Bases de données

Indexation

Nadime Francis

Université Gustave Eiffel
LIGM - 4B130 Copernic
nadime.francis@univ-eiffel.fr

Recherche d'un enregistrement dans un fichier

Objectif : évaluer efficacement la requête suivante :

```
SELECT * FROM maTable
WHERE attribut_1 = valeur_1 AND ... AND attribut_n = valeur_n;
```

Recherche d'un enregistrement dans un fichier

Objectif : évaluer efficacement la requête suivante :

```
SELECT * FROM maTable
WHERE attribut_1 = valeur_1 AND ... AND attribut_n = valeur_n;
```

Rappel : accès DRAM : 120 ns, accès disque magnétique : 1-10 ms

Recherche d'un enregistrement dans un fichier

Objectif : évaluer efficacement la requête suivante :

```
SELECT * FROM maTable
WHERE attribut_1 = valeur_1 AND ... AND attribut_n = valeur_n;
```

Rappel : accès DRAM : 120 ns, accès disque magnétique : 1-10 ms

Complexité évaluée en nombre d'accès au disque :

- Opération élémentaire : un transfert depuis ou vers le disque
- Toutes les opérations en mémoire principale sont négligeables

Recherche d'un enregistrement dans un fichier

Objectif : évaluer efficacement la requête suivante :

```
SELECT * FROM maTable
WHERE attribut_1 = valeur_1 AND ... AND attribut_n = valeur_n;
```

Rappel : accès DRAM : 120 ns, accès disque magnétique : 1-10 ms

Complexité évaluée en nombre d'accès au disque :

- Opération élémentaire : un transfert depuis ou vers le disque
- Toutes les opérations en mémoire principale sont négligeables

Approche naïve : parcours de tous les blocs de la table

Coût total : $\lceil nr/B \rceil$

- n : nombre de lignes de la table
- r : taille physique d'une ligne
- B : taille physique d'un bloc

Recherche d'un enregistrement dans un fichier

Objectif : évaluer efficacement la requête suivante :

```
SELECT * FROM maTable
WHERE attribut_1 = valeur_1 AND ... AND attribut_n = valeur_n;
```

Rappel : accès DRAM : 120 ns, accès disque magnétique : 1-10 ms

Complexité évaluée en nombre d'accès au disque :

- Opération élémentaire : un transfert depuis ou vers le disque
- Toutes les opérations en mémoire principale sont négligeables

Approche naïve : parcours de tous les blocs de la table

Coût total : $\lceil nr/B \rceil$

- n : nombre de lignes de la table
- r : taille physique d'une ligne
- B : taille physique d'un bloc

Question : comment faire mieux ?

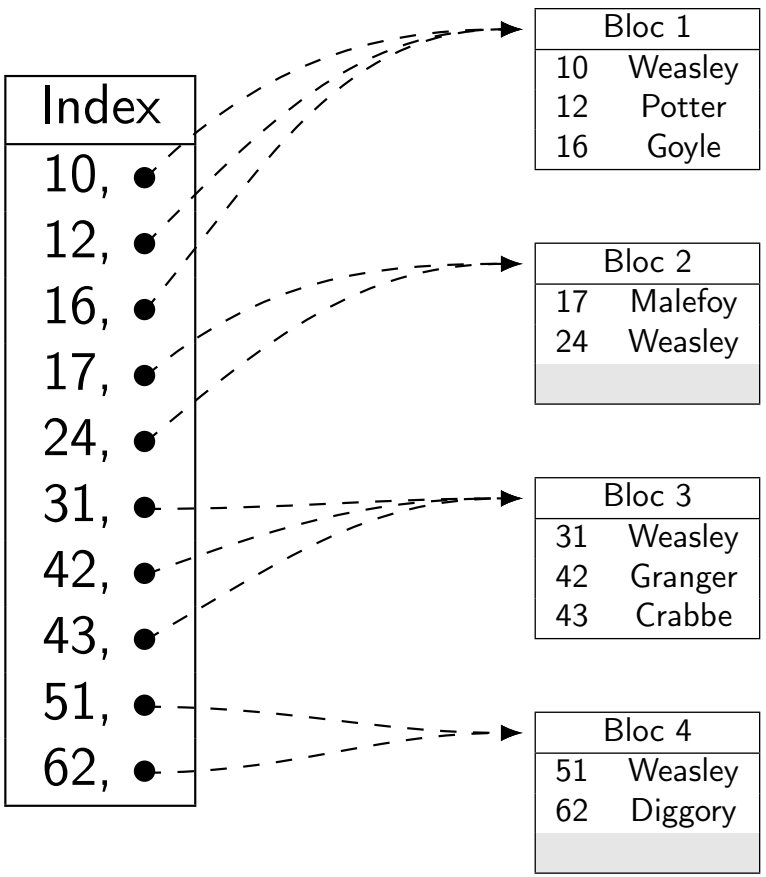
Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

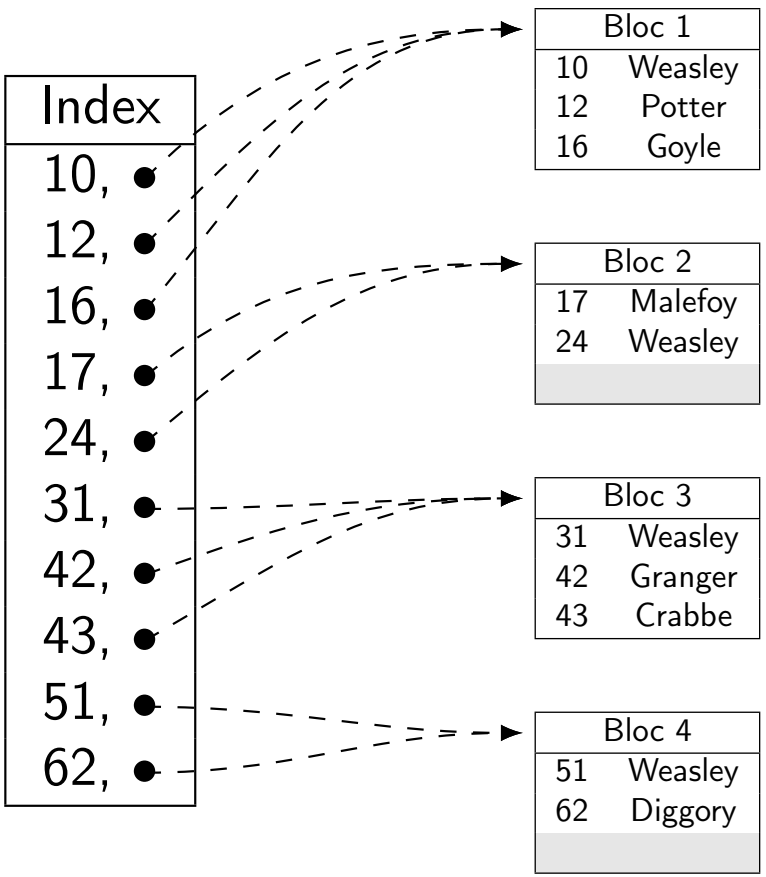
Un premier exemple simple :



Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

Un premier exemple simple :

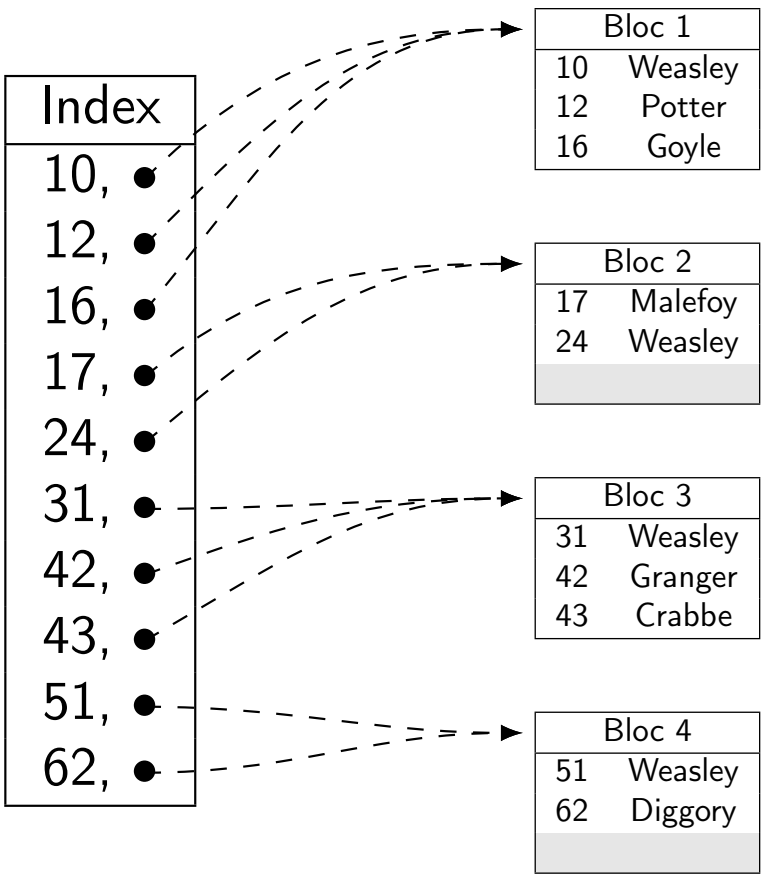


- Chaque entrée de l'index pointe sur le bloc contenant l'enregistrement correspondant

Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

Un premier exemple simple :

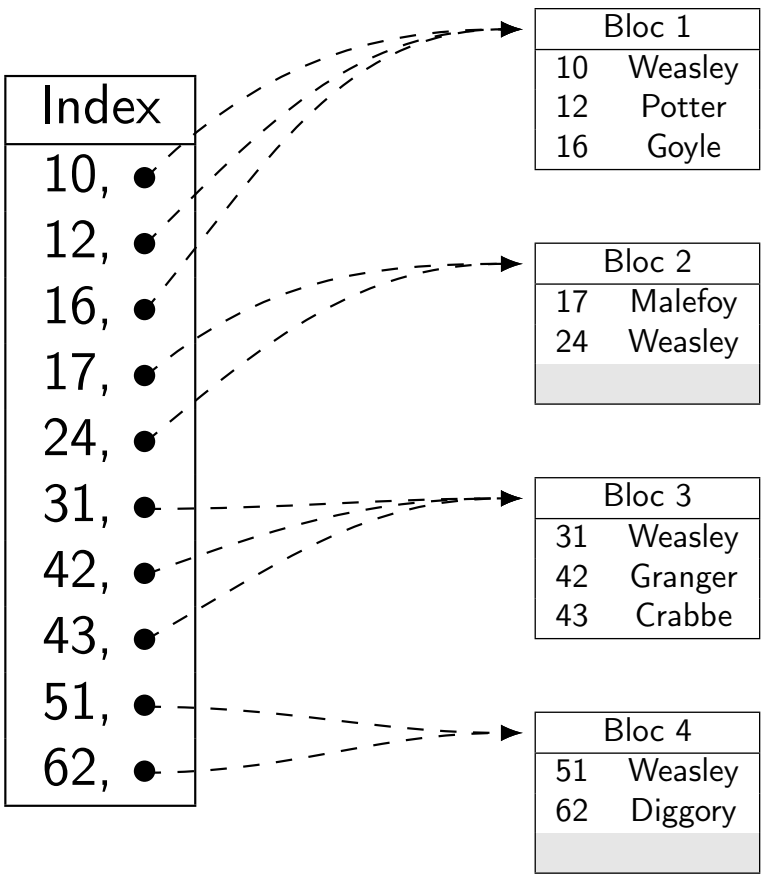


- Chaque entrée de l'index pointe sur le bloc contenant l'enregistrement correspondant
- Pour chercher un étudiant par son identifiant, seul l'index, beaucoup plus petit, est parcouru

Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

Un premier exemple simple :

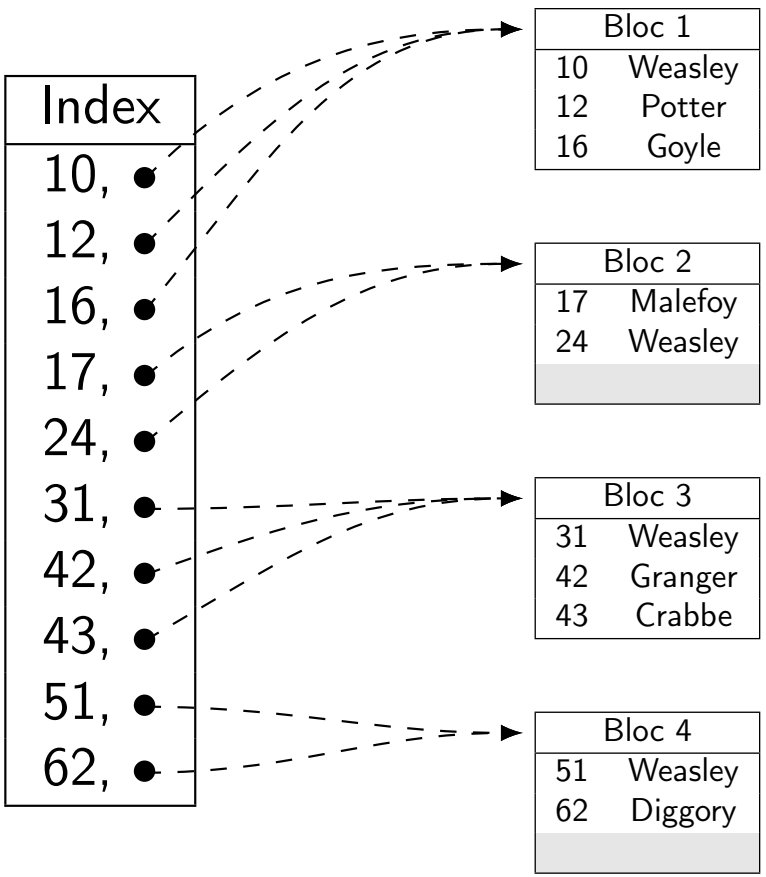


- Chaque entrée de l'index pointe sur le bloc contenant l'enregistrement correspondant
- Pour chercher un étudiant par son identifiant, seul l'index, beaucoup plus petit, est parcouru
- L'index n'est d'aucune aide pour chercher un étudiant par nom

Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

Un premier exemple simple :



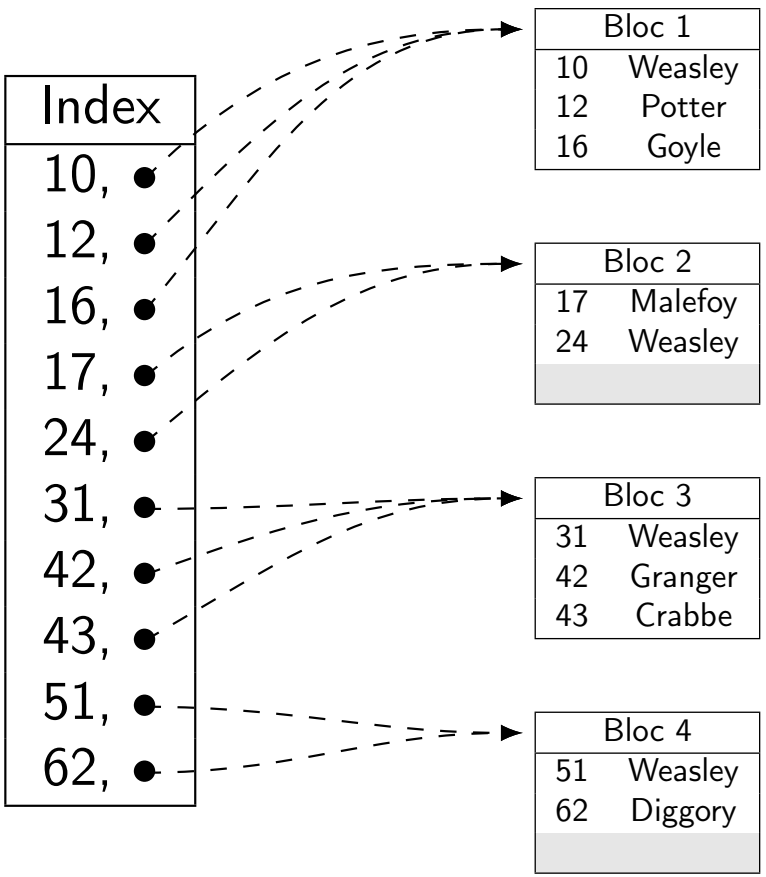
- Chaque entrée de l'index pointe sur le bloc contenant l'enregistrement correspondant
- Pour chercher un étudiant par son identifiant, seul l'index, beaucoup plus petit, est parcouru
- L'index n'est d'aucune aide pour chercher un étudiant par nom

Question : quel est alors le coût de la recherche ?

Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

Un premier exemple simple :



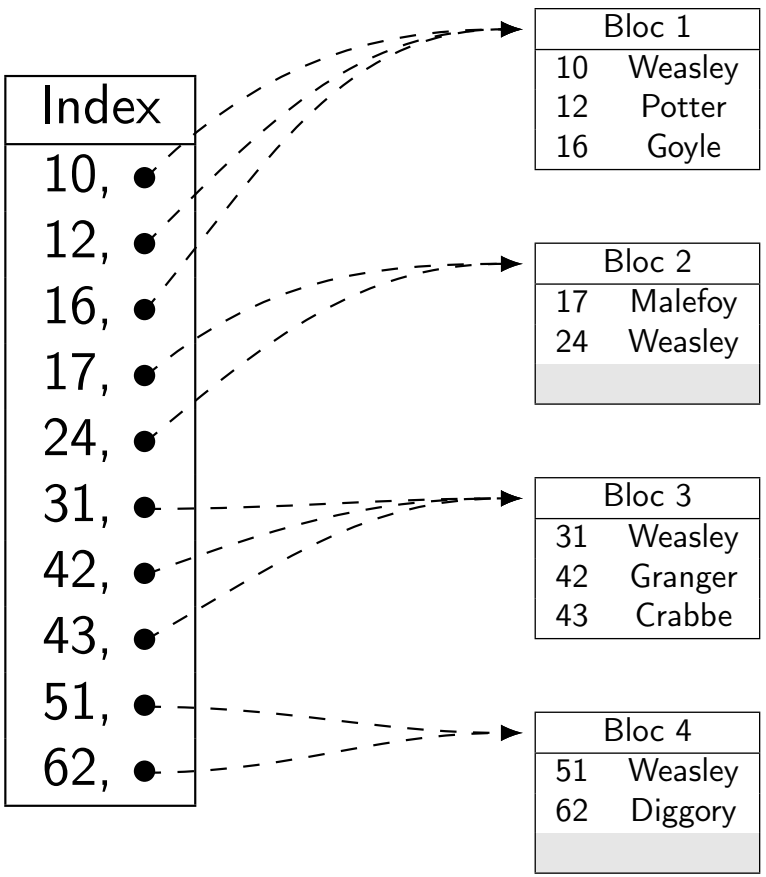
- Chaque entrée de l'index pointe sur le bloc contenant l'enregistrement correspondant
- Pour chercher un étudiant par son identifiant, seul l'index, beaucoup plus petit, est parcouru
- L'index n'est d'aucune aide pour chercher un étudiant par nom

Question : quel est alors le coût de la recherche ?
$\rightarrow$ dépend de l'implémentation de l'index

Index : principe général

Index : structure auxiliaire pour accélerer l'accès aux enregistrements d'une table par une clef de recherche (un ensemble d'attributs) choisie

Un premier exemple simple :



- Chaque entrée de l'index pointe sur le bloc contenant l'enregistrement correspondant
- Pour chercher un étudiant par son identifiant, seul l'index, beaucoup plus petit, est parcouru
- L'index n'est d'aucune aide pour chercher un étudiant par nom

Question : quel est alors le coût de la recherche ?
$\rightarrow$ dépend de l'implémentation de l'index

Au plus naïf : $\lceil ni/B \rceil$
i : taille d'une paire clef-pointeur

Index sous Postgres

- Index automatiquement créé sur la clef primaire de chaque table
  (Il s'agit de l'index de gestion d'un fichier séquentiel, voir chapitre stockage)
- Création ou suppression d'index supplémentaires :

  CREATE INDEX (nom) **ON** (nomTable) **USING** (méthode)(attributs);
  DROP INDEX (nom);

Index sous Postgres

- Index automatiquement créé sur la clef primaire de chaque table
  (Il s'agit de l'index de gestion d'un fichier séquentiel, voir chapitre stockage)
- Création ou suppression d'index supplémentaires :

  CREATE INDEX (nom) **ON** (nomTable) **USING** (méthode)(attributs);
  DROP INDEX (nom);

Ex : comparer les performances des requêtes suivantes :

```
SELECT tel FROM client WHERE numcli = 278;
```

```
SELECT tel FROM client WHERE nom = 'Delacour' AND prenom = 'Fleur';
```

Quelle différence observe-t-on ? Pourquoi ?
Construire un index et tester à nouveau.

Index sous Postgres

- Index automatiquement créé sur la clef primaire de chaque table
  (Il s'agit de l'index de gestion d'un fichier séquentiel, voir chapitre stockage)
- Création ou suppression d'index supplémentaires :

  CREATE INDEX (nom) **ON** (nomTable) **USING** (méthode)(attributs);
  DROP INDEX (nom);

Ex : comparer les performances des requêtes suivantes :

```
SELECT tel FROM client WHERE numcli = 278;
```

```
SELECT tel FROM client WHERE nom = 'Delacour' AND prenom = 'Fleur';
```

Quelle différence observe-t-on ? Pourquoi ?
Construire un index et tester à nouveau.

Question : pourquoi ne pas construire des index sur toutes les colonnes ?

Fonctionnement des index

Types d'index

Index primaire dense ou non-dense

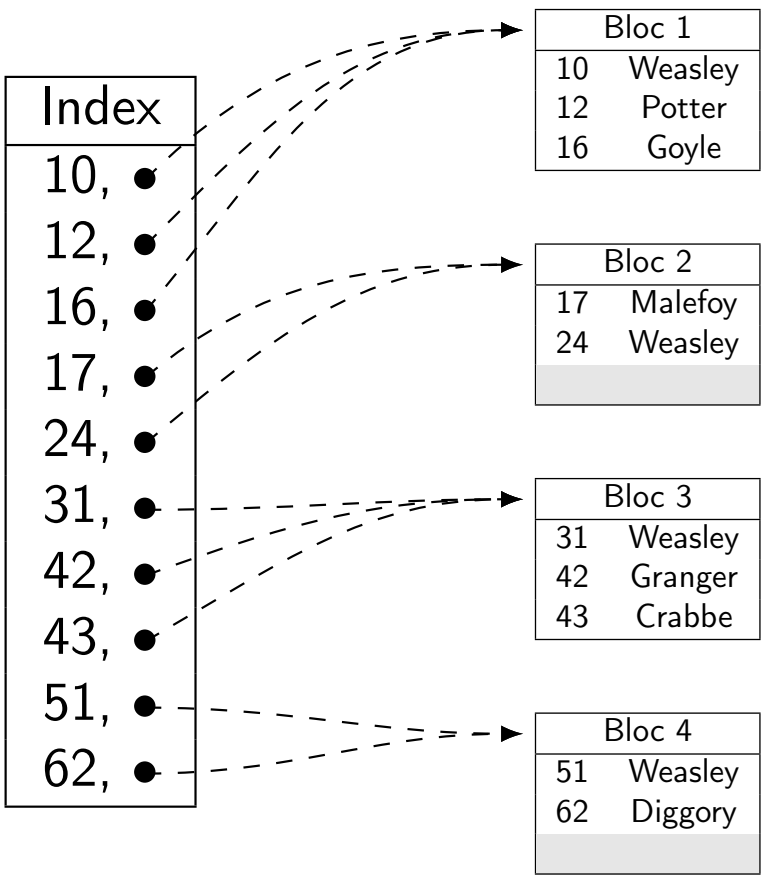
- Primaire : le fichier est trié par la clef de recherche de l'index
- Dense : une entrée de l'index par valeur possible de la clef
- Non-dense : seulement une entrée de l'index par bloc du fichier

Index secondaire (dense)

- Secondaire : le fichier n'est pas trié (tas de données par exemple) ou bien son critère de tri n'est pas la clef de recherche de l'index
- Un index secondaire est toujours dense !

Index primaire dense : recherche et mise-à-jour

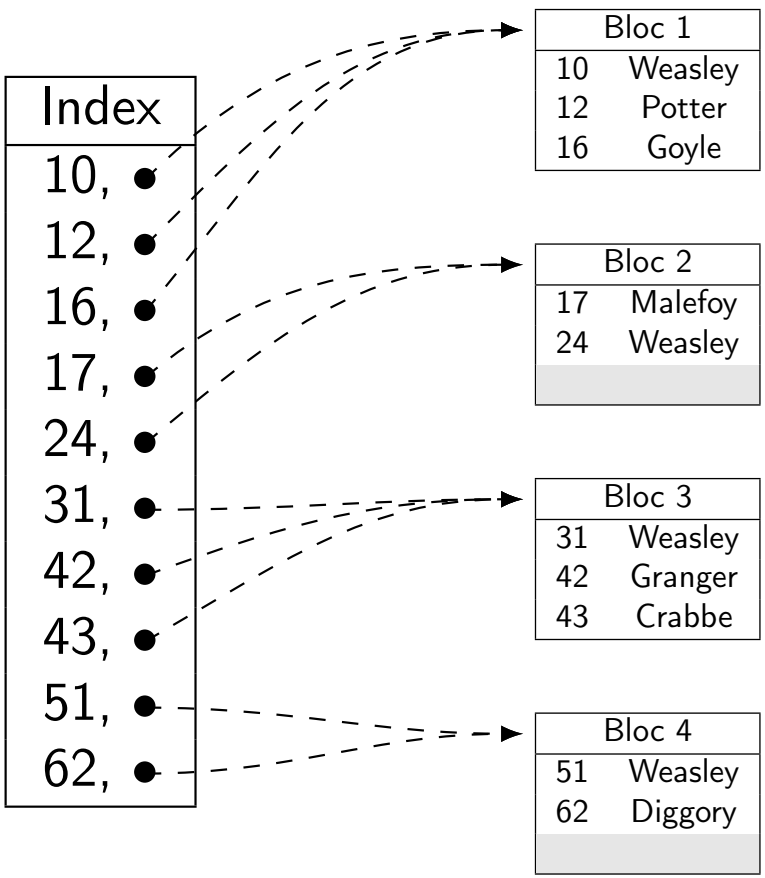
- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc



Ex :

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

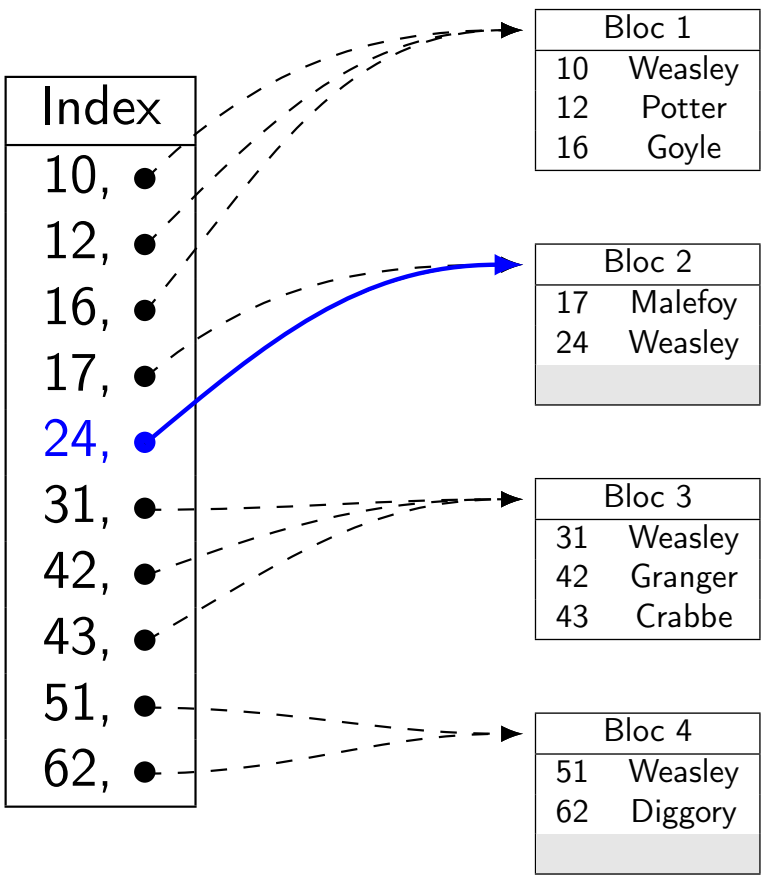


Ex :

- Recherche de 24

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

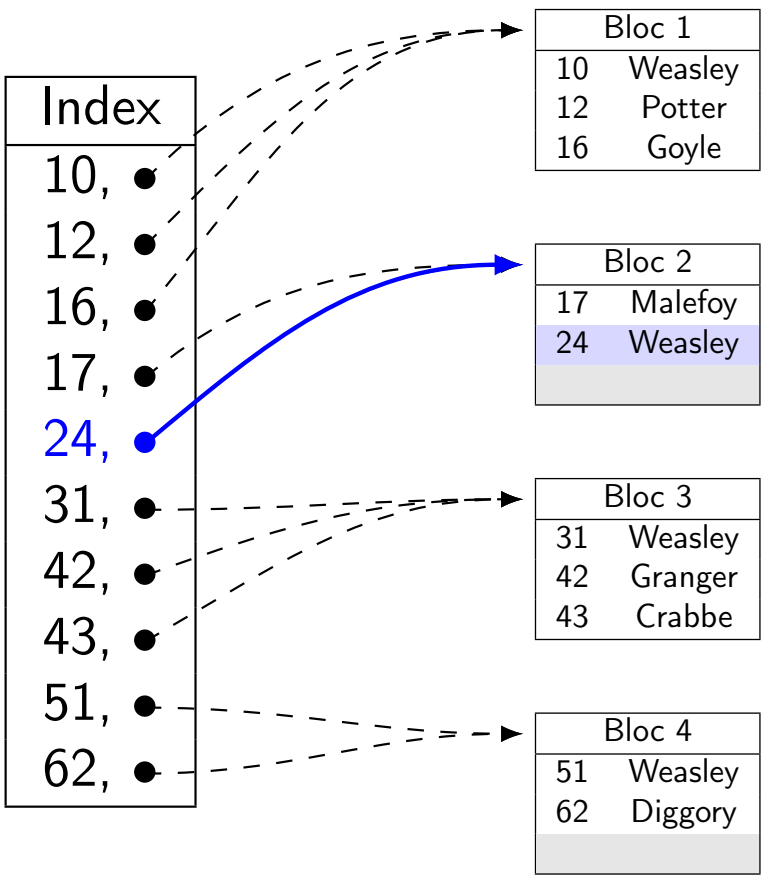


Ex :

- Recherche de 24

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

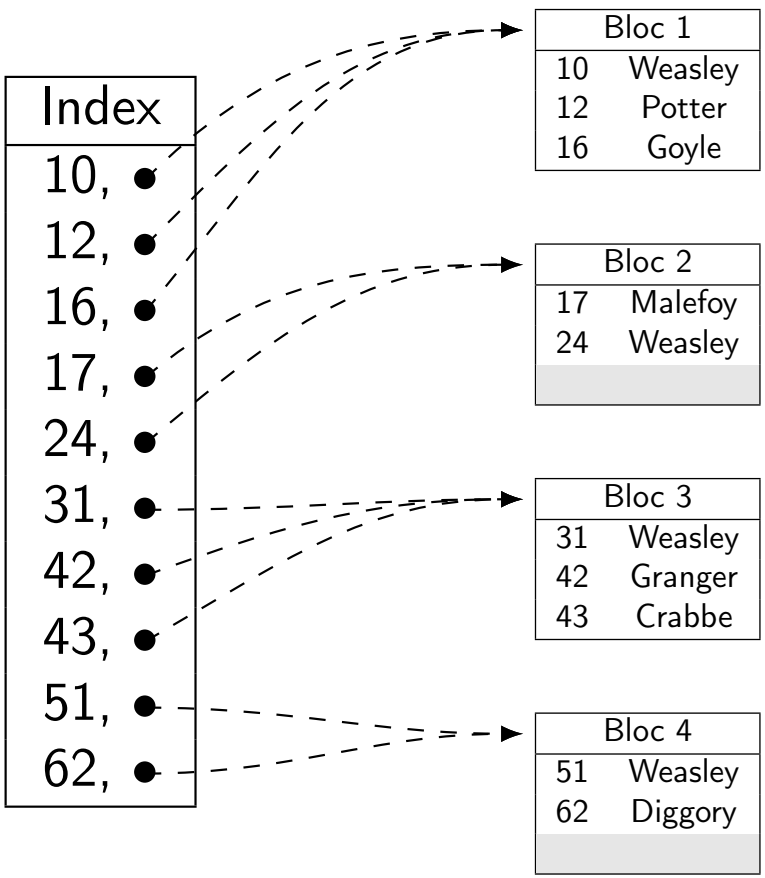


Ex :

- Recherche de 24

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

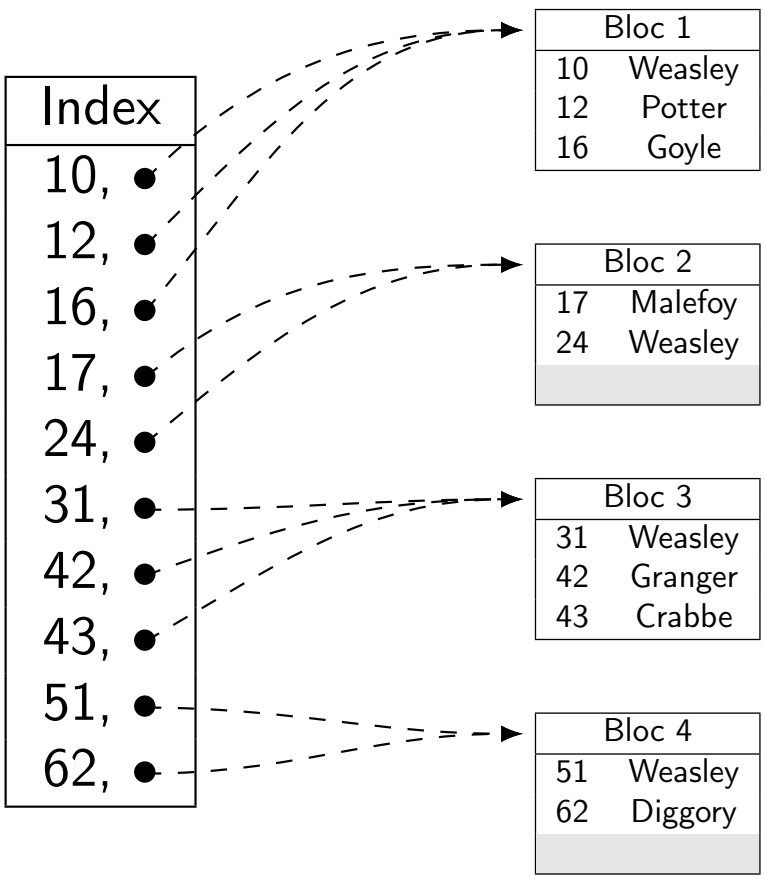


Ex :

- Recherche de 24
- Recherche de 27

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

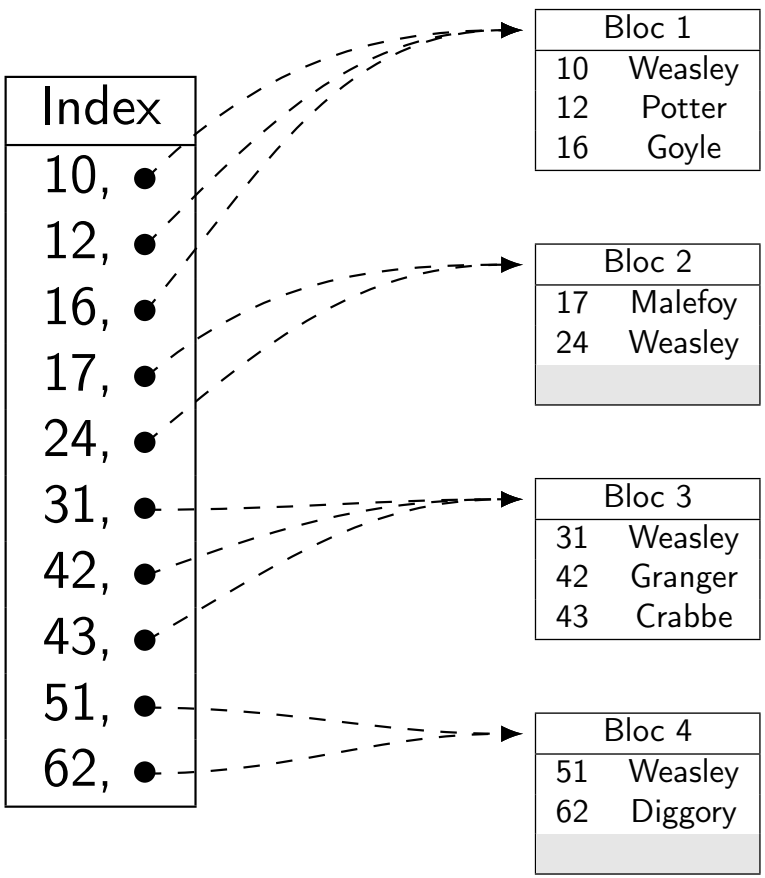


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

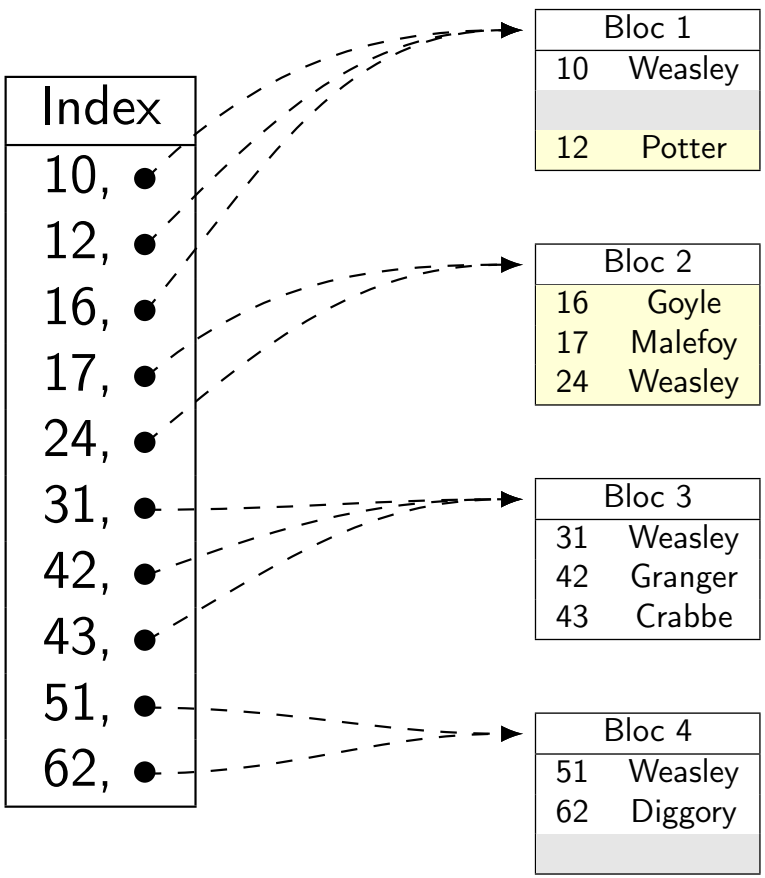


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

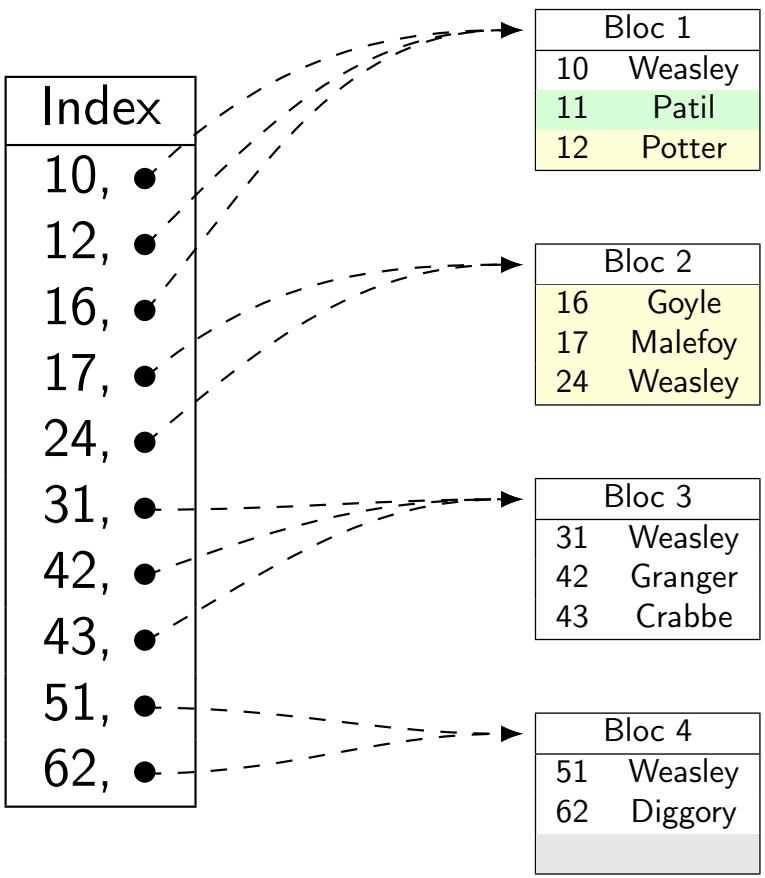


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

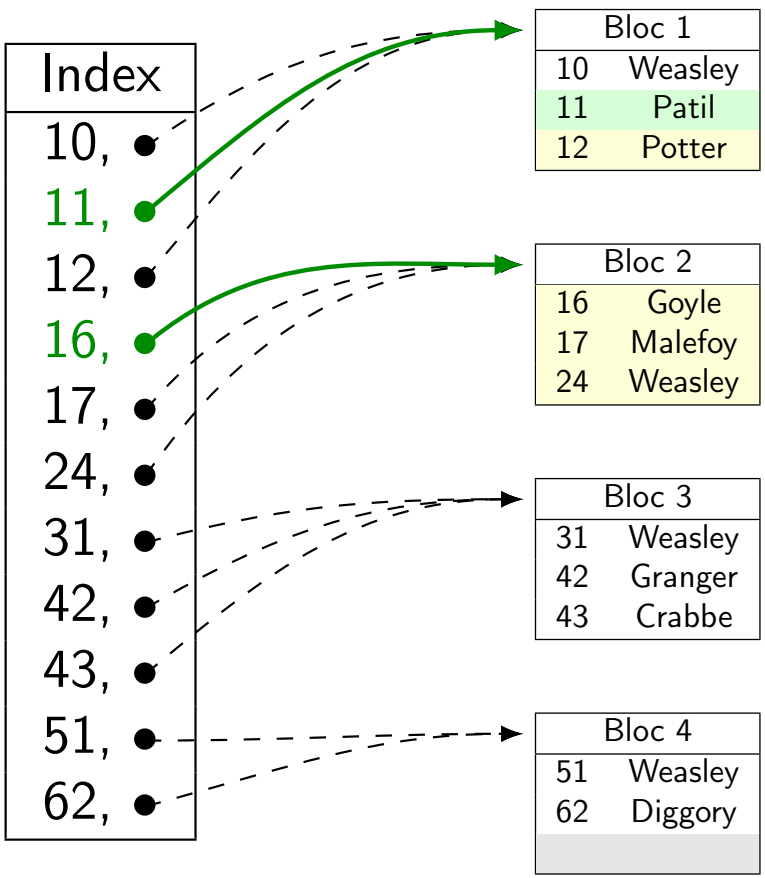


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

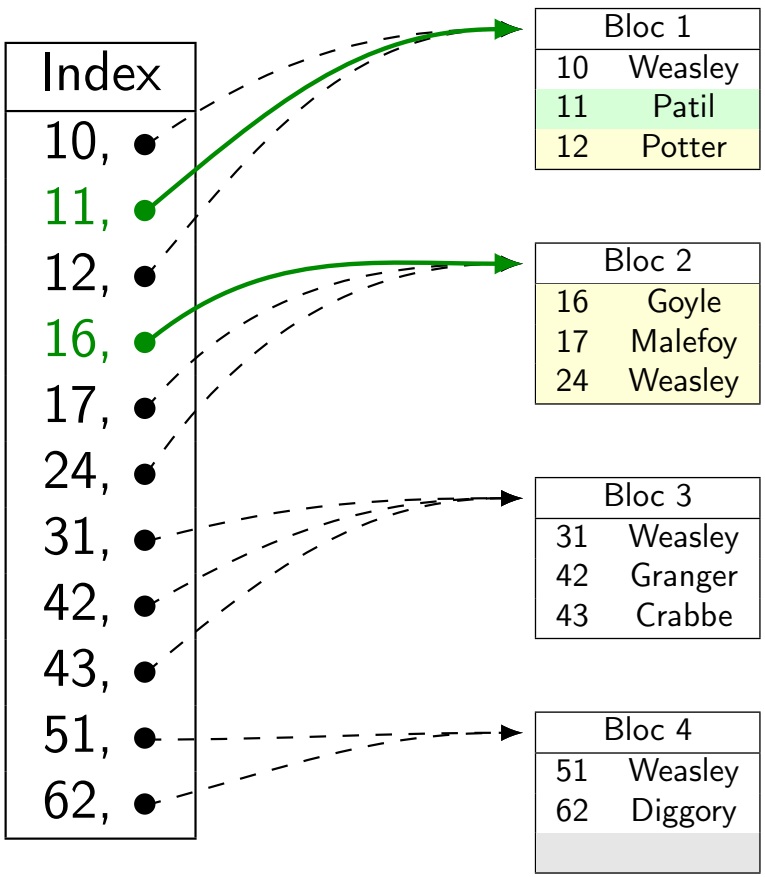


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

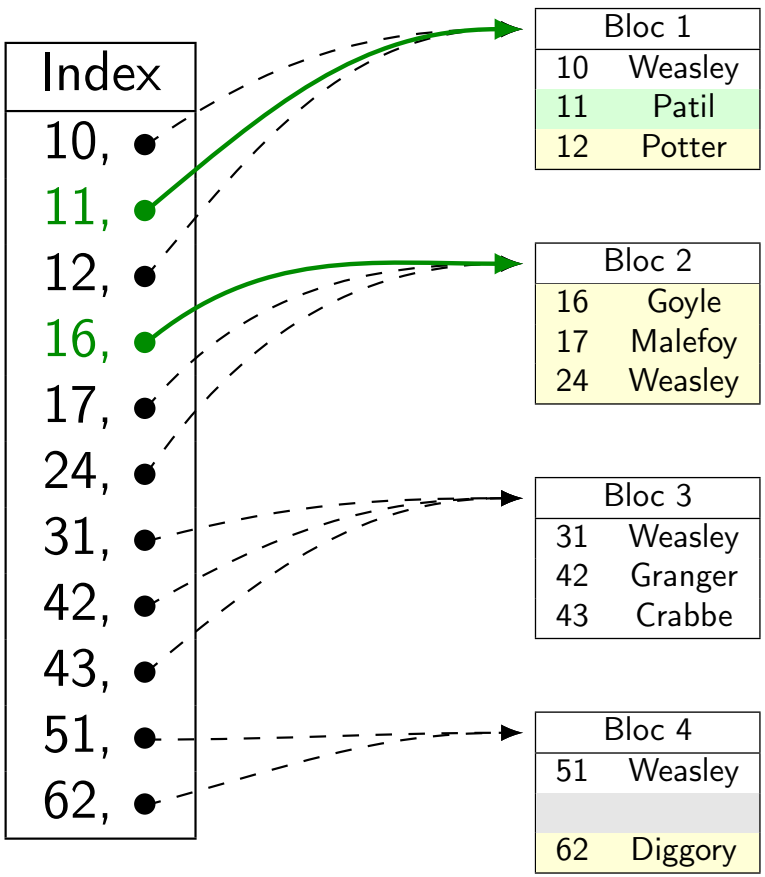


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

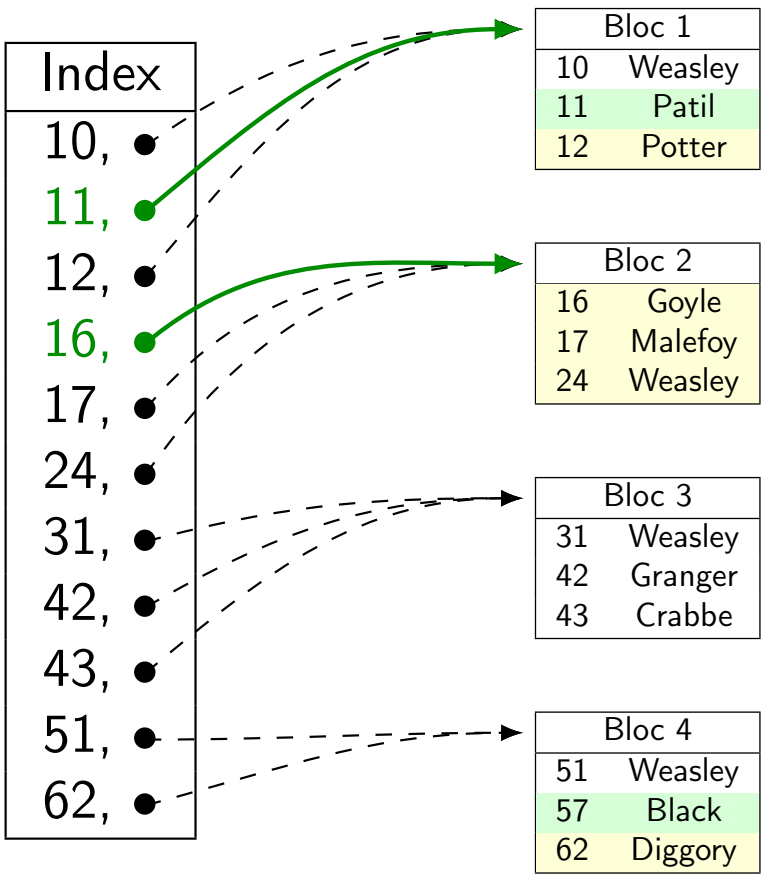


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

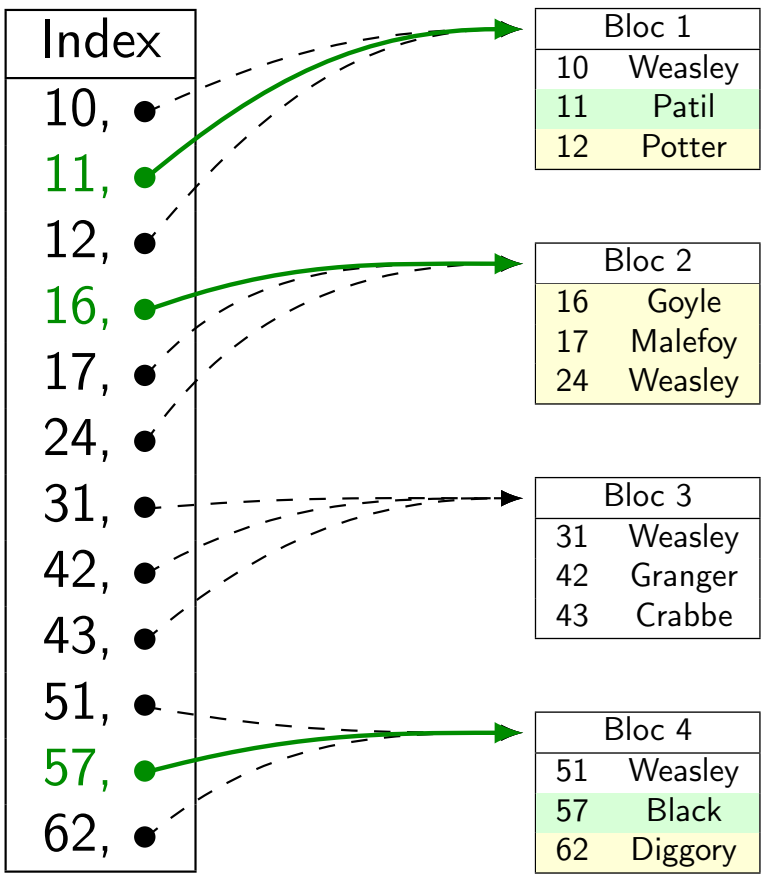


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)

Index primaire dense : recherche et mise-à-jour

- L'index fournit un pointeur par valeur possible de la clef de recherche
- Il suffit de suivre le pointeur pour trouver le bon bloc du fichier
- Doit être mis à jour à chaque insertion ou changement de bloc

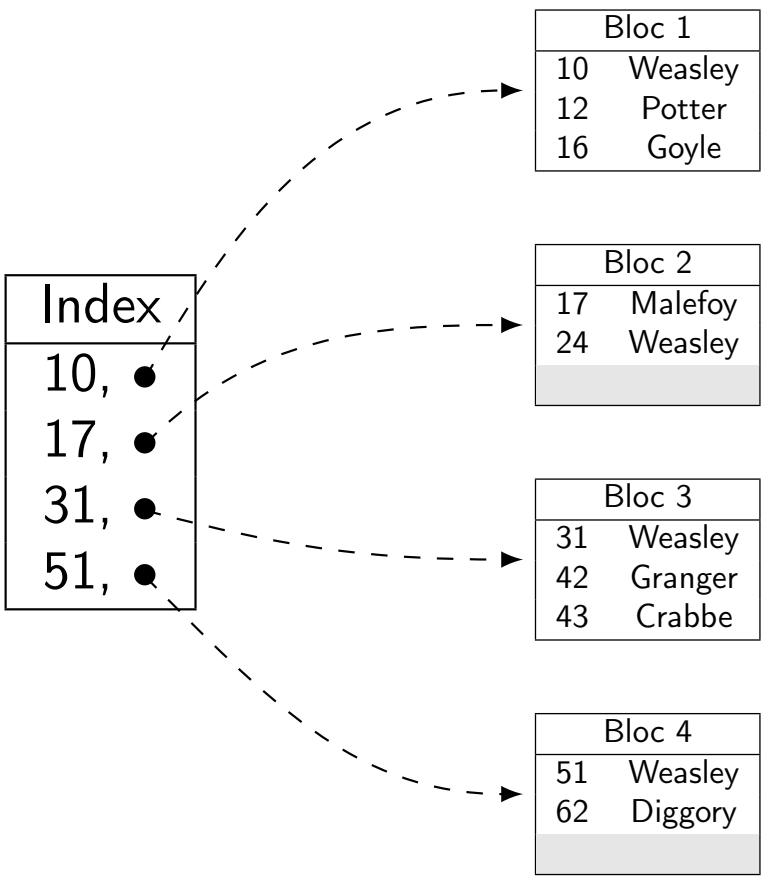


Ex :

- Recherche de 24
- Recherche de 27
  27 n'est pas dans l'index
  $\Rightarrow$ 27 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)

Index primaire non-dense : recherche et mise-à-jour

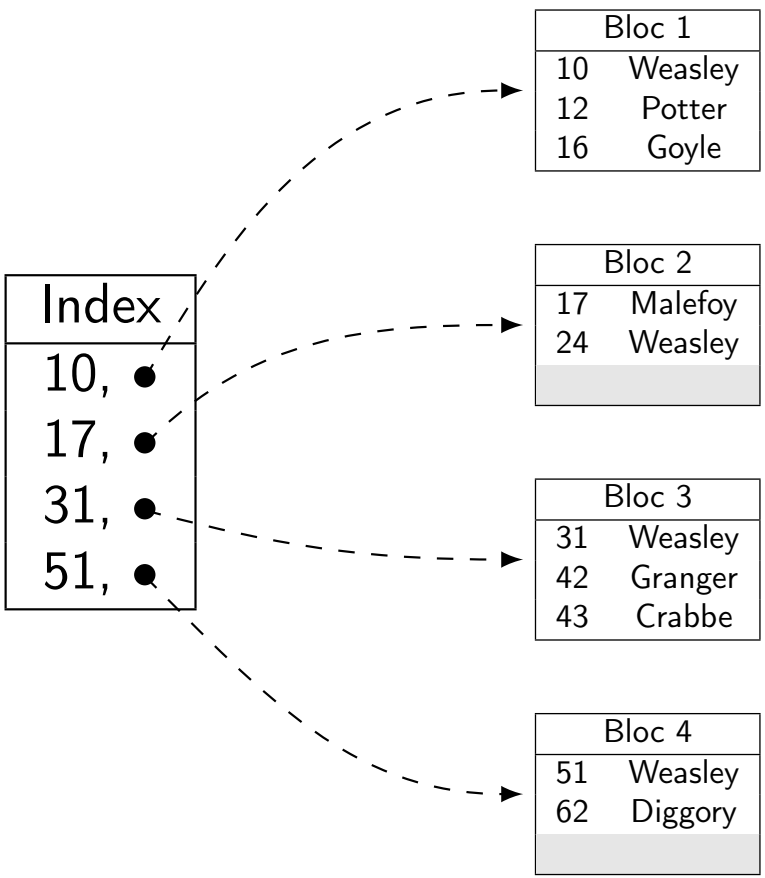
- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change



Ex :

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

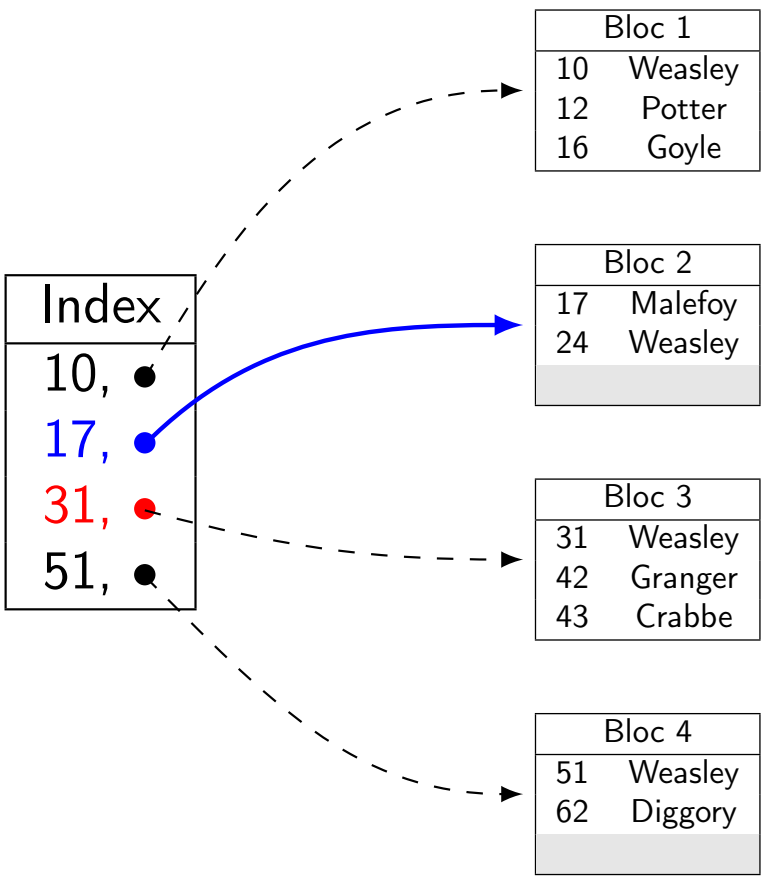


Ex :

- Recherche de 24

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change



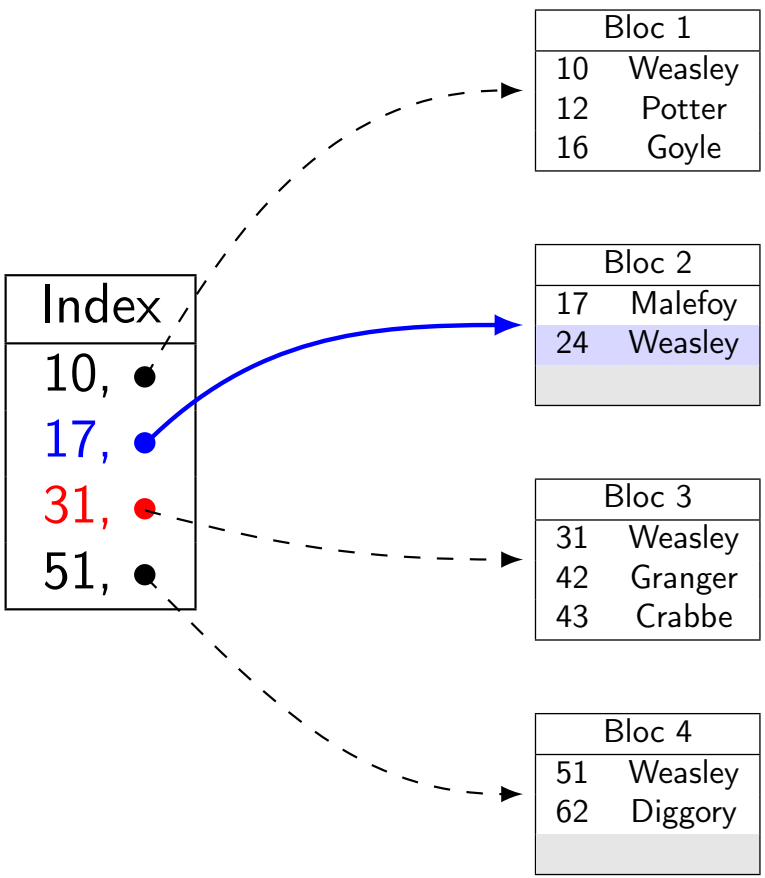
Ex :

- Recherche de 24

  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change



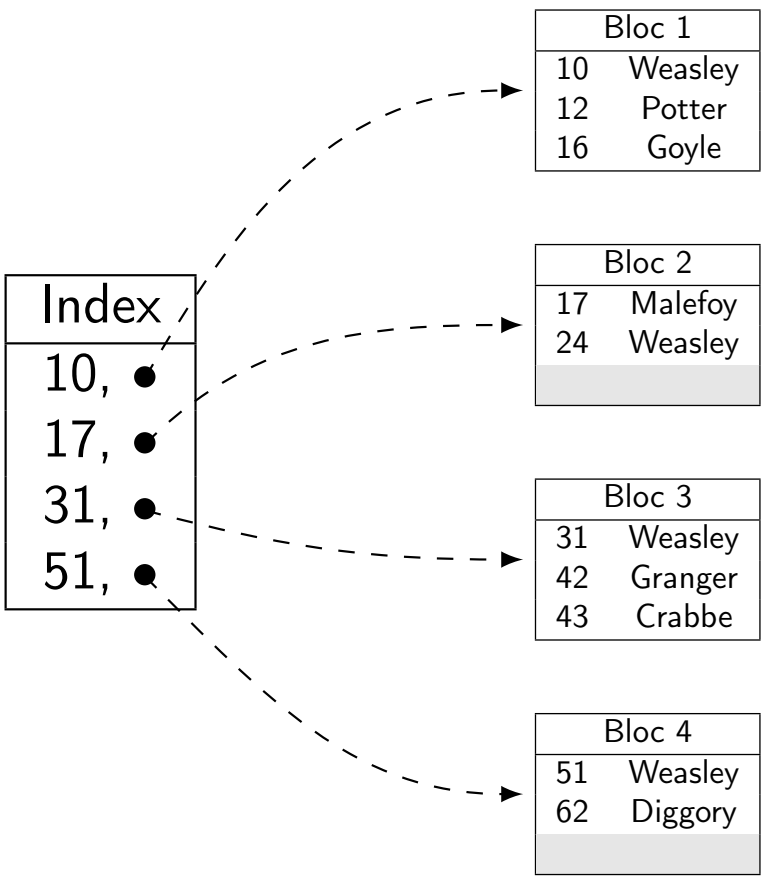
Ex :

- Recherche de 24

  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change



Ex :

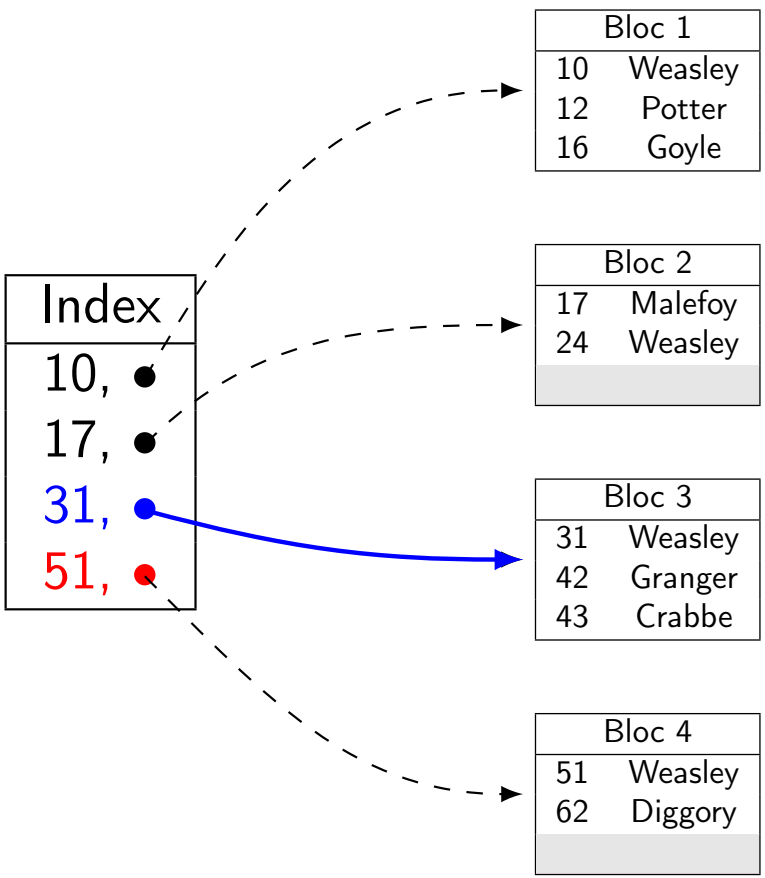
- Recherche de 24

  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2

- Recherche de 48

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

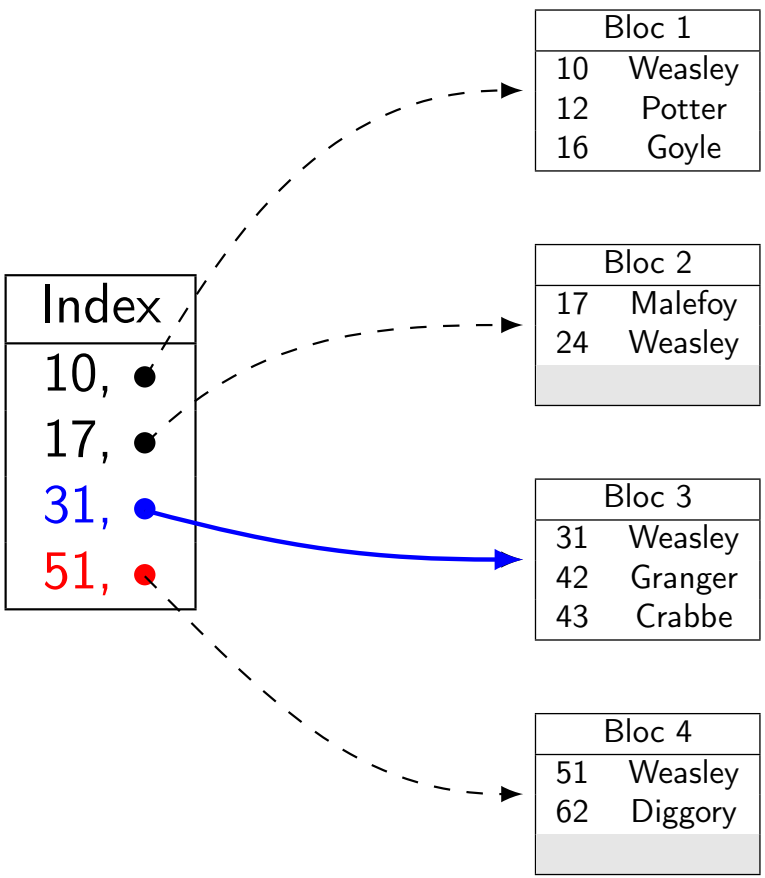


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change



Ex :

- Recherche de 24

  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2

- Recherche de 48

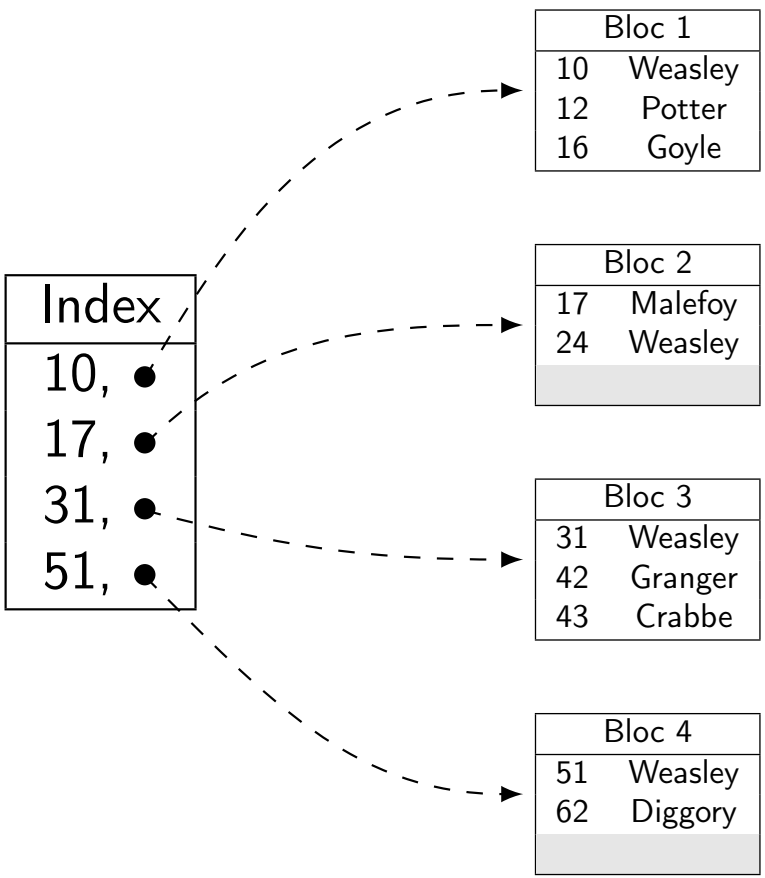
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3

  48 n'est pas trouvé dans le bloc

  $\Rightarrow$ 48 n'est pas dans la table

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

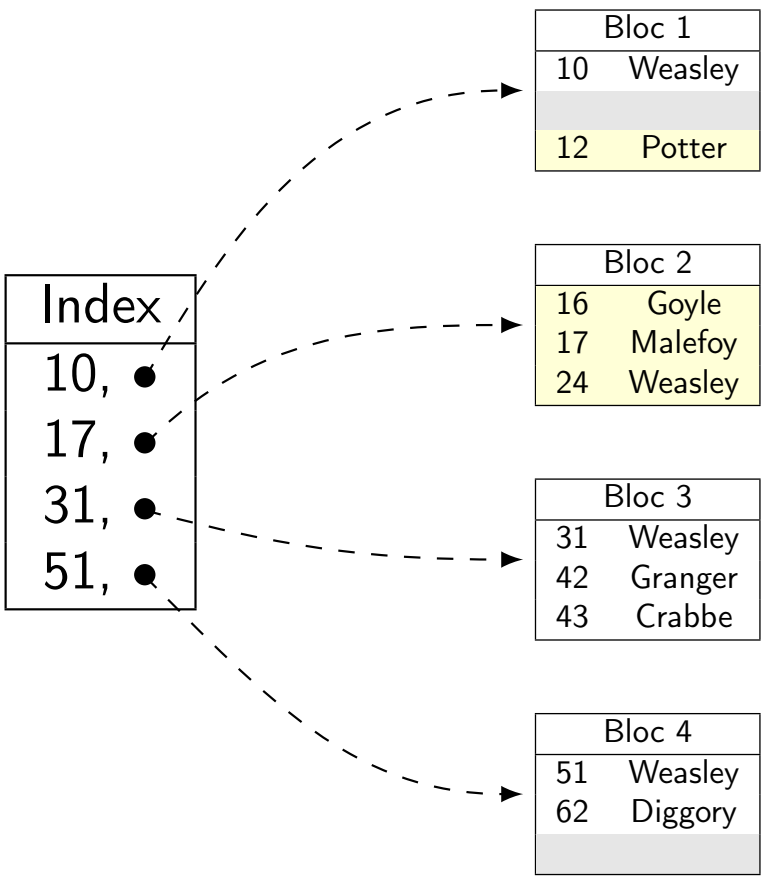


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

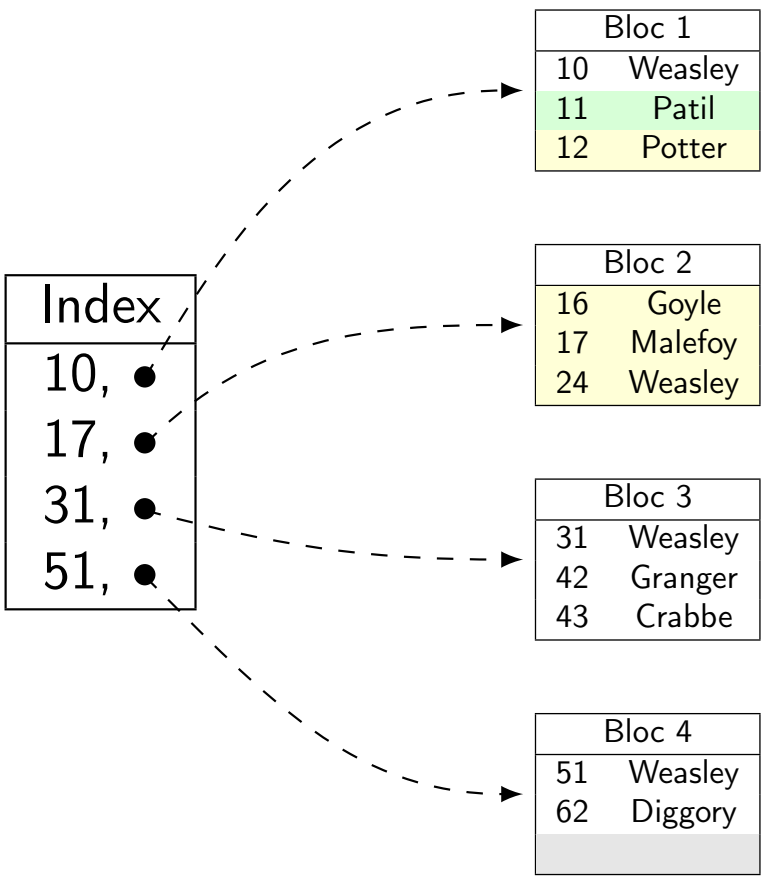


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

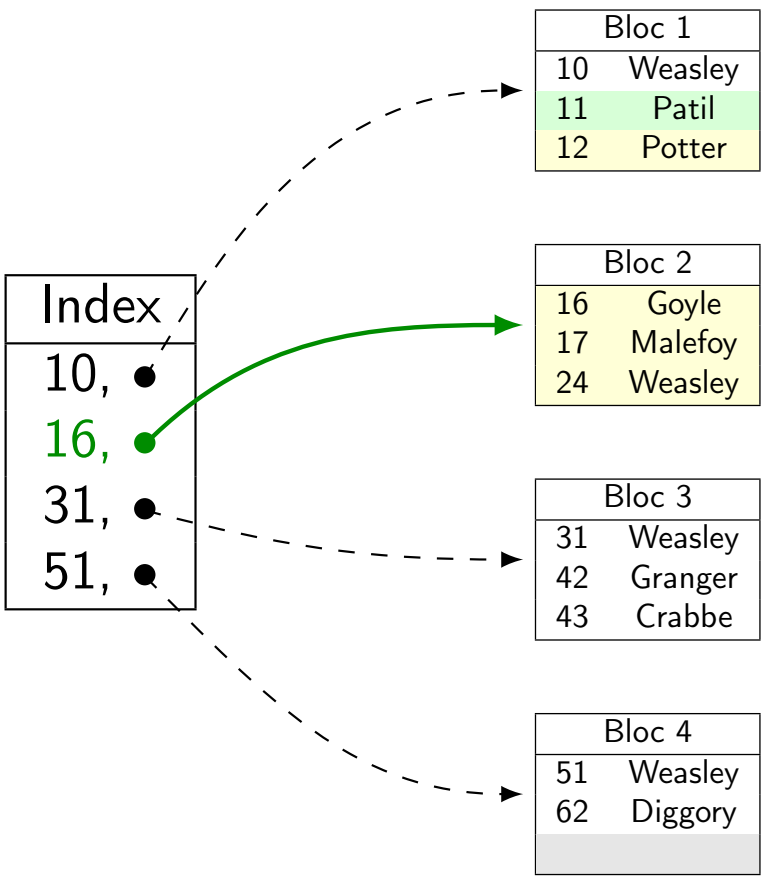


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

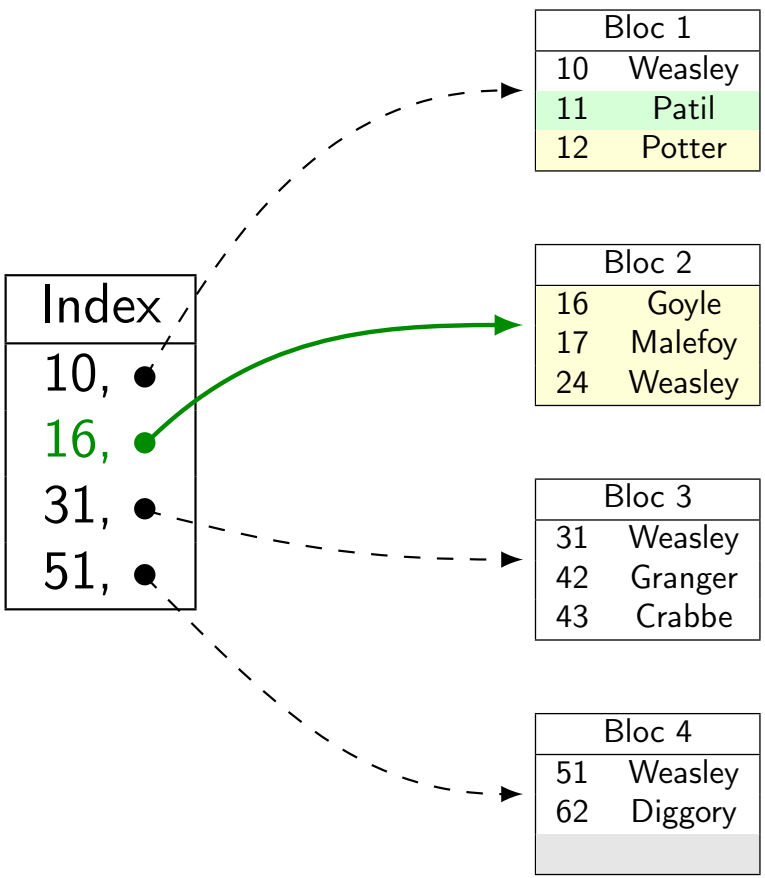


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

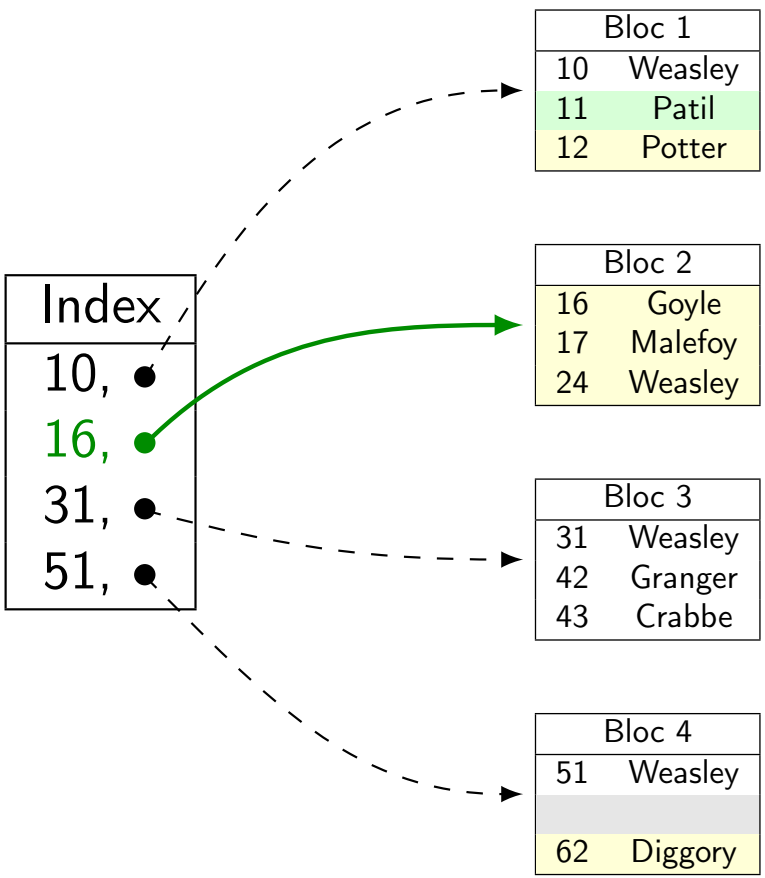


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

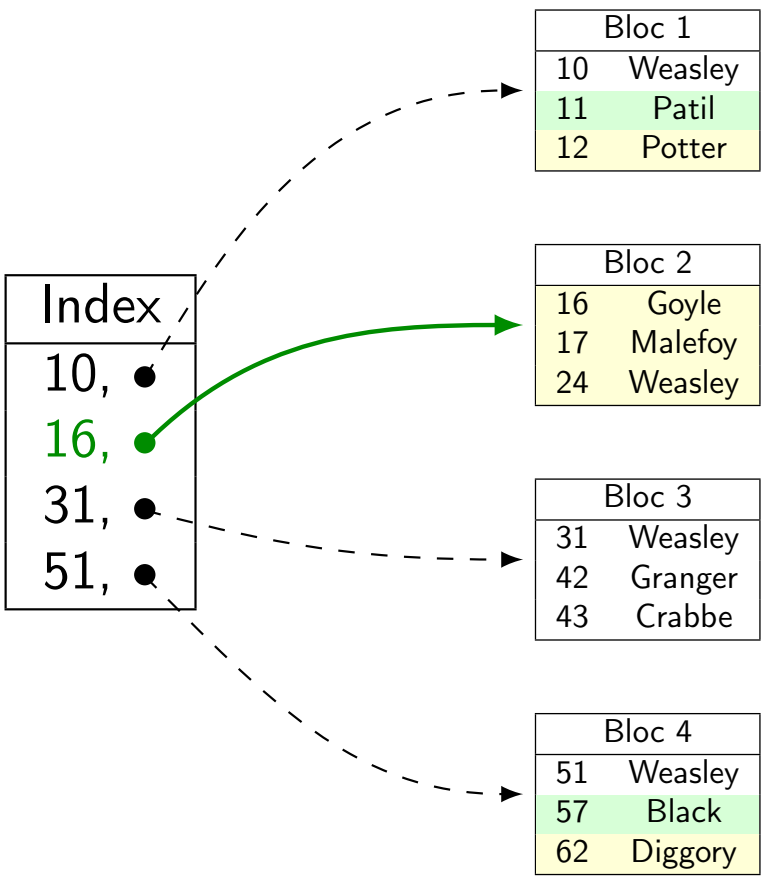


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change

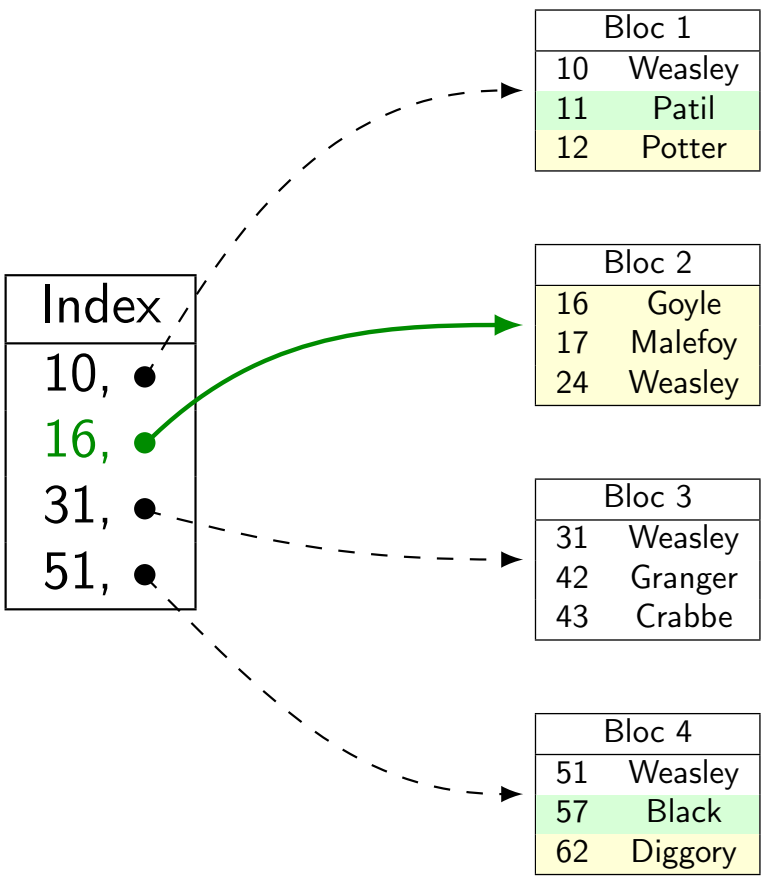


Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)

Index primaire non-dense : recherche et mise-à-jour

- L'index n'a qu'une entrée par bloc, associée à la première clef du bloc
- On déduit de l'index le bloc où peut se trouver la clef recherchée
- Doit être mis à jour quand la première clef d'un bloc change



Ex :

- Recherche de 24
  $17 \leq 24 < 31 \Rightarrow$ dans le bloc 2
- Recherche de 48
  $31 \leq 48 < 51 \Rightarrow$ dans le bloc 3
  48 n'est pas trouvé dans le bloc
  $\Rightarrow$ 48 n'est pas dans la table
- Insertion de (11, Patil)
- Insertion de (57, Black)
  Pas de changement de l'index

Index primaires : remarques

Dans la pratique, en général :

- L'index de gestion d'un fichier séquentiel est un index primaire non-dense, bien plus compact qu'un index dense.
- La clef de recherche est la clef primaire de la table.
  $\Rightarrow$ pas de problème de doublons dans l'index

Index primaires : remarques

Dans la pratique, en général :

- L'index de gestion d'un fichier séquentiel est un index primaire non-dense, bien plus compact qu'un index dense.
- La clef de recherche est la clef primaire de la table.
  $\Rightarrow$ pas de problème de doublons dans l'index

Sinon, il faut gérer les doublons !
Plusieurs solutions raisonnables, par ex :

- Cas dense : on garde la première occurrence puis on lit dans l'ordre
- Cas non-dense : même idée, mais attention à la recherche

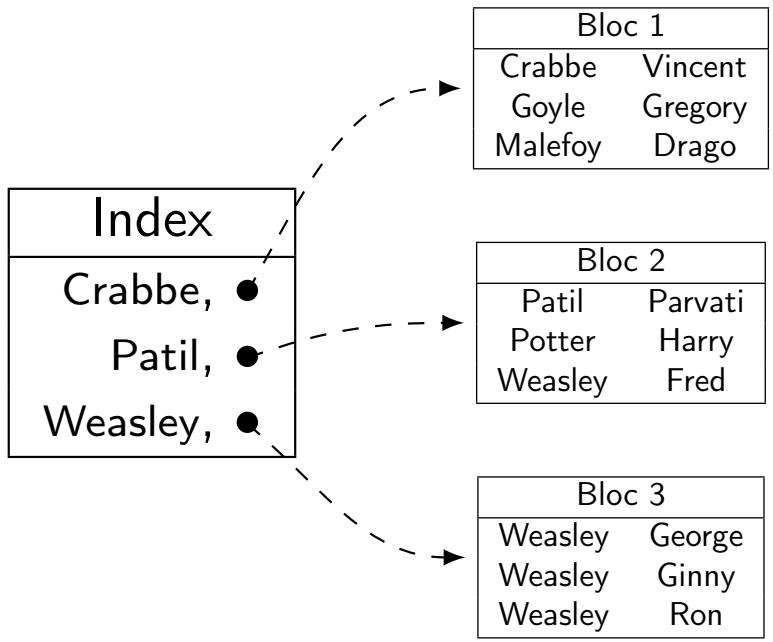
Index primaires : remarques

Dans la pratique, en général :

- L'index de gestion d'un fichier séquentiel est un index primaire non-dense, bien plus compact qu'un index dense.
- La clef de recherche est la clef primaire de la table.
  $\Rightarrow$ pas de problème de doublons dans l'index

Sinon, il faut gérer les doublons !
Plusieurs solutions raisonnables, par ex :

- Cas dense : on garde la première occurrence puis on lit dans l'ordre
- Cas non-dense : même idée, mais attention à la recherche



Index primaires : remarques

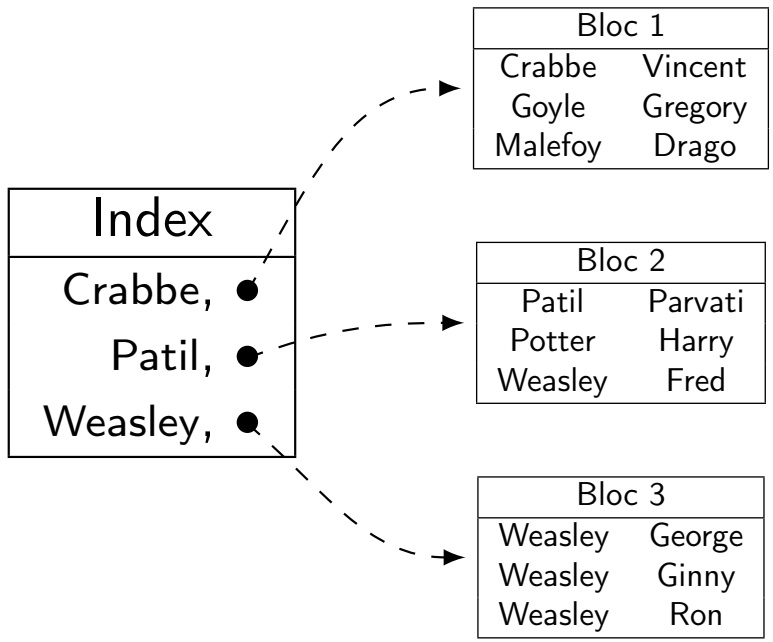
Dans la pratique, en général :

- L'index de gestion d'un fichier séquentiel est un index primaire non-dense, bien plus compact qu'un index dense.
- La clef de recherche est la clef primaire de la table.
  $\Rightarrow$ pas de problème de doublons dans l'index

Sinon, il faut gérer les doublons !
Plusieurs solutions raisonnables, par ex :

- Cas dense : on garde la première occurrence puis on lit dans l'ordre
- Cas non-dense : même idée, mais attention à la recherche

(Le premier Weasley est dans le bloc Patil !)



Index secondaire (dense)

- Pour accélérer l'accès aux enregistrements par un ou plusieurs attributs qui ne sont pas le critère de tri du fichier
  - Recherche par nom sur un fichier trié par numéro d'étudiant
  - Index sur un tas de données (qui n'est donc pas ordonné)

Index secondaire (dense)

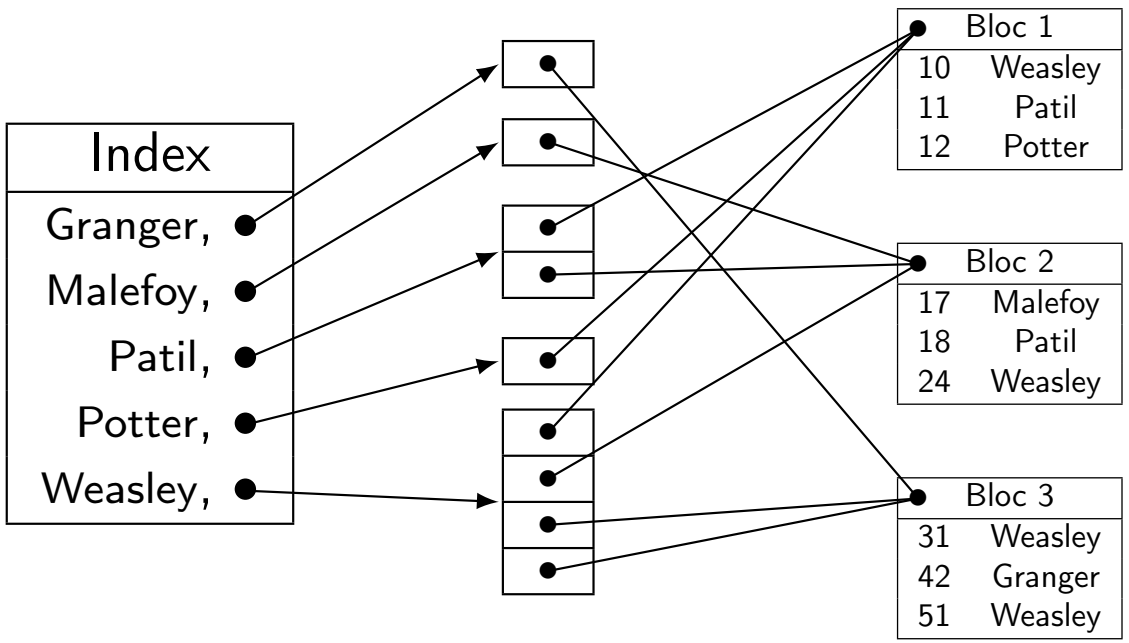
- Pour accélérer l'accès aux enregistrements par un ou plusieurs attributs qui ne sont pas le critère de tri du fichier
  - Recherche par nom sur un fichier trié par numéro d'étudiant
  - Index sur un tas de données (qui n'est donc pas ordonné)
- Accès et mises-à-jour comme dans un index primaire dense

Index secondaire (dense)

- Pour accélérer l'accès aux enregistrements par un ou plusieurs attributs qui ne sont pas le critère de tri du fichier
  - Recherche par nom sur un fichier trié par numéro d'étudiant
  - Index sur un tas de données (qui n'est donc pas ordonné)
- Accès et mises-à-jour comme dans un index primaire dense
- Attention : il faut gérer les doublons

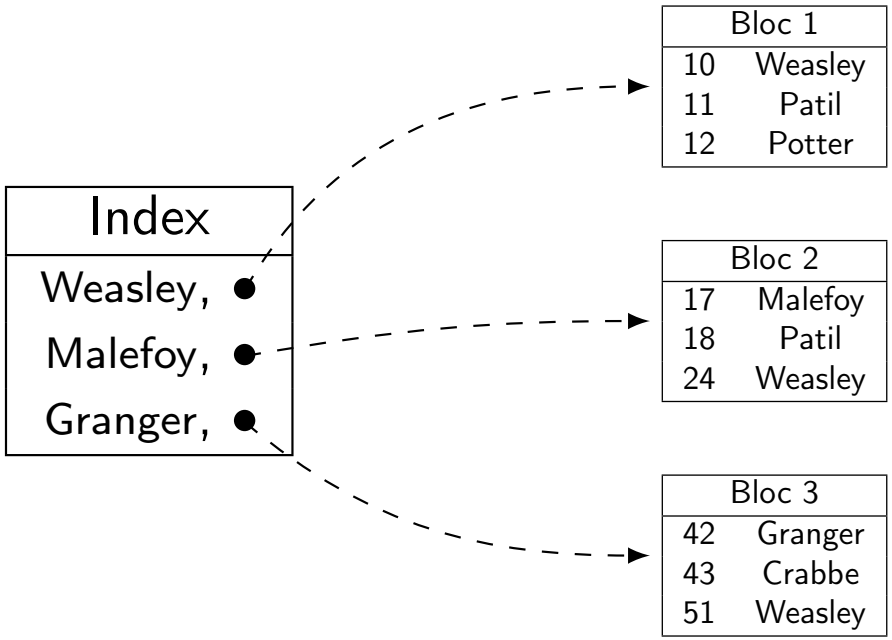
Index secondaire (dense)

- Pour accélérer l'accès aux enregistrements par un ou plusieurs attributs qui ne sont pas le critère de tri du fichier
  - Recherche par nom sur un fichier trié par numéro d'étudiant
  - Index sur un tas de données (qui n'est donc pas ordonné)
- Accès et mises-à-jour comme dans un index primaire dense
- Attention : il faut gérer les doublons

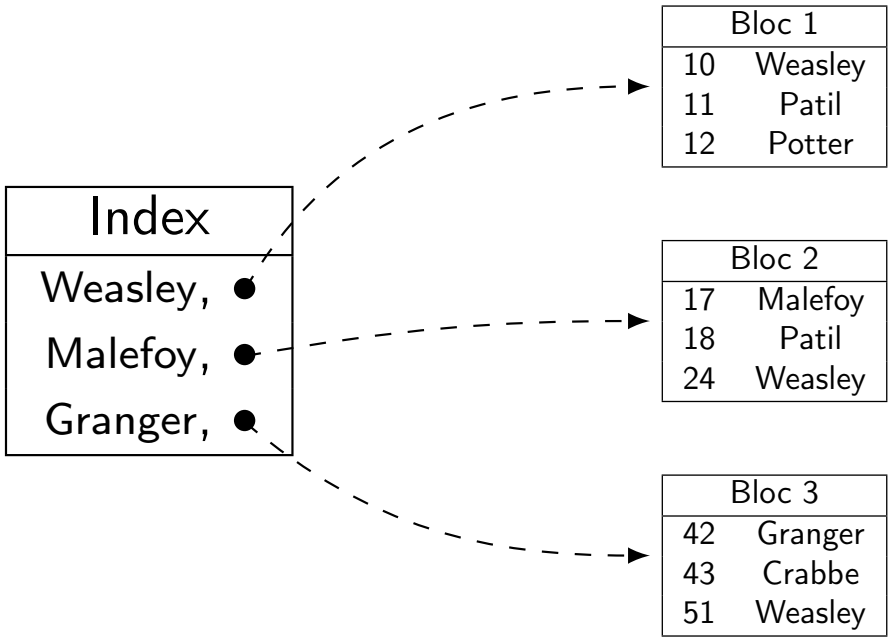


(Chaque clef de recherche pointe sur un tableau de pointeurs appelé bucket)

Et pourquoi pas des index secondaires non-denses ?



Et pourquoi pas des index secondaires non-denses ?



Cet index ne donne presque aucune information !

- Comment savoir où se trouvent les Patil ?
- Y a-t-il des Weasley ailleurs que dans le bloc indexé par Weasley ?

Implémentation des index

Un peu de recul...

Vus de loin, les index sont simplement des structures de données pour stocker et manipuler efficacement des paires clef-valeur $\langle c : v \rangle$

Toute implémentation fournit nécessairement les opérations suivantes :

- Recherche de la valeur v associée à une clef c
- Insertion de la paire $\langle c : v \rangle$
- Suppression de la paire $\langle c : v \rangle$

Un peu de recul...

Vus de loin, les index sont simplement des structures de données pour stocker et manipuler efficacement des paires clef-valeur $\langle c : v \rangle$

Toute implémentation fournit nécessairement les opérations suivantes :

- Recherche de la valeur v associée à une clef c
- Insertion de la paire $\langle c : v \rangle$
- Suppression de la paire $\langle c : v \rangle$

Question : connaissez-vous déjà une structure de données similaire ?

Un peu de recul...

Vus de loin, les index sont simplement des structures de données pour stocker et manipuler efficacement des paires clef-valeur $\langle c : v \rangle$

Toute implémentation fournit nécessairement les opérations suivantes :

- Recherche de la valeur v associée à une clef c
- Insertion de la paire $\langle c : v \rangle$
- Suppression de la paire $\langle c : v \rangle$

Question : connaissez-vous déjà une structure de données similaire ?

$\rightarrow$ les index sont simplement des dictionnaires !

Choix d'implémentation des index

Une bonne implémentation des index est simplement :

- Une implémentation des dictionnaires...
- ...qui soit efficace en terme d'accès disque...
- ...pour les opérations et volumes de données typiques de la BD

Choix d'implémentation des index

Une bonne implémentation des index est simplement :

- Une implémentation des dictionnaires...
- ...qui soit efficace en terme d'accès disque...
- ...pour les opérations et volumes de données typiques de la BD

Plusieurs choix possibles :

- Le choix naïf : liste chaînée et ordonnée, recherche dichotomique
- Les arbres B^+ : une variante des arbres binaires de recherche
- Des tables de hachage, construites sur mesure pour les BD

Choix d'implémentation des index

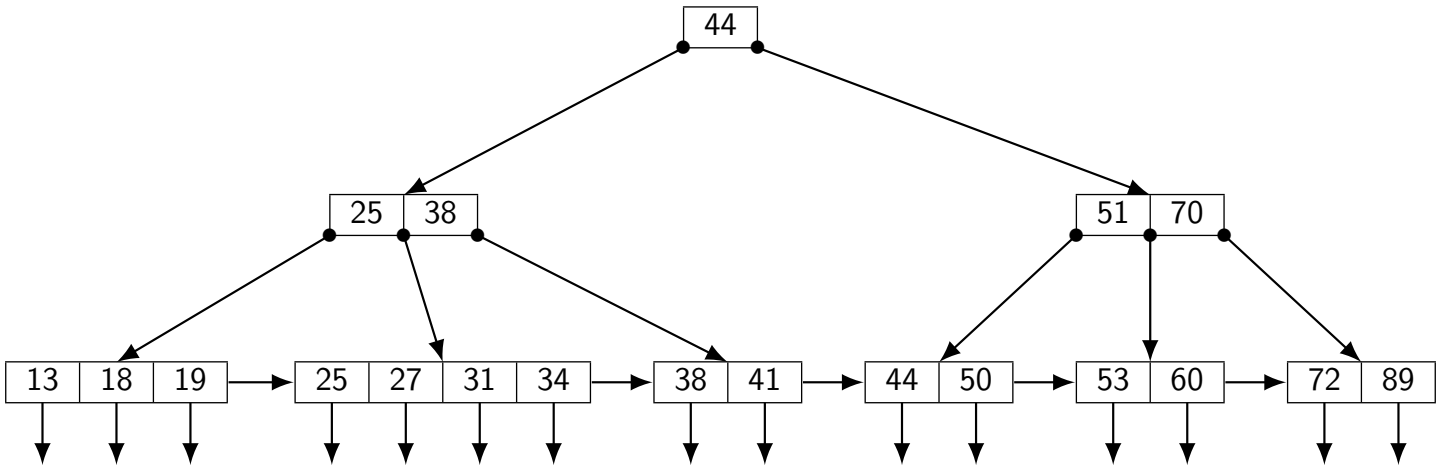
Une bonne implémentation des index est simplement :

- Une implémentation des dictionnaires...
- ...qui soit efficace en terme d'accès disque...
- ...pour les opérations et volumes de données typiques de la BD

Plusieurs choix possibles :

- Le choix naïf : liste chaînée et ordonnée, recherche dichotomique
- Les arbres B^+ : une variante des arbres binaires de recherche
- Des tables de hachage, construites sur mesure pour les BD
- ....et d'autres qui ne seront pas abordés dans ce cours...

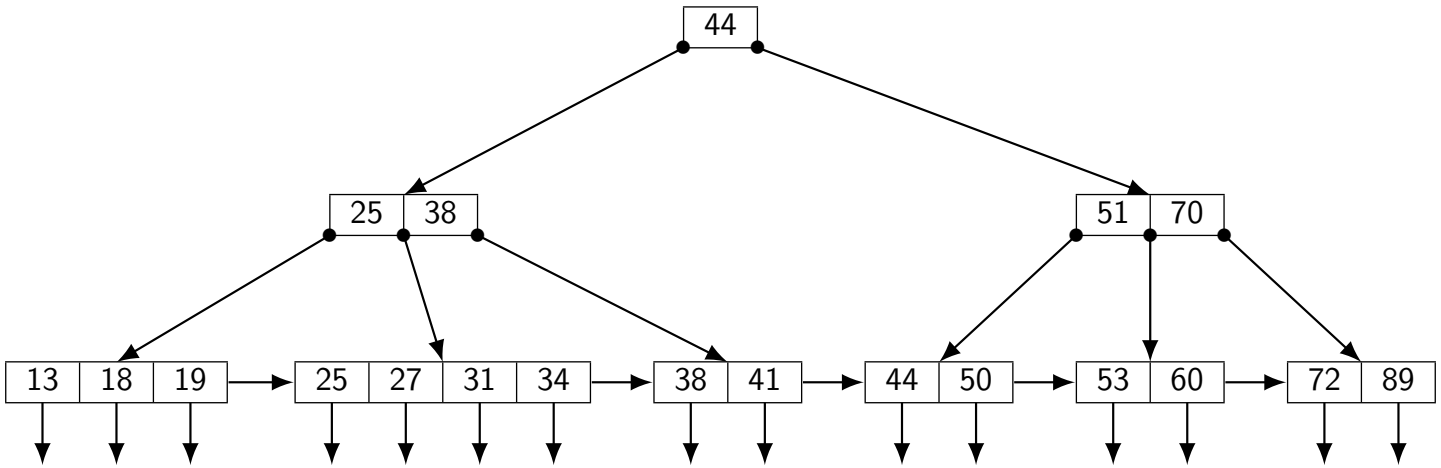
Arbres B^+



Un arbre B^+ (B^+ tree) est un arbre dans lequel :

- Chaque noeud (interne ou feuille) est stocké sur un bloc du disque
- Les noeuds internes servent de balise pour orienter la recherche
- Les feuilles contiennent les paires clef-pointeur

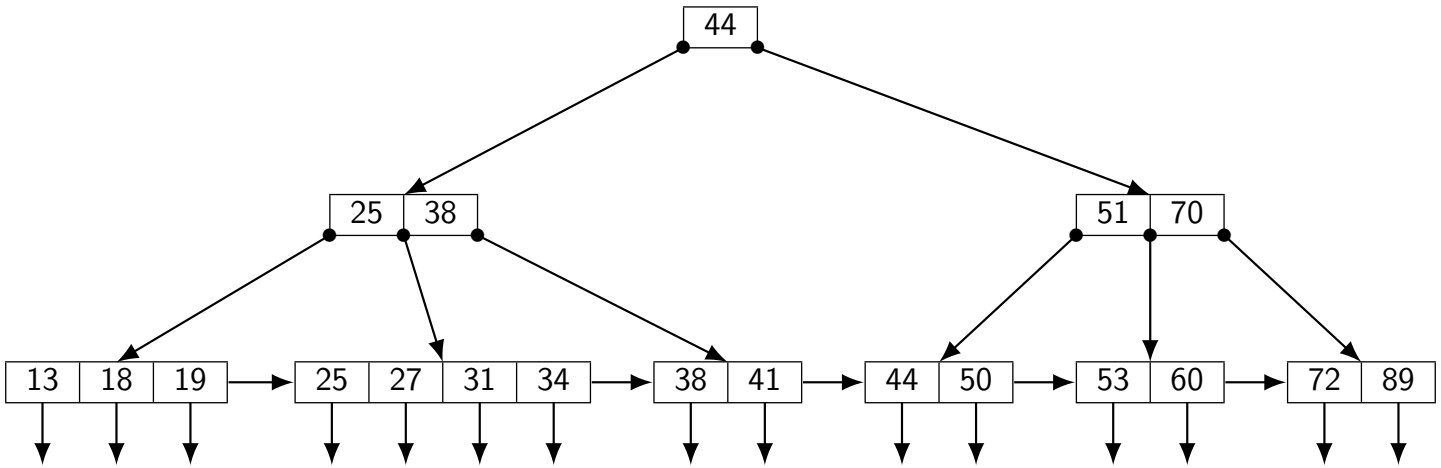
Arbres B^+



Si un noeud contient k clefs, il contient aussi $k+1$ pointeurs, avec :

- Aux feuilles : les k pointeurs vers les enregistrements de la base de données associés aux clefs et un pointeur vers la feuille suivante
- Aux noeuds internes : un pointeur entre chaque paire de clef C_i, C_{i+1} (et aux extrémités) vers le sous-arbre contenant les clefs $\geq C_i$ et $< all : C_{i+1}$

Arbres B^+



$N \stackrel{\text{def}}{=}$ maximum tel qu'un bloc peut contenir N clefs et $N+1$ pointeurs

Les noeuds doivent respecter des contraintes d'équilibrage :

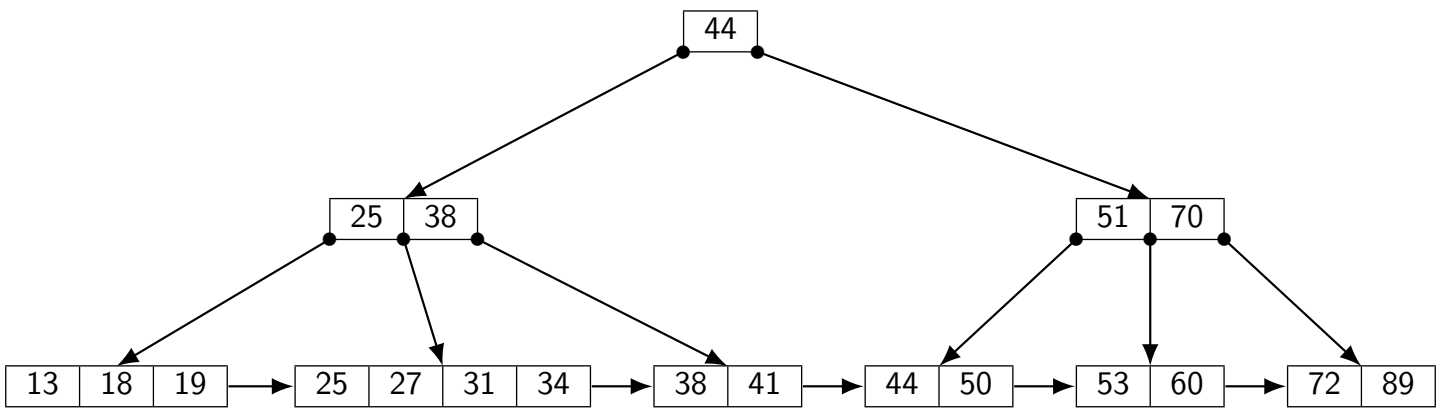
- Racine : entre 1 et N clefs
- Noeud interne : entre $\lfloor N/2 \rfloor$ et N clefs
- Feuille : entre $\lceil N/2 \rceil$ et N clefs

(arbre équilibré $\Rightarrow$ pas de blocs trop vides $\Rightarrow$ meilleures performances)

Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

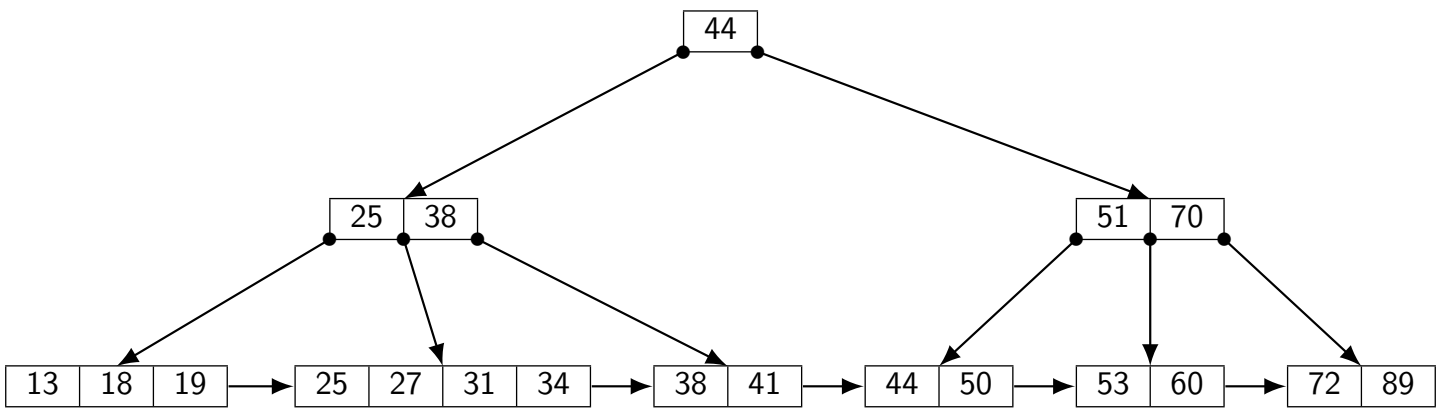


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34

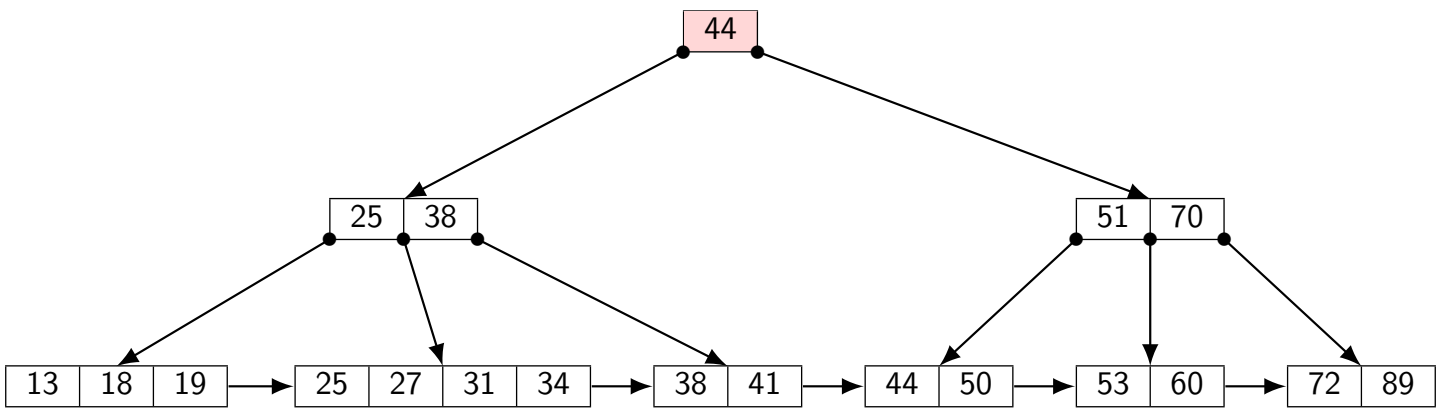


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34

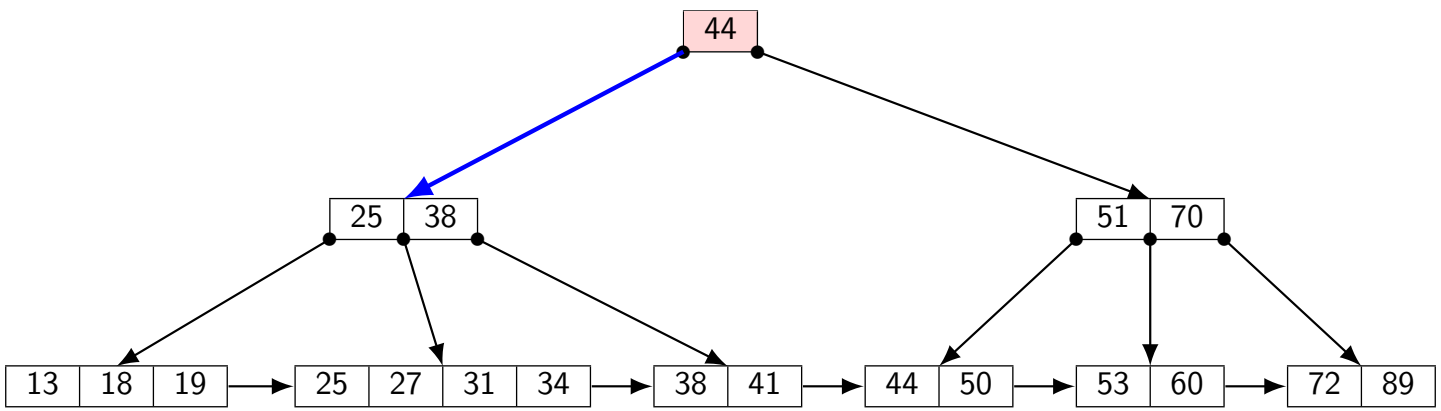


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34

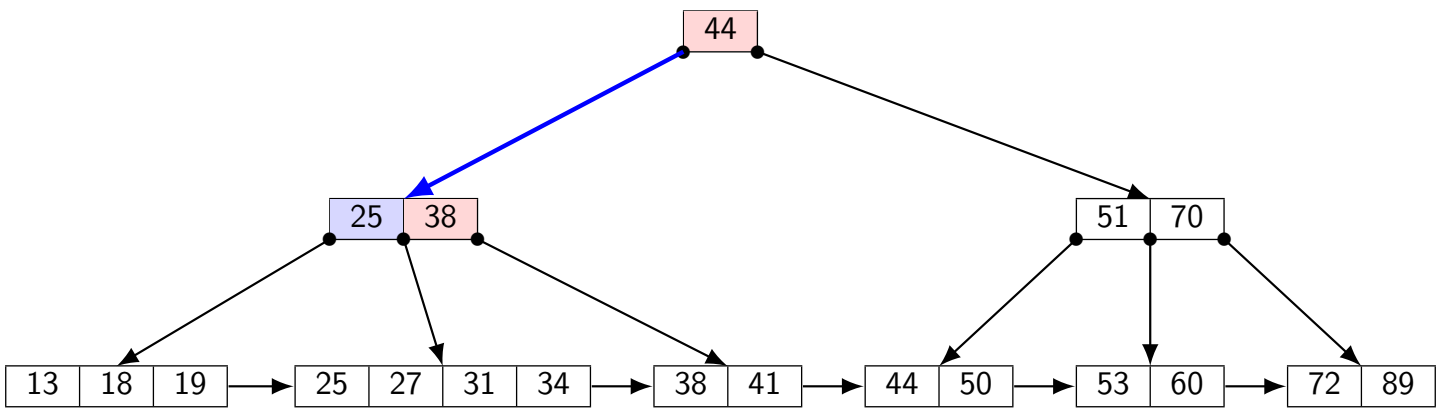


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34

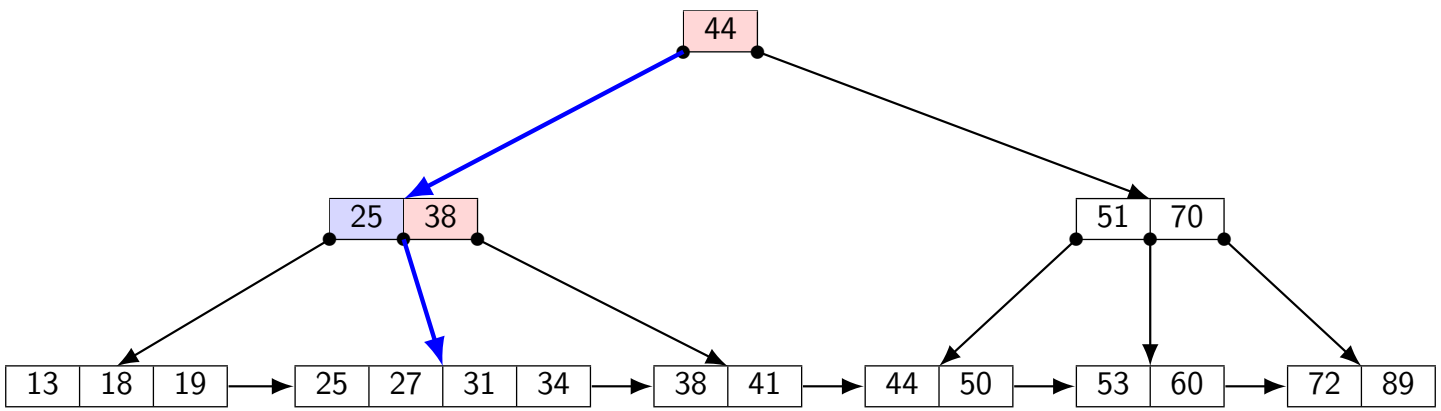


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34

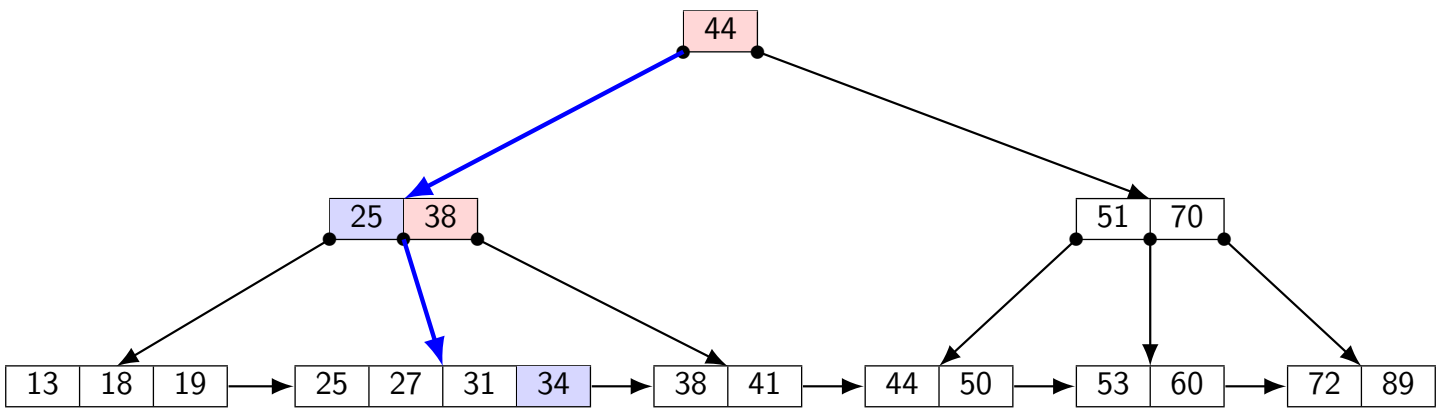


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34

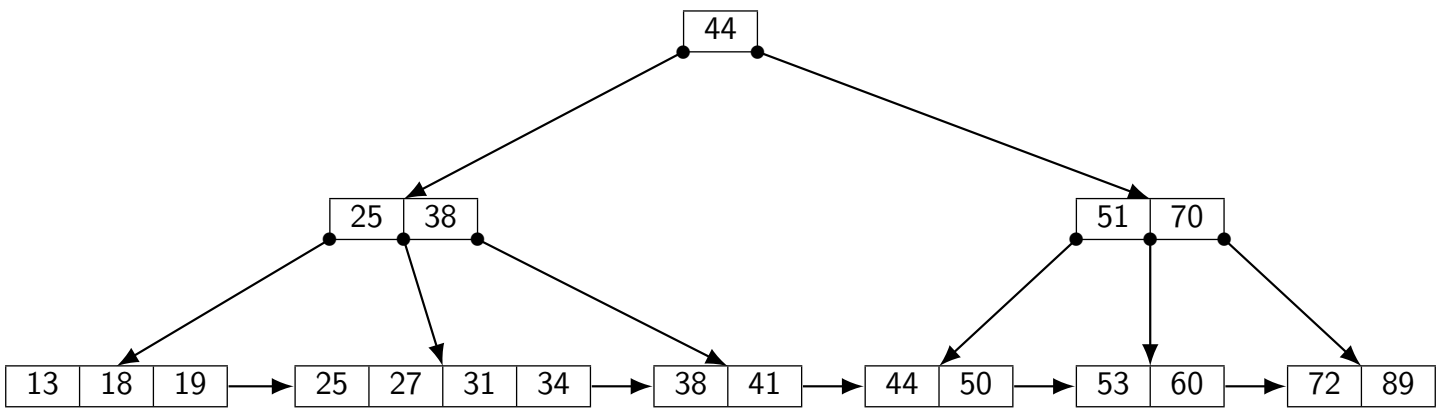


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49

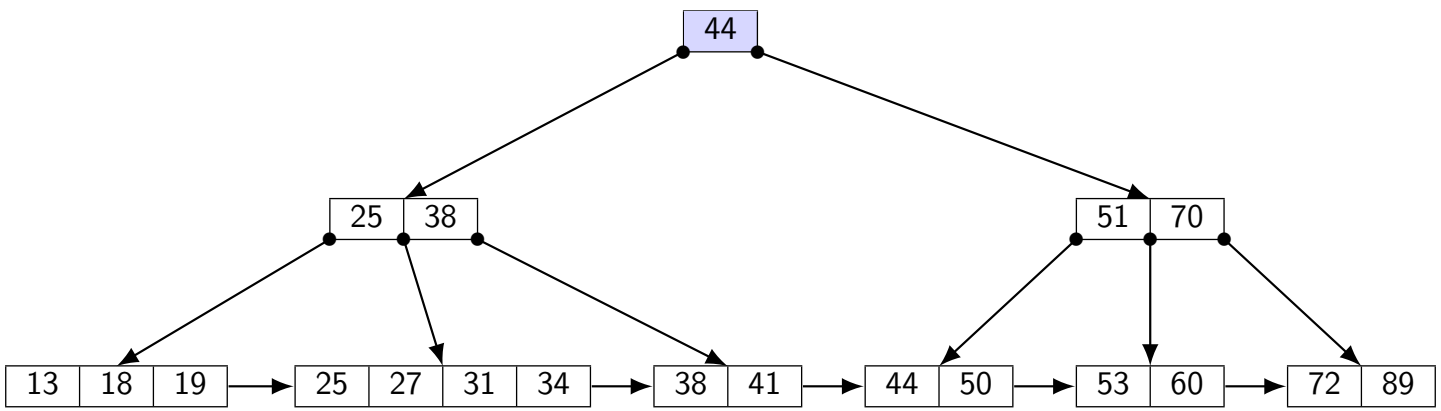


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49

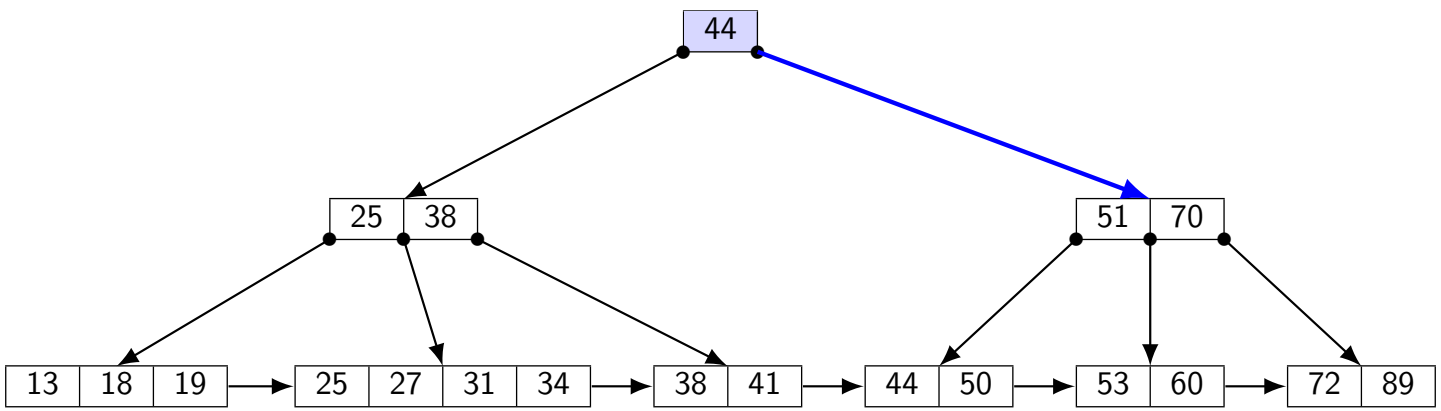


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49

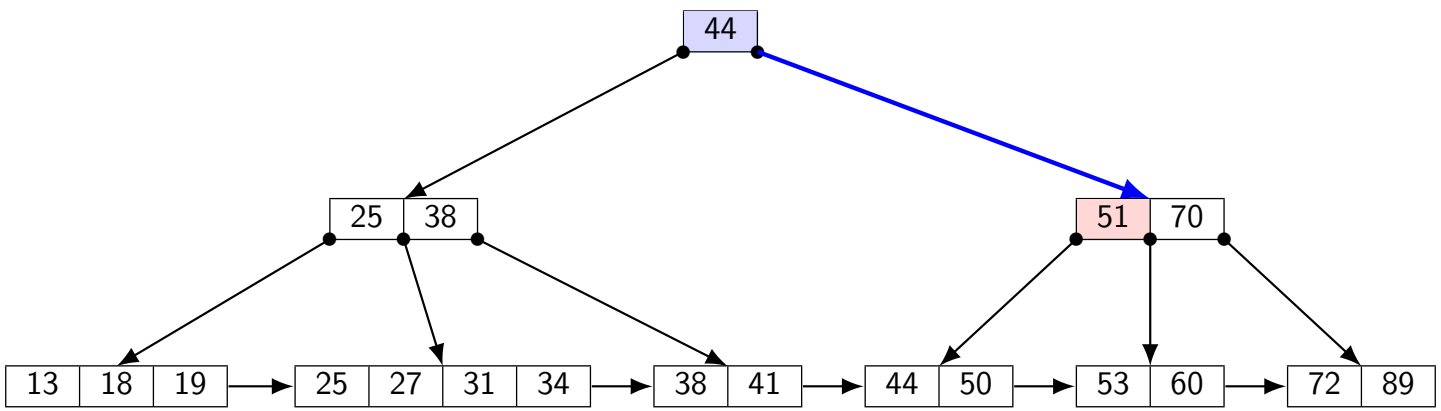


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49

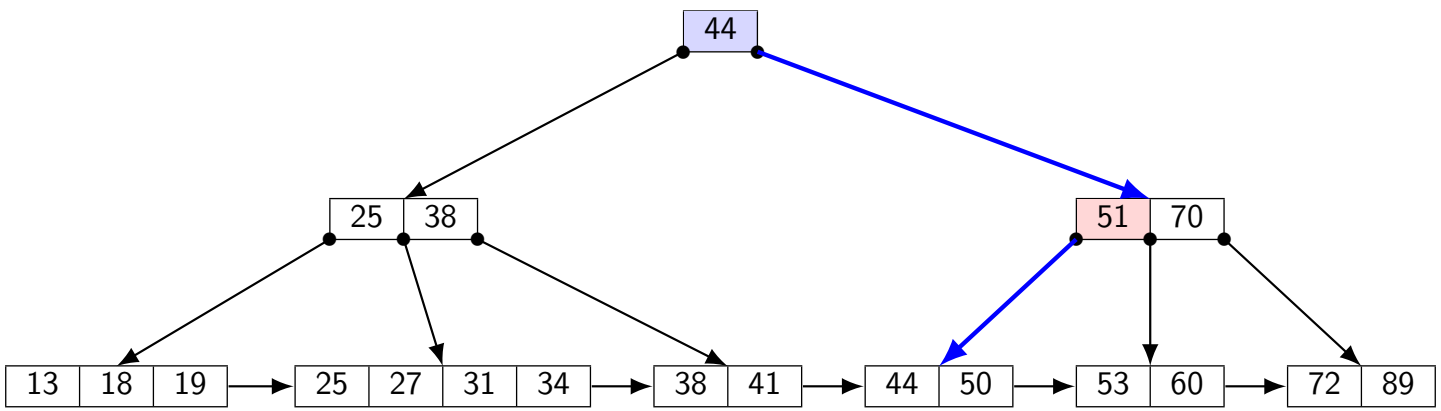


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49

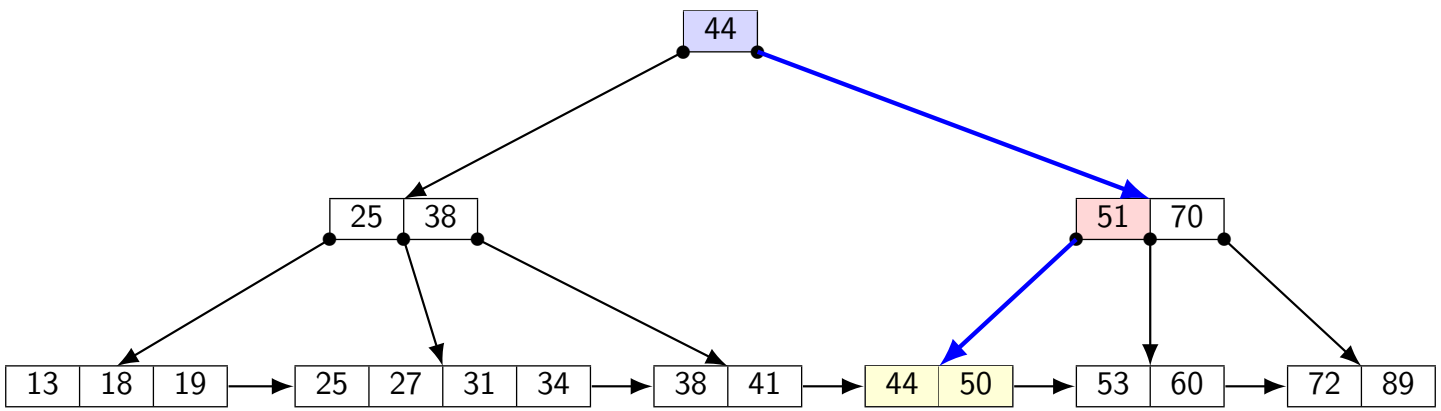


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49

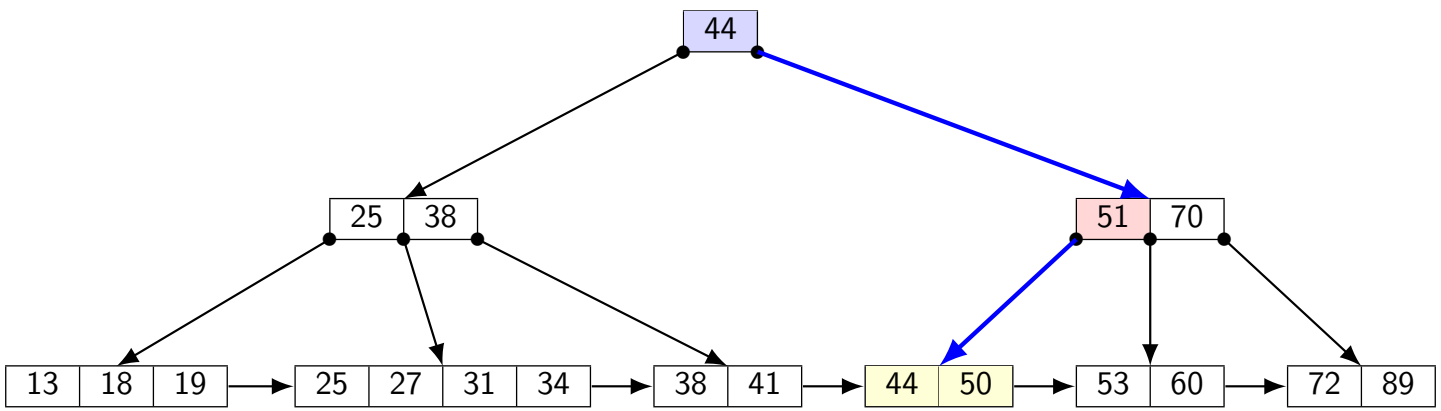


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé)

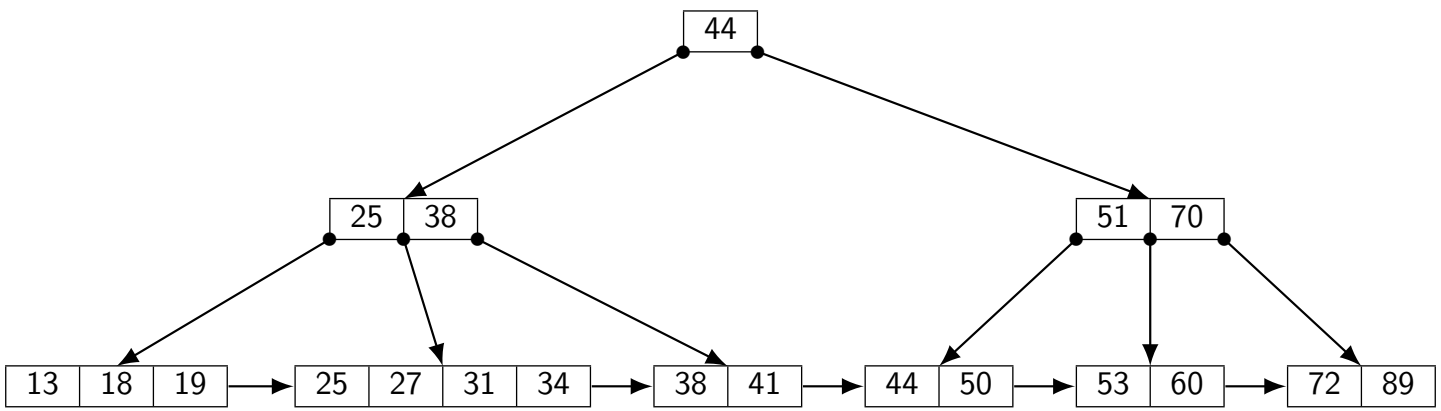


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51

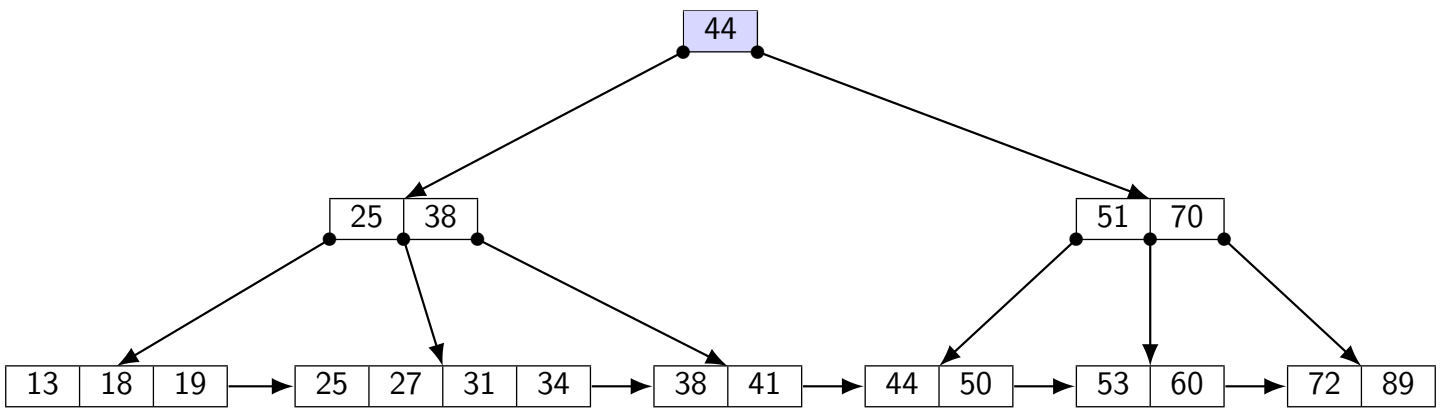


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51

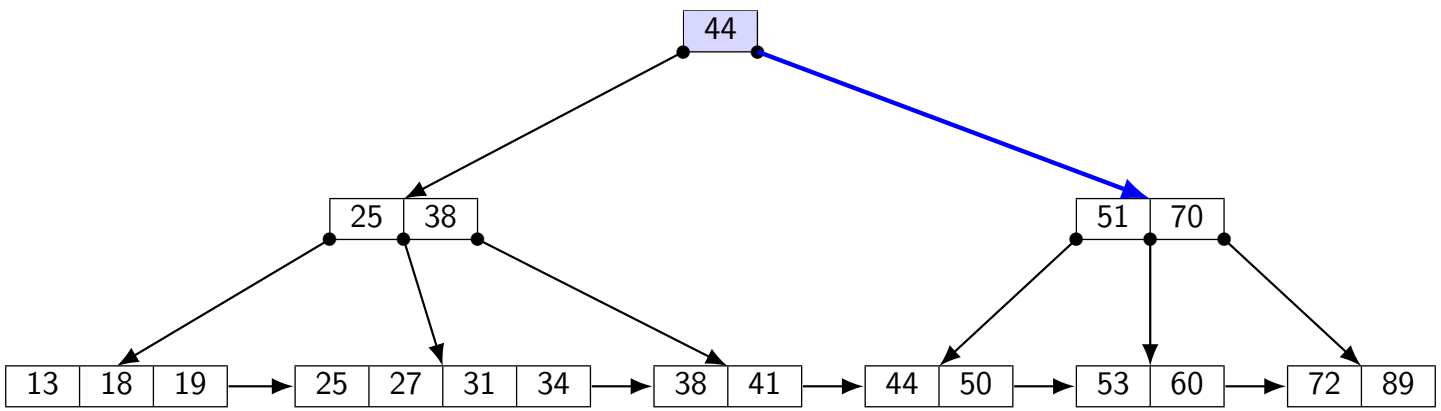


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51

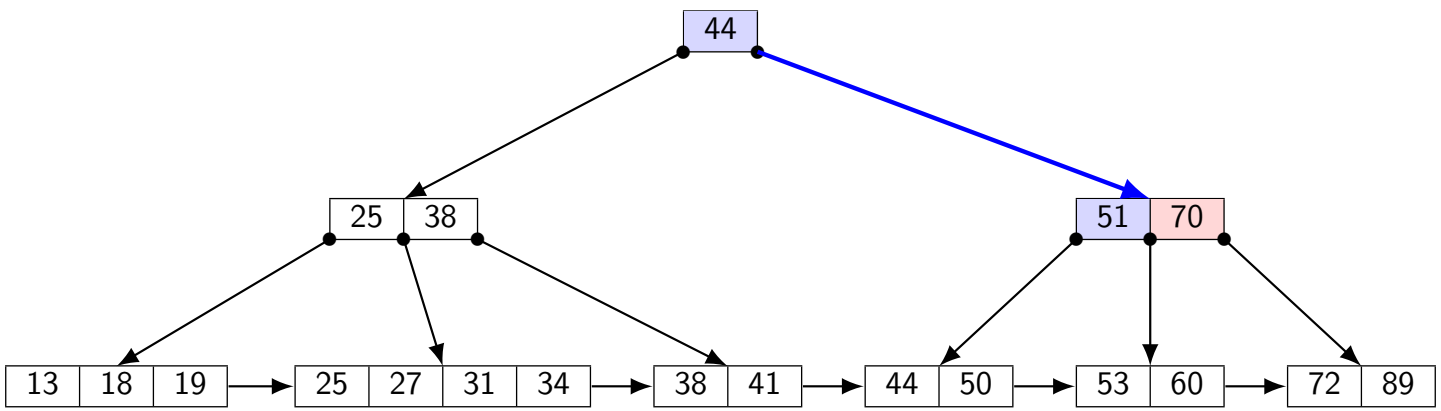


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51

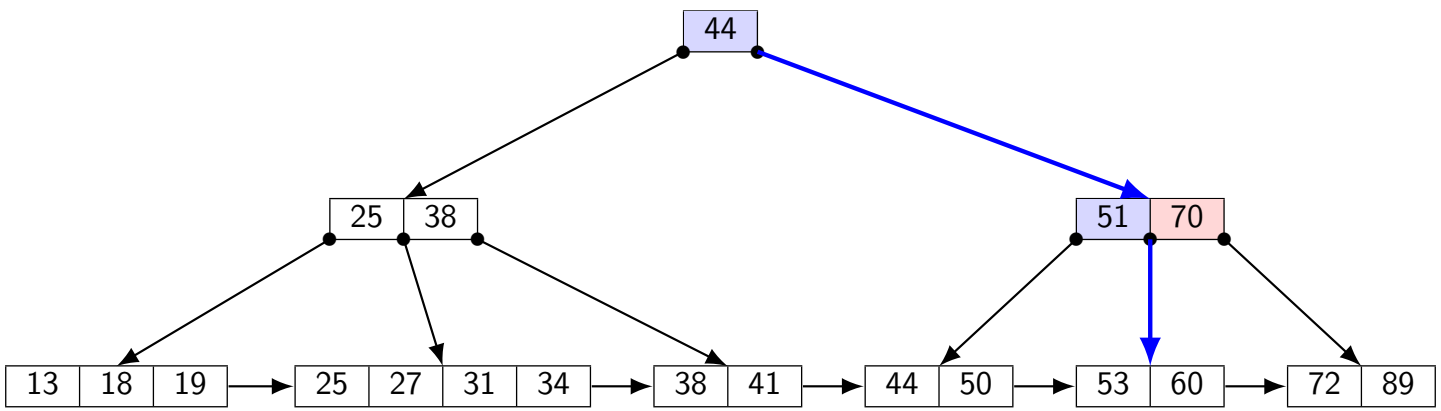


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51

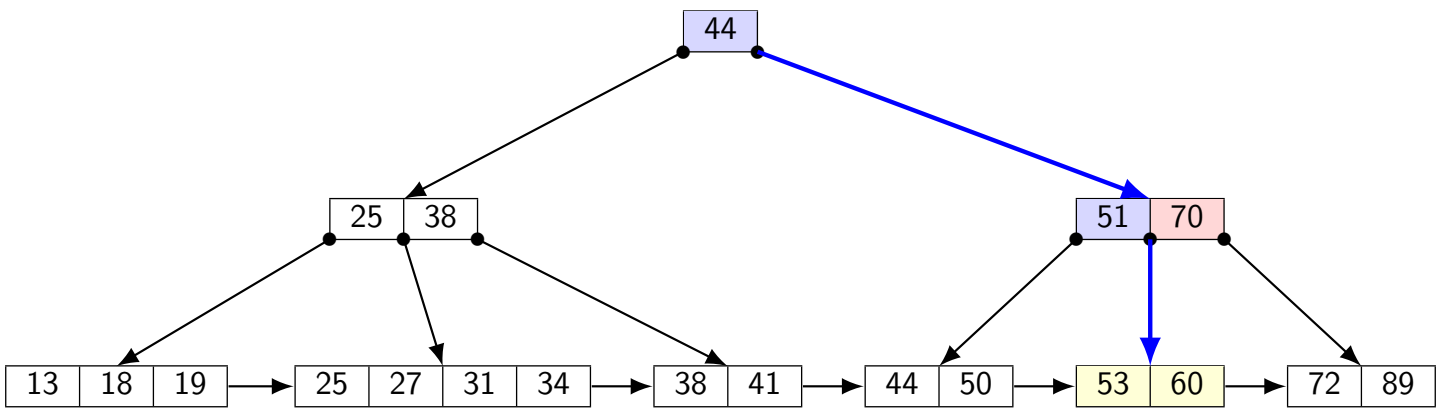


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51

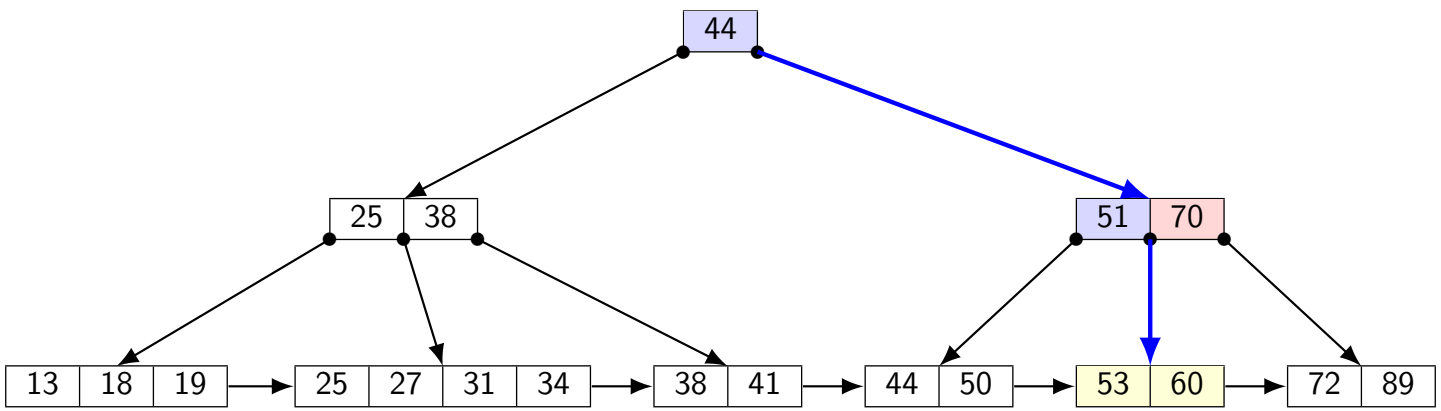


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51 (non trouvé)

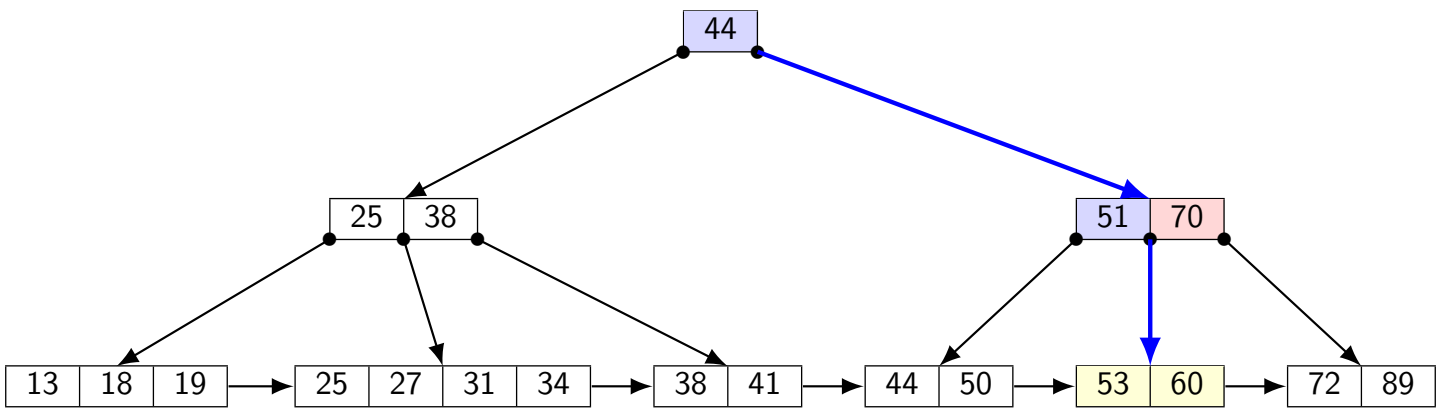


Arbres B^+ : recherche

Recherche d'une clef c :

- Recherche top-down : de la racine vers les feuilles
- À chaque noeud interne, suivre le pointeur entre C_i et C_{i+1} tel que $C_i \leq c < C_{i+1}$ (ou pointeurs aux extrémités si c est $<$ ou $\geq$ à toutes les clefs)
- Si c n'est pas dans la feuille atteinte, c n'est pas dans l'index

Ex : recherche de 34, 49 (non trouvé), 51 (non trouvé)

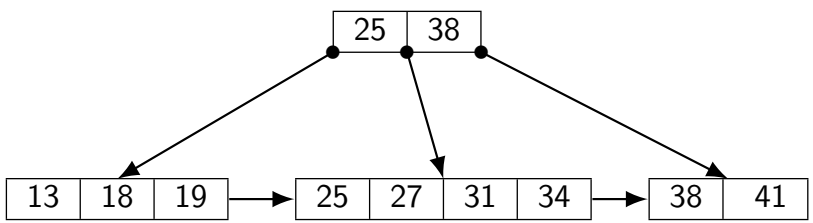


Complexité de la recherche : $O(hauteur)$

Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

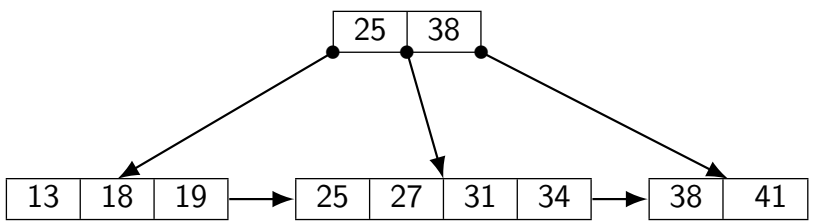


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39

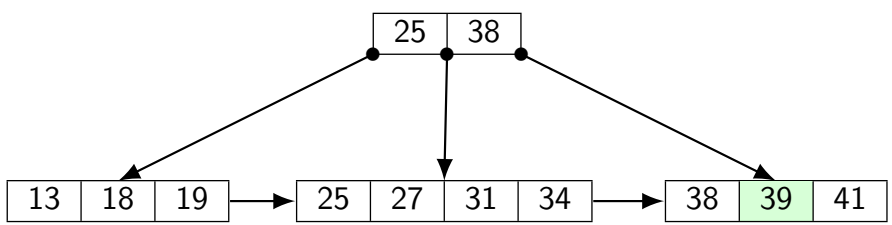


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39

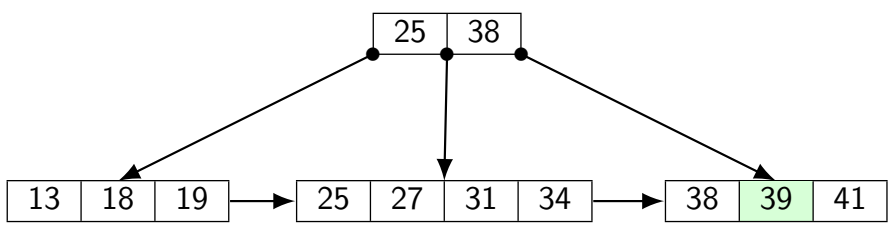


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20

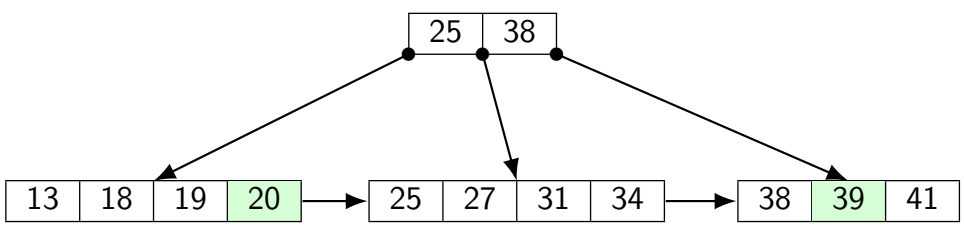


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20

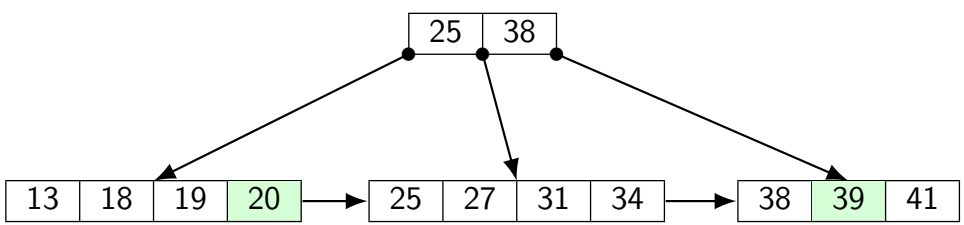


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10

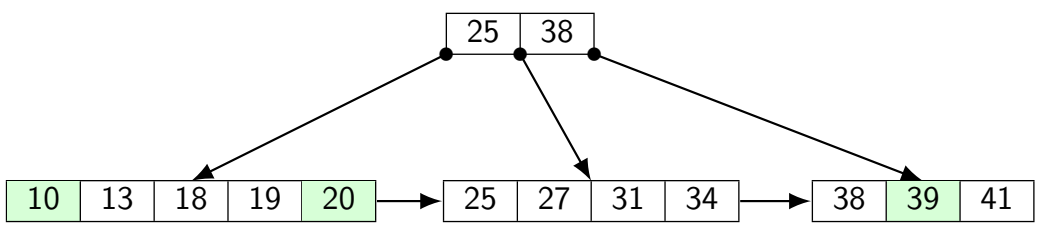


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10

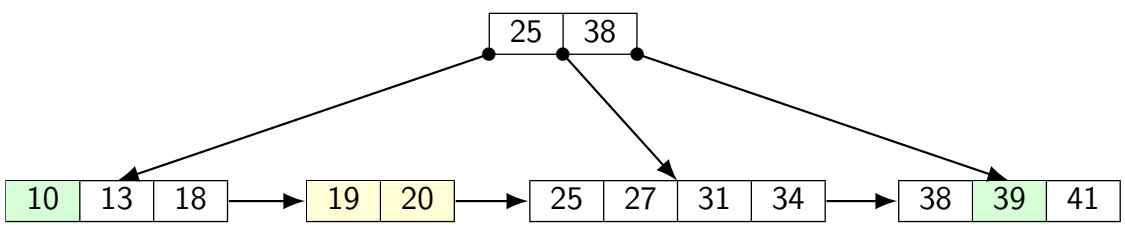


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10

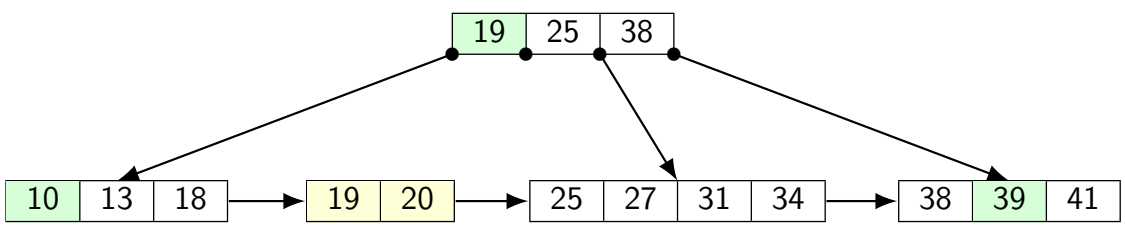


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10

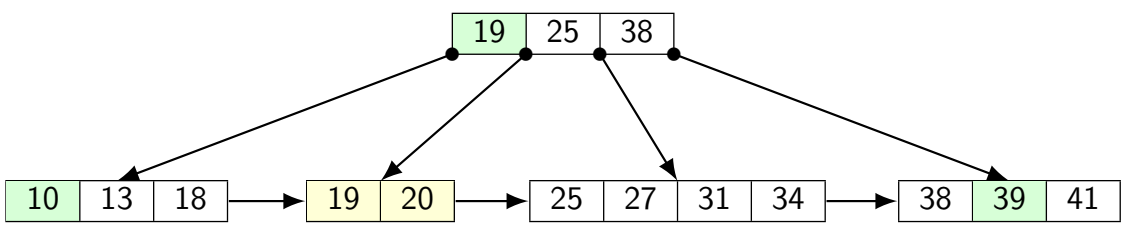


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10

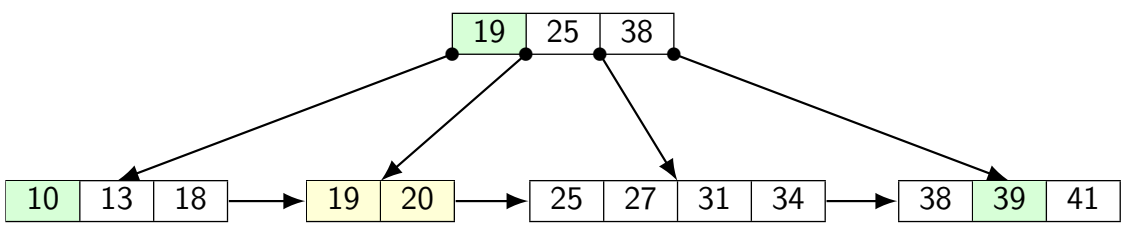


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29

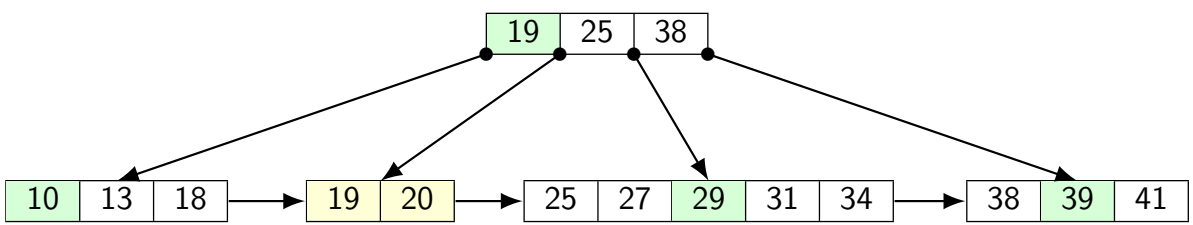


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29

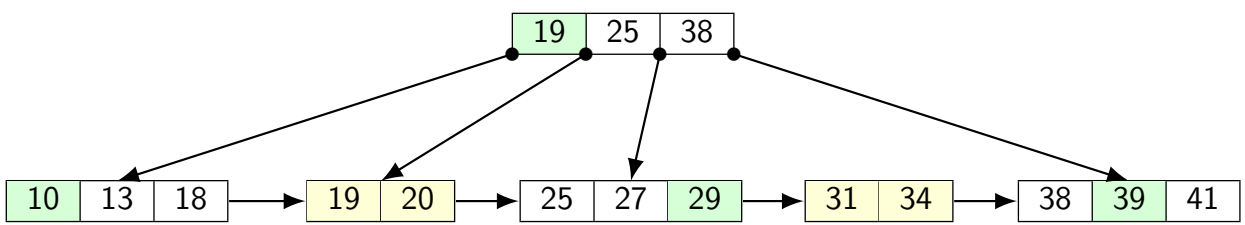


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29

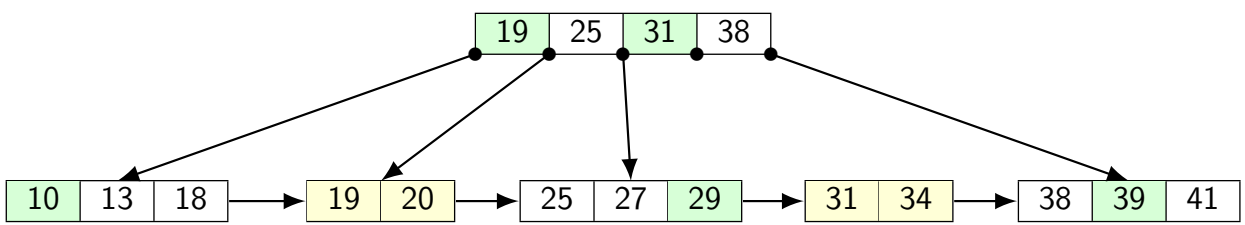


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29

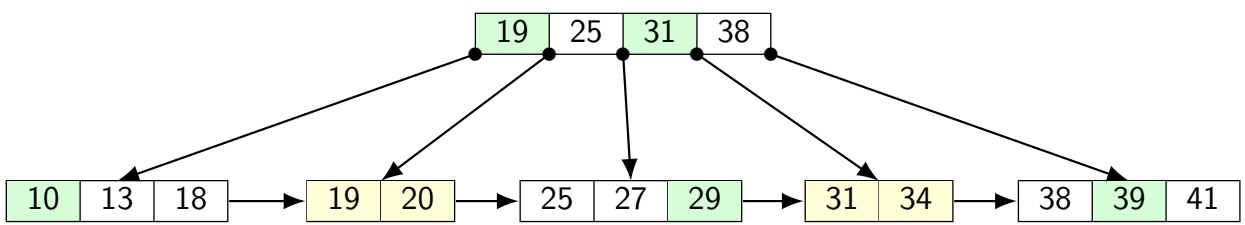


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29

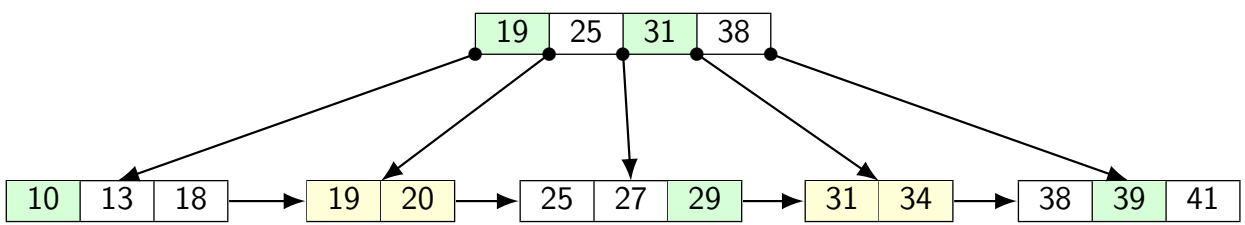


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4

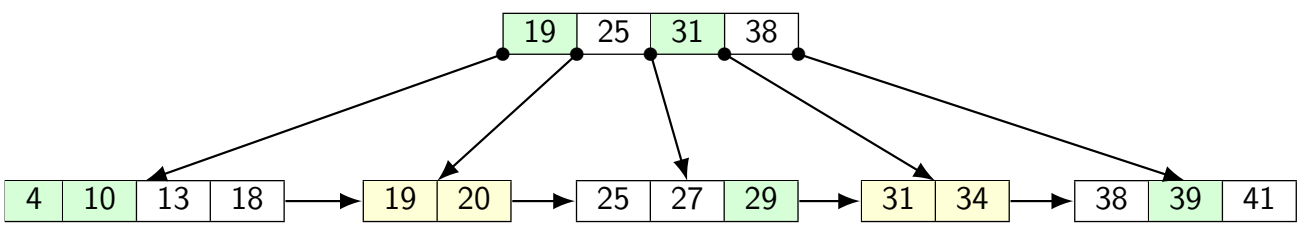


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4

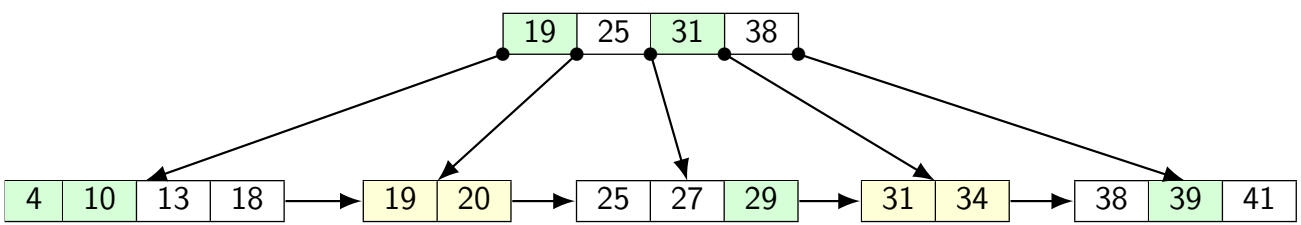


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4 et 6

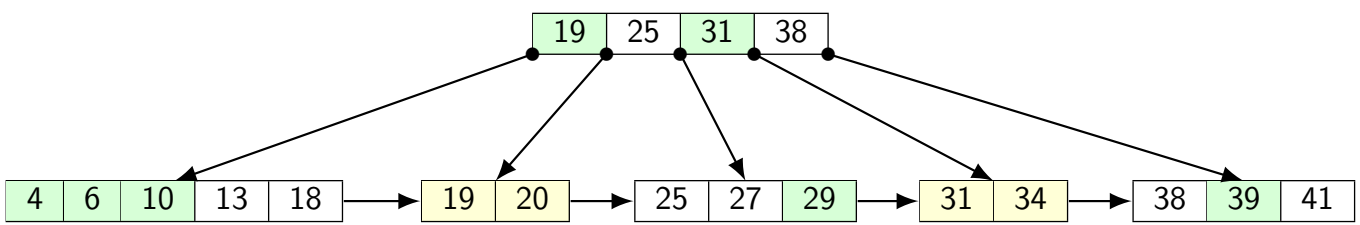


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4 et 6

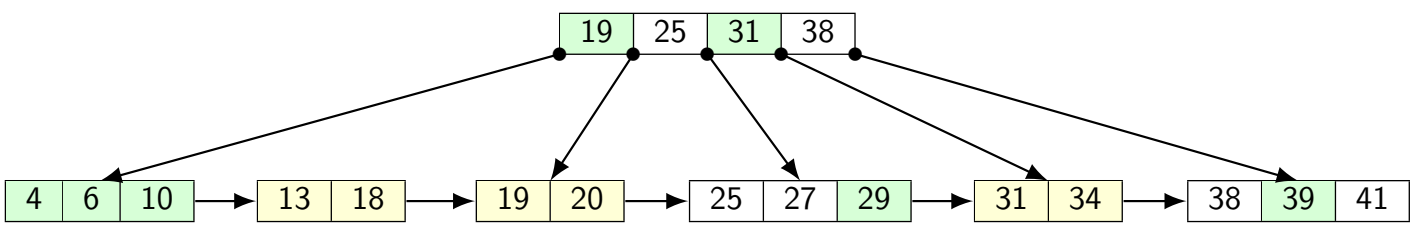


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4 et 6

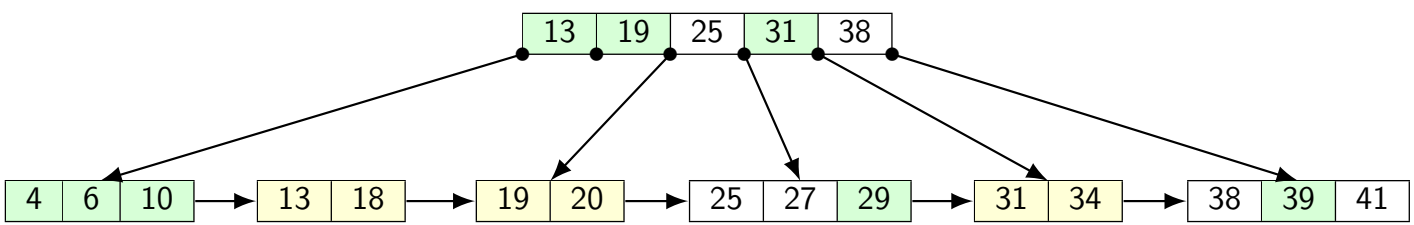


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4 et 6

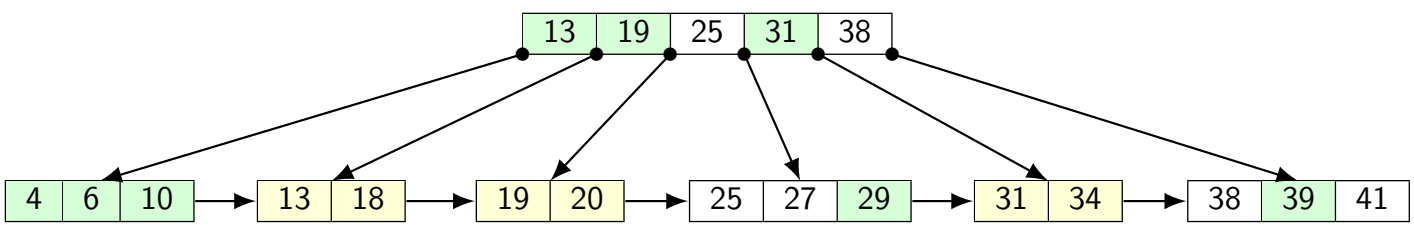


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4 et 6

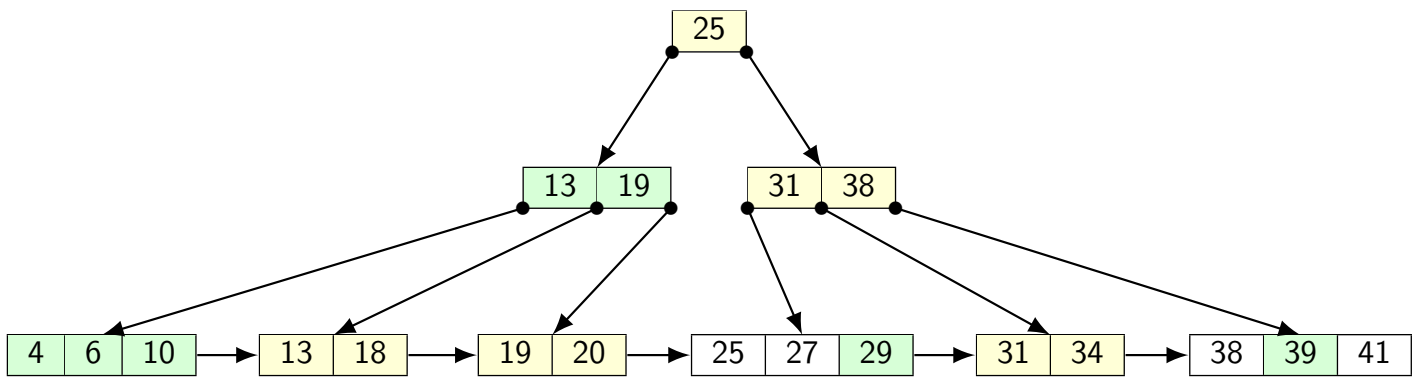


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4 et 6

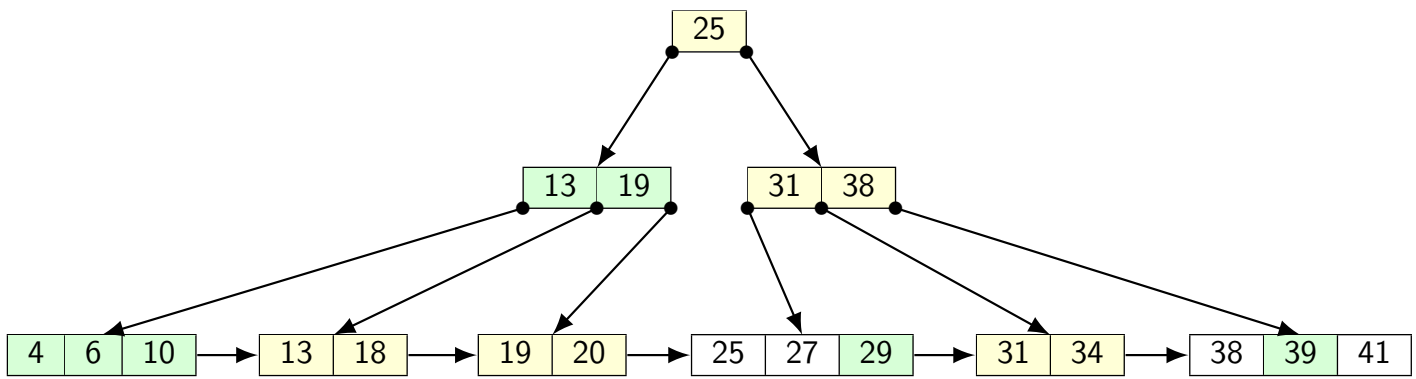


Arbres B^+ : insertion

Insertion d'une nouvelle clef c :

- Recherche de la feuille où doit se trouver c et insertion ordonnée
- Si la feuille déborde : rééquilibrage par une suite d'éclatements (split)
  - Feuille : division en deux feuilles de taille $\lceil \frac{N+1}{2} \rceil$ et $\lfloor \frac{N+1}{2} \rfloor$
    copie de la première clef de la deuxième feuille dans le père
  - Noeud interne : division en deux noeuds de taille $\lceil \frac{N}{2} \rceil$ et $\lfloor \frac{N}{2} \rfloor$
    déplacement de la clef $\lfloor \frac{N}{2} \rfloor + 1$ vers le père (ou nouvelle racine)

Ex (avec $N = 4$) : Insertion de 39, 20, 10, 29, 4 et 6

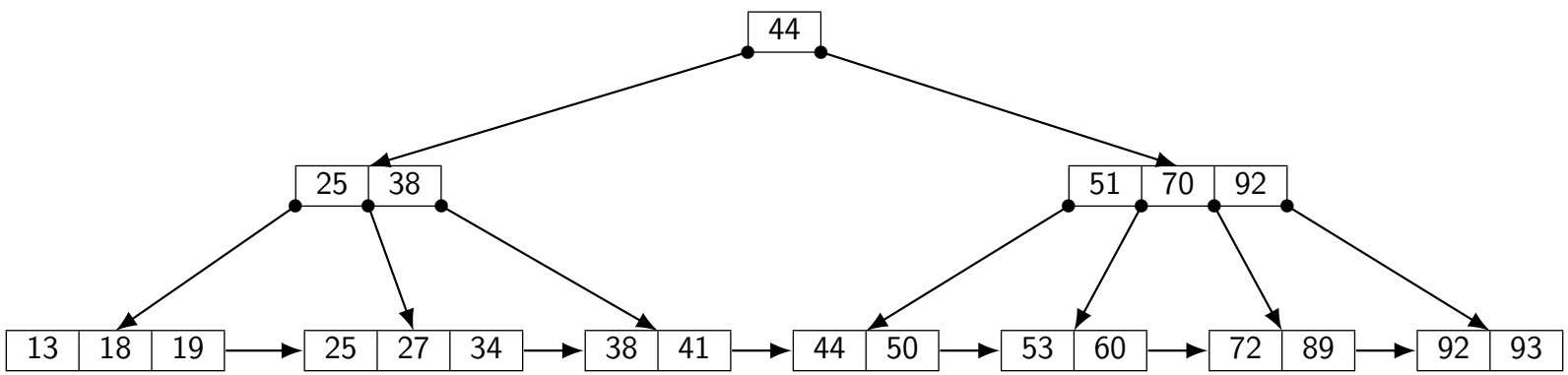


Complexité de l'insertion : $O(hauteur)$

Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

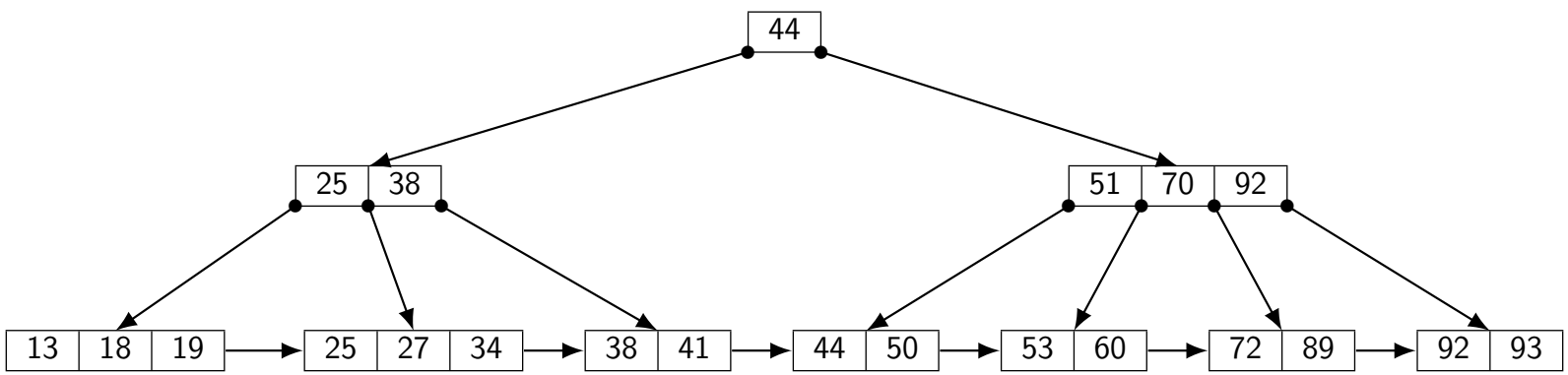


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19

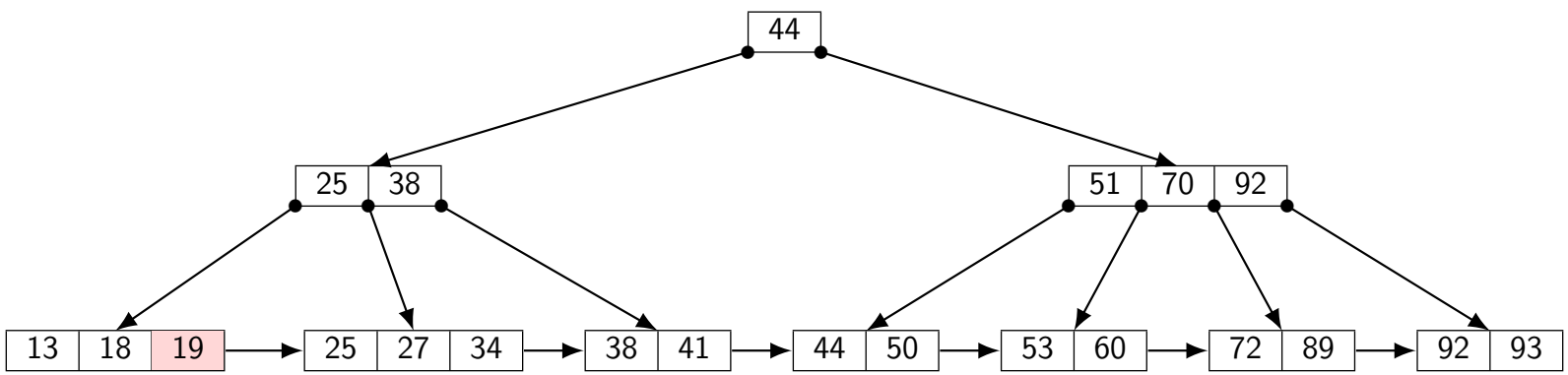


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19

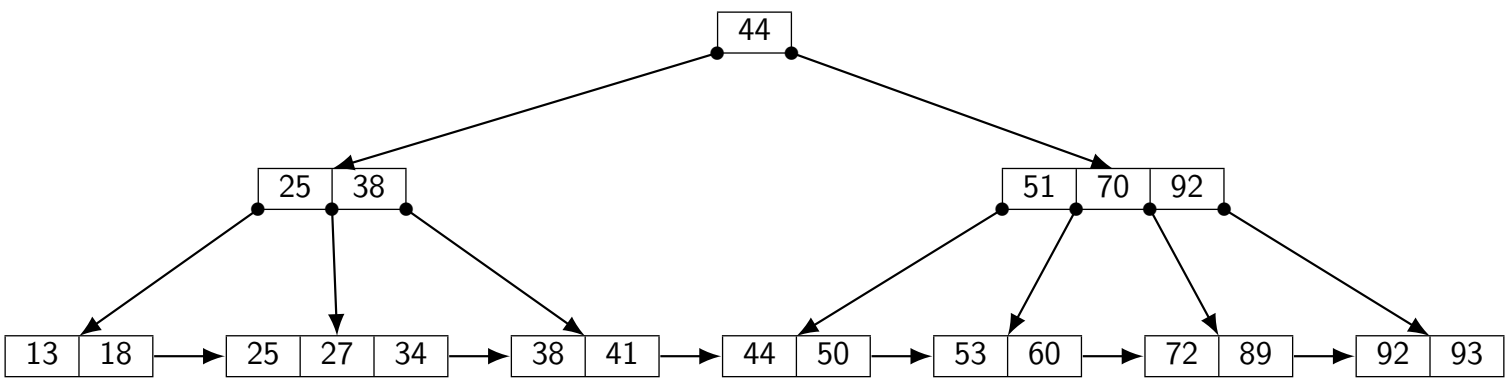


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19

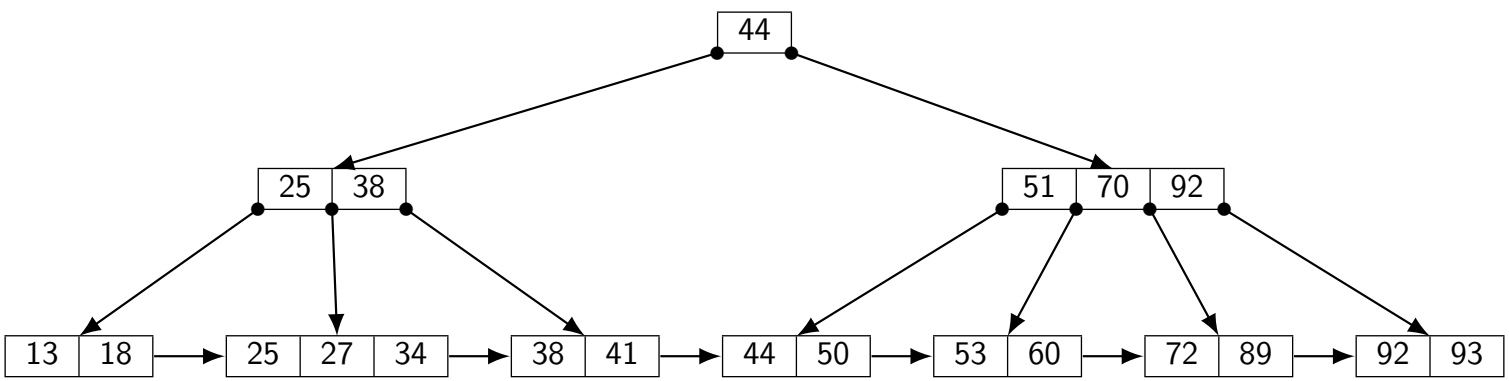


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13

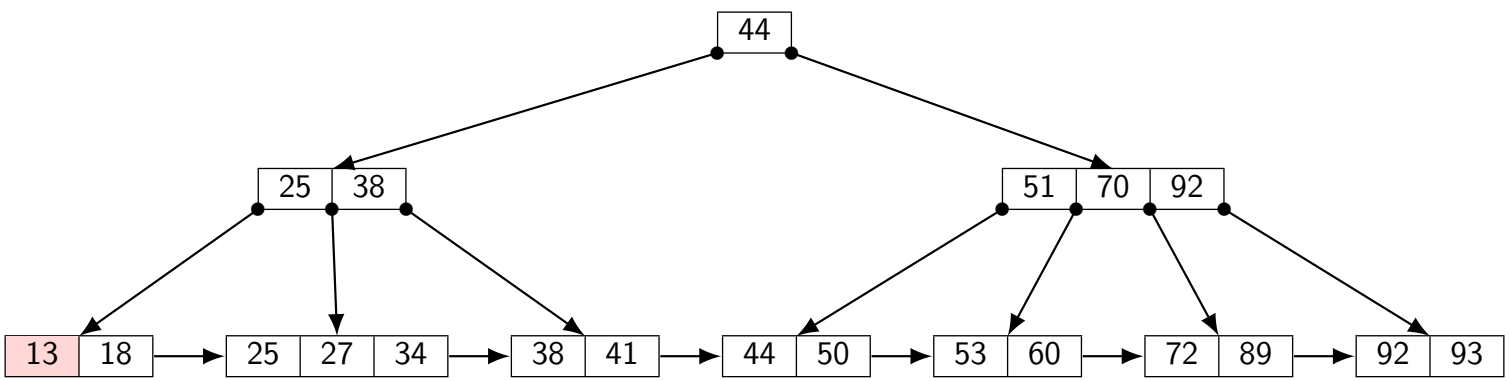


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13

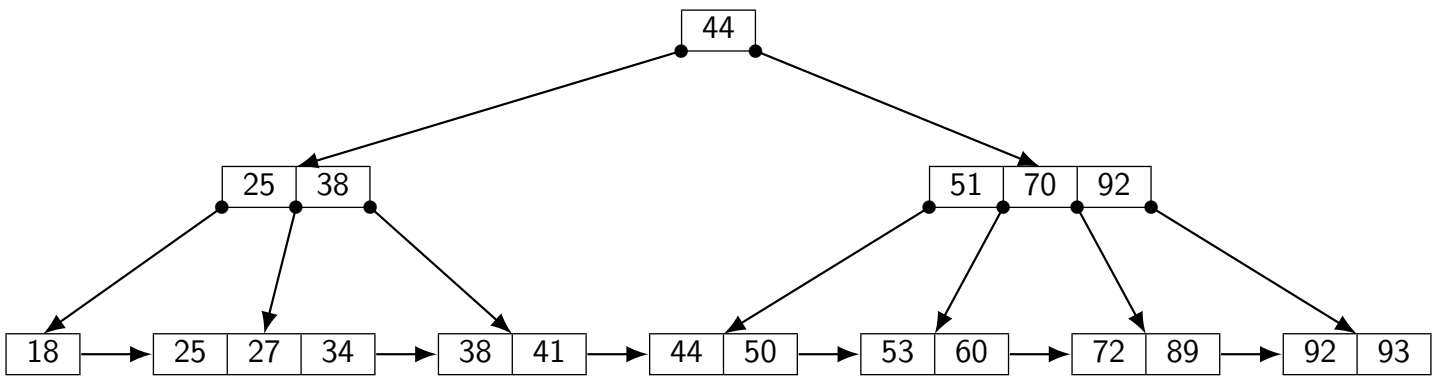


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13

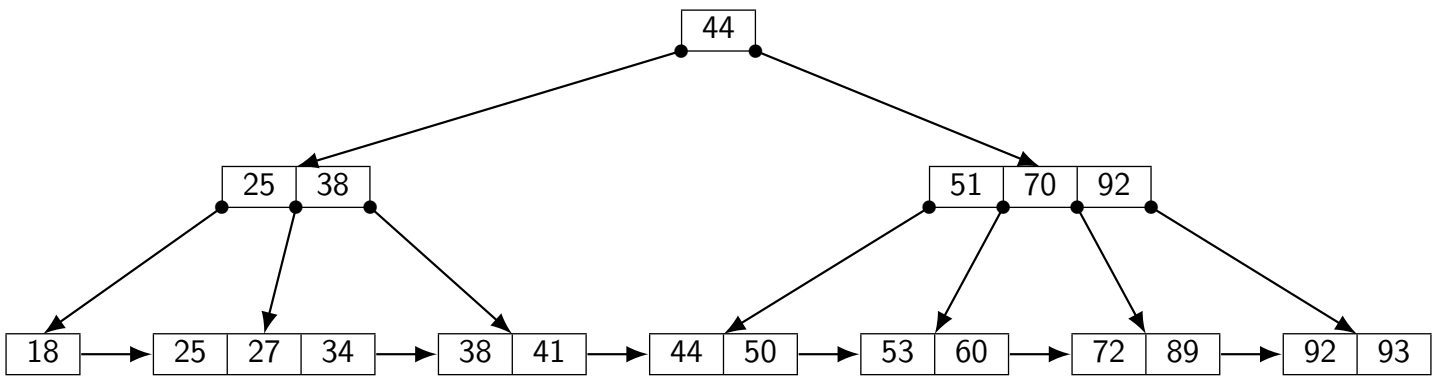


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
  Déplacement de la clef avec son sous-arbre vers le noeud trop vide
  Cas feuille : mise-à-jour de la clef du père
  Cas interne : échange père-fils des deux clefs erronées
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13

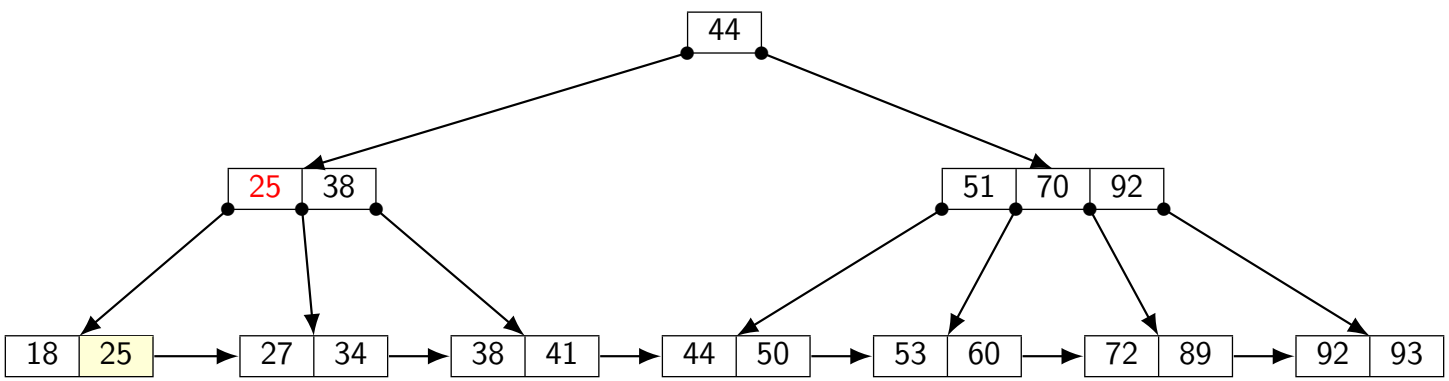


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
  Déplacement de la clef avec son sous-arbre vers le noeud trop vide
  Cas feuille : mise-à-jour de la clef du père
  Cas interne : échange père-fils des deux clefs erronées
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13

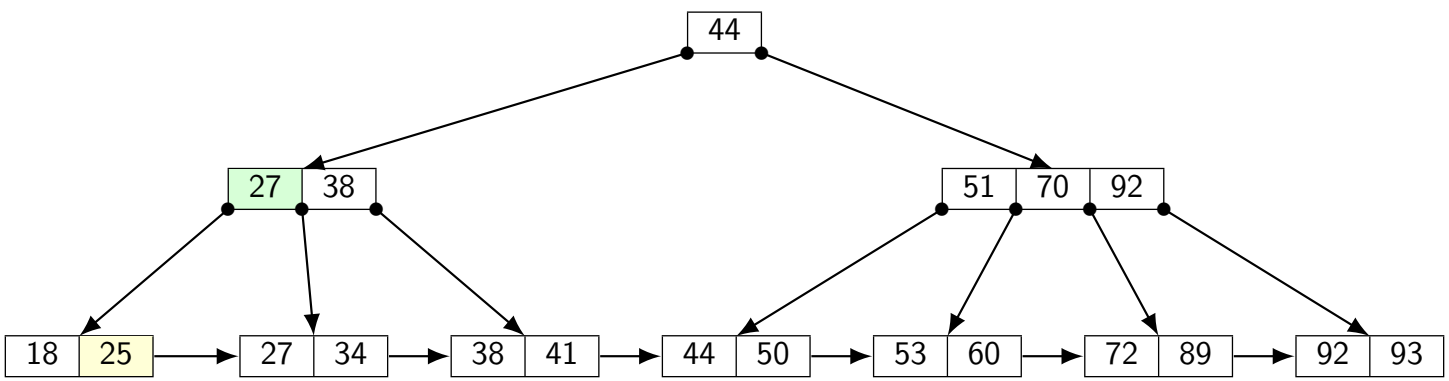


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
  Déplacement de la clef avec son sous-arbre vers le noeud trop vide
  Cas feuille : mise-à-jour de la clef du père
  Cas interne : échange père-fils des deux clefs erronées
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13

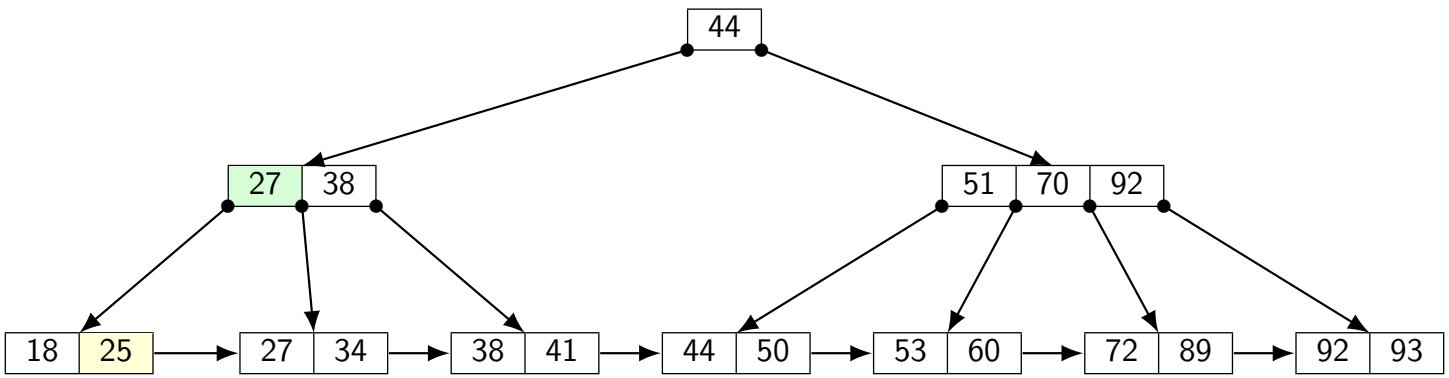


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34

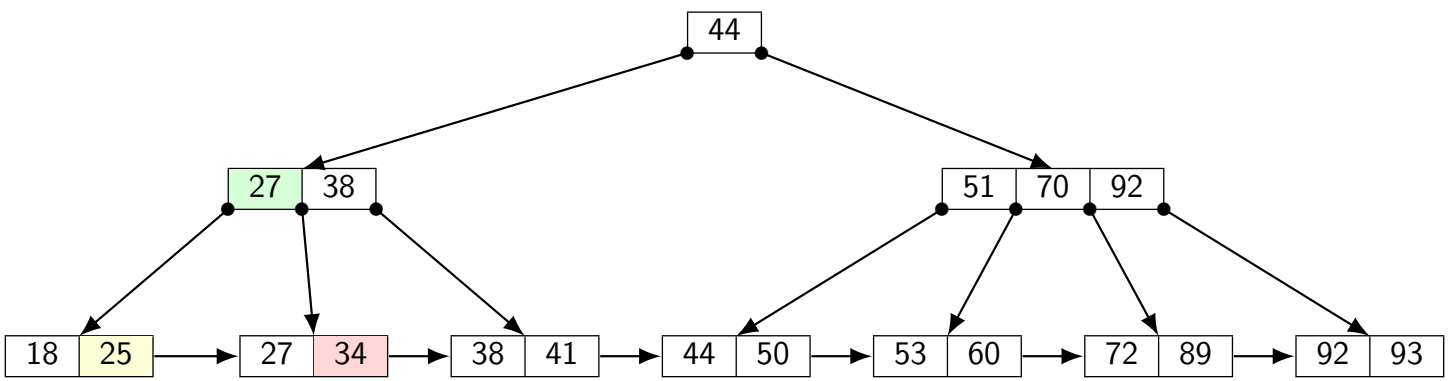


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34

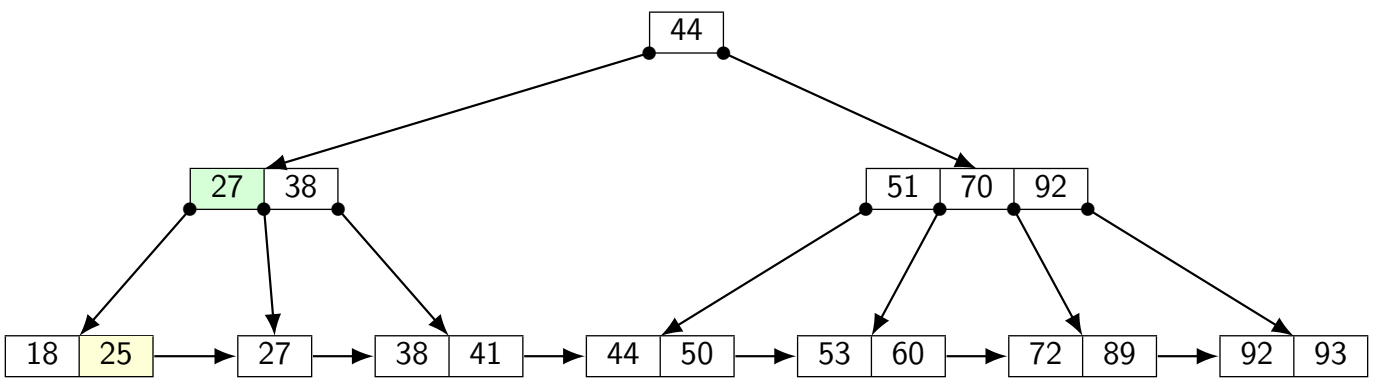


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34

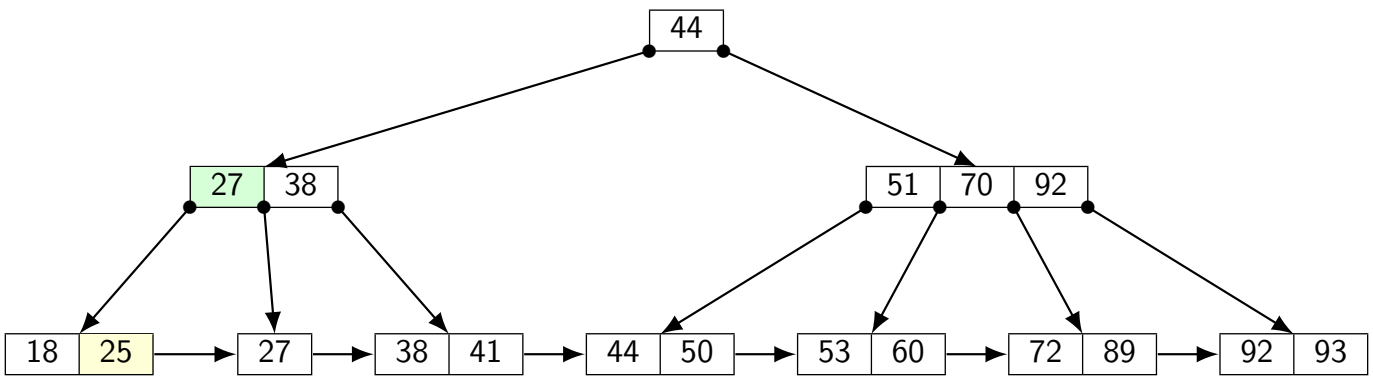


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34

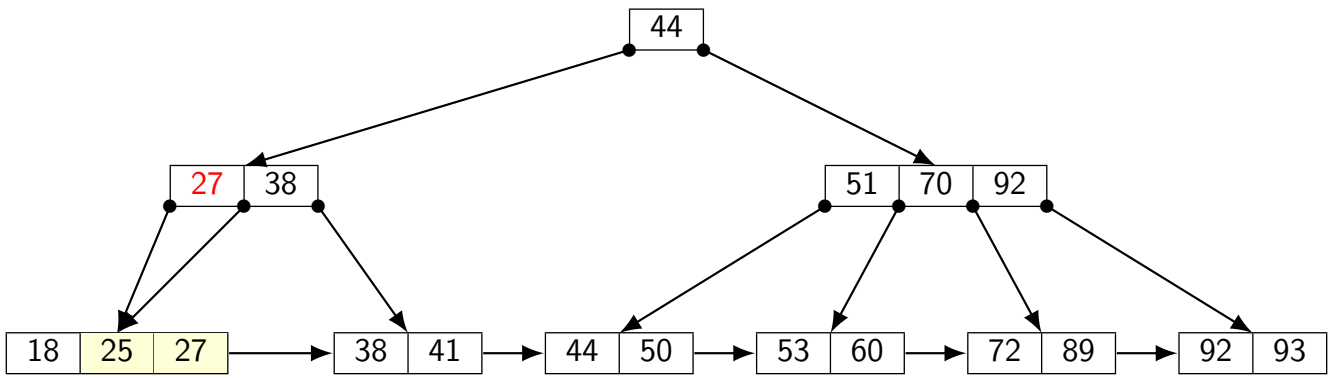


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34

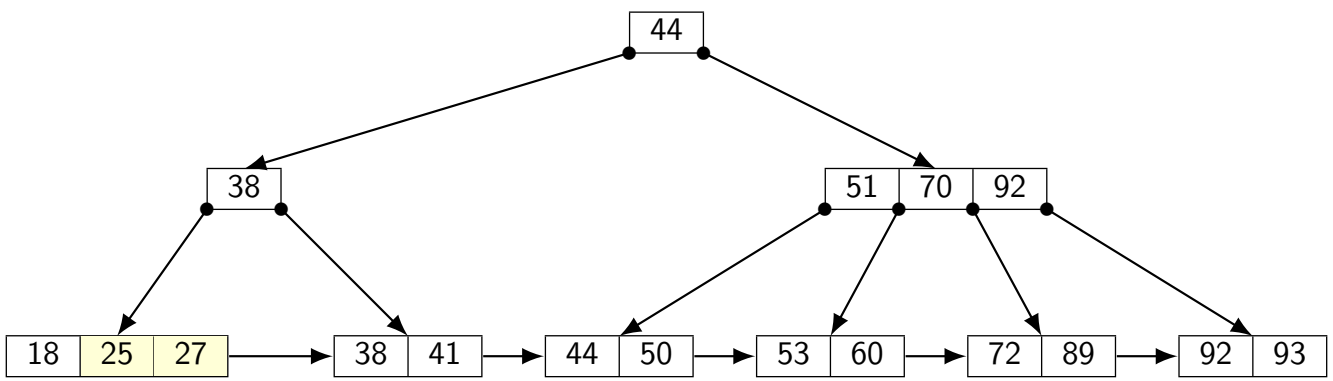


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34

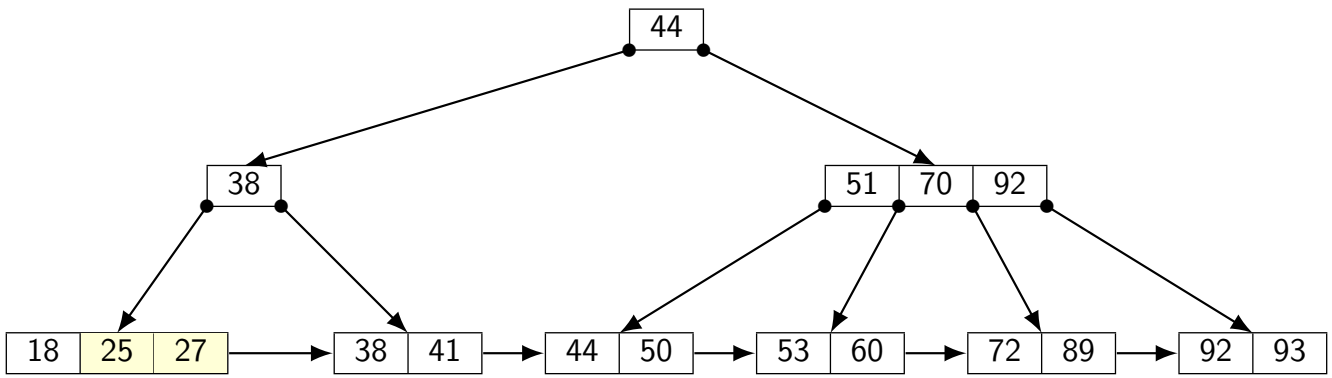


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
  Déplacement de la clef avec son sous-arbre vers le noeud trop vide
  Cas feuille : mise-à-jour de la clef du père
  Cas interne : échange père-fils des deux clefs erronées
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34

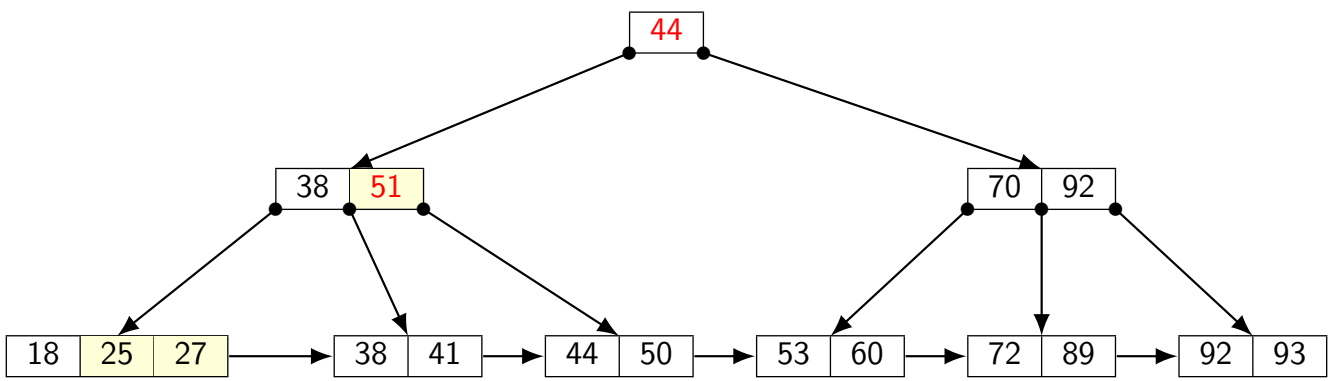


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
  Déplacement de la clef avec son sous-arbre vers le noeud trop vide
  Cas feuille : mise-à-jour de la clef du père
  Cas interne : échange père-fils des deux clefs erronées
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34

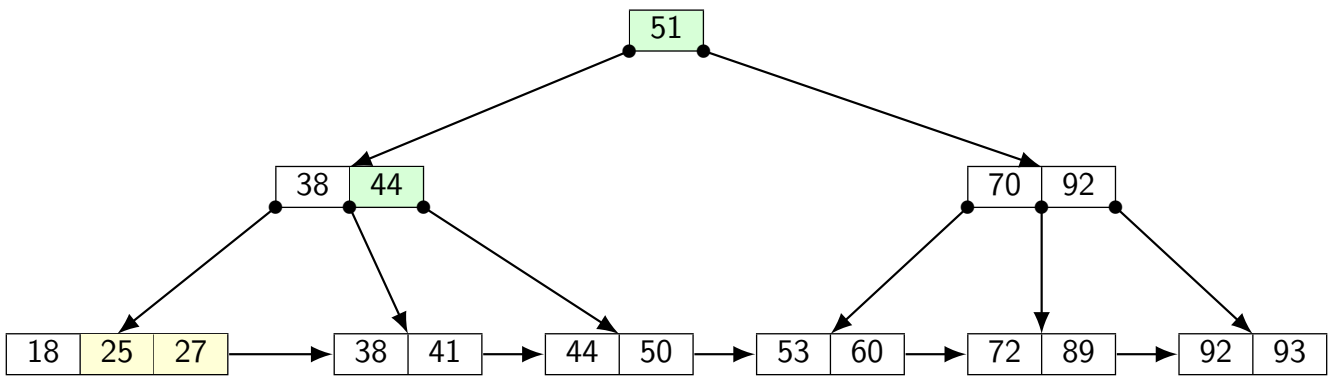


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
  Déplacement de la clef avec son sous-arbre vers le noeud trop vide
  Cas feuille : mise-à-jour de la clef du père
  Cas interne : échange père-fils des deux clefs erronées
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34

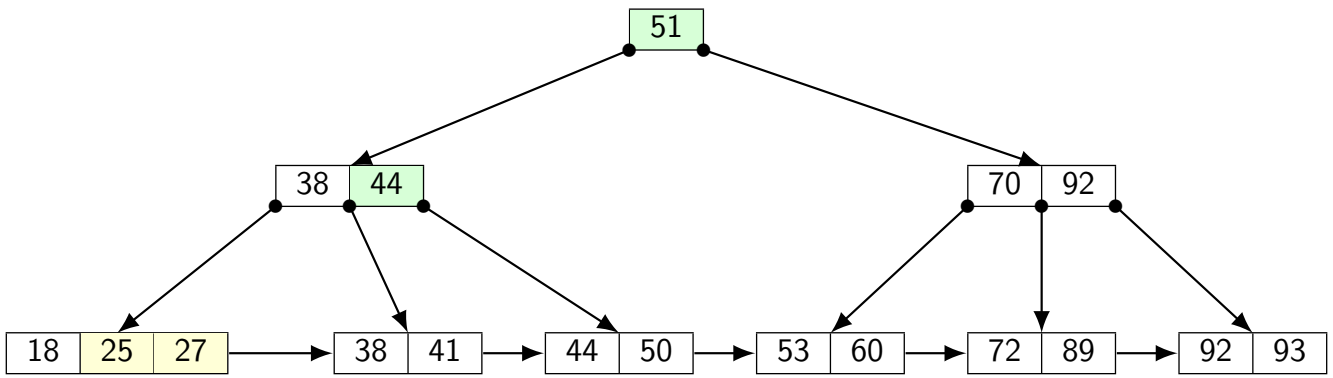


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70

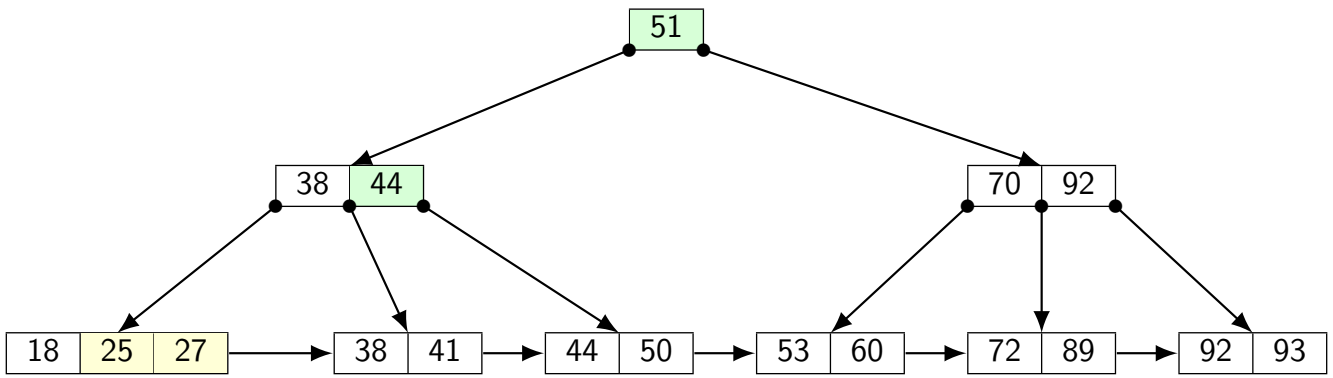


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé)

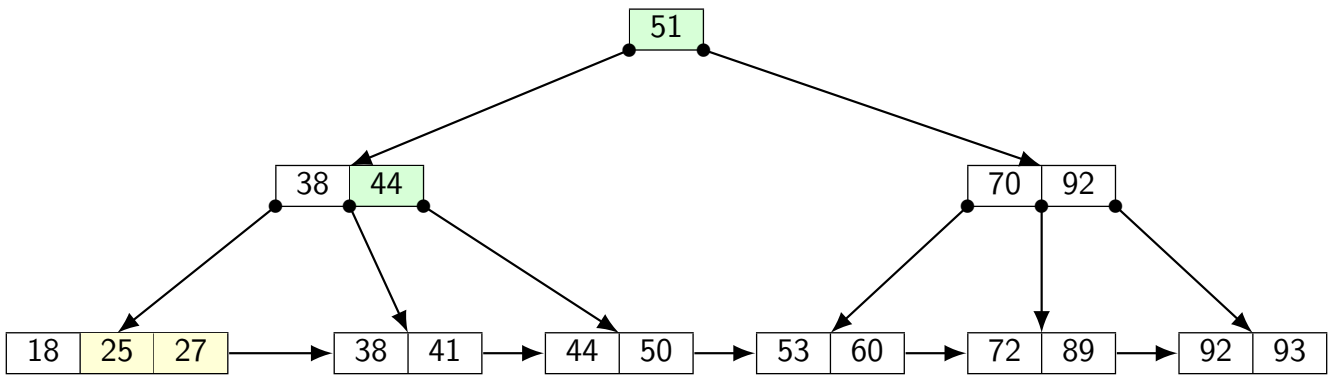


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

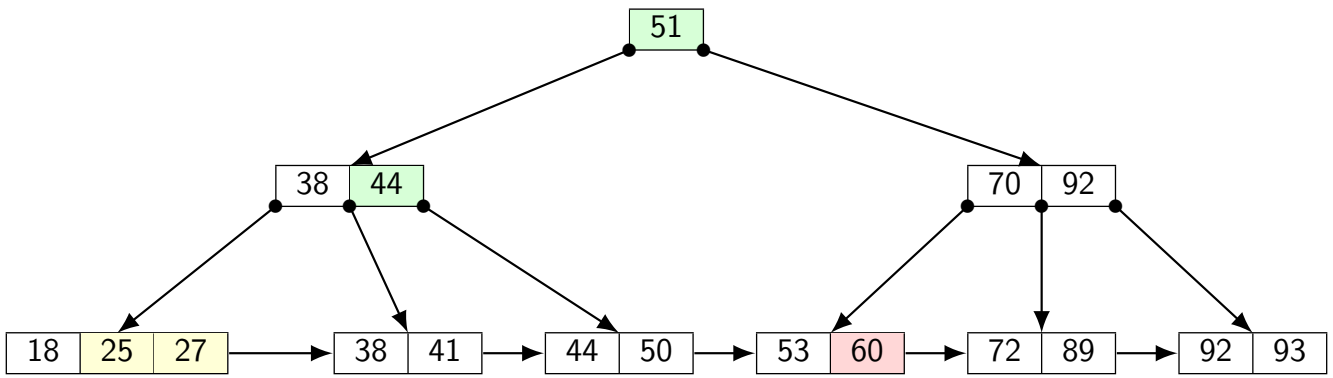


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

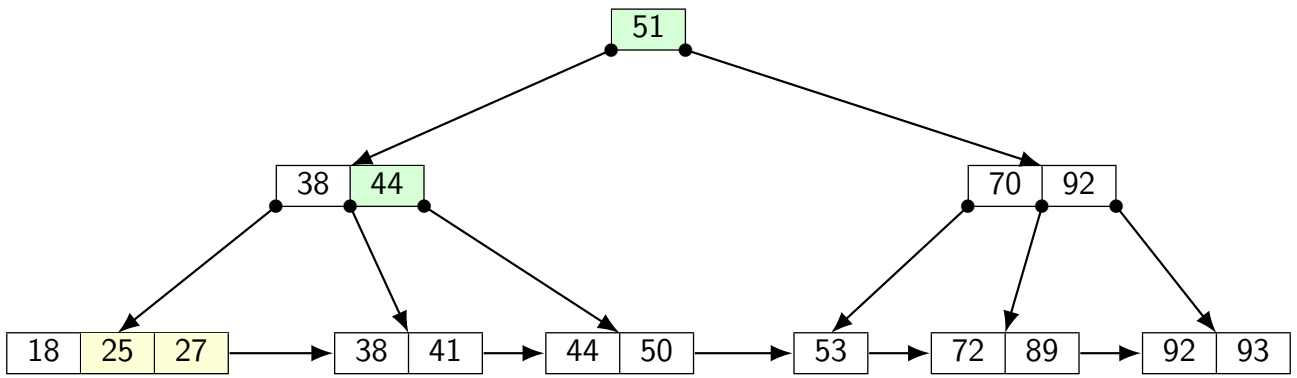


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

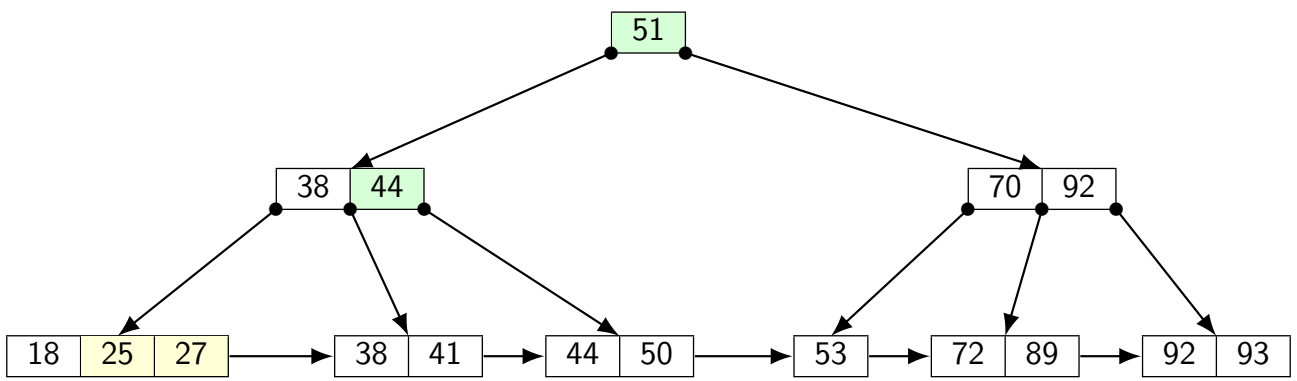


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

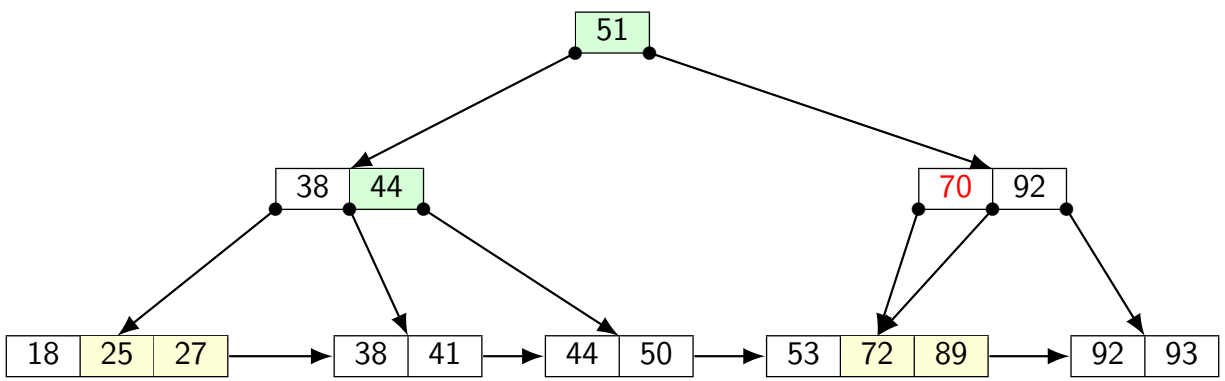


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

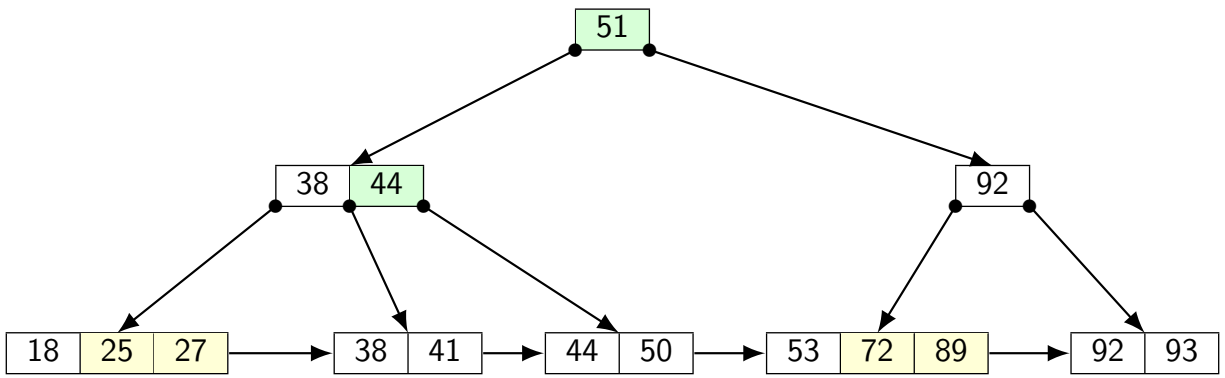


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

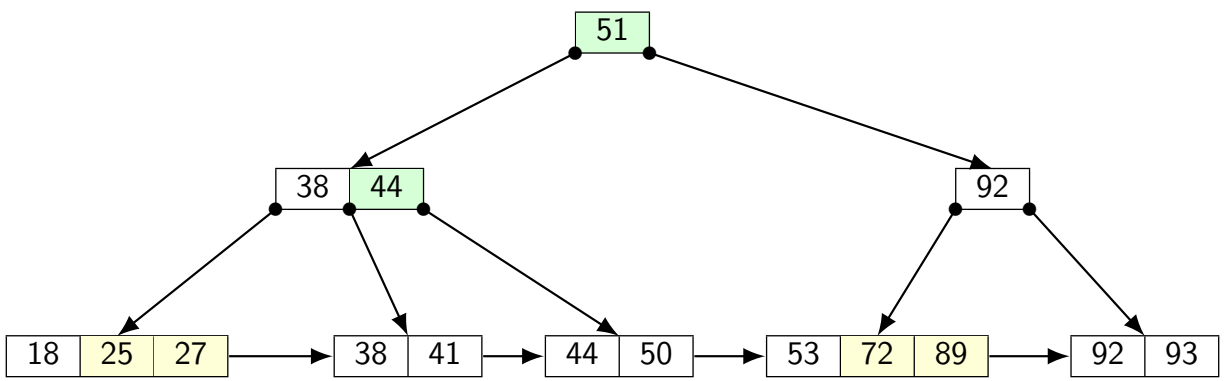


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

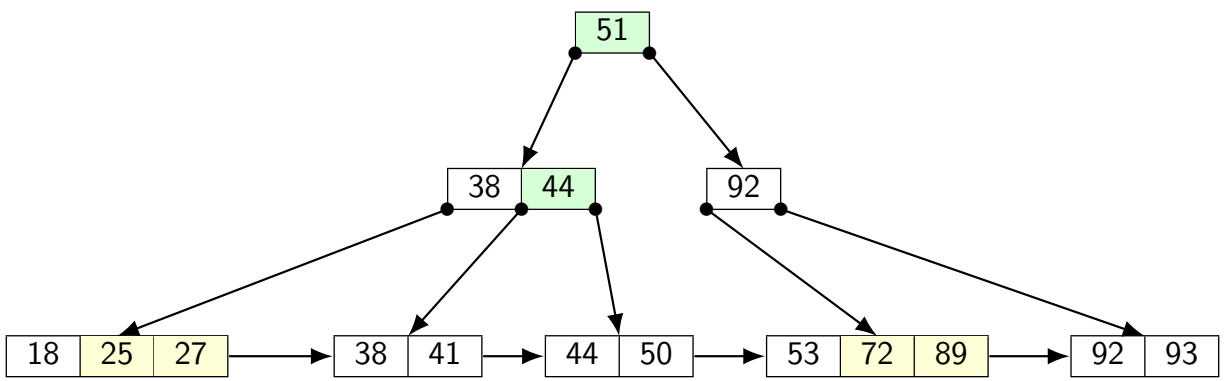


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

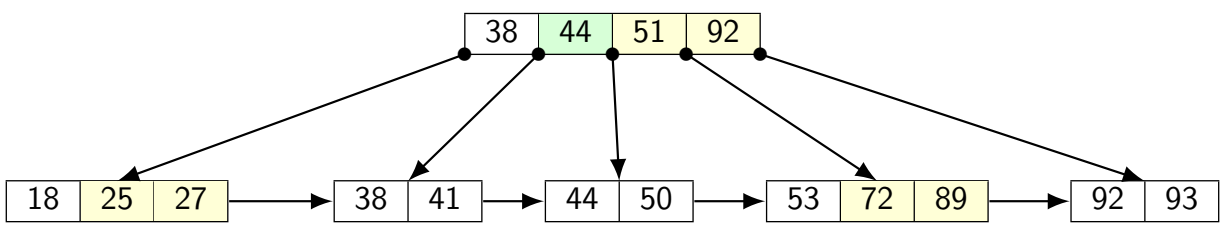


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum
  Split inversé : rassemble le noeud et un frère
  Redescend (cas interne) ou supprime (cas feuille) une clef du père
  Supprime la racine si elle ne contient plus de clefs

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60

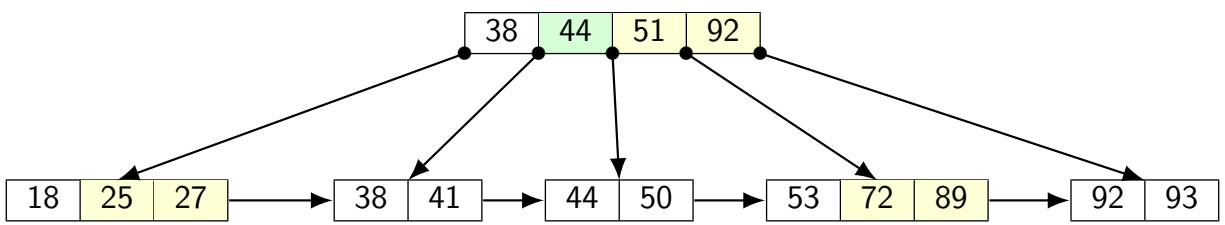


Arbres B^+ : suppression

Suppression d'une clef c :

- Recherche et suppression dans la feuille contenant c
  Si la feuille est trop vide : rééquilibrage par partages et fusions
- Partage (redistribute), si un frère peut se passer d'une clef
- Fusion (merge), si tous les frères sont remplis au minimum

Ex (avec $N = 4$) : Suppression de 19, 13, 34, 70 (non trouvé) et 60



Complexité de la suppression : $O(hauteur)$

Arbres B^+ : performances

Recherche, insertion et suppression en $O(\textit{hauteur})$

(et la constante est toute petite ! 1 pour recherche, < 2 pour insertion et suppression)

Estimation de la hauteur de l'arbre :

Arbres B^+ : performances

Recherche, insertion et suppression en $O(hauteur)$

(et la constante est toute petite ! 1 pour recherche, < 2 pour insertion et suppression)

Estimation de la hauteur de l'arbre :

- Une paire clef-pointeur : 8 octets

Arbres B^+ : performances

Recherche, insertion et suppression en $O(\textit{hauteur})$

(et la constante est toute petite ! 1 pour recherche, < 2 pour insertion et suppression)

Estimation de la hauteur de l'arbre :

- Une paire clef-pointeur : 8 octets
- Taille d'un bloc : 4096 octets

Arbres B^+ : performances

Recherche, insertion et suppression en $O(hauteur)$

(et la constante est toute petite ! 1 pour recherche, < 2 pour insertion et suppression)

Estimation de la hauteur de l'arbre :

- Une paire clef-pointeur : 8 octets
- Taille d'un bloc : 4096 octets
- $N \sim 500$. Remplissage moyen d'un bloc ~ 400 clefs

Arbres B^+ : performances

Recherche, insertion et suppression en $O(hauteur)$

(et la constante est toute petite ! 1 pour recherche, < 2 pour insertion et suppression)

Estimation de la hauteur de l'arbre :

- Une paire clef-pointeur : 8 octets
- Taille d'un bloc : 4096 octets
- $N \sim 500$. Remplissage moyen d'un bloc ~ 400 clefs
- La hauteur de l'arbre est de l'ordre de $\log_{400}(\text{nbr clefs})$

Arbres B^+ : performances

Recherche, insertion et suppression en $O(hauteur)$
(et la constante est toute petite ! 1 pour recherche, < 2 pour insertion et suppression)

Estimation de la hauteur de l'arbre :

- Une paire clef-pointeur : 8 octets
- Taille d'un bloc : 4096 octets
- $N \sim 500$. Remplissage moyen d'un bloc ~ 400 clefs
- La hauteur de l'arbre est de l'ordre de $\log_{400}(\text{nbr clefs})$

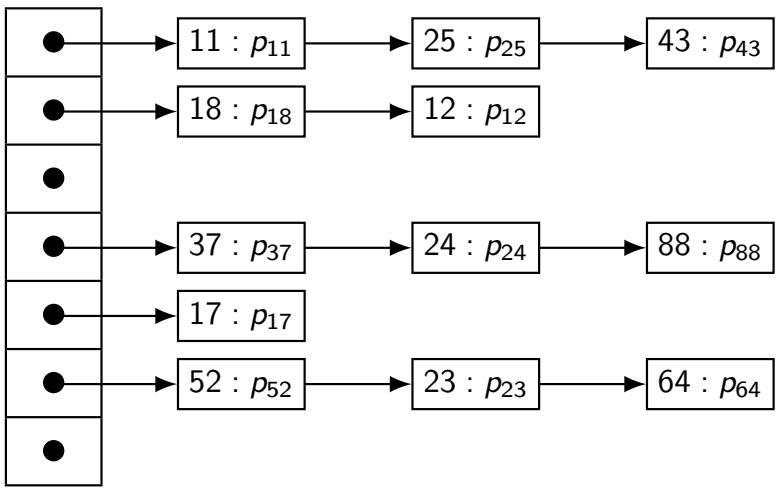
$\Rightarrow$ Un arbre de hauteur 5 suffit pour $400^5 = 10240$ milliards de clefs !

$\Rightarrow$ Complexité "constante" en pratique, "jamais" plus de 8 accès disque

Tables de hachage

Fonctionnement habituel des tables de hachage :

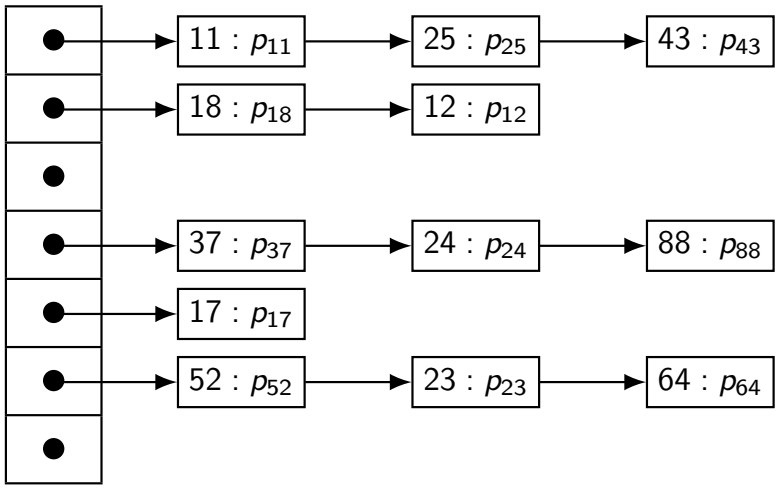
- Tableau T de listes chaînées
- Indexé par l'image d'une fonction de hachage h
- Recherche de c dans $T[h(c)]$ en $O(1)$ en moyenne



Tables de hachage

Fonctionnement habituel des tables de hachage :

- Tableau T de listes chaînées
- Indexé par l'image d'une fonction de hachage h
- Recherche de c dans $T[h(c)]$ en $O(1)$ en moyenne



Mais implémentation adaptée à l'utilisation en mémoire secondaire :

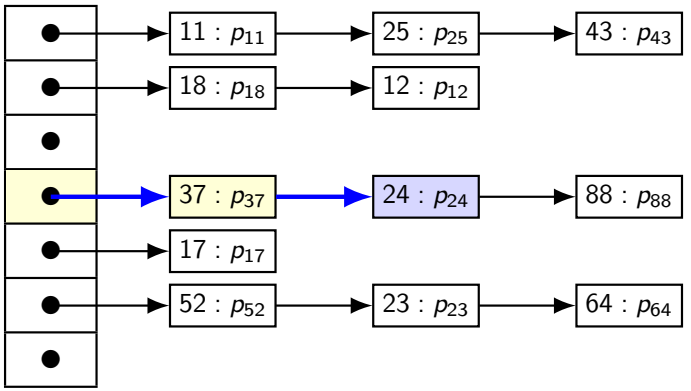
- Listes chaînées de blocs du disque
- Hachage dynamique : techniques pour éviter un rehachage complet $\rightarrow$ hachage linéaire, hachage extensible

Arbres B^+ et hachage en pratique

Tables de hachage

Accès généralement plus efficace pour des recherches ponctuelles

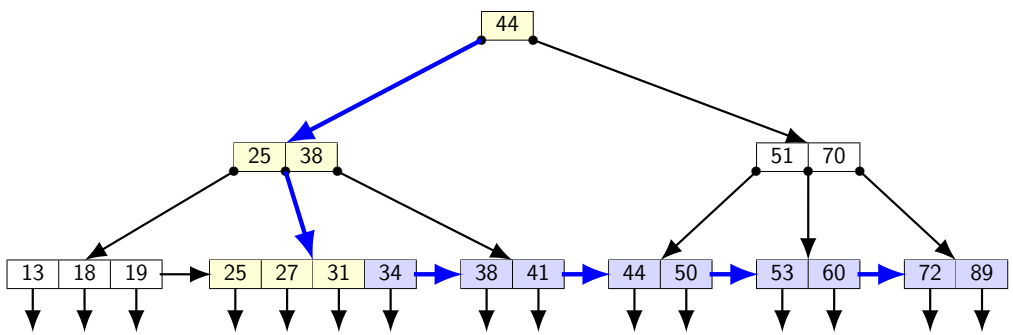
```
SELECT * FROM maTable
WHERE attr = 24;
```



Arbres B^+

Optimisation des lectures séquentielles grace au chaînage des feuilles

```
SELECT * FROM maTable
WHERE attr >= 32;
```

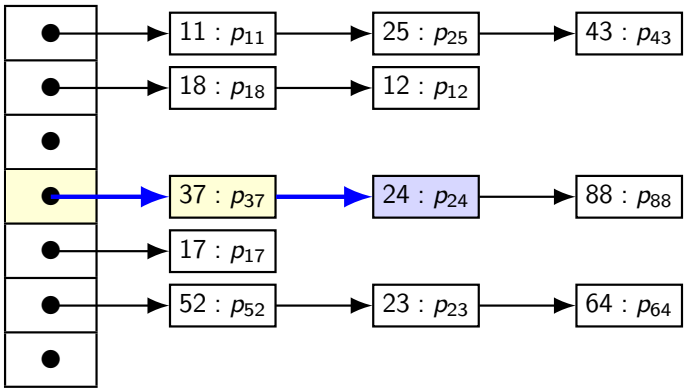


Arbres B^+ et hachage en pratique

Tables de hachage

Accès généralement plus efficace pour des recherches ponctuelles

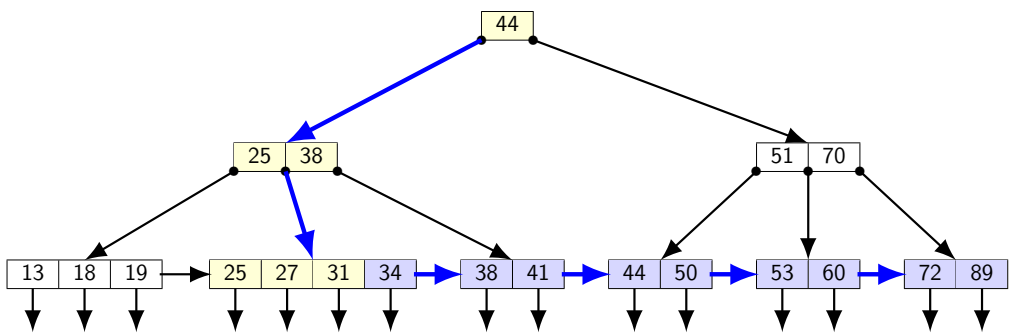
```
SELECT * FROM maTable
WHERE attr = 24;
```



Arbres B^+

Optimisation des lectures séquentielles grace au chaînage des feuilles

```
SELECT * FROM maTable
WHERE attr >= 32;
```



Remarque : les btrees de Postgres sont en réalité des arbres B^+
Les arbres B (sans le +) sont une variante peu utilisée en pratique