B-trees (Arbres-B)

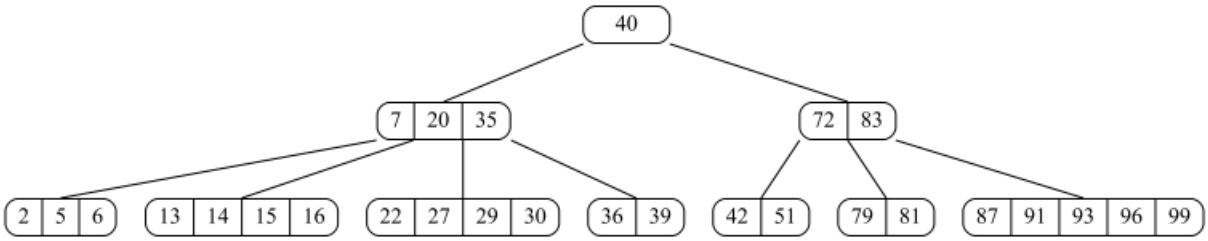

Figure 1: B-tree?

1 Preliminaries

Exercise 1.1 (Trees: search ones, B-trees, B+, 2-3-4...)

What is it? What that's for?

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

Exercise 1.2 (B-trees implementation)

1. What are the informations needed to represent a B-tree?
2. Among the implementations of general trees, which one is the more suitable for B-trees? What modifications have to be done on the chosen implementation?

Exercise 1.3 (B-trees: Linear Representation – *Midterm S3# - 2018*)

Remaining: A general tree $A =< o, A_1, A_2, ..., A_n >$ can be represented by $(o\ A_1\ A_2\ ...\ A_n)$
With B-trees, a node o is represented as a list of its keys: $< x_1, ..., x_{k-1} >$.

1. Give the linear representation of the tree in figure 2.
2. Draw the tree : "$(< 13, 32, 44 > (< 3 >)(< 18, 25 >)(< 35, 40 >)(< 46, 49, 50 >))$"
3. Write the function that builds the linear representation (as a string) from a B-tree
4. **Bonus:**Write the reciprocal function, that builds the B-tree from its linear representation.

Exercise 1.4 (B-tree or not B-tree... – *Midterm S3 - Oct. 2017*)

Write a function that tests whether a B-tree is "well-ordered", i.e. if the pre-defined range is respected everywhere.

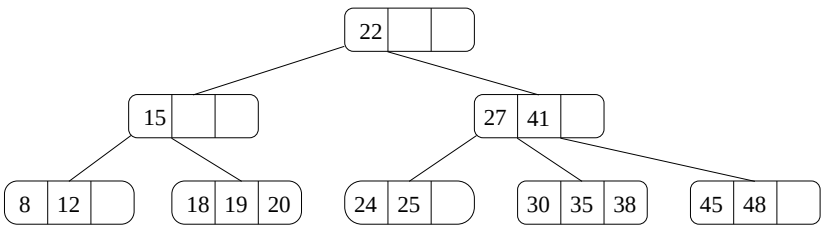

Figure 2: B-tree?

2 Classics

To make explanations simpler: in a node, for each key #i, the child $\#i$ will be called the *left child* and the child $\#i + 1$ the *right child*.

Exercise 2.1 (Minimum and maximum)

1. Where are the minimum and the maximum values in a B-tree?
2. Write the two functions that give the maximum (resp. minimum) value of a non empty B-tree.

Exercise 2.2 (Searching)

Write a function that searches a value x in a B-tree. The function returns the pair `(B, i)` such that `B.keys[i] == x` in case of a positive search, the value `None` otherwise.

Exercise 2.3 (Inserting a new element: the classic method)

1. (a) At which place a new element can be inserted in a B-tree so that its properties are preserved?
   (b) What problem can occur?
   (c) Which transformation can be used to solve this problem? What are the conditions to use this transformation?
   (d) Write the function that makes this transformation in the *perfect case*[1].
2. There are two ways of using this transformation: in going down (*precautionary principle*) or in going up.

   For each method:

   (a) Give the insertion principle.
   (b) Insert successively the following values to create a new 2-4 tree:
   5, 15, 40, 25, 18, 45, 38, 42, 9.
3. Write the insertion function in a B-tree applying the precautionary principle.

Exercise 2.4 (Deleting an element: in going down)

1. How can we delete an element that is not in a leaf? Use, for example, the deletion of 27 in the tree of figure 2 (we will prefer the left side, but this choice might be reconsidered later!).
2. (a) What is the problem with the deletion of 24 in the new tree? What transformation can be used to solve this problem (draw your inspiration from AVL trees)? Delete this value, then the value 25, using this new transformation.
   (b) Write the two functions of this transformation (on the left, on the right). Precise the possible limits of calling.
3. (a) What problem occurs when deleting the value 35 in the last tree? What new transformation must be used?
   (b) Write the function for the new transformation. Precise the possible limits of calling.
4. Use a precautionary principle as in insertion to delete successively the values 15, 22, 8, 20, 30, 18, 45, 12, 48, 19, 41. Try to limit as much as possible node "destructions".
   At the same time, build the principle of the deletion in a B-tree.
5. Write the function deleting an element.

[1] Not the root, no "space" problem.

Bonus

Exercise 2.5 (Insertion, a new method: for a little change. . .)

The insertion we work on before can imply useless modifications of the tree structure. The aim here is to find new transformations to avoid increasing the tree node number when inserting a new element.

1. A new transformation?
   (a) In the example of the exercise 2.3, how to avoid the splitting when the second to last element (42) is inserted?
   (b) Use the same method to insert successively the values 33 and 36 in the tree of figure 2 (always prefer the left side. . .). Can we use the same method to insert the value 42?
2. The insertion:
   (a) Insert in the obtained tree the values 42 and 16. Precise the conditions in which the transformations can be used for an insertion.
   (b) Finally, insert the values 37 and 23. Do we have to use the precautionary principle?
   (c) Deduce the new principle for insertion algorithm.
   (d) Write the insertion function.

Exercise 2.6 (Deleting an element: in going up!)

The deleting function that uses the precautionary principle requires too many modifications: for instance when trying to delete a non-present key, the tree can be even so modified.

Here deleting an element will be done without the precautionary principle. That is to say the modifications will be used only when necessarily: in going up.

1. Here, we use the same deletions as in exercise 2.4 on the tree from figure 2, but this time they will be done in going up.
   (a) Start with the deletions of 27, 24, 25, 35. Remarks?
   (b) Then, continue with 15, 22, 8, 20, 30, 18, 45, 12, 48, 19, 41.
2. Deduce the principle of deletion in a B-tree with modifications in going up.
3. Write this deletion function.