# Data structures and algorithms
# Project documentation

**Blaj Andrei-Sorin – Group 911**

# SortedBag ADT

(implementation on a Binary Search Tree)

# 1. Problem statement

A family needs to store their daily expenses in a program and may chose to sort them in ascending order, add expenses, remove expenses. We will use a SortedBag ADT (implemented on a Binary Search Tree) to ensure the storing of similar elements.

# 2. SortedBag ADT

## Domain:

SB = { sb | *sb* is a set of *TElement type elements* }

## Operations (interface):

- **create (*root*)**
  {creates an empty SB}
*Pre:* *true*
*Post:* *sb* ∈ *SB, sb*=Φ*(empty sb)*

- **Insert *(root, data)***
  {insert an element into the SB}
*Pre:* *sb* ∈ SB*, data* ∈ *TElement*
*Post:* *sb'* ∈ SB*, sb'* = *sb* ∪ {*e}*

- **Delete *(root, data)***
  {removes an element from the SB}
*Pre:* *sb* ∈ *SB, data* ∈ *TElement*
*Post:* *sb'* ∈ *SB, sb'*=*sb* ∈ *data*

- **count *(root)***
  {returns the number of elements in the SB}
*Pre:* *sb* ∈ *SB*
*Post:* *count* = |*SB*|

- **Search *(root, data)***
  {search for an element in the SB}
*Pre: sb* ∈ *SB, data* ∈ *TElement*
*Post:* seach = $\begin{cases} true, & if\ data \in \Phi \\ false, & if\ data \notin \Phi \end{cases}$

- **isEmpty*(root)***

*Pre: sb ∈ SB*

*Post: vidă=* $\begin{cases} true, & if\ sb \in \Phi \\ false, & if\ sb \notin \Phi \end{cases}$

- **Iterator*(root, i)***

{iterates through all the elements of the SB}

*Pre:* *sb ∈ SB*

*Post:* *i∈I, I is an iterator on SB*

- **Destroy*(root)***

{destructor}

*Pre:* *root∈M*

*Post:* *m* has been destroyed (the allocated memory has been freed)

# 3. SortedBag Iterator

## Domain:

*I* = { *i* | *i* is an iterator on *root* ∈ *M* }

## Operații (interfață):

- **create(*it*)**

*Pre:* *true*

*Post:* *it ∈ I*

- ***isV*alid*(it)***

*Pre:* *it ∈ I*

*Post:* *isValid <— if iterator is Valid*

- **getValue*(it)***

*Pre:* *it ∈ I*

*Post:* *getValue <— the value of the iterator*

- **next*(it)***

*Pre:* *it ∈ I*

*Post:* *it' = I*

# 4. Representation

## Node:

*data: Integer*
*parent: Node*
*left: Node*
*right: Node*

## SortedBag ADT

*root = Node*

## C++ implementation of the interface operations:

```
getIterator() {
        return IteratorBST{root};
}
```
**O(1)**

```
createNode() {
        root = new Node();
        root -> data = data;
        root -> parent = parent;
        root -> left = root -> right = NULL;
}
```
**O(1)**

```
Delete(struct Node *root, int data) {
        if(root == NULL) return root;
        else if(data < root->data) root->left = Delete(root->left,data);
        else if (data > root->data) root->right = Delete(root->right,data);
        else {
                // Case 1:  No child
                if(root->left == NULL && root->right == NULL) {
                delete root;
                root = NULL;
         }
        //Case 2: One child
        else if(root->left == NULL) {
                struct Node *temp = root;
                root = root->right;
                delete temp;
         }
        else if(root->right == NULL) {
                struct Node *temp = root;
                root = root->left;
                delete temp;
        }
```

```
        // case 3: 2 children
        else {
                struct Node *temp = FindMin(root->right);
                root->data = temp->data;
                root->right = Delete(root->right,temp->data);
        }

        return root;

} O(log n)

searchElement(Node *root, int data) {
        if (root == NULL)
                return root;

        else if (data < root -> data)
                root -> left = Search(root -> left, data);
        else if (data > root -> data)
                root -> right = Search(root -> right, data);

        return root;

} O(log n)

Insert(Node* parent, Node *root, int data) {
        if(root == NULL) {
                root = new Node();
                root -> data = data;
                root -> parent = parent;
                root -> left = root -> right = NULL;
        }
        else if (data <= root -> data)
                root -> left = Insert(root, root -> left, data);
        else
                root -> right = Insert(root, root -> right, data);

        return root;

} O(log n)

isEmpty() {
        if (root == NULL)
                return 1; // it's empty
         return 0; // it's not empty

} O(1)
```

## SortedBagIterator ADT

*index : întreg*

## C++ implementation of the interface operations

```
isValid() {
        return root != NULL;
} O(1)

getValue() {
        ireturn root->data;
} O(1)

next() {
        if(!goDeeper(root->right)){
                if(root->parent != NULL){
                while(root->parent != NULL && root == root->parent->right){
                        root = root->parent;
                }
                root = root->parent;
                }
        }
} O(1)

goDeeper() {
        if(here == NULL){
                return false;
        }
        if(!goDeeper(here->left)){
                root = here;
        }
        return true;
} O(n)
```