
# 查 找

## 顺序查找法

**基本思想：**

一列数放在数组 a[1]～a[n]中，待查找的数放在 x 中，把 x 与 a 数组中的元素从头到尾一一进行比较查找。用变量 i 表示 a 数组元素下标，i 初值为 1，使 x 与 a[i]比较，如果 x 不等于 a[i]，则使 i++，不断重复这个过程；一旦 x 等于 a[i]则退出循环；另外，如果 p 大于数组长度，循环也应该停止。

**程序清单：**

```
#include "stdio.h"
int find(int *a,int x)
{
  int i;
  for(i=0;i<10;i++)
    if(a[i]==x) return i+1;
  return 0;
}

main()
{
  int a[10]={75,87,68,92,88,61,77,96,80,72};
  int x,i,ans;
  printf("Input x:");
  scanf("%d",&x);
  ans=find(a,x);
  if(ans!=0)
    printf("%d index is %d",x,ans);
  else
    printf("Not found!");
  getch();
}
```

**算法分析：**

对于 n 个元素一维数组，元素的查找在等概率的前提下，查找成功的概率 $p_i = \frac{1}{n}$。因此，顺序查找的缺点是速度较慢，平均查找长度为 O(n)。

# 折半查找法

**基本思想：**（只能对有序数列进行查找）

二分查找法又称为。它的基本思想是：设n个有序数（从小到大）存放在数组a[1]～a[n]中，要查找的数为x。用变量left、right、mid分别表示查找数据范围的底部（数组下界）、顶部（数组的上界）和中间，mid=(left+right)/2，折半查找的算法如下：

（1）x=a[mid]，则已找到退出循环，否则进行下面的判断；

（2）x<a[mid]，x必定落在left和mid-1的范围之内，即right=mid-1；

（3）x>a[mid]，x必定落在mid+1和right的范围之内，即lefts=mid+1；

（4）在确定了新的查找范围后，重复进行以上比较，直到找到或者left<=right。

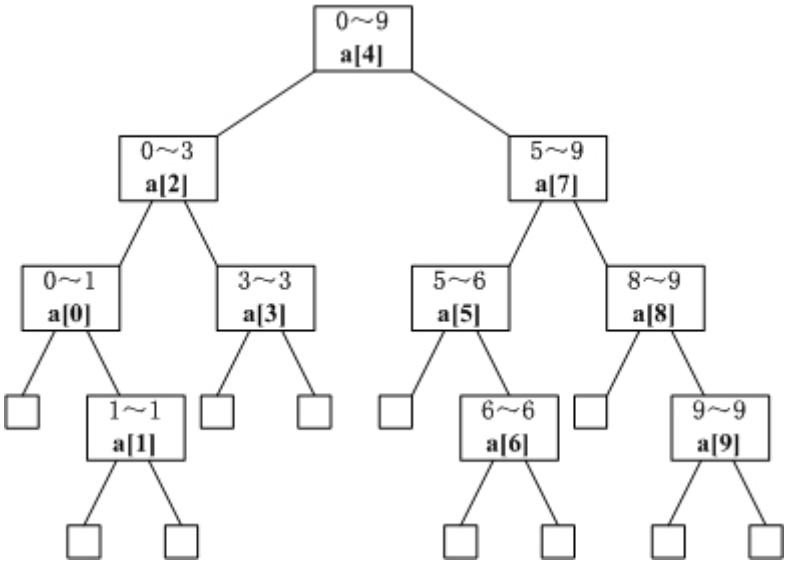
**程序清单：**

```
#include "stdio.h"
main()
{
  int a[10]={2,4,7,9,10,14,18,26,32,40};
  int left,mid,right,i,x,find;
  printf("Input x:");
  scanf("%d",&x);
  left=0; right=9; find=0;
  while(left<=right && find==0)
  {
    mid=(left+right)/2;
    if(x==a[mid])
    {
      find=mid+1;
      break;
    }
    else if(x<a[mid])
      right=mid-1;
    else
      left=mid+1;
  }
  if(find!=0)
    printf("%d index is %d",x,find);
  else
    printf("Not found!");
  getch();
}
```

**算法分析：**

二分查找过程构成一个判定树，把当前查找区间的中间位置上的元素作为根，左子表和右子表中的元

素分别作为根的左子树和右子树。上例中各元素对就的判定树如下图所示。图中大方形表示内部结点，内部结点中的数字表示该元素在有序表中的位置，如 0～9 表示当前的查找区间是 a[0]～a[9]，其中间元素为 a[4]。小方形结点表示外部结点，外部结点表示查找不成功出现的情况，如 a[3]结点的左边小方形，表示当查找元素小于 a[2]而大于 a[3]时查找失败的情况。




由此可见，成功的二分查找过程恰好是走了一条从判定树的根到被查元素的路径，经历比较的元素次数恰好为该元素在树中的层数。若查找失败，则其比较过程是经历了一条从判定树根到某个外部结点的路径，所需的元素比较次数是该路径上内部结点的总数。

在二分查找过程中，每比较一次查找范围就缩小一半。每次比较可能涉及的元素个数如下表所示。

| 比较次数 | 1 | 2 | 3 | 4 | …… | j |
|---|---|---|---|---|---|---|
| 可能涉及的元素个数 | 1 | 2 | 4 | 8 | …… | $2^{j-1}$ |

二分查找的平均查找长度为 $O(\log_2 n)$，比顺序查找速度快。