Лабораторная работа №7
ИССЛЕДОВАНИЕ И ОЦЕНКА АЛГОРИТМОВ ПОИСКА
Краткая теория

Цель работы. Разработка программ, реализующих различные алгоритмы поиска, и оценка их временной и пространственной сложности.

Поиск. Основные определения и классы алгоритмов

Процедуры поиска широко используются в информационно-справочных системах (базах и банках данных), при разработке синтаксических анализаторов и компиляторов (поиск служебных слов, команд и т.п. в таблицах) и др. В простейшем случае считается, что исходными данными для этой процедуры являются массив (чисел, слов и т.д.) и некоторое значение, которое принято называть *аргументом поиска*.

Эталоны в таблице (массиве) могут иметь сложную структуру, но характеризуются неповторяющимися значениями некоторых *ключей*. Искомый *аргумент* сравнивается с каждым из этих ключей. Если они совпадают, то *аргумент* найден. Результат поиска может быть булевский (аргумент есть в таблице или нет) или числовой (номер ключа в таблице).

Наиболее сложным является случай, когда аргумента поиска нет в таблице. Суждение об этом можно сделать только по окончании просмотра всего массива.

Поскольку в процессе поиска участвуют только ключи, а информационная часть элементов не важна, в дальнейшем будем рассматривать только *алгоритмы поиска ключей*, значения которых обычно являются *целыми числами*.

В общем случае для выполнения этой операции могут использоваться различные ***условия***. В зависимости от способа их задания и организации поиска различают большое количество видов таких операций. Наибольшее распространение получили следующие ***виды поиска*** по простому условию:

a) совпадению,
b) близости снизу
c) близости сверху.

Первый вид предполагает нахождение элемента с заданным значением ключа *k*. Если такого ключа нет, то операция закончилась безуспешно.

При поиске ***по близости*** операция всегда заканчивается успешно. Если он выполняется снизу, то результатом будет элемент, ключ которого является ближайшим меньшим искомого. При поиске по близости сверху результат представляет собой элемент с ключом, ближайшим большим искомого.

Мы рассмотрим только алгоритмы поиска по совпадению, а по близости наиболее просто реализуются с помощью двоичных деревьев, которые будут изучаться позднее.

Общий алгоритм поиска по совпадению ключа можно представить так.

1. Ввести исходный массив (ключей).
2. Повторять
   ввести аргумент поиска и вывести результат
   пока не надоест.
3. Закончить.

Предположим в дальнейшем, что для окончания работы с

программой (когда искать надоело) необходимо ввести отрицательное число (несуществующий ключ).

Наиболее распространенными являются три алгоритма поиска:

1) Линейный;
2) Дихотомический:
3) Интерполирующий.

1. Алгоритм линейного поиска

В соответствии с этим алгоритмом эталонный массив просматривается последовательно от первого до последнего элемента. Наиболее сложным, как уже отмечалось, является случай, когда аргумента (ключа) нет в таблице (не найден).

Уточненный алгоритм будет таким.

1. Ввести исходный массив (ключей).
2. Выполнять
   2. 1. Ввести *аргумент поиска* (целое число);
   2. 2. Если *аргумент поиска* больше или равен нулю,
      2.2.1. Результат (номер в массиве, *num*) = - 1 (не найден, такого номера нет);
      2.2.2. Для индекса массива (*i)* от 0 до Длина.массива
         Если *аргумент поиска* = ключ[*i*],
            номер в массиве (*num*) = *i*;
      2.2.3. Если номер *num* = - 1,
         вывести: «Такого ключа в массиве нет»,
         Иначе
         вывести: «Ключ найден под номером *num*».

пока будет *аргумент поиска* больше или равен нулю.

3. Закончить.

Основной недостаток алгоритма линейного поиска – большое время. Он предполагает использование оператора **For** в пункте 2.2.2. При этом ВСЕГДА выполняется ровно ***n*** операций сравнения, не зависимо от того, найден ключ или нет. Программа, обнаружив

аргумент в начале массива, продолжает его просмотр до конца, т.е. выполняет бесполезную работу. Время поиска может быть существенно сокращено, если обеспечить его прекращение, когда ключ найден. При равномерном распределении элементов в таблице эталонов (ключей) среднее время поиска может стать пропорциональным величине **n / 2**.

Этого можно достичь, изменив пункт 2.2 в рассмотренном выше алгоритме следующим образом.

Ускорение линейного поиска

2. Выполнять

2. 1. Ввести *аргумент поиска* (целое число);

2. 2. Если *аргумент поиска* больше или равен нулю,

2.2.1. Результат (*num*) = - 1 (не найден);

2.2.2. Начальный индекс, *i*=0;

2.2.3. Пока (num=-1) и (i<=n – Длины.массива)

a) Если *аргумент поиска* = ключ[*i*],
номер в массиве (*num*) = *i*;

b) *i*=*i* + 1

2.2.4. Если номер *num* = - 1,
вывести: «Такого ключа в массиве нет»,
Иначе
вывести: «Ключ *num* найден».

пока будет *аргумент поиска* больше или равен нулю.

Трудоемкость (временная сложность) алгоритма линейного поиска определяется числом операций сравнения, выполняемых при просмотре таблицы эталонов. В лучшем случае количество таких операций равно 1, в худшем – *n*, а в среднем, если возможные значения ключей равновероятны, - *n* / 2. Таким образом, асимптотическая оценка O(*n*) = *n*.

2. Алгоритм дихотомического поиска

Этот алгоритм является более быстрым, чем линейный, но ***применяется только к упорядоченным массивам***. Среднее время дихотомического поиска пропорционально величине $\mathbf{log_2 n}$, где n — количество элементов таблицы эталонов. Таким образом, ускорение достигается за счет дополнительной информации о расположении элементов, а асимптотическая оценка алгоритма $O(n) = log_2 n$.

Метод основан на последовательном делении на 2 диапазона поиска. При этом на каждом шаге либо находится элемент, либо происходит переход в одну из половин диапазона. В процессе поиска выполняется не только сравнение на равенство, но и на больше - меньше. Последняя операция позволяет выбрать очередную половину диапазона таблицы. Если массив эталонов не упорядочен, то выбор будет сделан неверно, и результат можно не получить никогда (говорят, что алгоритм расходится).

Общий алгоритм поиска будет таким же, как в п. 1.

1. Ввести исходный массив (ключей).
2. Повторять
   ввести аргумент поиска и вывести результат
   пока не надоест.
3. Закончить.

Для корректности работы алгоритма необходимо упорядочить ключи (массив эталонов) по возрастанию. Для простоты их значения можно задать с помощью датчика случайных чисел. После этого необходимо выполнить операцию сортировки полученных величин.

Уточненный ***алгоритм*** можно представить в следующем виде.

1.1. Ввести количество элементов массива (n).
1.2. Инициировать датчик случайных чисел.
1.3. Для номера элемента (i) от 0 до n
   1.3.1. Вычислить ключ[i].

2. Упорядочить ключи по возрастанию:

2.1. Для номера просмотра (*k*) от 1 до *n* - 1

2.1.1. Для номера слова (*i*) от 0 до *n* - *k*

Если ключ [*i*] > ключ [*i*+1],

поменять местами ключ[*i*] и ключ[*i*+1].

3. Выполнять

3. 1. Ввести *аргумент поиска* (целое число);

3. 2. Если *аргумент поиска* больше или равен нулю,

3.2.1 Граница_левая (диапазона поиска) = 0;

3.2.2. Граница_правая (диапазона поиска) = *n* - 1 (Длина.массива – 1);

3.2.3. Если *аргумент поиска* = ключ [*n* - 1],

a) Признак = «Найдено»;

b) *k* = *n* – 1.

Иначе

3.2.4. Признак = «Не найдено».

3.2.5. Выполнять

a) Номер *аргумент поиска* (*k*) = Целое ((Граница_левая + Граница_правая)/2);

b) Если *аргумент поиска* = ключ [*k*],

Признак:= «Найдено»

Иначе

Если *аргумент поиска* > ключ [*k*],

Граница_левая = *k*

Иначе

Граница_правая = *k*.

Пока не «Найдено» Или (Граница_левая = Граница_правая - 1).

3.3. Если «Найдено»,

вывести: «Ключ найден под номером *k*»,

Иначе

вывести: «Такого ключа в массиве нет».

Пока будет *аргумент поиска* больше или равен нулю.
4. Закончить.

В алгоритме пункт 3.2.3 применен для обеспечения нахождения последнего элемента массива. Дело в том, что при целочисленном делении на 2 дробная часть частного отбрасывается, и результат всегда будет на 1 меньше, чем длина таблицы.

3. Алгоритм интерполирующего поиска

Интерполирующий поиск основан на принципе поиска в телефонной книге или, например, в словаре. Вместо сравнения каждого элемента с искомым как при линейном поиске, производится предсказание местонахождения элемента. Процедура похожа на двоичную, но вместо деления области поиска на две части, производится оценка новой области по расстоянию между ключом и текущим значением аргумента. Другими словами, бинарный поиск учитывает лишь знак разности между ключом и текущим значением, а интерполирующий - ещѐ и модуль этой разности и по этому значению производит предсказание позиции следующего элемента для проверки.

В среднем, интерполирующий поиск производит $\log(\log(n))$ операций, где n - число элементов в массиве. Количество необходимых операций зависит от равномерности распределения значений среди элементов. В плохом случае (например, когда значения элементов экспоненциально возрастают) интерполяционный поиск может потребовать до $O(n)$ операций.

На практике, интерполяционный поиск часто быстрее бинарного, так как их отличают, прежде всего, применяемые арифметические операции:

a) интерполирование — в интерполирующем поиске и
b) деление на два — в двоичном.

При этом скорость вычисления отличается незначительно. С другой стороны интерполирующий поиск использует такое

принципиальное свойство данных, как однородность распределения значений. Ключом может быть не только номер, число, но и, например, текстовая строка, тогда становится понятна аналогия с телефонной книгой: если мы ищем имя в телефонной книге, начинающееся на «А», то нужно искать его в начале, но никак не в середине. В принципе, ключом может быть всѐ что угодно, так как те же строки, например, кодируются посимвольно, в простейшем случае символ можно закодировать значением от 1 до 33 (только русские символы) или, например, от 1 до 26 (только латинский алфавит) и т. д.

Интерполяция может производиться на основе функции, аппроксимирующей распределение значений, либо набора кривых, выполняющих аппроксимацию на отдельных участках. В этом случае поиск может завершиться за несколько проверок. Преимущества метода состоят в уменьшении запросов на чтение медленной памяти (такой, например, как жесткий диск), если запросы происходят часто.

Часто анализ и построение аппроксимирующих кривых не требуется, например, когда все элементы отсортированы по возрастанию. В таком списке минимальное значение будет по индексу 0, а максимальное по индексу n - 1. В этом случае аппроксимирующую кривую можно принять за прямую и применять линейную интерполяцию.

Общий ***алгоритм интерполирующего поиска*** без задания исходного массива может быть таким. На каждой стадии рассчитывается позиция mid (средняя) для следующей проверки, по формуле:

mid = low + (аргумент - ключ [low])*(high - low) /
(ключ [high] - ключ[low])

где low – нижняя граница диапазона,
high – его верхняя граница.

Затем в зависимости от результата сравнения переносится верхняя или нижняя граница, определяя новую область поиска. Процесс

заканчивается, если элемент с индексом mid равен искомому ключу или нижняя и верхняя границы поиска совпали.

Пошаговый алгоритм интерполирующего поиска можно записать следующим образом.

1. Выполнять
   1.1. Ввести *аргумент поиска* (целое число);
   1.2. Если *аргумент поиска* больше или равен нулю,
      1.2.1. Искомый номер (num) = - 1.
      1.2.1. Нижняя_граница (low) = 0;
      1.2.2. Верхняя_граница (high) = *n* - 1;
      1.2.3. Пока (ключ[low] < *аргумент* И ключ [high] > *аргумент*)
         a) Средний(mid) = low + (*аргумент* - ключ [low])*(high - low) / (ключ [high] - ключ[low]);
         b) Если ключ [mid]< *аргумента,*
            low = mid + 1;
         Иначе
            Если ключ [mid]< *аргумента,*
               high = mid - 1;
            Иначе
               mid – искомый номер.
      1.2.4. Если ключ[low] = *аргумент,*
         low – искомый номер num
      Иначе
         Если ключ[high] = *аргумент,*
            high – искомый номер num
         Иначе
            Номер num не найден

В заключение можно отметить, что рассмотренные алгоритмы могут применяться не только к числам, но и к строкам. Примером являются задачи поиска слова в словаре, справочнике, таблице и т.д. Применение дихотомического поиска требует упорядочения слов по алфавиту.

Порядок выполнения лабораторной работы

Работа предполагает выполнение следующих этапов.

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на разработку программы для алгоритмов поиска (см. прил. 2).
3. Разработка и отладка заданной программы.
4. Получение верхней и экспериментальной оценки времени выполнения заданного алгоритма и программы.
5. Нахождение предельной оценки емкости памяти, необходимой для выполнения разработанной программы.

Содержание отчета о выполненной работе

Отчет о выполненной работе должен содержать.

1. Название и цель работы.
2. Словесное описание заданного алгоритма поиска.
3. Текст программы.
4. Формулы верхней оценки временной и емкостной сложности заданного алгоритма.
5. Результаты экспериментальной оценки временной и емкостной сложности заданного алгоритма.

Контрольные вопросы

1. Что такое поиск и для чего он нужен?
2. Что является исходными данными для поиска?
3. Какие алгоритмы поиска Вы знаете?
4. Приведите словесное описание простейшего алгоритма линейного поиска (с циклами For).
5. Приведите словесное описание ускоренного алгоритма линейного поиска.
6. Приведите словесное описание алгоритма дихотомического поиска.
7. Приведите словесное описание алгоритма интерполирующего поиска.
8. Какова верхняя оценка трудоемкости алгоритма линейного поиска?

9. Какова верхняя оценка трудоемкости алгоритма дихотомического поиска?
10. Какова верхняя оценка трудоемкости алгоритма интерполирующего поиска?
11. Какова верхняя оценка емкостной сложности алгоритма линейного поиска?
12. Какова верхняя оценка емкостной сложности алгоритма дихотомического поиска?
13. Какова верхняя оценка емкостной сложности алгоритма интерполирующего поиска?
14. На сколько отличаются результаты оценки трудоемкости предложенного Вам алгоритма, полученные аналитическими (по формулам) и экспериментальными методами?
15. На сколько отличаются результаты оценки емкостной сложности предложенного Вам алгоритма, полученные аналитическими (по формулам) и экспериментальными методами?

Практические задания к лабораторной работе

1. Разработать алгоритм и программу простого линейного поиска с циклом For. В качестве исходных данных использовать строку текста, из которой необходимо выделить слова. Аргумент поиска – слово.
2. Разработать алгоритм и программу ускоренного линейного поиска. В качестве исходных данных использовать строку текста, из которой необходимо выделить слова. Аргумент поиска – слово.
3. Разработать алгоритм и программу дихотомического поиска. В качестве исходных данных использовать массив целых чисел, который вводится с клавиатуры. Аргумент поиска – число.
4. Разработать алгоритм и программу дихотомического поиска. В качестве исходных данных использовать массив целых чисел, который формируется с помощью датчика случайных чисел с диапазоном от 0 до 100. Аргумент поиска – число.
5. Разработать алгоритм и программу интерполирующего поиска.

В качестве исходных данных использовать массив целых чисел, который вводится с клавиатуры. Аргумент поиска – число.

6. Разработать алгоритм и программу интерполирующего поиска. В качестве исходных данных использовать массив целых чисел, который формируется с помощью датчика случайных чисел с диапазоном от 0 до 100. Аргумент поиска – число.

7. Разработать алгоритм и программу простого линейного поиска с циклом For. В качестве исходных данных использовать строку текста, из которой необходимо выделить слова. Затем слова упорядочить по алфавиту. Аргумент поиска – слово.

8. Разработать алгоритм и программу ускоренного линейного поиска. В качестве исходных данных использовать строку текста, из

которой необходимо выделить слова. Затем слова упорядочить по алфавиту. Аргумент поиска – слово.