15.11.21

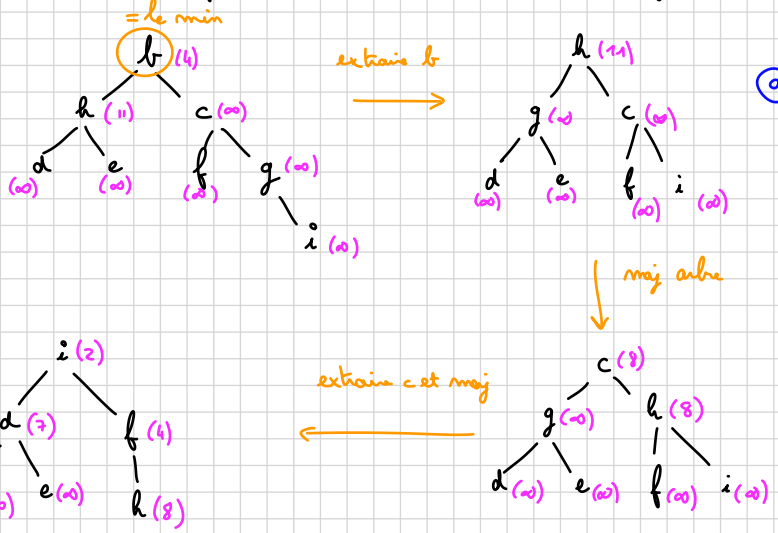
ALS TD8 ACN et Algo de Prim

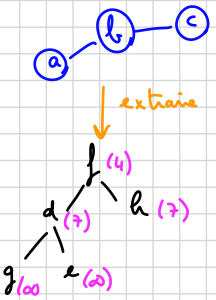
Exercice 1

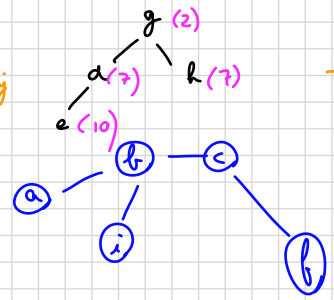
file	a	b	c	d	e	f	g	h	i
	∞ ∅	∞ ∅	∞ ∅	∞ ∅	∞ ∅	∞ ∅	∞ ∅	∞ ∅	∞ ∅
a	0	4 a						11 a	
b, h		4 a	8 b					8 b	
c, h									

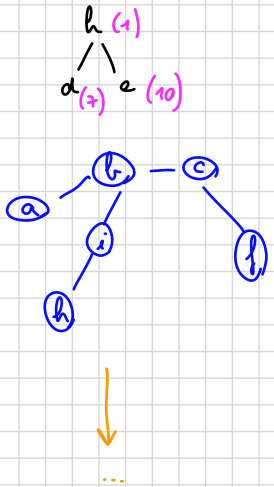
↑ OK représentation → complexité n^2
mais on peut aussi faire ça avec un tas binaire → complexité $n \log(n)$

(bleu = l'ACN en construction)









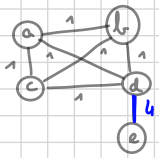
Coût

* extraire min : $\log(n) \times n$
(il faut refaire l'arbre)
* maj tas : $\log(n) \times m$
(on met à jour une clé à chaque fois qu'on regarde une arrête ; et on ne voit chaque arrête qu'une seule fois).

$\Rightarrow m \log(n)$ ($m \geq n$ en général donc on ne garde que ce terme)

Exercice 2

1) FAUX ✓



(a,e) est de poids maximal et est forcément dans l'ACM.

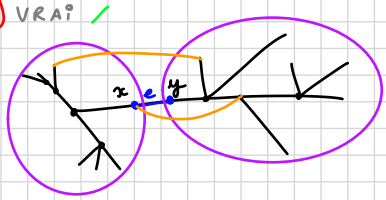
2) VRAI
→ algo de Kruskal : on ordonne les arêtes de façon à prendre e en premier. ✓

3) VRAI ✓ (si on construit l'ACM avec Kruskal, on prendra bien cette arête en premier) → pas vraiment une preuve (sauf si on considère que c'est une notion du cours)

↳ Soit e l'arête unique de poids minimal.
Soit T un ACM.
Supposons par l'absurde que $e \notin E(T)$.
∃ un cycle dans le graphe T+e.
Soit e' une arrête quelconque de ce cycle ($\neq e$).
Par hypothèse de e, $w(e) < w(e')$.
On obtient une contradiction car T+e-e' est un arbre couvrant plus petit que T.

Donc on a bien montré $e \in E(T)$.

4) VRAI ✓



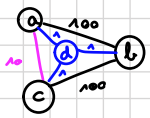
coupe = partitionner les sommets en deux patates.

Si un des arrêtes orange était < e, alors on pourrait faire un ACM - e + orange plus petit que l'ACM avec e.

Formellement : Soit T un ACM et soit $e \in E(T)$
Soient V_1 et V_2 les deux composantes connexes de T-e
Par l'absurde : si il existe $f \neq e$ entre V_1 et V_2 tq $w(f) < w(e)$
alors T-e+f est un ACM de poids strictement inférieur à T ⇒ CONTRADICTION

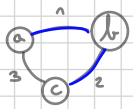
Donc e est minimal dans la coupe.

5) FAUX



Cycle = {a,b,c}
e = (a,c)

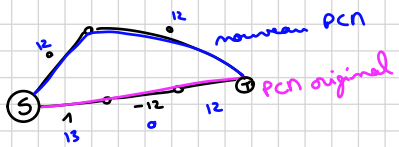
6) FAUX $(a,c) \notin$ ACM ✓



7) FAUX
Dans l'exercice 1, le ⊛ court chemin $a \rightsquigarrow h$ n'est pas dans l'ACM.

8) VRAI
→ on peut se ramener à un graphe OK en ajoutant à tous les poids le poids minimal.
Ça marche parce que tous les arbres sont décalés de la même façon de $\min \times (n-1)$
↑ ce qu'on a ajouté à tous les poids ; ↑ nb d'arêtes

⚠ Cette astuce ne fonctionne pas pour Dijkstra.



Exercice 3

1) OUI ✓

→ on obtient un arbre pour les mêmes raisons que l'algo original.

→ l'arbre est maximal : pour chaque sommet on emprunte l'arrête entrante maximale ✓

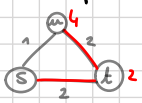
⇒ cela revient à faire Prim sur l'arbre où on a les arrêtes de signe opposé (= les ⊕ grds deviennent les plus petites) ✓

↳ Faire Prim_max sur (G, w) revient à faire Prim sur $(G, -w)$.

Un ACPrim pour (G, w') est un ACMax pour (G, w)

2) NON

Contre-exemple :



$s \to t \to u$ OK max

$s \to t$ ≠ le max

(il faudrait $s \to u \to t$)

En général, trouver les chemins les plus longs n'est pas solvable en tps raisonnable.