Тема 4. Методы поиска

Поиск является одной из основных операций над множествами и обычно формулируется следующим образом. Имеется конечное множество $T = \{x_1, x_2, ..., x_n\}$ из n элементов, которое будем называть *таблицей*. В общем случае элементы таблицы могут иметь сложную структуру, но с каждым элементом ассоциирован некоторый *ключ* (*имя*), используемый для того, чтобы отличить один элемент от другого. Поскольку рассматривается в первую очередь процесс поиска, а обнаруженные данные с точки зрения поиска не представляют интереса, будем считать, что элементами таблицы являются имена (ключи). Поиск представляет собой запрос, который для заданной таблицы T и некоторого имени z, называемого *ключом*, или *аргументом поиска*, возвращает указатель на имя из таблицы, совпадающее с z (*успешный* поиск). Если такого имени нет, возвращает информацию (например, пустое значение указателя) о *безуспешном* поиске.

Таблица T является конечным подмножеством некоторого множества S, называемого *пространством имен*, которое может быть конечным или бесконечным. На множестве S должно быть определено отношение линейного (*естественного*) порядка, т. е. любые два элемента $x, y \in S$ сравнимы (должно выполняться в точности одно из трех условий: $x < y$, $x = y$, $x > y$). Отношение порядка должно обладать также свойством транзитивности, т. е. если $x < y$ и $y < z$, то $x < z$ для любых элементов $x, y, z \in S$. При анализе алгоритмов будем предполагать, что исход сравнений элементов $x, y \in S$ получается за время, не зависящее от мощности пространства имен и от мощности n исследуемой таблицы T.

Порядок, определенный на таблице T, будем называть *табличным*. Табличный порядок часто совпадает с естественным порядком, определенным на пространстве имен, однако такое совпадение не обязательно.

Таблицы, которые после их построения никогда не изменяются, называются *статическими*. Такие таблицы необходимо строить так, чтобы максимально минимизировать время поиска.

На большинстве таблиц, однако, производятся и другие операции. Наиболее важными из них являются операции модификации (добавление или исключение имени, сортировка), объединение нескольких таблиц в одну и т. д. В таких таблицах, называемых *динамическими*, когда встает вопрос об эффективности выполнения тех или иных операций, нельзя ограничиваться оптимизацией структуры таблицы для достижения наибыстрейшего поиска. Это связано с тем, что общая эффективность работы с таблицами снижается за счет неэффективного выполнения других операций. Необходимо искать компромиссное решение, когда время поиска удлиняется из соображений более гибкой организации таблиц, которая позволяет выполнять некоторые виды операций более эффективно.

Ограничимся рассмотрением следующих четырех операций над таблицами:

Поиск z: если $z \in T$, то отметить его указателем, в противном случае указать, что $z \notin T$.

Включение z: поиск места включения; если $z \notin T$, то поместить его на соответствующее место.

Исключение z: поиск имени z в таблице; если $z \in T$, исключить его.

Распечатка: распечатка всех имен из таблицы T в их естественном порядке. Если табличный порядок не соответствует естественному, предполагается предварительная сортировка.

Детали реализации этих операций во многом зависят от структуры данных, используемой для представления таблицы.

4.1. Последовательный поиск

Последовательный (*линейный*) поиск подразумевает исследование имен в том порядке, в котором они встречаются в таблице, т. е. в худшем случае осуществляется просмотр всей таблицы. Для больших таблиц последовательный поиск нельзя отнести к методам быстрого поиска, поскольку даже в среднем он имеет тенденцию к использованию числа операций, пропорционального размеру n таблицы. Несмотря на это последовательный поиск является единственным методом поиска, применимым к отдельным устройствам памяти и к тем таблицам, которые строятся на пространстве имен без линейного порядка. Кроме того, последовательный поиск является быстрым для достаточно малых таблиц.

Наиболее очевидный процесс последовательного поиска в таблице $T = \{x_1, x_2, \ldots, x_n\}$ первого вхождения заданного имени z (имя z – ключ поиска) представлен алгоритмом 4.1. Алгоритм возвращает позицию i имени $x_i = z$ в случае успешного поиска и нулевое значение указателя – в случае безуспешного поиска.

```
for i ← 1 to n do
   if z = x_i then return(i)  // найдено: i указывает на z
return(0)  // не найдено: z не входит в T
```

Алгоритм 4.1. Последовательный поиск

Логика алгоритма показана для таблицы, представленной с помощью массива. Очевидно, что она остается той же самой и для таблицы, представленной с помощью связного списка. В этом случае i будет указывать на соответствующий узел списка при успешном поиске или иметь пустое значение Λ при безуспешном. Перемещение по списку реализуется с помощью поля *next* узла, т. е. вместо присваивания $i \leftarrow i + 1$ (скрытого в операторе **for**) следует использовать оператор $i \leftarrow i.next$.

Чтобы детализировать выполняемые внутри цикла операции, представим алгоритм в форме, близкой к машинному языку:

```
        i ← 1
цикл: if i > n then // не найдено
      if z = x_i then // найдено
      i ← i + 1
      goto цикл
```

За каждую итерацию выполняется до четырех команд: два сравнения, одна операция увеличения i и одна операция передачи управления. Любое ускорение работы алгоритма должно быть следствием сокращения числа операций в цикле, поскольку время выполнения цикла определяет время работы всего алгоритма.

Для ускорения цикла общим приемом является добавление в таблицу специальных имен (называемых *сторожами*, или *часовыми*), которые делают необязательной явную проверку того, достиг ли указатель границы таблицы. Этот прием можно применить и к рассматриваемому алгоритму. Если перед поиском добавить искомое имя z в конец таблицы в качестве сторожа, то цикл всегда будет завершаться отысканием имени z, т. е. в цикле нет необходимости каждый раз выполнять проверку $i > n$. После выхода из цикла проверка условия $i > n$ выполняется только один раз для определения, является ли найденное имя z истинным или специальным добавленным именем-сторожем. В результате получается улучшенный алгоритм последовательного поиска (алгоритм 4.2), в котором при каждой итерации выполняются только три операции вместо четырех.

$x_{n+1} \leftarrow z$ // добавление сторожа
$i \leftarrow 1$
while $z \neq x_i$ **do** $i \leftarrow i+1$
if $i > n$
  then **return**(0) // не найдено
  else **return**(i) // найдено: i указывает на z

Алгоритм 4.2. Улучшенный алгоритм последовательного поиска

Одной из особенностей алгоритма является то, что добавление перед поиском ключа z в конец таблицы возможно только, если таблица хранится в памяти с произвольным доступом, где имеется прямой непосредственный доступ к концу таблицы. Алгоритм неприменим, когда используется связное размещение имен таблицы (связный список с одним указателем на первый элемент) или память с последовательным доступом. В случае представления таблицы связным списком можно выйти из положения, если дополнить список указателем на последний элемент. Тогда можно реализовать добавление в конец списка узла с именем-сторожем (перед началом поиска), а после завершения поиска следует исключить этот узел.

Недостатком двух рассмотренных алгоритмов является то, что при безуспешном поиске всегда просматривается вся таблица. Если такой поиск возникает часто, имена необходимо хранить в естественном порядке. Это позволяет завершить поиск, когда при просмотре попадается первое имя, большее или равное ключу поиска. В этом случае в конец таблицы необходимо добавить имя-сторож ∞ (предполагается, что имя ∞ больше любого имени из пространства имен S), чтобы гарантировать выполнение условия завершения. В результате получается алгоритм 4.3.

$x_{n+1} \leftarrow \infty$
$i \leftarrow 1$
while $z > x_i$ **do** $i \leftarrow i + 1$
if $z = x_i$
  then **retutn**(i) // найдено: i указывает на z
  else **return**(0) // не найдено

Алгоритм 4.3. Последовательный поиск в упорядоченной таблице

Для анализа времени последовательного поиска за единицу времени примем время, необходимое для исследования одного элемента таблицы. Для нахождения в таблице i-го имени x_i требуется i единиц времени. Предполагая, что частота обращения к именам таблицы распределена равномерно, можно вычислить среднее время успешного поиска S_n как

$$S_n = \frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2}.$$

Для безуспешного поиска среднее время равно примерно $n/2$, если имена в таблице хранятся в естественном порядке, или равно n в противном случае. Таким образом, временная сложность алгоритмов последовательного поиска составляет $O(n)$.

Операции включения и исключения в большей степени зависят от структуры данных, выбранной для представления таблицы, чем от порядка имен в таблице. Наиболее важными структурами данных для последовательного поиска являются связные списки и последовательно распределенные таблицы. Операции включения и исключения в связном списке требуют только изменения нескольких указателей, т. е. если известно местоположение, эти операции могут быть выполнены за время, не зависящее от размера таблицы. В последовательно распределенной таблице имена хранятся в последовательных ячейках. Поэтому исключение имени образует пропуск, который должен быть заполнен, а включение имени требует предварительного образования пропуска. В обоих случаях необходимо передвижение имен, что требует времени $O(n)$.

Распечатка имен заключается в последовательном просмотре имен таблицы с временем $O(n)$, если табличный порядок совпадает с естественным, в противном случае требуется предварительная сортировка с соответствующим временем.

Можно уменьшить среднее время успешного поиска, если имена в таблице хранить в порядке невозрастания частот обращения. Распределение частот обращения обычно не известно. Если таблица является статической, то можно собрать статистику о частоте обращений, а затем реорганизовать таблицу в соответствии с полученным распределением частот. Если нет возможности собрать подобную статистику, таблицы можно сделать *самоорганизующимися*. В таких таблицах имена, к которым обращаются часто, передвигаются в направлении начала таблицы, т. е. по мере получения данных о частотах обращений происходит изменение табличного порядка. Следует отметить, что такое улучшение среднего времени успешного поиска приводит к увеличению времени безуспешного поиска, поскольку табличный порядок не совпадает с естественным.