B-arbres

Algorithmique et structures de données, 2022-2023

P. Albuquerque (B410), P. Künzli et O. Malaspinas (A401), ISC, HEPIA

2023-05-19

En partie inspirés des supports de cours de P. Albuquerque

Rappel: Les B-arbres

Pourquoi utiliser un B-arbre?

Rappel: Les B-arbres

Pourquoi utiliser un B-arbre?

À quoi ressemble un B-arbre?

Rappel: Les B-arbres

Pourquoi utiliser un B-arbre?

À quoi ressemble un B-arbre?

Qu'est-ce qu'un B-arbre d'ordre n

- Chaque page d'un arbre contient au plus $2 \cdot n$ *clés*;
- Chaque page (excepté la racine) contient au moins n clés;
- Chaque page qui contient m clés contient soit:
  - 0 descendants;
  - $m + 1$ descendants.
- Toutes les pages terminales apparaissent au même niveau.

Rappel: Les B-arbres

Quelques propriétés

- Dans chaque nœud les clés sont **triées**.
- Chaque page contient au plus n nœuds;
- Chaque nœud avec m clés a $m + 1$ descendants;
- Toutes les feuilles apparaissent au même niveau.

Les B-arbres

Structure de données

- Chaque page a une contrainte de remplissage, par rapport à l'ordre de l'arbre;
- Un nœud (page) est composé d'un tableau de clés/pointeurs vers les enfants;

```
P_0 | K_1 | P_1 | K_2 | .. | P_i | K_{i+1} | .. | P_{m-1} | K_m | P_m
```

- `P_0`, ..., `P_m` pointeurs vers enfants;
- `K_1`, ..., `K_m` les clés.
- Il y a `m+1` pointeurs mais `m` clés.
- Comment faire pour gérer l'insertion?

Les B-arbres

Faire un dessin de la structure de données (3min matrix)?

Les B-arbres

Faire un dessin de la structure de données (3min matrix)?

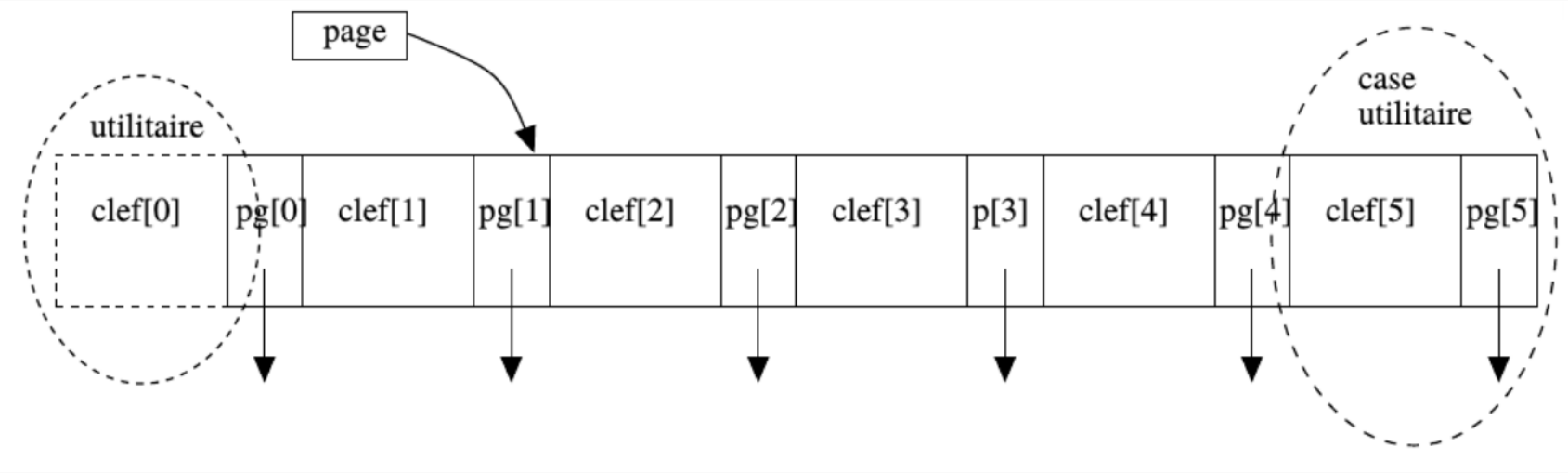
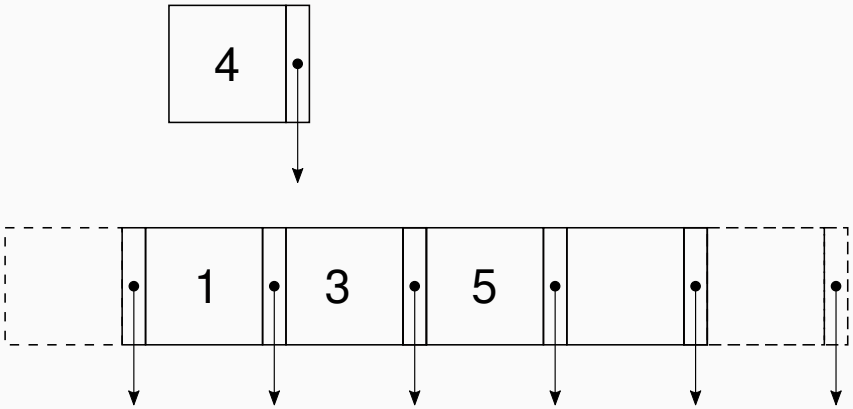


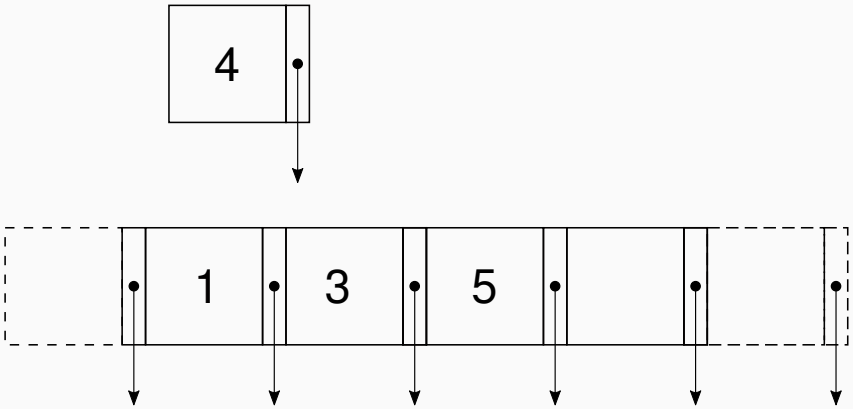
Figure 1: Structure d'une page de B-arbre d'ordre 2.

1. On veut un tableau de `P_i, K_i => struct`;
2. `K_0` va être en "trop";
3. Pour simplifier l'insertion dans une page, on ajoute un élément de plus.

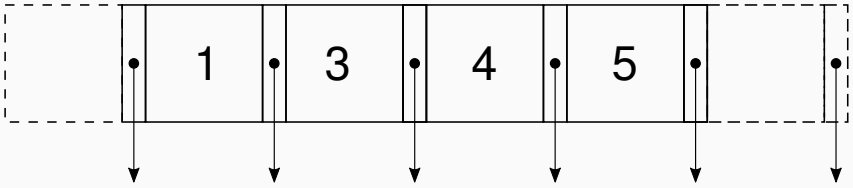
L'insertion cas nœud pas plein, insertion 4?



L'insertion cas nœud pas plein, insertion 4?



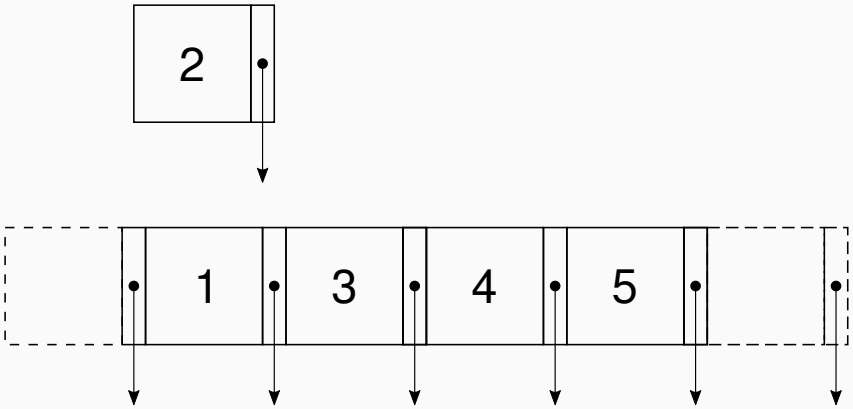
Solution



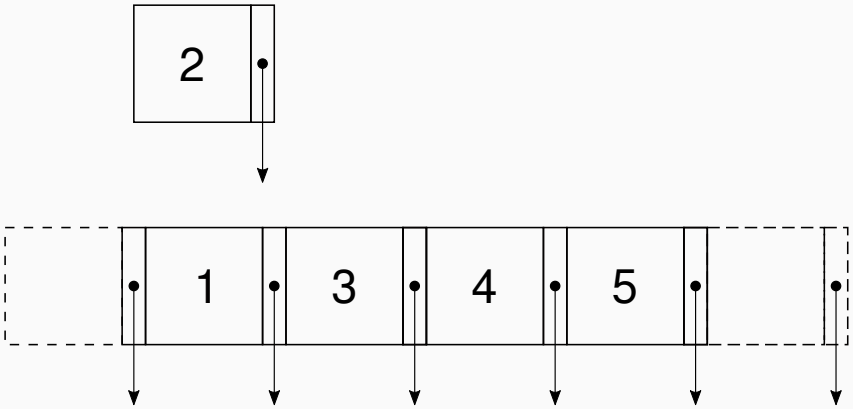
L'insertion cas nœud pas plein, insertion `N`

- On décale les éléments plus grand que `N`;
- On insère `N` dans la place "vide";
- Si la page n'est pas pleine, on a terminé.

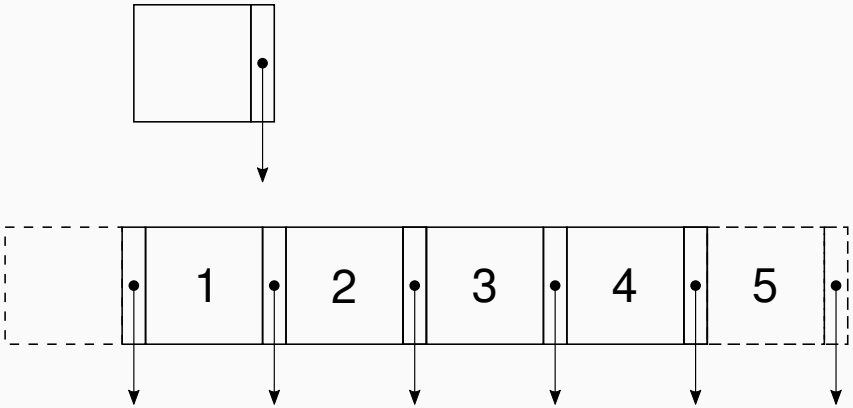
L'insertion cas nœud plein, insertion 2?



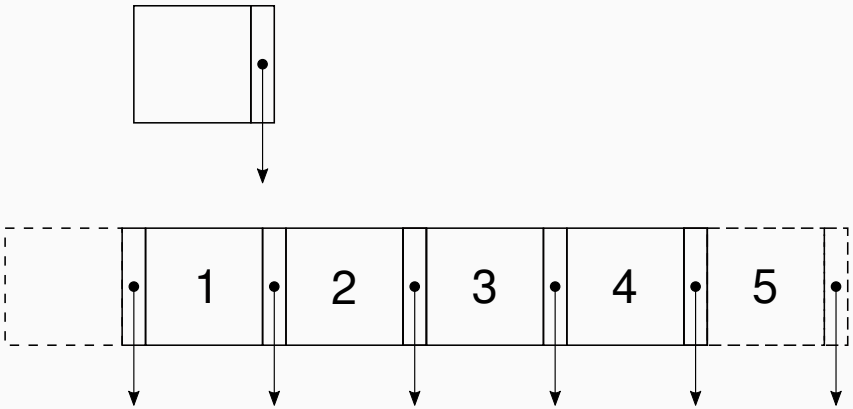
L'insertion cas nœud plein, insertion 2?



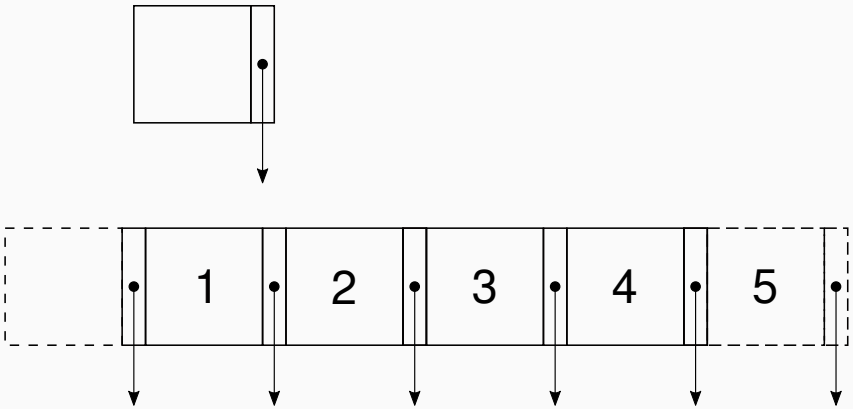
Solution



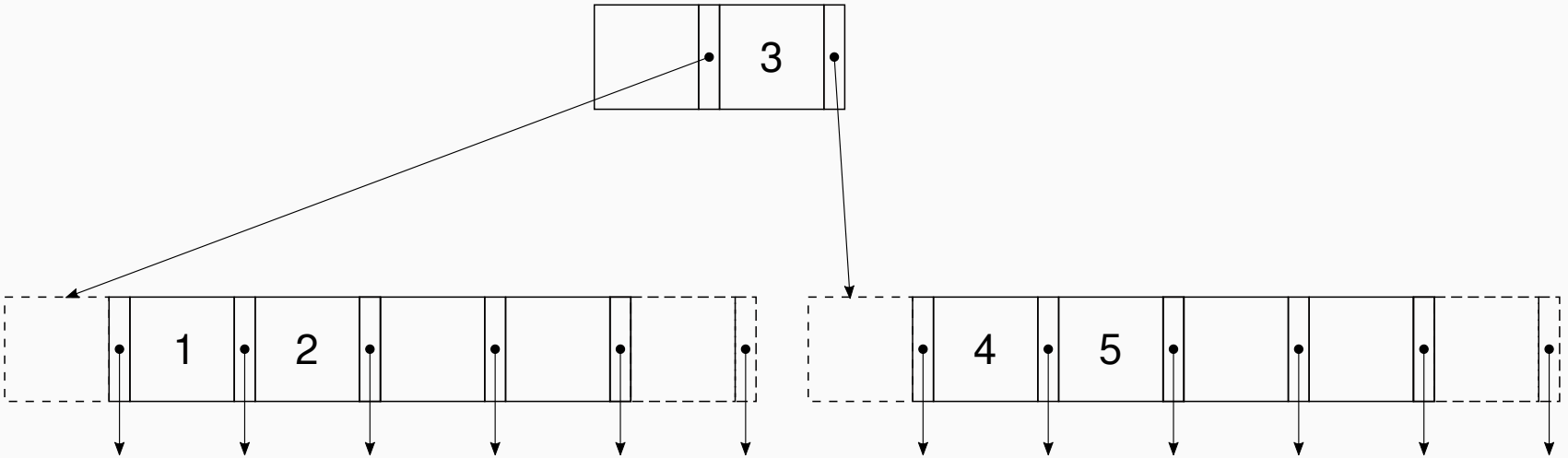
L'insertion cas nœud plein, promotion 3?



L'insertion cas nœud plein, promotion 3?



Solution



L'insertion cas nœud plein, insertion `N`

- On décale les éléments plus grand que `N`;
- On insère `N` dans la place "vide";
- Si la page est pleine:
  - On trouve la valeur médiane `M` de la page (quel indice?);
  - On crée une nouvelle page de droite;
  - On copie les valeur à droite de `M` dans la nouvelle page;
  - On promeut `M` dans la page du dessus;
  - On connecte le pointeur de gauche de `M` et de droite de `M` avec l'ancienne et la nouvelle page respectivement.

Les B-arbres

Pseudo-code structure de données (3min, matrix)?

Les B-arbres

Pseudo-code structure de données (3min, matrix)?

```
struct page
    entier ordre, nb
    element tab[2*ordre + 2]
```

```
struct element
    entier clé
    page pg
```

Les fonctions utilitaires (5min matrix)

```
booléen est_feuille(page)      // la page est elle une feuille?
entier position(page, valeur) // à quelle indice on insère?
booléen est_dans_page(page, valeur) // la valeur est dans la page
```

Les fonctions utilitaires (5min matrix)

```
booléen est_feuille(page)      // la page est elle une feuille?
entier position(page, valeur) // à quelle indice on insère?
booléen est_dans_page(page, valeur) // la valeur est dans la page
```

```
booléen est_feuille(page)
    retourne (page.tab[0].pg == vide)

entier position(page, valeur)
    i = 0
    tant que i < page.nb && val >= page.tab[i+1].clef
        i += 1
    retourne i

booléen est_dans_page(page, valeur)
    i = position(page, valeur)
    retourne (page.nb > 0 && page.tab[i].val == valeur)
```

Les B-arbres

Les fonctions utilitaires (5min matrix)

```
page nouvelle_page(ordre)  // créer une page
rien liberer_memoire(page) // libérer tout un arbre!
```

Les B-arbres

Les fonctions utilitaires (5min matrix)

```
page nouvelle_page(ordre)  // créer une page
rien liberer_memoire(page) // libérer tout un arbre!
```

```
page nouvelle_page(ordre)
    page = allouer(page)
    page.ordre = ordre
    page.nb = 0
    page.tab = allouer(2*ordre+2)
    retourner page

rien liberer_memoire(page)
    si est_feuille(page)
        liberer(page.tab)
        liberer(page)
    sinon
        pour fille dans page.tab
            liberer_memoire(fille)
        liberer(page.tab)
        liberer(page)
```

Les B-arbres

Les fonctions (5min matrix)

```
page recherche(page, valeur) // retourner la page contenant
                             // la valeur ou vide
```

Les fonctions (5min matrix)

```
page recherche(page, valeur) // retourner la page contenant
                             // la valeur ou vide
```

```
page recherche(page, valeur)
    si est_dans_page(page, valeur)
        retourne page
    sinon si est_feuille(page)
        retourne vide
    sinon
        recherche(page.tab[position(page, valeur) - 1], valeur)
```

Les fonctions

```
page inserer_valeur(page, valeur) // insérer une valeur
```

Les fonctions

```
page inserer_valeur(page, valeur) // insérer une valeur
```

```
page inserer_valeur(page, valeur)
    element = nouvel_element(valeur)
    // ici élément est modifié pour savoir
    // s'il faut le remonter
    inserer_element(page, element)
    si element.page != vide && page.nb > 2*page.ordre
        // si on atteint le sommet!
        page = ajouter_niveau(page, element)
    retourne page
```

Les B-arbres

Les fonctions

```
rien inserer_element(page, element) // insérer un element
                                    // et voir s'il remonte
```

Les fonctions

```
rien inserer_element(page, element) // insérer un element
                                    // et voir s'il remonte
```

```
rien inserer_element(page, element)
    si est_feuille(page)
        placer(page, element)
    sinon
        sous_page = page.tab[position(page, element.clé) - 1].pa
        inserer_element(sous_page, element)
        // un element a été promu
        si element.page != vide
            placer(page, element)
```

Les B-arbres

Les fonctions (5min matrix)

```
rien placer(page, element) // inserer un élément
```

Les B-arbres

Les fonctions (5min matrix)

```
rien placer(page, element) // inserer un élément
```

```
rien placer(page, element)
    pos = position(page, element.clé)
    pour i de 2*page.ordre à pos+1
        page.tab[i+1] = page.tab[i]
    page.tab[pos+1] = element
    page.nb += 1
    si page.nb > 2*page.ordre
        scinder(page, element)
```

Les B-arbres

Les fonctions (5min matrix)

```
rien scinder(page, element) // casser une page et remonter
```

Les fonctions (5min matrix)

```
rien scinder(page, element) // casser une page et remonter
```

```
rien scinder(page, element)
    nouvelle_page = nouvelle_page(page.ordre)
    nouvelle_page.nb = page.ordre
    pour i de 0 à ordre inclu
        nouvelle_page.tab[i] = page.tab[i+ordre+1]
    element.clé = page.tab[ordre+1].clé
    element.page = nouvelle_page
```

Les fonctions (5min matrix)

```
page ajouter_niveau(page, element) // si on remonte à la
                                   // racine, on doit créer
                                   // une nouvelle racine
```

Les fonctions (5min matrix)

```
page ajouter_niveau(page, element) // si on remonte à la
                                   // racine, on doit créer
                                   // une nouvelle racine
```

```
page ajouter_niveau(page, element)
    tmp = nouvelle_page(page.ordre)
    tmp.tab[0].page = page
    tmp.tab[1].clé = element.clé
    tmp.tab[1].page = element.page
    retourne tmp
```