МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
Київський національний університет імені Тараса Шевченка
Кафедра програмних систем і технологій

Звіт з лабораторної роботи 1.9
тема: «Бінарне дерево»

Варіант 2

Виконав: студент групи ІПЗ - 12
Богатько Олександр
Геннадійович
Перевірила: викладач
Юрчук Ірина Аркадіївна

Київ 2020

Контрольні питання

1. Що таке абстрактна структура даних?

2. Що таке бінарне пошукове дерево?

3. Які основні властивості і способи побудови структури даних бінарне пошукове дерево?

4. Напишіть алгоритм основних операцій для структури даних бінарне пошукове дерево?
5. В якому порядку відвідуються вузли бінарного дерева у випадку:

- ✓ симетричного обходу – *inorder traversal*;
- ✓ обходу в прямому порядку (*в ширину*)– *preorder traversal*;
- ✓ обходу в зворотньому порядку (*в глибину*)– *postorder traversal*.

Відповіді:

1. **Абстрактний тип даних** — це математична модель для типів даних, де тип даних визначається поведінкою з точки зору користувача даних, а саме в термінах можливих значень, можливих операцій над даними цього типу і поведінки цих операцій.

2. Бінарне дерево — це структура даних, що складається з множини вузлів, які об'єднані зв'язками типу батько --> дитина

3.

Нехай *x* — довільна вершина двійкового дерева пошуку. Якщо вершина *y* знаходиться в лівому піддереві вершини *x*, то val[*y*] ≤ val[*x*].

Якщо *y* знаходиться у правому піддереві *x*, то val[*y*] ≥ val[*x*].

Таке структурування дозволяє надрукувати усі значення у зростаючому порядку за допомогою простого алгоритму центрованого обходу дерева.

Представляється таке дерево вузлами наступного вигляду:

*Node = (element, key, left, right, parent). Доступ до дерева *T* здійснюється за допомогою посилання *root*.

4. Алгоритм пошуку

```
1. if x = nil або k = key [x]
 2.    then return x
 3. if k < key [x]
 4.    then return Tree_Search (left [x], k)
 5.    else return Tree_Search (right [x], k)
```

Алгоритм пошуку мінімального елемента

```
1. while left[x] ≠ NIL
 2.    do x ← left[x]
 3. return x
```

Максимального

```
1. while right[x] ≠ NIL
 2.    do x ← right[x]
 3. return x
```

Алгоритм пошуку наступного та попереднього елемента

```
1. if right[x] ≠  NIL
 2.    then return TREE_MINIMUM(right[x])
 3. y ← p[x]
 4. while y ≠ NIL та x = right[y]
 5.    do     x ← y
 6.           y ← p[y]
 7. return y
```

Алгоритм додавання елемента

```
1. y ← NIL
 2. x ← root[T]
 3. while x ≠ NIL
 4.    do y ← x
 5.           if key[z] < key [x]
 6.                    then x ← left[x]
 7.                   else x ← right[x]
 8. p [z] ← y
 9. if y = NIL
 10.   then root[T] ← z                    // Дерево T - пусте
 11.   else if key[z] <key[y]
 12.          then left[y] ← z
 13.          else right[y] ← z
```

Алгоритм видалення елемента

```
DELETE (T, z)
1  if left[z] = NULL or right[z]=NULL
2    then y:=z
3    else y:=SUCCESSOR(z)
4  if left[y] <> NULL
5    then x:=left[y]
6    else x:= right[y]
7  if x <> NULL
8    then p[x]:=p[y]
```

```
9  if p[y]:=NULL
10   then root[T]:=x
11   else if y=left[p[y] ]
12     then left[p[y] ] :=x
13     else right[p[y] ]:=x
14 if y <> z
15   then val[z]:=val[y]
16  //копіювання додаткових даних з y
17 return y
```

5.

1.**Симетричний обхід**

1.1. Обхід лівого піддерева даного вузла.

1.2. Відвідування даного вузла.
1.3. Обхід правого піддерева даного вузла.

2. **Обхід у прямому порядку або обхід в ширину.**

2.1. Відвідування даного вузла.
2.2. Обхід лівого піддерева даного вузла.
2.3. Обхід правого піддерева даного вузла.

3. **Обхід у зворотньому порядку або обхід в глубину.**

3.1. Обхід лівого піддерева даного вузла.
3.2. Обхід правого піддерева даного вузла.

3.3. Відвідування даного вузла.

Лабораторне завдання:

Реалізувати операції:

1. Створенні бінарного дерева.
2. Додавання елемента до бінарного дерева.
3. Видалення елемента з бінарного дерева.
4. Видалення бінарного дерева.
5. Перевірка бінарного дерева на пустоту.
6. Знаходження та видобування даних.
7. Копіювання бінарного дерева.
8. Обхід бінарного дерева за трьома різними маршрутами.
9. Виведення на консоль всіх елементів дерева.

```
#include <iostream>
#include <cmath>
using namespace std;
struct BinTree {
    int value; //має в собі значення
    BinTree* left;//адрес левого підерева
    BinTree* right;//адрес правого підерева
    BinTree(int val) : left(NULL), right(NULL), value(-1) {}
};

void newBinTree(int val, BinTree** Tree) {
    if ((*Tree) == NULL)
    {
        (*Tree) = new BinTree(0);
        (*Tree)->value = val;
        (*Tree)->left = (*Tree)->right = NULL;
        return;
    }
    if (val > (*Tree)->value) newBinTree(val, &(*Tree)->right);
    else newBinTree(val, &(*Tree)->left);
}
void push(int a, BinTree **t)
{
    if ((*t) == NULL)
    {
        (*t) = new BinTree(0);
        (*t)->value = a;
        (*t)->left = (*t)->right = NULL;
        return;
    }
    if (a > (*t)->value) push(a, &(*t)->right);
    else push(a, &(*t)->left);
}

BinTree* DeleteNode(BinTree* node, int value){
    if(node == NULL)
        return node;
    if(value == node->value){
        BinTree* tmp;
        if(node->right == NULL)
            tmp = node->left;
        else {
            BinTree* ptr = node->right;
            if(ptr->left == NULL){
                ptr->left = node->left;
                tmp = ptr;
            } else {
                BinTree* pmin = ptr->left;
                while(pmin->left != NULL){
                    ptr  = pmin;
                    pmin = ptr->left;
```

```
                }
                ptr->left   = pmin->right;
                pmin->left  = node->left;
                pmin->right = node->right;
                tmp = pmin;
            }
        }
        delete node;
        return tmp;
    } else if(value < node->value)
        node->left  = DeleteNode(node->left, value);
    else
        node->right = DeleteNode(node->right, value);
    return node;
}
void Print(BinTree**Tree, int l)
{
    int i;

    if (*Tree != NULL)
    {
        Print(&((**Tree).right), l + 1);
        for (i = 1; i <= l; i++) cout << "    ";
        cout << (**Tree).value << endl;
        Print(&((**Tree).left), l + 1);
    }
}
BinTree * CopyTree(BinTree * node)
{
    if (node == NULL)
        return NULL;
    BinTree * newnode = new BinTree(node->value);
    newnode->left = CopyTree(node->left);
    newnode->right = CopyTree(node->right);

    return newnode;
}
void TreeTraversalAndPrint(BinTree* Root) {
    if (Root != NULL) {
        cout << Root->value << endl;
        TreeTraversalAndPrint(Root->left);
        TreeTraversalAndPrint(Root->right);

    }
}
void TreeTraversalAndPrint2(BinTree* Root) {
    if (Root != NULL) {
        TreeTraversalAndPrint2(Root->left);
        TreeTraversalAndPrint2(Root->right);
        cout << Root->value << endl;
    }
}
void TreeTraversalAndPrint3(BinTree* Root) {
    if (Root != NULL) {
        TreeTraversalAndPrint2(Root->left);
        cout << Root->value << endl;
        TreeTraversalAndPrint2(Root->right);
    }
}
BinTree* Search(BinTree* Tree, int key) {
    if (Tree == NULL) return NULL;
    if  (Tree->value == key) return Tree;
    if (key < Tree->value) return Search(Tree->left, key);
    else
        return Search(Tree->right, key);
}
void DestroyBTree(BinTree* Tree) {
    if (Tree != NULL) {
        DestroyBTree(Tree->left);
        DestroyBTree(Tree->right);
        delete(Tree);
    }
}
int main() {
    int command;
    BinTree* Tree = NULL;
    cout<<"Примітка: Дерево виводиться зліва направо\n"
          "1. Створення бінарного дерева.\n"
```

```
        "2. Видалення елемента з бінарного дерева.\n"
        "3. Видалення бінарного дерева.\n"
        "4. Перевірка бінарного дерева на пустоту.\n"
        "5. Знаходження та видобування даних.\n"
        "6. Копіювання бінарного дерева.\n"
        "7. Обхід бінарного дерева за трьома різними маршрутами.\n"
        "8. Виведення на консоль всіх елементів дерева.\n"
        "0. Вийти.\n";
    while(command!=0){
        cout<<"Введіть команду: ";
        cin>>command;
          switch(command){
              case 1:
                  int s,n;
                  cout << "Введіть кількість елементів: ";
                  cin >> n;
                  for (int i=0; i<n; ++i)
                  {
                      cout << "Введіть число:  ";
                      cin >> s;
                      push(s, &Tree);
                  }
                  Print(&Tree, 0);
                  break;
              case 2:
                  int el;
                  cout<<"Введіть елемент: ";
                  cin>>el;
                  DeleteNode(Tree, el);
                  cout<<"Елемент видалено.\n";
                  break;
              case 3:
                  DestroyBTree(Tree);
                  cout<<"Дерево видалено.\n";
                  break;
              case 4:
                  if(Tree == NULL)
                      cout<<"Дерево порожнє\n";
                else cout<<"Дерево не порожнє\n";
                  break;
              case 5:
              int key;
              cout << "Введіть значення елемента для пошуку-> ";
              cin >> key;
              Search(Tree, key);
                  if (Tree == NULL)
                      cout << "Елемент не знайдено\n";
                  else
                      cout << "Ваш елемент->" << Tree->value<<"\n";
                  break;
              case 6:CopyTree(Tree);
                  cout<<"Дерево скопійовано.\n";
                  break;
              case 7:
                  int com;
                  cout<<"1.Прямий обхід дерева\n2.Зворотній обхід дерева\n"
                        "3.Симетричний обхід дерева\nВведіть команлу:";
                  cin>>com;
              switch(com){
                  case 1:
                      TreeTraversalAndPrint(Tree);
                      break;
                  case 2:
                      TreeTraversalAndPrint2(Tree);
                      break;
                  case 3:
                      TreeTraversalAndPrint3(Tree);
                      break;
              }
                  break;
              case 8:
                  Print(&Tree, 0);
                  break;
      }
    }
    return 0;
}
```

```
Примітка: Дерево виводиться зліва направо
1. Створення бінарного дерева.
2. Видалення елемента з бінарного дерева.
3. Видалення бінарного дерева.
4. Перевірка бінарного дерева на пустоту.
5. Знаходження та видобування даних.
6. Копіювання бінарного дерева.
7. Обхід бінарного дерева за трьома різними маршрутами.
8. Виведення на консоль всіх елементів дерева.
0. Вийти.
Введіть команду: 1
Введіть кількість елементів: 10
Введіть число:  10
Введіть число:  6
Введіть число:  8
Введіть число:  3
Введіть число:  4
Введіть число:  2
Введіть число:  14
Введіть число:  18
Введіть число:  16
Введіть число:  12
      18
         16
   14
      12
10
      8
   6
         4
      3
         2
Введіть команду: 2
Введіть елемент: 2
Елемент видалено.
Введіть команду: 8
      18
         16
   14
      12
10
      8
   6
         4
      3
```

```
Введіть команду: 7
1.Прямий обхід дерева
2.Зворотній обхід дерева
3.Симетричний обхід дерева
Введіть команлу:1
10
6
3
4
8
14
12
18
16
Введіть команду: 7
1.Прямий обхід дерева
2.Зворотній обхід дерева
3.Симетричний обхід дерева
Введіть команлу:2
4
3
8
6
12
16
18
14
10
Введіть команду: 7
1.Прямий обхід дерева
2.Зворотній обхід дерева
3.Симетричний обхід дерева
Введіть команлу:3
4
3
8
6
10
12
16
18
14
Введіть команду: 6
Дерево скопійовано.
Введіть команду: 3
Дерево видалено.
```