**СПб НИУ ИТМО**

**Кафедра ИПМ**

Комбинаторные алгоритмы

Лабораторная работа №2

Тема: «Поиск»

***Вариант 5***

- бинарный поиск

- поиск по бинарному дереву

Выполнил:

Студент 1-го курса

Группы № 1125

Журавлев Виталий

Санкт - Петербург

2013 г.

**Описание:**

*Бинарный поиск:*

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.

*Двоичное дерево поиска:*

Это двоичное дерево, для которого выполняются следующие дополнительные условия:

* Оба поддерева — левое и правое, являются двоичными деревьями поиска.
* У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.
* В то время, как у всех узлов правого поддерева того же узла X значения ключей данных не меньше, нежели значение ключа данных узла X.

Поиск осуществляется с помощью ключа, сравниваемый со значением вершины. Если же ключ меньше вершины, ключ сравнивается с левым деревом, если же наоборот, то с правым и т.д. пока ключ не будет равен определенной вершине или же не закончатся вершины дерева.

**Текст программы:**

class Program

{

static Stopwatch stopwatch = new Stopwatch();

public static List<double> GetKeys(string filename)

{

string[] NumArray = File.ReadAllText(filename).Split(new[] { "\n", "\t" }, StringSplitOptions.RemoveEmptyEntries);

List<double> ResultList = new List<double>();

for (int i = 1; i <= double.Parse(NumArray[0]); i++)

{

ResultList.Add(double.Parse(NumArray[i]));

}

return ResultList;

}

public static List<double> GetData(string filename)

{

string[] NumArray = File.ReadAllText(filename).Split(new[] { "\n", "\t" }, StringSplitOptions.RemoveEmptyEntries);

List<double> ResultList = new List<double>();

for (int i = int.Parse(NumArray[0]) + 1; i < NumArray.Length; i++)

{

ResultList.Add(double.Parse(NumArray[i]));

}

ResultList.Sort();

return ResultList;

}

public static Tree GetTree(List<double> Data)

{

Tree tree = new Tree();

List<double> LeftList;

List<double> RightList;

int mid = (int)Math.Floor((decimal)(Data.Count / 2));

if (Data.Count >= 2)

{

tree.Add(Data[mid]);

LeftList = Data.GetRange(0, mid);

RightList = Data.GetRange(mid + 1, Data.Count - mid - 1);

tree.left = GetTree(LeftList);

tree.right = GetTree(RightList);

}

else if (Data.Count == 1) tree.Add(Data[0]);

return tree;

}

public static string GetResult(string filename)

{

string Elements = File.ReadAllText(filename).Split(new[] { "\n" }, StringSplitOptions.RemoveEmptyEntries)[0];

string Variant = filename[filename.Length - 1].ToString();

string Distribution = filename[filename.Length - 3].ToString();

string Lucky = filename[9].ToString();

return Variant + "\t" + Elements + "\t" + Lucky + "\t" + Distribution;

}

public static double BinarySearch(List<double> DataList, List<double> KeysList)

{

stopwatch.Restart();

foreach (double key in KeysList)

{

int FirstElement = 0;

int LastElement = DataList.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 (key == DataList[i]) break;

else if (key > DataList[i]) FirstElement = i + 1;

else LastElement = i; }

}

stopwatch.Stop();

return stopwatch.Elapsed.TotalMilliseconds;

}

public static double BinaryTreeSearch(List<double> KeysList, Tree tree)

{

stopwatch.Restart();

foreach (double key in KeysList)

{

tree.Search(key);

}

stopwatch.Stop();

return stopwatch.Elapsed.TotalMilliseconds;

}

static void Main(string[] args)

{ string[] FileNames = Directory.GetFiles(@"E:\Поиск");

File.WriteAllText(@"E:\Поиск.xls", "Search\t\t\t\tBinary\tBinTree\nVariant\tElements\tLucky\tDistribution\tTime\tTime\n");

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

{

List<double> Keys = GetKeys(FileNames[i]);

List<double> Data = GetData(FileNames[i]);

Tree tree = GetTree(Data);

double bintime = BinarySearch(Data, Keys);

double treetime = BinaryTreeSearch(Keys, tree);

File.AppendAllText(@"E:\Поиск.xls", GetResult(FileNames[i]) + "\t" + bintime.ToString() + "\t" + treetime.ToString() + " \n");

}

Console.WriteLine("Searched success!");

Console.ReadKey();

}

}

class Tree

{

private double value;

public Tree left;

public Tree right;

public void Add(double value)

{

if (this.value == 0) this.value = value;

}

public void Search(double value)

{

if (this.value == value) return;

else if (this.value > value)

if (left != null) this.left.Search(value);

else return;

else if (right != null) this.right.Search(value);

else return;

} }

**Результаты замеров:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Search |  | | | Binary | BinTree |
| Variant | Elements | Lucky | Distribution | Time | Time |
| 1 | 64 | N | E | 0,1642 | 0,2097 |
| 2 | 64 | N | E | 0,1369 | 0,0334 |
| 3 | 64 | N | E | 0,1477 | 0,0392 |
| 4 | 64 | N | E | 0,2092 | 0,0513 |
| 1 | 64 | N | Z | 0,1427 | 0,037 |
| 2 | 64 | N | Z | 0,1419 | 0,0441 |
| 3 | 64 | N | Z | 0,1396 | 0,0383 |
| 4 | 64 | N | Z | 0,1419 | 0,0365 |
| 1 | 128 | N | E | 0,2605 | 0,0611 |
| 2 | 128 | N | E | 0,2802 | 0,0589 |
| 3 | 128 | N | E | 0,2605 | 0,0611 |
| 4 | 128 | N | E | 0,3377 | 0,0798 |
| 1 | 128 | N | Z | 0,2583 | 0,0606 |
| 2 | 128 | N | Z | 0,2641 | 0,062 |
| 3 | 128 | N | Z | 0,2784 | 0,058 |
| 4 | 128 | N | Z | 0,2601 | 0,0593 |
| 1 | 256 | N | E | 0,5296 | 0,132 |
| 2 | 256 | N | E | 0,5194 | 0,1227 |
| 3 | 256 | N | E | 0,518 | 0,1213 |
| 4 | 256 | N | E | 0,5858 | 0,124 |
| 1 | 256 | N | Z | 0,4761 | 0,1124 |
| 2 | 256 | N | Z | 0,4752 | 0,1142 |
| 3 | 256 | N | Z | 0,4774 | 0,1155 |
| 4 | 256 | N | Z | 0,4801 | 0,112 |
| 1 | 512 | N | E | 1,3119 | 0,273 |
| 2 | 512 | N | E | 0,9852 | 0,236 |
| 3 | 512 | N | E | 0,9803 | 0,2668 |
| 4 | 512 | N | E | 1,0053 | 0,2275 |
| 1 | 512 | N | Z | 1,17 | 0,2396 |
| 2 | 512 | N | Z | 0,9495 | 0,2195 |
| 3 | 512 | N | Z | 0,9544 | 0,2949 |
| 4 | 512 | N | Z | 0,9446 | 0,219 |
| 1 | 1024 | N | E | 1,9843 | 0,5863 |
| 2 | 1024 | N | E | 1,9009 | 0,4506 |
| 3 | 1024 | N | E | 1,9549 | 0,5756 |
| 4 | 1024 | N | E | 1,8937 | 0,4564 |
| 1 | 1024 | N | Z | 2,1606 | 0,5385 |
| 2 | 1024 | N | Z | 1,9629 | 0,4453 |
| 3 | 1024 | N | Z | 1,9232 | 0,4319 |
| 4 | 1024 | N | Z | 1,9616 | 0,4417 |
| 1 | 2048 | N | E | 3,8415 | 0,8795 |
| 2 | 2048 | N | E | 3,9647 | 0,9794 |
| 3 | 2048 | N | E | 3,8741 | 0,8955 |
| 4 | 2048 | N | E | 3,8817 | 0,95 |
| 1 | 2048 | N | Z | 3,846 | 0,8772 |
| 2 | 2048 | N | Z | 3,9593 | 0,8871 |
| 3 | 2048 | N | Z | 3,9165 | 0,8813 |
| 4 | 2048 | N | Z | 3,8518 | 0,8799 |
| 1 | 64 | Y | E | 0,0522 | 0,0116 |
| 2 | 64 | Y | E | 0,0526 | 0,0116 |
| 3 | 64 | Y | E | 0,0517 | 0,012 |
| 4 | 64 | Y | E | 0,0517 | 0,0116 |
| 1 | 64 | Y | Z | 0,0597 | 0,0133 |
| 2 | 64 | Y | Z | 0,0736 | 0,024 |
| 3 | 64 | Y | Z | 0,0593 | 0,0133 |
| 4 | 64 | Y | Z | 0,0584 | 0,0129 |
| 1 | 128 | Y | E | 0,1204 | 0,0285 |
| 2 | 128 | Y | E | 0,128 | 0,0303 |
| 3 | 128 | Y | E | 0,1155 | 0,0267 |
| 4 | 128 | Y | E | 0,1374 | 0,0272 |
| 1 | 128 | Y | Z | 0,1343 | 0,0307 |
| 2 | 128 | Y | Z | 0,1338 | 0,0303 |
| 3 | 128 | Y | Z | 0,1334 | 0,0307 |
| 4 | 128 | Y | Z | 0,1343 | 0,0307 |
| 1 | 256 | Y | E | 0,2704 | 0,0709 |
| 2 | 256 | Y | E | 0,2605 | 0,0606 |
| 3 | 256 | Y | E | 0,2793 | 0,0606 |
| 4 | 256 | Y | E | 0,3382 | 0,0624 |
| 1 | 256 | Y | Z | 0,3012 | 0,0691 |
| 2 | 256 | Y | Z | 0,3123 | 0,0696 |
| 3 | 256 | Y | Z | 0,3168 | 0,0691 |
| 4 | 256 | Y | Z | 0,3177 | 0,0687 |
| 1 | 512 | Y | E | 0,6006 | 0,1477 |
| 2 | 512 | Y | E | 0,6215 | 0,1338 |
| 3 | 512 | Y | E | 0,5809 | 0,1454 |
| 4 | 512 | Y | E | 0,5979 | 0,136 |
| 1 | 512 | Y | Z | 0,6921 | 0,1584 |
| 2 | 512 | Y | Z | 0,6947 | 0,1597 |
| 3 | 512 | Y | Z | 0,6916 | 0,1588 |
| 4 | 512 | Y | Z | 0,6903 | 0,1584 |
| 1 | 1024 | Y | E | 1,3257 | 0,3114 |
| 2 | 1024 | Y | E | 1,34 | 0,2998 |
| 3 | 1024 | Y | E | 1,3003 | 0,3016 |
| 4 | 1024 | Y | E | 1,3203 | 0,3029 |
| 1 | 1024 | Y | Z | 1,5435 | 0,3498 |
| 2 | 1024 | Y | Z | 1,5292 | 0,3676 |
| 3 | 1024 | Y | Z | 1,5278 | 0,3694 |
| 4 | 1024 | Y | Z | 1,5426 | 0,3511 |
| 1 | 2048 | Y | E | 2,8915 | 0,6863 |
| 2 | 2048 | Y | E | 2,9049 | 0,6894 |
| 3 | 2048 | Y | E | 2,9045 | 0,6876 |
| 4 | 2048 | Y | E | 2,8982 | 0,6693 |
| 1 | 2048 | Y | Z | 3,2976 | 0,7358 |
| 2 | 2048 | Y | Z | 3,4881 | 0,9125 |
| 3 | 2048 | Y | Z | 3,2512 | 0,7585 |
| 4 | 2048 | Y | Z | 3,2547 | 0,7657 |

**Графики:**

*N – неудачный поиск, Y – неудачный поиск;*

*E – равномерное распределение, Z - распределение по закону Зипфа.*

*Бинарный поиск:*

*Поиск по бинарному дереву:*

Аппроксимирующие функции:

1. Бинарный поиск:
   1. Неудачный поиск

t ≈ 0.843\* Log(N)

* 1. Удачный поиск

t ≈ 0.71 \* Log(N)

1. Поиск по бинарному дереву:
   1. Неудачный поиск

t ≈ 0.23 \* Log(N)

* 1. Удачный поиск

t ≈ 0.19 \* Log(N)

**Вывод:**

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

* *Бинарный поиск:*

*Преимуществом* данного алгоритма является нечувствительность к типу распределения вероятностей, как в случае с удачным поиском, так и с неудачным. Данный поиск имеет широкое распространение в информатике применительно к поиску в структурах данных.

Так же он имеет и *недостатки*, например:

* в случае с массивом с нечетным количеством ключей.
* иногда требуется, чтобы, если  в цепочке существует в нескольких экземплярах,

находило не любой, а обязательно первый (или последний, как в данном случае).

* не будет работать на пустом массиве (он просто не выполнится)

Данный метод достаточно прост, но имеет некоторые трудные особенности при программировании, поэтому у неопытных пользователей могут возникнуть трудности при его написании, но его универсальность играет свою роль.

* *Поиск по бинарному дереву:*

Основным *преимуществом* двоичного дерева поиска перед другими структурами данных является возможная высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки. Двоичное дерево поиска применяется для построения более абстрактных структур, таких как множества, мультимножества, ассоциативные массивы.

*Недостатком* же является очень трудная реализация построения дерева, хотя при этом сам метод поиска достаточно прост, не считая рекурсивности. Так же могут возникнуть проблемы при разбивании оставшихся элементов (1-го или 2-х) на деревья.

Данный метод несколько схож с методом бинарного поиска, но при этом работает значительно быстрее. Но это не считая затрат времени на построение этого самого дерева.

В целом оба метода показали неплохие результаты. Время выполнения у них при неудачном поиске самое большое, причем практически независимо от распределения. При выполнении же удачного поиска с распределением по закону Зипфра поиск выполняется быстрее, чем с обычным.

Графики показывают примерно одинаковые отношения количества ключей от удачности поиска и метода их распределения и имеют аппроксимирующие функции

t ≈ О \* Log(N), однако различаются по скорости выполнения ≈ в 4 раза.

Поэтому по скорости предпочтительней использовать метод бинарного дерева поиска. Если необходим легкий, простой в структуре и универсальный код, то подойдет метод бинарного поиска.