☆查找

1. 设集合采用位向量存储结构，实现集合的删除、并和交算法。

这里的“位”是指数据位，不是数字位。

```
typedef struct Set{
    Char *Array;
    Int MaxSize;
}Set;

Void Delete(int index)
{
    Int r=index%8;
    Array[index/8]&=511-2^r;
}
Set SetUnion(Set s1,Set s2)
{
    Set RtSet;
    For(int i=0;i<MaxSize;i++)
    {
        RtSet.Array[i]=s1.Array[i]|s2.Array[i];
    }
    Return RtSet;
}
Set SetIntersection(Set s1,Set s2)
{
    Set RtSet;
    For(int i=0;i<MaxSize;i++)
    {
        RtSet.Array[i]=s1.Array[i]&s2.Array[i];
    }
    Return RtSet;

}
```

2. 设集合采用动态分配数组表示，写出集合的删除、交和并算法。

```
Typedef struct Set{
    Int *Array;
    Int MaxSize;
```

```
}Set;

Int Delete(int item)
{
    For(int i=0;i<MaxSize;i++)
    {
        If(Array[i]==item)
        {
            Array[i+1…MaxSize-1]→Array[i…MaxSize -2];
            MaxSize--;
            Return 1;
        }
    }
    Return 0;
}

Set SetUnion(Set s1,Set s2)
{
    Set RtSet=s1;
    For(int i=0;i<s2.MaxSize;i++)
    {
        Int t=s2.Array[i];
        For(int j=0;j<RtSet.MaxSize;j++)
        {
            If(RtSet.Array[j]==t)
                Break;
        }
        If(j==RtSet.MaxSize)
        {
            RtSet.Array[MaxSize]=t;
            RtSet.MaxSize++;
        }
    }
    Return RtSet;
}

Set SetIntersection(Set s1,Set s2)
{
    Set RtSet;
    For(int i=0;i<s1.MaxSize;i++)
    {
        For(int j=0;j<s2.MaxSize;j++)
        {
            If(s1.Array[i]==(t=s2.Array[j]))
```

```
            {
                RtSet.Array[MaxSize]=t;
                RtSet.MaxSize++;
                Break;
            }
        }
    }
    Return RtSet;
}
```

3. 设集合采用有序数组表示，写出集合的删除、成员测试、交和并算法，并分析其时间复杂性。

```
Typedef struct Set{
    Int *Array;        //sorted
    Int MaxSize;
}Set;

int Delete(int item)
{
    For(int i=0;i<MaxSize;i++)        //also may use binary search
    {
        If(Array[i]==item)            //成员测试
        {
            Array[i+1…MaxSize-1]→Array[i…MaxSize -2];
            MaxSize--;
            Return 1;
        }
    }
    Return 0;
}
Set SetUnion(Set s1,Set s2)
{
    Set RtSet=s1;
    For(int i=0,j=0;i<RtSet.MaxSize && j<s2.MaxSize; )
    {
        If(RtSet.Array[i]<s2.Array[j])
        {
            I++;
            Continue;
        }
```

```
            If(RtSet.Array[i]==s2.Array[j])
            {
                I++;
                J++;
                Continue;
            }
            If(RtSet.Array[i]>s2.Array[j])
            {
                RtSet.Array[i…MaxSize-1]→RtSet.Array[i+1…MaxSize];
                RtSet.Array[i]=s2.Array[j];
                RtSet.MaxSize++;
                I++;
                J++;
                Continue;
            }
        }
        If(j!=s2.MaxSize)
        {
            S2.Array[j…s2.MaxSize-1]→
RtSet.Array[RtSet.MaxSize…RtSet.MaxSize+s2.MaxSize-2-j];
RtSet.MaxSize+=s2.MaxSize-j;
        }
        Return RtSet;
}
Set SetIntersection(Set s1,Set s2)
{
        Set RtSet=s1;
        For(int i=0,j=0;i<RtSet.MaxSize || j<s2.MaxSize; )
        {
            If(RtSet.Array[i]<s2.Array[j])
            {
                RtSet.Array[i+1…MaxSize-1]→RetSet.Array[i…MaxSize-2];
                RtSet.MaxSize--;
                I++;
                Continue;
            }
            If(RtSet.Array[i]==s2.Array[j])
            {
                I++;
                J++;
                Continue;
            }
            If(RtSet.Array[i]>s2.Array[j])
            {
```

```
            J++;
            Continue;
        }
    }
    If(i!=RtSet.MaxSize)
    {
        Delete RtSet.Array[i…MaxSize-1];
        RtSet.MaxSize-=MaxSize-I;
    }
    Return RtSet;
}
```

删除：O(logn)，成员测试：O(logn)，交：O(n)，并：O(n)。

4. 设集合用二叉树表示，写出集合的交和并算法，并分析其时间复杂性。

因不是二叉排序树，很容易实现。略。

5. 设集合采用 Hash 表表示，用开放定址法处理冲突，写出集合的插入、删除、成员测试、交和并算法。

```
Status InsertHash(HashTable &H,Elemtype e)
{
    C=0;
    If(HashSearch(H,e.key,p,c))
        Return DUPLICATE;
    Else if(c<hashsize[H.sizeindex]/2)
    {
        H.elem[p]=e;
        H.count++;
        Return OK;
    }
    Else
        RecreateHashTable(H);
}
Status DeleteHash(HashTable &H,Elemtype e)
{
    C=0;
```

```
    If(HashSearch(H,e.key,p,c))
    {
        Delete H.Elem[p];
        Return OK;
    }
    Else
        Return NOTFOUND;
}
HashTable IntersectionHash(HashTable h1,HashTable h2)
{
    HashTable RtH;
    For(int i=0;i<h1.count;i++)
    {
        For(int j=0;j<h2.count;j++)
        {
            C=0;
            If(HashSearch(h1,h2.Elem[j],p,c))
            {
                InsertHash(RtH,h2.Elem[j]);
            }
        }
    }
    Return RtH;
}
HashTable UnionHash(HashTable h1,HashTable h2)
{
    HashTable RtH=h1;
    For(int i=0;i<h2.count;i++)
    {
        For(int j=0;j<RtH.count;j++)
        {
            C=0;
            If(HashSearch(h2,RtH.Elem[j],p,c))
                Break;
        }
        If(j==RtH.count)
        {
            InsertHash(RtH,h2.Elem[i]);
        }
    }
    Return RtH;
}
```

6. 设输入的关键字序列及其 Hash 函数为

关键字：22,41,53,33,46,30,13,01,67

Hash 函数：H = Key MOD 11

表长 = 11

试用线形探测法解决冲突。请将各关键字按输入次序填入 Hash 表中。

0	1	2	3	4	5	6	7	8	9	10
22										
22								41		
22								41	53	
22	33							41	53	
22	33	46						41	53	
22	33	46						41	53	30
22	33	46	13					41	53	30
22	33	46	13	01				41	53	30
22	33	46	13	01	67			41	53	30

7. 写出按中序遍历方法遍历一棵 m 阶 B 树的算法。

```
Typedef struct BTNode{
    Int keynum;
    Struct BTNode *parent;
    KeyType key[m+1];
    Struct BTNode *ptr[m+1];
}BTNode,*BTree;
```

```
Typedef struct{
    BTNode *pt;
    Int i;
    Int tag;
}Result;

Result SearchInOrder(BTNode *TNode,KeyType K,int i)
{
    If(TNode==NULL)
        Return NULL;
    If(i>=TNode.keynum)
        Return NULL;
    If((Result rt=SearchInOrder(TNode.ptr[i],K,i))!=NULL)
        Return rt;
    If(TNode.key[i]==K)
    {
        Return (TNode,I,1);
    }
    Return SearchInOrder(TNode,K,i+1);
}
```