Arbres-B (B-trees)

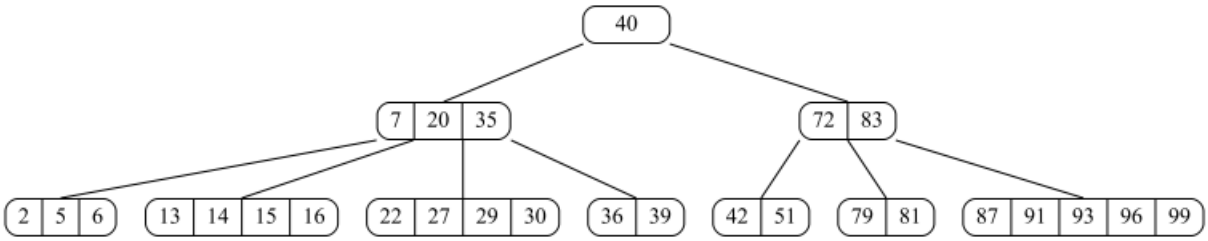

Figure 1 – B-arbre ?

1 Préliminaires

Exercice 1.1 (Arbres : de recherche, B, B+, 2-3-4. . .)

Qu'est-ce ? A quoi ça sert ?

1. Arbre (général) de recherche (M-way search tree or (general) search tree)
2. Arbre B (B-tree), Arbre B+ (B+ tree)
3. Arbre 2-3-4 (2-4 tree)

Exercice 1.2 (Implémentation des B-arbres)

1. Quelles sont les informations nécessaires à la représentation d'un B-arbre ?
2. Parmis les implémentations des arbres généraux laquelle est la plus appropriée pour les B-arbres ? Quelles sont les modifications à apporter à l'implémentation choisie.

Exercice 1.3 (En ordre)

Écrire une fonction qui construit une liste des clés d'un B-arbre en ordre croissant.

Exercice 1.4 (Représentation linéaire)

Rappel : Un arbre général $A =< o, A_1, A_2, ..., A_n >$ peut être représenté par $(o\ A_1\ A_2\ ...\ A_n)$.
Pour les B-arbres, un nœud o est représenté par la liste de ses clés : $< x_1, ..., x_{k-1} >$.

1. Donner la représentation linéaire de l'arbre de la figure 2.
2. Dessiner l'arbre : "$(< 13, 32, 44 > (< 3 >)(< 18, 25 >)(< 35, 40 >)(< 46, 49, 50 >))$"
3. Écrire la fonction qui construit à partir d'un B-arbre sa représentation linéaire (sous forme de chaîne de caractères).
4. Bonus :Écrire la fonction réciproque : qui construit le B-arbre à partir de sa représentation linéaire.

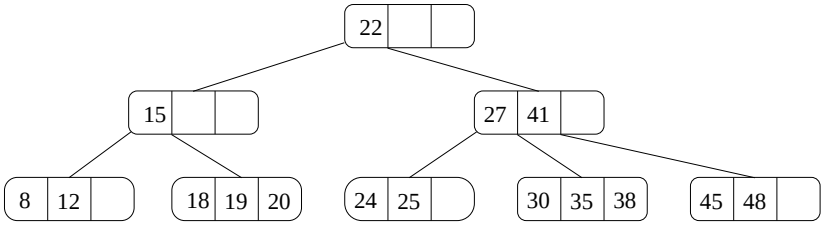

Figure 2 – B-arbre ?

2 Classiques

Pour simplifier les explications : dans un nœud, pour chaque clé n° i on nommera *fils gauche* le fils n° i, et *fils droit* le fils n° $i+1$.

Exercice 2.1 (Minimum et maximum)

1. Où se trouvent la valeur minimum et la valeur maximum d'un B-arbre ?
2. Écrire les deux fonctions permettant de récupérer la valeur minimum (resp. maximum) d'un B-arbre non vide.

Exercice 2.2 (Recherche)

Écrire une fonction qui recherche une valeur x dans un B-arbre. La fonction retourne le couple `(B, i)` tel que `B.keys[i] == x` si la recherche est positive, la valeur `None` sinon.

Exercice 2.3 (Inserer un nouvel élément : la méthode classique)

1. (a) Où peut-on insérer un nouvel élément dans un B-arbre afin qu'il conserve ses propriétés ?
   (b) Quel problème cela pose ?
   (c) Quelle transformation peut être appliquée à l'arbre pour le résoudre ? Dans quelles conditions peut-on effectuer cette transformation ?
   (d) Écrire la fonction réalisant cette transformation dans le *cas idéal*[1].
2. On peut utiliser cette transformation de deux manières : à la descente (*principe de précaution*) ou à la remontée.
   Pour chacune de ces méthodes :
   (a) Donner le principe d'insertion.
   (b) Construire un nouvel arbre 2-3-4 par insertions successives des éléments suivants : 5, 15, 40, 25, 18, 45, 38, 42, 9.
3. Écrire la fonction d'insertion dans un B-arbre en appliquant le principe de précaution.

Exercice 2.4 (Suppression d'un élément : à la descente)

1. Comment supprimer une clé lorsque l'on n'est pas en feuille ? Utiliser comme exemple la suppression de la valeur 27 dans l'arbre de la figure 2 (on préférera ici systématiquement le coté gauche, mais ce choix pourra être remis en cause plus tard !).
2. (a) Quel problème pose la suppression de la valeur 24 dans l'arbre obtenu ? Quelle transformation permet de résoudre le problème (s'inspirer des transformations sur les AVL) ? Supprimer cette valeur, puis la valeur 25, en utilisant cette nouvelle transformation.
   (b) Écrire les deux fonctions de cette transformation (à gauche, à droite), en précisant les conditions d'appels.
3. (a) Quel problème pose la suppression de la valeur 35 dans le dernier arbre ? Quelle nouvelle transformation faut-il appliquer ?
   (b) Écrire la fonction de la nouvelle transformation, en précisant les conditions d'appel.
4. En appliquant un principe de précaution similaire à celui vu pour l'insertion, supprimer successivement les valeurs 15, 22, 8, 20, 30, 18, 45, 12, 48, 19, 41. Essayer de limiter les destructions de nœuds.
   Construire en même temps le principe de la suppression d'un élément dans un B-arbre.
5. Écrire la fonction de suppression.

1. Pas la racine de l'arbre, pas de soucis de "place".

3 From previous midterms

Exercice 3.1 (B-arbres et mystère – *3 points*)

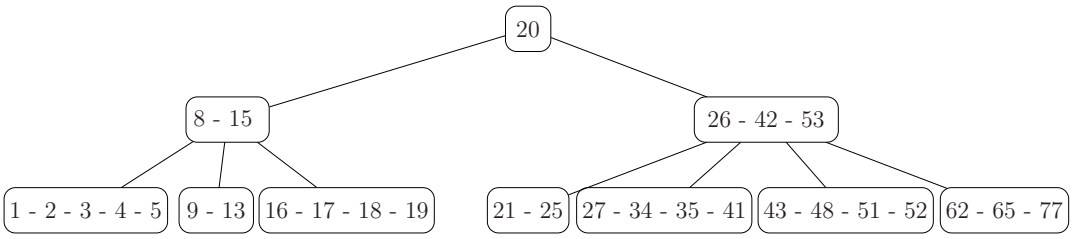

FIGURE 3 – B-arbre B_1

```
def mystery(B, a, b):
    i = 0
    while i < B.nbKeys and B.keys[i] < a:
        i += 1
    c = 0
    if B.children == []:
        while i < B.nbKeys and b > B.keys[i]:
            i += 1
            c += 1
    else:
        c += mystery(B.children[i], a, b)
        while i < B.nbKeys and b > B.keys[i]:
            c += mystery(B.children[i+1], a, b) + 1
            i += 1
    return c
```

1. Pour chacun des appels suivants, avec B_1 l'arbre de la figure 3 :
   — quel est le résultat retourné ?
   — combien d'appels à `mystery` ont été effectués ?

   (a) `mystery(`B_1`, 1, 77)`

   (b) `mystery(`B_1`, 10, 30)`

2. Soient `B` un B-arbre non vide contenant des entiers, et `a` et `b` deux valeurs entières telles que `a` $<$ `b`.
   Que calcule la fonction `mystery(B, a, b)` ?

Exercice 3.2 (Au suivant – *5 points*)

Écrire la fonction `findNext(B, x)` qui retourne la clé de `B` immédiatement supérieure à `x`. La fonction renvoie `None` si une telle valeur n'existe pas.

Dans l'arbre de la figure 1 :

— `findNext(B, 5)` retourne 6

— `findNext(B, 6)` retourne 7

— `findNext(B, 7)` retourne 13

— `findNext(B, 40)` retourne 42

— `findNext(B, 41)` retourne 42

— `findNext(B, 99)` retourne `None`

Bonus

Exercice 3.3 (Insertion, une nouvelle méthode : pour changer un peu. . .)

L'insertion vue ci-dessus entraîne quelquefois des modifications de la structure de l'arbre inutiles. Le but ici est de trouver d'autres transformations qui permettent l'insertion sans pour autant augmenter le nombre de nœuds de l'arbre.

1. Une nouvelle transformation ?
   (a) Comment éviter d'effectuer un éclatement lors de l'insertion de l'avant-dernier élément (42) dans l'exemple de l'exercice 2.3 ?
   (b) Utiliser la même méthode pour ajouter successivement les valeurs 33 et 36 dans l'arbre de la figure 2 (préférer le côté gauche). Peut-on utiliser la même méthode pour insérer la valeur 42 ?
2. L'ajout :
   (a) Ajouter à l'arbre obtenu les valeurs 42 et 16. Préciser les conditions dans lesquelles on pourra utiliser les transformations pour une insertion.
   (b) Ajouter enfin les valeur 37 puis 23. Doit-on utiliser le principe de précaution ?
   (c) En déduire le nouveau principe de l'algorithme d'insertion.
   (d) Écrire la fonction d'insertion.

Exercice 3.4 (Suppression d'un élément : à la remontée !)

La fonction de suppression utilisant le principe de précaution est trop coûteuse en modifications : par exemple, la suppression d'une clé qui n'existe pas dans le B-arbre peut quand même modifier l'arbre.

Ici la suppression se fera sans utiliser le principe de précaution, c'est à dire en effectuant les modifications seulement lorsque cela est nécessaire : à la remontée.

1. On reprend ici les suppressions effectuées à l'exercice 2.4 sur l'arbre de la figure 2, mais cette fois-ci elles seront effectuées avec modifications à la remontée.
   (a) Commencer par supprimer 27, 24, 25, 35. Des remarques ?
   (b) Continuer les suppressions : 15, 22, 8, 20, 30, 18, 45, 12, 48, 19, 41.
2. En déduire le principe de la suppression d'un élément dans un B-arbre avec modifications à la remontée.
3. Écrire cette fonction de suppression.