \Leftrightarrow, \top, \bot, \dots$

Exemple

$\boldsymbol{x_1} \land (\boldsymbol{x_2} \lor \neg \boldsymbol{x_3}) \lor (\neg \boldsymbol{x_1} \land \neg \boldsymbol{x_2})$

Un littéral : $\boldsymbol{x_i}$ ou $\neg \boldsymbol{x_i}$

φ Calcul prop.

$\exists\, x_1 . \exists\, x_2 . \dots \exists\, x_n .\ \varphi$?

$\exists\, x_1 . \forall\, x_2\ \exists\, x_3\ \forall$

3 SAT : $(\boldsymbol{x_1} \lor \boldsymbol{x_2} \lor \neg \boldsymbol{x_1}) \land (\neg \boldsymbol{x_1} \lor \neg \boldsymbol{x_2} \lor \boldsymbol{x_5}) \dots$

Formules de Horn

= une Conjonction de clauses de Horn (au plus un littéral « positif »)

$$\begin{cases} \overline{x_1} \lor \overline{x_2} \lor \overline{x_5} \lor \overline{x_6} & 0\ positif. \\ x_4 & 0\ neg., 1\ positif. \\ \overline{x_1} \lor \overline{x_3} \lor x_6 \lor \overline{x_4} & des\ neg., 1\ positif \end{cases}$$

$$(\boldsymbol{x_1} \land \boldsymbol{x_3} \land \boldsymbol{x_4}) \Rightarrow \boldsymbol{x_6}$$

Ou :

- Des clauses purement négatives : $(\overline{\boldsymbol{x_1}} \lor \overline{\boldsymbol{x_2}} \lor \overline{\boldsymbol{x_5}})$
- Des implications avec :

$\mathbf{T}$

  - **A gauche** : une conj. (potentielle^{v} vide) de litt. positifs.
  - **A droite** : un litt. positif.

Formules de Horn

Exemple :

$(x_1 \wedge x_4) \Rightarrow x_3$

$\mathrm{T} \Rightarrow x_3$

$x_1 \Rightarrow x_2$

$(x_2 \wedge x_3) \Rightarrow x_4$

$x_1 = \top$

$x_2 = \top$

$x_3 = \bot$

$x_4 = \bot$

$\top = \mathrm{Top} = \mathrm{True}$

$\bot = \mathrm{bottom} = \mathrm{false}$

$\overline{x_1} \vee \overline{x_3} \vee \overline{x_2}$

$\overline{x_4} \vee \overline{x_2}$

Algo

$$\boldsymbol{v} = \{\bot, \dots, \bot\} \qquad [\,\boldsymbol{i.e.}\ \boldsymbol{v}(\boldsymbol{x_i}) = \bot \quad \forall \boldsymbol{i}\,]$$

Tant qu'il y a une <u>implication</u> non satisfaite,
└ mettre la var. à droite de $\Longrightarrow$ à $\top$

Vérifier que toutes les clauses négatives sont vérifiées :

Si oui : la formule est satisfaisable.
Si non : la formule est non satisfaisable.