МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
АДЫГЕЙСКИЙГОСУДАРСТВЕННЫЙУНИВЕРСИТЕТ

Инженерно-физический факультет
Кафедра автоматизированных систем обработки информации и
управления

Отчет по практике

Написать программу красно-черные деревья, вариант 5.

2 курс, группа 2ИВТ2

Выполнили:

_____ Виноградов К.Е

«___» _____ 2022 г.

Руководитель: _____

С.В. Теплоухов «___» _____ 2022 г.

Майкоп, 2022 г.

1. Введение

1) Текстовая формулировка задачи
2) Код данной задачи
3) Скриншот программы

2. Вариант 5

Задание

Красно-черные деревья.

Теория

Красно-черные деревья - один из способов балансировки деревьев. Название происходит от стандартной раскраски узлов таких деревьев в красный и черный цвета. Цвета узлов используются при балансировке дерева. Во время операций вставки и удаления поддеревья может понадобиться повернуть, чтобы достигнуть сбалансированности дерева. Оценкой как среднего время, так и наихудшего является O(log n).

Красно-черное дерево - это бинарное дерево с следующими свойствами:

- Каждый узел покрашен либо в черный, либо в красный цвет.
- Листьями объявляются NIL-узлы (т.е. "виртуальные" узлы, наследники узлов, которые обычно называют листьями; на них "указывают" NULL указатели). Листья покрашены в черный цвет.
- Если узел красный, то оба его потомка черны.
- На всех ветвях дерева, ведущих от его корня к листьям, число черных узлов одинаково.

3. Ход работы

3.1. Код приложения

Красно-черные деревья.

```
#include<iostream>
using namespace std;

struct node
{
   int key;
   node* parent;
   char color;
   node* left;
   node* right;
};
class RBtree
{
   node* root;
   node* q;
public:
   RBtree()
   {
      q = NULL;
      root = NULL;
   }
   void insert();
   void insertfix(node*);
   void leftrotate(node*);
   void rightrotate(node*);
   void del();
   node* successor(node*);
   void delfix(node*);
   void disp();
   void display(node*);
```

```
    void search();
};
void RBtree::insert()
{
    int z, i = 0;
    cout << "\nEnter key of the node to be inserted: ";
    cin >> z;
    node* p, * q;
    node* t = new node;
    t->key = z;
    t->left = NULL;
    t->right = NULL;
    t->color = 'r';
    p = root;
    q = NULL;
    if (root == NULL)
    {
        root = t;
        t->parent = NULL;
    }
    else
    {
        while (p != NULL)
        {
            q = p;
            if (p->key < t->key)
                p = p->right;
            else
                p = p->left;
        }
        t->parent = q;
        if (q->key < t->key)
            q->right = t;
```

```
        else
            q->left = t;
    }
    insertfix(t);
}
void RBtree::insertfix(node* t)
{
    node* u;
    if (root == t)
    {
        t->color = 'b';
        return;
    }
    while (t->parent != NULL && t->parent->color == 'r')
    {
        node* g = t->parent->parent;
        if (g->left == t->parent)
        {
            if (g->right != NULL)
            {
                u = g->right;
                if (u->color == 'r')
                {
                    t->parent->color = 'b';
                    u->color = 'b';
                    g->color = 'r';
                    t = g;
                }
            }
            else
            {
                if (t->parent->right == t)
                {
```

```
            t = t->parent;
            leftrotate(t);
        }
        t->parent->color = 'b';
        g->color = 'r';
        rightrotate(g);
    }
}
else
{
    if (g->left != NULL)
    {
        u = g->left;
        if (u->color == 'r')
        {
            t->parent->color = 'b';
            u->color = 'b';
            g->color = 'r';
            t = g;
        }
    }
    else
    {
        if (t->parent->left == t)
        {
            t = t->parent;
            rightrotate(t);
        }
        t->parent->color = 'b';
        g->color = 'r';
        leftrotate(g);
    }
}
```

```
        root->color = 'b';
    }
}

void RBtree::del()
{
    if (root == NULL)
    {
        cout << "\nEmpty Tree.";
        return;
    }
    int x;
    cout << "\nEnter the key of the node to be deleted: ";
    cin >> x;
    node* p;
    p = root;
    node* y = NULL;
    node* q = NULL;
    int found = 0;
    while (p != NULL && found == 0)
    {
        if (p->key == x)
            found = 1;
        if (found == 0)
        {
            if (p->key < x)
                p = p->right;
            else
                p = p->left;
        }
    }
    if (found == 0)
    {
```

```
    cout << "\nElement Not Found.";
    return;
}
else
{
    cout << "\nDeleted Element: " << p->key;
    cout << "\nColour: ";
    if (p->color == 'b')
        cout << "Black\n";
    else
        cout << "Red\n";

    if (p->parent != NULL)
        cout << "\nParent: " << p->parent->key;
    else
        cout << "\nThere is no parent of the node.  ";
    if (p->right != NULL)
        cout << "\nRight Child: " << p->right->key;
    else
        cout << "\nThere is no right child of the node.  ";
    if (p->left != NULL)
        cout << "\nLeft Child: " << p->left->key;
    else
        cout << "\nThere is no left child of the node.  ";
    cout << "\nNode Deleted.";
    if (p->left == NULL || p->right == NULL)
        y = p;
    else
        y = successor(p);
    if (y->left != NULL)
        q = y->left;
    else
    {
```

```
            if (y->right != NULL)
                q = y->right;
            else
                q = NULL;
        }
        if (q != NULL)
            q->parent = y->parent;
        if (y->parent == NULL)
            root = q;
        else
        {
            if (y == y->parent->left)
                y->parent->left = q;
            else
                y->parent->right = q;
        }
        if (y != p)
        {
            p->color = y->color;
            p->key = y->key;
        }
        if (y->color == 'b')
            delfix(q);
    }
}

void RBtree::delfix(node* p)
{
    node* s;
    while (p != root && p->color == 'b')
    {
        if (p->parent->left == p)
        {
```

```
    s = p->parent->right;
    if (s->color == 'r')
    {
        s->color = 'b';
        p->parent->color = 'r';
        leftrotate(p->parent);
        s = p->parent->right;
    }
    if (s->right->color == 'b' && s->left->color == 'b')
    {
        s->color = 'r';
        p = p->parent;
    }
    else
    {
        if (s->right->color == 'b')
        {
            s->left->color == 'b';
            s->color = 'r';
            rightrotate(s);
            s = p->parent->right;
        }
        s->color = p->parent->color;
        p->parent->color = 'b';
        s->right->color = 'b';
        leftrotate(p->parent);
        p = root;
    }
}
else
{
    s = p->parent->left;
    if (s->color == 'r')
```

```
        {
            s->color = 'b';
            p->parent->color = 'r';
            rightrotate(p->parent);
            s = p->parent->left;
        }
        if (s->left->color == 'b' && s->right->color == 'b')
        {
            s->color = 'r';
            p = p->parent;
        }
        else
        {
            if (s->left->color == 'b')
            {
                s->right->color = 'b';
                s->color = 'r';
                leftrotate(s);
                s = p->parent->left;
            }
            s->color = p->parent->color;
            p->parent->color = 'b';
            s->left->color = 'b';
            rightrotate(p->parent);
            p = root;
        }
    }
    p->color = 'b';
    root->color = 'b';
  }
}

void RBtree::leftrotate(node* p)
```

```
{
   if (p->right == NULL)
      return;
   else
   {
      node* y = p->right;
      if (y->left != NULL)
      {
         p->right = y->left;
         y->left->parent = p;
      }
      else
         p->right = NULL;
      if (p->parent != NULL)
         y->parent = p->parent;
      if (p->parent == NULL)
         root = y;
      else
      {
         if (p == p->parent->left)
            p->parent->left = y;
         else
            p->parent->right = y;
      }
      y->left = p;
      p->parent = y;
   }
}
void RBtree::rightrotate(node* p)
{
   if (p->left == NULL)
      return;
   else
```

```
    {
        node* y = p->left;
        if (y->right != NULL)
        {
            p->left = y->right;
            y->right->parent = p;
        }
        else
            p->left = NULL;
        if (p->parent != NULL)
            y->parent = p->parent;
        if (p->parent == NULL)
            root = y;
        else
        {
            if (p == p->parent->left)
                p->parent->left = y;
            else
                p->parent->right = y;
        }
        y->right = p;
        p->parent = y;
    }
}

node* RBtree::successor(node* p)
{
    node* y = NULL;
    if (p->left != NULL)
    {
        y = p->left;
        while (y->right != NULL)
            y = y->right;
```

```
    }
    else
    {
        y = p->right;
        while (y->left != NULL)
            y = y->left;
    }
    return y;
}

void RBtree::disp()
{
    display(root);
}
void RBtree::display(node* p)
{
    if (root == NULL)
    {
        cout << "\nEmpty Tree.";
        return;
    }
    if (p != NULL)
    {
        cout << "\n\t NODE: ";
        cout << "\n Key: " << p->key;
        cout << "\n Colour: ";
        if (p->color == 'b')
            cout << "Black";
        else
            cout << "Red";
        if (p->parent != NULL)
            cout << "\n Parent: " << p->parent->key;
        else
```

```
        cout << "\n There is no parent of the node.  ";
      if (p->right != NULL)
        cout << "\n Right Child: " << p->right->key;
      else
        cout << "\n There is no right child of the node.  ";
      if (p->left != NULL)
        cout << "\n Left Child: " << p->left->key;
      else
        cout << "\n There is no left child of the node.  ";
      cout << endl;
      if (p->left)
      {
        cout << "\n\nLeft:\n";
        display(p->left);
      }
      /*else
       cout<<"\nNo Left Child.\n";*/
      if (p->right)
      {
        cout << "\n\nRight:\n";
        display(p->right);
      }
      /*else
       cout<<"\nNo Right Child.\n"*/
   }
}
void RBtree::search()
{
   if (root == NULL)
   {
      cout << "\nEmpty Tree\n";
      return;
   }
```

```
int x;
cout << "\n Enter key of the node to be searched: ";
cin >> x;
node* p = root;
int found = 0;
while (p != NULL && found == 0)
{
   if (p->key == x)
      found = 1;
   if (found == 0)
   {
      if (p->key < x)
         p = p->right;
      else
         p = p->left;
   }
}
if (found == 0)
   cout << "\nElement Not Found.";
else
{
   cout << "\n\t FOUND NODE: ";
   cout << "\n Key: " << p->key;
   cout << "\n Colour: ";
   if (p->color == 'b')
      cout << "Black";
   else
      cout << "Red";
   if (p->parent != NULL)
      cout << "\n Parent: " << p->parent->key;
   else
      cout << "\n There is no parent of the node.  ";
   if (p->right != NULL)
```

```
        cout << "\n Right Child: " << p->right->key;
      else
        cout << "\n There is no right child of the node.  ";
      if (p->left != NULL)
        cout << "\n Left Child: " << p->left->key;
      else
        cout << "\n There is no left child of the node.  ";
      cout << endl;

   }
}
int main()
{
   int ch, y = 0;
   RBtree obj;
   do
   {
      cout << "\n\t RED BLACK TREE ";
      cout << "\n 1. Insert in the tree ";
      cout << "\n 2. Delete a node from the tree";
      cout << "\n 3. Search for an element in the tree";
      cout << "\n 4. Display the tree ";
      cout << "\n 5. Exit ";
      cout << "\nEnter Your Choice: ";
      cin >> ch;
      switch (ch)
      {
      case 1: obj.insert();
        cout << "\nNode Inserted.\n";
        break;
      case 2: obj.del();
        break;
      case 3: obj.search();
```

```
            break;
        case 4: obj.disp();
            break;
        case 5: y = 1;
            break;
        default: cout << "\nEnter a Valid Choice.";
        }
        cout << endl;

    } while (y != 1);
    return 1;
}
```

4. Пример скриншота программы

Красно-черные деревья

Рис. 1. скриншот программы

Рис. 2. скриншот программы

5. библиографические ссылки

Для изучения «внутренностей» TEX необходимо изучить [1], а для использования LATEX лучше почитать [2, 3].

Список литературы

[1] Кнут Д.Э. Всё про TEX. —- Москва: Изд. Вильямс, 2003 г. 550 с.

[2] Львовский С.М. Набор и верстка в системе LATEX. —- 3-е издание, исправленное и дополненное, 2003 г.

[3] Воронцов К.В. LATEX в примерах. 2005 г.