中山大学数据科学与计算机学院本科生实验报告

课程名称：**算法设计与分析**　　　任课教师：**张子臻**

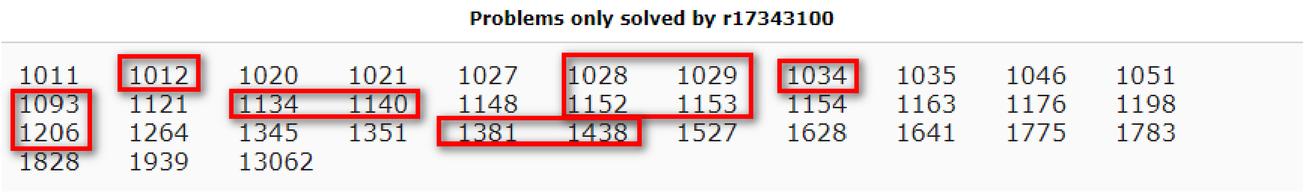
年级	**2017级**	**专业（方向）**	**软件工程**
学号	17343100	姓名	仕润昊
电话	13280152626	Email	1056627011@qq.com
开始日期	2019/4/3	完成日期	2019/4/10

完成情况——12题

本次实验共完成12道题

1152 1153 1093 1134 1140 1438
1028 1029 1381 1206 1012 1034

sicily截图如下



Problems only solved by r17343100

```
1011  1012  1020  1021  1027  1028  1029  1034  1035  1046  1051
1093  1121  1134  1140  1148  1152  1153  1154  1163  1176  1198
1206  1264  1345  1351  1381  1438  1527  1628  1641  1775  1783
1828  1939  13062
```

1.实验题目

1152 简单的马周游问题

题意：在给定大小的棋盘里，求一条从特定的点出发的一条马周游路线。马周游路线指的是“马”经过棋盘上所有格子，且每个格子只访问一次的路线。

约束：1152限制棋盘大小为56，*1153为88*

1153 马的周游问题

题意：在给定大小的棋盘里，求一条从特定的点出发的一条马周游路线。马周游路线指的是“马”经过棋盘上所有格子，且每个格子只访问一次的路线。

约束：1152限制棋盘大小为5*6，1153为8*8

1093 Air Express

题意：给出4个重量区间和各个区间的单位重量运输价，问对于一个背包，需要添加多少重量使得运输代价最小。

约束：所有的数都为正数，且不超过1000

1134 积木分发

题意：n个小朋友需要积木完成任务，其中第i个小朋友现手上有ai个积木，还需要bi个积木才能完成任务。现在老师手上有s块积木，她可以将积木分给其中一个小朋友让他先完成任务后，再将他手上的所有积木回收。问是否能回收所有小朋友的积木。

约束：n<=10^5,ai,bi<=10^9,s<=10^6

1140 国王的遗产

题意：

有一个国王拥有一个n个金块组成的树，在他死后由他的k个儿子轮流分金块。

每个人可以选择一条边将它断开，然后选择金块数量少的那一块，如果金块数量相同，则选择剩余编号小的金块所在的那一块。

约束：n<=3*10^4,k<=10^2

1438 Shopaholic

题意：有个购物狂在商场中购物，他想要买n个商品，商场在做促销，每买三个商品，最便宜的一个可以免费，现在问最多可以省多少钱。

约束：n<=*210^4 price*<=210^4

1028 Hanoi Tower Sequence

题意：

汉诺塔问题，将从上到下的方向从小到大排列的盘子，从第一个柱子中移动到另一个柱子，其中可以借助第三个柱子，并且每次移动后每根柱子从上到下方向柱子大小总是从小到大的。

现在给定了一个移动规则，问第p个移动的盘子编号为多少

约束：1<=p<=10^100

1029 Rabbit

题意：

开始有一对成年兔子

每对成年兔子每个月产生一对小兔子

每只小兔子经过m个月变成成年兔子

问经过d个月后有多少兔子

约束：1 <= m <= 10, 1 <= d <= 100

1381 a*b

题意：

题目简单暴力，就是要实现高精度与低精度的乘法

约束：0 <= a <= 10^100, 0 <= b <= 10^4

1206 Stacking Cylinders

题意：

给出最底层的n个圆柱的位置，求最顶层的圆柱的位置

圆柱半径都为1

约束：1 <= n <= 10

1012 Stacking Cylinders

题意：

给出最底层的n个圆柱的位置，求最顶层的圆柱的位置

圆柱半径都为1

约束：1 <= n <= 10

1034 Forest

题意：

给一个n个点，m条有向边的图

问这个图是不是森林，如果是输出这个森林的最大深度和最大宽度

森林的定义为：一个有向图，没有指向同一个节点的边也没有重边

对于入度为0的点，被称为为根，它们处于第0层

对于一条有向边，u->v，如果u处于第k层，那么v处于第k+1层

最大深度指的是存在点最深一层

最大宽度是指处于同一层的最多点数

限制：1<=n<=100, 0<=m<=100,m<=n*n

2.实验目的

练习深度优先搜索、回溯法和贪心算法，更熟练的掌握搜索算法、回溯法和贪心算法。

3.程序设计

1152 简单的马周游问题

深搜即可，枚举所有可能的马行走路线，直到找到一条能完成马周游的路线，即回溯法。

深搜过程如下

```
void dfs(int x, int y, int cnt) {
    if (cnt == N * M) {
        showTrail();
        flag = true;
        return;
    }
    if (flag) {
        return;
    }
    for (int i = 0; i<8; ++i) {
        int move[8][2] = { { 1,-2 },{ 2,-1 },{ 2,1 },{ 1,2 },{ -1,2 },{ -2,1 },{ -2,-1 },{ -1,-2
} };
        int x1 = x + move[i][0];
        int y1 = y + move[i][1];
        if (0 == chess[x1][y1] && x1 >= 0 && x1<N&&y1 >= 0 && y1<M) {
            chess[x1][y1] = cnt + 1;
            dfs(x1, y1, cnt + 1);
            chess[x1][y1] = 0;
        }
    }
    return;
}
```

1153 马的周游问题

因为棋盘变为8*8，如果直接暴力搜索会超时，所以需要一定的剪枝，我们先走后序分支少的子树，即先加一个判断。

我们采用先枚举所有下一可行分支的所有可行分支的数目，然后进行排序，先走拥有较少可行分支的下一步可行分支。

```
    vector<next_node> order;
    for (int i = 0; i < 8; i++) {
        if (isValid(n, i)) {
            next_node next;
            next.index = get_n(n, i);
            next.count = get_c(next.index);
            if (next.count != 0) {
                order.push_back(next);
            }
        }
    }
    for (int i = 0; i < order.size(); i++) {
        for (int j = 0; j < order.size()-1; j++) {
            if (order[j].count > order[j + 1].count) {
                swap(order[j], order[j + 1]);
            }
        }
    }
    for (int i = 0; i < order.size(); i++) {
        dfs(order[i].index);
        if (count_order >= 64) {
            return;
        }
        visited[order[i].index] = false;
        count_order--;
    }
```

1093 Air Express

本题目就是简单的枚举，每次最多枚举四种情况，比较得出哪种情况花费最少。

```
if (f == 0) {
    temp_cost[0] = cost[0] * n;
    for (int i = 1; i < 4; i++) {
        add_pounds[i] = weight[i - 1] + 1 - n;
        temp_cost[i] = cost[i] * (weight[i-1] + 1);
    }
    int min = temp_cost[0];
    int index = 0;
    for (int i = 1; i < 4; i++) {
        if (min > temp_cost[i]) {
            min = temp_cost[i];
            index = i;
        }
    }
    print(n, min, add_pounds[index]);
}
else if (f == 1) {
    temp_cost[1] = cost[1] * n;
    for (int i = 2; i < 4; i++) {
        add_pounds[i] = weight[i - 1] + 1 - n;
        temp_cost[i] = cost[i] * (weight[i - 1] + 1);
    }
    int min = temp_cost[1];
    int index = 1;
    for (int i = 2; i < 4; i++) {
        if (min > temp_cost[i]) {
            min = temp_cost[i];
            index = i;
        }
    }
    print(n, min, add_pounds[index]);
}
else if (f == 2) {
    temp_cost[2] = cost[2] * n;
    for (int i = 3; i < 4; i++) {

        add_pounds[i] = weight[i - 1] + 1 - n;
        temp_cost[i] = cost[i] * (weight[i - 1] + 1);
    }
    int min = temp_cost[2];
    int index = 2;
    for (int i = 3; i < 4; i++) {
        if (min > temp_cost[i]) {
            min = temp_cost[i];
            index = i;
        }
    }
    print(n, min, add_pounds[index]);
}
else{
    print(n, cost[3] * n, 0);
}
```

1134 积木分发

排序之后贪心即可。按bi从小到大排序，模拟分发过程就可以了。

```
int main() {
    ios::sync_with_stdio(false);
    int n;
    while (true) {
        cin >> n;
        if (n == 0) {
            return 0;
        }
        long long s;
        cin >> s;
        for (int i = 0; i < n; i++) {
            cin >> puz[i].has_get >> puz[i].to_get;
        }
        bool flag = true;
        sort(puz, puz + n,Cmp());
        for (int i = 0; i < n; i++) {
            if (s < puz[i].to_get) {
                cout << "NO" << endl;
                flag = false;
                break;
            }
            s += puz[i].has_get;
        }
        if (flag) {
            cout << "YES" << endl;
        }
    }
}
```

1140 国王的遗产

整个链可以看做是一棵树，我们可以把这个题看做树的分治。

定义树的结构如下

```
struct node {
    int in;
    vector<int> out;
    int root_count;     //以该节点为根的子树的节点数目
    bool exist;         // 该节点是否存在
    int min_index;     // 以该节点为根的子树的最小节点
    node() {
        in = -1;
        root_count = -1;
        exist = false;
        min_index = -1;
    }
};
```

本题的思路如下

1. 使用递归算法递归的算出每一棵子树包含的节点数目，递归的时候保留子树状态，递归一次需要的时间复杂度为O（n）
2. 递归的计算每棵子树的最小节点，并记录下来，防止在循环里面重复计算。
3. 穷举去掉每条变后两棵子树的节点数和最小节点，根据规则选出要保留的子树。
4. 对1,2,3步循环k-1次，每次输出子树的节点数。
5. 最后输出剩下子树的节点数。

如果不把子树的节点数和最小节点记录下来，每次循环都穷举搜索将会超时（(3000+x)*(3000+y)*100），所以还是需要一点剪枝的。

1438 Shopaholic

可以直接使用贪心算法，将所有的花费从大到小排序，然后从第三个数开始，隔两个取一个，将所有取出数字相加即为最省结果。

可以直接使用sort函数，将结构体cmp（也可以用greater）作为参数。

```
bool cmp(int a, int b)
{
    return a>b;
}
```

```
sort(v, v + n ,cmp);
int sum = 0;
if (n < 3) {
    cout << 0 << endl;
    continue;
}
for (int i = 2; i < n; i += 3) {
    sum += v[i];
}
cout << sum << endl;
```

1028 Hanoi Tower Sequence

汉诺塔问题所需要的步数为

F[n] = F[n-1] * 2 + 1

F[1] = 1

可以得到通项公式为F[n] = 2 ^ n – 1

这启示着我们可以从二进制上考虑

移动序列：

1

1 2 1

1 2 1 3 1 2 1

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

F((0001)2) = 1 F((0010)2) = 2 F((0011)2) = 1 F((0100)2) = 3...

观察发现，二进制中有从低位数起连续的0的数量加1就是当前移动的盘子的编号

【即第p项的值为p能被2整除的次数加一】

```
    while (T--) {
        C++;
        string str;
        cin >> str;
        int k = 2;
        int count = 1;
        //str是大数，求str%k,ans即为结果
        int ans = 0;
        for (int i = str.length()-1; i >= 0; i--) {
            a[i] = int(str[str.length()-i-1] - '0');
        }
        int l = str.length()-1;
        while (a[0] % 2 == 0) {
            int rem = 0;
            while (a[l] == 0) l--;

            for (int i = l; i >= 0; i--) {
                a[i] = a[i] + rem * 10;
                rem = a[i] % 2;
                a[i] /= 2;
            }
            count++;
        }
        cout << "Case " << C << ": " << count << endl;
        if (T > 0) {
            cout << endl;
        }
    }
```

1029 Rabbit

这个数据量可以直接用暴力方法计算，所以最后实现高精度加法即可。

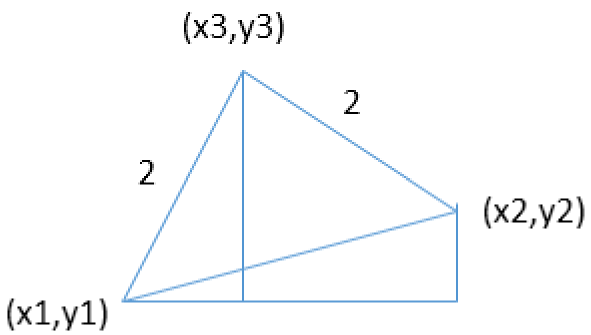
```
string get_sum(string adult, string children) {
    reverse(adult.begin(), adult.end());
    reverse(children.begin(), children.end());
    string temp = (adult.length() > children.length() ? adult : children);
    if (children.length() < adult.length()) {
        int c_l = children.length();
        for (int i = c_l; i < adult.length(); i++) {
            children += '0';
        }
    }
    else if (children.length() > adult.length()) {
        int a_l = adult.length();
        for (int i = a_l; i < children.length(); i++) {
            adult += '0';
        }
    }
    children += '0';
    adult += '0';
    temp += '0';
    int c = 0;
    for (int i = 0; i < temp.length(); i++) {
        int cur_sum = children[i] + adult[i] - '0' - '0' + c;
        c = (cur_sum - 9 > 0 ? 1 : 0);
        if (cur_sum > 9) {
            cur_sum -= 10;
        }
        temp[i] = cur_sum + '0';
    }
    int count = temp.length();
    while ( count > 0 && temp[count-1] == '0') {
        count--;
    }
    string sum = temp.substr(0, count);
    reverse(sum.begin(), sum.end());
    return sum;

}
```

1381 a*b

本题是高精度乘法，可以通过上一题的高精度加法实现

```
string get_single_mul(string a, char b) {
    string sum = "0";
    int n = b - '0';
    for (int i = 0; i < n; i++) {
        sum = get_sum(sum, a);
    }
    return sum;
}

string get_mul(string a, string b) {
    string sum = "0";
    string mul = "0";
    reverse(b.begin(), b.end());
    for (int i = 0; i < b.length(); i++) {
        mul = get_single_mul(a, b[i]);
        for (int j = 0; j < i; j++) {
            mul += "0";
        }
        sum = get_sum(sum, mul);
    }
    return sum;
}
```

1206 Stacking Cylinders

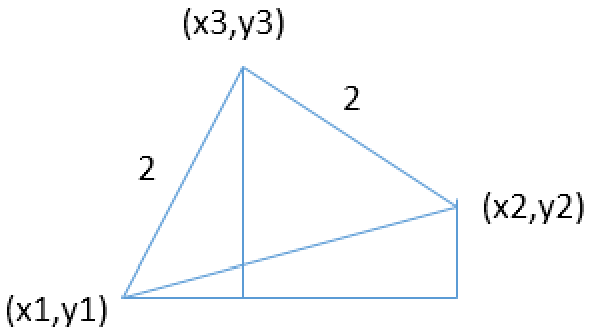


本题非常简单，只需要从第一层到最高层挨着计算圆心位置即可。根据几何知识，我们可以根据角度求得上一层的圆心横纵坐标，利用stl里面的三角函数即可求得。

```
        for (int i = n - 1; i > 0; i--) {
            vector<cycle> arr_temp;
            for (int k = 0; k < i; k++) {
                cycle node1, node2, node;
                node1 = arr[k];
                node2 = arr[k + 1];
                double a = atan(abs(node1.y - node2.y) / abs(node1.x - node2.x));
                double b = acos(sqrt((node2.y - node1.y)*(node2.y - node1.y) + (node2.x -
node1.x)*(node2.x - node1.x)) / 4);
                a = a + b;
                if (node2.y > node1.y) {
                    node.x = node1.x + 2 * cos(a);
                    node.y = node1.y + 2 * sin(a);
                }
                else if (node2.y < node1.y) {
                    node.x = node2.x - 2 * cos(a);
                    node.y = node2.y + 2 * sin(a);
                }
                else {
                    node.x = node1.x + 2 * cos(b);
                    node.y = node1.y + 2 * sin(b);
                }
                arr_temp.push_back(node);
            }
            arr.clear();
            arr = arr_temp;
        }
```

1012 Stacking Cylinders

本题跟上一题相同



本题非常简单，只需要从第一层到最高层挨着计算圆心位置即可。根据几何知识，我们可以根据角度求得上一层的圆心横纵坐标，利用stl里面的三角函数即可求得。

```
        for (int i = n - 1; i > 0; i--) {
            vector<cycle> arr_temp;
            for (int k = 0; k < i; k++) {
                cycle node1, node2, node;
                node1 = arr[k];
                node2 = arr[k + 1];

                double a = atan(abs(node1.y - node2.y) / abs(node1.x - node2.x));
                double b = acos(sqrt((node2.y - node1.y)*(node2.y - node1.y) + (node2.x - node1.x)*(node2.x - node1.x)) / 4);
                a = a + b;
                if (node2.y > node1.y) {
                    node.x = node1.x + 2 * cos(a);
                    node.y = node1.y + 2 * sin(a);
                }
                else if (node2.y < node1.y) {
                    node.x = node2.x - 2 * cos(a);
                    node.y = node2.y + 2 * sin(a);
                }
                else {
                    node.x = node1.x + 2 * cos(b);
                    node.y = node1.y + 2 * sin(b);
                }
                arr_temp.push_back(node);
            }
            arr.clear();
            arr = arr_temp;
        }
```

1034 Forest

树的深度我是通过DFS计算的，通过DFS可以判断是否有回路，如果没有回路再计算树的最大宽度，暴力搜索即可。

在计算深度的时候，我们把计算好的节点的深度保存下来，防止以后的重复计算。

```
void get_depth(int index) {
    if (tree_node[index].in == -1) {
        tree_node[index].depth = 0;
        return;
    }
    if (tree_node[index].depth != -1) {
        return;
    }
    if (tree_node[tree_node[index].in].depth == -1) {
        get_depth(tree_node[index].in);
    }
    tree_node[index].depth = tree_node[tree_node[index].in].depth + 1;


}
```

根据求好的深度，我们再遍历树求得最大深度

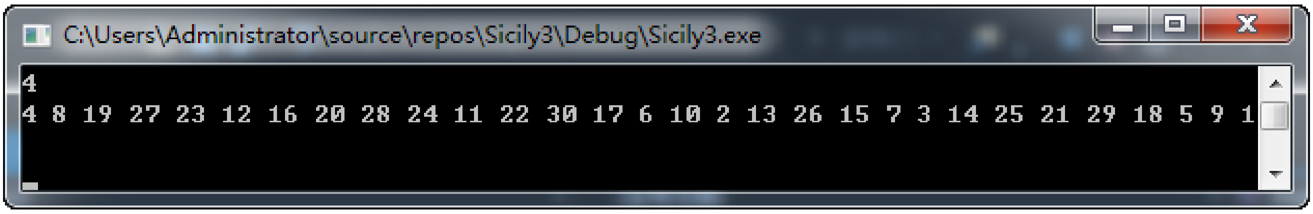
```
int dfs_depth(int index) {
    if (valid_flag == false) {

        return -1;
    }
    if (visited[index]) {
        valid_flag = false;
        return -1;
    }
    visited[index] = true;
    if (tree_node[index].out.size() == 0) {
        return 0;
    }
    int max_depth = dfs_depth(tree_node[index].out[0]) + 1;
    int visit_node = tree_node[index].out[0];
    for (int i = 1; i < tree_node[index].out.size(); i++) {
        max_depth = max(max_depth, dfs_depth(tree_node[index].out[i]) + 1);
    }
    return max_depth;
}
```

下面是计算最大宽度的过程，width[i]代表深度为i时的宽度，遍历树即可。

```
        if (valid_flag) {
            int width[200];
            for (int i = 0; i <= max_depth; i++) {
                width[i] = 0;
            }
            for (int i = 1; i <= n; i++) {
                get_depth(i);
            }
            for (int i = 1; i <= n; i++) {
                width[tree_node[i].depth]++;
            }
            int max_width = 0;
            for (int i = 0; i <= max_depth; i++) {
                max_width = max(max_width, width[i]);
            }
            cout << max_depth << " " << max_width << endl;
        }
        else {
            cout << "INVALID" << endl;
        }
```

4.程序运行与测试

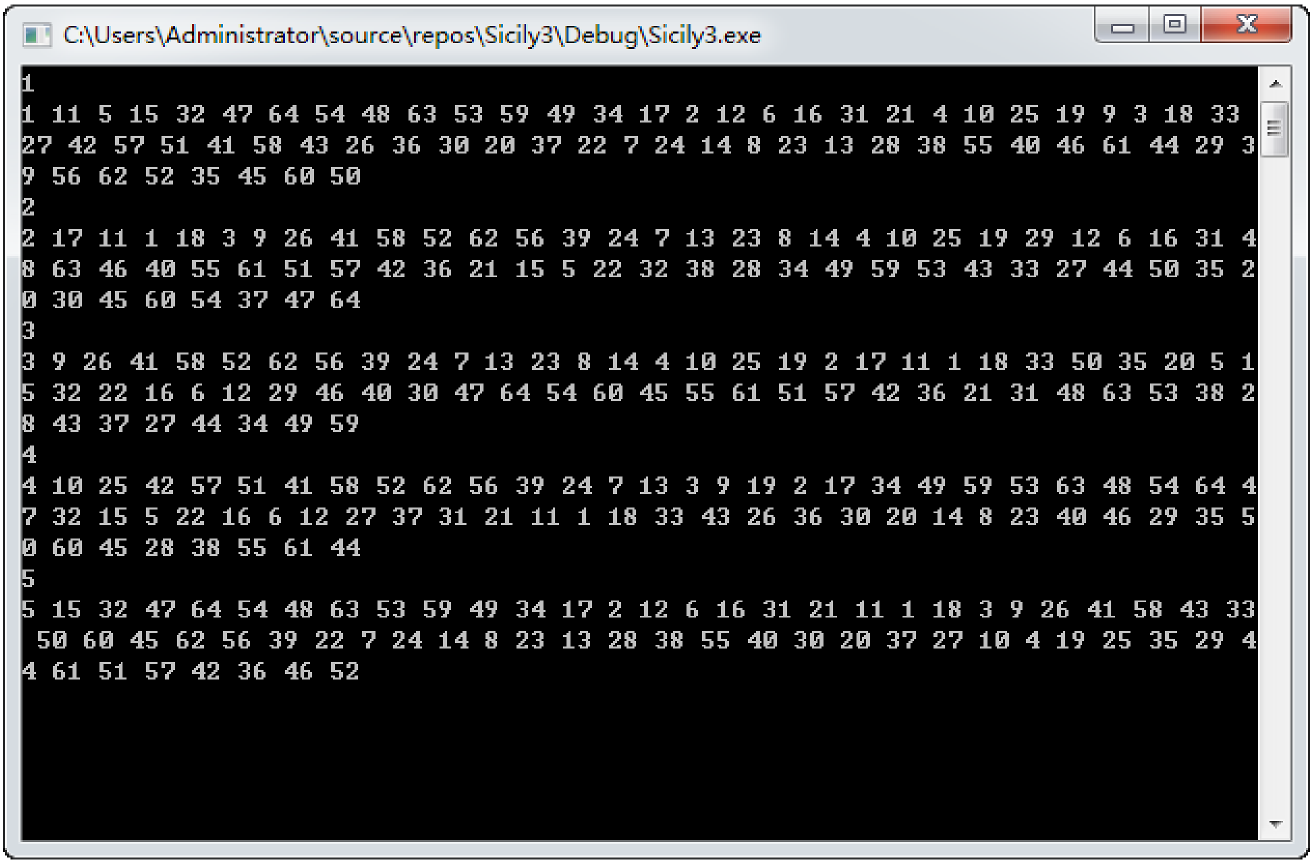
1152 简单的马周游问题



C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

```
4
4 8 19 27 23 12 16 20 28 24 11 22 30 17 6 10 2 13 26 15 7 3 14 25 21 29 18 5 9 1
```

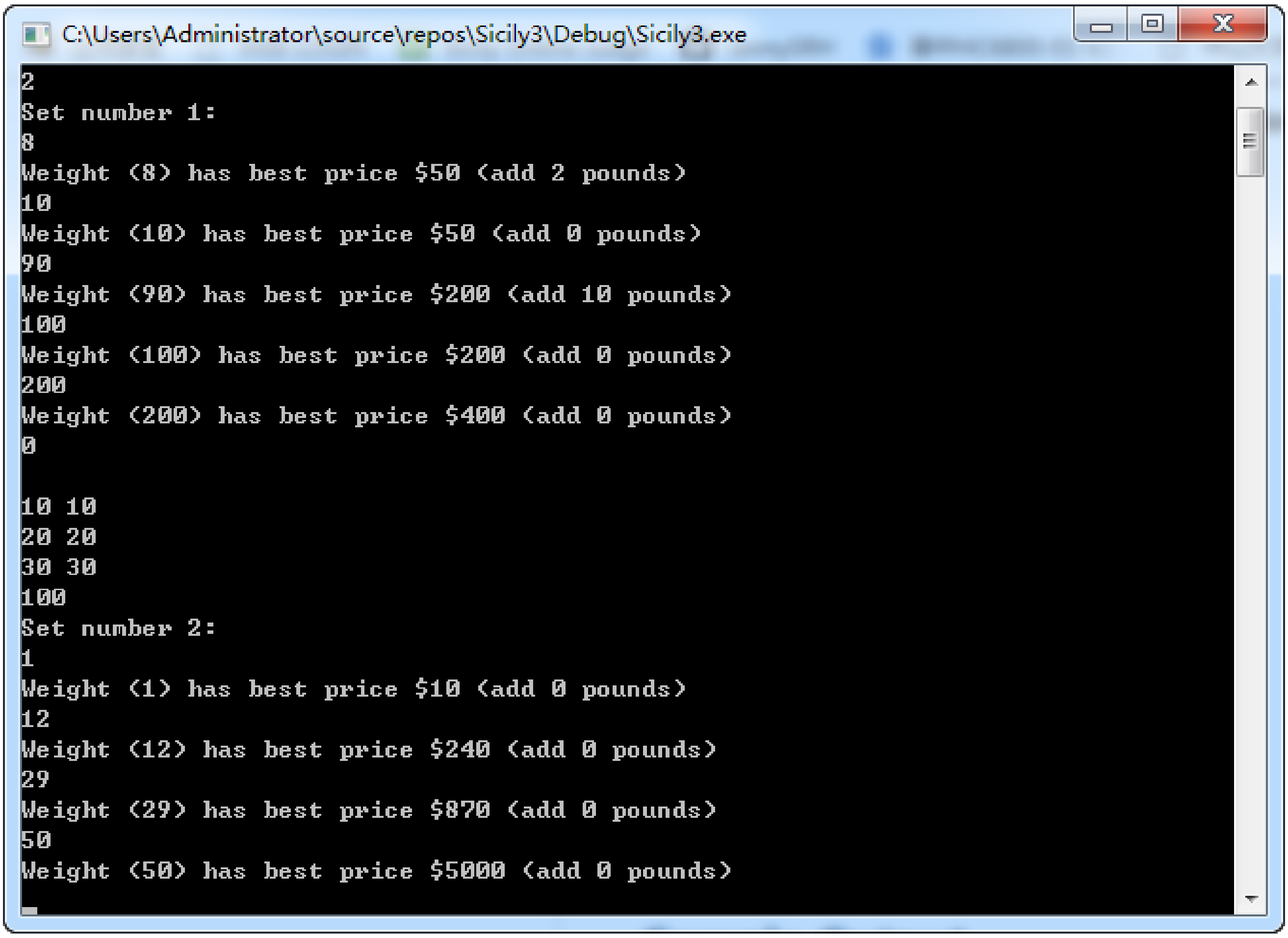
1153 马的周游问题



C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

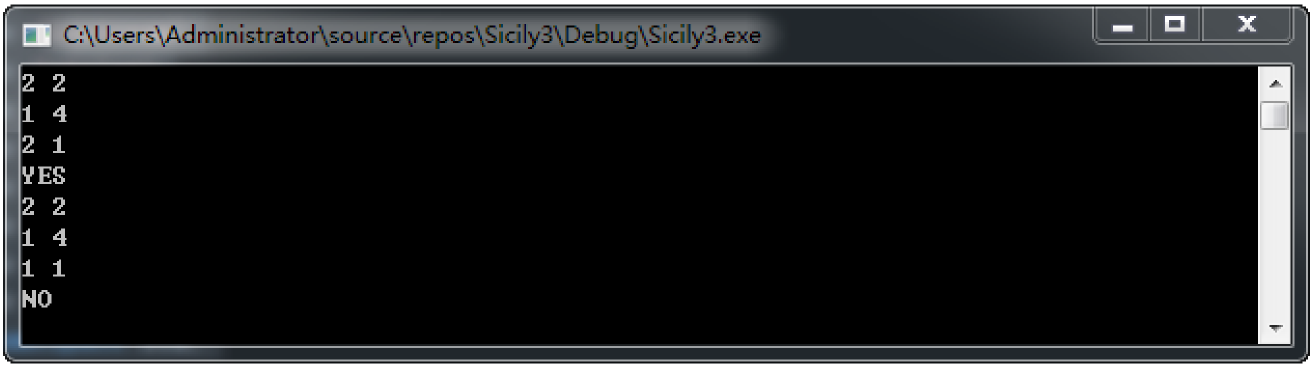
```
1
1 11 5 15 32 47 64 54 48 63 53 59 49 34 17 2 12 6 16 31 21 4 10 25 19 9 3 18 33
27 42 57 51 41 58 43 26 36 30 20 37 22 7 24 14 8 23 13 28 38 55 40 46 61 44 29 3
9 56 62 52 35 45 60 50
2
2 17 11 1 18 3 9 26 41 58 52 62 56 39 24 7 13 23 8 14 4 10 25 19 29 12 6 16 31 4
8 63 46 40 55 61 51 57 42 36 21 15 5 22 32 38 28 34 49 59 53 43 33 27 44 50 35 2
0 30 45 60 54 37 47 64
3
3 9 26 41 58 52 62 56 39 24 7 13 23 8 14 4 10 25 19 2 17 11 1 18 33 50 35 20 5 1
5 32 22 16 6 12 29 46 40 30 47 64 54 60 45 55 61 51 57 42 36 21 31 48 63 53 38 2
8 43 37 27 44 34 49 59
4
4 10 25 42 57 51 41 58 52 62 56 39 24 7 13 3 9 19 2 17 34 49 59 53 63 48 54 64 4
7 32 15 5 22 16 6 12 27 37 31 21 11 1 18 33 43 26 36 30 20 14 8 23 40 46 29 35 5
0 60 45 28 38 55 61 44
5
5 15 32 47 64 54 48 63 53 59 49 34 17 2 12 6 16 31 21 11 1 18 3 9 26 41 58 43 33
 50 60 45 62 56 39 22 7 24 14 8 23 13 28 38 55 40 30 20 37 27 10 4 19 25 35 29 4
4 61 51 57 42 36 46 52
```

1093 Air Express


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

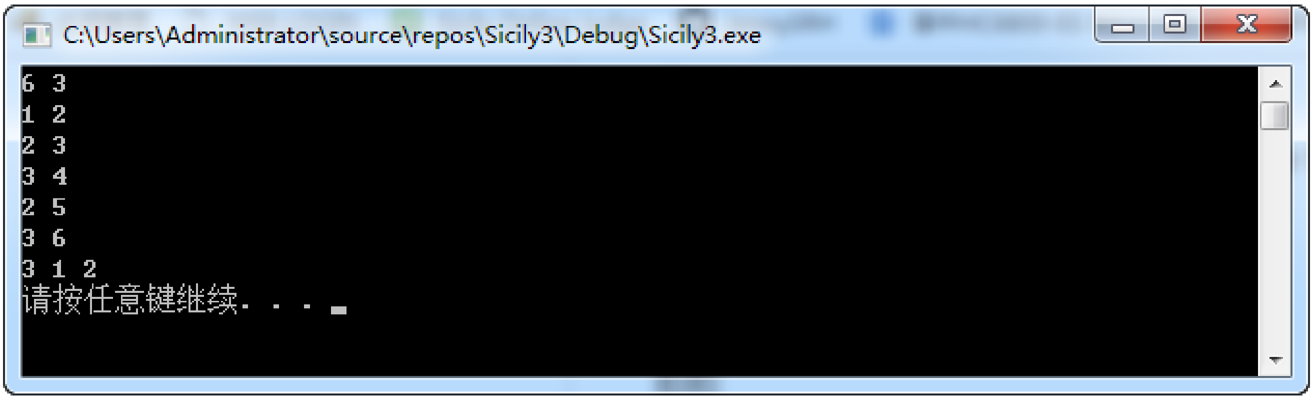
```
2
Set number 1:
8
Weight (8) has best price $50 (add 2 pounds)
10
Weight (10) has best price $50 (add 0 pounds)
90
Weight (90) has best price $200 (add 10 pounds)
100
Weight (100) has best price $200 (add 0 pounds)
200
Weight (200) has best price $400 (add 0 pounds)
0

10 10
20 20
30 30
100
Set number 2:
1
Weight (1) has best price $10 (add 0 pounds)
12
Weight (12) has best price $240 (add 0 pounds)
29
Weight (29) has best price $870 (add 0 pounds)
50
Weight (50) has best price $5000 (add 0 pounds)
```

1134 积木分发


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

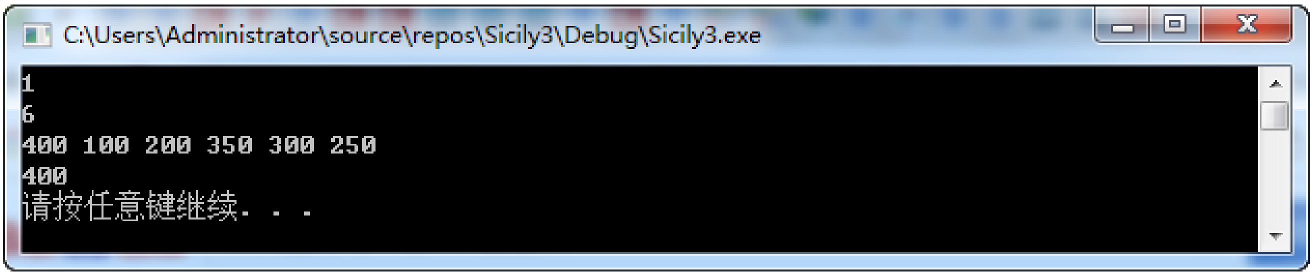
```
2 2
1 4
2 1
YES
2 2
1 4
1 1
NO
```

1140 国王的遗产


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

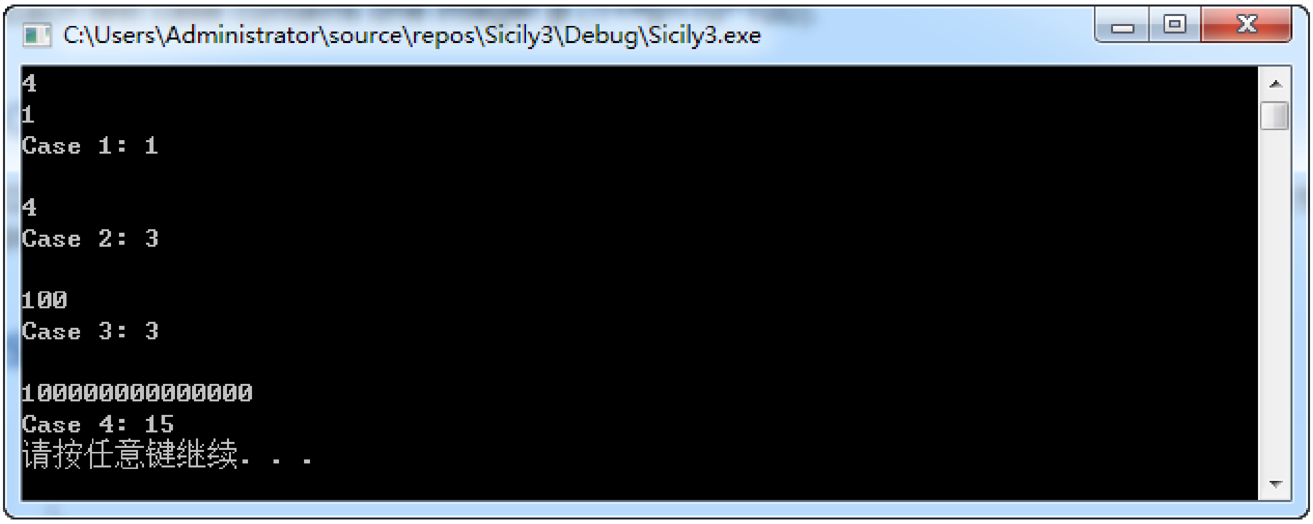
```
6 3
1 2
2 3
3 4
2 5
3 6
3 1 2
请按任意键继续. . .
```

1438 Shopaholic


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

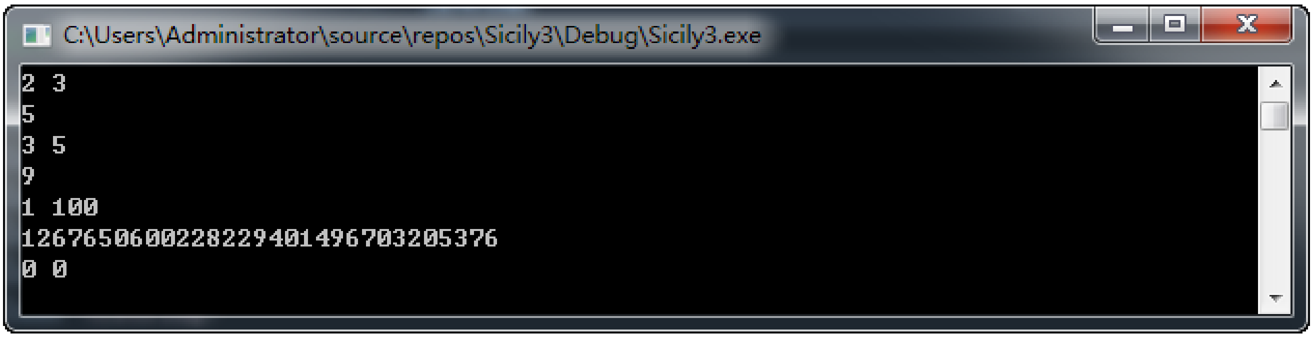
```
1
6
400 100 200 350 300 250
400
请按任意键继续. . .
```

1028 Hanoi Tower Sequence


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

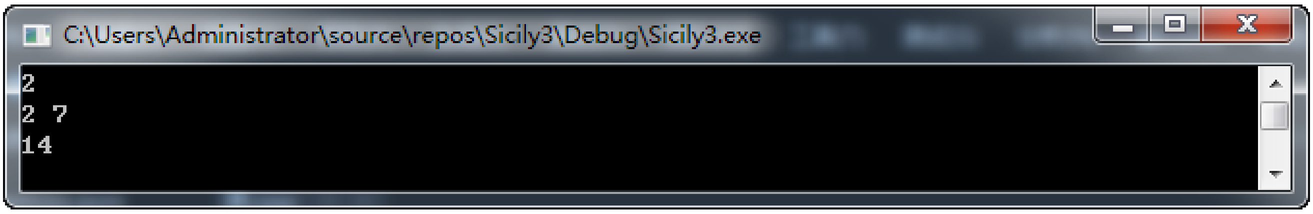
```
4
1
Case 1: 1

4
Case 2: 3

100
Case 3: 3

1000000000000000
Case 4: 15
请按任意键继续. . .
```

1029 Rabbit


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

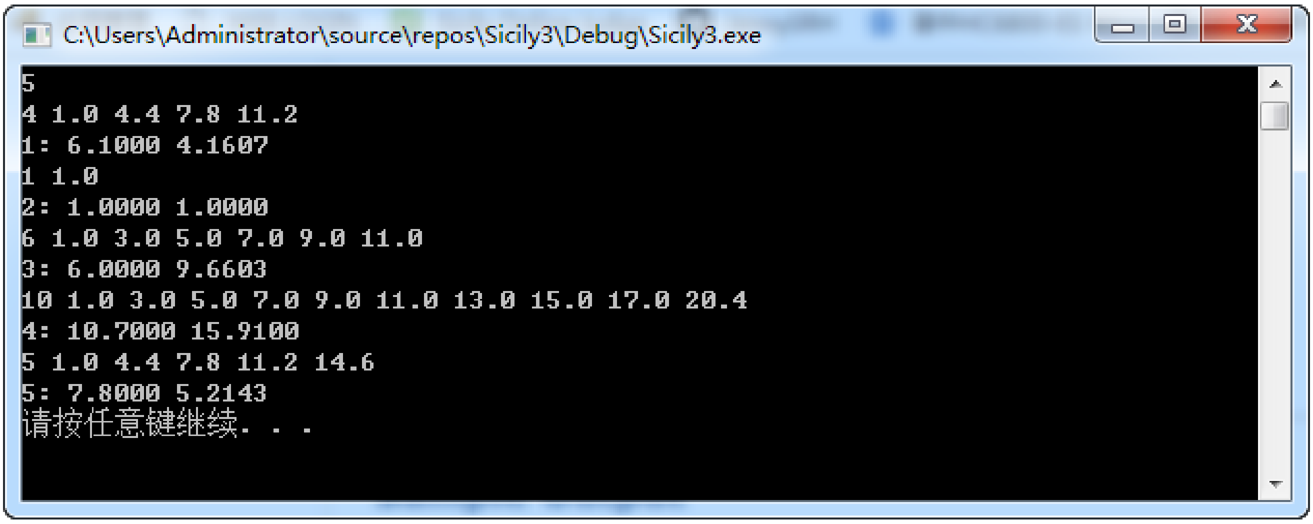
```
2 3
5
3 5
9
1 100
1267650600228229401496703205376
0 0
```

1381 a*b


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

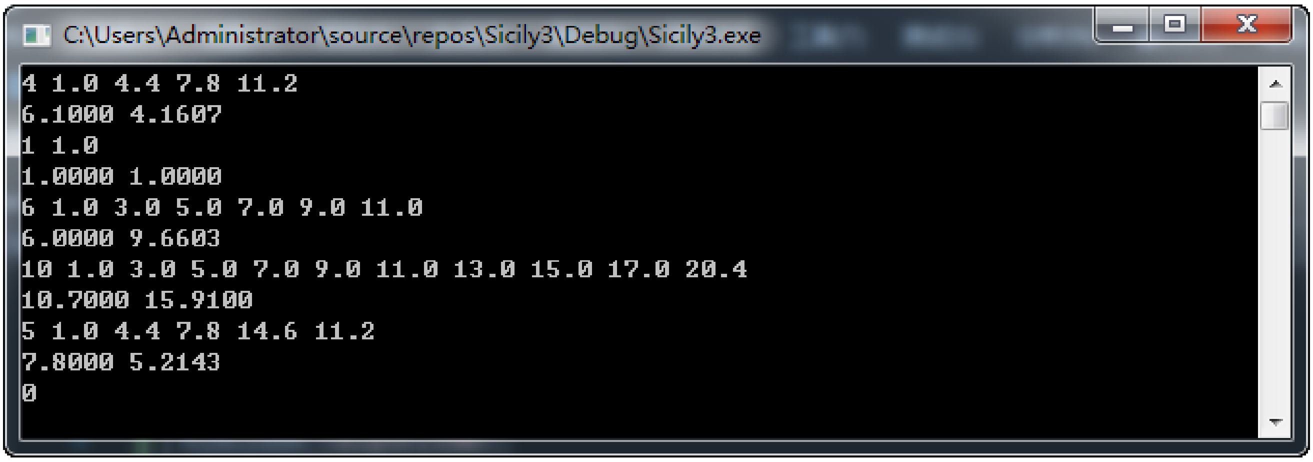
```
2
2 7
14
```

1206 Stacking Cylinders


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

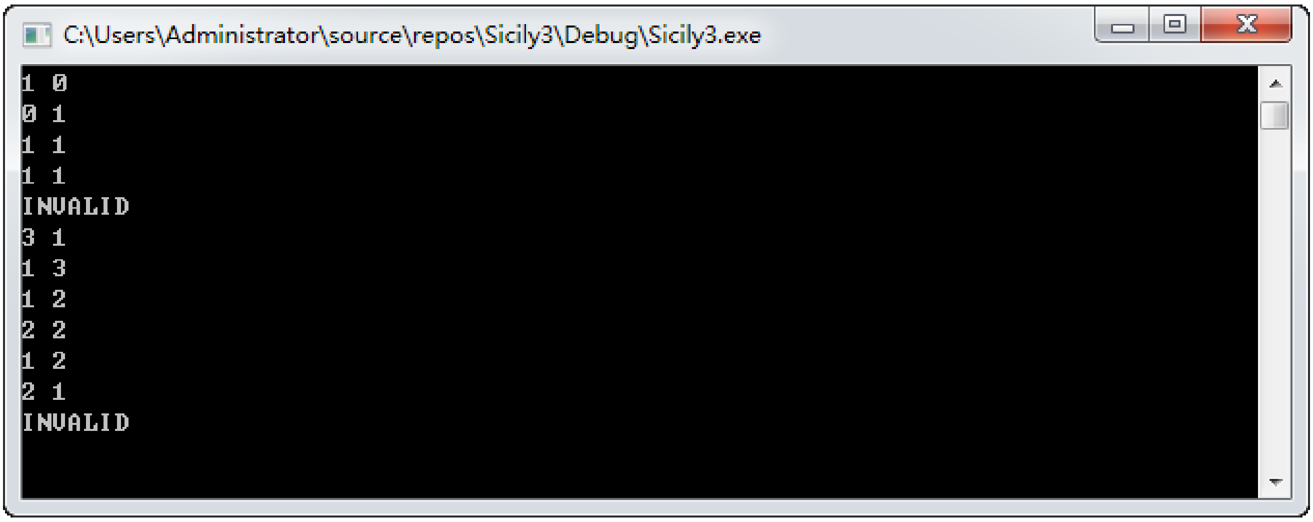
```
5
4 1.0 4.4 7.8 11.2
1: 6.1000 4.1607
1 1.0
2: 1.0000 1.0000
6 1.0 3.0 5.0 7.0 9.0 11.0
3: 6.0000 9.6603
10 1.0 3.0 5.0 7.0 9.0 11.0 13.0 15.0 17.0 20.4
4: 10.7000 15.9100
5 1.0 4.4 7.8 11.2 14.6
5: 7.8000 5.2143
请按任意键继续. . .
```

1012 Stacking Cylinders


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

```
4 1.0 4.4 7.8 11.2
6.1000 4.1607
1 1.0
1.0000 1.0000
6 1.0 3.0 5.0 7.0 9.0 11.0
6.0000 9.6603
10 1.0 3.0 5.0 7.0 9.0 11.0 13.0 15.0 17.0 20.4
10.7000 15.9100
5 1.0 4.4 7.8 14.6 11.2
7.8000 5.2143
0
```

1034 Forest


C:\Users\Administrator\source\repos\Sicily3\Debug\Sicily3.exe

```
1 0
0 1
1 1
1 1
INVALID
3 1
1 3
1 2
2 2
1 2
2 1
INVALID
```

5.实验总结与心得

本次实验耗时最长的是1140，树的分治，一开始总是超时，最后剪枝了终于ac了，二百行代码重写了四遍。通过这次实验，我掌握了基本的回溯法，搜索算法以及贪心，收获很大。

最后还是需要多加练习才行，思路转换为代码的效率还是太低了。

附录、提交文件清单

1152 简单的马周游问题

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <vector>
using namespace std;

const int N = 5;//棋盘的边长
const int M = 6;
int chess[N][M];//标记
vector<int> order;
void showTrail() {
    int count = 1;
    while (count <= 30) {
        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < M; ++j) {
                if (count == chess[i][j]) {
                    cout << i * 6 + j + 1;
                    if (count < 30) {
                        cout << " ";
                    }
                    count++;
                }
            }
        }
    }
    cout << endl;

}

bool flag = false;

void dfs(int x, int y, int cnt) {
    if (cnt == N * M) {
        showTrail();
        flag = true;
        return;
    }
    if (flag) {
        return;
    }
    for (int i = 0; i<8; ++i) {
        int move[8][2] = { { 1,-2 },{ 2,-1 },{ 2,1 },{ 1,2 },{ -1,2 },{ -2,1 },{ -2,-1 },{ -1,-2
} };
        int x1 = x + move[i][0];
        int y1 = y + move[i][1];
        if (0 == chess[x1][y1] && x1 >= 0 && x1<N&&y1 >= 0 && y1<M) {
            chess[x1][y1] = cnt + 1;
            dfs(x1, y1, cnt + 1);
            chess[x1][y1] = 0;
        }
    }
    return;
}


int main()
{
    int start;
    while (true) {
        cin >> start;
        if (start <= -1 || start > 30) {
            return 0;
        }
        flag = false;
        int x = (start - 1) / 6;
        int y = (start - 1) % 6;
        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < M; ++j) {
                chess[i][j] = 0;
            }
        }
        chess[x][y] = 1;
        order.push_back(start);
        dfs(x, y, 1);
    }
}
```

1153 马的周游问题

```
// 1153
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <math.h>
#include <iomanip>
using namespace std;


int map[8][8];

int offset_x[] = { -1, -2,-2,-1,1, 2, 2, 1 };
int offset_y[] = { -2, -1, 1, 2,2, 1,-1,-2 };


bool visited[65];
```

```
int visit_order[65];
int count_order = 1;

int get_n(int n, int i) {
    int x = (n - 1) / 8;
    int y = (n - 1) % 8;
    x = x + offset_x[i];
    y = y + offset_y[i];
    return map[x][y];
}


bool isValid(int n, int i) {
    int x = (n-1) / 8;
    int y = (n-1) % 8;
    x = x + offset_x[i];
    y = y + offset_y[i];
    if (x >= 0 && x <= 7 && y >= 0 && y <= 7 ) {
        int n_next = get_n(n, i);
        if (visited[n_next] == false) {
            return true;
        }
    }
    return false;
}


int get_c(int n) {
    int  count = 0;
    for (int i = 0; i < 8; i++) {
        if (isValid(n, i)) {
            count++;
        }
    }
    return count;
}


struct  next_node {
    int index;
    int count;
};

void swap(next_node & a, next_node & b) {
    next_node temp = a;
    a = b;
    b = temp;
}


void dfs(int n) {

    if (count_order > 64) {
        return;
    }
    visited[n] = true;
    visit_order[n] = count_order++;
    if (count_order > 64) {
        return;
    }
    vector<next_node> order;
    for (int i = 0; i < 8; i++) {
        if (isValid(n, i)) {
            next_node next;
            next.index = get_n(n, i);
            next.count = get_c(next.index);
            if (next.count != 0) {
                order.push_back(next);
            }
        }
    }
    for (int i = 0; i < order.size(); i++) {
        for (int j = 0; j < order.size()-1; j++) {
            if (order[j].count > order[j + 1].count) {
                swap(order[j], order[j + 1]);
            }
        }
    }
    for (int i = 0; i < order.size(); i++) {
        dfs(order[i].index);
        if (count_order >= 64) {
            return;
        }
        visited[order[i].index] = false;
        count_order--;
    }

}

int main() {
    int count = 1;
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            map[i][j] = count++;
        }
    }
    while (true) {
        int n;
        cin >> n;
        if (n == -1) {
            return 0;
        }
        count_order = 1;
        memset(visited, false, sizeof(visited));
        memset(visit_order, 0, sizeof(visit_order));

        dfs(n);
        int  c = 1;
        while(true) {
            if (c == 64) {
                break;
            }
            for (int i = 1; i < 65; i++) {
                if (visit_order[i] == c) {
                    c++;
                    cout << i << " ";
                    break;
                }
            }
        }
        for (int i = 1; i < 65; i++) {
            if (visit_order[i] == 0 || visit_order[i] == 64) {
                cout << i << endl;
                break;
            }
        }
    }
}
```

1093 Air Express

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <iomanip>
using namespace std;

int weight[3];
int cost[4];

int get_min_w(int n) {
    if (n <= weight[0]) {
        return 0;
    }
    else if (n > weight[0] && n <= weight[1]) {
        return 1;
    }
    else if (n <= weight[2]) {
        return 2;

    }
    else {
        return 3;
    }
}



void print(int w, int p, int add) {
    //  cout << "Weight(" << w << ") has best price $" << p << "(add " << add << " pounds)" << endl;
    cout << "Weight (" << w << ") has best price $"
        << p << " (add " << add << " pounds)\n";
}

int temp_cost[4];
int add_pounds[4];

int main() {
    int count = 0;
    while (cin >> weight[0]) {
        count++;
        cin >> cost[0];
        cin >> weight[1] >> cost[1] >> weight[2] >> cost[2] >> cost[3];
        cout << "Set number " << count << ":" << endl;
        int n;
        while (true) {
            cin >> n;
            if (n == 0) {
                break;
            }
            int f = get_min_w(n);
            for (int i = 0; i < 4; i++) {
                add_pounds[i] = 0;
            }
            if (f == 0) {
                temp_cost[0] = cost[0] * n;
                for (int i = 1; i < 4; i++) {
                    add_pounds[i] = weight[i - 1] + 1 - n;
                    temp_cost[i] = cost[i] * (weight[i - 1] + 1);
                }
                int min = temp_cost[0];
                int index = 0;
                for (int i = 1; i < 4; i++) {
                    if (min > temp_cost[i]) {
                        min = temp_cost[i];
                        index = i;
                    }
                }
                print(n, min, add_pounds[index]);
            }
            else if (f == 1) {
                temp_cost[1] = cost[1] * n;

                for (int i = 2; i < 4; i++) {
                    add_pounds[i] = weight[i - 1] + 1 - n;
                    temp_cost[i] = cost[i] * (weight[i - 1] + 1);
                }
                int min = temp_cost[1];
                int index = 1;
                for (int i = 2; i < 4; i++) {
                    if (min > temp_cost[i]) {
                        min = temp_cost[i];
                        index = i;
                    }
                }
                print(n, min, add_pounds[index]);
            }
            else if (f == 2) {
                temp_cost[2] = cost[2] * n;
                for (int i = 3; i < 4; i++) {
                    add_pounds[i] = weight[i - 1] + 1 - n;
                    temp_cost[i] = cost[i] * (weight[i - 1] + 1);
                }
                int min = temp_cost[2];
                int index = 2;
                for (int i = 3; i < 4; i++) {
                    if (min > temp_cost[i]) {
                        min = temp_cost[i];
                        index = i;
                    }
                }
                print(n, min, add_pounds[index]);
            }
            else {
                print(n, cost[3] * n, 0);
            }
        }
        cout << endl;
    }
}
```

1134 积木分发

```
// 1134
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <iomanip>
using namespace std;
```

```
struct puzzle {
    long long has_get;
    long long to_get;
};

puzzle puz[10050];

struct Cmp {
    bool operator ()(puzzle a, puzzle b) {
        return a.to_get < b.to_get;
    }
};

int main() {
    ios::sync_with_stdio(false);
    int n;
    while (true) {
        cin >> n;
        if (n == 0) {
            return 0;
        }
        long long s;
        cin >> s;
        for (int i = 0; i < n; i++) {
            cin >> puz[i].has_get >> puz[i].to_get;
        }
        bool flag = true;
        sort(puz, puz + n,Cmp());
        for (int i = 0; i < n; i++) {
            if (s < puz[i].to_get) {
                cout << "NO" << endl;
                flag = false;
                break;
            }
            s += puz[i].has_get;
        }
        if (flag) {
            cout << "YES" << endl;
        }
    }
}
```

1140 国王的遗产

```
// 1140
#include <iostream>
#include <string>

#include <vector>
```

```
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <iomanip>
using namespace std;

int min(int a, int b) {
    return a < b ? a : b;
}

struct node {
    int in;
    vector<int> out;
    int root_count;     //以该节点为根的子树的节点数目
    bool exist;         // 该节点是否存在
    int min_index;  // 以该节点为根的子树的最小节点
    node() {
        in = -1;
        root_count = -1;
        exist = false;
        min_index = -1;
    }
};

node tree_node[30050];


void build_tree(int n) {
    for (int i = 1; i < n; i++) {
        int a, b;
        cin >> a >> b;
        tree_node[a].exist = true;
        tree_node[b].exist = true;
        if (tree_node[a].in == -1) {
            tree_node[a].in = b;
            tree_node[b].out.push_back(a);
        }
        else {
            tree_node[b].in = a;
            tree_node[a].out.push_back(b);
        }
    }
}

int set_subtree_count(int root) {
    if (tree_node[root].exist == false) {
        return 0;
    }
    if (tree_node[root].out.size() == 0) {
        tree_node[root].root_count = 1;
        return 1;

    }
    if (tree_node[root].root_count != -1) {
        return tree_node[root].root_count;
    }
    tree_node[root].root_count = 1;
    for (int i = 0; i < tree_node[root].out.size(); i++) {
        tree_node[root].root_count += set_subtree_count(tree_node[root].out[i]);
    }
    return tree_node[root].root_count;
}

int set_min_index(int root) {
    if (tree_node[root].exist == false) {
        return 99999;
    }
    if (tree_node[root].out.size() == 0) {
        tree_node[root].min_index = root;
    }
    if (tree_node[root].min_index != -1) {
        return tree_node[root].min_index;
    }
    tree_node[root].min_index = root;
    for (int i = 0; i < tree_node[root].out.size(); i++) {
        tree_node[root].min_index = min(tree_node[root].min_index,
set_min_index(tree_node[root].out[i]));
    }
    return tree_node[root].min_index;
}

void remove_edge(int index) { // 去掉一条边
    int edge_from = tree_node[index].in;
    int edge_to = index;
    tree_node[edge_to].in = -1;
    for (int k = 0; k < tree_node[edge_from].out.size(); k++) {
        if (tree_node[edge_from].out[k] == edge_to) {
            for (int j = k; j < tree_node[edge_from].out.size() - 1; j++) {
                tree_node[edge_from].out[j] = tree_node[edge_from].out[j + 1];
            }
            tree_node[edge_from].out.pop_back();
            break;
        }
    }
}

void add_edge(int edge_from, int edge_to) {
    tree_node[edge_from].out.push_back(edge_to);
    tree_node[edge_to].in = edge_from;
}

int get_min_index_sub(int root) {
    if (tree_node[root].exist == false) {
        return 999999;
    }

    if (tree_node[root].out.size() == 0) {
        return root;
    }
    int min_index = root;
    for (int i = 0; i < tree_node[root].out.size(); i++) {
        min_index = min(min_index, get_min_index_sub(tree_node[root].out[i]));
    }
    return min_index;
}

int get_min_index(int root, int k) {
    int edge_from = tree_node[k].in;
    int edge_to = k;
    remove_edge(k);
    int min_index = get_min_index_sub(root);
    add_edge(edge_from, edge_to);
    return min_index;
}

void init_false(int n) {
    for (int i = 1; i <= n; i++) {
        tree_node[i].exist = false;
    }
}

void init(int root) {
    tree_node[root].exist = true;
    tree_node[root].min_index = -1;
    tree_node[root].root_count = -1;
    for (int i = 0; i < tree_node[root].out.size(); i++) {
        init(tree_node[root].out[i]);
    }
    return;
}


int main() {
    ios::sync_with_stdio(false);
    int n, k;
    cin >> n >> k;
    build_tree(n);
    int root;
    for (int i = 1; i <= n; i++) {
        if (tree_node[i].in == -1) { // 找到根
            root = i;
            break;
        }
    }
    int sum_count = n;
    for (int i = 0; i < k - 1; i++) {
        set_subtree_count(root);    // O(n)
        set_min_index(root);    // O(n)


        int delete_root;
        int save_root;
        int delete_root_min_index = 9999999;
        int delete_count = -1;
        for (int k = 1; k <= n; k++) {
            if (k == root) {
                continue;
            }
            if (tree_node[k].exist) {
                int delete_left_count = sum_count - tree_node[k].root_count;
                if (delete_left_count < tree_node[k].root_count) {
                    if (delete_left_count > delete_count) {
                        int delete_left_min_index = get_min_index(root, k);
                        delete_count = delete_left_count;
                        delete_root = k;
                        save_root = k;
                        delete_root_min_index = delete_left_min_index;
                    }
                    else if (delete_left_count == delete_count) {
                        int delete_left_min_index = get_min_index(root, k);
                        if (delete_left_min_index < delete_root_min_index) {
                            delete_root = k;
                            save_root = k;
                            delete_count = delete_left_count;
                            delete_root_min_index = delete_left_min_index;
                        }
                    }
                }
                else if (delete_left_count > tree_node[k].root_count) {
                    if (tree_node[k].root_count > delete_count || (tree_node[k].root_count ==
delete_count && tree_node[k].min_index < delete_root_min_index)) {
                        delete_count = tree_node[k].root_count;
                        delete_root = k;
                        save_root = root;
                        delete_root_min_index = tree_node[k].min_index;
                    }
                }
                else {
                    if (tree_node[k].root_count > delete_count ) {
                        int delete_left_min_index = get_min_index(root, k);
                        if (tree_node[k].min_index < delete_left_min_index) {
                                delete_count = tree_node[k].root_count;
                                delete_root = k;
                                save_root = root;
                                delete_root_min_index = tree_node[k].min_index;
                        }
                        else {
                                delete_count = tree_node[k].root_count;
                                delete_root = k;
                                save_root = k;
                                delete_root_min_index = delete_left_min_index;
                        }
                    }

                    else if (tree_node[k].root_count == delete_count) {
                        int delete_left_min_index = get_min_index(root, k);
                        if (delete_count < delete_left_min_index && delete_count <
tree_node[k].min_index) {
                            if (tree_node[k].min_index < delete_left_min_index) {
                                if (tree_node[k].min_index < delete_root_min_index) {
                                    delete_count = tree_node[k].root_count;
                                    delete_root = k;
                                    save_root = root;
                                    delete_root_min_index = tree_node[k].min_index;
                                }
                            }
                            else {
                                if (delete_left_min_index < delete_root_min_index) {
                                    delete_count = tree_node[k].root_count;
                                    delete_root = k;
                                    save_root = k;
                                    delete_root_min_index = delete_left_min_index;
                                }
                            }
                        }
                    }
                }
            }
        }
        remove_edge(delete_root);
        root = save_root;
        init_false(n);
        init(root);
        cout << delete_count << " ";
        sum_count = sum_count - delete_count;
//      for (int i = 1; i <= n; i++) {
//          cout << "index " << i << " count: " << tree_node[i].root_count << " min_index: " <<
tree_node[i].min_index << endl;
//      }
    }
    cout << sum_count << endl;
//  system("pause");
}
```

1438 Shopaholic

```
// 1438
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>

#include <iomanip>
```

```
using namespace std;

int v[20010];

bool cmp(int a, int b)
{
    return a>b;
}

int main() {
    int T;
    cin >> T;
    while (T--) {
        int n;
        cin >> n;
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        sort(v, v + n, cmp);
        int sum = 0;
        if (n < 3) {
            cout << 0 << endl;
            continue;
        }
        for (int i = 2; i < n; i += 3) {
            sum += v[i];
        }
        cout << sum << endl;
    }
}
```

1028 Hanoi Tower Sequence

```
// 1028
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <iomanip>
using namespace std;

int a[110];

int main() {
    int T;
    cin >> T;

    int C = 0;
    while (T--) {
        C++;
        string str;
        cin >> str;
        int k = 2;
        int count = 1;
        //str是大数，求str%k,ans即为结果
        int ans = 0;
        for (int i = str.length()-1; i >= 0; i--) {
            a[i] = int(str[str.length()-i-1] - '0');
        }
        int l = str.length()-1;
        while (a[0] % 2 == 0) {
            int rem = 0;
            while (a[l] == 0) l--;
            for (int i = l; i >= 0; i--) {
                a[i] = a[i] + rem * 10;
                rem = a[i] % 2;
                a[i] /= 2;
            }
            count++;
        }
        cout << "Case " << C << ": " << count << endl;
        if (T > 0) {
            cout << endl;
        }
    }
}
```

1029 Rabbit

```
// 1029
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <iomanip>
using namespace std;

string adult;

string get_sum(string adult, string children) {
    reverse(adult.begin(), adult.end());
    reverse(children.begin(), children.end());
    string temp = (adult.length() > children.length() ? adult : children);
    if (children.length() < adult.length()) {

        int c_l = children.length();
        for (int i = c_l; i < adult.length(); i++) {
            children += '0';
        }
    }
    else if (children.length() > adult.length()) {
        int a_l = adult.length();
        for (int i = a_l; i < children.length(); i++) {
            adult += '0';
        }
    }
    children += '0';
    adult += '0';
    temp += '0';
    int c = 0;
    for (int i = 0; i < temp.length(); i++) {
        int cur_sum = children[i] + adult[i] - '0' - '0' + c;
        c = (cur_sum - 9 > 0 ? 1 : 0);
        if (cur_sum > 9) {
            cur_sum -= 10;
        }
        temp[i] = cur_sum + '0';
    }
    int count = temp.length();
    while ( count > 0 && temp[count-1] == '0') {
        count--;
    }
    string sum = temp.substr(0, count);
    reverse(sum.begin(), sum.end());
    return sum;
}

string children[20];

int main() {
    int m, d;
    while (true) {
        cin >> m >> d;
        if (m == 0) {
            return 0;
        }
        int temp_count = m;
        for (int i = 0; i < 20; i++) {
            children[i] = "0";  // 还有i个月长大的孩子的个数
        }
        children[m] = "1";
        adult = "1";
        for (int i = 1; i < d; i++) {
            adult = get_sum(children[1], adult);
            for (int k = 1; k < m; k++) {
                children[k] = children[k + 1];
            }
            children[m] = adult;

        }
        string sum = "0";
        for (int i = 1; i <= m; i++) {
            sum = get_sum(sum, children[i]);
        }
        cout << get_sum(sum, adult) << endl;
    }
}
```

1381 a*b

```
// 1381
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <iomanip>
using namespace std;


string get_sum(string adult, string children) {
    reverse(adult.begin(), adult.end());
    reverse(children.begin(), children.end());
    string temp = (adult.length() > children.length() ? adult : children);
    if (children.length() < adult.length()) {
        int c_l = children.length();
        for (int i = c_l; i < adult.length(); i++) {
            children += '0';
        }
    }
    else if (children.length() > adult.length()) {
        int a_l = adult.length();
        for (int i = a_l; i < children.length(); i++) {
            adult += '0';
        }
    }
    children += '0';
    adult += '0';
    temp += '0';
    int c = 0;
    for (int i = 0; i < temp.length(); i++) {
        int cur_sum = children[i] + adult[i] - '0' - '0' + c;
        c = (cur_sum - 9 > 0 ? 1 : 0);
        if (cur_sum > 9) {
            cur_sum -= 10;
        }
        temp[i] = cur_sum + '0';
    }
    int count = temp.length();
    while (count > 0 && temp[count - 1] == '0') {
        count--;
    }
    string sum = temp.substr(0, count);
    reverse(sum.begin(), sum.end());
    return sum;
}

string get_single_mul(string a, char b) {
    string sum = "0";
    int n = b - '0';
    for (int i = 0; i < n; i++) {
        sum = get_sum(sum, a);
    }
    return sum;
}

string get_mul(string a, string b) {
    string sum = "0";
    string mul = "0";
    reverse(b.begin(), b.end());
    for (int i = 0; i < b.length(); i++) {
        mul = get_single_mul(a, b[i]);
        for (int j = 0; j < i; j++) {
            mul += "0";
        }
        sum = get_sum(sum, mul);
    }
    return sum;
}

int main() {
    int T;
    cin >> T;
    while (T--) {
        string a, b;
        cin >> a >> b;
        string ans = get_mul(a, b);
        if (ans.length() == 0) {
            cout << 0 << endl;
        }
        else {
            cout << get_mul(a, b) << endl;
        }
    }
}
```

1206 Stacking Cylinders

```
// 1206
#include <iostream>
#include <string>
#include <vector>
```

```
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <math.h>
#include <iomanip>
using namespace std;
#define PI 3.14159265358979323846
struct cycle {
    double x, y;
    cycle() {

    }
    cycle(double x, double y) {
        this->x = x;
        this->y = y;
    }
};

int main() {
    int T;
    cin >> T;
    double temp_arr[1010];
    int count = 1;
    while (T--) {
        int n;
        cin >> n;
        vector<cycle> arr;
        for (int i = 0; i < n; i++) {
            cin >> temp_arr[i];
        }
        sort(temp_arr, temp_arr + n);
        for (int i = 0; i < n; i++) {
            arr.push_back(cycle(temp_arr[i], 1));
        }
        for (int i = n-1; i > 0; i--) {
            vector<cycle> arr_temp;
            for (int k = 0; k < i; k++) {
                cycle node1, node2, node;
                node1 = arr[k];
                node2 = arr[k + 1];
                double a = atan(abs(node1.y - node2.y) / abs(node1.x - node2.x));
                double b = acos(sqrt((node2.y - node1.y)*(node2.y - node1.y) + (node2.x - node1.x)*(node2.x - node1.x)) / 4);
                a = a + b;
                if (node2.y > node1.y) {
                    node.x = node1.x + 2 * cos(a);
                    node.y = node1.y + 2 * sin(a);
                }
                else if (node2.y < node1.y) {
                    node.x = node2.x - 2 * cos(a);
                    node.y = node2.y + 2 * sin(a);

                }
                else {
                    node.x = node1.x + 2 * cos(b);
                    node.y = node1.y + 2 * sin(b);
                }
                arr_temp.push_back(node);
            }
            arr.clear();
            arr = arr_temp;
        }
        cout.precision(4);
        cout << count << ": "  << fixed <<  arr[0].x << " " << arr[0].y << endl;
        count++;
    }
    //system("pause");
}
```

1012 Stacking Cylinders

```
// 1012
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#include <cstring>
#include <math.h>
#include <iomanip>
using namespace std;
#define PI 3.14159265358979323846
struct cycle {
    double x, y;
    cycle() {

    }
    cycle(double x, double y) {
        this->x = x;
        this->y = y;
    }
};

int main() {

    double temp_arr[1010];
    int count = 1;
    while (true) {
        int n;
        cin >> n;
        if (n == 0) {
            return 0;
        }

        vector<cycle> arr;
        for (int i = 0; i < n; i++) {
            cin >> temp_arr[i];
        }
        sort(temp_arr, temp_arr + n);
        for (int i = 0; i < n; i++) {
            arr.push_back(cycle(temp_arr[i], 1));
        }
        for (int i = n - 1; i > 0; i--) {
            vector<cycle> arr_temp;
            for (int k = 0; k < i; k++) {
                cycle node1, node2, node;
                node1 = arr[k];
                node2 = arr[k + 1];
                double a = atan(abs(node1.y - node2.y) / abs(node1.x - node2.x));
                double b = acos(sqrt((node2.y - node1.y)*(node2.y - node1.y) + (node2.x - node1.x)*(node2.x - node1.x)) / 4);
                a = a + b;
                if (node2.y > node1.y) {
                    node.x = node1.x + 2 * cos(a);
                    node.y = node1.y + 2 * sin(a);
                }
                else if (node2.y < node1.y) {
                    node.x = node2.x - 2 * cos(a);
                    node.y = node2.y + 2 * sin(a);
                }
                else {
                    node.x = node1.x + 2 * cos(b);
                    node.y = node1.y + 2 * sin(b);
                }
                arr_temp.push_back(node);
            }
            arr.clear();
            arr = arr_temp;
        }
        cout.precision(4);
        cout <<  fixed << arr[0].x << " " << arr[0].y << endl;
        count++;
    }
    //system("pause");
}
```

1034 Forest

```
// 1034
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <queue>

#include <stack>
```

```
#include <cstring>
#include <iomanip>
using namespace std;

int max(int a, int b) {
    return (a > b ? a : b);
}

struct node {
    int in;
    int depth;
    vector<int> out;
    node() {
        in = -1;
        depth = -1;
    }
};

node tree_node[200];

bool valid_flag;

bool visited[200];

int dfs_depth(int index) {
    if (valid_flag == false) {
        return -1;
    }
    if (visited[index]) {
        valid_flag = false;
        return -1;
    }
    visited[index] = true;
    if (tree_node[index].out.size() == 0) {
        return 0;
    }
    int max_depth = dfs_depth(tree_node[index].out[0]) + 1;
    int visit_node = tree_node[index].out[0];
    for (int i = 1; i < tree_node[index].out.size(); i++) {
        max_depth = max(max_depth, dfs_depth(tree_node[index].out[i]) + 1);
    }
    return max_depth;
}

void get_depth(int index) {
    if (tree_node[index].in == -1) {
        tree_node[index].depth = 0;
        return;
    }
    if (tree_node[index].depth != -1) {
        return;
    }

    if (tree_node[tree_node[index].in].depth == -1) {
        get_depth(tree_node[index].in);
    }
    tree_node[index].depth = tree_node[tree_node[index].in].depth + 1;


}

int main() {
    int n, m;
    while (true) {
        cin >> n >> m;
        valid_flag = true;
        if (n == 0) {
            return 0;
        }
        for (int i = 1; i <= n; i++) {
            visited[i] = false;
            tree_node[i].depth = -1;
            tree_node[i].in = -1;
            tree_node[i].out.clear();
        }
        for (int i = 0; i < m; i++) {
            int a, b;
            cin >> a >> b;
            if (a == b) {
                valid_flag = false;
            }
            tree_node[a].out.push_back(b);
            tree_node[b].in = a;
        }
        int max_depth = 0;
        bool has_root_flag = false;
        for (int i = 1; i <= n; i++) {
            if (tree_node[i].in == -1) {
                int temp = dfs_depth(i);
                visited[i] = true;
                max_depth = max(max_depth, temp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (visited[i] == false) {
                valid_flag = false;
                break;
            }
        }

        if (valid_flag) {
            int width[200];
            for (int i = 0; i <= max_depth; i++) {
                width[i] = 0;
            }
            for (int i = 1; i <= n; i++) {

                get_depth(i);
            }
            for (int i = 1; i <= n; i++) {
                width[tree_node[i].depth]++;
            }
            int max_width = 0;
            for (int i = 0; i <= max_depth; i++) {
                max_width = max(max_width, width[i]);
            }
            cout << max_depth << " " << max_width << endl;
        }
        else {
            cout << "INVALID" << endl;
        }
    }
}
```