A photograph of a wooden desk with a laptop and an open notebook. A black pen with 'PRECISE' and 'W5' written on it lies on the notebook. The background is slightly blurred, showing a window with blinds. A semi-transparent dark circle is overlaid on the right side of the image.

Разработка методов и алгоритмов поисковой системы

Магистрант: Жұмадін Д.Е.
Научный руководитель: Синчев Б.К.

Содержание

- ▶ Актуальность темы диссертации
- ▶ Цели и задачи диссертации
- ▶ Определение объекта и предмета исследования
- ▶ Научная новизна
- ▶ Методология исследования
- ▶ Заключение

АКТУАЛЬНОСТЬ ТЕМЫ

**Повышения эффективности
информационного поиска**

**Грамотное применение методов и
подходов, применяемые для
решение информационного
поиска**

**Колоссальный
рост объема
данных**



Цели и задачи

Цель

Целью диссертационного исследования является разработка основных методов и программных средств поиска информации для ИПС

Задачи

- исследование основных подходов и методов поиска
- изучение основных характеристик и особенностей ИПС
- изучение различных методов поиска в ИПС

Определение объекта и предмета исследования

- ▶ **Объектом исследования** в данной работе являются основные методы поиска информации в ИПС
- ▶ **Предметом исследования** являются методы, применяемые для решения задач информационного поиска



Научная новизна

Предложен метод информационного-поиска, который позволяет существенно повысить эффективность и точность информационного поиска, а также удобочитаемость результатов поиска.



Базовые отличия СВД и ИПС

	СВД	ИПС
соответствие данных поисковому запросу	точное	частичное
классификация документов	детерминированная	вероятностная
язык запросов	искусственный	естественный
критерии выборки документов	булева функция релевантности	вероятностная функция релевантности
устойчивость к ошибкам в данных и запросах	неустойчивы	устойчивы

Существующие информационные системы, работающие с электронными тестовыми документами, можно условно разделить на две категории: информационно-поисковые системы (information retrieval systems), и системы выборки данных (data retrieval systems).

Методы поиска данных и Постановка задачи

Все алгоритмы поиска делятся на

- поиск в неупорядоченном множестве данных;
- поиск в упорядоченном множестве данных.

Упорядоченность – наличие отсортированного ключевого поля.

Цель сортировки – облегчить последующий поиск элементов в отсортированном множестве при обработке данных.

Задачу поиска можно сформулировать так: найти один или несколько элементов в множестве, причем искомые элементы должны обладать определенным свойством.

Анализ методов поиска данных

Последовательный поиск

Последовательный поиск

Начать с начала и продолжать, пока не будет найден искомый ключ, затем остановиться - это простейший из алгоритмов поиска. Этот алгоритм не очень эффективен, однако он работает на произвольном списке.

Процедура `SequentialSearch` выполняет последовательный поиск элемента `z` в массиве `A[1..n]`.

```
SequentialSearch(A, z, n)
```

```
(1) for i ← 1 to n  
(2) do if z = A[i].key  
(3) then return i  
(4) return 0
```

```
(4) return 0
```

Последовательный поиск

Анализ наихудшего случая.

У алгоритма последовательного поиска два наихудших случая.

- 1. В первом случае целевой элемент стоит в списке последним.**
 - 2. Во втором его вовсе нет в списке.**
- В обоих случаях алгоритм выполнит n сравнений.**

Анализ среднего случая.

Целевое значение может занимать одно из n возможных положений. Будем предполагать, что все эти положения равновероятны.

Анализ методов поиска данных

Логарифмический
(бинарный) поиск

Пусть упорядоченный массив $x(1:n)$ содержит, например, элементы 5, 7, 11, 18, 26, 32, 44, 57, 81, 90, 94, 97, 107, 116, 129, 147, 179 и пусть задан аргумент поиска key , равный, например, 129.

Идея алгоритма бинарного поиска такова:

- сравнить аргумент поиска key со значением среднего элемента $x(mid)$ массива x , где $mid = \lfloor n/2 \rfloor$, а $\lfloor c \rfloor$ – целая часть числа c ;
- если они равны, то поиск завершен, иначе, если $key < x(mid)$, выполнить аналогичным образом поиск в позициях массива x , предшествующих позиции mid , в противном случае, если $key \geq x(mid)$, выполнить аналогичным образом поиск в позициях массива x , следующих за позицией mid .

Логарифмический (бинарный или метод делением пополам) поиск

BinarySearch(A, z, n)

(1) $p \leftarrow 1$

(2) $r \leftarrow n$

(3) **while** $p \leq r$ **do**

(4) $q \leftarrow \lfloor (p+r)/2 \rfloor$

(5) **if** $A[q].key = z$

(6) **then return** q

(7) **else if** $A[q].key < z$

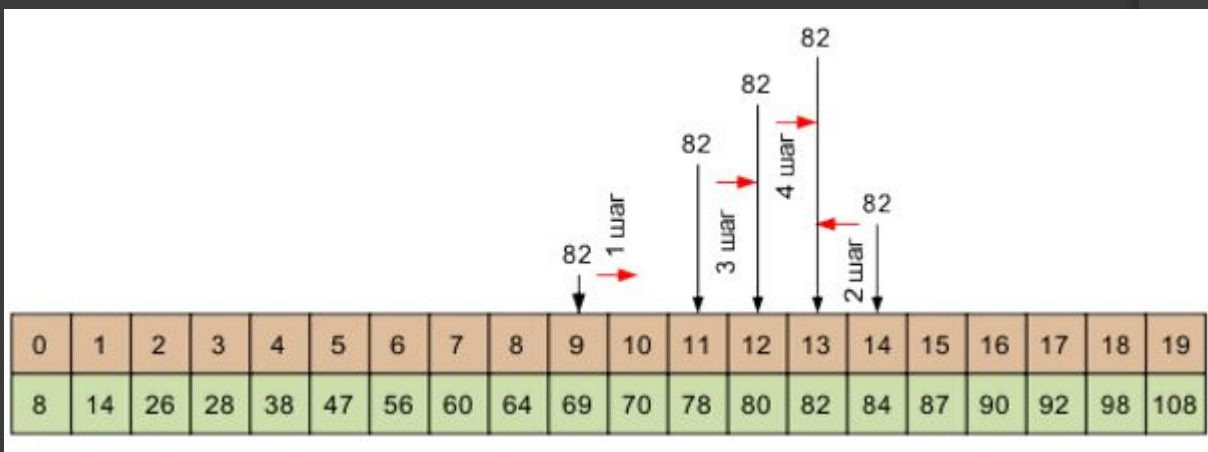
(8) **then** $p \leftarrow q+1$

(9) **else** $r \leftarrow q-1$

(10) **return** 0

Исключить из дальнейшего рассмотрения часть массива позволяет тот факт, что массив упорядочен.

Процедура BinarySearch выполняет бинарный поиск элемента z в отсортированном массиве $A[1..n]$.



Логарифмический (бинарный или метод делением пополам) поиск

Анализ наихудшего случая.

Поскольку алгоритм всякий раз делит список пополам, будем предполагать при анализе, что $n = 2^k - 1$ для некоторого k . Ясно, что на некотором проходе цикл имеет дело со списком из $2^j - 1$ элементов.

Анализ среднего случая.

Рассмотрим два случая. В первом случае целевое значение наверняка содержится в списке, а во втором его может там и не быть. В первом случае у целевого значения n возможных положений.

Результат выполнения

Последовательный поиск

```

c:\Projects\index-serial-search\Debug\index-serial-search.exe
0. k[0]= 8: r[0]=7
1. k[1]= 14: r[1]=3
2. k[2]= 26: r[2]=4
3. k[3]= 28: r[3]=2
4. k[4]= 38: r[4]=6
5. k[5]= 47: r[5]=5
6. k[6]= 56: r[6]=8
7. k[7]= 60: r[7]=9
8. k[8]= 64: r[8]=1
9. k[9]= 69: r[9]=12
10. k[10]= 70: r[10]=14
11. k[11]= 78: r[11]=10
12. k[12]= 80: r[12]=55
13. k[13]= 82: r[13]=34
14. k[14]= 84: r[14]=25
15. k[15]= 87: r[15]=19
16. k[16]= 90: r[16]=78
17. k[17]= 92: r[17]=90
18. k[18]= 98: r[18]=86
19. k[19]= 108: r[19]=100
Введите key: 87
15. key= 87. r[15]=19_
  
```

Логарифмический поиск

```

c:\Projects\binary-search\Debug\binary-search.exe
0. k[0]= 8: r[0]=1
1. k[1]= 14: r[1]=2
2. k[2]= 26: r[2]=3
3. k[3]= 28: r[3]=4
4. k[4]= 38: r[4]=5
5. k[5]= 47: r[5]=6
6. k[6]= 56: r[6]=7
7. k[7]= 60: r[7]=8
8. k[8]= 64: r[8]=9
9. k[9]= 69: r[9]=10
10. k[10]= 70: r[10]=11
11. k[11]= 78: r[11]=12
12. k[12]= 80: r[12]=13
13. k[13]= 82: r[13]=14
14. k[14]= 84: r[14]=15
15. k[15]= 87: r[15]=16
16. k[16]= 90: r[16]=17
17. k[17]= 92: r[17]=18
18. k[18]= 98: r[18]=19
19. k[19]= 108: r[19]=20
Введите key: 82
13. key= 82. r[13]=14_
  
```

```

T2: key= 87: L[T2]=19
Введите key: 87
T2: k[T2]= 87: r[T2]=100
  
```

```

T3: key= 82: L[T3]=14
Введите key: 82
T3: k[T3]= 82: r[T3]=100
  
```

Заключение

В результате выполнения работы сравним алгоритмы последовательного и бинарного поиска. Пусть файл, в котором выполняется поиск, отсортирован и содержит 1024 (210) элемента. В случае последовательного поиска наибольшее число итераций будет равно 1024, а бинарного – 11. То есть разница в два порядка.

Сравним теперь временные затраты на поиск в случае неотсортированного файла. При последовательном поиске максимальное число итераций, разумеется, сохраняется. Бинарный поиск неприменим. Выполним, однако, быструю сортировку файла.

В этом исследовании мы показали, что если файл неотсортирован и в процессе вычислений задача поиска в файле возникает сравнительно, то можно применять последовательный поиск, в противном случае более целесообразно прежде отсортировать файл и при вычислениях применять бинарный поиск.

***Спасибо за
внимание!***