МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Українськийдержавний університет  
науки і технологій**

Кафедра «Комп’ютерніінформаційнітехнології»

**Лабораторна робота №3**

**з дисципліни «Якість ПЗ та тестування»**

**на тему: «Проектування структур данихефективних за показникамиобчислювальноїскладності»**

Дніпро, 2023

**ЛАБОРАТОРНА РОБОТА №3**

**Тема.**Проектування структур данихефективних за показникамиобчислювальноїскладності.

**Мета.**Вивчитикомбінаторно-ймовірносніметодивизначенняпоказниківобчислювальноїскладностіалгоритмів та отриматинавичкизастосуванняцихметодів при вирішенні задач розробкиефективних структур даних.

**Звіт**

1. **Постановка задачi**

Розробитиоб’єктно-орієнтованупрограму, щореалізує три структуриданих, визначивши для них операціїдодавання, видалення, пошукуелементів.

Для текступрограми на заданіймовіпрограмуваннявиконатипошукусіх

лексем класу, визначеноговаріантом. Знайденілексеми занести в розроблені

структури, виключаючи повтор елементів.

Виконатиймовірнісно-комбінаторнуоцінкуфункцій для роботизі структурами, на основіякоїзробитивисновок про оптимальністьструктури з точки зорунайменшоїобчислювальноїскладності.

Класи лексем відповідноваріанту:

1. ідентифікатори;
2. зарезервовані слова;
3. операції;
4. роздільники;
5. коментарі;
6. літерали.

Мову програмування, на якій буде представлено текст програми для лексичногорозбору, обирає студент самостійно за погодженням з викладачем.

**Варіант4:** (бінарне дерево, стек зв’язний, стек послідовний). Клас – роздільники.Мова тексту вхідної програми – С#.

1. **Фрагмент коду для використання у роботі**

using System;

namespace TestProgram

{

// Клас Soldier

class Soldier

{

// Атрибути класу

private string name\_;

private int health\_;  
private int damage\_;

// Конструктор

public Soldier(string name, int health, int damage)

{

name\_ = name;

health\_ = health;

damage\_ = damage;

}

// Гетери

public string GetName()

{

return name\_;

}

public int GetHealth()

{

return health\_;

}

public int GetDamage()

{

return damage\_;

}

// Сетери

public void SetName(string name)

{

name\_ = name;

}

public void SetHealth(int health)

{

health\_ = health;

}

public void SetDamage(int damage)

{

damage\_ = damage;

}

}

}

1. **Інформація про структури даних**

* Бінарне дерево – структура даних, що базується на представленні даних у вигляді дерева, сортуючі менші значення елементів – вліво, більші – вправо, утворюючи «листя» з елементів.
  + Переваги: швидкий пошук.
  + Недоліки: додавання-видалення є складним.
* Стек зв’язний – структура даних, у якій є список елементів, що організовується таким чином, що елементи, додані останніми, видаляються першими.
  + Переваги: швидке додавання, видалення.
  + Недоліки: поганий пошук.
* Стек послідовний – структура даних, у якій реалізація базується на основі масиву.
  + Переваги: швидке додавання та видалення елементів у неповному масиві
  + Недоліки: поганий пошук, складне додавання елементів у повному масиві

1. **Критерії вибору мови програмування**

Було обрано мову С++ через ту причину, що завдяки їй можна вправно оперувати пам’яттю пристрою, а також дана мова є об’єктно-орієнтованою, що дозволить розробити класи перелічених структур даних.

1. **Текст програми**

*Main.cpp*

#include"pch.h"

#include"BinaryTree.h"

#include"CoherentStack.h"

#include"SequentialStack.h"

staticstd::stringprogramText=

    "usingSystem;\n"

    "namespaceTestProgram\n"

    "{\n"

    "// КласSoldier\n"

    "classSoldier\n"

    "{\n"

    "// Атрибутикласу\n"

    "privatestringname\_;\n"

    "privateinthealth\_;\n"

    "privateintdamage\_;\n"

    "// Конструктор\n"

    "publicSoldier(stringname, inthealth, intdamage)\n"

    "{\n"

    "name\_ = name;\n"

    "health\_ = health;\n"

    "damage\_ = damage;\n"

    "}\n"

    "// Гетери\n"

    "publicstringGetName()\n"

    "{\n"

    "returnname\_;\n"

    "}\n"

    "publicintGetHealth()\n"

    "{\n"

    "returnhealth\_;\n"

    "}\n"

    "publicintGetDamage()\n"

    "{\n"

    "returndamage\_;\n"

    "}\n"

    "// Cетери\n"

    "publicvoidSetName(stringname)\n"

    "{\n"

    "name\_ = name;\n"

    "}\n"

    "publicvoidSetHealth(inthealth)\n"

    "{\n"

    "health\_ = health;\n"

    "}\n"

    "publicintSetDamage()\n"

    "{\n"

    "damage\_ = damage;\n"

    "}\n"

    "}\n"

    "}"

    ;

staticBinaryTree\*tree=newBinaryTree;

staticCoherentStack\*coherentStack=newCoherentStack;

staticSequentialStack\*sequentialStack=newSequentialStack;

staticconststd::vector<std::string>delimiters{",",";","{","}","(",")","[","]",":"};

boolIsDelimiter(conststd::stringtoken)

{

    BinaryTreetrees;

    if(token.size()!=1)

    {

        throw"Notsinglesymbol";

    }

    for(constautodelimiter:delimiters)

    {

        if(token.find(delimiter)!=std::string::npos)

        {

            returntrue;

        }

    }

    returnfalse;

}

voidShowStructsList()

{

    std::cout<<std::endl;

    std::cout<<"[0] - Exit"<<std::endl;

    std::cout<<"[1] - BinaryTree"<<std::endl;

    std::cout<<"[2] - CoherentStack"<<std::endl;

    std::cout<<"[3] - SequentialStack"<<std::endl;

}

voidShowStructActions()

{

    std::cout<<std::endl;

    std::cout<<"[0] - Exit"<<std::endl;

    std::cout<<"[1] - Add"<<std::endl;

    std::cout<<"[2] - Remove"<<std::endl;

    std::cout<<"[3] - Check elementexistance"<<std::endl;

    std::cout<<"[4] - Proceedexamplecodetext"<<std::endl;

    std::cout<<"[5] - Show"<<std::endl;

}

voidProceedBinaryTree()

{

    intinput=0;

    do

    {

        ShowStructActions();

        std::cout<<std::endl;

        std::cout<<"Select: ";

        std::cin>>input;

        switch(input)

        {

            case0:

            {

                break;

            }

            case1:

            {

                chardelimiter{};

                do

                {

                    std::cout<<std::endl;

                    std::cout<<"Enter a delimiter: ";

                    std::cin>>delimiter;

                }while(!IsDelimiter(std::string{delimiter}));

                // Check whetherbinarytreeisalreadycontainingthespecifieddelimiter

                if(tree->IsContain(delimiter))

                {

                    std::cout<<std::endl<<"The specifieddelimiteralreadyexists"<<std::endl;

                }

                else

                {

                    tree->AddElement(delimiter);

                    std::cout<<std::endl<<"Elementhasbeenaddedsuccessfully!"<<std::endl;

                }

                break;

            }

            case2:

            {

                chardelimiter{};

                do

                {

                    std::cout<<std::endl;

                    std::cout<<"Enter a delimiter: ";

                    std::cin>>delimiter;

                }while(!IsDelimiter(std::string{delimiter}));

                if(tree->DeleteElement(delimiter))

                {

                    std::cout<<std::endl<<"The elementhasbeendeleted"<<std::endl;

                }

                else

                {

                    std::cout<<std::endl<<"Couldnotdeletetheelement"<<std::endl;

                }

                break;

            }

            case3:

            {

                chardelimiter{};

                do

                {

                    std::cout<<std::endl;

                    std::cout<<"Enter a delimiter: ";

                    std::cin>>delimiter;

                }while(!IsDelimiter(std::string{delimiter}));

                // Check whetherbinarytreeisalreadycontainingthespecifieddelimiter

                if(tree->IsContain(delimiter))

                {

                    std::cout<<std::endl<<"The specifieddelimiteralreadyexists"<<std::endl;

                }

                else

                {

                    std::cout<<std::endl<<"No suchdelimiterisfound"<<std::endl;

                }

                break;

            }

            case4:

            {

                uint16\_tadds=0;

                uint16\_tsearches=0;

                for(int i =0; i <programText.size(); i++)

                {

                    if(IsDelimiter({programText.at(i)}))

                    {

                        searches++;

                        if(!tree->IsContain({programText.at(i)}))

                        {

                            tree->AddElement({programText.at(i)});

                            adds++;

                        }

                    }

                }

                std::cout<<"Searchesdone: "<<searches<<std::endl;

                std::cout<<"Addsdone: "<<adds<<std::endl;

                std::cout<<std::endl<<"Current treestate: "<<std::endl;

                tree->OutputTree();

                break;

            }

            case5:

            {

                tree->OutputTree();

                break;

            }

        }

    }while(input!=0);

}

voidProceedCoherentStack()

{

    intinput=0;

    do

    {

        ShowStructActions();

        std::cout<<std::endl;

        std::cout<<"Select: ";

        std::cin>>input;

        switch(input)

        {

            case0:

            {

                break;

            }

            case1:

            {

                chardelimiter{};

                do

                {

                    std::cout<<std::endl;

                    std::cout<<"Enter a delimiter: ";

                    std::cin>>delimiter;

                }while(!IsDelimiter(std::string{delimiter}));

               // Check whethercoherentstackisalreadycontainingthespecifieddelimiter

                if(coherentStack->IsContain(delimiter))

                {

                    std::cout<<std::endl<<"The specifieddelimiteralreadyexists"<<std::endl;

                }

                else

                {

                    coherentStack->AddElement(delimiter);

                    std::cout<<std::endl<<"Elementhasbeenaddedsuccessfully!"<<std::endl;

                }

                break;

            }

            case2:

            {

                if(coherentStack->DeleteElement())

                {

                    std::cout<<std::endl<<"The elementhasbeendeleted"<<std::endl;

                }

                else

                {

                    std::cout<<std::endl<<"Couldnotdeletetheelement"<<std::endl;

                }

                break;

            }

            case3:

            {

                chardelimiter{};

                do

                {

                    std::cout<<std::endl;

                    std::cout<<"Enter a delimiter: ";

                    std::cin>>delimiter;

                }while(!IsDelimiter(std::string{delimiter}));

                // Check whethercoherentstackisalreadycontainingthespecifieddelimiter

                if(coherentStack->IsContain(delimiter))

                {

                    std::cout<<std::endl<<"The specifieddelimiteralreadyexists"<<std::endl;

                }

                else

                {

                    std::cout<<std::endl<<"No suchdelimiterisfound"<<std::endl;

                }

                break;

            }

            case4:

            {

                uint16\_tadds=0;

                uint16\_tsearches=0;

                for(int i =0; i <programText.size(); i++)

                {

                    if(IsDelimiter({programText.at(i)}))

                    {

                        searches++;

                        if(!coherentStack->IsContain({programText.at(i)}))

                        {

                            coherentStack->AddElement({programText.at(i)});

                            adds++;

                        }

                    }

                }

                std::cout<<"Searchesdone: "<<searches<<std::endl;

                std::cout<<"Addsdone: "<<adds<<std::endl;

                std::cout<<std::endl<<"Current stackstate: "<<std::endl;

                coherentStack->OutputElements();

                break;

            }

            case5:

            {

                coherentStack->OutputElements();

                break;

            }

        }

    }while(input!=0);

}

voidProceedSequentialStack()

{

    intinput=0;

    do

    {

        std::cout<<std::endl;

        std::cout<<"Select: ";

        std::cin>>input;

        switch(input)

        {

            case1:

            {

                chardelimiter{};

                do

                {

                    std::cout<<std::endl;

                    std::cout<<"Enter a delimiter: ";

                    std::cin>>delimiter;

                }while(!IsDelimiter(std::string{delimiter}));

               // Check whethersequentialstackisalreadycontainingthespecifieddelimiter

                if(sequentialStack->IsContain(delimiter))

                {

                    std::cout<<std::endl<<"The specifieddelimiteralreadyexists"<<std::endl;

                }

                else

                {

                    sequentialStack->AddElement(delimiter);

                    std::cout<<std::endl<<"Elementhasbeenaddedsuccessfully!"<<std::endl;

                }

                break;

            }

            case2:

            {

                if(sequentialStack->DeleteElement())

                {

                    std::cout<<std::endl<<"The elementhasbeendeleted"<<std::endl;

                }

                else

                {

                    std::cout<<std::endl<<"Couldnotdeletetheelement"<<std::endl;

                }

                break;

            }

            case3:

            {

                chardelimiter{};

                do

                {

                    std::cout<<std::endl;

                    std::cout<<"Enter a delimiter: ";

                    std::cin>>delimiter;

                }while(!IsDelimiter(std::string{delimiter}));

                // Check whethersequentialstackisalreadycontainingthespecifieddelimiter

                if(sequentialStack->IsContain(delimiter))

                {

                    std::cout<<std::endl<<"The specifieddelimiteralreadyexists"<<std::endl;

                }

                else

                {

                    std::cout<<std::endl<<"No suchdelimiterisfound"<<std::endl;

                }

                break;

            }

            case4:

            {

                uint16\_tadds=0;

                uint16\_tsearches=0;

                for(int i =0; i <programText.size(); i++)

                {

                    if(IsDelimiter({programText.at(i)}))

                    {

                        searches++;

                        if(!sequentialStack->IsContain({programText.at(i)}))

                        {

                            sequentialStack->AddElement({programText.at(i)});

                            adds++;

                        }

                    }

                }

                std::cout<<"Searchesdone: "<<searches<<std::endl;

                std::cout<<"Addsdone: "<<adds<<std::endl;

                std::cout<<std::endl<<"Current stackstate: "<<std::endl;

                sequentialStack->OutputElements();

                break;

            }

            case5:

                sequentialStack->OutputElements();

                break;

        }

    }while(input!=0);

}

intmain()

{

    intinput=0;

    do

    {

        ShowStructsList();

        std::cout<<std::endl;

        std::cout<<"Select: ";

        std::cin>>input;

        switch(input)

        {

            case1:

                ProceedBinaryTree();

                break;

            case2:

                ProceedCoherentStack();

                break;

            case3:

                ProceedSequentialStack();

                break;

        }

    }while(input!=0);

}

*BinaryTree.h*

#pragmaonce

#include"pch.h"

classBinaryTree

{

    classNode

    {

    public:

        chardata;

        Node\*left=nullptr;

        Node\*right=nullptr;

    };

    //

    // Public interface

    //

public:

    //! Addsnewelementtothetree.

    voidAddElement(chartoken);

    //! Deletesexistingelemnt. Ifnoelementexists - returnsfalse.

    boolDeleteElement(chartoken);

    //! Checkswhethertheelementexists. Returnsfalseifnot. Otherwise - true.

    boolIsContain(chartoken)const;

    //! Showswholetree.

    voidOutputTree()const;

    //

    // Private methods

    //

private:

    //! Showsthespecificnode'sdata.

    voidOutputElement(Node\*node)const;

    //! Movesnodeafterdeletion.

    voidMoveNodeAfterDelete(Node\*node);

    //

    // Private members

    //

private:

    Node\*root=nullptr;

};

*SequentialStack.h*

#pragmaonce

#include"pch.h"

classSequentialStack

{

    //

    // Public interface

    //

public:

    //! Addsnewelement.

    voidAddElement(chartoken);

    //! Deletesexistingelement. Ifnoelementexists - returnsfalse.

    boolDeleteElement();

    //! Checkswhetherstackcontainsanelement. Returnstrueifyes, false - otherwise.

    boolIsContain(chartoken);

    //! Showsalltheelementsofthestack.

    voidOutputElements();

    //

    // Private members

    //

private:

    intcurrSize=10;

    intcurrIndex=0;

    char\*massive=newchar[currSize];

};

*CoherentStack.h*

#pragmaonce

#include"pch.h"

classCoherentStack

{

    classNode

    {

    public:

        chardata;

        Node\*lowerElem=nullptr;

    };

    //

    // Public interface

    //

public:

    //! Addsnewelement.

    voidAddElement(chartoken);

    //! Checkswhetherstackcontainsanelement. Returnstrueifyes, false - otherwise.

    boolIsContain(chartoken);

    //! Deletesexistingelement. Ifnoelementexists - returnsfalse.

    boolDeleteElement();

    //! Showsalltheelementsofthestack.

    voidOutputElements();

    //

    // Private members

    //

private:

    Node\*upperElem=nullptr;

};

*pch.h*

#pragmaonce

#include<iostream>

#include<string>

#include<vector>

*BinaryTree.cpp*

#include"BinaryTree.h"

voidBinaryTree::AddElement(chartoken)

{

    if(root==nullptr)

    {

        root=newNode();

        root->data=token;

        root->right=root->left=nullptr;

        return;

    }

    Node\*temp=root;

    do

    {

        if(temp->data<token)

        {

            if(temp->left!=nullptr)

            {

                temp=temp->left;

                continue;

            }

            temp->left=newNode();

            temp->left->data=token;

            temp->left->left=temp->left->right=nullptr;

            return;

        }

        if(temp->right!=nullptr)

        {

            temp=temp->right;

        }

        else

        {

            temp->right=newNode();

            temp->right->data=token;

            temp->right->left=temp->right->right=nullptr;

            return;

        }

    }while(true);

}

boolBinaryTree::DeleteElement(chartoken)

{

    if(root==nullptr)

    {

        returnfalse;

    }

    if(root->data==token)

    {

        Node\*leftSideBranch=root->left;

        root=root->right;

        MoveNodeAfterDelete(leftSideBranch);

        returntrue;

    }

    Node\*temp=root;

    do

    {

        if(temp->data<token)

        {

            if(temp->left==nullptr)

            {

                returnfalse;

            }

            if(temp->left->data!=token)

            {

                temp=temp->left;

                continue;

            }

            else

            {

                Node\*leftSideBranch=temp->left->left;

                temp->left=temp->left->right;

                MoveNodeAfterDelete(leftSideBranch);

                returntrue;

            }

        }

        if(temp->right==nullptr)

        {

            returnfalse;

        }

        if(temp->right->data!=token)

        {

            temp=temp->right;

        }

        else

        {

            Node\*leftSideBranch=temp->right->left;

            temp->right=temp->right->right;

            MoveNodeAfterDelete(leftSideBranch);

            returntrue;

        }

    }while(true);

}

boolBinaryTree::IsContain(chartoken)const

{

    Node\*temp=root;

    do

    {

        if(temp==nullptr)

        {

            returnfalse;

        }

        if(temp->data==token)

        {

            returntrue;

        }

        if(temp->data<token)

        {

            temp=temp->left;

        }

        else

        {

            temp=temp->right;

        }

    }while(true);

}

voidBinaryTree::OutputTree()const

{

    OutputElement(root);

}

voidBinaryTree::OutputElement(Node\*node)const

{

    if(node==nullptr)

    {

        return;

    }

    OutputElement(node->left);

    std::cout<<node->data;

    OutputElement(node->right);

}

voidBinaryTree::MoveNodeAfterDelete(Node\*node)

{

    if(node==nullptr)

    {

        return;

    }

    AddElement(node->data);

    MoveNodeAfterDelete(node->left);

    MoveNodeAfterDelete(node->right);

}

*SequentialStack.cpp*

#include<stdio.h>

#include"SequentialStack.h"

voidSequentialStack::AddElement(chartoken)

{

    if(currIndex<currSize)

    {

        massive[currIndex]=token;

        currIndex++;

    }

    else

    {

        char\*newMassive=newchar[currSize+1];

        for(inti=0;i<currSize;i++)

        {

            newMassive[i]=massive[i];

        }

        newMassive[currSize]=token;

        massive=newMassive;

        currSize++;

        currIndex++;

    }

}

boolSequentialStack::DeleteElement()

{

    if(currIndex<=0)

    {

        returnfalse;

    }

    currIndex--;

    returntrue;

 }

boolSequentialStack::IsContain(chartoken)

{

    intn=0;

    char\*temp=newchar[currSize];

    while(currIndex>0)

    {

        if(massive[currIndex-1]!=token)

        {

            temp[n]=massive[currIndex-1];

            n++;

            currIndex--;

            continue;

    }

    while(n>0)

    {

        massive[currIndex]=temp[n-1];

        currIndex++;

        n--;

    }

    returntrue;

    }

    while(n>0)

    {

        massive[currIndex]=temp[n-1];

        currIndex++;

        n--;

    }

    returnfalse;

}

voidSequentialStack::OutputElements()

{

    char\*temp=newchar[currSize];

    intn=0;

    while(currIndex>0)

    {

        std::cout<<massive[currIndex-1]<<std::endl;

        temp[n]=massive[currIndex-1];

        n++;

        currIndex--;

    }

    while(n>0)

    {

        massive[currIndex]=temp[n-1];

        currIndex++;

        n--;

    }

}

*CoherentStack.cpp*

#include"CoherentStack.h"

voidCoherentStack::AddElement(chartoken)

{

    Node\*newNode=newNode;

    newNode->data=token;

    newNode->lowerElem=upperElem;

    upperElem=newNode;

}

boolCoherentStack::DeleteElement()

{

    if(upperElem==nullptr)

    {

        returnfalse;

    }

    upperElem=upperElem->lowerElem;

    returntrue;

}

boolCoherentStack::IsContain(chartoken)

{

    if(upperElem==nullptr)

    {

        returnfalse;

    }

    if(upperElem->data==token)

    {

        returntrue;

    }

    Node\*temp=newNode;

    temp->data=upperElem->data;

    upperElem=upperElem->lowerElem;

    while(upperElem!=nullptr)

    {

        if(upperElem->data==token)

        {

            while(temp!=nullptr)

            {

                Node\*newNode=newNode;

                newNode->data=temp->data;

                newNode->lowerElem=upperElem;

                upperElem=newNode;

                temp=temp->lowerElem;

            }

            returntrue;

        }

        Node\*newNode=newNode;

        newNode->data=upperElem->data;

        newNode->lowerElem=temp;

        temp=newNode;

        upperElem=upperElem->lowerElem;

    }

    while(temp!=nullptr)

    {

        Node\*newNode=newNode;

        newNode->data=temp->data;

        newNode->lowerElem=upperElem;

        upperElem=newNode;

        temp=temp->lowerElem;

    }

    returnfalse;

  }

voidCoherentStack::OutputElements()

{

    Node\*temp=nullptr;

    while(upperElem!=nullptr)

    {

        std::cout<<upperElem->data<<std::endl;

        Node\*newNode=newNode;

        newNode->data=upperElem->data;

        newNode->lowerElem=temp;

        temp=newNode;

        upperElem=upperElem->lowerElem;

    }

    while(temp!=nullptr)

    {

        Node\*newNode=newNode;

        newNode->data=temp->data;

        newNode->lowerElem=upperElem;

        upperElem=newNode;

        temp=temp->lowerElem;

    }

}

*CMakeLists.txt*

cmake\_minimum\_required(VERSION 3.0.0)

project(LR3 VERSION 0.1.0)

add\_executable(LR3

    main.cpp

    BinaryTree.h

    BinaryTree.cpp

    SequentialStack.h

    SequentialStack.cpp

    CoherentStack.h

    CoherentStack.cpp

)

set(CPACK\_PROJECT\_NAME ${PROJECT\_NAME})

set(CPACK\_PROJECT\_VERSION ${PROJECT\_VERSION})

include(CPack)

1. **Підрахуноккількостівиконанихоператорів**
2. **Обробкабінарного дерева.**

**Найкращим**випадком для бінарного дерева буде той випадок, коли бінарне дерево буде пустим.

**Середнім**випадком для бінарного дерева буде той випадок, коли бінарне дерево збалансоване.

**Найгіршим**випадком для бінарного дерева буде той випадок, коли елементибінарного дерева будутьзнаходитисяу порядкузростанняабоспадання, і таким чином дерево матимевигляд списку.

* 1. **Операціядодавання**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| if(root==nullptr)      {          root=newNode();          root->data=token;          root->right=root->left=nullptr;          return;      }      Node\*temp=root;      do      {          if(temp->data<token)          {              if(temp->left!=nullptr)              {                  temp=temp->left;                  continue;              }                temp->left=newNode();              temp->left->data=token;              temp->left->left=temp->left->right=nullptr;              return;          }            if(temp->right!=nullptr)          {              temp=temp->right;          }          else          {              temp->right=newNode();              temp->right->data=token;              temp->right->left=temp->right->right=nullptr;              return;          }      }while(true); | 1  1  1  1  1 | 1  1  log(n)  log(n/2)  log(n/2)  log(n/2)  1  1  1  1  log(n/2)  log(n/2)  log(n/2) | 1  1  n  n/2  n/2  n/2  n/2  n/2 - 1  1  1  1  1  n |
| **Сума:** | 5 | log(n) + 5\*log(n/2) + 6 | 4,5\*n + 5 |

* 1. **Операціяпошуку**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| Node\*temp=root;      do      {          if(temp==nullptr)          {              returnfalse;          }          if(temp->data==token)          {              returntrue;          }          if(temp->data<token)          {              temp=temp->left;          }          else          {              temp=temp->right;          }      }while(true); | 1  1  1 | 1  log2(n)  log2(n)  1  log2(n/2)  log2(n/2)  log2(n/2)  log2(n/2)  log2(n) | 1  n  n  1  n/2  n/2  n/2  n/2  n |
| **Сума:** | 3 | 3\*log2(n) + 4\*log2(n/2)  + 2 | 5\*n + 2 |

* 1. **Операціявидалення**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| if(root==nullptr)      {          returnfalse;      }      if(root->data==token)      {          Node\*leftSideBranch=root->left;          root=root->right;          MoveNodeAfterDelete(leftSideBranch);            returntrue;      }      Node\*temp=root;      do      {          if(temp->data<token)          {              if(temp->left==nullptr)              {                  returnfalse;              }              if(temp->left->data!=token)              {                  temp=temp->left;                    continue;              }              else              {                  Node\*leftSideBranch=temp->left->left;                  temp->left=temp->left->right;                  MoveNodeAfterDelete(leftSideBranch);                    returntrue;              }          }            if(temp->right==nullptr)          {              returnfalse;          }            if(temp->right->data!=token)          {              temp=temp->right;          }          else          {              Node\*leftSideBranch=temp->right->left;              temp->right=temp->right->right;              MoveNodeAfterDelete(leftSideBranch);                returntrue;          }      }while(true); | 1  1 | 1  1  1  log(n)  log(n/2)  log(n/2)  log(n/2)  log(n/2)  1  1  1 (2n)  1  log(n/2)  log(n/2)  log(n/2)  log(n) | 1  1  1  n  n/2  n/2  n/2  n/2  1  1  1 (4n)  1  n/2  n/2  n/2  n |
| **Сума:** | 2 | 2\*log(n) + 7\*log(n/2) + 2\*n + 6 | 9,5\*n + 6 |

* 1. **MoveNodeAfterDelete(…)**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| if(node==nullptr)      {          return;      }      AddElement(node->data);      MoveNodeAfterDelete(node->left);      MoveNodeAfterDelete(node->right); | 1  1 | N  n/2  n/2 | 2\*n  n  n |
| **Сума:** | 2 | 2\*n | 4\*n |

1. **Обробказв’язного стеку**

**Найкращих випадків** додавання у зв’язномустеку **не існує**, додавання є константним (додавання елементів тільки у одні конкретну позицію)

**Найкращий випадок** для видалення у зв’язному стеку буде той випадок, коли стек пустий.

**Середній або гірший випадки** для видалення у зв’язномустеку будуть такі випадки, коли є хоча б один елемент.

**Найкращий випадок** для пошуку у зв’язному стеку буде такий випадок, коли стек путсий.

**Середній випадок** для пошуку у зв’язному стеку буде такий випадок, коли елемент по середині стеку.

**Найгірший випадок** для пошуку у зв’язному стеку буде такий випадок, коли елемента не існує (стек не пустий).

* 1. **Операція додавання**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| Node\*newNode=newNode;  newNode->data=token;  newNode->lowerElem=upperElem;  upperElem=newNode; | 1  1  1  1 | 1  1  1  1 | 1  1  1  1 |
| **Сума:** | 4 | 4 | 4 |

* 1. **Операція пошуку**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| if(upperElem==nullptr)      {          returnfalse;      }      if(upperElem->data==token)      {          returntrue;      }      Node\*temp=newNode;      temp->data=upperElem->data;      upperElem=upperElem->lowerElem;      while(upperElem!=nullptr)      {          if(upperElem->data==token)          {              while(temp!=nullptr)              {                  Node\*newNode=newNode;                  newNode->data=temp->data;                  newNode->lowerElem=upperElem;                  upperElem=newNode;                  temp=temp->lowerElem;              }                returntrue;          }          Node\*newNode=newNode;          newNode->data=upperElem->data;          newNode->lowerElem=temp;          temp=newNode;          upperElem=upperElem->lowerElem;      }      while(temp!=nullptr)      {          Node\*newNode=newNode;          newNode->data=temp->data;          newNode->lowerElem=upperElem;          upperElem=newNode;          temp=temp->lowerElem;      }      returnfalse; | 1  1 | 1  1  1  1  1  n/2  n/2  n/2  n/2  n/2  n/2  n/2  n/2  1  n/2  n/2  n/2  n/2  n/2 | 1  1  1  1  1  n  n  n  n  n  n  n  n  n  n  n  n  n  1 |
| **Сума:** | 2 | 6,5\*n + 6 | 13\*n+6 |

* 1. **Операція видалення**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| if(upperElem==nullptr)      {          returnfalse;      }        upperElem=upperElem->lowerElem;      returntrue; | 1  1 | 1  1  1 | 1  1  1 |
| **Сума:** | 2 | 3 | 3 |

1. **Обробка послідовного стеку**

**Найкращий випадок** для додавання у послідовному стеку буде той випадок, коли буде вільне місце.

**Середній або гірший випадки** для додавання у послідовному стеку будуть такі випадки, коли місця немає. Необхідне збільшення масиву.

**Найкращий випадок** для видалення у послідовному стеку буде той випадок, коли стек пустий.

**Середній або гірший випадки** для видалення у послідовному стеку будуть такі випадки, коли є хоча б один елемент.

**Найкращий випадок** для пошуку у послідовному стеку буде такий випадок, коли стек пустий.

**Середній випадок** для пошуку у послідовному стеку буде такий випадок, коли елемент по середині стеку.

**Найгірший випадок** для пошуку у послідовному стеку буде такий випадок, коли елемента не існує (стек не пустий).

* 1. **Операція додавання**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| if(currIndex<currSize)  {      massive[currIndex]=token;      currIndex++;  }  else  {      char\*newMassive=newchar[currSize+1];      for(inti=0;i<currSize;i++)      {          newMassive[i]=massive[i];      }        newMassive[currSize]=token;      massive=newMassive;      currSize++;      currIndex++;  } | 1  1  1 | 1  1  n  n  1  1  1  1 | 1  1  n  n  1  1  1  1 |
| **Сума:** | 3 | 2\*n+6 | 2\*n+6 |

* 1. **Операція пошуку**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| intn=0;  char\*temp=newchar[currSize];  while(currIndex>0)  {      if(massive[currIndex-1]!=token)      {          temp[n]=massive[currIndex-1];          n++;          currIndex--;          continue;      }      while(n>0)      {          massive[currIndex]=temp[n-1];          currIndex++;          n--;      }      returntrue;  }  while(n>0)  {  massive[currIndex]=temp[n-1];        currIndex++;        n--;  }  returnfalse; | 1  1  1  1  1 | 1  1  n/2  n/2  n/2  n/2  n/2  n/2  n/2  n/2  n/2  n/2  1 | 1  1  n  n  n  n  n  n  n  n  n  n  1 |
| **Сума:** | 5 | 5\*n+3 | 10\*n+3 |

* 1. **Операція видалення**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції і оператори | Кількістьвиконанихоператорів | | |
| Найкращийвипадок | Середнійвипадок | Найгіршийвипадок |
| if(currIndex<=0)   {      returnfalse;  }  currIndex--;  returntrue; | 1  1 | 1  1  1 | 1  1  1 |
| **Сума:** | 2 | 3 | 3 |

1. **Зразок програми для аналізу**

|  |  |  |
| --- | --- | --- |
| Текстпрограми | Кількістьвикористань | |
| Операціядодавання | Операціяпошуку |
| using System;  namespace TestProgram  {  // Клас Soldier  class Soldier  {  // Атрибути класу  private string name\_;  private int health\_; private int damage\_;  // Конструктор  public Soldier(string name, int health, int damage)  {  name\_ = name;  health\_ = health;  damage\_ = damage;  }  // Гетери  public string GetName()  {  return name\_;  }  public int GetHealth()  {  return health\_;  }  public int GetDamage()  {  return damage\_;  }  // Сетери  public void SetName(string name)  {  name\_ = name;  }  public void SetHealth(int health)  {  health\_ = health;  }  public void SetDamage(int damage)  {  damage\_ = damage;  }  }  } | 1  1  1  2  1 | 1  1  1  1  1  1  4  1  1  1  1  1  2  1  1  1  2  1  1  1  2  1  1  1  2  1  1  1  2  1  1  1  2  1  1  1  1  1 |
| **Сума:** | 6 | 47 |

1. **Обчислення**

У кінці відбувається зворотнє видалення доданих 6 елементів.

Дерево:

Стек зв’язний:

Стек послідовний:

1. **Результатиобчислень**

|  |  |  |  |
| --- | --- | --- | --- |
| **Структураданих** | **Обчислювальнаскладність** | | |
| **Найкращийвипадок** | **Середнійвипадок** | **Найгіршийвипадок** |
| Бінарне дерево | 3.1 | 8.2 | 35.1 |
| Стекзізв’язнимпредставленням | 2.2 | 36.7 | 67.9 |
| Стекізпослідовнимпредставленням | 2.8 | 27 | 51 |

1. **Висновки**