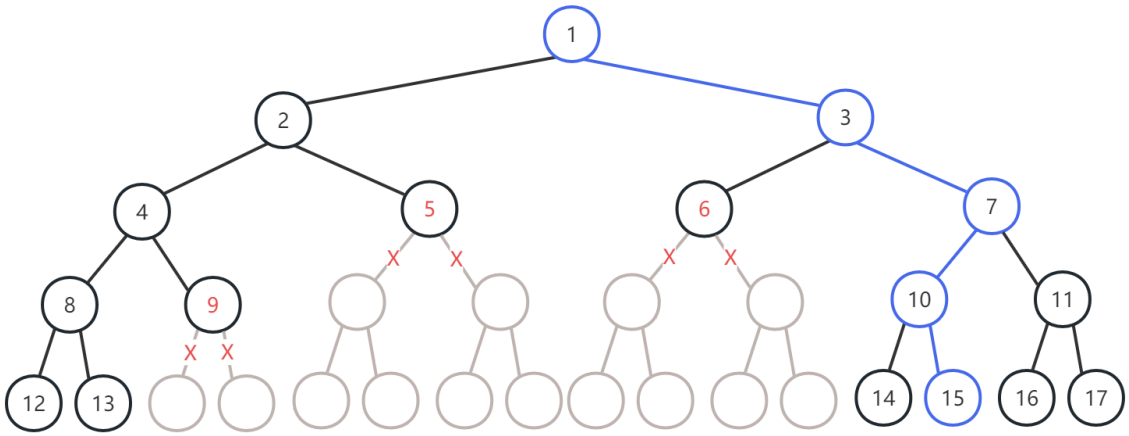
# OJ4 缺损二叉树

## 问题描述

输入给定一系列缺损点，将一棵无限深度的满二叉树进行重新编号，编号方式为从1开始逐行从左至右顺序编号，缺损点得到编号后，其子树被删去，后续编号中该子树的结点跳过不编号，编号后的树被称为“缺损二叉树”。例如下图为缺损点为{5，6，9}的缺损二叉树的一部分（下方可无限延长），红色数字标注了缺损点，其下方灰色子树被删去。对于输入给定的目标编号，输出缺损二叉树上根节点到目标结点的路径，该路径用经过的所有结点的编号表示。下图中，蓝色表示根结点到目标结点15的路径，路径用途径结点编号{1,3,7,10,15}表示。




## 输入格式

第一行为两个整数N和M，N表示缺损点数目，M表示目标点的数目。

第二行为N个缺损点的编号，按从小到大顺序排列。

第三行为M个目标点的编号，按从小到大顺序排列。

下方给出的输入样例表示缺损点为{5,6,9}，目标点为{3,9,15,22}。

## 输出格式

每一行输出根节点到一个目标点的最短路径，路径用途径节点编号表示。

如果该路径不存在，则输出0。

共输出M行。

## 输入样例

3 4
5 6 9
3 9 15 22

# 输出样例

1 3
1 2 4 9
1 3 7 10 15
1 3 7 10 14 22

# 提示

题目假定缺损点及目标点数目不超过100个，目标节点编号均小于$2^{32}$。

对本题，可以根据题目要求逐行构造该缺损二叉树。对于当前行，记录所有非缺损点，在下一行依次构造子节点，并排除缺损点，直到构造到目标节点为止。利用构造节点时记录的父亲节点编号，可以从目标节点搜索到回到根节点的路径，将该路径翻转，即可得到所求路径。

上述方法在目标节点编号过大时会占用过多内存，事实上，若某棵子树或树中的某连续部分不存在缺损点，可以不必将其每个节点均储存在内存中。你可以自行设计省略表示方法以满足性能要求。

# 思路

一些思考算法时的想法：

1. 由于缺损点最多100个，通过简单计算可知，该缺损二叉树的高度最多为133。
2. 构造树的时候，只需构造到节点为最大目标点即可停止，或者这一层全是缺损点，后面就没有树了。
3. 用一个数组存放每一层的起始和末尾，根据夹逼法，可以判断某一个缺损点或目标点位于的层数。

   对于这个层数类型的实现，可以使用结构体：

   ```
   typedef struct
   {
       long long start; // 某一层的头节点
       long long end;   // 某一层的尾节点
   } Floor;
   ```

   再另外一个二维数组`defect[134][100]`，存放每一行的缺损点及其个数。对于这个二维数组所代表的矩阵，每一行的第一个数存放这一层缺损点个数，后面依次从小到大存放缺损点数值。

   再用一个数组，存放每一个目标点及其所在层数。这个目标点类型可以使用另外一个结构体：

   ```
   typedef struct
   {
       long long data; // 某个目标点
       int floor;      // 该目标点在第几层
   } Node;
   ```

   此时遇上一个结构体不同的是：由于层数上限很低，只有133，远小于节点数值上限，因此用`int`类型可以节省空间。（如果忽略pragma对齐）
4. 如果已知某一行的起点`floor[f-1].start`和终点`floor[f-1].end`，以及上一行的缺损点数目`def_num`，可以确定下一行的起点和终点，起点为：

   ```
   floor[f].start = floor[f - 1].end + 1
   ```

   终点为：

```
floor[f].end = floor[f].start + (floor[f - 1].end - floor[f - 1].start + 1 - def_num) * 2 - 1
```

然后，将缺损点与该层的起点和终点进行比较，位于起点、终点之间的即为该层的缺损点，计入并更新该层的缺损点个数`def_num`。重复此操作可以构建出完整的树。

5. 向上回溯寻找某目标点的父亲节点的时候，这个过程比较复杂，不好讲解：

   对于当前位于层数`f`的节点`tar`，运算`(tar-floor[f].start)/2`，得到结果`cnt`，从上一层的起点`floor[f-1].start`开始，向后移动`cnt`步，但是途中没遇到一个缺损点就要额外移动一步，最后得到的数即为`tar`的父亲节点。将它重新赋给`tar`，依次循环向上查找，直到到达根节点1为止。

   此段代码实现如下：

```
while (cur_fl > 0)
{
    temp[cur_fl] = new_ta; //更新得到这一层的、将要向上寻找的节点
    long long cnt = (new_ta - floor[cur_fl].start) / 2; //预期向后移动步数
    int flag = 0;
    for (int j = 1; j <= df_fl[cur_fl - 1][0]; j++)
    {
        if (cnt + j <= df_fl[cur_fl - 1][j] - floor[cur_fl - 1].start)
        {
            break;
        }
        else
        {
            flag = j; //记录上一个循环中缺损点的数组索引
        }
    }
    new_ta = floor[cur_fl - 1].start + cnt + flag;
    cur_fl--; //继续到上一层
}
```

6. 还需要考虑某些特殊情况：构造到某一层的时候全是缺损点，标志为`floor[f].end - floor[f].start + 1 == df_fl[f][0]`，缺损二叉树到此停止，后面不会再有节点，如果有目标点大于最后一个构造的节点，那么它到根结点的路径为0。

   如果1为缺损点，那么该二叉树只有1，目标点1的路径为1，其他所有目标点回到1的路径为0。

综合以上想法，就可以完整的解决这道题了~

# 代码

```
#include <stdio.h>

typedef struct
{
    long long start; // 某一层的头节点
    long long end;   // 某一层的尾节点
} Floor;

typedef struct
{
    long long data; // 某个目标点
```

```
    int floor;      // 该目标点在第几层
} Node;

int main()
{
    int N, M;                          // 缺损点数目和目标点数目
    long long defect[100];             // 记录缺损点及其所在层数
    long long df_fl[134][101] = {0}; // 记录每一层缺损点及其个数
    Node target[100];                  // 记录目标点及其所在层数
    Floor floor[134];                  // 经计算可得缺损树最多133层，第0层为1
    floor[0].start = floor[0].end = 1;

    scanf("%d%d", &N, &M);
    for (int i = 0; i < N; i++)
    {
        scanf("%lld", &defect[i]);
    }
    for (int i = 0; i < M; i++)
    {
        scanf("%lld", &target[i].data);
    }

    if (defect[0] == 1) // 如果根节点1缺损，那么后续全都不存在了
    {
        for (int i = 0; i < M; i++)
        {
            if (target[i].data == 1)
                printf("%d\n", 1);
            else
                printf("%d\n", 0);
        }
        return 0;
    }

    // 开始构造缺损树
    int cur_fl = 1; // 当前层数
    int df_num = 0; // 当前层的所有缺损点数
    int flag_d = 0; // 当前将要遇到哪个缺损点defect[flag_d]
    int flag_t = 0; // 当前将要遇到哪个目标点target[flag_t].data
    long long last_nd;

    if (target[0].data == 1)
    {
        target[0].floor = 0;
        flag_t++;
    }

    for (cur_fl = 1; cur_fl < 134; cur_fl++)
    {
        floor[cur_fl].start = floor[cur_fl - 1].end + 1;
                                                // 确定起点
        floor[cur_fl].end = floor[cur_fl].start + (floor[cur_fl - 1].end -
floor[cur_fl - 1].start - df_num) * 2 + 1; // 确定终点
        df_num = 0;
                                               // 重置这一行的缺损点数

        int i;
        for (i = flag_d; i < N; i++)
```

```
        {
            // 判断这一行有几个缺损点
            if (floor[cur_fl].start <= defect[i] && defect[i] <= floor[cur_fl].end)
            {
                df_fl[cur_fl][i - flag_d + 1] = defect[i];
                df_num++;
            }
            else
                break;
        }
        // 此时，这一行的缺损点数目为df_num
        df_fl[cur_fl][0] = df_num; // 这一行的缺损点个数存放在数组第一位
        flag_d = i;

        int j;
        for (j = flag_t; j < M; j++)
        {
            // 记录该层所有目标点的层数
            if (floor[cur_fl].start <= target[j].data && target[j].data <= floor[cur_fl].end)
            {
                target[j].floor = cur_fl;
            }
            else
                break;
        }
        flag_t = j;

        if (floor[cur_fl].end >= target[M - 1].data || floor[cur_fl].end - floor[cur_fl].start + 1 == df_num)
        {
            // 如果已经构造到最后一个目标点，或者这一行全是缺损点，停止构造
            last_nd = floor[cur_fl].end; // 记录构造到的最后一个节点
            break;
        }
    }

    // 开始搜索回到根结点的路径
    for (int i = 0; i < M; i++)
    {
        if (target[i].data > last_nd)
        {
            printf("%d\n", 0);
            continue;
        } // 路径不存在的情况，直接输出0

        long long temp[134] = {1};
        // 该目标点层数为target[i].floor
        // 在df_fl[floor]中还要判断可能碰到的缺损点
        int cur_fl = target[i].floor;      // 当前层数
        long long new_ta = target[i].data; // 当前节点
        while (cur_fl > 0)
        {
            temp[cur_fl] = new_ta;
            long long cnt = (new_ta - floor[cur_fl].start) / 2;
            int flag = 0;
            for (int j = 1; j <= df_fl[cur_fl - 1][0]; j++)
```

```
            {
                if (cnt + j <= df_fl[cur_fl - 1][j] - floor[cur_fl -
1].start)
                {
                    break;
                }
                else
                {
                    flag = j;
                }
            }
            new_ta = floor[cur_fl - 1].start + cnt + flag;
            cur_fl--;
        }
        printf("%d", 1); // 最先输出根节点
        for (int j = 1; j <= target[i].floor; j++)
        {
            printf(" %lld", temp[j]);
        }
        printf("\n");
    }

    return 0;
}
```

附上作者的通过图片~

**提交详情（缺损二叉树）**

提交者: 2322022010597　　创建时间: 2023-11-15 18:03:35

**运行结果**　　**分数** 100.00

| # | 状态 | 时间 | 内存 |
|---|---|---|---|
| 1 | Accepted | 0 ms | 864 KB |
| 2 | Accepted | 0 ms | 872 KB |
| 3 | Accepted | 0 ms | 872 KB |
| 4 | Accepted | 0 ms | 868 KB |
| 5 | Accepted | 0 ms | 868 KB |
| 6 | Accepted | 0 ms | 868 KB |
| 7 | Accepted | 0 ms | 872 KB |
| 8 | Accepted | 0 ms | 868 KB |
| 9 | Accepted | 0 ms | 868 KB |
| 10 | Accepted | 0 ms | 864 KB |