**Отчет по практике по дисциплине «Структуры и алгоритмы компьютерной обработки данных»**

Подготовила студентка

214 группы

факультета КНиИТ

Акимов Артем

Саратов 2015

**Деревья бинарного поиска**

**I.I.** В файле input.txt хранится последовательность целых чисел. По входной последовательности построить дерево бинарного поиска и найти для него: сумму нечетных значений узлов дерева;

using System;

namespace Tree1\_1

{

public class BinaryTree //класс, реализующий АТД «дерево бинарного поиска»

{

//вложенный класс, отвечающий за узлы и операции допустимы для дерева бинарного

//поиска

private class Node

{

public object inf; //информационное поле

public Node left; //ссылка на левое поддерево

public Node rigth; //ссылка на правое поддерево

//конструктор вложенного класса, создает узел дерева

public Node(object nodeInf)

{

inf = nodeInf;

left = null;

rigth = null;

}

//добавляет узел в дерево так, чтобы дерево оставалось деревом бинарного поиска

public static void Add(ref Node r, object nodeInf)

{

if (r == null)

{

r = new Node(nodeInf);

}

else

{

if (((IComparable)(r.inf)).CompareTo(nodeInf) > 0)

{

Add(ref r.left, nodeInf);

}

else

{

Add(ref r.rigth, nodeInf);

}

}

}

public static void Preorder(Node r) //прямой обход дерева

{

if (r != null)

{

Console.Write("{0} ", r.inf);

Preorder(r.left);

Preorder(r.rigth);

}

}

/////////////////////////////////////

public static void Sum(Node r, ref int s) //reshenie zadachi

{

if (r != null)

{

if ((int)(r.inf) % 2 == 1)

s += (int)r.inf;

Sum(r.left, ref s);

Sum(r.rigth, ref s);

}

}

/////////////////////////////////////

public static void Inorder(Node r) //симметричный обход дерева

{

if (r != null)

{

Inorder(r.left);

Console.Write("{0} ", r.inf);

Inorder(r.rigth);

}

}

public static void Postorder(Node r) //обратный обход дерева

{

if (r != null)

{

Postorder(r.left);

Postorder(r.rigth);

Console.Write("{0} ", r.inf);

}

}

//поиск ключевого узла в дереве

public static void Search(Node r, object key, out Node item)

{

if (r == null)

{

item = null;

}

else

{

if (((IComparable)(r.inf)).CompareTo(key) == 0)

{

item = r;

}

else

{

if (((IComparable)(r.inf)).CompareTo(key) > 0)

{

Search(r.left, key, out item);

}

else

{

Search(r.rigth, key, out item);

}

}

}

}

//методы Del и Delete позволяют удалить узел в дереве так, чтобы дерево при этом

//оставалось деревом бинарного поиска

private static void Del(Node t, ref Node tr)

{

if (tr.rigth != null)

{

Del(t, ref tr.rigth);

}

else

{

t.inf = tr.inf;

tr = tr.left;

}

}

public static void Delete(ref Node t, object key)

{

if (t == null)

{

throw new Exception("Данное значение в дереве отсутствует");

}

else

{

if (((IComparable)(t.inf)).CompareTo(key) > 0)

{

Delete(ref t.left, key);

}

else

{

if (((IComparable)(t.inf)).CompareTo(key) < 0)

{

Delete(ref t.rigth, key);

}

else

{

if (t.left == null)

{

t = t.rigth;

}

else

{

if (t.rigth == null)

{

t = t.left;

}

else

{

Node tr = t.left;

Del(t, ref tr);

}

}

}

}

}

}

} //конец вложенного класса

Node tree; //ссылка на корень дерева

//свойство позволяет получить доступ к значению информационного поля корня дерева

public object Inf

{

set { tree.inf = value; }

get { return tree.inf; }

}

public BinaryTree() //открытый конструктор

{

tree = null;

}

private BinaryTree(Node r) //закрытый конструктор

{

tree = r;

}

public void Add(object nodeInf) //добавление узла в дерево

{

Node.Add(ref tree, nodeInf);

}

//организация различных способов обхода дерева

public void Preorder()

{

Node.Preorder(tree);

}

public void Sum(ref int s)//reshenie

{

Node.Sum(tree, ref s);

}

public void Inorder()

{

Node.Inorder(tree);

}

public void Postorder()

{

Node.Postorder(tree);

}

//поиск ключевого узла в дереве

public BinaryTree Search(object key)

{

Node r;

Node.Search(tree, key, out r);

BinaryTree t = new BinaryTree(r);

return t;

}

//удаление ключевого узла в дереве

public void Delete(object key)

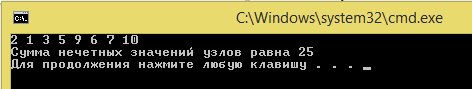
{

Node.Delete(ref tree, key);

}

}

}

****

**II.I.** В файле input.txt хранится последовательность целых чисел. По входной последовательности построить дерево бинарного поиска и: распечатать узлы k-го уровня дерева;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace ConsoleApplication1

{

public class BinaryTree

{

private class Node

{

public object inf;

public Node left;

public Node rigth;

public Node(object nodeInf)

{

inf = nodeInf;

left = null;

rigth = null;

}

public static void RotationRigth(ref Node t)

{

Node x = t.left;

t.left = x.rigth;

x.rigth = t;

t = x;

}

public static void RotationLeft(ref Node t)

{

Node x = t.rigth;

t.rigth = x.left;

x.left = t;

t = x;

}

public static void InsertToRoot(ref Node t, object item)

{

if (t == null)

{

t = new Node(item);

}

else if (((IComparable)(t.inf)).CompareTo(item) > 0)

{

InsertToRoot(ref t.left, item);

RotationRigth(ref t);

}

else

{

InsertToRoot(ref t.rigth, item);

RotationLeft(ref t);

}

}

public static void Add(ref Node r, object nodeInf)

{

if (r == null)

{

r = new Node(nodeInf);

}

else

{

if (((IComparable)(r.inf)).CompareTo(nodeInf) > 0)

{

Add(ref r.left, nodeInf);

}

else

{

Add(ref r.rigth, nodeInf);

}

}

}

public static void Preorder(Node r) //прямой обход дерева

{

if (r != null)

{

Console.Write("{0} ", r.inf);

Preorder(r.left);

Preorder(r.rigth);

}

}

public static void Search(Node r, object key, out Node item)

{

if (r == null)

{

item = null;

}

else

{

if (((IComparable)(r.inf)).CompareTo(key) == 0)

{

item = r;

}

else

{

if (((IComparable)(r.inf)).CompareTo(key) > 0)

{

Search(r.left, key, out item);

}

else

{

Search(r.rigth, key, out item);

}

}

}

}

public static void GetItemsOnALevel(Node t, int deep, ref int n, ref int counter)

{

if (t != null)

{

if (deep == n)

{

Console.Write(t.inf+" ");

counter++;

}

n++;

GetItemsOnALevel(t.rigth, deep, ref n, ref counter);

GetItemsOnALevel(t.left, deep, ref n, ref counter);

n--;

}

}

}

Node tree;

public object Inf

{

set { tree.inf = value; }

get { return tree.inf; }

}

public BinaryTree()

{

tree = null;

}

private BinaryTree(Node r)

{

tree = r;

}

public void Add(object nodeInf)

{

Node.Add(ref tree, nodeInf);

}

public void Preorder()

{

Node.Preorder(tree);

}

public BinaryTree Search(object key)

{

Node r;

Node.Search(tree, key, out r);

BinaryTree t = new BinaryTree(r);

return t;

}

public void InsertToRoot(object item)

{

Node.InsertToRoot(ref tree, item);

}

public int ItemsOnAlevel(int deep)

{

int number = 0;

int n = 0;

Node.GetItemsOnALevel(tree, deep, ref n, ref number);

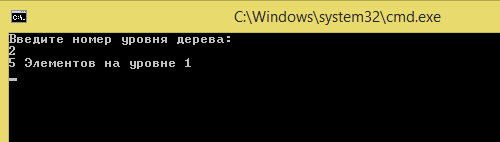
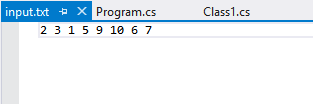
Console.Write("Элементов на уровне ");

return number;

}

}

}



**Графы**

**I.I.** В входном файле указывается количество вершин графа/орграфа и матрица смежности: Для заданного графа: подсчитать количество вершин, смежных с данной;

using System;

using System.IO;

namespace Pract16.II

{

public class Graph

{

private class Node //вложенный класс для скрытия данных и алгоритмов

{

private int[,] array; //матрица смежности

public int this[int i, int j] //индексатор для обращения к матрице смежности

{

get

{

return array[i, j];

}

set

{

array[i, j] = value;

}

}

public int Size //свойство для получения размерности матрицы смежности

{

get

{

return array.GetLength(0);

}

}

private bool[] nov; //вспомогательный массив: если i-ый элемент массива равен

//true, то i-ая вершина еще не просмотрена; если i-ый

//элемент равен false, то i-ая вершина просмотрена

public void NovSet() //метод помечает все вершины графа как непросмотреные

{

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

{

nov[i] = true;

}

}

//конструктор вложенного класса, инициализирует матрицу смежности и

// вспомогательный массив

public Node(int[,] a)

{

array = a;

nov = new bool[a.GetLength(0)];

}

public void AddTop(int N)

{

int[,] c = new int[Size+1, Size+1];

}

} //конец вложенного клаcса

private Node graph; //закрытое поле, реализующее АТД «граф»

public Graph(string name) //конструктор внешнего класса

{

using (StreamReader file = new StreamReader(name))

{

int n = int.Parse(file.ReadLine());

int[,] a = new int[n, n];

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

{

string line = file.ReadLine();

string[] mas = line.Split(' ');

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

{

a[i, j] = int.Parse(mas[j]);

}

}

graph = new Node(a);

}

}

//метод выводит матрицу смежности на консольное окно

public void Show()

{

for (int i = 0; i < graph.Size; i++)

{

int count = 0;

for (int j = 0; j < graph.Size; j++)

{

if (graph[i, j] != 0)

count++;

}

Console.Write("Количество смежных вершин для вершины {1} {0}", count,i);

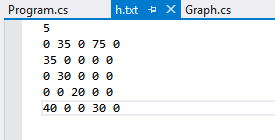
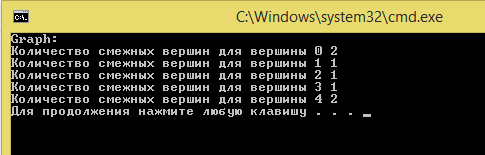
Console.WriteLine();

}

}

}

}

**II.I**. Найти кратчайший путь, соединяющие города А и В, проходящий только через заданное множество городов.

using System;

using System.IO;

using System.Collections;

using System.Text;

namespace Example

{

class Program

{

static void Main()

{

Graph g = new Graph("C:/Users/1/Documents/Visual Studio 2013/input.txt");

Console.WriteLine("Graph:");

g.Show();

Console.WriteLine();

Console.WriteLine("Floyd:");

g.Floyd();

}

}

public class Stack

{

private class Node //вложенный класс, реализующий элемент стека

{

private object inf;

private Node next;

public Node(object nodeInfo)

{

inf = nodeInfo;

next = null;

}

public Node Next

{

get { return next; }

set { next = value; }

}

public object Inf

{

get { return inf; }

set { inf = value; }

}

} //конец класса Node

private Node head; //ссылка на вершину стека

public Stack() //конструктор класса, создает пустой стек

{

head = null;

}

public void Push(object nodeInfo) // добавляет элемент в вершину стека

{

Node r = new Node(nodeInfo);

r.Next = head;

head = r;

}

public object Pop() //извлекает элемент из вершины стека, если он не пуст

{

if (head == null)

{

throw new Exception("Стек пуст");

}

else

{

Node r = head;

head = r.Next;

return r.Inf;

}

}

public bool IsEmpty //определяет пуст или нет стек

{

get

{

if (head == null)

{

return true;

}

else

{

return false;

}

}

}

}

public class Queue

{

private class Node //вложенный класс, реализующий базовый элемент очереди

{

private object inf;

private Node next;

public Node(object nodeInfo)

{

inf = nodeInfo;

next = null;

}

public Node Next

{

get { return next; }

set { next = value; }

}

public object Inf

{

get { return inf; }

set { inf = value; }

}

} //конец класса Node

private Node head;

private Node tail;

public Queue()

{

head = null;

tail = null;

}

public void Add(object nodeInfo)

{

Node r = new Node(nodeInfo);

if (head == null)

{

head = r;

tail = r;

}

else

{

tail.Next = r;

tail = r;

}

}

public object Take()

{

if (head == null)

{

throw new Exception("Очередь пуста.");

}

else

{

Node r = head;

head = head.Next; if (head == null)

{

tail = null;

}

return r.Inf;

}

}

public bool IsEmpty

{

get

{

if (head == null)

{

return true;

}

else

{

return false;

}

}

}

}

public class Graph

{

private class Node //вложенный класс для скрытия данных и алгоритмов

{

private int[,] array; //матрица смежности

private static string[] s;

public string this[int i]

{

get

{

return s[i];

}

}

public int this[int i, int j] //индексатор для обращения к матрице смежности

{

get

{

return array[i, j];

}

set

{

array[i, j] = value;

}

}

public int Size //свойство для получения размерности матрицы смежности

{

get

{

return array.GetLength(0);

}

}

private bool[] nov; //вспомогательный массив: если i-ый элемент массива равен

//true, то i-ая вершина еще не просмотрена; если i-ый

//элемент равен false, то i-ая вершина просмотрена

public void NovSet() //метод помечает все вершины графа как непросмотреные

{

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

{

nov[i] = true;

}

}

//конструктор вложенного класса, инициализирует матрицу смежности и

// вспомогательный массив

public Node(int[,] a, string[] p)

{

s = p;

array = a;

nov = new bool[a.GetLength(0)];

}

//реализация алгоритма Флойда

public long[,] Floyd(out int[,] p)

{

int i, j, k;

//создаем массивы р и а

long[,] a = new long[Size, Size];

p = new int[Size, Size];

for (i = 0; i < Size; i++)

{

for (j = 0; j < Size; j++)

{

if (i == j)

{

a[i, j] = 0;

}

else

{

if (array[i, j] == 0)

{

a[i, j] = int.MaxValue;

}

else

{

a[i, j] = array[i, j];

}

}

p[i, j] = -1;

}

}

//осуществляем поиск кратчайших путей

for (k = 0; k < Size; k++)

{

for (i = 0; i < Size; i++)

{

for (j = 0; j < Size; j++)

{

long distance = a[i, k] + a[k, j];

if (a[i, j] > distance)

{

a[i, j] = distance;

p[i, j] = k;

}

}

}

}

return a;//в качестве результата возвращаем массив кратчайших путей между

} //всеми парами вершин

public bool WayFloyd(int a, int b, int[,] p, ref Queue items)

{

int k = p[a, b];

if (k != -1)

{

// рекурсивно восстанавливаем путь между вершинами а и k

WayFloyd(a, k, p, ref items);

items.Add(k); //помещаем вершину к в очередь

// рекурсивно восстанавливаем путь между вершинами k и b

WayFloyd(k, b, p, ref items);

return false;

}

else return true;

}

public string name(int v)

{

return s[v];

}

}

private Node graph; //закрытое поле, реализующее АТД «граф»

public Graph(string name) //конструктор внешнего класса

{

using (StreamReader file = new StreamReader(name, Encoding.GetEncoding(1251)))

{

int n = int.Parse(file.ReadLine());

string[] p = new string[n];

string line = file.ReadLine();

p = line.Split(' ');

int[,] a = new int[n, n];

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

{

line = file.ReadLine();

string[] mas = line.Split(' ');

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

{

a[i, j] = int.Parse(mas[j]);

}

}

graph = new Node(a, p);

}

}

//метод выводит матрицу смежности на консольное окно

public void Show()

{

for (int i = 0; i < graph.Size; i++)

{

for (int j = 0; j < graph.Size; j++)

{

Console.Write("{0,4}", graph[i, j]);

}

Console.WriteLine();

}

}

public void Floyd()

{

int[,] p;

long[,] a = graph.Floyd(out p); //запускаем алгоритм Флойда

int i, j;

//анализируем полученные данные и выводим их на экран

for (i = 0; i < graph.Size; i++)

{

for (j = 0; j < graph.Size; j++)

{

if (i != j)

{

if (a[i, j] == int.MaxValue)

{

Console.WriteLine("Пути из вершины {0} в вершину {1} не существует", graph.name(i), graph.name(j));

}

else

{

Console.Write("Кратчайший путь от вершины {0} до вершины {1} равен {2}, ", graph.name(i), graph.name(j), a[i, j]);

Console.Write(" путь ");

Queue items = new Queue();

items.Add(i);

graph.WayFloyd(i, j, p, ref items);

items.Add(j);

while (!items.IsEmpty)

{

Console.Write("{0} ", items.Take());

}

Console.WriteLine();

}

}

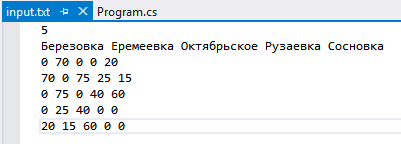
}

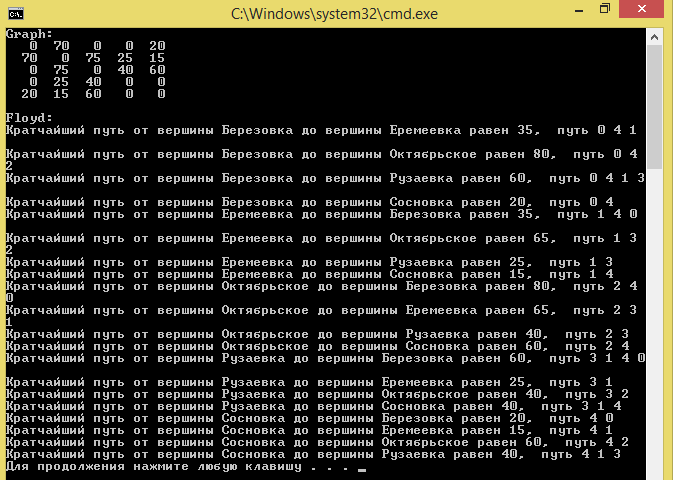
}

}

}

}





**Код Хаффмана**

**Класс HuffmanTest**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace HuffmanTest

{

public class HuffmanTree

{

private List<Node> nodes = new List<Node>();

public Node Root { get; set; }

public Dictionary<char, int> Frequencies = new Dictionary<char, int>();

public void Build(string source)

{

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

{

if (!Frequencies.ContainsKey(source[i]))

{

Frequencies.Add(source[i], 0);

}

Frequencies[source[i]]++;

}

foreach (KeyValuePair<char, int> symbol in Frequencies)

{

nodes.Add(new Node() { Symbol = symbol.Key, Frequency = symbol.Value });

}

while (nodes.Count > 1)

{

List<Node> orderedNodes = nodes.OrderBy(node => node.Frequency).ToList<Node>();

if (orderedNodes.Count >= 2)

{

// берем первые два узла

List<Node> taken = orderedNodes.Take(2).ToList<Node>();

// создаем узел-родитель, складываем частоты

Node parent = new Node()

{

Symbol = '\*',

Frequency = taken[0].Frequency + taken[1].Frequency,

Left = taken[0],

Right = taken[1]

};

nodes.Remove(taken[0]);

nodes.Remove(taken[1]);

nodes.Add(parent);

}

this.Root = nodes.FirstOrDefault();

}

}

public BitArray Encode(string source)

{

List<bool> encodedSource = new List<bool>();

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

{

List<bool> encodedSymbol = this.Root.Traverse(source[i], new List<bool>());

encodedSource.AddRange(encodedSymbol);

}

BitArray bits = new BitArray(encodedSource.ToArray());

return bits;

}

public string Decode(BitArray bits)

{

Node current = this.Root;

string decoded = "";

foreach (bool bit in bits)

{

if (bit)

{

if (current.Right != null)

{

current = current.Right;

}

}

else

{

if (current.Left != null)

{

current = current.Left;

}

}

if (IsLeaf(current))

{

decoded += current.Symbol;

current = this.Root;

}

}

return decoded;

}

public bool IsLeaf(Node node)

{

return (node.Left == null && node.Right == null);

}

}

}

**Класс Node**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace HuffmanTest

{

public class Node

{

public char Symbol { get; set; }

public int Frequency { get; set; }

public Node Right { get; set; }

public Node Left { get; set; }

public List<bool> Traverse(char symbol, List<bool> data)

{

//вышли на лист

if (Right == null && Left == null)

{

if (symbol.Equals(this.Symbol))

{

return data;

}

else

{

return null;

}

}

else

{

List<bool> left = null;

List<bool> right = null;

if (Left != null)

{

List<bool> leftPath = new List<bool>();

leftPath.AddRange(data);

leftPath.Add(false);

left = Left.Traverse(symbol, leftPath);

}

if (Right != null)

{

List<bool> rightPath = new List<bool>();

rightPath.AddRange(data);

rightPath.Add(true);

right = Right.Traverse(symbol, rightPath);

}

if (left != null)

{

return left;

}

else

{

return right;

}

}

}

}

}

**Класс Program**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace HuffmanTest

{

class Program

{

static void Main(string[] args)

{

string input = "to be or not to be";

HuffmanTree huffmanTree = new HuffmanTree();

// Build the Huffman tree

huffmanTree.Build(input);

// Encode

BitArray encoded = huffmanTree.Encode(input);

Console.Write("Encoded: ");

foreach (bool bit in encoded)

{

Console.Write((bit ? 1 : 0) + "");

}

Console.WriteLine();

// Decode

string decoded = huffmanTree.Decode(encoded);

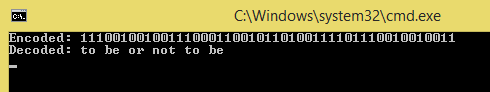
Console.WriteLine("Decoded: " + decoded);

Console.ReadLine();

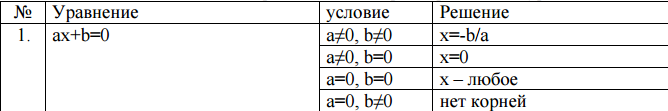
}

}

}

C:\Users\1\Dropbox\Скриншоты\Скриншот 2015-05-26 23.16.06.png

**Вычислительная математика**

****

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace \_19.II

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

double x;

double a = double.Parse(textBox1.Text);

double b = double.Parse(textBox2.Text);

if (a!=0)

{

x = -b / a;

label3.Text = String.Format("Корень = {0}", x);

}

else if (b==0)

label3.Text = String.Format("Корень - любое число");

else

label3.Text = String.Format("Корней нет");

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

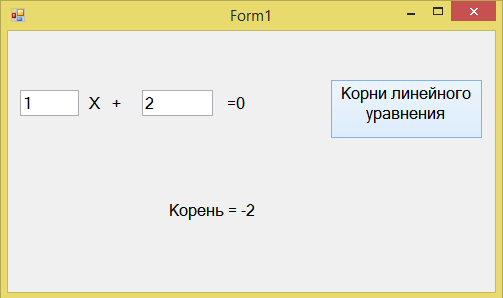
private void label2\_Click(object sender, EventArgs e)

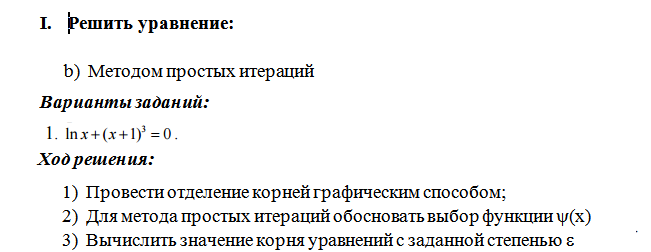
{

}

}

}

****

****

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace \_20.II

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

static double psi(double xPrev)

{

double x = 1 / Math.Pow(xPrev+1, 0.5);

return x;

}

static double Solution(double x)

{

double xPrev = 0;

do

{

xPrev = x;

x = psi(xPrev);

}

while (Math.Abs(x - xPrev) > 0.000001);

return x;

}

private void button1\_Click(object sender, EventArgs e)

{

double a = 0.1;

double b = 1;

double x = (a + b) / 2;

label1.Text = String.Format("Корень = {0}", Solution(x));

}

private void label1\_Click(object sender, EventArgs e)

{

}

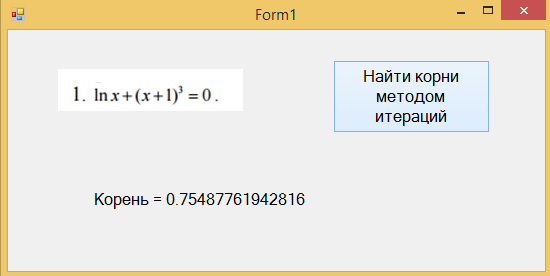
private void pictureBox1\_Click(object sender, EventArgs e)

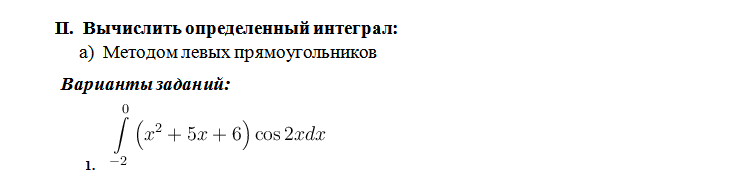
{

}

}

}

****

****

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace \_21.II

{

public partial class Form1 : Form

{

public static double Function(double x)

{

return (x \* x + 5 \* x + 6) \* Math.Cos(2 \* x);

}

public static double LeftRectangle(double a, double b, int n)

{

double res = 0;

double h = (b - a) / (double)n;

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

res += Function(a + i \* h);

res \*= h;

return res;

}

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

int n = 1000;

int n2 = 100;

double ideal = 1.60254;

double a = -2, b = 0;

double res = LeftRectangle(a, b, n);

double res2 = LeftRectangle(a, b, n2);

label1.Text = String.Format("{0} при n=1000, {1} при n=100", res,res2);

double Abs = Math.Abs(ideal - res);

double Abs2 = Math.Abs(ideal - res2);

double Otnos=Abs/res\*100;

double Otnos2=Abs/res2\*100;

label2.Text = String.Format("АП при n= 1000 {0} ОП {1},", Abs,Otnos);

}

private void pictureBox1\_Click(object sender, EventArgs e)

{

}

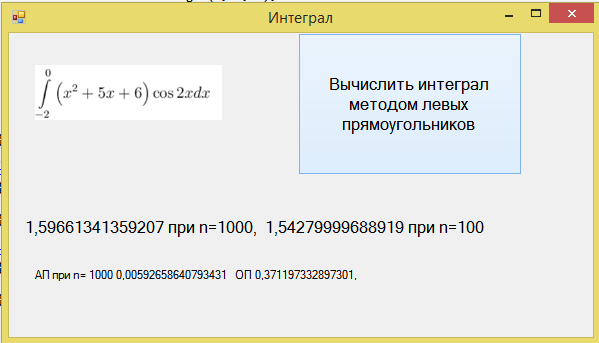
private void label2\_Click(object sender, EventArgs e)

{

}

}

}

****