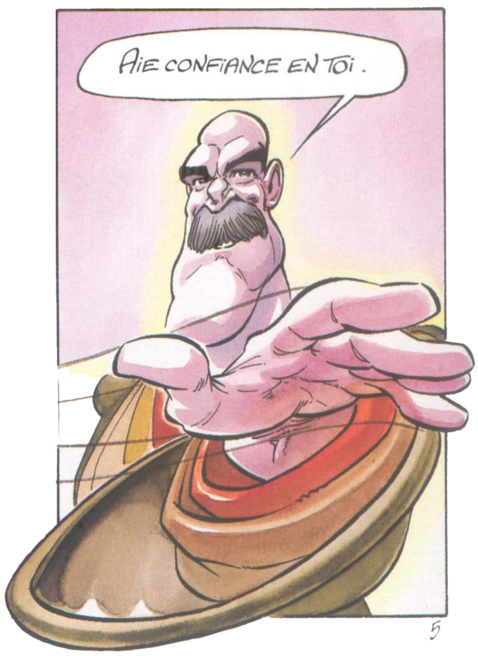
Algorithmics
Mid-term Exam #3

S3
EPITA

26 October 2015 - 14:00 (D.S. 308818.03 BW)

Instructions (read it) :

- ☐ You must answer on **the answer sheets provided.**
  - No other sheet will be collected. Keep your rough drafts.
  - Answer within the provided space. **Outside answers will not be marked**: Use your drafts!
  - Do not separate the sheets unless they can be re-stapled before handing in.
  - Penciled answers will not be marked.
- ☐ The presentation is negatively marked, which means that you are marked out of 20 points and the presentation points (maximum of 2) are taken off this grade.
- ☐ **Algorithms:**
  - All algorithms must be written in the language ALGO (no C, CAML or anything else).
  - All ALGO code not indented will not be marked.
  - All that you need (types, routines) is indicated in the **appendix** (the last page)!
- ☐ Duration : 2h



Exercise 1 (Some questions – *5 points*)

1. Give 2 required properties of a hash function.
2. Give a direct method of hashing.
3. Give an indirect method of hashing.
4. Which collision resolution method does not need a hash table whose size is greater than the number of keys to be hashed?
5. Which kind of search is incompatible with the hashing?
6. With which collision resolution method do secondary collisions appear?

Exercise 2 (General Trees: Prefix - Suffix – *7 points*)

The aim here is to fill a vector with the keys of a general tree. Each key is put **twice** in the vector: at the first encounter (prefix order) and at the second encounter (suffix order) during the depth first traversal.

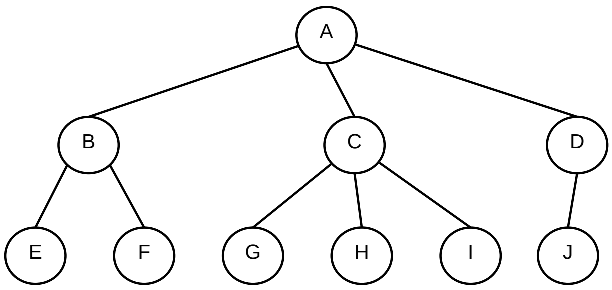


Figure 1: A general tree

After the depth first traversal of the tree in figure 1, the array will be filled as follows :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	B	E	E	F	F	B	C	G	G	H	H	I	I	C	D	J	J	D	A

1. *Tuples of pointers:*
   (a) Write the procedure `ps_stat(`T`,`c`,`V`)` that performs a depth first traversal of the tree T and fills the vector V according to the described order. (Each key is put in prefix then in suffix.) The integer c is the current position in the vector.
   (b) Write the function `filling_stat(`A`, `V`)` that uses the procedure `ps_stat(`A`, `C`, `V`)` to fill the vector V. This function returns the tree size.

2. *Left child - right sibling:*
   Write again the procedure of the question 1.(a), this time with the *left child - right sibling* implementation. This one is called `ps_dyn(A,c,v)`.
   The "calling function" would be the same (just change the type).

Exercise 3 (B-trees: Insertions – *2 + 6 points*)

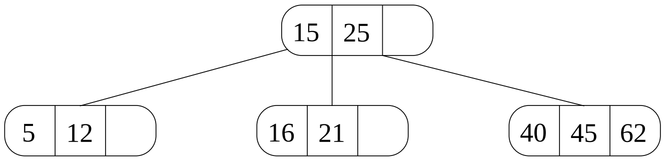


Figure 2: Btree with minimal degree 2

1. Successively insert the keys 18, 42 and 23 into the tree in figure 2 using the "in going down" method. Draw **only** the tree that you get after each insertion.

2. Complete the recursive function `insert_rec` that inserts a new element in a B-tree unless this element is already present.

   ▷ Use the "in going down" method.

   ▷ Use the split procedure with the following specifications:
   The procedure `split (`B`, `i`)` splits the child n°i of the tree B (type `t_Btree`).
   – B is a nonempty tree and its root is not a $2t$-node.
   – The child i of B exists and its root is a $2t$-node.

   ▷ We assume the function `search_pos` is written. Its specifications:
   The function `search_pos (t_element `x`, t_Btree `B`)` searches for the value x in the root of the nonempty B-tree B. It returns x position in the node if it is found, its "virtual" position otherwise.

   ▷ The function `insert_rec` will be called by the following:

```
algorithm function insertion_Btree : boolean
    local parameters
        t_element    x
    global parameters
        t_Btree      B

    variables
        integer      i
        t_Btree    R
begin
    if B = NUL then
        malloc (B)
        B↑.nbkeys ← 1
        B↑.keys[1] ← x
        B↑.children[1] ← NUL
        B↑.children[2] ← NUL
        return true
    else
        if B↑.nbkeys = 2*t-1 then
            malloc (R)           /* new root */
            R↑.nbkeys ← 0
            R↑.children[1] ← B
            B ← R
            split (B, 1)
        end if
        return insert_rec (x, B)
    end if
end algorithm function insertion_Btree
```

Appendix

Key vector

```
constants
    Max = ...
types
    t_element = ...
    t_elts_vect  = Max t_element
```

General tree implementations

Tuples of Pointers:

```
constants
  MaxCh = ...
types
  t_tree_tuples = ↑t_tuple_node

  t_vect_children = MaxCh t_tree_tuples

  t_tuple_node = record
    t_element       key
    integer        nbChildren
    t_vect_children    children
  end record t_tuple_node
```

Left Child - Right Sibling:

```
types
  t_dyn_tree = ↑t_dyn_node

  t_dyn_node = record
    t_element   key
    t_dyn_tree child, sibling
  end record t_dyn_node
```

B-trees implementation

```
constants
    t    = /* minimal degree */
types
     /* t_element */
    t_Btree    = ↑ t_node_Btree
    t_vect_keys  = (2*t-1) t_element
    t_vect_children  = (2*t) t_Btree
    t_node_Btree = record
        integer     nbkeys
        t_vect_keys  keys
        t_vect_children  children
    end record t_node_Btree
```

Reminder : in the children vector, the k first children are NUL for the external k-nodes.