Ingénierie des systèmes d'information. B-arbre.

Sergey Kirgizov

ESIREM, LIB

2021

Plan

Rappel : Systèmes d'information, indices, B-arbres.

Rappel : Systèmes d'information de bonne qualité

Leurs propriétés

- **Disponibilité :** vitesse de chargement, la latence...
- **Cohérence :** pas des réponses contradictoires...
- **Robustesse :** sauvegarde, archivage, lutte contre les fausses manips...
- **Confidentialité, sécurité :** cryptage, signatures, chiffrement homomorphe, RGPD
- **Ergonomie et esthétique :** interface conviviale Homme-Machine.

src Dr. Marcus Gossler @ Wiki

Rappel : Indices

- **Disponibilité**
- **Cohérence**

Pourquoi ?

- accélère les opérations de recherche
- vérifier rapidement les propriétés des données,
- ... de tri
- ... de jointure

Technologies sous-jacentes :

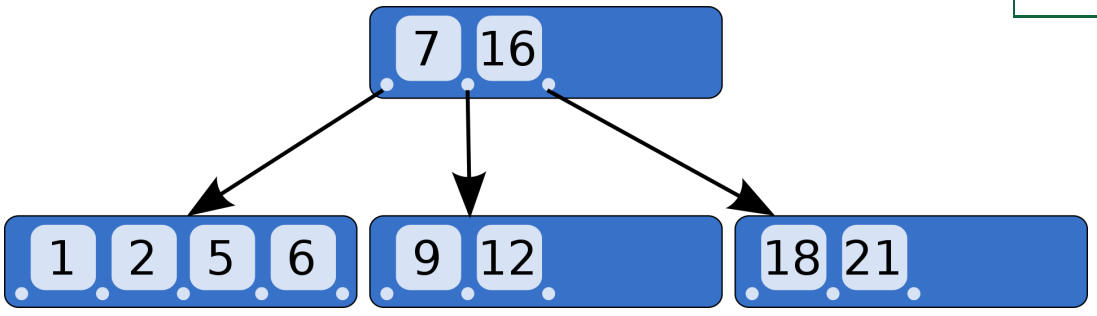
- **B-arbres**
- bitmaps
- tables de hachage

Il y an impact sur les performances en modification/création.

B-arbre

B-tree

- **Disponibilité**
- **Cohérence**



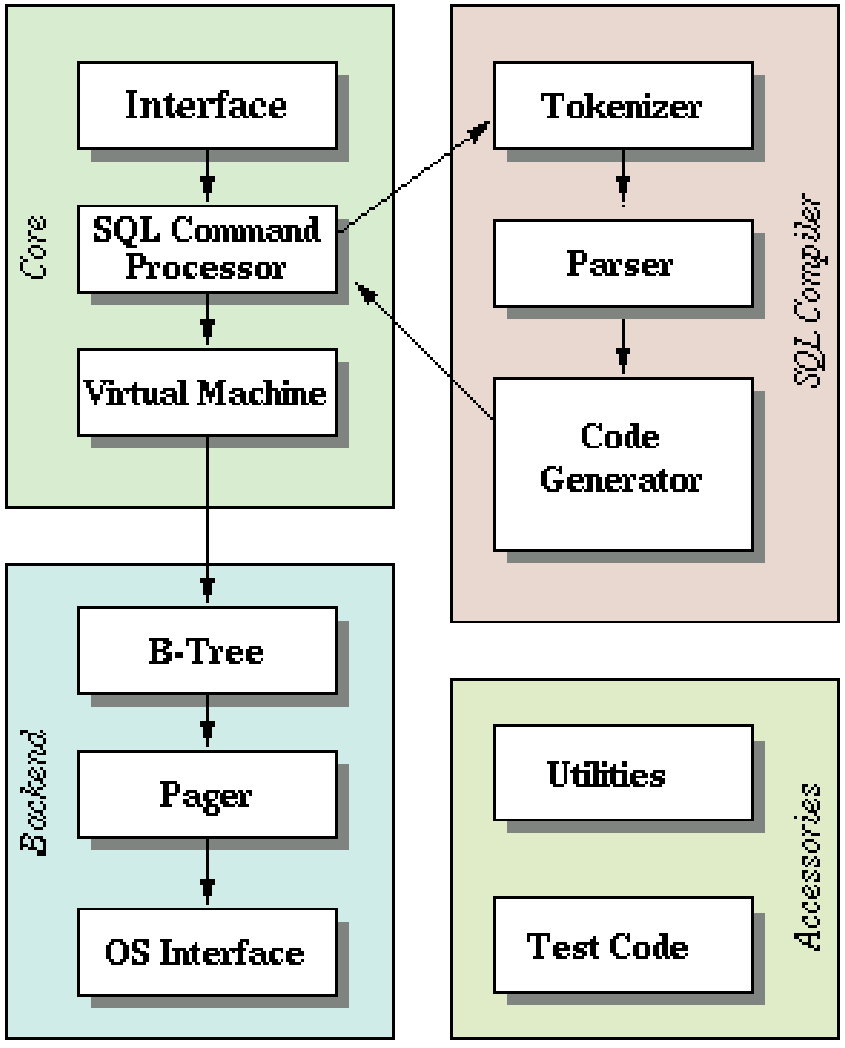
Inventé par : Rudolf Bayer, Edward M. McCreight en **1970** dans BOEING Research Labs.

Applications :

- Base de données : CouchDB, MySQL, Oracle, PostgreSQL, LMDB, SQLite, ...
- Systèmes de fichiers : HFS+, NTFS, jfs2, btrfs, Ext4, ...

Variantes : B+-tree et B*-tree

Architecture de SQLite



https://www.sqlite.org/arch.html

Les arbres... ?!

Système unaire

Hanakapiai Beach

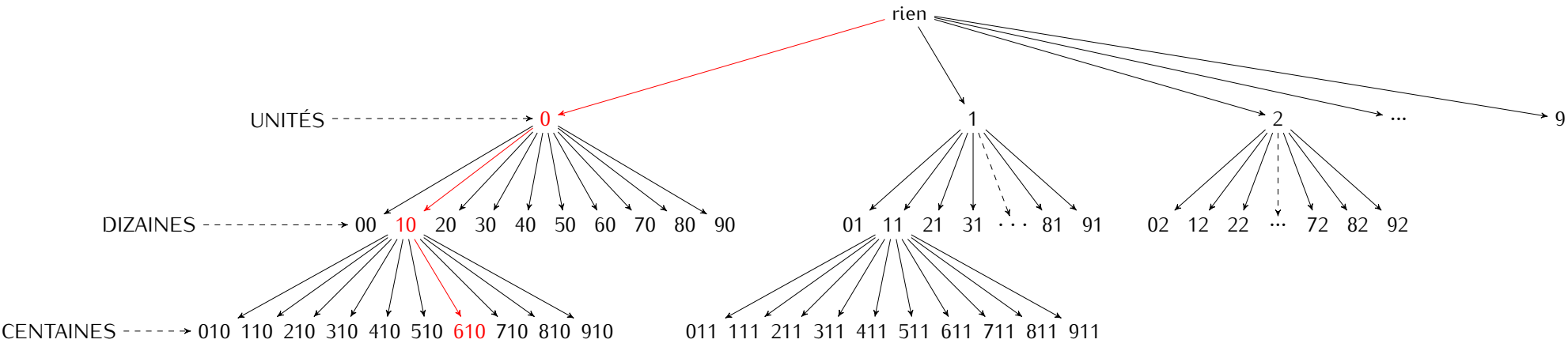
Hanakāpīʻai Beach in 1995

Location	Hawaiian islands
Geology	Beach
Access	No road access

Le système unaire est pratique en certain sens : *lorsque vous augmentez la valeur, vous n'avez rien à effacer, il suffit juste ajouter un marque de dénombrement.*

Inconvénient : il n'est pas pratique de lire ça lorsque les nombres deviennent importants.

Système décimal ≃ un arbre



En lisant le nombre en notation décimale de droite à gauche, nous obtenons le chemin de la racine au nœud souhaité dans cet arbre. Exemple : 610

Livres

- **Algorithms** de Robert Sedgewick et Kevin Wayne `https://algs4.cs.princeton.edu/home/`
- **Introduction to Algorithms** de Thomas H. Cormen, Charles E. Leiserson, et Ronald L. Rivest `http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/toc.htm`
- **The Art of Computer Programming, Volume 3 : Sorting and Searching** de Donald Knuth

Structure de B-arbre

Définition : B-arbre (variante L-U)

Définition : B-arbre (variante L-U)

- Chaque nœud a plusieurs étiquettes (clés) ordonnés $(e_1, e_2, \ldots)$
  - au moins $L - 1$ étiquettes ;
  - au plus $U - 1$ étiquettes.

Définition : B-arbre (variante L-U)

- Chaque nœud a plusieurs étiquettes (clés) ordonnés $(e_1, e_2, \ldots)$
  - au moins $L-1$ étiquettes ;
  - au plus $U-1$ étiquettes.
- Chaque nœud, sauf la racine et feuilles, possède
  - au moins L enfants ;
  - au plus U enfants.
- Si un nœud ($\neq$ feuille) a $k-1$ étiquettes $(e_1, e_2, \ldots e_{k-1})$, alors il a k enfants $(T_1, T_2, \ldots, T_k)$. Chaque étiquette $c_{j,i}$ du sous-arbre T_j respecte la propriété suivante

$$\begin{cases} c_{j,i} < e_1 & \text{si } j = 1, \\ e_{j-1} < c_{j,i} < e_j & \text{si } 1 < j < k, \\ e_{k-1} < c_{j,i} & \text{si } j = k. \end{cases}$$

- Un arbre B est toujours équilibré

En plus des étiquettes on peut aussi stocker la charge utile.

Questions ?

Soit T un B-arbre contenant N clés (variante L-U)

- Quelle sera la hauteur de l'arbre dans le pire des cas ?
- Quelle sera la hauteur de l'arbre dans le meilleur des cas ?

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.
Niveau 1 contient 2 nœuds, $2(L-1)$ clés, il a $2L$ fils.

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.
Niveau 1 contient 2 nœuds, $2(L-1)$ clés, il a $2L$ fils.
Niveau 2 contient $2L$ nœuds, $2L(L-1)$ clés, il a $2LL$ fils.

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.
Niveau 1 contient 2 nœuds, $2(L-1)$ clés, il a $2L$ fils.
Niveau 2 contient $2L$ nœuds, $2L(L-1)$ clés, il a $2LL$ fils.
...
Niveau i contient $2L^{i-1}$ nœuds, $2L^{i-1}(L-1)$ clés, il a $2L^i$ fils.
La hauteur h de l'arbre T vérifie l'inégalité suivante :

$$N \geq 1 + \sum_{i=0}^{h-1} 2L^i(L-1)$$

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.
Niveau 1 contient 2 nœuds, $2(L-1)$ clés, il a $2L$ fils.
Niveau 2 contient $2L$ nœuds, $2L(L-1)$ clés, il a $2LL$ fils.
...
Niveau i contient $2L^{i-1}$ nœuds, $2L^{i-1}(L-1)$ clés, il a $2L^i$ fils.
La hauteur h de l'arbre T vérifie l'inégalité suivante :

$$N \geq 1 + \sum_{i=0}^{h-1} 2L^i(L-1) \geq 1 + 2(L-1)\frac{L^h - 1}{L-1}$$

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.
Niveau 1 contient 2 nœuds, $2(L-1)$ clés, il a $2L$ fils.
Niveau 2 contient $2L$ nœuds, $2L(L-1)$ clés, il a $2LL$ fils.
...
Niveau i contient $2L^{i-1}$ nœuds, $2L^{i-1}(L-1)$ clés, il a $2L^i$ fils.
La hauteur h de l'arbre T vérifie l'inégalité suivante :

$$N \geq 1 + \sum_{i=0}^{h-1} 2L^i(L-1) \geq 1 + 2(L-1)\frac{L^h - 1}{L-1} \geq 2L^h - 1.$$

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.
Niveau 1 contient 2 nœuds, $2(L-1)$ clés, il a $2L$ fils.
Niveau 2 contient $2L$ nœuds, $2L(L-1)$ clés, il a $2LL$ fils.
...
Niveau i contient $2L^{i-1}$ nœuds, $2L^{i-1}(L-1)$ clés, il a $2L^i$ fils.
La hauteur h de l'arbre T vérifie l'inégalité suivante :

$$N \geq 1 + \sum_{i=0}^{h-1} 2L^i(L-1) \geq 1 + 2(L-1)\frac{L^h - 1}{L-1} \geq 2L^h - 1.$$

$$N + 1 \geq 2L^h$$

$$h \leq \log_L \left(\frac{N+1}{2} \right).$$

Le pire de cas.

Soit T un B-arbre contenant N clés (variante L-U).

Niveau 0 contient 1 nœuds, 1 clé, il a deux fils.
Niveau 1 contient 2 nœuds, $2(L-1)$ clés, il a $2L$ fils.
Niveau 2 contient $2L$ nœuds, $2L(L-1)$ clés, il a $2LL$ fils.
...
Niveau i contient $2L^{i-1}$ nœuds, $2L^{i-1}(L-1)$ clés, il a $2L^i$ fils.
La hauteur h de l'arbre T vérifie l'inégalité suivante :

$$N \geq 1 + \sum_{i=0}^{h-1} 2L^i(L-1) \geq 1 + 2(L-1)\frac{L^h - 1}{L-1} \geq 2L^h - 1.$$

$$N + 1 \geq 2L^h$$

$$h \leq \log_L \left(\frac{N+1}{2} \right).$$

Le meilleur des cas est similaire, on peut montrer que

$$h \geq \log_U (N+1) - 1.$$

La charge utile

En plus des étiquettes on peut aussi stocker la charge utile !

Variante minimale d'arbre B : arbre 2-3

Inventé par : John Hopcroft en 1970.

IBM Professor of Engineering and Applied Mathematics.
Cornell, Princeton Stanford Universities.

Interprétation de B-arbre

Interprétation de B-arbre

Opérations :

- Recherche
- Insertion
- Parcours
- Suppression

Recherche dans l'arbre B

Chaque nœud contient l'ensemble d'étiquettes avec les charges utiles associées.

Entrée : k, une clé que nous recherchons dans l'arbre.
Sortie : v, la valeur de la charge utile associée à la clé k s'il existe, NULL sinon.

Recherche dans l'arbre B

Chaque nœud contient l'ensemble d'étiquettes avec les charges utiles associées.

Entrée : k, une clé que nous recherchons dans l'arbre.
Sortie : v, la valeur de la charge utile associée à la clé k s'il existe, NULL sinon.

1. Prendre la racine. La racine est notre *nœud courant*.

Recherche dans l'arbre B

Chaque nœud contient l'ensemble d'étiquettes avec les charges utiles associées.

Entrée : k, une clé que nous recherchons dans l'arbre.
Sortie : v, la valeur de la charge utile associée à la clé k s'il existe, NULL sinon.

1. Prendre la racine. La racine est notre *nœud courant*.
2. Si nœud courant est vide, on retourne NULL.

Recherche dans l'arbre B

Chaque nœud contient l'ensemble d'étiquettes avec les charges utiles associées.

Entrée : k, une clé que nous recherchons dans l'arbre.
Sortie : v, la valeur de la charge utile associée à la clé k s'il existe, NULL sinon.

1. Prendre la racine. La racine est notre *nœud courant*.
2. Si nœud courant est vide, on retourne NULL.
3. Si le nœud courant contient la clé k, alors on termine en retournant la valeur de la charge utile associée.

Recherche dans l'arbre B

Chaque nœud contient l'ensemble d'étiquettes avec les charges utiles associées.

Entrée : k, une clé que nous recherchons dans l'arbre.
Sortie : v, la valeur de la charge utile associée à la clé k s'il existe, NULL sinon.

1. Prendre la racine. La racine est notre *nœud courant*.
2. Si nœud courant est vide, on retourne NULL.
3. Si le nœud courant contient la clé k, alors on termine en retournant la valeur de la charge utile associée.
4. Sinon on trouve un enfant susceptible de contenir la clé k. Cet enfant devient le nœud courant.

Recherche dans l'arbre B

Chaque nœud contient l'ensemble d'étiquettes avec les charges utiles associées.

Entrée : k, une clé que nous recherchons dans l'arbre.
Sortie : v, la valeur de la charge utile associée à la clé k s'il existe, NULL sinon.

1. Prendre la racine. La racine est notre *nœud courant*.
2. Si nœud courant est vide, on retourne NULL.
3. Si le nœud courant contient la clé k, alors on termine en retournant la valeur de la charge utile associée.
4. Sinon on trouve un enfant susceptible de contenir la clé k. Cet enfant devient le nœud courant.
5. On revient vers 2.

Recherche dans l'arbre B

Chaque nœud contient l'ensemble d'étiquettes avec les charges utiles associées.

Entrée : k, une clé que nous recherchons dans l'arbre.
Sortie : v, la valeur de la charge utile associée à la clé k s'il existe, NULL sinon.

1. Prendre la racine. La racine est notre *nœud courant*.
2. Si nœud courant est vide, on retourne NULL.
3. Si le nœud courant contient la clé k, alors on termine en retournant la valeur de la charge utile associée.
4. Sinon on trouve un enfant susceptible de contenir la clé k. Cet enfant devient le nœud courant.
5. On revient vers 2.

Deux façons de la mise en œuvre de cet algorithme existent : **itérative** et **récursive**. Pour plus d'info : voir récursion terminale.

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.
3. Sinon

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.
3. Sinon
   3.1 Trouver une médiane m parmi les clés de A et la nouvelle clé.

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.
3. Sinon
   - 3.1 Trouver une médiane m parmi les clés de A et la nouvelle clé.
   - 3.2 Transformer A en deux nœuds séparés par la médiane m.

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.
3. Sinon
   - 3.1 Trouver une médiane m parmi les clés de A et la nouvelle clé.
   - 3.2 Transformer A en deux nœuds séparés par la médiane m.
   - 3.3 Répéter l'insertion récursivement, c'est-à-dire

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.
3. Sinon
   - 3.1 Trouver une médiane m parmi les clés de A et la nouvelle clé.
   - 3.2 Transformer A en deux nœuds séparés par la médiane m.
   - 3.3 Répéter l'insertion récursivement, c'est-à-dire
     - $A \leftarrow$ parent de A

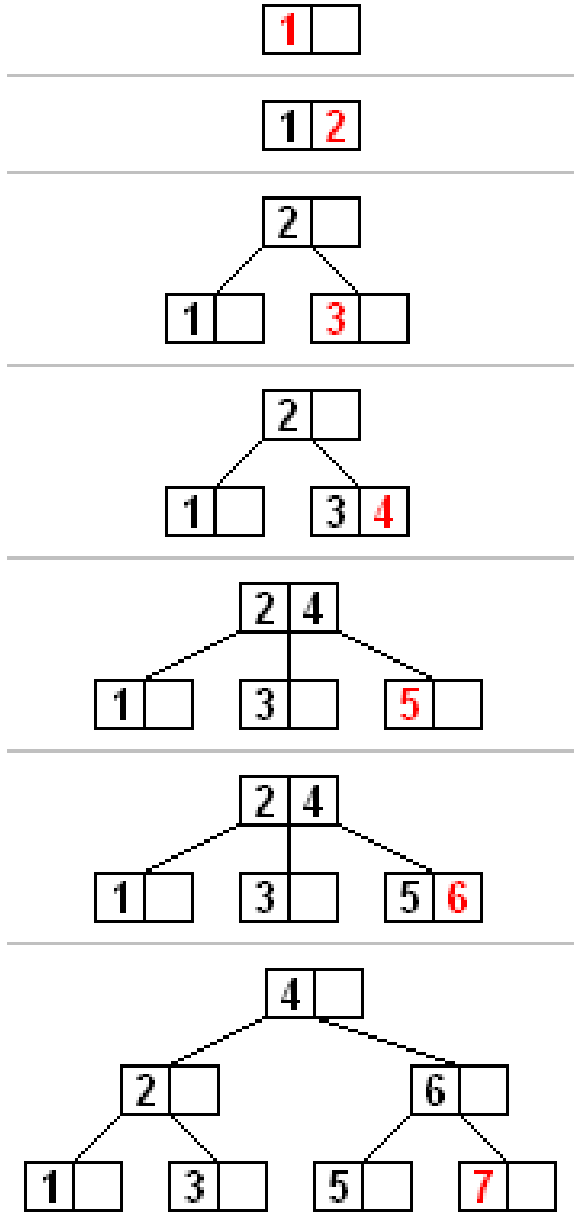
Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.
3. Sinon
   - 3.1 Trouver une médiane m parmi les clés de A et la nouvelle clé.
   - 3.2 Transformer A en deux nœuds séparés par la médiane m.
   - 3.3 Répéter l'insertion récursivement, c'est-à-dire
     - $A \leftarrow$ parent de A
     - Répéter les étapes à partir de 2.

Insertion dans l'arbre B

1. Trouver une feuille où la nouvelle clé k devrait être insérée. Cette feuille est notre *nœud courant* A.
2. Insérer la clé dans A et terminer l'algorithme si A possède un nombre acceptable de clés.
3. Sinon
   - 3.1 Trouver une médiane m parmi les clés de A et la nouvelle clé.
   - 3.2 Transformer A en deux nœuds séparés par la médiane m.
   - 3.3 Répéter l'insertion récursivement, c'est-à-dire
     - $A \leftarrow$ parent de A
     - Répéter les étapes à partir de 2.

Si la clé est insérée dans la racine, qui contient déjà le nombre maximal d'éléments, une nouvelle racine avec un élément sera créée.



src : Maxtremus @ Wikipediaxs

Parcours

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche
  (R) Traverser récursivement son sous-arbre droit

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche
  (R) Traverser récursivement son sous-arbre droit
  (N) Traiter le nœud lui-même

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche
  (R) Traverser récursivement son sous-arbre droit
  (N) Traiter le nœud lui-même
  Trois variantes :
  - préordre (NLR)

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche
  (R) Traverser récursivement son sous-arbre droit
  (N) Traiter le nœud lui-même
  Trois variantes :
  - préordre (NLR)
  - inordre (LNR)

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche
  (R) Traverser récursivement son sous-arbre droit
  (N) Traiter le nœud lui-même
  Trois variantes :
  - préordre (NLR)
  - inordre (LNR)
  - postordre (LRN)

Parcours d'arbre

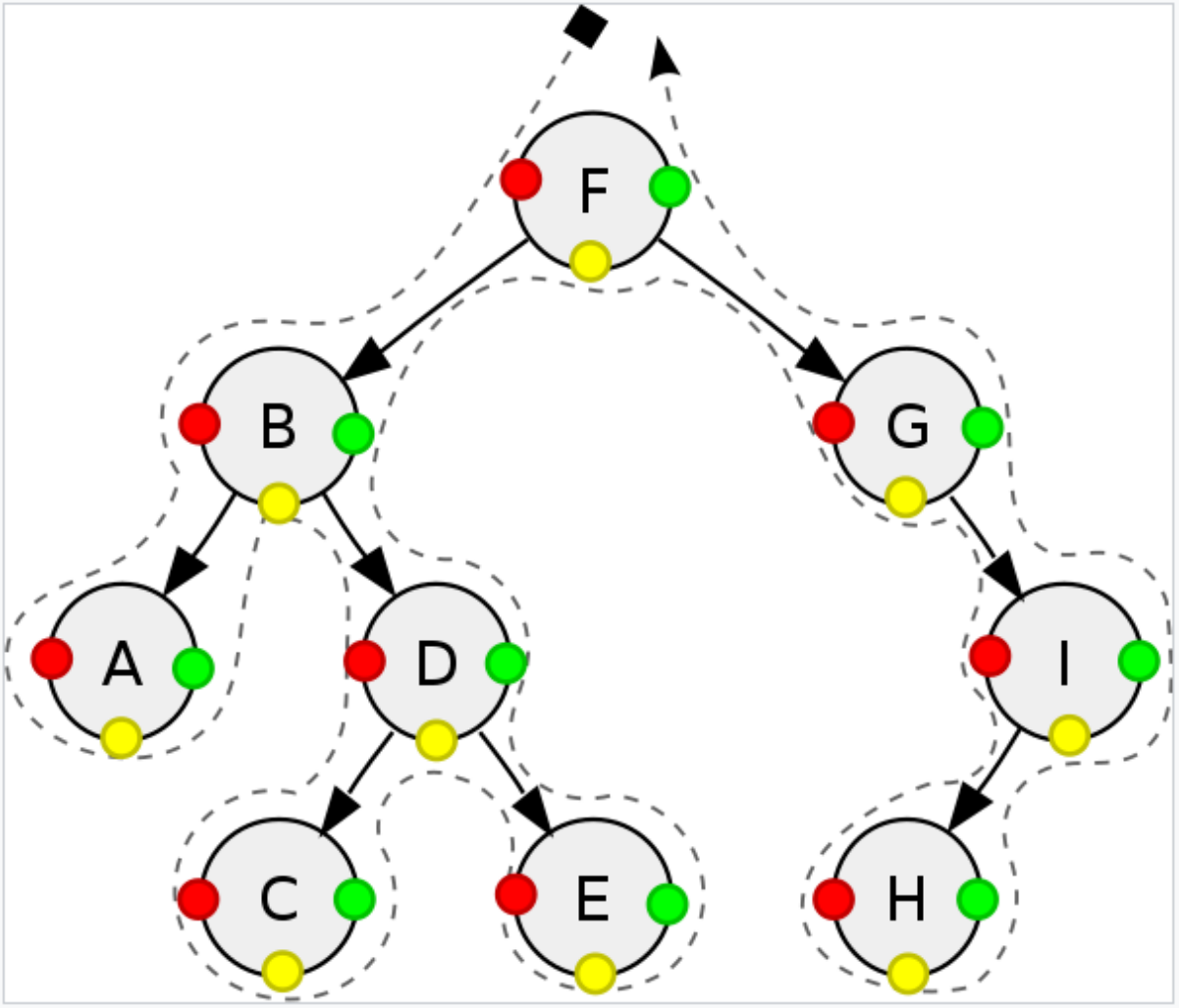
- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche
  (R) Traverser récursivement son sous-arbre droit
  (N) Traiter le nœud lui-même
  Trois variantes :
  - préordre (NLR)
  - inordre (LNR)
  - postordre (LRN)
- parcours en largeur, BFS, Breadth-first search

Parcours d'arbre

- parcours en profondeur, DFS, Depth-First Search
  (L) Traverser récursivement son sous-arbre gauche
  (R) Traverser récursivement son sous-arbre droit
  (N) Traiter le nœud lui-même
  Trois variantes :
  - préordre (NLR)
  - inordre (LNR)
  - postordre (LRN)
- parcours en largeur, BFS, Breadth-first search

Code : `https://en.wikipedia.org/wiki/Tree_traversal#Depth-first_search`

Tous ces algorithmes sont faciles à adapter pour les arbres B.



Depth-first traversal of an example tree:
pre-order (red): F, B, A, D, C, E, G, I, H;
in-order (yellow): A, B, C, D, E, F, G, H, I;
post-order (green): A, C, E, D, B, H, I, G, F.

src : Miles, Pluke, Jochen Burghardt @ Wikipediaxs

Suppression

https://en.wikipedia.org/wiki/B-tree#Deletion

Visualisation

Essayer en-ligne !

David Galles
`https://www.cs.usfca.edu/~galles/visualization/BTree.html`

Complexité

La complexité des algorithmes

La notation grand O de Bachmann-Landau.

On dit

$$f(x) \in O\big(g(x)\big)$$

lorsqu'il existe des constantes $N > 0$ et $C > 0$ telles que pour tout

$$x > N$$

on a

$$|f(x)| \leq C|g(x)|.$$

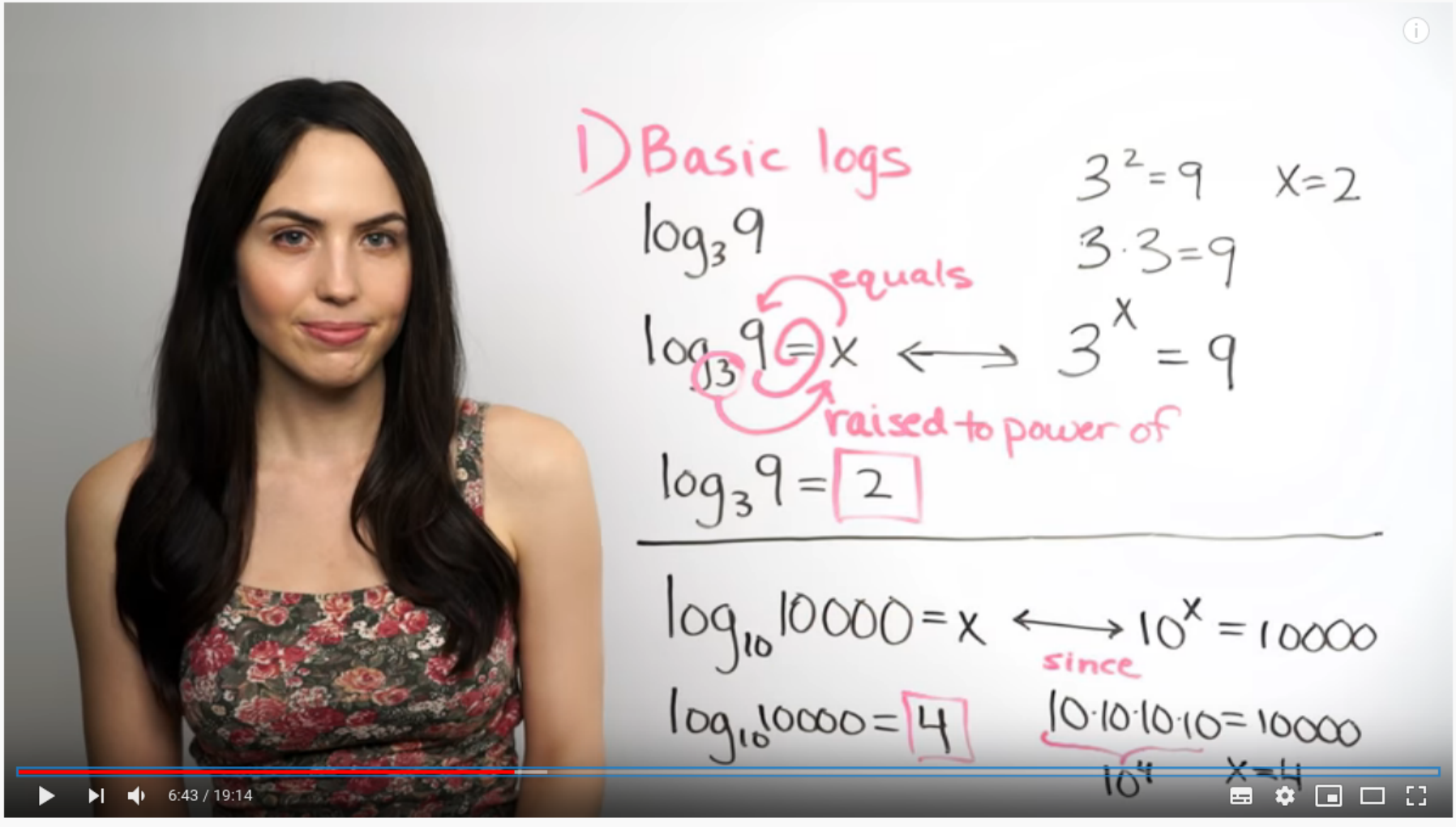
The big-O originally stands for "order of" ("Ordnung")

Complexité

Espace : $O(n)$

Opérations de Recherche, Insertion,
Suppression : $O(log(n))$

Logarithmes ?



Logarithms... How? (NancyPi)

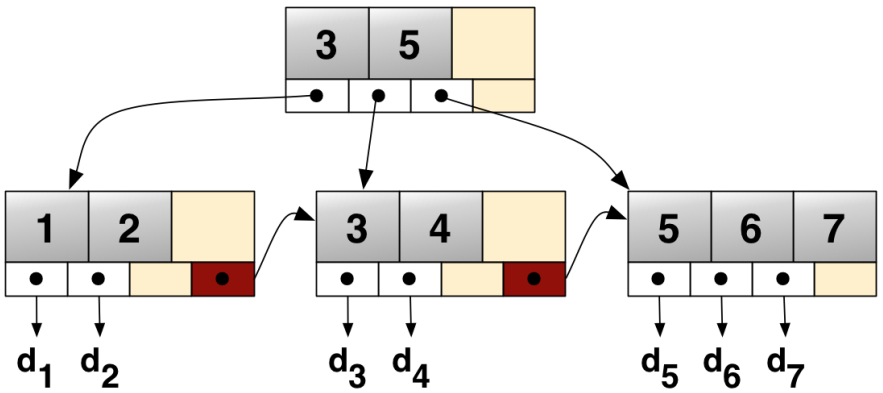
https://www.youtube.com/watch?v=Zw5t6BTQYRU

Variantes

Variantes

Arbre B+ $\approx$ structure linéaire + B-arbre.

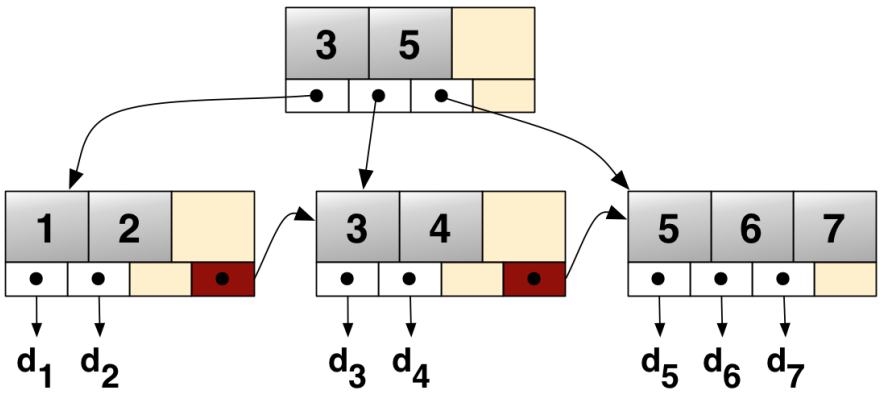
L'arbre B+ diffère légèrement de l'arbre B, en ceci que toutes les données sont stockées exclusivement dans des feuilles, et celles-ci sont reliées entre elles.



src Grundprinzip @ Wiki

Variantes

Arbre B+ ≈ structure linéaire + B-arbre.
L'arbre B+ diffère légèrement de l'arbre B, en ceci que toutes les données sont stockées exclusivement dans des feuilles, et celles-ci sont reliées entre elles.



src Grundprinzip @ Wiki

Arbre B* garantit que les nœuds (sauf la racine) soient remplis au moins aux 2/3 au lieu de 1/2 (Knuth 1998, The Art of Computer Programming, p. 488).

Les applications

Les applications

- **Base de données :** CouchDB, MySQL, Oracle, PostgreSQL, LMDB, SQLite, ... Liste des enregistrements avec plusieurs arbres associés (un arbre pour chaque index).

- **Systèmes de fichiers :** HFS+, NTFS, jfs2, btrfs, Ext4, ...

Questions ?