Михаил Густокашин. Поиск в неупорядоченных массивах.

Самым простым вариантом поиска можно считать поиск элемента в одномерном неупорядоченном массиве. Сформулируем задачу следующим образом: дан одномерный неупорядоченный массив, состоящий из целых чисел, и необходимо проверить, содержится ли данное число в этом массиве.

Пусть массив называется a и состоит из n элементов, а искомое число равно k. Тогда код, осуществляющий поиск, можно записать так:

```
int j = -1;
for (i=0; i < n; i++)
  if (a[i] ==  k) j = i;
```

В случае если число k ни разу не встречалось в массиве, j будет равно -1. Приведенная выше функция будет искать последнее вхождение числа k в массиве a. Если нам необходимо искать первое вхождение, то вместо присваивания $j = i$ следует добавить оператор `break;` (в этом случае искомый индекс будет храниться в переменной i).

И в том и в другом случае алгоритм будет иметь сложность $O(N)$.

На этом примере можно рассмотреть «барьерный» метод, который может быть полезен в очень многих задачах. Для использования барьерного метода наш массив должен иметь один дополнительный элемент (т.е. его длина должна быть не меньше, чем $n + 1$ элемент). Отметим, что таким способом можно искать только первое вхождение элемента:

```
a[n+1] = k;
for (i=0; a[i] != k; i++);
```

Если элемент k встречается в массиве, то его индекс будет находиться в переменной i, если же такой элемент в массиве не встречается, то i будет равно $n + 1$.

Рассмотрим отдельно задачу поиска минимума и максимума в массиве. Так же как и при поиске вхождения элемента, будем искать не само значения минимума или максимума, а индекс минимального (максимального) элемента. Это избавит нас от многих проблем и позволит совершать меньшее количество ошибок при программировании. Поиск минимального элемента в массиве a будет выглядеть следующим образом:

```
imin = 0;
For (i, n)
  if (a[i] < a[imin]) imin = i;
```

Индекс минимального элемента будет храниться в переменной $imin$, а сам минимум равен $a[imin]$. Минимум и максимум следует обязательно искать по индексу, а не по значению. Например, если мы будем пытаться хранить непосредственно значение минимума или максимума, то можем легко ошибиться с начальной инициализацией. Например, для массива вещественных чисел определить значения, которыми изначально следует инициализировать минимум и максимум.

Теперь рассмотрим задачу поиска минимума и максимума одновременно. Можно реализовать такой поиск аналогично:

```
imin = 0;
imax = 0;
for (i=1; i<n; i++)
{
  if (a[i] < a[imin]) imin = i;
  if (a[i] > a[imax]) imax = i;
}
```

Такая реализация требует $2 \times N - 2$ сравнения. Но эту задачу можно решить и за меньшее количество сравнений. Разобьем все элементы на пары, и будем искать в каждой паре минимум и максимум ($N/2$ сравнений), затем минимум будем искать только среди минимальных элементов пар, а максимум — среди максимальных. Общее количество сравнений будет около $3 \times N/2$ (проблема возникает, когда количество элементов нечетное — один из элементов остается без пары). Точно это можно записать как $\lceil 3 \times N/2 \rceil - 2$, где $\lceil\ \rceil$ — округление до большего целого.

Рассмотрим еще один способ поиска максимума. После разбиения элементов на пары будем продолжать этот процесс, аналогично турниру «на вылет». Т.е. заново разобьем максимальные элементы из пар на пары и снова найдем максимум и т.д. Для поиска максимального элемента будет по-прежнему требовать $N - 1$ операция сравнения, но сам максимальный элемент будет участвовать только в $\log N$ сравнениях. И одно из этих сравнений обязательно будет со вторым по величине элементом. Таким образом, для поиска второго по величине элемента будет требоваться $\lceil \log N \rceil - 1$ сравнение (при условии, что все сравнения для максимального элемента проведены).

Обычно такие методы используются в особых случаях, когда это непосредственно требуется в решении задачи. Для общего случая подходят более простые методы, где количество сравнений не играет такой важной роли.