B-arbres

Algorithmique et programmation séquentielle

1 Buts

- Mise en place d'une structure de B-arbre.
- Implémentation de l'insertion, de la recherche, et de la suppression dans un B-arbre
- Utilisation de la récursivité.

2 Énoncé

Écrire un programme qui implémente les opérations d'insertion, de recherche et d'affichage d'un B-arbre. Les clés stockées sont des entiers. Une page d'un B-arbre est constituée d'un tableau de cases contenant une clé et un pointeur sur une page enfant, et d'un pointeur sur la page enfant toute à gauche. La structure de page proposée prévoit deux cases supplémentaires (voir fig. 1) qui sont justes utilisées pour l'implémentation de l'insertion.

```
typedef struct element {
    int clef;        //information stockée
    struct page* pg; //pointeur sur une page
} element;

typedef struct page {
    int ordre; //ordre de la page
    int nb;    //compteur de clés dans la page
    element* tab;
} page;
```

3 Cahier des charges

Votre module de manipulation de B-arbre (fichiers `b_arbre.h` et `b_arbre.c`) doit implémenter la structure définie précédemment et comporter plusieurs fonctions.

1. Implémenter une fonction

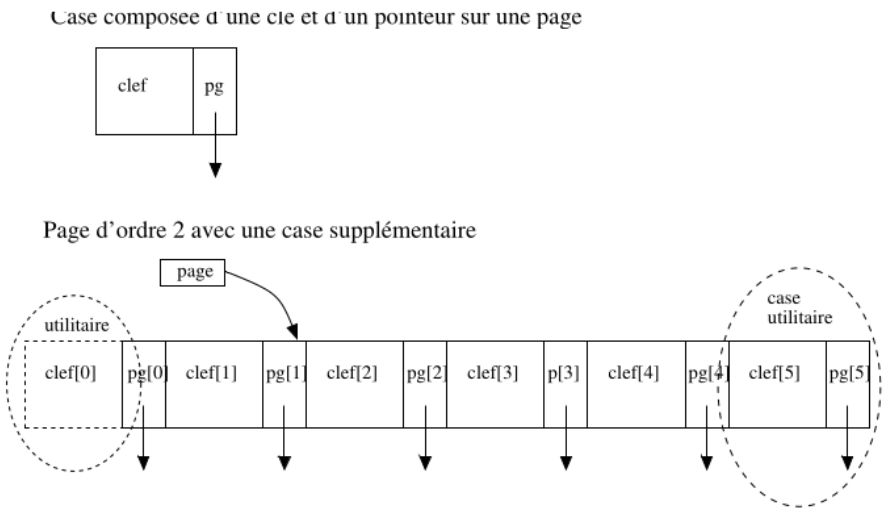

Figure 1: Structure d'une page (B-arbre d'ordre 2)

```
page* new_page(int ordre);
```

qui alloue dynamiquement une page dont l'ordre est passé en paramètre.

2. Implémenter une fonction

   ```
   page* inserer(page* b_arbre, int clef);
   ```

   qui insère dans un B-arbre une clé qui ne s'y trouve pas encore (pas de clé à double dans un B-arbre).

   Pour que le code soit bien structuré, on suggère d'écrire :

   - Une fonction

     ```
     int position(page* pg, int clef);
     ```

     qui retourne la position où insérer une clé dans une page. Utiliser une recherche dichotomique.

   - Une fonction

     ```
     int placer(page* pg, element* cell);
     ```

     qui place une case dans une page. Elle retourne la valeur `1`, si cette clé n'est pas déjà présente dans la page, et 0 sinon. Cette fonction trouve la position où insérer la clef dans la page, décale les cases depuis cette position, y insère la case et incrémente le compteur de clés de la page. Si le nombre de clés dans la page dépasse `2*ordre`,

alors on crée une nouvelle page contenant les cases d'indices `ordre+1` à `2*ordre` sur laquelle pointe
`cell->pg`, et on stocke la clé du milieu de page dans `cell->clef`; sinon `cell->pg` est simplement mise à `NULL` (voir fig. 2 et fig. 3).

- Une fonction

  ```
  page* inserer_case(page* b_arbre, element* cell, int depth);
  ```

  qui insère une case dans un B-arbre. Cette fonction descend récursivement jusqu'à une feuille dans laquelle on place la case. Le paramètre `depth` permet de savoir à quelle profondeur on est dans l'arbre lors de la récursion. En remontant le chemin d'insertion, s'il y a eu division d'une page (c.-à-d. si `cell->pg` est non nul), on place la paire valeur promue / nouvelle page (paire stockée par cell) dans la page parente. Lorsque depth égal `0`, on est dans la page au sommet du B-arbre. Finalement, la fonction inserer se résume essentiellement à un appel à `inserer_case()`.

3. Implémenter une fonction `search()` de recherche d'une clé dans un B-arbre. Si la clé recherchée s'y trouve, elle renvoie un pointeur sur la page contenant la clé; sinon elle renvoie `NULL`.
4. Écrire une fonction `display_RGD()` qui affiche un B-arbre en effectuant un parcours de type RGD. Chaque page est imprimée sur une ligne avec un décalage d'autant plus grand que la page est profonde dans l'arbre.
5. Écrire une fonction `display_GRD()` qui affiche un B-arbre en effectuant un parcours de type GRD. Les clés sont alors imprimées à l'écran par ordre croissant. Afficher une valeur par ligne.
6. Une fonction

   ```
   void free_b_arbre(page* b_arbre);
   ```

   qui libère la mémoire allouée pour la création d'un B-arbre. Ceci nécessite d'effectuer un parcours de type GDR du B-arbre.
7. Lorsque toutes les fonctions précédentes ont été réalisées, alors implémenter une fonction `delete()` pour supprimer une clé dans un B-arbre.

4 Format d'entrées/sorties

Votre programme devra pouvoir être lancé avec la syntaxe suivante :

`./test_b_arbre <ordre> <liste_de_valeurs> <operation> <parametre>`

où les arguments sont les suivants :

- `ordre` spécifie l'ordre du B-arbre ;
- `liste_de_valeurs` donne la liste des valeurs à insérer dans le B-arbre ;

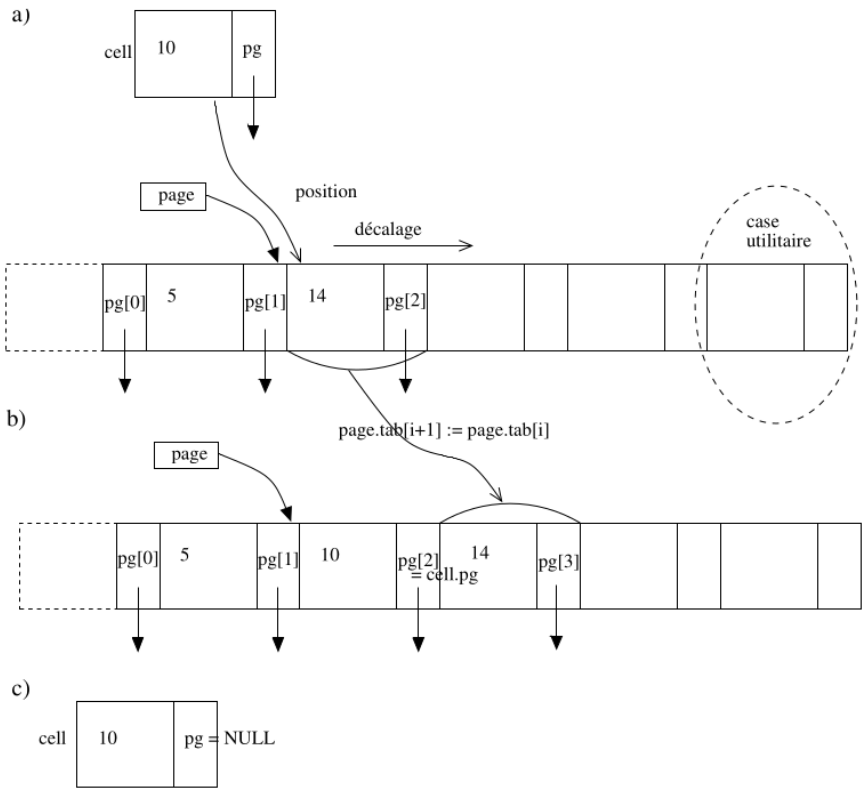

Figure 2: Placement d'une case dans une page pas pleine (B-arbre d'ordre 2).

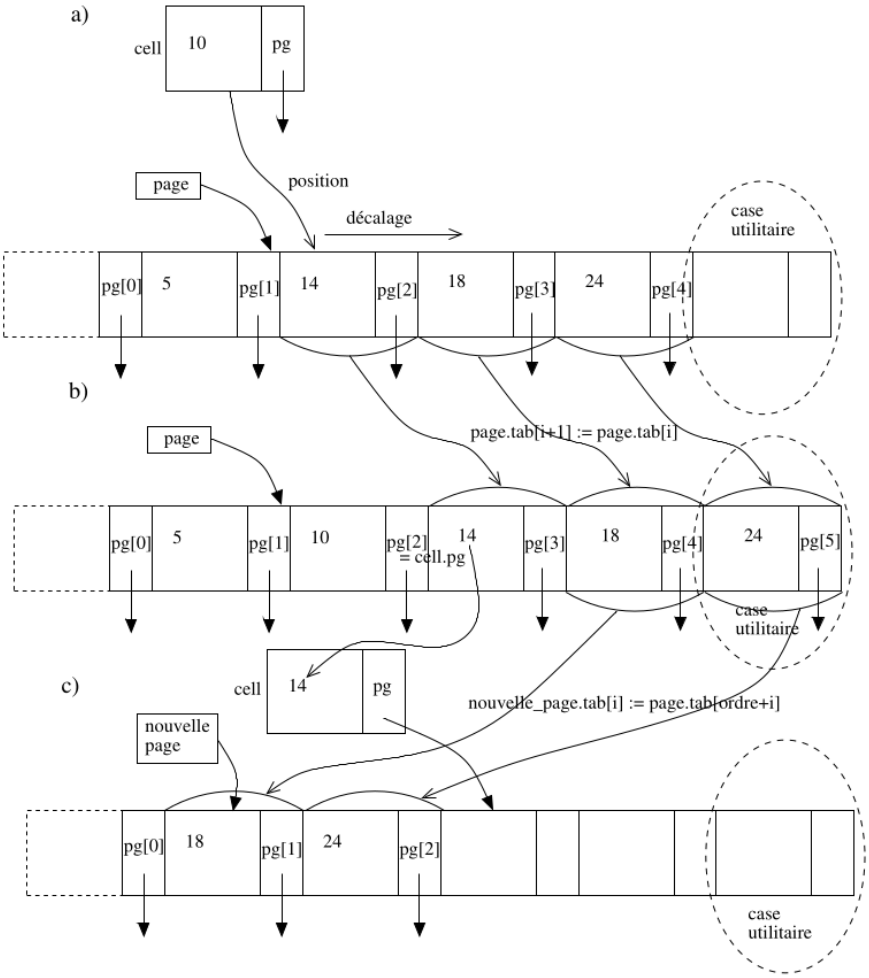

Figure 3: Placement d'une case dans une page pleine (B-arbre d'ordre 2).

- `operation` spécidfie l'opération à effectuer sur le B-arbre (`display`, `search`) ;
- `parametre` est l'argument pour l'opération à effectuer (GRD ou RGD pour `display`, la valeur à chercher pour `search`).

Par exemple, la commande :

```
./test_b_arbre 2 4 7 9 12 20 13 100 -12 -5 -6 50 60 10 15 14 29 search 5
```

doit produire la sortie `0` à l'écran, car la valeur est absente du B-arbre (`1` sinon).

La commande

```
./test_b_arbre 2 4 7 9 12 20 13 100 -12 -5 17 66 -6 50 60 10 15 14 29 display
RGD
```

doit produire la sortie à l'écran :

```
    13
        -5  9
                -12 -6
                4   7
                10  12
        16  60
                14  15
                20  29  50
                66  100
```

et la commande

```
./test_b_arbre 2 4 7 9 20 13 -12 -5 60 10 15 29 display GRD
```

doit produire la sortie à l'écran :

```
-12
-5
4
7
9
10
13
15
20
29
60
```

Les commandes précédentes peuvent aussi être lancées avec la syntaxe :

```
echo 2 4 7 9 20 13 100 -12 -5 60 10 15 29 display GRD | xargs ./test_b_arbre
```

ou, si les arguments sont placés dans un fichier `input.dat`, avec

```
cat input.dat | xargs ./test_b_arbre
```

5 Fichiers offerts

Pour vous aider nous mettons à votre disposition les deux fichiers suivants b_arbre.h et b_arbre_skel.c.