Звіт

Автор: Момот Р. КІТ-119а

Дата: 09.05.2020

ЛАБОРАТОРНА РОБОТА № 13. ДЕРЕВА

Мета: набути досвід практичної роботи з бінарними деревами.

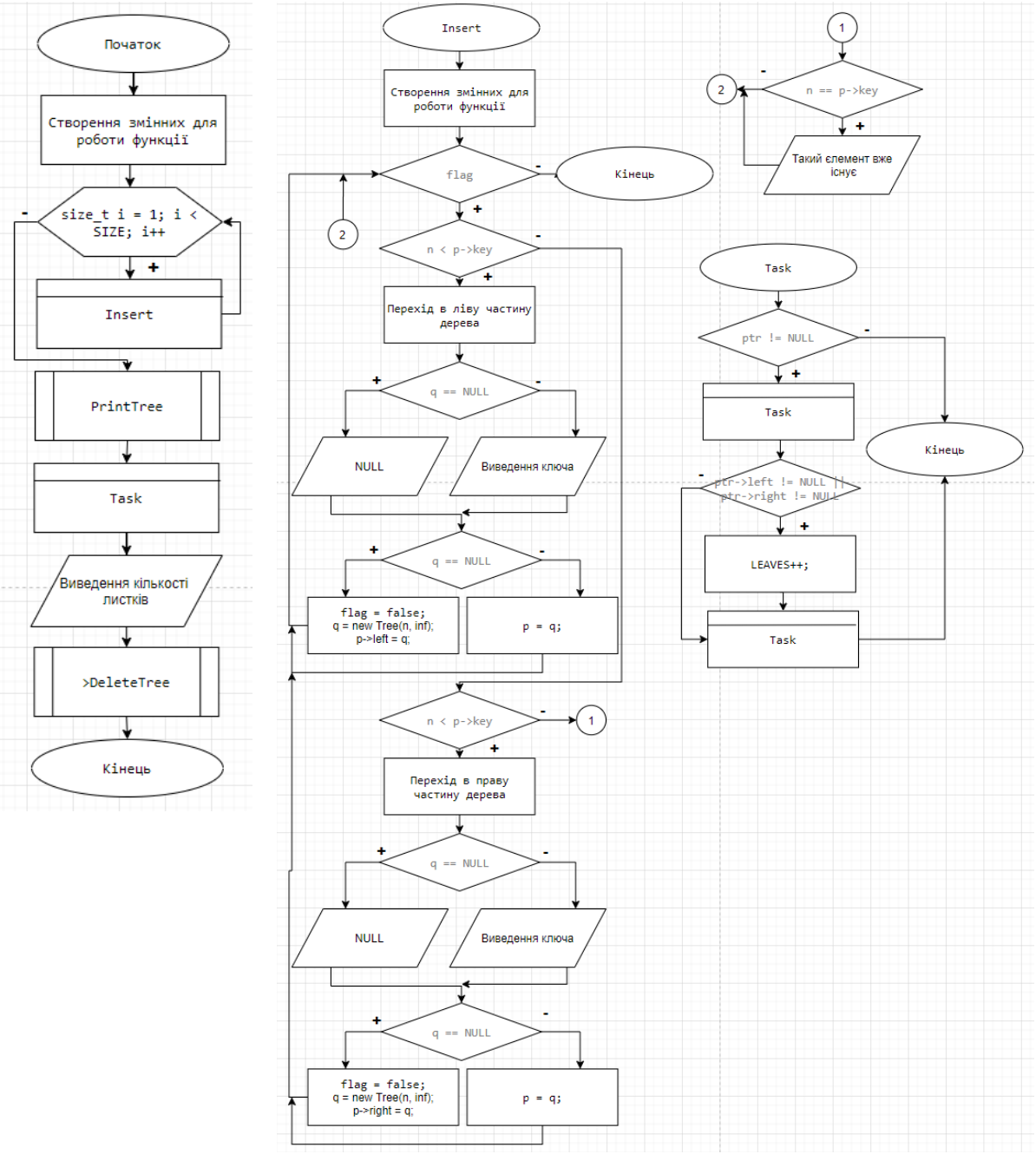
Індивідуальне завдання

Розробити програму, що дозволяє створити бінарне дерево та вирішити індивідуальне завдання.

В інформаційну частину вузлів записати ключ, що є цілим числом. Видати вміст дерева та результати індивідуального завдання на екран.

Індивідуальне завдання: визначити кількість листів в дереві.

Блок-схема алгоритму програми



Текст програми

```
#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#define DEBUG_NEW new(_NORMAL_BLOCK, __FILE__, __LINE__)
#define new DEBUG_NEW

#include<conio.h>
#include<stdlib.h>
#include<iostream>
using namespace std;
static int LEAVES = 0;
class Tree
{
	int key;			//ключ
	int info;			//информационное поле
	Tree* left, * right;//указатель на левый и правый узел

public:
	Tree()					//конструктор по умолчанию
	{
		key = 0;
		left = right = NULL;
	}
	Tree(int key, int info)	//конструктор с параметр
	{
		this->key = key;
		this->info = info;
		left = right = NULL;
	}
	~Tree() {}				//Деструктор

	void Insert(int, int, Tree*);			//функция добавления элемента в дерево
	void PrintTree(Tree*, int) const;		//вывод дерева на экран
	void DeleteTree(Tree*);					//удаление дерева
	void Task(Tree*);						//индивидуальное задание
	void PrintTreeLikeMas(Tree*& Tree)const; //Функция обхода
};

int main()
{
	setlocale(LC_ALL, "Rus");
	int SIZE = 20;
	int number, info;

	number = rand() % 100 + 1;				//случайные значения корня
	info = rand() % 100 + 1;

	Tree* root = new Tree(number, info);	//создание корня

	for (size_t i = 1; i < SIZE; i++)
	{
		number = rand() % 100 + 1;			//случайные значения
		info = rand() % 100 + 1;
		root->Insert(number, info, root);
	}

	cout << endl;
	root->PrintTree(root, 5);
	root->Task(root);
	cout << endl << "Количество листьев: " << LEAVES;
	root->DeleteTree(root);					//освобождение памяти

	if (_CrtDumpMemoryLeaks())
```

```
		cout << "\nУтечка памяти обнаружена." << endl;
	else
		cout << "\nУтечка памяти отсутствует." << endl;

	return 0;
}

void Tree::Insert(int n, int inf, Tree* ptr)
{
	cout << "\n Добавляется элемент (" << n << ";" << inf << ")\n";
	bool flag = true;
	Tree* p, * q;
	p = ptr;
	while (flag)
	{
		if (n < p->key)
		{
			q = p->left;   //a3
			cout << "Найден узел " << p->key << endl << "Переход влево" << endl;

			if (q == NULL)
				cout << "NULL" << endl;
			else
				cout << q->key << endl;

			if (q == NULL)
			{
				flag = false;
				cout << "Создается новый элемент (" << n << "," << inf << ")\n";
				q = new Tree(n, inf);
				cout << "Обновление связей: " << p->key << "->left = " << q->key <<
endl;
				p->left = q;
			}
			else
				p = q;
		}
		else if (n > p->key)
		{
			q = p->right;         //a4
			cout << "Найден узел " << p->key << endl << "Переход вправо" << endl;

			if (q == NULL)
				cout << "NULL" << endl;
			else
				cout << q->key << endl;

			if (q == NULL)
			{
				flag = false;
				cout << "Создается новый элемент (" << n << "," << inf << ")\n ";
				q = new Tree(n, inf);
				cout << "Обновление связей: " << p->key << "->right = " << q->key <<
endl;
				p->right = q;
			}
			else
				p = q;
		}
		else if (n == p->key)
		{
			cout << "Такой элемент уже существует " << endl;
			flag = false;        //выход из цикла WHILE
		}
	}
}
```

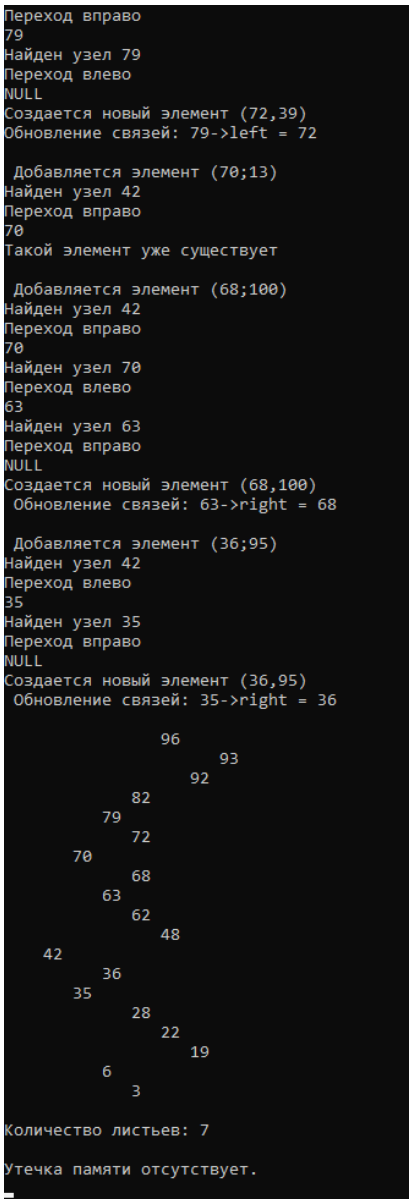
```
void Tree::PrintTree(Tree* ptr, int n) const
{
	if (ptr)
	{
		PrintTree(ptr->right, n + 3);		//вызов для правого поддерева
		for (size_t i = 1; i < n; i++)
			cout << " ";

		cout << ptr->key << endl;
		PrintTree(ptr->left, n + 3);		//вызов для левого поддерева
	}
}
void Tree::Task(Tree* ptr)
{
	if (ptr != NULL)
	{
		Task(ptr->left);

		if (ptr->left == NULL && ptr->right == NULL)
			LEAVES++;

		Task(ptr->right);
	}
}
void Tree::DeleteTree(Tree* ptr)
{
	if (ptr != NULL)
	{
		DeleteTree(ptr->left);
		DeleteTree(ptr->right);
		delete(ptr);
	}
}
void Tree::PrintTreeLikeMas(Tree*& Tree) const  //Функция обхода
{
	if (Tree != NULL)			//Пока не встретится пустое звено
	{
		PrintTreeLikeMas(Tree->left); //Рекурсивная функция для вывода левого поддерева
		cout << Tree->key << "  ";    //Отображаем корень дерева
		PrintTreeLikeMas(Tree->right);//Рекурсивная функци для вывода правого поддерева
	}
}
```

Результат роботи програми



Висновок

У результаті лабораторної роботи програми було розроблено програму, яка створює бінарне дерево, заповнюючи елементи випадковими значеннями, виводить його на екран та знаходить кількість листів.

Відповіді на питання

1. Що таке дерево?

Дерево – це такий граф, в якому існує єдиний елемент, на який не посилається ніякий інший елемент (корінь), на кожен елемент, крім кореня, мається тільки єдине посилання і є вузли, що не посилаються на інші вузли дерева (листи).

2. Чим визначається арність дерева?

Арність N-арних дерев визначається полустепінню результату кожної вершини, яка менше або дорівнює N.

3. Які операції притаманні деревам?

Над деревами визначені наступні основні операції:

1) пошук вузла з заданим ключем;
2) додавання нового вузла;
3) видалення вузла (поддерева);
4) обхід дерева в певному порядку:
   - спадний обхід;
   - змішаний;
   - всхідний обхід.

4. Що таке «обхід» дерева, які операції обходу відомі?

Обхід дерева - систематичний перегляд всіх його вузлів в певному порядку.

Існують три види обходу:

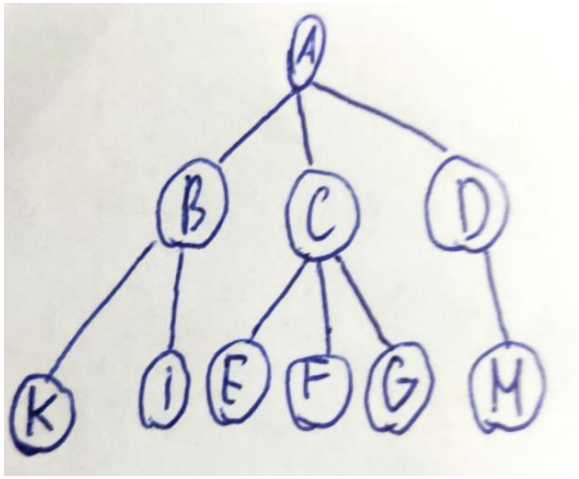
- Прямий порядок;
- Зворотний порядок;
- Центрований (центральний) порядок;

5. Яке дерево називається деревом пошуку?

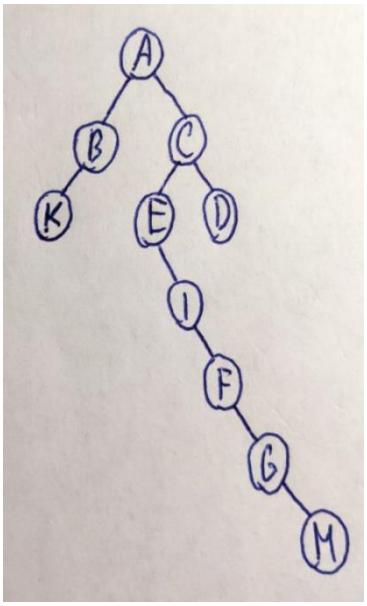
Деревом пошуку називається дерево, для кожної вершини t_i якого справедливим є твердження, що всі ключі лівого піддерева менше ключа вузла t_i, а всі ключі правого піддерева більше його.

6. Напишіть структуру m-арного дерева за своїм розсудом. Перетворіть його в бінарне.

m-арне дерево



бінарне дерево



7. Які виділяють типи дерев?

Бінарні дерева: бінарні, повні бінарні, ідеально збалансовані дерева, збалансовані дерева, не збалансовані дерева;

m-арні дерева (m>2)

8. Опишіть алгоритм висхідного обходу дерева.

1) спуститися по лівій гілці із запам'ятовуванням вершини в стеці як 1-й вид стекових записів;

2) якщо стек порожній, то перейти до п.5;

3) вибрати вершину з стека, якщо це перший вид стекових записів, то повернути його в стек як 2-й вид стекових записів; перейти до правого синові; перейти до п.1, інакше перейти до п.4;

4) опрацювати дані вершини і перейти до п.2;

5) кінець алгоритму.

При рекурсивному обході дерева спочатку виконується висхідний обхід лівого піддерева, потім виконується обхід правого піддерева, потім виконується обробка коріння.

9. Опишіть алгоритм спадного обходу дерева.

Схематично алгоритм обходу двійкового дерева відповідно до низхідним способом може виглядати наступним чином:

1). Взяти корінь дерева.

2). Провести обробку чергової вершини відповідно до вимог завдання.

3). Якщо чергова вершина має обидві гілки, то в якості нової вершини вибрати ту вершину, на яку посилається ліва гілка, а вершину, на яку посилається права гілка, занести в стек; перейти до пункту 2. Якщо чергова вершина є кінцевою, то вибрати в якості нової чергової вершини вершину з стека, якщо він не порожній, і перейти до пункту 2. Якщо ж стек порожній, то це означає, що обхід всього дерева закінчено. Якщо чергова вершина має тільки одну гілку, то в якості чергової вершини вибрати ту вершину, на яку ця гілка вказує, перейти до пункту 2.

Алгоритм істотно спрощується при використанні рекурсії. Так, спадний обхід можна описати таким чином:

1) обробка кореневої вершини;

2) спадний обхід лівого піддерева;

3) спадний обхід правого піддерева.

10.Опишіть алгоритм змішаного обходу дерева.

Змішаний обхід можна описати таким чином:

1) спуститися по лівій гілці із запам'ятовуванням вершин в стеці;

2) якщо стек порожній - кінець;

3) вибрати вершину з стека і обробити дані вершини;

4) якщо вершина має правого сина, то перейти до нього; перейти до п.1.

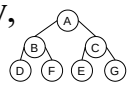
У разі рекурсії змішаний обхід описується наступним чином:

1) змішаний обхід лівого піддерева;

2) обробка кореневої вершини;

3) змішаний обхід правого піддерева.

11.Вкажіть послідовність вузлів даного дерева при спадному, висхідному та змішаному його обходах.



Спадний обхід: ABDFCEG

Змішаний обхід: DBFAECG

Висхідний обхід: DFBEGCA

12.Яке дерево називається повним m-арним?

Якщо полустепінь результату кожної вершини в точності дорівнює або m, або нулю, то таке дерево називається повним m-арним деревом.

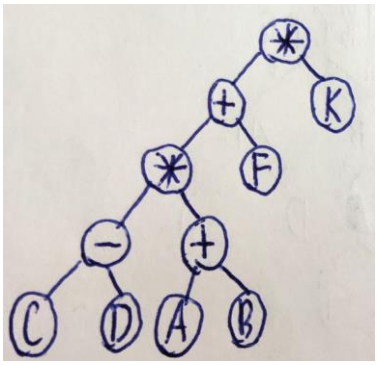
13. Яке дерево називається ідеально збалансовоним?

За визначенням Вірта дерево називається ідеально збалансованим, якщо число вершин в його лівих і правих піддерева відрізняється не більше, ніж на одну вершину. Таке дерево має мінімальну глибину (кількість рівнів). Мінімальна глибина досягається, якщо на всіх рівнях, крім останнього, міститься максимально можливе число вершин.

14. Яке дерево називається повним бінарним?

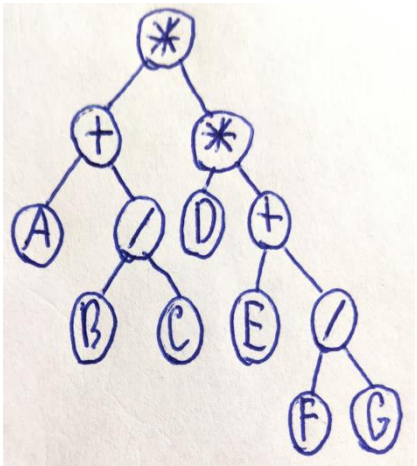
Якщо полустепені результату кожної вершини в точності дорівнює або m, або нулю, то таке дерево називається повним m-арним деревом. При m = 2 такі дерева називаються відповідно бінарними, або повними бінарними.

15. Дано арифметичний вираз ((a + b) * (c – d) +f) * k. Побудуйте для нього бінарне дерево.



16. Запишіть арифметичний вираз за своїм розсудом. Створіть відповідне для нього дерево. Виконавши обход дерева, запишіть префіксну, інфіксну та посфіксну форми запису заданого виразу.

(A+B/C)*(D*(E+F/G)



Префіксна: *+A/BC*D+E/FG
Інфіксна: A+B/C*D*E+F/G
Постфіксна: ABC/+DEFG/+**

17.Функція якого обходу дерева наведена в програмному прикладі?

```
void Order(Tree* t)
{
  if (!t) return;
  if (t->left) Order(t->left);
  . . .  // Обробка даних вузла дерева
        if (t->right) Order(t->right);
}
```

В даному коді наведен приклад висхідного рекурсивного обходу дерева.

18.Які дерева називаються червоно-чорними, які їх властивості?

Червоно-чорне дерево - один з видів самобалансірующіхся довічних дерев пошуку, що гарантують логарифмический зростання висоти дерева від числа вузлів і дозволяє швидко виконувати основні операції дерева пошуку: додавання, видалення і пошук вузла. Збалансованість досягається за рахунок введення додаткового атрибута вузла дерева - «кольору». Цей атрибут може приймати одне з двох можливих значень - «чорний» або «червоний».

1) Кожен вузол забарвлений або в червоний, або в чорний колір (в структурі даних вузла з'являється додаткове поле - біт кольору).

2) Корінь забарвлений в чорний колір.

3) Листя (так звані NULL-вузли) пофарбовані в чорний колір.

4) Кожен червоний вузол повинен мати два чорних дочірніх вузла. Потрібно відзначити, що у чорного вузла можуть бути чорні дочірні вузли. Червоні вузли в якості дочірніх можуть мати тільки чорні.

5) Шляхи від вузла до його листя повинні містити однакову кількість чорних вузлів (це чорна висота).

19.Яке призначення операції балансування дерев?

Так як основні операції над двійковими деревами пошуку (пошук, вставка і видалення вузлів) лінійно залежать від його висоти, то виходить гарантована логарифмічна залежність часу роботи цих алгоритмів від числа ключів, що зберігаються в дереві.

20.Якими властивостями характеризуються АВЛ-дерева?

Для будь-якого вузла дерева висота його правого піддерева відрізняється від висоти лівого піддерева не більше ніж на одиницю.

Ключ будь-якого вузла дерева не менш будь-якого ключа в лівому піддереві даного вузла і не більше будь-якого ключа в правому піддереві цього вузла.

21.Які властивості мають ідеально сбалансовані дерева?

Для будь-якого вузла дерева висота його правого піддерева відрізняється від висоти лівого піддерева не більше ніж на одиницю.