Algorithmics
Correction Mid-term Exam #3

S3 – Epita

10 Nov. 2014 - 10:00

Solution 1 **(Some questions –** ***5 points*****)**

1. A hash function has to be uniform, easy and fast to compute and consistent.
2. The linear probing or the double hashing.
3. The hashing with separate chaining or the coalesced hashing.
4. The hashing with separate chaining. Le hachage avec chainage sÃľparÃľ. the elements are chained together outside the hash table.
5. The search by interval is incompatible with the hashing due to the dispersion of thes elements.
6. The secondary collisions appear with the coalesced hashing.

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

1. (a) **Specifications:**
   The procedure ps_stat(T,c,V) fills the vector V with the keys of the tree T in prefix then in suffix. The integer c is the current position in the vector.

```
algorithm procedure ps_stat
    local parameters
        t_tree_tuples    T
    global parameters
        integer  c
        t_elts_vect    V

    variables
        integer  i
begin
    c ← c + 1
    V[c] ← T↑.key
    for i ← 1 to A↑.nbChildren do
        ps_stat(T↑.children[i], c, V)
    end for
    c ← c + 1
    V[c] ← T↑.key
end algorithm procedure ps_stat
```

(b) **Specifications:**
The function `filling_stat(`T`, `V`)` fills the vector V with the keys of the tree T in prefix then in suffix. It returns the tree size.

```
algorithm function filling_stat : integer
    local parameters
        t_tree_tuples    T
    global parameters
        t_elts_vect    V

    variables
        integer                        c
begin
    c ← 0
    ps_stat(T, c, V)
    return (c div 2)
end algorithm function filling_stat
```

2. **Specifications:**
The procedure `ps_dyn(`T`,`c`,`V`)` fills the vector V with the keys of the tree T in prefix then in suffix. The integer c is the current position in the vector.

```
algorithm procedure ps_dyn
    local parameters
        t_dyn_tree T
    global parameters
        integer  c
        t_elts_vect    V
begin
    if A <> NUL then
        c ← c + 1
        V[c] ← T↑.key
        ps_dyn(T↑.child, c, V)
        c ← c + 1
        V[c] ← A↑.key
        ps_dyn(T↑.sibling, c, V)
    end if
end algorithm procedure ps_dyn
```

Classical depth first traversal:

```
algorithm procedure ps_dyn
    local parameters
        t_dyn_tree T
    global parameters
        integer  c
        t_elts_vect    V
    variables
        t_dyn_tree F
begin
    c ← c + 1
    V[c] ← T↑.key
    F ← T↑.child
    while F <> NUL do
        ps_dyn(F, c, V)
        F ← F↑.sibling
    end while
    c ← c + 1
    V[c] ← A↑.key
end algorithm procedure ps_dyn
```

***Solution 3* (Arbre 2-3-4 : insertions – *2 + 6 points*)**

1.

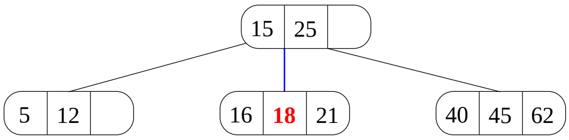

Figure 1: After insertion of 18

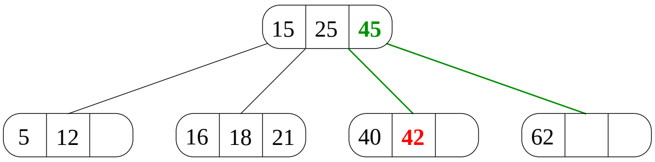

Figure 2: After insertion of 42 - Node (40-45-62) has been split

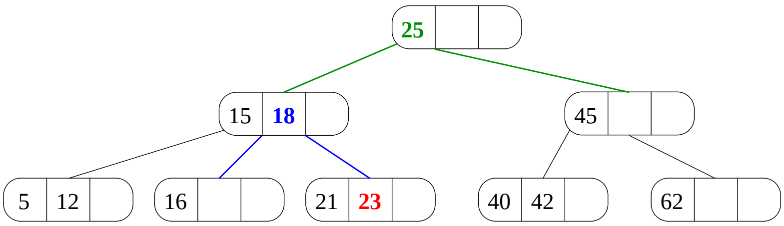

Figure 3: After insertion of 23 - The root then the node (40-45-62) have been split

2. **Specifications:**
   The function `insert_rec` (x, B) inserts the key x in the tree B of type `t_Btree`, unless x is already in the tree. B is nonempty, and its root is not a full node (not a $2t$-node).

La fonction récursive ci-dessous ne prend pas en compte les cas particuliers de l'arbre vide et d'un $2t$-nœud en racine. Ces cas sont gérés par la fonction d'appel donnée dans l'énoncé.

```
algorithm function insert_rec : boolean
   local parameters
      t_element    x
      t_Btree      B

   variables
      integer     i, j, m

begin
   i ← search_pos (x, B)                        /* search where to insert x */

   if (i > B↑.nbkeys) or (B↑.keys[i] <> x) then

      if B↑.children[1] <> NUL then             /* x ∉ node */

          if B↑.children[i]↑.nbkeys = 2*t-1 then
            if B↑.children[i]↑.keys[t] = x then
               return false
            end if
            split (B, i)
            if x > B↑.keys[i] then
               i ← i + 1
            end if
         end if

         return insert_rec (x, B↑.children[i])

      else                                                  /* insertion */

         for j ← B↑.nbkeys downto i do
            B↑.keys[j+1] ← B↑.keys[j]
         end for

         B↑.keys[i] ← x
         B↑.nbkeys ← B↑.nbkeys + 1
         B↑.children[B↑.nbkeys+1] ← NUL

         return true

      end if
   else

      return false

   end if
end algorithm function insert_rec
```