Algoritmi i strukture podataka 2019./2020.

Zadatci za vježbu – binarno stablo

U zadatcima se koristi razred `BinaryTree` te pomoćni razred `Node`:

```
template <typename T>
class BinaryTree {
public:

   BinaryTree() : root(nullptr) {}
   ~BinaryTree() {}
   bool isEmpty() const;
   void insert(const T& item) { insert (root, item); }
   bool remove(const T& item) { return remove(root, item); }
   bool search(const T& item) { return search(root, item); }

private:
   struct Node {
      T item;
      shared_ptr<Node> left;
      shared_ptr<Node> right;
       Node(const T& newItem) : item(newItem), left(nullptr), right(nullptr) {}
      ~Node();
   };
   shared_ptr<Node> root;
   void insert(shared_ptr<Node>&, const T&);
   bool remove(shared_ptr<Node>&, const T&);
   bool search(shared_ptr<Node>&, const T&);

};
```

U zadatcima 1. – 5. binarno stablo je uređeno tako da lijevo idu manji, a desno veći elementi. U 6. i 7. zadatku vidjeti tekst zadatka za uređenost stabla. Pretpostavite da su za tip T definirani svi operatori usporedbe.

Naputak: u zadatcima 1. - 7. po potrebi definirati privatne funkcijske članove, koji se pozivaju iz javnih funkcijskih članova zadanih prototipovima (za primjer vidjeti 1. zadatak).
Za svaki zadatak napišite i odsječak glavnog programa kojim ćete testirati funkcijske članove.

1. Napišite javni funkcijski član razreda `BinaryTree` koji će vratiti broj čvorova u binarnom stablu, a zadan je prototipom:
   ```
   int countNodes();
   ```

   Naputak: definirajte pomoćni privatni funkcijski član čiji je prototip:
   ```
   int countNodes(shared_ptr<Node>&);
   ```

2. Napišite javni funkcijski član razreda `BinaryTree` koji će čvorove u stablu ispisati inorder postupkom te vratiti broj ispisanih elemenata. Prototip je:
   ```
   int inorder2();
   ```

3. Napišite javni funkcijski član razreda `BinaryTree` koji će prebrojati sve čvorove stabla na zadanoj razini, a čiji je prototip:
```
int countLevel(int level);
```

Napomena: pretpostavite da je korijen stabla na razini 1.

4. Napišite javni funkcijski član razreda `BinaryTree` koji će osloboditi svu memoriju koju su zauzeli čvorovi stabla, a čiji je prototip:

```
void freeAllNodes();
```

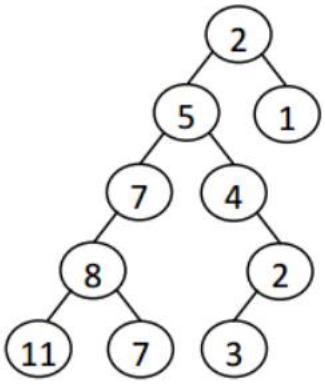
5. (prema zadatku za drugi ispitni rok 2013./2014.)
Napišite javni funkcijski član razreda `BinaryTree` koji će pronaći n-ti po veličini član stabla, a čiji je prototip:

```
T nthValue(int n);
```

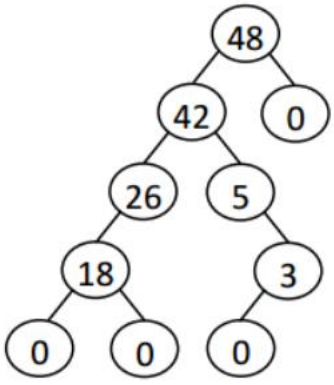
Cijeli broj n je redni broj po veličini elementa kojeg tražimo (1 za najveći član, 2 za drugi najveći itd.). Pretpostaviti da je broj n uvijek manji ili jednak broju čvorova stabla.

6. (prema zadatku za završni ispit 2011./2012.)
Napišite javni funkcijski član koji će svaki element stabla zamijeniti sumom elemenata u njegovom lijevom i desnom podstablu (tj. sumom njegovih potomaka) prije zamjene vrijednosti u tim potomcima, a čiji je prototip:
```
void changeValues();
```

Pretpostaviti da je definiran operator + za T.
Primjer: zadan je niz brojeva 2, 5, 7, 8, 11, 1, 4, 2, 3, 7 koji su spremljeni u čvorove binarnog stabla. Stablo sa slike Slika 1 transformirat će se u stablo na slici Slika 2.



Slika 1

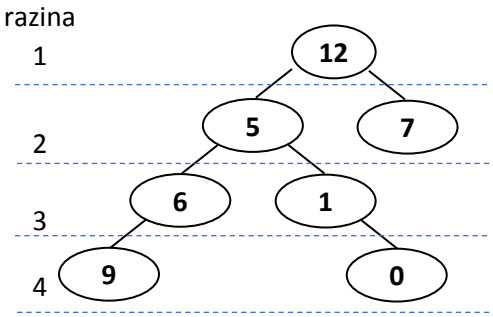


Slika 2

7. (prema zadatku za završni ispit 2018./2019.)

Napišite javni funkcijski član koji vraća zbroj vrijednosti čvorova stabla na zadanoj razini. Ako na zadanoj razini nema čvorova, treba vratiti 0. Pretpostaviti da su vrijednosti čvorova tipa int. Prototip funkcije je:

```
int sumLevel(int level);
```



Primjer: za stablo s lijeve strane **sumLevel** treba vratiti za razine 1 i 2 vrijednost 12, za razinu 3 vraća vrijednost 7, a za razinu 4 vrijednost 9.

Rješenja:

1.

```
// public
template <typename T>
int BinaryTree<T>::countNodes() { return countNodes(root); }

// private
template <typename T>
int BinaryTree<T>::countNodes(shared_ptr<Node>& node) {
	if (node) {
		return countNodes(node->left) + 1 + countNodes(node->right);
	}
	return 0;
}
```

2.

```
// public
template <typename T>
int BinaryTree<T>::inorder2() { return inorder2(root); }

// private
template <typename T>
int BinaryTree<T>::inorder2(shared_ptr<Node>& node) {
	if (node) {
		int cnt = inorder2(node->left);
		++cnt;
		// operator << treba biti preopterećen za tip T
		std::cout << node->item << " ";
		cnt += inorder2(node->right);
		return cnt;
	}
	return 0;
}
```

3.

```
// public
template <typename T>
int BinaryTree<T>::countLevel(int level) { return countLevel(root, level); }

// private
template <typename T>
int BinaryTree<T>::countLevel(shared_ptr<Node>& node, int level) {
	if (node && level >= 1) {
		if (level == 1) {
			return 1;
		}
		else {
			return countLevel(node->left, level - 1)
				+ countLevel(node->right, level - 1);
		}
	}
	return 0;
}
```

4.

```
// public
template <typename T>
void BinaryTree<T>::freeAllNodes { freeAllNodes(root); }

// private
template <typename T>
void BinaryTree<T>::freeAllNodes(shared_ptr<Node>& node) {
	if (node) {
		freeAllNodes(node->left);
		freeAllNodes(node->right);
		node->right = node->left = nullptr;
		node = nullptr;
	}
}
```

No, može samo i:

```
// public
template <typename T>
void BinaryTree<T>::freeAllNodes { root = nullptr; }
```

Zašto?

5.

```
// public
template <typename T>
T BinaryTree<T>::nthValue(int n) {
	T item;
	if (n < 1 || n > countNodes(root)) {
		throw std::invalid_argument("Argument out of range.");
	}
	int curr = 0;
	nthValue(root, n, item, curr);
```

```
	return item;
}

// private
template <typename T>
void BinaryTree<T>::nthValue(shared_ptr<Node>& node, int n, T& item, int& curr) {
	if (node && n >= 1) {
		if (node->right) nthValue(node->right, n, item, curr);
		++curr;
		if (n == curr) item = node->item;
		if (node->left) nthValue(node->left, n, item, curr);
	}
}
```

6.

```
// public
template <typename T>
void BinaryTree<T>::changeValues() { T item = changeValues(root); }

// private
template <typename T>
T BinaryTree<T>::changeValues(shared_ptr<Node>& node) {
	if (node) {
		T oldValue = node->item;
		node->item = changeValues(node->left) + changeValues(node->right);
		return oldValue + node->item;
	}
	return 0;
}
```

7.

```
// public
template <>
int BinaryTree<int>::sumLevel(int level) { return sumLevel(root, level); }

// private
template <>
int BinaryTree<int>::sumLevel(shared_ptr<Node>& node, int level) {
	if (node) {
		if (level > 1) {
			int sumLeft = node->left ? sumLevel(node->left, level - 1) : 0;
			int sumRight = node->right ? sumLevel(node->right, level - 1) : 0;
			return sumLeft + sumRight;
		}
		else if (level == 1) { return node->item; }
	}
	return 0;
}
```