阅读：5,574　　作者：解学武

分块查找算法（索引顺序查找）及C语言实现

本节介绍一种在顺序查找的基础上对其进行改进的算法——分块查找算法。

分块查找，也叫索引顺序查找，算法实现除了需要查找表本身之外，还需要根据查找表建立一个索引表。例如图 1，给定一个查找表，其对应的索引表如图所示：

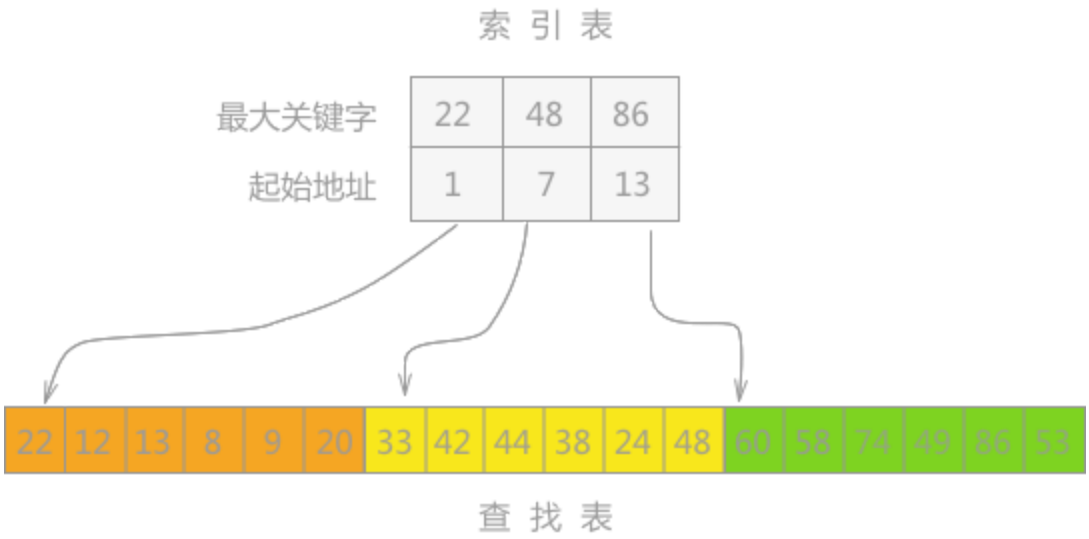

图 1 查找表及其对应的索引表

图 1 中，查找表中共 18 个查找关键字，将其平均分为 3 个子表，对每个子表建立一个索引，索引中包含中两部分内容：该子表部分中最大的关键字以及第一个关键字在总表中的位置，即该子表的起始位置。

> 建立的索引表要求按照关键字进行升序排序，查找表要么整体有序，要么分块有序。

分块有序指的是第二个子表中所有关键字都要大于第一个子表中的最大关键字，第三个子表的所有关键字都要大于第二个子表中的最大关键字，依次类推。

块（子表）中各关键字的具体顺序，根据各自可能会被查找到的概率而定。如果各关键字被查找到的概率是相等的，那么可以随机存放；否则可按照被查找概率进行降序排序，以提高算法运行效率。

分块查找的具体实现

所有前期准备工作完成后，开始在此基础上进行分块查找。分块查找的过程分为两步进行：

1. 确定要查找的关键字可能存在的具体块（子表）；
2. 在具体的块中进行顺序查找。

以图 1 中的查找表为例，假设要查找关键字 38 的具体位置。首先将 38 依次和索引表中各最大关键字进行比较，因为 22 < 38 < 48，所以可以确定 38 如果存在，肯定在第二个子表中。

由于索引表中显示第二子表的起始位置在查找表的第 7 的位置上，所以从该位置开始进行顺序查找，一直查找到该子表最后一个关键字（一般将查找表进行等分，具体子表个数根据实际情况而定）。结果在第 10 的位置上确定该关键字即为所找。

> 提示：在第一步确定块（子表）时，由于索引表中按照关键字有序，所有可以采用折半查找算法。而在第二步中，由于各子表中关键字没有严格要求有序，所以只能采用顺序查找的方式。

具体实现代码:

```
#include <stdio.h>
#include <stdlib.h>
struct index {  //定义块的结构
    int key;
    int start;
} newIndex[3];   //定义结构体数组

int search(int key, int a[]);

int cmp(const void *a,const void* b){
    return (*(struct index*)a).key>(*(struct index*)b).key?1:-1;
}
int main(){
    int i, j=-1, k, key;
    int a[] = {33,42,44,38,24,48, 22,12,13,8,9,20,  60,58,74,49,86,53};
    //确认模块的起始值和最大值
    for (i=0; i<3; i++) {
        newIndex[i].start = j+1;  //确定每个块范围的起始值
        j += 6;
        for (int k=newIndex[i].start; k<=j; k++) {
            if (newIndex[i].key<a[k]) {
                newIndex[i].key=a[k];
            }
        }
    }
    //对结构体按照 key 值进行排序
    qsort(newIndex,3, sizeof(newIndex[0]), cmp);

    //输入要查询的数，并调用函数进行查找
    printf("请输入您想要查找的数: \n");
    scanf("%d", &key);
    k = search(key, a);
    //输出查找的结果
```

```
    if (k>0) {
        printf("查找成功！您要找的数在数组中的位置是：%d\n",k+1);
    }else{
        printf("查找失败！您要找的数不在数组中。\n");
    }
    return 0;
}
int search(int key, int a[]){
    int i, startValue;
    i = 0;
    while (i<3 && key>newIndex[i].key) { //确定在哪个块中，遍历每个块，确定key在哪个块中
        i++;
    }
    if (i>=3) {  //大于分得的块数，则返回0
        return -1;
    }
    startValue = newIndex[i].start;  //startValue等于块范围的起始值
    while (startValue <= startValue+5 && a[startValue]!=key)
    {
        startValue++;
    }
    if (startValue>startValue+5) {  //如果大于块范围的结束值，则说明没有要查找的数
        return -1;
    }
    return startValue;
}
```

运行结果：

```
请输入您想要查找的数：
22
查找成功！您要找的数在数组中的位置是：7
```

分块查找的性能分析

分块查找算法的运行效率受两部分影响：查找块的操作和块内查找的操作。查找块的操作可以采用顺序查找，也可以采用折半查找（更优）；块内查找的操作采用顺序查找的方式。相比于折半查找，分块查找时间效率上更低一些；相比于顺序查找，由于在子表中进行，比较的子表个数会不同程度的减少，所有分块查找算法会更优。

> 总体来说，分块查找算法的效率介于顺序查找和折半查找之间。

联系方式　购买教程（带答疑）