



INFO0501

ALGORITHMIQUE AVANCÉE

COURS 4

ARBRES COUVRANTS DE POIDS MINIMAL



UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

Pierre Delisle
Département de Mathématiques, Mécanique et Informatique
Septembre 2019

Plan de la séance

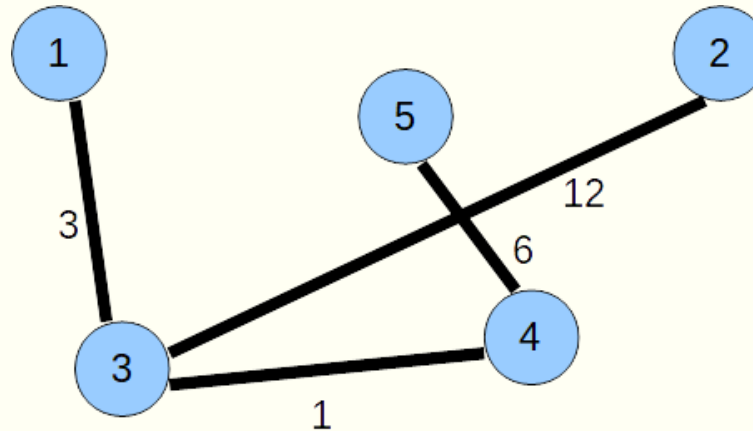
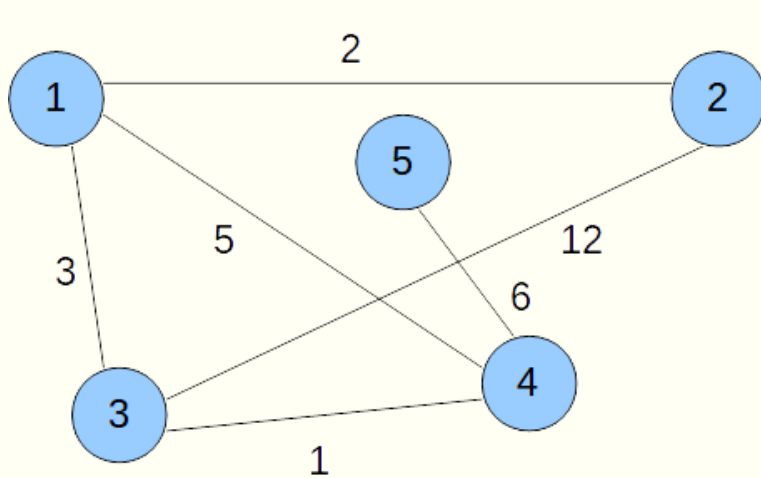
- Arbres couvrants de poids minimal
- Algorithme de Kruskal
 - Structure de données pour ensembles disjoints
- Algorithme de Prim
 - Structure de file de priorités
- Bibliographie
 - T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Algorithmique", 3^e édition, Dunod, 2010



PROBLÈME DE L'ARBRE COUVRANT DE POIDS MINIMAL

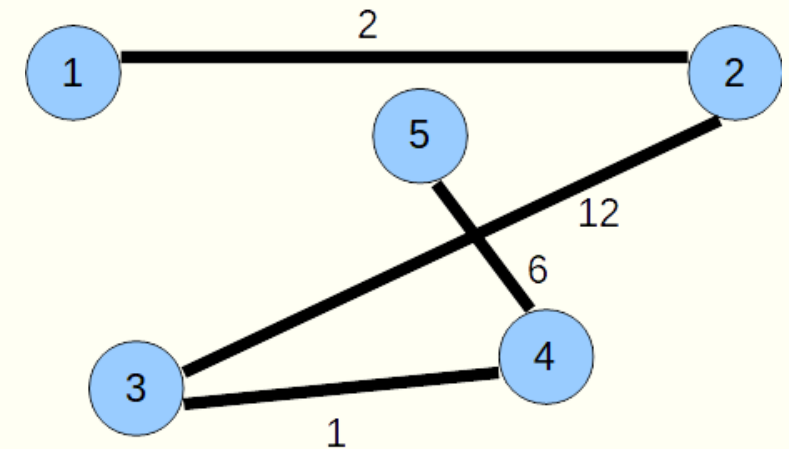
Définition du problème

- Étant donné un graphe non orienté pondéré
- On cherche un graphe partiel de ce graphe
 - ... qui soit un arbre ...
 - ... et qui soit de coût minimum
- Graphe partiel : l'arbre a pour ensemble de sommets tous les sommets du graphe initial
 - On dit que c'est un arbre couvrant
- En anglais : minimum spanning tree
- On suppose le graphe initial connexe



Arbre couvrant
(de poids 22)

Info0501 - Cours 4



Arbre couvrant
(de poids 21)

Applications

- Réseaux
 - On estime le coût des liaisons directes entre toutes les machines à relier
 - Puis on cherche à réaliser un réseau connexe à coût minimum
- Traitement d'images
 - On représente une image sous forme de pixels
 - On veut déterminer des régions dans l'image
 - Graphe : chaque pixel est un sommet à 8 voisins
 - On value les arêtes par la différence de gris
 - On construit l'arbre couvrant minimum, puis on sépare

Construction d'un arbre couvrant minimal

- Soit un graphe $G = (S, A)$
 - Non orienté
 - Connexe
 - Fonction de pondération $p \rightarrow$ attribue un poids $p(u, v)$ à chaque arête (u, v)
- Stratégies gloutonnes
 - On construit l'arbre arête par arête
 - ... en ajoutant à chaque étape une arête appropriée de A

ACM-GÉNÉRIQUE (G, p)

$E = \emptyset$

Tant que E ne forme pas un arbre couvrant

Trouver une arête (u, v) appropriée

$E = E \cup (u, v)$

Retourner E

- 2 algorithmes principaux \rightarrow Kruskal et Prim
 - Utilisent cette stratégie gloutonne
 - Utilisent une règle différente pour choisir l'arête (u, v) appropriée



ALGORITHME DE KRUSKAL

Principe de l'algorithme

- Pour déterminer un arbre couvrant de poids minimal d'un graphe connexe à $|S|$ sommets, ...
- ... on sélectionne les arêtes d'un graphe partiel initialement sans arête ...
- ... en itérant $|S| - 1$ fois l'opération suivante
 - Choisir une arête de poids minimal ne formant pas un cycle avec les arêtes précédemment choisies
- Exemple 1 : fonctionnement de l'algorithme de Kruskal

Mise en oeuvre

- Trier les arêtes par ordre de poids croissants
- Tant qu'on n'a pas retenu $|S| - 1$ arêtes
 - Considérer, dans l'ordre du tri, la 1ère arête non examinée
 - Si elle forme un cycle avec celles précédentes, la rejeter, sinon la garder
- Il reste à résoudre un problème
 - Comment déterminer si une arête choisie à l'étape i forme un cycle avec les arêtes précédemment choisies ?

Principe : gérer l'évolution des composantes connexes

- Pour qu'une arête (u, v) ferme un cycle, il faut que ses extrémités aient été précédemment reliées par une chaîne
 - Donc aient été dans une même composante connexe
- Nous allons donc gérer l'évolution des composantes connexes au fur et à mesure du choix des arêtes
 - Par tableau \rightarrow plus simple mais moins efficace
 - Par ensembles disjoints \rightarrow efficace

Gérer l'évolution des composantes connexes par tableau

- Initialement, le graphe ne contient aucune arête
 - Chaque sommet est une composante connexe
 - On initialise chaque sommet à son indice i dans un tableau de longueur $|S|$
- Chaque fois qu'une arête $\{u, v\}$ est candidate
 - On compare les valeurs aux indices u et v dans le tableau
 - Si valeurs égales, u et v sont dans la même composante connexe \rightarrow ajouter l'arête créerait alors un cycle donc on ne la retient pas
 - Si valeurs différentes, on garde l'arête $\{u, v\}$, on donne à l'indice de v la valeur de l'indice de u , ainsi que tout sommet d'indice v
 - Autrement dit, après sélection de l'arête $\{u, v\}$, tous les sommets de la composante connexe de u et de la composante connexe de v ne forment qu'une seule composante connexe et ont le même indice
- Exemple 2 : Gestion des composantes connexes par tableau

Remarques

- Après la sélection d'une arête valide
 - On doit parcourir tout le tableau pour mettre à jour les indices de composantes connexes
 - $\alpha(S)$
- On peut réduire ce temps par l'utilisation d'une structure de données pour ensembles disjoints



STRUCTURES DE DONNÉES POUR ENSEMBLES DISJOINTS

Ensembles disjoints

- On veut regrouper n éléments distincts dans une collection $S = \{S_1, S_2, \dots, S_k\}$ d'ensembles dynamiques disjoints
 - Chaque élément fait partie d'un seul ensemble
 - Chaque ensemble est identifié par un représentant
- Opérations sur les ensembles disjoints
 - CRÉER-ENSEMBLE (x)
 - TROUVER-ENSEMBLE (x)
 - UNION (x, y)

Opérations sur les ensembles disjoints

▪ CRÉER-ENSEMBLE (x)

- À partir d'un pointeur sur un objet x
- Crée un nouvel ensemble dont le seul membre (et le représentant) est x
- Ensembles disjoints
 - x ne doit pas être déjà membre d'un autre ensemble

▪ TROUVER-ENSEMBLE (x)

- Retourne un pointeur vers le représentant de l'ensemble contenant x

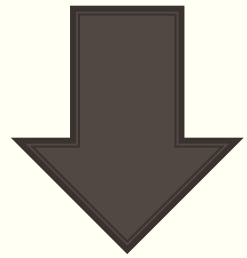
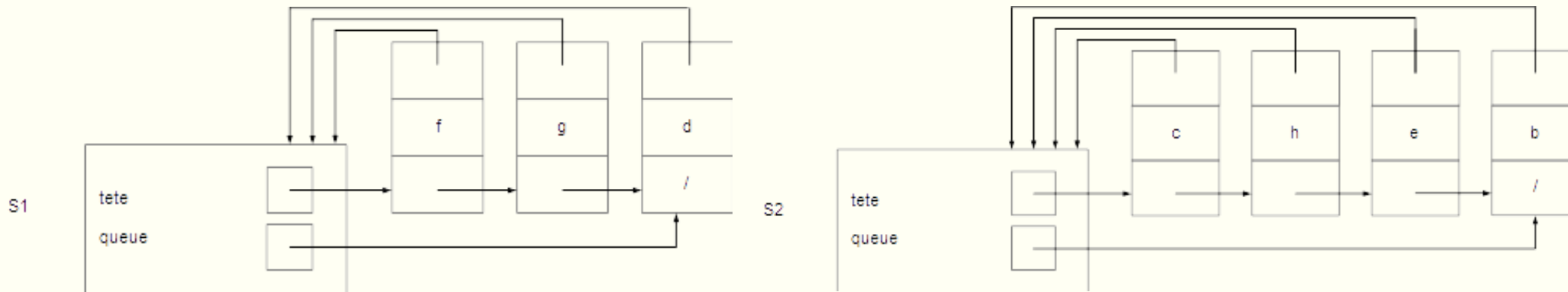
▪ UNION (x, y)

- Réunit les ensembles dynamiques qui contiennent x et y ...
- ... appelés S_x et S_y
- ... dans un nouvel ensemble qui est l'union de S_x et S_y
- Le représentant du nouvel ensemble sera un élément quelconque d'un des 2 ensembles
- Détruit les ensembles initiaux S_x et S_y

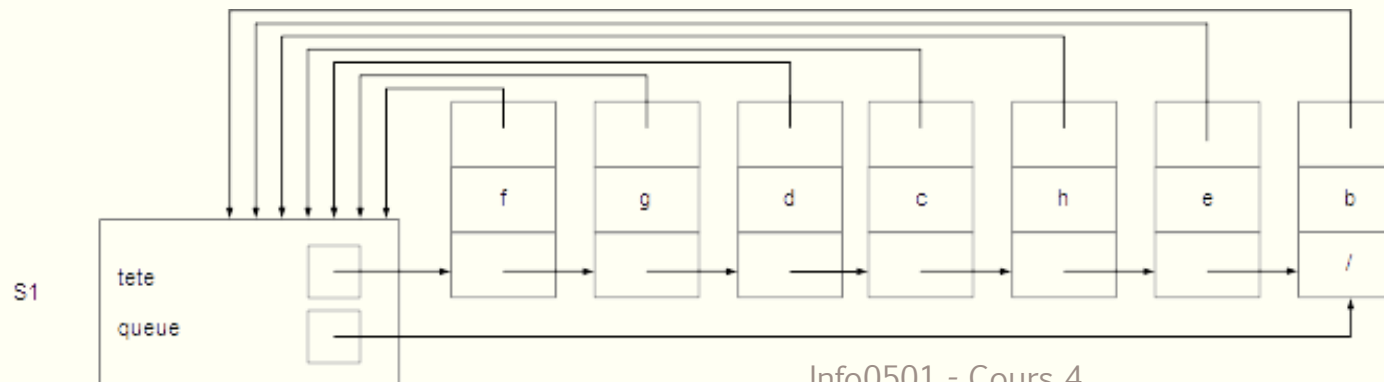
▪ Exemple 3

- Application d'une structure de données d'ensembles disjoints
- Détermination des composantes connexes d'un graphe non orienté

Représentation d'ensembles disjoints par listes chaînées

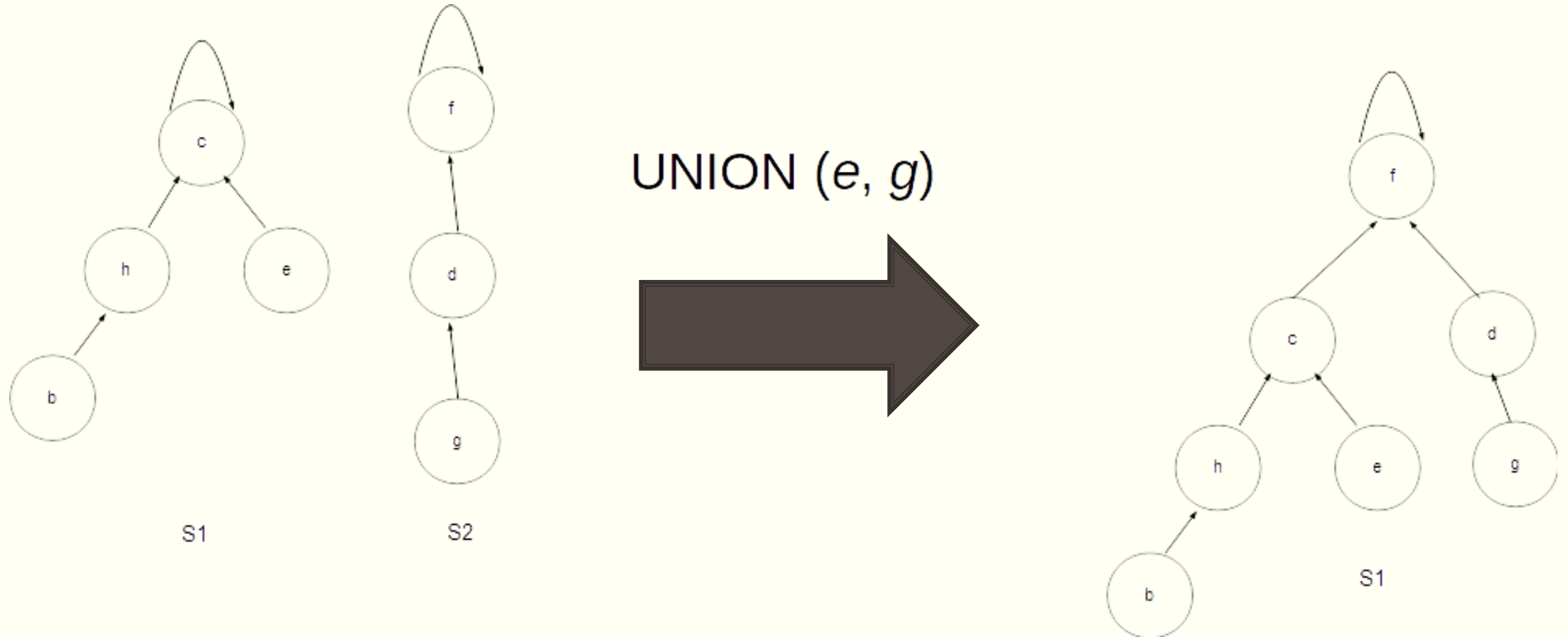


UNION (g, e)



- Temps d'exécution
 - CRÉER-ENSEMBLE(x) ?
 - $\mathcal{O}(1)$
 - TROUVER-ENSEMBLE(x) ?
 - $\mathcal{O}(1)$
 - UNION(x, y) ?
 - $\mathcal{O}(n)$

Représentation d'ensembles disjoints par arbres



Algorithme de Kruskal avec utilisation d'ensembles disjoints

- On crée un ensemble disjoint pour chaque sommet de G
- On trie les arêtes de G
- Pour chaque arête prise par ordre croissant
 - Si l'origine et l'extrémité ne font pas partie du même ensemble, conserver l'arête et réunir les deux ensembles
- Les arêtes retenues constituent alors un arbre couvrant de poids minimal de G



ALGORITHME DE PRIM

Principe de l'algorithme

- Permet de trouver un arbre couvrant de poids minimum sans devoir trier les arêtes
- On étend, de proche en proche, un arbre couvrant des parties des sommets du graphe
 - En atteignant un sommet supplémentaire à chaque étape ...
 - ... en prenant l'arête la plus légère parmi celles qui joignent l'ensemble des sommets déjà couverts ...
 - ... à l'ensemble des sommets non encore couverts
- Un tableau de distances d permet de connaître la distance entre chaque sommet non couvert et l'ensemble des sommets couverts
- Exemple 4 : Fonctionnement de l'algorithme de Prim – Implémentation par tableau

Remarques

- Après la mise à jour des distances entre le pivot actuel et ses voisins
 - On doit parcourir tout le tableau d pour déterminer le pivot de l'itération suivante
 - $\alpha(S)$
- On peut réduire ce temps par l'utilisation d'une file de priorités min implémentée par tas



FLASHBACK

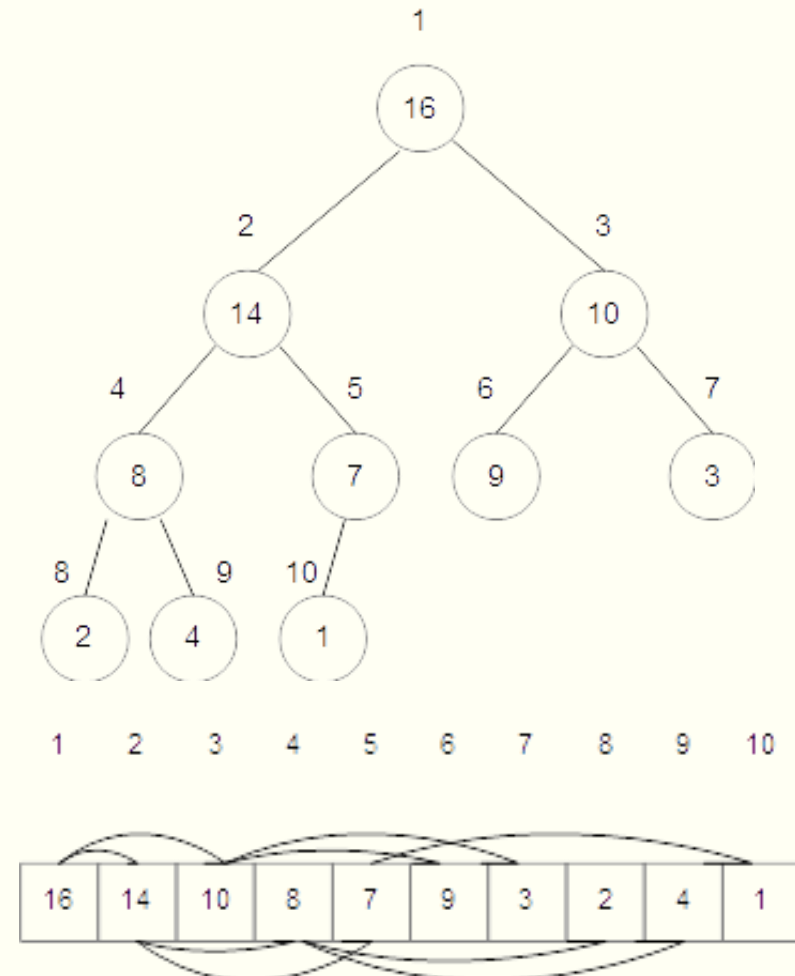
Tas

Tas (binaire)

- Tableau ayant des caractéristiques spécifiques sur l'emplacement des éléments
 - Peut être vu (mais non stocké) comme un arbre binaire presque complet
- Plusieurs utilisations
 - Tri
 - Files de priorités
 - ...

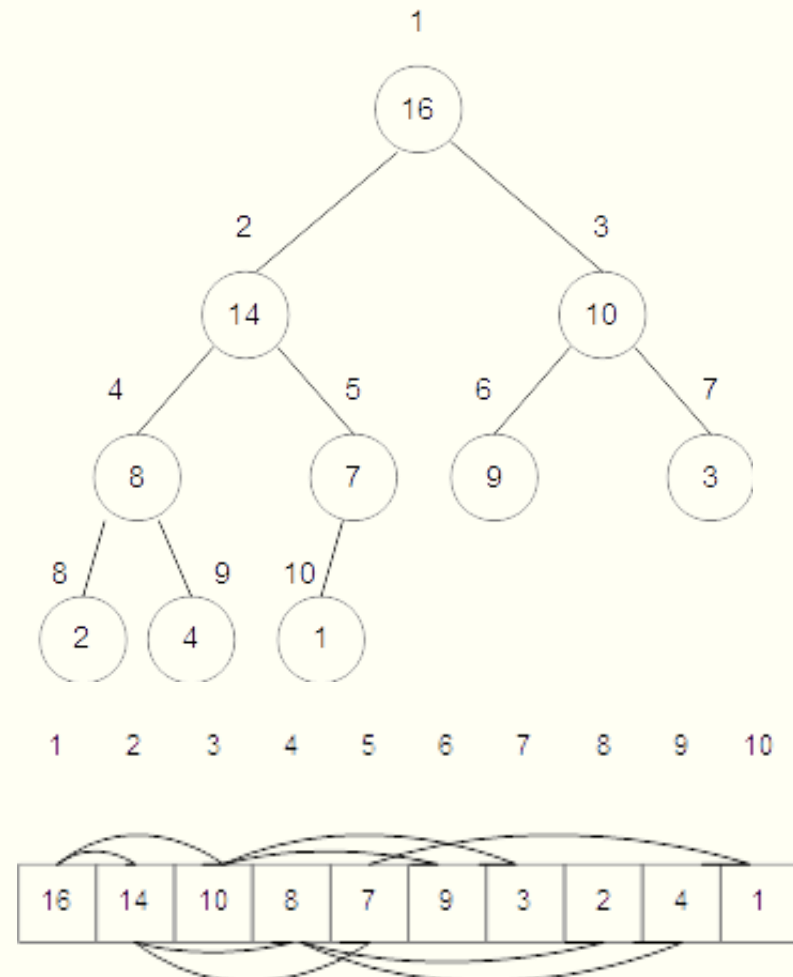
Tas (binaire)

- Chaque nœud de l'arbre correspond à un élément du tableau
- L'arbre est complètement rempli à tous les niveaux
 - Sauf éventuellement le dernier qui est rempli de gauche à droite
- Un tableau t représentant un tas possède 2 attributs
 - **longueur** : nombre maximum d'éléments
 - **taille** : nombre d'éléments du tas effectivement rangés dans le tableau
- Éléments valides du tas $\rightarrow t[1 .. t.taille]$



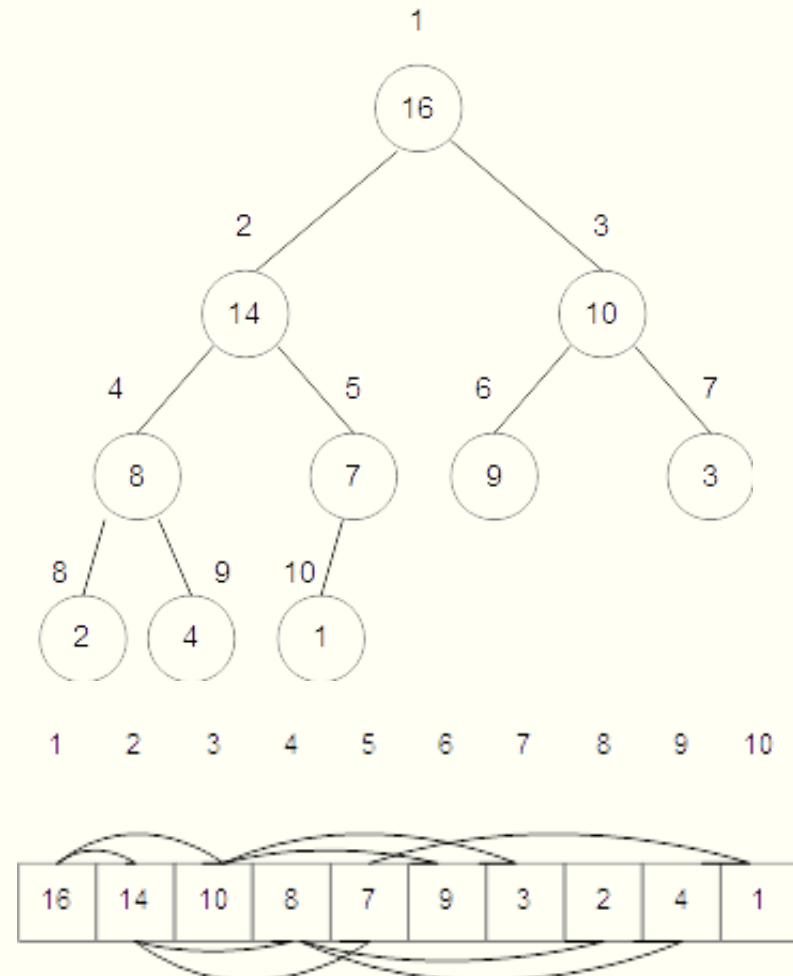
Tas (binaire)

- Racine de l'arbre
 - $t[1]$
- Étant donné l'indice i d'un nœud, on peut calculer
 - L'indice de son parent $\text{PARENT}(i)$
 - $\lfloor i / 2 \rfloor$
 - L'indice de son enfant de gauche $\text{GAUCHE}(i)$
 - $2i$
 - L'indice de son enfant de droite $\text{DROITE}(i)$
 - $2i + 1$



Tas (binaire)

- Propriété de tas (ici pour un tas max)
 - Pour chaque nœud i autre que la racine
 - La valeur d'un nœud est au plus égale à celle du parent
 - $t[\text{PARENT}(i)] \geq t[i]$
- Plus grand élément stocké à la racine
- Hauteur d'un nœud
 - Nombre d'arcs sur le chemin le plus long reliant le nœud à une feuille
- Hauteur d'un tas de n éléments
 - Hauteur de la racine
 - $\Theta(\lg n)$



Conservation de la propriété de tas

- Une modification du tableau doit assurer la conservation de la propriété de tas
- Procédure ENTASSER-MAX(t, i)
 - Suppose que les arbres binaires enracinés en GAUCHE(i) et DROITE(i) sont des tas max
 - ... mais que $t[i]$ puisse être plus petit que ses enfants
 - ... violant ainsi la propriété de tas max
 - On fait alors descendre la valeur de $t[i]$ dans le tas max de sorte à rétablir la propriété de tas
- Fonctionnement de ENTASSER-MAX
 - À chaque étape, on détermine le plus grand des éléments $t[i]$, GAUCHE(i) et DROITE(i)
 - Si $t[i]$ est le max, on a déjà un tas max
 - Sinon, on échange $t[i]$ avec l'enfant qui est le max et on rappelle ENTASSER-MAX récursivement
- Exemple 5
 - ENTASSER-MAX avec $t = \langle 16, 4, 10, 14, 7, 9, 3, 2, 8, 1 \rangle$
- Temps d'exécution proportionnel à la hauteur de l'arbre
 - $O(\lg n)$

Construction d'un tas

- Pour convertir un tableau $t[1..n]$ (avec $n = t.\text{longueur}$) en tas max
 - On utilise la propriété suivante
 - Les éléments du sous-tableau $t[(n / 2 + 1) .. n]$ sont tous des feuilles de l'arbre, donc des tas max à 1 élément
 - On utilise la procédure ENTASSER-MAX sur les autres nœuds de l'arbre, à l'envers
- Exemple 6
 - CONSTRUIRE-TAS-MAX avec $t = \langle 4, 1, 3, 2, 16, 9, 10, 14, 8, 7 \rangle$
- Temps d'exécution : $O(n)$ appels à ENTASSER-MAX
 - $O(n \lg n)$ (une analyse plus fine peut démontrer que c'est en fait $\mathcal{O}(n)$)

Tri par tas

- Démarre en construisant un tas max avec le tableau d'entrée
- Principe
 - Élément maximal du tableau \rightarrow racine de l'arbre $t[1]$
 - On peut donc le placer à sa position finale correcte, à la fin du tableau, en l'échangeant avec $t[n]$
 - ... et en enlevant le nœud n du tas en décrémentant $t.taille$
 - On rétablit ensuite la propriété de tas max en appelant ENTASSER-MAX sur la racine
 - Et on répète le processus jusqu'à arriver à un tas de taille 2
- Exemple 7
 - TRI-PAR-TAS avec $t = \langle 16, 14, 10, 8, 7, 9, 3, 2, 4, 1 \rangle$
- CONSTRUIRE-TAS-MAX
 - $O(n \lg n)$ (ou $O(n)$, mais ça ne change pas le résultat)
- n appels à ENTASSER-MAX
 - $O(n \lg n)$
- Temps d'exécution
 - $\Theta(n \lg n)$
 - (Comme le tri par fusion)
- Trie sur place
 - (Comme le tri par insertion)

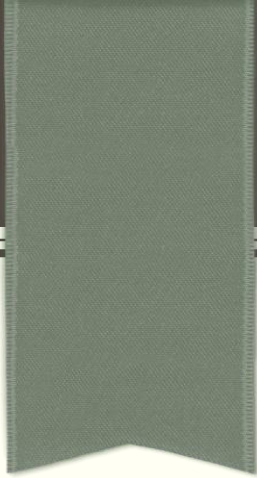


FIN DU FLASHBACK

Tas

Algorithme de Prim – Implémentation par file de priorités

- Utilise un tas-min à la place du tableau d pour stocker les distances des sommets non encore couverts
 - Cette version de l'algorithme sera vue en TD



PROCHAIN COURS
PLUS COURTS CHEMINS
