Laboratory Module 7
2-3 Trees

Purpose:
− understand the notion of 2-3 trees
− to build, in C, a 2-3 tree

1 2-3 Trees

1.1 General Presentation

2-3 Trees represent the simplest type of multiway trees – trees with nodes that have more than two descendents. This kind of trees have the following structural properties:
- A non-leaf node has two keys and three children.
- A node with one key has exactly two children
- A leaf node has one or two keys

The next figures present a sample structure of a node with one or two keys.

![Sample structure of a node from a 2-3 Tree](image)

Here is the structure of a node from a 2-3 Tree.

```c
typedef struct 23Node{
    int k1, k2;
    struct 23Node *left, *middle, *right, up;
};
```

The rules that are followed by the keys within a 2-3 tree are:
- The keys within a node are ordered. This means that \( k_1 \) is smaller than \( k_2 \).
- The keys from left subtree are smaller than first key from the parent: \( X < k_1 \).
- The keys from middle subtree are greater than first key but smaller than the second key from the parent: \( k_1 < Y < k_2 \).
- The keys from right subtree are greater than second key from the parent: \( k_2 < Z \).
1.2 Operations on 2-3 Trees

1.2.1 Searching a Key

Searching a key $x$ within a 2-3 tree starts from the root node. If key $x$ is not found in the root than the key needs to be searched within one subtree. The child node of the root where the key is searched is determined based on the relation between key $x$ and the keys $k_1$ and $k_2$ from the root. The searching continues in the same manner until a leaf node is reached.

For example, searching key 50 in the 2-3 Tree from figure 2 is performed in the following way. Firstly, the key 50 is searched in the root node. It is not found there and that is why the searching procedure needs to go to a lower level. Because the root node has only one key and searched key is greater the searching procedure will resume from the middle child node of the root. So, the current node where key 50 is searched becomes the node with keys 51 and 68. The key 50 is not found in this node and thus the searching proceeds at a lower level, at node with keys 45 and 50. This node is taken into consideration because key 50 is smaller than key 51 which is the smallest key in the node. This meant that the left child was taken into consideration. Once the node with keys 45 and 50 is reached the key 50 is found and the procedure ends.

The pseudocode for searching a key in a 2-3 tree is:

```
SEARCH(a, r)
{
if ( children of r are leaves )
    return r
else
    if ( a <= k_1[r] )  return Search(a, r->left)
    else if ( a <= k_2[r] )  return Search(a, r->middle)
    else              return Search(a, r->right)
}
```

1.2.2 Inserting a Key

Insertion of a key into a 2-3 tree is performed only in a leaf node. That is why, the first operation that needs to be performed is to search for the leaf where the key may be inserted. The leaf where the key needs to be inserted may have one or two keys.

**Case 1.** If the leaf has one key than the new key is placed in proper position. If the new key is greater than $k_1$ (the existing key) than the new key is placed in second position as $k_2$. If the new key is smaller than $k_1$ than $k_1$ is placed in second position as $k_2$ and the new key is placed as $k_1$ in the first position.

**Case 2.** If the leaf node has two keys than the new key may NOT be placed directly in it. In this situation a key from the leaf needs to be lifted up to the parent and a new brother of the leaf containing the new key is created.

The pseudocode for inserting a key in a 2-3 tree is:
**Insert** \((a, r)\) \\
if ( \(r\) consists of a single leaf \(l\) labeled \(b\) ) // that is, if \(l\) is the root  
create a new root \(r'\) // interior node  
create a new leaf \(v\) labeled \(a\)  
make \(l\) and \(v\) children of \(r'\) in proper order  
update \(L\) and \(M\) for \(r'\)  
else  
Set \(f\) to \(\text{SEARCH}(a, r)\)  
create a new leaf \(l\) labeled \(a\)  
if \(f\) has 2 children  
insert \(l\) into proper position  
update \(L\) and \(M\)  
else  
create a transitory 4-node tree at \(f\) (\(l\) in proper position)  
\(\text{ADDCHILD}(f)\)  
}  

**ADDCHILD** \((v)\) \\
create new interior node \(v'\)  
move 2 rightmost children of \(v\) to \(v'\)  
if ( \(v\) has no parent ) // that is, if \(v\) is the root  
create new root \(r'\)  
make \(v\) the left child and \(v'\) the right child  
update \(L\) and \(M\)  
else  
let \(f\) be the parent of \(v\)  
make \(v'\) the child of \(f\) immediately to the right of \(v\)  
if \(f\) now has 4 children  
\(\text{ADDCHILD}(f)\)  
else  
update \(L\) and \(M\)  
}  

Here is an example of performing insert operations into a 2-3 Tree. Suppose we have the tree from figure 2 and we want to insert key 30.

![Figure 3. 2-3 Tree after inserting key 30](image)

The steps that are performed for inserting key 30 are:

**Step 1.** Leaf node with keys 24 and 28 is determined by the search procedure. The leaf node is full and thus a key needs to be lifted up to the parent node.

**Step 2.** Key 30 is greater than both existing keys: 24 and 28. Key 28 is lifted up to the parent node where it lays in the second position. A new brother for the leaf node is created and key 30 is placed in it.

Having the 2-3 tree presented in figure 3 we want to insert key 10. The necessary steps are:
**Step 1.** Leaf node with keys 9 and 15 is determined by the search procedure. The leaf node is full and thus a key needs to be lifted up to the parent node.  
**Step 2.** Key 10 is greater than 9 (which represents $k_1$) but smaller than 15 (which represents $k_2$). Key 10 (middle key) is lifted up to the parent node.  
**Step 3.** The parent node (with keys 20 and 28) is full and that is why a brother is created (node with key 28). This node will take with him two children (nodes with keys 24 and 30). In the parent node it remains only key 10 with two children (nodes with keys 9 and 15), while key 20 is lifted up to the parent node.  
**Step 4.** Key 20 is placed in parent node in first position since is smaller than key 42. Placing is possible due to the fact that the node has only one key. If the root node had two keys than a new root is needed thus leading to an increase in tree’s height.

![2-3 Tree after inserting key 10](image)

**1.2.3 Deleting a Key**

Deleting a key from a 2-3 tree is a complex operation due to the fact that after deletion all the constraints reading the structure of the tree must still hold.

The first step that is performed is to search the key that needs to be deleted. The deleted key may belong to a leaf node or to an internal node. If the key belongs to an internal node than a predecessor or a successor key needs to be determined. This key must belong to a leaf node and will be copied over the key that needs to be deleted. Finally, the key from the leaf must be deleted.

As conclusion, deletion of a key from a 2-3 tree may be treated only for the case when we want to delete a key from a leaf node. The procedure for deleting a key from the leaf must make sure the remaining tree is perfectly balanced and the structure is well preserved. This means nodes with one key have exactly two children and nodes with two keys have exactly three children. The situation that may arise are:

**Situation 1.** The leaf node contains TWO keys. In this situation the deletion is straight. If $k_2$ key needs to be deleted than it is set to the maximum integer value. If $k_1$ needs to be deleted than $k_2$ is copied over $k_1$ and maximum integer value is set over $k_2$.

**Situation 2.** The leaf node contains ONE key. The deletion can not be straight because the node will remain without keys and thus the parent will remain without a child. This situation is not acceptable due to structural consistency reasons. A brother of the node where deletion takes place is inspected regarding the number of keys.

**Case 1.** If the brother has two keys than $k_2$ is moved to the parent and a corresponding key from the parent is brought down to the node where deletion takes place. Once we have two keys in the node where deletion takes place we are in the situation 1.

**Case 2.** If the brother has one key than a key from the parent node needs to be brought down along the key from the brother, thus being possible the deletion. This situation may lead to decreasing the height of the tree. Bringing down the key from the parent may is treated as deletion the key from a leaf node.
The pseudocode for deleting a key from a 2-3 tree is:

```
Delete()
Let f be the parent of the node just deleted (or NULL if leaf was root)
while(f is an illegal interior node) //not NULL or leaf, with one child
  if ( f has no parent )
    make the single child of f the new root
    delete f
    Set f to the root //for while loop condition
  else
    let g be parent of f //known to have 2 or 3 children
    if ( one of f's siblings is a 3-node )
      move one child from the 3-node into f  //That is, f becomes legal
      update k_1 and k_2 everywhere
    else
      give f's remaining child to one of f's siblings
      delete f
      update k_1 and k_2 everywhere
    Set f to g //for while loop condition
}//end-while
```

Here is an example of performing delete operations into a 2-3 Tree. Suppose we have the tree from figure 4 and we want to delete key 45 and 50.

Deletion of key 45 is simple. The leaf node containing keys 45 and 50 will remain only with key 50.
Deletion of key 50 is a bit more complex since it can not be accomplished in a straight manner.

Closest brother (node with keys 55 and 59 – see figure 4) is inspected and is observed that it has two keys. Key k_1 (with value 55) is lifted up to the parent and key 51 from the parent is brought down into the leaf node where deletion takes place. After these modes the deletion is performed in a straight way and the tree will look like in figure 5.

Figure 5. 2-3 Tree after deleting keys 45 and 50

2. Assignments

1) Write a program that creates and manages a 2-3 tree. The program must implement the following operations: creation, insertion, deletion and tree display. The program should present a menu where user may choose from implemented options.

2) It is given the 2-3 tree from the following figure.
Present the shape of the tree after each of the following operations:
- INSERT 8
- INSERT 7
- INSERT 12
- INSERT 51
- DELETE 25
- DELETE 88

For each insertion operation there must be presented the following:
- Initial position of the key into the 2-3 tree
- Analysis regarding the number of keys
- The necessary movements of the keys
- The final shape of the tree

For each deletion operation there must be presented the following:
- The search of the tree for the key that needs to be deleted
- The analysis regarding the type (internal or root) of the node
- The necessary movements of the keys
- The final shape of the tree

References:
Aho, Hopcroft, and Ullman