1. Сортировка пузырьком

static void Sorting(int[] arr)

{

int i = 0;

while (true)

{

if (i == arr.Length - 1)

break;

if (arr[i] > arr[i + 1])

{

Swap(arr, i,i+1);

i = 0;

continue;

}

i++;

}

}

1. Сортировка вставками

static void Sort(int[] arr)

{

for (int i = 1; i < arr.Length; i++)

{

int temp = arr[i];

int j = i;

while (j > 0 && temp < arr[j - 1])

{

arr[j] = arr[j - 1];

j--;

}

arr[j] = temp;

}

}

1. Сортировка выбором

static void Sorting(int[] arr)

{

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

{

int minIndex = FindMin(arr,i);

Swap(arr, i, minIndex);

}

}

static void Swap(int[] arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

static int FindMin(int[] arr,int i)

{

int min = arr[i];

int minIndex = i;

for (int j = i+1; j < arr.Length; j++)

{

if(min > arr[j])

{

min = arr[j];

minIndex = j;

}

}

return minIndex;

}

1. Шейкерная сортировка

static void Sorting(int[] arr)

{

int from = 0;

int before = arr.Length;

while (true)

{

SortRight(arr, from, before);

before--;

SortLeft(arr, from, before);

from++;

if (from == before)

break;

}

}

static void SortRight(int[] arr, int from, int before)

{

for (int i = from; i < before-1; i++)

{

if ( arr[i] > arr[i + 1])

{

Swap(arr, i, i + 1);

}

}

}

static void SortLeft(int[] arr, int from, int before)

{

for (int i = before; i > from; i--)

{

if (arr[i] < arr[i - 1])

{

Swap(arr, i, i - 1);

}

}

}

1. Сортировка Шелла

static void Sorting(int[] arr)

{

int j;

int step = arr.Length / 2;

while (step > 0)

{

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

{

j = i;

while ((j >= 0) && (arr[j] > arr[j + step]))

{

Swap(arr, j, j + step);

j -= step;

}

}

step = step / 2;

}

}

1. Алгоритм бинарного поиска

static int FindNum(int[] arr,int num)

{

int from = 0;

int before = arr.Length;

while (true)

{

int aver = (before + from) / 2;

if (num == arr[aver])

{

return aver;

}

else if (num > arr[aver])

{

from = aver;

}

else

{

before = aver;

}

}

}

1. Быстрая сортировка

static int Partition(int[] arr, int start, int end)

{

int marker = start;

for (int i = start; i <= end; i++)

{

if (arr[i] <= arr[end])

{

Swap(arr, i, marker);

marker += 1;

}

}

return marker - 1;

}

static void Sorting(int[] arr, int start, int end)

{

if (start >= end)

{

return;

}

int pivot = Partition(arr, start, end);

Sorting(arr, start, pivot - 1);

Sorting(arr, pivot + 1, end);

}

1. Внешняя сортировка слиянием

public static void SortByFiles(string pathToSort)

{

int numOfLines = 0;

using (var reader = new StreamReader(pathToSort))

while (reader.ReadLine() != null)

numOfLines++;

int runTo = (int)Math.Ceiling(Math.Log(numOfLines, 2));

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

TakeElements((int)Math.Pow(2, i));

void TakeElements(int switching)

{

//Наполнение временных файлов

var writerA = new StreamWriter("A.txt");

var writerB = new StreamWriter("B.txt");

int counter = 0;

bool writeToA = true;

using (var reader = new StreamReader(pathToSort))

{

string? line = reader.ReadLine();

while (line != null)

{

if (counter % switching == 0)

writeToA = !writeToA;

if (writeToA)

writerA.WriteLine(line);

else

writerB.WriteLine(line);

counter++;

line = reader.ReadLine();

}

}

writerB.Dispose();

writerA.Dispose();

//Сбрасывание в файл

File.Create(pathToSort).Dispose();

using (var writer = new StreamWriter(pathToSort))

{

var readerA = new StreamReader("A.txt");

var readerB = new StreamReader("B.txt");

while (!readerA.EndOfStream || !readerB.EndOfStream)

{

PopTo(readerA, readerB, switching);

}

readerA.Dispose();

readerB.Dispose();

void PopTo(StreamReader readerA, StreamReader readerB, int popFromFile)

{

int stepsA = 0, stepsB = 0;

string? a\_line = readerA.ReadLine();

string? b\_line = readerB.ReadLine();

while ((stepsA < popFromFile || stepsB < popFromFile) && (a\_line != null || b\_line != null))

{

int compare;

if (a\_line == null) compare = 1;

else if (b\_line == null) compare = -1;

else compare = int.Parse(a\_line).CompareTo(int.Parse(b\_line));

if (compare < 0 || compare == 0)

{

writer.WriteLine(a\_line);

stepsA++;

a\_line = stepsA >= popFromFile ? null : readerA.ReadLine();

}

if (compare > 0 || compare == 0)

{

writer.WriteLine(b\_line);

stepsB++;

b\_line = stepsB >= popFromFile ? null : readerB.ReadLine();

}

}

}

}

}

}

1. Сортировка с помощью двоичного дерева

static void Sorting(int[] arr, BinaryTree binaryTree)

{

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

{

Node node = binaryTree.root;

while (true)

{

if (arr[i] < node.Value)

{

if (node.LeftNode == null)

{

Node newNode = new Node(arr[i]);

node.LeftNode = newNode;

break;

}

else

{

node = node.LeftNode;

}

}

else

{

if (node.RightNode == null)

{

Node newNode = new Node(arr[i]);

node.RightNode = newNode;

break;

}

else

{

node = node.RightNode;

}

}

}

}

}

1. Поразрядная сортировка (radix sort)

public static void Sorting(int[] arr)

{

int i, j;

int[] tmp = new int[arr.Length];

for (int shift = sizeof(int) \* 8 - 1; shift >= 0; shift--)

{

j = 0;

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

{

bool move = (arr[i] << shift) >= 0;

if (shift == 0 ? !move : move)

arr[i - j] = arr[i];

else

tmp[j++] = arr[i];

}

Array.Copy(tmp, 0, arr, arr.Length - j, j);

}

}

1. Хэш-таблицы с разрешением коллизий методом цепочек

class MyDictionary

{

List<Cell>[] table;

int size = 0;

public MyDictionary(int size)

{

table = new List<Cell>[size];

this.size = size;

}

public void Add(int value)

{

int key = FindHash(value);

if (table[key] == null)

table[key] = new List<Cell>();

table[key].Add(new Cell(value));

}

int FindHash(int key)

{

double value = 0.59756416 \* key;

var trunc = Math.Truncate(value);

return (int)(size \* (value - trunc));

}

}

class Cell

{

int value;

public Cell(int value)

{

this.value = value;

}

}

1. Хэш-таблицы с разрешением коллизий методом открытой адресации

class MyDictionary

{

Cell[] table;

int size = 0;

public MyDictionary(int size)

{

table = new Cell[size];

this.size = size;

}

public void Add(int value)

{

int j = 1;

int key = FindHash(value);

while (true)

{

while (key > size)

{

key -= size;

}

try

{

if (table[key] != null)

throw new Exception("Cell not null");

table[key] = new Cell(value);

break;

}

catch

{

key += (int)Math.Pow(j, 2);

j++;

}

}

}

int FindHash(int key)

{

double value = 0.59756416 \* key;

var trunc = Math.Truncate(value);

return (int)(size \* (value - trunc));

}

}

class Cell

{

int value;

public Cell(int value)

{

this.value = value;

}

}

1. ABC-сортировка (для строк)

public static List<string> ABSSort(List<string> words, int rank = 0)

{

if (words.Count <= 1)

return words;

var square = new Dictionary<char, List<string>>(62);

var result = new List<string>();

int shortWordsCounter = 0;

foreach (var word in words)

{

if (rank < word.Length)

{

if (square.ContainsKey(word[rank]))

square[word[rank]].Add(word);

else

square.Add(word[rank], new List<string> { word });

}

else

{

result.Add(word);

shortWordsCounter++;

}

}

if (shortWordsCounter == words.Count)

return words;

for (char i = 'А'; i <= 'я'; i++)

{

if (square.ContainsKey(i))

{

foreach (var word in ABSSort(square[i], rank + 1))

result.Add(word);

}

}

return result;

}

}

1. Реализовать стек и базовые операции работы со стеком, с использованием собственного двусвязного списка

class MyStack<T>

{

MyDoubleLinkedList<T> list = new MyDoubleLinkedList<T>();

public void Push(T value) => list.Add(value);

public T Pop()

{

T value = list.ReturnLast();

list.RemoveLast();

return value;

}

public T Top() => list.ReturnLast();

public bool isEmpty() => list.Count == 0 ? true : false;

public void Print() => list.Print();

}

class MyDoubleLinkedList<T>

{

Node<T> firtsNode;

Node<T> lastNode;

int count;

public int Count

{

get

{

return count;

}

set

{

count = value;

}

}

public T ReturnLast() => lastNode.Value;

public T ReturnFirst() => firtsNode.Value;

public void Add(T value)

{

Node<T> node = new Node<T>(value);

if (firtsNode == null && lastNode == null)

{

firtsNode = node;

}

else

{

node.nodeBack = lastNode;

lastNode.nodeNext = node;

}

lastNode = node;

count++;

}

public void Remove(T value)

{

Node<T> node = FindNode(value);

Node<T> temp = node.nodeBack;

temp.nodeNext = node.nodeNext;

temp = node.nodeNext;

temp.nodeBack = node.nodeBack;

count--;

}

public void RemoveLast()

{

Node<T> node = lastNode.nodeBack;

node.nodeNext = null;

lastNode = node;

count--;

}

public void RemoveFirst()

{

Node<T> node = firtsNode.nodeNext;

node.nodeBack = null;

firtsNode = node;

count--;

}

Node<T> FindNode(T value)

{

Node<T> node = firtsNode;

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

{

if (node.Value.Equals(value))

{

return node;

}

node = node.nodeNext;

}

return null;

}

public void Print()

{

Node<T> node = firtsNode;

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

{

Console.WriteLine(node.Value);

node = node.nodeNext;

}

}

}

class Node<T>

{

T value;

public T Value {

get

{

return value;

}

set

{

this.value = value;

}

}

public Node<T> nodeBack;

public Node<T> nodeNext;

public Node(T value)

{

this.value = value;

}

}

1. Реализовать очередь и базовые операции работы с очередью, с использованием собственного двусвязного списка

class MyQueue<T>

{

MyDoubleLinkedList<T> list = new MyDoubleLinkedList<T>();

public void Enqueue(T value) => list.Add(value);

public T Dequeue()

{

T value = list.ReturnFirst();

list.RemoveFirst();

return value;

}

public T Top() => list.ReturnLast();

public bool isEmpty() => list.Count == 0 ? true : false;

public void Print() => list.Print();

}

1. Алгоритм Крускала (поиска минимального остовного дерева)

public class Edge

{

public int node1, node2, weight;

public Edge(int node1, int node2, int weight)

{

this.node1 = node1;

this.node2 = node2;

this.weight = weight;

}

public override string ToString() => $"{node1} {node2}, {weight}";

}

public static class TreeGraphMethods

{

// Алгоритм Краскала. Обходим все отсортированные по весу ребра, которые не образуют цикл, ребра записываем в образуемые множества.

public static List<Edge> Kruskals(ICollection<Edge> edges)

{

var nodes = new List<int>(); // ищем все вершины

foreach (var edge in edges)

{

if (!nodes.Contains(edge.node1)) nodes.Add(edge.node1);

if (!nodes.Contains(edge.node2)) nodes.Add(edge.node2);

}

var sortedByWeight = edges.OrderBy(e => e.weight);

var nodeSets = new Dictionary<int, int>(); // множества которые хранят номера вершин

var resultEdges = new List<Edge>();

foreach (var edge in sortedByWeight)

{

int node1 = edge.node1, node2 = edge.node2;

if (nodeSets.ContainsKey(node1))

{

if (nodeSets.ContainsKey(node2))

{

if (nodeSets[node1] == nodeSets[node2]) // две вершины из одного множества - будет цикл

continue;

else

{ // соединяем два множества в одно

var nodesOfSecondSet = nodeSets.Where(x => x.Value == nodeSets[node2]).Select(x => x.Key);

foreach (var node in nodesOfSecondSet)

nodeSets[node] = nodeSets[node1];

}

}

else

{

nodeSets.Add(node2, nodeSets[node1]);

}

}

else

{

if (nodeSets.ContainsKey(node2))

nodeSets.Add(node1, nodeSets[node2]);

else

{

int newSet = 0; // ни в одном множестве нет этих вершин - создаем новое множество

while (nodeSets.ContainsValue(newSet)) // ищем уникальный номер для нового множества

newSet++;

nodeSets.Add(node1, newSet);

nodeSets.Add(node2, newSet);

}

}

resultEdges.Add(edge); // все проверки пройдены и ребро может быть добавлено в ответ

var sets = nodeSets.GroupBy(x => x.Value);

foreach (var set in sets)

if (set.Count() == nodes.Count)

return resultEdges;

}

throw new Exception();

}

}

1. Алгоритм Прима (поиска минимального остовного дерева)

public static class TreeGraphMethods

{

// Алгоритм Эль Примо. Выбираем случайно начальную вершину, затем идем по наименьшему по весу ребру, которое не образует цикл, до того как не обойдем все ноды

public static List<Edge> Prima(Edge[] edges)

{

var nodes = new List<int>(); // ищем все вершины

foreach (var edge in edges)

{

if (!nodes.Contains(edge.node1)) nodes.Add(edge.node1);

if (!nodes.Contains(edge.node2)) nodes.Add(edge.node2);

}

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

visitedNodes.Add(edges[0].node1);

IOrderedEnumerable<Edge> toVisit = edges.Where(x => x.node1 == visitedNodes[0] || x.node2 == visitedNodes[0]).OrderBy(x => x.weight);

List<Edge> resultEdges = new List<Edge>();

while (visitedNodes.Count < nodes.Count)

{

if (toVisit.FirstOrDefault() == null) throw new Exception();

Edge minEdge = toVisit.First();

resultEdges.Add(minEdge);

if (visitedNodes.Contains(minEdge.node1))

visitedNodes.Add(minEdge.node2);

else

visitedNodes.Add(minEdge.node1);

toVisit = edges.Where(x => visitedNodes.Contains(x.node1) ^ visitedNodes.Contains(x.node2)).OrderBy(x => x.weight);

}

return resultEdges;

}

}

public class Edge

{

public int node1, node2, weight;

public Edge(int node1, int node2, int weight)

{

this.node1 = node1;

this.node2 = node2;

this.weight = weight;

}

public override string ToString() => $"{node1} {node2}, {weight}";

}

1. Обход графа в глубину

class Graph

{

public Node[] graph;

public Graph(int countNode)

{

graph = new Node[countNode];

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

graph[i] = new Node();

}

public void addEdge(int id1, int id2)

{

graph[id1].conected.Add(graph[id2]);

}

public Node[] DFS(int start)

{

var startNode = graph[start];

var visited = new List<Node> { startNode };

var stack = new Stack<Node>();

stack.Push(startNode);

while (stack.Count > 0)

{

var node = stack.Pop();

var neighbours = node.conected;

foreach (var n in neighbours)

if (!visited.Contains(n))

{

stack.Push(n);

visited.Add(n);

}

}

return visited.ToArray();

}

}

class Node

{

public List<Node> conected = new List<Node>();

}

1. Обход графа в ширину

public Node[] BFS(int start)

{

var startNode = graph[start];

var visited = new List<Node> { startNode };

var queue = new Queue<Node>();

queue.Enqueue(startNode);

while (queue.Count > 0)

{

var node = queue.Dequeue();

var neighbours = node.conected;

foreach (var n in neighbours)

if (!visited.Contains(n))

{

queue.Enqueue(n);

visited.Add(n);

}

}

return visited.ToArray();

}

1. Алгоритм Дейкстры (поиск кратчайшего пути)

static void Main(string[] args)

{

int[,] graph = {

{ 0, 6, 0, 0, 0, 0, 0, 9, 0 },

{ 6, 0, 9, 0, 0, 0, 0, 11, 0 },

{ 0, 9, 0, 5, 0, 6, 0, 0, 2 },

{ 0, 0, 5, 0, 9, 16, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 6, 0, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 16, 0, 2, 0, 1, 6 },

{ 9, 11, 0, 0, 0, 0, 1, 0, 5 },

{ 0, 0, 2, 0, 0, 0, 6, 5, 0 }

};

DijkstraAlgo(graph, 0, 5);

Console.ReadKey();

}

private static int MinimumDistance(int[] distance, bool[] shortestPathTreeSet, int verticesCount)

{

int min = int.MaxValue;

int minIndex = 0;

for (int v = 0; v < verticesCount; ++v)

{

if (shortestPathTreeSet[v] == false && distance[v] <= min)

{

min = distance[v];

minIndex = v;

}

}

return minIndex;

}

private static void Print(int[] distance, int verticesCount)

{

Console.WriteLine("Вершина Расстояние от источника");

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

Console.WriteLine("{0}\t {1}", i, distance[i]);

}

public static void DijkstraAlgo(int[,] graph, int source, int verticesCount)

{

int[] distance = new int[verticesCount];

bool[] shortestPathTreeSet = new bool[verticesCount];

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

{

distance[i] = int.MaxValue;

shortestPathTreeSet[i] = false;

}

distance[source] = 0;

for (int count = 0; count < verticesCount - 1; ++count)

{

int u = MinimumDistance(distance, shortestPathTreeSet, verticesCount);

shortestPathTreeSet[u] = true;

for (int v = 0; v < verticesCount; ++v)

if (!shortestPathTreeSet[v] && Convert.ToBoolean(graph[u, v]) && distance[u] != int.MaxValue && distance[u] + graph[u, v] < distance[v])

distance[v] = distance[u] + graph[u, v];

}

Print(distance, verticesCount);

}

8 Внешняя

using System;

using System.IO;

using System.Linq;

using System.Threading;

namespace ReHomeWork\_4.\_2

{

class Program

{

public static int numColumnSort { get; set; }

public static bool TypeSortUp { get; set; }

public static int countRow { get; set; }

public static bool IsDigit { get; set; }

public static int pow { get; set; }

public static int delay { get; set; }

public static void Main(string[] args)

{

string file = "testTable3.txt";

countRow = File.ReadAllLines(file).Length;

FindPow();

DrawTable(file);

WorkFile.ReadData(file);

WorkFile.SaveData(file);

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

{

SplitThrough(i);

Sort(i+1);

}

DrawNewTable(file);

WorkFile.SaveNewTable(file);

Console.ReadLine();

}

static void FindPow()

{

int count = countRow - 1;

while (count != 1)

{

if (count % 2==1)

{

Console.WriteLine("Недопустимое кол-во строк, не степень 2");

Thread.Sleep(200000);

}

count = count / 2;

pow++;

}

}

static void DrawTable(string file)

{

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

{

string[] text = File.ReadAllLines(file)[i].Split(";");

for (int j = 0; j < text.Length; j++)

{

Console.Write($"{text[j]} ");

}

Console.WriteLine();

}

}

static void SplitThrough(int pow)

{

WorkFile.ClearFile("B.txt");

WorkFile.ClearFile("C.txt");

Console.WriteLine($"Раскидываем по файлам B и С через {Math.Pow(2,pow)}");

int j = 0;

for (int i = 0; i < (countRow - 1) / Math.Pow(2,pow); i++)

{

Thread.Sleep(delay);

if (i % 2 == 1)

{

for (int k = 0; k < Math.Pow(2, pow); k++)

{

string text = File.ReadAllLines("A.txt")[j];

using (StreamWriter sw = new StreamWriter("B.txt", true, System.Text.Encoding.Default))

{

Console.WriteLine($"{text.Split(";")[0]} в B");

sw.WriteLine($"{text}");

}

j++;

}

}

else

{

for (int k = 0; k < Math.Pow(2, pow); k++)

{

string text = File.ReadAllLines("A.txt")[j];

using (StreamWriter sw = new StreamWriter("C.txt", true, System.Text.Encoding.Default))

{

Console.WriteLine($"{text.Split(";")[0]} в C");

sw.WriteLine($"{text}");

}

j++;

}

}

}

Console.WriteLine();

}

static void Sort(int num)

{

WorkFile.ClearFile("A.txt");

int i = 0;

int j = 0;

string elB = "";

string elC = "";

int limit = 0;

int countLinesInB = File.ReadAllLines("B.txt").Length;

int countLinesInC = File.ReadAllLines("C.txt").Length;

Console.WriteLine($"Соединяем из файлов B и C в файл А при этом сортирием их по { num} элементов из каждого файла"); ;

for (int k = 0; k < (countRow - 1) / Math.Pow(2,num); k++)

{

elB = "";

elC = "";

limit += Convert.ToInt32(Math.Pow(2, num-1)); // (k+1)\*num\*2-1;

for (int n = 0; n < Math.Pow(2, num); n++)

{

Thread.Sleep(delay);

if (i < limit && j < limit) //j <= 2 \* (k + 1) -1) // туууууууууууут

{

if (i < countLinesInB)

{

elB = File.ReadAllLines("B.txt")[i];

}

if (j < countLinesInC)

{

elC = File.ReadAllLines("C.txt")[j];

}

Console.WriteLine($"Сравниваем " + elB.Split(';')[0] + " из файла B и " + elC.Split(';')[0] + " из файла C");

SelectSort(elB, elC, ref i, ref j);

Console.WriteLine();

}

else

{

Console.WriteLine("---Записываем остаток");

Console.Write("---");

if (i < limit)

{

elB = File.ReadAllLines("B.txt")[i];

WriteSort(elB, "B");

i++;

}

else if (j < limit)

{

elC = File.ReadAllLines("C.txt")[j];

WriteSort(elC, "C");

j++;

}

Console.WriteLine();

}

}

}

Console.WriteLine();

}

static void SelectSort(string elB, string elC, ref int i, ref int j)

{

if (IsDigit && TypeSortUp)

{

if (Convert.ToInt32(elB.Split(";")[0]) < Convert.ToInt32(elC.Split(";")[0]))

{

WriteSort(elB, "B");

i++;

}

else

{

WriteSort(elC, "C");

j++;

}

}

else if (IsDigit && !TypeSortUp)

{

if (Convert.ToInt32(elB.Split(";")[0]) > Convert.ToInt32(elC.Split(";")[0]))

{

WriteSort(elB, "B");

i++;

}

else

{

WriteSort(elC, "C");

j++;

}

}

else if (!IsDigit && TypeSortUp)

{

if ((int)elB.Split(";")[0][0] < (int)elC.Split(";")[0][0])

{

WriteSort(elB, "B");

i++;

}

else

{

WriteSort(elC, "C");

j++;

}

}

else if (!IsDigit && !TypeSortUp)

{

if ((int)elB.Split(";")[0][0] > (int)elC.Split(";")[0][0])

{

WriteSort(elB, "B");

i++;

}

else

{

WriteSort(elC, "C");

j++;

}

}

}

static void WriteSort(string FirstEl, string file)

{

using (StreamWriter sw = new StreamWriter("A.txt", true, System.Text.Encoding.Default))

{

Console.WriteLine($"Записываем {FirstEl.Split(";")[0]} в A из файла {file}");

sw.WriteLine($"{FirstEl}");

}

}

static void DrawNewTable(string file)

{

int[] arrBanRow = new int[Program.countRow];

int countBan = 0;

int countRowIn = 1;

string textTable = File.ReadAllLines(file)[0];

Console.WriteLine(textTable);

while (countRowIn != countRow )

{

string[] text = File.ReadAllLines("A.txt")[countRowIn - 1].Split(";");

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

{

textTable = File.ReadAllLines(file)[j];

if (text[0] == textTable.Split(";")[numColumnSort] && !WorkFile.IsBanRow(arrBanRow, j))

{

arrBanRow[countBan] = j;

countBan++;

Console.WriteLine(textTable);

countRowIn++;

break;

}

}

}

}

}

}

using System;

using System.IO;

using System.Linq;

using System.Threading;

namespace ReHomeWork\_4.\_2

{

class WorkFile

{

public static void ReadData(string file)

{

Console.WriteLine("\n\nВведите название столбца по которому будете сортировать");

string nameColumnSort = Console.ReadLine();

Console.WriteLine();

string[] text = File.ReadAllLines(file)[0].Split(";");

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

{

if (nameColumnSort == text[i])

{

Program.numColumnSort = i;

}

}

Console.WriteLine("Если хотите сортировать по возрастанию напишите UP, по убыванию DOWN ");

string TypeSort = Console.ReadLine();

Console.WriteLine();

if (TypeSort == "UP")

{

Program.TypeSortUp = true;

}

else if (TypeSort == "DOWN")

{

Program.TypeSortUp = false;

}

Console.WriteLine("Введите задержку в милисекундах ");

Program.delay = Convert.ToInt32(Console.ReadLine());

}

public static void SaveData(string file)

{

ClearFile("A.txt");

using (StreamWriter sw = new StreamWriter("A.txt", true, System.Text.Encoding.Default))

{

Console.WriteLine("Записываем выбранный столбец в файл A");

for (int i = 1; i < Program.countRow; i++)

{

string[] text = File.ReadAllLines(file)[i].Split(";");

Console.Write($"{text[Program.numColumnSort]} ");

sw.WriteLine($"{text[Program.numColumnSort]};{i}");

}

Console.WriteLine();

string[] type = File.ReadAllLines(file)[1].Split(";");

Program.IsDigit = type[Program.numColumnSort].Length == type[Program.numColumnSort].Where(c => char.IsDigit(c)).Count();

Console.WriteLine();

}

}

public static void ClearFile(string file)

{

File.Delete(file);

var myFile = File.Create(file);

myFile.Close();

}

public static void SaveNewTable(string file)

{

ClearFile("NewTable.txt");

using (StreamWriter sw = new StreamWriter("NewTable.txt", true, System.Text.Encoding.Default))

{

int[] arrBanRow = new int[Program.countRow];

int countBan = 0;

int countRowIn = 1;

string textTable = File.ReadAllLines(file)[0];

sw.WriteLine(textTable);

while (countRowIn != Program.countRow )

{

string[] text = File.ReadAllLines("A.txt")[countRowIn - 1].Split(";");

for (int j = 0; j < Program.countRow; j++)

{

textTable = File.ReadAllLines(file)[j];

if (text[0] == textTable.Split(";")[Program.numColumnSort] && !IsBanRow(arrBanRow,j))

{

arrBanRow[countBan] = j;

countBan++;

sw.WriteLine(textTable);

countRowIn++;

break;

}

}

}

}

}

public static bool IsBanRow(int[] arrBanRow, int numRow)

{

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

{

if (numRow == arrBanRow[i])

return true;

}

return false;

}

}

}