

Fin des tables de hachages et arbres

Algorithmes et structures de données, 2025-2026

P. Albuquerque (B410) et O. Malaspinas (A401), ISC, HEPIA
2026-02-25

En partie inspiré des supports de cours de P. Albuquerque

Les tables de hachage

Rappel (1/2)

Qu'est-ce qu'une table de hachage ?

Structure de données abstraite où chaque *valeur* (ou élément) est associée à une *clé* (ou argument).

On parle de paires *clé-valeur* (*key-value pairs*).

Quelles opérations principales sur les tables ?

- Insertion d'élément (`insert(clé, valeur)`), insère la paire clé-valeur
- Consultation (`get(clé)`), retourne la valeur correspondant à clé
- Suppression (`remove(clé)`), supprime la paire clé-valeur

Rappel (2/2)

Objectif

Trouver une méthode efficace pour faire

`get(table, clé) -> retourne une valeur`

sans avoir à parcourir le tableau (séquentiel ou binary get)

Transformation de clé (hashing)

Problématique : Numéro AVS (13 chiffres)

- Format : 106.3123.8492.13

Numéro AVS	Nom
0000000000000	-----
...	...
1063123849213	Paul
...	...
3066713878328	Orestis
...	...
9999999999999	-----

Quelle est la clé ? Quelle est la valeur ?

Transformation de clé (hashing)

Problématique : Numéro AVS (13 chiffres)

- Format : 106.3123.8492.13

Numéro AVS	Nom
0000000000000	-----
...	...
1063123849213	Paul
...	...
3066713878328	Orestis
...	...
9999999999999	-----

Quelle est la clé ? Quelle est la valeur ?

- Clé : Numéro AVS, Valeur : Nom.

Nombre de clés ? Nombre de citoyens ? Rapport ?

Transformation de clé (hashing)

Problématique : Numéro AVS (13 chiffres)

- Format : 106.3123.8492.13

Numéro AVS	Nom
0000000000000	-----
...	...
1063123849213	Paul
...	...
3066713878328	Orestis
...	...
9999999999999	-----

Quelle est la clé ? Quelle est la valeur ?

- Clé : Numéro AVS, Valeur : Nom.

Nombre de clés ? Nombre de citoyens ? Rapport ?

- 10^{13} clés, 10^7 citoyens, 10^{-6} ($10^{-4}\%$ de la table est occupée) \Rightarrow *inefficace*.
- Pire : 10^{13} entrées ne rentrent pas dans la mémoire d'un ordinateur.

Transformation de clé (hashing)

Problématique 2 : Identificateurs d'un programme

- Format : 8 caractères (simplification)

Identificateur		Adresse
aaaaaaaa		-----
...		...
resultat		3aeff
compteur		4fedc
...		...
zzzzzzzz		-----

Quelle est la clé ? Quelle est la valeur ?

Transformation de clé (hashing)

Problématique 2 : Identificateurs d'un programme

- Format : 8 caractères (simplification)

Identificateur		Adresse
aaaaaaaa		-----
...		...
resultat		3aeff
compteur		4fedc
...		...
zzzzzzzz		-----

Quelle est la clé ? Quelle est la valeur ?

- Clé : Identificateur, Valeur : Adresse.

Nombre de clés ? Nombre d'identificateurs d'un programme ?

Rapport ?

Transformation de clé (hashing)

Problématique 2 : Identificateurs d'un programme

- Format : 8 caractères (simplification)

Identificateur		Adresse
aaaaaaaa		-----
...		...
resultat		3aeff
compteur		4fedc
...		...
zzzzzzzz		-----

Quelle est la clé ? Quelle est la valeur ?

- Clé : Identificateur, Valeur : Adresse.

Nombre de clés ? Nombre d'identificateurs d'un programme ?

Rapport ?

- $26^8 \sim 2 \cdot 10^{11}$ clés, 2000 identificateurs, 10^{-8} ($10^{-6}\%$ de la table est occupée) \Rightarrow *un peu inefficace*.

Fonctions de transformation de clé (hash functions)

- La table est représentée avec un tableau.
- La taille du tableau est bien plus petite que le nombre de clés possibles.
- On produit un indice du tableau à partir d'une clé :

$$h(key) = n, \quad n \in \mathbb{N}.$$

En français : on transforme *key* en nombre entier qui sera l'indice dans le tableau correspondant à *key*.

La fonction de hash

- La taille du domaine des clés est beaucoup plus grande que le domaine des indices.
- Plusieurs **clés** peuvent correspondre au **même indice** :
 - Il faut traiter les **collisions**.
- L'ensemble des indices doit être plus petit ou égal à la taille de la table.

Une bonne fonction de hash

Fonctions de transformation de clés : exemples

Méthode par troncature

$$h : [0, 9999] \rightarrow [0, 9]$$

$h(key)$ = troisième chiffre du nombre.

Key		Index
0003		0
1123		2 \
1234		3 -> collision.
1224		2 /
1264		6

Quelle est la taille de la table ?

Fonctions de transformation de clés : exemples

Méthode par troncature

$$h : [0, 9999] \rightarrow [0, 9]$$

$h(key)$ = troisième chiffre du nombre.

Key		Index
0003		0
1123		2 \
1234		3 -> collision.
1224		2 /
1264		6

Quelle est la taille de la table ?

C'est bien dix oui.

Fonctions de transformation de clés : exemples

Méthode par découpage

Taille de l'index : 3 chiffres.

```
key = 321 991 24 -> 321
                        991
                        + 24
                        ----
                        1336 -> index = 336
```

Devinez l'algorithme ?

Fonctions de transformation de clés : exemples

Méthode par découpage

Taille de l'index : 3 chiffres.

```
key = 321 991 24 -> 321
                        991
                        + 24
                        ----
                        1336 -> index = 336
```

Devinez l'algorithme ?

On part de la gauche :

1. On découpe la clé en tranche de longueur égale à celle de l'index.
2. On somme les nombres obtenus.
3. On tronque à la longueur de l'index.

Fonctions de transformation de clés : exemples

Méthode multiplicative

Taille de l'index : 2 chiffres.

$\text{key} = 5486 \rightarrow \text{key}^2 = 30096196 \rightarrow \text{index} = 96$

On prend le carré de la clé et on garde les chiffres du milieu du résultat.

Fonctions de transformation de clés : exemples

Méthode par division modulo

Taille de l'index : N chiffres.

$$h(\text{key}) = \text{key} \% N.$$

Quelle doit être la taille de la table ?

Fonctions de transformation de clés : exemples

Méthode par division modulo

Taille de l'index : N chiffres.

$$h(\text{key}) = \text{key} \% N.$$

Quelle doit être la taille de la table ?

Oui comme vous le pensiez au moins N.

Traitement des collisions

La collision

`key1 != key2, h(key1) == h(key2)`

Traitement (une idée ?)

Traitement des collisions

La collision

`key1 != key2, h(key1) == h(key2)`

Traitement (une idée ?)

- La première clé occupe la place prévue dans le tableau.
- La deuxième (troisième, etc.) est placée ailleurs de façon **déterministe**.

Dans ce qui suit la taille de la table est `table_size`.

La méthode séquentielle

Comment ça marche ?

- Quand l'index est déjà occupé on regarde sur la position suivante, jusqu'à en trouver une libre.

```
index = h(key);  
while (table[index].state == occupied && table[index].key != key) {  
    index = (index + 1) % table_size; // attention à pas dépasser  
}  
table[index].key = key;  
table[index].state = occupied;
```

Problème ?

La méthode séquentielle

Comment ça marche ?

- Quand l'index est déjà occupé on regarde sur la position suivante, jusqu'à en trouver une libre.

```
index = h(key);  
while (table[index].state == occupied && table[index].key != key) {  
    index = (index + 1) % table_size; // attention à pas dépasser  
}  
table[index].key = key;  
table[index].state = occupied;
```

Problème ?

- Regroupement d'éléments (clustering).

Comment ça marche ?

- Comme la méthode séquentielle mais on “saute” de k .

```
index = h(key);  
while (table[index].state == occupied && table[index].key != key) {  
    index = (index + k) % table_size; // attention à pas dépasser  
}  
table[index].key = key;  
table[index].state = occupied;
```

Quelle valeur de k éviter ?

Méthode linéaire

Comment ça marche ?

- Comme la méthode séquentielle mais on “saute” de k .

```
index = h(key);  
while (table[index].state == occupied && table[index].key != key) {  
    index = (index + k) % table_size; // attention à pas dépasser  
}  
table[index].key = key;  
table[index].state = occupied;
```

Quelle valeur de k éviter ?

- Une valeur où `table_size` est multiple de k .

Cette méthode répartit mieux les regroupements au travers de la table.

Méthode du double hashing

Comment ça marche ?

- Comme la méthode linéaire, mais $k = h_2(\text{key})$ (variable).

```
index = h(key);  
while (table[index].state == occupied && table[index].key != key) {  
    index = (index + h2(key)) % table_size; // attention à pas dépasser  
}  
table[index].key = key;  
table[index].state = occupied;
```

Quelle propriété doit avoir h_2 ?

Exemple

```
h2(key) = (table_size - 2) - key % (table_size - 2)
```

Méthode pseudo-aléatoire

Comment ça marche ?

- Comme la méthode linéaire mais on génère k pseudo-aléatoirement.

```
index = h(key);  
while (table[index].state == occupied && table[index].key != key) {  
    index = (index + random_number) % table_size;  
}  
table[index].key = key;  
table[index].state = occupied;
```

Comment s'assurer qu'on va bien retrouver la bonne clé ?

Méthode pseudo-aléatoire

Comment ça marche ?

- Comme la méthode linéaire mais on génère k pseudo-aléatoirement.

```
index = h(key);  
while (table[index].state == occupied && table[index].key != key) {  
    index = (index + random_number) % table_size;  
}  
table[index].key = key;  
table[index].state = occupied;
```

Comment s'assurer qu'on va bien retrouver la bonne clé ?

- Le germe (seed) de la séquence pseudo-aléatoire doit être le même.
- Le germe à choisir est l'index retourné par $h(key)$.

```
srand(h(key));  
while {  
    random_number = rand();  
}
```

Méthode quadratique

- La fonction des indices de collision est de degré 2.
- Soit $J_0 = h(key)$, les indices de collision se construisent comme :

```
J_i = J_0 + i^2 % table_size, i > 0,  
J_0 = 100, J_1 = 101, J_2 = 104, J_3 = 109, ...
```

Problème possible ?

Méthode quadratique

- La fonction des indices de collision est de degré 2.
- Soit $J_0 = h(key)$, les indices de collision se construisent comme :

```
J_i = J_0 + i^2 % table_size, i > 0,  
J_0 = 100, J_1 = 101, J_2 = 104, J_3 = 109, ...
```

Problème possible ?

- Calculer le carré peut être “lent”.
- En fait on peut ruser un peu.

Méthode quadratique

```
J_i = J_0 + i^2 % table_size, i > 0,  
J_0 = 100  
    \  
    d_0 = 1  
  /    \  
J_1 = 101    Delta = 2  
  \  
  d_1 = 3  
 /    \  
J_2 = 104    Delta = 2  
  \  
  d_2 = 5  
 /    \  
J_3 = 109    Delta = 2  
  \  
  d_3 = 7  
 /  
J_4 = 116  
  
-----  
J_{i+1} = J_i + d_i,  
d_{i+1} = d_i + Delta, d_0 = 1, i > 0.
```

Méthode de chaînage

Comment ça marche ?

- Chaque index de la table contient un pointeur vers une liste chaînée contenant les paires clés-valeurs.

Un petit dessin

Méthode de chaînage

Exemple

On hash avec la fonction $h(\text{key}) = \text{key} \% 11$ (key est le numéro de la lettre de l'alphabet)

U		N		E		X		E		M		P		L		E		D		E		T		A		B		L		E
10		3		5		2		5		2		5		1		5		4		5		9		1		2		1		5

Comment on représente ça ? (à vous)

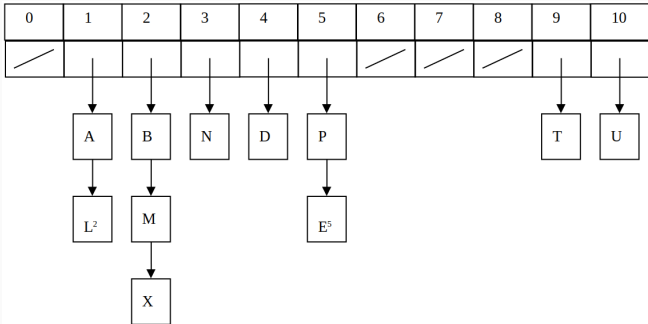
Méthode de chaînage

Example

On hash avec la fonction $h(\text{key}) = \text{key} \% 11$ (key est le numéro de la lettre de l'alphabet)

U	N	E	X	E	M	P	L	E	D	E	T	A	B	L	E
10	3	5	2	5	2	5	1	5	4	5	9	1	2	1	5

Comment on représente ça? (à vous)



Méthode de chaînage

Avantages :

- Si les clés sont grandes l'économie de place est importante (les places vides sont NULL).
- La gestion des collisions est conceptuellement simple.
- Pas de problème de regroupement (clustering).

Exercice 1

- Construire une table à partir de la liste de clés suivante : R, E, C, O, U, P, A, N, T
- On suppose que la table est initialement vide, de taille $n = 13$.
- Utiliser la fonction $h_1(k) = k \bmod 13$ où k est la k -ème lettre de l'alphabet et un traitement séquentiel des collisions.

Exercice 2

- Reprendre l'exercice 1 et utiliser la technique de double hachage pour traiter les collisions avec

$$\begin{aligned}h_1(k) &= k \bmod 13, \\h_2(k) &= 1 + (k \bmod 11).\end{aligned}$$

- La fonction de hachage est donc $h(k) = (h_1(k) + h_2(k)) \% 13$ en cas de collision

Exercice 3

- Stocker les numéros de téléphones internes d'une entreprise suivants dans un tableau de 10 positions.
- Les numéros sont compris entre 100 et 299.
- Soit N le numéro de téléphone, la fonction de hachage est

$$h(N) = N \bmod 10.$$

- La fonction de gestion des collisions est

$$C_i(N) = (h(N) + 3 \cdot i) \bmod 10,$$

avec i le numéro de la collision.

- Placer 145, 167, 110, 175, 210, 215 (mettre son état à occupé).
- Supprimer 175 (rechercher 175, et mettre son état à supprimé).
- Rechercher 135, 215.
- Les cases ni supprimées, ni occupées sont vides.

Implémentation d'une table de hachage

Préambule

- Ici, on ne considère pas le cas du chaînage en cas de collisions.
- L'insertion est construite avec une forme du type

```
index = h(key);  
while (table[index].state == occupied  
      && table[index].key != key) {  
    index = (index + k) % table_size; // attention à pas dépasser  
}  
table[index].key = key;  
table[index].state = occupied;
```

- Gestion de l'état d'une case *explicite*

```
enum state_t {empty, occupied, deleted};
```

Pseudocode ?

L'insertion

Pseudocode ?

```
rien insertion(table, clé, valeur) {  
    index = hash(clé)  
    tant que table[index].état == occupé  
        et table[index].clé != clé:  
        index = rehash(index, clé)  
  
    table[index].état = occupé  
    table[index].clé = clé  
    table[index].valeur = valeur  
}
```

Pseudocode ?

La suppression

Pseudocode ?

```
valeur suppression(table, clé):  
    index = hash(clé)  
    tant que table[index].état != vide:  
        si table[index].état == occupé  
            et table[index].clé == clé:  
                table[index].état = supprimé  
        sinon  
            index = rehash(index, clé)  
    }
```

Pseudocode ?

Pseudocode ?

```
booléen recherche(table, clé) {  
    index = hash(clé)  
    tant que table[index].état != vide:  
        si table[index].état == occupé  
            et table[index].clé == clé:  
                retourner vrai  
        sinon  
            index = rehash(index, clé)  
    retourner faux  
}
```

Écrivons le code !

- Mais avant :
 - Quelles sont les structures de données dont nous avons besoin ?
 - Y a-t-il des fonctions auxiliaires à écrire ?
 - Écrire les signatures des fonctions.

Écrivons le code !

- Mais avant :
 - Quelles sont les structures de données dont nous avons besoin ?
 - Y a-t-il des fonctions auxiliaires à écrire ?
 - Écrire les signatures des fonctions.

Structures de données

Écrivons le code !

- Mais avant :
 - Quelles sont les structures de données dont nous avons besoin ?
 - Y a-t-il des fonctions auxiliaires à écrire ?
 - Écrire les signatures des fonctions.

Structures de données

```
enum state_t {empty, deleted, occupied};
typedef char * key_t; // could be something different
typedef char * value_t;
struct cell_t {
    key_t key;
    value_t value;
    enum state_t state;
};
struct hm {
    struct cell_t *table;
    int capacity;
    int size;
};
```


Écrivons le code !

Fonctions auxiliaires

Écrivons le code !

Fonctions auxiliaires

```
static int hash(key_t key);  
static int rehash(int index, key_t key);  
static int find_index(hm h, key_t key);
```

Signature de l'API

Écrivons le code !

Fonctions auxiliaires

```
static int hash(key_t key);  
static int rehash(int index, key_t key);  
static int find_index(hm h, key_t key);
```

Signature de l'API

```
void hm_init(struct hm *h, int capacity);  
void hm_destroy(struct hm *h);  
bool hm_set(struct hm *h, key_t key, value_t *value);  
bool hm_get(struct hm h, key_t key, value_t *value);  
bool hm_remove(struct hm *h, key_t key, value_t *value);  
bool hm_search(struct hm h, key_t key);  
void hm_print(struct hm h);
```

Live code session !

0. Offered to you by ProtonVPN¹ !

1. The fastest way to connect to BBB !

Live code session !

0. Offered to you by ProtonVPN¹ !
1. Like the video.
2. Subscribe to the channel.
3. Use our one time voucher for ProtonVPN : PAULISAWESOME.
4. Consider donating on our patreon.

1. The fastest way to connect to BBB !

Les arbres

Les arbres : définition

“Un arbre est un graphe acyclique orienté possédant une unique racine, et tel que tous les nœuds sauf la racine ont un unique parent.”

Les arbres : définition

“Un arbre est un graphe acyclique orienté possédant une unique racine, et tel que tous les nœuds sauf la racine ont un unique parent.”

Santé !

Plus sérieusement

- Ensemble de **nœuds** et d'**arêtes** (graphe).
- Les arêtes relient les nœuds entre eux, mais pas n'importe comment : chaque nœud a au plus un **parent**.
- Le seul nœud sans parent est la **racine**.
- Chaque nœud a un nombre fini d'**enfants**.
- La hiérarchie des nœuds rend les arêtes **orientées** (parent -> enfants), et empêche les **cycles** (acyclique, orienté).
- La **feuille** ou **nœud terminal** est un nœud sans enfants.
- Le **niveau** est 1 à la racine et **niveau+1** pour les enfants.
- Le **degré** d'un nœud est le nombre d'enfants du nœud.

Les arbres : définition

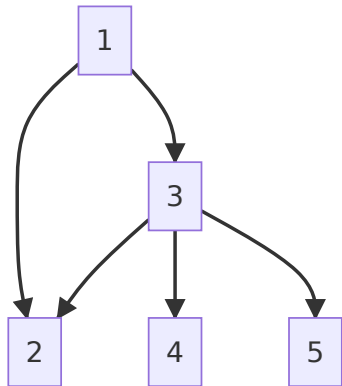
“Un arbre est un graphe acyclique orienté possédant une unique racine, et tel que tous les nœuds sauf la racine ont un unique parent.”

Santé !

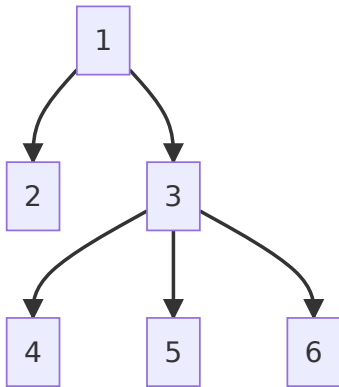
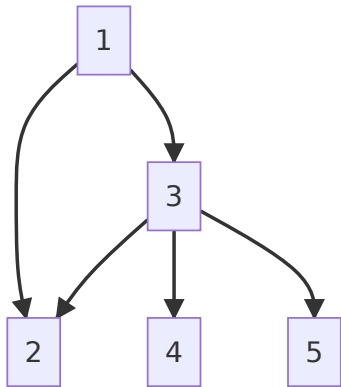
Plus sérieusement

- Ensemble de **nœuds** et d'**arêtes** (graphe).
- Les arêtes relient les nœuds entre eux, mais pas n'importe comment : chaque nœud a au plus un **parent**.
- Le seul nœud sans parent est la **racine**.
- Chaque nœud a un nombre fini d'**enfants**.
- La hiérarchie des nœuds rend les arêtes **orientées** (parent -> enfants), et empêche les **cycles** (acyclique, orienté).
- La **feuille** ou **nœud terminal** est un nœud sans enfants.
- Le **niveau** est 1 à la racine et **niveau+1** pour les enfants.
- Le **degré** d'un nœud est le nombre d'enfants du nœud.
- Chaque nœud est un arbre en lui-même.
- La **récurtivité** sera très utile !

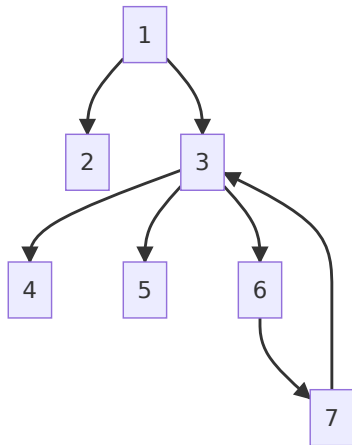
Arbre ou pas arbre ?



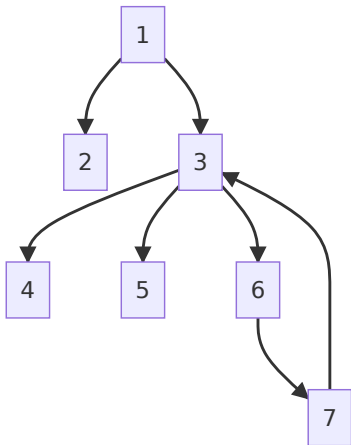
Arbre ou pas arbre ?



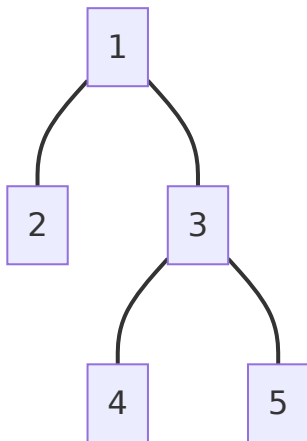
Arbre ou pas arbre ?



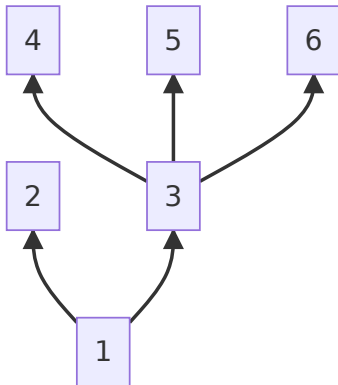
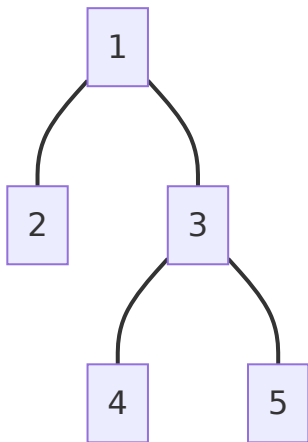
Arbre ou pas arbre ?



Arbre ou pas arbre ?

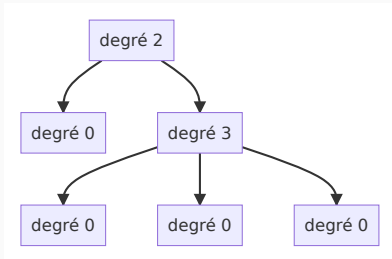


Arbre ou pas arbre ?



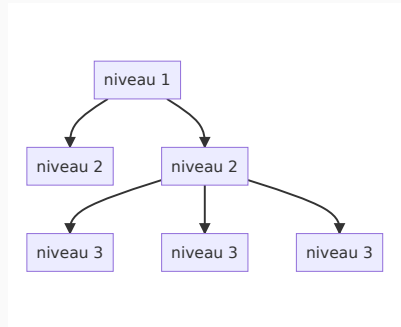
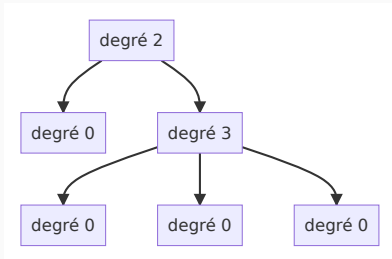
Degré et niveau

- Illustration du degré (nombre d'enfants) et du niveau (profondeur)



Degré et niveau

- Illustration du degré (nombre d'enfants) et du niveau (profondeur)



- Les nœuds de degré 0 sont des feuilles.

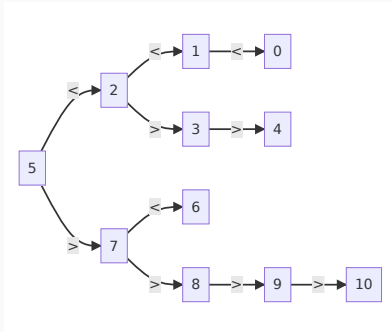
Application : recherche rapide

Pouvez-vous construire un arbre pour résoudre le nombre secret ?

Application : recherche rapide

Pouvez-vous construire un arbre pour résoudre le nombre secret ?

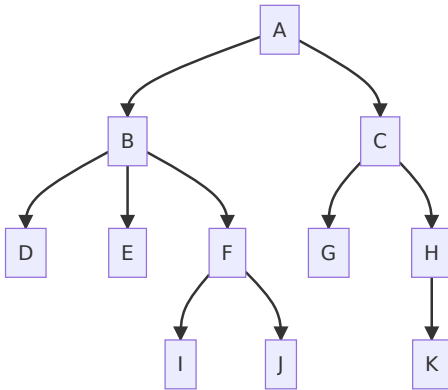
- Le nombre secret ou la recherche dichotomique (nombre entre 0 et 10).



Question : Quelle est la complexité pour trouver un nombre ?

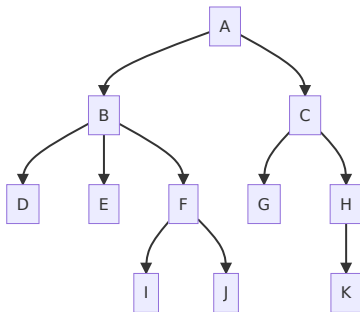
Autres représentations

- Botanique
- **Exercice** : Ajouter les degrés/niveaux et feuilles



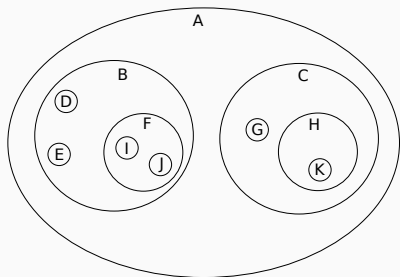
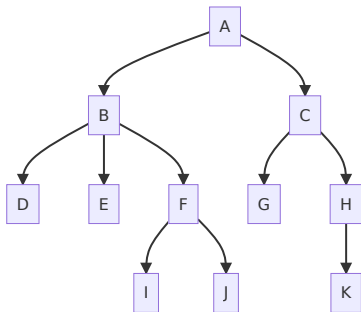
Autres représentations

- Ensembliste



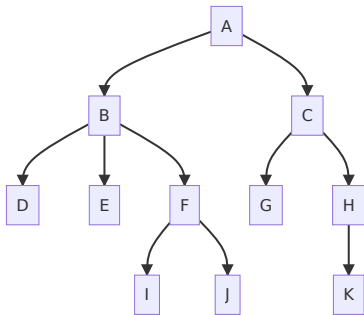
Autres représentations

- Ensembliste



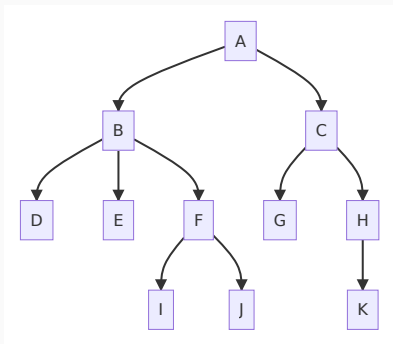
Autres représentations

- Liste



Autres représentations

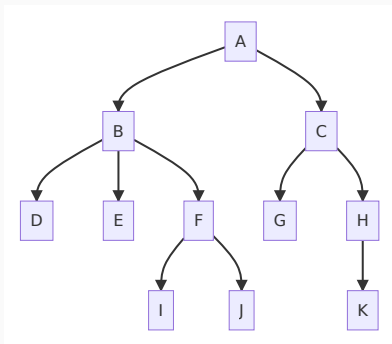
■ Liste



(A
 (B
 (D)
 (E)
 (F
 (I)
 (J)
)
)
 (C
 (G)
 (H
 (K)
)
)
)

Autres représentations

- Par niveau



Autres représentations

- Par niveau

