UVA240　Variable Radix Huffman Encoding

题目：可变基数霍夫曼编码

哈夫曼编码是一种最优编码方法。根据已知源字母表中字符出现的频率，将源字母表中字符编码为目标字母表中字符，最优的意思是编码信息的平均长度最小。在该问题中，你需要将 N 个大写字母（源字母 $S_1 \cdots S_N$，频率 $f_1 \cdots f_N$）转换成 R 进制数字（目标字母 $T_1 \cdots T_R$）。

当 R=2 时，编码过程分几个步骤，每个步骤中，有两个最低频率的源字符 S_1、S_2，合并成一个新的“组合字母”，频率为 S_1、S_2 的频率之和。如果最低频率和次低频率相等，则字母表中最早出现的字母被选中。经过一系列的步骤后，最后只剩两个字母合并，每次合并的字母分配一个目标字符，较低频率的分配 0，另一个分配 1。（如果一个合并中，每个字母有相同的频率，最早出现的分配 0，出于比较的目的，组合字母的值为合并中最早出现的字母的值。）源符号的最终编码由每次形成的目标字符组成。

目标字符以相反顺序连接，最终编码序列中第一个字符为分配给组合字母的最后一个目标字符。

下面的两个插图展示了 R = 2 的过程。

Symbol	Frequency
A	5
B	7
C	8
D	15

Pass 1: A and B grouped
Pass 2: {A,B} and C grouped
Pass 3: {A,B,C} and D grouped
Resulting codes: A=110, B=111, C=10, D=0
Avg. length=(3*5+3*7+2*8+1*15)/35=1.91

Symbol	Frequency
A	7
B	7
C	7
D	7

Pass 1: A and B grouped
Pass 2: C and D grouped
Pass 3: {A,B} and {C,D} grouped
Resulting codes: A=00, B=01, C=10, D=11
Avg. length=(2*7+2*7+2*7+2*7)/28=2.00

当 R>2 时，每一个步骤分配 R 个符号。由于每个步骤将 R 个字母或组合字母合并为一个组合字母，并且最后一次合并必须合并 R 个字母和组合字母，源字母必须包含 k*(R-1)+R 个字母， k 为整数。由于 N 可能不是很大，因此必须包括适当数量具有零频率的虚拟字母。这些虚拟的字母不包含在输出中。在进行比较时，虚拟字母晚于字母表中的任何字母。

霍夫曼编码的基本过程与 R = 2 情况相同。在每次合并中，将具有最低频率的 R 个字母合并，形成新的组合字母，其频率等于组中包括的字母频率的总和。被合并的字母被分配目标字母符号 0 到 R-1。0 被分配给具有最低频率的组合中的字母，1 被分配给下一个最低频率，等等。 如果组中的几个字母具有相同的频率，则字母表中最早的字母被分配较小的目标符号，依此类推。

下图说明了 R = 3 的过程：

Symbol	Frequency
A	5
B	7
C	8
D	15

Pass 1: ? (fictitious symbol), A and B are grouped
Pass 2: {?,A,B}, C and D are grouped
Resulting codes: A=11, B=12, C=0, D=2
Avg. length=(2*5+2*7+1*8+1*15)/35=1.34

输入：

输入将包含一个或多个数据集，每行一个。 每个数据集都包含一个整数值 R（2≤R≤10），

整数值 N（2≤R≤26）和整数频率 $f_1 \cdots f_N$，每个都在 1 到 999 之间。

整个输入的数据结束是 R 为 0；它不被认为是单独的数据集。

输出：

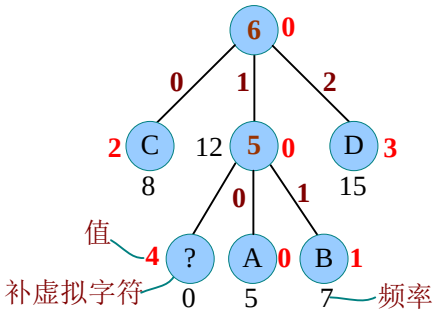
对于每个数据集，在一行上显示其编号（编号从 1 开始按顺序排列）和平均目标符号长度（四舍五入到小数点后两位）。 然后显示 N 个源字母和相应的霍夫曼代码，每行一个字母和代码。在每个测试用例后打印一个空行。

题解：

可变基哈夫曼编码，普通的哈夫曼编码为 R=2。

举例：

3 4 5 7 8 15　// R=3，N=4，A: 5　B: 7　C: 8　D: 15



算法设计：

（1）　先补充虚拟字符，使 N 满足 k*(R-1)+R，k 为整数，即(N-R)%(R-1)=0;

```
while((n-R)%(R-1)!=0)//补虚拟结点
    n++;
```

（2）　定义结点结构体，包含 3 个域：int frequency,va,id;//频率，优先值，序号

（3）　定义优先级：

```
bool operator <(const node &b) const {
    if(frequency==b.frequency)
        return va>b.va;
    return frequency>b.frequency;
}
```

（4）　将所有结点入优先队列

```
for(int i=0;i<n;i++)//所有结点入队
    Q.push(node(fre[i],i,i));
```

（5）　构建哈夫曼树

```
c=n;//新合成结点编号
int rec=0;//统计所有和值
while(Q.size()!=1)//剩余一个结点停止合并
{
    int sum=0,minva=n;
    for(int i=0;i<R;i++)
    {
        sum+=Q.top().frequency;//统计频率和
        minva=min(Q.top().va,minva);//求最小优先值
        father[Q.top().id]=c;//记录双亲
        code[Q.top().id]=i;//记录编码
        Q.pop(); //出队
    }
    Q.push(node(sum,minva,c));//新结点入队
    c++;
    rec+=sum;
```

```
            }
            c--;
            printf("Set %d; average length %0.2f\n",cas,1.0*rec/total);
```

（6） 进行哈夫曼编码

```
        for(int i=0;i<N;i++)
        {
            int cur=i;
            string s;
            while(cur!=c)
            {
                s.push_back(code[cur]+'0');
                cur=father[cur];
            }
            reverse(s.begin(),s.end());//翻转编码
            cout<<"        "<<char('A'+i)<<": "<<s<<endl;
        }
```

特别注意：需要补虚拟结点，值和存储序号不同。