**Phần 1: ( bt buổi 3)**

using System;

using System.Collections.Generic;

using System.IO;

using System.Text.RegularExpressions;

class InfixToPostfixConverter

{

//Bài tập Stack1

static int Precedence(char op)

{

if (op == '+' || op == '-')

return 1;

else if (op == '\*' || op == '/')

return 2;

else

return 0;

}

static bool IsOperand(char c)

{

return char.IsLetterOrDigit(c);

}

static string ConvertToPostfix(string infix)

{

Stack<char> stack = new Stack<char>();

List<string> output = new List<string>();

for (int i = 0; i < infix.Length; i++)

{

char token = infix[i];

if (IsOperand(token))

{

output.Add(token.ToString());

}

else if (token == '(')

{

stack.Push(token);

}

else if (token == ')')

{

while (stack.Count > 0 && stack.Peek() != '(')

{

output.Add(stack.Pop().ToString());

}

stack.Pop();

}

else

{

while (stack.Count > 0 && Precedence(stack.Peek()) >= Precedence(token))

{

output.Add(stack.Pop().ToString());

}

stack.Push(token);

}

}

while (stack.Count > 0)

{

output.Add(stack.Pop().ToString());

}

return string.Join(" ", output);

}

static void Main()

{

Console.Write("Nhap bieu thuc Infix: ");

string infixExpression = Console.ReadLine();

string postfixExpression = ConvertToPostfix(infixExpression.Replace(" ", ""));

Console.WriteLine("Bieu thuc Postfix: " + postfixExpression);

}

}

class PostfixCalculator

{

//Bài tập Stack2

static double ApplyOperator(double operand1, double operand2, char op)

{

switch (op)

{

case '+':

return operand1 + operand2;

case '-':

return operand1 - operand2;

case '\*':

return operand1 \* operand2;

case '/':

if (operand2 == 0)

throw new DivideByZeroException("Khong the chia cho so 0");

return operand1 / operand2;

default:

throw new InvalidOperationException("Toan tu khong duoc ho tro: " + op);

}

}

static double CalculatePostfix(string expression)

{

Stack<double> stack = new Stack<double>();

foreach (char token in expression)

{

if (char.IsDigit(token))

{

stack.Push(double.Parse(token.ToString()));

}

else if (token == '+' || token == '-' || token == '\*' || token == '/')

{

double operand2 = stack.Pop();

double operand1 = stack.Pop();

double result = ApplyOperator(operand1, operand2, token);

stack.Push(result);

}

}

return stack.Pop();

}

static void Main()

{

Console.Write("Nhap bieu thuc Postfix: ");

string postfixExpression = Console.ReadLine().Replace(" ", "");

double result = CalculatePostfix(postfixExpression);

Console.WriteLine("Ket qua cua bieu thuc Postfix: " + result);

}

}

class HtmlTagValidator

{

//Bài tập Queue

static bool IsOpeningTag(string tag)

{

return !tag.StartsWith("</");

}

static bool IsClosingTag(string tag)

{

return tag.StartsWith("</");

}

static bool ValidateHtmlTags(string htmlContent)

{

Stack<string> tagStack = new Stack<string>();

Regex tagRegex = new Regex(@"</?([a-zA-Z0-9]+)[^>]\*>");

MatchCollection matches = tagRegex.Matches(htmlContent);

foreach (Match match in matches)

{

string tag = match.Value;

if (IsOpeningTag(tag))

{

string tagName = tag.Substring(1, tag.Length - 2);

tagStack.Push(tagName);

}

else if (IsClosingTag(tag))

{

string tagName = tag.Substring(2, tag.Length - 3);

if (tagStack.Count == 0 || tagStack.Peek() != tagName)

{

Console.WriteLine($"Loi: The dong <{tagName}> khong khop the mo.");

return false;

}

tagStack.Pop();

}

}

if (tagStack.Count > 0)

{

Console.WriteLine("Loi: The mo chua duoc dong.");

return false;

}

return true;

}

static void Main()

{

Console.Write("Nhap duong dan den file HTML: ");

string filePath = Console.ReadLine();

try

{

string htmlContent = File.ReadAllText(filePath);

bool isValid = ValidateHtmlTags(htmlContent);

if (isValid)

{

Console.WriteLine("Tat ca the HTML deu hop le.");

}

else

{

Console.WriteLine("Co loi trong the HTML.");

}

}

catch (Exception ex)

{

Console.WriteLine("Loi khi doc file: " + ex.Message);

}

}

}

**Phần 2: (Bài tập buổi 3)**  
using System.Collections.Generic;

using System;

public class Node

{

public Node next;

public object data;

}

public class MyStack

{

Node top;

public bool IsEmpty()

{

return top == null;

}

public void Push(object ele)

{

Node n = new Node();

n.data = ele;

n.next = top;

top = n;

}

public Node Pop()

{

if (top == null)

return null;

Node d = top;

top = top.next;

return d;

}

public int Sum()

{

int sum = 0;

MyStack temp = new MyStack();

while (!this.IsEmpty())

{

object n = this.Pop().data;

temp.Push(n);

sum += (int)n;

}

while (!temp.IsEmpty())

{

object n = temp.Pop().data;

this.Push(n);

}

return sum;

}

public void Swap(int i, int j)

{

int t = Math.Max(i, j);

List<int> list = new List<int>();

int count = 0;

while (count <= t)

{

int d = (int)this.Pop().data;

count++;

list.Add(d);

}

for (int k = 0; k < list.Count; k++)

{

if (k == 0)

this.Push(list[list.Count - 1]);

else if (k == list.Count - 1)

this.Push(list[0]);

else

this.Push(list[k]);

}

}

public void Sort()

{

List<int> list = new List<int>();

while (!this.IsEmpty())

{

list.Add((int)this.Pop().data);

}

list.Sort();

foreach (var item in list)

{

this.Push(item);

}

}

public void Reverse()

{

List<int> list = new List<int>();

while (!this.IsEmpty())

{

list.Add((int)this.Pop().data);

}

for (int i = list.Count - 1; i >= 0; i--)

{

this.Push(list[i]);

}

}

}

public class Node2

{

public Node2 prev, next;

public object data;

}

public class MyQueue

{

Node2 rear, front;

public bool IsEmpty()

{

return rear == null || front == null;

}

public void Enqueue(object ele)

{

Node2 n = new Node2();

n.data = ele;

if (rear == null)

{

rear = n; front = n;

}

else

{

rear.prev = n;

n.next = rear; rear = n;

}

}

public Node2 Dequeue()

{

if (front == null) return null;

Node2 d = front;

front = front.prev;

if (front == null)

rear = null;

else

front.next = null;

return d;

}

public int Sum()

{

int sum = 0;

MyQueue temp = new MyQueue();

while (!this.IsEmpty())

{

int n = (int)this.Dequeue().data;

temp.Enqueue(n);

sum += n;

}

while (!temp.IsEmpty())

this.Enqueue((int)temp.Dequeue().data);

return sum;

}

public void Swap(int i, int j)

{

List<int> list = new List<int>();

while (!this.IsEmpty())

{

list.Add((int)this.Dequeue().data);

}

if (i < list.Count && j < list.Count)

{

int temp = list[i];

list[i] = list[j];

list[j] = temp;

}

foreach (var item in list)

{

this.Enqueue(item);

}

}

public void Sort()

{

List<int> list = new List<int>();

while (!this.IsEmpty())

{

list.Add((int)this.Dequeue().data);

}

list.Sort();

foreach (var item in list)

{

this.Enqueue(item);

}

}

public void Reverse()

{

List<int> list = new List<int>();

while (!this.IsEmpty())

{

list.Add((int)this.Dequeue().data);

}

for (int i = list.Count - 1; i >= 0; i--)

{

this.Enqueue(list[i]);

}

}

}

public class Program

{

public static void Main(string[] args)

{

Console.Clear();

/\*MyStack stack = new MyStack();

System.Console.WriteLine(stack.IsEmpty());

stack.Push(1);

System.Console.WriteLine(stack.IsEmpty());

stack.Push(2);

stack.Push(3);

stack.Push(4);

stack.Push(5);

;

stack.Swap(2, 4);

;\*/

/\*System.Console.WriteLine(stack.IsEmpty());

System.Console.WriteLine("Sum: " + stack.Sum());

System.Console.WriteLine(stack.IsEmpty());\*/

MyQueue queue = new MyQueue();

queue.Enqueue(1);

queue.Enqueue(2);

queue.Enqueue(3);

System.Console.WriteLine(queue.Sum());

}

}

**Bài tập buổi 4:**

using System;

using System.Diagnostics;

class Program

{

static void Main()

{

//Câu 1

int[] arr = { 1, 3, 5, 7, 9, 11, 15, 18, 20, 25 };

int target = 15;

//SeqSearch

MeasureTime(() =>

{

int result = SequentialSearch(arr, target);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Sequential Search");

//RecuSearch

MeasureTime(() =>

{

int result = RecursiveSearch(arr, target, 0);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Recursive Search");

//SenSearch

MeasureTime(() =>

{

int result = SentinelSearch(arr, target);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Sentinel Search");

//BinSearch

MeasureTime(() =>

{

int result = BinarySearch(arr, target);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Binary Search");

}

//Hàm đo thời gian

static void MeasureTime(Action searchMethod, string searchName)

{

Stopwatch stopwatch = Stopwatch.StartNew();

searchMethod();

stopwatch.Stop();

Console.WriteLine($"{searchName} thoi gian thuc hien: {stopwatch.ElapsedTicks} ticks\n");

}

//SeqSearch

static int SequentialSearch(int[] arr, int target)

{

for (int i = 0; i < arr.Length; i++)

{

if (arr[i] == target)

return i;

}

return -1;

}

//RecuSearch

static int RecursiveSearch(int[] arr, int target, int index)

{

if (index >= arr.Length) return -1;

if (arr[index] == target) return index;

return RecursiveSearch(arr, target, index + 1);

}

//SenSearch

static int SentinelSearch(int[] arr, int target)

{

int last = arr[arr.Length - 1];

arr[arr.Length - 1] = target;

int i = 0;

while (arr[i] != target) i++;

arr[arr.Length - 1] = last;

return (i < arr.Length - 1 || arr[arr.Length - 1] == target) ? i : -1;

}

//BinSearch

static int BinarySearch(int[] arr, int target)

{

int left = 0, right = arr.Length - 1;

while (left <= right)

{

int mid = left + (right - left) / 2;

if (arr[mid] == target)

return mid;

if (arr[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

class SinhVien

{

public int Id { get; set; }

public string HoTen { get; set; }

public double DiemTB { get; set; }

public SinhVien(int id, string hoten, double diemtb)

{

Id = id;

HoTen = hoten;

DiemTB = diemtb;

}

}

}

class SinhVien

{

//Câu 2

public int Id { get; set; }

public string HoTen { get; set; }

public double DiemTB { get; set; }

public SinhVien(int id, string hoten, double diemtb)

{

Id = id;

HoTen = hoten;

DiemTB = diemtb;

}

}

class Program

{

static void Main()

{

SinhVien[] svList = {

new SinhVien(101, "Nguyen Van A", 7.5),

new SinhVien(102, "Nguyen Thi B", 8.0),

new SinhVien(103, "phan Van C", 6.5),

new SinhVien(104, "Pham Thi D", 9.0),

new SinhVien(105, "Tran Van E", 7.2)

};

int targetId = 103;

//SeqSearch

MeasureTime(() =>

{

int result = SequentialSearch(svList, targetId);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Sequential Search");

//RecuSearch

MeasureTime(() =>

{

int result = RecursiveSearch(svList, targetId, 0);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Recursive Search");

//SenSearch

MeasureTime(() =>

{

int result = SentinelSearch(svList, targetId);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Sentinel Search");

//BinSearch

Array.Sort(svList, (a, b) => a.Id.CompareTo(b.Id));

MeasureTime(() =>

{

int result = BinarySearch(svList, targetId);

Console.WriteLine(result != -1 ? $"Tim thay tai {result}" : "Khong tim thay");

}, "Binary Search");

}

//Hàm đo thời gian

static void MeasureTime(Action searchMethod, string searchName)

{

Stopwatch stopwatch = Stopwatch.StartNew();

searchMethod();

stopwatch.Stop();

Console.WriteLine($"{searchName} thoi gian thuc hien: {stopwatch.ElapsedTicks} ticks\n");

}

// SeqSearch

static int SequentialSearch(SinhVien[] svList, int targetId)

{

for (int i = 0; i < svList.Length; i++)

{

if (svList[i].Id == targetId)

return i;

}

return -1;

}

// RecuSearch

static int RecursiveSearch(SinhVien[] svList, int targetId, int index)

{

if (index >= svList.Length) return -1;

if (svList[index].Id == targetId) return index;

return RecursiveSearch(svList, targetId, index + 1);

}

// SenSearch

static int SentinelSearch(SinhVien[] svList, int targetId)

{

int last = svList[svList.Length - 1].Id;

svList[svList.Length - 1].Id = targetId;

int i = 0;

while (svList[i].Id != targetId) i++;

svList[svList.Length - 1].Id = last;

return (i < svList.Length - 1 || svList[svList.Length - 1].Id == targetId) ? i : -1;

}

// BinSearch

static int BinarySearch(SinhVien[] svList, int targetId)

{

int left = 0, right = svList.Length - 1;

while (left <= right)

{

int mid = left + (right - left) / 2;

if (svList[mid].Id == targetId)

return mid;

if (svList[mid].Id < targetId)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

}

using System;

public class Program

{

static int SeqSearch(int[] arr, int value)

{

int i = 0;

while (i < arr.Length && arr[i] != value)

i++;

return (i < arr.Length) ? i : -1;

}

static int RecuSearch(int[] arr, int from, int value)

{

if (from >= arr.Length) return -1;

if (arr[from] == value) return from;

return RecuSearch(arr, from + 1, value);

}

//Bài tập 1: Cài đặt SenSearch bằng đệ quy (Github)

static int SenSearchRecursive(int[] arr, int index, int value)

{

if (index >= arr.Length - 1)

return (arr[index] == value) ? index : -1;

if (arr[index] == value)

return index;

return SenSearchRecursive(arr, index + 1, value);

}

static int SenSearch(int[] arr, int value)

{

int last = arr[arr.Length - 1];

arr[arr.Length - 1] = value;

int result = SenSearchRecursive(arr, 0, value);

return result;

}

// Bài tập 2: Cài đặt BinSearch bằng đệ quy (Github)

static int BinSearchRecursive(int[] sarr, int value, int left, int right)

{

if (left > right) return -1;

int mid = (left + right) / 2;

if (sarr[mid] == value)

return mid;

if (sarr[mid] < value)

return BinSearchRecursive(sarr, value, mid + 1, right);

return BinSearchRecursive(sarr, value, left, mid - 1);

}

static int BinSearch(int[] sarr, int value)

{

return BinSearchRecursive(sarr, value, 0, sarr.Length - 1);

}

public static void Main(string[] args)

{

Console.Clear();

int[] arr = { 3, 9, 2, 5, 6 };

int value = 6;

// SeqSearch

int index = SeqSearch(arr, value);

Console.WriteLine($"[SeqSearch] Gia tri {value} o vi tri: {index}");

// RecuSearch

index = RecuSearch(arr, 0, value);

Console.WriteLine($"[RecuSearch] Gia tri {value} o vi tri: {index}");

// SenSearch

index = SenSearch(arr, value);

Console.WriteLine($"[SenSearch] Gia tri {value} o vi tri: {index}");

// BinSearch

int[] sarr = { 2, 3, 5, 6, 9 };

index = BinSearch(sarr, value);

Console.WriteLine($"[BinSearch] Gia tri {value} o vi tri: {index}");

}

}

using System;

class Program

{

//BT1

static int SequentialSearchNthOccurrence(int[] arr, int target, int occurrence)

{

int count = 0;

for (int i = 0; i < arr.Length; i++)

{

if (arr[i] == target)

{

count++;

if (count == occurrence)

return i;

}

}

return -1;

}

static void Main()

{

int[] arr = { 1, 2, 3, 2, 4, 2, 5 };

int target = 2;

int occurrence = 2;

int result = SequentialSearchNthOccurrence(arr, target, occurrence);

Console.WriteLine($"Lan xuat hien thu {occurrence} cua {target} nam o vi tri: {result}");

}

}

using System;

class Program

{

//BT2

static int SequentialSearchLastOccurrence(int[] arr, int target)

{

int lastIndex = -1;

for (int i = 0; i < arr.Length; i++)

{

if (arr[i] == target)

lastIndex = i;

}

return lastIndex;

}

static void Main()

{

int[] arr = { 3, 5, 2, 7, 5, 9, 5 };

int target = 5;

int result = SequentialSearchLastOccurrence(arr, target);

Console.WriteLine($"Lan xuat hien cuoi cung cua {target} nam o vi tri: {result}");

}

}

using System;

class Program

{

//BT3

static int BinarySearch(int[] arr, int value)

{

int L = 0, R = arr.Length - 1;

while (L <= R)

{

int mid = (L + R) / 2;

if (arr[mid] == value)

return mid;

else if (arr[mid] < value)

L = mid + 1;

else

R = mid - 1;

}

return -1;

}

static void Main()

{

int[] arr = { 9, 3, 7, 1, 5, 2 };

int target = 5;

int result = BinarySearch(arr, target);

Console.WriteLine($"Ket qua tim kiem nhi phan tren mang khong sap xep: {result}");

}

}

using System;

class CArray

{

//BT4

public int[] arr;

public int compCount;

public CArray(int size)

{

arr = new int[size];

compCount = 0;

Random rand = new Random();

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

arr[i] = rand.Next(1, 10000);

}

public int SeqSearch(int target)

{

compCount = 0;

for (int i = 0; i < arr.Length; i++)

{

compCount++;

if (arr[i] == target)

return i;

}

return -1;

}

public int BinSearch(int target)

{

compCount = 0;

Array.Sort(arr);

int L = 0, R = arr.Length - 1;

while (L <= R)

{

compCount++;

int mid = (L + R) / 2;

if (arr[mid] == target)

return mid;

else if (arr[mid] < target)

L = mid + 1;

else

R = mid - 1;

}

return -1;

}

}

class Program

{

static void Main()

{

CArray array = new CArray(1000);

int target = 738;

int seqResult = array.SeqSearch(target);

int seqComparisons = array.compCount;

Console.WriteLine($"Sequential Search: {seqResult}, So lan so sanh: {seqComparisons}");

int binResult = array.BinSearch(target);

int binComparisons = array.compCount;

Console.WriteLine($"Binary Search: {binResult}, So lan so sanh: {binComparisons}");

}

}