Structures pour ensembles disjoints
(Classe-Union)
(« Union-Find », angl.)

Irena.Rusu@univ-nantes.fr
LINA, bureau 123, 02.51.12.58.16

Sommaire

- Un exemple, pour commencer
- “Union/Find” ou comment gérer les ensembles disjoints
- Représentation à l’aide de tableaux/arbres
- Union pondérée
- Compression de chemins
- Les algorithmes en pratique

Sources : © D. Kaplan (U. Washington), C. Evrendilek (Izmir U. Economy), M.A. Patwary (U. Bergen), J. Blair (U.S. Military Academy), F. Manne (U. Bergen)

Sommaire

- Un exemple, pour commencer
- “Union/Find” ou comment gérer les ensembles disjoints
- Représentation à l’aide de tableaux/arbres
- Union pondérée
- Compression de chemins
- Les algorithmes en pratique

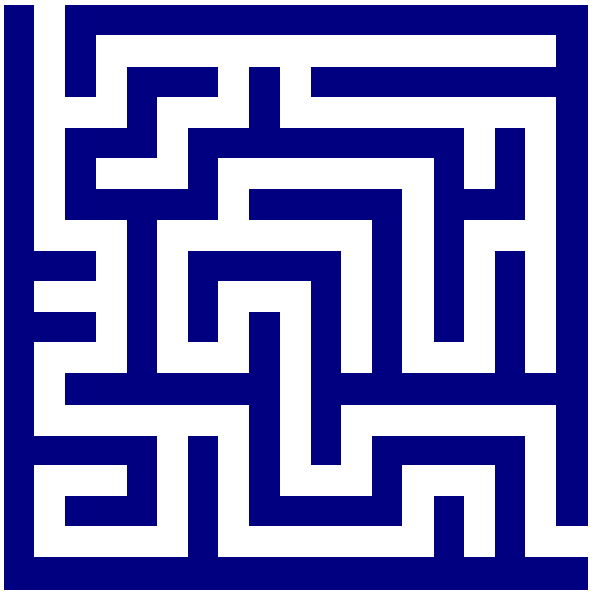
Construction d'un labyrinthe (1)

Labyrinthe **{V,E}**

- Ensemble de pièces: **V**
- Portes entre les pièces (toutes fermeés initiallement): **E**

Construire un labyrinthe tel que :

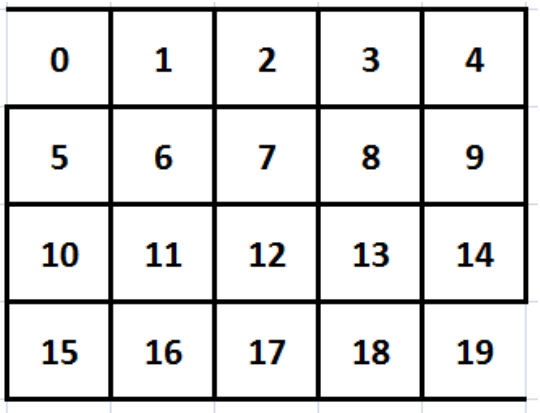
- toutes les pièces de V soient connectées en ouvrant des portes
- Une pièce soit désignée comme entrée, **i**$\in$**V**, et une autre comme sortie, **o**$\in$**V**
- l'ensemble **E'** $\subseteq$ **E** des portes ouvertes assure l'existence d'un chemin unique entre toute paire de pièces

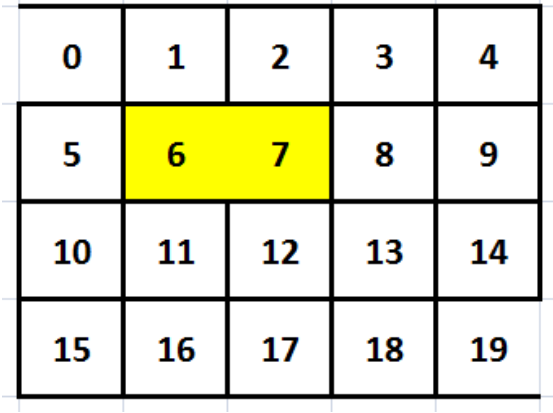


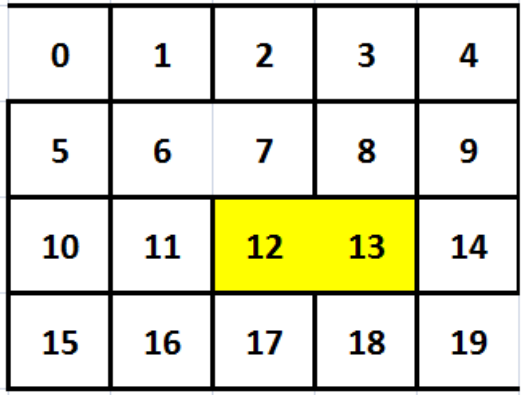
Ici, 1 pièce = 1 pixel blanc/bleu
1 porte ouverte ssi pixels voisins blancs

Construction d’un labyrinthe (2)

- Idées :
  - Ouvrir au hasard une porte
  - En s’assurant que les deux pièces qu’elle sépare ne sont pas connectées par ailleurs
  - … Jusqu’à connecter toutes les pièces







Construction d'un labyrinthe (3)

tant que (il reste des cases non-connectées) **faire**

soit (A,B) de E-E'

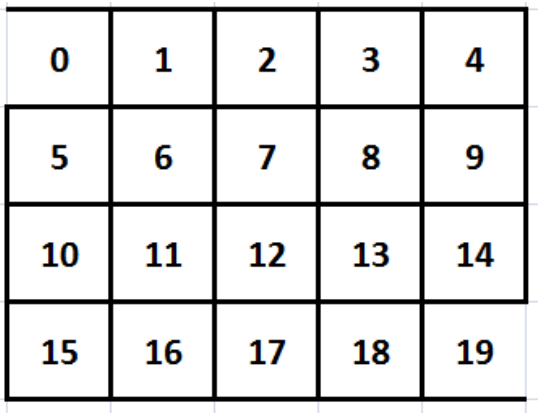
si A, B non connectées **alors**

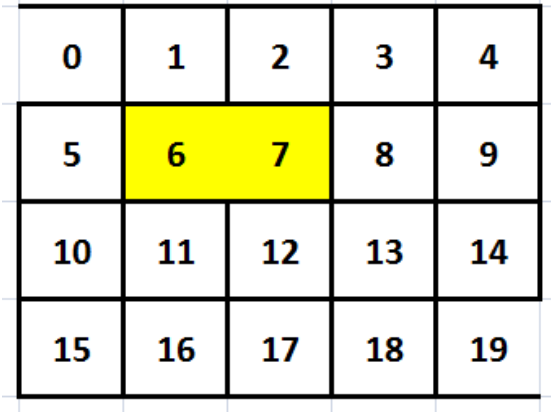
E'← E' ∪ {(A,B)}

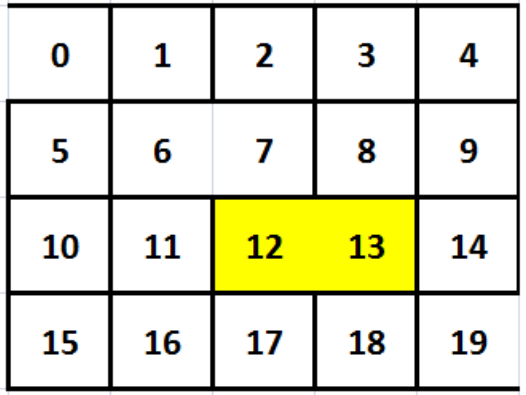
finsi

fintantque

Comment tester la connexion ?

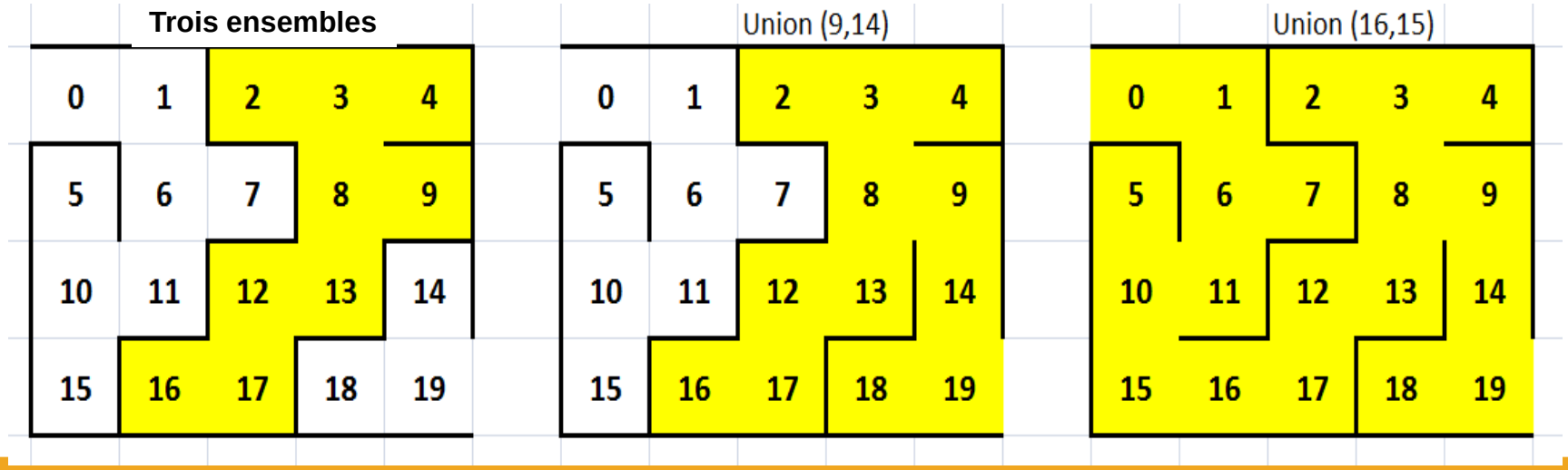






Construction d'un labyrinthe (3)

- Remarques sur l'exemple :
  - Une partie connectée est un ensemble de cases
  - Les ensembles sont disjoints
  - Ouvrir une porte = réunir deux ensembles disjoints
  - Tester une connexion = vérifier si appartenance au même ensemble
  - Tester s'il reste des cases non-connectées = vérifier si on a un seul ensemble ou plusieurs



Sommaire

- Un exemple, pour commencer
- “Classe-Union” ou comment gérer les ensembles disjoints
- Représentation à l’aide de tableaux/arbres
- Union pondérée
- Compression de chemins
- Les algorithmes en pratique

Gestion d'ensembles disjoints

- Collection S = {S_1, …, S_k} d'ensembles disjoints dynamiques (= classes).
- Chaque classe a un représentant.
- **Opérations (anglais : *M*ake-set, *U*nion, *F*ind) :**
  - Créer(x): crée l'ensemble {x}, dont x est le représentant.
  - Union(x, y): réunit les ensembles contenant les éléments x et y.
  - Classe(x): retourne le représentant de la classe contenant x
- **Complexité de chaque opération** En fonction de
  - n = nombre d'operations Créer (ç.a.d. du nombre d'éléments)
- **Complexité globale de plusieurs opérations** En fonction de n et
  - m = nombre total d'opérations Union et Classe.
- **Note:** $m \geq n$, puisqu'on ne crée que ce qu'on utilise.

Exemple

Au début (les représentants sont soulignés) :

S={{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}} (10 appels à Créer)

Après quelques opérations d'union :

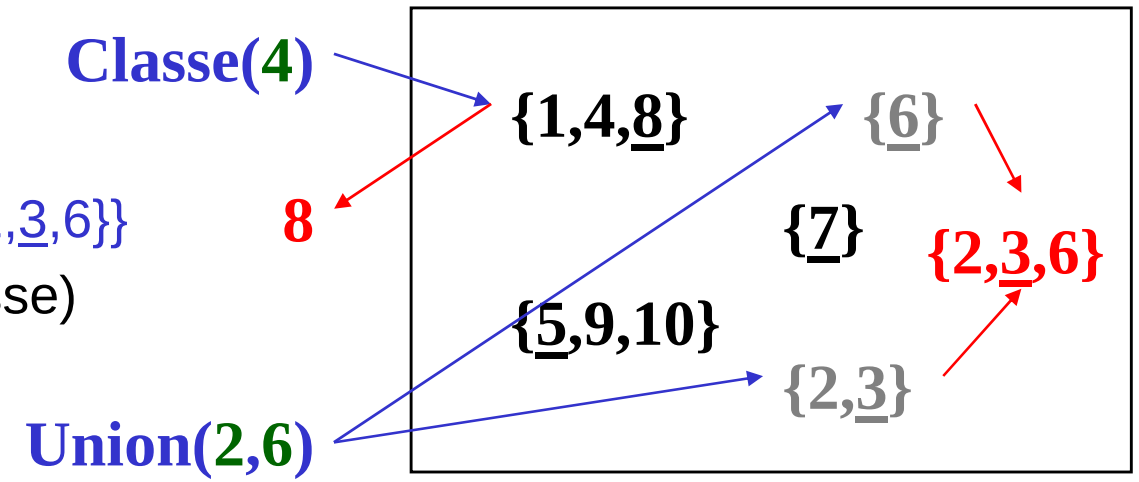
S={{1, 4, 8}, {6}, {5, 9, 10}, {7}, {2,3}}

Union(2,6):

S={{1, 4, 8},

{5, 9, 10}, {7}, {2,3,6}}

(fait 2 appels à Classe)



Retour au labyrinthe (1)

Etat initial : 1 classe par case

{a}{b}{c}{d}{e}{f}{g}{h}{i}

Algorithme :

tant que (il reste des cases non-connectées) **faire**

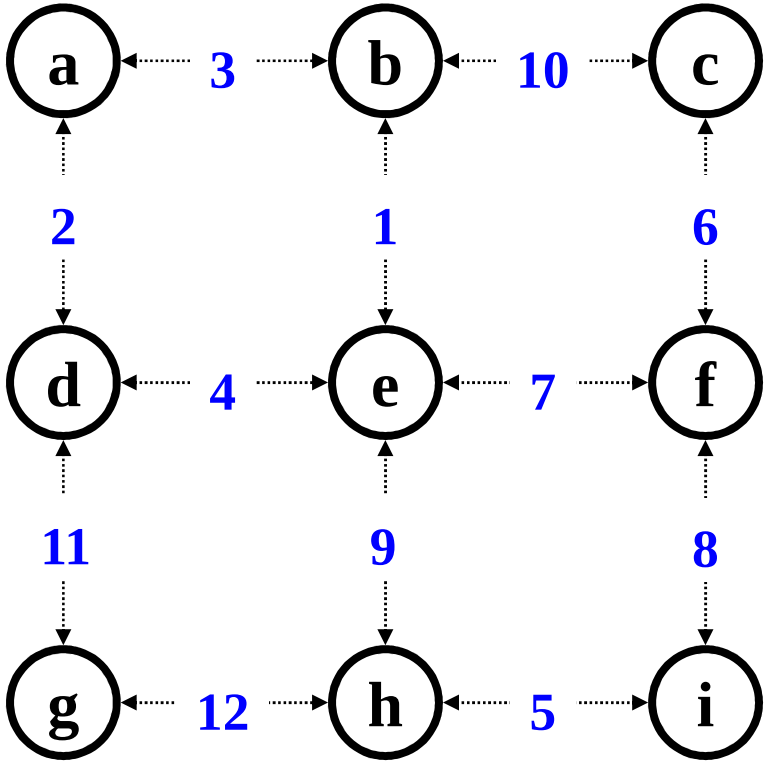
soit (A,B) de E-E'

si A, B non connectées **alors**

E'← E' ∪ {(A,B)}

finsi

fintantque



Pris au hasard, disons selon l'ordre de la figure

Retour au labyrinthe (2)

Etat initial : 1 classe par case

{a}{b}{c}{d}{e}{f}{g}{h}{i}

Algorithme :

tant que (nombre de classes > 1) **faire**

soit (A,B) de E-E'

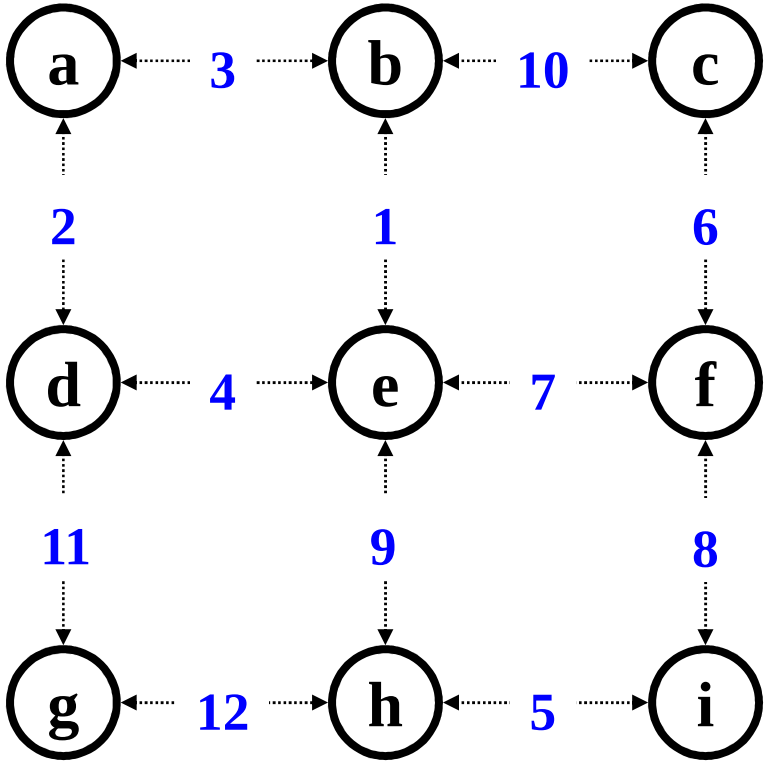
si classe(A)≠classe(B) **alors**

union(A,B)

E'← E' ∪ {(A,B)}

finsi

fintantque



Pris au hasard, disons selon l'ordre de la figure

Une étape: (b,e)

{a}{b}{c}{d}{e}{f}{g}{h}{i}

classe(b) ⇒ b

classe(e) ⇒ e

classe(b) ≠ classe(e) donc:

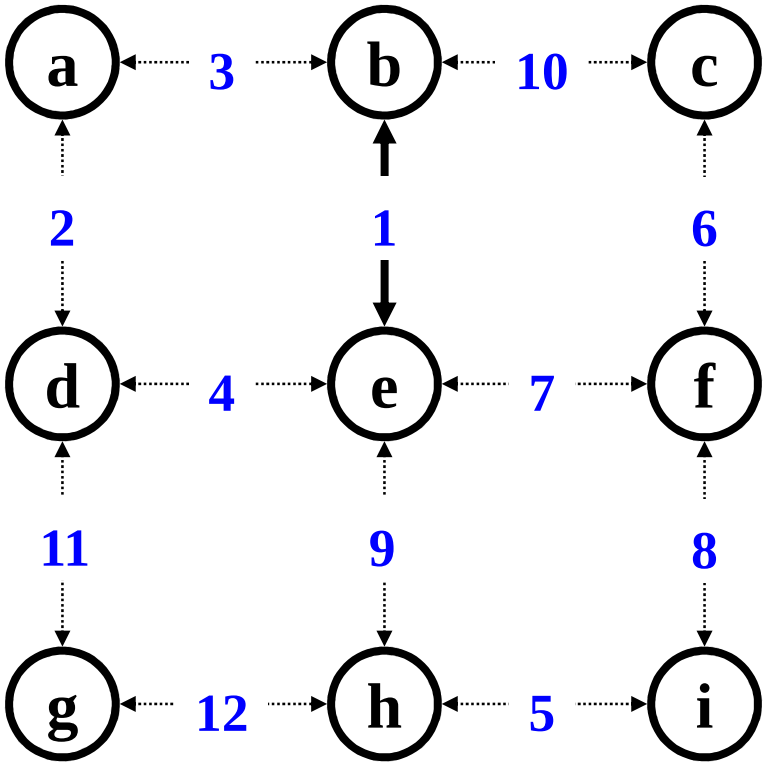
ajouter (b,e) to **E′**

union(b, e)

Résultat: {a}{b,e}{c}{d}{f}{g}{h}{i}

Ensuite : {a,d}{b,e}{c}{f}{g}{h}{i}

{a,b,d,e}{c}{f}{g}{h}{i} etc.



Besoin d'une représentation efficace

Sommaire

- Un exemple, pour commencer
- “Classe-Union” ou comment gérer les ensembles disjoints
- Représentation à l’aide de tableaux/arbres
- Union pondérée
- Compression de chemins
- Les algorithmes en pratique

Une première idée : par tableaux

	a	b	c	d	e	f	g
`Classe`	**1**	**1**	**2**	**2**	**3**	**3**	**4**

pour {a,b} {c,d} {e,f} {g}

`Union` des classes (disjointes) de *p* et *q*

```
{       x ← Classe [p] ;  y ← Classe [q] ;
        pour k ← 1 à n faire
                si Classe [k] = y alors
                        Classe [k] ← x ;
}
```

Complexité

`Classe` : O(1)

`Union` : O(*n*)

	a	b	c	d	e	f	g
`Classe`	**1**	**1**	**1**	**1**	**5**	**5**	**7**

pour {a,b,c,d} {e,f} {g}

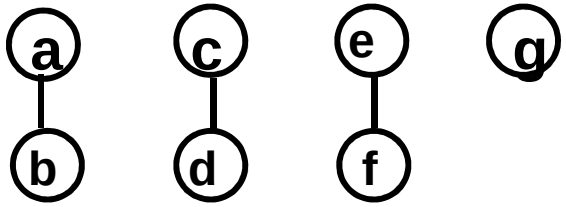
(après Union(b,c))

Une meilleure idée : par arbres

partition {a,b} {c,d} {e,f} {g}

Classe,Taille a,2 c,2 e,2 g,1

arbres



	a	b	c	d	e	f	g
P	-	a	-	c	-	e	-

(contient l'élément du père)

```
Classe(i){
        k ← i ;
        tant que P[k] défini faire k ← P[k] ;
        retour(Classe[k]);}
```

Complexité

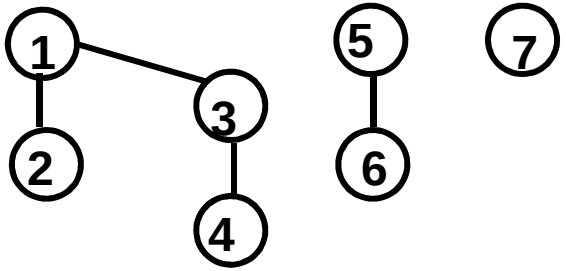
Classe : O(*n*)

Union : O(1)

partition {a,b,c,d} {e,f} {g}

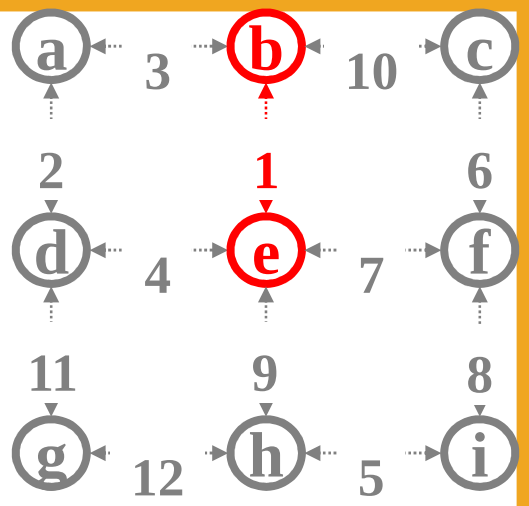
Classe,Taille a,4 e,2 g,1

arbres

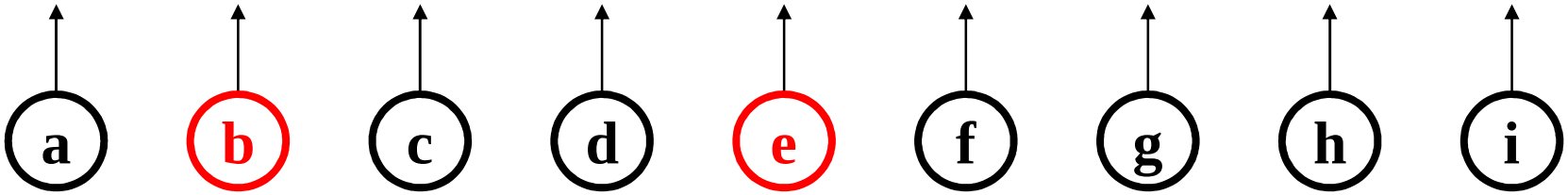


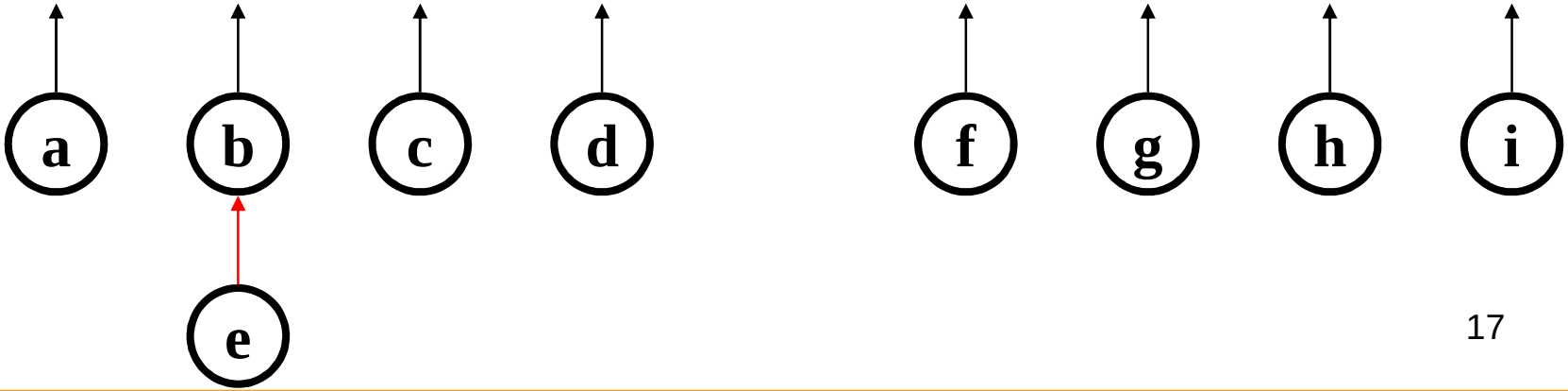
	a	b	c	d	e	f	g
P	-	a	a	c	-	e	-

Exemple (1/11)

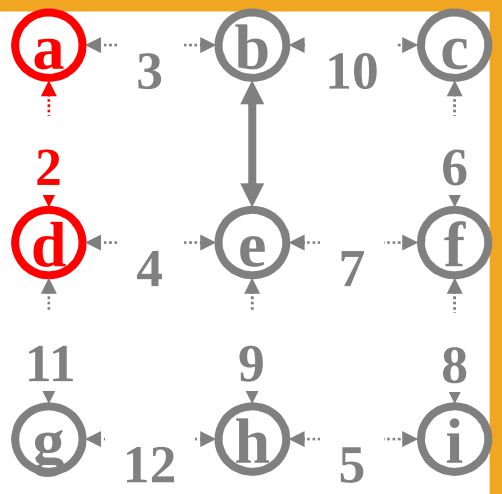


Union(b,e)

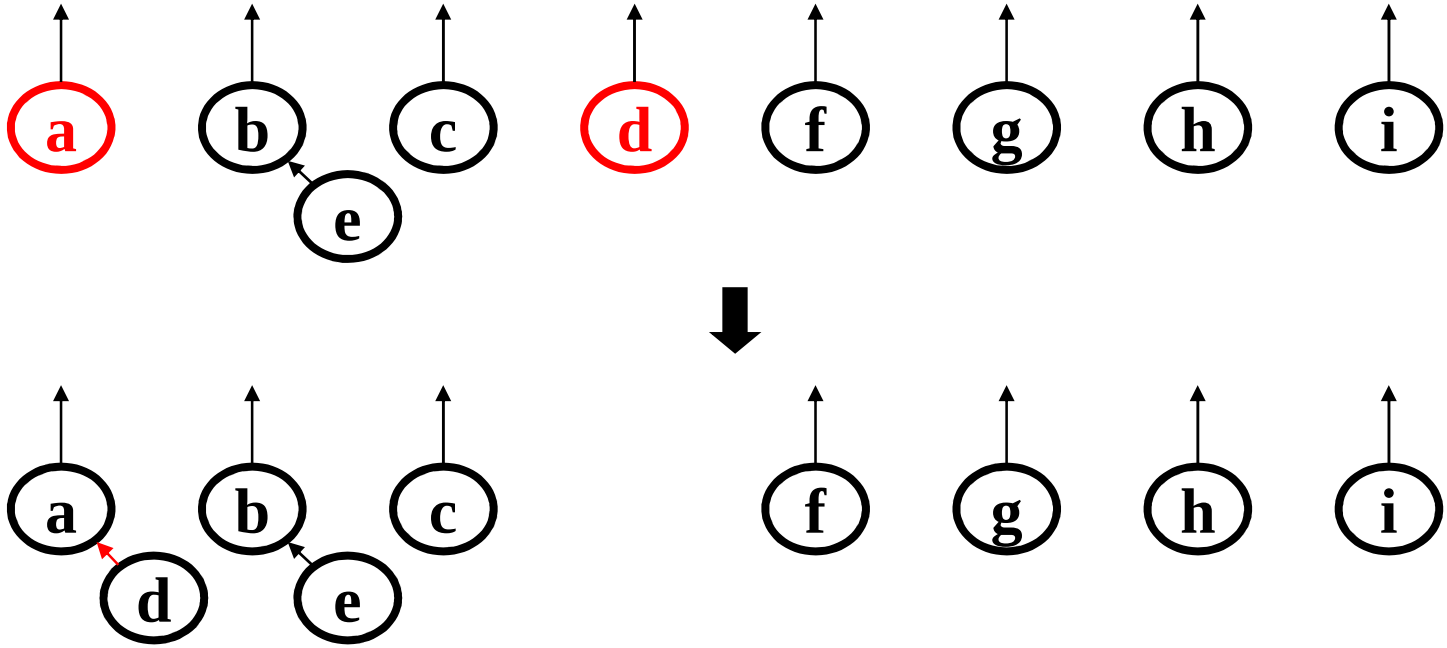




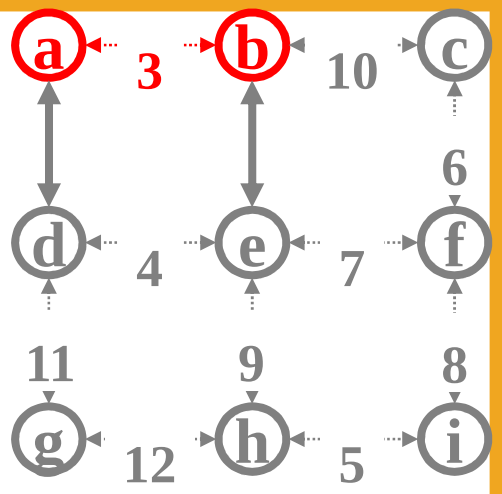
Exemple (2/11)



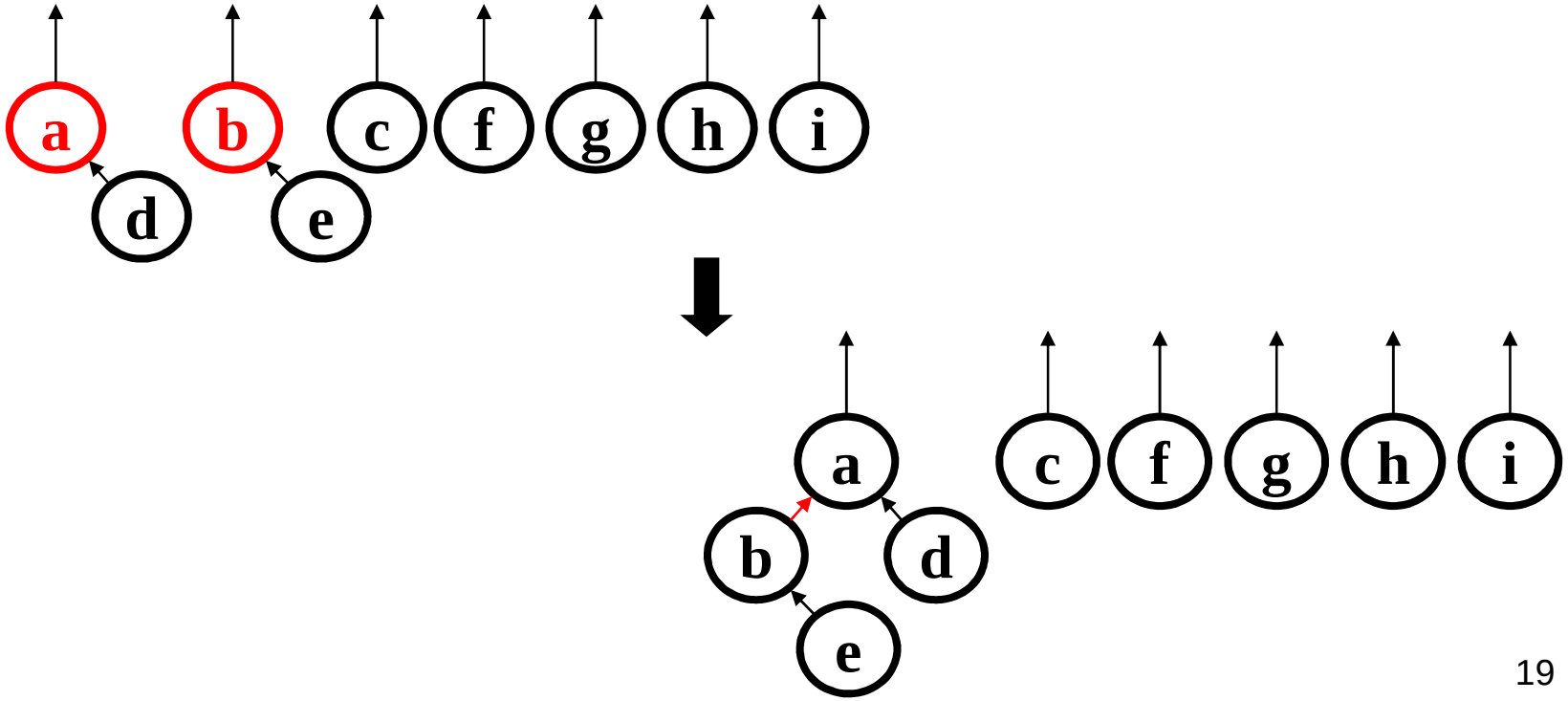
Union(a,d)



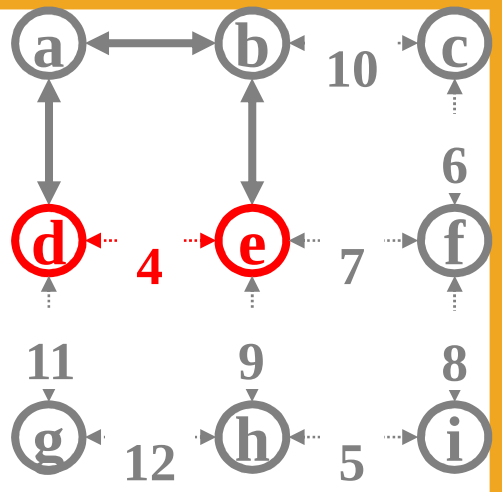
Exemple (3/11)



Union(a,b)

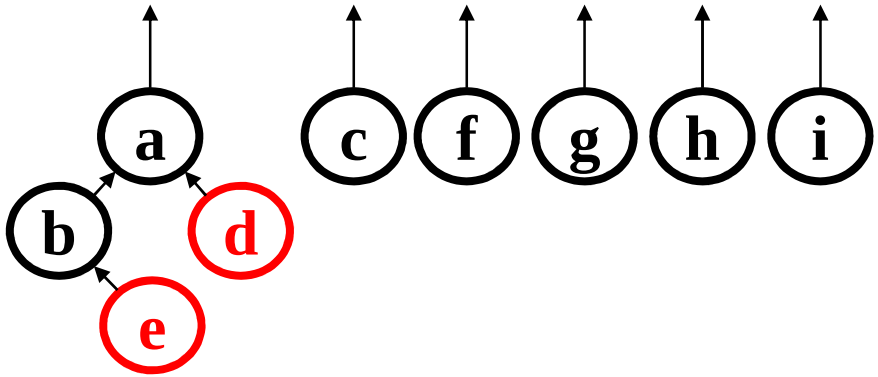


Exemple (4/11)

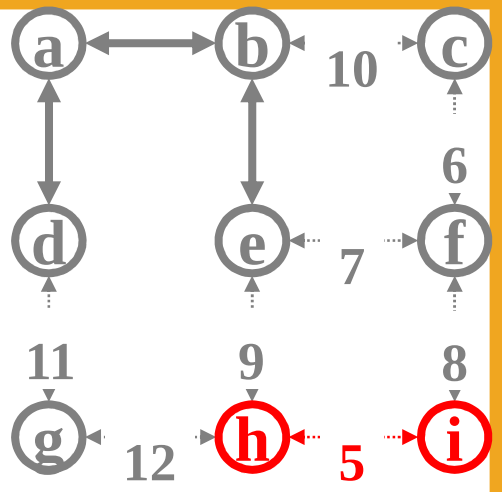


Classe(d) = Classe(e)

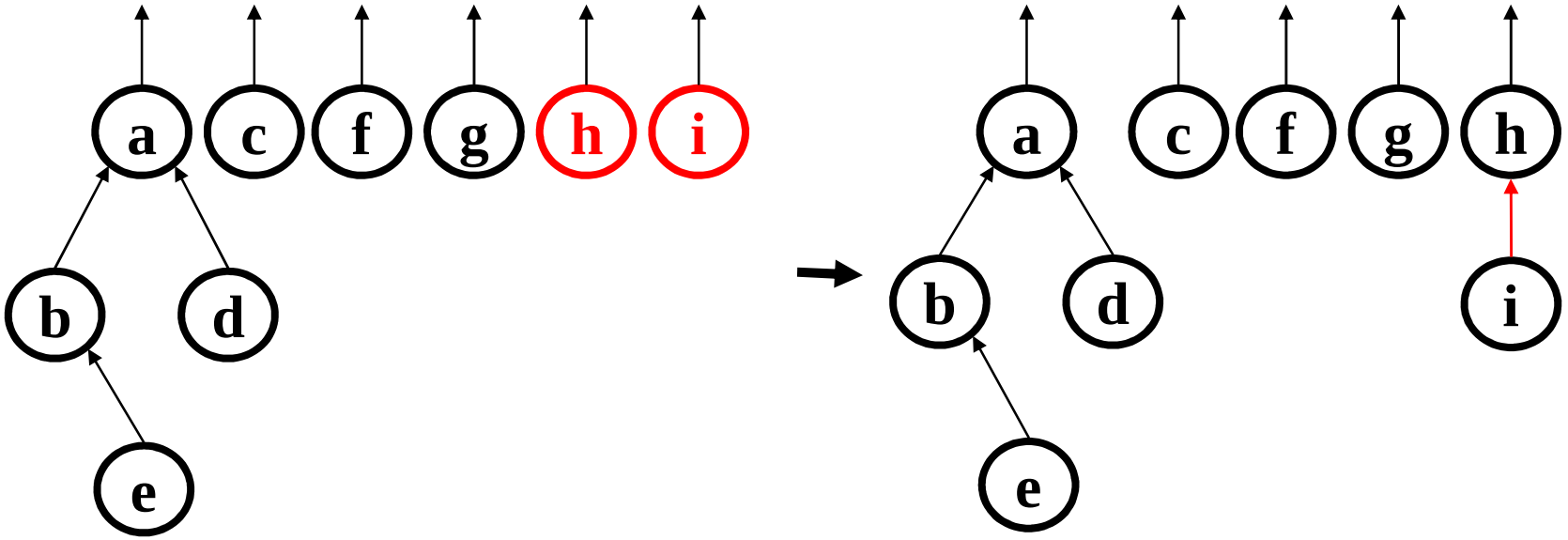
Pas d'union!



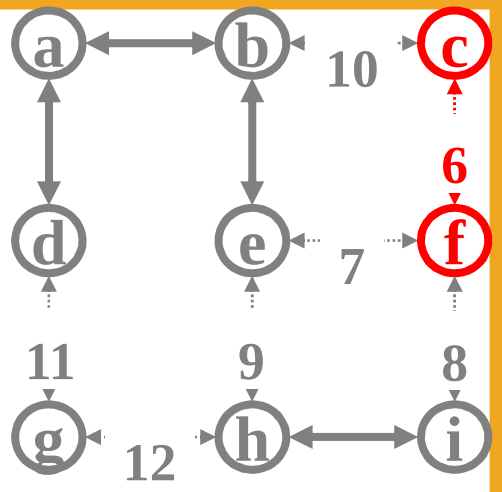
Exemple (5/11)



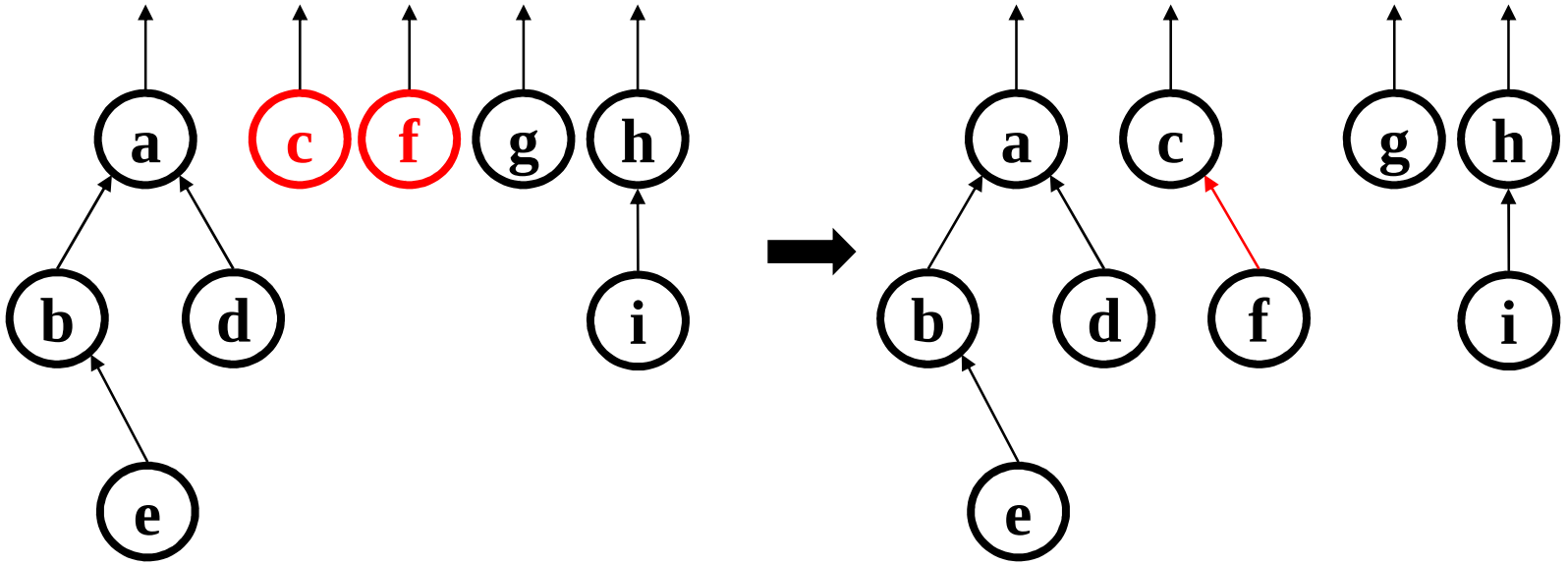
Union(h,i)



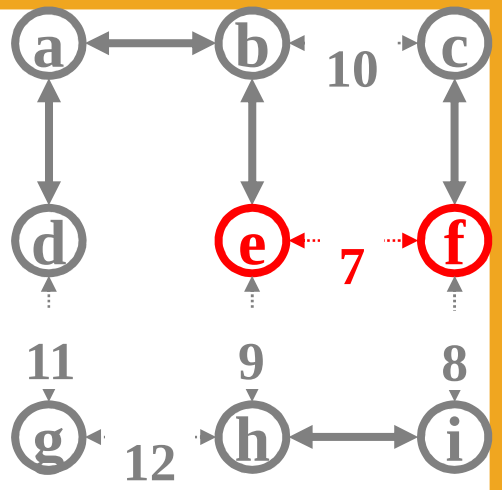
Exemple (6/11)



Union(c,f)



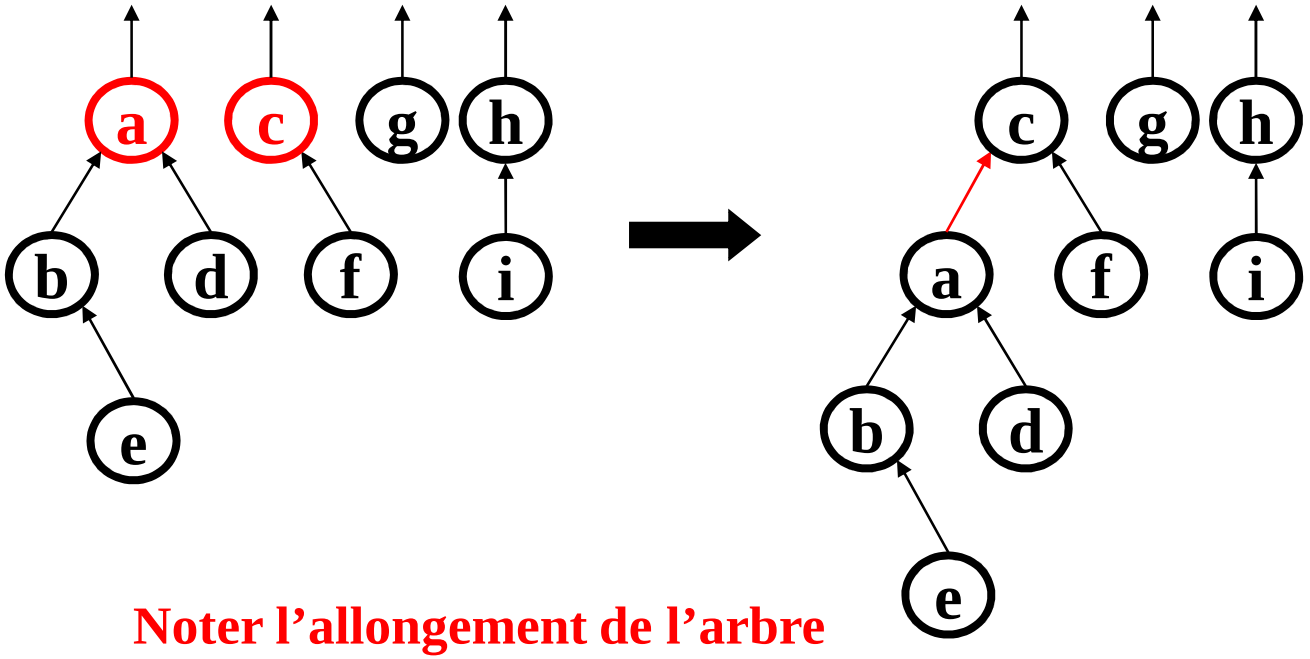
Exemple (7/11)



Classe(e) = a

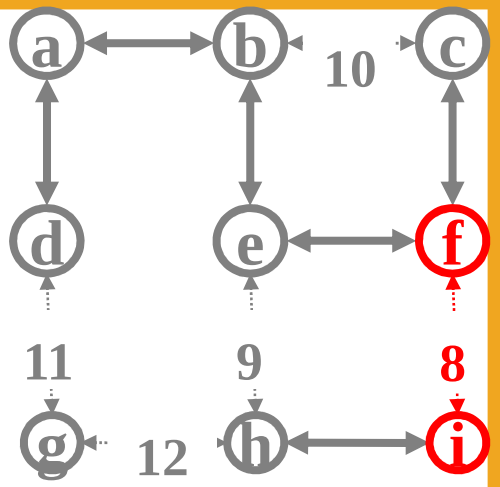
Classe(f) = c

Union(e,f) réunit les arbres de a et de c



Noter l'allongement de l'arbre

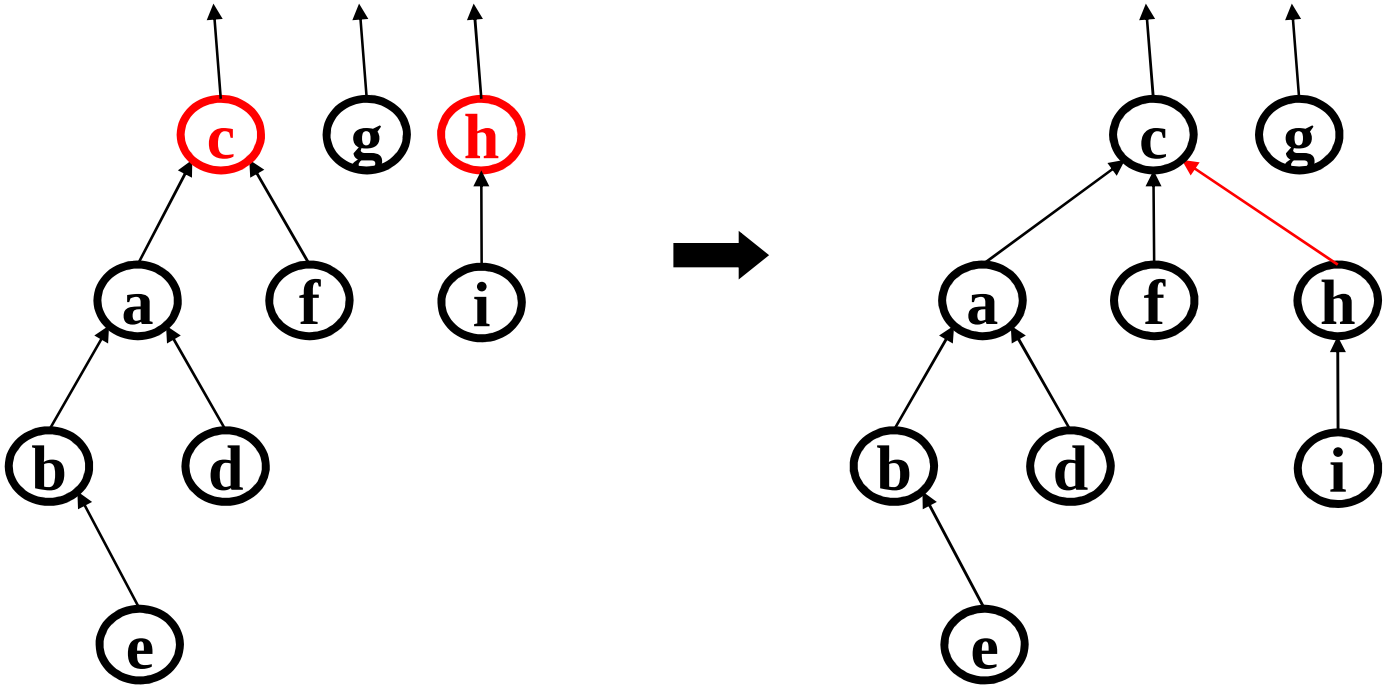
Exemple (8/11)



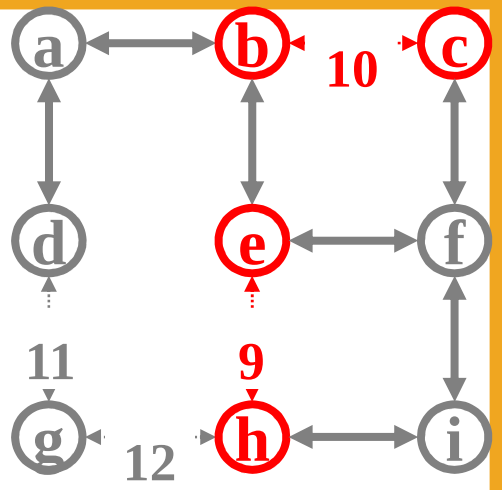
Classe(f)=c

Classe(i)=h

Union(f,i) réunit les arbres de c et de h

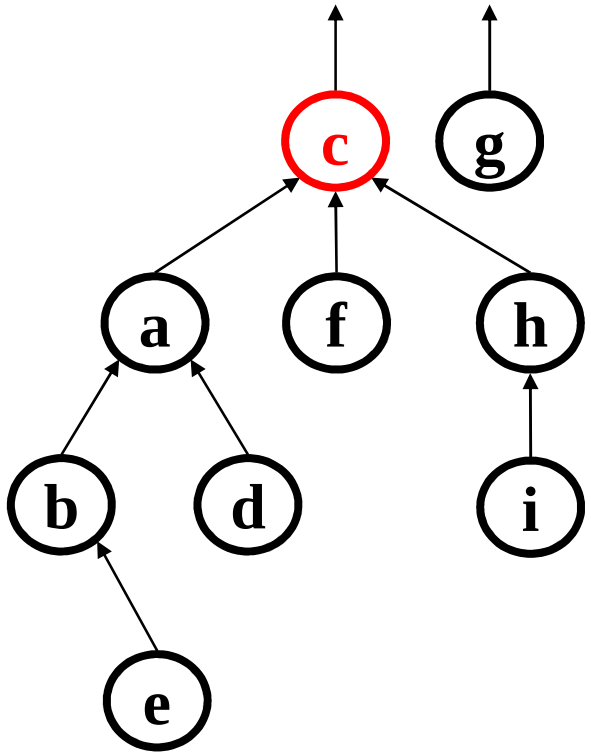


Exemple (9/11)

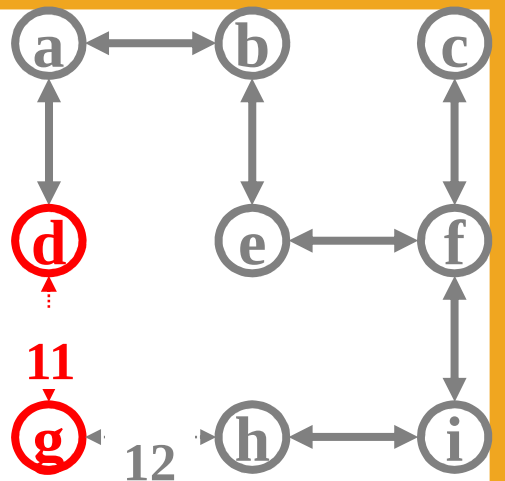


Classe(e) = Classe(h) et Classe(b) = Classe(c)

Pas d'union pour ces deux choix.



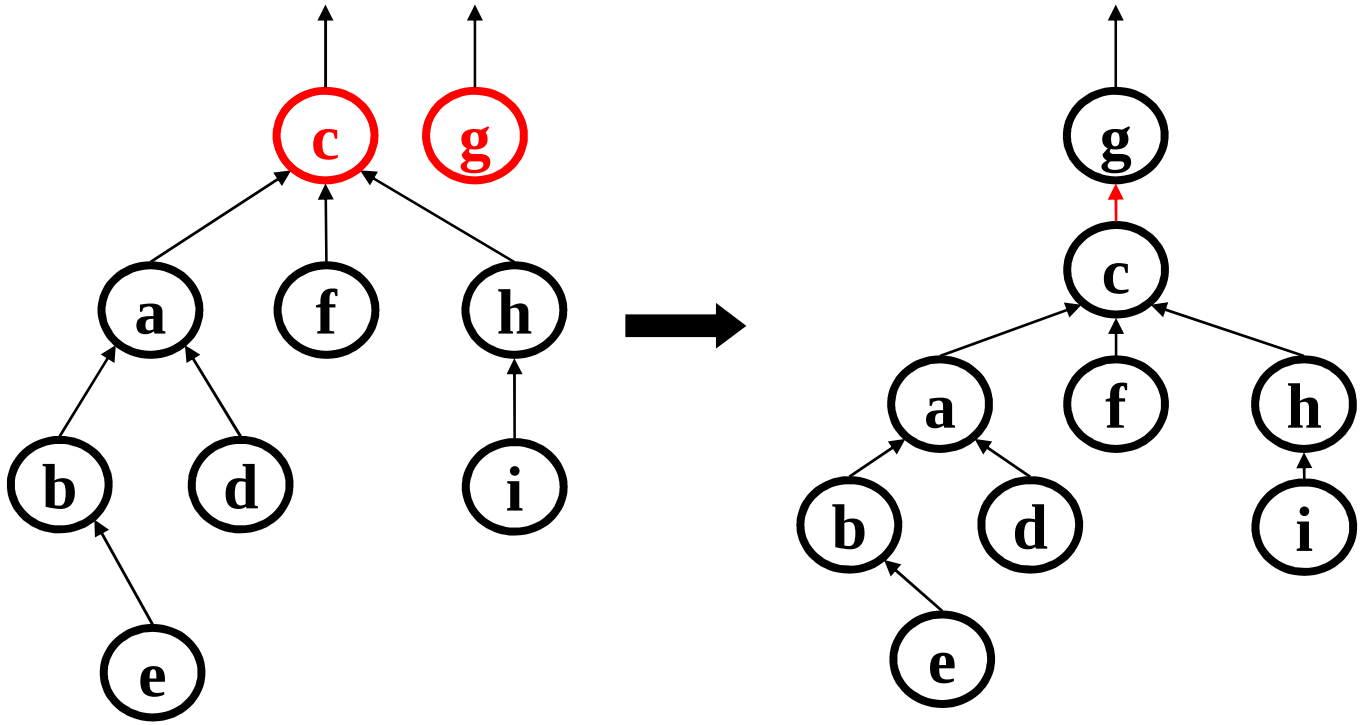
Exemple (10/11)



Classe(d)=c

Classe(g)=g

Union(d, g) réunit les arbres de c et de g

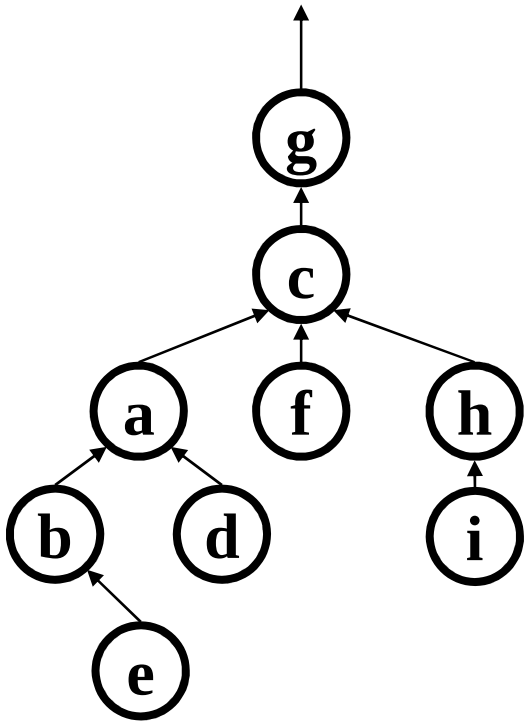


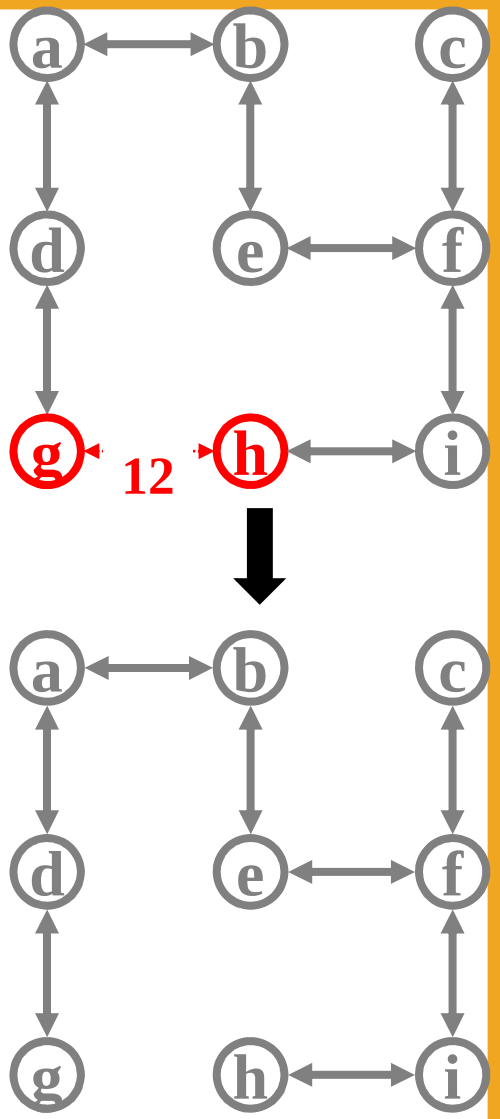
Exemple (11/11)

Classe(g) = Classe(h)

Pas d'union

Comme il ne reste qu'une classe, on a fini.





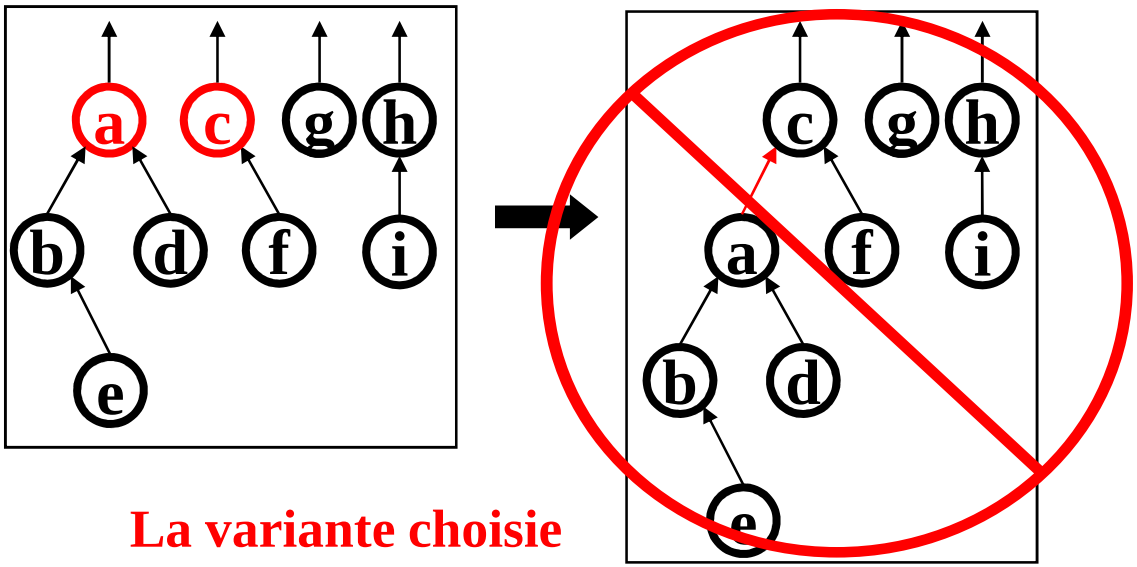
Le labyrinthe ainsi trouvé.

Sommaire

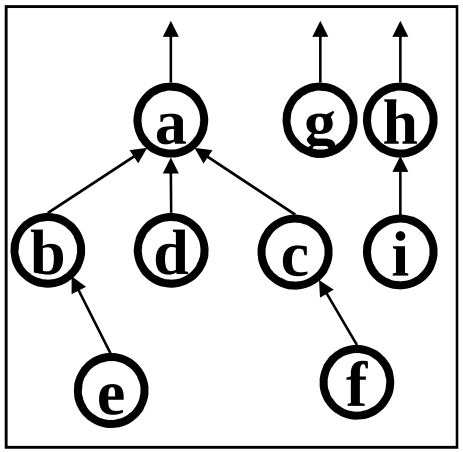
- Un exemple, pour commencer
- “Classe-Union” ou comment gérer les ensembles disjoints
- Représentation à l’aide de tableaux/arbres
- Union pondérée
- Compression de chemins
- Les algorithmes en pratique

Observation

- L'union de deux arbres influe sur la complexité de Classe()
- Hauteur plus importante de l'arbre → Classe() moins efficace



La variante choisie allonge l'arbre



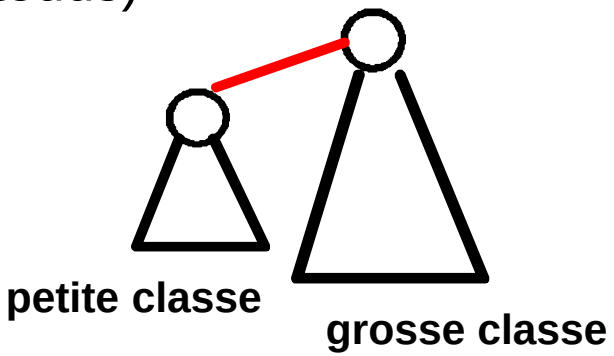
Meilleure variante: garde la hauteur de l'arbre a.

Amélioration de l'Union : Union par rang

Éviter des arbres filiformes pour réduire le temps de calcul de Classe(*i*)

Stratégie pour Union : toujours mettre le petit arbre enfant de la racine du gros (en termes du nombre de nœuds)

Complexité
Classe : O(log*n*)
Union : O(1)



Preuve

On montre que la hauteur de l'arbre à la fin (et donc à chaque moment) est en O(log n).

Le niveau(*i*) d'un élément dans son arbre augmente de 1 quand

on fait l'union de *P* et *Q*,avec card *P* ≤ card *Q* et $i \in P$

i.e., quand la taille de la classe de *i* double au moins.

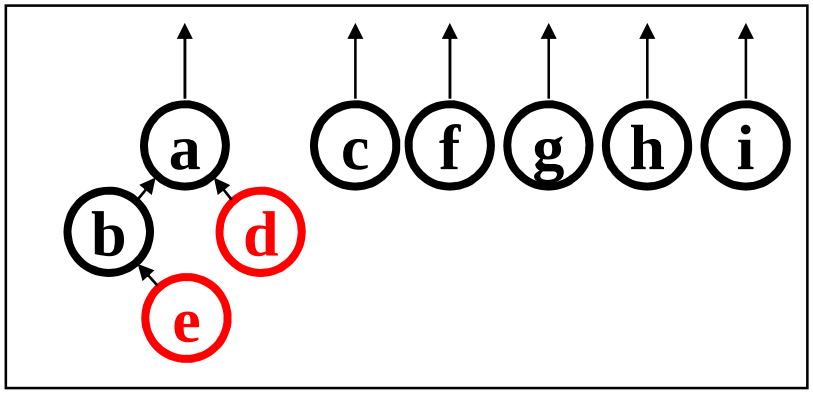
Ceci ne peut arriver que $\lfloor \log_2 n \rfloor$ fois au plus

Sommaire

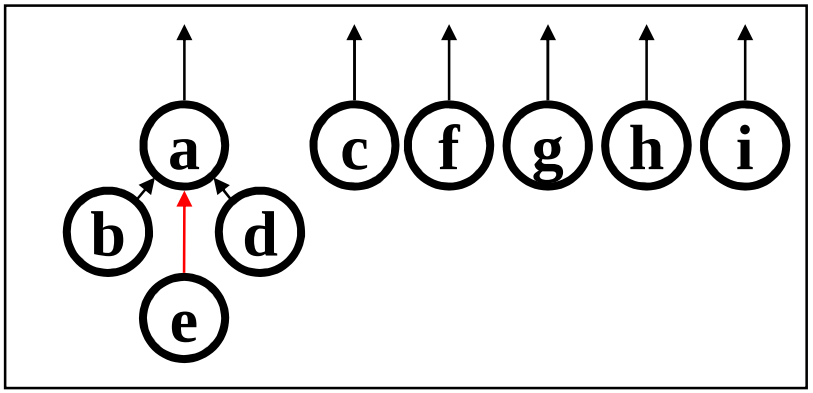
- Un exemple, pour commencer
- “Classe-Union” ou comment gérer les ensembles disjoints
- Représentation à l’aide de tableaux/arbres
- Union pondérée
- Compression de chemins
- Les algorithmes en pratique

Observation

- L'appel à Classe() produit un parcours du noeud vers la racine
- Pour tous les noeuds du chemin on calcule leur classe en même temps
- Mais on ne garde pas ces informations.



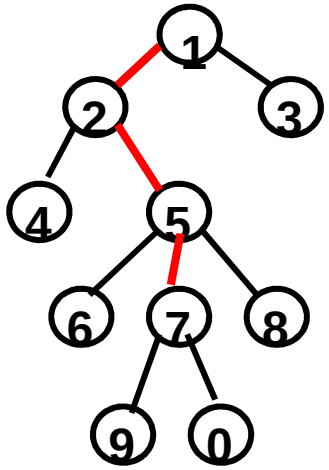
On appelle Classe(e) mais on ne garde pas l'information Classe(e)=a



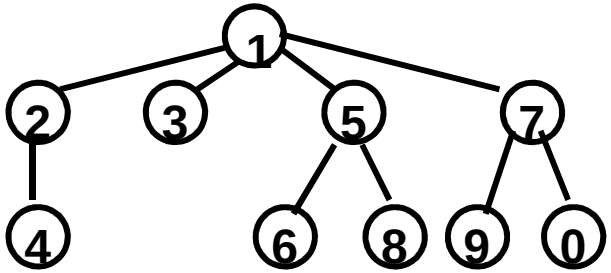
Accrocher e directement à a permettrait d'avoir Classe(e) en O(1) au prochain appel de Classe(e).

Amélioration de Classe : compression de chemins

Stratégie pour Classe(i) : accrocher à la racine tous les nœuds rencontrés sur le chemin vers la racine.



après calcul de `Classe(7)`:

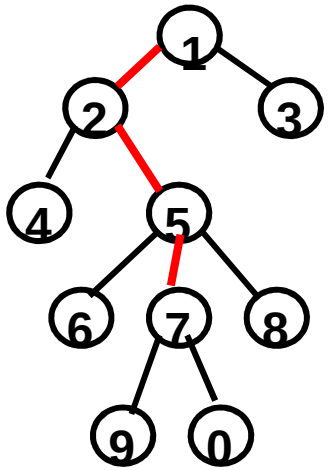


Complexité ?

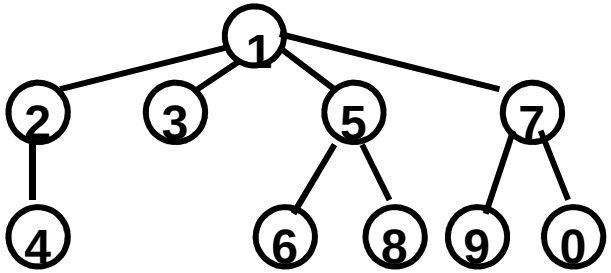
- Évidemment Classe() n'est toujours pas en O(1)
- Alors qu'Union() l'est toujours

Complexité amortie (1)

Idée : faire le calcul de complexité globalement sur plusieurs opérations du même type, et voir la moyenne pour une opération.



après calcul de `Classe(7)`:



Classe(7)=O(longueur du chemin vers la racine)

Mais : pour tout nœud x de ce chemin le calcul de Classe(x) va se faire ensuite en O(1)

Donc: payer plus cher une première fois → des économies par la suite

Complexité amortie (2)

- Temps de n calculs de `Classe` : $O(n\,\alpha(n))$

  où $\alpha(n)$ est le plus petit entier k tel que $n \leq 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ (tour de 2 en k exemplaires)

  ***k* fois**

- Preuve [Aho, Hopcroft, Ullman, 1974]
- $\alpha(n)$ est très-très-très petit par rapport à n. En pratique, il peut être assimilé à une constante. La complexité amortie de `Classe` est donc en $0(\alpha(n))$, donc quasi-constant.
- Exemple : $\alpha(2^{65536})=5$ (nombre à 20 000 chiffres, alors que le nombre d'atomes dans l'univers est estimé à $O(2^{300})$)
- En théorie, $\alpha(n)\neq$cst, donc $O(n\,\alpha(n)) \neq O(n)$

Conclusion jusqu'ici

- Avec union par rang (complexité au pire) :
  - Union en O(1)
  - Classe en O(log n)
- Avec union par rang et compression de chemins (complexité amortie) :
  - Union en O(1)
  - Classe en O($\alpha(n)$), presque constant

En théorie, meilleur algorithme (équivalent à 2 ou 3 autres, mais pas dépassé)

- La pratique ?!

Sommaire

- Un exemple, pour commencer
- “Classe-Union” ou comment gérer les ensembles disjoints
- Représentation à l’aide de tableaux/arbres
- Union pondérée
- Compression de chemins
- Les algorithmes en pratique

Un test sur des réseaux

- Problème résolu : connectivité d'un réseau
- Machine :Dell, Intel Core 2 CPU (2.40 GHz), Fedora 10.
- Trois types de réseaux (= graphes) :
  - rw: 9 graphes "réels"
    - Médicine
    - Ingénierie
    - Industrie automobile.
  - sw: 5 graphes dits "small-world" (tous les sommets sont relativement proches les uns des autres), engendrés par un programme
  - er: 6 graphes sans structure particulière, engendrés par un programme de manière aléatoire, sur un modèle dit d'Erdös-Rényi
- Pour chaque graphe : 5 exécutions avec 5 ordres différents

Propriétés structurelles des graphes

Graph	$\|V\|$	$\|E\|$	Comp	Max Deg	Avg Deg	# Edges Processed
rw1 (m_t1)	97,578	4,827,996	1	236	99	692,208
rw2 (crankseg_2)	63,838	7,042,510	1	3,422	221	803,719
rw3 (inline_1)	503,712	18,156,315	1	842	72	5,526,149
rw4 (ldoor)	952,203	22,785,136	1	76	48	7,442,413
rw5 (af_shell10)	1,508,065	25,582,130	1	34	34	9,160,083
rw6 (boneS10)	914,898	27,276,762	1	80	60	11,393,426
rw7 (bone010)	986,703	35,339,811	2	80	72	35,339,811
rw8 (audikw_1)	943,695	38,354,076	1	344	81	10,816,880
rw9 (spal_004)	321,696	45,429,789	1	6,140	282	28,262,657
sw1	50,000	6,897,769	17,233	6,241	276	6,897,769
sw2	75,000	12,039,043	9,467	8,624	321	12,039,043
sw3	100,000	16,539,557	34,465	10,470	331	16,539,557
sw4	175,000	26,985,391	43,931	14,216	308	26,985,391
sw5	200,000	34,014,275	68,930	16,462	340	34,014,275
er1	100,000	453,803	24	25	9	453,803
er2	100,000	1,650,872	1	61	33	603,141
er3	500,000	2,904,660	8	30	12	2,904,660
er4	1,000,000	5,645,880	31	31	11	5,645,880
er5	500,000	9,468,353	1	70	38	3,476,740
er6	1,000,000	20,287,048	1	76	41	7,347,376

Id. graphe.	Nb. sommets	Nb. arêtes	Nb. composantes	Degré max.	Degré moyen	Nb. arêtes traitées

Diverses implémentations de Classe-Union

CL	Compression Technique							
Union	NF	PC	PH	PS	CO	R0	R1	SP
NL								X
LR		❶	❷					X
LS								X
Rem			X		X	X	X	❸
TVL			X		X	X	X	

Table: 29 variations of classical algorithms. Each cell is an algorithm.

En rouge : variante avec union par rang (LR) et compression des chemins (PC)

En bleu: variante avec union par rang (LR) et un autre type de compression (PH) (non vue en cours)

En vert : algorithme de Rem (1976), non vu en cours et légèrement oublié …

Comparaison d’algorithmes

- Un algorithme X domine un algorithme Y si X a des résultats au moins aussi bons que Y en terme de temps d’exécution, sur chaque graphe.

- Les variantes rouge et bleue sont les meilleures d’un point de vue théorique

Diverses implémentations de Classe-Union

CL	Compression Technique							
Union	NF	PC	PH	PS	CO	R0	R1	SP
NL								X
LR		❶	❷					X
LS								X
Rem			X		X	X	X	❸
TVL			X		X	X	X	

Table: 29 variations of classical algorithms. Each cell is an algorithm.

En rouge : variante avec union par rang (LR) et compression des chemins (PC)

En bleu: variante avec union par rang (LR) et un autre type de compression (PH) (non vue en cours)

En vert : algorithme de Rem (1976), non vu en cours et légèrement oublié …

Variante rouge gagne souvent

CL	Compression Technique							
Union	NF	PC	PH	PS	CO	R0	R1	SP
NL	$LRPC_1$				$LRPC_1$	$LRPC_1$	$LRPC_1$	✗
LR	$LRPC_1$	❶	❷			$LRPC_1$	$LRPC_1$	✗
LS	$LRPC_1$	$LRPC_1$				$LRPC_1$	$LRPC_1$	✗
Rem	$LRPC_1$		✗		✗	✗	✗	❸
TVL	$LRPC_1$	$LRPC_1$	✗		✗	✗	✗	

Table: LRPC dominates 14 algorithms.

En rouge : variante avec union par rang (LR) et compression des chemins (PC)

En bleu: variante avec union par rang (LR) et un autre type de compression (PH) (non vue en cours)

En vert : algorithme de Rem (1976), non vu en cours et légèrement oublié …

Variante bleue gagne encore plus souvent

CL	Compression Technique							
Union	NF	PC	PH	PS	CO	R0	R1	SP
NL	$LRPC_1$	$LRPH_2$			$LRPC_1$	$LRPC_1$	$LRPC_1$	X
LR	$LRPC_1$	❶ $LRPH_2$	❷			$LRPC_1$	$LRPC_1$	X
LS	$LRPC_1$	$LRPC_1$				$LRPC_1$	$LRPC_1$	X
Rem	$LRPC_1$		X		X	X	X	❸
TVL	$LRPC_1$	$LRPC_1$	X		X	X	X	$LRPH_2$

Table: LRPH dominates 3 additional, including LRPC - Total 17.

En rouge : variante avec union par rang (LR) et compression des chemins (PC)

En bleu: variante avec union par rang (LR) et un autre type de compression (PH) (non vue en cours)

En vert : algorithme de Rem (1976), non vu en cours et légèrement oublié …

L'algorithme de Rem est le meilleur en pratique

CL	Compression Technique							
Union	NF	PC	PH	PS	CO	R0	R1	SP
NL	LRPC$_1$	LRPH$_2$	RemSP$_3$	RemSP$_3$	LRPC$_1$	LRPC$_1$	LRPC$_1$	×
LR	LRPC$_1$	❶ LRPH$_2$	❷ RemSP$_3$	RemSP$_3$	RemSP$_3$	LRPC$_1$	LRPC$_1$	×
LS	LRPC$_1$	LRPC$_1$	RemSP$_3$	RemSP$_3$	RemSP$_3$	LRPC$_1$	LRPC$_1$	×
Rem	LRPC$_1$	RemSP$_3$	×		×	×	×	❷
TVL	LRPC$_1$	LRPC$_1$	×	RemSP$_3$	×	×	×	LRPH$_2$

Table: RemSP dominates 10 of remaining, including LRPH - Total 27.

En rouge : variante avec union par rang (LR) et compression des chemins (PC)

En bleu: variante avec union par rang (LR) et un autre type de compression (PH) (non vue en cours)

En vert : algorithme de Rem (1976), non vu en cours et légèrement oublié …

Améliorations possibles de l'existant

Reference	# of Algorithms	Recommended Algorithm	REMSP improves by
[Liu, 1990]	2	NLPC	56%
[Gilbert et al., 1994]	6	NLPH	45%
[Wassenberg et al., 2008]	8	LRCO	24%
[Wu et al., 2009]	3	LIPC	48%
[Hynes, 1998]	18	LICO, LSCO	28%, 24%
[Osipov et al., 2009]	2	IPC-LRPC	29%
-	-	LRPC	52%
-	-	LRPH	28%

Réf. application

Algorithme proposé comme étant le plus adapté

Gain avec l'algo oublié de Rem

Conclusion

- Structure de données simple, et simple à implémenter (du moins dans la version union par rang et compression de chemins)
- Analyse de la complexité difficile
- En pratique, temps constant pour chaque opération
- En théorie, pour le pire des cas, Classe() en O(logn), Union() en O(1).
- Beaucoup d'applications : connexion des réseaux, collections de pages Web, organisation des pixels sur une photo, assemblage de puzzles, jeux (Go, Hex) etc.