PARTIE I

LE TYPE ABSTRAIT "PARTITION D'ENSEMBLES DISJOINTS"

C'EST QUOI LE TYPE ABSTRAIT DU JOUR ?

PARTITION D'ENSEMBLES DISJOINTS

Exemple : $\{\{3,7,1\}, \{2,0\}, \{5\}, \{4,6\}\}$

ou

$\{\{0,1,2\}\}$

RÈGLES

- → ensemble d'ensembles non vides
- → ordre sans importance
- → deux ensembles n'ont jamais un élément en commun
- → chaque entier entre 0 et $n-1$ se trouve dans un ensemble

n = nombre d'éléments

☆ QUELLES OPÉRATIONS RÉALISER SUR CE TYPE ABSTRAIT ?

OPÉRATIONS POUR LES PARTITIONS D'ENSEMBLES DISJOINTS (UNION-FIND)

OPÉRATION 1 CRÉER UNE PARTITION VIDE → $\emptyset$

OPÉRATION 2 AJOUTER UN SINGLETON (= ENSEMBLE DE TAILLE 1)

Le singleton ajouté sera $\{n\}$, où n = nombre d'entiers avant

AVANT: $\{\{3,7,1\},\{2,0\},\{5\},\{4,6\}\}$ APRÈS $\{\{3,7,1\},\{2,0\},\{5\},\{4,6\},\{8\}\}$

OPÉRATION 3 FAIRE L'UNION DE 2 ENSEMBLES étant donnés deux entiers

AVANT: $\{\{3,7,1\},\{2,0\},\{5\},\{4,6\}\}$

ENTRÉES
0 et 4

→ APRÈS $\{\{3,7,1\},\{0,2,4,6\},\{5\}\}$

ENTRÉES
7 et 1

→ APRÈS $\{\{3,7,1\},\{2,0\},\{5\},\{4,6\}\}$
(si entiers dans le même ensemble, on ne fait rien)

~~**OPÉRATION 4** EST-CE QUE 2 ENTIERS APPARTIENNENT AU MÊME ENSEMBLE?~~

~~$\{\{3,7,1\},\{2,0\},\{5\},\{4,6\}\}$~~

ENTRÉES
0 et 4

→ Renvoie Faux

ENTRÉES
7 et 1

→ Renvoie Vrai

Tester si x et y sont dans le même ensemble, c'est pareil que tester Trouver(x) == Trouver(y)

OPÉRATION 4 (+ FORTE) TROUVER UN REPRÉSENTANT DE L'ENSEMBLE

Entrée : Un entier x Sortie : Un entier y dans le même ensemble que x

Contrainte : Les entiers d'un même ensemble doivent avoir le même représentant.

$\{\{3,7,1\},\{2,0\},\{5\},\{4,6\}\}$ Par ex:

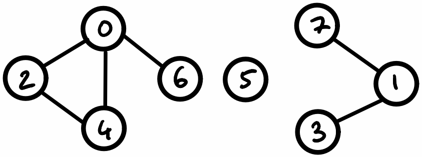
Entier x	0	1	2	3	4	5	6	7
Trouver(x)	2	3	2	3	4	5	4	3

on ne peut pas changer le 3 en 1

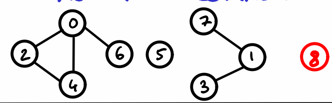
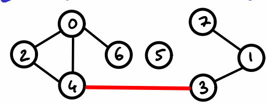
MAIS À QUOI ÇA SERT ?

Pour de l'algo de graphes ! Les ensembles correspondent aux composantes connexes.

GRAPHE



PARTITION {{3,7,1},{0,2,4,6},{5}}

	CHEZ LES PARTITIONS	CHEZ LES GRAPHES
OPÉRATION 1	CRÉER PARTITION VIDE	CRÉER GRAPHE VIDE
OPÉRATION 2	AJOUTER SINGLETON {{3,7,1},{0,2,4,6},{5},{8}}	AJOUTER SOMMET 
OPÉRATION 3	UNION ENTRE ENSEMBLE DE x ET ENSEMBLE DE y {{3,7,1,0,2,4,6},{5}}	AJOUTER ARÊTE ENTRE x et y 
OPÉRATION 4	x et y SONT DANS LE MÊME ENSEMBLE ?	x et y SONT DANS LA MÊME COMPOSANTE CONNEXE ?

En particulier : Compter les composantes connexes, détection de cycle, algo de Kruskal...

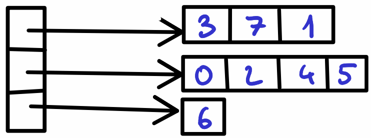
PARTIE II

IMPLÉMENTATIONS SOUS-OPTIMALES

IDÉE 1 : TABLEAU DE TABLEAUX (DYNAMIQUES)

Partition $\{\{3,7,1\},\{0,2,4,5\},\{6\}\}$

En mémoire



	COMPLEXITÉ
Création d'une partition vide	$O(1)$
Ajout d'un singleton	$O(1)^{*}$
Union	$O(n)$
Trouver (un représentant)	$O(n)$

Problème : on ne sait pas où chercher notre élément

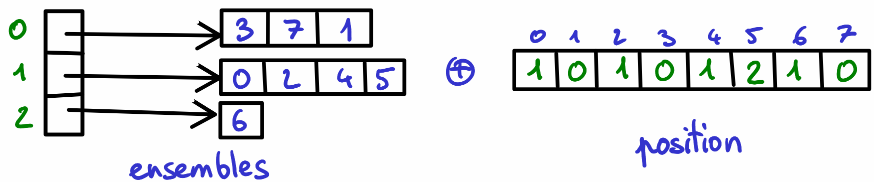
n = nombre d'éléments

* : en amorti

IDÉE 2 : TABLEAU DE TABLEAUX + TABLEAU POSITIONS

Partition $\{\{3,7,1\},\{0,2,4,5\},\{6\}\}$

En mémoire

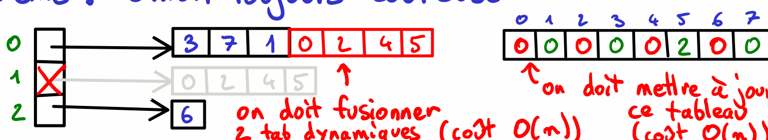


	COMPLEXITÉ
Création d'une partition vide	$O(1)$
Ajout d'un singleton	$O(1)^*$
Union	$O(n)$
Trouver (un représentant)	$O(1)$

$*$: en amorti

Problème : Union toujours coûteuse

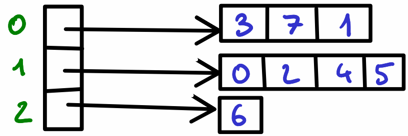
Ex : Union de 3 et 4



IDÉE 3 : LISTE CHAÎNÉE EN TABLEAUX

Partition $\{\{3,7,1\},\{0,2,4,5\},\{6\}\}$

Tableau de tableaux (Idée n°2)

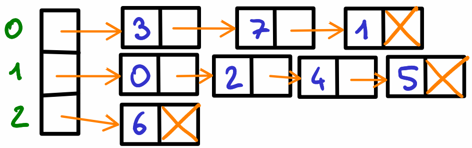


⊕

0	1	2	3	4	5	6	7
1	0	1	0	1	2	1	0

meilleure structure

Tableau de listes chaînées



0	1	2	3	4	5	6	7
1	0	1	0	1	2	1	0

pour une concaténation ⊕ facile ?

meilleure structure

Listes chaînées en tableaux !

	0	1	2	3	4	5	6	7
suivant	2	-1	4	7	5	-1	-1	1
tête_liste	0	3	0	3	0	0	6	3
fin_liste	5	1	5	1	5	5	6	1

IDÉE 3 : LISTES CHAÎNÉES EN TABLEAUX

Partition {{3,7,1},{0,2,4,5},{6}}

Vision "liste chaînée"
(plus compréhensible pour nous)

3 → 7 → 1 ×

6 ×

0 → 2 → 4 → 5 ×

En mémoire

	0	1	2	3	4	5	6	7
suivant	2	-1	4	7	5	-1	-1	1

	0	1	2	3	4	5	6	7
tête_liste	0	3	0	3	0	0	6	3
fin_liste	5	1	5	1	5	5	6	1

	COMPLEXITÉ
Création d'une partition vide	$O(1)$
Ajout d'un singleton	$O(1)^*$
Union	$O(n)$
Trouver (un représentant)	$O(1)$

* : en amorti

Problème : Union toujours toujours coûteuse

Ex : Union de 3 et 4

3 → 7 → 1 → 0 → 2 → 4 → 5 ×

6 ×

	0	1	2	3	4	5	6	7
suivant	2	0	4	7	5	-1	-1	1
tête_liste	3	3	3	3	3	3	6	3
fin_liste	5	5	5	5	5	5	6	5

modification de ce tableau $O(1)$

Ceux-là $O(n)$

IDÉE 4: LISTES CHAÎNÉES EN TABLEAUX ASTUCIEUSES

Partition $\{\{3,7,1\},\{0,2,4,5\},\{6\}\}$

Vision "liste chaînée"
(plus compréhensible pour nous)

3 → 7 → 1 ×   6 ×
0 → 2 → 4 → 5 ×

En mémoire

	0	1	2	3	4	5	6	7
suivant	2	-1	4	7	5	-1	-1	1
tête liste	0	3	0	3	0	0	6	3

	0	1	2	3	4	5	6	7
taille	4	?	?	3	?	?	1	?
fin_liste	5	?	?	1	?	?	6	?

ASTUCE 1 Lors de l'union, on met la liste chaînée la plus petite à la fin de la plus grande

ASTUCE 2 On met à jour la fin et la taille des listes chaînées uniquement pour les têtes de listes.

Exemple

Union de 3 et 4

3 → 7 → 1 ×
0 → 2 → 4 → 5
6 ×

	0	1	2	3	4	5	6	7	
suivant	2	-1	4	7	5	3	-1	1	O(1)
tête liste	0	0	0	0	0	0	6	0	O(taille ⊕ petit)
taille	7	?	?	?	?	?	1	?	O(1)
fin_liste	1	?	?	?	?	?	6	?	O(1)

COMPLEXITÉ DANS LE PIRE DES CAS : O(taille de la ⊕ petite liste chaînée)

IDÉE 4 : LISTES CHAÎNÉES EN TABLEAUX ASTUCIEUSES

Partition $\{\{3,7,1\},\{0,2,4,5\},\{6\}\}$

Vision "liste chaînée"
(plus compréhensible pour nous)

$3 \to 7 \to 1 \times$ $\qquad$ $6 \times$

$0 \to 2 \to 4 \to 5 \times$

En mémoire

	0	1	2	3	4	5	6	7
suivant	2	-1	4	7	5	-1	-1	1
tête_liste	0	3	0	3	0	0	6	3

	0	1	2	3	4	5	6	7
taille	4	?	?	3	?	?	1	?
fin_liste	5	?	?	1	?	?	6	?

	COMPLEXITÉ
Création d'une partition vide	$O(1)$
Ajout d'un singleton	$O(1)^{*}$
Union d'un ensemble de taille l1 et un ensemble de taille l2	$O(\min(l_1, l_2))$ soit $O(\log(n))^{*}$
Trouver (un représentant)	$O(1)$

* : en amorti

POURQUOI COMPLEXITÉ AMORTIE DE L'UNION $= O(\log(n))$ AVEC LES MAINS

Imaginons qu'on a effectué m unions

Après 1 union, on a réuni ≤ 2 entiers ; après 2 unions, on a réuni ≤ 4 entiers...
Donc au bout de m unions on a réuni $\leq 2m$ entiers

On peut considérer que le nombre d'entiers n vérifie $n \leq 2m$
(les autres n'interviennent pas dans la complexité)

En outre, lors de l'union, le plus coûteux est la mise à jour du tableau tête_liste. On néglige le reste.

Autrement dit complexité totale des m unions $\simeq$ nombre de fois qu'une case de tête_liste a été modifiée

$$\boxed{\begin{array}{l}\text{Combien de fois la case de tête\_liste à la position } i \text{ a été modifiée ?}\\ \text{La 1ère fois, } i \text{ passe d'un singleton à un ensemble de taille} \geq 2\\ \text{La 2ième fois, } i \text{ passe d'un ensemble de taille} \geq 2 \text{ à un ensemble de taille } 4,\\ \vdots\\ \text{La } k\text{ième fois, } i \text{ passe d'un ensemble de taille} \geq 2^{k-1} \text{ à un ensemble de taille } 2^k.\\ \text{Comme les ensembles ont une taille} \leq n, \text{ nb de fois que } i \text{ a été modifiée} \leq \log_2(n)\end{array}}$$

Au final complexité totale des m unions $\leq m \log_2(n) \leq 2m \log_2(n)$

$$\boxed{\text{complexité amortie de l'union} \leq \frac{2m\log_2(n)}{m} = O(\log_2(n))}$$

IDÉE 4 : LISTES CHAÎNÉES EN TABLEAUX ASTUCIEUSES

Partition $\{\{3,7,1\},\{0,2,4,5\},\{6\}\}$

Vision "liste chaînée" (plus compréhensible pour nous)

$3 \to 7 \to 1$ × 6 ×

$0 \to 2 \to 4 \to 5$ ×

En mémoire

	0	1	2	3	4	5	6	7
suivant	2	-1	4	7	5	-1	-1	1
tête_liste	0	3	0	3	0	0	6	3

	0	1	2	3	4	5	6	7
taille	4	?	?	3	?	?	1	?
fin_liste	5	?	?	1	?	?	6	?

	COMPLEXITÉ
Création d'une partition vide	$O(1)$
Ajout d'un singleton	$O(1)^*$
Union d'un ensemble de taille $\ell1$ et un ensemble de taille $\ell2$	$O(\log(n))^*$
Trouver (un représentant)	$O(1)$

* : en amorti

ON PEUT FAIRE MIEUX ?!

Ce qui est coûteux, c'est de mettre à jour le représentant de chaque entier

Idée pour plus tard ? Ne mettre à jour que lorsqu'on en a vraiment besoin

PARTIE III

FORÊTS UNION-FIND

FORÊTS UNION-FIND

Partition $\{\{3,7,1\},\{0,2,4,5\},\{6\}\}$

Vision "Forêt"

À chaque ensemble correspond un arbre orienté des noeuds vers la racine

Les représentants sont les racines
(= noeuds dont leurs parents sont eux-mêmes)

On aimerait que chaque noeud pointe vers la racine
mais on le fera uniquement quand on aura l'occasion.

En mémoire

parent

0	1	2	3	4	5	6	7
0	3	0	3	0	2	6	3

taille_arbre

0	1	2	3	4	5	6	7
4	?	?	3	?	?	1	?

tableaux dynamiques

les tailles des arbres (stockées aux positions des racines)

DESCRIPTION DES OPÉRATIONS

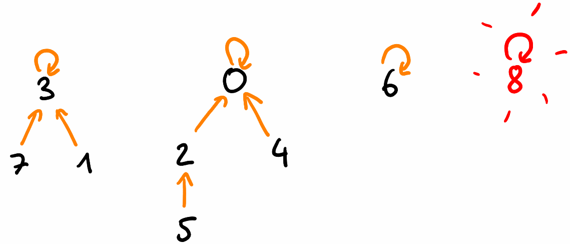
OPÉRATION 1 CRÉER UNE PARTITION VIDE

[] [] 2 tableaux dynamiques vides

parent taille_arbre

OPÉRATION 2 AJOUTER UN SINGLETON

Vision "Forêt"



En mémoire

parent

0	1	2	3	4	5	6	7	8
0	3	0	3	0	2	6	3	8

taille_arbre

0	1	2	3	4	5	6	7	8
4	?	?	3	?	?	1	?	1

DESCRIPTION DES OPÉRATIONS

OPÉRATION 3 — UNION

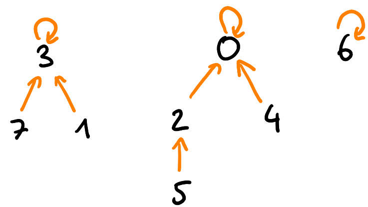
On greffe l'arbre avec le moins de sommets à la racine du plus gros

(on reprend l'astuce de la partie précédente sauf qu'ici c'est pour ne pas trop augmenter la hauteur)

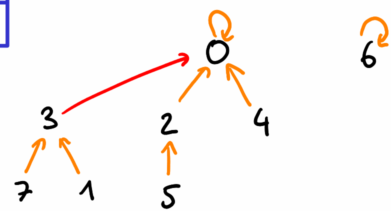
UNION de 3 et 4

Vision "Forêt"

AVANT



APRÈS



En mémoire

AVANT

parent

0	1	2	3	4	5	6	7
0	3	0	3	0	2	6	3

taille_arbre

0	1	2	3	4	5	6	7
4	?	?	3	?	?	1	?

APRÈS

parent

0	1	2	3	4	5	6	7
0	3	0	0	0	2	6	3

taille_arbre

0	1	2	3	4	5	6	7
7	?	?	?	?	?	1	?

D'autres versions de l'union existent: classiquement on peut greffer l'arbre le moins haut à l'arbre le + haut, c'est l'"union par rang"

DESCRIPTION DES OPÉRATIONS

OPÉRATION 3 UNION

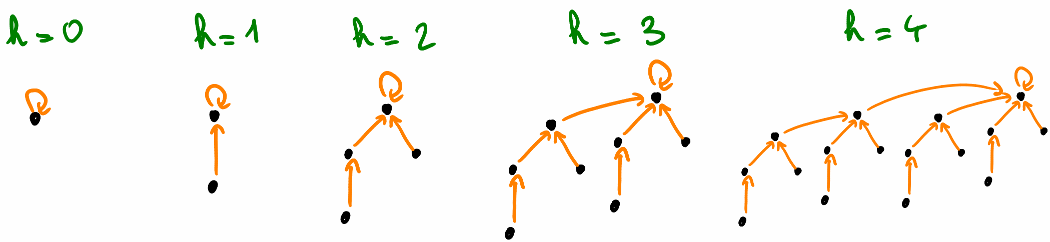
On greffe l'arbre avec le moins de sommets à la racine du plus gros

(on reprend l'astuce de la partie précédente sauf qu'ici c'est pour ne pas trop augmenter la hauteur)

Avec cette contrainte, les arbres de hauteur h ont plus de 2^h noeuds.

(Autrement dit, la hauteur est au plus logarithmique en la taille)

Arbres les plus petits à hauteur fixée :



DESCRIPTION DES OPÉRATIONS

OPÉRATION 4 — TROUVER UN REPRÉSENTANT

Quand on remonte le long d'un arbre pour trouver la racine (=représentant) (y compris lors de l'opération 3 UNION), on fait pointer tous les noeuds rencontrés sur le chemin vers la racine

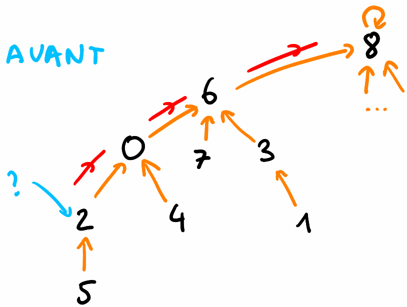
On parle de **compression de chemins**.

Ex :

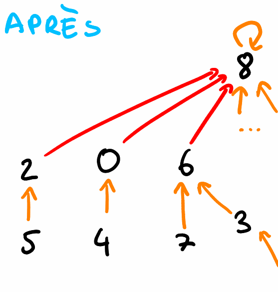
TROUVER
2

Vision "Forêt"

AVANT



APRÈS



En mémoire

AVANT :

parent

0	1	2	3	4	5	6	7	8	
6	3	0	6	0	2	8	6	8	...

taille_arbre

0	1	2	3	4	5	6	7	8	
?	?	?	?	?	?	?	?	23	...

APRÈS :

parent

0	1	2	3	4	5	6	7	8	
8	3	8	6	0	2	8	6	8	...

taille_arbre

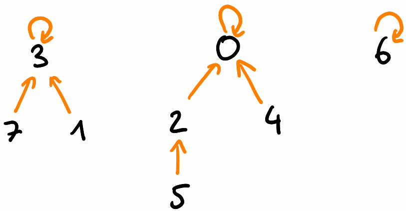
0	1	2	3	4	5	6	7	8	
?	?	?	?	?	?	?	?	23	...

PARTIE IV

COMPLEXITÉ UNION FIND

COMPLEXITÉ DES FORÊTS UNION-FIND

Forêt



En mémoire

parent

0	1	2	3	4	5	6	7
0	3	0	3	0	2	6	3

taille_arbre

0	1	2	3	4	5	6	7
4	?	?	3	?	?	1	?

	COMPLEXITÉ
Création d'une partition vide	$O(1)$
Ajout d'un singleton	$O(1)^*$
Union	$O(\alpha(n))^*$
Trouver (un représentant)	$O(\alpha(n))^*$

* : en amorti

$\alpha(n)$ est une fonction qui croît très (très) lentement :
la fonction inverse d'Ackermann

FONCTION ~~INVERSE~~ D'ACKERMANN : KEZAKO ?

La fonction d'Ackermann est une fonction qui croît très très vite...

On définit $$A_m(n) = \begin{cases} n+1 & \text{si } m=0 \\ \underbrace{A_{m-1} \circ \dots \circ A_{m-1}}_{n+1 \text{ fois}}(1) & \text{sinon} \end{cases}$$

Premières valeurs

$A_0(1) = 2$ $\quad A_0(2) = 3$ $\quad A_0(3) = 4$

$A_1(1) = 3$ $\quad A_1(3) = 4$ $\quad A_1(3) = 5$ —— $A_1(n) = n+2$

$A_2(1) = 5$ $\quad A_2(2) = 7$ $\quad A_2(3) = 9$ —— $A_2(n) = 2(n+3) - 3$

$A_3(1) = 13$ $\quad A_2(3) = 29$ $\quad A_3(3) = 61$ —— $A_3(n) = 2^{n+3} - 3$

—— $A_4(n) = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n+3} - 3$

$A_4(2)$ est plus grand que le nombre d'atomes dans l'univers

La fonction qu'on va considérer est $A(n,n)$

FONCTION INVERSE D'ACKERMANN : KEZAKO ?

La fonction d'Ackermann inverse $\alpha(n)$ est défini comme le plus petit entier k tel que

$$A(k, k) > n$$

En d'autre termes, $\alpha(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } 1 \leq n < 3 \\ 2 & \text{si } 3 \leq n < 7 \\ 3 & \text{si } 7 \leq n < 61 \\ 4 & \text{si } 61 \leq n < \underbrace{A(4,4)}_{\text{ultra grand}} \end{cases}$

Pour tous les n raisonnables, $\alpha(n) \leq 4$

COMPLEXITÉ DES FORÊTS UNION-FIND

	COMPLEXITÉ
Création d'une partition vide	$O(1)$
Ajout d'un singleton	$O(1)^*$
Union	$O(\alpha(n))^*$
Trouver (un représentant)	$O(\alpha(n))^*$

* : en amorti

$\alpha(n)$ est la fonction inverse d'Ackermann

Théorème de Fredman-Saks (89) : On est obligés de faire ça en $\Omega(\alpha(n))$.

Prouvons cette complexité...

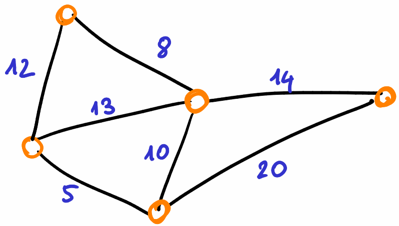
HAHA JE SUIS TROP DRÔLE.

PARTIE V

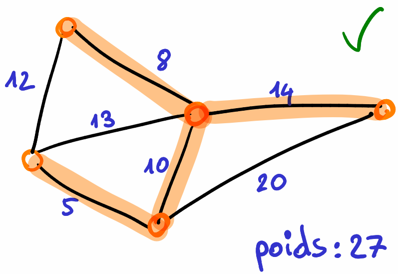
ALGORITHME DE KRUSKAL

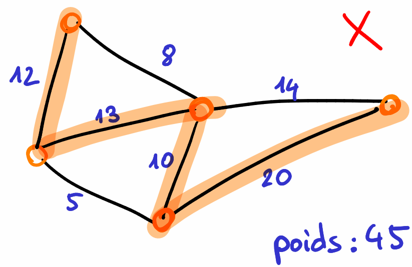
PROBLÈME DE L'ARBRE COUVRANT DE POIDS MINIMAL

<u>Entrée</u> : Graphe non orienté pondéré



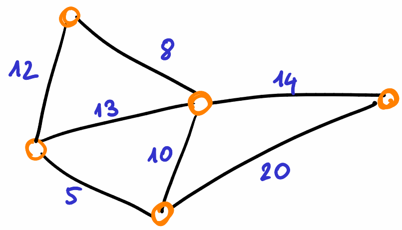
<u>Sortie</u> Arbre couvrant dont la somme des poids est minimale





ALGORITHME DE KRUSKAL

La stratégie gloutonne marche : Il faut prendre les + petites arêtes possibles

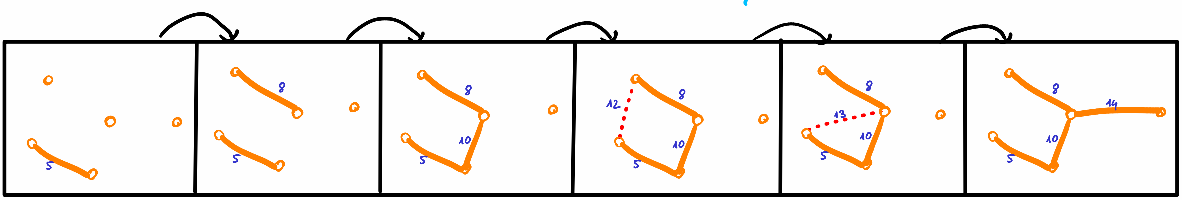


Autrement dit, on construit l'arbre en choisissant les plus petits poids en priorité. On n'ajoute pas l'arête si elle forme un cycle.

$5 < 8 < 10 < 12 < 13 < 14 < 20$

forment des cycles

on peut s'arrêter là



ALGORITHME DE KRUSKAL

On peut utiliser Union-Find pour coder Kruskal !

Complexité

→ Initialiser une partition des sommets en singletons — $O(|S|)$

$+$

→ Trier les arêtes — $O(|A| \log |A|)$

$+$

→ Pour chaque arête,

ajoutez l'arête (avec l'union)

si elle ne forme pas de cycle

(i.e si les extrêmités ne sont pas dans la même composante) (On fait 2 fois "trouver")

— $|A| \times O(\alpha(|S|) + \alpha(|S|))$

$\boxed{\text{COMPLEXITÉ AU FINAL } O(|A| \log |A|)}$