деленный перечень функций, которые удовлетворяют установленные или предполагаемые потребности в соответствии с назначением ПО;

надежность функционирования - свойства ПО, обусловливающие способность ПО сохранять работоспособность и преобразовывать исходные данные в искомый результат в заданных условиях за установленный период времени;

удобство использования - свойства ПО, обеспечивающие пользователям необходимые условия для использования ПО;

рациональность - свойства ПО, характеризующиеся степенью соответствия используемых ресурсов среды функционирования уровню качества функционирования ПО при заданных условиях применения;

переносимость - свойства ПО, обусловливающие его приспособленность для переноса из одной среды функционирования в другую.

5. Выводы

Проанализовав в данной статье преимущества и недостатки этих двух современных технлогий PHP и ASP.NET по отношению к потребителю следует сделать вывод, что для автоматизации поставленной задачи необходимо использовать технологию PHP, так как фактор надежности, бесплатности, эффективности, безопасности и стабильной работы сайта построенного на данной технологии, важен для стабильной работы всей компании.

Литература

1. Котеров Д. В., Костарев А. Ф. PHP 5: В Подлиннике - СПб.: БХВ-Петербург, 2006 - 1120 стр.
2. Дуглас Дж. Рейли. Создание приложений Microsoft ASP. NET – СПб.: Русская Редакция, 2006 г. – 464 стр.

УДК 65.011.56

ВЫБОР МЕТОДА ПОИСКА В БАЗЕ ДАННЫХ

А.Н. Толстиков*
Контактный тел.: (057) 335-24-29

Е.П. Павленко
Кандидат технических наук, доцент*
Контактный тел.: (057) 702-14-51
E-mail:evg-pavl@mail.ru
*Кафедра информационных управляющих систем
Харьковский национальный университет радиоэлектроники
пр-т Ленина, 14, г. Харьков, Украина, 61166

Досліджені основні переваги і недоліки методів пошуку в базах даних. Була розглянута задача обліку бібліотечного фонду в бібліотеці на ОАО «Автрамат» і вибраний метод для її ефективного вирішення

Ключові слова: двійковий пошук, послідовний пошук, база даних

Исследованы основные преимущества и недостатки методов поиска в базах данных. Была рассмотрена задача учета библиотечного фонда в библиотеке на ОАО «Автрамат» и выбран метод для её эффективного решения

Ключевые слова: двоичный поиск, последовательный поиск, база данных

Advantages and disadvantages of methods of search in the bases of information were analyzed. The problem of account of library fund on enterprise «Avtramat» was considered and also was chosen a method for solving this problem

Keywords: binary search, one-at-time search, databas

1.Введение

Информационные системы с использованием концепций баз данных широко применяются в настоящее время, так как предприятиям необходимо обрабатывать большое количество разнообразной информации. Одна из составляющих информатизации – создание автоматизированных библиотечно–информационных систем (АБИС). Основная задача, которая ставится перед этими системами, – реализация максимально возможного числа библиотечных технологических процессов и операций, которые поддаются автоматизации. АБИС состоит из реляционной базы данных, программного обеспечения, которое взаимодействует с базой данных, и графических пользовательских интерфейсов.

Основу АБИС составляет база данных (БД). Существуют методы поиска информации в неупорядо-

ченном и упорядоченном массиве. Для нахождения информации в неупорядоченном массиве требуется последовательный поиск, а если данные уже отсортированы, можно применить двоичный поиск.

2. Постановка задачи

ОАО «Автрамат» – это предприятие по выпуску поршней, являющееся лидером в своей области в Украине. Для того чтобы поддерживать свой статус лидера необходимо быть обеспеченным современными технологиями и квалифицированными специалистами, которые помогут предприятию выйти на качественно новый уровень развития. Поэтому активно развивается библиотека, которая способствует развитию кадров предприятия. Одной из важных задач библиотеки является задача учета библиотечного фонда библиотеки. Необходимо эффективно управлять библиотечным фондом, номенклатура которого все время увеличивается. Сейчас неэкономично и сложно вести учетные документы библиотеки вручную. Поэтому было предложено разработать и внедрить АБИС, которая сможет эффективно, быстро и качественно решать поставленную задачу, а также автоматически формировать и сохранять выходные документы, такие, как «Книга суммарного учета библиотечного фонда», «Инвентарная книга» и др.

Рассмотрим методы поиска в базах данных для выбора эффективного способа поиска.

3. Сравнение двоичного и последовательного методов поиска данных в базах данных

Поиск – это нахождение какой-либо конкретной информации в большом объеме ранее собранных данных. Данные делятся на записи, и каждая запись имеет хотя бы один ключ. Ключ используется для того, чтобы отличить одну запись от другой. Записи, или элементы списка, идут в массиве последовательно и между ними нет промежутков. Целью поиска является нахождение всех записей подходящих к заданному ключу поиска.

Суть различных методов поиска сосредоточена в методах перебора, в стратегии поиска. Возникает ряд вопросов, ответы на которые зависят от структуры данных, в которой хранится множество элементов. Накладывая ограничения на структуру исходных данных, можно получить множество разнообразных стратегий поиска различной степени эффективности.

Последовательный поиск. Суть метода – последовательно перебираются все элементы массива, если на каком-то шаге цикла обнаруживается, что массив закончился или обнаруживается искомый элемент, то цикл заканчивается. Перед алгоритмом поиска стоит задача определения расположения ключа, потому он возвращает индекс записи, содержащей нужный ключ. Если ключевое значение не найдено, когда поиск завершился, то алгоритм поиска обычно возвращает значение индекса, которое превышает верхнюю границу массива.

У алгоритма последовательного поиска есть два недостатка или затруднительных случая. В первом случае целевой элемент стоит в списке последним. Во втором его вовсе нет в списке. В обоих случаях алгоритм выполнит n сравнений, где n – число элементов в списке.

Возможно проанализировать средний случай данного алгоритма.

Для алгоритмов поиска возможны два средних случая. В первом предполагается, что поиск всегда завершается успешно, во втором — что иногда целевое значение в списке отсутствует. Целевое значение, если оно содержится в списке, может занимать одно из N возможных положений: первое, второе и т.п. Предположим, что все эти положения равновероятны (вероятность встретить каждое 1/n) . Тогда, для каждого из N случаев число сравнений совпадает с номером искомого элемента. Получаем равенство

$$T(n)=\frac{1}{n}\sum_{i=1}^{n} i=\frac{n+1}{2}$$

Целевое значение может не содержаться в списке. В результате число возможностей возрастет до n + 1 (если же элемент отсутствует в списке. то его поиск требует n сравнений). Предположим, что все эти положения равновероятны

$$T(n)=\frac{1}{n+1}\left(\sum_{i=1}^{n} i+n\right)\approx\frac{n+2}{2}$$

Возможность отсутствия элемента в списке увеличивает сложность среднего случая лишь на 1/2, хотя эта величина пренебрежимо мала по сравнению с длиной списка, которая может быть очень велика.

Если вероятность встретить искомый элемент в списке равна

p_i , где $p_i \ge 0$ и $\sum_{i=1}^{n} p_i = 1$,

то средняя сложность поиска элемента равна $\sum_{i=1}^{n} i p_i$. Рассмотрим закон Зипфа: $p_i = \frac{c}{i}$ для $i = 1, 2, ..., n$ с нормирующей константой $\sum_{i=1}^{n} p_i = 1$

Допустим, что элементы массива A упорядочены согласно указанным частотам, тогда получим $c \approx \frac{1}{\ln n}$ и среднее время успешного поиска составит

$$\sum_{i=1}^{n} i p_i = \sum_{i=1}^{n} i \frac{c}{i} = nc \approx \frac{n}{\ln n}$$

что меньше (n+1)/2. Исходя из этого можно сказать, что даже простой последовательный поиск требует выбора разумной структуры данных множества для более эффективной работы алгоритма.

Сложность алгоритма линейна – O(n).

Двоичный (бинарный или метод делением пополам).

Суть метода в следующем: сначала производится проверка среднего элемента. Если его ключ равен ключу требуемого элемента, то поиск завершен. Если его ключ больше ключа требуемого элемента, то поиск необходимо продолжить в левой части массива. В противном случае поиск необходимо продолжить в правой части массива. Этот процесс повторяется до тех пор, пока не будет найден требуемый элемент или не будет больше элементов для проверки.

У алгоритма бинарного поиска есть определенный затруднительный случай. Так как алгоритм всякий раз делит список пополам, можно предположить, что $n = 2^k - 1$ для k. На некотором проходе цикл имеет дело со списком из $2^j - 1$ элементов. При j = 1 производится

последняя итерация цикла, когда размер оставшейся части становится равным единице. Можно сделать вывод, что количество проходов не превышает k при $n = 2^k - 1$. Таким образом, в наихудшем случае количество проходов – $k = \log_2(n+1)$.

Следовательно, алгоритм двоичного поиска имеет сложность по времени $O(\log_2 n)$, где n – число элементов отсортированного массива.

Метод двоичного поиска предназначен для сортированных элементов на смежной памяти фиксированного размера. Таким образом, если размерность вектора динамически изменяется, то экономия от использования двоичного поиска не компенсирует затрат на поддержание упорядоченного расположения $a1 < a2 < ... < an$.

4. Выводы – выбор метода поиска.

Выбор того или другого метода поиска зависит от определенных условий.

Если массив является неупорядоченным, то единственный алгоритм поиска, применимый в этом случае – последовательный. Этот метод применим для неупорядоченной информации, но также можно использовать его и на отсортированных данных. Когда данных немного, последовательный поиск работает достаточно быстро. Но если информация хранится на диске, поиск может занимать продолжительное время. Последовательный поиск достаточно прост и его легко запрограммировать, но время его работы прямо пропорционально количеству данных, которые нужно просмотреть; удвоение количества элементов приведет к удвоению времени на поиск, если искомого элемента в массиве нет. Это линейное соотношение (время выполнения является линейной функцией от размера данных), поэтому такой метод также называется линейным поиском. Потому эффективность этого метода очень низкая, так как для отыскания одного элемента в массиве размерности N в среднем нужно сделать N/2 сравнений. В лучшем случае он проверяет только один элемент, а в худшем — n.

Кроме поиска иногда нужно вставлять и удалять элементы. Однако массив плохо приспособлен для выполнения этих операций.

Приведенная ниже функция производит поиск в массиве символов известной длины, пока не найдет элемент с заданным ключом:

```
int sequential_search(char *items, int count, char kl)
{
 register int m;
 for(m=0; m < count; ++m)
 if(kl == items[m]) return m;
 return -1;
}
```

В данном случае items — это указатель на массив, который содержит информацию. Функция возвращает индекс подходящего элемента, если такой существует, либо -1, если такового нет.

Однако если данные уже отсортированы, можно применить двоичный поиск, который находит данные быстрее. Алгоритм двоичного поиска является систематизированной версией поиска слова в словаре. Для массива большого объема более эффективно было бы использовать двоичный поиск. Недостаток бинарного поиска заключается в необходимости последовательного хранения списка, что усложняет операции добавления и исключения элементов.

Рассмотрим функцию двоичного поиска для массивов символов в виде примера. Поиск возможно адаптировать для произвольных структур данных, если изменить фрагмент сравнения.

```
int binary_search(char *items, int count, char kl)
{
 int low, high, sred;
 low = 0; high = count-1;
 while(low <= high) {
   sred = (low+high)/2;
 if(kl < items[sred]) high = sred -1;
 else if(kl > items[sred]) low = sred +1;
 else return sred;
 }
 return -1;
}
```

Двоичный поиск отбрасывает за каждый шаг половину данных, поэтому количество шагов пропорционально тому, сколько раз мы можем поделить n на 2, пока у нас не останется один элемент. Без учета округления это число равно $\log_2 n$. Если у нас в массиве 1000 элементов, то линейный поиск займет до 1000 шагов, в то время как двоичный — только около 10; при миллионе элементов линейный поиск займет миллион шагов, а двоичный — 20. Очевидно, чем больше число элементов, тем больше преимущество двоичного поиска, который работает быстрее, чем линейный.

База данных для задачи учета библиотечного фонда представляет собой электронный каталог с множеством данных, для эффективного выбора информации из которой, лучше всего использовать двоичный метод поиска. Так как он работает быстрее, чем линейный и имеет преимущество в использовании для упорядоченных данных, представленных в виде набора электронных документов библиотеки, помещенных в БД согласно выведенному внутрисистемному формату согласно стандарта UNIMARC.

Литература

1. Макконнелл Дж. Основы современных алгоритмов. 2-е дополненное издание. Пер. с англ. под ред. С.К.Ландо, дополнение М.В.Ульянова. – Москва, Техносфера, 2004 – 368 с.
2. Кнут Д. Искусство программирования. Т. 3. Сортировка и поиск. – М.: Издательский дом «Вильямс», 2003 – 801 с.