**我抱枕没有抄袭别人作业**

# 第 10 章 检索

## 书面作业

1. 将关键字序列（7、8、30、11、18、9、14）散列存储到散列表中。散列表是一个下标从 0 开始的一维数组，散列函数为：H(key) = (key * 3) MOD 7，处理冲突采用线性探测法，要求装填（载）因子为 0.7。

(1) 请画出所构造的散列表。

(2) 分别计算等概率情况下查找成功和查找不成功的平均查找长度。

解:

(1) 共 7 个元素,装载因子为 0.7 即散列表长度为 10,所构造的散列表为

| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 值 | 7 | 14 | | 8 | | 11 | 30 | 18 | 9 | |

(2)查找成功:(1 + 1 + 1 + 1 + 3 + 3 + 2) / 7 = 12 / 7

查找不成功:(3 + 2 + 1 + 2 + 1 + 5 + 4) / 7 = 18 / 7

2. 两个整数集合 S1、S2，大小分别为 N、M，M=O(logN)，试借助于排序来求 S1 和 S2 的交集。请给出算法的伪代码描述和时间复杂度分析。

```
void intersection(set s1, set s2
                  , int size1, int size2, set& ans){
    /* 排序的时间开销最低 O(nlogn)(基于比较的排序) */
    s1.sort();
    /* 遍历 s2, 需要 O(logn)的时间 */
    for each i in s2{
       /* 在 s1 中二分查找 i 时间开销为 O(logn) */
        if(s1.find(i)){
            ans.insert(i);
        }
    }
    return ;
}
总的时间复杂度为 O(nlogn) + O(logn) *O(logn) = O(nlogn)
```

3. 给一个整数集合 S，定义 S 的子集 D 为连续子集当且仅当 D 中的整数构成连续的整数序列。求 S 的最大连续子集，即包含连续整数最多的子集。如{1, 3, 4, 100, 200, 2}的最大连续子集为{1,2,3,4}。

```
int main(){
    int a[64] = {};
    int size = 0;
    cin >> size;
    for(int i = 0; i < size; i += 1){
        cin >> a[i];
    }
    sort(a, a + size);
    int maxlength = -1, maxidx = -1;
    for(int idx = 0; idx < size; ){
        int tmplength = 1;
        int i = idx + 1;
        for( ; i < size && a[i] == a[i - 1] + 1; i += 1){
            tmplength += 1;
        }
        if(tmplength > maxlength){
            maxlength = tmplength;
            maxidx = idx;
        }
        idx = i;
    }
    for(int i = 0; i < maxlength; i += 1){
        cout << a[maxidx + i] << ' ';
    }
    cout << endl;
    return 0;
}
```

如果题目认为只有一个整数不构成连续的序列,则只需再判断 maxlength 是否等于 1

| 测试用例 | 输出 |
| --- | --- |
| 6  1 3 4 100 200 2 | 1 2 3 4 |
| 9  1 2 3 4 5 6 7 8 9 | 1 2 3 4 5 6 7 8 9 |
| 3  1 3 5 | 1 (根据题意也可以不输出数据) |
| 0 | (一个空行) |

4. 现在有一个文本编辑器，具有如下的操作：
MOVE k：将光标移动到第 k 个字符之前，如果 k=0，那么移动到文档开头
PRINT n：输出光标之后的 n 个字符
PREV：光标前移一位
NEXT：光标后移一位
请基于线性数据机构设计一套合理的算法，来实现这些操作，并且分析每个操作的性能。
假定：文本最大的长度为 L<10^9。

```
long long cursor_pos = 0;
char text = (char* )malloc(size * sizeof(char));
/* 单纯改变变量 cursor_pos 的值 */
void move(long long k){
    cursor_pos = k;
    if(cursor_pos <= 0) cursor_pos = 0;
    if(cursor_pos >= size) cursor_pos = size - 1;
}
/* 将字符一个一个输出 */
void print(long long n){
    for(int p = 0; cursor_pos + p < size && p < n; p += 1){
        cout << text[p];
    }
}

void prev(){
    cursor_pos -= 1;
    if(cursor_pos <= 0) cursor_pos = 0;
}

void next(){
    cursor_pos += 1;
    if(cursor_pos >= size) cursor_pos = size - 1;
}
```

认为字符从第 0 个开始计数，
Move,perv,next 都可以在 O(1)时间内完成
Print 可以在 O(n)时间内完成

5. 仍然是上面的题目背景，添加 2 个操作：
INSERT n, s：在当前光标之后插入长度为 n 的字符串 s
DELETE n：删除光标之后的 n 个字符
此时数据结构应当作出什么样的改变来适应这一变化？

用链表组织文本,插入删除时按需要增加或删除结点
每一个结点内可以存储至多固定大小的字符,使用现在所在的结点指针和节点内的偏移量给光标寻址

```
int bufsize = 1024;
class nodes{
     char* buf;
     int usedlength;
     node* perv;
     node* next;
};
void move(long long k){/* 从头遍历寻找位置,至多O(k)时间 */
   curr_node = first node;
    while(char_sum + curr_node.usedlength < k){
        char_sum += curr_node.usedlength;
        curr_node = curr_node.next;
    }
    curr_pos = k - char_sum;
}
Void print(long long n){/* 从当前块向后数,至多O(n)时间 */
   print chars after curr_pos;//输出当前块的后面
   while(sum + tmpnode.usedlength < n){
      print tmpnode.buf;
      tmpnode = tmpnode.next
   }
   print n - sum chars in tmpnode;//输出最后块的前面
}
Void next(){/* O(1) */
   if(curr_pos + 1 < currnode.usedlength){
      curr_pos += 1;
   }else{
      currnode = currnode.next;
      corrpos = 0;
   }
}
Prev 的实现类似(时间复杂度 O(1))
Insert(long long n, char* s){/* O(n) */
   split_node(curr_node, currpos);
   append(currnode.buf, s);
}
```

```
Delete(long long n){/* O(n) */
   delete_char(curr_node, offset);//在当前块删除字符
   While(del_sum + curr_node.next.usedlength <= n){
      Delete_node(curr_node.next);
   }
   delete_char(curr_node.next, n - del_sum);//删除后面块的前面
}
```

在块内的增删都可以用字符串的操作完成