**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ**

**“ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет информационных технологий

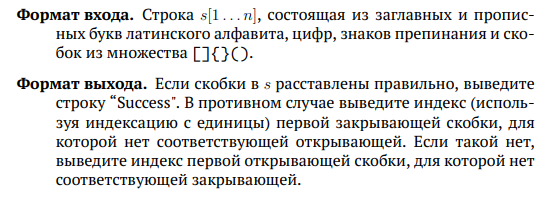
Кафедра технологий программирования

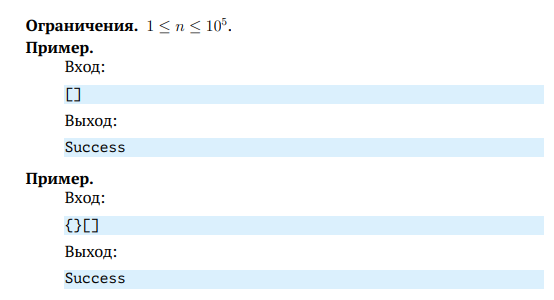
**Алгоритмы: теория и практика. Структуры данных**

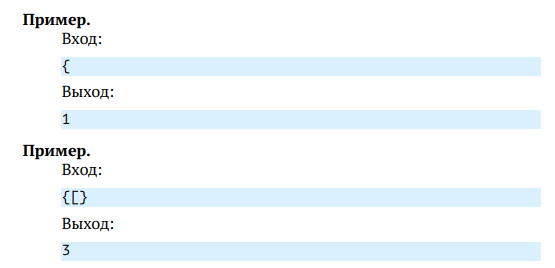
**Модуль №1**

Выполнила студент 2 курса, группа 21-ИТ-1 Макеёнок Д.И

Проверила Виноградова А.Д

Полоцк, 2022 г.





Решение:

using System;

using System.Collections;

public class MainClass

{

public static void Main()

{

var inputString = Console.ReadLine();

int index = 0;

if (isBalanced(inputString, ref index))

Console.WriteLine("Success");

else

Console.WriteLine("{0}", index);

}

private class Pair

{

public char Ch { get; set; }

public int Index { get; set; }

}

public static bool isBalanced(String str, ref int index)

{

Stack st = new Stack();

bool balanced = false;

foreach (var ch in str)

{

index++;

if (ch == '[' || ch == '{' || ch == '(')

{

st.Push(new Pair { Ch = ch, Index = index });

}

else

{

if (ch == ']' || ch == '}' || ch == ')')

{

if (st.Count == 0) return false;

Pair top = (Pair)st.Pop();

if ((top.Ch == '(' && ch != ')') ||

(top.Ch == '[' && ch != ']') ||

(top.Ch == '{' && ch != '}'))

return false;

}

else

continue;

}

}

balanced = st.Count == 0;

if (!balanced)

{

Pair top = (Pair)st.Pop();

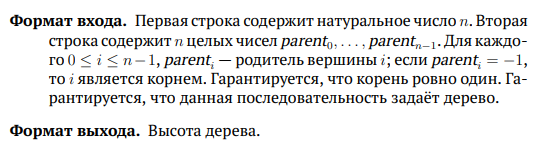
index = top.Index;

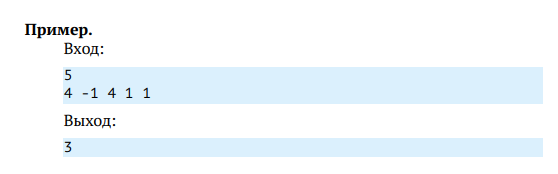
}

return balanced;

}

}





Решение

using System;

using System.Collections.Generic;

namespace HeightTree

{

class Program

{

static void Main(string[] args)

{

var n = Convert.ToInt32(Console.ReadLine());

var nodeArray = GetArray(Console.ReadLine());

var node = new Node[n];

int root = -1;

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

{

int cn = nodeArray[i];

if (cn == -1)

root = i;

else if (node[cn] == null)

node[cn] = new Node(i);

else node[cn].children.Add(i);

}

int size = node[root].GetHeight(node);

Console.WriteLine(size);

//Console.ReadKey();

}

private static int[] GetArray(string v)

{

var numbers = v.Split(' ');

var array = new int[numbers.Length];

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

array[i] = int.Parse(numbers[i]);

return array;

}

}

class Node

{

public List<int> children;

public Node(int child)

{

children = new List<int>() { child };

}

internal int GetHeight(Node[] node)

{

int s = 0;

foreach (var nods in children)

if (node[nods] != null)

s = Math.Max(s, node[nods].GetHeight(node));

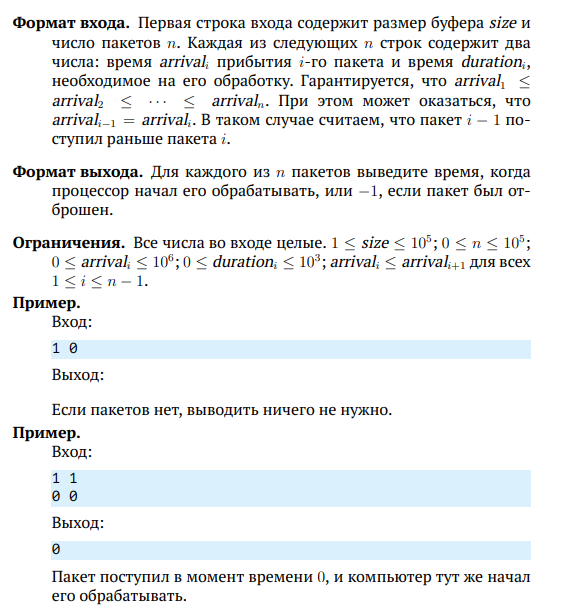
else s = Math.Max(s, 1); ;

return s + 1;

}

}

}



Решение

#include <iostream>

#include <list>

int main()

{

std::list<int> buffer{ };

int start = 0, size = 0, n = 0, arrival = 0, duration = 0;

std::cin >> size >> n;

while (std::cin >> arrival >> duration)

{

start = start < arrival ? arrival : start;

while (!buffer.empty() && arrival >= buffer.front())

{

buffer.pop\_front();

}

int iRes = -1;

if (buffer.size() < size)

{

iRes = start;

start += duration;

buffer.push\_back(start);

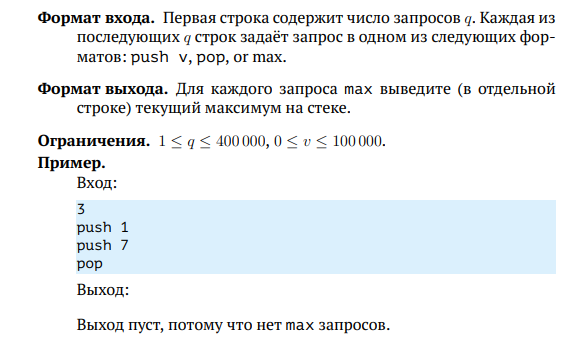
}

std::cout << iRes << '\n'; // print

}

return 0;

}



Решение

using System;

using System.Collections.Generic;

namespace StackMax

{

class Program

{

static void Main(string[] args)

{

var stack = new Stack<int>();

var countLines = Int32.Parse(Console.ReadLine());

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

{

var command = Console.ReadLine();

Process(command, stack);

}

}

private static void Process(string command, Stack<int> stack)

{

var words = command.Split(' ');

if (words[0] == "push")

stack.Push(Int32.Parse(words[1]));

else if (words[0] == "pop" && !stack.Empty)

stack.Pop();

else if (words[0] == "max" && !stack.Empty)

Console.WriteLine(stack.Max);

return;

}

}

public class Stack<T> where T : IComparable<T>

{

LinkedList<Contain<T>> item = new LinkedList<Contain<T>>();

public void Push(T value)

{

item.AddLast(new Contain<T>(value, Maximum(value)));

}

private T Maximum(T val)

{

if (!this.Empty && val.CompareTo(item.Last.Value.Maximum) < 0)

return item.Last.Value.Maximum;

else

return val;

}

public T Pop()

{

if (this.Empty)

{

throw new InvalidOperationException("The stack is empty");

}

T result = item.Last.Value.Item;

item.RemoveLast();

return result;

}

public T Peek()

{

if (this.Empty)

{

throw new InvalidOperationException("The stack is empty");

}

return item.Last.Value.Item;

}

public int Count

{

get

{

return item.Count;

}

}

public bool Empty

{

get

{

return item.Count == 0;

}

}

private T max;

public T Max

{

get

{

if (item.Count != 0)

return item.Last.Value.Maximum;

else

return default(T);

}

set

{

if (item.Count != 0 || max.CompareTo(value) == 0)

max = value;

}

}

public struct Contain<T> where T : IComparable<T>

{

private T item;

private T maximum;

public T Item

{

get { return item; }

set { item = value; }

}

public T Maximum

{

get { return maximum; }

set { maximum = value; }

}

public Contain(T val1, T val2)

{

item = val1;

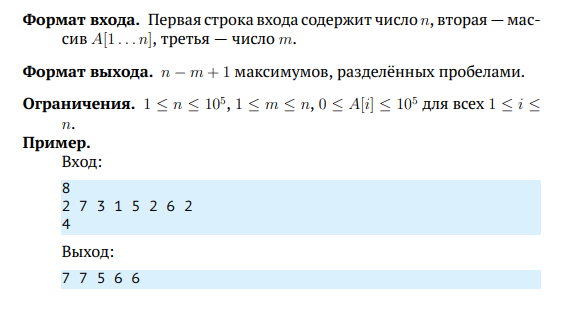
maximum = val2;

}

}

}

}



using System;

using System.Collections.Generic;

namespace StackMax

{

class Program

{

static void Main(string[] args)

{

var stack = new Stack<int>();

var countLines = Int32.Parse(Console.ReadLine());

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

{

var command = Console.ReadLine();

Process(command, stack);

}

}

private static void Process(string command, Stack<int> stack)

{

var words = command.Split(' ');

if (words[0] == "push")

stack.Push(Int32.Parse(words[1]));

else if (words[0] == "pop" && !stack.Empty)

stack.Pop();

else if (words[0] == "max" && !stack.Empty)

Console.WriteLine(stack.Max);

return;

}

}

public class Stack<T> where T : IComparable<T>

{

LinkedList<Contain<T>> item = new LinkedList<Contain<T>>();

public void Push(T value)

{

item.AddLast(new Contain<T>(value, Maximum(value)));

}

private T Maximum(T val)

{

if (!this.Empty && val.CompareTo(item.Last.Value.Maximum) < 0)

return item.Last.Value.Maximum;

else

return val;

}

public T Pop()

{

if (this.Empty)

{

throw new InvalidOperationException("The stack is empty");

}

T result = item.Last.Value.Item;

item.RemoveLast();

return result;

}

public T Peek()

{

if (this.Empty)

{

throw new InvalidOperationException("The stack is empty");

}

return item.Last.Value.Item;

}

public int Count

{

get

{

return item.Count;

}

}

public bool Empty

{

get

{

return item.Count == 0;

}

}

private T max;

public T Max

{

get

{

if (item.Count != 0)

return item.Last.Value.Maximum;

else

return default(T);

}

set

{

if (item.Count != 0 || max.CompareTo(value) == 0)

max = value;

}

}

public struct Contain<T> where T : IComparable<T>

{

private T item;

private T maximum;

public T Item

{

get { return item; }

set { item = value; }

}

public T Maximum

{

get { return maximum; }

set { maximum = value; }

}

public Contain(T val1, T val2)

{

item = val1;

maximum = val2;

}

}

}

}