САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ МЕХАНИКИ И ОПТИКИ

**Лабораторная работа**

по комбинаторным алгоритмам на тему:

*«Поиск»*

Выполнил:

студент группы 1125

Припадчев Артём

2013

**Вариант 9**

- Интерполяционный поиск;

- Бинарный поиск;

- Поиск с помощью стандартного алгоритма в стандартном контейнере.

*1) Интерполяционный поиск*

Поиск проходит подобно двоичному поиску, но вместо деления области на две примерно равные части, интерполирующий поиск производит оценку деления новой области поиска по расстоянию между ключом и текущим значением элемента. Если известно, что x лежит между a_l и a_r, то следующая проверка выполняется примерно на расстоянии \frac{x - a_l}{a_r - a_l} \cdot (r - l) от l.

*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 значение 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.

*3) Стандартный метод Find класса List<T>*

По сути является видоизмененным алгоритмом последовательного поиска по неупорядоченному списку.

**Реализация методов сортировки**

class Find

{

public static int InterpolationSearch(List<int> Figures, List<int> RequestList)

{

foreach (var Request in RequestList)

{

int left = 0;

int right = Figures.Count() - 1;

int i;

while (Figures[left] <= Request && Request <= Figures[right])

{

i = left + ((Request - Figures[left]) \* (right - left)) / (Figures[right] - Figures[left]);

if (Request < Figures[i])

{

right = i - 1;

continue;

}

if (Request > Figures[i])

{

left = i + 1;

continue;

}

if (Request == Figures[i])

break;

}

}

return 0;

}

public static int BinarySearch(List<int> Figures, List<int> RequestList)

{

foreach (var Request in RequestList)

{

int left = 0;

int right = Figures.Count() - 1;

int i;

while (left < right)

{

i = (left + right + 1) / 2;

if (Request < Figures[i])

{

right = i - 1;

continue;

}

if (Request > Figures[i])

{

left = i + 1;

continue;

}

if (Request == Figures[i])

break;

}

}

return 0;

}

public static int InsertSearch(List<int> Figures, List<int> RequestList)

{

foreach (var Request in RequestList)

{

Figures.Find(delegate(int Figure) { return Request == Figure; });

}

return 0;

}

**Результаты замеров времени выполнения (время в тиках процессора)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Поиск неудачен (распределение E) | | | | | | |
|  | **64** | **128** | **256** | **512** | **1024** | **2048** |
| Интерполяционный(Е) | 69 | 116 | 202 | 334 | 628 | 1013 |
|  | 69 | 116 | 202 | 334 | 587 | 1112 |
|  | 70 | 116 | 201 | 375 | 608 | 992 |
|  | **69** | **116** | **202** | **348** | **608** | **1039** |
| Бинарный(Е) | 56 | 114 | 227 | 453 | 900 | 1782 |
|  | 57 | 113 | 226 | 450 | 897 | 1896 |
|  | 56 | 110 | 227 | 447 | 914 | 1789 |
|  | **56** | **112** | **227** | **450** | **904** | **1822** |
| Встроенный(Е) | 8193 | 13449 | 27021 | 52235 | 104423 | 208998 |
|  | 6526 | 13221 | 27862 | 52228 | 104446 | 214295 |
|  | 6909 | 13295 | 26511 | 24827 | 106402 | 218492 |
|  | **7209** | **13322** | **27131** | **43097** | **105090** | **213928** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Поиск неудачен (распределение Z) | | | | | | |
|  | **64** | **128** | **256** | **512** | **1024** | **2048** |
| Интерполяционный(Z) | 82 | 161 | 307 | 585 | 1203 | 2157 |
|  | 86 | 153 | 307 | 585 | 1116 | 2129 |
|  | 83 | 153 | 313 | 602 | 1139 | 2175 |
|  | **84** | **156** | **309** | **591** | **1153** | **2154** |
| Бинарный(Z) | 59 | 119 | 336 | 455 | 894 | 1898 |
|  | 62 | 119 | 224 | 454 | 895 | 1775 |
|  | 59 | 117 | 227 | 442 | 892 | 1800 |
|  | **60** | **118** | **262** | **450** | **894** | **1824** |
| Встроенный(Z) | 6771 | 13077 | 27945 | 52540 | 104503 | 214251 |
|  | 6776 | 13536 | 27393 | 52192 | 104653 | 208957 |
|  | 6852 | 13255 | 26203 | 52307 | 104542 | 209143 |
|  | **6800** | **13289** | **27180** | **52346** | **104566** | **210784** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Поиск удачен (распределение E) | | | | | | |
|  | **64** | **128** | **256** | **512** | **1024** | **2048** |
| Интерполяционный(Е) | 20 | 50 | 95 | 200 | 393 | 837 |
|  | 19 | 49 | 99 | 190 | 394 | 837 |
|  | 20 | 49 | 89 | 190 | 410 | 827 |
|  | **20** | **49** | **94** | **193** | **399** | **834** |
| Бинарный(Е) | 32 | 67 | 149 | 330 | 717 | 1692 |
|  | 31 | 67 | 150 | 329 | 721 | 1692 |
|  | 31 | 67 | 150 | 331 | 840 | 1600 |
|  | **31** | **67** | **150** | **330** | **759** | **1661** |
| Встроенный(Е) | 86 | 274 | 1151 | 4135 | 14290 | 53061 |
|  | 83 | 273 | 1075 | 3900 | 15052 | 53034 |
|  | 71 | 271 | 1059 | 4172 | 15208 | 61123 |
|  | **80** | **273** | **1095** | **4069** | **14850** | **55739** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Поиск удачен (распределение Z) | | | | | | |
|  | **64** | **128** | **256** | **512** | **1024** | **2048** |
| Интерполяционный(Z) | 38 | 93 | 223 | 476 | 1092 | 1915 |
|  | 42 | 92 | 214 | 448 | 977 | 2020 |
|  | 41 | 91 | 221 | 447 | 1038 | 2007 |
|  | **40** | **92** | **219** | **457** | **1036** | **1981** |
| Бинарный(Z) | 35 | 76 | 172 | 489 | 811 | 1803 |
|  | 37 | 75 | 168 | 379 | 812 | 1675 |
|  | 33 | 76 | 168 | 376 | 987 | 1705 |
|  | **35** | **76** | **169** | **415** | **870** | **1728** |
| Встроенный(Z) | 193 | 688 | 3013 | 12581 | 38624 | 93923 |
|  | 181 | 692 | 2897 | 11898 | 36963 | 93511 |
|  | 183 | 730 | 2831 | 12589 | 41108 | 93417 |
|  | **186** | **703** | **2914** | **12356** | **38898** | **93617** |

**Графики зависимостей**

1. Интерполяционный поиск

*Неудачный поиск*

*Удачный поиск*

2. Бинарный поиск

*Неудачный поиск*

*Удачный поиск*

3. Метод Find

*Неудачный поиск*

*Удачный поиск*

**Сравнительные графики зависимостей**

*Неудачный поиск*

*Удачный поиск*

**Аппроксимирующие функции**

*Интеполяционный поиск:*

*1. Неудачный поиск*

t ≈ 0.23\* *Log(Log(N)) (Равномерное распределение вероятности)*

*2. Неудачный поиск (Распределение вероятности по закону Зипфа)*

t ≈ 0.14\**Log(Log(N))*

*3. Удачный поиск (Равномерное распределение вероятности)*

t ≈ 0.23\* *Log(Log(N))*

*4. Удачный поиск (Распределение вероятности по закону Зипфа)*

t ≈ 0.135\**Log(Log(N))*

*Бинарный поиск:*

*1. Неудачный поиск*

*t* ≈ *0.81\* Log(N)*

*2. Удачный поиск*

*t* ≈ *0.76 \* Log(N)*

*Метод Find:*

*1. Неудачный поиск*

t ≈0.01 \*N

*2. Удачный поиск (Равномерное распределение вероятности)*

t ≈ 0.0052\* N

*3. Удачный поиск (Распределение вероятности по закону Зипфа)*

t ≈ 0.0025\*N

**Вывод:**

В процессе выполнения работы были выявлены следующие особенности алгоритмов:

*Интерполяционный:* довольно быстродейственный алгоритм, причем в то же время легкий для программирования. Время выполнения не практически не зависит от того, будет ли поиск удачным или же нет. Однако зависит от распределения: при распределении по закону Зипфа работает в два раза медленней, чем при относительном.

*Бинарный:* нечувствителен к типу распределения вероятностей при удачном, и при неудачном поиске. Сам алгоритм довольно прост в реализации. Работает быстрее интерполяционного поиска, но только в случае распределения по закону Зипфа.

*Стандартный метод Find:* практически нечувствителен к типу распределения вероятностей. При удачном поиске время сокращается значительно, что и логично, т.к. этот метод похож на последовательный поиск. Однако время выполнения слишком велико.

Таким образом, в случае если нам известно примерное распределение вероятностей, и оно подчиняется закону Зипфа, то лучше использовать бинарный поиск. Если же распределение нам не известно, или же оно равномерное – интеполяционный.