université PARIS-SACLAY

Bases de Données & Optimisation : Les arbres B+

Thomas Gerald

October 5, 2026

Laboratoire Interdisciplinaire des Sciences du Numérique – LISN, CNRS
thomas.gerald@lisn.upsaclay.fr

Les index : Structure et algorithmes

Index trié et arbre de recherche

Index trié ?

Supposons que nous ayons un très grand nombre de données pouvant être triés sur une clef

- L'accès aux données est rapide $O(log_2(n))$
- La mise à jour implique de déplacer en moyenne la moitié des données (soit de l'index soit de du fichier si type 1)

Peut-on à la fois avoir un accès rapide et une insertion/suppression performante ?

Index trié et arbre de recherche

Index trié ?

Supposons que nous ayons un très grand nombre de données pouvant être triés sur une clef

- L'accès aux données est rapide $O(log_2(n))$
- La mise à jour implique de déplacer en moyenne la moitié des données (soit de l'index soit de du fichier si type 1)

Peut-on à la fois avoir un accès rapide et une insertion/suppression performante ?

Index trié et arbre de recherche

Index trié ?

Supposons que nous ayons un très grand nombre de données pouvant être triés sur une clef de recherche

- L'accès aux données est rapide $O(log_2(n))$
- La mise à jour implique de déplacer en moyenne la moitié des données (soit de l'index soit de du fichier si type 1)

Peut-on à la fois avoir un accès rapide et une insertion/suppression performante ?

Structure arborescentes

Index trié et arbre de recherche

Index trié ?

Supposons que nous ayons un très grand nombre de données pouvant être triés sur une clef de recherche

- L'accès aux données est rapide $O(log_2(n))$
- La mise à jour implique de déplacer en moyenne la moitié des données (soit de l'index soit de du fichier si type 1)

Peut-on à la fois avoir un accès rapide et une insertion/suppression performante ?

Structure arborescentes :

- Index triés non dense successifs

Index trié et arbre de recherche

Index trié ?

Supposons que nous ayons un très grand nombre de données pouvant être triés sur une clef de recherche

- L'accès aux données est rapide $O(log_2(n))$
- La mise à jour implique de déplacer en moyenne la moitié des données (soit de l'index soit de du fichier si type 1)

Peut-on à la fois avoir un accès rapide et une insertion/suppression performante ?

Les Structures arborescentes :

- Index triés non dense successifs
- B-Tree (bayer ou balanced tree)

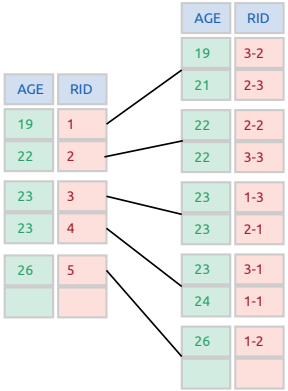
  → "'Organization and maintenance of large ordered indices", R. Bayer et E. McCreight, 1970

Imbrication d'index non-dense

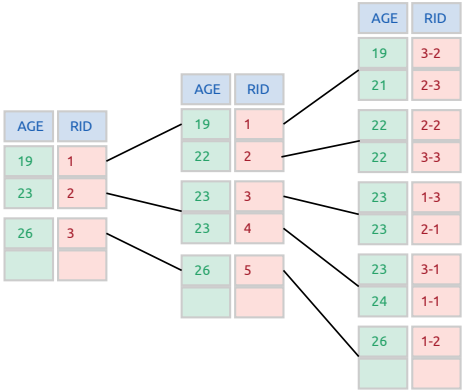
Imbrication d'index non-dense

AGE	RID
19	3-2
21	2-3
22	2-2
22	3-3
23	1-3
23	2-1
23	3-1
24	1-1
26	1-2

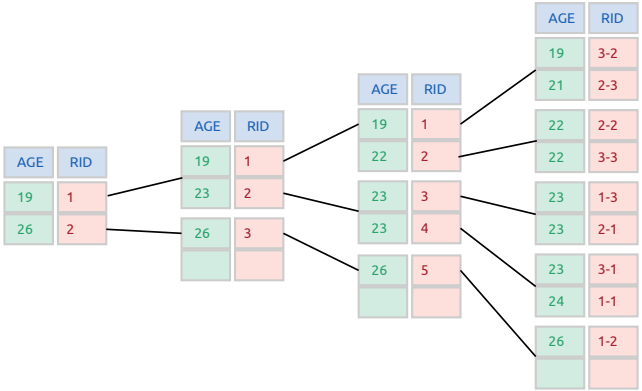
Imbrication d'index non-dense



Imbrication d'index non-dense



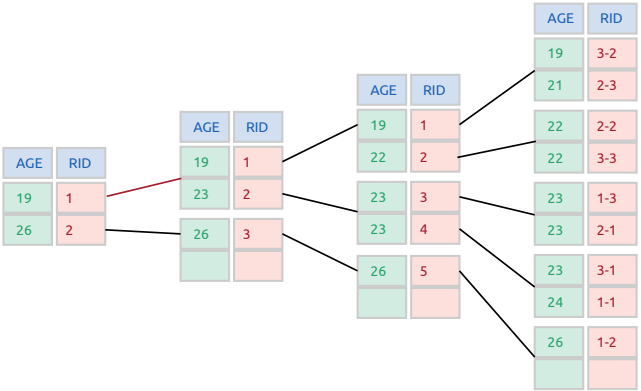
Imbrication d'index non-dense



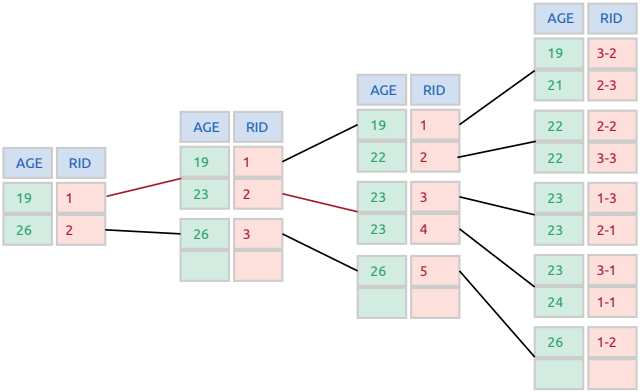
Les arbre B+ : Index trié

Recherche d'une valeur (24)?

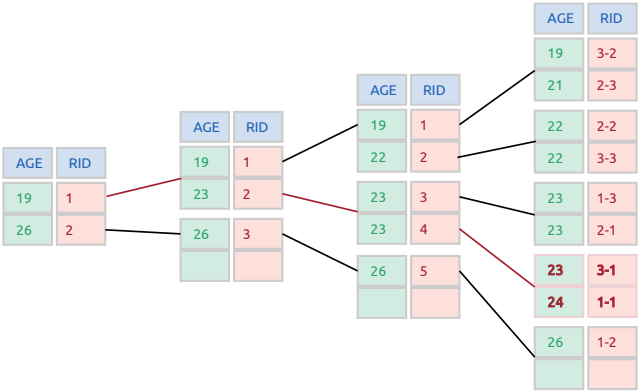
Recherche d'une valeur (24)?



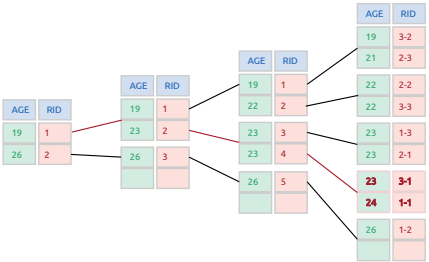
Recherche d'une valeur (24)?



Recherche d'une valeur (24)?

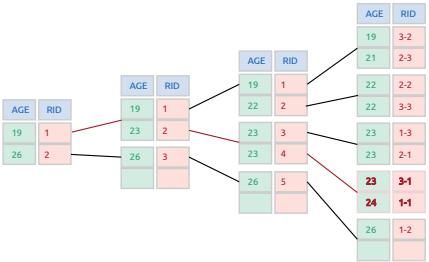


Recherche d'une valeur (24)?



$\rightarrow$ Le nombre de pages ouvertes correspond à la hauteur de l'arbre $h + 1$!!!

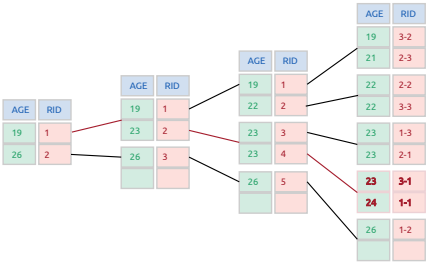
Recherche d'une valeur (24)?



→ Le nombre de pages ouvertes correspond à la hauteur de l'arbre $h + 1$!!!

→ Chaque noeud/étage à environ k (ici 2) enfants

Recherche d'une valeur (24)?

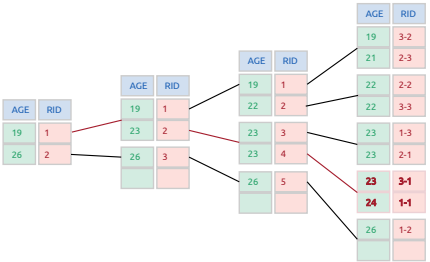


→ Le nombre de pages ouvertes correspond à la hauteur de l'arbre $h + 1$!!!

→ Chaque noeud/étage à environ k (ici 2) enfants

→ Il y a entre 2^{k-1} et 2^k feuilles (multiplication par k à chaque étage)

Recherche d'une valeur (24)?



→ Le nombre de pages ouvertes correspond à la hauteur de l'arbre $h + 1$!!!

→ Chaque noeud/étage à environ k (ici 2) enfants

→ Il y a entre 2^{k-1} et 2^k feuilles (multiplication par k à chaque étage)

→ La hauteur de l'arbre est $sup(log(P))$ où P est le nombre de page

→ L'arbre B utilise la même idée pour retrouver l'information

Les arbre B : Définition

Arbre B :

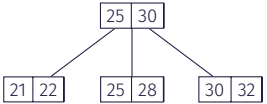
Un arbre B est une généralisation de l'arbre binaire de recherche

Les arbre B : Définition

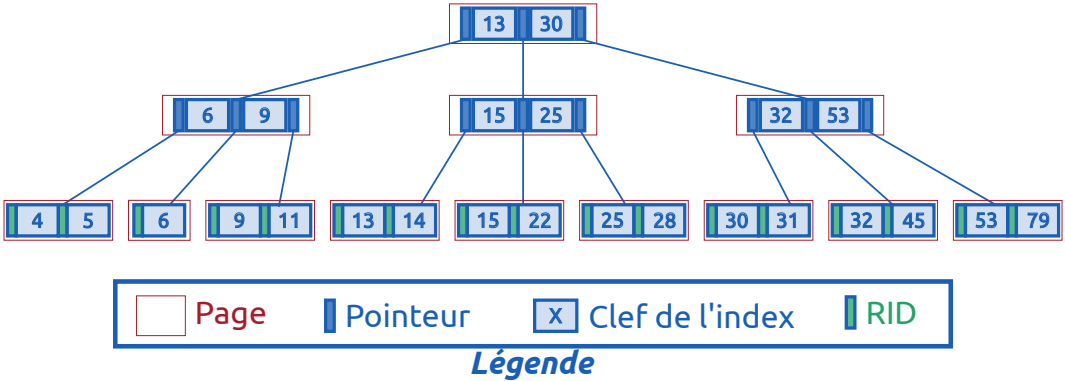
Arbre B :

Un arbre B est une généralisation de l'arbre binaire de recherche

- Les noeuds de l'arbre correspondent à des pages de l'index
  - Arbre d'ordre N $\rightarrow$ nombre éléments dans un noeuds $\leq 2N$
  - Noeuds internes $\rightarrow N \leq$ nombre d'éléments $\leq 2N$
  - La racine $\rightarrow 1 \leq$ nombre d'éléments $\leq 2N$
- Si un noeud à *m* éléments alors il a m+1 fils



- Toutes les feuilles de l'arbre B sont au même niveau (arbre totalement équilibré)
- Les feuilles contiennent les entrées de l'index (lien vers l'emplacement des données)


13
30
6
9
15
25
32
53
4
5
6
9
11
13
14
15
22
25
28
30
31
32
45
53
79
Page
Pointeur
X
Clef de l'index
RID
Légende

Algorithme de recherche

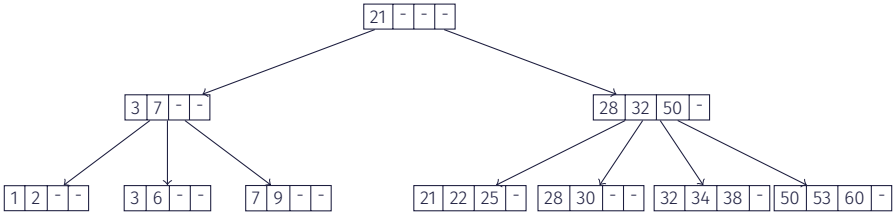
Algorithme de descente dans l'arbre de recherche

Algorithm 1 Find

Require: La clef de recherche K
  $P \leftarrow TreeSearch(root, K)$
  if $K \in P.values()$ **then**
    return P ▷ on retourne le rid
  else
    return *NULL* ▷ non trouvé

Algorithm 2 TreeSearch

Require: un noeud P, la clef de recherche K
  if $P \in leaf$ **then**
    return P
  else if $K < K_1$ **then**
    return $TreeSearch(P_0, K)$
  else if $K > K_n$ **then**
    return $TreeSearch(P_n, K)$
  else
    $k \leftarrow i$ tq $K_i \leq K < K_{i+1}$
    return $TreeSearch(P_k, K)$
  end if=0



Recherche d'une entrées d'index :

Rechercher l'enregistrement de valeur **34**

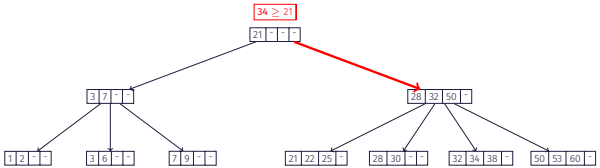


Figure 1: Descente dans le sous-arbre de droite ($34 > 21$)

Algorithm 3 TreeSearch

Require: un noeud P, la clef de recherche K

```
if P ∈ leaf then
    return P
else if K < K_1 then
    return TreeSearch(P_0, K)
else if K > K_n then
    return TreeSearch(P_n, K)
else
    k ← i tq K_i ≤ K < K_{i+1}
    return TreeSearch(P_i, K)
end if
```

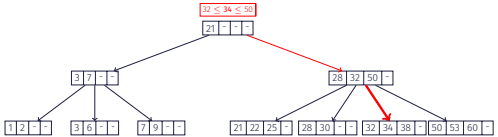
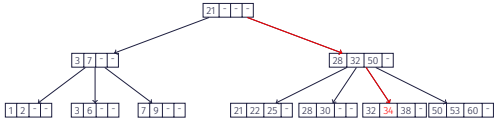


Figure 2: Descente dans le troisième fils sous-arbre de droite ($32 \leq$ **34** ≤ 50)

Algorithm 4 TreeSearch

Require: un noeud P, la clef de recherche K

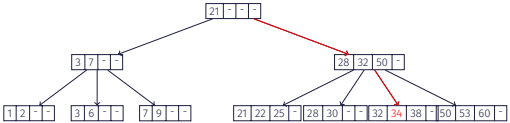
```
if P ∈ leaf then
    return P
else if K < K_1 then
    return TreeSearch(P_0, K)
else if K > K_n then
    return TreeSearch(P_n, K)
else
    k ← i tq K_i ≤ K < K_{i+1}
    return TreeSearch(P_i, K)
end if
```



Algorithm 5 TreeSearch

Require: un noeud P, la clef de recherche K
  if $P \in leaf$ **then**
    return P
  else if $K < K_1$ **then**
    return $TreeSearch(P_0, K)$
  else if $K > K_n$ **then**
    return $TreeSearch(P_n, K)$
  else
    $k \leftarrow i$ tq $K_i \leq K < K_{i+1}$
    return $TreeSearch(P_i, K)$
  end if

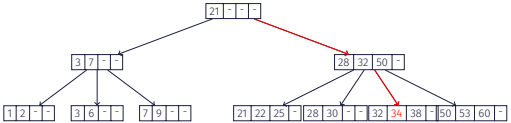
Arbre B : Recherche



Nombre d'opérations

- Nombre de pages ouvertes ?
  - Le nombre de niveaux de l'arbre (hauteur + 1)
- Comparaison de recherche dans les noeuds ?
  - log_2 de la taille des noeuds ($< log_2(4)$ dans notre cas) $\rightarrow$ recherche par dichotomie

Arbre B : Recherche

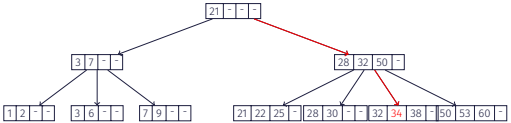


Nombre d'opérations

- Nombre de pages ouvertes ?
  - Le nombre de niveaux de l'arbre (hauteur + 1)
- Comparaison de recherche dans les noeuds ?
  - log_2 de la taille des noeuds ($< log_2(4)$ dans notre cas) $\rightarrow$ recherche par dichotomie

Temps (max) de recherche avec T_p et T_e (lecture page, lecture enregistrement)

Arbre B : Recherche



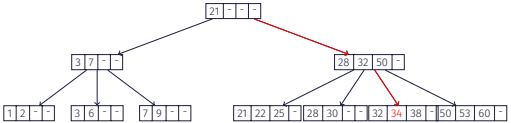
Nombre d'opérations

- Nombre de pages ouvertes ?
  - Le nombre de niveaux de l'arbre (hauteur + 1)
- Comparaison de recherche dans les noeuds ?
  - log_2 de la taille des noeuds ($< log_2(4)$ dans notre cas) $\rightarrow$ recherche par dichotomie

Temps (max) de recherche avec T_p et T_e (lecture page, lecture enregistrement)

- $(h+1) \times (T_p + \log_2(2N) \times T_e)$ (notons que $T_p >> T_e$)

Arbre B : Recherche



Nombre d'opérations

- Nombre de pages ouvertes ?
  - Le nombre de niveaux de l'arbre (hauteur + 1)
- Comparaison de recherche dans les noeuds ?
  - log_2 de la taille des noeuds ($< log_2(4)$ dans notre cas) $\rightarrow$ recherche par dichotomie

Temps (max) de recherche avec T_p et T_e (lecture page, lecture enregistrement)

- $(h+1) \times (T_p + \log_2(2N) \times T_e)$ (notons que $T_p >> T_e$)
- $(h+1)$ ouvertures de pages

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles
$\rightarrow \approx \frac{R}{2N}$ (les feuilles sont remplies)

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles
$\rightarrow \approx \frac{R}{2N}$ (les feuilles sont remplies)
Calculez p_{max}^{h-1} le nombre maximal de parent

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles
$\rightarrow \approx \frac{R}{2N}$ (les feuilles sont remplies)
Calculez p_{max}^{h-1} le nombre maximal de parent
$\rightarrow \approx \frac{L_{max}}{N+1}$ (chaque noeud parent à au moins $N+1$ enfants)

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles
$\rightarrow \approx \frac{R}{2N}$ (les feuilles sont remplies)
Calculez p_{max}^{h-1} le nombre maximal de parent
$\rightarrow \approx \frac{L_{max}}{N+1}$ (chaque noeud parent à au moins $N+1$ enfants)
Calculez p_{max}^{h-2} le nombre maximal de noeuds de profondeur h-2

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles
$\rightarrow \approx \frac{R}{2N}$ (les feuilles sont remplies)
Calculez p_{max}^{h-1} le nombre maximal de parent
$\rightarrow \approx \frac{L_{max}}{N+1}$ (chaque noeud parent à au moins $N+1$ enfants)
Calculez p_{max}^{h-2} le nombre maximal de noeuds de profondeur h-2
$\rightarrow \approx \frac{p_{max}^{h-1}}{N+1} = \frac{L_{max}}{(N+1)^2}$ (chaque noeud parent à entre $N+1$ et $2N+1$ enfants)

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles
$\rightarrow \approx \frac{R}{2N}$ (les feuilles sont remplies)
Calculez p_{max}^{h-1} le nombre maximal de parent
$\rightarrow \approx \frac{L_{max}}{N+1}$ (chaque noeud parent à au moins $N+1$ enfants)
Calculez p_{max}^{h-2} le nombre maximal de noeuds de profondeur h-2
$\rightarrow \approx \frac{p_{max}^{h-1}}{N+1} = \frac{L_{max}}{(N+1)^2}$ (chaque noeud parent à entre $N+1$ et $2N+1$ enfants)
Calculez h_{max} la profondeur maximal de l'arbre

Arbre B : Recherche

Hauteur maximal de l'arbre ?

- R le nombre d'enregistrement
- N le degré de l'arbre (nombre de clefs minimum par noeud)

Calculez L_{max} le nombre maximal de feuilles
$\rightarrow \approx \frac{R}{N}$ (les feuilles sont à moitié remplies)
Calculez L_{min} le nombre minimal de feuilles
$\rightarrow \approx \frac{R}{2N}$ (les feuilles sont remplies)
Calculez p_{max}^{h-1} le nombre maximal de parent
$\rightarrow \approx \frac{L_{max}}{N+1}$ (chaque noeud parent à au moins $N+1$ enfants)
Calculez p_{max}^{h-2} le nombre maximal de noeuds de profondeur h-2
$\rightarrow \approx \frac{p_{max}^{h-1}}{N+1} = \frac{L_{max}}{(N+1)^2}$ (chaque noeud parent à entre $N+1$ et $2N+1$ enfants)
Calculez h_{max} la profondeur maximal de l'arbre
$\rightarrow \approx \lceil log_{N+1}(L_{max}) \rceil$ (valeur entière)

Arbre B : un intervalle ?

Recherche d'un intervalle de valeurs

Supposons que nous souhaitons obtenir les pages/enregistrements ou la valeur de la clef de l'index est v_i tel que $a \leq v_i < b$

Arbre B : un intervalle ?

Recherche d'un intervalle de valeurs

Supposons que nous souhaitons obtenir les pages/enregistrements ou la valeur de la clef de l'index est v_i tel que $a \leq v_i < b$

1. Tester les valeurs possibles $v_i \in \{a, a+1, a+2, \ldots b-1\}$?

Arbre B : un intervalle ?

Recherche d'un intervalle de valeurs

Supposons que nous souhaitons obtenir les pages/enregistrements ou la valeur de la clef de l'index est v_i tel que $a \leq v_i < b$

1. Tester les valeurs possibles $v_i \in \{a, a+1, a+2, \dots b-1\}$?
   - ⚠ $b - a$ recherches Opération trop couteuse en ressources

Arbre B : un intervalle ?

Recherche d'un intervalle de valeurs

Supposons que nous souhaitons obtenir les pages/enregistrements ou la valeur de la clef de l'index est v_i tel que $a \leq v_i < b$

1. Tester les valeurs possibles $v_i \in \{a, a+1, a+2, \dots b-1\}$?
   - ⚠ $b - a$ recherches Opération trop couteuse en ressources
   - Irréalisable ⚠ ne fonctionne pas pour des valeurs dans des espaces "continues" (flotants)

Arbre B : un intervalle ?

Recherche d'un intervalle de valeurs

Supposons que nous souhaitons obtenir les pages/enregistrements ou la valeur de la clef de l'index est v_i tel que $a \leq v_i < b$

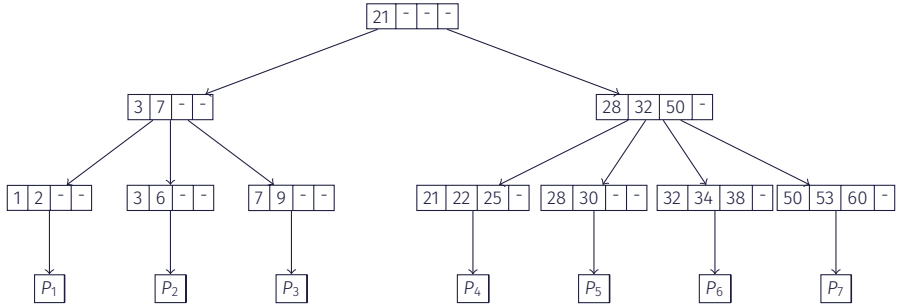
1. Tester les valeurs possibles $v_i \in \{a, a+1, a+2, \dots b-1\}$?
   - ⚠ $b - a$ recherches Opération trop couteuse en ressources
   - Irréalisable ⚠ ne fonctionne pas pour des valeurs dans des espaces "continues" (flotants)
2. Faire en sorte que les données soient contigus pour les entrées de l'index ?

Arbre B : un intervalle ?

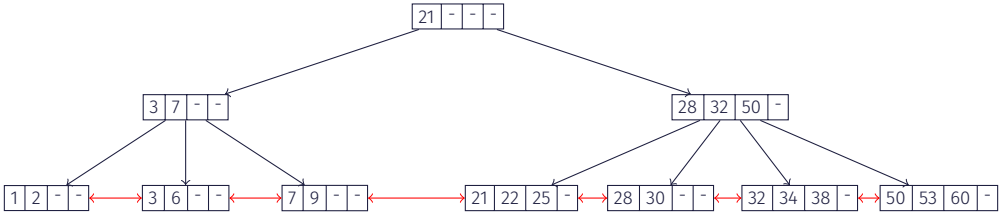
Recherche d'un intervalle de valeurs

Supposons que nous souhaitons obtenir les pages/enregistrements ou la valeur de la clef de l'index est v_i tel que $a \leq v_i < b$

1. Tester les valeurs possibles $v_i \in \{a, a+1, a+2, \ldots b-1\}$?
   - ⚠ $b - a$ recherches Opération trop couteuse en ressources
   - Irréalisable ⚠ ne fonctionne pas pour des valeurs dans des espaces "continues" (flotants)
2. Faire en sorte que les données soient contigus pour les entrées de l'index ?
   - Rapide
   - Maintenir la structure.... (décalage)



- $\forall x \in P_i, y \in P_j$ ssi $i < j \Rightarrow x < y$ (par exemple toutes les valeurs de P_2 sont inférieurs aux valeurs de $P4$)
- Si on sélectionne deux valeurs a et b alors on veut retrouver la page de la valeur a et les pages suivantes jusqu'à la valeur b (pour $a = 7$ et $b = 38$ on souhaite récupérer les pages **successives** P_3, P_4, P_5, P_6)



Arbre B+ : un intervalle

- Ajout d'une liste doublement chainée sur les feuilles de l'arbre !
- Retrouver la valeur max/min puis parcours de la liste chainée

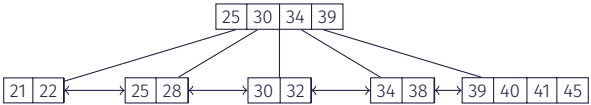
Arbre B+ : Insertion

Insertion :

- Ajouter l'élément dans la bonne feuille
- Restructurer l'arbre si besoin (contrainte sur nombre d'éléments par feuille non respectée)

Dans la suite :

- On va considérer $N = 2$ (maximum de 4 valeurs par noeud)



Arbre B+ : Ajout d'un élément

Procédure pour l'ajout d'un élément

- Rechercher la feuille correspondante à la nouvelle valeurs (tree search)

Arbre B+ : Ajout d'un élément

Procédure pour l'ajout d'un élément

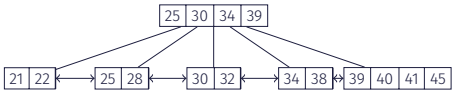
- Rechercher la feuille correspondante à la nouvelle valeurs (tree search)
- Ajouter dans la page correspondant à la feuille l'enregistrement (ou l'entrée de l'index) !!!

Arbre B+ : Ajout d'un élément

Procédure pour l'ajout d'un élément

- Rechercher la feuille correspondante à la nouvelle valeurs (tree search)
- Ajouter dans la page correspondant à la feuille l'enregistrement (ou l'entrée de l'index) !!!

Ajouter un enregistrement de clef d'index 23 :



Arbre B+ : Ajout d'un élément

Procédure pour l'ajout d'un élément

- Rechercher la feuille correspondante à la nouvelle valeurs (tree search)
- Ajouter dans la page correspondant à la feuille l'enregistrement (ou l'entrée de l'index) !!!

Ajouter un enregistrement de clef d'index 23 :

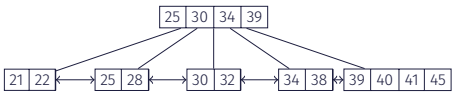


Arbre B+ : Ajout d'un élément

Procédure pour l'ajout d'un élément

- Rechercher la feuille correspondante à la nouvelle valeurs (tree search)
- Ajouter dans la page correspondant à la feuille l'enregistrement (ou l'entrée de l'index) !!!

Ajouter un enregistrement de clef d'index 43 :



La contrainte sur les noeuds n'est pas respectée !!!

Procédure pour l'ajout d'un élément

- Rechercher la feuille correspondante à la nouvelle valeurs (tree search)
- Ajouter dans la page correspondant à la feuille l'enregistrement (ou l'entrée de l'index) !!!

Ajouter un enregistrement de clef d'index 43 :



La contrainte sur les noeuds n'est pas respectée !!!

Ajouter un enregistrement avec une valeur de clef 43 :



La contrainte sur les noeuds n'est pas respectée !!!

Arbre B+ : Ajout d'un élément

Ajouter un enregistrement avec une valeur de clef 43 :



La contrainte sur les noeuds n'est pas respectée !!!

Solution : Éclatement d'un noeud

- Nous voulons "éclater" le noeud suivant 39 | 40 | 41 | 43 | 45
- On choisit l'élément médian 41
- On crée un arbre où 41 est la racine et les valeurs inférieurs 39 | 40 correspondent à un nouveau noeud (gauche) et les valeurs supérieurs 41 | 43 | 45 correspondent au noeud droit du nouvel arbre

Ajouter un enregistrement avec une valeur de clef 43 :



La contrainte sur les noeuds n'est pas respectée !!!

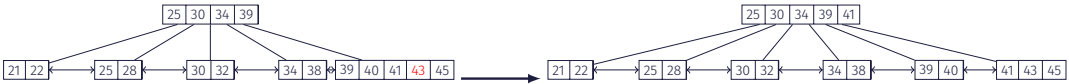
Solution : Éclatement d'un noeud



Figure 3: Exemple d'éclatement d'un noeud

- **Feuille :** 41 est présent dans le noeud fils mais aussi dans le noeud parent
- **Il faut maintenant insérer le sous arbre dans le noeud parent**

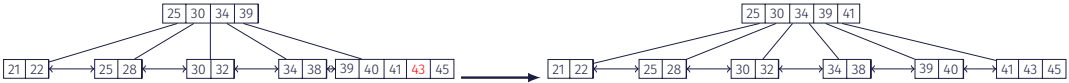
Ajouter un enregistrement avec une valeur de clef 43 (suite):



La contrainte sur la racine n'est pas respectée (5 valeurs)!!

Arbre B+ : Ajout d'un élément

Ajouter un enregistrement avec une valeur de clef 43 (suite):

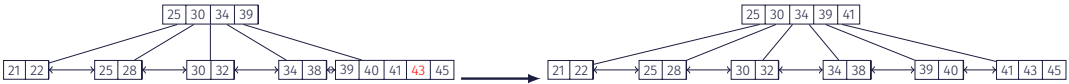


La contrainte sur la racine n'est pas respectée (5 valeurs)!!

Solution : Éclatement d'un noeud interne

- Nous voulons "éclater" le noeud suivant 25 | 30 | 34 | 39 | 41
- On choisit l'élément médian 34
- On crée un arbre où 34 est la racine et les valeurs inférieurs 35 | 30 correspondent à un nouveau noeud (gauche) et les valeurs strictements supérieurs 39 | 41 correspondent au noeud droit du nouvel arbre

Ajouter un enregistrement avec une valeur de clef 43 (suite):



La contrainte sur la racine n'est pas respectée (5 valeurs)!!

Solution : Éclatement d'un noeud interne



Figure 4: Exemple d'éclatement d'un noeud

- **Interne :** Pas de redondance de la valeur médiane !!!
- **On remplace la racine**

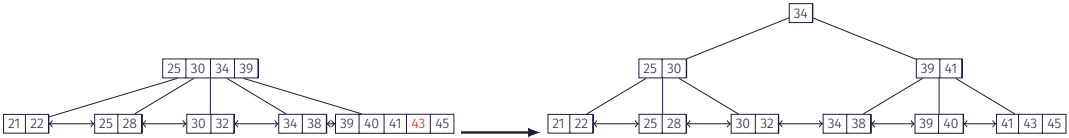
Solution : Éclatement d'un noeud interne



Figure 5: Exemple d'éclatement d'un noeud

- **Interne :** Pas de redondance de la valeur médiane !!!
- **On remplace la racine**

Ajouter un enregistrement avec une valeur de clef 43 (suite):



Arbre B+ : Ajout d'un élément (Synthèse)

Algorithme d'insertion

1. **Rechercher** la feuille d'insertion de F

Arbre B+ : Ajout d'un élément (Synthèse)

Algorithme d'insertion

1. **Rechercher** la feuille d'insertion de F
2. Si F non pleine, alors on ajoute l'entrée à la bonne place

Arbre B+ : Ajout d'un élément (Synthèse)

Algorithme d'insertion

1. **Rechercher** la feuille d'insertion de F
2. Si F non pleine, alors on ajoute l'entrée à la bonne place
3. Sinon **Éclatement de F** en G(fils gauche) et D(fils droit)

Arbre B+ : Ajout d'un élément (Synthèse)

Algorithme d'insertion

1. **Rechercher** la feuille d'insertion de F
2. Si F non pleine, alors on ajoute l'entrée à la bonne place et Fin
3. Sinon **Éclatement de F** en G(fils gauche) et D(fils droit)
   - 3.1 Retrouver l'élément médian M de F (avec l'entrée à la bonne place)
   - 3.2 Ajouter dans G les élément de F $< M$
   - 3.3 Ajouter dans D les éléments de F
     - $\geq M$ si feuille
     - $> M$ si noeud interne
   - 3.4 Insérer le sous arbre constituer de M comme racine et de G et D dans le noeud parent P
   - 3.5 Reprendre à 2 (avec $F \leftarrow P$)

Arbre B+ : Ajout d'un élément (Synthèse)

Algorithme d'insertion

1. **Rechercher** la feuille d'insertion de F
2. Si F non pleine, alors on ajoute l'entrée à la bonne place
3. Sinon **Éclatement de F** en G(fils gauche) et D(fils droit)
   - 3.1 Retrouver l'élément médian M de F (avec l'entrée à la bonne place)
   - 3.2 Ajouter dans G les élément de F $< M$
   - 3.3 Ajouter dans D les éléments de F
     - $\geq M$ si feuille
     - $> M$ si noeud interne
   - 3.4 Insérer le sous arbre constituer de M comme racine et de G et D dans le noeud parent P
   - 3.5 Reprendre à 2 (avec $F \leftarrow P$)

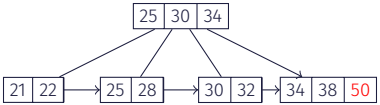
Arbre B+ : Ajout d'un élément (Compléxité)

Complexité

- Coût de la recherche $O(D)$
- Coût de l'insertion $O(D)$ (dans le pire cas D éclatements)

Suppression d'un élément
Il existe plusieurs cas de suppression certains nécéssitent une réorganisation de la structure (restructuration)

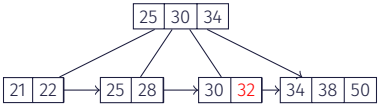
Suppression d'un élément



Cas 1 : Suppression sans restructuration

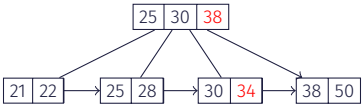
- On supprime la feuilles (pas de rééquilibrage nécessaire)
- $log(D)$ lectures + une écriture

Suppression d'un élément

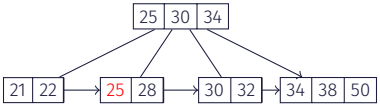


Cas 2 : suppression d'un élément menant un sous remplissage de feuille

- Rééquilibrage nécessaire
  - échanger une entrée avec le voisin de droite

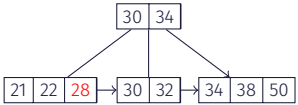


Suppression d'un élément



Cas 3 : suppression d'un élément menant un sous remplissage de feuille et aucun voisins pour échange

- Rééquilibrage nécessaire
  - Fusion de feuilles



Arbre B + : Création

Construction d'un index B-tree

Objectif : Créer un index sur un fichier de données.

Arbre B + : Création

Construction d'un index B-tree

Objectif : Créer un index sur un fichier de données.

1. **Par insertion successive ?** (répéter l'insertion autant qu'il y a de données)

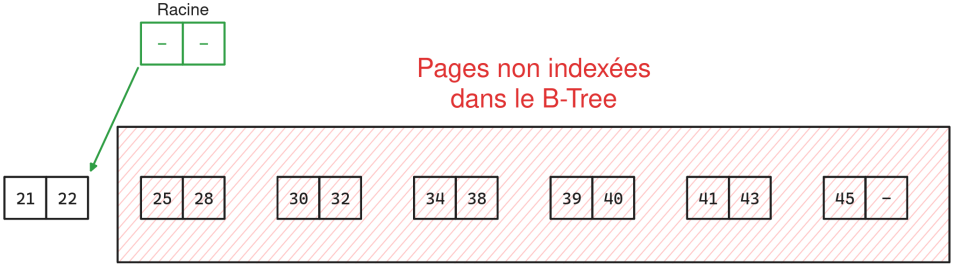
Construction d'un index B-tree

Objectif : Créer un index sur un fichier de données.

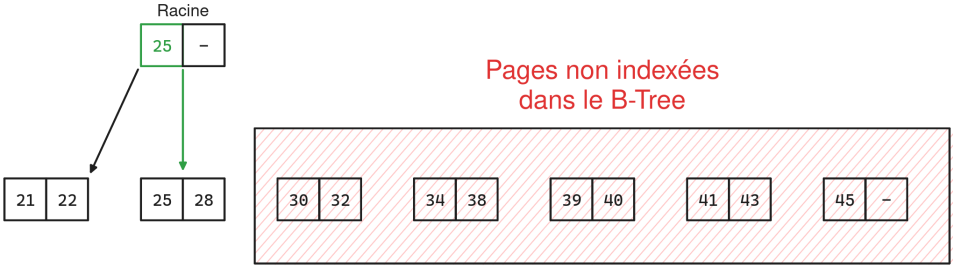
1. **Par insertion successive ?** (répéter l'insertion autant qu'il y a de données)
   Pas très efficace (Rechargement de pages)
2. **Bulk Loading** (répéter l'insertion autant qu'il y a de données)
   Création depuis les feuilles jusqu'à la racine à partir des données triés
   - **ETAPE 1 :** trie des entrées de l'index
   - **ETAPE 2 :** Successivement rassembler les pages d'entrées de l'index (index non dense)

   Les feuilles sont fixées et ne seront pas modifiées, les dernières pages ouvertes sont (presque) les mêmes d'une étape à l'autre (utilisation du cache)

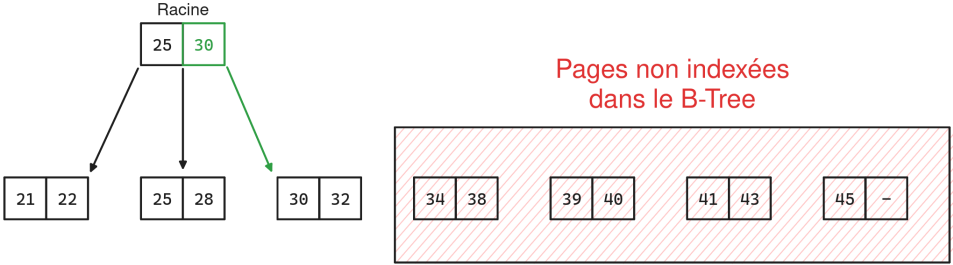
Arbre B + : Bulkloading



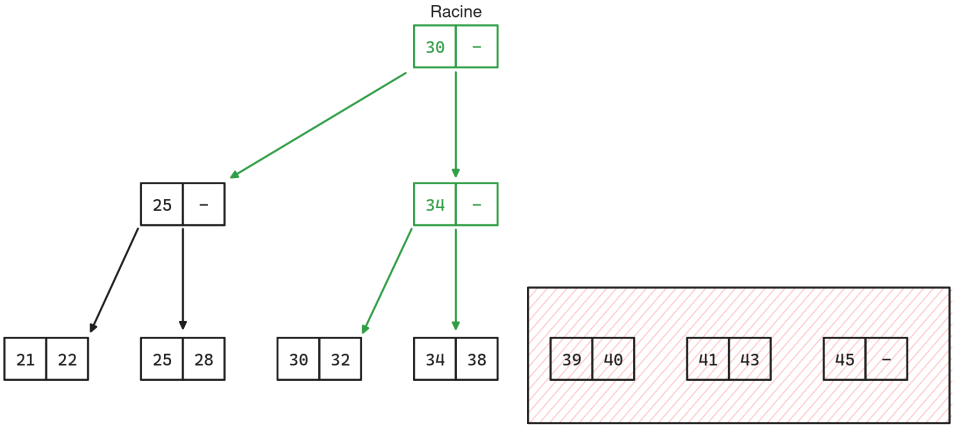
Arbre B + : Bulkloading



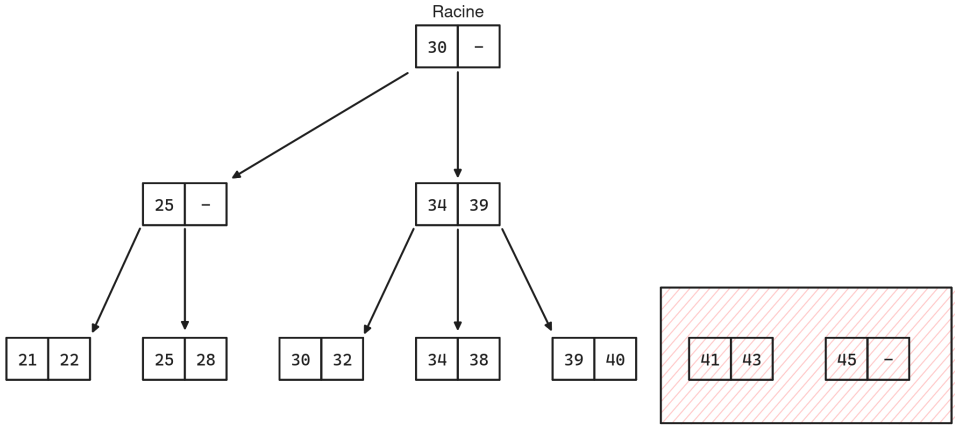
Arbre B + : Bulkloading



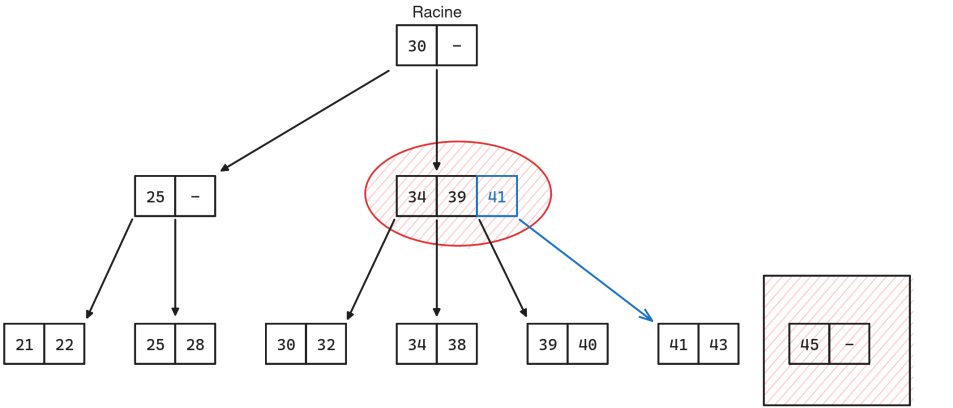
Arbre B + : Bulkloading



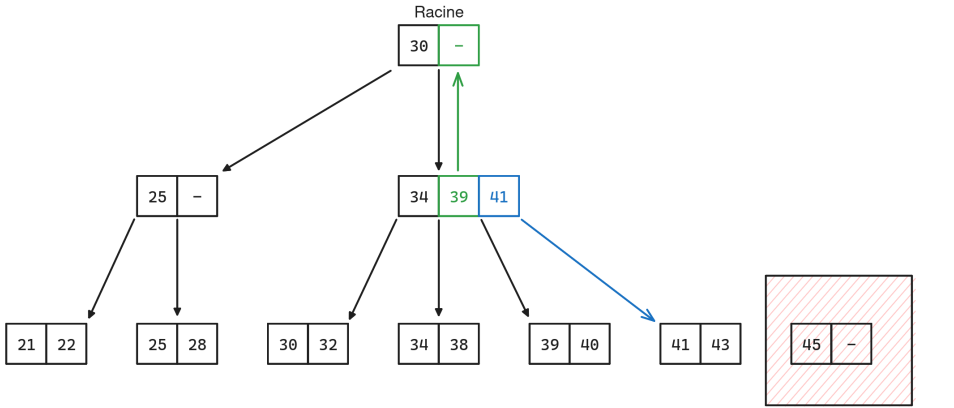
Arbre B + : Bulkloading



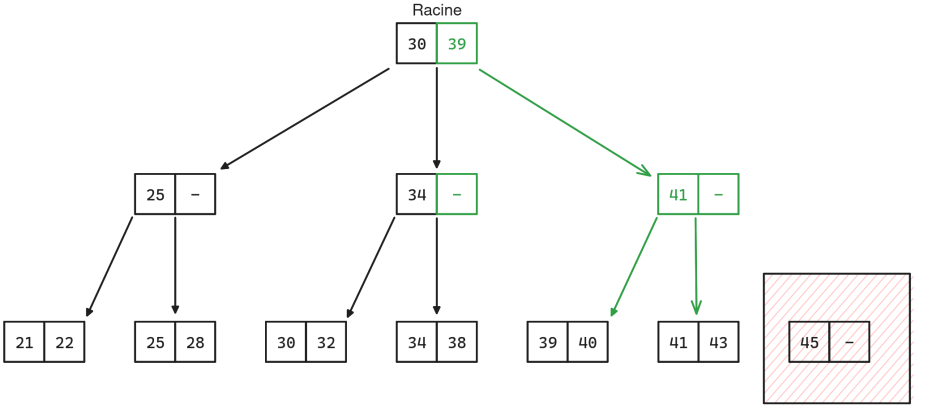
Arbre B + : Bulkloading



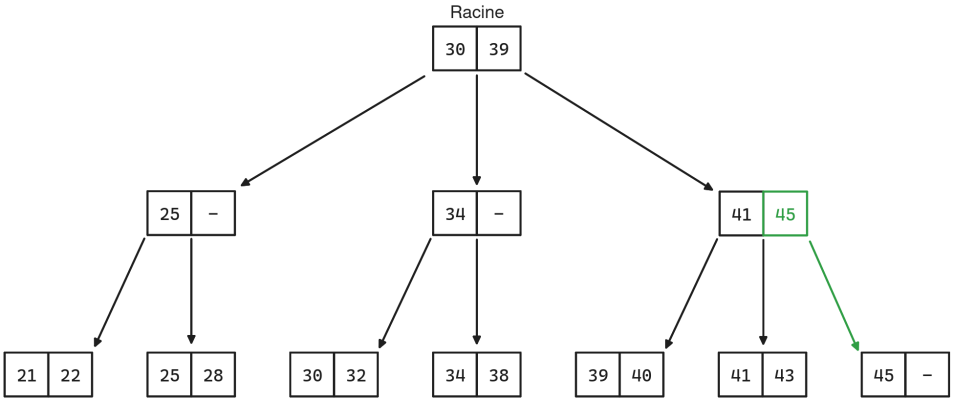
Arbre B + : Bulkloading



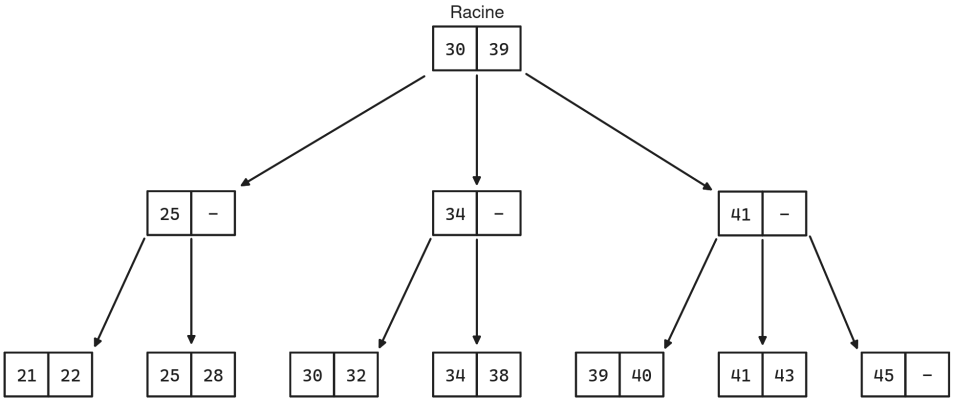
Arbre B + : Bulkloading



Arbre B + : Bulkloading



Arbre B + : Bulkloading



Arbre B + : Bulkloading

Avantages :

- Feuilles jamais modifiées $\rightarrow$ plus rapide que non trié
- Connaissances des éléments qui seront modifiés à l'avance (optimisation possible) !!!
- Conserver en mémoire les noeuds de "droites" !
- On ouvre qu'une seule fois chaque page (si cache suffisant $\approx$ la hauteur de l'arbre)

Les Arbres B+ en pratique

En pratique :

- Degré entre 100 et 1000 (fan-out)
- Hauteur entre 3 et 4 (height)
- remplissage $\approx$ 67% (fill-factor)

Les Arbres B+ en pratique

En pratique :

- Degré entre 100 et 1000 (fan-out)
- Hauteur entre 3 et 4 (height)
- remplissage $\approx$ 67% (fill-factor)

Noeuds interne Vs Feuilles

- **Niveau 1 (racine)** 1 page $\rightarrow$ 8KB (Postgres)

Facile de garder en mémoire les pages intermédiaires !!!

Les Arbres B+ en pratique

En pratique :

- Degré entre 100 et 1000 (fan-out)
- Hauteur entre 3 et 4 (height)
- remplissage $\approx$ 67% (fill-factor)

Noeuds interne Vs Feuilles

- **Niveau 1 (racine)** 1 page $\rightarrow$ 8KB (Postgres)
- **Niveau 2** $\approx$ 500 pages $\rightarrow$ 8MB

Facile de garder en mémoire les pages intermédiaires !!!

Les Arbres B+ en pratique

En pratique :

- Degré entre 100 et 1000 (fan-out)
- Hauteur entre 3 et 4 (height)
- remplissage $\approx$ 67% (fill-factor)

Noeuds interne Vs Feuilles

- **Niveau 1 (racine)** 1 page $\rightarrow$ 8KB (Postgres)
- **Niveau 2** $\approx$ 500 pages $\rightarrow$ 8MB
- **Niveau 3** $\approx 500^2$ pages $\rightarrow$ 2GB

Les Arbres B+ en pratique

En pratique :

- Degré entre 100 et 1000 (fan-out)
- Hauteur entre 3 et 4 (height)
- remplissage $\approx$ 67% (fill-factor)

Noeuds interne Vs Feuilles

- **Niveau 1 (racine)** 1 page $\rightarrow$ 8KB (Postgres)
- **Niveau 2** $\approx$ 500 pages $\rightarrow$ 8MB
- **Niveau 3** $\approx 500^2$ pages $\rightarrow$ 2GB

On peut garder en mémoire les noeuds internes !!!

Les Arbres B+ en pratique

En pratique :

- Degré entre 100 et 1000 (fan-out)
- Hauteur entre 3 et 4 (height)
- remplissage $\approx$ 67% (fill-factor)

Noeuds interne Vs Feuilles

- **Niveau 1** 1 page $\rightarrow$ 8KB (Postgres)
- **Niveau 2** $\approx$ 500 pages $\rightarrow$ 8MB
- **Niveau 3** $\approx 500^2$ pages $\rightarrow$ 2GB

On peut garder en mémoire les noeuds internes !!!

Optimisation

- Compression des noeuds (avoir des noeuds contenant plus de valeurs)
- Plusieurs pages pour un noeud (organisées séquentiellements sur le stockage)

Lien

- https://colab.research.google.com/drive/1EYxLfDPXPMmme-lTIA2Kf_3bDr7PDrM3?usp=sharing