|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
|  | | |
| Домашнее задание № 1 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| **Сортировка и поиск** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-81 |
| Варианты: | 3 |
| Студент: | Кондратьев Игорь |
|  |  |
|  |  |
| Преподаватель: | Георгий Анатольевич Щукин |
|  |  |
|  | | |
| Новосибирск | | |
| 2019 | | |

Постановка задачи  
Дано множество записей о людях, каждая запись содержит такие основные поля, как имя, фамилия и год рождения. Для множества записей требуется реализовать сортировку или поиск с использованием алгоритма/структуры данных, указанного в варианте. Реализовать алгоритм пирамидальной сортировки массива записей. Должна быть возможность сортировки записей по строковому полю (например, имя или фамилия) и по числовому полю (например, год рождения).

1. Описание использованных алгоритмов  
   Использован алгоритм пирамидальной сортироки. Общая идея пирамидальной сортировки заключается в том, что сначала строится пирамида из элементов исходного массива, а затем осуществляется сортировка элементов.  
   1 этап: Построение пирамиды. Определяем правую часть дерева, начиная с n/2-1 (нижний уровень дерева). Берем элемент левее этой части массива и просеиваем его сквозь пирамиду по пути, где находятся меньшие его элементы, которые одновременно поднимаются вверх; из двух возможных путей выбираете путь через меньший элемент.  
   2 этап: Сортировка на построенной пирамиде. Берем последний элемент массива в качестве текущего. Меняем верхний (наименьший) элемент массива и текущий местами. Текущий элемент (он теперь верхний) просеиваем сквозь n-1 элементную пирамиду. Затем берем предпоследний элемент и т.д.  
     
   Главный алгоритм:  
   humans = СканироватьМассивЛюдейИзФайла(“Humans.txt”)  
   hs = экземпляр класса HeapSort  
   hs.Сортировать(СравниватьПоИмени, humans)  
   ЗаписатьМассивВФайл(“SortedHumans.txt”, humans)  
     
   hs.Сортировать(СравниватьПоФамилии, humans)  
   ЗаписатьМассивВФайл(“SortedHumans.txt”, humans)  
     
   hs.Сортировать(СравниватьПоДатеРождения, humans)  
   ЗаписатьМассивВФайл(“SortedHumans.txt”, humans)
2. Текст реализации алгоритмов на языке программирования

Human.cs:  
  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MPA\_HW1

{

public class Human : IComparer<Human>

{

private string name;

private string secondName;

private int birthYear;

public Human(string name, string secondName, int birthYear)

{

this.name = name;

this.secondName = secondName;

this.birthYear = birthYear;

}

public override bool Equals(object obj)

{

if (obj is Human)

{

Human equatableObject = obj as Human;

return this.Name == equatableObject.Name &&

this.SecondName == equatableObject.SecondName &&

this.BirthYear == equatableObject.BirthYear;

}

else

{

return false;

}

}

public override int GetHashCode()

{

int hashCode = 1;

foreach (char letter in this.Name)

{

hashCode += letter;

}

foreach (char letter in this.SecondName)

{

hashCode += letter;

}

hashCode \*= this.BirthYear;

return hashCode;

}

public int Compare(Human x, Human y)

{

return (x.BirthYear).CompareTo(y.birthYear);

}

public string Name

{

get { return name; }

}

public string SecondName

{

get { return secondName; }

}

public int BirthYear

{

get { return birthYear; }

}

}

}  
  
  
HeapSort.cs:  
  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MPA\_HW1

{

public class HeapSort

{

private int heapSize;

private Compare compareDelegate;

public void Sort(List<Human> arr, Compare compareDelegate)

{

this.compareDelegate = compareDelegate;

BuildHeap(arr);

for (int i = arr.Count - 1; i >= 0; i--)

{

Swap(arr, 0, i);

heapSize--;

Heapify(arr, 0);

}

}

private void BuildHeap(List<Human> arr)

{

heapSize = arr.Count - 1;

for (int i = heapSize / 2; i >= 0; i--)

{

Heapify(arr, i);

}

}

private void Swap(List<Human> arr, int x, int y)

{

Human temp = arr[x];

arr[x] = arr[y];

arr[y] = temp;

}

private void Heapify(List<Human> arr, int index)

{

int left = 2 \* index + 1;

int right = 2 \* index + 2;

int largest = index;

if (left <= heapSize && compareDelegate.Invoke(arr[left], arr[index]) > 0)

{

largest = left;

}

if (right <= heapSize && compareDelegate.Invoke(arr[right], arr[largest]) > 0)

{

largest = right;

}

if (largest != index)

{

Swap(arr, index, largest);

Heapify(arr, largest);

}

}

}

}

Program.cs:  
using System;

using System.Diagnostics;

using System.IO;

using System.Collections.Generic;

using System.Net;

using System.Text;

using System.Linq;

using System.Text.RegularExpressions;

namespace MPA\_HW1

{

public delegate int Compare(Human human1, Human human2);

public class Program

{

public static int CompareByName(Human human1, Human human2)

{

return human1.Name.CompareTo(human2.Name);

}

public static int CompareBySecondName(Human human1, Human human2)

{

return human1.SecondName.CompareTo(human2.SecondName);

}

public static int CompareByYearBirth(Human human1, Human human2)

{

return human1.BirthYear.CompareTo(human2.BirthYear);

}

static private List<Human> ScanHumanList(string fileName)

{

List<Human> humans = new List<Human>();

foreach (string line in File.ReadLines(fileName))

{

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

humans.Add(new Human(lines[1], lines[0], Convert.ToInt32(lines[2])));

}

return humans;

}

static private void WriteArray(StreamWriter streamWriter, List<Human> arr)

{

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

{

streamWriter.WriteLine(arr[i].SecondName + " " + arr[i].Name + " " + arr[i].BirthYear);

}

streamWriter.WriteLine();

}

public static string getBetween(string strSource, string strStart, string strEnd)

{

int Start, End;

if (strSource.Contains(strStart) && strSource.Contains(strEnd))

{

Start = strSource.IndexOf(strStart, 0) + strStart.Length;

End = strSource.IndexOf(strEnd, Start);

return strSource.Substring(Start, End - Start);

}

else

{

return "";

}

}

static private List<Human> GenerateHumansList(int count)

{

string[] names = File.ReadAllLines("names.txt");

string[] secondNames = File.ReadAllLines("secondnames.txt");

Random random = new Random();

List<Human> randomHumans = new List<Human>(count);

int randomNumber;

string randomName, randomSecondName;

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

{

randomNumber = random.Next(5490);

randomName = names[random.Next(names.Length)];

randomSecondName = secondNames[random.Next(secondNames.Length)];

randomHumans.Add(new Human(randomName, randomSecondName, random.Next(1930, 2019)));

}

return randomHumans;

}

static void Main(string[] args)

{

List<Human> humans = ScanHumanList("Humans.txt");

HeapSort hs = new HeapSort();

FileStream fileStream = new FileStream("SortedHumans.txt", FileMode.Create);

StreamWriter streamWriter = new StreamWriter(fileStream);

fileStream.Seek(0, SeekOrigin.End);

hs.Sort(humans, CompareByName);

streamWriter.WriteLine("Sorted by name:");

WriteArray(streamWriter, humans);

hs.Sort(humans, CompareBySecondName);

streamWriter.WriteLine("Sorted by secondname:");

WriteArray(streamWriter, humans);

hs.Sort(humans, CompareByYearBirth);

streamWriter.WriteLine("Sorted by year birth:");

WriteArray(streamWriter, humans);

streamWriter.Close();

//Time measurement

Stopwatch stopwatch = new Stopwatch();

humans = GenerateHumansList(10000);

stopwatch.Start();

hs.Sort(humans, CompareByName);

stopwatch.Stop();

Console.WriteLine(stopwatch.Elapsed);

stopwatch.Reset();

humans = GenerateHumansList(10000);

stopwatch.Start();

humans.Sort(CompareByName);

stopwatch.Stop();

Console.WriteLine(stopwatch.Elapsed);

stopwatch.Reset();

}

}

}

1. Результаты тестирования

using System;

using MPA\_HW1;

using System.Collections.Generic;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace MPA\_HW1.Tests

{

[TestClass]

public class ProgramTest

{

[TestMethod]

public void Sort\_Empty\_EmptyReturned\_ByName()

{

//arrange

HeapSort heapSort = new HeapSort();

List<Human> actual = new List<Human>();

List<Human> expected = new List<Human>();

//act

heapSort.Sort(actual, Program.CompareByName);

//assert

CollectionAssert.AreEqual(expected, actual);

}

[TestMethod]

public void Sort\_Empty\_EmptyReturned\_BySecondName()

{

//arrange

HeapSort heapSort = new HeapSort();

List<Human> actual = new List<Human>();

List<Human> expected = new List<Human>();

//act

heapSort.Sort(actual, Program.CompareBySecondName);

//assert

CollectionAssert.AreEqual(expected, actual);

}

[TestMethod]

public void Sort\_Empty\_EmptyReturned\_ByYearBirth()

{

//arrange

HeapSort heapSort = new HeapSort();

List<Human> actual = new List<Human>();

List<Human> expected = new List<Human>();

//act

heapSort.Sort(actual, Program.CompareByYearBirth);

//assert

CollectionAssert.AreEqual(expected, actual);

}

[TestMethod]

public void Sort\_HumansList\_SortedHumansList\_ByName()

{

//arrange

HeapSort heapSort = new HeapSort();

List<Human> expected = new List<Human>()

{

new Human("Alesha", "Popov", 2003),

new Human("Amin", "Avarin", 2000),

new Human("Igor", "Baranov", 1990),

new Human("Ivan", "Kuleshov", 2010),

new Human("Ivan", "Ivanov", 2000),

new Human("Petr", "Kuleshov", 1999),

new Human("Vlad", "Klemenko", 2001),

new Human("Yan", "Koght", 1998),

};

//act

List<Human> actual = new List<Human>()

{

new Human("Ivan", "Ivanov", 2000),

new Human("Ivan", "Kuleshov", 2010),

new Human("Petr", "Kuleshov", 1999),

new Human("Igor", "Baranov", 1990),

new Human("Yan", "Koght", 1998),

new Human("Alesha", "Popov", 2003),

new Human("Amin", "Avarin", 2000),

new Human("Vlad", "Klemenko", 2001),

};

heapSort.Sort(actual, Program.CompareByName);

//assert

CollectionAssert.AreEqual(expected, actual);

}

[TestMethod]

public void Sort\_HumansList\_SortedHumansList\_BySecondName()

{

//arrange

HeapSort heapSort = new HeapSort();

List<Human> expected = new List<Human>()

{

new Human("Amin", "Avarin", 2000),

new Human("Igor", "Baranov", 1990),

new Human("Ivan", "Ivanov", 2000),

new Human("Vlad", "Klemenko", 2001),

new Human("Yan", "Koght", 1998),

new Human("Petr", "Kuleshov", 1999),

new Human("Ivan", "Kuleshov", 2010),

new Human("Alesha", "Popov", 2003),

};

//act

List<Human> actual = new List<Human>()

{

new Human("Alesha", "Popov", 2003),

new Human("Amin", "Avarin", 2000),

new Human("Igor", "Baranov", 1990),

new Human("Ivan", "Kuleshov", 2010),

new Human("Ivan", "Ivanov", 2000),

new Human("Petr", "Kuleshov", 1999),

new Human("Vlad", "Klemenko", 2001),

new Human("Yan", "Koght", 1998),

};

heapSort.Sort(actual, Program.CompareBySecondName);

//assert

CollectionAssert.AreEqual(expected, actual);

}

[TestMethod]

public void Sort\_HumansList\_SortedHumansList\_ByYearBirth()

{

//arrange

HeapSort heapSort = new HeapSort();

List<Human> expected = new List<Human>()

{

new Human("Igor", "Baranov", 1990),

new Human("Yan", "Koght", 1998),

new Human("Petr", "Kuleshov", 1999),

new Human("Ivan", "Ivanov", 2000),

new Human("Amin", "Avarin", 2000),

new Human("Vlad", "Klemenko", 2001),

new Human("Alesha", "Popov", 2003),

new Human("Ivan", "Kuleshov", 2010),

};

//act

List<Human> actual = new List<Human>()

{

new Human("Alesha", "Popov", 2003),

new Human("Amin", "Avarin", 2000),

new Human("Igor", "Baranov", 1990),

new Human("Ivan", "Kuleshov", 2010),

new Human("Ivan", "Ivanov", 2000),

new Human("Petr", "Kuleshov", 1999),

new Human("Vlad", "Klemenko", 2001),

new Human("Yan", "Koght", 1998),

};

heapSort.Sort(actual, Program.CompareByYearBirth);

//assert

CollectionAssert.AreEqual(expected, actual);

}

}

}

1. Результаты исследования времени работы алгортмов

сек

Кол-во элементов

сек

Кол-во элементов

1. Сравнение полученных результатов с ожидаемыми:

Ожидаемые результаты для пирамидальной сортировки в среднем вычисляются, как **O((n/2)·log2n)**.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1000000 | 2000000 | 3000000 | 4000000 | 5000000 | 6000000 | 7000000 | 8000000 | 9000000 | 10000000 |
| 6.66 | 13.2 | 19.0 | 23.6 | 31.9 | 42.72 | 55.33 | 66.3 | 80.00 | 101.56 |

Полученные результаты пирамидальной сортировки по фамилиям.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1000000 | 2000000 | 3000000 | 4000000 | 5000000 | 6000000 | 7000000 | 8000000 | 9000000 | 10000000 |
| 5.74 | 12.75 | 18.17 | 23.36 | 30.29 | 40.27 | 51.83 | 61.3 | 75.5 | 96.4 |

Полученные результаты близки с ожидаемыми. Это подтверждает представленные значения в таблицах и графиках, представленных выше.

1. Вывод  
   В процессе выполнения задачи я научился применять делегаты для сортировки по любому из признаку, провел модульное тестирование программы, а также построил графики, демонстрирующие зависимость времени работы от размера массива и сравение время работы написанной сортировки со встроенной сортировкой c# .NET. Полученные результаты совпадают с ожидаемыми/теоретическими.