小白专场：堆中的路径

题意理解

即最小堆

将一系列给定数字插入一个初始为空的小顶堆H[]。随后对任意给定的下标`i`，打印从H[i]到根结点的路径。

将输入5个数据

输入样例:

5 3　　要查询的个数

46 23 26 24 10　　输入构造堆的元素

5 4 3

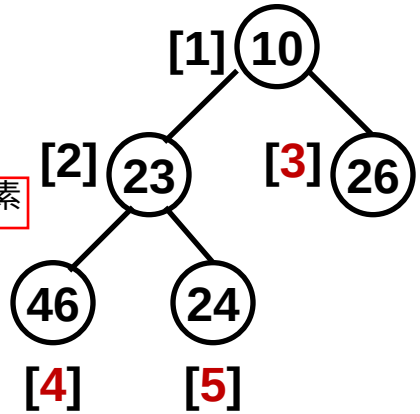
输入查询元素的编号（注意从1开始，没有0）

输出样例:

24 23 10

46 23 10

26 10



堆的表示及其操作

```
#define MAXN 1001
#define MINH -10001

int H[MAXN], size;

void Create ()
{
      size = 0;
      H[0] = MINH;
          /*设置“岗哨”*/
}
```

就不用判别下标i有没有越界：
MINH设定为比将要插入的元素都要小的值，这样如果插入的元素到根节点，一定会停下

```
void Insert ( int X )
{
    /* 将X插入H。这里省略检查堆是否已满的代码 */
          int i;

          for (i=++size; H[i/2] > X; i/=2)
                H[i] = H[i/2];
          H[i] = X;
}
```

堆的最大优点：可以很容易找到相关节点。
比如，编号i的节点（i>=1，0是空缺的），其父节点编号一定是i/2（退1法整除），左子节点一定是2i，右子节点一定是2i+1

主程序

```
int main()
{
   ●读入n和m
   ●根据输入序列建堆
   ●对m个要求：打印到
   根的路径

    return 0;
}
```

```
int main()
{    int n, m, x, i, j;

     scanf("%d %d", &n, &m);
     Create();                  /* 堆初始化 */
     for (i=0; i<n; i++) {      /*以逐个插入方式建堆 */
          scanf("%d", &x);
          Insert(x);
     }
     for (i=0; i<m; i++) {
          scanf("%d", &j);
          printf("%d", H[j]);
          while (j>1) {  /*沿根方向输出各结点*/
               j /= 2;
               printf(" %d", H[j]);
          }
          printf("\n");
     }
     return 0;
}
```

可见解题的关键是找到一个合适的存储方法，而不是寻找路径本身