

**KAUNO TECHNOLOGIJOS**

**UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**ALGORITMŲ SUDARYMAS IR ANALIZĖ**

**P170B400**

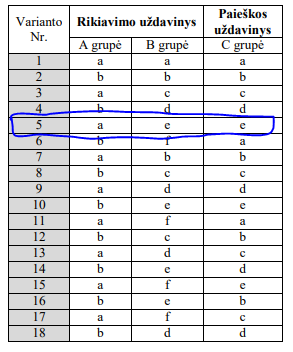
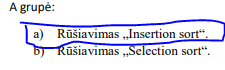
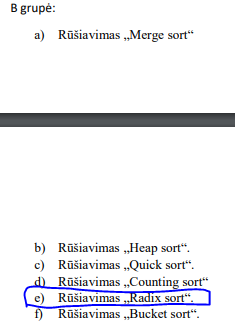
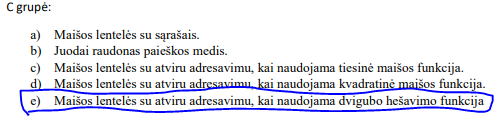
**Individualus darbas**

**Atliko:**

Tadas Laurinaitis IFF-6/8

**Priėmė:**

doc. Mikuckienė Irena

Užduotis

Insertion Sort

Kartai

Kaina

static MyFileArray PerformInsertionSortD(MyFileArray array)

{

int length = array.Length;

1

2N +1

for (int i = 1; i < length; i++)

{

int j = i;

length

N+2

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

{

int k = j - 1;

double temp = array[k];

//array[k] = array[j];

//array[j] = temp;

array.Swap(j, array[j], array[k]);

length \* j

2

j--;

}

}

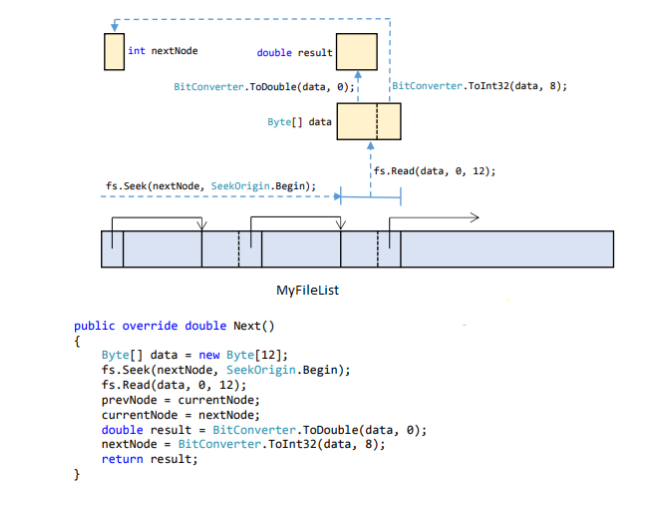
return array;

}

Teorinis sudėtingumas: T = O(N2)

Gautas sudėtingumas: T = 2N2 + 4

|  |  |  |  |
| --- | --- | --- | --- |
| Insertion Sort | | | |
|  | Operatyvinė | | Diskinė |
|  | Masyvas | LinkedList | Masyvas |
| kiekis | laikas | laikas | laikas |
| 100 | 0.0002 | 0.0018 | 0.037 |
| 500 | 0.0005 | 0.0379 | 0.8659 |
| 1000 | 0.0016 | 0.2741 | 3.3445 |
| 1500 | 0.0034 | 1.0411 | 7.2713 |
| 2000 | 0.0061 | 2.402 | 12.931 |
| 2500 | 0.0094 | 4.8906 | 20.1291 |
| 3000 | 0.0132 | 9.439 | 28.9849 |



Radix Sort

Kaina

Kartai

static void PerformRadixSort(int[] arr)

{

int i, j;

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

1

2N +1

for (int shift = 31; shift > -1; --shift)

{

j = 0;

length

d + d + 1

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);

}

}

Teorinis sudėtingumas: O(2d N)

Gautas sudėtingumas: T = 4d\*N+2

|  |  |
| --- | --- |
| Radix Sort | |
| Operatyvinė | |
| kiekis | laikas |
| 1000 | 0.0005 |
| 2000 | 0.0008 |
| 3000 | 0.0012 |
| 4000 | 0.0014 |
| 5000 | 0.0019 |
| 6000 | 0.0021 |
| 7000 | 0.0029 |

Maišos lentelės su atviru adresavimu, kai naudojama dvigubo hešavimo funkcija

|  |  |
| --- | --- |
| Maišos lentelė | |
| Operatyvinė atmintis | |
| kiekis | laikas |
| 0 | 0 |
| 1000 | 0.2168 |
| 2000 | 0.44888 |
| 3000 | 0.6318 |
| 4000 | 0.8848 |
| 5000 | 1.1091 |
| 6000 | 1.2744 |
| 7000 | 1.4798 |

Išvada: Atlikę šį laboratorinį darbą puikiai galime matyti, kad veiksmai greičiau vyksta masyvuose negu sąrašuose, taip pat veiksmai gerokai greičiau vyksta operatyvinėje atmintyje negu diskinėje.

Programos kodas:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Laboratorinis1

{

public class HashTable

{

private DataItem[] hashArray;

private int arraySize;

private int actualSize;

private DataItem bufItem; // for deleted items

public HashTable(int size)

{

arraySize = size;

hashArray = new DataItem[arraySize];

actualSize = 0;

bufItem = new DataItem(-1, "", "");

}

public int getSize()

{

return arraySize;

}

public int GetActualSize()

{

return actualSize;

}

public void displayTable()

{

Console.WriteLine("Table: ");

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

{

if (hashArray[j] != null)

Console.WriteLine("No. " +(j+1) +" Key: " +hashArray[j].GetKey() + "Name and surname: " +hashArray[j].GetName() +" " + hashArray[j].GetSurname());

//else

//Console.WriteLine("\*\* ");

}

Console.WriteLine("");

}

public int hashFunc1(int key)

{

return key % arraySize;

}

public int hashFunc2(int key)

{

return 6 - key % 6;

}

public void Insert(DataItem item)

{

int hashVal = hashFunc1(item.GetKey()); // hash the key

int stepSize = hashFunc2(item.GetKey()); // get step size

// until empty cell or -1

while (hashArray[hashVal] != null && hashArray[hashVal].GetKey() != -1)

{

hashVal += stepSize; // add the step

hashVal %= arraySize; // for wraparound

}

actualSize++;

hashArray[hashVal] = item; // insert item

//return hashVal;

}

public DataItem Delete(int key)

{

int hashVal = hashFunc1(key);

int stepSize = hashFunc2(key); // get step size

while (hashArray[hashVal] != null)

{

if (hashArray[hashVal].GetKey() == key)

{

DataItem temp = hashArray[hashVal]; // save item

hashArray[hashVal] = bufItem; // delete item

actualSize--;

return temp; // return item

}

hashVal += stepSize; // add the step

hashVal %= arraySize; // for wraparound

}

return null; // can't find item

}

public DataItem Find(int key)

{

int hashVal = hashFunc1(key); // hash the key

int stepSize = hashFunc2(key); // get step size

while (hashArray[hashVal] != null)

{

if (hashArray[hashVal].GetKey() == key)

return hashArray[hashVal]; // yes, return item

hashVal += stepSize; // add the step

hashVal %= arraySize; // for wraparound

}

Console.WriteLine("Can't find item with key: " +key);

return null; // can't find item

}

}

public class DataItem

{

private string name;

private string surname;

private int key;

public DataItem(int key, string name, string surname)

{

this.name = name;

this.surname = surname;

this.key = key;

}

public int GetKey()

{

return key;

}

public string GetName()

{

return name;

}

public string GetSurname()

{

return surname;

}

public void SetName(string nameToSet)

{

name = nameToSet;

}

public void SetKey(int keyToSet)

{

key = keyToSet;

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

namespace Laboratorinis1

{

class Program

{

public static void Main(string[] args)

{

//int n = 200;

//string file = "doc.dat";

//MyFileArray arr = new MyFileArray(file, n, 5);

//using (arr.fs = new FileStream(file, FileMode.Open,

//FileAccess.ReadWrite))

//{

// Console.WriteLine("\n FILE ARRAY \n");

// arr.Print(n);

// arr = PerformInsertionSortD(arr);

// arr.Print(n);

//}

//int n = 20;

//string file = "doc.dat";

//MyFileList arr = new MyFileList(file, n, 5);

//using (arr.fs = new FileStream(file, FileMode.Open,

//FileAccess.ReadWrite))

//{

// Console.WriteLine("\n FILE ARRAY \n");

// arr.Print(n);

// //PerformInsertionSortLLD(arr);

// //arr.Print(n);

// double a = arr.get(1);

// Console.WriteLine(a);

//}

//HashTest(7000);

//TestInsertion(1000);

TestRadix(7000);

}

//Insertion/Masyvas/OA

static int[] PerformInsertionSort(int[] array)

{

int length = array.Length;

for (int i = 1; i < length; i++)

{

int j = i;

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

{

int k = j - 1;

int temp = array[k];

array[k] = array[j];

array[j] = temp;

j--;

}

}

return array;

}

//Insertion/LinkedList/OA

public static void InsertSortLL(LinkedList<int> ll)

{

LinkedListNode<int> n = ll.First.Next;

if (n == null)

{

Console.WriteLine("Error");

}

while (n != null)

{

LinkedListNode<int> nn = n;

while ((nn.Previous != null) && (nn.Previous.Value > nn.Value))

{

LinkedListNode<int> t1 = nn.Previous;

LinkedListNode<int> t2 = nn;

ll.Remove(t2.Value);

ll.AddBefore(t1, t2);

nn = t1.Previous;

}

n = n.Next;

}

}

//Radix/Masyvas/OA

static void PerformRadixSort(int[] arr)

{

int i, j;

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

for (int shift = 31; shift > -1; --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);

}

}

//Insertion/Masyvas/Diskine

static MyFileArray PerformInsertionSortD(MyFileArray array)

{

int length = array.Length;

for (int i = 1; i < length; i++)

{

int j = i;

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

{

int k = j - 1;

double temp = array[k];

//array[k] = array[j];

//array[j] = temp;

array.Swap(j, array[j], array[k]);

j--;

}

}

return array;

}

//Insertion/LinkedList/Diskine

static void PerformInsertionSortLLD(MyFileList array)

{

int length = array.Length;

for (int i = 1; i < length; i++)

{

int j = i;

while ((j > 0) && (array.get(j) < array.get(j - 1)))

{

int k = j - 1;

double temp = array.get(k);

double temp2 = array.get(j);

//array[k] = array[j];

//array[j] = temp;

array.Swap(temp2, temp);

j--;

}

}

//return array;

}

static public void TestInsertion(int n)

{

string file = "doc.dat";

Stopwatch watch = new Stopwatch();

int[] array = GenerateArray(n);

LinkedList<int> list = GenerateLinkedList(n);

MyFileArray arr = new MyFileArray("doc.dat", n, 5);

watch.Start();

array = PerformInsertionSort(array);

watch.Stop();

Console.WriteLine("Insertion/Masyvas/OA truko: " + watch.Elapsed);

watch.Reset();

watch.Start();

InsertSortLL(list);

watch.Stop();

Console.WriteLine("Insertion/LinkedList/OA truko: " +watch.Elapsed);

watch.Reset();

watch.Start();

using (arr.fs = new FileStream(file, FileMode.Open,

FileAccess.ReadWrite))

{

//Console.WriteLine("\n FILE ARRAY \n");

//arr.Print(n);

arr = PerformInsertionSortD(arr);

//arr.Print(n);

}

watch.Stop();

Console.WriteLine("Insertion/Masyvas/Diskine truko: " + watch.Elapsed);

}

static public void TestRadix(int n)

{

Stopwatch watch = new Stopwatch();

int[] array = GenerateArray(n);

watch.Start();

PerformRadixSort(array);

watch.Stop();

Console.WriteLine("Radix/Masyvas/Operatyvine truko: " +watch.Elapsed);

}

static public void HashTest(int n)

{

string[] surnames = { "Giedraitis", "Liutenis", "Spokas", "Giedriunas", "Milikonauskas", "Matijosaitis", "Bendrauskas", "Cvirplys", "Asgardietis", "Polukauskis" };

string[] names = {"Jonas", "Antanas", "Robertas", "Kasparas", "Paulius", "Benediktas", "Kirilas", "Mantas", "Vilmantas", "Lukas"};

HashTable hash = new HashTable(n);

Random rand = new Random();

for (int i = 0; i < hash.getSize(); i++)

{

int s1 = rand.Next(10);

int s2 = rand.Next(10);

DataItem nn = new DataItem(i, names[s1], surnames[s2]);

hash.Insert(nn);

}

//hash.displayTable();

Stopwatch watch = new Stopwatch();

Console.WriteLine("---------------Trukmes pradzia--------------");

watch.Start();

for (int i = 0; i < surnames.Length-1; i++)

{

HashTable temp = new HashTable(n);

for (int j = 0; j < hash.GetActualSize(); j++)

{

DataItem item = hash.Find(j);

if(surnames[i] == item.GetSurname())

{

temp.Insert(item);

}

}

temp.displayTable();

}

watch.Stop();

Console.WriteLine("Truko: " +watch.Elapsed);

}

static int[] GenerateArray(int n)

{

Random rnd = new Random();

int[] array = new int[n];

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

{

int c = rnd.Next(100000000);

array[i] = c;

//Console.WriteLine(array[i]);

}

//Console.WriteLine("-------------");

return array;

}

static LinkedList<int> GenerateLinkedList(int n)

{

LinkedList<int> ll = new LinkedList<int>();

Random rnd = new Random();

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

{

int numb = rnd.Next(100000000);

ll.AddLast(numb);

}

return ll;

}

}

//ikopijuotos klases skirtos diskinei atminciai

abstract class DataArray

{

protected int length;

public int Length { get { return length; } }

public abstract double this[int index] { get; }

public abstract void Swap(int j, double a, double b);

public void Print(int n)

{

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

Console.Write(" {0:F5} ", this[i]);

Console.WriteLine();

}

}

abstract class DataList

{

protected int length;

public int Length { get { return length; } }

public abstract double Head();

public abstract double Next();

public abstract void Swap(double a, double b);

public abstract double get(int ind);

public void Print(int n)

{

Console.Write(" {0:F5} ", Head());

for

(int i = 1; i < n; i++)

Console.Write(" {0:F5} ", Next());

Console.WriteLine();

}

}

class MyFileArray : DataArray

{

public MyFileArray(string filename, int n, int seed)

{

double[] data = new double[n];

length = n;

Random rand = new Random(seed);

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

{

data[i] = rand.NextDouble();

}

if (File.Exists(filename)) File.Delete(filename);

try

{

using (BinaryWriter writer = new BinaryWriter(File.Open(filename,

FileMode.Create)))

{

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

writer.Write(data[j]);

}

}

catch (IOException ex)

{

Console.WriteLine(ex.ToString());

}

}

public FileStream fs { get; set; }

public override double this[int index]

{

get

{

Byte[] data = new Byte[8];

fs.Seek(8 \* index, SeekOrigin.Begin);

fs.Read(data, 0, 8);

double result = BitConverter.ToDouble(data, 0);

return result;

}

}

public override void Swap(int j, double a, double b)

{

Byte[] data = new Byte[16];

BitConverter.GetBytes(a).CopyTo(data, 0);

BitConverter.GetBytes(b).CopyTo(data, 8);

fs.Seek(8 \* (j - 1), SeekOrigin.Begin);

fs.Write(data, 0, 16);

}

}

class MyFileList : DataList

{

int prevNode;

int currentNode;

int nextNode;

public MyFileList(string filename, int n, int seed)

{

length = n;

Random rand = new Random(seed);

if (File.Exists(filename)) File.Delete(filename);

try

{

using (BinaryWriter writer = new BinaryWriter(File.Open(filename,

FileMode.Create)))

{

writer.Write(4);

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

{

writer.Write(rand.NextDouble());

writer.Write((j + 1) \* 12 + 4);

}

}

}

catch (IOException ex)

{

Console.WriteLine(ex.ToString());

}

}

public FileStream fs { get; set; }

public override double Head()

{

Byte[] data = new Byte[12];

fs.Seek(0, SeekOrigin.Begin);

fs.Read(data, 0, 4);

currentNode = BitConverter.ToInt32(data, 0);

prevNode = -1;

fs.Seek(currentNode, SeekOrigin.Begin);

fs.Read(data, 0, 12);

double result = BitConverter.ToDouble(data, 0);

nextNode = BitConverter.ToInt32(data, 8);

return result;

}

public override double Next()

{

Byte[] data = new Byte[12];

fs.Seek(nextNode, SeekOrigin.Begin);

fs.Read(data, 0, 12);

prevNode = currentNode;

currentNode = nextNode;

double result = BitConverter.ToDouble(data, 0);

nextNode = BitConverter.ToInt32(data, 8);

return result;

}

public override double get(int ind)

{

Byte[] data = new Byte[12];

fs.Seek(ind, SeekOrigin.Begin);

fs.Read(data, 0, 12);

double result = BitConverter.ToDouble(data, 0);

return result;

}

public override void Swap(double a, double b)

{

Byte[] data;

fs.Seek(prevNode, SeekOrigin.Begin);

data = BitConverter.GetBytes(a);

fs.Write(data, 0, 8);

fs.Seek(currentNode, SeekOrigin.Begin);

data = BitConverter.GetBytes(b);

fs.Write(data, 0, 8);

}

}

}