Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Комбинаторные алгоритмы

Лабораторная работа №2

«Поиск»

Выполнил Кудряшов А.А.

Группа 1121

Преподаватель

Павловская Т.А.

Санкт - Петербург

2012 г.

Вариант 1

- последовательный поиск по неупорядоченному файлу
- бинарный поиск
- поиск с помощью стандартного алгоритма в стандартном контейнере.

1) Последовательный поиск в неупорядоченном файле

Сравнение каждого элемента файла (ключа) с запросом, в случае совпадения поиск считается удачным, в противном случае – неудачным.

2) Бинарный поиск в упорядоченном файле.

Алгоритм заключается в делении массива (списка) ключей и сравнение запроса с крайним элементом, нахождение половины, в диапазон значений ключей которой попадает занчение запроса, и повторение деления, до момента когда количество элементов остаточного массива будет равным 1. В случае, если данный элемент равен запросу – поиск удачен, в противном случае – нет.

1) Стандартный метод Find класса List<T>.

По сути является видоизмененным алгоритмом последовательного поиска по неупорядоченному списку

Текст программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using System.Text.RegularExpressions;

namespace ConsoleApplication1
{
   class Find
   {
      public static List<double> FileToDoubleArray_Figures(string Adress)
      {
         string[] HelpVeriable = File.ReadAllText(Adress).Split(new[] { "\n", "\r", " " },
StringSplitOptions.RemoveEmptyEntries);
         List<double> ResultOfMethod = new List<double>();
         for (int i = 1; i <= double.Parse(HelpVeriable[0]); ++i)
         {
            ResultOfMethod.Add(double.Parse(HelpVeriable[i]));
         }
         return ResultOfMethod;
      }
      public static List<double> FileToDoubleArray_Requst(string Adress)
      {
         string[] HelpVeriable = File.ReadAllText(Adress).Split(new[] { "\n", "\r", " " },
StringSplitOptions.RemoveEmptyEntries);
         List<double> ResultOfMethod = new List<double>();
```

```
            for (int i = int.Parse(HelpVeriable[0]) + 1; i < HelpVeriable.Length; ++i)
            {
                ResultOfMethod.Add(double.Parse(HelpVeriable[i]));
            }
            return ResultOfMethod;
        }



//Последовательный поиск


            public static string FirstFind(List<double> Figures, List<double> Requst)

        {
            Stopwatch timer = new Stopwatch();
            timer.Start();
            for (int i_1 = 0; i_1 < Requst.Count; ++i_1)
            {
                for (int i_2 = 0; i_2 < Figures.Count; ++i_2)
                {
                    if (Requst[i_1] == Figures[i_2])
                        break;
                }
            }
            timer.Stop();
            return timer.Elapsed.TotalMilliseconds.ToString();
        }

//Бинарный поиск

        public static string SecondFind(List<double> Figures, List<double> Requst)
        {
            Figures.Sort();
            Stopwatch timer = new Stopwatch();
            timer.Start();
            for (int i_1 = 0; i_1 < Requst.Count; ++i_1)
            {
                int FirstElement = 0;
                int LastElement = Figures.Count - 1;

                for (int i = FirstElement + (int)Math.Floor((decimal)(LastElement - FirstElement)/2); LastElement >
FirstElement; i = FirstElement +(int)Math.Floor((decimal)(LastElement - FirstElement)/2))
                {
                    if (Requst[i_1] == Figures[i])
                        break;
                    if (Requst[i_1] > Figures[i])
                    {
                        FirstElement = i + 1;
                        continue;
                    }
                    if (Requst[i_1] < Figures[i])
                    {
                        LastElement = i;
                        continue;
                    }
                }
            }
            timer.Stop();
```

```
            return timer.Elapsed.TotalMilliseconds.ToString();
        }



// Метод Find

        public static string ThirdFind(List<double> Figures, List<double> Requst)
        {

            Stopwatch timer = new Stopwatch();
            timer.Start();
            for (int i_1 = 0; i_1 < Requst.Count; ++i_1)
            {
                Figures.Find(delegate(double Figure) { return Requst[i_1] == Figure; });
            }
            timer.Stop();
            return timer.Elapsed.TotalMilliseconds.ToString();
        }




        public static string GetOutString(string Adress)
        {
            string[] HelpVeriable = File.ReadAllText(Adress).Split(new[] { "\n", "\r", " " },
StringSplitOptions.RemoveEmptyEntries);
            string Elements = HelpVeriable[0];
            string Var = Adress[Adress.Length - 1].ToString();
            string Distribution = Adress[Adress.Length - 3].ToString();
            string Lucky = Adress[17].ToString();
            return Var + "\t" + Elements + "\t" + Lucky + "\t" + Distribution;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                string[] Files = Directory.GetFiles(@"d:\Артем\F_POISK");
                File.AppendAllText(@"d:\Поиск.xls", "\nFind 1\nVar\tElements\tLucky\tDistribution\tTime\n");
                for (int f = 0; f < Files.Length; ++f)
                {
                    List<double> Figures = Find.FileToDoubleArray_Figures(Files[f]);
                    List<double> Requst = Find.FileToDoubleArray_Requst(Files[f]);
                    string time = Find.FirstFind(Figures, Requst);
                    File.AppendAllText(@"d:\Поиск.xls", Find.GetOutString(Files[f]) + "\t" + time + " \n");
                    time = Find.ThirdFind(Figures, Requst);

                }
                File.AppendAllText(@"d:\Поиск.xls", "\nFind 2\nVar\tElements\tLucky\tDistribution\tTime\n");
                for (int f = 0; f < Files.Length; ++f)
                {
                    List<double> Figures = Find.FileToDoubleArray_Figures(Files[f]);
                    List<double> Requst = Find.FileToDoubleArray_Requst(Files[f]);
                    string time = Find.SecondFind(Figures, Requst);
                    File.AppendAllText(@"d:\Поиск.xls", Find.GetOutString(Files[f]) + "\t" + time + " \n");
                }
                File.AppendAllText(@"d:\Поиск.xls", "\nFind 3\nVar\tElements\tLucky\tDistribution\tTime\n");
```

```
            for (int f = 0; f < Files.Length; ++f)
            {
                List<double> Figures = Find.FileToDoubleArray_Figures(Files[f]);
                List<double> Requst = Find.FileToDoubleArray_Requst(Files[f]);
                string time = Find.ThirdFind(Figures, Requst);
                File.AppendAllText(@"d:\Поиск.xls", Find.GetOutString(Files[f]) + "\t" + time + " \n");
            }
        }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }
  }
}
```

Результаты замеров

Последовательный поиск | Бинарный поиск | Метод Find

Var	Elem	N/Y	Dis.	Time	Var	Elem	N/Y	Dis.	Time	Var	Elem	Lucky	Dis.	Time
1	64	N	E	0,3837	1	64	N	E	2,0839	1	64	N	E	0,6649
2	64	N	E	0,3813	2	64	N	E	1,5766	2	64	N	E	0,6839
3	64	N	E	0,3853	3	64	N	E	1,5527	3	64	N	E	0,6653
1	64	N	Z	0,3808	1	64	N	Z	1,5653	1	64	N	Z	0,6702
2	64	N	Z	0,3808	2	64	N	Z	1,558	2	64	N	Z	0,6698
3	64	N	Z	0,3808	3	64	N	Z	1,5511	3	64	N	Z	0,6645
1	128	N	E	0,8217	1	128	N	E	1,8104	1	128	N	E	1,3327
2	128	N	E	0,7715	2	128	N	E	1,8096	2	128	N	E	1,3323
3	128	N	E	0,7694	3	128	N	E	1,7918	3	128	N	E	1,359
1	128	N	Z	0,7703	1	128	N	Z	1,7894	1	128	N	Z	1,3339
2	128	N	Z	0,7731	2	128	N	Z	1,789	2	128	N	Z	1,3319
3	128	N	Z	0,769	3	128	N	Z	1,7967	3	128	N	Z	1,3821
1	256	N	E	1,5256	1	256	N	E	2,0252	1	256	N	E	2,6411
2	256	N	E	1,5268	2	256	N	E	2,0244	2	256	N	E	2,6391
3	256	N	E	1,5268	3	256	N	E	2,0337	3	256	N	E	2,6638
1	256	N	Z	1,528	1	256	N	Z	2,0268	1	256	N	Z	2,6358
2	256	N	Z	1,5276	2	256	N	Z	2,026	2	256	N	Z	2,6363
3	256	N	Z	1,6192	3	256	N	Z	2,0179	3	256	N	Z	2,6395
1	512	N	E	3,039	1	512	N	E	2,2643	1	512	N	E	5,2746
2	512	N	E	3,039	2	512	N	E	2,2679	2	512	N	E	5,3933
3	512	N	E	3,1164	3	512	N	E	2,3113	3	512	N	E	5,2527
1	512	N	Z	3,0905	1	512	N	Z	2,2785	1	512	N	Z	5,2653
2	512	N	Z	3,1813	2	512	N	Z	2,2845	2	512	N	Z	5,2527

3	512	N	Z	3,0419	3	512	N	Z	2,2999	3	512	N	Z	5,273
1	1024	N	E	6,1223	1	1024	N	E	2,5321	1	1024	N	E	10,6234
2	1024	N	E	6,2244	2	1024	N	E	2,5033	2	1024	N	E	10,6165
3	1024	N	E	6,1875	3	1024	N	E	2,4973	3	1024	N	E	10,6651
1	1024	N	Z	6,0789	1	1024	N	Z	2,5021	1	1024	N	Z	10,6112
2	1024	N	Z	6,1324	2	1024	N	Z	2,5029	2	1024	N	Z	10,7121
3	1024	N	Z	6,0716	3	1024	N	Z	2,4916	3	1024	N	Z	10,5816
1	64	Y	E	0,2119	1	64	Y	E	1,2059	1	64	Y	E	0,3695
2	64	Y	E	0,2086	2	64	Y	E	1,2156	2	64	Y	E	0,3646
3	64	Y	E	0,2131	3	64	Y	E	1,2249	3	64	Y	E	0,3731
1	64	Y	Z	0,0992	1	64	Y	Z	1,2938	1	64	Y	Z	0,173
2	64	Y	Z	0,0984	2	64	Y	Z	1,2379	2	64	Y	Z	0,1795
3	64	Y	Z	0,1021	3	64	Y	Z	1,2885	3	64	Y	Z	0,1762
1	128	Y	E	0,4072	1	128	Y	E	1,4482	1	128	Y	E	0,7366
2	128	Y	E	0,393	2	128	Y	E	1,4457	2	128	Y	E	0,6864
3	128	Y	E	0,4072	3	128	Y	E	1,483	3	128	Y	E	0,6985
1	128	Y	Z	0,1584	1	128	Y	Z	1,4879	1	128	Y	Z	0,2743
2	128	Y	Z	0,1539	2	128	Y	Z	1,4773	2	128	Y	Z	0,2662
3	128	Y	Z	0,2086	3	128	Y	Z	1,5175	3	128	Y	Z	0,2832
1	256	Y	E	0,7682	1	256	Y	E	1,7075	1	256	Y	E	1,3335
2	256	Y	E	0,7654	2	256	Y	E	1,669	2	256	Y	E	1,3242
3	256	Y	E	0,7792	3	256	Y	E	1,6613	3	256	Y	E	1,3457
1	256	Y	Z	0,265	1	256	Y	Z	1,6864	1	256	Y	Z	0,4546
2	256	Y	Z	0,2564	2	256	Y	Z	1,7586	2	256	Y	Z	0,4449
3	256	Y	Z	0,2723	3	256	Y	Z	1,7148	3	256	Y	Z	0,4692
1	512	Y	E	1,5345	1	512	Y	E	1,9	1	512	Y	E	2,6533
2	512	Y	E	4,8325	2	512	Y	E	2,0953	2	512	Y	E	2,6249
3	512	Y	E	1,5252	3	512	Y	E	2,1666	3	512	Y	E	2,6699
1	512	Y	Z	0,4659	1	512	Y	Z	1,9826	1	512	Y	Z	0,7986
2	512	Y	Z	0,4424	2	512	Y	Z	1,9215	2	512	Y	Z	0,8639
3	512	Y	Z	0,4639	3	512	Y	Z	2,3368	3	512	Y	Z	0,8047
1	1024	Y	E	3,086	1	1024	Y	E	2,2708	1	1024	Y	E	5,3726
2	1024	Y	E	3,0791	2	1024	Y	E	2,1585	2	1024	Y	E	5,3021

3	1024	Y	E	3,1541	3	1024	Y	E	2,1545	3	1024	Y	E	5,2223
1	1024	Y	Z	0,7974	1	1024	Y	Z	2,1925	1	1024	Y	Z	1,5333
2	1024	Y	Z	0,7929	2	1024	Y	Z	2,1601	2	1024	Y	Z	1,3724
3	1024	Y	Z	0,8464	3	1024	Y	Z	2,1609	3	1024	Y	Z	1,4648

Сводные таблицы

Метод последовательного поиска						
	N		N Итог	Y		Y Итог
	E	Z		E	Z	
64	0,383433	0,3808	0,382117	0,2112	0,0999	0,15555
128	0,787533	0,7708	0,779167	0,402467	0,173633	0,28805
256	1,5264	1,558267	1,542333	0,770933	0,264567	0,51775
512	3,0648	3,104567	3,084683	2,630733	0,4574	1,544067
1024	6,178067	6,0943	6,136183	3,1064	0,812233	1,959317

Метод бинарного поиска						
	N		N Итог	Y		Y Итог
	E	Z		E	Z	
64	0,383433	0,3808	0,382117	0,2112	0,0999	0,15555
128	0,787533	0,7708	0,779167	0,402467	0,173633	0,28805
256	1,5264	1,558267	1,542333	0,770933	0,264567	0,51775
512	3,0648	3,104567	3,084683	2,630733	0,4574	1,544067
1024	6,178067	6,0943	6,136183	3,1064	0,812233	1,959317

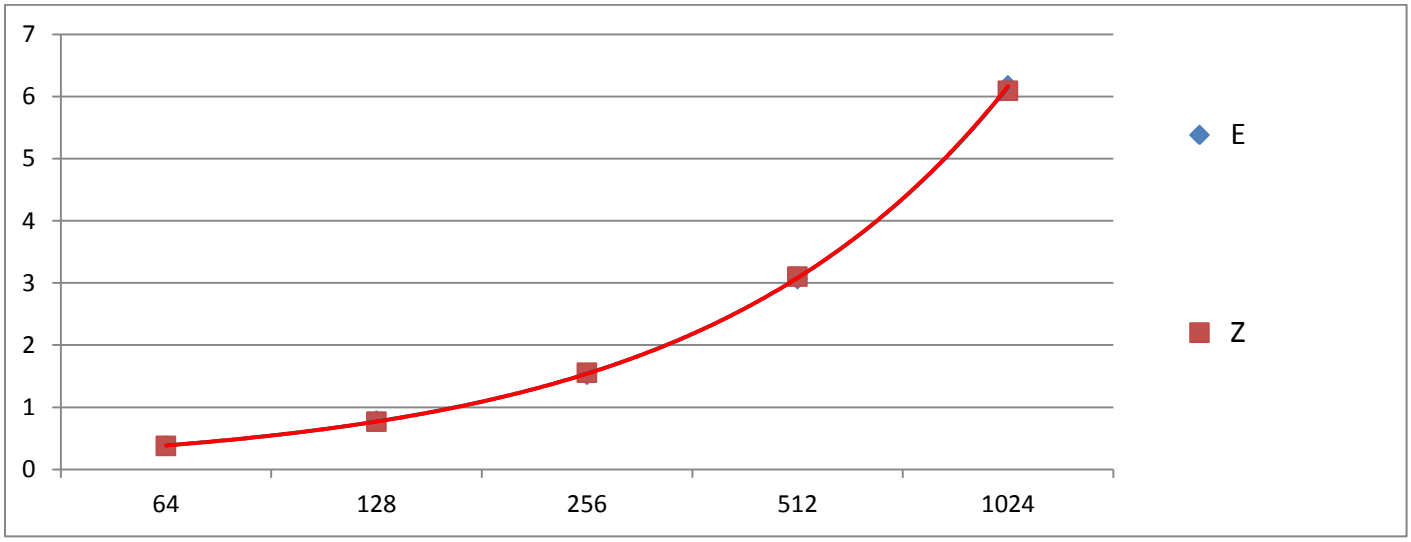
Метод Find						
	N		N Итог	Y		Y Итог
	E	Z		E	Z	
64	0,671367	0,668167	0,669767	0,369067	0,176233	0,27265
128	1,341333	1,3493	1,345317	0,707167	0,274567	0,490867
256	2,648	2,6372	2,6426	1,334467	0,456233	0,89535
512	5,306867	5,263667	5,285267	2,649367	0,8224	1,735883
1024	10,635	10,63497	10,63498	5,299	1,456833	3,377917

Графики

1. Последовательный поиск

Неудачный поиск

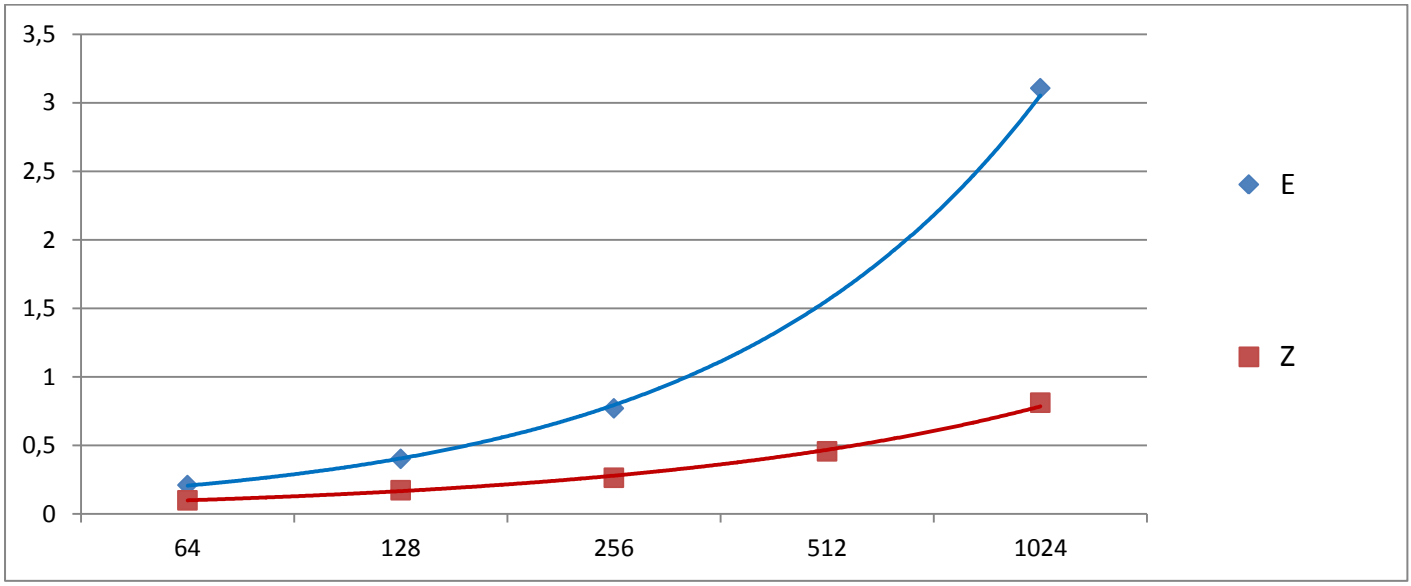
E – равномерное распределение; Z - распределение по закону Зипфа



- Так как, в данном случае, горизонтальная ось имеет логарифмический вид, а функция Y(x) – линейная, то визуализация аппроксимирующей функции имеет экспоненциальный вид.

Удачный поиск

E – равномерное распределение; Z - распределение по закону Зипфа

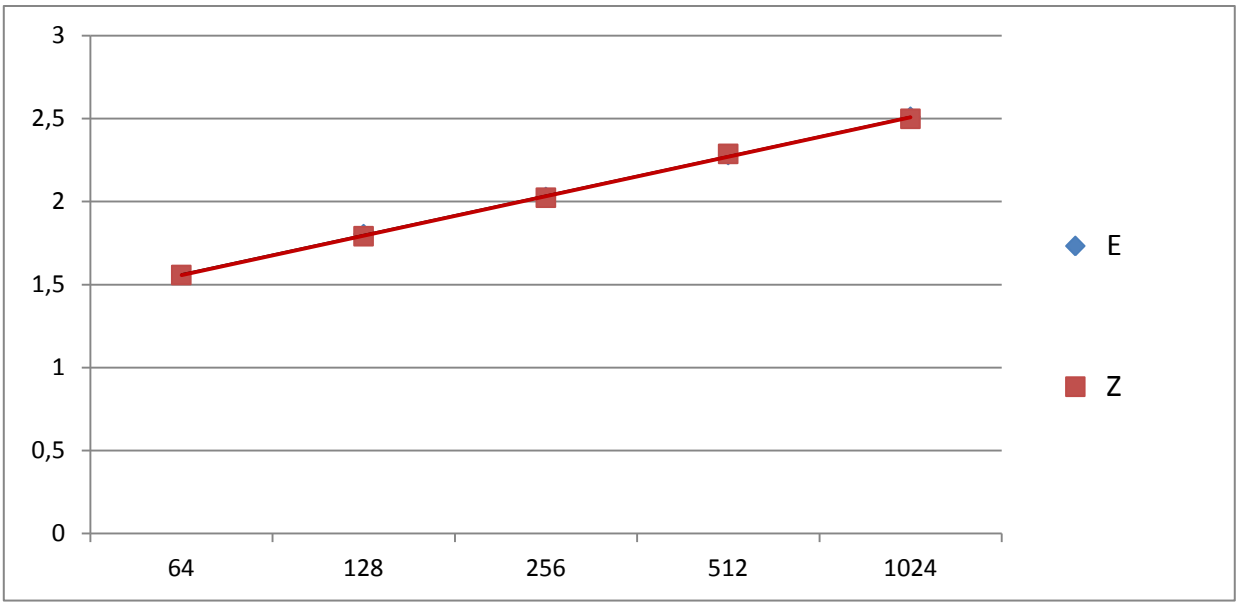


- Так как, в данном случае, горизонтальная ось имеет логарифмический вид, а функция Y(x) – линейная, то визуализация аппроксимирующей функции имеет экспоненциальный вид.

2.Бинарный поиск

Неудачный поиск

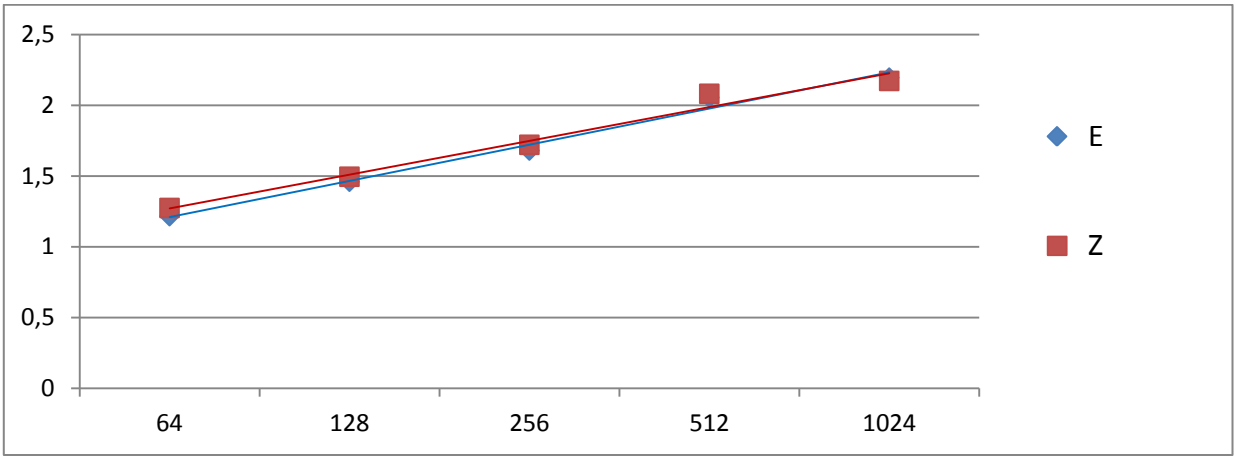
E – равномерное распределение; Z - распределение по закону Зипфа



- Так как, в данном случае, горизонтальная ось имеет логарифмический вид, а функция Y(x) – логарифмическая, то визуализация аппроксимирующей функции имеет линейный вид.

Удачный поиск

E – равномерное распределение; Z - распределение по закону Зипфа

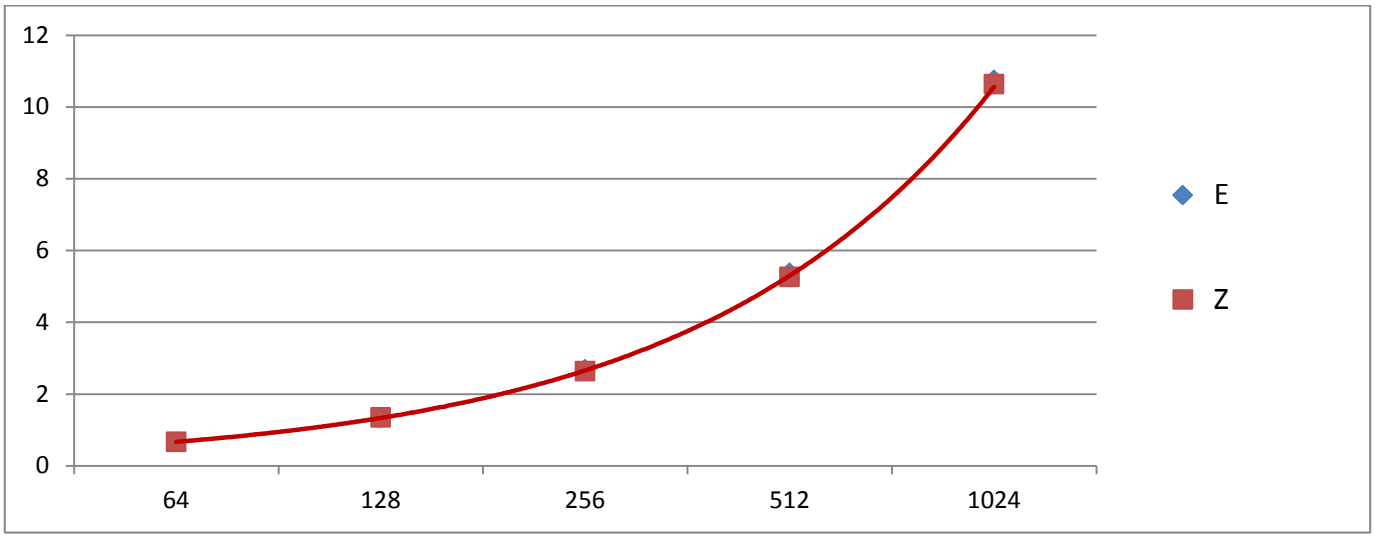


- Так как, в данном случае, горизонтальная ось имеет логарифмический вид, а функция Y(x) – логарифмическая, то визуализация аппроксимирующей функции имеет линейный вид.

3. Метод Find

Неудачный поиск

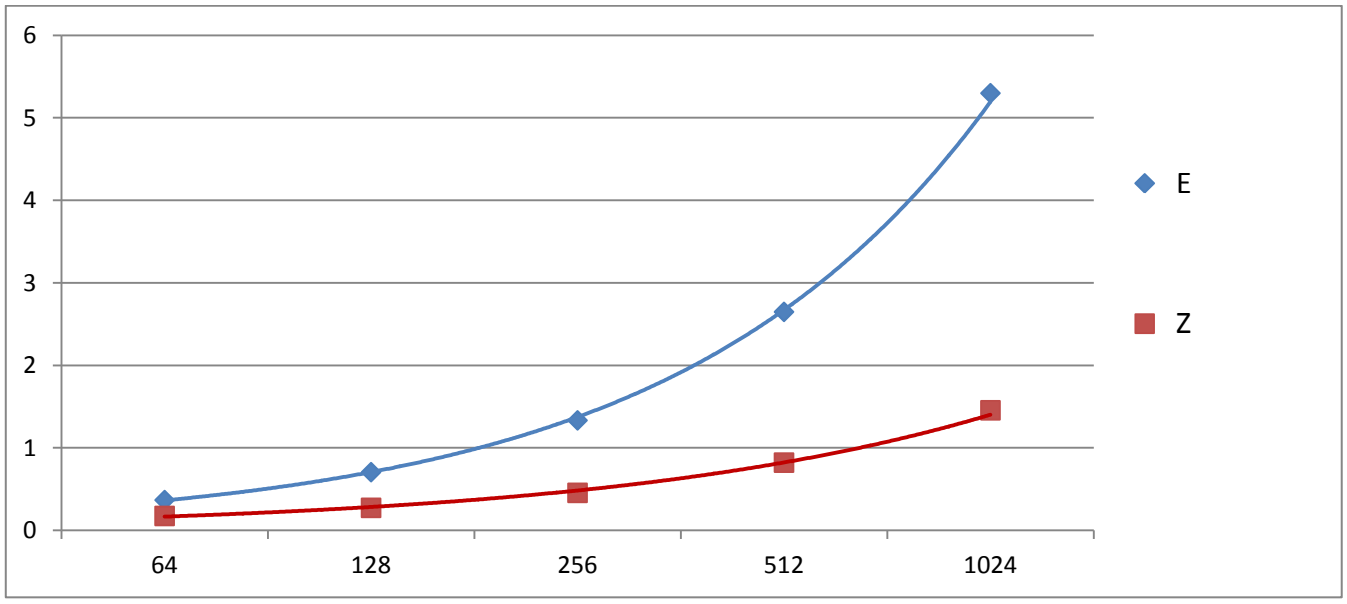
E – равномерное распределение; Z - распределение по закону Зипфа



- Так как, в данном случае, горизонтальная ось имеет логарифмический вид, а функция Y(x) – линейная, то визуализация аппроксимирующей функции имеет экспоненциальный вид.

Удачный поиск

E – равномерное распределение; Z - распределение по закону Зипфа



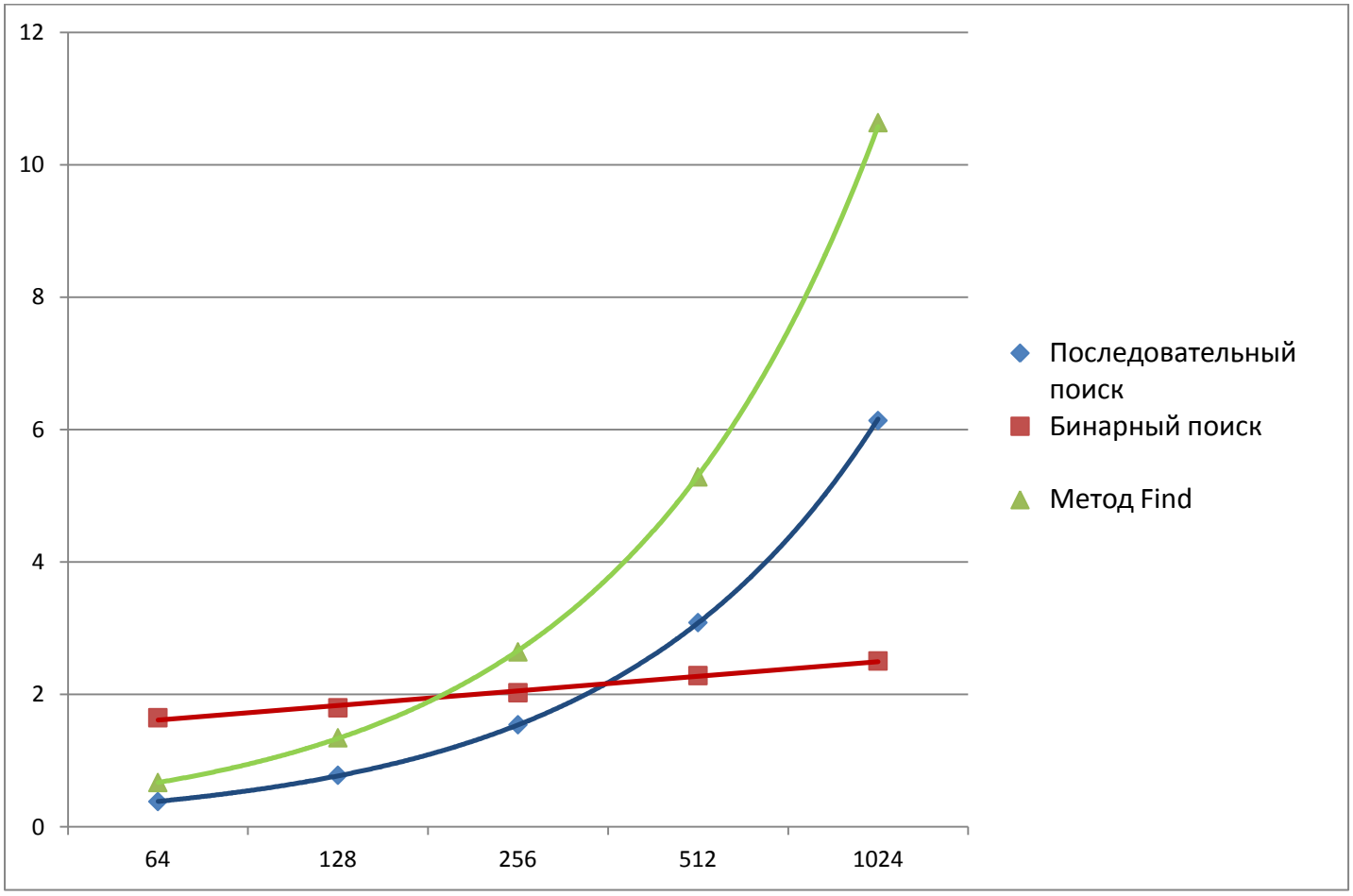
- Так как, в данном случае, горизонтальная ось имеет логарифмический вид, а функция Y(x) – линейная, то визуализация аппроксимирующей функции имеет экспоненциальный вид.

Сравнительные графики

Неудачный поиск:

В данном случае достаточно поострить 1 график от каждого типа поиска, как так, для всех рассматриваемых типов время, затраченное на поиск, в случае неудачного исхода, не зависит от типа распределения вероятности.

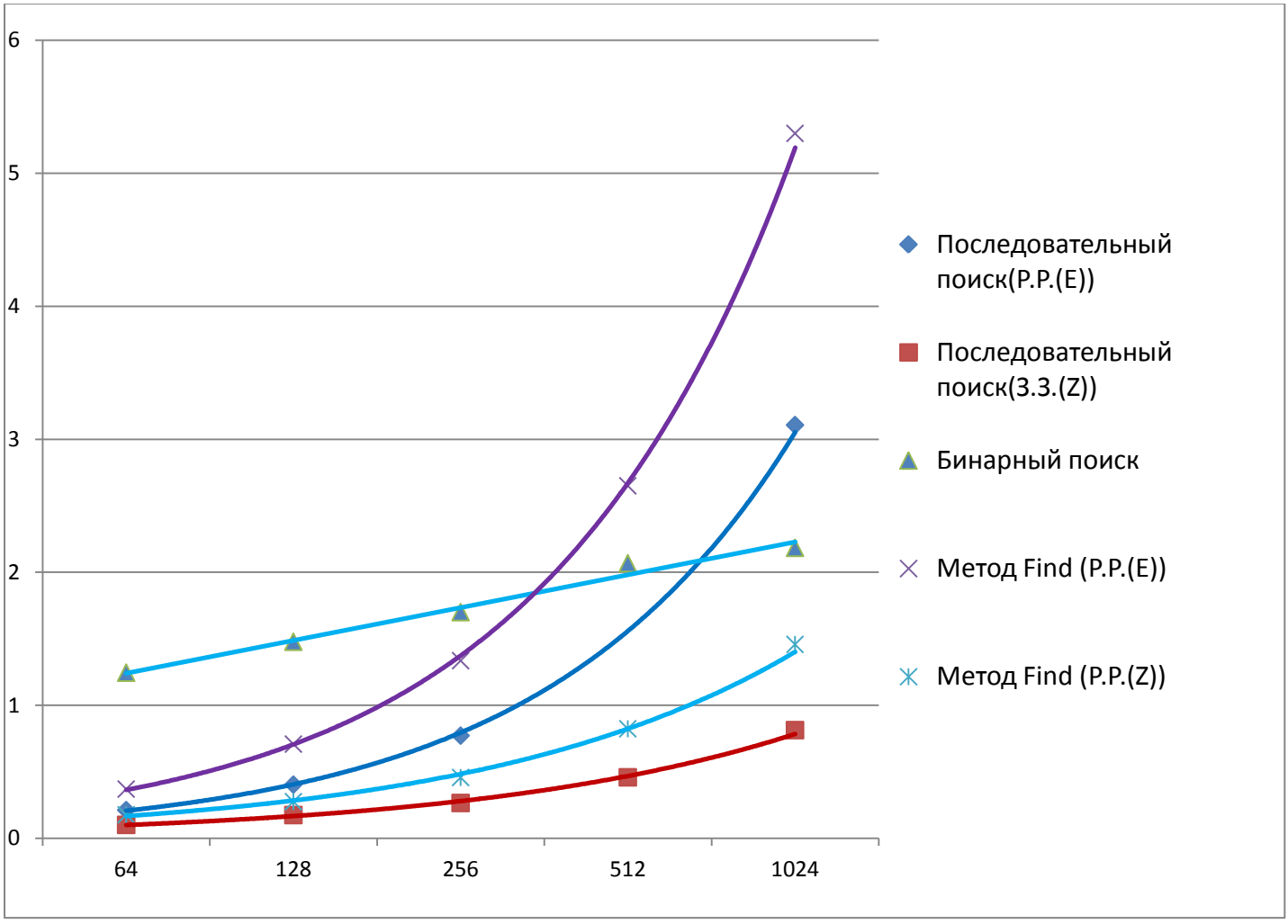
	Последовательный поиск	Бинарный поиск	Метод Find
64	0,382117	1,647933	0,669767
128	0,779167	1,797817	1,345317
256	1,542333	2,025667	2,6426
512	3,084683	2,2844	5,285267
1024	6,136183	2,504883	10,63498



Удачный поиск:

В данном случае необходимо совместить на одной плоскости по два графика зависимости для разных распределений вероятностей Последовательного поиска и Метода Find, так как данные зависимости имеют разные коэффициенты и 1 график для Бинарного поиска, так как и в данном случае распределение вероятности не важно.

	Последовательный поиск(P.P.(E))	Последовательный поиск(3.3.(Z))	Бинарный поиск	Метод Find (P.P.(E))	Метод Find (P.P.(Z))
64	0,2112	0,0999	1,244433333	0,369066667	0,176233333
128	0,402466667	0,173633333	1,4766	0,707166667	0,274566667
256	0,770933333	0,264566667	1,6996	1,334466667	0,456233333
512		0,4574	2,067133333	2,649366667	0,8224
1024	3,1064	0,812233333	2,182883333	5,299	1,456833333



Аппроксимирующие функции:

Последовательный поиск:

1. *Неудачный поиск*
   $t \approx 0.006*N$

2. *Удачный поиск (Равномерное распределение вероятности)*
   $t \approx 0.0031 * N$

3. *Удачный поиск (Распределение вероятности по закону Зипфа)*
   $t \approx 0.0011*N$

Бинарный поиск:

1. *Неудачный поиск*
   $t \approx 0.843* Log(N)$

2. *Удачный поиск*
   $t \approx 0.71 * Log(N)$

Метод Find:

1. *Неудачный поиск*
   $t \approx 0.0104 *N$

2. *Удачный поиск (Равномерное распределение вероятности)*
   $t \approx 0.0052* N$

3. *Удачный поиск (Распределение вероятности по закону Зипфа)*
   $t \approx 0.0017*N$

Вывод:

В процессе выполнения работы были выявлены следующие особенности исследуемых алгоритмов поиска:

Метод последовательного поиска:

Легкий для программирования и интуитивно понятный алгоритм, с случае неудачного поиска нечувствителен к типу распределения вероятности, а так же в данном случае является наиболее быстродейственным для массивом ключей размером приблизительно до 400 элементов (в зависимости от размера массива ключе й от 64 до ≈400 выигрывает изначально относительно Бинарного поиска ≈в 3 раза, относительно Метода Find чуть менее чем в 2, по мере увеличения количества ключей преимущество сокращается), далее начинает уступать по времени бинарному поиску, на максимально больших массивах ключей (в нашем случае 1024) приблизительно в 2,5 раза, но продолжает выигрывать относительно стандартного метода Find.

В ситуации, когда поиск является удачным, данный алгоритм становиться чувствительным к распределению вероятностей, и в случае с распределением вероятности по закону Зипфа, так же является наиболее быстродейственным, среди рассматриваемых вариантов(на малых массивах ключей относительно Бинарного поиска выигрывает приблизительно в 13 раз, на максимальных в нашем случае почти в 3), но с возрастанием количества элементов всё же проиграет Бинарному поиску(на ≈5500 ключей). Если же распределение равномерное, то, выигрывая вначале, поравняется с методом бинарного поиска на массивах с ≈700 ключей.

Так же стоит отметить линейность зависимости t от N, что облегчает поверхностную, сравнительную оценку данного алгоритма.

Метод Бинарного поиска:

Преимуществом данного алгоритма является нечувствительность к типу распределения вероятностей, как в случае с удачным поиском, так и с неудачным.

Данный метод достаточно прост, но имеет некоторые трудные особенности при программирование, например, в случае с массивом с нечетным количеством ключей.

Является менее быстродейственным для массивов ключей с небольшим количеством элементов, нежели метод последовательного поиска, но в свою очередь на большом количестве элементов имеет большое преимущество. В случае с неудачным поиском становится приоритетным при массивах ключей $\geq\approx 400$, при удачном поиске по массиву ключей с равномерным распределением $\geq\approx 700$, с распределением вероятностей по закону Зипфа $\geq\approx 5500$.

Стандартный метод Find(Последовательная сортировка):

Так, как данный метод является, по сути Последовательным поиском, то отличительные особенности у него такие же, на графиках видно, что данный метод имеет тот же характер зависимости. Проигрыш в быстродействии, думаю, стоит связать с организацией данного метода в конкретном языке, которая, является не очень тривиальной.

Таким образом, считаю, что следует использовать Последовательный поиск (или Метод Find) случае, когда мы работаем с небольшими массивами ключей и использовать Бинарный поиск при работе с большими.