

Поиск информации

Задача поиска:

где в заданной совокупности данных находится элемент, обладающий заданным свойством?

Большинство задач поиска сводится к поиску в таблице элемента с заданным значением.

Алгоритмы поиска информации

Линейный поиск

Пример:

Написать программу поиска элемента x в массиве из n элементов. Значение элемента x вводится с клавиатуры.

Решение:

Дано:

Const $n = 10$;

Var a : *Array*[1.. n] of *integer*;

x : *integer*;

В данном случае известно только значение разыскиваемого элемента, никакой дополнительной информации о нем или о массиве, в котором его надо искать, нет. Поэтому для решения задачи разумно применить последовательный просмотр массива и сравнение значения очередного рассматриваемого элемента с данным. Если значение очередного элемента совпадает с x , то запомним его номер в переменной k .

For $i:=1$ to n do

if $a[i] = x$ then $k:=i$;

Недостатки данного метода:

- если значение x встречается в массиве несколько раз, то найдено будет последнее из них;
- после того, как нужное значение уже найдено, массив просматривается до конца, т.е. всегда выполняется n сравнений.

Прервем просмотр сразу же после обнаружения заданного элемента!

Используем цикл с предусловием.

While (i ≤ n) and (a[i] ≠ x) do inc(i);

В результате:

- либо будет найден искомый элемент, т.е. найдется такой индекс i , что $a[i] = x$;
- либо будет просмотрен весь массив, и искомый элемент не обнаружится.

Поскольку поиск заканчивается только в случае, когда $i = n + 1$ или когда искомый элемент найден, то из этого следует, что если в массиве есть несколько элементов, совпадающих с элементом x , то в результате работы программы будет найден первый из них, т.е. элемент с наименьшим индексом.

Задание

- оформить программу и проследить ее работу в режиме пошагового просмотра при различных значениях X ;
- модифицировать программу для поиска элемента массива, равного X , с максимально возможным индексом.

Линейный поиск с использованием барьера

Недостатком нашей программы является то, что в заголовке цикла записано достаточно сложное условие, которое проверяется перед каждым увеличением индекса, что замедляет поиск. Чтобы ускорить его необходимо максимально упростить логическое выражение.

Для этого используем искусственный прием!

В массиве на $n + 1$ место запишем искомый элемент x , который будет являться барьерным. Тогда если в процессе работы программы

$a[n + 1] := x; i := 1;$

***While* $a[i] \neq x$ *do* $inc(i);$**

обнаружится такой индекс i , что $a[i] = x$, то элемент будет найден. Но если $a[i] = x$ будет только при $i = n + 1$, то, значит, интересующего нас элемента в массиве нет.

В случае наличия в массиве нескольких элементов, удовлетворяющих заданному свойству, будет также найден элемент с наименьшим номером.

Задание

Изменить программу так, чтобы был найден элемент с максимально возможным индексом.

Если никаких дополнительных сведений о массиве, в котором хранится массив нет, то ускорить поиск нельзя.

Если же известна некоторая информация о данных, среди которых ведется поиск, например, массив данных отсортирован, удастся существенно сократить время поиска, применяя непоследовательные методы поиска.

Бинарный поиск

Иначе **двоичный поиск** или **метод половинного деления.**

При его использовании на каждом шаге область поиска сокращается вдвое.

Задача

Дано целое число x и массив $a[1..n]$,
отсортированный в порядке неубывания
чисел, то есть для любого k : $1 \leq k < n$:
 $a[k-1] \leq a[k]$.

Найти такое i , что $a[i] = x$ или сообщить,
что элемента x в массиве нет.

Идея бинарного метода

- проверить, является ли ***X*** ***средним элементом*** массива. Если да, то ответ получен. Если нет, то возможны два случая:
 - ***X*** ***меньше среднего элемента***.
Следовательно, после этого данный метод можно применить к левой половине массива.
 - ***X*** ***больше среднего элемента***.
Аналогично, теперь этот метод следует применить к правой части массива.

Пример:

Массив **a**:

3 5 6 8 12 15 17 18 20 25

x = 6

Шаг 1.

Найдем номер среднего элемента:

$$m = \frac{1 + 10}{2} = 5 \quad \text{Так как } 6 < a[5]$$

3 5 6 8 ~~12 15 17 18 20 25~~

Шаг 2. Рассмотрим лишь первые 4 элемента массива. Индекс среднего элемента:

$$m = \frac{1 + 4}{2} = 2 \quad \text{Аналогично:}$$

~~3 5 6 8 12 15 17 18 20 25~~

Шаг 3. Рассматриваем два элемента

$$m = \frac{3 + 4}{2} = 3$$

~~3 5 6 8 12 15 17 18 20 25~~

A[3] = 6! Его номер – 3

В общем случае формула поиска значения среднего элемента **m**:

$$m = \frac{L + R}{2}$$

Где **L** – индекс первого, а **R** – индекс последнего элемента рассматриваемой части массива.

Фрагмент программной реализации бинарного поиска:

Begin

L:= 1; R:= n; {на первом шаге – весь массив}

f:= false; {признак того, что x не найден}

while (L<=R) and not f do

begin

m:= (L + R) div 2;

if a[m] = x then f:= true { элемент найден.

Поиск надо прекратить}

else if a[m] < x then L:= m + 1 {отбрасывается левая
часть}

else R:= m – 1 {отбрасывается правая часть}

end

End;

Бинарный поиск с использованием фиктивного «барьерного» элемента.

Begin

$a[0] := x;$

$L := 1; R := n;$

Repeat

$m := (L + R) \text{ div } 2;$

if $L > R$ then $m := 0$

else if $a[m] < x$ then $L := m + 1$

else $R := m - 1$

until $a[m] = x;$

ans := m;

End;

(Списать в тетрадь. Добавить комментарий)

Задание:

Использование идеи двоичного поиска позволяет значительно улучшить **алгоритм сортировки массива методом простого включения**. Учитывая, что готовая последовательность, в которую надо вставлять элемент, является упорядоченной, можно методом деления пополам определить позицию включения нового элемента в неё. Такой модифицированный алгоритм сортировки называется **методом двоичного включения**.

Написать программу, реализующую этот метод.