# 查找

## 定义

根据给定的某个值，在查找表中确定一个其关键字等于给定值的数据元素（或记录）。

## 分类

1）静态查找和动态查找；

注：静态或者动态都是针对查找表而言的。动态表指查找表中有删除和插入操作的表。

　　2）无序查找和有序查找。

无序查找：被查找数列有序无序均可；

有序查找：被查找数列必须为有序数列。

## 性能

平均查找长度（Average Search Length，ASL）：需和指定key进行比较的关键字的个数的期望值，称为查找算法在查找成功时的平均查找长度。

　　对于含有n个数据元素的查找表，查找成功的平均查找长度为：ASL = Pi\*Ci的和。

　　Pi：查找表中第i个数据元素的概率。

Ci：找到第i个数据元素时已经比较过的次数。

# 顺序表查找

## 基本思想

顺序查找也称为线形查找，属于无序查找算法。从数据结构线形表的一端开始，顺序扫描，依次将扫描到的结点关键字与给定值k相比较，若相等则表示查找成功；若扫描结束仍没有找到关键字等于k的结点，表示查找失败。

## 具体步骤

## 代码实现

## 性能分析

查找成功时的平均查找长度为：（假设每个数据元素的概率相等） ASL = 1/n(1+2+3+…+n) = (n+1)/2 ;

　　当查找不成功时，需要n+1次比较，时间复杂度为O(n);

　　所以，顺序查找的时间复杂度为O(n)。

# 有序表查找

## 折半查找

### 基本思想

也称为是折半查找/二分查找，属于有序查找算法。用给定值k先与中间结点的关键字比较，中间结点把线形表分成两个子表，若相等则查找成功；若不相等，再根据k与该中间结点关键字的比较结果确定下一步查找哪个子表，这样递归进行，直到查找到或查找结束发现表中没有这样的结点。

前提条件：

### 具体步骤

### 代码实现

#include <stdio.h>

#include <stdlib.h>

//二分查找

/\*

在一个有序序列中 查找某一个元素是否存在

如果存在，返回该元素在该序列中的索引位置

如果不存在，返回-1

\*/

/\*开区间实现：

如果high = n-1,while(high >= low) high = middle -1;

如果high = n while(high > low) high = middle;

\*/

int binary\_search(int\* array,int n,int key)

{

/\*闭区间实现\*/

int low = 0;

int high = n-1;

int mid;

if(high < low)

return -1;

while(high >= low)

{

mid = low+((high-low)>>1);

/\*二分查找中间值获取的标准写法，常见不规范的写法：

(high-low)/2或low+(high-low)/2\*/

if(array[mid] == key)

return mid;

else if(array[mid] > key)

high = mid-1;

else

low = mid+1;

}

return -1;

}

int main()

{

int array[]={1,2,3,4,5,6,7,8,9,10};

printf("%d\n",binary\_search(array,10,4));

return 0;

}

### 性能分析

### 应用

#### 查找元素的上下限

题目要求：在一个有序序列中查找某一元素的上下限。

代码：

#include <iostream>

#include <vector>

using namespace std;

//在一个有序序列中，查找某一个元素在该序列中的下限索引

//找到第一个不小于该元素的下标索引

int lower\_bound(int\* array,int low,int high,int key)

{

int mid;

if(high < low)

return -1;

//利用二分查找的策略

while(high >= low)

{

mid = low+((high-low)>>1);

if(array[mid] < key)

low = mid+1;

else

high = mid-1;

}

return low;

}

/\*

在一个有序序列中，查找某一个元素在该序列中的上限索引

找到第一个大于该元素的下标索引

\*/

int upper\_bound(int\* array,int low,int high,int key)

{

int mid;

if(high < low)

return -1;

while(high >= low)

{

mid = low+((high-low)>>1);

if(array[mid]>key)

high = mid-1;

else

low = mid+1;

}

return low;

}

/\*

查找某一个元素在一个有序序列总的范围

包含该缘故的上下限 该区间为两端闭区间

如果没有找到 就返回-1

\*/

pair<int,int> Findvalue(vector<int>& vec,int key)

{

pair<int,int> pos(-1,-1);

int mid,begin = 0,end = vec.size();

//首先判断该元素是否存在

while(begin <= end)

{

mid = begin + ((end-begin)>>1);

if(vec[mid] == key)

{

pos.first = mid;

pos.second = mid;

break;

}

else if(vec[mid] > key)

end = mid-1;

else

begin =mid+1;

}

if(vec[mid] != key)

return pos;

int low = mid-1;

int high = mid+1;

//找低地址

while(begin <= low)

{

mid = begin +(low-begin)/2;

if(vec[mid] < key)

begin = mid+1;

else

low = mid-1;

}

pos.first = begin;

//找高地址

while(high <= end)

{

mid = high +(end-high)/2;

if(vec[mid] > key)

end = mid-1;

else

high = mid+1;

}

pos.second = high -1;

return pos;

}

int main()

{

int array[]={1,2,3,3,3,3,3,3,4,5,7};

cout<<lower\_bound(array,0,10,3)<<endl;

cout<<upper\_bound(array,0,10,3)<<endl;

vector<int> vec(array,array+sizeof(array)/sizeof(int));

pair<int,int> pos = Findvalue(vec,3);

cout<<pos.first<<endl<<pos.second<<endl;

return 0;

}

拓展：熟悉STL模板库中查找元素上下限的函数，并阅读其代码实现。

#### 寻找多个集合的交集

#### 旋转数组中的查找

## 差值查找

### 基本思想

### 具体步骤

### 代码实现

### 性能分析

## 斐波那契查找

### 基本思想

### 具体步骤

### 代码实现

### 性能分析

# 线性索引查找

## 稠密索引

## 分块索引

## 倒排索引

# 二叉排序树

## 平衡二叉树（AVL树）

## 多路查找树（B树）

# 散列表查找（哈希表）