**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ**

**“ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет информационных технологий

Кафедра технологий программирования

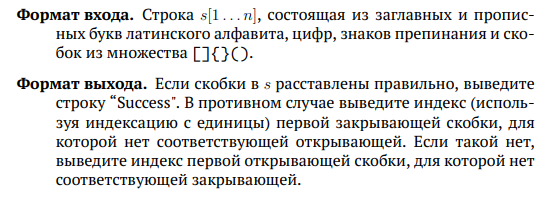
**Лабораторная работа №2**

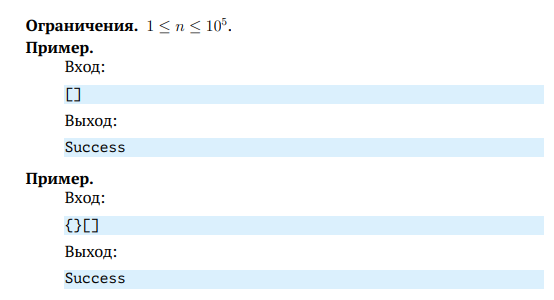
Выполнила студент 2 курса, группа 21-ИТ-1 Макеёнок Д.И.

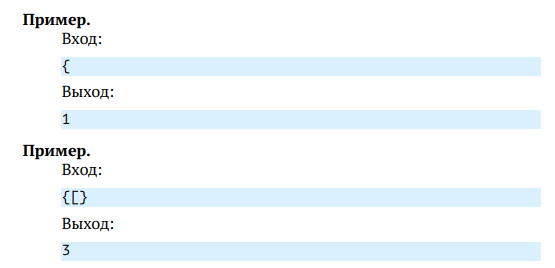
Проверила Виноградова А.Д.

Полоцк, 2022 г.

**Модуль 1**







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();

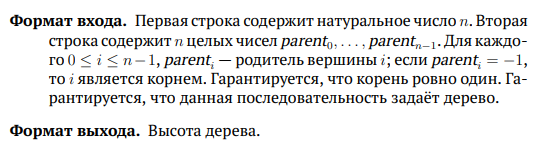
index = top.Index;

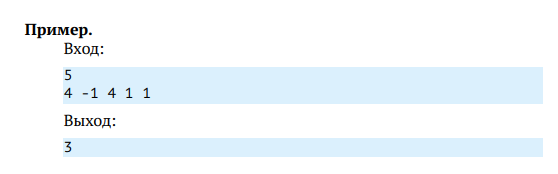
}

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));

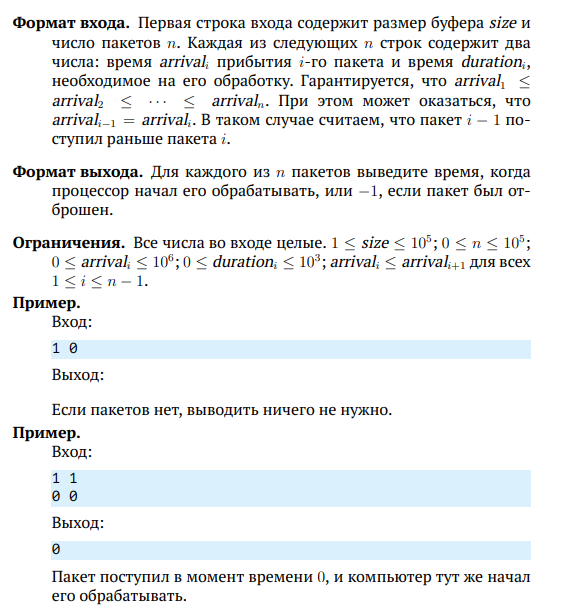
else s = Math.Max(s, 1); ;

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);

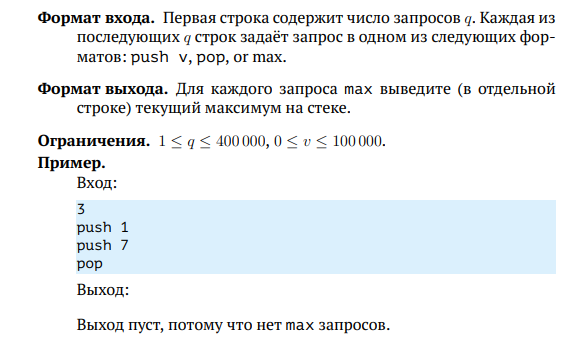
}

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;

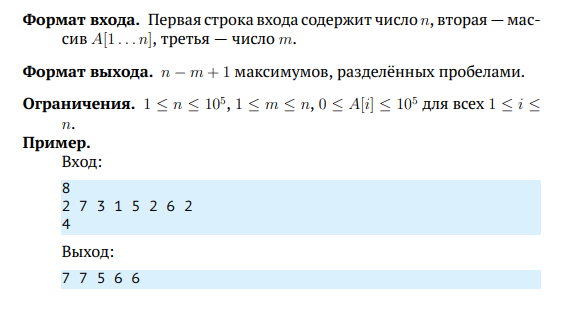
maximum = val2;

}

}

}

}



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;

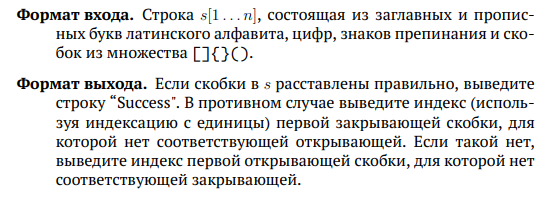
}

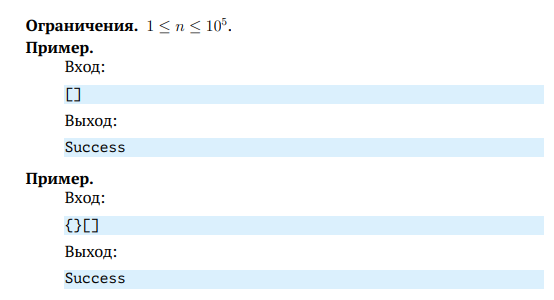
}

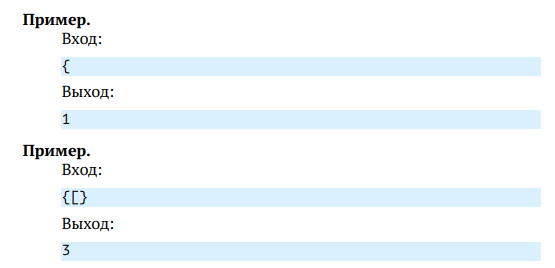
}

}

**Модуль 2**







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();

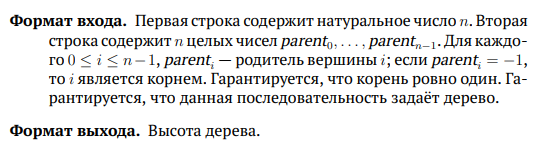
index = top.Index;

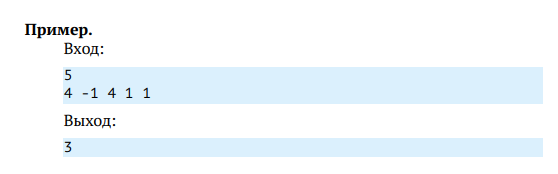
}

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));

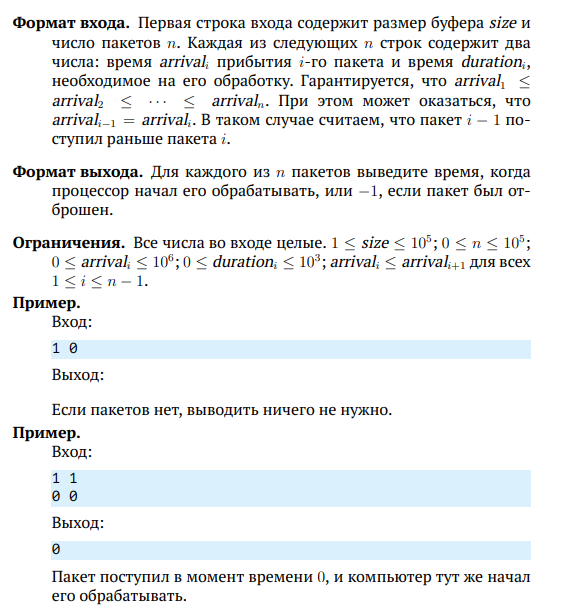
else s = Math.Max(s, 1); ;

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);

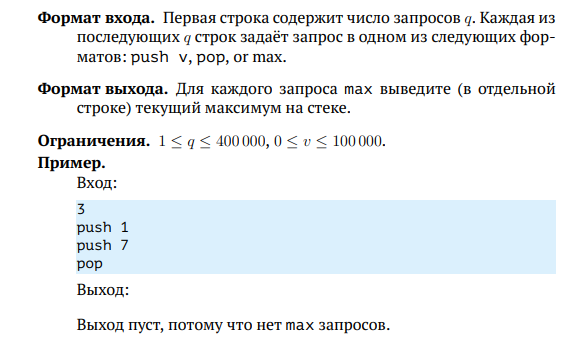
}

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;

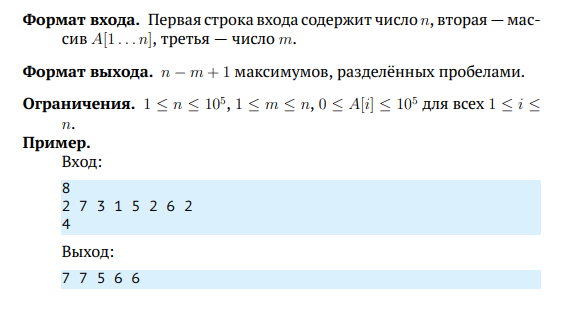
maximum = val2;

}

}

}

}



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;

}

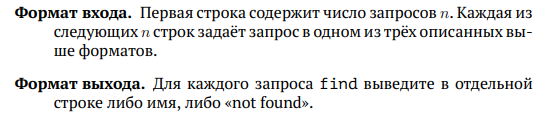
}

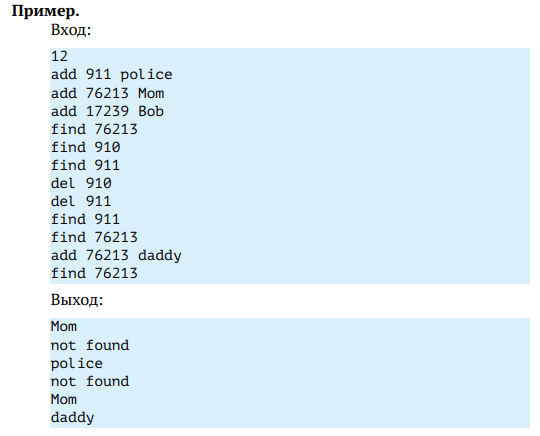
}

}

**Модуль 3**

Телефонная книга





Решение

#include <iostream>

#include <string>

#include <map>

using namespace std;

int main()

{

long n, key;

string cmd, value;

map<long, string> db;

cin >> n;

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

{

cin >> cmd;

if (cmd == "add")

{

cin >> key >> value;

db[key] = value;

}

else if (cmd == "find")

{

cin >> key;

if (db.find(key) != db.end()) { cout << db[key] << endl; }

else { cout << "not found" << endl; }

}

else if (cmd == "del")

{

cin >> key;

db.erase(key);

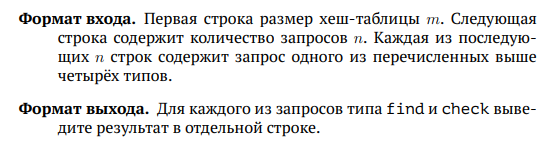
}

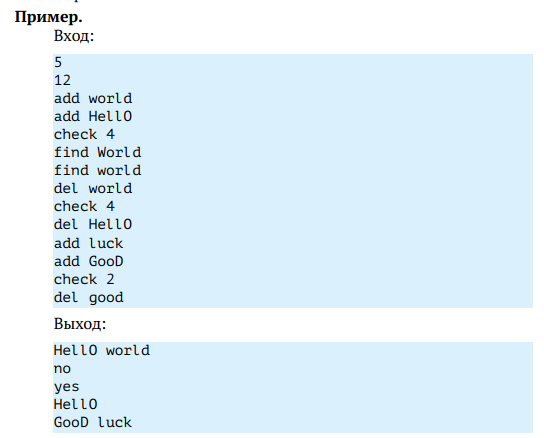
}

return 0;

}

Хеширование цепочками





Решение

using System;

using System.Collections.Generic;

public class MainClass

{

static int m = 1;

static int n = 1;

static int p = 1000000007;

static int x = 263;

static List<String>[] map;

static int GetHash(string str)

{

long result = 0;

for (int i = str.Length - 1; i >= 0; i--)

{

result = ((int)str[i] + result \* x) % p;

}

result = result % m;

return (int)result;

}

static void Add(string word)

{

if (Find(word) == "no")

{

int index = GetHash(word);

map[index].Insert(0, word);

}

}

static string Find(string word)

{

int index = GetHash(word);

int flag = map[index].IndexOf(word);

if (flag == -1)

{

return "no";

}

else

{

return "yes";

}

}

static void Del(string word)

{

int index = GetHash(word);

map[index].Remove(word);

}

static string Check(string number)

{

int index;

int.TryParse(number, out index);

string result = "";

Array.ForEach(map[index].ToArray(), element => {

result += element + " ";

});

return result;

}

public static void Main()

{

string mInput = Console.ReadLine();

int.TryParse(mInput, out m);

map = new List<String>[m];

for (var i = 0; i < m; i++)

{

map[i] = new List<String>();

}

string nInput = Console.ReadLine();

int.TryParse(nInput, out n);

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

{

string commandInput = Console.ReadLine();

string[] commandSplit = commandInput.Split(' ');

string command = commandSplit[0];

if (command == "add")

{

Add(commandSplit[1]);

}

if (command == "find")

{

string result = Find(commandSplit[1]);

Console.WriteLine(result);

}

if (command == "del")

{

Del(commandSplit[1]);

}

if (command == "check")

{

string result = Check(commandSplit[1]);

Console.WriteLine(result);

}

}

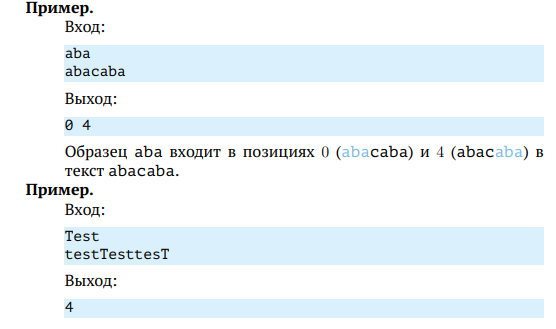
}

}

Поиск образца в тексте

**Формат входа.** Образец Pattern и текст Text.

**Формат выхода.** Индексы вхождений строки Pattern в строку Text в возрастающем порядке, используя индексацию с нуля.



Решение

#include <iostream>

#include <string>

using namespace std;

long M = 1000000007;

long p = 263;

int main()

{

string pattern, text;

cin >> pattern >> text;

long patternHash = 0, curHash = 0;

long power = 1;

for (char c : pattern)

{

patternHash = ((patternHash % M \* p % M) % M + c) % M;

power = (power \* p) % M;

}

for (int i = 0; i < text.length(); ++i)

{

curHash = ((curHash \* p) % M + text[i]);

if (i >= pattern.length())

{

long last = (text[i - pattern.length()] \* power) % M;

curHash = (curHash + M - last) % M;

}

if (curHash == patternHash) { cout << i - pattern.length() + 1 << " "; }

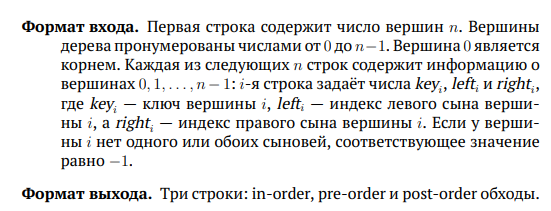
}

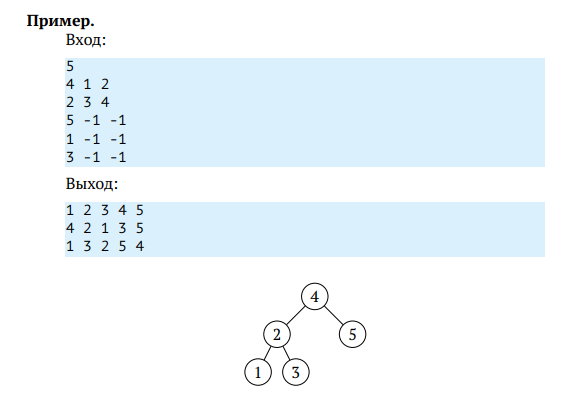
return 0;

}

**Модуль 4**

Обход двоичного дерева





Решение

# include <iostream>

# include <vector>

struct Node

{

int val;

int left;

int right;

};

class Tree

{

public:

Tree(int n)

{

\_tree.resize(n);

int val, left, right;

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

{

std::cin >> val >> left >> right;

\_tree[i].val = val;

\_tree[i].left = left;

\_tree[i].right = right;

}

}

~Tree() { }

void printInOrder(int v)

{

if (v == -1)

return;

printInOrder(\_tree[v].left);

std::cout << \_tree[v].val << " ";

printInOrder(\_tree[v].right);

}

void printPreOrder(int v)

{

if (v == -1)

return;

std::cout << \_tree[v].val << " ";

printPreOrder(\_tree[v].left);

printPreOrder(\_tree[v].right);

}

void printPostOrder(int v)

{

if (v == -1)

return;

printPostOrder(\_tree[v].left);

printPostOrder(\_tree[v].right);

std::cout << \_tree[v].val << " ";

}

private:

std::vector<Node> \_tree;

};

int main(void)

{

int n;

std::cin >> n;

Tree tree(n);

tree.printInOrder(0);

std::cout << std::endl;

tree.printPreOrder(0);

std::cout << std::endl;

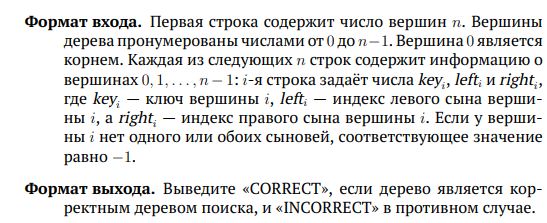
tree.printPostOrder(0);

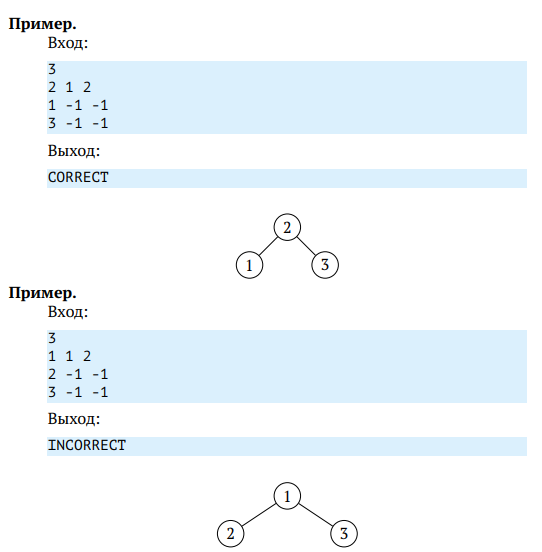
std::cout << std::endl;

return 0;

}

Проверка свойства дерева поиска





Решение

# include <iostream>

# include <vector>

struct Node

{

int val;

int left;

int right;

};

class Tree

{

public:

Tree(int n)

{

\_tree.resize(n);

int val, left, right;

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

{

std::cin >> val >> left >> right;

\_tree[i].val = val;

\_tree[i].left = left;

\_tree[i].right = right;

}

}

~Tree() { }

void InOrder(int v)

{

if (v == -1)

return;

InOrder(\_tree[v].left);

\_output.push\_back(\_tree[v].val);

InOrder(\_tree[v].right);

}

bool check()

{

InOrder(0);

int prev = \_output[0];

for (int i = 1; i < \_output.size(); ++i)

{

int current = \_output[i];

if (!(current > prev))

return false;

prev = current;

}

return true;

}

private:

std::vector<Node> \_tree;

std::vector<int> \_output;

};

int main(void)

{

int n;

std::string res;

std::cin >> n;

if (!n)

{

std::cout << "CORRECT" << std::endl;

return 0;

}

Tree tree(n);

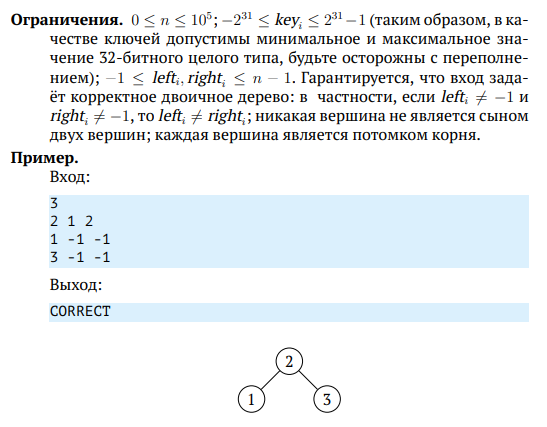
res = tree.check() ? "CORRECT" : "INCORRECT";

std::cout << res << std::endl;

return 0;

}

Проверка более общего свойства дерева поиска



Решение

#include <iostream>

#include <vector>

#include <climits>

struct Vertex

{

long key = -1;

int left = -1;

int right = -1;

};

typedef std::vector<Vertex> TreeV;

bool is\_bst(TreeV& T, int i = 0, long min = LONG\_MIN, long max = LONG\_MAX)

{

if (i == -1) return true;

if (T[i].key >= min && T[i].key < max)

return is\_bst(T, T[i].left, min, T[i].key)

&& is\_bst(T, T[i].right, T[i].key, max);

return false;

}

int main()

{

int n;

std::cin >> n;

TreeV T(n);

while (n)

{

std::cin >> T[T.size() - n].key

>> T[T.size() - n].left

>> T[T.size() - n].right;

n--;

}

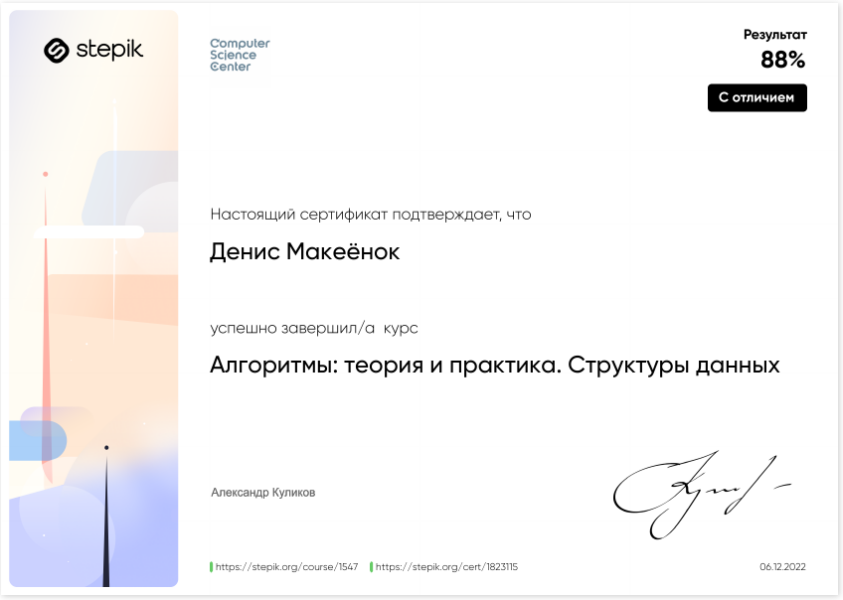
if (!T.size()) std::cout << "CORRECT\n";

else std::cout << (is\_bst(T) ? "CORRECT" : "INCORRECT") << "\n";

return 0;

}

**Сертификат**

****