



小白专场： 电话聊天狂人

浙江大学 陈 越

题意理解

输入样例:

4

13005711862 13588625832

13505711862 13088625832

13588625832 18087925832

15005713862 13588625832

13005711862 1

13588625832 3

13505711862 1

13088625832 1

18087925832 1

15005713862 1

输出样例:

13588625832 3

解法1 – 排序

第1步：读入最多 2×10^5 个电话号码，每个号码存为长度为11的字符串

第2步：按字符串非递减顺序排序

第3步：扫描有序数组，累计同号码出现的次数，并且更新最大次数



编程简单快捷



无法拓展解决动态插入的问题

解法2 – 直接映射

第1步：创建有 2×10^{10} 个单元的整数数组，保证每个电话号码对应唯一的单元下标；数组初始化为0

第2步：对读入的每个电话号码，找到以之为下标的单元，数值累计1次

第3步：顺序扫描数组，找出累计次数最多的单元

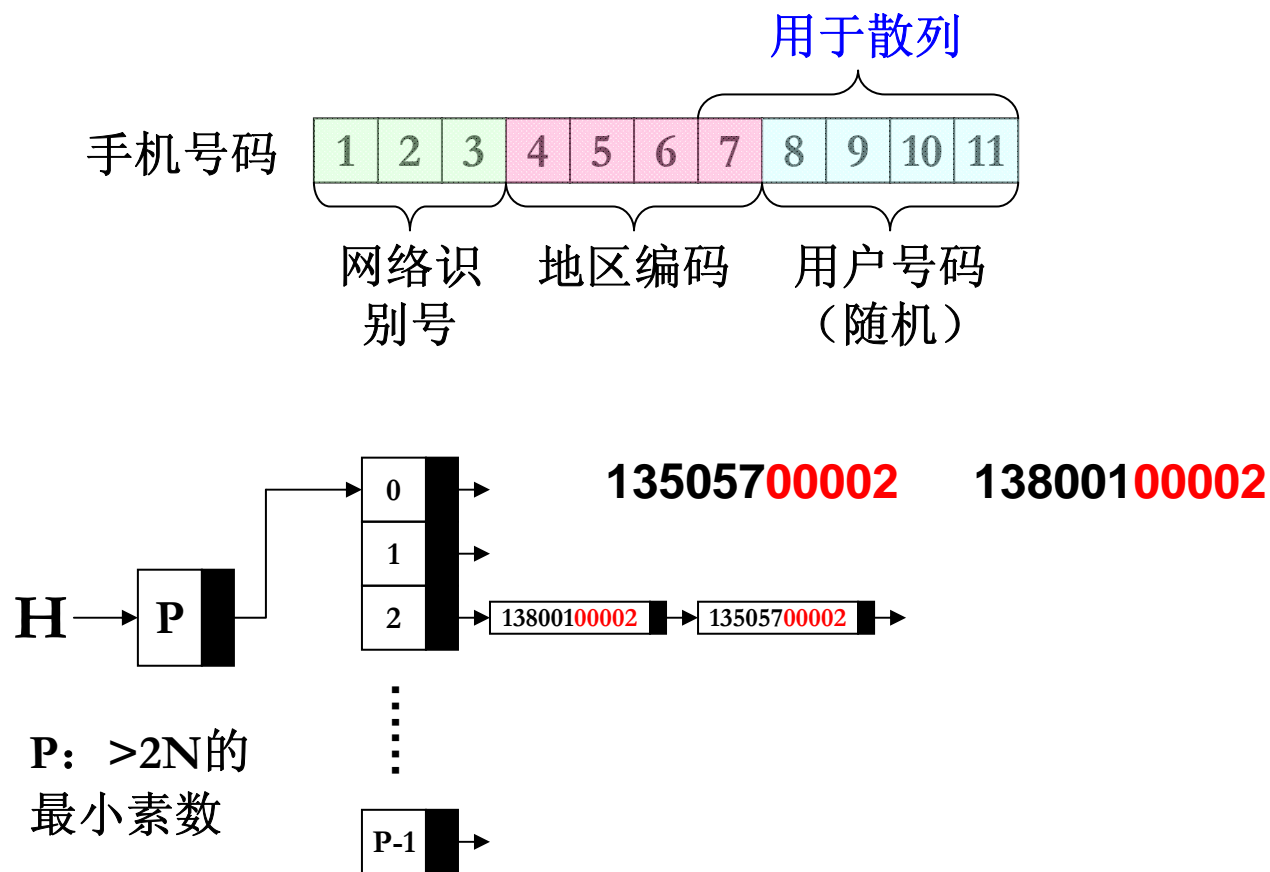
👍 编程简单快捷，动态插入快

👎 下标超过了 `unsigned long`

需要 $2 \times 10^{10} \times 2 \text{ bytes} \approx 37\text{GB}$

为了 2×10^5 个号码扫描 2×10^{10} 个单元

解法3 – 带智商的散列



程序框架搭建

```
int main()
{
    创建散列表;
    读入号码插入表中;
    扫描表输出狂人;
    return 0;
}
```

HashTable的定义

NextPrime

CreateTable

Hash

Find

Insert

```
int main()
{
    int N, i;
    ElementType Key;
    HashTable H;

    scanf("%d", &N);
    H = CreateTable(N*2); /* 创建一个散列表 */
    for (i=0; i<N; i++) {
        scanf("%s", Key); Insert( H, Key );
        scanf("%s", Key); Insert( H, Key );
    }
    ScanAndOutput( H );
    DestroyTable( H );
    return 0;
}
```

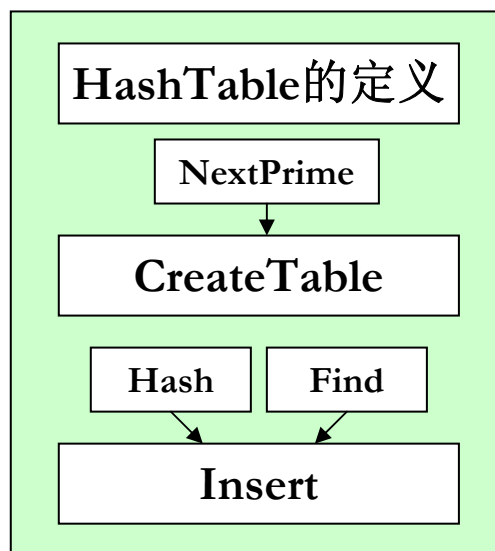
扫描整个散列表

更新最大通话次数;
更新最小号码 + 统计人数;

输出狂人

```
void ScanAndOutput( HashTable H )
{
    int i, MaxCnt = PCnt = 0;
    ElementType MinPhone;
    List Ptr;
    MinPhone[0] = '\0';
    for (i=0; i<H->TableSize; i++) { /* 扫描链表 */
        Ptr = H->Heads[i].Next;
        while (Ptr) {
            if (Ptr->Count > MaxCnt) { /* 更新最大通话次数 */
                MaxCnt = Ptr->Count;
                strcpy(MinPhone, Ptr->Data);
                PCnt = 1;
            }
            else if (Ptr->Count == MaxCnt) {
                PCnt ++; /* 狂人计数 */
                if ( strcmp(MinPhone, Ptr->Data)>0 )
                    strcpy(MinPhone, Ptr->Data); /* 更新狂人的最小手机号码 */
            }
            Ptr = Ptr->Next;
        }
    }
    printf("%s %d", MinPhone, MaxCnt);
    if ( PCnt > 1 ) printf(" %d", PCnt);
    printf("\n");
}
```

模块的引用与裁剪



```
#define KEYLENGTH 11 /* 关键词字符串的最大长度 */  
/* 关键词类型用字符串 */
```

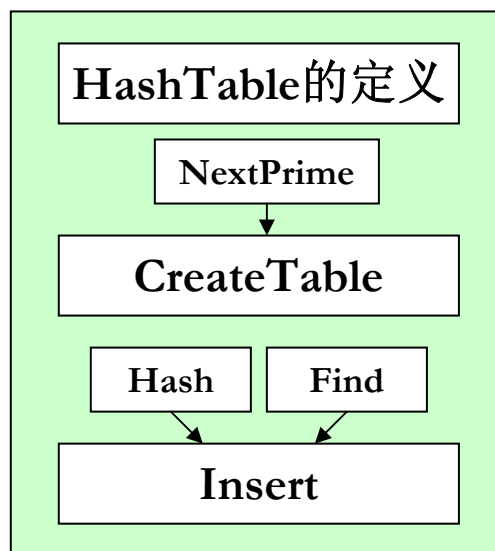
```
typedef char ElementType[KEYLENGTH+1];  
typedef int Index; /* 散列地址类型 */
```

```
typedef struct LNode *PtrToLNode;  
struct LNode {  
    ElementType Data;  
    PtrToLNode Next;  
    int Count;  
};
```

```
typedef PtrToLNode Position;  
typedef PtrToLNode List;
```

```
typedef struct TblNode *HashTable;  
struct TblNode { /* 散列表结点定义 */  
    int TableSize; /* 表的最大长度 */  
    List Heads; /* 指向链表头结点的数组 */  
};
```


模块的引用与裁剪



```
int Hash ( int Key, int P )
{ /* 除留余数法散列函数 */
    return Key%P;
}
```

```
#define MAXTABLESIZE 1000000
int NextPrime( int N )
{ /* 返回大于N且不超过MAXTABLESIZE的最小素数 */
    int i, p = (N%2)? N+2 : N+1; /*从大于N的下一个奇数开始 */
    while( p <= MAXTABLESIZE ) {
        for( i=(int)sqrt(p); i>2; i-- )
            if ( !(p%i) ) break; /* p不是素数 */
        if ( i==2 ) break; /* for正常结束,说明p是素数 */
        else p += 2; /* 否则试探下一个奇数 */
    }
    return p;
}
```

```
HashTable CreateTable( int TableSize )
{
    HashTable H;
    int i;
    H = (HashTable)malloc(sizeof(struct TblNode));
    H->TableSize = NextPrime(TableSize);
    H->Heads = (List)malloc(H->TableSize*sizeof(struct LNode));
    for( i=0; i<H->TableSize; i++ ) {
        H->Heads[i].Data[0] = '\0'; H->Heads[i].Next = NULL;
        H->Heads[i].Count = 0;
    }
    return H;
}
```

模块的引用与裁剪

```
Position Find( HashTable H, ElementType Key )
{
    Position P;
    Index Pos;

    /* 初始散列位置 */
    Pos = Hash(atoi(Key+KEYLENGTH-MAXD), H->TableSize);

    P = H->Heads[Pos].Next; /* 从该链表的第1个结点开始 */
    /* 当未到表尾, 并且key未找到时 */
    while( P && strcmp(P->Data, Key) )
        P = P->Next;

    return P; /* 此时P或者指向找到的结点, 或者为NULL */
}
```

模块的引用与裁剪

```
bool Insert( HashTable H, ElementType Key )
{
    Position P, NewCell;
    Index Pos;

    P = Find( H, Key );
    if ( !P ) { /* 关键词未找到, 可以插入 */
        NewCell = (Position)malloc(sizeof(struct LNode));
        strcpy(NewCell->Data, Key);
        NewCell->Count = 1;
        Pos = Hash(atoi(Key+KEYLENGTH-MAXD), H->TableSize);
        /* 将NewCell插入为H->Heads[Pos]链表的第1个结点 */
        NewCell->Next = H->Heads[Pos].Next;
        H->Heads[Pos].Next = NewCell;
        return true;
    }
    else { /* 关键词已存在 */
        P->Count++;
        return false;
    }
}
```