1001. A+B Format (20) [字符串处理]

Calculate a + b and output the sum in standard format — that is, the digits must be separated into groups of three by commas (unless there are less than four digits).

Input

Each input file contains one test case. Each case contains a pair of integers a and b where -1000000 <= a, b <= 1000000. The numbers are separated by a space.

Output

For each test case, you should output the sum of a and b in one line. The sum must be written in the standard format.

Sample Input

-1000000 9

Sample Output

-999,991

题目大意：计算A+B的和，然后以每三位加一个”,”的格式输出～

分析：把a+b的和转为字符串s～除了第一位是负号的情况，只要当前位的下标i满足(i + 1) % 3 == len % 3并且i不是最后一位，就在逐位输出的时候在该位输出后的后面加上一个逗号～

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b;
    string s = to_string(a + b);
    int len = s.length();
    for (int i = 0; i < len; i++) {
        cout << s[i];
        if (s[i] == '-') continue;
        if ((i + 1) % 3 == len % 3 && i != len - 1) cout << ",";
    }
    return 0;
}
```

1002. A+B for Polynomials (25) [模拟]

This time, you are supposed to find A+B where A and B are two polynomials.

Input

Each input file contains one test case. Each case occupies 2 lines, and each line contains the information of a polynomial: K N1 aN1 N2 aN2 … NK aNK, where K is the number of nonzero terms in the polynomial, Ni and aNi (i=1, 2, …, K) are the exponents and coefficients, respectively. It is given that 1 <= K <= 10， 0 <= NK < … < N2 < N1 <=1000.

Output

For each test case you should output the sum of A and B in one line, with the same format as the input. Notice that there must be NO extra space at the end of each line. Please be accurate to 1 decimal place.

Sample Input

2 1 2.4 0 3.2

2 2 1.5 1 0.5

Sample Output

3 2 1.5 1 2.9 0 3.2

题目大意：计算多项式A+B的和～

分析：设立c数组，长度为指数的最大值，c[i] = j表示指数i的系数为j，接收a和b输入的同时将对应指数的系数加入到c中，累计c中所有非零系数的个数，然后从后往前输出所有系数不为0的指数和系数～

```
#include <iostream>
using namespace std;
int main() {
    float c[1001] = {0};
    int m, n, t;
    float num;
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d%f", &t, &num);
        c[t] += num;
    }
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d%f", &t, &num);
        c[t] += num;
    }
    int cnt = 0;
    for (int i = 0; i < 1001; i++) {
        if (c[i] != 0) cnt++;
    }
    printf("%d", cnt);
    for (int i = 1000; i >= 0; i--) {
        if (c[i] != 0.0)
            printf(" %d %.1f", i, c[i]);
    }
    return 0;
}
```

1003. Emergency (25) [Dijkstra算法]

As an emergency rescue team leader of a city, you are given a special map of your country. The map shows several scattered cities connected by some roads. Amount of rescue teams in each city and the length of each road between any pair of cities are marked on the map. When there is an emergency call to you from some other city, your job is to lead your men to the place as quickly as possible, and at the

mean time, call up as many hands on the way as possible.

Input

Each input file contains one test case. For each test case, the first line contains 4 positive integers: N (<= 500) – the number of cities (and the cities are numbered from 0 to N-1), M – the number of roads, C1 and C2 – the cities that you are currently in and that you must save, respectively. The next line contains N integers, where the i-th integer is the number of rescue teams in the i-th city. Then M lines follow, each describes a road with three integers c1, c2 and L, which are the pair of cities connected by a road and the length of that road, respectively. It is guaranteed that there exists at least one path from C1 to C2.

Output

For each test case, print in one line two numbers: the number of different shortest paths between C1 and C2, and the maximum amount of rescue teams you can possibly gather.

All the numbers in a line must be separated by exactly one space, and there is no extra space allowed at the end of a line.

Sample Input

5 6 0 2

1 2 1 5 3

0 1 1

0 2 2

0 3 1

1 2 1

2 4 1

3 4 1

Sample Output

2 4

题目大意：n个城市m条路，每个城市有救援小组，所有的边的边权已知。给定起点和终点，求从起点到终点的最短路径条数以及最短路径上的救援小组数目之和。如果有多条就输出点权（城市救援小组数目）最大的那个～

分析：用一遍Dijkstra算法～救援小组个数相当于点权，用Dijkstra求边权最小的最短路径的条数，以及这些最短路径中点权最大的值～dis[i]表示从出发点到i结点最短路径的路径长度，num[i]表示从出发点到i结点最短路径的条数，w[i]表示从出发点到i点救援队的数目之和～当判定dis[u] + e[u][v] < dis[v]的时候，不仅仅要更新dis[v]，还要更新num[v] = num[u], w[v] = weight[v] + w[u]; 如果dis[u] + e[u][v] == dis[v]，还要更新num[v] += num[u]，而且判断一下是否权重w[v]更小，如果更小了就更新w[v] = weight[v] + w[u];

```
#include <iostream>
#include <algorithm>
using namespace std;
int n, m, c1, c2;
```

```
int e[510][510], weight[510], dis[510], num[510], w[510];
bool visit[510];
const int inf = 99999999;
int main() {
    scanf("%d%d%d%d", &n, &m, &c1, &c2);
    for(int i = 0; i < n; i++)
        scanf("%d", &weight[i]);
    fill(e[0], e[0] + 510 * 510, inf);
    fill(dis, dis + 510, inf);
    int a, b, c;
    for(int i = 0; i < m; i++) {
        scanf("%d%d%d", &a, &b, &c);
        e[a][b] = e[b][a] = c;
    }
    dis[c1] = 0;
    w[c1] = weight[c1];
    num[c1] = 1;
    for(int i = 0; i < n; i++) {
        int u = -1, minn = inf;
        for(int j = 0; j < n; j++) {
            if(visit[j] == false && dis[j] < minn) {
                u = j;
                minn = dis[j];
            }
        }
        if(u == -1) break;
        visit[u] = true;
        for(int v = 0; v < n; v++) {
            if(visit[v] == false && e[u][v] != inf) {
                if(dis[u] + e[u][v] < dis[v]) {
                    dis[v] = dis[u] + e[u][v];
                    num[v] = num[u];
                    w[v] = w[u] + weight[v];
                } else if(dis[u] + e[u][v] == dis[v]) {
                    num[v] = num[v] + num[u];
                    if(w[u] + weight[v] > w[v])
                        w[v] = w[u] + weight[v];
                }
            }
        }
    }
    printf("%d %d", num[c2], w[c2]);
    return 0;
}
```

1004. Counting Leaves (30) [BFS， DFS， 树的层序遍历]

A family hierarchy is usually presented by a pedigree tree. Your job is to count those family members who have no child.

Input

Each input file contains one test case. Each case starts with a line containing 0 < N < 100, the number of nodes in a tree, and M (< N), the number of non-leaf nodes. Then M lines follow, each in the format:

ID K ID[1] ID[2] … ID[K]

where ID is a two-digit number representing a given non-leaf node, K is the number of its children, followed by a sequence of two-digit ID's of its children. For the sake of simplicity, let us fix the root ID to be 01.

Output

For each test case, you are supposed to count those family members who have no child for every seniority level starting from the root. The numbers must be printed in a line, separated by a space, and there must be no extra space at the end of each line.

The sample case represents a tree with only 2 nodes, where 01 is the root and 02 is its only child. Hence on the root 01 level, there is 0 leaf node; and on the next level, there is 1 leaf node. Then we should output "0 1" in a line.

Sample Input

2 1

01 1 02

Sample Output

0 1

题目大意：给出一棵树，问每一层各有多少个叶子结点～

分析：可以用dfs也可以用bfs～如果用dfs，用二维数组保存每一个有孩子结点的结点以及他们的孩子结点，从根结点开始遍历，直到遇到叶子结点，就将当前层数depth的book[depth]++；标记第depth层拥有的叶子结点数，最后输出～

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> v[100];
int book[100], maxdepth = -1;
void dfs(int index, int depth) {
    if(v[index].size() == 0) {
        book[depth]++;
        maxdepth = max(maxdepth, depth);
        return ;
    }
    for(int i = 0; i < v[index].size(); i++)
        dfs(v[index][i], depth + 1);
}
int main() {
    int n, m, k, node, c;
    scanf("%d %d", &n, &m);
    for(int i = 0; i < m; i++) {
        scanf("%d %d",&node, &k);
```

```
            for(int j = 0; j < k; j++) {
                scanf("%d", &c);
                v[node].push_back(c);
            }
        }
        dfs(1, 0);
        printf("%d", book[0]);
        for(int i = 1; i <= maxdepth; i++)
            printf(" %d", book[i]);
        return 0;
    }
```

1005. Spell It Right (20) [字符串处理]

Given a non-negative integer N, your task is to compute the sum of all the digits of N, and output every digit of the sum in English.

Input Specification:

Each input file contains one test case. Each case occupies one line which contains an N (<= 10100).

Output Specification:

For each test case, output in one line the digits of the sum in English words. There must be one space between two consecutive words, but no extra space at the end of a line.

Sample Input:

12345

Sample Output:

one five

题目大意：给一个非负正数N，计算N的每一位相加的和，然后输出和的每一位的英文读音～

分析：1、求出每一位相加的和sum

2、将sum转换为string s 3、将string s的每一位输出对应的英文读音～

```
#include <iostream>
using namespace std;
int main() {
    string a;
    cin >> a;
    int sum = 0;
    for (int i = 0; i < a.length(); i++)
        sum += (a[i] - '0');
    string s = to_string(sum);
    string arr[10] = {"zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};
    cout << arr[s[0] - '0'];
    for (int i = 1; i < s.length(); i++)
        cout << " " << arr[s[i] - '0'];
    return 0;
```

```
15    }
```

1006. Sign In and Sign Out (25) [查找元素]

At the beginning of every day, the first person who signs in the computer room will unlock the door, and the last one who signs out will lock the door. Given the records of signing in's and out's, you are supposed to find the ones who have unlocked and locked the door on that day.

Input Specification:

Each input file contains one test case. Each case contains the records for one day. The case starts with a positive integer M, which is the total number of records, followed by M lines, each in the format:

ID_number Sign_in_time Sign_out_time

where times are given in the format HH:MM:SS, and ID number is a string with no more than 15 characters.

Output Specification:

For each test case, output in one line the ID numbers of the persons who have unlocked and locked the door on that day. The two ID numbers must be separated by one space.

Note: It is guaranteed that the records are consistent. That is, the sign in time must be earlier than the sign out time for each person, and there are no two persons sign in or out at the same moment.

Sample Input:

3

CS301111 15:30:28 17:00:10

SC3021234 08:00:00 11:25:25

CS301133 21:45:00 21:58:40

Sample Output:

SC3021234 CS301133

题目大意：给出n个人的id、sign in时间、sign out时间，求最早进来的人和最早出去的人的ID～

分析：将时间都转换为总秒数，最早和最迟的时间保存在变量minn和maxn中，并同时保存当前最早和最迟的人的ID，最后输出～

```
#include <iostream>
#include <climits>
using namespace std;
int main() {
    int n, minn = INT_MAX, maxn = INT_MIN;
    scanf("%d", &n);
    string unlocked, locked;
    for(int i = 0; i < n; i++) {
        string t;
        cin >> t;
        int h1, m1, s1, h2, m2, s2;
```

```
            scanf("%d:%d:%d %d:%d:%d", &h1, &m1, &s1, &h2, &m2, &s2);
            int tempIn = h1 * 3600 + m1 * 60 + s1;
            int tempOut = h2 * 3600 + m2 * 60 + s2;
            if (tempIn < minn) {
                minn = tempIn;
                unlocked = t;
            }
            if (tempOut > maxn) {
                maxn = tempOut;
                locked = t;
            }
        }
        cout << unlocked << " " << locked;
        return 0;
    }
```

1007. Maximum Subsequence Sum(25) [动态规划，最大连续子序列和]

Given a sequence of K integers { N1, N2, …, NK }. A continuous subsequence is defined to be { Ni, Ni+1, …, Nj } where 1 <= i <= j <= K. The Maximum Subsequence is the continuous subsequence which has the largest sum of its elements. For example, given sequence { -2, 11, -4, 13, -5, -2 }, its maximum subsequence is { 11, -4, 13 } with the largest sum being 20.

Now you are supposed to find the largest sum, together with the first and the last numbers of the maximum subsequence.

Input Specification:

Each input file contains one test case. Each case occupies two lines. The first line contains a positive integer K (<= 10000). The second line contains K numbers, separated by a space.

Output Specification:

For each test case, output in one line the largest sum, together with the first and the last numbers of the maximum subsequence. The numbers must be separated by one space, but there must be no extra space at the end of a line. In case that the maximum subsequence is not unique, output the one with the smallest indices i and j (as shown by the sample case). If all the K numbers are negative, then its maximum sum is defined to be 0, and you are supposed to output the first and the last numbers of the whole sequence.

Sample Input:

10

-10 1 2 3 4 -5 -23 3 7 -21

Sample Output:

10 1 4

题目大意：求最大连续子序列和，输出最大的和以及这个子序列的开始值和结束值。如果所有数都小于0，那么认为最大的和为0，并且输出首尾元素～

分析：sum为要求的最大和，temp为临时最大和，leftindex和rightindex为所求的子序列的下标，tempindex标记left的临时下标～

temp = temp + v[i]，当temp比sum大，就更新sum的值、leftindex和rightindex的值；当temp < 0，那么后面不管来什么值，都应该舍弃temp < 0前面的内容，因为负数对于总和只可能拉低总和，不可能增加总和，还不如舍弃～

舍弃后，直接令temp = 0，并且同时更新left的临时值tempindex。因为对于所有的值都为负数的情况要输出0，第一个值，最后一个值，所以在输入的时候用flag判断是不是所有的数字都是小于0的，如果是，要在输入的时候特殊处理～

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n;
    scanf("%d", &n);
    vector<int> v(n);
    int leftindex = 0, rightindex = n - 1, sum = -1, temp = 0, tempindex = 0;
    for (int i = 0; i < n; i++) {
        scanf("%d", &v[i]);
        temp = temp + v[i];
        if (temp < 0) {
            temp = 0;
            tempindex = i + 1;
        } else if (temp > sum) {
            sum = temp;
            leftindex = tempindex;
            rightindex = i;
        }
    }
    if (sum < 0) sum = 0;
    printf("%d %d %d", sum, v[leftindex], v[rightindex]);
    return 0;
}
```

1008. Elevator (20) [数学问题]

The highest building in our city has only one elevator. A request list is made up with N positive numbers. The numbers denote at which floors the elevator will stop, in specified order. It costs 6 seconds to move the elevator up one floor, and 4 seconds to move down one floor. The elevator will stay for 5 seconds at each stop.

For a given request list, you are to compute the total time spent to fulfill the requests on the list. The elevator is on the 0th floor at the beginning and does not have to return to the ground floor when the requests are fulfilled.

Input Specification:

Each input file contains one test case. Each case contains a positive integer N, followed by N positive numbers. All the numbers in the input are less than 100.

Output Specification:

For each test case, print the total time on a single line.

Sample Input:

3 2 3 1

Sample Output:

41

题目大意：电梯从0层开始向上，给出该电梯依次按顺序停的楼层数，并且已知上升需要6秒/层，下降需要4秒/层，停下来的话需要停5秒，问走完所有需要停的楼层后总共花了多少时间～

分析：累加计算输出～now表示现在的层数，a表示将要去的层数，当a > now，电梯上升，需要6 * (a – now)秒，当a < now，电梯下降，需要4 * (now – a)秒，每一次需要停5秒，最后输出累加的结果sum～

```cpp
#include <iostream>
using namespace std;
int main() {
    int a, now = 0, sum = 0;
    cin >> a;
    while(cin >> a) {
        if(a > now)
            sum = sum + 6 * (a - now);
        else
            sum = sum + 4 * (now - a);
        now = a;
        sum += 5;
    }
    cout << sum;
    return 0;
}
```

1009. Product of Polynomials (25) [模拟]

This time, you are supposed to find A*B where A and B are two polynomials.

Input Specification:

Each input file contains one test case. Each case occupies 2 lines, and each line contains the information of a polynomial: K N1 aN1 N2 aN2 … NK aNK, where K is the number of nonzero terms in the polynomial, Ni and aNi (i=1, 2, …, K) are the exponents and coefficients, respectively. It is given that 1 <= K <= 10, 0 <= NK < … < N2 < N1 <=1000.

Output Specification:

For each test case you should output the product of A and B in one line, with the same format as the input. Notice that there must be NO extra space at the end of each line. Please be accurate up to 1 decimal place.

Sample Input

2 1 2.4 0 3.2

2 2 1.5 1 0.5

Sample Output

3 3 3.6 2 6.0 1 1.6

题目大意：给出两个多项式A和B，求A*B的结果～

分析：简单模拟～

double类型的arr数组保存第一组数据，ans数组保存结果。当输入第二组数据的时候，一边进行运算一边保存结果。最后按照指数递减的顺序输出所有不为0的项～

注意：因为相乘后指数可能最大为2000，所以ans数组最大要开到2001

```
#include <iostream>
using namespace std;
int main() {
    int n1, n2, a, cnt = 0;
    scanf("%d", &n1);
    double b, arr[1001] = {0.0}, ans[2001] = {0.0};
    for(int i = 0; i < n1; i++) {
        scanf("%d %lf", &a, &b);
        arr[a] = b;
    }
    scanf("%d", &n2);
    for(int i = 0; i < n2; i++) {
        scanf("%d %lf", &a, &b);
        for(int j = 0; j < 1001; j++)
                ans[j + a] += arr[j] * b;
    }
    for(int i = 2000; i >= 0; i--)
        if(ans[i] != 0.0) cnt++;
    printf("%d", cnt);
    for(int i = 2000; i >= 0; i--)
        if(ans[i] != 0.0)
            printf(" %d %.1f", i, ans[i]);
    return 0;
}
```

1010. Radix (25) [二分法]

Given a pair of positive integers, for example, 6 and 110, can this equation 6 = 110 be true? The answer is “yes”, if 6 is a decimal number and 110 is a binary number.

Now for any pair of positive integers N1 and N2, your task is to find the radix of one number while that of the other is given.

Input Specification:

Each input file contains one test case. Each case occupies a line which contains 4 positive integers:

N1 N2 tag radix

Here N1 and N2 each has no more than 10 digits. A digit is less than its radix and is chosen from the set {0-9, a-z} where 0-9 represent the decimal numbers 0-9, and a-z represent the decimal numbers 10-35. The last number “radix” is the radix of N1 if “tag” is 1, or of N2 if “tag” is 2.

Output Specification:

For each test case, print in one line the radix of the other number so that the equation N1 = N2 is true. If the equation is impossible, print “Impossible”. If the solution is not unique, output the smallest possible radix.

Sample Input 1:

6 110 1 10

Sample Output 1:

2

Sample Input 2:

1 ab 1 2

Sample Output 2:

Impossible

分析：convert函数：给定一个数值和一个进制，将它转化为10进制。转化过程中可能产生溢出

find_radix函数：找到令两个数值相等的进制数。在查找的过程中，需要使用二分查找算法，如果使用当前进制转化得到数值比另一个大或者小于0，说明这个进制太大～

```
#include <iostream>
#include <cctype>
#include <algorithm>
#include <cmath>
using namespace std;
long long convert(string n, long long radix) {
    long long sum = 0;
    int index = 0, temp = 0;
    for (auto it = n.rbegin(); it != n.rend(); it++) {
        temp = isdigit(*it) ? *it - '0' : *it - 'a' + 10;
        sum += temp * pow(radix, index++);
    }
    return sum;
}
long long find_radix(string n, long long num) {
    char it = *max_element(n.begin(), n.end());
    long long low = (isdigit(it) ? it - '0': it - 'a' + 10) + 1;
    long long high = max(num, low);
    while (low <= high) {
        long long mid = (low + high) / 2;
        long long t = convert(n, mid);
        if (t < 0 || t > num) high = mid - 1;
        else if (t == num) return mid;
        else low = mid + 1;
```

```
        }
        return -1;
    }
    int main() {
        string n1, n2;
        long long tag = 0, radix = 0, result_radix;
        cin >> n1 >> n2 >> tag >> radix;
        result_radix = tag == 1 ? find_radix(n2, convert(n1, radix)) :
    find_radix(n1, convert(n2, radix));
        if (result_radix != -1) {
            printf("%lld", result_radix);
        } else {
            printf("Impossible");
        }
        return 0;
    }
```

1011. World Cup Betting (20) [查找元素]

With the 2010 FIFA World Cup running, football fans the world over were becoming increasingly excited as the best players from the best teams doing battles for the World Cup trophy in South Africa. Similarly, football betting fans were putting their money where their mouths were, by laying all manner of World Cup bets.

Chinese Football Lottery provided a “Triple Winning” game. The rule of winning was simple: first select any three of the games. Then for each selected game, bet on one of the three possible results — namely W for win, T for tie, and L for lose. There was an odd assigned to each result. The winner’s odd would be the product of the three odds times 65%.

For example, 3 games’ odds are given as the following:

W T L

1.1 2.5 1.7

1.2 3.0 1.6

4.1 1.2 1.1

To obtain the maximum profit, one must buy W for the 3rd game, T for the 2nd game, and T for the 1st game. If each bet takes 2 yuans, then the maximum profit would be (4.1*3.0*2.5*65%-1)*2 = 37.98 yuans (accurate up to 2 decimal places).

Input

Each input file contains one test case. Each case contains the betting information of 3 games. Each game occupies a line with three distinct odds corresponding to W, T and L.

Output

For each test case, print in one line the best bet of each game, and the maximum profit accurate up to 2 decimal places. The characters and the number must be separated by one space.

Sample Input

1.1 2.5 1.7

1.2 3.0 1.6

4.1 1.2 1.1

Sample Output

T T W 37.98

题目大意：给出三场比赛以及每场比赛的W、T、L的赔率，选取每一场比赛中赔率最大的三个数a b c，先输出三行各自选择的是W、T、L中的哪一个，然后根据计算公式 (a * b * c * 0.65 – 1) * 2 得出最大收益～

分析：以三个数一组的形式读取，读取完一组后输出最大值代表的字母，然后同时ans累乘该最大值，最后根据公式输出结果～

```
#include <cstdio>
using namespace std;
int main() {
    char c[4] = {"WTL"};
    double ans = 1.0;
    for(int i = 0; i < 3; i++) {
        double maxvalue = 0.0;
        int maxchar = 0;
        for(int j = 0; j < 3; j++) {
            double temp;
            scanf("%lf", &temp);
            if(maxvalue <= temp) {
                maxvalue = temp;
                maxchar = j;
            }
        }
        ans *= maxvalue;
        printf("%c ", c[maxchar]);
    }
    printf("%.2f", (ans * 0.65 - 1) * 2);
    return 0;
}
```

1012. The Best Rank (25) [排序]

To evaluate the performance of our first year CS majored students, we consider their grades of three courses only: C – C Programming Language, M – Mathematics (Calculus or Linear Algebra), and E – English. At the mean time, we encourage students by emphasizing on their best ranks — that is, among the four ranks with respect to the three courses and the average grade, we print the best rank for each student.

For example, The grades of C, M, E and A – Average of 4 students are given as the following:

StudentID C M E A

310101 98 85 88 90

310102 70 95 88 84

310103 82 87 94 88

310104 91 91 91 91

Then the best ranks for all the students are No.1 since the 1st one has done the best in C Programming Language, while the 2nd one in Mathematics, the 3rd one in English, and the last one in average.

Input

Each input file contains one test case. Each case starts with a line containing 2 numbers N and M (<=2000), which are the total number of students, and the number of students who would check their ranks, respectively. Then N lines follow, each contains a student ID which is a string of 6 digits, followed by the three integer grades (in the range of [0, 100]) of that student in the order of C, M and E. Then there are M lines, each containing a student ID.

Output

For each of the M students, print in one line the best rank for him/her, and the symbol of the corresponding rank, separated by a space.

The priorities of the ranking methods are ordered as A > C > M > E. Hence if there are two or more ways for a student to obtain the same best rank, output the one with the highest priority.

If a student is not on the grading list, simply output “N/A”.

Sample Input

5 6

310101 98 85 88

310102 70 95 88

310103 82 87 94

310104 91 91 91

310105 85 90 90

310101

310102

310103

310104

310105

999999

Sample Output

1 C

1 M

1 E

1 A

3 A

N/A

题目大意：现已知n个考生的3门分数，平均分可以按照这三门算出来。然后分别对这四个分数从高到低排序，这样对每个考生来说有4个排名。k个查询，对于每一个学生id，输出当前id学生的最好的排名和它对应的分数，如果名次相同，按照A>C>M>E的顺序输出～如果当前id不存在，输出N/A～

分析：

1、用结构体存储学生的id、四门成绩、四门排名、最好的排名的对应的科目下标～

2、排名并列应该1、1、3、4、5，而不是1、1、2、3、4，否则会有一个测试点不过

3、平均分是四舍五入的，所以需要按照+0.5后取整，保证是四舍五入的（听说不四舍五入也能通过...）

4、存储的时候就按照ACME的顺序存储可以简化程序逻辑～

5、用exist数组保存当前id是否存在，这个id对应的stu结构体的下标是多少。用i+1可以保证为0的都是不存在的可以直接输出N/A，其余不为0的保存的值是对应的结构体index + 1的值～

```
#include <cstdio>
#include <algorithm>
using namespace std;
struct node {
    int id, best;
    int score[4], rank[4];
}stu[2005];
int exist[1000000], flag = -1;
bool cmp1(node a, node b) {return a.score[flag] > b.score[flag];}
int main() {
    int n, m, id;
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++) {
        scanf("%d %d %d %d", &stu[i].id, &stu[i].score[1], &stu[i].score[2], &stu[i].score[3]);
        stu[i].score[0] = (stu[i].score[1] + stu[i].score[2] + stu[i].score[3]) / 3.0 + 0.5;
    }
    for(flag = 0; flag <= 3; flag++) {
        sort(stu, stu + n, cmp1);
        stu[0].rank[flag] = 1;
        for(int i = 1; i < n; i++) {
            stu[i].rank[flag] = i + 1;
            if(stu[i].score[flag] == stu[i-1].score[flag])
                stu[i].rank[flag] = stu[i-1].rank[flag];
        }
    }
    for(int i = 0; i < n; i++) {
        exist[stu[i].id] = i + 1;
```

```
            stu[i].best = 0;
            int minn = stu[i].rank[0];
            for(int j = 1; j <= 3; j++) {
                if(stu[i].rank[j] < minn) {
                    minn = stu[i].rank[j];
                    stu[i].best = j;
                }
            }
        }
        char c[5] = {'A', 'C', 'M', 'E'};
        for(int i = 0; i < m; i++) {
            scanf("%d", &id);
            int temp = exist[id];
            if(temp) {
                int best = stu[temp-1].best;
                printf("%d %c\n", stu[temp-1].rank[best], c[best]);
            } else {
                printf("N/A\n");
            }
        }
        return 0;
    }
```

1013. Battle Over Cities (25) [图的遍历，统计连通分量的个数，DFS]

It is vitally important to have all the cities connected by highways in a war. If a city is occupied by the enemy, all the highways from/toward that city are closed. We must know immediately if we need to repair any other highways to keep the rest of the cities connected. Given the map of cities which have all the remaining highways marked, you are supposed to tell the number of highways need to be repaired, quickly.

For example, if we have 3 cities and 2 highways connecting city1-city2 and city1-city3. Then if city1 is occupied by the enemy, we must have 1 highway repaired, that is the highway city2-city3.

Input

Each input file contains one test case. Each case starts with a line containing 3 numbers N (<1000), M and K, which are the total number of cities, the number of remaining highways, and the number of cities to be checked, respectively. Then M lines follow, each describes a highway by 2 integers, which are the numbers of the cities the highway connects. The cities are numbered from 1 to N. Finally there is a line containing K numbers, which represent the cities we concern.

Output

For each of the K cities, output in a line the number of highways need to be repaired if that city is lost.

Sample Input

3 2 3

1 2

1 3

1 2 3

Sample Output

1

0

0

题目大意：给出n个城市之间有相互连接的m条道路，当删除一个城市和其连接的道路的时候，问其他几个剩余的城市至少要添加多少个路线才能让它们重新变为连通图

分析：添加的最少的路线，就是他们的连通分量数-1，因为当a个互相分立的连通分量需要变为连通图的时候，只需要添加a-1个路线，就能让他们相连。所以这道题就是求去除了某个结点之后其他的图所拥有的连通分量数

使用邻接矩阵存储，对于每一个被占领的城市，去除这个城市结点，就是把它标记为已经访问过，这样在深度优先遍历的时候，对于所有未访问的结点进行遍历，就能求到所有的连通分量的个数～记得因为有很多个要判断的数据，每一次输入被占领的城市之前要把visit数组置为false～～

```
#include <cstdio>
#include <algorithm>
using namespace std;
int v[1010][1010];
bool visit[1010];
int n;
void dfs(int node) {
    visit[node] = true;
    for(int i = 1; i <= n; i++) {
        if(visit[i] == false && v[node][i] == 1)
            dfs(i);
    }
}
int main() {
    int m, k, a, b;
    scanf("%d%d%d", &n, &m, &k);
    for(int i = 0; i < m; i++) {
        scanf("%d%d", &a, &b);
        v[a][b] = v[b][a] = 1;
    }
    for(int i = 0; i < k; i++) {
        fill(visit, visit + 1010, false);
        scanf("%d", &a);
        int cnt = 0;
        visit[a] = true;
        for(int j = 1; j <= n; j++) {
            if(visit[j] == false) {
                dfs(j);
                cnt++;
            }
        }
        printf("%d\n", cnt - 1);
    }
```

```
34        return 0;
35    }
```

1014. Waiting in Line (30) [queue的应用]

Suppose a bank has N windows open for service. There is a yellow line in front of the windows which devides the waiting area into two parts. The rules for the customers to wait in line are:

The space inside the yellow line in front of each window is enough to contain a line with M customers. Hence when all the N lines are full, all the customers after (and including) the (NM+1)st one will have to wait in a line behind the yellow line.

Each customer will choose the shortest line to wait in when crossing the yellow line. If there are two or more lines with the same length, the customer will always choose the window with the smallest number.

Customer[i] will take T[i] minutes to have his/her transaction processed.

The first N customers are assumed to be served at 8:00am.

Now given the processing time of each customer, you are supposed to tell the exact time at which a customer has his/her business done.

For example, suppose that a bank has 2 windows and each window may have 2 custmers waiting inside the yellow line. There are 5 customers waiting with transactions taking 1, 2, 6, 4 and 3 minutes, respectively. At 08:00 in the morning, customer1 is served at window1 while customer2 is served at window2. Customer3 will wait in front of window1 and customer4 will wait in front of window2. Customer5 will wait behind the yellow line.

At 08:01, customer1 is done and customer5 enters the line in front of window1 since that line seems shorter now. Customer2 will leave at 08:02, customer4 at 08:06, customer3 at 08:07, and finally customer5 at 08:10.

Input

Each input file contains one test case. Each case starts with a line containing 4 positive integers: N (<=20, number of windows), M (<=10, the maximum capacity of each line inside the yellow line), K (<=1000, number of customers), and Q (<=1000, number of customer queries).

The next line contains K positive integers, which are the processing time of the K customers.

The last line contains Q positive integers, which represent the customers who are asking about the time they can have their transactions done. The customers are numbered from 1 to K.

Output

For each of the Q customers, print in one line the time at which his/her transaction is finished, in the format HH:MM where HH is in [08, 17] and MM is in [00, 59]. Note that since the bank is closed everyday after 17:00, for those customers who cannot be served before 17:00, you must output “Sorry” instead.

Sample Input

2 2 7 5

1 2 6 4 3 534 2

3 4 5 6 7

Sample Output

08:07

08:06

08:10

17:00

Sorry

题目大意：n个窗口，每个窗口可以排队m人。有k位用户需要服务，给出了每位用户需要的minute数，所有客户在8点开始服务，如果有窗口还没排满就入队，否则就在黄线外等候。如果有某一列有一个用户走了服务完毕了，黄线外的人就进来一个。如果同时就选窗口数小的。求q个人的服务结束时间。

如果一个客户在17:00以及以后还没有开始服务（此处不是结束服务是开始17:00）就不再服务输出sorry；如果这个服务已经开始了，无论时间多长都要等他服务完毕。

分析：设立结构体，里面包含poptime为队首的人出队（结束）的时间，和endtime为队尾的人结束的时间。poptime是为了让黄线外的人可以计算出哪一个队列先空出人来（poptime最小的那个先有人服务完毕），endtime是为了入队后加上自己本身的服务所需时间可以计算出自己多久才能被服务完毕～且前一个人的endtime可以得知自己是不是需要被Sorry（如果前一个人服务结束时间超过17:00，自己当前入队的人就是sorry），还有一个queue表示所有当前该窗口的排队队列。

对于前m*n个人，也就是排的下的情况下，所有人依次到窗口前面排队。对于m*n之后的人，当前人选择poptime最短的入队，让队伍的第一个人出列），如果前面一个人导致的endtime超过17点就标记自己的sorry为true。

计算时间的时候按照分钟计算，最后再考虑08点开始和转换为小时分钟的形式会比较简便。

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct node {
    int poptime, endtime;
    queue<int> q;
};
int main() {
    int n, m, k, q, index = 1;
    scanf("%d%d%d%d", &n, &m, &k, &q);
    vector<int> time(k + 1), result(k + 1);
    for(int i = 1; i <= k; i++) {
        scanf("%d", &time[i]);
    }
    vector<node> window(n + 1);
    vector<bool> sorry(k + 1, false);
    for(int i = 1; i <= m; i++) {
        for(int j = 1; j <= n; j++) {
            if(index <= k) {
```

```
                window[j].q.push(time[index]);
                if(window[j].endtime >= 540)
                    sorry[index] = true;
                window[j].endtime += time[index];
                if(i == 1)
                    window[j].poptime = window[j].endtime;
                result[index] = window[j].endtime;
                index++;
            }

        }
    }
    while(index <= k) {
        int tempmin = window[1].poptime, tempwindow = 1;
        for(int i = 2; i <= n; i++) {
            if(window[i].poptime < tempmin) {
                tempwindow = i;
                tempmin = window[i].poptime;
            }
        }
        window[tempwindow].q.pop();
        window[tempwindow].q.push(time[index]);
        window[tempwindow].poptime +=  window[tempwindow].q.front();
        if(window[tempwindow].endtime >= 540)
            sorry[index] = true;
        window[tempwindow].endtime += time[index];
        result[index] = window[tempwindow].endtime;
        index++;
    }
    for(int i = 1; i <= q; i++) {
        int query, minute;
        scanf("%d", &query);
        minute = result[query];
        if(sorry[query] == true) {
            printf("Sorry\n");
        } else {
            printf("%02d:%02d\n",(minute + 480) / 60, (minute + 480) % 60);
        }
    }
    return 0;
}
```

1015. Reversible Primes (20) [素数]

A reversible prime in any number system is a prime whose “reverse” in that number system is also a prime. For example in the decimal system 73 is a reversible prime because its reverse 37 is also a prime.

Now given any two positive integers N (< 105) and D (1 < D <= 10), you are supposed to tell if N is a reversible prime with radix D.

Input Specification:

The input file consists of several test cases. Each case occupies a line which contains two integers N and D. The input is finished by a negative N.

Output Specification:

For each test case, print in one line “Yes” if N is a reversible prime with radix D, or “No” if not.

Sample Input:

73 10

23 2

23 10

-2

Sample Output:

Yes

Yes

No

题目大意：如果一个数本身是素数，而且在d进制下反转后的数在十进制下也是素数，就输出Yes，否则就输出No

分析：判断输入是否为负数，判断n是否为素数，把n转换为d进制再反过来转换为10进制，判断是否为素数

```
#include <cstdio>
#include <cmath>
using namespace std;
bool isprime(int n) {
    if(n <= 1) return false;
    int sqr = int(sqrt(n * 1.0));
    for(int i = 2; i <= sqr; i++) {
        if(n % i == 0)
            return false;
    }
    return true;
}
int main() {
    int n, d;
    while(scanf("%d", &n) != EOF) {
        if(n < 0) break;
        scanf("%d", &d);
        if(isprime(n) == false) {
            printf("No\n");
            continue;
        }
        int len = 0, arr[100];
        do{
            arr[len++] = n % d;
            n = n / d;
```

```
        }while(n != 0);
        for(int i = 0; i < len; i++)
            n = n * d + arr[i];
        printf("%s", isprime(n) ? "Yes\n" : "No\n");
    }
    return 0;
}
```

1016. Phone Bills (25) [排序]

A long-distance telephone company charges its customers by the following rules:

Making a long-distance call costs a certain amount per minute, depending on the time of day when the call is made. When a customer starts connecting a long-distance call, the time will be recorded, and so will be the time when the customer hangs up the phone. Every calendar month, a bill is sent to the customer for each minute called (at a rate determined by the time of day). Your job is to prepare the bills for each month, given a set of phone call records.

Input Specification:

Each input file contains one test case. Each case has two parts: the rate structure, and the phone call records.

The rate structure consists of a line with 24 non-negative integers denoting the toll (cents/minute) from 00:00 – 01:00, the toll from 01:00 – 02:00, and so on for each hour in the day.

The next line contains a positive number N (<= 1000), followed by N lines of records. Each phone call record consists of the name of the customer (string of up to 20 characters without space), the time and date (mm:dd:hh:mm), and the word “on-line” or “off-line”.

For each test case, all dates will be within a single month. Each “on-line” record is paired with the chronologically next record for the same customer provided it is an “off-line” record. Any “on-line” records that are not paired with an “off-line” record are ignored, as are “off-line” records not paired with an “on-line” record. It is guaranteed that at least one call is well paired in the input. You may assume that no two records for the same customer have the same time. Times are recorded using a 24-hour clock.

Output Specification:

For each test case, you must print a phone bill for each customer.

Bills must be printed in alphabetical order of customers’ names. For each customer, first print in a line the name of the customer and the month of the bill in the format shown by the sample. Then for each time period of a call, print in one line the beginning and ending time and date (dd:hh:mm), the lasting time (in minute) and the charge of the call. The calls must be listed in chronological order. Finally, print the total charge for the month in the format shown by the sample.

Sample Input:

10 10 10 10 10 10 20 20 20 15 15 15 15 15 15 15 20 30 20 15 15 10 10 10

10

CYLL 01:01:06:01 on-line

CYLL 01:28:16:05 off-line

CYJJ 01:01:07:00 off-line

CYLL 01:01:08:03 off-line

CYJJ 01:01:05:59 on-line

aaa 01:01:01:03 on-line

aaa 01:02:00:01 on-line

CYLL 01:28:15:41 on-line

aaa 01:05:02:24 on-line

aaa 01:04:23:59 off-line

Sample Output:

CYJJ 01

01:05:59 01:07:00 61 $12.10

Total amount: $12.10

CYLL 01

01:06:01 01:08:03 122 $24.40

28:15:41 28:16:05 24 $3.85

Total amount: $28.25

aaa 01

02:00:01 04:23:59 4318 $638.80

Total amount: $638.80

分析：将给出的数据先按照姓名排序，再按照时间的先后顺序排列，这样遍历的时候，前后两个名字相同且前面的状态为on-line后面一个的状态为off-line的就是合格数据～

注意：【关于最后一个测试点】计算费用从00:00:00到dd:hh:mm计算可以避免跨天的问题，比如01:12:00到02:02:00

```
#include <iostream>
#include <map>
#include <vector>
#include <algorithm>
using namespace std;
struct node {
    string name;
    int status, month, time, day, hour, minute;
};
bool cmp(node a, node b) {
    return a.name != b.name ? a.name < b.name : a.time < b.time;
}
double billFromZero(node call, int *rate) {
    double total = rate[call.hour] * call.minute + rate[24] * 60 * call.day;
```

```
    for (int i = 0; i < call.hour; i++)
        total += rate[i] * 60;
    return total / 100.0;
}
int main() {
    int rate[25] = {0}, n;
    for (int i = 0; i < 24; i++) {
        scanf("%d", &rate[i]);
        rate[24] += rate[i];
    }
    scanf("%d", &n);
    vector<node> data(n);
    for (int i = 0; i < n; i++) {
        cin >> data[i].name;
        scanf("%d:%d:%d:%d", &data[i].month, &data[i].day, &data[i].hour,
&data[i].minute);
        string temp;
        cin >> temp;
        data[i].status = (temp == "on-line") ? 1 : 0;
        data[i].time = data[i].day * 24 * 60 + data[i].hour * 60 +
data[i].minute;
    }
    sort(data.begin(), data.end(), cmp);
    map<string, vector<node> > custom;
    for (int i = 1; i < n; i++) {
        if (data[i].name == data[i - 1].name && data[i - 1].status == 1 &&
data[i].status == 0) {
            custom[data[i - 1].name].push_back(data[i - 1]);
            custom[data[i].name].push_back(data[i]);
        }
    }
    for (auto it : custom) {
        vector<node> temp = it.second;
        cout << it.first;
        printf(" %02d\n", temp[0].month);
        double total = 0.0;
        for (int i = 1; i < temp.size(); i += 2) {
            double t = billFromZero(temp[i], rate) - billFromZero(temp[i - 1],
rate);
            printf("%02d:%02d:%02d %02d:%02d:%02d %d $%.2f\n", temp[i - 1].day,
temp[i - 1].hour, temp[i - 1].minute, temp[i].day, temp[i].hour,
temp[i].minute, temp[i].time - temp[i - 1].time, t);
            total += t;
        }
        printf("Total amount: $%.2f\n", total);
    }
    return 0;
}
```

1017. Queueing at Bank (25) [模拟]

Suppose a bank has K windows open for service. There is a yellow line in front of the windows which devides the waiting area into two parts. All the customers have to wait in line behind the yellow line, until it is his/her turn to be served and there is a window available. It is assumed that no window can be occupied by a single customer for more than 1 hour.

Now given the arriving time T and the processing time P of each customer, you are supposed to tell the average waiting time of all the customers.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 numbers: N (<=10000) – the total number of customers, and K (<=100) – the number of windows. Then N lines follow, each contains 2 times: HH:MM:SS – the arriving time, and P – the processing time in minutes of a customer. Here HH is in the range [00, 23], MM and SS are both in [00, 59]. It is assumed that no two customers arrives at the same time.

Notice that the bank opens from 08:00 to 17:00. Anyone arrives early will have to wait in line till 08:00, and anyone comes too late (at or after 17:00:01) will not be served nor counted into the average.

Output Specification:

For each test case, print in one line the average waiting time of all the customers, in minutes and accurate up to 1 decimal place.

Sample Input:

7 3

07:55:00 16

17:00:01 2

07:59:59 15

08:01:00 60

08:00:00 30

08:00:02 2

08:03:00 10

Sample Output:

8.2

题目大意：有n个客户，k个窗口。已知每个客户的到达时间和需要的时长，如果有窗口就依次过去，如果没有窗口就在黄线外等候（黄线外只有一个队伍，先来先服务），求客户的平均等待时长。银行开放时间为8点到17点，再8点之前不开门，8点之前来的人都要等待，在17点后来的人不被服务。

分析：客户结构体node，里面有come为到达时间和time为需要服务的时长。先将所有满足条件的（到来时间点在17点之前的）客户放入数组中，数组的长度就是需要服务的客户的个数。window数组表示某个窗口的结束时间，每一个客户到来的时候，选择最早结束时间的窗口，如果最早结束时间比他还早，那么他一来就能被服务，更新window的值；如果最早结束时间比他晚，他需要等待，累加等待的时间，然后更新window的值。

一开始所有窗口的值都为8点整。对所有客户要先进行排序，按来的时间的早晚排序，之后再先来先服务。因为涉及分钟和秒数，可以把所有时间暂时先化解成秒数的形式，便于计算。如果一个可以被服务的客户都没有，需要直接输出0.0，因为任何数除以0都没有意义。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct node {
    int come, time;
} tempcustomer;
bool cmp1(node a, node b) {
    return a.come < b.come;
}
int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    vector<node> custom;
    for(int i = 0; i < n; i++) {
        int hh, mm, ss, time;
        scanf("%d:%d:%d %d", &hh, &mm, &ss, &time);
        int cometime = hh * 3600 + mm * 60 + ss;
        if(cometime > 61200) continue;
        tempcustomer = {cometime, time * 60};
        custom.push_back(tempcustomer);
    }
    sort(custom.begin(), custom.end(), cmp1);
    vector<int> window(k, 28800);
    double result = 0.0;
    for(int i = 0; i < custom.size(); i++) {
        int tempindex = 0, minfinish = window[0];
        for(int j = 1; j < k; j++) {
            if(minfinish > window[j]) {
                minfinish = window[j];
                tempindex = j;
            }
        }
        if(window[tempindex] <= custom[i].come) {
            window[tempindex] = custom[i].come + custom[i].time;
        } else {
            result += (window[tempindex] - custom[i].come);
            window[tempindex] += custom[i].time;
        }
    }
    if(custom.size() == 0)
        printf("0.0");
    else
        printf("%.1f", result / 60.0 / custom.size());
    return 0;
}
```

1018. Public Bike Management (30) [Dijkstra算法 + DFS]

There is a public bike service in Hangzhou City which provides great convenience to the tourists from all over the world. One may rent a bike at any station and return it to any other stations in the city.

The Public Bike Management Center (PBMC) keeps monitoring the real-time capacity of all the stations. A station is said to be in perfect condition if it is exactly half-full. If a station is full or empty, PBMC will collect or send bikes to adjust the condition of that station to perfect. And more, all the stations on the way will be adjusted as well.

When a problem station is reported, PBMC will always choose the shortest path to reach that station. If there are more than one shortest path, the one that requires the least number of bikes sent from PBMC will be chosen.

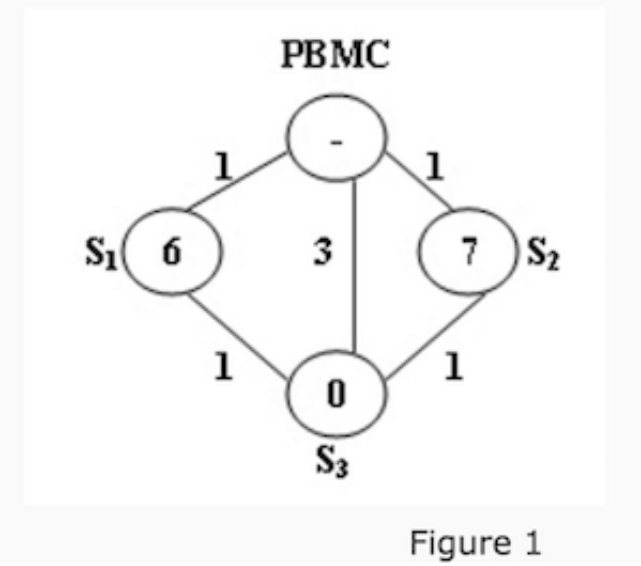


Figure 1

1. PBMC -> S1 -> S3. In this case, 4 bikes must be sent from PBMC, because we can collect 1 bike from S1 and then take 5 bikes to S3, so that both stations will be in perfect conditions.
2. PBMC -> S2 -> S3. This path requires the same time as path 1, but only 3 bikes sent from PBMC and hence is the one that will be chosen.

Input Specification:

Each input file contains one test case. For each case, the first line contains 4 numbers: Cmax (<= 100), always an even number, is the maximum capacity of each station; N (<= 500), the total number of stations; Sp, the index of the problem station (the stations are numbered from 1 to N, and PBMC is represented by the vertex 0); and M, the number of roads. The second line contains N non-negative numbers Ci (i=1,…N) where each Ci is the current number of bikes at Si respectively. Then M lines follow, each contains 3 numbers: Si, Sj, and Tij which describe the time Tij taken to move betwen stations Si and Sj. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print your results in one line. First output the number of bikes that PBMC must send. Then after one space, output the path in the format: 0->S1->…->Sp. Finally after another space, output the number of bikes that we must take back to PBMC after the condition of Sp is adjusted to perfect.

Note that if such a path is not unique, output the one that requires minimum number of bikes that we must take back to PBMC. The judge’s data guarantee that such a path is unique.

Sample Input:

10 3 3 5

6 7 0

0 1 1

0 2 1

0 3 3

1 3 1

2 3 1

Sample Output:

3 0->2->3 0

题目大意：每个自行车车站的最大容量为一个偶数cmax，如果一个车站里面自行车的数量恰好为cmax / 2，那么称处于完美状态。如果一个车站容量是满的或者空的，控制中心（处于结点0处）就会携带或者从路上收集一定数量的自行车前往该车站，一路上会让所有的车站沿途都达到完美。现在给出cmax，车站的数量n，问题车站sp，m条边，还有距离，求最短路径。如果最短路径有多个，求能带的最少的自行车数目的那条。如果还是有很多条不同的路，那么就找一个从车站带回的自行车数目最少的（带回的时候是不调整的）～

分析：Dijkstra + DFS。如果只有Dijkstra是不可以的，因为minNeed和minBack在路径上的传递不满足最优子结构，不是简单的相加的过程，只有在所有路径都确定了之后才能区选择最小的need和最小的back～

Dijkstra求最短路径，dfs求minNeed和minBack和path，dfs的时候模拟一遍需要调整的过程，求出最后得到的need和back，与minNeed和minBack比较然后根据情况更新path，最后输出minNeed path 和 minBack，记得path是从最后一个结点一直到第一个结点的，所以要倒着输出～

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
const int inf = 99999999;
int cmax, n, sp, m;
int minNeed = inf, minBack = inf;
int e[510][510], dis[510], weight[510];
bool visit[510];
vector<int> pre[510];
vector<int> path, temppath;
void dfs(int v) {
    temppath.push_back(v);
    if(v == 0) {
        int need = 0, back = 0;
        for(int i = temppath.size() - 1; i >= 0; i--) {
            int id = temppath[i];
            if(weight[id] > 0) {
                back += weight[id];
            } else {
                if(back > (0 - weight[id])) {
```

```
                    back += weight[id];
                } else {
                    need += ((0 - weight[id]) - back);
                    back = 0;
                }
            }
        }
        if(need < minNeed) {
            minNeed = need;
            minBack = back;
            path = temppath;
        } else if(need == minNeed && back < minBack) {
            minBack = back;
            path = temppath;
        }
        temppath.pop_back();
        return ;
    }
    for(int i = 0; i < pre[v].size(); i++)
        dfs(pre[v][i]);
    temppath.pop_back();
}
int main() {
    fill(e[0], e[0] + 510 * 510, inf);
    fill(dis, dis + 510, inf);
    scanf("%d%d%d%d", &cmax, &n, &sp, &m);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &weight[i]);
        weight[i] = weight[i] - cmax / 2;
    }
    for(int i = 0; i < m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        scanf("%d", &e[a][b]);
        e[b][a] = e[a][b];
    }
    dis[0] = 0;
    for(int i = 0; i <= n; i++) {
        int u = -1, minn = inf;
        for(int j = 0; j <= n; j++) {
            if(visit[j] == false && dis[j] < minn) {
                u = j;
                minn = dis[j];
            }
        }
        if(u == -1) break;
        visit[u] = true;
        for(int v = 0; v <= n; v++) {
            if(visit[v] == false && e[u][v] != inf) {
                if(dis[v] > dis[u] + e[u][v]) {
                    dis[v] = dis[u] + e[u][v];
                    pre[v].clear();
```

```
                    pre[v].push_back(u);
                }else if(dis[v] == dis[u] + e[u][v]) {
                    pre[v].push_back(u);
                }
            }
        }
    }
    dfs(sp);
    printf("%d 0", minNeed);
    for(int i = path.size() - 2; i >= 0; i--)
        printf("->%d", path[i]);
    printf(" %d", minBack);
    return 0;
}
```

1019. General Palindromic Number (20) [回文数]

A number that will be the same when it is written forwards or backwards is known as a Palindromic Number. For example, 1234321 is a palindromic number. All single digit numbers are palindromic numbers.

Although palindromic numbers are most often considered in the decimal system, the concept of palindromicity can be applied to the natural numbers in any numeral system. Consider a number N > 0 in base b >= 2, where it is written in standard notation with k+1 digits ai as the sum of (aibi) for i from 0 to k. Here, as usual, 0 <= ai < b for all i and ak is non-zero. Then N is palindromic if and only if ai = ak-i for all i. Zero is written 0 in any base and is also palindromic by definition.

Given any non-negative decimal integer N and a base b, you are supposed to tell if N is a palindromic number in base b.

Input Specification:

Each input file contains one test case. Each case consists of two non-negative numbers N and b, where 0 <= N <= 109 is the decimal number and 2 <= b <= 109 is the base. The numbers are separated by a space.

Output Specification:

For each test case, first print in one line “Yes” if N is a palindromic number in base b, or “No” if not. Then in the next line, print N as the number in base b in the form “ak ak-1 … a0”. Notice that there must be no extra space at the end of output.

Sample Input 1:

27 2

Sample Output 1:

Yes

1 1 0 1 1

Sample Input 2:

121 5

Sample Output 2:

No

4 4 1

题目大意：给出两个整数a和b，问十进制的a在b进制下是否为回文数。是的话输出Yes，不是输出No。并且输出a在b进制下的表示，以空格隔开

分析：将a转换为b进制形式，保存在int的数组里面，比较数组左右两端是否对称。

注意：如果是0，要输出Yes和0

```
#include <cstdio>
using namespace std;
int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    int arr[40], index = 0;
    while(a != 0) {
        arr[index++] = a % b;
        a = a / b;
    }
    int flag = 0;
    for(int i = 0; i < index / 2; i++) {
        if(arr[i] != arr[index-i-1]) {
            printf("No\n");
            flag = 1;
            break;
        }
    }
    if(!flag) printf("Yes\n");
    for(int i = index - 1; i >= 0; i--) {
        printf("%d", arr[i]);
        if(i != 0) printf(" ");
    }
    if(index == 0)
        printf("0");
    return 0;
}
```

1020. Tree Traversals (25) [二叉树的遍历，后序中序转层序]

Suppose that all the keys in a binary tree are distinct positive integers. Given the postorder and inorder traversal sequences, you are supposed to output the level order traversal sequence of the corresponding binary tree.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=30), the total number of nodes in the binary tree. The second line gives the postorder sequence and the third line gives the inorder sequence. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the level order traversal sequence of the corresponding binary tree. All the numbers in a line must be separated by exactly one space, and there must be no extra space at the end of the line.

Sample Input:

7

2 3 1 5 7 6 4 1 2 3 4 5 6 7

Sample Output:

4 1 6 3 5 7 2

题目大意：给定一棵二叉树的后序遍历和中序遍历，请你输出其层序遍历的序列。这里假设键值都是互不相等的正整数。

分析：与后序中序转换为前序的代码相仿（无须构造二叉树再进行广度优先搜索～），只不过加一个变量index，表示当前根结点在二叉树中所对应的下标（从0开始），所以进行一次输出先序的递归的时候，把节点的index和value放进结构体里，再装进容vector里，这样在递归完成后，vector中按照index排序就是层序遍历的顺序～

如果你不知道如何将后序和中序转换为先序，请看：https://www.liuchuo.net/archives/2090

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
struct node {
    int index, value;
};
bool cmp(node a, node b) {
    return a.index < b.index;
}
vector<int> post, in;
vector<node> ans;
void pre(int root, int start, int end, int index) {
    if (start > end) return;
    int i = start;
    while (i < end && in[i] != post[root]) i++;
    ans.push_back({index, post[root]});
    pre(root - 1 - end + i, start, i - 1, 2 * index + 1);
    pre(root - 1, i + 1, end, 2 * index + 2);
}
int main() {
    int n;
    scanf("%d", &n);
    post.resize(n);
    in.resize(n);
    for (int i = 0; i < n; i++) scanf("%d", &post[i]);
    for (int i = 0; i < n; i++) scanf("%d", &in[i]);
    pre(n - 1, 0, n - 1, 0);
    sort(ans.begin(), ans.end(), cmp);
    for (int i = 0; i < ans.size(); i++) {
```

```
            if (i != 0) cout << " ";
            cout << ans[i].value;
        }
        return 0;
    }
```

1021. Deepest Root (25) [图的遍历，DFS，计算连通分量的个数]

A graph which is connected and acyclic can be considered a tree. The height of the tree depends on the selected root. Now you are supposed to find the root that results in a highest tree. Such a root is called the deepest root.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<=10000) which is the number of nodes, and hence the nodes are numbered from 1 to N. Then N-1 lines follow, each describes an edge by given the two adjacent nodes' numbers.

Output Specification:

For each test case, print each of the deepest roots in a line. If such a root is not unique, print them in increasing order of their numbers. In case that the given graph is not a tree, print "Error: K components" where K is the number of connected components in the graph.

Sample Input 1:

5

1 2

1 3

1 4

2 5

Sample Output 1:

3

4

5

Sample Input 2:

5

1 3

1 4

2 5

3 4

Sample Output 2:

Error: 2 components

题目大意：给出n个结点（1~n）之间的n条边，问是否能构成一棵树，如果不能构成则输出它有的连通分量个数，如果能构成一棵树，输出能构成最深的树的高度时，树的根结点。如果有多个，按照从小到大输出。

分析：首先深度优先搜索判断它有几个连通分量。如果有多个，那就输出Error: x components，如果只有一个，就两次深度优先搜索，先从一个结点dfs后保留最高高度拥有的结点们，然后从这些结点中的其中任意一个开始dfs得到最高高度的结点们，这两个结点集合的并集就是所求

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;
int n, maxheight = 0;
vector<vector<int>> v;
bool visit[10010];
set<int> s;
vector<int> temp;
void dfs(int node, int height) {
    if(height > maxheight) {
        temp.clear();
        temp.push_back(node);
        maxheight = height;
    } else if(height == maxheight){
        temp.push_back(node);
    }
    visit[node] = true;
    for(int i = 0; i < v[node].size(); i++) {
        if(visit[v[node][i]] == false)
            dfs(v[node][i], height + 1);
    }
}
int main() {
    scanf("%d", &n);
    v.resize(n + 1);
    int a, b, cnt = 0, s1 = 0;
    for(int i = 0; i < n - 1; i++) {
        scanf("%d%d", &a, &b);
        v[a].push_back(b);
        v[b].push_back(a);
    }
    for(int i = 1; i <= n; i++) {
        if(visit[i] == false) {
            dfs(i, 1);
            if(i == 1) {
                if (temp.size() != 0) s1 = temp[0];
                for(int j = 0; j < temp.size(); j++)
                    s.insert(temp[j]);
            }
            cnt++;
```

```
            }
        }
        if(cnt >= 2) {
            printf("Error: %d components", cnt);
        } else {
            temp.clear();
            maxheight = 0;
            fill(visit, visit + 10010, false);
            dfs(s1, 1);
            for(int i = 0; i < temp.size(); i++)
                s.insert(temp[i]);
            for(auto it = s.begin(); it != s.end(); it++)
                printf("%d\n", *it);
        }
        return 0;
    }
```

1022. Digital Library (30) [map映射，STL的使用]

A Digital Library contains millions of books, stored according to their titles, authors, key words of their abstracts, publishers, and published years. Each book is assigned an unique 7-digit number as its ID. Given any query from a reader, you are supposed to output the resulting books, sorted in increasing order of their ID's.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<=10000) which is the total number of books. Then N blocks follow, each contains the information of a book in 6 lines:

Line #1: the 7-digit ID number;

Line #2: the book title — a string of no more than 80 characters;

Line #3: the author — a string of no more than 80 characters;

Line #4: the key words — each word is a string of no more than 10 characters without any white space, and the keywords are separated by exactly one space;

Line #5: the publisher — a string of no more than 80 characters;

Line #6: the published year — a 4-digit number which is in the range [1000, 3000].

It is assumed that each book belongs to one author only, and contains no more than 5 key words; there are no more than 1000 distinct key words in total; and there are no more than 1000 distinct publishers.

After the book information, there is a line containing a positive integer M (<=1000) which is the number of user's search queries. Then M lines follow, each in one of the formats shown below:

1: a book title

2: name of an author

3: a key word

4: name of a publisher

5: a 4-digit number representing the year

Output Specification:

For each query, first print the original query in a line, then output the resulting book ID's in increasing order, each occupying a line. If no book is found, print "Not Found" instead.

Sample Input:

3

1111111

The Testing Book

Yue Chen

test code debug sort keywords

ZUCS Print

2011

3333333

Another Testing Book

Yue Chen

test code sort keywords

ZUCS Print2

2012

2222222

The Testing Book

CYLL

keywords debug book

ZUCS Print2

2011

6

1: The Testing Book

2: Yue Chen

3: keywords

4: ZUCS Print

5: 2011

3: blablabla

Sample Output:

1: The Testing Book

1111111

2222222

2: Yue Chen

1111111

3333333

3: keywords

1111111

2222222

3333333

4: ZUCS Print

1111111

5: 2011

1111111

2222222

3: blablabla

Not Found

题目大意：模拟数字图书馆的查询功能。会给出n本书的信息，以及m个需要查询的命令，数字标号对应相应的命令，数字编号后面的字符串是查询的搜索词，要求输出这行命令以及输出满足条件的书的id，如果一个都没有找到，输出Not Found

分析：1、对除了id之外的其他信息都建立一个map<string, set>，把相应的id插入对应搜索词的map的集合里面，形成一个信息对应一个集合，集合里面是复合条件的书的id

2、因为对于输入的关键词（可以重复，算是书本对应的tag标签吧～）没有给定关键词的个数，可以使用while(cin >> s)并且判断c = getchar()，c是否等于\n，如果是再退出循环～

3、建立query，通过传参的形式可以将不同的map名称统一化，先要判断map.find()和m.end()是否相等，如果不等再去遍历整个map，输出所有满足条件的id，如果相等就说明不存在这个搜索词对应的id，那么就要输出Not Found～

4、传参一定要用引用，否则最后一组数据可能会超时

```
#include <iostream>
#include <map>
#include <set>
using namespace std;
map<string, set<int> > title, author, key, pub, year;
```

```
void query(map<string, set<int> > &m, string &str) {
    if(m.find(str) != m.end()) {
        for(auto it = m[str].begin(); it != m[str].end(); it++)
            printf("%07d\n", *it);
    } else
        cout << "Not Found\n";
}
int main() {
    int n, m, id, num;
    scanf("%d", &n);
    string ttitle, tauthor, tkey, tpub, tyear;
    for(int i = 0; i < n; i++) {
        scanf("%d\n", &id);
        getline(cin, ttitle);
        title[ttitle].insert(id);
        getline(cin, tauthor);
        author[tauthor].insert(id);
        while(cin >> tkey) {
            key[tkey].insert(id);
            char c = getchar();
            if(c == '\n') break;
        }
        getline(cin, tpub);
        pub[tpub].insert(id);
        getline(cin, tyear);
        year[tyear].insert(id);
    }
    scanf("%d", &m);
    for(int i = 0; i < m; i++) {
        scanf("%d: ", &num);
        string temp;
        getline(cin, temp);
        cout << num << ": " << temp << "\n";
        if(num == 1) query(title, temp);
        else if(num == 2) query(author, temp);
        else if(num == 3) query(key, temp);
        else if(num == 4) query(pub,temp);
        else if(num ==5) query(year, temp);
    }
    return 0;
}
```

1023. Have Fun with Numbers (20) [大整数运算]

Notice that the number 123456789 is a 9-digit number consisting exactly the numbers from 1 to 9, with no duplication. Double it we will obtain 246913578, which happens to be another 9-digit number consisting exactly the numbers from 1 to 9, only in a different permutation. Check to see the result if we double it again!

Now you are suppose to check if there are more numbers with this property. That is, double a given number with k digits, you are to tell if the resulting number consists of only a permutation of the digits in the original number.

Input Specification:

Each input file contains one test case. Each case contains one positive integer with no more than 20 digits.

Output Specification:

For each test case, first print in a line “Yes” if doubling the input number gives a number that consists of only a permutation of the digits in the original number, or “No” if not. Then in the next line, print the doubled number.

Sample Input:

1234567899

Sample Output:

Yes

2469135798

题目大意：给出一个长度不超过20的整数，问这个整数两倍后的数位是否为原数位的一个排列。不管是yes还是no最后都要输出整数乘以2的结果

分析：使用char数组存储这个数，没个数的数位乘以2 + 进位，同时设立book来标记数位出现的情况。只有最后book的每个元素都是0的时候才说明这两个数字是相等的一个排列结果～～

```cpp
#include <cstdio>
#include <string.h>
using namespace std;
int book[10];
int main() {
    char num[22];
    scanf("%s", num);
    int flag = 0, len = strlen(num);
    for(int i = len - 1; i >= 0; i--) {
        int temp = num[i] - '0';
        book[temp]++;
        temp = temp * 2 + flag;
        flag = 0;
        if(temp >= 10) {
            temp = temp - 10;
            flag = 1;
        }
        num[i] = (temp + '0');
        book[temp]--;
    }
    int flag1 = 0;
    for(int i = 0; i < 10; i++) {
        if(book[i] != 0)
            flag1 = 1;
    }
    printf("%s", (flag == 1 || flag1 == 1) ? "No\n" : "Yes\n");
    if(flag == 1) printf("1");
    printf("%s", num);
```

```
29        return 0;
30    }
```

1024. Palindromic Number (25) [大整数相加]

A number that will be the same when it is written forwards or backwards is known as a Palindromic Number. For example, 1234321 is a palindromic number. All single digit numbers are palindromic numbers.

Non-palindromic numbers can be paired with palindromic ones via a series of operations. First, the non-palindromic number is reversed and the result is added to the original number. If the result is not a palindromic number, this is repeated until it gives a palindromic number. For example, if we start from 67, we can obtain a palindromic number in 2 steps: 67 + 76 = 143, and 143 + 341 = 484.

Given any positive integer N, you are supposed to find its paired palindromic number and the number of steps taken to find it.

Input Specification:

Each input file contains one test case. Each case consists of two positive numbers N and K, where N (<= 10^10) is the initial numer and K (<= 100) is the maximum number of steps. The numbers are separated by a space.

Output Specification:

For each test case, output two numbers, one in each line. The first number is the paired palindromic number of N, and the second number is the number of steps taken to find the palindromic number. If the palindromic number is not found after K steps, just output the number obtained at the Kth step and K instead.

Sample Input 1:

67 3

Sample Output 1:

484

2

Sample Input 2:

69 3

Sample Output 2:

1353

3

题目大意：给定一个数字，和允许翻转后相加的次数cnt，求要多少次才能变成一个回文数字，输出那个回文数字和翻转相加了多少次，如果本身就是回文数字就输出0次，如果超过给定的次数cnt了，就输出那个不是回文的结果，并且输出给定的次数cnt

分析：1、会超出long int类型（会有两个点溢出错误），所以用字符串存储，大整数相加

2、可以通过对字符串翻转后比较来判断是否为回文串

```
#include <iostream>
#include <algorithm>
using namespace std;
string s;
void add(string t) {
    int len = s.length(), carry = 0;
    for(int i = 0; i < len; i++) {
        s[i] = s[i] + t[i] + carry - '0';
        carry = 0;
        if(s[i] > '9') {
            s[i] = s[i] - 10;
            carry = 1;
        }
    }
    if(carry) s += '1';
    reverse(s.begin(), s.end());
}
int main() {
    int cnt, i;
    cin >> s >> cnt;
    for(i = 0; i <= cnt; i++) {
        string t = s;
        reverse(t.begin(), t.end());
        if(s == t || i == cnt) break;
        add(t);
    }
    cout << s << endl << i;
    return 0;
}
```

1025. PAT Ranking (25) [排序]

Programming Ability Test (PAT) is organized by the College of Computer Science and Technology of Zhejiang University. Each test is supposed to run simultaneously in several places, and the ranklists will be merged immediately after the test. Now it is your job to write a program to correctly merge all the ranklists and generate the final rank.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive number N (<=100), the number of test locations. Then N ranklists follow, each starts with a line containing a positive integer K (<=300), the number of testees, and then K lines containing the registration number (a 13-digit number) and the total score of each testee. All the numbers in a line are separated by a space.

Output Specification:

For each test case, first print in one line the total number of testees. Then print the final ranklist in the following format:

registration_number final_rank location_number local_rank

The locations are numbered from 1 to N. The output must be sorted in nondecreasing order of the final ranks. The testees with the same score must have the same rank, and the output must be sorted in nondecreasing order of their registration numbers.

Sample Input:

2

5

1234567890001 95

1234567890005 100

1234567890003 95

1234567890002 77

1234567890004 85

4

1234567890013 65

1234567890011 25

1234567890014 100

1234567890012 85

Sample Output:

9

1234567890005 1 1 1

1234567890014 1 2 1

1234567890001 3 1 2

1234567890003 3 1 2

1234567890004 5 1 4

1234567890012 5 2 2

1234567890002 7 1 5

1234567890013 8 2 3

1234567890011 9 2 4

题目大意：有n个考场，每个考场有若干数量的学生，给出每个考场中考生的编号和分数，要求算排名，输出所有考生的编号、排名、考场号、考场内排名

分析：先按照考场内排名 然后赋值给总数组fin，然后总排名，最后输出。注意相同的分数情况下按照学号的从小到大排列，但是他们的排名应该是一样的数字～

```
1   #include <cstdio>
2   #include <algorithm>
```

```cpp
#include <vector>
using namespace std;
struct student {
    long long int no;
    int score, finrank, loca, locarank;
};
bool cmp1(student a, student b) {
    return a.score != b.score ? a.score > b.score : a.no < b.no;
}
int main() {
    int n, m;
    scanf("%d", &n);
    vector<student> fin;
    for(int i = 1; i <= n; i++) {
        scanf("%d", &m);
        vector<student> v(m);
        for(int j = 0; j < m; j++) {
            scanf("%lld %d", &v[j].no, &v[j].score);
            v[j].loca = i;
        }
        sort(v.begin(), v.end(), cmp1);
        v[0].locarank = 1;
        fin.push_back(v[0]);
        for(int j = 1; j < m; j++) {
            v[j].locarank = (v[j].score == v[j - 1].score) ? (v[j - 1].locarank) : (j + 1);
            fin.push_back(v[j]);
        }
    }
    sort(fin.begin(), fin.end(), cmp1);
    fin[0].finrank = 1;
    for(int j = 1; j < fin.size(); j++)
        fin[j].finrank = (fin[j].score == fin[j - 1].score) ? (fin[j - 1].finrank) : (j + 1);
    printf("%d\n", fin.size());
    for(int i = 0; i < fin.size(); i++)
        printf("%013lld %d %d %d\n", fin[i].no, fin[i].finrank, fin[i].loca, fin[i].locarank);
    return 0;
}
```

1026. Table Tennis (30) [模拟，排序]

A table tennis club has N tables available to the public. The tables are numbered from 1 to N. For any pair of players, if there are some tables open when they arrive, they will be assigned to the available table with the smallest number. If all the tables are occupied, they will have to wait in a queue. It is assumed that every pair of players can play for at most 2 hours.

Your job is to count for everyone in queue their waiting time, and for each table the number of players it has served for the day.

One thing that makes this procedure a bit complicated is that the club reserves some tables for their VIP members. When a VIP table is open, the first VIP pair in the queue will have the priviledge to take it. However, if there is no VIP in the queue, the next pair of players can take it. On the other hand, if when it is the turn of a VIP pair, yet no VIP table is available, they can be assigned as any ordinary players.

Input Specification:

Each input file contains one test case. For each case, the first line contains an integer N (<=10000) – the total number of pairs of players. Then N lines follow, each contains 2 times and a VIP tag: HH:MM:SS – the arriving time, P – the playing time in minutes of a pair of players, and tag – which is 1 if they hold a VIP card, or 0 if not. It is guaranteed that the arriving time is between 08:00:00 and 21:00:00 while the club is open. It is assumed that no two customers arrives at the same time. Following the players' info, there are 2 positive integers: K (<=100) – the number of tables, and M (< K) – the number of VIP tables. The last line contains M table numbers.

Output Specification:

For each test case, first print the arriving time, serving time and the waiting time for each pair of players in the format shown by the sample. Then print in a line the number of players served by each table. Notice that the output must be listed in chronological order of the serving time. The waiting time must be rounded up to an integer minute(s). If one cannot get a table before the closing time, their information must NOT be printed.

Sample Input:

9

20:52:00 10 0

08:00:00 20 0

08:02:00 30 0

20:51:00 10 0

08:10:00 5 0

08:12:00 10 1

20:50:00 10 0

08:01:30 15 1

20:53:00 10 1

3 1

2

Sample Output:

08:00:00 08:00:00 0

08:01:30 08:01:30 0

08:02:00 08:02:00 0

08:12:00 08:16:30 5

08:10:00 08:20:00 10

20:50:00 20:50:00 0

20:51:00 20:51:00 0

20:52:00 20:52:00 0

3 3 2

题目大意：k张桌子，球员到达后总是选择编号最小的桌子。如果训练时间超过2h会被压缩成2h，如果到达时候没有球桌空闲就变成队列等待。

k张桌子中m张是vip桌，如果vip桌子有空闲，而且队列里面有vip成员，那么等待队列中的第一个vip球员会到最小的vip球桌训练。如果vip桌子空闲但是没有vip来，那么就分配给普通的人。如果没有vip球桌空闲，那么vip球员就当作普通人处理。

给出每个球员的到达时间、要玩多久、是不是vip（是为1不是为0）。给出球桌数和所有vip球桌的编号，QQ所有在关门前得到训练的球员的到达时间、训练开始时间、等待时长（取整数，四舍五入），营业时间为8点到21点。如果再21点后还没有开始玩的人，就不再玩，不需要输出～

分析：建立两个结构体，person 和 tablenode，分别创建他们的结构体数组player和table。

因为给出的输入是无序的所以要先排序。可以根据最早空闲的桌子是不是vip桌进行分类讨论。

如果最早空闲的桌子index是vip桌，那么寻找队列里面第一个vip球员。根据他的到达时间继续分类讨论。

如果最早空闲的桌子index不是vip桌，那么看队首的球员是不是vip球员。如果是普通人，就直接把球桌分配给他，如果是vip，那么需要找到最早空闲的vip桌子的号vipindex，根据vip球员的到达时间和vipindex桌子的空闲时间继续分类讨论。

分配的时候标记table的num++，统计该table服务的人数。

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
struct person {
    int arrive, start, time;
    bool vip;
}tempperson;
struct tablenode {
    int end = 8 * 3600, num;
    bool vip;
};
bool cmp1(person a, person b) {
    return a.arrive < b.arrive;
}
bool cmp2(person a, person b) {
    return a.start < b.start;
}
```

```
vector<person> player;
vector<tablenode> table;
void alloctable(int personid, int tableid) {
    if(player[personid].arrive <= table[tableid].end)
        player[personid].start = table[tableid].end;
    else
        player[personid].start = player[personid].arrive;
    table[tableid].end = player[personid].start + player[personid].time;
    table[tableid].num++;
}
int findnextvip(int vipid) {
    vipid++;
    while(vipid < player.size() && player[vipid].vip == false) vipid++;
    return vipid;
}
int main() {
    int n, k, m, viptable;
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        int h, m, s, temptime, flag;
        scanf("%d:%d:%d %d %d", &h, &m, &s, &temptime, &flag);
        tempperson.arrive = h * 3600 + m * 60 + s;
        tempperson.start = 21 * 3600;
        if(tempperson.arrive >= 21 * 3600) continue;
        tempperson.time = temptime <= 120 ? temptime * 60 : 7200;
        tempperson.vip = ((flag == 1) ? true : false);
        player.push_back(tempperson);
    }
    scanf("%d%d", &k, &m);
    table.resize(k + 1);
    for(int i = 0; i < m; i++) {
        scanf("%d", &viptable);
        table[viptable].vip = true;
    }
    sort(player.begin(), player.end(), cmp1);
    int i = 0, vipid = -1;
    vipid = findnextvip(vipid);
    while(i < player.size()) {
        int index = -1, minendtime = 999999999;
        for(int j = 1; j <= k; j++) {
            if(table[j].end < minendtime) {
                minendtime = table[j].end;
                index = j;
            }
        }
        if(table[index].end >= 21 * 3600) break;
        if(player[i].vip == true && i < vipid) {
            i++;
            continue;
        }
        if(table[index].vip == true) {
            if(player[i].vip == true) {
```

```
72                  alloctable(i, index);
73                  if(vipid == i) vipid = findnextvip(vipid);
74                  i++;
75              } else {
76                  if(vipid < player.size() && player[vipid].arrive <=
    table[index].end) {
77                      alloctable(vipid, index);
78                      vipid = findnextvip(vipid);
79                  } else {
80                      alloctable(i, index);
81                      i++;
82                  }
83              }
84          } else {
85              if(player[i].vip == false) {
86                  alloctable(i, index);
87                  i++;
88              } else {
89                  int vipindex = -1, minvipendtime = 999999999;
90                  for(int j = 1; j <= k; j++) {
91                      if(table[j].vip == true && table[j].end < minvipendtime) {
92                          minvipendtime = table[j].end;
93                          vipindex = j;
94                      }
95                  }
96                  if(vipindex != -1 && player[i].arrive >= table[vipindex].end)
    {
97                      alloctable(i, vipindex);
98                      if(vipid == i) vipid = findnextvip(vipid);
99                      i++;
100                 } else {
101                     alloctable(i, index);
102                     if(vipid == i) vipid = findnextvip(vipid);
103                     i++;
104                 }
105             }
106         }
107     }
108     sort(player.begin(), player.end(), cmp2);
109     for(i = 0; i < player.size() && player[i].start < 21 * 3600; i++) {
110         printf("%02d:%02d:%02d ", player[i].arrive / 3600, player[i].arrive %
    3600 / 60, player[i].arrive % 60);
111         printf("%02d:%02d:%02d ", player[i].start / 3600, player[i].start %
    3600 / 60, player[i].start % 60);
112         printf("%.0f\n", round((player[i].start - player[i].arrive) / 60.0));
113     }
114     for(int i = 1; i <= k; i++) {
115         if(i != 1) printf(" ");
116         printf("%d", table[i].num);
117     }
118     return 0;
119 }
```

1027. Colors in Mars (20) [进制转换]

People in Mars represent the colors in their computers in a similar way as the Earth people. That is, a color is represented by a 6-digit number, where the first 2 digits are for Red, the middle 2 digits for Green, and the last 2 digits for Blue. The only difference is that they use radix 13 (0-9 and A-C) instead of 16. Now given a color in three decimal numbers (each between 0 and 168), you are supposed to output their Mars RGB values.

Input

Each input file contains one test case which occupies a line containing the three decimal color values.

Output

For each test case you should output the Mars RGB value in the following format: first output “#, then followed by a 6-digit number where all the English characters must be upper-cased. If a single color is only 1-digit long, you must print a “0” to the left.

Sample Input

15 43 71

Sample Output

#123456

题目大意：给三个十进制的数，把它们转换为十三进制的数输出。要求在前面加上一个”#”号

分析：因为0~168的十进制转换为13进制不会超过两位数，所以这个两位数为(num / 13)(num % 13)构成的数字

```
#include <cstdio>
using namespace std;
int main() {
    char c[14] = {"0123456789ABC"};
    printf("#");
    for(int i = 0; i < 3; i++) {
        int num;
        scanf("%d", &num);
        printf("%c%c", c[num/13], c[num%13]);
    }
    return 0;
}
```

1028. List Sorting (25) [排序]

Excel can sort records according to any column. Now you are supposed to imitate this function.

Input

Each input file contains one test case. For each case, the first line contains two integers N (<=100000) and C, where N is the number of records and C is the column that you are supposed to sort the records with. Then N lines follow, each contains a record of a student. A student’s record consists of his or her distinct ID (a 6-digit number), name (a string with no more than 8 characters without space), and grade (an integer between 0 and 100, inclusive).

Output

For each test case, output the sorting result in N lines. That is, if C = 1 then the records must be sorted in increasing order according to ID's; if C = 2 then the records must be sorted in non-decreasing order according to names; and if C = 3 then the records must be sorted in non-decreasing order according to grades. If there are several students who have the same name or grade, they must be sorted according to their ID's in increasing order.

Sample Input 1

3 1

000007 James 85

000010 Amy 90

000001 Zoe 60

Sample Output 1

000001 Zoe 60

000007 James 85

000010 Amy 90

Sample Input 2

4 2

000007 James 85

000010 Amy 90

000001 Zoe 60

000002 James 98

Sample Output 2

000010 Amy 90

000002 James 98

000007 James 85

000001 Zoe 60

Sample Input 3

4 3

000007 James 85

000010 Amy 90

000001 Zoe 60

000002 James 90

Sample Output 3

000001 Zoe 60

000007 James 85

000002 James 90

000010 Amy 90

题目大意：根据c的值是1还是2还是3，对相应的列排序。第一列升序，第二列不降序，第三列不降序。

分析：注意，按照名称的不降序排序，因为strcmp比较的是ACSII码，所以A < Z。写cmp函数的时候return strcmp(a.name, b.name) <= 0; return语句返回的是true或者false的值，所以要写 <= 0 这样的形式。比较ACSII码的大小，strcmp('a', 'z')返回负值，因为a<z a – z < 0

按照分数的不降序排序，a.score <= b.score

```
#include <iostream>
#include <algorithm>
#include <string.h>
using namespace std;
const int maxn = 100001;
struct NODE {
    int no, score;
    char name[10];
}node[maxn];
int c;
int cmp1(NODE a, NODE b) {
    if(c == 1) {
        return a.no < b.no;
    } else if(c == 2) {
        if(strcmp(a.name, b.name) == 0) return a.no < b.no;
        return strcmp(a.name, b.name) <= 0;
    } else {
        if(a.score == b.score) return a.no < b.no;
        return a.score <= b.score;
    }
}
int main() {
    int n;
    scanf("%d%d", &n, &c);
    for(int i = 0; i < n; i++)
        scanf("%d %s %d", &node[i].no, node[i].name, &node[i].score);
    sort(node, node + n, cmp1);
    for(int i = 0; i < n; i++)
        printf("%06d %s %d\n", node[i].no, node[i].name, node[i].score);
    return 0;
}
```

1029. Median (25) [two pointers]

Given an increasing sequence S of N integers, the median is the number at the middle position. For example, the median of S1={11, 12, 13, 14} is 12, and the median of S2={9, 10, 15, 16, 17} is 15. The median of two sequences is defined to be the median of the nondecreasing sequence which contains all the elements of both sequences. For example, the median of S1 and S2 is 13.

Given two increasing sequences of integers, you are asked to find their median.

Input

Each input file contains one test case. Each case occupies 2 lines, each gives the information of a sequence. For each sequence, the first positive integer N (<=1000000) is the size of that sequence. Then N integers follow, separated by a space. It is guaranteed that all the integers are in the range of long int.

Output

For each test case you should output the median of the two given sequences in a line.

Sample Input

4 11 12 13 14

5 9 10 15 16 17

Sample Output

13

题目大意：给出两个已排序序列，求这两个序列合并后的中间数

分析：开一个数组，在线处理第二个数组。 第一二个数组（下标从1开始）分别有n，m个元素，中间数在(n + m + 1) / 2的位置。所以只要从小到大数到(n + m + 1) / 2的位置就行了~ count计总个数 ， 给第一个数组设置指针i，每次从第二个数组中读入temp，检查第一个数组中前几个数是不是比temp小，小就count+1并判断是否到数了中间数，到了就输出。 如果数完比temp小的数还没到中间数，count+1，检查temp是不是中间数，是就输出。循环上述过程。如果第二个数组读取完了，还没数到中间数，说明中间数在剩下的第一个数组中，就在剩下的数组中数到中间数位置即可

PS：感谢LittleMeepo提供的更优解～

```
#include <iostream>
using namespace std;
int k[200005];
int main(){
    int n, m, temp, count = 0;
    cin >> n;
    for (int i = 1; i <= n; i++)
        scanf("%d", &k[i]);
    k[n + 1] = 0x7fffffff;
    cin >> m;
    int midpos = (n + m + 1) / 2, i = 1;
    for (int j = 1; j <= m; j++) {
        scanf("%d", &temp);
        while (k[i] < temp) {
            count++;
            if (count == midpos) cout << k[i];
            i++;
```

```
            }
            count++;
            if (count == midpos) cout << temp;
        }
        while (i <= n) {
            count++;
            if (count == midpos) cout << k[i];
            i++;
        }
        return 0;
    }
```

1030. Travel Plan (30) [Dijkstra算法 + DFS，最短路径，边权]

A traveler's map gives the distances between cities along the highways, together with the cost of each highway. Now you are supposed to write a program to help a traveler to decide the shortest path between his/her starting city and the destination. If such a shortest path is not unique, you are supposed to output the one with the minimum cost, which is guaranteed to be unique.

Input Specification:

Each input file contains one test case. Each case starts with a line containing 4 positive integers N, M, S, and D, where N (<=500) is the number of cities (and hence the cities are numbered from 0 to N-1); M is the number of highways; S and D are the starting and the destination cities, respectively. Then M lines follow, each provides the information of a highway, in the format:

City1 City2 Distance Cost

where the numbers are all integers no more than 500, and are separated by a space.

Output Specification:

For each test case, print in one line the cities along the shortest path from the starting point to the destination, followed by the total distance and the total cost of the path. The numbers must be separated by a space and there must be no extra space at the end of output.

Sample Input

4 5 0 3

0 1 1 20

1 3 2 30

0 3 4 10

0 2 2 20

2 3 1 20

Sample Output

0 2 3 3 40

题目大意：求起点到终点的最短路径最短距离和花费，要求首先路径最短，其次花费最少，要输出完整路径

分析：Dijksta + DFS。 Dijkstra记录路径pre数组，然后用dfs求最短的一条mincost以及它的路径path，最后输出path数组和mincost

注意路径path因为是从末端一直压入push_back到path里面的，所以要输出路径的时候倒着输出

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
int n, m, s, d;
int e[510][510], dis[510], cost[510][510];
vector<int> pre[510];
bool visit[510];
const int inf = 99999999;
vector<int> path, temppath;
int mincost = inf;
void dfs(int v) {
    temppath.push_back(v);
    if(v == s) {
        int tempcost = 0;
        for(int i = temppath.size() - 1; i > 0; i--) {
            int id = temppath[i], nextid = temppath[i-1];
            tempcost += cost[id][nextid];
        }
        if(tempcost < mincost) {
            mincost = tempcost;
            path = temppath;
        }
        temppath.pop_back();
        return ;
    }
    for(int i = 0; i < pre[v].size(); i++)
        dfs(pre[v][i]);
    temppath.pop_back();
}
int main() {
    fill(e[0], e[0] + 510 * 510, inf);
    fill(dis, dis + 510, inf);
    scanf("%d%d%d%d", &n, &m, &s, &d);
    for(int i = 0; i < m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        scanf("%d", &e[a][b]);
        e[b][a] = e[a][b];
        scanf("%d", &cost[a][b]);
        cost[b][a] = cost[a][b];
    }
    pre[s].push_back(s);
    dis[s] = 0;
    for(int i = 0; i < n; i++) {
        int u = -1, minn = inf;
        for(int j = 0; j < n; j++) {
```

```
            if(visit[j] == false && dis[j] < minn) {
                u = j;
                minn = dis[j];
            }
        }
        if(u == -1) break;
        visit[u] = true;
        for(int v = 0; v < n; v++) {
            if(visit[v] == false && e[u][v] != inf) {
                if(dis[v] > dis[u] + e[u][v]) {
                    dis[v] = dis[u] + e[u][v];
                    pre[v].clear();
                    pre[v].push_back(u);
                } else if(dis[v] == dis[u] + e[u][v]) {
                    pre[v].push_back(u);
                }
            }
        }
    }
    dfs(d);
    for(int i = path.size() - 1; i >= 0; i--)
        printf("%d ", path[i]);
    printf("%d %d", dis[d], mincost);
    return 0;
}
```

1031. Hello World for U (20) [图形打印]

Given any string of N (>=5) characters, you are asked to form the characters into the shape of U. For example, “helloworld” can be printed as:

h d

e l

l r

lowo

That is, the characters must be printed in the original order, starting top-down from the left vertical line with n1 characters, then left to right along the bottom line with n2 characters, and finally bottom-up along the vertical line with n3 characters. And more, we would like U to be as squared as possible — that is, it must be satisfied that n1 = n3 = max { k| k <= n2 for all 3 <= n2 <= N } with n1 + n2 + n3 − 2 = N.

Input Specification:

Each input file contains one test case. Each case contains one string with no less than 5 and no more than 80 characters in a line. The string contains no white space.

Output Specification:

For each test case, print the input string in the shape of U as specified in the description.

Sample Input:

helloworld!

Sample Output:

h !

e d

l l

lowor

题目大意：用所给字符串按U型输出。n1和n3是左右两条竖线从上到下的字符个数，n2是底部横线从左到右的字符个数。

要求：

1. n1 == n3
2. n2 >= n1
3. n1为在满足上述条件的情况下的最大值

分析：假设n = 字符串长度 + 2，因为2 * n1 + n2 = n，且要保证n2 >= n1, n1尽可能地大，分类讨论：

1. 如果n % 3 == 0，n正好被3整除，直接n1 == n2 == n3;
2. 如果n % 3 == 1，因为n2要比n1大，所以把多出来的那1个给n2
3. 如果n % 3 == 2, 就把多出来的那2个给n2

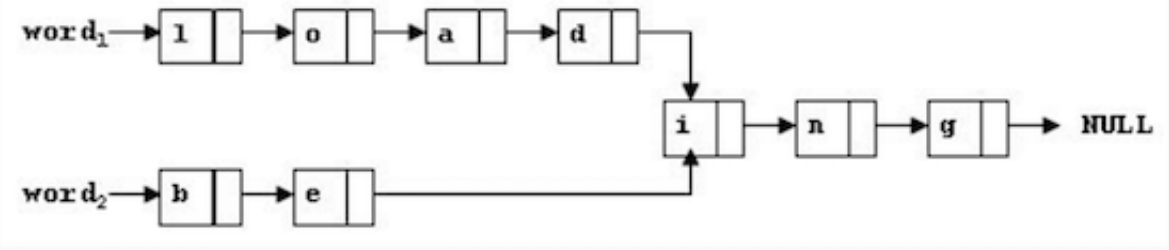
所以得到公式：n1 = n / 3，n2 = n / 3 + n % 3

把它们存储到二维字符数组中，一开始初始化字符数组为空格，然后按照u型填充进去，最后输出这个数组u～～

```
#include <iostream>
#include <string.h>
using namespace std;
int main() {
    char c[81], u[30][30];
    memset(u, ' ', sizeof(u));
    scanf("%s", c);
    int n = strlen(c) + 2;
    int n1 = n / 3, n2 = n / 3 + n % 3, index = 0;
    for(int i = 0; i < n1; i++) u[i][0] = c[index++];
    for(int i = 1; i <= n2 - 2; i++) u[n1-1][i] = c[index++];
    for(int i = n1 - 1; i >= 0; i--) u[i][n2-1] = c[index++];
    for(int i = 0; i < n1; i++) {
        for(int j = 0; j < n2; j++)
            printf("%c", u[i][j]);
        printf("\n");
    }
    return 0;
}
```

1032. Sharing (25) [链表]

To store English words, one method is to use linked lists and store a word letter by letter. To save some space, we may let the words share the same sublist if they share the same suffix. For example, “loading” and “being” are stored as showed in Figure 1.



Input Specification:

Each input file contains one test case. For each case, the first line contains two addresses of nodes and a positive N (<= 105), where the two addresses are the addresses of the first nodes of the two words, and N is the total number of nodes. The address of a node is a 5-digit positive integer, and NULL is represented by -1.

Then N lines follow, each describes a node in the format:

Address Data Next

where Address is the position of the node, Data is the letter contained by this node which is an English letter chosen from {a-z, A-Z}, and Next is the position of the next node.

Output Specification:

For each case, simply output the 5-digit starting position of the common suffix. If the two words have no common suffix, output “-1” instead.

Sample Input 1:

11111 22222 9

67890 i 00002

00010 a 12345

00003 g -1

12345 D 67890

00002 n 00003

22222 B 23456

11111 L 00001

23456 e 67890

00001 o 00010

Sample Output 1:

67890

Sample Input 2:

00001 00002 4

00001 a 10001

10001 s -1

00002 a 10002

10002 t -1

Sample Output 2:

-1

题目大意：求两个链表的首个共同结点的地址。如果没有，就输出-1

分析：用结构体数组存储，node[i]表示地址为i的结点，key表示值，next为下一个结点的地址，flag表示第一条链表有没有该结点

遍历第一条链表，将访问过的结点的flag都标记为true，当遍历第二条结点的时候，如果遇到了true的结点就输出并结束程序，没有遇到就输出-1

```
#include <cstdio>
using namespace std;
struct NODE {
    char key;
    int next;
    bool flag;
}node[100000];
int main() {
    int s1, s2, n, a, b;
    scanf("%d%d%d", &s1, &s2, &n);
    char data;
    for(int i = 0; i < n; i++) {
        scanf("%d %c %d", &a, &data, &b);
        node[a] = {data, b, false};
    }
    for(int i = s1; i != -1; i = node[i].next)
        node[i].flag = true;
    for(int i = s2; i != -1; i = node[i].next) {
        if(node[i].flag == true) {
            printf("%05d", i);
            return 0;
        }
    }
    printf("-1");
    return 0;
}
```

1033. To Fill or Not to Fill (25) [贪心算法]

With highways available, driving a car from Hangzhou to any other city is easy. But since the tank capacity of a car is limited, we have to find gas stations on the way from time to time. Different gas station may give different price. You are asked to carefully design the cheapest route to go.

Input Specification:

Each input file contains one test case. For each case, the first line contains 4 positive numbers: Cmax (<= 100), the maximum capacity of the tank; D (<=30000), the distance between Hangzhou and the destination city; Davg (<=20), the average distance per unit gas that the car can run; and N (<= 500), the total number of gas stations. Then N lines follow, each contains a pair of non-negative numbers: Pi, the unit gas price, and Di (<=D), the distance between this station and Hangzhou, for i=1,…N. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the cheapest price in a line, accurate up to 2 decimal places. It is assumed that the tank is empty at the beginning. If it is impossible to reach the destination, print “The maximum travel distance = X” where X is the maximum possible distance the car can run, accurate up to 2 decimal places.

Sample Input 1:

50 1300 12 8

6.00 1250

7.00 600

7.00 150

7.10 0

7.20 200

7.50 400

7.30 1000

6.85 300

Sample Output 1:

749.17

Sample Input 2:

50 1300 12 2

7.10 0

7.00 600

Sample Output 2:

The maximum travel distance = 1200.00

题目大意：汽车从杭州出发可以通过高速公路去任何城市，但是油箱的容量是有限的，路上有很多加油站，每个加油站的价格不同，为汽车设计一个从杭州到终点的最便宜的路线，Cmax表示油箱最大容量，D表示杭州到目的地的距离，Davg表示平均每单位的汽油可以让汽车行驶的距离，N表示汽车的站点数量，每个站点都会给出它的单位油价Pi和汽车站点和杭州的距离Di，求汽车从杭州到终点的最小花费，如果不能够到达，就输出汽车能够行驶的最大距离

分析：贪心算法。

0.假设增加一个目的地处的加油站，距离为目的地的距离，价格为0，考虑从0距离开始能否到达最后一个加油站的问题

1.因为先开始没有油，所以如果所有的加油站距离都没有等于0的，那么说明车哪也去不了，直接输出并return

2.将加油站按照距离dis从小到大排序

3.先去第一个加油站，设置变量nowdis表示当前所在的距离，maxdis是能够到达的最大距离，nowprice是当前的站点的价格，totalPrice是总的价格。

贪心思想：

0.寻找比自己距离远的，到能够到达的最大距离之间的加油站，看他们的油价。如果找到了更低价格的油价，就加油到刚好能到达那个加油站的距离的油，然后去那个更低价格的加油站（有更低的我一分都不想多花在别的距离上，只加到刚好满足更低价格的加油站的距离就行，那样以后的路程我就可以以更低的价格行驶啦）

1.如果找不到更低的，就找尽可能低的油价的加油站，在当前加油站加满油之后过去。因为想要让路程上使用的尽可能是低价的油，既然没有比当前更低价格的了，就让油箱加到最大值，这样能保证利益最大化，保证最大的距离使用的是便宜的油。

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
const int inf = 99999999;
struct station {
    double price, dis;
};
bool cmp1(station a, station b) {
    return a.dis < b.dis;
}
int main() {
    double cmax, d, davg;
    int n;
    scanf("%lf%lf%lf%d", &cmax, &d, &davg, &n);
    vector<station> sta(n + 1);
    sta[0] = {0.0, d};
    for(int i = 1; i <= n; i++)
        scanf("%lf%lf", &sta[i].price, &sta[i].dis);
    sort(sta.begin(), sta.end(), cmp1);
    double nowdis = 0.0, maxdis = 0.0, nowprice = 0.0, totalPrice = 0.0,
leftdis = 0.0;
    if(sta[0].dis != 0) {
        printf("The maximum travel distance = 0.00");
        return 0;
    } else {
        nowprice = sta[0].price;
    }
    while(nowdis < d) {
        maxdis = nowdis + cmax * davg;
        double minPriceDis = 0, minPrice = inf;
```

```
31          int flag = 0;
32          for(int i = 1; i <= n && sta[i].dis <= maxdis; i++) {
33              if(sta[i].dis <= nowdis) continue;
34              if(sta[i].price < nowprice) {
35                  totalPrice += (sta[i].dis - nowdis - leftdis) * nowprice /
    davg;
36                  leftdis = 0.0;
37                  nowprice = sta[i].price;
38                  nowdis = sta[i].dis;
39                  flag = 1;
40                  break;
41              }
42              if(sta[i].price < minPrice) {
43                  minPrice = sta[i].price;
44                  minPriceDis = sta[i].dis;
45              }
46          }
47          if(flag == 0 && minPrice != inf) {
48              totalPrice += (nowprice * (cmax - leftdis / davg));
49              leftdis = cmax * davg - (minPriceDis - nowdis);
50              nowprice = minPrice;
51              nowdis = minPriceDis;
52          }
53          if(flag == 0 && minPrice == inf) {
54              nowdis += cmax * davg;
55              printf("The maximum travel distance = %.2f", nowdis);
56              return 0;
57          }
58      }
59      printf("%.2f", totalPrice);
60      return 0;
61  }
```

1034. Head of a Gang (30) [图的遍历，DFS]

One way that the police finds the head of a gang is to check people's phone calls. If there is a phone call between A and B, we say that A and B is related. The weight of a relation is defined to be the total time length of all the phone calls made between the two persons. A "Gang" is a cluster of more than 2 persons who are related to each other with total relation weight being greater than a given threshold K. In each gang, the one with maximum total weight is the head. Now given a list of phone calls, you are supposed to find the gangs and the heads.

Input Specification:

Each input file contains one test case. For each case, the first line contains two positive numbers N and K (both less than or equal to 1000), the number of phone calls and the weight threthold, respectively. Then N lines follow, each in the following format:

Name1 Name2 Time

where Name1 and Name2 are the names of people at the two ends of the call, and Time is the length of the call. A name is a string of three capital letters chosen from A-Z. A time length is a positive integer which is no more than 1000 minutes.

Output Specification:

For each test case, first print in a line the total number of gangs. Then for each gang, print in a line the name of the head and the total number of the members. It is guaranteed that the head is unique for each gang. The output must be sorted according to the alphabetical order of the names of the heads.

Sample Input 1:

8 59

AAA BBB 10

BBB AAA 20

AAA CCC 40

DDD EEE 5

EEE DDD 70

FFF GGG 30

GGG HHH 20

HHH FFF 10

Sample Output 1:

2

AAA 3

GGG 3

Sample Input 2:

8 70

AAA BBB 10

BBB AAA 20

AAA CCC 40

DDD EEE 5

EEE DDD 70

FFF GGG 30

GGG HHH 20

HHH FFF 10

Sample Output 2:

0

题目大意：给出1000条以内的通话记录A B和权值w，和阈值k，如果一个团伙人数超过2人并且通话总权值超过k，令团伙里面的自身权值的最大值为头目，输出所有满足条件的团伙的头目，和他们团伙里面的人数

分析：总的来说是一个判断一个图的连通分量的个数，用图的遍历解决，深度优先遍历。

1.因为给的是字母，要用两个map把它们转换成数字，从1开始排列命名所有不同的人的id，存储在两个map里面，一个字符串对应id，一个id对应字符串，方便查找，正好顺便统计了总共的人数idNumber。

2.建立两个数组，weight和G，分别存储每条边的权值和每个结点的权值，因为这两个题目中都要求得后判断。

3.用传递引用的方法深度优先dfs，这样传入的参数在dfs后还能保存想要求得的值

4.遍历过一条边之后就把这条边的权值设为0（ G[u][v] = G[v][u] = 0;）防止出现回路遍历死循环

```
#include <iostream>
#include <map>
using namespace std;
map<string, int> stringToInt;
map<int, string> intToString;
map<string, int> ans;
int idNumber = 1, k;
int stoifunc(string s) {
    if(stringToInt[s] == 0) {
        stringToInt[s] = idNumber;
        intToString[idNumber] = s;
        return idNumber++;
    } else {
        return stringToInt[s];
    }
}
int G[2010][2010], weight[2010];
bool vis[2010];
void dfs(int u, int &head, int &numMember, int &totalweight) {
    vis[u] = true;
    numMember++;
    if(weight[u] > weight[head])
        head = u;
    for(int v = 1; v < idNumber; v++) {
        if(G[u][v] > 0) {
            totalweight += G[u][v];
            G[u][v] = G[v][u] = 0;
            if(vis[v] == false)
                dfs(v, head, numMember, totalweight);
        }
    }
}
void dfsTrave() {
    for(int i = 1; i < idNumber; i++) {
```

```
35          if(vis[i] == false) {
36              int head = i, numMember = 0, totalweight = 0;
37              dfs(i, head, numMember, totalweight);
38              if(numMember > 2 && totalweight > k)
39                  ans[intToString[head]] = numMember;
40          }
41      }
42  }
43  int main() {
44      int n, w;
45      cin >> n >> k;
46      string s1, s2;
47      for(int i = 0; i < n; i++) {
48          cin >> s1 >> s2 >> w;
49          int id1 = stoifunc(s1);
50          int id2 = stoifunc(s2);
51          weight[id1] += w;
52          weight[id2] += w;
53          G[id1][id2] += w;
54          G[id2][id1] += w;
55      }
56      dfsTrave();
57      cout << ans.size() << endl;
58      for(auto it = ans.begin(); it != ans.end(); it++)
59          cout << it->first << " " << it->second << endl;
60      return 0;
61  }
```

1035. Password (20) [字符串处理]

To prepare for PAT, the judge sometimes has to generate random passwords for the users. The problem is that there are always some confusing passwords since it is hard to distinguish 1 (one) from l (L in lowercase), or 0 (zero) from O (o in uppercase). One solution is to replace 1 (one) by @, 0 (zero) by %, l by L, and O by o. Now it is your job to write a program to check the accounts generated by the judge, and to help the juge modify the confusing passwords.

Input Specification:

Each input file contains one test case. Each case contains a positive integer N (<= 1000), followed by N lines of accounts. Each account consists of a user name and a password, both are strings of no more than 10 characters with no space.

Output Specification:

For each test case, first print the number M of accounts that have been modified, then print in the following M lines the modified accounts info, that is, the user names and the corresponding modified passwords. The accounts must be printed in the same order as they are read in. If no account is modified, print in one line “There are N accounts and no account is modified” where N is the total number of accounts. However, if N is one, you must print “There is 1 account and no account is modified” instead.

Sample Input 1:

3

Team000002 Rlsp0dfa

Team000003 perfectpwd

Team000001 R1spOdfa

Sample Output 1:

2

Team000002 RLsp%dfa

Team000001 R@spodfa

Sample Input 2:

1

team110 abcdefg332

Sample Output 2:

There is 1 account and no account is modified

Sample Input 3:

2

team110 abcdefg222

team220 abcdefg333

Sample Output 3:

There are 2 accounts and no account is modified

题目大意：给定n个用户的姓名和密码，把密码中的1改为@，0改为%，l改为L，O改为o。

如果不存在需要修改的密码，则输出There are n accounts and no account is modified。注意单复数，如果只有一个账户，就输出There is 1 account and no account is modified

分析：把需要改变的字符串改变后存储在字符串数组vector里面，根据数组里面元素的个数是否为0输出相应的结果。

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n;
    scanf("%d", &n);
    vector<string> v;
    for(int i = 0; i < n; i++) {
        string name, s;
        cin >> name >> s;
        int len = s.length(), flag = 0;
        for(int j = 0; j < len; j++) {
            switch(s[j]) {
                case '1' : s[j] = '@'; flag = 1; break;
```

```
                case '0' : s[j] = '%'; flag = 1; break;
                case 'l' : s[j] = 'L'; flag = 1; break;
                case 'O' : s[j] = 'o'; flag = 1; break;
            }
        }
        if(flag) {
            string temp = name + " " + s;
            v.push_back(temp);
        }
    }
    int cnt = v.size();
    if(cnt != 0) {
        printf("%d\n", cnt);
        for(int i = 0; i < cnt; i++)
            cout << v[i] << endl;
    } else if(n == 1) {
        printf("There is 1 account and no account is modified");
    } else {
        printf("There are %d accounts and no account is modified", n);
    }
    return 0;
}
```

1036. Boys vs Girls (25) [查找元素]

This time you are asked to tell the difference between the lowest grade of all the male students and the highest grade of all the female students.

Input Specification:

Each input file contains one test case. Each case contains a positive integer N, followed by N lines of student information. Each line contains a student's name, gender, ID and grade, separated by a space, where name and ID are strings of no more than 10 characters with no space, gender is either F (female) or M (male), and grade is an integer between 0 and 100. It is guaranteed that all the grades are distinct.

Output Specification:

For each test case, output in 3 lines. The first line gives the name and ID of the female student with the highest grade, and the second line gives that of the male student with the lowest grade. The third line gives the difference gradeF-gradeM. If one such kind of student is missing, output "Absent" in the corresponding line, and output "NA" in the third line instead.

Sample Input 1:

3

Joe M Math990112 89

Mike M CS991301 100

Mary F EE990830 95

Sample Output 1:

Mary EE990830

Joe Math990112

6

Sample Input 2:

1

Jean M AA980920 60

Sample Output 2:

Absent

Jean AA980920

NA

题目大意：给出N个同学的信息，输出女生中的最高分获得者的信息与男生中最低分获得者的信息，并输出他们的分数差。如果不存在女生或者男生，则对应获得者信息处输出Absent，而且差值处输出NA。

分析：用string类型的female和male保存要求的学生的信息，femalescore和malescore处保存男生的最低分和女生的最高分

一开始设femalescore为最低值-1，malescore为最高值101，最后根据分值是否为-1或者101来判断是否有相应的女生或者男生。

```
#include <iostream>
using namespace std;
int main() {
    int n;
    scanf("%d", &n);
    string female, male;
    int femalescore = -1, malescore = 101;
    for(int i = 0; i < n; i++) {
        string name, sex, num;
        int score;
        cin >> name >> sex >> num;
        scanf("%d", &score);
        if(sex == "F") {
            if(femalescore < score) {
                femalescore = score;
                female = name + " " + num;
            }
        } else if(malescore > score) {
                malescore = score;
                male = name + " " + num;
            }
    }
    if(femalescore != -1)
        cout << female << endl;
    else
        printf("Absent\n");
    if(malescore != 101)
```

```
28            cout << male << endl;
29        else
30            printf("Absent\n");
31        if(femalescore != -1 && malescore != 101)
32            printf("%d", femalescore - malescore);
33        else
34            printf("NA");
35        return 0;
36    }
```

1037. Magic Coupon (25) [贪心算法]

The magic shop in Mars is offering some magic coupons. Each coupon has an integer N printed on it, meaning that when you use this coupon with a product, you may get N times the value of that product back! What is more, the shop also offers some bonus product for free. However, if you apply a coupon with a positive N to this bonus product, you will have to pay the shop N times the value of the bonus product… but hey, magically, they have some coupons with negative N's!

For example, given a set of coupons {1 2 4 -1}, and a set of product values {7 6 -2 -3} (in Mars dollars M$) where a negative value corresponds to a bonus product. You can apply coupon 3 (with N being 4) to product 1 (with value M$7) to get M$28 back; coupon 2 to product 2 to get M$12 back; and coupon 4 to product 4 to get M$3 back. On the other hand, if you apply coupon 3 to product 4, you will have to pay M$12 to the shop.

Each coupon and each product may be selected at most once. Your task is to get as much money back as possible.

Input Specification:

Each input file contains one test case. For each case, the first line contains the number of coupons NC, followed by a line with NC coupon integers. Then the next line contains the number of products NP, followed by a line with NP product values. Here 1<= NC, NP <= 105, and it is guaranteed that all the numbers will not exceed 230.

Output Specification:

For each test case, simply print in a line the maximum amount of money you can get back.

Sample Input:

4

1 2 4 -1

4

7 6 -2 -3

Sample Output:

43

题目大意：给出两个数字序列，从这两个序列中分别选取相同数量的元素进行一对一相乘，问能得到的乘积之和最大为多少～

分析：把这两个序列都从小到大排序，将前面都是负数的数相乘求和，然后将后面都是正数的数相乘求和～

```cpp
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
    int m, n, ans = 0, p = 0, q = 0;
    scanf("%d", &m);
    vector<int> v1(m);
    for(int i = 0; i < m; i++)
        scanf("%d", &v1[i]);
    scanf("%d", &n);
    vector<int> v2(n);
    for(int i = 0; i < n; i++)
        scanf("%d", &v2[i]);
    sort(v1.begin(), v1.end());
    sort(v2.begin(), v2.end());
    while(p < m && q < n && v1[p] < 0 && v2[q] < 0) {
        ans += v1[p] * v2[q];
        p++; q++;
    }
    p = m - 1, q = n - 1;
    while(p >= 0 && q >= 0 && v1[p] > 0 && v2[q] > 0) {
        ans += v1[p] * v2[q];
        p--; q--;
    }
    printf("%d", ans);
    return 0;
}
```

1038. Recover the Smallest Number (30) [贪心算法]

Given a collection of number segments, you are supposed to recover the smallest number from them. For example, given {32, 321, 3214, 0229, 87}, we can recover many numbers such like 32-321-3214-0229-87 or 0229-32-87-321-3214 with respect to different orders of combinations of these segments, and the smallest number is 0229-321-3214-32-87.

Input Specification:

Each input file contains one test case. Each case gives a positive integer N (<=10000) followed by N number segments. Each segment contains a non-negative integer of no more than 8 digits. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the smallest number in one line. Do not output leading zeros.

Sample Input:

5 32 321 3214 0229 87

Sample Output:

22932132143287

题目大意：给一些字符串，求它们拼接起来构成最小数字的方式

分析：贪心算法。让我们一起来见证cmp函数的强大之处！！～～不是按照字典序排列就可以的，必须保证两个字符串构成的数字是最小的才行，所以cmp函数写成return a + b < b + a;的形式，保证它排列按照能够组成的最小数字的形式排列。

因为字符串可能前面有0，这些要移除掉（用s.erase(s.begin())就可以了～嗯～string如此神奇～～）。输出拼接后的字符串即可。

注意：如果移出了0之后发现s.length() == 0了，说明这个数是0，那么要特别地输出这个0，否则会什么都不输出～

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
bool cmp0(string a, string b) {
    return a + b < b + a;
}
string str[10010];
int main() {
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        cin >> str[i];
    sort(str, str + n, cmp0);
    string s;
    for(int i = 0; i < n; i++)
        s += str[i];
    while(s.length() != 0 && s[0] == '0')
        s.erase(s.begin());
    if(s.length() == 0) cout << 0;
    cout << s;
    return 0;
}
```

1039. Course List for Student (25) [不定长数组vector，STL的使用]

Zhejiang University has 40000 students and provides 2500 courses. Now given the student name lists of all the courses, you are supposed to output the registered course list for each student who comes for a query.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers: N (<=40000), the number of students who look for their course lists, and K (<=2500), the total number of courses. Then the student name lists are given for the courses (numbered from 1 to K) in the following format: for each course i, first the course index i and the number of registered students Ni (<= 200) are given in a line. Then in the next line, Ni student names are given. A student name consists of 3 capital English letters plus a one-digit number. Finally the last line contains the N names of students who come for a query. All the names and numbers in a line are separated by a space.

Output Specification:

For each test case, print your results in N lines. Each line corresponds to one student, in the following format: first print the student's name, then the total number of registered courses of that student, and finally the indices of the courses in increasing order. The query results must be printed in the same order as input. All the data in a line must be separated by a space, with no extra space at the end of the line.

Sample Input:

11 5

4 7

BOB5 DON2 FRA8 JAY9 KAT3 LOR6 ZOE1

1 4

ANN0 BOB5 JAY9 LOR6

2 7

ANN0 BOB5 FRA8 JAY9 JOE4 KAT3 LOR6

3 1

BOB5

5 9

AMY7 ANN0 BOB5 DON2 FRA8 JAY9 KAT3 LOR6 ZOE1

ZOE1 ANN0 BOB5 JOE4 JAY9 FRA8 DON2 AMY7 KAT3 LOR6 NON9

Sample Output:

ZOE1 2 4 5

ANN0 3 1 2 5

BOB5 5 1 2 3 4 5

JOE4 1 2

JAY9 4 1 2 4 5

FRA8 3 2 4 5

DON2 2 4 5

AMY7 1 5

KAT3 3 2 4 5

LOR6 4 1 2 4 5

NON9 0

题目大意：有N个学生，K门课，给出选择每门课的学生姓名，最后对于给出的N个学生的姓名的选课情况进行询问，要求按顺序给出每个学生的选课情况～

分析：考虑到string、cin、cout会超时，可以使用hash(26*26*26*10+10)将学生姓名变为int型，然后存储在vector里面～～

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;
int getid(char *name) {
    int id = 0;
    for(int i = 0; i < 3; i++)
        id = 26 * id + (name[i] - 'A');
    id = id * 10 + (name[3] - '0');
    return id;
}
const int maxn = 26 * 26 * 26 * 10 + 10;
vector<int> v[maxn];

int main() {
    int n, k, no, num, id = 0;
    char name[5];
    scanf("%d %d", &n, &k);
    for(int i = 0; i < k; i++) {
        scanf("%d %d", &no, &num);
        for(int j = 0; j < num; j++) {
            scanf("%s", name);
            id = getid(name);
            v[id].push_back(no);
        }
    }
    for(int i = 0; i < n; i++) {
        scanf("%s", name);
        id = getid(name);
        sort(v[id].begin(), v[id].end());
        printf("%s %lu", name, v[id].size());
        for(int j = 0; j < v[id].size(); j++)
            printf(" %d", v[id][j]);
        printf("\n");
    }
    return 0;
}
```

1040. Longest Symmetric String (25) [动态规划]

Given a string, you are supposed to output the length of the longest symmetric sub-string. For example, given “Is PAT&TAP symmetric?”, the longest symmetric sub-string is “s PAT&TAP s”, hence you must output 11.

Input Specification:

Each input file contains one test case which gives a non-empty string of length no more than 1000.

Output Specification:

For each test case, simply print the maximum length in a line.

Sample Input:

Is PAT&TAP symmetric?

Sample Output:

11

分析：dp[i][j]表示s[i]到s[j]所表示的字串是否是回文字串。只有0和1，递推方程为：

1 当s[i] == s[j] : dp[i][j] = dp[i+1][j-1]

2 当s[i] != s[j] : dp[i][j] =0

3 边界：dp[i][i] = 1, dp[i][i+1] = (s[i] == s[i+1]) ? 1 : 0

首先初始化dp[i][i] = 1, dp[i][i+1]，把长度为1和2的都初始化好，然后从L = 3开始一直到 L <= len 根据动态规划的递归方程来判断

```
#include <iostream>
using namespace std;
int dp[1010][1010];
int main() {
    string s;
    getline(cin, s);
    int len = s.length(), ans = 1;
    for(int i = 0; i < len; i++) {
        dp[i][i] = 1;
        if(i < len - 1 && s[i] == s[i+1]) {
            dp[i][i+1] = 1;
            ans = 2;
        }
    }
    for(int L = 3; L <= len; L++) {
        for(int i = 0; i + L - 1 < len; i++) {
            int j = i + L -1;
            if(s[i] == s[j] && dp[i+1][j-1] == 1) {
                dp[i][j] = 1;
                ans = L;
            }
        }
    }
    printf("%d", ans);
    return 0;
}
```

1041. Be Unique (20) [Hash散列]

Being unique is so important to people on Mars that even their lottery is designed in a unique way. The rule of winning is simple: one bets on a number chosen from [1, 104]. The first one who bets on a unique number wins. For example, if there are 7 people betting on 5 31 5 88 67 88 17, then the second one who bets on 31 wins.

Input Specification:

Each input file contains one test case. Each case contains a line which begins with a positive integer N (<=105) and then followed by N bets. The numbers are separated by a space.

Output Specification:

For each test case, print the winning number in a line. If there is no winner, print “None” instead.

Sample Input 1:

7 5 31 5 88 67 88 17

Sample Output 1:

31

Sample Input 2:

5 888 666 666 888 888

Sample Output 2:

None

题目大意：给n个数字，按照读入顺序，哪个数字是第一个在所有数字中只出现一次的数字。如果所有数字出现都超过了一次，则输出None

分析：建立一个数组，存储每个数字出现的次数，然后遍历一遍输入的顺序看是否有出现次数为1的数字

```
#include <cstdio>
using namespace std;
int a[100001], m[100000];
int main() {
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
        m[a[i]]++;
    }
    for(int i = 0; i < n; i++) {
        if(m[a[i]] == 1) {
            printf("%d", a[i]);
            return 0;
        }
    }
    printf("None");
    return 0;
}
```

1042. Shuffling Machine (20) [模拟]

Shuffling is a procedure used to randomize a deck of playing cards. Because standard shuffling techniques are seen as weak, and in order to avoid “inside jobs” where employees collaborate with gamblers by performing inadequate shuffles, many casinos employ automatic shuffling machines. Your task is to simulate a shuffling machine.

The machine shuffles a deck of 54 cards according to a given random order and repeats for a given number of times. It is assumed that the initial status of a card deck is in the following order:

S1, S2, …, S13, H1, H2, …, H13, C1, C2, …, C13, D1, D2, …, D13, J1, J2

where “S” stands for “Spade”, “H” for “Heart”, “C” for “Club”, “D” for “Diamond”, and “J” for “Joker”. A given order is a permutation of distinct integers in [1, 54]. If the number at the i-th position is j, it means to move the card from position i to position j. For example, suppose we only have 5 cards: S3, H5, C1, D13 and J2. Given a shuffling order {4, 2, 5, 3, 1}, the result will be: J2, H5, D13, S3, C1. If we are to repeat the shuffling again, the result will be: C1, H5, S3, J2, D13.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer K (<= 20) which is the number of repeat times. Then the next line contains the given order. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the shuffling results in one line. All the cards are separated by a space, and there must be no extra space at the end of the line.

Sample Input:

2

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

Sample Output:

S7 C11 C10 C12 S1 H7 H8 H9 D8 D9 S11 S12 S13 D10 D11 D12 S3 S4 S6 S10 H1 H2 C13 D2 D3 D4 H6 H3 D13 J1 J2 C1 C2 C3 C4 D1 S5 H5 H11 H12 C6 C7 C8 C9 S2 S8 S9 H10 D5 D6 D7 H4 H13 C5

分析：简单模拟。使用start和end数组保存每一次变换的开始顺序和结束顺序（以1~54的编号存储），最后根据编号与扑克牌字母数字的对应关系输出end数组

```
#include <cstdio>
using namespace std;
int main() {
    int cnt;
    scanf("%d", &cnt);
    int start[55], end[55], scan[55];
    for(int i = 1; i < 55; i++) {
        scanf("%d", &scan[i]);
        end[i] = i;
    }
    for(int i = 0; i < cnt; i++) {
        for(int j = 1; j < 55; j++)
            start[j] = end[j];
```

```
            for(int k = 1; k < 55; k++)
                end[scan[k]] = start[k];
        }
        char c[6] = {"SHCDJ"};
        for(int i = 1; i < 55; i++) {
            end[i] = end[i] - 1;
            printf("%c%d", c[end[i]/13], end[i]%13+1);
            if(i != 54) printf(" ");
        }
        return 0;
    }
```

1043. Is It a Binary Search Tree (25) [二叉查找树BST]

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

The left subtree of a node contains only nodes with keys less than the node's key.

The right subtree of a node contains only nodes with keys greater than or equal to the node's key.

Both the left and right subtrees must also be binary search trees.

If we swap the left and right subtrees of every node, then the resulting tree is called the Mirror Image of a BST.

Now given a sequence of integer keys, you are supposed to tell if it is the preorder traversal sequence of a BST or the mirror image of a BST.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<=1000). Then N integer keys are given in the next line. All the numbers in a line are separated by a space.

Output Specification:

For each test case, first print in a line "YES" if the sequence is the preorder traversal sequence of a BST or the mirror image of a BST, or "NO" if not. Then if the answer is "YES", print in the next line the postorder traversal sequence of that tree. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

Sample Input 1:

7

8 6 5 7 10 8 11

Sample Output 1:

YES

5 7 6 8 11 10 8

Sample Input 2:

7

8 10 11 8 6 7 5

Sample Output 2:

YES

11 8 10 7 5 6 8

Sample Input 3:

7

8 6 8 5 10 9 11

Sample Output 3:

NO

题目大意：给定一个整数键值序列，现请你编写程序，判断这是否是对一棵二叉搜索树或其镜像进行前序遍历的结果。

分析：假设它是二叉搜索树，一开始isMirror为FALSE，根据二叉搜索树的性质将已知的前序转换为后序，转换过程中，如果发现最后输出的后序数组长度不为n，那就设isMirror为true，然后清空后序数组，重新再转换一次（根据镜面二叉搜索树的性质），如果依旧转换后数组大小不等于n，就输出no否则输出yes

```
#include <cstdio>
#include <vector>
using namespace std;
bool isMirror;
vector<int> pre, post;
void getpost(int root, int tail) {
    if(root > tail) return ;
    int i = root + 1, j = tail;
    if(!isMirror) {
        while(i <= tail && pre[root] > pre[i]) i++;
        while(j > root && pre[root] <= pre[j]) j--;
    } else {
        while(i <= tail && pre[root] <= pre[i]) i++;
        while(j > root && pre[root] > pre[j]) j--;
    }
    if(i - j != 1) return ;
    getpost(root + 1, j);
    getpost(i, tail);
    post.push_back(pre[root]);
}
int main() {
    int n;
    scanf("%d", &n);
    pre.resize(n);
    for(int i = 0; i < n; i++)
        scanf("%d", &pre[i]);
    getpost(0, n - 1);
    if(post.size() != n) {
        isMirror = true;
        post.clear();
```

```
            getpost(0, n - 1);
        }
        if(post.size() == n) {
            printf("YES\n%d", post[0]);
            for(int i = 1; i < n; i++)
                printf(" %d", post[i]);
        } else {
            printf("NO");
        }
        return 0;
    }
```

1044. Shopping in Mars (25) [二分查找]

Shopping in Mars is quite a different experience. The Mars people pay by chained diamonds. Each diamond has a value (in Mars dollars M$). When making the payment, the chain can be cut at any position for only once and some of the diamonds are taken off the chain one by one. Once a diamond is off the chain, it cannot be taken back. For example, if we have a chain of 8 diamonds with values M$3, 2, 1, 5, 4, 6, 8, 7, and we must pay M$15. We may have 3 options:

1. Cut the chain between 4 and 6, and take off the diamonds from the position 1 to 5 (with values 3+2+1+5+4=15).
2. Cut before 5 or after 6, and take off the diamonds from the position 4 to 6 (with values 5+4+6=15).
3. Cut before 8, and take off the diamonds from the position 7 to 8 (with values 8+7=15).

Now given the chain of diamond values and the amount that a customer has to pay, you are supposed to list all the paying options for the customer.

If it is impossible to pay the exact amount, you must suggest solutions with minimum lost.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 numbers: N (<=105), the total number of diamonds on the chain, and M (<=108), the amount that the customer has to pay. Then the next line contains N positive numbers D1 … DN (Di<=103 for all i=1, …, N) which are the values of the diamonds. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print “i-j” in a line for each pair of i <= j such that Di + … + Dj = M. Note that if there are more than one solution, all the solutions must be printed in increasing order of i.

If there is no solution, output “i-j” for pairs of i <= j such that Di + … + Dj > M with (Di + … + Dj – M) minimized. Again all the solutions must be printed in increasing order of i.

It is guaranteed that the total value of diamonds is sufficient to pay the given amount.

Sample Input 1:

16 15

3 2 1 5 4 6 8 7 16 10 15 11 9 12 14 13

Sample Output 1:

1-5

4-6

7-8

11-11

Sample Input 2:

5 13

2 4 5 7 9

Sample Output 2:

2-4

4-5

题目大意，求一串的数字中连续的一段，使得这个连续的段内数字的和恰好等于所期望的值m。如果不能找到恰好等于，就找让自己付出最少的价格（总和必须大于等于所给值）的那段区间。求所有可能的结果。

分析：一开始用的简单模拟，有两个超时了。后来想到因为sum数组是递增的，所以改用二分法查找～Func函数的作用是二分查找，修改tempsum和j的值～一开始接收输入的时候可以直接保存它的sum函数，即sum[i]表示1～i的所有数字的总和～

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> sum, resultArr;
int n, m;
void Func(int i, int &j, int &tempsum) {
    int left = i, right = n;
    while(left < right) {
        int mid = (left + right) / 2;
        if(sum[mid] - sum[i-1] >= m)
            right = mid;
        else
            left = mid + 1;
    }
    j = right;
    tempsum = sum[j] - sum[i-1];
}
int main() {
    scanf("%d%d", &n, &m);
    sum.resize(n+1);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &sum[i]);
        sum[i] += sum[i-1];
    }
    int minans = sum[n];
    for(int i = 1; i <= n; i++) {
        int j, tempsum;
```

```
            Func(i, j, tempsum);
            if(tempsum > minans) continue;
            if(tempsum >= m) {
                if(tempsum < minans) {
                    resultArr.clear();
                    minans = tempsum;
                }
                resultArr.push_back(i);
                resultArr.push_back(j);
            }
        }
        for(int i = 0; i < resultArr.size(); i += 2)
            printf("%d-%d\n", resultArr[i], resultArr[i+1]);
        return 0;
    }
```

1045. Favorite Color Stripe (30) [动态规划，LIS / LCS]

Eva is trying to make her own color stripe out of a given one. She would like to keep only her favorite colors in her favorite order by cutting off those unwanted pieces and sewing the remaining parts together to form her favorite color stripe.

It is said that a normal human eye can distinguish about less than 200 different colors, so Eva's favorite colors are limited. However the original stripe could be very long, and Eva would like to have the remaining favorite stripe with the maximum length. So she needs your help to find her the best result.

Note that the solution might not be unique, but you only have to tell her the maximum length. For example, given a stripe of colors {2 2 4 1 5 5 6 3 1 1 5 6}. If Eva's favorite colors are given in her favorite order as {2 3 1 5 6}, then she has 4 possible best solutions {2 2 1 1 1 5 6}, {2 2 1 5 5 5 6}, {2 2 1 5 5 6 6}, and {2 2 3 1 1 5 6}.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<=200) which is the total number of colors involved (and hence the colors are numbered from 1 to N). Then the next line starts with a positive integer M (<=200) followed by M Eva's favorite color numbers given in her favorite order. Finally the third line starts with a positive integer L (<=10000) which is the length of the given stripe, followed by L colors on the stripe. All the numbers in a line are separated by a space.

Output Specification:

For each test case, simply print in a line the maximum length of Eva's favorite stripe.

Sample Input:

6

5 2 3 1 5 6

12 2 2 4 1 5 5 6 3 1 1 5 6

Sample Output:

7

题目大意：给出m中颜色作为喜欢的颜色（同时也给出顺序），然后给出一串长度为L的颜色序列，现在要去掉这个序列中的不喜欢的颜色，然后求剩下序列的一个子序列，使得这个子序列表示的颜色顺序符合自己喜欢的颜色的顺序，不一定要所有喜欢的颜色都出现。

分析：因为喜欢的颜色是不重复的，把喜欢的颜色的序列依次存储到数组中，book[i] = j表示i颜色的下标为j。先在输入的时候剔除不在喜欢的序列中的元素，然后把剩余的保存在数组a中。按照最长不下降子序列的方式做，对于从前到后的每一个i，如果它前面的所有的j，一下子找到了一个j的下标book[j]比book[i]小，此时就更新dp[i]使它 = max(dp[i], dp[j] + 1)；并且同时再每一次遍历完成一次j后更新maxn的值为长度的最大值，最后输出maxn～

```
#include <iostream>
#include <vector>
using namespace std;
int book[201], a[10001], dp[10001];
int main() {
    int n, m, x, l, num = 0, maxn = 0;
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= m; i++) {
        scanf("%d", &x);
        book[x] = i;
    }
    scanf("%d", &l);
    for(int i = 0; i < l; i++) {
        scanf("%d", &x);
        if(book[x] >= 1)
            a[num++] = book[x];
    }
    for(int i = 0; i < num; i++) {
        dp[i] = 1;
        for(int j = 0; j < i; j++)
            if(a[i] >= a[j])
                dp[i] = max(dp[i], dp[j] + 1);
        maxn = max(dp[i], maxn);
    }
    printf("%d", maxn);
    return 0;
}
```

1046. Shortest Distance (20) [模拟]

The task is really simple: given N exits on a highway which forms a simple cycle, you are supposed to tell the shortest distance between any pair of exits.

Input Specification:

Each input file contains one test case. For each case, the first line contains an integer N (in [3, 105]), followed by N integer distances D1 D2 … DN, where Di is the distance between the i-th and the (i+1)-st exits, and DN is between the N-th and the 1st exits. All the numbers in a line are separated by a space. The second line gives a positive integer M (<=104), with M lines follow, each contains a pair of exit numbers, provided that the exits are numbered from 1 to N. It is guaranteed that the total round trip distance is no more than 107.

Output Specification:

For each test case, print your results in M lines, each contains the shortest distance between the corresponding given pair of exits.

Sample Input:

5 1 2 4 14 9

3

1 3

2 5

4 1

Sample Output:

3

10

7

分析：简单模拟。所有结点连起来会形成一个环形，dis[i]存储第1个结点到第i个结点的下一个结点的距离，sum保存整个路径一圈的总和值。

求得结果就是dis[right – 1] – dis[left – 1]和 sum – dis[right – 1] – dis[left – 1]中较小的那一个～～

注意：可能left和right的顺序颠倒了，这时候要把left和right的值交换～

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n;
    scanf("%d", &n);
    vector<int> dis(n + 1);
    int sum = 0, left, right, cnt;
    for(int i = 1; i <= n; i++) {
        int temp;
        scanf("%d", &temp);
        sum += temp;
        dis[i] = sum;
    }
    scanf("%d", &cnt);
    for(int i = 0; i < cnt; i++) {
        scanf("%d %d", &left, &right);
        if(left > right)
            swap(left, right);
        int temp = dis[right - 1] - dis[left - 1];
        printf("%d\n", min(temp, sum - temp));
    }
    return 0;
}
```

1047. Student List for Course (25) [不定长数组vector， STL的使用]

Zhejiang University has 40000 students and provides 2500 courses. Now given the registered course list of each student, you are supposed to output the student name lists of all the courses.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 numbers: N (<=40000), the total number of students, and K (<=2500), the total number of courses. Then N lines follow, each contains a student's name (3 capital English letters plus a one-digit number), a positive number C (<=20) which is the number of courses that this student has registered, and then followed by C course numbers. For the sake of simplicity, the courses are numbered from 1 to K.

Output Specification:

For each test case, print the student name lists of all the courses in increasing order of the course numbers. For each course, first print in one line the course number and the number of registered students, separated by a space. Then output the students' names in alphabetical order. Each name occupies a line.

Sample Input:

10 5

ZOE1 2 4 5

ANN0 3 5 2 1

BOB5 5 3 4 2 1 5

JOE4 1 2

JAY9 4 1 2 5 4

FRA8 3 4 2 5

DON2 2 4 5

AMY7 1 5

KAT3 3 5 4 2

LOR6 4 2 4 1 5

Sample Output:

1 4

ANN0

BOB5

JAY9

LOR6

2 7

ANN0

BOB5

FRA8

JAY9

JOE4

KAT3

LOR6

3 1

BOB5

4 7

BOB5

DON2

FRA8

JAY9

KAT3

LOR6

ZOE1

5 9

AMY7

ANN0

BOB5

DON2

FRA8

JAY9

KAT3

LOR6

ZOE1

题目大意：给出选课人数和课程数目，然后再给出每个人的选课情况，请针对每门课程输出选课人数以及所有选该课的学生姓名，按照字典序～

分析：建立int的二维数组，course[i][j] = k表示第i号课程上的人的一个列表，k是上这个课的学生的姓名所在的字符数组name[i][j]的i下标～～ 注意：strcmp返回的不一定是-1，0，1这几个数字，要返回bool变量还需要在后面添加strcmp是大于0还是小于0～～

```
1   #include <iostream>
```

```cpp
#include <vector>
#include <algorithm>
#include <string.h>
using namespace std;
char name[40010][5];
vector<int> course[2510];
bool cmp1(int a, int b) {
    return strcmp(name[a], name[b]) < 0;
}
int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    for(int i = 0; i < n; i++) {
        int cnt, temp;
        scanf("%s %d", name[i], &cnt);
        for(int j = 0; j < cnt; j++) {
            scanf("%d", &temp);
            course[temp].push_back(i);
        }
    }
    for(int i = 1; i <= k; i++) {
        printf("%d %d\n", i, course[i].size());
        sort(course[i].begin(), course[i].end(), cmp1);
        for(int j = 0; j < course[i].size(); j++)
            printf("%s\n", name[course[i][j]]);
    }
    return 0;
}
```

1048. Find Coins (25) [Hash散列]

Eva loves to collect coins from all over the universe, including some other planets like Mars. One day she visited a universal shopping mall which could accept all kinds of coins as payments. However, there was a special requirement of the payment: for each bill, she could only use exactly two coins to pay the exact amount. Since she has as many as 105 coins with her, she definitely needs your help. You are supposed to tell her, for any given amount of money, whether or not she can find two coins to pay for it.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive numbers: N (<=105, the total number of coins) and M(<=103, the amount of money Eva has to pay). The second line contains N face values of the coins, which are all positive numbers no more than 500. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the two face values V1 and V2 (separated by a space) such that V1 + V2 = M and V1 <= V2. If such a solution is not unique, output the one with the smallest V1. If there is no solution, output “No Solution” instead.

Sample Input 1:

8 15

1 2 8 7 2 4 11 15

Sample Output 1:

4 11

Sample Input 2:

7 14

1 8 7 2 4 11 15

Sample Output 2:

No Solution

题目大意：给出n个正整数和一个正整数m，问n个数字中是否存在一对数字a和b(a <= b),使a+b=m成立。如果有多个，输出a最小的那一对。

分析：建立数组a保存每个数字出现的次数，然后判断输出

```
#include <iostream>
using namespace std;
int a[1001];
int main() {
    int n, m, temp;
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++) {
        scanf("%d", &temp);
        a[temp]++;
    }
    for(int i = 0; i < 1001; i++) {
        if(a[i]) {
            a[i]--;
            if(m > i && a[m-i]) {
                printf("%d %d", i, m - i);
                return 0;
            }
            a[i]++;
        }
    }
    printf("No Solution");
    return 0;
}
```

1049. Counting Ones (30) [数学问题]

The task is simple: given any positive integer N, you are supposed to count the total number of 1's in the decimal form of the integers from 1 to N. For example, given N being 12, there are five 1's in 1, 10, 11, and 12.

Input Specification:

Each input file contains one test case which gives the positive N (<=230).

Output Specification:

For each test case, print the number of 1's in one line.

Sample Input:

12

Sample Output:

5

题目大意：给出一个数字n，求1~n的所有数字里面出现1的个数

分析：这是一道数学问题。从第一位（个位）到最高位，设now为当前位的数字，left为now左边的所有数字构成的数字，right是now右边的所有数字构成的数字。只需要一次次累加对于当前位now来说可能出现1的个数，然后把它们累加即可。a表示当前的个位为1，十位为10，百位为100类推。

对于now，有三种情况：

1.now == 0 : 那么 ans += left * a; //因为now==0说明now位只有在left从0~left-1的时候会产生1，所以会产生left次，但是又因为右边会重复从0~999…出现a次

2.now == 1 : ans += left * a + right + 1;//now = 1的时候就要比上一步多加一个当now为1的时候右边出现0~right个数导致的now为1的次数

3.now >= 2 : ans += (left + 1) * a;//now大于等于2就左边0~left的时候会在now位置产生1，所以会产生left次，但是又因为右边会重复从0~999…出现a次

```
#include <iostream>
using namespace std;
int main() {
    int n, left = 0, right = 0, a = 1, now = 1, ans = 0;
    scanf("%d", &n);
    while(n / a) {
        left = n / (a * 10), now = n / a % 10, right = n % a;
        if(now == 0) ans += left * a;
        else if(now == 1) ans += left * a + right + 1;
        else ans += (left + 1) * a;
        a = a * 10;
    }
    printf("%d", ans);
    return 0;
}
```

1050. String Subtraction (20) [Hash散列]

Given two strings S1 and S2, S = S1 – S2 is defined to be the remaining string after taking all the characters in S2 from S1. Your task is simply to calculate S1 – S2 for any given strings. However, it might not be that simple to do it fast.

Input Specification:

Each input file contains one test case. Each case consists of two lines which gives S1 and S2, respectively. The string lengths of both strings are no more than 104. It is guaranteed that all the characters are visible ASCII codes and white space, and a new line character signals the end of a string.

Output Specification:

For each test case, print S1 – S2 in one line.

Sample Input:

They are students.

aeiou

Sample Output:

Thy r stdnts.

题目大意：给出两个字符串，在第一个字符串中删除第二个字符串中出现过的所有字符并输出～

分析：用flag[256]数组变量标记str2出现过的字符为true，输出str1的时候根据flag[str1[i]]是否为true,如果是true就不输出

注意：使用int lens1 = strlen(s1);int lens2 = strlen(s2);的形式，否则直接放在for循环里面会超时～ 如果使用gets前面请使用char str[100000]而非char *str～～

```
#include <iostream>
#include <string>
#include <string.h>
using namespace std;
char s1[100000], s2[100000];
int main() {
    cin.getline(s1, 100000);
    cin.getline(s2, 100000);
    int lens1 = strlen(s1), lens2 = strlen(s2);
    bool flag[256] = {false};
    for(int i = 0; i < lens2; i++)
        flag[s2[i]] = true;
    for(int i = 0; i < lens1; i++) {
        if(!flag[s1[i]])
            printf("%c", s1[i]);
    }
    return 0;
}
```

1051. Pop Sequence (25) [栈模拟]

Given a stack which can keep M numbers at most. Push N numbers in the order of 1, 2, 3, …, N and pop randomly. You are supposed to tell if a given sequence of numbers is a possible pop sequence of the stack. For example, if M is 5 and N is 7, we can obtain 1, 2, 3, 4, 5, 6, 7 from the stack, but not 3, 2, 1, 7, 5, 6, 4.

Input Specification:

Each input file contains one test case. For each case, the first line contains 3 numbers (all no more than 1000): M (the maximum capacity of the stack), N (the length of push sequence), and K (the number of pop sequences to be checked). Then K lines follow, each contains a pop sequence of N numbers. All the numbers in a line are separated by a space.

Output Specification:

For each pop sequence, print in one line “YES” if it is indeed a possible pop sequence of the stack, or “NO” if not.

Sample Input:

5 7 5

1 2 3 4 5 6 7

3 2 1 7 5 6 4

7 6 5 4 3 2 1

5 6 4 3 7 2 1

1 7 6 5 4 3 2

Sample Output:

YES

NO

NO

YES

NO

题目大意：有个容量限制为m的栈，分别把1，2，3，…，n入栈,给出一个系列出栈顺序，问这些出栈顺序是否可能～

分析：按照要求进行模拟。先把输入的序列接收进数组v。然后按顺序1~n把数字进栈，每进入一个数字，判断有没有超过最大范围，超过了就break。如果没超过，设current = 1，从数组的第一个数字开始，看看是否与栈顶元素相等，while相等就一直弹出栈，不相等就继续按顺序把数字压入栈～～最后根据变量flag的bool值输出yes或者no～

```
#include <iostream>
#include <stack>
#include <vector>
using namespace std;
int main() {
    int m, n, k;
    scanf("%d %d %d", &m, &n, &k);
    for(int i = 0; i < k; i++) {
        bool flag = false;
        stack<int> s;
        vector<int> v(n + 1);
        for(int j = 1; j <= n; j++)
```

```
            scanf("%d", &v[j]);
        int current = 1;
        for(int j = 1; j <= n; j++) {
            s.push(j);
            if(s.size() > m) break;
            while(!s.empty() && s.top() == v[current]) {
                s.pop();
                current++;
            }
        }
        if(current == n + 1) flag = true;
        if(flag) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

1052. Linked List Sorting (25) [链表]

A linked list consists of a series of structures, which are not necessarily adjacent in memory. We assume that each structure contains an integer key and a Next pointer to the next structure. Now given a linked list, you are supposed to sort the structures according to their key values in increasing order.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive N (< 105) and an address of the head node, where N is the total number of nodes in memory and the address of a node is a 5-digit positive integer. NULL is represented by -1.

Then N lines follow, each describes a node in the format:

Address Key Next

where Address is the address of the node in memory, Key is an integer in [-105, 105], and Next is the address of the next node. It is guaranteed that all the keys are distinct and there is no cycle in the linked list starting from the head node.

Output Specification:

For each test case, the output format is the same as that of the input, where N is the total number of nodes in the list and all the nodes must be sorted order.

Sample Input:

5 00001

11111 100 -1

00001 0 22222

33333 100000 11111

12345 -1 33333

22222 1000 12345

Sample Output:

5 12345

12345 -1 00001

00001 0 11111

11111 100 22222

22222 1000 33333

33333 100000 -1

题目大意：给出一个链表，将链表排序，然后把链表上的结点按照data值的从小到大顺序输出

分析：建立结构体数组，按照从首地址开始的顺序（直到-1）遍历一遍整个链表，将在链表中的结点的flag标记为true，并且统计cnt（有效结点的个数）。（因为有的结点根本不在链表中）

然后将链表进行排序，如果flag == false就把他们移动到后面（即：reuturn a.flag > b.flag）,最后只输出前cnt个链表的信息～

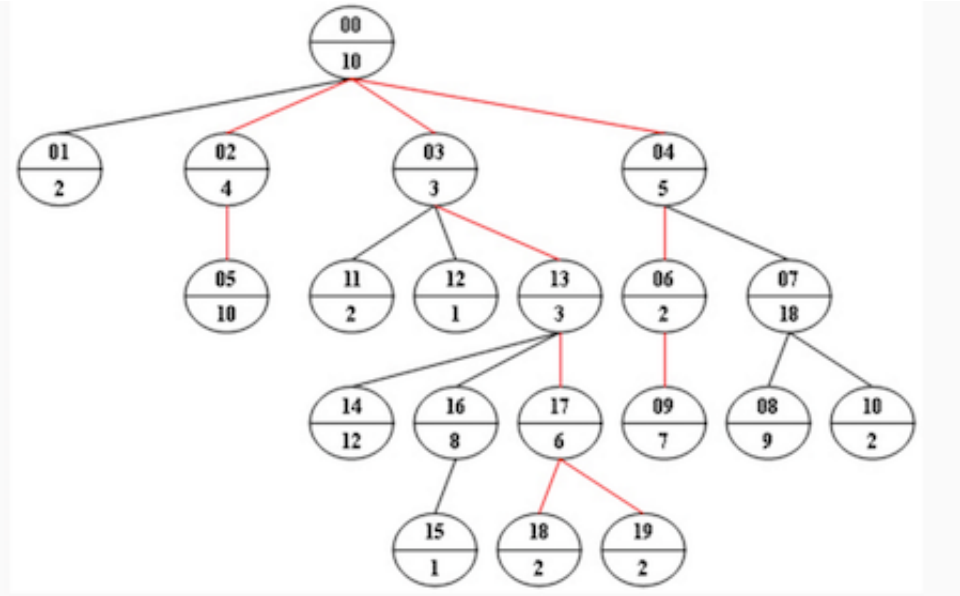
```
#include <iostream>
#include <algorithm>
using namespace std;
struct NODE {
    int address, key, next;
    bool flag;
}node[100000];
int cmp1(NODE a, NODE b) {
    return !a.flag || !b.flag ? a.flag > b.flag : a.key < b.key;
}
int main() {
    int n, cnt = 0, s, a, b, c;
    scanf("%d%d", &n, &s);
    for(int i = 0; i < n; i++) {
        scanf("%d%d%d", &a, &b, &c);
        node[a] = {a, b, c, false};
    }
    for(int i = s; i != -1; i = node[i].next) {
        node[i].flag = true;
        cnt++;
    }
    if(cnt == 0) {
        printf("0 -1");
    } else {
        sort(node, node + 100000, cmp1);
        printf("%d %05d\n", cnt, node[0].address);
        for(int i = 0; i < cnt; i++) {
            printf("%05d %d ", node[i].address, node[i].key);
            if(i != cnt - 1)
                printf("%05d\n", node[i + 1].address);
            else
                printf("-1\n");
        }
```

```
        }
        return 0;
    }
```

1053. Path of Equal Weight (30) [树的遍历]

Given a non-empty tree with root R, and with weight Wi assigned to each tree node Ti. The weight of a path from R to L is defined to be the sum of the weights of all the nodes along the path from R to any leaf node L.

Now given any weighted tree, you are supposed to find all the paths with their weights equal to a given number. For example, let's consider the tree showed in Figure 1: for each node, the upper number is the node ID which is a two-digit number, and the lower number is the weight of that node. Suppose that the given number is 24, then there exists 4 different paths which have the same given weight: {10 5 2 7}, {10 4 10}, {10 3 3 6 2} and {10 3 3 6 2}, which correspond to the red edges in Figure 1.



Input Specification:

Each input file contains one test case. Each case starts with a line containing 0 < N <= 100, the number of nodes in a tree, M (< N), the number of non-leaf nodes, and 0 < S < 230, the given weight number. The next line contains N positive numbers where Wi (<1000) corresponds to the tree node Ti. Then M lines follow, each in the format:

ID K ID[1] ID[2] … ID[K]

where ID is a two-digit number representing a given non-leaf node, K is the number of its children, followed by a sequence of two-digit ID's of its children. For the sake of simplicity, let us fix the root ID to be 00.

Output Specification:

For each test case, print all the paths with weight S in non-increasing order. Each path occupies a line with printed weights from the root to the leaf in order. All the numbers must be separated by a space with no extra space at the end of the line.

Note: sequence {A1, A2, …, An} is said to be greater than sequence {B1, B2, …, Bm} if there exists 1 <= k < min{n, m} such that Ai = Bi for i=1, … k, and Ak+1 > Bk+1.

Sample Input:

20 9 24

10 2 4 3 5 10 2 18 9 7 2 2 1 3 12 1 8 6 2 2

00 4 01 02 03 04

02 1 05

04 2 06 07

03 3 11 12 13

06 1 09

07 2 08 10

16 1 15

13 3 14 16 17

17 2 18 19

Sample Output:

10 5 2 7

10 4 10

10 3 3 6 2

10 3 3 6 2

题目大意：给出树的结构和权值，找从根结点到叶子结点的路径上的权值相加之和等于给定目标数的路径，并且从大到小输出路径

分析：对于接收孩子结点的数据时，每次完全接收完就对孩子结点按照权值进行排序（序号变，根据权值变），这样保证深度优先遍历的时候直接输出就能输出从大到小的顺序。记录路径采取这样的方式：首先建立一个path数组，传入一个nodeNum记录对当前路径来说这是第几个结点（这样直接在path[nodeNum]里面存储当前结点的孩子结点的序号，这样可以保证在先判断return的时候，path是从0~numNum-1的值确实是要求的路径结点）。然后每次要遍历下一个孩子结点的之前，令path[nodeNum] = 孩子结点的序号，这样就保证了在return的时候当前path里面从0~nodeNum-1的值就是要输出的路径的结点序号，输出这个序号的权值即可，直接在return语句里面输出。

注意：当sum==target的时候，记得判断是否孩子结点是空，要不然如果不空说明没有到底部，就直接return而不是输出路径。。。（这对接下来的孩子结点都是正数才有用，负权无用）。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int target;
struct NODE {
    int w;
    vector<int> child;
};
```

```
vector<NODE> v;
vector<int> path;
void dfs(int index, int nodeNum, int sum) {
    if(sum > target) return ;
    if(sum == target) {
        if(v[index].child.size() != 0) return;
        for(int i = 0; i < nodeNum; i++)
            printf("%d%c", v[path[i]].w, i != nodeNum - 1 ? ' ' : '\n');
        return ;
    }
    for(int i = 0; i < v[index].child.size(); i++) {
        int node = v[index].child[i];
        path[nodeNum] = node;
        dfs(node, nodeNum + 1, sum + v[node].w);
    }

}
int cmp1(int a, int b) {
    return v[a].w > v[b].w;
}
int main() {
    int n, m, node, k;
    scanf("%d %d %d", &n, &m, &target);
    v.resize(n), path.resize(n);
    for(int i = 0; i < n; i++)
        scanf("%d", &v[i].w);
    for(int i = 0; i < m; i++) {
        scanf("%d %d", &node, &k);
        v[node].child.resize(k);
        for(int j = 0; j < k; j++)
            scanf("%d", &v[node].child[j]);
        sort(v[node].child.begin(), v[node].child.end(), cmp1);
    }
    dfs(0, 1, v[0].w);
    return 0;
}
```

1054. The Dominant Color (20) [map映射， STL的使用]

Behind the scenes in the computer's memory, color is always talked about as a series of 24 bits of information for each pixel. In an image, the color with the largest proportional area is called the dominant color. A strictly dominant color takes more than half of the total area. Now given an image of resolution M by N (for example, 800×600), you are supposed to point out the strictly dominant color.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive numbers: M (<=800) and N (<=600) which are the resolutions of the image. Then N lines follow, each contains M digital colors in the range [0, 224). It is guaranteed that the strictly dominant color exists for each input image. All the numbers in a line are separated by a space.

Output Specification:

For each test case, simply print the dominant color in a line.

Sample Input:

5 3

0 0 255 16777215 24

24 24 0 0 24

24 0 24 24 24

Sample Output:

24

题目大意：选取主色调，就是M列N行的矩阵里面出现次数多余一半的那个数字～

分析：STL中map的应用～

使用arr[i] = j表示i元素在矩阵中出现了j次，在输入的同时比较arr当前的值是否已经超过半数，如果超过，就直接输出该数字并退出程序～

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    int m, n;
    scanf("%d %d", &m, &n);
    map<int, int> arr;
    int half = m * n / 2;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            int temp;
            scanf("%d", &temp);
            arr[temp]++;
            if(arr[temp] > half) {
                printf("%d", temp);
                return 0;
            }
        }
    }
    return 0;
}
```

1055. The World's Richest (25) [排序]

Forbes magazine publishes every year its list of billionaires based on the annual ranking of the world's wealthiest people. Now you are supposed to simulate this job, but concentrate only on the people in a certain range of ages. That is, given the net worths of N people, you must find the M richest people in a given range of their ages.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers: N (<=105) – the total number of people, and K (<=103) – the number of queries. Then N lines follow, each contains the name (string of no more than 8 characters without space), age (integer in (0, 200]), and the net worth (integer in [-106, 106]) of a person. Finally there are K lines of queries, each contains three positive integers: M (<= 100) – the maximum number of outputs, and [Amin, Amax] which are the range of ages. All the numbers in a line are separated by a space.

Output Specification:

For each query, first print in a line Case #X: where X is the query number starting from 1. Then output the M richest people with their ages in the range [Amin, Amax]. Each person's information occupies a line, in the format

Name Age Net_Worth

The outputs must be in non-increasing order of the net worths. In case there are equal worths, it must be in non-decreasing order of the ages. If both worths and ages are the same, then the output must be in non-decreasing alphabetical order of the names. It is guaranteed that there is no two persons share all the same of the three pieces of information. In case no one is found, output "None".

Sample Input:

12 4

Zoe_Bill 35 2333

Bob_Volk 24 5888

Anny_Cin 95 999999

Williams 30 -22

Cindy 76 76000

Alice 18 88888

Joe_Mike 32 3222

Michael 5 300000

Rosemary 40 5888

Dobby 24 5888

Billy 24 5888

Nobody 5 0

4 15 45

4 30 35

4 5 95

1 45 50

Sample Output:

Case #1:

Alice 18 88888

Billy 24 5888

Bob_Volk 24 5888

Dobby 24 5888

Case #2:

Joe_Mike 32 3222

Zoe_Bill 35 2333

Williams 30 -22

Case #3:

Anny_Cin 95 999999

Michael 5 300000

Alice 18 88888

Cindy 76 76000

Case #4:

None

题目大意：给出n个人的姓名、年龄和拥有的钱，然后进行k次查询，每次查询输出在年龄区间内的财富值的从大到小的前m个人的信息。如果财富值相同就就先输出年龄小的，如果年龄相同就把名字按照字典序排序输出～

分析：不能先排序然后根据每一个条件再新建一个数组、对新数组排序的方法，这样测试点2会超时～因为n和m的悬殊太大了，n有10的5次方，m却只有100个。所以先把所有的人按照财富值排序，再建立一个数组book标记每个年龄段拥有的人的数量，遍历数组并统计相应年龄的人数，当 当前年龄的人的数量不超过100的时候压入新的数组，多出来的不要压入新数组中（也就是说只取每个年龄的前100名，因为一个年龄段最小的就是一个年龄，即使这样也不会超过100个需要输出），再从这个新的数组里面取符合相应年龄的人的信息～～

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <cstring>
using namespace std;
struct node {
    char name[10];
    int age, money;
};
int cmp1(node a, node b) {
    if(a.money != b.money)
        return a.money > b.money;
    else if(a.age != b.age)
        return a.age < b.age;
```

```
    else
        return (strcmp(a.name, b.name) < 0);
}

int main() {
    int n, k, num, amin, amax;
    scanf("%d %d", &n, &k);
    vector<node> vt(n), v;
    vector<int> book(205, 0);
    for(int i = 0; i < n; i++)
        scanf("%s %d %d", vt[i].name, &vt[i].age, &vt[i].money);
    sort(vt.begin(), vt.end(), cmp1);
    for(int i = 0; i < n; i++) {
        if(book[vt[i].age] < 100) {
            v.push_back(vt[i]);
            book[vt[i].age]++;
        }
    }
    for(int i = 0; i < k; i++) {
        scanf("%d %d %d", &num, &amin, &amax);
        vector<node> t;
        for(int j = 0; j < v.size(); j++) {
            if(v[j].age >= amin && v[j].age <= amax)
                t.push_back(v[j]);
        }
        if(i != 0) printf("\n");
        printf("Case #%d:", i + 1);
        int flag = 0;
        for(int j = 0; j < num && j < t.size(); j++) {
            printf("\n%s %d %d", t[j].name, t[j].age, t[j].money);
            flag = 1;
        }
        if(flag == 0) printf("\nNone");
    }
    return 0;
}
```

1056. Mice and Rice (25) [queue的用法]

Mice and Rice is the name of a programming contest in which each programmer must write a piece of code to control the movements of a mouse in a given map. The goal of each mouse is to eat as much rice as possible in order to become a FatMouse.

First the playing order is randomly decided for NP programmers. Then every NG programmers are grouped in a match. The fattest mouse in a group wins and enters the next turn. All the losers in this turn are ranked the same. Every NG winners are then grouped in the next match until a final winner is determined.

For the sake of simplicity, assume that the weight of each mouse is fixed once the programmer submits his/her code. Given the weights of all the mice and the initial playing order, you are supposed to output the ranks for the programmers.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers: NP and NG (<= 1000), the number of programmers and the maximum number of mice in a group, respectively. If there are less than NG mice at the end of the player's list, then all the mice left will be put into the last group. The second line contains NP distinct non-negative numbers Wi (i=0,…NP-1) where each Wi is the weight of the i-th mouse respectively. The third line gives the initial playing order which is a permutation of 0,…NP-1 (assume that the programmers are numbered from 0 to NP-1). All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the final ranks in a line. The i-th number is the rank of the i-th programmer, and all the numbers must be separated by a space, with no extra space at the end of the line.

Sample Input:

11 3

25 18 0 46 37 3 19 22 57 56 10

6 0 8 7 10 5 9 1 4 2 3

Sample Output:

5 5 5 2 5 5 5 3 1 3 5

题目大意：np为老鼠的数量，ng为每组最多g个老鼠。先给出np个老鼠的重量，再给出老鼠的初始顺序（第i名的老鼠是第j号，j从0开始）。每ng个老鼠分为一组，对于每组老鼠，选出最重的那个，晋级下一轮比赛，然后依次再以np个老鼠一组分类，然后选出重量最大的。。。直到只剩下一只老鼠，排名为1.输出为老鼠的排名，这个排名是按照原输入老鼠的顺序输出的～

分析：设立结构体node表示老鼠，里面包括weight重量，index是按照排名后的顺序的老鼠的下标，index0是排名前老鼠的下标。rank是最终要输出的老鼠的排名。

先将所有的老鼠按照排名后的顺序依次放入队列中，对于队列中的每一个老鼠，按照分组后选择最大重量的比赛规则，将每次分组获得第一的老鼠放入队列尾部。此处有一点，如果当前剩下的老鼠可以分为group组，那么这一组里面没有晋级的老鼠排名就是group+1.此处解释一下：

因为对于共有group组的老鼠，每组晋级一个，也就是说最终这一轮能晋级的是group个老鼠，那么没有晋级的所有人就是group+1名，就像有4个人晋级下一轮，那么所有没晋级的这一轮就都是并列第5名。

group的计算方法是：如果当前轮次的人数正好可以被每组ng人的ng整除，那么就有人数/ng个组。如果不能被整除，就有剩下来的一些老鼠分为一组，就是人数/ng + 1组。（这是求得group的方法）

cnt用来控制当前的组内第几个人，如果cnt能够被ng整除了说明已经是下一组了，就cnt = 0；否则cnt不断++，同时将最重的老鼠体重赋值给maxn，最重的老鼠的node赋值给maxnode。

最后将结果结构体w排序，按照先前保存的index0的顺序排序，因为题目要求是必须按照题目所给的输入顺序输出的，排序后即可按序输出～

```
1   #include <iostream>
2   #include <queue>
3   #include <vector>
```

```cpp
#include <algorithm>
using namespace std;
struct node {
    int weight, index, rank, index0;
};
bool cmp1(node a, node b) {
    return a.index0 < b.index0;
}
int main() {
    int n, g, num;
    scanf("%d%d", &n, &g);
    vector<int> v(n);
    vector<node> w(n);
    for(int i = 0; i < n; i++)
        scanf("%d", &v[i]);
    for(int i = 0; i < n; i++) {
        scanf("%d", &num);
        w[i].weight = v[num];
        w[i].index = i;
        w[i].index0 = num;
    }
    queue<node> q;
    for(int i = 0; i < n; i++)
        q.push(w[i]);
    while(!q.empty()) {
        int size = q.size();
        if(size == 1) {
            node temp = q.front();
            w[temp.index].rank = 1;
            break;
        }
        int group = size / g;
        if(size % g != 0)
            group += 1;
        node maxnode;
        int maxn = -1, cnt = 0;
        for(int i = 0; i < size; i++) {
            node temp = q.front();
            w[temp.index].rank = group + 1;
            q.pop();
            cnt++;
            if(temp.weight > maxn) {
                maxn = temp.weight;
                maxnode = temp;
            }
            if(cnt == g || i == size - 1) {
                cnt = 0;
                maxn = -1;
                q.push(maxnode);
            }
        }
    }
```

```
56        sort(w.begin(), w.end(), cmp1);
57        for(int i = 0; i < n; i++) {
58            if(i != 0) printf(" ");
59            printf("%d", w[i].rank);
60        }
61        return 0;
62    }
```

1057. Stack (30) [树状数组]

Stack is one of the most fundamental data structures, which is based on the principle of Last In First Out (LIFO). The basic operations include Push (inserting an element onto the top position) and Pop (deleting the top element). Now you are supposed to implement a stack with an extra operation: PeekMedian — return the median value of all the elements in the stack. With N elements, the median value is defined to be the (N/2)-th smallest element if N is even, or ((N+1)/2)-th if N is odd.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<= 105). Then N lines follow, each contains a command in one of the following 3 formats:

Push key

Pop

PeekMedian

where key is a positive integer no more than 105.

Output Specification:

For each Push command, insert key into the stack and output nothing. For each Pop or PeekMedian command, print in a line the corresponding returned value. If the command is invalid, print “Invalid” instead.

Sample Input:

17

Pop

PeekMedian

Push 3

PeekMedian

Push 2

PeekMedian

Push 1

PeekMedian

Pop

Pop

Push 5

Push 4

PeekMedian

Pop

Pop

Pop

Pop

Sample Output:

Invalid

Invalid

3

2

2

1

2

4

4

5

3

Invalid

题目大意：现请你实现一种特殊的堆栈，它多了一种操作叫“查中值”，即返回堆栈中所有元素的中值。对于N个元素，若N是偶数，则中值定义为第N/2个最小元；若N是奇数，则中值定义为第(N+1)/2个最小元。

分析：用排序查询的方法会超时～～用树状数组，即求第k = (s.size() + 1) / 2大的数。查询小于等于x的数的个数是否等于k的时候用二分法更快～

```
#include <iostream>
#include <stack>
#define lowbit(i) ((i) & (-i))
const int maxn = 100010;
using namespace std;
int c[maxn];
stack<int> s;
void update(int x, int v) {
    for(int i = x; i < maxn; i += lowbit(i))
        c[i] += v;
```

```
}
int getsum(int x) {
    int sum = 0;
    for(int i = x; i >= 1; i -= lowbit(i))
        sum += c[i];
    return sum;
}
void PeekMedian() {
    int left = 1, right = maxn, mid, k = (s.size() + 1) / 2;
    while(left < right) {
        mid = (left + right) / 2;
        if(getsum(mid) >= k)
            right = mid;
        else
            left = mid + 1;
    }
    printf("%d\n", left);
}
int main() {
    int n, temp;
    scanf("%d", &n);
    char str[15];
    for(int i = 0; i < n; i++) {
        scanf("%s", str);
        if(str[1] == 'u') {
            scanf("%d", &temp);
            s.push(temp);
            update(temp, 1);
        } else if(str[1] == 'o') {
            if(!s.empty()) {
                update(s.top(), -1);
                printf("%d\n", s.top());
                s.pop();
            } else {
                printf("Invalid\n");
            }
        } else {
            if(!s.empty())
                PeekMedian();
            else
                printf("Invalid\n");
        }
    }
    return 0;
}
```

1058. A+B in Hogwarts (20) [进制转换]

If you are a fan of Harry Potter, you would know the world of magic has its own currency system — as Hagrid explained it to Harry, “Seventeen silver Sickles to a Galleon and twenty-nine Knuts to a Sickle, it’s easy enough.” Your job is to write a program to compute A+B where A and B are given in the standard form of “Galleon.Sickle.Knut” (Galleon is an integer in [0, 107], Sickle is an integer in [0, 17), and Knut is

an integer in [0, 29)).

Input Specification:

Each input file contains one test case which occupies a line with A and B in the standard form, separated by one space.

Output Specification:

For each test case you should output the sum of A and B in one line, with the same format as the input.

Sample Input:

3.2.1 10.16.27

Sample Output:

14.1.28

题目大意：17个Sickle对换一个Galleon，29个Knut对换一个Sickle。根据Galleon.Sickle.Knut的方式相加A和B

分析：把A和B都化为Knut的形式，然后相加，最后转换为Galleon.Sickle.Knut的形式～

注意：A和B相加有可能超出int范围，使用long long格式存储数据～

```
#include <iostream>
using namespace std;
int main() {
    long long a, b, c, d, e, f;
    scanf("%lld.%lld.%lld %lld.%lld.%lld", &a, &b, &c, &d, &e, &f);
    long long num = c + b * 29 + a * 29 * 17 + f + e * 29 + d * 29 * 17;
    long long g = num / (17 * 29);
    num = num % (17 * 29);
    printf("%lld.%lld.%lld", g, num / 29, num % 29);
    return 0;
}
```

1059. Prime Factors (25) [素数表的建立]

Given any positive integer N, you are supposed to find all of its prime factors, and write them in the format N = p1^k1 * p2^k2 *…*pm^km.

Input Specification:

Each input file contains one test case which gives a positive integer N in the range of long int.

Output Specification:

Factor N in the format N = p1^k1 * p2^k2 *…*pm^km, where pi's are prime factors of N in increasing order, and the exponent ki is the number of pi — hence when there is only one pi, ki is 1 and must NOT be printed out.

Sample Input:

97532468

Sample Output:

97532468=2^2*11*17*101*1291

题目大意：给出一个整数，按照从小到大的顺序输出其分解为质因数的乘法算式

分析：先建立个500000以内的素数表，然后从2开始一直判断是否为它的素数，如果是就将a=a/i继续判断i是否为a的素数，判断完成后输出这个素数因子和个数，用state判断是否输入过因子，输入过就要再前面输出“*”

```
#include <cstdio>
#include <vector>
using namespace std;
vector<int> prime(500000, 1);
int main() {
    for(int i = 2; i * i < 500000; i++)
        for(int j = 2; j * i < 500000; j++)
            prime[j * i] = 0;
    long int a;
    scanf("%ld", &a);
    printf("%ld=", a);
    if(a == 1) printf("1");
    bool state = false;
    for(int i = 2; a >= 2;i++) {
        int cnt = 0, flag = 0;
        while(prime[i] == 1 && a % i == 0) {
            cnt++;
            a = a / i;
            flag = 1;
        }
        if(flag) {
            if(state) printf("*");
            printf("%d", i);
            state = true;
        }
        if(cnt >= 2)
            printf("^%d", cnt);
    }
    return 0;
}
```

1060. Are They Equal (25) [科学计数法]

If a machine can save only 3 significant digits, the float numbers 12300 and 12358.9 are considered equal since they are both saved as 0.123*105 with simple chopping. Now given the number of significant digits on a machine and two float numbers, you are supposed to tell if they are treated equal in that machine.

Input Specification:

Each input file contains one test case which gives three numbers N, A and B, where N (<100) is the number of significant digits, and A and B are the two float numbers to be compared. Each float number is non-negative, no greater than 10100, and that its total digit number is less than 100.

Output Specification:

For each test case, print in a line “YES” if the two numbers are treated equal, and then the number in the standard form “0.d1…dN*10^k” (d1>0 unless the number is 0); or “NO” if they are not treated equal, and then the two numbers in their standard form. All the terms must be separated by a space, with no extra space at the end of a line.

Note: Simple chopping is assumed without rounding.

Sample Input 1:

3 12300 12358.9

Sample Output 1:

YES 0.123*10^5

Sample Input 2:

3 120 128

Sample Output 2:

NO 0.120*10^3 0.128*10^3

题目大意：给出两个数，问将它们写成保留N位小数的科学计数法后是否相等。如果相等，输出YES，输出他们的科学记数法表示方法。如果不相等输出NO，分别输出他们的科学计数法

分析：

1. cnta 和 cntb 通过扫描字符串得到小数点所在的下标（初始化cnta cntb为字符串长度，即下标为strlen(str)）
2. 考虑到可能前面有多余的零，用 p 和 q 通过扫描字符串使 p q 开始于第一个非0（且非小数点）处的下标
3. 如果cnta >= p ，说明小数点在第一个开始的非0数的下标的右边，那么科学计数法的指数为cnta – p ; 否则应该为cnta – p + 1; 字符串b同理。
4. 如果 p 和 q 等于字符串长度， 说明字符串是 0， 此时直接把 cnta（或者cntb）置为0，因为对于0来说乘以几次方都是相等的，如果不置为0可能会出现两个0比较导致判断为它们不相等
5. indexa = 0开始给新的A数组赋值，共赋值n位除去小数点外的正常数字，从p的下标开始。如果p大于等于strlen，说明字符串遍历完毕后依旧没能满足需要的位数，此时需要在A数组后面补上0直到满足n位数字。indexb同理，产生新的B数组
6. 判断A和B是否相等，且cnta和cntb是否相等。如果相等，说明他们用科学计数法表示后是相同的，输出YES，否则输出NO，同时输出正确的科学计数法

注意：

– 10的0次方和1次方都要写。

– 题目中说，无需四舍五入。

– 数组开大点，虽然只有100位，但是很有可能前面的0很多导致根本不止100位。一开始开的110，几乎没有AC的任何测试点。。后来开了10000就AC了~

```
#include <iostream>
#include <cstring>
```

```cpp
using namespace std;
int main() {
    int n, p = 0, q = 0;
    char a[10000], b[10000], A[10000], B[10000];
    scanf("%d%s%s", &n, a, b);
    int cnta = strlen(a), cntb = strlen(b);
    for(int i = 0; i < strlen(a); i++) {
        if(a[i] == '.') {
            cnta = i;
            break;
        }
    }
    for(int i = 0; i < strlen(b); i++) {
        if(b[i] == '.') {
            cntb = i;
            break;
        }
    }
    int indexa = 0, indexb = 0;
    while(a[p] == '0' || a[p] == '.') p++;
    while(b[q] == '0' || b[q] == '.') q++;
    if(cnta >= p)
        cnta = cnta - p;
    else
        cnta = cnta - p + 1;
    if(cntb >= q)
        cntb = cntb - q;
    else
        cntb = cntb - q + 1;
    if(p == strlen(a))
        cnta = 0;
    if(q == strlen(b))
        cntb = 0;
    while(indexa < n) {
        if(a[p] != '.' && p < strlen(a))
            A[indexa++] = a[p];
        else if(p >= strlen(a))
            A[indexa++] = '0';
        p++;
    }
    while(indexb < n) {
        if(b[q] != '.' && q < strlen(b))
            B[indexb++] = b[q];
        else if(q >= strlen(b))
            B[indexb++] = '0';
        q++;
    }
    if(strcmp(A, B) == 0 && cnta == cntb)
        printf("YES 0.%s*10^%d", A, cnta);
    else
        printf("NO 0.%s*10^%d 0.%s*10^%d" , A, cnta, B, cntb);
    return 0;
```

```
55    }
```

1061. Dating (20) [字符串处理]

Sherlock Holmes received a note with some strange strings: “Let’s date! 3485djDkxh4hhGE 2984akDfkkkkggEdsb s&hgsfdk d&Hyscvnm”. It took him only a minute to figure out that those strange strings are actually referring to the coded time “Thursday 14:04” — since the first common capital English letter (case sensitive) shared by the first two strings is the 4th capital letter ‘D’, representing the 4th day in a week; the second common character is the 5th capital letter ‘E’, representing the 14th hour (hence the hours from 0 to 23 in a day are represented by the numbers from 0 to 9 and the capital letters from A to N, respectively); and the English letter shared by the last two strings is ‘s’ at the 4th position, representing the 4th minute. Now given two pairs of strings, you are supposed to help Sherlock decode the dating time.

Input Specification:

Each input file contains one test case. Each case gives 4 non-empty strings of no more than 60 characters without white space in 4 lines.

Output Specification:

For each test case, print the decoded time in one line, in the format “DAY HH:MM”, where “DAY” is a 3-character abbreviation for the days in a week — that is, “MON” for Monday, “TUE” for Tuesday, “WED” for Wednesday, “THU” for Thursday, “FRI” for Friday, “SAT” for Saturday, and “SUN” for Sunday. It is guaranteed that the result is unique for each case.

Sample Input:

3485djDkxh4hhGE

2984akDfkkkkggEdsb

s&hgsfdk

d&Hyscvnm

Sample Output:

THU 14:04

题目大意：福尔摩斯接到一张奇怪的字条：“我们约会吧！3485djDkxh4hhGE 2984akDfkkkkggEdsb s&hgsfdk d&Hyscvnm”。大侦探很快就明白了，前面两字符串中第1对相同的大写英文字母（大小写有区分）是第4个字母D，代表星期四；第2对相同的字符是E，那是第5个英文字母，代表一天里的第14个钟头（于是一天的0点到23点由数字0到9、以及大写字母A到N表示）；后面两字符串第1对相同的英文字母s出现在第4个位置（从0开始计数）上，代表第4分钟。现给定两对字符串，要求解码得到约会的时间~

分析：按照题目所给的方法找到相等的字符后判断即可，如果输出的时间不足2位数要在前面添0，即用%02d输出～

```
1    #include <iostream>
2    #include <cctype>
3    using namespace std;
4    int main() {
```

```
    string a, b, c, d;
    cin >> a >> b >> c >> d;
    char t[2];
    int pos, i = 0, j = 0;
    while(i < a.length() && i < b.length()) {
        if (a[i] == b[i] && (a[i] >= 'A' && a[i] <= 'G')) {
            t[0] = a[i];
            break;
        }
        i++;
    }
    i = i + 1;
    while (i < a.length() && i < b.length()) {
        if (a[i] == b[i] && ((a[i] >= 'A' && a[i] <= 'N') || isdigit(a[i]))) {
            t[1] = a[i];
            break;
        }
        i++;
    }
    while (j < c.length() && j < d.length()) {
        if (c[j] == d[j] && isalpha(c[j])) {
            pos = j;
            break;
        }
        j++;
    }
    string week[7] = {"MON ", "TUE ", "WED ", "THU ", "FRI ", "SAT ", "SUN "};
    int m = isdigit(t[1]) ? t[1] - '0' : t[1] - 'A' + 10;
    cout << week[t[0]-'A'];
    printf("%02d:%02d", m, pos);
    return 0;
}
```

1062. Talent and Virtue (25) [排序]

About 900 years ago, a Chinese philosopher Sima Guang wrote a history book in which he talked about people's talent and virtue. According to his theory, a man being outstanding in both talent and virtue must be a "sage（圣人）"; being less excellent but with one's virtue outweighs talent can be called a "nobleman（君子）"; being good in neither is a "fool man（愚人）"; yet a fool man is better than a "small man（小人）" who prefers talent than virtue.

Now given the grades of talent and virtue of a group of people, you are supposed to rank them according to Sima Guang's theory.

Input Specification:

Each input file contains one test case. Each case first gives 3 positive integers in a line: N (<=105), the total number of people to be ranked; L (>=60), the lower bound of the qualified grades — that is, only the ones whose grades of talent and virtue are both not below this line will be ranked; and H (<100), the higher line of qualification — that is, those with both grades not below this line are considered as the "sages", and will be ranked in non-increasing order according to their total grades. Those with talent grades below H but virtue grades not are cosidered as the "noblemen", and are also ranked in non-increasing order

according to their total grades, but they are listed after the “sages”. Those with both grades below H, but with virtue not lower than talent are considered as the “fool men”. They are ranked in the same way but after the “noblemen”. The rest of people whose grades both pass the L line are ranked after the “fool men”.

Then N lines follow, each gives the information of a person in the format:

ID_Number Virtue_Grade Talent_Grade

where ID_Number is an 8-digit number, and both grades are integers in [0, 100]. All the numbers are separated by a space.

Output Specification:

The first line of output must give M (<=N), the total number of people that are actually ranked. Then M lines follow, each gives the information of a person in the same format as the input, according to the ranking rules. If there is a tie of the total grade, they must be ranked with respect to their virtue grades in non-increasing order. If there is still a tie, then output in increasing order of their ID’s.

Sample Input:

14 60 80

10000001 64 90

10000002 90 60

10000011 85 80

10000003 85 80

10000004 80 85

10000005 82 77

10000006 83 76

10000007 90 78

10000008 75 79

10000009 59 90

10000010 88 45

10000012 80 100

10000013 90 99

10000014 66 60

Sample Output:

12

10000013 90 99

10000012 80 100

10000003 85 80

10000011 85 80

10000004 80 85

10000007 90 78

10000006 83 76

10000005 82 77

10000002 90 60

10000014 66 60

10000008 75 79

10000001 64 90

题目大意：输入第1行给出3个正整数，分别为：N为考生总数；L为录取最低分数线，即德分和才分均不低于L的考生才有资格被考虑录取；H为优先录取线——德分和才分均不低于此线的被定义为“才德全尽”，此类考生按德才总分从高到低排序；才分不到但德分到线的一类考生属于“德胜才”，也按总分排序，但排在第一类考生之后；德才分均低于H，但是德分不低于才分的考生属于“才德兼亡”但尚有“德胜才”者，按总分排序，但排在第二类考生之后；其他达到最低线L的考生也按总分排序，但排在第三类考生之后。输出第1行首先给出达到最低分数线的考生人数M，随后M行，每行按照输入格式输出一位考生的信息，考生按输入中说明的规则从高到低排序。当某类考生中有多人总分相同时，按其德分降序排列；若德分也并列，则按准考证号的升序输出。

分析：用结构体存储。写好cmp函数～结构体数组vector v[4]中v[0]保存第一类考生，v[1]保存第二类考生……以此类推～写好cmp函数很重要，cmp函数中，排序先按照总分排序，然后按照德分排序，最后按照才分排序……最后输出符合条件的结果～

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
struct node {
    int num, de, cai;
};
int cmp(struct node a, struct node b) {
    if ((a.de + a.cai) != (b.de + b.cai))
        return (a.de + a.cai) > (b.de + b.cai);
    else if (a.de != b.de)
        return a.de > b.de;
    else
        return a.num < b.num;
}
int main() {
    int n, low, high;
    scanf("%d %d %d", &n, &low, &high);
    vector<node> v[4];
    node temp;
    int total = n;
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d %d %d", &temp.num, &temp.de, &temp.cai);
        if (temp.de < low || temp.cai < low)
            total--;
        else if (temp.de >= high && temp.cai >= high)
            v[0].push_back(temp);
        else if (temp.de >= high && temp.cai < high)
            v[1].push_back(temp);
        else if (temp.de < high && temp.cai < high && temp.de >= temp.cai)
            v[2].push_back(temp);
        else
            v[3].push_back(temp);
    }
    printf("%d\n", total);
    for (int i = 0; i < 4; i++) {
        sort(v[i].begin(), v[i].end(), cmp);
        for (int j = 0; j < v[i].size(); j++)
            printf("%d %d %d\n", v[i][j].num, v[i][j].de, v[i][j].cai);
    }
    return 0;
}
```

1063. Set Similarity (25) [集合set，STL的使用]

Given two sets of integers, the similarity of the sets is defined to be Nc/Nt*100%, where Nc is the number of distinct common numbers shared by the two sets, and Nt is the total number of distinct numbers in the two sets. Your job is to calculate the similarity of any given pair of sets.

Input Specification:

Each input file contains one test case. Each case first gives a positive integer N (<=50) which is the total number of sets. Then N lines follow, each gives a set with a positive M (<=104) and followed by M integers in the range [0, 109]. After the input of sets, a positive integer K (<=2000) is given, followed by K lines of queries. Each query gives a pair of set numbers (the sets are numbered from 1 to N). All the numbers in a line are separated by a space.

Output Specification:

For each query, print in one line the similarity of the sets, in the percentage form accurate up to 1 decimal place.

Sample Input:

3

3 99 87 101

4 87 101 5 87

7 99 101 18 5 135 18 99

2

1 2

1 3

Sample Output:

50.0%

33.3%

题目大意：给定两个整数集合，它们的相似度定义为：Nc/Nt*100%。其中Nc是两个集合都有的不相等整数的个数，Nt是两个集合一共有的不相等整数的个数。你的任务就是计算任意一对给定集合的相似度。

nc是两个集合的公共元素个数，nt是两个集合的所有包含的元素个数（其中元素个数表示各个元素之间互不相同）

分析：因为给出的集合里面含有重复的元素，而计算nc和nt不需要考虑两个集合里面是否分别有重复的元素，所以可以直接使用set存储每一个集合，然后把set放进一个数组里面存储。当需要计算集合a和集合b的相似度nc和nt的时候，遍历集合a中的每一个元素，寻找集合b中是否有该元素，如果有，说明是两个人公共的集合元素，则nc++，否则nt++（nt的初值为b集合里面本有的元素）

```
#include <cstdio>
#include <vector>
#include <set>
using namespace std;
int main() {
    int n, m, k, temp, a, b;
    scanf("%d", &n);
    vector<set<int>> v(n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &m);
        set<int> s;
        for(int j = 0; j < m; j++) {
            scanf("%d", &temp);
            s.insert(temp);
        }
        v[i] = s;
    }
    scanf("%d", &k);
    for(int i = 0; i < k; i++) {
        scanf("%d %d", &a, &b);
        int nc = 0, nt = v[b-1].size();
        for(auto it = v[a-1].begin(); it != v[a-1].end(); it++) {
            if(v[b-1].find(*it) == v[b-1].end())
                nt++;
            else
                nc++;
        }
        double ans = (double)nc / nt * 100;
        printf("%.1f%%\n", ans);
    }
    return 0;
}
```

1064. Complete Binary Search Tree (30) [二叉查找树BST]

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

The left subtree of a node contains only nodes with keys less than the node's key.

The right subtree of a node contains only nodes with keys greater than or equal to the node's key.

Both the left and right subtrees must also be binary search trees.

A Complete Binary Tree (CBT) is a tree that is completely filled, with the possible exception of the bottom level, which is filled from left to right.

Now given a sequence of distinct non-negative integer keys, a unique BST can be constructed if it is required that the tree must also be a CBT. You are supposed to output the level order traversal sequence of this BST.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<=1000). Then N distinct non-negative integer keys are given in the next line. All the numbers in a line are separated by a space and are no greater than 2000.

Output Specification:

For each test case, print in one line the level order traversal sequence of the corresponding complete binary search tree. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

Sample Input:

10

1 2 3 4 5 6 7 8 9 0

Sample Output:

6 3 8 1 5 7 9 0 2 4

题目大意：给一串构成树的序列，已知该树是完全二叉搜索树，求它的层序遍历的序列

分析：总得概括来说，已知中序，可求root下标，可以求出层序。

1. 因为二叉搜索树的中序满足：是一组序列的从小到大排列，所以只需排序所给序列即可得到中序
2. 因为根据完全二叉树的结点数，可以求出它的根结点在中序中对应的下标
3. 如此，已知了中序，又可以根据结点数求出根结点的下标，就可以递归求出左右子树的根结点的下标
4. i结点的左孩子为2 * i + 1，右孩子2 * i + 2，就可以根据结点下标和中序数组赋值level数组
5. 最后输出所有结点的层序数组level

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
```

```
vector<int> in, level;
void levelorder(int start, int end, int index) {
    if(start > end) return ;
    int n = end - start + 1;
    int l = log(n + 1) / log(2); // 除了最后一层的层数
    int leave = n - (pow(2, l) - 1);// 最后一层的叶子节点数
    int root = start + (pow(2, l - 1) - 1) + min((int)pow(2, l - 1), leave); // pow(2, l - 1) - 1是除了root结点所在层和最后一层外，左子树的结点个数，pow(2, l - 1) 是l+1层最多拥有的属于根结点左子树的结点个数，min(pow(2, l - 1), leave)是最后一个结点真正拥有的属于根结点左子树上的结点个数
    level[index] = in[root];
    levelorder(start, root - 1, 2 * index + 1);
    levelorder(root + 1, end, 2 * index + 2);
}
int main() {
    int n;
    scanf("%d", &n);
    in.resize(n);
    level.resize(n);
    for(int i = 0 ; i < n; i++)
        scanf("%d", &in[i]);
    sort(in.begin(), in.end());
    levelorder(0, n - 1, 0);
    printf("%d", level[0]);
    for(int i = 1; i < n; i++)
        printf(" %d", level[i]);
    return 0;
}
```

1065. A+B and C (64bit) (20) [模拟]

Given three integers A, B and C in [-263, 263], you are supposed to tell whether A+B > C.

Input Specification:

The first line of the input gives the positive number of test cases, T (<=10). Then T test cases follow, each consists of a single line containing three integers A, B and C, separated by single spaces.

Output Specification:

For each test case, output in one line “Case #X: true” if A+B>C, or “Case #X: false” otherwise, where X is the case number (starting from 1).”

Sample Input:

3

1 2 3

2 3 4

9223372036854775807 -9223372036854775808 0

Sample Output:

Case #1: false

Case #2: true

Case #3: false

分析：

因为A、B的大小为[-2^63, 2^63]，用long long 存储A和B的值，以及他们相加的值sum：

如果A > 0, B < 0 或者 A < 0, B > 0，sum是不可能溢出的

如果A > 0, B > 0，sum可能会溢出，sum范围理应为(0, 2^64 – 2]，溢出得到的结果应该是[-2^63, -2]是个负数，所以sum < 0时候说明溢出了

如果A < 0, B < 0，sum可能会溢出，同理，sum溢出后结果是大于0的，所以sum > 0 说明溢出了

```cpp
#include <cstdio>
using namespace std;
int main() {
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        long long a, b, c;
        scanf("%lld %lld %lld", &a, &b, &c);
        long long sum = a + b;
        if(a > 0 && b > 0 && sum < 0) {
            printf("Case #%d: true\n", i + 1);
        } else if(a < 0 && b < 0 && sum >= 0){
            printf("Case #%d: false\n", i + 1);
        } else if(sum > c) {
            printf("Case #%d: true\n", i + 1);
        } else {
            printf("Case #%d: false\n", i + 1);
        }
    }
    return 0;
}
```

1066. Root of AVL Tree (25) [平衡二叉树（AVL树）]

An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Figures 1-4 illustrate the rotation rules.

Now given a sequence of insertions, you are supposed to tell the root of the resulting AVL tree.

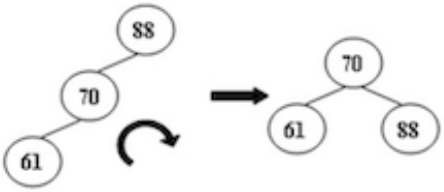


Figure 1: Insert 88, 70, 61

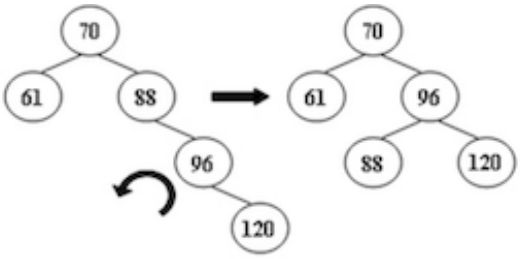


Figure 2: Insert 96, 120

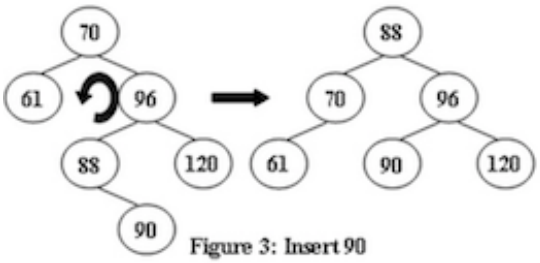


Figure 3: Insert 90

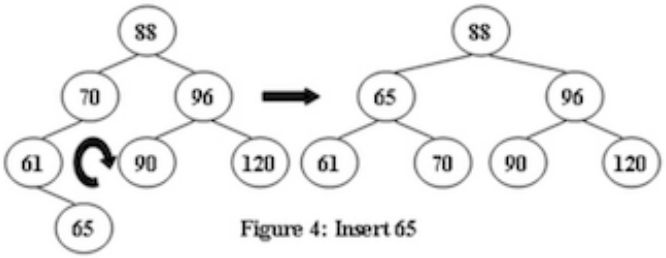


Figure 4: Insert 65

题目大意：AVL树是自平衡二叉搜索树。 在AVL树中，任何节点的两个子子树的高度最多相差一个；如果在任何时候它们相差多于一个，则重新平衡以恢复此属性。 图1-4说明了旋转规则～现在给出一系列插入，要求输出根节点的值～

分析：写出建AVL（平衡二叉搜索树）的代码模版即可，rotateLeft表示左旋，rotateRight表示右旋，rotateLeftRight表示先左旋后右旋，rotateRightLeft表示先右旋后左旋，getHeight表示获取传入结点的子树的高度，insert表示插入建树的过程，如果root为空，直接新建结点插入即可～如果当前要插入的值小于root->val，则插入root的左子树；如果当前要插入的值大于root->val，则插入root的右子树～如果插入后左右子树高度差大于1，再根据值的大小比较进行旋转调整使树平衡～插入完成后返回root指针赋值给main函数里的root～最后输出root的val值～

```
#include <iostream>
using namespace std;
struct node {
    int val;
    struct node *left, *right;
};
node *rotateLeft(node *root) {
    node *t = root->right;
    root->right = t->left;
    t->left = root;
    return t;
}
node *rotateRight(node *root) {
    node *t = root->left;
    root->left = t->right;
    t->right = root;
    return t;
}
node *rotateLeftRight(node *root) {
    root->left = rotateLeft(root->left);
    return rotateRight(root);
}
node *rotateRightLeft(node *root) {
    root->right = rotateRight(root->right);
    return rotateLeft(root);
}
```

```
int getHeight(node *root) {
    if(root == NULL) return 0;
    return max(getHeight(root->left), getHeight(root->right)) + 1;
}
node *insert(node *root, int val) {
    if(root == NULL) {
        root = new node();
        root->val = val;
        root->left = root->right = NULL;
    } else if(val < root->val) {
        root->left = insert(root->left, val);
        if(getHeight(root->left) - getHeight(root->right) == 2)
            root = val < root->left->val ? rotateRight(root) :
rotateLeftRight(root);
    } else {
        root->right = insert(root->right, val);
        if(getHeight(root->left) - getHeight(root->right) == -2)
            root = val > root->right->val ? rotateLeft(root) :
rotateRightLeft(root);
    }
    return root;
}
int main() {
    int n, val;
    scanf("%d", &n);
    node *root = NULL;
    for(int i = 0; i < n; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }
    printf("%d", root->val);
    return 0;
}
```

1067. Sort with Swap(0,*) (25) [贪心算法]

Given any permutation of the numbers {0, 1, 2,…, N-1}, it is easy to sort them in increasing order. But what if Swap(0, *) is the ONLY operation that is allowed to use? For example, to sort {4, 0, 2, 1, 3} we may apply the swap operations in the following way:

Swap(0, 1) => {4, 1, 2, 0, 3}

Swap(0, 3) => {4, 1, 2, 3, 0}

Swap(0, 4) => {0, 1, 2, 3, 4}

Now you are asked to find the minimum number of swaps need to sort the given permutation of the first N nonnegative integers.

Input Specification:

Each input file contains one test case, which gives a positive N (<=105) followed by a permutation sequence of {0, 1, …, N-1}. All the numbers in a line are separated by a space.

Output Specification:

For each case, simply print in a line the minimum number of swaps need to sort the given permutation.

Sample Input:

10 3 5 7 2 6 4 9 0 8 1

Sample Output:

9

题目大意：给出一个n个数的序列，数字为0~n-1的乱序，每次用两两交换的方式而且只能用0和另一个数交换，使序列变成有序的，问最少需要多少步骤。

分析：1.0号为哨兵， 用用哨兵与其他数字交换，使其他数字回到有序的位置（最后有序时所处的位置），则排序完成

2.a[t] = i; 表示t数字当前正在占着哪一个位置。 （如果想实时监测每个数字的位置，可以用 b 数组 {b[a[i]] = i } 缓存一下数据，输出查看的）

3.依次遍历每个位置i，如果当前位置不是与之对应的数字，那么我们让这哨兵来去该数执行操作回到正确位置

4.数字如何被哨兵执行操作回到序的位置：

如果哨兵此时不在自己有序的位置上，那就，先让哨兵去让他占的那个位置上的真正应该放的数字回到此位置，即交换哨兵和此数字，我们swap(a[0],a[a0])，直到哨兵在交换的过程中回到了自己的有序位置。字词再次检查该位置是不是应该放的数字（别的数字回到有序位置的时候即哨兵执行操作的过程中，有可能让此位置该有的数字回到位置）。如果此位置不是当前数字，那哨兵和他交换 swap(a[0],a[i])，就是让他先去哨兵的有序位置待一会，等下一轮操作，哨兵会把他交换回来的。如果此位置已经是该数字了，那就什么都不做。

5.当到达最后一个位置时候，两种情况 1）。如果第一个数字和最后一个数字都在自己的有序位置 那ok~ 2）.哨兵和最后一个数字互相占着对方的位置，那最后一个数字就是哨兵，交换一次后，哨兵在交换后的位置等待，就是已经回到自己的有序位置，此时排序也是完成的。此过程包括在4里面了，怕你们不理解，单独说一下～

```
#include <iostream>
using namespace std;
int main() {
    int n, t, cnt = 0, a[100010];
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> t;
        a[t] = i;
    }
    for(int i = 1; i < n; i++) {
        if(i != a[i]) {
            while(a[0] != 0) {
                swap(a[0],a[a[0]]);
                cnt++;
            }
            if(i != a[i]) {
```

```
                swap(a[0],a[i]);
                cnt++;
            }
        }
    }
    cout << cnt;
    return 0;
}
```

1068. Find More Coins (30) [01背包，动态规划]

Eva loves to collect coins from all over the universe, including some other planets like Mars. One day she visited a universal shopping mall which could accept all kinds of coins as payments. However, there was a special requirement of the payment: for each bill, she must pay the exact amount. Since she has as many as 104 coins with her, she definitely needs your help. You are supposed to tell her, for any given amount of money, whether or not she can find some coins to pay for it.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive numbers: N (<=104, the total number of coins) and M(<=102, the amount of money Eva has to pay). The second line contains N face values of the coins, which are all positive numbers. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the face values V1 <= V2 <= … <= Vk such that V1 + V2 + … + Vk = M. All the numbers must be separated by a space, and there must be no extra space at the end of the line. If such a solution is not unique, output the smallest sequence. If there is no solution, output “No Solution” instead.

Note: sequence {A[1], A[2], …} is said to be “smaller” than sequence {B[1], B[2], …} if there exists k >= 1 such that A[i]=B[i] for all i < k, and A[k] < B[k].

Sample Input 1:

8 9

5 9 8 7 2 3 4 1

Sample Output 1:

1 3 5

Sample Input 2:

4 8

7 2 4 3

Sample Output 2:

No Solution

题目大意：用n个硬币买价值为m的东西，输出使用方案，使得正好几个硬币加起来价值为m。从小到大排列，输出最小的那个排列方案

分析：01背包问题，因为要输出从小到大的排列，可以先把硬币面额从大到小排列，然后用bool类型的choice[i][j]数组dp[i][j]是否选取，如果选取了就令choice为true；然后进行01背包问题求解，如果最后求解的结果不是恰好等于所需要的价值的，就输出No Soultion，否则从choice[i][j]判断选取的情况，i从n到1表示从后往前看第i个物品的选取情况，j从m到0表示从容量m到0是否选取(j = j – w[i])，把选取情况压入arr数组中，最后输出arr数组

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int dp[10010], w[10010];
bool choice[10010][110];
int cmp1(int a, int b){return a > b;}
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++)
        scanf("%d", &w[i]);
    sort(w + 1, w + n + 1, cmp1);
    for(int i = 1; i <= n; i++) {
        for(int j = m; j >= w[i]; j--) {
            if(dp[j] <= dp[j-w[i]] + w[i]) {
                choice[i][j] = true;
                dp[j] = dp[j-w[i]] + w[i];
            }
        }
    }
    if(dp[m] != m) printf("No Solution");
    else {
        vector<int> arr;
        int v = m, index = n;
        while(v > 0) {
            if(choice[index][v] == true) {
                arr.push_back(w[index]);
                v -= w[index];
            }
            index--;
        }
        for(int i = 0; i < arr.size(); i++) {
            if(i != 0) printf(" ");
            printf("%d", arr[i]);
        }
    }
    return 0;
}
```

1069. The Black Hole of Numbers (20) [数学问题]

For any 4-digit integer except the ones with all the digits being the same, if we sort the digits in non-increasing order first, and then in non-decreasing order, a new number can be obtained by taking the second number from the first one. Repeat in this manner we will soon end up at the number 6174 — the “black hole” of 4-digit numbers. This number is named Kaprekar Constant.

For example, start from 6767, we’ll get:

7766 – 6677 = 1089

9810 – 0189 = 9621

9621 – 1269 = 8352

8532 – 2358 = 6174

7641 – 1467 = 6174

… …

Given any 4-digit number, you are supposed to illustrate the way it gets into the black hole.

Input Specification:

Each input file contains one test case which gives a positive integer N in the range (0, 10000).

Output Specification:

If all the 4 digits of N are the same, print in one line the equation “N – N = 0000”. Else print each step of calculation in a line until 6174 comes out as the difference. All the numbers must be printed as 4-digit numbers.

Sample Input 1:

6767

Sample Output 1:

7766 – 6677 = 1089

9810 – 0189 = 9621

9621 – 1269 = 8352

8532 – 2358 = 6174

Sample Input 2:

2222

Sample Output 2:

2222 – 2222 = 0000

分析：有一个测试用例注意点，如果当输入N值为6174的时候，依旧要进行下面的步骤，直到差值为6174才可以～所以用do while语句，无论是什么值总是要执行一遍while语句，直到遇到差值是0000或6174～

s.insert(0, 4 – s.length(), ‘0’);用来给不足4位的时候前面补0～

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
bool cmp(char a, char b) {return a > b;}
int main() {
    string s;
    cin >> s;
    s.insert(0, 4 - s.length(), '0');
    do {
        string a = s, b = s;
        sort(a.begin(), a.end(), cmp);
        sort(b.begin(), b.end());
        int result = stoi(a) - stoi(b);
        s = to_string(result);
        s.insert(0, 4 - s.length(), '0');
        cout << a << " - " << b << " = " << s << endl;
    } while (s != "6174" && s != "0000");
    return 0;
}
```

1070. Mooncake (25) [贪心算法]

Mooncake is a Chinese bakery product traditionally eaten during the Mid-Autumn Festival. Many types of fillings and crusts can be found in traditional mooncakes according to the regions culture. Now given the inventory amounts and the prices of all kinds of the mooncakes, together with the maximum total demand of the market, you are supposed to tell the maximum profit that can be made.

Note: partial inventory storage can be taken. The sample shows the following situation: given three kinds of mooncakes with inventory amounts being 180, 150, and 100 thousand tons, and the prices being 7.5, 7.2, and 4.5 billion yuans. If the market demand can be at most 200 thousand tons, the best we can do is to sell 150 thousand tons of the second kind of mooncake, and 50 thousand tons of the third kind. Hence the total profit is 7.2 + 4.5/2 = 9.45 (billion yuans).

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers N (<=1000), the number of different kinds of mooncakes, and D (<=500 thousand tons), the maximum total demand of the market. Then the second line gives the positive inventory amounts (in thousand tons), and the third line gives the positive prices (in billion yuans) of N kinds of mooncakes. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the maximum profit (in billion yuans) in one line, accurate up to 2 decimal places.

Sample Input:

3 200

180 150 100

7.5 7.2 4.5

Sample Output:

9.45

题目大意：N表示月饼种类，D表示月饼的市场最大需求量，给出每种月饼的数量和总价，问根据市场最大需求量，这些月饼的最大销售利润为多少～

分析：首先根据月饼的总价和数量计算出每一种月饼的单价，然后将月饼数组按照单价从大到小排序，根据需求量need的大小，从单价最大的月饼开始售卖，将销售掉这种月饼的价格累加到result中，最后输出result即可～

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
struct mooncake{
    float mount, price, unit;
};
int cmp(mooncake a, mooncake b) {
    return a.unit > b.unit;
}
int main() {
    int n, need;
    cin >> n >> need;
    vector<mooncake> a(n);
    for (int i = 0; i < n; i++) scanf("%f", &a[i].mount);
    for (int i = 0; i < n; i++) scanf("%f", &a[i].price);
    for (int i = 0; i < n; i++) a[i].unit = a[i].price / a[i].mount;
    sort(a.begin(), a.end(), cmp);
    float result = 0.0;
    for (int i = 0; i < n; i++) {
        if (a[i].mount <= need) {
            result = result + a[i].price;
        } else {
            result = result + a[i].unit * need;
            break;
        }
        need = need - a[i].mount;
    }
    printf("%.2f",result);
    return 0;
}
```

1071. Speech Patterns (25) [map映射，STL的使用]

People often have a preference among synonyms of the same word. For example, some may prefer “the police”, while others may prefer “the cops”. Analyzing such patterns can help to narrow down a speaker’s identity, which is useful when validating, for example, whether it’s still the same person behind an online avatar.

Now given a paragraph of text sampled from someone’s speech, can you find the person’s most commonly used word?

Input Specification:

Each input file contains one test case. For each case, there is one line of text no more than 1048576 characters in length, terminated by a carriage return '\n'. The input contains at least one alphanumerical character, i.e., one character from the set [0-9 A-Z a-z].

Output Specification:

For each test case, print in one line the most commonly occurring word in the input text, followed by a space and the number of times it has occurred in the input. If there are more than one such words, print the lexicographically smallest one. The word should be printed in all lower case. Here a "word" is defined as a continuous sequence of alphanumerical characters separated by non-alphanumerical characters or the line beginning/end.

Note that words are case insensitive.

Sample Input:

Can1: "Can a can can a can? It can!"

Sample Output:

can 5

题目大意：统计单词个数～大小写字母+数字的组合才是合法的单词，给出一个字符串，求出现的合法的单词的个数最多的那个单词，以及它出现的次数。如果有并列的，那么输出字典序里面的第一个～～

分析：用map很简单的~不过呢~有几个注意点～：

1. 大小写不区分，所以统计之前要先s[i] = tolower(s[i]);
2. [0-9 A-Z a-z]可以简写为cctype头文件里面的一个函数isalnum～～
3. 必须用getline读入一长串的带空格的字符串～～
4. 一定要当t不为空的时候m[t]++，因为t为空也会被统计的！！！～～
5. 最重要的是～如果i已经到了最后一位，不管当前位是不是字母数字，都得将当前这个t放到map里面（只要t长度不为0）～

```
#include <iostream>
#include <map>
#include <cctype>
using namespace std;
int main() {
    string s, t;
    getline(cin, s);
    map<string, int> m;
    for(int i = 0; i < s.length(); i++) {
        if(isalnum(s[i])) {
            s[i] = tolower(s[i]);
            t += s[i];
        }
        if(!isalnum(s[i]) || i == s.length() - 1){
            if(t.length() != 0) m[t]++;
            t = "";
        }
```

```
        }
        int maxn = 0;
        for(auto it = m.begin(); it != m.end(); it++) {
            if(it->second > maxn) {
                t = it->first;
                maxn = it->second;
            }
        }
        cout << t << " " << maxn;
        return 0;
    }
```

1072. Gas Station (30) [Dijkstra算法]

A gas station has to be built at such a location that the minimum distance between the station and any of the residential housing is as far away as possible. However it must guarantee that all the houses are in its service range.

Now given the map of the city and several candidate locations for the gas station, you are supposed to give the best recommendation. If there are more than one solution, output the one with the smallest average distance to all the houses. If such a solution is still not unique, output the one with the smallest index number.

Input Specification:

Each input file contains one test case. For each case, the first line contains 4 positive integers: N (<= 103), the total number of houses; M (<= 10), the total number of the candidate locations for the gas stations; K (<= 104), the number of roads connecting the houses and the gas stations; and DS, the maximum service range of the gas station. It is hence assumed that all the houses are numbered from 1 to N, and all the candidate locations are numbered from G1 to GM.

Then K lines follow, each describes a road in the format

P1 P2 Dist

where P1 and P2 are the two ends of a road which can be either house numbers or gas station numbers, and Dist is the integer length of the road.

Output Specification:

For each test case, print in the first line the index number of the best location. In the next line, print the minimum and the average distances between the solution and all the houses. The numbers in a line must be separated by a space and be accurate up to 1 decimal place. If the solution does not exist, simply output “No Solution”.

Sample Input 1:

4 3 11 5

1 2 2

1 4 2

1 G1 4

1 G2 3

2 3 2

2 G2 1

3 4 2

3 G3 2

4 G1 3

G2 G1 1

G3 G2 2

Sample Output 1:

G1

2.0 3.3

Sample Input 2:

2 1 2 10

1 G1 9

2 G1 20

Sample Output 2:

No Solution

题目大意：从m个加油站里面选取1个站点，让他离居民区的最近的人最远，并且没有超出服务范围ds之内。如果有很多个最远的加油站，输出距离所有居民区距离平均值最小的那个。如果平均值还是一样，就输出按照顺序排列加油站编号最小的那个

分析：

因为加油站之间也是彼此有路连接的，所以最短路径计算的时候也要把加油站算上。所以我们就是对n+m个点进行Dijkstra计算最短路径。要求计算出1~m号加油站距离其他站点的最短路径。这时候可以遍历dis数组，如果dis存在一个距离大于服务范围ds的距离，那么我们就舍弃这个加油站。取最最短的路径，这就是距离它最近的加油站mindis。如果mindis > ansdis，就是说找到了一个距离居民最小距离的加油站是更远的，那就选这个加油站，更新ansid为它的id。最后输出

对于加油站的字符串编号的处理：如果最近居民区最大的值没有变化但是找到了一个更小的平均距离，那就选这个。我们可以根据输入的是G还是数字，如果是数字就令编号为他自己，如果是G开头的，编号设为n+G后面的数字。 Update：Github用户littlesevenmo在issue中提出需要添加输入判断，题目中并没有说明两点之间最多只有一条路。也就是说，有可能两点之间有多条路，因此需要添加判断，只存储距离最短的路。另外，也有可能会出现 G1 G1 9999这样的测试数据，因此，自身与自身之间的距离要初始化为0。完善后的代码如下：

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;
```

```cpp
const int inf = 999999999;
int n, m, k, ds, station;
int e[1020][1020], dis[1020];
bool visit[1020];
int main() {
    fill(e[0], e[0] + 1020 * 1020, inf);
    fill(dis, dis + 1020, inf);
    scanf("%d%d%d%d", &n, &m, &k, &ds);
    for(int i = 0; i < k; i++) {
        int tempdis;
        string s, t;
        cin >> s >> t >> tempdis;
        int a, b;
        if(s[0] == 'G') {
            s = s.substr(1);
            a = n + stoi(s);
        } else {
            a = stoi(s);
        }
        if(t[0] == 'G') {
            t = t.substr(1);
            b = n + stoi(t);
        } else {
            b = stoi(t);
        }
        e[a][b] = e[b][a] = tempdis;
    }
    int ansid = -1;
    double ansdis = -1, ansaver = inf;
    for(int index = n + 1; index <= n + m; index++) {
        double mindis = inf, aver = 0;
        fill(dis, dis + 1020, inf);
        fill(visit, visit + 1020, false);
        dis[index] = 0;
        for(int i = 0; i < n + m; i++) {
            int u = -1, minn = inf;
            for(int j = 1; j <= n + m; j++) {
                if(visit[j] == false && dis[j] < minn) {
                    u = j;
                    minn = dis[j];
                }
            }
            if(u == -1) break;
            visit[u] = true;
            for(int v = 1; v <= n + m; v++) {
                if(visit[v] == false && dis[v] > dis[u] + e[u][v])
                    dis[v] = dis[u] + e[u][v];
            }
        }
        for(int i = 1; i <= n; i++) {
            if(dis[i] > ds) {
                mindis = -1;
```

```
                break;
            }
            if(dis[i] < mindis) mindis = dis[i];
            aver += 1.0 * dis[i];
        }
        if(mindis == -1) continue;
        aver = aver / n;
        if(mindis > ansdis) {
            ansid = index;
            ansdis = mindis;
            ansaver = aver;
        } else if(mindis == ansdis && aver < ansaver) {
            ansid = index;
            ansaver = aver;
        }
    }
    if(ansid == -1)
        printf("No Solution");
    else
        printf("G%d\n%.1f %.1f", ansid - n, ansdis, ansaver);
    return 0;
}
```

1073. Scientific Notation (20) [字符串处理]

Scientific notation is the way that scientists easily handle very large numbers or very small numbers. The notation matches the regular expression [+-][1-9]”.”[0-9]+E[+-][0-9]+ which means that the integer portion has exactly one digit, there is at least one digit in the fractional portion, and the number and its exponent’s signs are always provided even when they are positive.

Now given a real number A in scientific notation, you are supposed to print A in the conventional notation while keeping all the significant figures.

Input Specification:

Each input file contains one test case. For each case, there is one line containing the real number A in scientific notation. The number is no more than 9999 bytes in length and the exponent’s absolute value is no more than 9999.

Output Specification:

For each test case, print in one line the input number A in the conventional notation, with all the significant figures kept, including trailing zeros,

Sample Input 1:

+1.23400E-03

Sample Output 1:

0.00123400

Sample Input 2:

-1.2E+10

Sample Output 2:

-12000000000

题目大意：题目给出科学计数法的格式的数字A，要求输出普通数字表示法的A，并保证所有有效位都被保留，包括末尾的0

分析：n保存E后面的字符串所对应的数字，t保存E前面的字符串，不包括符号位。当n<0时表示向前移动，那么先输出0. 然后输出abs(n)-1个0，然后继续输出t中的所有数字；当n>0时候表示向后移动，那么先输出第一个字符，然后将t中尽可能输出n个字符，如果t已经输出到最后一个字符(j == t.length())那么就在后面补n-cnt个0，否则就补充一个小数点。 然后继续输出t剩余的没有输出的字符～

```
#include <iostream>
using namespace std;
int main() {
    string s;
    cin >> s;
    int i = 0;
    while (s[i] != 'E') i++;
    string t = s.substr(1, i-1);
    int n = stoi(s.substr(i+1));
    if (s[0] == '-') cout << "-";
    if (n < 0) {
        cout << "0.";
        for (int j = 0; j < abs(n) - 1; j++) cout << '0';
        for (int j = 0; j < t.length(); j++)
            if (t[j] != '.') cout << t[j];
    } else {
        cout << t[0];
        int cnt, j;
        for (j = 2, cnt = 0; j < t.length() && cnt < n; j++, cnt++) cout <<
t[j];
        if (j == t.length()) {
            for (int k = 0; k < n - cnt; k++) cout << '0';
        } else {
            cout << '.';
            for (int k = j; k < t.length(); k++) cout << t[k];
        }
    }
    return 0;
}
```

1074. Reversing Linked List (25) [链表]

Given a constant K and a singly linked list L, you are supposed to reverse the links of every K elements on L. For example, given L being 1→2→3→4→5→6, if K = 3, then you must output 3→2→1→6→5→4; if K = 4, you must output 4→3→2→1→5→6.

Input Specification:

Each input file contains one test case. For each case, the first line contains the address of the first node, a positive N (<= 105) which is the total number of nodes, and a positive K (<=N) which is the length of the sublist to be reversed. The address of a node is a 5-digit nonnegative integer, and NULL is represented by -1.

Then N lines follow, each describes a node in the format:

Address Data Next

where Address is the position of the node, Data is an integer, and Next is the position of the next node.

Output Specification:

For each case, output the resulting ordered linked list. Each node occupies a line, and is printed in the same format as in the input.

Sample Input:

00100 6 4 00000 4 99999 00100 1 12309 68237 6 -1 33218 3 00000 99999 5 68237 12309 2 33218

Sample Output:

00000 4 33218 33218 3 12309 12309 2 00100 00100 1 99999 99999 5 68237 68237 6 -1

分析：把地址为temp的数的数值存入data[temp]中，把temp的下一个结点的地址存入next[temp]中。

注意：不一定所有的输入的结点都是有用的，加个计数器sum

```
#include <iostream>
using namespace std;
int main() {
    int first, k, n, sum = 0;
    cin >> first >> n >> k;
    int temp, data[100005], next[100005], list[100005], result[100005];
    for (int i = 0; i < n; i++) {
        cin >> temp;
        cin >> data[temp] >> next[temp];
    }
    while (first != -1) {
        list[sum++] = first;
        first = next[first];
    }
    for (int i = 0; i < sum; i++) result[i] = list[i];
    for (int i = 0; i < (sum - sum % k); i++)
        result[i] = list[i / k * k + k - 1 - i % k];
    for (int i = 0; i < sum - 1; i++)
        printf("%05d %d %05d\n", result[i], data[result[i]], result[i + 1]);
    printf("%05d %d -1", result[sum - 1], data[result[sum - 1]]);
    return 0;
}
```

1075. PAT Judge (25) [排序]

The ranklist of PAT is generated from the status list, which shows the scores of the submittions. This time you are supposed to generate the ranklist for PAT.

Input Specification:

Each input file contains one test case. For each case, the first line contains 3 positive integers, N (<=104), the total number of users, K (<=5), the total number of problems, and M (<=105), the total number of submittions. It is then assumed that the user id's are 5-digit numbers from 00001 to N, and the problem id's are from 1 to K. The next line contains K positive integers p[i] (i=1, …, K), where p[i] corresponds to the full mark of the i-th problem. Then M lines follow, each gives the information of a submittion in the following format:

user_id problem_id partial_score_obtained

where partial_score_obtained is either -1 if the submittion cannot even pass the compiler, or is an integer in the range [0, p[problem_id]]. All the numbers in a line are separated by a space.

Output Specification:

For each test case, you are supposed to output the ranklist in the following format:

rank user_id total_score s[1] … s[K]

where rank is calculated according to the total_score, and all the users with the same total_score obtain the same rank; and s[i] is the partial score obtained for the i-th problem. If a user has never submitted a solution for a problem, then "-" must be printed at the corresponding position. If a user has submitted several solutions to solve one problem, then the highest score will be counted.

The ranklist must be printed in non-decreasing order of the ranks. For those who have the same rank, users must be sorted in nonincreasing order according to the number of perfectly solved problems. And if there is still a tie, then they must be printed in increasing order of their id's. For those who has never submitted any solution that can pass the compiler, or has never submitted any solution, they must NOT be shown on the ranklist. It is guaranteed that at least one user can be shown on the ranklist.

Sample Input:

7 4 20

20 25 25 30

00002 2 12

00007 4 17

00005 1 19

00007 2 25

00005 1 20

00002 2 2

00005 1 15

00001 1 18

00004 3 25

00002 2 25

00005 3 22

00006 4 -1

00001 2 18

00002 1 20

00004 1 15

00002 4 18

00001 3 4

00001 4 2

00005 2 -1

00004 2 0

Sample Output:

1 00002 63 20 25 – 18

2 00005 42 20 0 22 –

2 00007 42 – 25 – 17

2 00001 42 18 18 4 2

5 00004 40 15 0 25 –

分析：结构体数组中passnum统计完整通过的题目个数，isshown在用户有一题通过了编译器（不管得不得0分）的时候置为true。vector<int> score;记录每门课的最高分

因为没有通过编译器的分数为0，但是没有提交过的分数为“-”，所以把有一门课每次都是未通过编译器的那门课分数置为-2。初始化数组分数为-1，所以可以根据-1和-2判断当前分数是提交过了没通过编译器的，还是没提交过的题目

注意：因为最后一个测试样例是有一个人一开始得到了分数，后来提交了一次没有通过编译器的，所以要判断在分数每次更新最大值之后if(v[id].score[num] == -1)，说明最好成绩只是-1（也就是没通过编译器或者没有提交过），这个时候再置v[id].score[num] = -2，否则会误操作把已经提交过很好分数的人的成绩抹掉成了-2

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct node {
    int rank, id, total = 0;
    vector<int> score;
    int passnum = 0;
    bool isshown = false;
};
bool cmp1(node a, node b) {
    if(a.total != b.total)
        return a.total > b.total;
```

```
    else if(a.passnum != b.passnum)
        return a.passnum > b.passnum;
    else
        return a.id < b.id;
}

int main() {
    int n, k, m, id, num, score;
    scanf("%d %d %d", &n, &k, &m);
    vector<node> v(n + 1);
    for(int i = 1; i <= n; i++)
        v[i].score.resize(k + 1, -1);
    vector<int> full(k + 1);
    for(int i = 1; i <= k; i++)
        scanf("%d", &full[i]);
    for(int i = 0; i < m; i++) {
        scanf("%d %d %d", &id, &num, &score);
        v[id].id = id;
        v[id].score[num] = max(v[id].score[num], score);
        if(score != -1)
            v[id].isshown = true;
        else if(v[id].score[num] == -1)
            v[id].score[num] = -2;
    }
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= k; j++) {
            if(v[i].score[j] != -1 && v[i].score[j] != -2)
                v[i].total += v[i].score[j];
            if(v[i].score[j] == full[j])
                v[i].passnum++;
        }
    }
    sort(v.begin() + 1, v.end(), cmp1);
    for(int i = 1; i <= n; i++) {
        v[i].rank = i;
        if(i != 1 && v[i].total == v[i - 1].total)
            v[i].rank = v[i - 1].rank;
    }
    for(int i = 1; i <= n; i++) {
        if(v[i].isshown == true) {
            printf("%d %05d %d", v[i].rank, v[i].id, v[i].total);
            for(int j = 1; j <= k; j++) {
                if(v[i].score[j] != -1 && v[i].score[j] != -2)
                    printf(" %d", v[i].score[j]);
                else if(v[i].score[j] == -1)
                    printf(" -");
                else
                    printf(" 0");
            }
            printf("\n");
        }
    }
```

```
66          return 0;
67      }
```

1076. Forwards on Weibo (30) [图的遍历， BFS]

Weibo is known as the Chinese version of Twitter. One user on Weibo may have many followers, and may follow many other users as well. Hence a social network is formed with followers relations. When a user makes a post on Weibo, all his/her followers can view and forward his/her post, which can then be forwarded again by their followers. Now given a social network, you are supposed to calculate the maximum potential amount of forwards for any specific user, assuming that only L levels of indirect followers are counted.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers: N (<=1000), the number of users; and L (<=6), the number of levels of indirect followers that are counted. Hence it is assumed that all the users are numbered from 1 to N. Then N lines follow, each in the format:

M[i] user_list[i]

where M[i] (<=100) is the total number of people that user[i] follows; and user_list[i] is a list of the M[i] users that are followed by user[i]. It is guaranteed that no one can follow oneself. All the numbers are separated by a space.

Then finally a positive K is given, followed by K UserID’s for query.

Output Specification:

For each UserID, you are supposed to print in one line the maximum potential amount of forwards this user can triger, assuming that everyone who can view the initial post will forward it once, and that only L levels of indirect followers are counted.

Sample Input:

7 3

3 2 3 4

0

2 5 6

2 3 1

2 3 4

1 4

1 5

2 2 6

Sample Output:

4

5

题目大意：给出每个用户关注的人的id，和转发最多的层数，求一个id发了条微博最多会有多少个人转发

分析：带层数的广度优先，因为一个用户只能转发一次，所以用inq判断当前结点是否入队过了，如果入队过了就不能重复入队（重复转发消息），inq 邻接表v 都可以使用int只存储id，queue的数据类型必须为node，同时保存它的id和layer层数，控制不超过L层～

```
#include <cstdio>
#include <queue>
#include <vector>
using namespace std;
int n, l, m, k;
struct node {
    int id, layer;
};
vector<vector<int>> v;
int bfs(node tnode) {
    bool inq[1010] = {false};
    queue<node> q;
    q.push(tnode);
    inq[tnode.id] = true;
    int cnt = 0;
    while(!q.empty()) {
        node top = q.front();
        q.pop();
        int topid = top.id;
        for(int i = 0; i < v[topid].size(); i++) {
            int nextid = v[topid][i];
            if(inq[nextid] == false && top.layer < l) {
                node next = {nextid, top.layer + 1};
                q.push(next);
                inq[next.id] = true;
                cnt++;
            }
        }
    }
    return cnt;
}

int main() {
    scanf("%d %d", &n, &l);
    v.resize(n + 1);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &m);
        for(int j = 0; j < m; j++) {
            int temp;
            scanf("%d", &temp);
            v[temp].push_back(i);
        }
    }
    scanf("%d", &k);
    int tid;
```

```
46        for(int i = 0; i < k; i++) {
47            scanf("%d", &tid);
48            node tnode = {tid, 0};
49            printf("%d\n", bfs(tnode));
50        }
51        return 0;
52    }
```

1077. Kuchiguse (20) [字符串处理]

The Japanese language is notorious for its sentence ending particles. Personal preference of such particles can be considered as a reflection of the speaker's personality. Such a preference is called "Kuchiguse" and is often exaggerated artistically in Anime and Manga. For example, the artificial sentence ending particle "nyan~" is often used as a stereotype for characters with a cat-like personality:

Itai nyan~ (It hurts, nyan~)

Ninjin wa iyada nyan~ (I hate carrots, nyan~)

Now given a few lines spoken by the same character, can you find her Kuchiguse?

Input Specification:

Each input file contains one test case. For each case, the first line is an integer N (2<=N<=100). Following are N file lines of 0~256 (inclusive) characters in length, each representing a character's spoken line. The spoken lines are case sensitive.

Output Specification:

For each test case, print in one line the kuchiguse of the character, i.e., the longest common suffix of all N lines. If there is no such suffix, write "nai".

Sample Input 1:

3

Itai nyan~

Ninjin wa iyadanyan~

uhhh nyan~

Sample Output 1:

nyan~

Sample Input 2:

3

Itai!

Ninjinnwaiyada T_T

T_T

Sample Output 2:

nai

题目大意：给定N给字符串，求他们的公共后缀，如果不存在公共后缀，就输出“nai”

分析：因为是后缀，反过来比较太麻烦，所以每输入一个字符串，就把它逆序过来再比较会比较容易～

首先ans = s；后来每输入的一个字符串，都和ans比较，如果后面不相同的就把它截取掉，

最后输出ans即可（要逆序输出～，所以先将ans倒置reverse一下～）

```
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
    int n;
    scanf("%d\n", &n);
    string ans;
    for(int i = 0; i < n; i++) {
        string s;
        getline(cin, s);
        int lens = s.length();
        reverse(s.begin(), s.end());
        if(i == 0) {
            ans = s;
            continue;
        } else {
            int lenans = ans.length();
            int minlen = min(lens, lenans);
            for(int j = 0; j < minlen; j++) {
                if(ans[j] != s[j]) {
                    ans = ans.substr(0, j);
                    break;
                }
            }
        }
    }
    reverse(ans.begin(), ans.end());
    if (ans.length() == 0) ans = "nai";
    cout << ans;
    return 0;
}
```

1078. Hashing (25) [二次方探查法]

The task of this problem is simple: insert a sequence of distinct positive integers into a hash table, and output the positions of the input numbers. The hash function is defined to be “H(key) = key % TSize” where TSize is the maximum size of the hash table. Quadratic probing (with positive increments only) is used to solve the collisions.

Note that the table size is better to be prime. If the maximum size given by the user is not prime, you must re-define the table size to be the smallest prime number which is larger than the size given by the user.

Input Specification:

Each input file contains one test case. For each case, the first line contains two positive numbers: MSize (<=104) and N (<=MSize) which are the user-defined table size and the number of input numbers, respectively. Then N distinct positive integers are given in the next line. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the corresponding positions (index starts from 0) of the input numbers in one line. All the numbers in a line are separated by a space, and there must be no extra space at the end of the line. In case it is impossible to insert the number, print “-” instead.

Sample Input:

4 4

10 6 4 15

Sample Output:

0 1 4 –

题目大意：给出散列表长和要插入的元素，将这些元素按照读入的顺序插入散列表中，其中散列函数为h(key) = key % TSize，解决冲突采用只向正向增加的二次方探查法。如果题中给出的TSize不是素数，就取第一个比TSize大的素数作为TSize

分析：先解决size是否为素数，不是素数就要重新赋值的问题

然后根据二次方探查法：

– 如果hashTable里面key % size的下标对应的hashTable为false,说明这个下标没有被使用过，直接输出。否则step步长从1加到size-1，一次次尝试是否能使index = (key + step * step) % size;所对应的位置没有元素，如果都没有找到就输出“-”，否则就输出这个找到的元素的位置～ – 注意 是(key + step * step) % size 而不是(key % size + step * step)

```
#include <iostream>
using namespace std;
int size, n, hashTable[10100];
bool isprime(int num) {
    if(num == 1) return false;
    for(int i = 2; i * i <= num; i++)
        if(num % i == 0) return false;
    return true;
}
void insert(int key) {
    for(int step = 0; step < size; step++) {
        int index = (key + step * step) % size;
        if(hashTable[index] == 0) {
            hashTable[index] = 1;
            cout << index % size;
            return ;
        }
    }
    cout << '-';
```

```
}
int main() {
    cin >> size >> n;
    while(!isprime(size)) size++;
    for(int i = 0; i < n; i++) {
        int key;
        cin >> key;
        if(i != 0) cout << ' ';
        insert(key);
    }
    return 0;
}
```

1079. Total Sales of Supply Chain (25) [DFS，BFS，树的遍历]

A supply chain is a network of retailers（零售商）, distributors（经销商）, and suppliers（供应商）– everyone involved in moving a product from supplier to customer.

Starting from one root supplier, everyone on the chain buys products from one's supplier in a price P and sell or distribute them in a price that is r% higher than P. Only the retailers will face the customers. It is assumed that each member in the supply chain has exactly one supplier except the root supplier, and there is no supply cycle.

Now given a supply chain, you are supposed to tell the total sales from all the retailers.

Input Specification:

Each input file contains one test case. For each case, the first line contains three positive numbers: N (<=105), the total number of the members in the supply chain (and hence their ID's are numbered from 0 to N-1, and the root supplier's ID is 0); P, the unit price given by the root supplier; and r, the percentage rate of price increment for each distributor or retailer. Then N lines follow, each describes a distributor or retailer in the following format:

Ki ID[1] ID[2] … ID[Ki]

where in the i-th line, Ki is the total number of distributors or retailers who receive products from supplier i, and is then followed by the ID's of these distributors or retailers. Kj being 0 means that the j-th member is a retailer, then instead the total amount of the product will be given after Kj. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the total sales we can expect from all the retailers, accurate up to 1 decimal place. It is guaranteed that the number will not exceed 1010.

Sample Input:

10 1.80 1.00

3 2 3 5

1 9

1 4

1 7

0 7

2 6 1

1 8

0 9

0 4

0 3

Sample Output:

42.4

题目大意：给一棵树，在树根出货物的价格为p，然后从根结点开始每往下走一层，该层的货物价格将会在父亲结点的价格上增加r%，给出每个叶结点的货物量，求他们的价格之和

分析：树的遍历，可以采用dfs或者bfs两种方法。

采用dfs，建立结构体数组保存每一个结点的孩子结点的下标，如果没有孩子结点，就保存这个叶子结点的data（销售的量）。深度优先遍历的递归出口，即当前下标的结点没有孩子结点的时候，就把ans += data（货物量）* pow(1 + r, depth)计算货物量*价格引起的涨幅百分比。如果有孩子结点，就dfs深度优先遍历每一个孩子结点，并且在当前depth层数的基础上+1。最后输出ans * p（销售价格），即总价格

```
#include <cstdio>
#include <vector>
#include <cmath>
using namespace std;
struct node {
    double data;
    vector<int> child;
};
vector<node> v;
double ans = 0.0, p, r;

void dfs(int index, int depth) {
    if(v[index].child.size() == 0) {
        ans += v[index].data * pow(1 + r, depth);
        return ;
    }
    for(int i = 0; i < v[index].child.size(); i++)
        dfs(v[index].child[i], depth + 1);
}
int main() {
    int n, k, c;
    scanf("%d %lf %lf", &n, &p, &r);
    r = r / 100;
    v.resize(n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &k);
```

```
27            if(k == 0) {
28                scanf("%lf", &v[i].data);
29            } else {
30                for(int j = 0; j < k; j++) {
31                    scanf("%d", &c);
32                    v[i].child.push_back(c);
33                }
34            }
35        }
36        dfs(0, 0);
37        printf("%.1f", ans * p);
38        return 0;
39    }
```

1080. Graduate Admission (30) [排序]

It is said that in 2013, there were about 100 graduate schools ready to proceed over 40,000 applications in Zhejiang Province. It would help a lot if you could write a program to automate the admission procedure.

Each applicant will have to provide two grades: the national entrance exam grade GE, and the interview grade GI. The final grade of an applicant is (GE + GI) / 2. The admission rules are:

The applicants are ranked according to their final grades, and will be admitted one by one from the top of the rank list.

If there is a tied final grade, the applicants will be ranked according to their national entrance exam grade GE. If still tied, their ranks must be the same.

Each applicant may have K choices and the admission will be done according to his/her choices: if according to the rank list, it is one's turn to be admitted; and if the quota of one's most preferred shcool is not exceeded, then one will be admitted to this school, or one's other choices will be considered one by one in order. If one gets rejected by all of preferred schools, then this unfortunate applicant will be rejected.

If there is a tied rank, and if the corresponding applicants are applying to the same school, then that school must admit all the applicants with the same rank, even if its quota will be exceeded.

Input Specification:

Each input file contains one test case. Each case starts with a line containing three positive integers: N (<=40,000), the total number of applicants; M (<=100), the total number of graduate schools; and K (<=5), the number of choices an applicant may have.

In the next line, separated by a space, there are M positive integers. The i-th integer is the quota of the i-th graduate school respectively.

Then N lines follow, each contains 2+K integers separated by a space. The first 2 integers are the applicant's GE and GI, respectively. The next K integers represent the preferred schools. For the sake of simplicity, we assume that the schools are numbered from 0 to M-1, and the applicants are numbered from 0 to N-1.

Output Specification:

For each test case you should output the admission results for all the graduate schools. The results of each school must occupy a line, which contains the applicants' numbers that school admits. The numbers must be in increasing order and be separated by a space. There must be no extra space at the end of each line. If no applicant is admitted by a school, you must output an empty line correspondingly.

Sample Input:

11 6 3

2 1 2 2 2 3

100 100 0 1 2

60 60 2 3 5

100 90 0 3 4

90 100 1 2 0

90 90 5 1 3

80 90 1 0 2

80 80 0 1 2

80 80 0 1 2

80 70 1 3 2

70 80 1 2 3

100 100 0 2 4

Sample Output:

0 10

3

5 6 7

2 8

1 4

题目大意：每个考生有两个成绩：GE和GI，最终成绩为(GE + GI) / 2;按照最终成绩排名，如果最终成绩相同，就按照GE排名，如果仍然相同，他们的排名就是相同的。每个申请者有K个选择院校，每个学校也有招生人数限制。按照排名先后，如果当前考生的第一个志愿学校的名额还没满，就录取进去；如果当前志愿名额满了但是该校最后一个录取的人的排名和当前考生相同，则不管招生人数限制，依旧应该被录取；否则考虑该生的下一个志愿。如果所有志愿都没有能被录取，则该生落榜。

分析：

1.stu容器里放学生{id, ge, gi, fin, choice(容器里放学生报考学校的id)}, quota数组放招生计划的数量，cnt数组存放当前学校已经招收的学生数，sch数组里放的容器，容器里是学校已经招的学生的id～

2.对学生按照分数排序，依次学生遍历，分数最高的学生先挑学校～

3.对于每个学生录取到哪里：依次遍历学生的报考志愿，如果（没招满 || 他与已经招的学生的最后一名成绩并列）就把他招进去，该学生录取结果即可确定，更新该学校已经招生的人数，并把次学生加入该学校录取容器中～

4.输出学校录取情况时学生id顺序是乱的，要先从小到大排序，然后输出。每个学校占一行～

5.排序函数要用 & 引用传参，不然会超时～

6.因为分数 fin = ge + gi 不会超出int, fin / 2 和fin排名效果一样， 不除2不会影响结果，而且还可以巧妙躲避除2后double不能精确表示的问题～

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct peo{
    int id, ge, gi, fin;
    vector<int> choice;
};
bool cmp(peo& a, peo& b) {
    if (a.fin != b.fin) return a.fin > b.fin;
    return a.ge > b.ge;
}
bool cmp2(peo& a, peo& b) {
  return a.id < b.id;
}
int main(){
    int n, m, k, quota[110], cnt[110] = {0};
    scanf("%d%d%d", &n, &m, &k);
    vector<peo> stu(n), sch[110];
    for(int i = 0; i < m; i++)
        scanf("%d",&quota[i]);
    for(int i = 0; i < n; i++) {
        scanf("%d%d", &stu[i].ge, &stu[i].gi);
        stu[i].id = i;
        stu[i].fin = stu[i].ge + stu[i].gi;
        stu[i].choice.resize(k);
        for(int j = 0; j < k; j++)
            scanf("%d", &stu[i].choice[j]);
    }
    sort(stu.begin(), stu.end(), cmp);
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < k; j++) {
            int schid = stu[i].choice[j];
            int lastindex = cnt[schid] - 1;
            if(cnt[schid] < quota[schid] || (stu[i].fin == sch[schid]
[lastindex].fin) && stu[i].ge == sch[schid][lastindex].ge) {
                sch[schid].push_back(stu[i]);
                cnt[schid]++;
                break;
            }
        }
    }
```

```
    for(int i = 0; i < m; i++) {
        sort(sch[i].begin(), sch[i].end(), cmp2);
        for(int j = 0; j < cnt[i]; j++) {
            if(j != 0) printf(" ");
            printf("%d", sch[i][j].id);
        }
        printf("\n");
    }
    return 0;
}
```

1081. Rational Sum (20) [分数的四则运算]

Given N rational numbers in the form “numerator/denominator”, you are supposed to calculate their sum.

Input Specification:

Each input file contains one test case. Each case starts with a positive integer N (<=100), followed in the next line N rational numbers “a1/b1 a2/b2 …” where all the numerators and denominators are in the range of “long int”. If there is a negative number, then the sign must appear in front of the numerator.

Output Specification:

For each test case, output the sum in the simplest form “integer numerator/denominator” where “integer” is the integer part of the sum, “numerator” < “denominator”, and the numerator and the denominator have no common factor. You must output only the fractional part if the integer part is 0.

Sample Input 1:

5

2/5 4/15 1/30 -2/60 8/3

Sample Output 1:

3 1/3

Sample Input 2:

2

4/3 2/3

Sample Output 2:

2

Sample Input 3:

3

1/3 -1/6 1/8

Sample Output 3:

7/24

题目大意：给N个有理数（以分子/分母的形式给出），计算这N个数的总和，最后总和要以（整数 分子/分母）的形式给出～

分析：先根据分数加法的公式累加，后分离出整数部分和分数部分

分子和分母都在长整型内，所以不能用int存储，否则有一个测试点不通过

一开始一直是浮点错误，按理来说应该是出现了/0或者%0的情况，找了半天也不知道错在哪里，

后来注意到应该在累加的时候考虑是否会超出long long的范围，所以在累加每一步之前进行分子分母的约分处理，然后就AC了～

应该还要考虑整数和小数部分都为0时候输出0的情况，但是测试用例中不涉及，所以如果没有最后两句也是可以AC的（PS：据说更新后的系统已经需要考虑全零的情况了～）

```
#include <iostream>
#include <cstdlib>
using namespace std;
long long gcd(long long a, long long b) {return b == 0 ? abs(a) : gcd(b, a %
b);}
int main() {
    long long n, a, b, suma = 0, sumb = 1, gcdvalue;
    scanf("%lld", &n);
    for(int i = 0; i < n; i++) {
        scanf("%lld/%lld", &a, &b);
        gcdvalue = gcd(a, b);
        a = a / gcdvalue;
        b = b / gcdvalue;
        suma = a * sumb + suma * b;
        sumb = b * sumb;
        gcdvalue = gcd(suma, sumb);
        sumb = sumb / gcdvalue;
        suma = suma / gcdvalue;
    }
    long long integer = suma / sumb;
    suma = suma - (sumb * integer);
    if(integer != 0) {
        printf("%lld", integer);
        if(suma != 0) printf(" ");
    }
    if(suma != 0)
        printf("%lld/%lld", suma, sumb);
    if(integer == 0 && suma == 0)
        printf("0");
    return 0;
}
```

1082. Read Number in Chinese (25) [字符串处理]

Given an integer with no more than 9 digits, you are supposed to read it in the traditional Chinese way. Output “Fu” first if it is negative. For example, -123456789 is read as “Fu yi Yi er Qian san Bai si Shi wu Wan liu Qian qi Bai ba Shi jiu”. Note: zero (“ling”) must be handled correctly according to the Chinese tradition. For example, 100800 is “yi Shi Wan ling ba Bai”.

Input Specification:

Each input file contains one test case, which gives an integer with no more than 9 digits.

Output Specification:

For each test case, print in a line the Chinese way of reading the number. The characters are separated by a space and there must be no extra space at the end of the line.

Sample Input 1:

-123456789

Sample Output 1:

Fu yi Yi er Qian san Bai si Shi wu Wan liu Qian qi Bai ba Shi jiu

Sample Input 2:

100800

Sample Output 2:

yi Shi Wan ling ba Bai

题目大意：给定一个不超过9位的整数，你应该用传统的中文方式阅读它~ 如果是负的，首先输出“Fu”。 例如，-123456789被读作“Fu yi Yi er Qian san Bai si Shi wu Wan liu Qian qi Bai ba Shi jiu”。 注意：零（“ling”）必须根据中国传统正确处理。 例如，100800是“yi Shi Wan ling ba Bai”~

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
string num[10] = { "ling","yi", "er", "san", "si", "wu", "liu", "qi", "ba",
"jiu" };
string c[6] = { "Ge","Shi", "Bai", "Qian", "Yi", "Wan" };
int J[] = {1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000};
vector<string> res;
int main() {
    int n;
    cin >> n;
    if (n == 0) {
        cout << "ling";
        return 0;
    }
    if (n < 0) {
        cout << "Fu ";
        n = -n;
    }
    int part[3];
    part[0]= n / 100000000;
    part[1]= (n % 100000000) / 10000;
    part[2] = n % 10000;
    bool zero = false; //是否在非零数字前输出合适的ling
    int printCnt = 0; //用于维护单词前没有空格，之后输入的单词都在前面加一个空格。
```

```
    for (int i = 0; i < 3; i++) {
        int temp = part[i]; //三个部分，每部分内部的命名规则都一样，都是X千X百X十X
        for (int j = 3; j >= 0; j--) {
            int curPos = 8 - i * 4 + j; //当前数字的位置
            if (curPos >= 9) continue; //最多九位数
            int cur = (temp / J[j]) % 10;//取出当前数字
            if (cur != 0) {
                if (zero) {
                    printCnt++ == 0 ? cout<<"ling" : cout<<" ling";
                    zero = false;
                }
                if (j == 0)
                    printCnt++ == 0 ? cout << num[cur] : cout << ' ' << num[cur]; //在个位，直接输出
                else
                    printCnt++ == 0 ? cout << num[cur] << ' ' << c[j] : cout << ' ' << num[cur] << ' ' << c[j]; //在其他位，还要输出十百千
            } else {
                if (!zero && j != 0 && n / J[curPos] >= 10) zero = true;   //注意100020这样的情况
            }
        }
        if (i != 2 && part[i]>0) cout << ' ' << c[i + 4]; //处理完每部分之后，最后输出单位，Yi/Wan
    }
    return 0;
}
```

1083. List Grades (25) [排序]

Given a list of N student records with name, ID and grade. You are supposed to sort the records with respect to the grade in non-increasing order, and output those student records of which the grades are in a given interval.

Input Specification:

Each input file contains one test case. Each case is given in the following format:

N

name[1] ID[1] grade[1]

name[2] ID[2] grade[2]

… …

name[N] ID[N] grade[N]

grade1 grade2

where name[i] and ID[i] are strings of no more than 10 characters with no space, grade[i] is an integer in [0, 100], grade1 and grade2 are the boundaries of the grade's interval. It is guaranteed that all the grades are distinct.

Output Specification:

For each test case you should output the student records of which the grades are in the given interval [grade1, grade2] and are in non-increasing order. Each student record occupies a line with the student's name and ID, separated by one space. If there is no student's grade in that interval, output "NONE" instead.

Sample Input 1:

4

Tom CS000001 59

Joe Math990112 89

Mike CS991301 100

Mary EE990830 95

60 100

Sample Output 1:

Mike CS991301

Mary EE990830

Joe Math990112

Sample Input 2:

2

Jean AA980920 60

Ann CS01 80

90 95

Sample Output 2:

NONE

题目大意：给出n个考生的信息，按照分数从高到低排序，并且输出给定区间的考生信息。如果不存在满足条件的考生就输出NONE。

分析：建立结构体数组，将不满足条件的学生grade改为-1，并统计满足条件的学生的个数cnt，然后进行排序，输出前cnt个考生的信息～～ 注意：因为每个学生的成绩都不同，所以按照下降排列即可，return a.grade > b.grade;

注意：因为每个学生的成绩都不同，所以按照下降排列即可，return a.grade > b.grade;

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct stu {
    char name[12];
    char id[12];
    int grade;
```

```
};
int cmp1(stu a, stu b) {
    return a.grade > b.grade;
}
int main() {
    int n, low, high, cnt = 0;
    scanf("%d", &n);
    vector<stu> v(n);
    for(int i = 0; i < n; i++) {
        scanf("%s %s %d", v[i].name, v[i].id, &v[i].grade);
    }
    scanf("%d %d", &low, &high);
    for(int i = 0; i < n; i++) {
        if(v[i].grade < low || v[i].grade > high) {
            v[i].grade = -1;
        } else {
            cnt++;
        }
    }
    sort(v.begin(), v.end(), cmp1);
    for(int i = 0; i < cnt; i++) {
        printf("%s %s\n", v[i].name, v[i].id);
    }
    if(cnt == 0)
        printf("NONE");
    return 0;
}
```

1084. Broken Keyboard (20) [Hash散列]

On a broken keyboard, some of the keys are worn out. So when you type some sentences, the characters corresponding to those keys will not appear on screen.

Now given a string that you are supposed to type, and the string that you actually type out, please list those keys which are for sure worn out.

Input Specification:

Each input file contains one test case. For each case, the 1st line contains the original string, and the 2nd line contains the typed-out string. Each string contains no more than 80 characters which are either English letters [A-Z] (case insensitive), digital numbers [0-9], or “_” (representing the space). It is guaranteed that both strings are non-empty.

Output Specification:

For each test case, print in one line the keys that are worn out, in the order of being detected. The English letters must be capitalized. Each worn out key must be printed once only. It is guaranteed that there is at least one worn out key.

Sample Input:

7_This_is_a_test

hssaes

Sample Output:

7TI

题目大意：旧键盘上坏了几个键，于是在敲一段文字的时候，对应的字符就不会出现。现在给出应该输入的一段文字、以及实际被输入的文字，请你列出肯定坏掉的那些键～

题目大意：旧键盘上坏了几个键，于是在敲一段文字的时候，对应的字符就不会出现。现在给出应该输入的一段文字、以及实际被输入的文字，请你列出肯定坏掉的那些键～

分析：用string的find函数～遍历字符串s1，当当前字符s1[i]不在s2中，它的大写也不在ans中时，将当前字符的大写放入ans中，最后输出ans字符串即可～

ps：感谢github上的@xiaorong61给我发的pull request中strchr()函数带来的灵感～让我想到了曾经用过的string的find函数～

```
#include <iostream>
#include <cctype>
using namespace std;
int main() {
    string s1, s2, ans;
    cin >> s1 >> s2;
    for (int i = 0; i < s1.length(); i++)
        if (s2.find(s1[i]) == string::npos && ans.find(toupper(s1[i])) ==
string::npos)
            ans += toupper(s1[i]);
    cout << ans;
    return 0;
}
```

1085. Perfect Sequence (25) [二分，two pointers]

Given a sequence of positive integers and another positive integer p. The sequence is said to be a “perfect sequence” if M <= m * p where M and m are the maximum and minimum numbers in the sequence, respectively.

Now given a sequence and a parameter p, you are supposed to find from the sequence as many numbers as possible to form a perfect subsequence.

Input Specification:

Each input file contains one test case. For each case, the first line contains two positive integers N and p, where N (<= 105) is the number of integers in the sequence, and p (<= 109) is the parameter. In the second line there are N positive integers, each is no greater than 109.

Output Specification:

For each test case, print in one line the maximum number of integers that can be chosen to form a perfect subsequence.

Sample Input:

10 8

2 3 20 4 5 1 6 7 8 9

Sample Output:

8

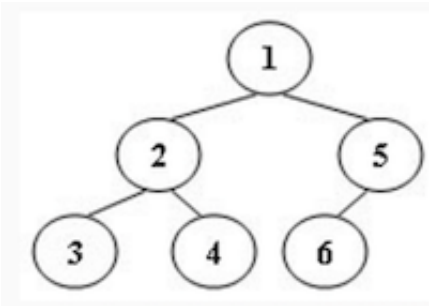
题目大意：给定一个正整数数列，和正整数p，设这个数列中的最大值是M，最小值是m，如果M <= m * p，则称这个数列是完美数列。现在给定参数p和一些正整数，请你从中选择尽可能多的数构成一个完美数列。输入第一行给出两个正整数N（输入正数的个数）和p（给定的参数），第二行给出N个正整数。在一行中输出最多可以选择多少个数可以用它们组成一个完美数列

分析：简单题。首先将数列从小到大排序，设当前结果为result = 0，当前最长长度为temp = 0；从i = 0～n，j从i + result到n，【因为是为了找最大的result，所以下一次j只要从i的result个后面开始找就行了】每次计算temp若大于result则更新result，最后输出result的值～

【solution 2】如果熟练使用upper_bound()，可以使用以下解法解决问题（代码由Github用户littlesevenmo提供，在此表达感谢）：

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main() {
    int n;
    long long p;
    scanf("%d%lld", &n, &p);
    vector<int> v(n);
    for (int i = 0; i < n; i++)
        cin >> v[i];
    sort(v.begin(), v.end());
    int result = 0, temp = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + result; j < n; j++) {
            if (v[j] <= v[i] * p) {
                temp = j - i + 1;
                if (temp > result)
                    result = temp;
            } else {
                break;
            }
        }
    }
    cout << result;
    return 0;
}
```

1086. Tree Traversals Again (25) [树的遍历]



Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<=30) which is the total number of nodes in a tree (and hence the nodes are numbered from 1 to N). Then 2N lines follow, each describes a stack operation in the format: “Push X” where X is the index of the node being pushed onto the stack; or “Pop” meaning to pop one node from the stack.

Output Specification:

For each test case, print the postorder traversal sequence of the corresponding tree in one line. A solution is guaranteed to exist. All the numbers must be separated by exactly one space, and there must be no extra space at the end of the line.

Sample Input:

6

Push 1

Push 2

Push 3

Pop

Pop

Push 4

Pop

Pop

Push 5

Push 6

Pop

Pop

Sample Output:

3 4 2 6 5 1

题目大意：用栈的形式给出一棵二叉树的建立的顺序，求这棵二叉树的后序遍历

分析：栈实现的是二叉树的中序遍历（左根右），而每次push入值的顺序是二叉树的前序遍历（根左右），所以该题可以用二叉树前序和中序转后序的方法做～

root为当前子树的根结点在前序pre中的下标，start和end为当前子树的最左边和最右边的结点在中序in中的下标。用i找到当前子树的根结点root在中序中的下标，然后左边和右边就分别为当前根结点root的左子树和右子树。递归实现～

Update：Github用户littlesevenmo给我发issue提出题目并没有说所有节点的值互不相同。因此，在有多个节点的值相同的情况下，之前的代码会输出错误的结果，所以修改后的代码中添加了key作为索引，前中后序中均保存索引值，然后用value存储具体的值，修改后的代码如下：

```
#include <cstdio>
#include <vector>
#include <stack>
#include <cstring>
using namespace std;
vector<int> pre, in, post,value;
void postorder(int root, int start, int end) {
    if (start > end) return;
    int i = start;
    while (i < end && in[i] != pre[root]) i++;
    postorder(root + 1, start, i - 1);
    postorder(root + 1 + i - start, i + 1, end);
    post.push_back(pre[root]);
}
int main() {
    int n;
    scanf("%d", &n);
    char str[5];
    stack<int> s;
    int key=0;
    while (~scanf("%s", str)) {
        if (strlen(str) == 4) {
            int num;
            scanf("%d", &num);
            value.push_back(num);
            pre.push_back(key);
            s.push(key++);
        } else {
            in.push_back(s.top());
            s.pop();
        }
    }
    postorder(0, 0, n - 1);
    printf("%d", value[post[0]]);
    for (int i = 1; i < n; i++)
        printf(" %d",value[post[i]]);
    return 0;
}
```

1087. All Roads Lead to Rome (30) [Dijkstra算法 + DFS，最短路径]

Indeed there are many different tourist routes from our city to Rome. You are supposed to find your clients the route with the least cost while gaining the most happiness.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers N (2<=N<=200), the number of cities, and K, the total number of routes between pairs of cities; followed by the name of the starting city. The next N-1 lines each gives the name of a city and an integer that represents the happiness one can gain from that city, except the starting city. Then K lines follow, each describes a route between two cities in the format “City1 City2 Cost”. Here the name of a city is a string of 3 capital English letters, and the destination is always ROM which represents Rome.

Output Specification:

For each test case, we are supposed to find the route with the least cost. If such a route is not unique, the one with the maximum happiness will be recommended. If such a route is still not unique, then we output the one with the maximum average happiness — it is guaranteed by the judge that such a solution exists and is unique.

Hence in the first line of output, you must print 4 numbers: the number of different routes with the least cost, the cost, the happiness, and the average happiness (take the integer part only) of the recommended route. Then in the next line, you are supposed to print the route in the format “City1->City2->…->ROM”.

Sample Input:

6 7 HZH

ROM 100

PKN 40

GDN 55

PRS 95

BLN 80

ROM GDN 1

BLN ROM 1

HZH PKN 1

PRS ROM 2

BLN HZH 2

PKN GDN 1

HZH PRS 1

Sample Output:

3 3 195 97

HZH->PRS->ROM

题目大意：有N个城市，M条无向边，从某个给定的起始城市出发，前往名为ROM的城市。每个城市（除了起始城市）都有一个点权（称为幸福值），和边权（每条边所需的花费）。求从起点到ROM所需要的最少花费，并输出其路径。如果路径有多条，给出幸福值最大的那条。如果仍然不唯一，选择路径上的城市平均幸福值最大的那条路径

分析：Dijkstra+DFS。Dijkstra算出所有最短路径的路线pre二维数组，DFS求最大happiness的路径path，并求出路径条数，最大happiness，最大average～

注意：average是除去起点的average。城市名称可以用m存储字符串所对应的数字，用m2存储数字对应的字符串

```
#include <iostream>
#include <map>
#include <vector>
#include <algorithm>
using namespace std;
int n, k;
const int inf = 999999999;
int e[205][205], weight[205], dis[205];
bool visit[205];
vector<int> pre[205], temppath, path;
map<string, int> m;
map<int, string> m2;
int maxvalue = 0, mindepth, cntpath = 0;
double maxavg;
void dfs(int v) {
    temppath.push_back(v);
    if(v == 1) {
        int value = 0;
        for(int i = 0; i < temppath.size(); i++) {
            value += weight[temppath[i]];
        }
        double tempavg = 1.0 * value / (temppath.size() - 1);
        if(value > maxvalue) {
            maxvalue = value;
            maxavg = tempavg;
            path = temppath;
        } else if(value == maxvalue && tempavg > maxavg) {
            maxavg = tempavg;
            path = temppath;
        }
        temppath.pop_back();
        cntpath++;
        return ;
    }
    for(int i = 0; i < pre[v].size(); i++) {
        dfs(pre[v][i]);
    }
    temppath.pop_back();
}
int main() {
    fill(e[0], e[0] + 205 * 205, inf);
```

```
    fill(dis, dis + 205, inf);
    scanf("%d %d", &n, &k);
    string s;
    cin >> s;
    m[s] = 1;
    m2[1] = s;
    for(int i = 1; i < n; i++) {
        cin >> s >> weight[i+1];
        m[s] = i+1;
        m2[i+1] = s;
    }
    string sa, sb;
    int temp;
    for(int i = 0; i < k; i++) {
        cin >> sa >> sb >> temp;
        e[m[sa]][m[sb]] = temp;
        e[m[sb]][m[sa]] = temp;
    }
    dis[1] = 0;
    for(int i = 0; i < n; i++) {
        int u = -1, minn = inf;
        for(int j = 1; j <= n; j++) {
            if(visit[j] == false && dis[j] < minn) {
                u = j;
                minn = dis[j];
            }
        }
        if(u == -1) break;
        visit[u] = true;
        for(int v = 1; v <= n; v++) {
            if(visit[v] == false && e[u][v] != inf) {
                if(dis[u] + e[u][v] < dis[v]) {
                    dis[v] = dis[u] + e[u][v];
                    pre[v].clear();
                    pre[v].push_back(u);
                } else if(dis[v] == dis[u] + e[u][v]) {
                    pre[v].push_back(u);
                }
            }
        }
    }
    int rom = m["ROM"];
    dfs(rom);
    printf("%d %d %d %d\n", cntpath, dis[rom], maxvalue, (int)maxavg);
    for(int i = path.size() - 1; i >= 1; i--) {
        cout << m2[path[i]] << "->";
    }
    cout << "ROM";
    return 0;
}
```

1088. Rational Arithmetic (20) [分数的四则运算]

For two rational numbers, your task is to implement the basic arithmetics, that is, to calculate their sum, difference, product and quotient.

Input Specification:

Each input file contains one test case, which gives in one line the two rational numbers in the format “a1/b1 a2/b2”. The numerators and the denominators are all in the range of long int. If there is a negative sign, it must appear only in front of the numerator. The denominators are guaranteed to be non-zero numbers.

Output Specification:

For each test case, print in 4 lines the sum, difference, product and quotient of the two rational numbers, respectively. The format of each line is “number1 operator number2 = result”. Notice that all the rational numbers must be in their simplest form “k a/b”, where k is the integer part, and a/b is the simplest fraction part. If the number is negative, it must be included in a pair of parentheses. If the denominator in the division is zero, output “Inf” as the result. It is guaranteed that all the output integers are in the range of long int.

Sample Input 1:

2/3 -4/2

Sample Output 1:

2/3 + (-2) = (-1 1/3)

2/3 – (-2) = 2 2/3

2/3 * (-2) = (-1 1/3)

2/3 / (-2) = (-1/3)

Sample Input 2:

5/3 0/6

Sample Output 2:

1 2/3 + 0 = 1 2/3

1 2/3 – 0 = 1 2/3

1 2/3 * 0 = 0

1 2/3 / 0 = Inf

题目大意：输入在一行中按照“a1/b1 a2/b2”的格式给出两个分数形式的有理数，其中分子和分母全是整型范围内的整数，负号只可能出现在分子前，分母不为0，分别在4行中按照“有理数1 运算符 有理数2 = 结果”的格式顺序输出2个有理数的和、差、积、商。注意输出的每个有理数必须是该有理数的最简形式“k a/b”，其中k是整数部分，a/b是最简分数部分；若为负数，则须加括号；若除法分母为0，则输出“Inf”～题目保证正确的输出中没有超过整型范围的整数～

分析：func(m, n)的作用是对m/n的分数进行化简，gcd(t1, t2)的作用是计算t1和t2的最大公约数～在func函数中，先看m和n里面是否有0（即m*n是否等于0），如果分母n=0，输出Inf，如果分子m=0，输出”0″～flag表示m和n是否异号，flag=true表示后面要添加负号”(-“和括号”)”，再将m和n都转为abs(m)和abs(n)，即取他们的正数部分方便计算～x = m/n为m和n的可提取的整数部分，先根据flag的结果判断是否要在前面追加”(-“，然后根据x是否等于0判断要不要输出这个整数位，接着根据m%n是否等于0的结果判断后面还有没有小分数，如果m能被n整除，表示没有后面的小分数，那么就根据flag的结果判断要不要加”)”，然后直接return～如果有整数位，且后面有小分数，则要先输出一个空格，接着处理剩下的小分数，先把m分子减去已经提取出的整数部分，然后求m和n的最大公约数t，让m和n都除以t进行化简～最后输出“m/n”，如果flag==true还要在末尾输出”)”

注意：判断m和n是否异号千万不要写成判断m*n是否小于0，因为m*n的结果可能超过了long long int的长度，导致溢出大于0，如果这样写的话会有一个测试点无法通过～（(⊙o⊙)嗯为了这一个测试点找bug找到凌晨两三点的就是我...我好菜啊.jpg）

```
#include <iostream>
#include <cmath>
using namespace std;
long long a, b, c, d;
long long gcd(long long t1, long long t2) {
    return t2 == 0 ? t1 : gcd(t2, t1 % t2);
}
void func(long long m, long long n) {
    if (m * n == 0) {
        printf("%s", n == 0 ? "Inf" : "0");
        return ;
    }
    bool flag = ((m < 0 && n > 0) || (m > 0 && n < 0));
    m = abs(m); n = abs(n);
    long long x = m / n;
    printf("%s", flag ? "(-" : "");
    if (x != 0) printf("%lld", x);
    if (m % n == 0) {
        if(flag) printf(")");
        return ;
    }
    if (x != 0) printf(" ");
    m = m - x * n;
    long long t = gcd(m, n);
    m = m / t; n = n / t;
    printf("%lld/%lld%s", m, n, flag ? ")" : "");
}
int main() {
    scanf("%lld/%lld %lld/%lld", &a, &b, &c, &d);
    func(a, b); printf(" + "); func(c, d); printf(" = "); func(a * d + b * c, b * d); printf("\n");
    func(a, b); printf(" - "); func(c, d); printf(" = "); func(a * d - b * c, b * d); printf("\n");
    func(a, b); printf(" * "); func(c, d); printf(" = "); func(a * c, b * d); printf("\n");
    func(a, b); printf(" / "); func(c, d); printf(" = "); func(a * d, b * c);
    return 0;
```

Mail: 820821725@qq.com UUID: e1343a18-aecf-31b5-baec-8398b3e0d2e7

```
35    }
```

1089. Insert or Merge (25) [two pointers]

According to Wikipedia:

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Merge sort works as follows: Divide the unsorted list into N sublists, each containing 1 element (a list of 1 element is considered sorted). Then repeatedly merge two adjacent sublists to produce new sorted sublists until there is only 1 sublist remaining.

Now given the initial sequence of integers, together with a sequence which is a result of several iterations of some sorting method, can you tell which sorting method we are using?

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=100). Then in the next line, N integers are given as the initial sequence. The last line contains the partially sorted sequence of the N numbers. It is assumed that the target sequence is always ascending. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in the first line either “Insertion Sort” or “Merge Sort” to indicate the method used to obtain the partial result. Then run this method for one more iteration and output in the second line the resulting sequence. It is guaranteed that the answer is unique for each test case. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

Sample Input 1:

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

Sample Output 1:

Insertion Sort

1 2 3 5 7 8 9 4 6 0

Sample Input 2:

10

3 1 2 8 7 5 9 4 0 6

1 3 2 8 5 7 4 9 0 6

Sample Output 2:

Merge Sort

1 2 3 8 4 5 7 9 0 6

题目大意：现给定原始序列和由某排序算法产生的中间序列，请你判断该算法是插入算法还是归并算法。首先在第1行中输出“Insertion Sort”表示插入排序、或“Merge Sort”表示归并排序；然后在第2行中输出用该排序算法再迭代一轮的结果序列

分析：先将i指向中间序列中满足从左到右是从小到大顺序的最后一个下标，再将j指向从i+1开始，第一个不满足a[j] == b[j]的下标，

如果j顺利到达了下标n，说明是插入排序，再下一次的序列是sort(a, a+i+2);否则说明是归并排序。归并排序就别考虑中间序列了，直接对原来的序列进行模拟归并时候的归并过程，i从0到n/k，每次一段段得sort(a + i * k, a + (i + 1) * k);最后别忘记还有最后剩余部分的sort(a + n / k * k, a + n);这样是一次归并的过程。直到有一次发现a的顺序和b的顺序相同，则再归并一次，然后退出循环～

注意：一开始第三个测试点一直不过，天真的我以为可以模拟一遍归并的过程然后在过程中判断下一步是什么。。然而真正的归并算法它是一个递归过程。。也就是先排左边一半，把左边的完全排列成正确的顺序之后，再排右边一半的。。而不是左右两边一起排列的。。后来改了自己的归并部分判断的代码就过了。。。。◕‿◕。

```
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
    int n, a[100], b[100], i, j;
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> a[i];
    for (int i = 0; i < n; i++)
        cin >> b[i];
    for (i = 0; i < n - 1 && b[i] <= b[i + 1]; i++);
    for (j = i + 1; a[j] == b[j] && j < n; j++);
    if (j == n) {
        cout << "Insertion Sort" << endl;
        sort(a, a + i + 2);
    } else {
        cout << "Merge Sort" << endl;
        int k = 1, flag = 1;
        while(flag) {
            flag = 0;
            for (i = 0; i < n; i++) {
                if (a[i] != b[i])
                    flag = 1;
            }
            k = k * 2;
            for (i = 0; i < n / k; i++)
                sort(a + i * k, a + (i + 1) * k);
            sort(a + n / k * k, a + n);
        }
    }
    for (j = 0; j < n; j++) {
        if (j != 0) printf(" ");
```

```
            printf("%d", a[j]);
        }
        return 0;
    }
```

1090. Highest Price in Supply Chain (25) [树的遍历]

A supply chain is a network of retailers（零售商）, distributors（经销商）, and suppliers（供应商）– everyone involved in moving a product from supplier to customer.

Starting from one root supplier, everyone on the chain buys products from one's supplier in a price P and sell or distribute them in a price that is r% higher than P. It is assumed that each member in the supply chain has exactly one supplier except the root supplier, and there is no supply cycle.

Now given a supply chain, you are supposed to tell the highest price we can expect from some retailers.

Input Specification:

Each input file contains one test case. For each case, The first line contains three positive numbers: N (<=105), the total number of the members in the supply chain (and hence they are numbered from 0 to N-1); P, the price given by the root supplier; and r, the percentage rate of price increment for each distributor or retailer. Then the next line contains N numbers, each number Si is the index of the supplier for the i-th member. Sroot for the root supplier is defined to be -1. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the highest price we can expect from some retailers, accurate up to 2 decimal places, and the number of retailers that sell at the highest price. There must be one space between the two numbers. It is guaranteed that the price will not exceed 1010.

Sample Input:

9 1.80 1.00

1 5 4 4 -1 4 5 3 6

Sample Output:

1.85 2

题目大意：给一棵树，在树根处货物的价格为p，然后每往下走一层，价格增加r%，求所有叶子结点的最高价格，以及这个价格的叶子结点个数。

分析：用二维数组v[i][j]存储，对于每一个结点i，它的孩子结点的下标push_back存储在v[i]中。用深度优先搜索dfs，保存当前结点的下标index以及当前结点所在层数，当 当前结点的孩子结点个数为0的时候说明是叶子结点，更新maxdepth和maxnum的值，最后输出～

注意：如果采用保存某个结点的父结点的下标的形式，然后一直遍历到根结点的深度/广度优先，会出现三个超时。因为从叶子结点往上遍历将会把所有路径都走一遍，很多都是重复走的路径，会超时，没有从根结点往下遍历的方式快～～

记得r是百分比，要除以100之后再计算复利～

```
#include <iostream>
```

```cpp
#include <cmath>
#include <vector>
using namespace std;
int n, maxdepth = 0, maxnum = 0, temp, root;
vector<int> v[100010];
void dfs(int index, int depth) {
    if(v[index].size() == 0) {
        if(maxdepth == depth)
            maxnum++;
        if(maxdepth < depth) {
            maxdepth = depth;
            maxnum = 1;
        }
        return ;
    }
    for(int i = 0; i < v[index].size(); i++)
        dfs(v[index][i], depth + 1);
}
int main() {
    double p, r;
    scanf("%d %lf %lf", &n, &p, &r);
    for(int i = 0; i < n; i++) {
        scanf("%d", &temp);
        if(temp == -1)
            root = i;
        else
            v[temp].push_back(i);
    }
    dfs(root, 0);
    printf("%.2f %d", p * pow(1 + r/100, maxdepth), maxnum);
    return 0;
}
```

1091. Acute Stroke (30) [广度优先搜索BFS]

One important factor to identify acute stroke (急性脑卒中) is the volume of the stroke core. Given the results of image analysis in which the core regions are identified in each MRI slice, your job is to calculate the volume of the stroke core.

Input Specification:

Each input file contains one test case. For each case, the first line contains 4 positive integers: M, N, L and T, where M and N are the sizes of each slice (i.e. pixels of a slice are in an M by N matrix, and the maximum resolution is 1286 by 128); L (<=60) is the number of slices of a brain; and T is the integer threshold (i.e. if the volume of a connected core is less than T, then that core must not be counted).

Then L slices are given. Each slice is represented by an M by N matrix of 0's and 1's, where 1 represents a pixel of stroke, and 0 means normal. Since the thickness of a slice is a constant, we only have to count the number of 1's to obtain the volume. However, there might be several separated core regions in a brain, and only those with their volumes no less than T are counted. Two pixels are "connected" and hence belong to the same region if they share a common side, as shown by Figure 1 where all the 6 red pixels are connected to the blue one.

Figure 1

Output Specification:

For each case, output in a line the total volume of the stroke core.

Sample Input:

3 4 5 2

1 1 1 1

1 1 1 1

1 1 1 1

0 0 1 1

0 0 1 1

0 0 1 1

1 0 1 1

0 1 0 0

0 0 0 0

1 0 1 1

0 0 0 0

0 0 0 0

0 0 0 1

0 0 0 1

1 0 0 0

Sample Output:

26

题目大意：给定一个三维数组，0表示正常1表示有肿瘤，肿瘤块的大小大于等于t才算作是肿瘤，让计算所有满足肿瘤块的大小

分析：三维的广度优先搜索～XYZ三个数组判断方向，对每一个点广度优先累计肿瘤块的大小，如果大于等于t就把结果累加。用visit数组标记当前的点有没有被访问过，被访问过的结点是不会再访问的。。judge判断是否超过了边界，或者是否当前结点为0不是肿瘤～

```
#include <cstdio>
#include <queue>
using namespace std;
struct node {
    int x, y, z;
};
int m, n, l, t;
int X[6] = {1, 0, 0, -1, 0, 0};
```

```
int Y[6] = {0, 1, 0, 0, -1, 0};
int Z[6] = {0, 0, 1, 0, 0, -1};
int arr[1300][130][80];
bool visit[1300][130][80];
bool judge(int x, int y, int z) {
    if(x < 0 || x >= m || y < 0 || y >= n || z < 0 || z >= l) return false;
    if(arr[x][y][z] == 0 || visit[x][y][z] == true) return false;
    return true;
}
int bfs(int x, int y, int z) {
    int cnt = 0;
    node temp;
    temp.x = x, temp.y = y, temp.z = z;
    queue<node> q;
    q.push(temp);
    visit[x][y][z] = true;
    while(!q.empty()) {
        node top = q.front();
        q.pop();
        cnt++;
        for(int i = 0; i < 6; i++) {
            int tx = top.x + X[i];
            int ty = top.y + Y[i];
            int tz = top.z + Z[i];
            if(judge(tx, ty, tz)) {
                visit[tx][ty][tz] = true;
                temp.x = tx, temp.y = ty, temp.z = tz;
                q.push(temp);
            }
        }
    }
    if(cnt >= t)
        return cnt;
    else
        return 0;

}
int main() {
    scanf("%d %d %d %d", &m, &n, &l, &t);
    for(int i = 0; i < l; i++)
        for(int j = 0; j < m; j++)
            for(int k = 0; k < n; k++)
                scanf("%d", &arr[j][k][i]);
    int ans = 0;
    for(int i = 0; i < l; i++) {
        for(int j = 0; j < m; j++) {
            for(int k = 0; k < n; k++) {
                if(arr[j][k][i] == 1 && visit[j][k][i] == false)
                    ans += bfs(j, k, i);
            }
        }
    }
```

```
61        printf("%d", ans);
62        return 0;
63    }
```

1092. To Buy or Not to Buy (20) [Hash散列]

Eva would like to make a string of beads with her favorite colors so she went to a small shop to buy some beads. There were many colorful strings of beads. However the owner of the shop would only sell the strings in whole pieces. Hence Eva must check whether a string in the shop contains all the beads she needs. She now comes to you for help: if the answer is “Yes”, please tell her the number of extra beads she has to buy; or if the answer is “No”, please tell her the number of beads missing from the string.

Figure 1

Output Specification:

For each test case, print your answer in one line. If the answer is “Yes”, then also output the number of extra beads Eva has to buy; or if the answer is “No”, then also output the number of beads missing from the string. There must be exactly 1 space between the answer and the number.

Sample Input 1:

ppRYYGrrYBR2258

YrR8RrY

Sample Output 1:

Yes 8

Sample Input 2:

ppRYYGrrYB225

YrR8RrY

Sample Output 1:

No 2

题目大意：小红想买些珠子做一串自己喜欢的珠串。卖珠子的摊主有很多串五颜六色的珠串，但是不肯把任何一串拆散了卖。于是小红要你帮忙判断一下，某串珠子里是否包含了全部自己想要的珠子？如果是，那么告诉她有多少多余的珠子；如果不是，那么告诉她缺了多少珠子～

分析：字符串a和b分别存储摊主的珠串和小红想做的珠串，遍历字符串a，将每一个字符出现的次数保存在book数组中，表示摊主的每个珠子的个数，遍历字符串b，如果book[b[i]]>0，表示小红要的珠子摊主有，则book[b[i]]-1，将这个珠子给小红～否则说明小红要的珠子摊主没有，则将统计缺了多少珠子的result++，如果result不等于0，说明缺珠子，则不可以买，输出No以及缺了的珠子个数result，否则说明不缺珠子，可以买，输出Yes以及摊主珠子多余的个数a.length() – b.length()～

```
#include <iostream>
using namespace std;
int book[256];
int main() {
    string a, b;
    cin >> a >> b;
    for (int i = 0; i < a.length(); i++)
        book[a[i]]++;
    int result = 0;
    for (int i = 0; i < b.length(); i++) {
        if (book[b[i]] > 0)
            book[b[i]]--;
        else
            result++;
    }
    if(result != 0)
        printf("No %d", result);
    else
        printf("Yes %d", a.length() - b.length());
    return 0;
}
```

1093. Count PAT's (25) [逻辑题]

The string APPAPT contains two PAT's as substrings. The first one is formed by the 2nd, the 4th, and the 6th characters, and the second one is formed by the 3rd, the 4th, and the 6th characters.

Now given any string, you are supposed to tell the number of PAT's contained in the string.

Input Specification:

Each input file contains one test case. For each case, there is only one line giving a string of no more than 105 characters containing only P, A, or T.

Output Specification:

For each test case, print in one line the number of PAT's contained in the string. Since the result may be a huge number, you only have to output the result moded by 1000000007.

Sample Input:

APPAPT

Sample Output:

2

分析：要想知道构成多少个PAT，那么遍历字符串后对于每一A，它前面的P的个数和它后面的T的个数的乘积就是能构成的PAT的个数。然后把对于每一个A的结果相加即可～辣么就简单啦，只需要先遍历字符串数一数有多少个T～然后每遇到一个T呢～countt–;每遇到一个P呢，countp++;然后一遇到字母A呢就countt * countp～～把这个结果累加到result中～～最后输出结果就好啦～对了别忘记要对10000000007取余哦～～

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s;
    cin >> s;
    int len = s.length(), result = 0, countp = 0, countt = 0;
    for (int i = 0; i < len; i++) {
        if (s[i] == 'T')
            countt++;
    }
    for (int i = 0; i < len; i++) {
        if (s[i] == 'P') countp++;
        if (s[i] == 'T') countt--;
        if (s[i] == 'A') result = (result + (countp * countt) % 1000000007) %
1000000007;
    }
    cout << result;
    return 0;
}
```

1094. The Largest Generation (25) [BFS，DFS，树的遍历]

A family hierarchy is usually presented by a pedigree tree where all the nodes on the same level belong to the same generation. Your task is to find the generation with the largest population.

Input Specification:

Each input file contains one test case. Each case starts with two positive integers N (<100) which is the total number of family members in the tree (and hence assume that all the members are numbered from 01 to N), and M (<N) which is the number of family members who have children. Then M lines follow, each contains the information of a family member in the following format:

ID K ID[1] ID[2] … ID[K]

where ID is a two-digit number representing a family member, K (>0) is the number of his/her children, followed by a sequence of two-digit ID's of his/her children. For the sake of simplicity, let us fix the root ID to be 01. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the largest population number and the level of the corresponding generation. It is assumed that such a generation is unique, and the root level is defined to be 1.

Sample Input:

23 13

21 1 23

01 4 03 02 04 05

03 3 06 07 08

06 2 12 13

13 1 21

08 2 15 16

02 2 09 10

11 2 19 20

17 1 22

05 1 11

07 1 14

09 1 17

10 1 18

Sample Output:

9 4

题目大意：输入树的结点个数N，结点编号为1~N，非叶子结点个数M，然后输出M个非叶子结点格子的孩子结点的编号，求结点个数最多的一层，根结点的层号为1，输出该层的结点个数以及层号
<(▰˘◡˘▰)>

分析：用DFS或者BFS，用DFS就用参数index和level标记当前遍历的结点的编号和层数，一个数组book标记当前层数level所含结点数，最后遍历一遍数组找出最大值。注意： book[level]++;这句话是发生在return语句判断之前的外面，即每遇到一个结点都要进行处理，而不是放在return语句的条件判断里面～～

如果是BFS，就用一个数组level[i]标记i结点所处的层数，它等于它的父亲结点的level的值+1，用一个数组book，book[i]标记i层所拥有的结点数，在遍历的时候每弹出一个结点就将当前结点的层数所对应的book值+1，最后遍历一遍book数组找出最大拥有的结点数和层数～

```
#include <cstdio>
#include <vector>
using namespace std;
vector<int> v[100];
int book[100];
void dfs(int index, int level) {
    book[level]++;
    for(int i = 0; i < v[index].size(); i++)
        dfs(v[index][i], level+1);
}
int main() {
    int n, m, a, k, c;
```

```
13        scanf("%d %d", &n, &m);
14        for(int i = 0; i < m; i++) {
15            scanf("%d %d",&a, &k);
16            for(int j = 0; j < k; j++) {
17                scanf("%d", &c);
18                v[a].push_back(c);
19            }
20        }
21        dfs(1, 1);
22        int maxnum = 0, maxlevel = 1;
23        for(int i = 1; i < 100; i++) {
24            if(book[i] > maxnum) {
25                maxnum = book[i];
26                maxlevel = i;
27            }
28        }
29        printf("%d %d", maxnum, maxlevel);
30        return 0;
31    }
```

1095. Cars on Campus (30) [map的用法， 排序]

Zhejiang University has 6 campuses and a lot of gates. From each gate we can collect the in/out times and the plate numbers of the cars crossing the gate. Now with all the information available, you are supposed to tell, at any specific time point, the number of cars parking on campus, and at the end of the day find the cars that have parked for the longest time period.

Input Specification:

Each input file contains one test case. Each case starts with two positive integers N (<= 10000), the number of records, and K (<= 80000) the number of queries. Then N lines follow, each gives a record in the format

plate_number hh:mm:ss status

where plate_number is a string of 7 English capital letters or 1-digit numbers; hh:mm:ss represents the time point in a day by hour:minute:second, with the earliest time being 00:00:00 and the latest 23:59:59; and status is either in or out.

Note that all times will be within a single day. Each “in” record is paired with the chronologically next record for the same car provided it is an “out” record. Any “in” records that are not paired with an “out” record are ignored, as are “out” records not paired with an “in” record. It is guaranteed that at least one car is well paired in the input, and no car is both “in” and “out” at the same moment. Times are recorded using a 24-hour clock.

Then K lines of queries follow, each gives a time point in the format hh:mm:ss. Note: the queries are given in ascending order of the times.

Output Specification:

For each query, output in a line the total number of cars parking on campus. The last line of output is supposed to give the plate number of the car that has parked for the longest time period, and the corresponding time length. If such a car is not unique, then output all of their plate numbers in a line in alphabetical order, separated by a space.

Sample Input:

16 7

JH007BD 18:00:01 in

ZD00001 11:30:08 out

DB8888A 13:00:00 out

ZA3Q625 23:59:50 out

ZA133CH 10:23:00 in

ZD00001 04:09:59 in

JH007BD 05:09:59 in

ZA3Q625 11:42:01 out

JH007BD 05:10:33 in

ZA3Q625 06:30:50 in

JH007BD 12:23:42 out

ZA3Q625 23:55:00 in

JH007BD 12:24:23 out

ZA133CH 17:11:22 out

JH007BD 18:07:01 out

DB8888A 06:30:50 in

05:10:00

06:30:50

11:00:00

12:23:42

14:00:00

18:00:00

23:59:00

Sample Output:

1

4

5

2

1

0

1

JH007BD ZD00001 07:20:09

题目大意：给出n个车牌号、时间点、进出状态的记录，然后查询k个时间点这时校园内的车辆个数。最后还要输出在校园里面呆的时间最长的车的车牌号，以及呆了多久的时间。如果有多辆车就按照它的字母从小到大输出车牌。

配对要求是，如果一个车多次进入未出，取最后一个值；如果一个车多次out未进入，取第一个值。

注意：一个车可能出入校园好多次，停车的时间应该取之和。

分析：为了简便，应该把小时和分钟都化简成秒数计算比较方便。

一开始所有车辆的id、时间和是进还是出（进的flag是1，出的flag是-1），对他们排序，先按照车牌号排序，再按照来的时间先后排序。

此后就能根据这样的排序后的顺序将所有满足条件（合法）的车辆进出记录保存到另一个数组里面。这个数组再按照时间先后排序。

因为多次询问值，为了避免超时，可以把他们的车辆数cnt数组先算出来。到时候直接取值就会比较快速。cnt[i]表示在i下标的记录的时间点的时候车辆的数量。数量可以由前一个数量+当前车辆的flag得到。

因为问询的时候是多个时间点按照从小到大的顺序，利用好这点能避免超时。如果上一个查询的index已经被记住，那么下一次就只需要从这个index开始找就可以了，避免重复寻找，浪费时间。

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstring>
#include <string>
#include <map>
using namespace std;
struct node {
    char id[10];
    int time, flag = 0;
};
bool cmp1(node a, node b) {
    if(strcmp(a.id, b.id) != 0)
        return strcmp(a.id, b.id) < 0;
    else
        return a.time < b.time;
}
bool cmp2(node a, node b) {
    return a.time < b.time;
}
```

```cpp
int main() {
    int n, k, maxtime = -1, tempindex = 0;
    scanf("%d%d\n", &n, &k);
    vector<node> record(n), car;
    for(int i = 0; i < n; i++) {
        char temp[5];
        int h, m, s;
        scanf("%s %d:%d:%d %s\n", record[i].id, &h, &m, &s, temp);
        int temptime = h * 3600 + m * 60 + s;
        record[i].time = temptime;
        record[i].flag = strcmp(temp, "in") == 0 ? 1 : -1;
    }
    sort(record.begin(), record.end(), cmp1);
    map<string, int> mapp;
    for(int i = 0; i < n - 1; i++) {
        if(strcmp(record[i].id, record[i+1].id) == 0 && record[i].flag == 1 &&
record[i+1].flag == -1) {
            car.push_back(record[i]);
            car.push_back(record[i+1]);
            mapp[record[i].id] += (record[i+1].time - record[i].time);
            if(maxtime < mapp[record[i].id]) {
                maxtime = mapp[record[i].id];
            }
        }
    }
    sort(car.begin(), car.end(), cmp2);
    vector<int> cnt(n);
    for(int i = 0; i < car.size(); i++) {
        if(i == 0)
            cnt[i] += car[i].flag;
         else
            cnt[i] = cnt[i - 1] + car[i].flag;
    }
    for(int i = 0; i < k; i++) {
        int h, m, s;
        scanf("%d:%d:%d", &h, &m, &s);
        int temptime = h * 3600 + m * 60 + s;
        int j;
        for(j = tempindex; j < car.size(); j++) {
            if(car[j].time > temptime) {
                printf("%d\n", cnt[j - 1]);
                break;
            } else if(j == car.size() - 1) {
                printf("%d\n", cnt[j]);
            }
        }
        tempindex = j;
    }
    for(map<string, int>::iterator it = mapp.begin(); it != mapp.end(); it++) {
        if(it->second == maxtime)
            printf("%s ", it->first.c_str());
    }
```

```
72      printf("%02d:%02d:%02d", maxtime / 3600, (maxtime % 3600) / 60, maxtime %
    60);
73      return 0;
74  }
```

1096. Consecutive Factors (20) [逻辑题]

Among all the factors of a positive integer N, there may exist several consecutive numbers. For example, 630 can be factored as 3*5*6*7, where 5, 6, and 7 are the three consecutive numbers. Now given any positive N, you are supposed to find the maximum number of consecutive factors, and list the smallest sequence of the consecutive factors.

Input Specification:

Each input file contains one test case, which gives the integer N (1<N<231).

Output Specification:

For each test case, print in the first line the maximum number of consecutive factors. Then in the second line, print the smallest sequence of the consecutive factors in the format “factor[1]*factor[2]*…*factor[k]”, where the factors are listed in increasing order, and 1 is NOT included.

Sample Input:

630

Sample Output:

3

5*6*7

题目大意：一个正整数N的因子中可能存在若干连续的数字。例如630可以分解为3*5*6*7，其中5、6、7就是3个连续的数字。给定任一正整数N，要求编写程序求出最长连续因子的个数，并输出最小的连续因子序列。

[Update v2.0] 由github用户littlesevenmo提供的更高效的解法：

不用算连续因子最多不会超过12个，也不需要三重循环，两重循环即可，直接去计算当前部分乘积能不能整除N

分析：1，如果只有一个因子，那么这个数只能为1或者质数。因此我们主要去计算两个及以上因数的情况。 2，在有两个及以上的数连乘中，因数的最大上限为sqrt(N)，即N的平方根。 3，因此思路就是，不断构造连乘，看连乘的积是否是N的因数，如果是，则看这部分连乘的数的个数是否比已记录的多。 4，用变量first记录连乘的第一个数字，这里我把它赋初值为0，如果在寻找N的因数过程中，first没有改变，那么就表明N是1或者是一个质数～

```
1   #include <iostream>
2   #include <cmath>
3   using namespace std;
4   long int num, temp;
5   int main(){
6       cin >> num;
7       int first = 0, len = 0, maxn = sqrt(num);
8       for (int i = 2; i <= maxn; i++){
```

```
            int j; temp = 1;
            for (j = i; j <= maxn; j++){
                temp *= j;
                if (num % temp != 0) break;
            }
            if (j - i > len){
                len = j - i;
                first = i;
            }
        }
        if (first == 0) cout << 1 << endl << num;
        else {
            cout << len << endl;
            for (int i = 0; i < len; i++){
                cout << first + i;
                if (i != len - 1) cout << '*';
            }
        }
        return 0;
    }
```

1097. Deduplication on a Linked List (25) [链表]

Given a singly linked list L with integer keys, you are supposed to remove the nodes with duplicated absolute values of the keys. That is, for each value K, only the first node of which the value or absolute value of its key equals K will be kept. At the mean time, all the removed nodes must be kept in a separate list. For example, given L being 21→-15→-15→-7→15, you must output 21→-15→-7, and the removed list -15→15.

Input Specification:

Each input file contains one test case. For each case, the first line contains the address of the first node, and a positive N (<= 105) which is the total number of nodes. The address of a node is a 5-digit nonnegative integer, and NULL is represented by -1.

Then N lines follow, each describes a node in the format:

Address Key Next

where Address is the position of the node, Key is an integer of which absolute value is no more than 104, and Next is the position of the next node.

Output Specification:

For each case, output the resulting linked list first, then the removed list. Each node occupies a line, and is printed in the same format as in the input.

Sample Input:

00100 5

99999 -7 87654

23854 -15 00000

87654 15 -1

00000 -15 99999

00100 21 23854

Sample Output:

00100 21 23854

23854 -15 99999

99999 -7 -1

00000 -15 87654

87654 15 -1

题目大意：给一个链表，去重（去掉值或者绝对值相等的），先输出删除后的链表，再输出删除了的链表。

分析：用结构体数组存储这个链表，大小为maxn = 100000，node[i]表示地址为i的结点。在结构体中定义一个num变量，将num变量先初始化为2 * maxn。通过改变num变量的值最后sort排序来改变链表的顺序。

将没有删除的结点的num标记为cnt1，cnt1为当前没有删除的结点的个数；将需要删除的结点的num标记为maxn + cnt2，cnt2表示当前删除了的结点的个数，因为一开始初始化为了2 * maxn，所以我们可以通过对num排序达到：num = 0~maxn为不删除结点，num = maxn~2maxn为删除结点，num = 2maxn为无效结点

这样sort后就会按照需要输出的顺序将结点排序，我们只需要输出前cnt1+cnt2个结点即可～～

```
#include <cstdio>
#include <stdlib.h>
#include <algorithm>
using namespace std;
const int maxn = 100000;
struct NODE {
    int address, key, next, num = 2 * maxn;
}node[maxn];
bool exist[maxn];
int cmp1(NODE a, NODE b){
    return a.num < b.num;
}
int main() {
    int begin, n, cnt1 = 0, cnt2 = 0, a;
    scanf("%d%d", &begin, &n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &a);
        scanf("%d%d", &node[a].key, &node[a].next);
        node[a].address = a;
    }
    for(int i = begin; i != -1; i = node[i].next) {
        if(exist[abs(node[i].key)] == false) {
            exist[abs(node[i].key)] = true;
```

```
            node[i].num = cnt1;
            cnt1++;
        }
        else {
            node[i].num = maxn + cnt2;
            cnt2++;
        }
    }
    sort(node, node + maxn, cmp1);
    int cnt = cnt1 + cnt2;
    for(int i = 0; i < cnt; i++) {
        if(i != cnt1 - 1 && i != cnt - 1) {
            printf("%05d %d %05d\n", node[i].address, node[i].key,
node[i+1].address);
        } else {
            printf("%05d %d -1\n", node[i].address, node[i].key);
        }
    }
    return 0;
}
```

1098. Insertion or Heap Sort (25) [heap sort（堆排序）]

According to Wikipedia:

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Heap sort divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. it involves the use of a heap data structure rather than a linear-time search to find the maximum.

Now given the initial sequence of integers, together with a sequence which is a result of several iterations of some sorting method, can you tell which sorting method we are using?

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=100). Then in the next line, N integers are given as the initial sequence. The last line contains the partially sorted sequence of the N numbers. It is assumed that the target sequence is always ascending. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in the first line either “Insertion Sort” or “Heap Sort” to indicate the method used to obtain the partial result. Then run this method for one more iteration and output in the second line the resuling sequence. It is guaranteed that the answer is unique for each test case. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

Sample Input 1:

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

Sample Output 1:

Insertion Sort

1 2 3 5 7 8 9 4 6 0

Sample Input 2:

10

3 1 2 8 7 5 9 4 6 0

6 4 5 1 0 3 2 7 8 9

Sample Output 2:

Heap Sort

5 4 3 1 0 2 6 7 8 9

题目大意：给出n和n个数的序列a和b，a为原始序列，b为排序其中的一个步骤，问b是a经过了堆排序还是插入排序的，并且输出它的下一步～

分析：插入排序的特点是：b数组前面的顺序是从小到大的，后面的顺序不一定，但是一定和原序列的后面的顺序相同～所以只要遍历一下前面几位，遇到不是从小到大的时候，开始看b和a是不是对应位置的值相等，相等就说明是插入排序，否则就是堆排序啦～

插入排序的下一步就是把第一个不符合从小到大的顺序的那个元素插入到前面已排序的里面的合适的位置，那么只要对前几个已排序的+后面一位这个序列sort排序即可～while(p <= n && b[p – 1] <= b[p]) p++；int index = p；找到第一个不满足条件的下标p并且赋值给index，b数组下标从1开始，所以插入排序的下一步就是sort(b.begin() + 1, b.begin() + index + 1)后的b数组～

堆排序的特点是后面是从小到大的，前面的顺序不一定，又因为是从小到大排列，堆排序之前堆为大顶堆，前面未排序的序列的最大值为b[1]，那么就可以从n开始往前找，找第一个小于等于b[1]的数字b[p]（while(p > 2 && b[p] >= b[1]) p–；），把它和第一个数字交换（swap(b[1], b[p])；），然后把数组b在1~p-1区间进行一次向下调整（downAdjust(b, 1，p – 1)；）～向下调整，low和high是需要调整的区间，因为是大顶堆，就是不断比较当前结点和自己的孩子结点哪个大，如果孩子大就把孩子结点和自己交换，然后再不断调整直到到达区间的最大值不能再继续了为止～

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
void downAdjust(vector<int> &b, int low, int high) {
    int i = 1, j = i * 2;
    while(j <= high) {
        if(j + 1 <= high && b[j] < b[j + 1]) j = j + 1;
        if (b[i] >= b[j]) break;
        swap(b[i], b[j]);
        i = j; j = i * 2;
    }
```

```
}
int main() {
    int n, p = 2;
    scanf("%d", &n);
    vector<int> a(n + 1), b(n + 1);
    for(int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for(int i = 1; i <= n; i++) scanf("%d", &b[i]);
    while(p <= n && b[p - 1] <= b[p]) p++;
    int index = p;
    while(p <= n && a[p] == b[p]) p++;
    if(p == n + 1) {
        printf("Insertion Sort\n");
        sort(b.begin() + 1, b.begin() + index + 1);
    } else {
        printf("Heap Sort\n");
        p = n;
        while(p > 2 && b[p] >= b[1]) p--;
        swap(b[1], b[p]);
        downAdjust(b, 1, p - 1);
    }
    printf("%d", b[1]);
    for(int i = 2; i <= n; i++)
        printf(" %d", b[i]);
    return 0;
}
```

1099. Build A Binary Search Tree (30) [二叉查找树BST]

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

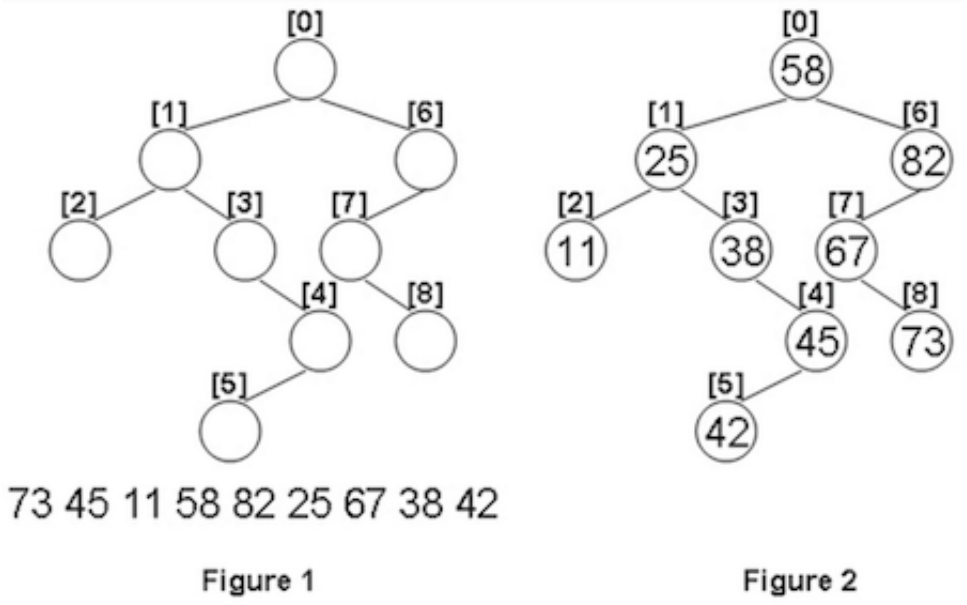


Figure 1 Figure 2

Each input file contains one test case. For each case, the first line gives a positive integer N (<=100) which is the total number of nodes in the tree. The next N lines each contains the left and the right children of a node in the format “left_index right_index”, provided that the nodes are numbered from 0 to N-1, and 0 is always the root. If one child is missing, then -1 will represent the NULL child pointer. Finally N distinct integer keys are given in the last line.

Output Specification:

For each test case, print in one line the level order traversal sequence of that tree. All the numbers must be separated by a space, with no extra space at the end of the line.

Sample Input:

9

1 6

2 3

-1 -1

-1 4

5 -1

-1 -1

7 -1

-1 8

-1 -1

73 45 11 58 82 25 67 38 42

Sample Output:

58 25 82 11 38 67 45 73 42

题目大意：给出一棵二叉搜索树（给出每个结点的左右孩子），且已知根结点为0，求并且给出应该插入这个二叉搜索树的数值，求这棵二叉树的层序遍历

分析：1. 用结构体data，left，right，index，level表示这棵树的结构，a数组存树的信息，b数组存这棵树节点的所有data，根据输入可知树a[i]的left和right～

2. 因为是二叉搜索树，所以中序遍历这棵树得到的结点顺序应该是给出的数值序列从小到大的排列顺序，所以把数值序列排序后，可以在中序遍历的时候直接赋值当前tree[root].data～
3. 然后根据节点的层数和下标，就可以排序输出层序了～

```
#include <iostream>
#include <algorithm>
using namespace std;
int n, cnt, b[100];
struct node {
    int data, l, r, index, lebel;
}a[110];
bool cmp(node x, node y) {
    if (x.lebel != y.lebel) return x.lebel < y.lebel;
    return x.index < y.index;
}
void dfs(int root, int index, int lebel) {
    if (a[root].l == -1 && a[root].r == -1) {
        a[root] = {b[cnt++], a[root].l, a[root].r, index, lebel};
```

```
        } else {
            if (a[root].l != -1) dfs(a[root].l, index * 2 + 1, lebel + 1);
            a[root] = {b[cnt++], a[root].l, a[root].r, index, lebel};
            if (a[root].r != -1) dfs(a[root].r, index * 2 + 2, lebel + 1);
        }
    }
    int main() {
        cin >> n;
        for (int i = 0; i < n; i++)
            cin >> a[i].l >> a[i].r;
        for (int i = 0; i < n; i++)
            cin >> b[i];
        sort(b, b + n);
        dfs(0, 0, 0);
        sort(a, a + n, cmp);
        for (int i = 0; i < n; i++) {
            if (i != 0) cout << " ";
            cout << a[i].data;
        }
        return 0;
    }
```

1100. Mars Numbers (20) [map映射，STL的使用]

People on Mars count their numbers with base 13:

Zero on Earth is called “tret” on Mars.

The numbers 1 to 12 on Earch is called “jan, feb, mar, apr, may, jun, jly, aug, sep, oct, nov, dec” on Mars, respectively.

For the next higher digit, Mars people name the 12 numbers as “tam, hel, maa, huh, tou, kes, hei, elo, syy, lok, mer, jou”, respectively.

For examples, the number 29 on Earth is called “hel mar” on Mars; and “elo nov” on Mars corresponds to 115 on Earth. In order to help communication between people from these two planets, you are supposed to write a program for mutual translation between Earth and Mars number systems.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (< 100). Then N lines follow, each contains a number in [0, 169), given either in the form of an Earth number, or that of Mars.

Output Specification:

For each number, print in a line the corresponding number in the other language.

Sample Input:

4

29

5

elo nov

tam

Sample Output:

hel mar

may

115

13

题目大意：火星人是以13进制计数的：地球人的0被火星人称为tret。地球人数字1到12的火星文分别为：jan, feb, mar, apr, may, jun, jly, aug, sep, oct, nov, dec。火星人将进位以后的12个高位数字分别称为：tam, hel, maa, huh, tou, kes, hei, elo, syy, lok, mer, jou。例如地球人的数字“29”翻译成火星文就是“hel mar”；而火星文“elo nov”对应地球数字“115”。为了方便交流，请你编写程序实现地球和火星数字之间的互译～

分析：因为给出的可能是数字（地球文）也有可能是字母（火星文），所以用字符串s保存每一次的输入，因为如果是火星文则会出现空格，所以用getline接收一行的输入～计算string s的长度len，判断s[0]是否是数字，如果是数字，表示是地球文，则需要转为火星文，执行func1()；如果不是数字，则说明是火星文，需要转为地球文，执行func2()；

func1(int t)中，传入的值是string转int后的结果stoi(s)，因为数字最大不超过168，所以最多只会输出两位火星文，如果t / 13不等于0，说明有高位，所以输出b[t/13]；如果输出了高位（t/13不等于0）并且t % 13不等于0，说明有高位且有低位，所以此时输出空格；如果t % 13不等于0，说明有低位，此时输出a[t % 13]；注意，还有个数字0没有考虑，因为数字0取余13等于0，但是要特别输出tret，所以在func1的最后一句判断中加一句t == 0，并将a[0]位赋值成tret即可解决0的问题～

func2()中，t1和t2一开始都赋值0，s1和s2用来分离火星文单词，因为火星文字符串只可能一个单词或者两个单词，而且一个单词不会超过4，所以先将一个单词的赋值给s1，即s1 = s.substr(0, 3)；如果len > 4，就将剩下的一个单词赋值给s2，即s2 = s.substr(4, 3)；然后下标j从1到12遍历a和b两个数组，如果a数组中有和s1或者s2相等的，说明低位等于j，则将j赋值给t2；如果b数组中有和s1相等的（b数组不会和s2相等，因为如果有两个单词，s2只可能是低位），说明高位有值，将j赋值给t1，最后输出t1 * 13 + t2即可～

```
#include <iostream>
#include <string>
using namespace std;
string a[13] = {"tret", "jan", "feb", "mar", "apr", "may", "jun", "jly", "aug", "sep", "oct", "nov", "dec"};
string b[13] = {"####", "tam", "hel", "maa", "huh", "tou", "kes", "hei", "elo", "syy", "lok", "mer", "jou"};
string s;
int len;
void func1(int t) {
    if (t / 13) cout << b[t / 13];
    if ((t / 13) && (t % 13)) cout << " ";
    if (t % 13 || t == 0) cout << a[t % 13];
}
```

```
void func2() {
    int t1 = 0, t2 = 0;
    string s1 = s.substr(0, 3), s2;
    if (len > 4) s2 = s.substr(4, 3);
    for (int j = 1; j <= 12; j++) {
        if (s1 == a[j] || s2 == a[j]) t2 = j;
        if (s1 == b[j]) t1 = j;
    }
    cout << t1 * 13 + t2;
}
int main() {
    int n;
    cin >> n;
    getchar();
    for (int i = 0; i < n; i++) {
        getline(cin, s);
        len = s.length();
        if (s[0] >= '0' && s[0] <= '9')
            func1(stoi(s));
        else
            func2();
        cout << endl;
    }
    return 0;
}
```

1101. Quick Sort (25) [快速排序]

There is a classical process named partition in the famous quick sort algorithm. In this process we typically choose one element as the pivot. Then the elements less than the pivot are moved to its left and those larger than the pivot to its right. Given N distinct positive integers after a run of partition, could you tell how many elements could be the selected pivot for this partition?

For example, given N = 5 and the numbers 1, 3, 2, 4, and 5. We have:

1 could be the pivot since there is no element to its left and all the elements to its right are larger than it;

3 must not be the pivot since although all the elements to its left are smaller, the number 2 to its right is less than it as well;

2 must not be the pivot since although all the elements to its right are larger, the number 3 to its left is larger than it as well;

and for the similar reason, 4 and 5 could also be the pivot.

Hence in total there are 3 pivot candidates.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<= 105). Then the next line contains N distinct positive integers no larger than 109. The numbers in a line are separated by spaces.

Output Specification:

For each test case, output in the first line the number of pivot candidates. Then in the next line print these candidates in increasing order. There must be exactly 1 space between two adjacent numbers, and no extra space at the end of each line.

Sample Input:

5

1 3 2 4 5

Sample Output:

3

1 4 5

题目大意：快速排序中，我们通常采用某种方法取一个元素作为主元，通过交换，把比主元小的元素放到它的左边，比主元大的元素放到它的右边。 给定划分后的N个互不相同的正整数的排列，请问有多少个元素可能是划分前选取的主元？例如给定N = 5, 排列是1、3、2、4、5。则：

1的左边没有元素，右边的元素都比它大，所以它可能是主元；

尽管3的左边元素都比它小，但是它右边的2它小，所以它不能是主元；

尽管2的右边元素都比它大，但其左边的3比它大，所以它不能是主元；

类似原因，4和5都可能是主元。

因此，有3个元素可能是主元。 给N个数，第一行输出可能是主元的个数，第二行输出这些元素～

分析：对原序列sort排序，逐个比较，当当前元素没有变化并且它左边的所有值的最大值都比它小的时候就可以认为它一定是主元（很容易证明正确性的，毕竟无论如何当前这个数要满足左边都比他大右边都比他小，那它的排名【当前数在序列中处在第几个】一定不会变）～

如果硬编码就直接运行超时了...后来才想到这种方法～

一开始有一个测试点段错误，后来才想到因为输出时候v[0]是非法内存，改正后发现格式错误（好像可以说明那个第2个测试点是0个主元？...）然后

加了最后一句printf(“\n”);才正确（难道是当没有主元的时候必须要输出空行吗...）

```
#include <iostream>
#include <algorithm>
#include <vector>
int v[100000];
using namespace std;
int main() {
    int n, max = 0, cnt = 0;
    scanf("%d", &n);
    vector<int> a(n), b(n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
        b[i] = a[i];
    }
    sort(a.begin(), a.end());
```

```
    for (int i = 0; i < n; i++) {
        if(a[i] == b[i] && b[i] > max)
            v[cnt++] = b[i];
        if (b[i] > max)
            max = b[i];
    }
    printf("%d\n", cnt);
    for(int i = 0; i < cnt; i++) {
        if (i != 0) printf(" ");
        printf("%d", v[i]);
    }
    printf("\n");
    return 0;
}
```

1102. Invert a Binary Tree (25) [树的遍历]

The following is from Max Howell @twitter:

Google: 90% of our engineers use the software you wrote (Homebrew), but you can't invert a binary tree on a whiteboard so fuck off.

Now it's your turn to prove that YOU CAN invert a binary tree!

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=10) which is the total number of nodes in the tree — and hence the nodes are numbered from 0 to N-1. Then N lines follow, each corresponds to a node from 0 to N-1, and gives the indices of the left and right children of the node. If the child does not exist, a "-" will be put at the position. Any pair of children are separated by a space.

Output Specification:

For each test case, print in the first line the level-order, and then in the second line the in-order traversal sequences of the inverted tree. There must be exactly one space between any adjacent numbers, and no extra space at the end of the line.

Sample Input:

8

1 –

– –

0 –

2 7

– –

– –

5 –

4 6

Sample Output:

3 7 2 6 4 0 5 1

6 5 7 4 3 2 0 1

题目大意：反转一棵二叉树，给出原二叉树的每个结点的左右孩子，输出它的层序和前序遍历～

分析：1. 反转二叉树就是存储的时候所有左右结点都交换。

2. 二叉树使用{id, l, r, index, level}存储每个结点的id, 左右结点,下标值，和当前层数～
3. 根结点是所有左右结点中没有出现的那个结点～
4. 已知根结点，用递归的方法可以把中序遍历的结果push_back到数组v1里面,直接输出就是中序，排序输出就是层序（排序方式，层数小的排前面，相同层数时，index大的排前面）

```
#include <vector>
#include <algorithm>
using namespace std;
struct node {
    int id, l, r, index, level;
} a[100];
vector<node> v1;
void dfs(int root, int index, int level) {
    if (a[root].r != -1) dfs(a[root].r, index * 2 + 2, level + 1);
    v1.push_back({root, 0, 0, index, level});
    if (a[root].l != -1) dfs(a[root].l, index * 2 + 1, level + 1);
}
bool cmp(node a, node b) {
    if (a.level != b.level) return a.level < b.level;
    return a.index > b.index;
}
int main() {
    int n, have[100] = {0}, root = 0;
    cin >> n;
    for (int i = 0; i < n; i++) {
        a[i].id = i;
        string l, r;
        cin >> l >> r;
        if (l != "-") {
            a[i].l = stoi(l);
            have[stoi(l)] = 1;
        } else {
            a[i].l = -1;
        }
        if (r != "-") {
            a[i].r = stoi(r);
            have[stoi(r)] = 1;
        } else {
            a[i].r = -1;
        }
    }
    while (have[root] == 1) root++;
```

```
    dfs(root, 0, 0);
    vector<node> v2(v1);
    sort(v2.begin(), v2.end(), cmp);
    for (int i = 0; i < v2.size(); i++) {
        if (i != 0) cout << " ";
        cout << v2[i].id;
    }
    cout << endl;
    for (int i = 0; i < v1.size(); i++) {
        if (i != 0) cout << " ";
        cout << v1[i].id;
    }
    return 0;
}
```

1103. Integer Factorization (30) [深度优先搜索DFS]

The K-P factorization of a positive integer N is to write N as the sum of the P-th power of K positive integers. You are supposed to write a program to find the K-P factorization of N for any positive integers N, K and P.

Input Specification:

Each input file contains one test case which gives in a line the three positive integers N (<=400), K (<=N) and P (1<P<=7). The numbers in a line are separated by a space.

Output Specification:

For each case, if the solution exists, output in the format:

N = n1^P + … nK^P

where ni (i=1, … K) is the i-th factor. All the factors must be printed in non-increasing order.

Note: the solution may not be unique. For example, the 5-2 factorization of 169 has 9 solutions, such as 122 + 42 + 22 + 22 + 12, or 112 + 62 + 22 + 22 + 22, or more. You must output the one with the maximum sum of the factors. If there is a tie, the largest factor sequence must be chosen — sequence { a1, a2, … aK } is said to be larger than { b1, b2, … bK } if there exists 1<=L<=K such that ai=bi for i<L and aL>bL

If there is no solution, simple output “Impossible”.

Sample Input 1:

169 5 2

Sample Output 1:

169 = 6^2 + 6^2 + 6^2 + 6^2 + 5^2

Sample Input 2:

169 167 3

Sample Output 2:

Impossible

题目大意：给三个正整数N、K、P，将N表示成K个正整数（可以相同，递减排列）的P次方和，如果有多种方案，选择底数n1+...+nk最大的方案，如果还有多种方案，选择底数序列的字典序最大的方案～

分析：dfs深度优先搜索。先把i从0开始所有的i的p次方的值存储在v[i]中，直到v[i] > n为止。然后深度优先搜索，记录当前正在相加的index（即v[i]的i的值），当前的总和tempSum，当前K的总个数tempK，以及因为题目中要求输出因子的和最大的那个，所以保存一个facSum为当前因子的和，让它和maxFacSum比较，如果比maxFacSum大就更新maxFacSum和要求的ans数组的值。

在ans数组里面存储因子的序列，tempAns为当前深度优先遍历而来的序列，从v[i]的最后一个index开始一直到index == 1，因为这样才能保证ans和tempAns数组里面保存的是从大到小的因子的顺序。一开始maxFacSum == -1，如果dfs后maxFacSum并没有被更新，还是-1，那么就输出Impossible，否则输出答案。

（PS：感谢github用户littlesevenmo提供的更优解）

分析：这道题考的是DFS+剪枝，我认为主要剪枝的地方有三个：

1. tempK==K但是tempSum!=n的时候需要剪枝
2. 在枚举的时候，按顺序枚举，上界或者下界可进行剪枝
3. 当且仅当tempSum + v[index] <= n时，进行下一层的DFS，而不要进入下一层DFS发现不满足条件再返回，这样开销会比较大～

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
int n, k, p, maxFacSum = -1;
vector<int> v, ans, tempAns;
void init() {
    int temp = 0, index = 1;
    while (temp <= n) {
        v.push_back(temp);
        temp = pow(index, p);
        index++;
    }
}
void dfs(int index, int tempSum, int tempK, int facSum) {
    if (tempK == k) {
        if (tempSum == n && facSum > maxFacSum) {
                ans = tempAns;
                maxFacSum = facSum;
        }
        return;
    }
    while(index >= 1) {
        if (tempSum + v[index] <= n) {
            tempAns[tempK] = index;
            dfs(index, tempSum + v[index], tempK + 1, facSum + index);
        }
        if (index == 1) return;
        index--;
```

```
        }
    }
    int main() {
        scanf("%d%d%d", &n, &k, &p);
        init();
        tempAns.resize(k);
        dfs(v.size() - 1, 0, 0, 0);
        if (maxFacSum == -1) {
            printf("Impossible");
            return 0;
        }
        printf("%d = ", n);
        for (int i = 0; i < ans.size(); i++) {
            if (i != 0) printf(" + ");
            printf("%d^%d", ans[i], p);
        }
        return 0;
    }
```

1104. Sum of Number Segments (20) [数学问题]

Given a sequence of positive numbers, a segment is defined to be a consecutive subsequence. For example, given the sequence {0.1, 0.2, 0.3, 0.4}, we have 10 segments: (0.1) (0.1, 0.2) (0.1, 0.2, 0.3) (0.1, 0.2, 0.3, 0.4) (0.2) (0.2, 0.3) (0.2, 0.3, 0.4) (0.3) (0.3, 0.4) (0.4).

Now given a sequence, you are supposed to find the sum of all the numbers in all the segments. For the previous example, the sum of all the 10 segments is 0.1 + 0.3 + 0.6 + 1.0 + 0.2 + 0.5 + 0.9 + 0.3 + 0.7 + 0.4 = 5.0.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N, the size of the sequence which is no more than 105. The next line contains N positive numbers in the sequence, each no more than 1.0, separated by a space.

Output Specification:

For each test case, print in one line the sum of all the numbers in all the segments, accurate up to 2 decimal places.

Sample Input:

4

0.1 0.2 0.3 0.4

Sample Output:

5.00

题目大意：给定一个正数数列，我们可以从中截取任意的连续的几个数，称为片段。例如，给定数列{0.1, 0.2, 0.3, 0.4}，我们有(0.1) (0.1, 0.2) (0.1, 0.2, 0.3) (0.1, 0.2, 0.3, 0.4) (0.2) (0.2, 0.3) (0.2, 0.3, 0.4) (0.3) (0.3, 0.4) (0.4) 这10个片段。给定正整数数列，求出全部片段包含的所有的数之和。如本例中10个片段总和是0.1 + 0.3 + 0.6 + 1.0 + 0.2 + 0.5 + 0.9 + 0.3 + 0.7 + 0.4 = 5.0，在一行中输出该序列所有片段包含的

数之和，精确到小数点后2位~

分析：将数列中的每个数字读取到temp中，假设我们选取的片段中包括temp，且这个片段的首尾指针分别为p和q，那么对于p，有i种选择，即12…i，对于q，有n-i-1种选择，即i, i+1, … n，所以p和q组合形成的首尾片段有i * (n-i-1)种，因为每个里面都会出现temp，所以temp引起的总和为temp * i * (n – i + 1)；遍历完所有数字，将每个temp引起的总和都累加到sum中，最后输出sum的值～

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cin >> n;
    double sum = 0.0, temp;
    for (int i = 1; i <= n; i++) {
        cin >> temp;
        sum = sum + temp * i * (n - i + 1);
    }
    printf("%.2f", sum);
    return 0;
}
```

1105. Spiral Matrix (25) [模拟]

This time your job is to fill a sequence of N positive integers into a spiral matrix in non-increasing order. A spiral matrix is filled in from the first element at the upper-left corner, then move in a clockwise spiral. The matrix has m rows and n columns, where m and n satisfy the following: m*n must be equal to N; m>=n; and m-n is the minimum of all the possible values.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N. Then the next line contains N positive integers to be filled into the spiral matrix. All the numbers are no more than 104. The numbers in a line are separated by spaces.

Output Specification:

For each test case, output the resulting matrix in m lines, each contains n numbers. There must be exactly 1 space between two adjacent numbers, and no extra space at the end of each line.

Sample Input:

12

37 76 20 98 76 42 53 95 60 81 58 93

Sample Output:

98 95 93

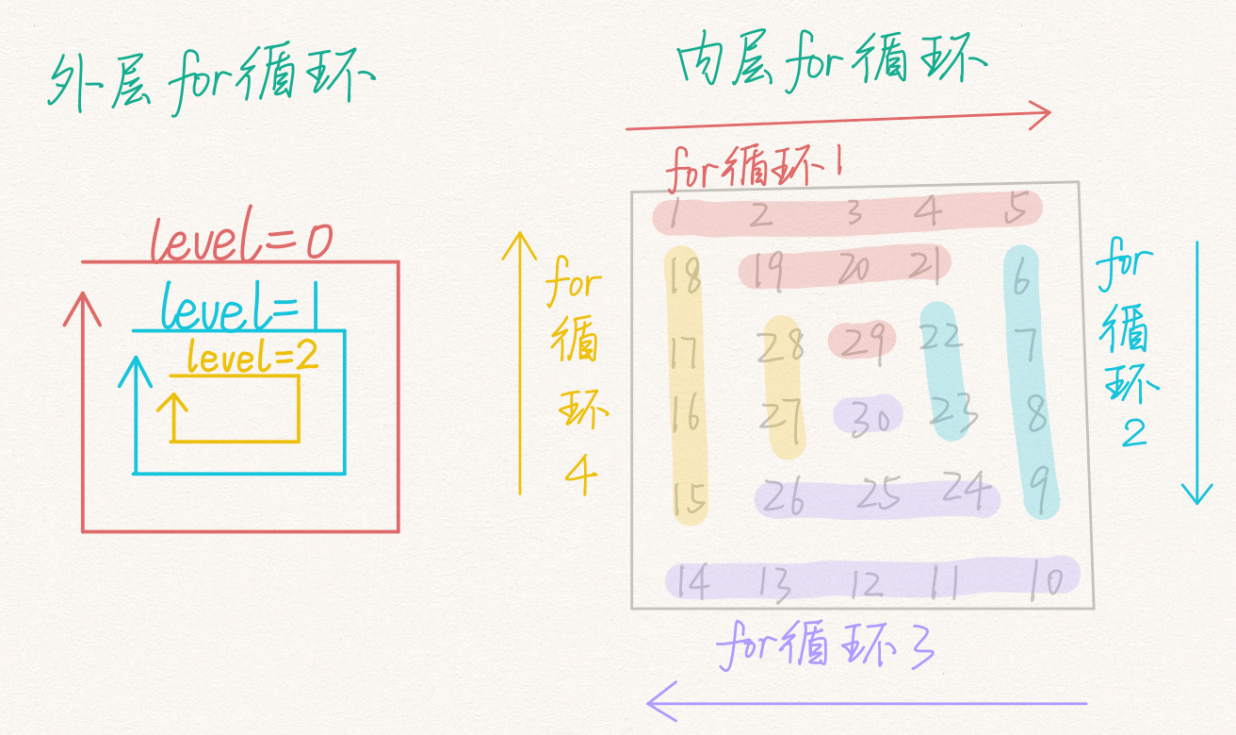
42 37 81

53 20 76

58 60 76

题目大意：将给定的N个正整数按非递增的顺序，填入“螺旋矩阵”~所谓“螺旋矩阵”，是指从左上角第1个格子开始，按顺时针螺旋方向填充~要求矩阵的规模为m行n列，满足条件：m*n等于N；m>=n；且m-n取所有可能值中的最小值~

分析：先计算行数m和列数n的值，n从根号N的整数部分开始，往前推一直到1，找到第一个满足N % n== 0的，m的值等于N/n～将N个给定的值输入数组a，并将a数组中的值按非递增排序，接着建立m行n列的数组b，填充时按层数填充，一个包裹矩阵的口字型为一层，计算螺旋矩阵的层数level，如果m的值为偶数，层数为m/2，如果m为奇数，层数为m/2+1，所以level = m / 2 + m % 2；因为是从左上角第1个格子开始，按顺时针螺旋方向填充，所以外层for循环控制层数i从0到level，内层for循环按左上到右上、右上到右下、右下到左下、左下到左上的顺序一层层填充，注意内层for循环中还要控制t <= N – 1，因为如果螺旋矩阵中所有的元素已经都填充完毕，就不能再重复填充～填充完毕后，输出整个矩阵～




```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <vector>
using namespace std;
int cmp(int a, int b) {return a > b;}
int main() {
    int N, m, n, t = 0;
    scanf("%d", &N);
    for (n = sqrt((double)N); n >= 1; n--) {
        if (N % n == 0) {
            m = N / n;
            break;
        }
    }
    vector<int> a(N);
    for (int i = 0; i < N; i++)
```

```
            scanf("%d", &a[i]);
        sort(a.begin(), a.end(), cmp);
        vector<vector<int> > b(m, vector<int>(n));
        int level = m / 2 + m % 2;
        for (int i = 0; i < level; i++) {
            for (int j = i; j <= n - 1 - i && t <= N - 1; j++)
                    b[i][j] = a[t++];
            for (int j = i + 1; j <= m - 2 - i && t <= N - 1; j++)
                    b[j][n - 1 - i] = a[t++];
            for (int j = n - i - 1; j >= i && t <= N - 1; j--)
                    b[m - 1 - i][j] = a[t++];
            for (int j = m - 2 - i; j >= i + 1 && t <= N - 1; j--)
                    b[j][i] = a[t++];
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0 ; j < n; j++) {
                printf("%d", b[i][j]);
                if (j != n - 1) printf(" ");
            }
            printf("\n");
        }
        return 0;
    }
```

1106. Lowest Price in Supply Chain (25) [DFS，BFS，树的遍历]

A supply chain is a network of retailers（零售商）, distributors（经销商）, and suppliers（供应商）– everyone involved in moving a product from supplier to customer.

Starting from one root supplier, everyone on the chain buys products from one's supplier in a price P and sell or distribute them in a price that is r% higher than P. Only the retailers will face the customers. It is assumed that each member in the supply chain has exactly one supplier except the root supplier, and there is no supply cycle.

Now given a supply chain, you are supposed to tell the lowest price a customer can expect from some retailers.

Input Specification:

Each input file contains one test case. For each case, The first line contains three positive numbers: N (<=105), the total number of the members in the supply chain (and hence their ID's are numbered from 0 to N-1, and the root supplier's ID is 0); P, the price given by the root supplier; and r, the percentage rate of price increment for each distributor or retailer. Then N lines follow, each describes a distributor or retailer in the following format:

Ki ID[1] ID[2] … ID[Ki]

where in the i-th line, Ki is the total number of distributors or retailers who receive products from supplier i, and is then followed by the ID's of these distributors or retailers. Kj being 0 means that the j-th member is a retailer. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the lowest price we can expect from some retailers, accurate up to 4 decimal places, and the number of retailers that sell at the lowest price. There must be one space between the two numbers. It is guaranteed that the all the prices will not exceed 1010.

Sample Input:

10 1.80 1.00

3 2 3 5

1 9

1 4

1 7

0

2 6 1

1 8

0

0

0

Sample Output:

1.8362 2

题目大意：提供一棵树，在树根处货物的价格为p，从根结点开始每往下一层，该层货物价格将会在父亲结点的价格上增加r%。求叶子结点出能获得的最低价格以及能提供最低价格的叶子结点数

分析：dfs。保存深度的最小值mindepth，以及最小值下该深度的个数minnum。深度优先搜索参数为index和depth，不断遍历index结点的孩子结点，直到当前结点没有孩子结点为止return～

```
#include <cstdio>
#include <vector>
#include <cmath>
using namespace std;
vector<int> v[100005];
int mindepth = 99999999, minnum = 1;
void dfs(int index, int depth) {
    if(mindepth < depth)
        return ;
    if(v[index].size() == 0) {
        if(mindepth == depth)
            minnum++;
        else if(mindepth > depth) {
            mindepth = depth;
            minnum = 1;
        }
    }
    for(int i = 0; i < v[index].size(); i++)
        dfs(v[index][i], depth + 1);
```

```
}
int main() {
    int n, k, c;
    double p, r;
    scanf("%d %lf %lf", &n, &p, &r);
    for(int i = 0; i < n; i++) {
        scanf("%d", &k);
        for(int j = 0; j < k; j++) {
            scanf("%d", &c);
            v[i].push_back(c);
        }
    }
    dfs(0, 0);
    printf("%.4f %d", p * pow(1 + r/100, mindepth), minnum);
    return 0;
}
```

1107. Social Clusters (30) [并查集]

When register on a social network, you are always asked to specify your hobbies in order to find some potential friends with the same hobbies. A “social cluster” is a set of people who have some of their hobbies in common. You are supposed to find all the clusters.

Input Specification:

Each input file contains one test case. For each test case, the first line contains a positive integer N (<=1000), the total number of people in a social network. Hence the people are numbered from 1 to N. Then N lines follow, each gives the hobby list of a person in the format:

Ki: hi[1] hi[2] … hi[Ki]

where Ki (>0) is the number of hobbies, and hi[j] is the index of the j-th hobby, which is an integer in [1, 1000].

Output Specification:

For each case, print in one line the total number of clusters in the network. Then in the second line, print the numbers of people in the clusters in non-increasing order. The numbers must be separated by exactly one space, and there must be no extra space at the end of the line.

Sample Input:

8

3: 2 7 10

1: 4

2: 5 3

1: 4

1: 3

1: 4

4: 6 8 1 5

1: 4

Sample Output:

3

4 3 1

题目大意：有n个人，每个人喜欢k个活动，如果两个人有任意一个活动相同，就称为他们处于同一个社交网络。求这n个人一共形成了多少个社交网络。

分析：并查集。先写好init、findFather、Union。

0. 每个社交圈的结点号是人的编号，而不是课程。课程是用来判断是否处在一个社交圈的。
1. course[t]表示任意一个喜欢t活动的人的编号。如果当前的课程t，之前并没有人喜欢过，那么就course[t] = i，i为它自己的编号，表示i为喜欢course[t]的一个人的编号
2. course[t]是喜欢t活动的人的编号，那么findFather(course[t])就是喜欢这个活动的人所处的社交圈子的根结点，合并根结点和当前人的编号的结点i。即Union(i, findFather(course[t]))，把它们处在同一个社交圈子里面
3. isRoot[i]表示编号i的人是不是它自己社交圈子的根结点，如果等于0表示不是根结点，如果不等于0，每次标记isRoot[findFather(i)]++，那么isRoot保存的就是如果当前是根结点，那么这个社交圈里面的总人数
4. isRoot中不为0的编号的个数cnt就是社交圈圈子的个数
5. 把isRoot从大到小排列，输出前cnt个，就是社交圈人数的从大到小的输出顺序

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> father, isRoot;
int cmp1(int a, int b){return a > b;}
int findFather(int x) {
    int a = x;
    while(x != father[x])
        x = father[x];
    while(a != father[a]) {
        int z = a;
        a = father[a];
        father[z] = x;
    }
    return x;
}
void Union(int a, int b) {
    int faA = findFather(a);
    int faB = findFather(b);
    if(faA != faB) father[faA] = faB;
}
int main() {
    int n, k, t, cnt = 0;
    int course[1001] = {0};
    scanf("%d", &n);
```

```
    father.resize(n + 1);
    isRoot.resize(n + 1);
    for(int i = 1; i <= n; i++)
        father[i] = i;
    for(int i = 1; i <= n; i++) {
        scanf("%d:", &k);
        for(int j = 0; j < k; j++) {
            scanf("%d", &t);
            if(course[t] == 0)
                course[t] = i;
            Union(i, findFather(course[t]));
        }
    }
    for(int i = 1; i <= n; i++)
        isRoot[findFather(i)]++;
    for(int i = 1; i <= n; i++) {
        if(isRoot[i] != 0) cnt++;
    }
    printf("%d\n", cnt);
    sort(isRoot.begin(), isRoot.end(), cmp1);
    for(int i = 0; i < cnt; i++) {
        printf("%d", isRoot[i]);
        if(i != cnt - 1) printf(" ");
    }
    return 0;
}
```

1108. Finding Average (20) [字符串处理]

The basic task is simple: given N real numbers, you are supposed to calculate their average. But what makes it complicated is that some of the input numbers might not be legal. A “legal” input is a real number in [-1000, 1000] and is accurate up to no more than 2 decimal places. When you calculate the average, those illegal numbers must not be counted in.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=100). Then N numbers are given in the next line, separated by one space.

Output Specification:

For each illegal input number, print in a line “ERROR: X is not a legal number” where X is the input. Then finally print in a line the result: “The average of K numbers is Y” where K is the number of legal inputs and Y is their average, accurate to 2 decimal places. In case the average cannot be calculated, output “Undefined” instead of Y. In case K is only 1, output “The average of 1 number is Y” instead.

Sample Input 1:

7

5 -3.2 aaa 9999 2.3.4 7.123 2.35

Sample Output 1:

ERROR: aaa is not a legal number

ERROR: 9999 is not a legal number

ERROR: 2.3.4 is not a legal number

ERROR: 7.123 is not a legal number

The average of 3 numbers is 1.38

Sample Input 2:

2

aaa -9999

Sample Output 2:

ERROR: aaa is not a legal number

ERROR: -9999 is not a legal number

The average of 0 numbers is Undefined

分析：用非常好用的sscanf和sprintf即可解决～ sscanf() – 从一个字符串中读进与指定格式相符的数据 sprintf() – 字符串格式化命令，主要功能是把格式化的数据写入某个字符串中

```
#include <iostream>
#include <cstdio>
#include <string.h>
using namespace std;
int main() {
    int n, cnt = 0;
    char a[50], b[50];
    double temp, sum = 0.0;
    cin >> n;
    for(int i = 0; i < n; i++) {
        scanf("%s", a);
        sscanf(a, "%lf", &temp);
        sprintf(b, "%.2f",temp);
        int flag = 0;
        for(int j = 0; j < strlen(a); j++)
            if(a[j] != b[j]) flag = 1;
        if(flag || temp < -1000 || temp > 1000) {
            printf("ERROR: %s is not a legal number\n", a);
            continue;
        } else {
            sum += temp;
            cnt++;
        }
    }
    if(cnt == 1)
        printf("The average of 1 number is %.2f", sum);
    else if(cnt > 1)
        printf("The average of %d numbers is %.2f", cnt, sum / cnt);
    else
```

```
30            printf("The average of 0 numbers is Undefined");
31        return 0;
32    }
```

1109. Group Photo (25) [逻辑题]

Formation is very important when taking a group photo. Given the rules of forming K rows with N people as the following:

The number of people in each row must be N/K (round down to the nearest integer), with all the extra people (if any) standing in the last row;

All the people in the rear row must be no shorter than anyone standing in the front rows;

In each row, the tallest one stands at the central position (which is defined to be the position (m/2+1), where m is the total number of people in that row, and the division result must be rounded down to the nearest integer);

In each row, other people must enter the row in non-increasing order of their heights, alternately taking their positions first to the right and then to the left of the tallest one (For example, given five people with their heights 190, 188, 186, 175, and 170, the final formation would be 175, 188, 190, 186, and 170. Here we assume that you are facing the group so your left-hand side is the right-hand side of the one at the central position.);

When there are many people having the same height, they must be ordered in alphabetical (increasing) order of their names, and it is guaranteed that there is no duplication of names.

Now given the information of a group of people, you are supposed to write a program to output their formation.

Input Specification:

Each input file contains one test case. For each test case, the first line contains two positive integers N (<=10000), the total number of people, and K (<=10), the total number of rows. Then N lines follow, each gives the name of a person (no more than 8 English letters without space) and his/her height (an integer in [30, 300]).

Output Specification:

For each case, print the formation — that is, print the names of people in K lines. The names must be separated by exactly one space, but there must be no extra space at the end of each line. Note: since you are facing the group, people in the rear rows must be printed above the people in the front rows.

Sample Input:

10 3

Tom 188

Mike 170

Eva 168

Tim 160

Joe 190

Ann 168

Bob 175

Nick 186

Amy 160

John 159

Sample Output:

Bob Tom Joe Nick

Ann Mike Eva

Tim Amy John

题目大意：拍集体照时队形很重要，这里对给定的N个人K排的队形设计排队规则如下：

每排人数为N/K（向下取整），多出来的人全部站在最后一排；后排所有人的个子都不比前排任何人矮；每排中最高者站中间（中间位置为m/2+1，其中m为该排人数，除法向下取整）；每排其他人以中间人为轴，按身高非增序，先右后左交替入队站在中间人的两侧（例如5人身高为190、188、186、175、170，则队形为175、188、190、186、170。这里假设你面对拍照者，所以你的左边是中间人的右边）；若多人身高相同，则按名字的字典序升序排列。这里保证无重名。现给定一组拍照人，请编写程序输出他们的队形。输出拍照的队形。即K排人名，其间以空格分隔，行末不得有多余空格。注意：假设你面对拍照者，后排的人输出在上方，前排输出在下方～

分析：建立结构体node，里面包含string类型的姓名name和int类型的身高height～将学生的信息输入到node类型的vector数组stu中～然后对stu数组进行排序（cmp函数表示排序规则，如果身高不等，就按照身高从大到小排列；如果身高相等，就按照名字从小到大的字典序排列～）然后用while循环排列每一行，将每一行应该排列的结果的姓名保存在ans数组中～

因为是面对拍照者，后排的人输出在上方，前排输出在下方，每排人数为N/K（向下取整），多出来的人全部站在最后一排，所以第一排输出的应该是包含多出来的人，所以while循环体中，当row == k时，表示当前是在排列第一行，那么这一行的人数m应该等于总人数n减去后面的k列*(k-1)行，即m = n – n / k * (k-1)；如果不是第一行，那么m直接等于n / k；最中间一个学生应该排在m/2的下标位置，即ans[m / 2] = stu[t].name；然后排左边一列，ans数组的下标 j 从m/2-1开始，一直往左j–，而对于stu的下标 i，是从t+1开始，每次隔一个人选取（即i = i+2，因为另一些人的名字是给右边的），每次把stu[i]的name赋值给ans[j–]；排右边的队伍同理，ans数组的下标 j 从m/2 + 1开始，一直往右j++，stu的下标 i，从t+2开始，每次隔一个人选取（i = i+2），每次把stu[i]的name赋值给ans[j++]，然后输出当前已经排好的ans数组～每一次排完一列row-1，直到row等于0时退出循环表示已经排列并输出所有的行～

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
struct node {
    string name;
    int height;
};
int cmp(struct node a, struct node b) {
    return a.height != b.height ? a.height > b.height : a.name < b.name;
}
```

```cpp
int main() {
    int n, k, m;
    cin >> n >> k;
    vector<node> stu(n);
    for(int i = 0; i < n; i++) {
        cin >> stu[i].name;
        cin >> stu[i].height;
    }
    sort(stu.begin(), stu.end(), cmp);
    int t = 0, row = k;
    while(row) {
        if(row == k) {
            m = n - n / k * (k - 1);
        } else {
            m = n / k;
        }
        vector<string> ans(m);
        ans[m / 2] = stu[t].name;
        // 左边一列
        int j = m / 2 - 1;
        for(int i = t + 1; i < t + m; i = i + 2)
            ans[j--] = stu[i].name;
        // 右边一列
        j = m / 2 + 1;
        for(int i = t + 2; i < t + m; i = i + 2)
            ans[j++] = stu[i].name;
        // 输出当前排
        cout << ans[0];
        for(int i = 1; i < m; i++)
            cout << " " << ans[i];
        cout << endl;
        t = t + m;
        row--;
    }
    return 0;
}
```

1110. Complete Binary Tree (25) [完全二叉树]

Given a tree, you are supposed to tell if it is a complete binary tree.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=20) which is the total number of nodes in the tree — and hence the nodes are numbered from 0 to N-1. Then N lines follow, each corresponds to a node, and gives the indices of the left and right children of the node. If the child does not exist, a “-” will be put at the position. Any pair of children are separated by a space.

Output Specification:

For each case, print in one line “YES” and the index of the last node if the tree is a complete binary tree, or “NO” and the index of the root if not. There must be exactly one space separating the word and the number.

Sample Input 1:

9

7 8

– –

– –

– –

0 1

2 3

4 5

– –

– –

Sample Output 1:

YES 8

Sample Input 2:

8

– –

4 5

0 6

– –

2 3

– 7

– –

– –

Sample Output 2:

NO 1

题目大意：给出一个n表示有n个结点，这n个结点为0~n-1，给出这n个结点的左右孩子，求问这棵树是不是完全二叉树

分析：递归出最大的下标值，完全二叉树一定把前面的下标充满： 最大的下标值 == 最大的节点数；不完全二叉树前满一定有位置是空，会往后挤： 最大的下标值 > 最大的节点数～

```
#include <iostream>
using namespace std;
struct node{
    int l, r;
```

```
}a[100];
int maxn = -1, ans;
void dfs(int root, int index) {
    if(index > maxn) {
        maxn = index;
        ans = root;
    }
    if(a[root].l != -1) dfs(a[root].l, index * 2);
    if(a[root].r != -1) dfs(a[root].r, index * 2 + 1);
}
int main() {
    int n, root = 0, have[100] = {0};
    cin >> n;
    for (int i = 0; i < n; i++) {
        string l, r;
        cin >> l >> r;
        if (l == "-") {
            a[i].l = -1;
        } else {
            a[i].l = stoi(l);
            have[stoi(l)] = 1;
        }
        if (r == "-") {
            a[i].r = -1;
        } else {
            a[i].r = stoi(r);
            have[stoi(r)] = 1;
        }
    }
    while (have[root] != 0) root++;
    dfs(root, 1);
    if (maxn == n)
        cout << "YES " << ans;
    else
        cout << "NO " << root;
    return 0;
}
```

1111. Online Map (30) [Dijkstra算法 + DFS]

Input our current position and a destination, an online map can recommend several paths. Now your job is to recommend two paths to your user: one is the shortest, and the other is the fastest. It is guaranteed that a path exists for any request.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers N (2 <= N <= 500), and M, being the total number of streets intersections on a map, and the number of streets, respectively. Then M lines follow, each describes a street in the format:

V1 V2 one-way length time

where V1 and V2 are the indices (from 0 to N-1) of the two ends of the street; one-way is 1 if the street is one-way from V1 to V2, or 0 if not; length is the length of the street; and time is the time taken to pass the street.

Finally a pair of source and destination is given.

Output Specification:

For each case, first print the shortest path from the source to the destination with distance D in the format:

Distance = D: source -> v1 -> … -> destination

Then in the next line print the fastest path with total time T:

Time = T: source -> w1 -> … -> destination

In case the shortest path is not unique, output the fastest one among the shortest paths, which is guaranteed to be unique. In case the fastest path is not unique, output the one that passes through the fewest intersections, which is guaranteed to be unique.

In case the shortest and the fastest paths are identical, print them in one line in the format:

Distance = D; Time = T: source -> u1 -> … -> destination

Sample Input 1:

10 15

0 1 0 1 1

8 0 0 1 1

4 8 1 1 1

3 4 0 3 2

3 9 1 4 1

0 6 0 1 1

7 5 1 2 1

8 5 1 2 1

2 3 0 2 2

2 1 1 1 1

1 3 0 3 1

1 4 0 1 1

9 7 1 3 1

5 1 0 5 2

6 5 1 1 2

3 5

Sample Output 1:

Distance = 6: 3 -> 4 -> 8 -> 5

Time = 3: 3 -> 1 -> 5

Sample Input 2:

7 9

0 4 1 1 1

1 6 1 1 3

2 6 1 1 1

2 5 1 2 2

3 0 0 1 1

3 1 1 1 3

3 2 1 1 2

4 5 0 2 2

6 5 1 1 2

3 5

Sample Output 2:

Distance = 3; Time = 4: 3 -> 2 -> 5

题目大意：给一张地图，两个结点中既有距离也有时间，有的单行有的双向，要求根据地图推荐两条路线：一条是最快到达路线，一条是最短距离的路线。

第一行给出两个整数N和M，表示地图中地点的个数和路径的条数。接下来的M行每一行给出：道路结点编号V1 道路结点编号V2 是否单行线 道路长度 所需时间

要求第一行输出最快到达时间Time和路径，第二行输出最短距离Distance和路径

分析：

1.用两个Dijkstra。一个求最短路径（如果相同求时间最短的那条），一个求最快路径（如果相同求结点数最小的那条）～～

2.求最短路径,和最快路径都可以在Dijkstra里面求前驱结点dispre和，Timepre数组～

3.dispre数组更新的条件是路径更短，或者路径相等的同时时间更短。

4.求最快路径时候要多维护一个NodeNum数组，记录在时间最短的情况下，到达此节点所需的节点数量。

Time数组更新的条件是，时间更短，时间相同的时候，如果此节点能让到达次节点是数目也变小，则更新Timepre，heNodeNum数组

5.最后根据dispre 和Timepre数组递归出两条路径，比较判断，输出最终答案～

注意：如果直接使用DFS的话，会导致最后一个测试用例“运行超时”～～

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
const int inf = 999999999;
int dis[510], Time[510], e[510][510], w[510][510], dispre[510],Timepre[510],
weight[510],NodeNum[510];
bool visit[510];
vector<int> dispath, Timepath, temppath;
int st, fin, minnode = inf;
void dfsdispath(int v) {
    dispath.push_back(v);
    if(v == st) return ;
    dfsdispath(dispre[v]);
}
void dfsTimepath(int v) {
    Timepath.push_back(v);
    if(v == st) return ;
    dfsTimepath(Timepre[v]);
}
int main() {
    fill(dis, dis + 510, inf);
    fill(Time, Time + 510, inf);
    fill(weight, weight + 510, inf);
    fill(e[0], e[0] + 510 * 510, inf);
    fill(w[0], w[0] + 510 * 510, inf);
    int n, m;
    scanf("%d %d", &n, &m);
    int a, b, flag, len, t;
    for(int i = 0; i < m; i++) {
        scanf("%d %d %d %d %d", &a, &b, &flag, &len, &t);
        e[a][b] = len;
        w[a][b] = t;
        if(flag != 1) {
            e[b][a] = len;
            w[b][a] = t;
        }
    }
    scanf("%d %d", &st, &fin);
    dis[st] = 0;
    for(int i = 0; i < n; i++)
        dispre[i] = i;
    for(int i = 0; i < n; i++) {
        int u = -1, minn = inf;
        for(int j = 0; j < n; j++) {
            if(visit[j] == false && dis[j] < minn) {
                u = j;
                minn = dis[j];
            }
        }
        if(u == -1) break;
        visit[u] = true;
```

```
52          for(int v = 0; v < n; v++) {
53              if(visit[v] == false && e[u][v] != inf) {
54                  if(e[u][v] + dis[u] < dis[v]) {
55                      dis[v] = e[u][v] + dis[u];
56                      dispre[v] = u;
57                      weight[v] = weight[u] + w[u][v];
58                  } else if(e[u][v] + dis[u] == dis[v] && weight[v] > weight[u]
    + w[u][v]) {
59                      weight[v] = weight[u] + w[u][v];
60                      dispre[v] = u;
61                  }
62              }
63          }
64      }
65      dfsdispath(fin);
66      Time[st] = 0;
67      fill(visit, visit + 510, false);
68      for(int i = 0; i < n; i++) {
69          int u = -1, minn = inf;
70          for(int j = 0; j < n; j++) {
71              if(visit[j] == false && minn > Time[j]) {
72                  u = j;
73                  minn = Time[j];
74              }
75          }
76          if(u == -1) break;
77          visit[u] = true;
78          for(int v = 0; v < n; v++) {
79              if(visit[v] == false && w[u][v] != inf) {
80                  if(w[u][v] + Time[u] < Time[v]) {
81                      Time[v] = w[u][v] + Time[u];
82                      Timepre[v]=(u);
83                      NodeNum[v]=NodeNum[u]+1;
84                  } else if(w[u][v] + Time[u] ==
    Time[v]&&NodeNum[u]+1<NodeNum[v]) {
85                      Timepre[v]=(u);
86                      NodeNum[v]=NodeNum[u]+1;
87                  }
88              }
89          }
90      }
91      dfsTimepath(fin);
92      printf("Distance = %d", dis[fin]);
93      if(dispath == Timepath) {
94          printf("; Time = %d: ", Time[fin]);
95      } else {
96          printf(": ");
97          for(int i = dispath.size() - 1; i >= 0; i--) {
98              printf("%d", dispath[i]);
99              if(i != 0) printf(" -> ");
100         }
101         printf("\nTime = %d: ", Time[fin]);
```

```
        }
        for(int i = Timepath.size() - 1; i >= 0; i--) {
            printf("%d", Timepath[i]);
            if(i != 0) printf(" -> ");
        }
        return 0;
    }
```

1112. Stucked Keyboard (20) [map映射， STL的使用]

On a broken keyboard, some of the keys are always stucked. So when you type some sentences, the characters corresponding to those keys will appear repeatedly on screen for k times.

Now given a resulting string on screen, you are supposed to list all the possible stucked keys, and the original string.

Notice that there might be some characters that are typed repeatedly. The stucked key will always repeat output for a fixed k times whenever it is pressed. For example, when k=3, from the string “thiiis iiisss a teeeeeest” we know that the keys “i” and “e” might be stucked, but “s” is not even though it appears repeatedly sometimes. The original string could be “this isss a teest”.

Input Specification:

Each input file contains one test case. For each case, the 1st line gives a positive integer k (1<k<=100) which is the output repeating times of a stucked key. The 2nd line contains the resulting string on screen, which consists of no more than 1000 characters from {a-z}, {0-9} and “_”. It is guaranteed that the string is non-empty.

Output Specification:

For each test case, print in one line the possible stucked keys, in the order of being detected. Make sure that each key is printed once only. Then in the next line print the original string. It is guaranteed that there is at least one stucked key.

Sample Input:

3

caseee1__thiiis_iiisss_a_teeeeeest

Sample Output:

ei

case1__this_isss_a_teest

题目大意：键盘某些键卡住了，按一次重复k次，要求找出可能的键，并且输出正确的字符串顺序。可能的键要求按照被发现的顺序输出。

分析：考察STL的应用～ map<char, bool>存储出现的键是否坏，set<char>存储输出可能坏的键的时候，当前字符是否已经被输出过，输出过的键放在set里面.

寻找坏键：遍历字符串的每个字符的时候，

与pre（字符串当前字符s[i]的前一个字符）相比较，如果相等就继续计数cnt++，如果不相等，令cnt = 1表示当前字符出现了一次～

如果cnt % k等于0 则令s[i]可能是坏键，置map对应的字符的bool值为true～

输出坏键：由于需要根据坏键发现的顺序输出，所以遍历整个字符串的方式输出～，并且确保不会重复输出～（用set集合确保，输出过了的放在set里面）

输出整个正确的字符串：如果当前s[i]是坏键，在输出一次后，令 i = i + k – 1，再输出，保证坏键出现k次只输出一次～

tips：谢谢在csdn博客的评论里的同学友情提醒，如果出现了先不是坏建后又判断是坏键的情况，这种情况会出现错误，因为前面已经认为它不是坏键了，说明它一定不是坏建，所以要加一个sureNoBroken，把确定不是坏键的键标记出来，在map都设置完成后把确定不是坏键的m标记为false。虽然测试用例里面没有考虑到这种情况，但是如果输入3 aabbaaa，应该输出没有坏键。

```
#include <iostream>
#include <map>
#include <cstdio>
#include <set>
using namespace std;
bool sureNoBroken[256];
int main() {
    int k, cnt = 1;
    scanf("%d", &k);
    string s;
    cin >> s;
    map<char, bool> m;
    set<char> printed;
    char pre = '#';
    s = s + '#';
    for(int i = 0; i < s.length(); i++) {
        if(s[i] == pre) {
            cnt++;
        } else {
            if(cnt % k != 0) {
                sureNoBroken[pre] = true;
            }
            cnt = 1;
        }
        if(i != s.length() - 1) m[s[i]] = (cnt % k == 0);
        pre = s[i];
    }
    for(int i = 0; i < s.length() - 1; i++) {
        if(sureNoBroken[s[i]] == true)
            m[s[i]] = false;
    }
    for(int i = 0; i < s.length() - 1; i++) {
        if(m[s[i]] && printed.find(s[i]) == printed.end()) {
            printf("%c", s[i]);
            printed.insert(s[i]);
        }
    }
```

```
        printf("\n");
        for(int i = 0; i < s.length() - 1; i++) {
            printf("%c", s[i]);
            if(m[s[i]])
                i = i + k - 1;
        }
        return 0;
    }
```

1113. Integer Set Partition (25) [排序]

Given a set of N (> 1) positive integers, you are supposed to partition them into two disjoint sets A1 and A2 of n1 and n2 numbers, respectively. Let S1 and S2 denote the sums of all the numbers in A1 and A2, respectively. You are supposed to make the partition so that |n1 – n2| is minimized first, and then |S1 – S2| is maximized.

Input Specification:

Each input file contains one test case. For each case, the first line gives an integer N (2 <= N <= 105), and then N positive integers follow in the next line, separated by spaces. It is guaranteed that all the integers and their sum are less than 231.

Output Specification:

For each case, print in a line two numbers: |n1 – n2| and |S1 – S2|, separated by exactly one space.

Sample Input 1:

10

23 8 10 99 46 2333 46 1 666 555

Sample Output 1:

0 3611

Sample Input 2:

13

110 79 218 69 3721 100 29 135 2 6 13 5188 85

Sample Output 2:

1 9359

题目大意：要求把一个集合分成两个不相交的集合，使得这两个集合的元素个数相差最小的前提下，两个集合的总和之差最大

分析：先把集合内n个元素排序，计算前n/2个元素的总和，然后用总的总和sum – 2 * halfsum即为 |S1 – S2|。

|n1 – n2|就是n % 2的结果，奇数为1，偶数为0。（总和sum的值其实可以在输入的时候就累加得到啦～）

```
#include <iostream>
#include <algorithm>
```

```cpp
#include <vector>
using namespace std;
int main() {
    int n, sum = 0, halfsum = 0;
    scanf("%d", &n);
    vector<int> v(n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &v[i]);
        sum += v[i];
    }
    sort(v.begin(), v.end());
    for(int i = 0; i < n / 2; i++)
        halfsum += v[i];
    printf("%d %d", n % 2, sum - 2 * halfsum);
    return 0;
}
```

1114. Family Property (25) [并查集]

This time, you are supposed to help us collect the data for family-owned property. Given each person's family members, and the estate（房产）info under his/her own name, we need to know the size of each family, and the average area and number of sets of their real estate.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=1000). Then N lines follow, each gives the infomation of a person who owns estate in the format:

ID Father Mother k Child1 … Childk M_estate Area

where ID is a unique 4-digit identification number for each person; Father and Mother are the ID's of this person's parents (if a parent has passed away, -1 will be given instead); k (0<=k<=5) is the number of children of this person; Childi's are the ID's of his/her children; M_estate is the total number of sets of the real estate under his/her name; and Area is the total area of his/her estate.

Output Specification:

For each case, first print in a line the number of families (all the people that are related directly or indirectly are considered in the same family). Then output the family info in the format:

ID M AVG_sets AVG_area

where ID is the smallest ID in the family; M is the total number of family members; AVG_sets is the average number of sets of their real estate; and AVG_area is the average area. The average numbers must be accurate up to 3 decimal places. The families must be given in descending order of their average areas, and in ascending order of the ID's if there is a tie.

Sample Input:

10

6666 5551 5552 1 7777 1 100

1234 5678 9012 1 0002 2 300

8888 -1 -1 0 1 1000

2468 0001 0004 1 2222 1 500

7777 6666 -1 0 2 300

3721 -1 -1 1 2333 2 150

9012 -1 -1 3 1236 1235 1234 1 100

1235 5678 9012 0 1 50

2222 1236 2468 2 6661 6662 1 300

2333 -1 3721 3 6661 6662 6663 1 100

Sample Output:

3

8888 1 1.000 1000.000

0001 15 0.600 100.000

5551 4 0.750 100.000

题目大意：给定每个人的家庭成员和其自己名下的房产，请你统计出每个家庭的人口数、人均房产面积及房产套数。首先在第一行输出家庭个数（所有有亲属关系的人都属于同一个家庭）。随后按下列格式输出每个家庭的信息：家庭成员的最小编号 家庭人口数 人均房产套数 人均房产面积。其中人均值要求保留小数点后3位。家庭信息首先按人均面积降序输出，若有并列，则按成员编号的升序输出。

分析：用并查集。分别用两个结构体数组，一个data用来接收数据，接收的时候顺便实现了并查集的操作union，另一个数组ans用来输出最后的答案，因为要计算家庭人数，所以用visit标记所有出现过的结点，对于每个结点的父结点，people++统计人数。标记flag == true，计算true的个数cnt就可以知道一共有多少个家庭。排序后输出前cnt个就是所求答案～～

```
#include <cstdio>
#include <algorithm>
using namespace std;
struct DATA {
    int id, fid, mid, num, area;
    int cid[10];
}data[1005];
struct node {
    int id, people;
    double num, area;
    bool flag = false;
}ans[10000];
int father[10000];
bool visit[10000];
int find(int x) {
    while(x != father[x])
        x = father[x];
    return x;
}
void Union(int a, int b) {
```

```
    int faA = find(a);
    int faB = find(b);
    if(faA > faB)
        father[faA] = faB;
    else if(faA < faB)
        father[faB] = faA;
}
int cmp1(node a, node b) {
    if(a.area != b.area)
        return a.area > b.area;
    else
        return a.id < b.id;
}
int main() {
    int n, k, cnt = 0;
    scanf("%d", &n);
    for(int i = 0; i < 10000; i++)
        father[i] = i;
    for(int i = 0; i < n; i++) {
        scanf("%d %d %d %d", &data[i].id, &data[i].fid, &data[i].mid, &k);
        visit[data[i].id] = true;
        if(data[i].fid != -1) {
            visit[data[i].fid] = true;
            Union(data[i].fid, data[i].id);
        }
        if(data[i].mid != -1) {
            visit[data[i].mid] = true;
            Union(data[i].mid, data[i].id);
        }
        for(int j = 0; j < k; j++) {
            scanf("%d", &data[i].cid[j]);
            visit[data[i].cid[j]] = true;
            Union(data[i].cid[j], data[i].id);
        }
        scanf("%d %d", &data[i].num, &data[i].area);
    }
    for(int i = 0; i < n; i++) {
        int id = find(data[i].id);
        ans[id].id = id;
        ans[id].num += data[i].num;
        ans[id].area += data[i].area;
        ans[id].flag = true;
    }
    for(int i = 0; i < 10000; i++) {
        if(visit[i])
            ans[find(i)].people++;
        if(ans[i].flag)
            cnt++;
    }
    for(int i = 0; i < 10000; i++) {
        if(ans[i].flag) {
            ans[i].num = (double)(ans[i].num * 1.0 / ans[i].people);
```

```
            ans[i].area = (double)(ans[i].area * 1.0 / ans[i].people);
        }
    }
    sort(ans, ans + 10000, cmp1);
    printf("%d\n", cnt);
    for(int i = 0; i < cnt; i++)
        printf("%04d %d %.3f %.3f\n", ans[i].id, ans[i].people, ans[i].num,
ans[i].area);
    return 0;
}
```

1115. Counting Nodes in a BST (30) [二叉树的遍历，DFS]

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

The left subtree of a node contains only nodes with keys less than or equal to the node's key.

The right subtree of a node contains only nodes with keys greater than the node's key.

Both the left and right subtrees must also be binary search trees.

Insert a sequence of numbers into an initially empty binary search tree. Then you are supposed to count the total number of nodes in the lowest 2 levels of the resulting tree.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=1000) which is the size of the input sequence. Then given in the next line are the N integers in [-1000 1000] which are supposed to be inserted into an initially empty binary search tree.

Output Specification:

For each case, print in one line the numbers of nodes in the lowest 2 levels of the resulting tree in the format:

n1 + n2 = n

where n1 is the number of nodes in the lowest level, n2 is that of the level above, and n is the sum.

Sample Input:

9

25 30 42 16 20 20 35 -5 28

Sample Output:

2 + 4 = 6

题目大意：输出一个二叉搜索树的最后两层结点个数a和b，以及他们的和c："a + b = c"

分析：用链表存储，递归构建二叉搜索树，深度优先搜索，传入的参数为结点和当前结点的深度depth，如果当前结点为NULL就更新最大深度maxdepth的值并return，将每一层所对应的结点个数存储在数组num中，输出数组的最后两个的值～～

```
#include <iostream>
#include <vector>
```

```
using namespace std;
struct node {
    int v;
    struct node *left, *right;
};
node* build(node *root, int v) {
    if(root == NULL) {
        root = new node();
        root->v = v;
        root->left = root->right = NULL;
    } else if(v <= root->v)
        root->left = build(root->left, v);
    else
        root->right = build(root->right, v);
    return root;
}
vector<int> num(1000);
int maxdepth = -1;
void dfs(node *root, int depth) {
    if(root == NULL) {
        maxdepth = max(depth, maxdepth);
        return ;
    }
    num[depth]++;
    dfs(root->left, depth + 1);
    dfs(root->right, depth + 1);

}
int main() {
    int n, t;
    scanf("%d", &n);
    node *root = NULL;
    for(int i = 0; i < n; i++) {
        scanf("%d", &t);
        root = build(root, t);
    }
    dfs(root, 0);
    printf("%d + %d = %d", num[maxdepth-1], num[maxdepth-2], num[maxdepth-1] +
num[maxdepth-2]);
    return 0;
}
```

1116. Come on! Let's C (20) [简单逻辑题]

"Let's C" is a popular and fun programming contest hosted by the College of Computer Science and Technology, Zhejiang University. Since the idea of the contest is for fun, the award rules are funny as the following:

0. The Champion will receive a "Mystery Award" (such as a BIG collection of students' research papers…).
1. Those who ranked as a prime number will receive the best award — the Minions (小黄人)!
2. Everyone else will receive chocolates.

Given the final ranklist and a sequence of contestant ID's, you are supposed to tell the corresponding awards.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=10000), the total number of contestants. Then N lines of the ranklist follow, each in order gives a contestant's ID (a 4-digit number). After the ranklist, there is a positive integer K followed by K query ID's.

Output Specification:

For each query, print in a line "ID: award" where the award is "Mystery Award", or "Minion", or "Chocolate". If the ID is not in the ranklist, print "Are you kidding?" instead. If the ID has been checked before, print "ID: Checked".

Sample Input:

6

1111

6666

8888

1234

5555

0001

6

8888

0001

1111

2222

8888

2222

Sample Output:

8888: Minion

0001: Chocolate

1111: Mystery Award

2222: Are you kidding?

8888: Checked

2222: Are you kidding?

分析：ran数组标记每个id对应的排名，集合ss存储所有已经询问过的id，如果发现当前id已经出现在ss中，则输出“Checked”，如果ran[id] == 0说明当前id不在排名列表中，所以输出“Are you kidding?”，如果ran[id]为1则输出“Minion”，如果ran[id]为素数则输出“Mystery Award”，否则输出“Chocolate”

```cpp
#include <iostream>
#include <set>
#include <cmath>
using namespace std;
int ran[10000];
bool isprime(int a) {
    if(a <= 1) return false;
    int Sqrt = sqrt((double)a);
    for(int i = 2; i <= Sqrt; i++) {
        if(a % i == 0)
            return false;
    }
    return true;
}
int main() {
    int n, k;
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        int id;
        scanf("%d", &id);
        ran[id] = i + 1;
    }
    scanf("%d", &k);
    set<int> ss;
    for(int i = 0; i < k; i++) {
        int id;
        scanf("%d", &id);
        printf("%04d: ", id);
        if(ran[id] == 0) {
            printf("Are you kidding?\n");
            continue;
        }
        if(ss.find(id) == ss.end()) {
            ss.insert(id);
        } else {
            printf("Checked\n");
            continue;
        }
        if(ran[id] == 1) {
            printf("Mystery Award\n");
        }else if(isprime(ran[id])) {
            printf("Minion\n");
        }else {
            printf("Chocolate\n");
        }
    }
    return 0;
}
```

1117. Eddington Number(25) [简单逻辑题]

British astronomer Eddington liked to ride a bike. It is said that in order to show off his skill, he has even defined an “Eddington number”, E — that is, the maximum integer E such that it is for E days that one rides more than E miles. Eddington’s own E was 87.

Now given everyday’s distances that one rides for N days, you are supposed to find the corresponding E (<=N).

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N(<=105), the days of continuous riding. Then N non-negative integers are given in the next line, being the riding distances of everyday.

Output Specification:

For each case, print in a line the Eddington number for these N days.

Sample Input:

10

6 7 6 9 3 10 8 2 7 8

Sample Output:

6

题目大意：英国天文学家爱丁顿很喜欢骑车。据说他为了炫耀自己的骑车功力，还定义了一个“爱丁顿数”E，即满足有E天骑车超过E英里的最大整数E。据说爱丁顿自己的E等于87。

分析：在数组a中存储n天的公里数，对n个数据从大到小排序，i表示了骑车的天数，那么满足a[i] > i+1的最大i+1即为所求

```
#include <iostream>
#include <algorithm>
using namespace std;
int a[1000000];
int main() {
    int n, e = 0;
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    sort(a, a+n, greater<int>());
    while(e < n && a[e] > e+1) e++;
    printf("%d", e);
    return 0;
}
```

1118. Birds in Forest (25) [并查集]

Some scientists took pictures of thousands of birds in a forest. Assume that all the birds appear in the same picture belong to the same tree. You are supposed to help the scientists to count the maximum number of trees in the forest, and for any pair of birds, tell if they are on the same tree.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive number N (<= 104) which is the number of pictures. Then N lines follow, each describes a picture in the format:

K B1 B2 … BK

where K is the number of birds in this picture, and Bi's are the indices of birds. It is guaranteed that the birds in all the pictures are numbered continuously from 1 to some number that is no more than 104.

After the pictures there is a positive number Q (<= 104) which is the number of queries. Then Q lines follow, each contains the indices of two birds.

Output Specification:

For each test case, first output in a line the maximum possible number of trees and the number of birds. Then for each query, print in a line "Yes" if the two birds belong to the same tree, or "No" if not.

Sample Input:

4

3 10 1 2

2 3 4

4 1 5 7 8

3 9 6 4

2

10 5

3 7

Sample Output:

2 10

Yes

No

题目大意：一幅画里面的鸟为同一棵树上的，问有多少棵树和多少只鸟，以及对于两只鸟判断是否在同一个树上～

分析：使用并查集就可以啦~将一幅画中鸟使用Union函数合并在同一个集合里面，cnt[i]数组保存以i为FindFather结点的集合里面的鸟的个数，exist[i]表示鸟的id——i在输入的鸟的序号里面是否出现过，遍历cnt数组并累加所有不为0的个数即可得知有多少棵树，累加所有出现过的鸟的id的cnt的值即可得知鸟的个数～判断两只鸟是否在同一棵树，只需要知道这两只鸟的findFather是否相等即可～

```
#include <iostream>
using namespace std;
int n, m, k;
const int maxn = 10010;
int fa[maxn] = {0}, cnt[maxn] = {0};
int findFather(int x) {
```

```
    int a = x;
    while(x != fa[x])
        x = fa[x];
    while(a != fa[a]) {
        int z = a;
        a = fa[a];
        fa[z] = x;
    }
    return x;
}
void Union(int a, int b) {
    int faA = findFather(a);
    int faB = findFather(b);
    if(faA != faB) fa[faA] = faB;
}
bool exist[maxn];
int main() {
    scanf("%d", &n);
    for(int i = 1; i <= maxn; i++)
        fa[i] = i;
    int id, temp;
    for(int i = 0; i < n; i++) {
        scanf("%d%d", &k, &id);
        exist[id] = true;
        for(int j = 0; j < k-1; j++) {
            scanf("%d", &temp);
            Union(id, temp);
            exist[temp] = true;
        }
    }
    for(int i = 1; i <= maxn; i++) {
        if(exist[i] == true) {
            int root = findFather(i);
            cnt[root]++;
        }
    }
    int numTrees = 0, numBirds = 0;
    for(int i = 1; i <= maxn; i++) {
        if(exist[i] == true && cnt[i] != 0) {
            numTrees++;
            numBirds += cnt[i];
        }
    }
    printf("%d %d\n", numTrees, numBirds);
    scanf("%d", &m);
    int ida, idb;
    for(int i = 0; i < m; i++) {
        scanf("%d%d", &ida, &idb);
        printf("%s\n", (findFather(ida) == findFather(idb)) ? "Yes" : "No");
    }
    return 0;
}
```

1119. Pre- and Post-order Traversals (30) [树的遍历，前序后序转中序]

Suppose that all the keys in a binary tree are distinct positive integers. A unique binary tree can be determined by a given pair of postorder and inorder traversal sequences, or preorder and inorder traversal sequences. However, if only the postorder and preorder traversal sequences are given, the corresponding tree may no longer be unique.

Now given a pair of postorder and preorder traversal sequences, you are supposed to output the corresponding inorder traversal sequence of the tree. If the tree is not unique, simply output any one of them.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=30), the total number of nodes in the binary tree. The second line gives the preorder sequence and the third line gives the postorder sequence. All the numbers in a line are separated by a space.

Output Specification:

For each test case, first printf in a line “Yes” if the tree is unique, or “No” if not. Then print in the next line the inorder traversal sequence of the corresponding binary tree. If the solution is not unique, any answer would do. It is guaranteed that at least one solution exists. All the numbers in a line must be separated by exactly one space, and there must be no extra space at the end of the line.

Sample Input 1:

7

1 2 3 4 6 7 5

2 6 7 4 5 3 1

Sample Output 1:

Yes

2 1 6 4 7 3 5

Sample Input 2:

4

1 2 3 4

2 4 3 1

Sample Output 2:

No

2 1 3 4

题目大意：给出一棵树的结点个数n，以及它的前序遍历和后序遍历，输出它的中序遍历，如果中序遍历不唯一就输出No，且输出其中一个中序即可，如果中序遍历唯一就输出Yes，并输出它的中序

分析：用unique标记是否唯一，如果为true就表示中序是唯一的～

已知二叉树的前序和后序是无法唯一确定一颗二叉树的，因为可能会存在多种情况，这种情况就是一个结点可能是根的左孩子也有可能是根的右孩子，如果发现了一个无法确定的状态，置unique = false，又因为题目只需要输出一个方案，可以假定这个不可确定的孩子的状态是右孩子，接下来的问题是如何求根结点和左右孩子划分的问题了，首先我们需要知道树的表示范围，需要四个变量，分别是前序的开始的地方prel，前序结束的地方prer，后序开始的地方postl，后序结束的地方postr，前序的开始的第一个应该是后序的最后一个是相等的，这个结点就是根结点，以后序的根结点的前面一个结点作为参考，寻找这个结点在前序的位置，就可以根据这个位置来划分左右孩子，递归处理。

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> in, pre, post;
bool unique = true;
void getIn(int preLeft, int preRight, int postLeft, int postRight) {
    if(preLeft == preRight) {
        in.push_back(pre[preLeft]);
        return;
    }
    if (pre[preLeft] == post[postRight]) {
        int i = preLeft + 1;
        while (i <= preRight && pre[i] != post[postRight-1]) i++;
        if (i - preLeft > 1)
            getIn(preLeft + 1, i - 1, postLeft, postLeft + (i - preLeft - 1) - 1);
        else
            unique = false;
        in.push_back(post[postRight]);
        getIn(i, preRight, postLeft + (i - preLeft - 1), postRight - 1);
    }
}
int main() {
    int n;
    scanf("%d", &n);
    pre.resize(n), post.resize(n);
    for (int i = 0; i < n; i++)
        scanf("%d", &pre[i]);
    for (int i = 0; i < n; i++)
        scanf("%d", &post[i]);
    getIn(0, n-1, 0, n-1);
    printf("%s\n%d", unique == true ? "Yes" : "No", in[0]);
    for (int i = 1; i < in.size(); i++)
        printf(" %d", in[i]);
    printf("\n");
    return 0;
}
```

1120. Friend Numbers (20) [set的应用]

Two integers are called “friend numbers” if they share the same sum of their digits, and the sum is their “friend ID”. For example, 123 and 51 are friend numbers since 1+2+3 = 5+1 = 6, and 6 is their friend ID. Given some numbers, you are supposed to count the number of different friend ID’s among them. Note: a number is considered a friend of itself.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N. Then N positive integers are given in the next line, separated by spaces. All the numbers are less than 10^4.

Output Specification:

For each case, print in the first line the number of different frind ID’s among the given integers. Then in the second line, output the friend ID’s in increasing order. The numbers must be separated by exactly one space and there must be no extra space at the end of the line.

Sample Input:

8

123 899 51 998 27 33 36 12

Sample Output:

4

3 6 9 26

分析：在接收输入数据的时候就把该数字的每一位相加，并把结果插入一个set集合中。因为set是有序的、不重复的，所以set的size值就是输出的个数，set中的每一个数字即所有答案的数字序列

```
#include <iostream>
#include <set>
using namespace std;
int getFriendNum(int num) {
    int sum = 0;
    while(num != 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}
int main() {
    set<int> s;
    int n, num;
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &num);
        s.insert(getFriendNum(num));
    }
    printf("%d\n", s.size());
    for(auto it = s.begin(); it != s.end(); it++) {
        if(it != s.begin()) printf(" ");
        printf("%d", *it);
    }
```

```
25        return 0;
26    }
```

1121. Damn Single (25) [set的应用]

“Damn Single (单身狗)” is the Chinese nickname for someone who is being single. You are supposed to find those who are alone in a big party, so they can be taken care of.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=50000), the total number of couples. Then N lines of the couples follow, each gives a couple of ID’s which are 5-digit numbers (i.e. from 00000 to 99999). After the list of couples, there is a positive integer M (<=10000) followed by M ID’s of the party guests. The numbers are separated by spaces. It is guaranteed that nobody is having bigamous marriage (重婚) or dangling with more than one companion.

Output Specification:

First print in a line the total number of lonely guests. Then in the next line, print their ID’s in increasing order. The numbers must be separated by exactly 1 space, and there must be no extra space at the end of the line.

Sample Input:

3

11111 22222

33333 44444

55555 66666

7

55555 44444 10000 88888 22222 11111 23333

Sample Output:

5

10000 23333 44444 55555 88888

分析： 设立数组couple[i] = j表示i的对象是j。一开始先设置为都是-1。设立数组isExist表示某人的对象是否来到了派对上。接收数据的时候，对于每一对a和b，将couple的a设置为b，b设置为a，表示他俩是一对。对于每一个需要判断的人，将其存储在guest数组里面，如果它不是单身的（也就是如果它的couple[guest[i]] != -1）那么就将它对象的isExist设置为1，表示他对象的对象（也就是他自己）来到了派对。这样所有isExist不为1的人，对象是没有来到派对的。把所有的人遍历后插入一个集合set里面，set的size就是所求的人数，set里面的所有数就是所求的人的递增排列～～～

```
1   #include <iostream>
2   #include <set>
3   #include <vector>
4   using namespace std;
5   int main() {
6       int n, a, b, m;
```

```
    scanf("%d", &n);
    vector<int> couple(100000);
    for (int i = 0; i < 100000; i++)
        couple[i] = -1;
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &a, &b);
        couple[a] = b;
        couple[b] = a;
    }
    scanf("%d", &m);
    vector<int> guest(m), isExist(100000);
    for (int i = 0; i < m; i++) {
        scanf("%d", &guest[i]);
        if (couple[guest[i]] != -1)
            isExist[couple[guest[i]]] = 1;
    }
    set<int> s;
    for (int i = 0; i < m; i++)
        if (!isExist[guest[i]]) s.insert(guest[i]);
    printf("%d\n", s.size());
    for (auto it = s.begin(); it != s.end(); it++) {
        if (it != s.begin()) printf(" ");
        printf("%05d", *it);
    }
    return 0;
}
```

1122. Hamiltonian Cycle (25) [图论]

The “Hamilton cycle problem” is to find a simple cycle that contains every vertex in a graph. Such a cycle is called a “Hamiltonian cycle”.

In this problem, you are supposed to tell if a given cycle is a Hamiltonian cycle.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers N (2< N <= 200), the number of vertices, and M, the number of edges in an undirected graph. Then M lines follow, each describes an edge in the format “Vertex1 Vertex2”, where the vertices are numbered from 1 to N. The next line gives a positive integer K which is the number of queries, followed by K lines of queries, each in the format:

n V1 V2 … Vn

where n is the number of vertices in the list, and Vi‘s are the vertices on a path.

Output Specification:

For each query, print in a line “YES” if the path does form a Hamiltonian cycle, or “NO” if not.

Sample Input:

6 10 6 2 3 4 1 5 2 5 3 1 4 1 1 6 6 3 1 2 4 5 6 7 5 1 4 3 6 2 5 6 5 1 4 3 6 2 9 6 2 1 6 3 4 5 2 6 4 1 2 5 1 7 6 1 3 4 5 2 6 7 6 1 2 5 4 3 1

Sample Output:

YES

NO

NO

NO

YES

NO

题目大意：给出一个图，判断给定的路径是不是哈密尔顿路径

分析：1.设置falg1 判断节点是否多走、少走、或走成环

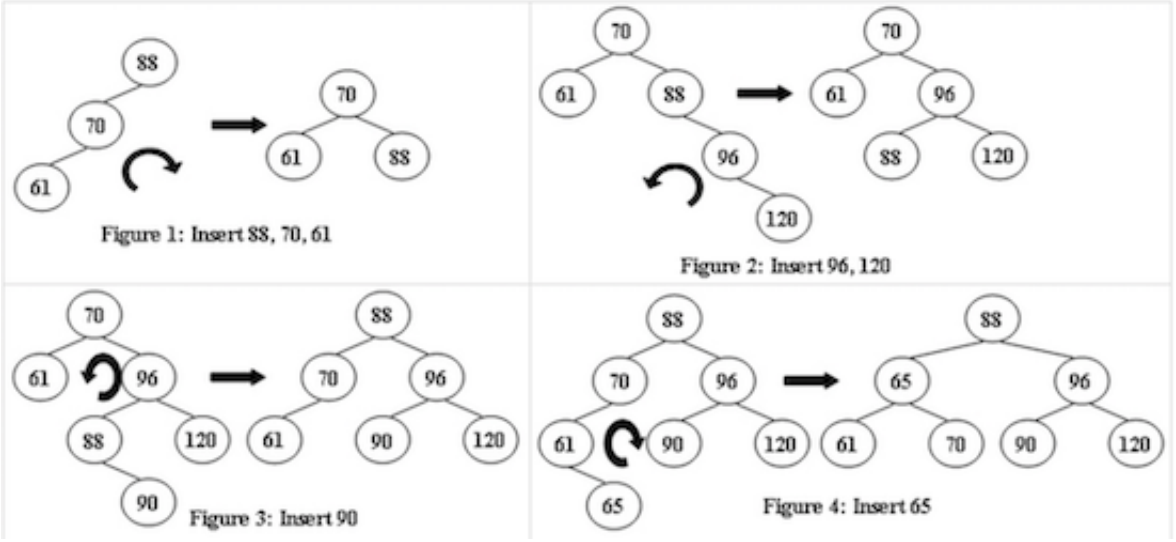
2.设置flag2 判断这条路能不能走通

3.当falg1、flag2都为1时是哈密尔顿路径，否则不是

```
#include <iostream>
#include <set>
#include <vector>
using namespace std;
int main() {
    int n, m, cnt, k, a[210][210] = {0};
    cin >> n >> m;
    for(int i = 0; i < m; i++) {
        int t1, t2;
        scanf("%d%d", &t1, &t2);
        a[t1][t2] = a[t2][t1] = 1;
    }
    cin >> cnt;
    while(cnt--) {
        cin >> k;
        vector<int> v(k);
        set<int> s;
        int flag1 = 1, flag2 = 1;
        for(int i = 0; i < k; i++) {
            scanf("%d", &v[i]);
            s.insert(v[i]);
        }
        if(s.size() != n || k - 1 != n || v[0] != v[k-1]) flag1 = 0;
        for(int i = 0; i < k - 1; i++)
            if(a[v[i]][v[i+1]] == 0) flag2 = 0;
        printf("%s",flag1 && flag2 ? "YES\n" : "NO\n");
    }
    return 0;
}
```

1123. Is It a Complete AVL Tree (30) [AVL树]

An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Figures 1-4 illustrate the rotation rules.



Now given a sequence of insertions, you are supposed to output the level-order traversal sequence of the resulting AVL tree, and to tell if it is a complete binary tree.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (<= 20). Then N distinct integer keys are given in the next line. All the numbers in a line are separated by a space.

Output Specification:

For each test case, insert the keys one by one into an initially empty AVL tree. Then first print in a line the level-order traversal sequence of the resulting AVL tree. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line. Then in the next line, print “YES” if the tree is complete, or “NO” if not.

Sample Input 1:

5

88 70 61 63 65

Sample Output 1:

70 63 88 61 65

YES

Sample Input 2:

8

88 70 61 96 120 90 65 68

Sample Output 2:

88 65 96 61 70 90 120 68

NO

分析：这道题实际上考察AVL树和层序遍历两个知识点～～

判断是不是完全二叉树，就看在出现了一个孩子为空的结点之后是否还会出现孩子结点不为空的结点，如果出现了就不是完全二叉树。

AVL树一共有四种情况，这里我把发现树不平衡的那个结点叫做A结点，A发现树不平衡的情况有四种：

新来的结点插入到A的左子树的左子树

新来的结点插入到A的左子树的右子树

新来的结点插入到A的右子树的左子树

新来的结点插入到A的右子树的右子树

发现不平衡时就需要处理，第1种情况只要简单的右旋，第4种情况只需左旋一下，第2种情况需要先对A的左子树左旋一下，然后对A右旋，同理第3种情况需要对A的右子树右旋一下，然后对A左旋就可以啦～～～

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
struct node {
    int val;
    struct node *left, *right;
};
node* leftRotate(node *tree) {
    node *temp = tree->right;
    tree->right = temp->left;
    temp->left = tree;
    return temp;
}
node* rightRotate(node *tree) {
    node *temp = tree->left;
    tree->left = temp->right;
    temp->right = tree;
    return temp;
}
node* leftRightRotate(node *tree) {
    tree->left = leftRotate(tree->left);
    return rightRotate(tree);
}
node* rightLeftRotate(node *tree) {
    tree->right = rightRotate(tree->right);
    return leftRotate(tree);
}
int getHeight(node *tree) {
    if (tree == NULL) return 0;
    int l = getHeight(tree->left);
    int r = getHeight(tree->right);
    return max(l, r) + 1;
}
```

```
node* insert(node *tree, int val) {
    if (tree == NULL) {
        tree = new node();
        tree->val = val;
    }else if (tree->val > val) {
        tree->left = insert(tree->left, val);
        int l = getHeight(tree->left), r = getHeight(tree->right);
        if (l - r >= 2) {
            if (val < tree->left->val)
                tree = rightRotate(tree);
            else
                tree = leftRightRotate(tree);
        }
    } else {
        tree->right = insert(tree->right, val);
        int l = getHeight(tree->left), r = getHeight(tree->right);
        if (r - l >= 2) {
            if (val > tree->right->val)
                tree = leftRotate(tree);
            else
                tree = rightLeftRotate(tree);
        }
    }
    return tree;
}
int isComplete = 1, after = 0;
vector<int> levelOrder(node *tree) {
    vector<int> v;
    queue<node *> queue;
    queue.push(tree);
    while (!queue.empty()) {
        node *temp = queue.front();
        queue.pop();
        v.push_back(temp->val);
        if (temp->left != NULL) {
            if (after) isComplete = 0;
            queue.push(temp->left);
        } else {
            after = 1;
        }
        if (temp->right != NULL) {
            if (after) isComplete = 0;
            queue.push(temp->right);
        } else {
            after = 1;
        }
    }
    return v;
}
int main() {
    int n, temp;
    scanf("%d", &n);
```

```
87        node *tree = NULL;
88        for (int i = 0; i < n; i++) {
89            scanf("%d", &temp);
90            tree = insert(tree, temp);
91        }
92        vector<int> v = levelOrder(tree);
93        for (int i = 0; i < v.size(); i++) {
94            if (i != 0) printf(" ");
95            printf("%d", v[i]);
96        }
97        printf("\n%s", isComplete ? "YES" : "NO");
98        return 0;
99    }
```

1124. Raffle for Weibo Followers (20) [map映射]

John got a full mark on PAT. He was so happy that he decided to hold a raffle（抽奖） for his followers on Weibo — that is, he would select winners from every N followers who forwarded his post, and give away gifts. Now you are supposed to help him generate the list of winners.

Input Specification:

Each input file contains one test case. For each case, the first line gives three positive integers M (<= 1000), N and S, being the total number of forwards, the skip number of winners, and the index of the first winner (the indices start from 1). Then M lines follow, each gives the nickname (a nonempty string of no more than 20 characters, with no white space or return) of a follower who has forwarded John’s post.

Note: it is possible that someone would forward more than once, but no one can win more than once. Hence if the current candidate of a winner has won before, we must skip him/her and consider the next one.

Output Specification:

For each case, print the list of winners in the same order as in the input, each nickname occupies a line. If there is no winner yet, print “Keep going…” instead.

Sample Input 1:

9 3 2

Imgonnawin!

PickMe

PickMeMeMeee

LookHere

Imgonnawin!

TryAgainAgain

TryAgainAgain

Imgonnawin!

TryAgainAgain

Sample Output 1:

PickMe

Imgonnawin!

TryAgainAgain

Sample Input 2:

2 3 5

Imgonnawin!

PickMe

Sample Output 2:

Keep going...

题目大意：小明PAT考了满分，高兴之余决定发起微博转发抽奖活动，从转发的网友中按顺序每隔N个人就发出一个红包。请你编写程序帮助他确定中奖名单。注意：可能有人转发多次，但不能中奖多次。所以如果处于当前中奖位置的网友已经中过奖，则跳过他顺次取下一位。按照输入的顺序输出中奖名单，每个昵称占一行。如果没有人中奖，则输出“Keep going...”

分析：用mapp存储当前用户有没有已经中奖过～当输入的时候，判断当前字符串是否已经在mapp中出现过，如果出现过就将s+1。每次判断i是否等于s，如果等于s且当前用户没有中过奖，就将它的名字输出，并且s = s + n～并将mapp[str]标记为1，且flag标记为true表示有过人中奖。最后flag如果依然是false说明要输出Keep going...

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    int m, n, s;
    scanf("%d%d%d", &m, &n, &s);
    string str;
    map<string, int> mapp;
    bool flag = false;
    for (int i = 1; i <= m; i++) {
        cin >> str;
        if (mapp[str] == 1) s = s + 1;
        if (i == s && mapp[str] == 0) {
            mapp[str] = 1;
            cout << str << endl;
            flag = true;
            s = s + n;
        }
    }
    if (flag == false) cout << "Keep going...";
    return 0;
}
```

1125. Chain the Ropes (25) [排序，贪心]

Given some segments of rope, you are supposed to chain them into one rope. Each time you may only fold two segments into loops and chain them into one piece, as shown by the figure. The resulting chain will be treated as another segment of rope and can be folded again. After each chaining, the lengths of the original two segments will be halved.

Your job is to make the longest possible rope out of N given segments.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (2 <= N <= 104). Then N positive integer lengths of the segments are given in the next line, separated by spaces. All the integers are no more than 104.

Output Specification:

For each case, print in a line the length of the longest possible rope that can be made by the given segments. The result must be rounded to the nearest integer that is no greater than the maximum length.

Sample Input:

8

10 15 12 3 4 13 1 15

Sample Output:

14

题目大意：给定一段一段的绳子，你需要把它们串成一条绳。每次串连的时候，是把两段绳子对折，再如下图所示套接在一起。这样得到的绳子又被当成是另一段绳子，可以再次对折去跟另一段绳子串连。每次串连后，原来两段绳子的长度就会减半。给定N段绳子的长度，你需要找出它们能串成的绳子的最大长度~

分析：因为所有长度都要串在一起，每次都等于(旧的绳子长度+新的绳子长度)/2，所以越是早加入绳子长度中的段，越要对折的次数多，所以既然希望绳子长度是最长的，就必须让长的段对折次数尽可能的短。所以将所有段从小到大排序，然后从头到尾从小到大分别将每一段依次加入结绳的绳子中，最后得到的结果才会是最长的结果～

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main() {
    int n;
    scanf("%d", &n);
    vector<int> v(n);
    for (int i = 0; i < n; i++)
```

```
        scanf("%d", &v[i]);
    sort(v.begin(), v.end());
    int result = v[0];
    for (int i = 1; i < n; i++)
        result = (result + v[i]) / 2;
    printf("%d", result);
    return 0;
}
```

1126. Eulerian Path (25) [连通图]

In graph theory, an Eulerian path is a path in a graph which visits every edge exactly once. Similarly, an Eulerian circuit is an Eulerian path which starts and ends on the same vertex. They were first discussed by Leonhard Euler while solving the famous Seven Bridges of Konigsberg problem in 1736. It has been proven that connected graphs with all vertices of even degree have an Eulerian circuit, and such graphs are called Eulerian. If there are exactly two vertices of odd degree, all Eulerian paths start at one of them and end at the other. A graph that has an Eulerian path but not an Eulerian circuit is called semi-Eulerian. (Cited from https://en.wikipedia.org/wiki/Eulerian_path)

Given an undirected graph, you are supposed to tell if it is Eulerian, semi-Eulerian, or non-Eulerian.

Input Specification:

Each input file contains one test case. Each case starts with a line containing 2 numbers N (<= 500), and M, which are the total number of vertices, and the number of edges, respectively. Then M lines follow, each describes an edge by giving the two ends of the edge (the vertices are numbered from 1 to N).

Output Specification:

For each test case, first print in a line the degrees of the vertices in ascending order of their indices. Then in the next line print your conclusion about the graph — either “Eulerian”, “Semi-Eulerian”, or “Non-Eulerian”. Note that all the numbers in the first line must be separated by exactly 1 space, and there must be no extra space at the beginning or the end of the line.

Sample Input 1:

7 12

5 7

1 2

1 3

2 3

2 4

3 4

5 2

7 6

6 3

4 5

6 4

5 6

Sample Output 1:

2 4 4 4 4 4 2

Eulerian

Sample Input 2:

6 10

1 2

1 3

2 3

2 4

3 4

5 2

6 3

4 5

6 4

5 6

Sample Output 2:

2 4 4 4 3 3

Semi-Eulerian

Sample Input 3:

5 8

1 2

2 5

5 4

4 1

1 3

3 2

3 4

5 3

Sample Output 3:

3 3 4 3 3

Non-Eulerian

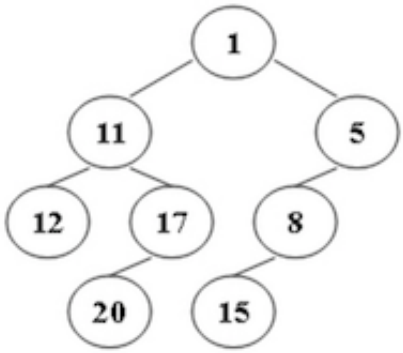
题目大意：如果一个连通图的所有结点的度都是偶数，那么它就是Eulerian，如果除了两个结点的度是奇数其他都是偶数，那么它就是Semi-Eulerian，否则就是Non-Eulerian～

分析：用邻接表存储图，判断每个结点的度【也就是每个结点i的v[i].size()】是多少即可得到最终结果～注意：图必须是连通图，所以要用一个深搜判断一下连通性，从结点1开始深搜，如果最后发现统计的连通结点个数cnt != n说明是不是连通图，要输出Non-Eulerian～

```
#include <iostream>
#include <vector>
using namespace std;
vector<vector<int> > v;
vector<bool> visit;
int cnt = 0;
void dfs(int index) {
    visit[index] = true;
    cnt++;
    for (int i = 0; i < v[index].size(); i++)
        if (visit[v[index][i]] == false)
            dfs(v[index][i]);
}
int main() {
    int n, m, a, b, even = 0;
    scanf("%d%d", &n, &m);
    v.resize(n + 1);
    visit.resize(n + 1);
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &a, &b);
        v[a].push_back(b);
        v[b].push_back(a);
    }
    for (int i = 1; i <= n; i++) {
        if (i != 1) printf(" ");
        printf("%d", v[i].size());
        if (v[i].size() % 2 == 0) even++;
    }
    printf("\n");
    dfs(1);
    if (even == n && cnt == n)
        printf("Eulerian");
    else if(even == n - 2 && cnt == n)
        printf("Semi-Eulerian");
    else
        printf("Non-Eulerian");
    return 0;
}
```

1127. ZigZagging on a Tree (30) [中序后序建树，层序遍历]

Suppose that all the keys in a binary tree are distinct positive integers. A unique binary tree can be determined by a given pair of postorder and inorder traversal sequences. And it is a simple standard routine to print the numbers in level-order. However, if you think the problem is too simple, then you are too naive. This time you are supposed to print the numbers in “zigzagging order” — that is, starting from the root, print the numbers level-by-level, alternating between left to right and right to left. For example, for the following tree you must output: 1 11 5 8 17 12 20 15.



Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<= 30), the total number of nodes in the binary tree. The second line gives the inorder sequence and the third line gives the postorder sequence. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print the zigzagging sequence of the tree in a line. All the numbers in a line must be separated by exactly one space, and there must be no extra space at the end of the line.

Sample Input:

8

12 11 20 17 1 15 8 5

12 20 17 11 15 8 5 1

Sample Output:

1 11 5 8 17 12 20 15

题目大意：给出一个树的中序和后序遍历结果，求它的Z字型层序遍历，也就是偶数层从右往左，奇数层从左往右遍历～

分析：分为3步：1.根据中序和后序建树 保存在tree二维数组中，比如：tree[i][0] = val表示post[i]的左孩子是post[val]，tree[i][1] = val表示post[i]的右孩子是post[val]～

2.进行广度优先搜索，将树从根结点开始所有结点层序遍历，保存在result二维数组中，比如：result[i]保存第i层所有结点的序列～

3.进行z字型输出，根据当前层号的奇偶性分别从左往右、从右往左遍历输出～

1. dfs：因为post(后序)是按照左、右、根的顺序遍历的，所以从右往左，最右边的肯定是根结点～所以postRight是当前子树的根结点的下标，将它的赋值给index，并继续dfs tree[index][0]和tree[index][1]～

根据post[postRight]的结点在in里面的下标位置i，可以得到i的左边是左子树，即inLeft 到 i – 1，右边是右子树：i + 1 到 inRight。而对于post来说，根据左子树的结点个数i – inLeft可以得到[postLeft, postLeft + (i – inLeft) – 1]是post中左子树的范围，[postLeft + (i – inLeft), postRight – 1]是post中右子树的范围～

2.广度优先搜索，采用队列q，q中保存的是node结点，node.index表示当前节点在post中的下标，node.depth表示当前结点在树中的层数～

3.当 i % 2 == 0的时候倒序输出，否则正序输出～

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
vector<int> in, post, result[35];
int n, tree[35][2], root;
struct node {
    int index, depth;
};
void dfs(int &index, int inLeft, int inRight, int postLeft, int postRight) {
    if (inLeft > inRight) return;
    index = postRight;
    int i = 0;
    while (in[i] != post[postRight]) i++;
    dfs(tree[index][0], inLeft, i - 1, postLeft, postLeft + (i - inLeft) - 1);
    dfs(tree[index][1], i + 1, inRight, postLeft + (i - inLeft), postRight -
1);
}
void bfs() {
    queue<node> q;
    q.push(node{root, 0});
    while (!q.empty()) {
        node temp = q.front();
        q.pop();
        result[temp.depth].push_back(post[temp.index]);
        if (tree[temp.index][0] != 0)
            q.push(node{tree[temp.index][0], temp.depth + 1});
        if (tree[temp.index][1] != 0)
            q.push(node{tree[temp.index][1], temp.depth + 1});
    }
}
int main() {
    cin >> n;
    in.resize(n + 1), post.resize(n + 1);
    for (int i = 1; i <= n; i++) cin >> in[i];
    for (int i = 1; i <= n; i++) cin >> post[i];
    dfs(root, 1, n, 1, n);
    bfs();
    printf("%d", result[0][0]);
    for (int i = 1; i < 35; i++) {
        if (i % 2 == 1) {
            for (int j = 0; j < result[i].size(); j++)
                printf(" %d", result[i][j]);
```

```
43            } else {
44                for (int j = result[i].size() - 1; j >= 0; j--)
45                    printf(" %d", result[i][j]);
46            }
47        }
48        return 0;
49    }
```

1128. N Queens Puzzle (20) [逻辑题]

The “eight queens puzzle” is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general N queens problem of placing N non-attacking queens on an N×N chessboard. (From Wikipedia – “Eight queens puzzle”.)

Here you are NOT asked to solve the puzzles. Instead, you are supposed to judge whether or not a given configuration of the chessboard is a solution. To simplify the representation of a chessboard, let us assume that no two queens will be placed in the same column. Then a configuration can be represented by a simple integer sequence (Q1, Q2, …, QN), where Qi is the row number of the queen in the i-th column. For example, Figure 1 can be represented by (4, 6, 8, 2, 7, 1, 3, 5) and it is indeed a solution to the 8 queens puzzle; while Figure 2 can be represented by (4, 6, 7, 2, 8, 1, 9, 5, 3) and is NOT a 9 queens’ solution.

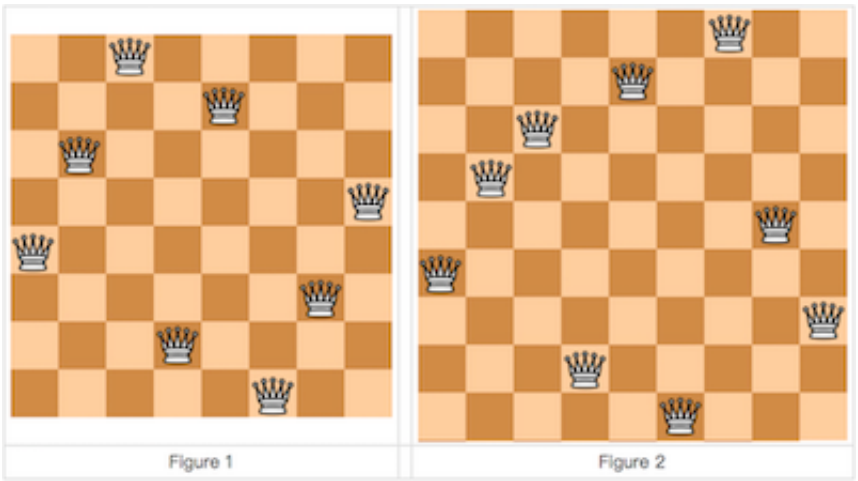


Figure 1 Figure 2

Input Specification:

Each input file contains several test cases. The first line gives an integer K (1 < K <= 200). Then K lines follow, each gives a configuration in the format “N Q1 Q2 … QN”, where 4 <= N <= 1000 and it is guaranteed that 1 <= Qi <= N for all i=1, …, N. The numbers are separated by spaces.

Output Specification:

For each configuration, if it is a solution to the N queens problem, print “YES” in a line; or “NO” if not.

Sample Input:

4

8 4 6 8 2 7 1 3 5

9 4 6 7 2 8 1 9 5 3

6 1 5 2 6 4 3

5 1 3 5 2 4

Sample Output:

YES

NO

NO

YES

题目大意：给出一个皇后图，以这样的方式给出：一个数组包含n个数字，每个数字表示该列的皇后所在的行数～判断给出的皇后图是否满足不会互相攻击（任意两个皇后都要不在同一行或者同一列，且不在斜对角线上～）

分析：用v[n]存储一张图给出的数字～对于第j个数字，判断前0～j-1个数字中是否有在同一行的（v[j] == v[t]）和在斜对角线上的（abs(v[j]-v[t]) == abs(j-t)）【因为已经告诉肯定不在同一列了，所以不需要判断是否在同一列啦～】如果发现了不满足的情况，就将result由true标记为false～最后根据result是true还是false输出对应的YES还是NO即可～

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
int main() {
    int k, n;
    cin >> k;
    for (int i = 0; i < k; i++) {
        cin >> n;
        vector<int> v(n);
        bool result = true;
        for (int j = 0; j < n; j++) {
            cin >> v[j];
            for (int t = 0; t < j; t++) {
                if (v[j] == v[t] || abs(v[j]-v[t]) == abs(j-t)) {
                    result = false;
                    break;
                }
            }
        }
        cout << (result == true ? "YES\n" : "NO\n");
    }
    return 0;
}
```

1129. Recommendation System (25) [set的应用，运算符重载]

Recommendation system predicts the preference that a user would give to an item. Now you are asked to program a very simple recommendation system that rates the user's preference by the number of times that an item has been accessed by this user.

Input Specification:

Each input file contains one test case. For each test case, the first line contains two positive integers: N (<= 50000), the total number of queries, and K (<= 10), the maximum number of recommendations the system must show to the user. Then given in the second line are the indices of items that the user is accessing — for the sake of simplicity, all the items are indexed from 1 to N. All the numbers in a line are separated by a space.

Output Specification:

For each case, process the queries one by one. Output the recommendations for each query in a line in the format:

query: rec[1] rec[2] … rec[K]

where query is the item that the user is accessing, and rec[i] (i = 1, … K) is the i-th item that the system recommends to the user. The first K items that have been accessed most frequently are supposed to be recommended in non-increasing order of their frequencies. If there is a tie, the items will be ordered by their indices in increasing order.

Note: there is no output for the first item since it is impossible to give any recommendation at the time. It is guaranteed to have the output for at least one query.

Sample Input:

12 3

3 5 7 5 5 3 2 1 8 3 8 12

Sample Output:

5: 3

7: 3 5

5: 3 5 7

5: 5 3 7

3: 5 3 7

2: 5 3 7

1: 5 3 2

8: 5 3 1

3: 5 3 1

8: 3 5 1

12: 3 5 8

题目大意：根据用户每次点击的东西的编号，输出他在点当前编号之前应该给这个用户推荐的商品的编号～只推荐k个～也就是输出用户曾经点击过的商品编号的最多的前k个～如果恰好两个商品有相同的点击次数，就输出编号较小的那个～

分析：【并不复杂～只是写的比较详细～不怕不怕～(摸头...】因为每个商品有两个属性：编号value和出现的次数cnt，编号具有唯一性，然后set又会根据大小自动排序，所以我们可以考虑将value和cnt组成一个node属性，把所有商品编号和它对应的次数变成node放入set里面，重载小于号，使<根据set中node的cnt排序，如果cnt相等就按照node的value排序～

【并不复杂～只是写的比较详细～不怕不怕～(摸头...】

这样set里面就是按照出现次数排序好的商品node，每次输出set的前k个node的value值就可以～（要记住，因为是点击前推荐，所以我们在接收完num的值后，应该先输出再插入让set帮忙排序当前num，当前的这个点击编号暂时先不算在输出结果里面的～）

首先在struct node里面重载<号，如果当前cnt不等于a.cnt就将cnt大的排在前，否则将value小的排在前面～

每次输入的时候，先不插入，先输出，当i != 0时候开始输出，因为i==0时候用户才第一次点击，没有可以推荐的～(沮丧脸...) 输出的同时记录输出过的个数tempCnt，当tempCnt < k的时候输出，因为只要前k个～所以就从头到尾依次输出set中的值就可以啦～

book[num]标记num出现的次数，每次寻找set中当前值为num和次数为book[num]的那个值，如果找到了就把他移除，（找到说明这个数已经出现过啦，cnt已经不对啦，先移除掉吧～）然后将book[num]+1，在将node(num, book[num])插入到set中，set会帮忙根据我们自定义的<的规则自动排序哒～

```
#include <iostream>
#include <set>
using namespace std;
int book[50001];
struct node {
    int value, cnt;
    bool operator < (const node &a) const {
        return (cnt != a.cnt) ? cnt > a.cnt : value < a.value;
    }
};
int main() {
    int n, k, num;
    scanf("%d%d", &n, &k);
    set<node> s;
    for (int i = 0; i < n; i++) {
        scanf("%d", &num);
        if (i != 0) {
            printf("%d:", num);
            int tempCnt = 0;
            for(auto it = s.begin(); tempCnt < k && it != s.end(); it++) {
                printf(" %d", it->value);
                tempCnt++;
            }
            printf("\n");
        }
        auto it = s.find(node{num, book[num]});
        if (it != s.end()) s.erase(it);
        book[num]++;
        s.insert(node{num, book[num]});
```

```
30        }
31        return 0;
32    }
```

1130. Infix Expression (25) [dfs深度优先搜索]

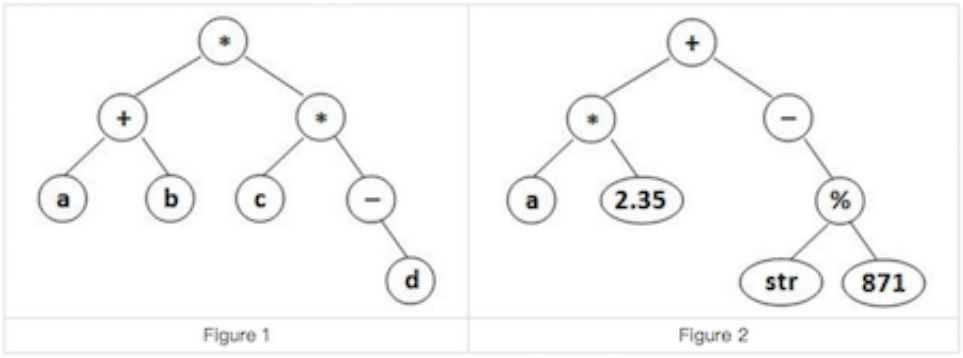
Given a syntax tree (binary), you are supposed to output the corresponding infix expression, with parentheses reflecting the precedences of the operators.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<= 20) which is the total number of nodes in the syntax tree. Then N lines follow, each gives the information of a node (the i-th line corresponds to the i-th node) in the format:

data left_child right_child

where data is a string of no more than 10 characters, left_child and right_child are the indices of this node's left and right children, respectively. The nodes are indexed from 1 to N. The NULL link is represented by -1. The figures 1 and 2 correspond to the samples 1 and 2, respectively.



题目大意：给一个二叉树，输出中缀表达式，且加上括号表示运算的优先级～

分析：首先根据所有孩子结点编号寻找1～n中没有出现过的编号标记为root，即树的根结点～然后进行从root结点开始dfs～

dfs递归拼接 “(” + 左子树 + 根 + 右子树 + “)” 递归有四种情况（有效的只有三种）：

1. 左右子树都空 返回 “(” + 根 + “)”
2. 左空右不空 返回 “(” + 根 + 右子树 + “)”
3. 左不空右空 这种情况不存在
4. 左右都不空 返回 “(” + 左子树 + 根 + 右子树 + “)” 最后递归返回的ans，最外层可能会被括号包起来，也可能不被包起来。要判断一下，如果被包起来，把最外层括号去掉即可～

```
#include <iostream>
using namespace std;
struct node {
    string data;
    int l, r;
}a[100];
string dfs(int root) {
    if(a[root].l == -1 && a[root].r == -1) return a[root].data;
    if(a[root].l == -1 && a[root].r != -1)  return "(" +  a[root].data +
dfs(a[root].r) + ")";
```

```
10        if(a[root].l != -1 && a[root].r != -1) return "(" +  dfs(a[root].l) +
    a[root].data + dfs(a[root].r) + ")";
11    }
12    int main() {
13        int have[100] = {0}, n, root = 1;
14        cin >> n;
15        for(int i  = 1; i <= n; i++) {
16            cin >> a[i].data >> a[i].l >> a[i].r;
17            if(a[i].l != -1) have[a[i].l] = 1;
18            if(a[i].r != -1) have[a[i].r] = 1;
19        }
20        while(have[root] == 1) root++;
21        string ans = dfs(root);
22        if(ans[0] == '(') ans = ans.substr(1,ans.size()-2);
23        cout << ans;
24        return 0;
25    }
```

1131. Subway Map (30) [dfs深度优先搜索]

In the big cities, the subway systems always look so complex to the visitors. To give you some sense, the following figure shows the map of Beijing subway. Now you are supposed to help people with your computer skills! Given the starting position of your user, your task is to find the quickest way to his/her destination.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer N (< =100), the number of subway lines. Then N lines follow, with the i-th (i = 1, …, N) line describes the i-th subway line in the format:

M S[1] S[2] … S[M]

where M (<= 100) is the number of stops, and S[i]'s (i = 1, … M) are the indices of the stations (the indices are 4-digit numbers from 0000 to 9999) along the line. It is guaranteed that the stations are given in the correct order — that is, the train travels between S[i] and S[i+1] (i = 1, …, M-1) without any stop.

Note: It is possible to have loops, but not self-loop (no train starts from S and stops at S without passing through another station). Each station interval belongs to a unique subway line. Although the lines may cross each other at some stations (so called “transfer stations”), no station can be the conjunction of more than 5 lines.

After the description of the subway, another positive integer K (<= 10) is given. Then K lines follow, each gives a query from your user: the two indices as the starting station and the destination, respectively.

The following figure shows the sample map.

Note: It is guaranteed that all the stations are reachable, and all the queries consist of legal station numbers.

Output Specification:

For each query, first print in a line the minimum number of stops. Then you are supposed to show the optimal path in a friendly format as the following:

Take Line#X1 from S1 to S2.

Take Line#X2 from S2 to S3.

……

where Xi’s are the line numbers and Si’s are the station indices. Note: Besides the starting and ending stations, only the transfer stations shall be printed.

If the quickest path is not unique, output the one with the minimum number of transfers, which is guaranteed to be unique.

Sample Input:

4

7 1001 3212 1003 1204 1005 1306 7797

9 9988 2333 1204 2006 2005 2004 2003 2302 2001

13 3011 3812 3013 3001 1306 3003 2333 3066 3212 3008 2302 3010 3011

4 6666 8432 4011 1306

3

3011 3013

6666 2001

2004 3001

Sample Output:

2

Take Line#3 from 3011 to 3013.

10

Take Line#4 from 6666 to 1306.

Take Line#3 from 1306 to 2302.

Take Line#2 from 2302 to 2001.

6

Take Line#2 from 2004 to 1204.

Take Line#1 from 1204 to 1306.

Take Line#3 from 1306 to 3001.

题目大意：找出一条路线，使得对任何给定的起点和终点，可以找出中途经停站最少的路线；如果经停站一样多，则取需要换乘线路次数最少的路线～

【很简单的，别跑^_^】

分析：一遍DFS即可～DFS过程中要维护两个变量：minCnt-中途经停的最少的站; minTransfer-需要换乘的最小次数～

0.可以这样计算出一条线路的换乘次数：在line[10000][10000]的数组中保存每两个相邻站中间的线路是几号线～从头到尾遍历最终保存的路径，preLine为前一小段的线路编号，如果当前的结点和前一个结点组成的这条路的线路编号和preLine不同，说明有一个换乘，就将cnt+1，最后遍历完累加的cnt即是换乘的次数～ update：由于新的PAT系统中会显示原解法有一个测试用例内存超限，考虑到line[10000][10000]存储的方式太暴力了，所以将line换成了unordered_map<int, int> line存储的方式，第一个int用来存储线路，每次将前四位存储第一个线路，后四位存储第二个线路，使用a[i-1]*10000+a[i]的方式存储，第二个int用来保存两个相邻中间的线路是几号线～

update：由于新的PAT系统中会显示原解法有一个测试用例内存超限，考虑到line[10000][10000]存储的方式太暴力了，所以将line换成了unordered_map<int, int> line存储的方式，第一个int用来存储线路，每次将前四位存储第一个线路，后四位存储第二个线路，使用a[i-1]*10000+a[i]的方式存储，第二个int用来保存两个相邻中间的线路是几号线～

1.可以这样计算出一条线路中途停站的次数：在dfs的时候有个变量cnt，表示当前路线是所需乘的第几个站，每次dfs时候将cnt+1表示向下遍历一层～cnt就是当前中途停站的次数～

2.可以这样输出结果：和计算线路换乘次数的思路一样，每当preLine和当前Line值不同的时候就输出一句话～保存preTransfer表示上一个换乘站，最后不要忘记输出preTransfer和最后一个站之间的路即使最后一个站并不是换乘站～

喵呜～

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
vector<vector<int>> v(10000);
int visit[10000], minCnt, minTransfer, start, end1;
unordered_map<int, int> line;
vector<int> path, tempPath;
int transferCnt(vector<int> a) {
    int cnt = -1, preLine = 0;
    for (int i = 1; i < a.size(); i++) {
        if (line[a[i-1]*10000+a[i]] != preLine) cnt++;
        preLine = line[a[i-1]*10000+a[i]];
```

```
    }
    return cnt;
}
void dfs(int node, int cnt) {
    if (node == end1 && (cnt < minCnt || (cnt == minCnt &&
transferCnt(tempPath) < minTransfer))) {
        minCnt = cnt;
        minTransfer = transferCnt(tempPath);
        path = tempPath;
    }
    if (node == end1) return;
    for (int i = 0; i < v[node].size(); i++) {
        if (visit[v[node][i]] == 0) {
            visit[v[node][i]] = 1;
            tempPath.push_back(v[node][i]);
            dfs(v[node][i], cnt + 1);
            visit[v[node][i]] = 0;
            tempPath.pop_back();
        }
    }
}
int main() {
    int n, m, k, pre, temp;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &m, &pre);
        for (int j = 1; j < m; j++) {
            scanf("%d", &temp);
            v[pre].push_back(temp);
            v[temp].push_back(pre);
            line[pre*10000+temp] = line[temp*10000+pre] = i + 1;
            pre = temp;
        }
    }
    scanf("%d", &k);
    for (int i = 0; i < k; i++) {
        scanf("%d%d", &start, &end1);
        minCnt = 99999, minTransfer = 99999;
        tempPath.clear();
        tempPath.push_back(start);
        visit[start] = 1;
        dfs(start, 0);
        visit[start] = 0;
        printf("%d\n", minCnt);
        int preLine = 0, preTransfer = start;
        for (int j = 1; j < path.size(); j++) {
            if (line[path[j-1]*10000+path[j]] != preLine) {
                if (preLine != 0) printf("Take Line#%d from %04d to %04d.\n",
preLine, preTransfer, path[j-1]);
                preLine = line[path[j-1]*10000+path[j]];
                preTransfer = path[j-1];
            }
```

```
64          }
65          printf("Take Line#%d from %04d to %04d.\n", preLine, preTransfer,
    end1);
66      }
67      return 0;
68  }
```

1132. Cut Integer (20) [数学问题]

Cutting an integer means to cut a K digits long integer Z into two integers of (K/2) digits long integers A and B. For example, after cutting Z = 167334, we have A = 167 and B = 334. It is interesting to see that Z can be devided by the product of A and B, as 167334 / (167 x 334) = 3. Given an integer Z, you are supposed to test if it is such an integer.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<= 20). Then N lines follow, each gives an integer Z (10<=Z<=231). It is guaranteed that the number of digits of Z is an even number.

Output Specification:

For each case, print a single line “Yes” if it is such a number, or “No” if not.

Sample Input:

3

167334

2333

12345678

Sample Output:

Yes

No

No

题目大意：给一个偶数个位的正整数num，把它从中间分成左右两个整数a、b，问num能不能被a和b的乘积整除，能的话输出yes，不能的话输出no

分析：要注意a*b如果为0的时候不能取余，否则会浮点错误～

直接用int保存num的值，计算出num的长度len，则令d = pow(10, len / 2)时，num取余d能得到后半部分的整数，num除以d能得到前半部分的整数，计算num % (a*b)是否等于0就可以得知是否可以被整除～

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       int n, num;
5       scanf("%d", &n);
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d", &num);
        string s = to_string(num);
        int len = s.length();
        int a = stoi(s.substr(0, len/2));
        int b = stoi(s.substr(len/2));
        if (a * b != 0 && num % (a * b) == 0)
            printf("Yes\n");
        else
            printf("No\n");
    }
    return 0;
}
```

1133. Splitting A Linked List (25) [链表]

Given a singly linked list, you are supposed to rearrange its elements so that all the negative values appear before all of the non-negatives, and all the values in [0, K] appear before all those greater than K. The order of the elements inside each class must not be changed. For example, given the list being 18→7→-4→0→5→-6→10→11→-2 and K being 10, you must output -4→-6→-2→7→0→5→10→18→11.

Input Specification:

Each input file contains one test case. For each case, the first line contains the address of the first node, a positive N (<= 105) which is the total number of nodes, and a positive K (<=1000). The address of a node is a 5-digit nonnegative integer, and NULL is represented by -1.

Then N lines follow, each describes a node in the format:

Address Data Next

where Address is the position of the node, Data is an integer in [-105, 105], and Next is the position of the next node. It is guaranteed that the list is not empty.

Output Specification:

For each case, output in order (from beginning to the end of the list) the resulting linked list. Each node occupies a line, and is printed in the same format as in the input.

Sample Input:

00100 9 10

23333 10 27777

00000 0 99999

00100 18 12309

68237 -6 23333

33218 -4 00000

48652 -2 -1

99999 5 68237

27777 11 48652

12309 7 33218

Sample Output:

33218 -4 68237

68237 -6 48652

48652 -2 12309

12309 7 00000

00000 0 99999

99999 5 23333

23333 10 00100

00100 18 27777

27777 11 -1

题目大意：给一个链表和K，遍历链表后将<0的结点先输出，再将0～k区间的结点输出，最后输出>k的结点

分析：1.所有节点用结构体｛id, data, next｝存储

2.遍历链表，找出在此链表中的节点，放入容器v中

3.把节点分三类｛（-无穷，0），[0,k], (k,+无穷)｝,把他们按段，按先后顺序依次放进容器ans中，最后输出即可～

```
#include <iostream>
#include <vector>
using namespace std;
struct node {
    int id, data, next;
};
int main() {
    int begin, n, k, s, d, e;
    scanf("%d%d%d", &begin, &n, &k);
    node a[100010];
    vector<node> v, ans;
    for (int i = 0; i < n; i++) {
        scanf("%d%d%d", &s, &d, &e);
        a[s] = {s, d, e};
    }
    for (; begin != -1; begin = a[begin].next)
        v.push_back(a[begin]);
    for (int i = 0; i < v.size(); i++)
        if (v[i].data < 0) ans.push_back(v[i]);
    for (int i = 0; i < v.size(); i++)
        if (v[i].data >= 0 && v[i].data <= k) ans.push_back(v[i]);
    for (int i = 0; i < v.size(); i++)
```

```
            if (v[i].data > k) ans.push_back(v[i]);
        for (int i = 0; i < ans.size() - 1; i++)
            printf("%05d %d %05d\n", ans[i].id, ans[i].data, ans[i + 1].id);
        printf("%05d %d -1\n", ans[ans.size() - 1].id, ans[ans.size() - 1].data);
        return 0;
    }
```

1134. Vertex Cover (25) [hash散列]

A vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set. Now given a graph with several vertex sets, you are supposed to tell if each of them is a vertex cover or not.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers N and M (both no more than 104), being the total numbers of vertices and the edges, respectively. Then M lines follow, each describes an edge by giving the indices (from 0 to N-1) of the two ends of the edge.

After the graph, a positive integer K (<= 100) is given, which is the number of queries. Then K lines of queries follow, each in the format:

Nv v[1] v[2] … v[Nv]

where Nv is the number of vertices in the set, and v[i]’s are the indices of the vertices.

Output Specification:

For each query, print in a line “Yes” if the set is a vertex cover, or “No” if not.

Sample Input:

10 11

8 7

6 8

4 5

8 4

8 1

1 2

1 4

9 8

9 1

1 0

2 4

5

4 0 3 8 4

6 6 1 7 5 4 9

3 1 8 4

2 2 8

7 9 8 7 6 5 4 2

Sample Output:

No

Yes

Yes

No

No

题目大意：给n个结点m条边，再给k个集合。对这k个集合逐个进行判断。每个集合S里面的数字都是结点编号，求问整个图所有的m条边两端的结点，是否至少一个结点出自集合S中。如果是，输出Yes否则输出No

分析：用vector v[n]保存某结点属于的某条边的编号，比如a b两个结点构成的这条边的编号为0，则v[a].push_back(0)，v[b].push_back(0)——表示a属于0号边，b也属于0号边。对于每一个集合做判断，遍历集合中的每一个元素，将当前元素能够属于的边的编号i对应的hash[i]标记为1，表示这条边是满足有一个结点出自集合S中的。最后判断hash数组中的每一个值是否都是1，如果有不是1的，说明这条边的两端结点没有一个出自集合S中，则输出No。否则输出Yes～

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n, m, k, nv, a, b, num;
    scanf("%d%d", &n, &m);
    vector<int> v[n];
    for (int i = 0;i < m; i++) {
        scanf("%d%d", &a, &b);
        v[a].push_back(i);
        v[b].push_back(i);
    }
    scanf("%d", &k);
    for (int i = 0; i < k; i++) {
        scanf("%d", &nv);
        int flag = 0;
        vector<int> hash(m, 0);
        for (int j = 0; j < nv; j++) {
            scanf("%d", &num);
            for (int t = 0; t < v[num].size(); t++)
                hash[v[num][t]] = 1;
        }
        for (int j = 0; j < m; j++) {
```

```
24                 if (hash[j] == 0) {
25                     printf("No\n");
26                     flag = 1;
27                     break;
28                 }
29             }
30             if (flag == 0) printf("Yes\n");
31         }
32         return 0;
33     }
```

1135. Is It A Red-Black Tree (30) [红黑树]

There is a kind of balanced binary search tree named red-black tree in the data structure. It has the following 5 properties:

(1) Every node is either red or black.

(2) The root is black.

(3) Every leaf (NULL) is black.

(4) If a node is red, then both its children are black.

(5) For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

For example, the tree in Figure 1 is a red-black tree, while the ones in Figure 2 and 3 are not.

For each given binary search tree, you are supposed to tell if it is a legal red-black tree.

Input Specification:

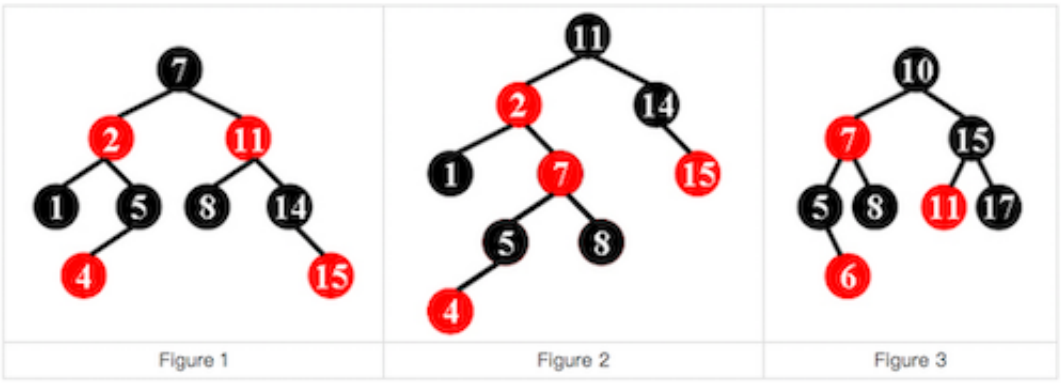


Figure 1 Figure 2 Figure 3

Output Specification:

For each test case, print in a line “Yes” if the given tree is a red-black tree, or “No” if not.

Sample Input:

3

9

7 -2 1 5 -4 -11 8 14 -15

9

11 -2 1 -7 5 -4 8 14 -15

8

10 -7 5 -6 8 15 -11 17

Sample Output:

Yes

No

No

题目大意：给一棵二叉搜索树的前序遍历，判断它是否为红黑树，是输出Yes，否则输出No。

分析：判断以下几点：

1.根结点是否为黑色

2.如果一个结点是红色，它的孩子节点是否都为黑色

3.从任意结点到叶子结点的路径中，黑色结点的个数是否相同

所以分为以下几步：

0. 根据先序建立一棵树，用链表表示
1. 判断根结点（题目所给先序的第一个点即根结点）是否是黑色【arr[0] < 0】
2. 根据建立的树，从根结点开始遍历，如果当前结点是红色，判断它的孩子节点是否为黑色，递归返回结果【judge1函数】
3. 从根节点开始，递归遍历，检查每个结点的左子树的高度和右子树的高度（这里的高度指黑色结点的个数），比较左右孩子高度是否相等，递归返回结果【judge2函数】

注意：终于知道自己PAT考试的时候错在哪了。。。维基百科定义：红黑树（英语：Red–black tree）是一种自平衡二叉查找树。AVL树：在计算机科学中，AVL树是最先发明的自平衡二叉查找树。在AVL树中任何节点的两个子树的高度最大差别为1，所以它也被称为高度平衡树。所以说红黑树不是一种AVL树，红黑树相对于AVL树来说，牺牲了部分平衡性以换取插入/删除操作时少量的旋转操作，整体来说性能要优于AVL树。而我根据先序遍历直接建树后判断了是否AVL平衡，把判断是否平衡的那段代码注释掉就AC了～

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
vector<int> arr;
struct node {
    int val;
    struct node *left, *right;
};
node* build(node *root, int v) {
    if(root == NULL) {
        root = new node();
        root->val = v;
        root->left = root->right = NULL;
    } else if(abs(v) <= abs(root->val))
```

```
            root->left = build(root->left, v);
        else
            root->right = build(root->right, v);
        return root;
    }
    bool judge1(node *root) {
        if (root == NULL) return true;
        if (root->val < 0) {
            if (root->left != NULL && root->left->val < 0) return false;
            if (root->right != NULL && root->right->val < 0) return false;
        }
        return judge1(root->left) && judge1(root->right);
    }
    int getNum(node *root) {
        if (root == NULL) return 0;
        int l = getNum(root->left);
        int r = getNum(root->right);
        return root->val > 0 ? max(l, r) + 1 : max(l, r);
    }
    bool judge2(node *root) {
        if (root == NULL) return true;
        int l = getNum(root->left);
        int r = getNum(root->right);
        if(l != r) return false;
        return judge2(root->left) && judge2(root->right);
    }
    int main() {
        int k, n;
        scanf("%d", &k);
        for (int i = 0; i < k; i++) {
            scanf("%d", &n);
            arr.resize(n);
            node *root = NULL;
            for (int j = 0; j < n; j++) {
                scanf("%d", &arr[j]);
                root = build(root, arr[j]);
            }
            if (arr[0] < 0 || judge1(root) == false || judge2(root) == false)
                printf("No\n");
            else
                printf("Yes\n");
        }
        return 0;
    }
```

1136. A Delayed Palindrome (20) [水题]

Consider a positive integer N written in standard notation with k+1 digits ai as ak...a1a0 with 0 <= ai < 10 for all i and ak > 0. Then N is palindromic if and only if ai = ak-i for all i. Zero is written 0 and is also palindromic by definition.

Non-palindromic numbers can be paired with palindromic ones via a series of operations. First, the non-palindromic number is reversed and the result is added to the original number. If the result is not a palindromic number, this is repeated until it gives a palindromic number. Such number is called a delayed palindrome. (Quoted from https://en.wikipedia.org/wiki/Palindromic_number)

Given any positive integer, you are supposed to find its paired palindromic number.

Input Specification:

Each input file contains one test case which gives a positive integer no more than 1000 digits.

Output Specification:

For each test case, print line by line the process of finding the palindromic number. The format of each line is the following:

A + B = C

where A is the original number, B is the reversed A, and C is their sum. A starts being the input number, and this process ends until C becomes a palindromic number — in this case we print in the last line "C is a palindromic number."; or if a palindromic number cannot be found in 10 iterations, print "Not found in 10 iterations." instead.

Sample Input 1:

97152

Sample Output 1:

97152 + 25179 = 122331

122331 + 133221 = 255552

255552 is a palindromic number.

Sample Input 2:

196

Sample Output 2:

196 + 691 = 887

887 + 788 = 1675

1675 + 5761 = 7436

7436 + 6347 = 13783

13783 + 38731 = 52514

52514 + 41525 = 94039

94039 + 93049 = 187088

187088 + 880781 = 1067869

1067869 + 9687601 = 10755470

10755470 + 07455701 = 18211171

Not found in 10 iterations.

分析：1 将字符串倒置与原字符串比较看是否相等可知s是否为回文串

2 字符串s和它的倒置t相加，只需从头到尾相加然后再倒置（记得要处理最后一个进位carry，如果有进位要在末尾+'1'）

3 倒置可采用algorithm头文件里面的函数reverse(s.begin(), s.end())直接对s进行倒置

```
#include <iostream>
#include <algorithm>
using namespace std;
string rev(string s) {
    reverse(s.begin(), s.end());
    return s;
}
string add(string s1, string s2) {
    string s = s1;
    int carry = 0;
    for (int i = s1.size() - 1; i >= 0; i--) {
        s[i] = (s1[i] - '0' + s2[i] - '0' + carry) % 10 + '0';
        carry = (s1[i] - '0' + s2[i] - '0' + carry) / 10;
    }
    if (carry > 0) s = "1" + s;
    return s;
}
int main() {
    string s, sum;
    int n = 10;
    cin >> s;
    if (s == rev(s)) {
        cout << s << " is a palindromic number.\n";
        return 0;
    }
    while (n--) {
        sum = add(s, rev(s));
        cout << s << " + " << rev(s) << " = " << sum << endl;
        if (sum == rev(sum)) {
            cout << sum << " is a palindromic number.\n";
            return 0;
        }
        s = sum;
    }
    cout << "Not found in 10 iterations.\n";
    return 0;
}
```

1137. Final Grading (25) [map映射，排序]

For a student taking the online course “Data Structures” on China University MOOC (http://www.icourse163.org/), to be qualified for a certificate, he/she must first obtain no less than 200 points from the online programming assignments, and then receive a final grade no less than 60 out of 100. The final grade is calculated by G = (Gmid-termx 40% + Gfinalx 60%) if Gmid-term > Gfinal, or Gfinal will be taken as the final grade G. Here Gmid-term and Gfinal are the student’s scores of the mid-term and the final exams, respectively.

The problem is that different exams have different grading sheets. Your job is to write a program to merge all the grading sheets into one.

Input Specification:

Each input file contains one test case. For each case, the first line gives three positive integers: P , the number of students having done the online programming assignments; M, the number of students on the mid-term list; and N, the number of students on the final exam list. All the numbers are no more than 10,000.

Then three blocks follow. The first block contains P online programming scores Gp’s; the second one contains M mid-term scores Gmid-term’s; and the last one contains N final exam scores Gfinal’s. Each score occupies a line with the format: StudentID Score, where StudentID is a string of no more than 20 English letters and digits, and Score is a nonnegative integer (the maximum score of the online programming is 900, and that of the mid-term and final exams is 100).

Output Specification:

For each case, print the list of students who are qualified for certificates. Each student occupies a line with the format:

StudentID Gp Gmid-term Gfinal G

If some score does not exist, output “-1” instead. The output must be sorted in descending order of their final grades (G must be rounded up to an integer). If there is a tie, output in ascending order of their StudentID’s. It is guaranteed that the StudentID’s are all distinct, and there is at least one qualified student.

Sample Input:

6 6 7

01234 880

a1903 199

ydjh2 200

wehu8 300

dx86w 220

missing 400

ydhfu77 99

wehu8 55

ydjh2 98

dx86w 88

a1903 86

01234 39

ydhfu77 88

a1903 66

01234 58

wehu8 84

ydjh2 82

missing 99

dx86w 81

Sample Output:

missing 400 -1 99 99

ydjh2 200 98 82 88

dx86w 220 88 81 84

wehu8 300 55 84 84

分析：1 因为所有人必须要G编程>=200分，所以用v数组保存所有G编程>=200的人，（一开始gm和gf都为-1），用map映射保存名字所对应v中的下标（为了避免与“不存在”混淆，保存下标+1，当为0时表示该学生的姓名在v中不存在）

2 G期中中出现的名字，如果对应的map并不存在（==0），说明该学生编程成绩不满足条件，则无须保存该学生信息。将存在的人的期中考试成绩更新

3 G期末中出现的名字，也必须保证在map中存在。先更新G期末和G总为新的成绩，当G期末<G期中时再将G总更新为(G期中x 40% + G期末x 60%)

4 将v数组中所有G总满足条件的放入ans数组中，对ans排序（总分递减，总分相同则姓名递增），最后输出ans中的学生信息

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
using namespace std;
struct node {
    string name;
    int gp, gm, gf, g;
};
bool cmp(node a, node b) {
    return a.g != b.g ? a.g > b.g : a.name < b.name;
}
map<string, int> idx;
int main() {
```

```
    int p, m, n, score, cnt = 1;
    cin >> p >> m >> n;
    vector<node> v, ans;
    string s;
    for (int i = 0; i < p; i++) {
        cin >> s >> score;
        if (score >= 200) {
            v.push_back(node{s, score, -1, -1, 0});
            idx[s] = cnt++;
        }
    }
    for (int i = 0; i < m; i++) {
        cin >> s >> score;
        if (idx[s] != 0) v[idx[s] - 1].gm = score;
    }
    for (int i = 0; i < n; i++) {
        cin >> s >> score;
        if (idx[s] != 0) {
            int temp = idx[s] - 1;
            v[temp].gf = v[temp].g = score;
            if (v[temp].gm > v[temp].gf) v[temp].g = int(v[temp].gm * 0.4 +
v[temp].gf * 0.6 + 0.5);
        }
    }
    for (int i = 0; i < v.size(); i++)
        if (v[i].g >= 60) ans.push_back(v[i]);
    sort(ans.begin(), ans.end(), cmp);
    for (int i = 0; i < ans.size(); i++)
        printf("%s %d %d %d %d\n", ans[i].name.c_str(), ans[i].gp, ans[i].gm,
ans[i].gf, ans[i].g);
    return 0;
}
```

1138. Postorder Traversal (25) [树的遍历，前序中序转后序]

Suppose that all the keys in a binary tree are distinct positive integers. Given the preorder and inorder traversal sequences, you are supposed to output the first number of the postorder traversal sequence of the corresponding binary tree.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=50000), the total number of nodes in the binary tree. The second line gives the preorder sequence and the third line gives the inorder sequence. All the numbers in a line are separated by a space.

Output Specification:

For each test case, print in one line the first number of the postorder traversal sequence of the corresponding binary tree.

Sample Input:

7

1 2 3 4 5 6 7

2 3 1 5 4 7 6

Sample Output:

3

分析：一道简单的前序中序转后序，而且只需要知道后序的第一个值，所以可以定义一个变量flag，当post的第一个值已经输出，则flag为true，递归出口处判断flag，可以提前return

ps：经过测试发现没有flag按照正常前序中序转后序也可以AC，但是我在考场上第一次尝试直接转没有提前退出递归的写法并没有能够AC，而是有一部分运行超时，后来增加了flag才AC。可能pat考试时更为严苛，所以对于准备pat考试的小伙伴，建议如果能够缩小运行时间尽量精益求精

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> pre, in;
bool flag = false;
void postOrder(int prel, int inl, int inr) {
    if (inl > inr || flag == true) return;
    int i = inl;
    while (in[i] != pre[prel]) i++;
    postOrder(prel+1, inl, i-1);
    postOrder(prel+i-inl+1, i+1, inr);
    if (flag == false) {
        printf("%d", in[i]);
        flag = true;
    }
}
int main() {
    int n;
    scanf("%d", &n);
    pre.resize(n);
    in.resize(n);
    for (int i = 0; i < n; i++) scanf("%d", &pre[i]);
    for (int i = 0; i < n; i++) scanf("%d", &in[i]);
    postOrder(0, 0, n-1);
    return 0;
}
```

1139. First Contact (30) [水题]

Unlike in nowadays, the way that boys and girls expressing their feelings of love was quite subtle in the early years. When a boy A had a crush on a girl B, he would usually not contact her directly in the first place. Instead, he might ask another boy C, one of his close friends, to ask another girl D, who was a friend of both B and C, to send a message to B — quite a long shot, isn't it? Girls would do analogously.

Here given a network of friendship relations, you are supposed to help a boy or a girl to list all their friends who can possibly help them making the first contact.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers N (1 < N <= 300) and M, being the total number of people and the number of friendship relations, respectively. Then M lines follow, each gives a pair of friends. Here a person is represented by a 4-digit ID. To tell their genders, we use a negative sign to represent girls.

After the relations, a positive integer K (<= 100) is given, which is the number of queries. Then K lines of queries follow, each gives a pair of lovers, separated by a space. It is assumed that the first one is having a crush on the second one.

Output Specification:

For each query, first print in a line the number of different pairs of friends they can find to help them, then in each line print the IDs of a pair of friends.

If the lovers A and B are of opposite genders, you must first print the friend of A who is of the same gender of A, then the friend of B, who is of the same gender of B. If they are of the same gender, then both friends must be in the same gender as theirs. It is guaranteed that each person has only one gender.

The friends must be printed in non-decreasing order of the first IDs, and for the same first ones, in increasing order of the seconds ones.

Sample Input:

10 18

-2001 1001

-2002 -2001

1004 1001

-2004 -2001

-2003 1005

1005 -2001

1001 -2003

1002 1001

1002 -2004

-2004 1001

1003 -2002

-2003 1003

1004 -2002

-2001 -2003

1001 1003

1003 -2001

1002 -2001

-2002 -2003

5

1001 -2001

-2003 1001

1005 -2001

-2002 -2004

1111 -2003

Sample Output:

4

1002 2004

1003 2002

1003 2003

1004 2002

4

2001 1002

2001 1003

2002 1003

2002 1004

0

1

2003 2001

0

分析：1.用数组arr标记两个人是否是朋友（邻接矩阵表示），用v标记所有人的同性朋友（邻接表表示）

2.对于一对想要在一起的A和B，他们需要先找A的所有同性朋友C，再找B的所有同性朋友D，当C和D两人是朋友的时候则可以输出C和D

3.A在寻找同性朋友时，需要避免找到他想要的伴侣B，所以当当前朋友就是B或者B的同性朋友就是A时舍弃该结果

4.输出时候要以4位数的方式输出，所以要%04d

5.如果用int接收一对朋友，-0000和0000对于int来说都是0，将无法得知这个人的性别，也就会影响他找他的同性朋友的那个步骤，所以考虑用字符串接收，因为题目中已经表示会以符号位加四位的方式给出输入，所以只要判断字符串是否大小相等，如果大小相等说明这两个人是同性

6.结果数组因为必须按照第一个人的编号从小到大排序（当第一个相等时按照第二个人编号的从小到大排序），所以要用sort对ans数组排序后再输出结果

Update: 新PAT系统中原代码导致了一个测试点内存超限，使用unordered_map<int, bool> arr 替代二维数组可避免内存超限（2018-05-28更新）

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <unordered_map>
using namespace std;
unordered_map<int, bool> arr;
struct node {
    int a, b;
};
bool cmp(node x, node y) {
    return x.a != y.a ? x.a < y.a : x.b < y.b;
}
int main() {
    int n, m, k;
    scanf("%d%d", &n, &m);
    vector<int> v[10000];
    for (int i = 0; i < m; i++) {
        string a, b;
        cin >> a >> b;
        if (a.length() == b.length()) {
            v[abs(stoi(a))].push_back(abs(stoi(b)));
            v[abs(stoi(b))].push_back(abs(stoi(a)));
        }
        arr[abs(stoi(a)) * 10000 + abs(stoi(b))] = arr[abs(stoi(b)) * 10000 + abs(stoi(a))] = true;
    }
    scanf("%d", &k);
    for (int i = 0; i < k; i++) {
        int c, d;
        cin >> c >> d;
        vector<node> ans;
        for (int j = 0; j < v[abs(c)].size(); j++) {
            for (int k = 0; k < v[abs(d)].size(); k++) {
                if (v[abs(c)][j] == abs(d) || abs(c) == v[abs(d)][k]) continue;
                if (arr[v[abs(c)][j] * 10000 + v[abs(d)][k]] == true)
                    ans.push_back(node{v[abs(c)][j], v[abs(d)][k]});
            }
        }
        sort(ans.begin(), ans.end(), cmp);
        printf("%d\n", int(ans.size()));
        for(int j = 0; j < ans.size(); j++)
            printf("%04d %04d\n", ans[j].a, ans[j].b);
    }
    return 0;
}
```

Mail: 820821725@qq.com UUID: e1343a18-aecf-31b5-baec-8398b3e0d2e7

1140. Look-and-say Sequence (20) [字符串处理]

Look-and-say sequence is a sequence of integers as the following:

D, D1, D111, D113, D11231, D112213111, …

where D is in [0, 9] except 1. The (n+1)st number is a kind of description of the nth number. For example, the 2nd number means that there is one D in the 1st number, and hence it is D1; the 2nd number consists of one D (corresponding to D1) and one 1 (corresponding to 11), therefore the 3rd number is D111; or since the 4th number is D113, it consists of one D, two 1's, and one 3, so the next number must be D11231. This definition works for D = 1 as well. Now you are supposed to calculate the Nth number in a look-and-say sequence of a given digit D.

Input Specification:

Each input file contains one test case, which gives D (in [0, 9]) and a positive integer N (<=40), separated by a space.

Output Specification:

Print in a line the Nth number in a look-and-say sequence of D.

Sample Input:

1 8

Sample Output:

1123123111

题目大意：给两个数字D和n，第一个序列是D，后一个序列描述前一个序列的所有数字以及这个数字出现的次数，比如D出现了1次，那么第二个序列就是D1，对于第二个序列D1，第三个序列这样描述：D出现1次，1出现1次，所以是D111……以此类推，输出第n个序列～

分析：用string s接收所需变幻的数字，每次遍历s，从当前位置i开始，看后面有多少个与s[i]相同，设j处开始不相同，那么临时字符串 t += s[i] + to_string(j – i);然后再将t赋值给s，cnt只要没达到n次就继续加油循环下一次，最后输出s的值～

```
#include <iostream>
using namespace std;
int main() {
    string s;
    int n, j;
    cin >> s >> n;
    for (int cnt = 1; cnt < n; cnt++) {
        string t;
        for (int i = 0; i < s.length(); i = j) {
            for (j = i; j < s.length() && s[j] == s[i]; j++);
            t += s[i] + to_string(j - i);
        }
        s = t;
    }
    cout << s;
    return 0;
}
```

1141. PAT Ranking of Institutions (25) [排序， map STL]

After each PAT, the PAT Center will announce the ranking of institutions based on their students' performances. Now you are asked to generate the ranklist.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<=105), which is the number of testees. Then N lines follow, each gives the information of a testee in the following format:

ID Score School

where "ID" is a string of 6 characters with the first one representing the test level: "B" stands for the basic level, "A" the advanced level and "T" the top level; "Score" is an integer in [0, 100]; and "School" is the institution code which is a string of no more than 6 English letters (case insensitive). Note: it is guaranteed that "ID" is unique for each testee.

Output Specification:

For each case, first print in a line the total number of institutions. Then output the ranklist of institutions in nondecreasing order of their ranks in the following format:

Rank School TWS Ns

where "Rank" is the rank (start from 1) of the institution; "School" is the institution code (all in lower case); "TWS" is the total weighted score which is defined to be the integer part of "ScoreB/1.5 + ScoreA + ScoreT*1.5", where "ScoreX" is the total score of the testees belong to this institution on level X; and "Ns" is the total number of testees who belong to this institution.

The institutions are ranked according to their TWS. If there is a tie, the institutions are supposed to have the same rank, and they shall be printed in ascending order of Ns. If there is still a tie, they shall be printed in alphabetical order of their codes.

Sample Input:

10

A57908 85 Au

B57908 54 LanX

A37487 60 au

T28374 67 CMU

T32486 24 hypu

A66734 92 cmu

B76378 71 AU

A47780 45 lanx

A72809 100 pku

A03274 45 hypu

Sample Output:

5

1 cmu 192 2

1 au 192 3

3 pku 100 1

4 hypu 81 2

4 lanx 81 2

题目大意：给出每个学生的id、分数、学校，学校名称不区分大小写，输出学校排名、学校名称、总加权成绩、学校参赛人数。学校名称输出时候以小写方式输出。

分析：两个map，一个cnt用来存储某学校名称对应的参赛人数，另一个sum计算某学校名称对应的总加权成绩。每次学校名称string school都要转化为全小写，将map中所有学校都保存在vector ans中，类型为node，node中包括学校姓名、加权总分、参赛人数。对ans数组排序，根据题目要求写好cmp函数，最后按要求输出。对于排名的处理：设立pres表示前一个学校的加权总分，如果pres和当前学校的加权总分不同，说明rank等于数组下标+1，否则rank不变～

注意：总加权分数取整数部分是要对最后的总和取整数部分，不能每次都直接用int存储，不然会有一个3分测试点不通过～ PS：之前直接使用map，导致在新的PAT系统中提交后最后一个测试点超时，改成了unordered_map即可AC～

```
#include <iostream>
#include <algorithm>
#include <cctype>
#include <vector>
#include <unordered_map>
using namespace std;
struct node {
    string school;
    int tws, ns;
};
bool cmp(node a, node b) {
    if (a.tws != b.tws)
        return a.tws > b.tws;
    else if (a.ns != b.ns)
        return a.ns < b.ns;
    else
        return a.school < b.school;
}
int main() {
    int n;
    scanf("%d", &n);
    unordered_map<string, int> cnt;
    unordered_map<string, double> sum;
    for (int i = 0; i < n; i++) {
        string id, school;
```

```
        cin >> id;
        double score;
        scanf("%lf", &score);
        cin >> school;
        for (int j = 0; j < school.length(); j++)
            school[j] = tolower(school[j]);
        if (id[0] == 'B')
            score = score / 1.5;
        else if (id[0] == 'T')
            score = score * 1.5;
        sum[school] += score;
        cnt[school]++;
    }
    vector<node> ans;
    for (auto it = cnt.begin(); it != cnt.end(); it++)
        ans.push_back(node{it->first, (int)sum[it->first], cnt[it->first]});
    sort(ans.begin(), ans.end(), cmp);
    int rank = 0, pres = -1;
    printf("%d\n", (int)ans.size());
    for (int i = 0; i < ans.size(); i++) {
        if (pres != ans[i].tws) rank = i + 1;
        pres = ans[i].tws;
        printf("%d ", rank);
        cout << ans[i].school;
        printf(" %d %d\n", ans[i].tws, ans[i].ns);
    }
    return 0;
}
```

1142. Maximal Clique (25) [图论，无向完全图]

A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. A maximal clique is a clique that cannot be extended by including one more adjacent vertex. (Quoted from https://en.wikipedia.org/wiki/Clique_ (graph_theory))

Now it is your job to judge if a given subset of vertices can form a maximal clique.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers Nv (<= 200), the number of vertices in the graph, and Ne, the number of undirected edges. Then Ne lines follow, each gives a pair of vertices of an edge. The vertices are numbered from 1 to Nv.

After the graph, there is another positive integer M (<= 100). Then M lines of query follow, each first gives a positive number K (<= Nv), then followed by a sequence of K distinct vertices. All the numbers in a line are separated by a space.

Output Specification:

For each of the M queries, print in a line “Yes” if the given subset of vertices can form a maximal clique; or if it is a clique but not a maximal clique, print “Not Maximal”; or if it is not a clique at all, print “Not a Clique”.

Sample Input:

8 10

5 6

7 8

6 4

3 6

4 5

2 3

8 2

2 7

5 3

3 4

6

4 5 4 3 6

3 2 8 7

2 2 3

1 1

3 4 3 6

3 3 2 1

Sample Output:

Yes

Yes

Yes

Yes

Not Maximal

Not a Clique

题目大意：clique是一个点集，在一个无向图中，这个点集中任意两个不同的点之间都是相连的。maximal clique是一个clique，这个clique不可以再加入任何一个新的结点构成新的clique。点编号从1～nv，给出ne条边，以一对结点编号的方式给出。然后给出m条询问，每个询问是一个点集合，问这个点集合是否是maximal clique、是否是clique～

分析：先判断是否是clique，即判断是否任意两边都相连；之后判断是否是maximal，即遍历所有不在集合中的剩余的点，看是否存在一个点满足和集合中所有的结点相连，最后如果都满足，那就输出Yes表示是Maximal clique～

```cpp
#include <iostream>
#include <vector>
using namespace std;
int e[210][210];
int main() {
    int nv, ne, m, ta, tb, k;
    scanf("%d %d", &nv, &ne);
    for (int i = 0; i < ne; i++) {
        scanf("%d %d", &ta, &tb);
        e[ta][tb] = e[tb][ta] = 1;
    }
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d", &k);
        vector<int> v(k);
        int hash[210] = {0}, isclique = 1, isMaximal = 1;
        for (int j = 0; j < k; j++) {
            scanf("%d", &v[j]);
            hash[v[j]] = 1;
        }
        for (int j = 0; j < k; j++) {
            if (isclique == 0) break;
            for (int l = j + 1; l < k; l++) {
                if (e[v[j]][v[l]] == 0) {
                    isclique = 0;
                    printf("Not a Clique\n");
                    break;
                }
            }
        }
        if (isclique == 0) continue;
        for (int j = 1; j <= nv; j++) {
            if (hash[j] == 0) {
                for (int l = 0; l < k; l++) {
                    if (e[v[l]][j] == 0) break;
                    if (l == k - 1) isMaximal = 0;
                }
            }
            if (isMaximal == 0) {
                printf("Not Maximal\n");
                break;
            }
        }
        if (isMaximal == 1) printf("Yes\n");
    }
    return 0;
}
```

1143. Lowest Common Ancestor (30) [水题]

The lowest common ancestor (LCA) of two nodes U and V in a tree is the deepest node that has both U and V as descendants.

Mail: 820821725@qq.com UUID: e1343a18-aecf-31b5-baec-8398b3e0d2e7

A binary search tree (BST) is recursively defined as a binary tree which has the following properties:

The left subtree of a node contains only nodes with keys less than the node's key.

The right subtree of a node contains only nodes with keys greater than or equal to the node's key.

Both the left and right subtrees must also be binary search trees.

Given any two nodes in a BST, you are supposed to find their LCA.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers: M (<= 1000), the number of pairs of nodes to be tested; and N (<= 10000), the number of keys in the BST, respectively. In the second line, N distinct integers are given as the preorder traversal sequence of the BST. Then M lines follow, each contains a pair of integer keys U and V. All the keys are in the range of int.

Output Specification:

For each given pair of U and V, print in a line "LCA of U and V is A." if the LCA is found and A is the key. But if A is one of U and V, print "X is an ancestor of Y." where X is A and Y is the other node. If U or V is not found in the BST, print in a line "ERROR: U is not found." or "ERROR: V is not found." or "ERROR: U and V are not found.".

Sample Input:

6 8

6 3 1 2 5 4 8 7

2 5

8 7

1 9

12 -3

0 8

99 99

Sample Output:

LCA of 2 and 5 is 3.

8 is an ancestor of 7.

ERROR: 9 is not found.

ERROR: 12 and -3 are not found.

ERROR: 0 is not found.

ERROR: 99 and 99 are not found.

题目大意：给出一棵二叉搜索树的前序遍历，问结点u和v的共同最低祖先是谁～

分析：map<int, bool> mp用来标记树中所有出现过的结点，遍历一遍pre数组，将当前结点标记为a，如果u和v分别在a的左、右，或者u、v其中一个就是当前a，即(a >= u && a <= v) || (a >= v && a <= u)，说明找到了这个共同最低祖先a，退出当前循环～最后根据要求输出结果即可～

PS：30分的题目30行代码解决，1行1分，惊不惊喜？意不意外？（真的是水题啊...）

```
#include <iostream>
#include <vector>
#include <map>
using namespace std;
map<int, bool> mp;
int main() {
    int m, n, u, v, a;
    scanf("%d %d", &m, &n);
    vector<int> pre(n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &pre[i]);
        mp[pre[i]] = true;
    }
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &u, &v);
        for(int j = 0; j < n; j++) {
            a = pre[j];
            if ((a >= u && a <= v) || (a >= v && a <= u)) break;
        }
        if (mp[u] == false && mp[v] == false)
            printf("ERROR: %d and %d are not found.\n", u, v);
        else if (mp[u] == false || mp[v] == false)
            printf("ERROR: %d is not found.\n", mp[u] == false ? u : v);
        else if (a == u || a == v)
            printf("%d is an ancestor of %d.\n", a, a == u ? v : u);
        else
            printf("LCA of %d and %d is %d.\n", u, v, a);
    }
    return 0;
}
```

1144. The Missing Number (20) [STL， map]

Given N integers, you are supposed to find the smallest positive integer that is NOT in the given list.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (<= 105). Then N integers are given in the next line, separated by spaces. All the numbers are in the range of int.

Output Specification:

Print in a line the smallest positive integer that is missing from the input list.

Sample Input:

10

5 -25 9 6 1 3 4 2 5 17

Sample Output:

7

题目大意：给n个数字，找到不在这个数字列表里面的最小的正整数

分析：将每个数字出现的次数存储在map里面，num从1开始，如果m[num] == 0说明不存在，则输出这个num～

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    int n, a, num = 0;
    cin >> n;
    map<int, int> m;
    for (int i = 0; i < n; i++) {
        cin >> a;
        m[a]++;
    }
    while(++num)
        if (m[num] == 0) break;
    cout << num;
    return 0;
}
```

1145. Hashing – Average Search Time (25) [哈希映射，哈希表，平方探测法]

The task of this problem is simple: insert a sequence of distinct positive integers into a hash table first. Then try to find another sequence of integer keys from the table and output the average search time (the number of comparisons made to find whether or not the key is in the table). The hash function is defined to be “H(key) = key % TSize” where TSize is the maximum size of the hash table. Quadratic probing (with positive increments only) is used to solve the collisions.

Note that the table size is better to be prime. If the maximum size given by the user is not prime, you must re-define the table size to be the smallest prime number which is larger than the size given by the user.

Input Specification:

Each input file contains one test case. For each case, the first line contains two positive numbers: MSize, N, and M, which are the user-defined table size, the number of input numbers, and the number of keys to be found, respectively. All the three numbers are no more than 104. Then N distinct positive integers are given in the next line. All the numbers in a line are separated by a space and are no more than 105.

Output Specification:

For each test case, in case it is impossible to insert some number, print in a line “X cannot be inserted.” where X is the input number. Finally print in a line the average search time for all the M keys, accurate up to 1 decimal place.

Sample Input:

4 5 4

10 6 4 15 11

11 4 15 2

Sample Output:

15 cannot be inserted.

2.8

题目大意：给定一个序列，用平方探测法解决哈希冲突，然后给出m个数字，如果这个数字不能够被插入就输出”X cannot be inserted.”，然后输出这m个数字的平均查找时间

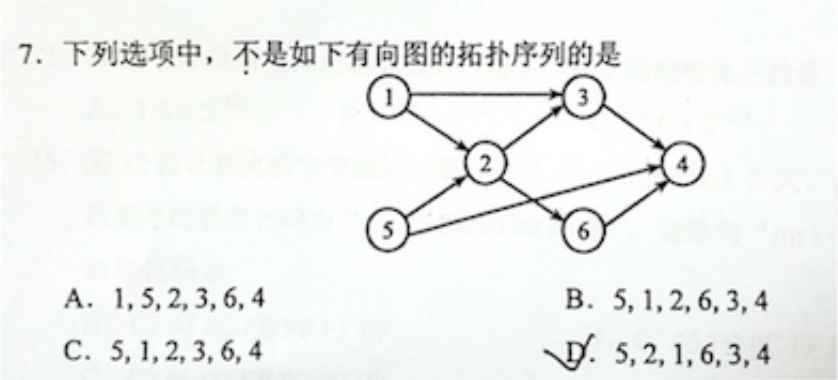
分析：先找到大于tsize的最小的素数为真正的tsize，然后建立一个tsize长度的数组。首先用平方探测法插入数字a，每次pos = (a + j * j) % tsize，j是从0～tsize-1的数字，如果当前位置可以插入就将a赋值给v[pos]，如果一次都没有能够插入成功就输出”X cannot be inserted.”。其次计算平均查找时间，每次计算pos = (a + j * j) % tsize，其中j <= tsize，如果v[pos]处正是a则查找到了，则退出循环，如果v[pos]处不存在数字表示没查找到，那么也要退出循环。每次查找的时候，退出循环之前的j就是这个数字的查找长度。最后ans除以m得到平均查找时间然后输出～

```
#include <iostream>
#include <vector>
using namespace std;
bool isprime(int n) {
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return false;
    return true;
}
int main() {
    int tsize, n, m, a;
    scanf("%d %d %d", &tsize, &n, &m);
    while(!isprime(tsize)) tsize++;
    vector<int> v(tsize);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a);
        int flag = 0;
        for (int j = 0; j < tsize; j++) {
            int pos = (a + j * j) % tsize;
            if (v[pos] == 0) {
                v[pos] = a;
                flag = 1;
                break;
            }
        }
        if (!flag) printf("%d cannot be inserted.\n", a);
    }
    int ans = 0;
    for (int i = 0; i < m; i++) {
        scanf("%d", &a);
        for (int j = 0; j <= tsize; j++) {
            ans++;
            int pos = (a + j * j) % tsize;
```

```
            if (v[pos] == a || v[pos] == 0) break;
        }
    }
    printf("%.1lf\n", ans * 1.0 / m);
    return 0;
}
```

1146. Topological Order (25) [拓扑排序]

This is a problem given in the Graduate Entrance Exam in 2018: Which of the following is NOT a topological order obtained from the given directed graph? Now you are supposed to write a program to test each of the options.



题目大意：给一个有向图，判断给定序列是否是拓扑序列～

分析：用邻接表v存储这个有向图，并将每个节点的入度保存在in数组中。对每一个要判断是否是拓扑序列的结点遍历，如果当前结点的入度不为0则表示不是拓扑序列，每次选中某个点后要将它所指向的所有结点的入度-1，最后根据是否出现过入度不为0的点决定是否要输出当前的编号i～flag是用来判断之前是否输出过现在是否要输出空格的～judge是用来判断是否是拓扑序列的～

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n, m, a, b, k, flag = 0, in[1010];
    vector<int> v[1010];
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &a, &b);
        v[a].push_back(b);
        in[b]++;
    }
    scanf("%d", &k);
    for (int i = 0; i < k; i++) {
        int judge = 1;
        vector<int> tin(in, in+n+1);
        for (int j = 0; j < n; j++) {
            scanf("%d", &a);
            if (tin[a] != 0) judge = 0;
            for (int it : v[a]) tin[it]--;
        }
        if (judge == 1) continue;
```

```
            printf("%s%d", flag == 1 ? " ": "", i);
            flag = 1;
        }
        return 0;
    }
```

1147. Heaps (30) [堆，树的遍历]

In computer science, a heap is a specialized tree-based data structure that satisfies the heap property: if P is a parent node of C, then the key (the value) of P is either greater than or equal to (in a max heap) or less than or equal to (in a min heap) the key of C. A common implementation of a heap is the binary heap, in which the tree is a complete binary tree. (Quoted from Wikipedia at https://en.wikipedia.org/wiki/Heap_ (data_structure))

Your job is to tell if a given complete binary tree is a heap.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers: M (<= 100), the number of trees to be tested; and N (1 < N <= 1000), the number of keys in each tree, respectively. Then M lines follow, each contains N distinct integer keys (all in the range of int), which gives the level order traversal sequence of a complete binary tree.

Output Specification:

For each given tree, print in a line “Max Heap” if it is a max heap, or “Min Heap” for a min heap, or “Not Heap” if it is not a heap at all. Then in the next line print the trees postorder traversal sequence. All the numbers are separated by a space, and there must no extra space at the beginning or the end of the line.

Sample Input:

3 8

98 72 86 60 65 12 23 50

8 38 25 58 52 82 70 60

10 28 15 12 34 9 8 56

Sample Output:

Max Heap

50 60 65 72 12 23 86 98

Min Heap

60 58 52 38 82 70 25 8

Not Heap

56 12 34 28 9 8 15 10

题目大意：给一个树的层序遍历，判断它是不是堆，是大顶堆还是小顶堆。输出这个树的后序遍历～

分析：30分大题，28行代码，一行代码一分……（(⊙o⊙)嗯） // 我为什么这么机智可爱又伶俐?

首先根据v[0]和v[1]的大小比较判断可能是大顶还是小顶，分别赋值flag为1和-1，先根据层序遍历，从0～(n-1)/2【所有有孩子的结点】判断他们的孩子是不是满足flag的要求，如果有一个结点不满足，那就将flag=0表示这不是一个堆。根据flag输出是否是堆，大顶堆还是小顶堆，然后后序遍历，根据index分别遍历index*2+1和index*2+2，即他们的左右孩子，遍历完左右子树后输出根结点，即完成了后序遍历～

```
#include <iostream>
#include <vector>
using namespace std;
int m, n;
vector<int> v;
void postOrder(int index) {
    if (index >= n) return;
    postOrder(index * 2 + 1);
    postOrder(index * 2 + 2);
    printf("%d%s", v[index], index == 0 ? "\n" : " ");
}
int main() {
    scanf("%d%d", &m, &n);
    v.resize(n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) scanf("%d", &v[j]);
        int flag = v[0] > v[1] ? 1 : -1;
        for (int j = 0; j <= (n-1) / 2; j++) {
            int left = j * 2 + 1, right = j * 2 + 2;
            if (flag == 1 && (v[j] < v[left] || (right < n && v[j] < v[right]))) flag = 0;
            if (flag == -1 && (v[j] > v[left] || (right < n && v[j] > v[right]))) flag = 0;
        }
        if (flag == 0) printf("Not Heap\n");
        else printf("%s Heap\n", flag == 1 ? "Max" : "Min");
        postOrder(0);
    }
    return 0;
}
```

1148. Werewolf – Simple Version (20) [水题]

Werewolf（狼人杀） is a game in which the players are partitioned into two parties: the werewolves and the human beings. Suppose that in a game,

player #1 said: “Player #2 is a werewolf.”;

player #2 said: “Player #3 is a human.”;

player #3 said: “Player #4 is a werewolf.”;

player #4 said: “Player #5 is a human.”; and

player #5 said: “Player #4 is a human.”.

Given that there were 2 werewolves among them, at least one but not all the werewolves were lying, and there were exactly 2 liers. Can you point out the werewolves?

Now you are asked to solve a harder version of this problem: given that there were N players, with 2 werewolves among them, at least one but not all the werewolves were lying, and there were exactly 2 liers. You are supposed to point out the werewolves.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N ($5 \le N \le 100$). Then N lines follow and the i-th line gives the statement of the i-th player ($1 \le i \le N$), which is represented by the index of the player with a positive sign for a human and a negative sign for a werewolf.

Output Specification:

If a solution exists, print in a line in ascending order the indices of the two werewolves. The numbers must be separated by exactly one space with no extra spaces at the beginning or the end of the line. If there are more than one solution, you must output the smallest solution sequence — that is, for two sequences $A=a[1],\ldots,a[M]$ and $B=b[1],\ldots,b[M]$, if there exists $0 \le k[k+1] < b[k+1]$, then A is said to be smaller than B. In case there is no solution, simply print No Solution.

Sample Input 1:

5

-2

+3

-4

+5

+4

Sample Output 1:

1 4

Sample Input 2:

6

+6

+3

+1

-5

-2

+4

Sample Output 2 (the solution is not unique):

1 5

Sample Input 3:

5

-2

-3

-4

-5

-1

Sample Output 3:

No Solution

题目大意：已知 N 名玩家中有 2 人扮演狼人角色，有 2 人说的不是实话，有狼人撒谎但并不是所有狼人都在撒谎。要求你找出扮演狼人角色的是哪几号玩家，如果有解，在一行中按递增顺序输出 2 个狼人的编号；如果解不唯一，则输出最小序列解；若无解则输出 No Solution～

分析：每个人说的数字保存在v数组中，i从1～n、j从i+1～n遍历，分别假设i和j是狼人，a数组表示该人是狼人还是好人，等于1表示是好人，等于-1表示是狼人。k从1～n分别判断k所说的话是真是假，k说的话和真实情况不同（即v[k] * a[abs(v[k])] < 0）则表示k在说谎，则将k放在lie数组中；遍历完成后判断lie数组，如果说谎人数等于2并且这两个说谎的人一个是好人一个是狼人（即a[lie[0]] + a[lie[1]] == 0）表示满足题意，此时输出i和j并return，否则最后的时候输出No Solution～

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<int> v(n+1);
    for (int i = 1; i <= n; i++) cin >> v[i];
    for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            vector<int> lie, a(n + 1, 1);
            a[i] = a[j] = -1;
            for (int k = 1; k <= n; k++)
                if (v[k] * a[abs(v[k])] < 0) lie.push_back(k);
            if (lie.size() == 2 && a[lie[0]] + a[lie[1]] == 0) {
                cout << i << " " << j;
                return 0;
            }
        }
    }
    cout << "No Solution";
    return 0;
}
```

1149. Dangerous Goods Packaging (25) [STL的应用]

When shipping goods with containers, we have to be careful not to pack some incompatible goods into the same container, or we might get ourselves in serious trouble. For example, oxidizing agent （氧化剂） must not be packed with flammable liquid （易燃液体）, or it can cause explosion.

Now you are given a long list of incompatible goods, and several lists of goods to be shipped. You are supposed to tell if all the goods in a list can be packed into the same container.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers: N (≤10^4), the number of pairs of incompatible goods, and M (≤100), the number of lists of goods to be shipped.

Then two blocks follow. The first block contains N pairs of incompatible goods, each pair occupies a line; and the second one contains M lists of goods to be shipped, each list occupies a line in the following format:

K G[1] G[2] … G[K]

where K (≤1,000) is the number of goods and G[i]'s are the IDs of the goods. To make it simple, each good is represented by a 5-digit ID number. All the numbers in a line are separated by spaces.

Output Specification:

For each shipping list, print in a line Yes if there are no incompatible goods in the list, or No if not.

Sample Input:

6 3

20001 20002

20003 20004

20005 20006

20003 20001

20005 20004

20004 20006

4 00001 20004 00002 20003

5 98823 20002 20003 20006 10010

3 12345 67890 23333

Sample Output:

No

Yes

Yes

题目大意：集装箱运输货物时，我们必须特别小心，不能把不相容的货物装在一只箱子里。比如氧化剂绝对不能跟易燃液体同箱，否则很容易造成爆炸。给定一张不相容物品的清单，需要你检查每一张集装箱货品清单，判断它们是否能装在同一只箱子里。对每箱货物清单，判断是否可以安全运输。如果没有不相容物品，则在一行中输出 Yes，否则输出 No～

分析：用map存储每一个货物的所有不兼容货物～在判断给出的一堆货物是否是相容的时候，判断任一货物的不兼容货物是否在这堆货物中～如果存在不兼容的货物，则这堆货物不能相容～如果遍历完所有的货物，都找不到不兼容的两个货物，则这堆货物就是兼容的～

```
#include <iostream>
#include <vector>
#include <map>
using namespace std;
int main() {
    int n, k, t1, t2;
    map<int,vector<int>> m;
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &t1, &t2);
        m[t1].push_back(t2);
        m[t2].push_back(t1);
    }
    while (k--) {
        int cnt, flag = 0, a[100000] = {0};
        scanf("%d", &cnt);
        vector<int> v(cnt);
        for (int i = 0; i < cnt; i++) {
            scanf("%d", &v[i]);
            a[v[i]] = 1;
        }
        for (int i = 0; i < v.size(); i++)
            for (int j = 0; j < m[v[i]].size(); j++)
                if (a[m[v[i]][j]] == 1) flag = 1;
        printf("%s\n",flag ? "No" :"Yes");
    }
    return 0;
}
```

1150. Travelling Salesman Problem (25) [图论]

The “travelling salesman problem” asks the following question: “Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?” It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science. (Quoted from “ https://en.wikipedia.org/wiki/Travelling_salesman_problem ”.)

In this problem, you are supposed to find, from a given list of cycles, the one that is the closest to the solution of a travelling salesman problem.

Input Specification:

Each input file contains one test case. For each case, the first line contains 2 positive integers N (2<N≤200), the number of cities, and M, the number of edges in an undirected graph. Then M lines follow, each describes an edge in the format City1 City2 Dist, where the cities are numbered from 1 to N and the distance Dist is positive and is no more than 100. The next line gives a positive integer K which is the number of paths, followed by K lines of paths, each in the format:

n C1 C2 ……Cn

where n is the number of cities in the list, and C

i

‘s are the cities on a path.

Output Specification:

For each path, print in a line Path X: TotalDist (Description) where X is the index (starting from 1) of that path, TotalDist its total distance (if this distance does not exist, output NA instead), and Description is one of the following:

TS simple cycle if it is a simple cycle that visits every city;

TS cycle if it is a cycle that visits every city, but not a simple cycle;

Not a TS cycle if it is NOT a cycle that visits every city.

Finally print in a line Shortest Dist(X) = TotalDist where X is the index of the cycle that is the closest to the solution of a travelling salesman problem, and TotalDist is its total distance. It is guaranteed that such a solution is unique.

Sample Input:

6 10

6 2 1

3 4 1

1 5 1

2 5 1

3 1 8

4 1 6

1 6 1

6 3 1

1 2 1

4 5 1

7

7 5 1 4 3 6 2 5

7 6 1 3 4 5 2 6

6 5 1 4 3 6 2

9 6 2 1 6 3 4 5 2 6

4 1 2 5 1

7 6 1 2 5 4 3 1

7 6 3 2 5 4 1 6

Sample Output:

Path 1: 11 (TS simple cycle)

Path 2: 13 (TS simple cycle)

Path 3: 10 (Not a TS cycle)

Path 4: 8 (TS cycle)

Path 5: 3 (Not a TS cycle)

Path 6: 13 (Not a TS cycle)

Path 7: NA (Not a TS cycle)

Shortest Dist(4) = 8

题目大意：给出一条路径，判断这条路径是这个图的旅行商环路、简单旅行商环路还是非旅行商环路～

分析：如果给出的路径存在某两个连续的点不可达或者第一个结点和最后一个结点不同或者这个路径没有访问过图中所有的点，那么它就不是一个旅行商环路(flag = 0)～如果满足了旅行商环路的条件，那么再判断这个旅行商环路是否是简单旅行商环路，即是否访问过n+1个结点（源点访问两次）～最后输出这些旅行商环路中经过的路径最短的路径编号和路径长度～

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;
int e[300][300], n, m, k, ans = 99999999, ansid;
vector<int> v;
void check(int index) {
    int sum = 0, cnt, flag = 1;
    scanf("%d", &cnt);
    set<int> s;
    vector<int> v(cnt);
    for (int i = 0; i < cnt; i++) {
        scanf("%d", &v[i]);
        s.insert(v[i]);
    }
    for (int i = 0; i < cnt - 1; i++) {
        if(e[v[i]][v[i+1]] == 0) flag = 0;
        sum += e[v[i]][v[i+1]];
    }
    if (flag == 0) {
```

```
            printf("Path %d: NA (Not a TS cycle)\n", index);
        } else if(v[0] != v[cnt-1] || s.size() != n) {
            printf("Path %d: %d (Not a TS cycle)\n", index, sum);
        } else if(cnt != n + 1) {
            printf("Path %d: %d (TS cycle)\n", index, sum);
            if (sum < ans) {
                ans = sum;
                ansid = index;
            }
        } else {
            printf("Path %d: %d (TS simple cycle)\n", index, sum);
            if (sum < ans) {
                ans = sum;
                ansid = index;
            }
        }
    }
    int main() {
        scanf("%d%d", &n, &m);
        for (int i = 0; i < m; i++) {
            int t1, t2, t;
            scanf("%d%d%d", &t1, &t2, &t);
            e[t1][t2] = e[t2][t1] = t;
        }
        scanf("%d", &k);
        for (int i = 1; i <= k; i++) check(i);
        printf("Shortest Dist(%d) = %d\n", ansid, ans);
        return 0;
    }
```

1151. LCA in a Binary Tree (30) [树的遍历，LCA算法]

The lowest common ancestor (LCA) of two nodes U and V in a tree is the deepest node that has both U and V as descendants.

Given any two nodes in a binary tree, you are supposed to find their LCA.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers: M (≤ 1,000), the number of pairs of nodes to be tested; and N (≤ 10,000), the number of keys in the binary tree, respectively. In each of the following two lines, N distinct integers are given as the inorder and preorder traversal sequences of the binary tree, respectively. It is guaranteed that the binary tree can be uniquely determined by the input sequences. Then M lines follow, each contains a pair of integer keys U and V. All the keys are in the range of int.

Output Specification:

For each given pair of U and V, print in a line LCA of U and V is A. if the LCA is found and A is the key. But if A is one of U and V, print X is an ancestor of Y. where X is A and Y is the other node. If U or V is not found in the binary tree, print in a line ERROR: U is not found. or ERROR: V is not found. or ERROR: U and V are not found..

Sample Input:

6 8

7 2 3 4 6 5 1 8

5 3 7 2 6 4 8 1

2 6

8 1

7 9

12 -3

0 8

99 99

Sample Output:

LCA of 2 and 6 is 3.

8 is an ancestor of 1.

ERROR: 9 is not found.

ERROR: 12 and -3 are not found.

ERROR: 0 is not found.

ERROR: 99 and 99 are not found.

题目大意：给出中序序列和先序序列，再给出两个点，求这两个点的最近公共祖先～

分析：不用建树～已知某个树的根结点，若a和b在根结点的左边，则a和b的最近公共祖先在当前子树根结点的左子树寻找，如果a和b在当前子树根结点的两边，在当前子树的根结点就是a和b的最近公共祖先，如果a和b在当前子树根结点的右边，则a和b的最近公共祖先就在当前子树的右子树寻找。中序加先序可以唯一确定一棵树，在不构建树的情况下，在每一层的递归中，可以得到树的根结点，在此时并入lca算法可以确定两个结点的公共祖先～

```
#include <iostream>
#include <vector>
#include <map>
using namespace std;
map<int, int> pos;
vector<int> in, pre;
void lca(int inl, int inr, int preRoot, int a, int b) {
    if (inl > inr) return;
    int inRoot = pos[pre[preRoot]], aIn = pos[a], bIn = pos[b];
    if (aIn < inRoot && bIn < inRoot)
        lca(inl, inRoot-1, preRoot+1, a, b);
    else if ((aIn < inRoot && bIn > inRoot) || (aIn > inRoot && bIn < inRoot))
        printf("LCA of %d and %d is %d.\n", a, b, in[inRoot]);
    else if (aIn > inRoot && bIn > inRoot)
        lca(inRoot+1, inr, preRoot+1+(inRoot-inl), a, b);
```

```
        else if (aIn == inRoot)
                printf("%d is an ancestor of %d.\n", a, b);
        else if (bIn == inRoot)
                printf("%d is an ancestor of %d.\n", b, a);
    }
    int main() {
        int m, n, a, b;
        scanf("%d %d", &m, &n);
        in.resize(n + 1), pre.resize(n + 1);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &in[i]);
            pos[in[i]] = i;
        }
        for (int i = 1; i <= n; i++) scanf("%d", &pre[i]);
        for (int i = 0; i < m; i++) {
            scanf("%d %d", &a, &b);
            if (pos[a] == 0 && pos[b] == 0)
                printf("ERROR: %d and %d are not found.\n", a, b);
            else if (pos[a] == 0 || pos[b] == 0)
                printf("ERROR: %d is not found.\n", pos[a] == 0 ? a : b);
            else
                lca(1, n, 1, a, b);
        }
        return 0;
    }
```

1152. Google Recruitment (20) [字符串处理]

In July 2004, Google posted on a giant billboard along Highway 101 in Silicon Valley (shown in the picture below) for recruitment. The content is super-simple, a URL consisting of the first 10-digit prime found in consecutive digits of the natural constant e. The person who could find this prime number could go to the next step in Google's hiring process by visiting this website.

The natural constant e is a well known transcendental number（超越数）. The first several digits are: e = 2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742746639193200305992 1… where the 10 digits in bold are the answer to Google's question.

Now you are asked to solve a more general problem: find the first K-digit prime in consecutive digits of any given L-digit number.

Input Specification:

Each input file contains one test case. Each case first gives in a line two positive integers: L (≤ 1,000) and K (< 10), which are the numbers of digits of the given number and the prime to be found, respectively. Then the L-digit number N is given in the next line.

Output Specification:

For each test case, print in a line the first K-digit prime in consecutive digits of N. If such a number does not exist, output 404 instead. Note: the leading zeroes must also be counted as part of the K digits. For example, to find the 4-digit prime in 200236, 0023 is a solution. However the first digit 2 must not be treated as a solution 0002 since the leading zeroes are not in the original number.

Sample Input 1:

20 5

23654987725541023819

Sample Output 1:

49877

Sample Input 2:

10 3

2468024680

Sample Output 2:

404

题目大意：给出一个l长度的字符串，求出其中第一个k位的素数

分析: 枚举每个k位的子串，转换成整数，判断是否是素数（判断素数的时候要把0和1也考虑进去）～

```
#include <iostream>
#include <string>
using namespace std;
bool isPrime(int n) {
    if (n == 0 || n == 1) return false;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return false;
    return true;
}
int main() {
    int l, k;
    string s;
    cin >> l >> k >> s;
    for (int i = 0; i <= l - k; i++) {
        string t = s.substr(i, k);
        int num = stoi(t);
```

```
            if (isPrime(num)) {
                cout << t;
                return 0;
            }
        }
        cout << "404\n";
        return 0;
    }
```

1153. Decode Registration Card of PAT (25) [模拟，排序，map]

A registration card number of PAT consists of 4 parts:

the 1st letter represents the test level, namely, T for the top level, A for advance and B for basic;

the 2nd – 4th digits are the test site number, ranged from 101 to 999;

the 5th – 10th digits give the test date, in the form of yymmdd;

finally the 11th – 13th digits are the testee's number, ranged from 000 to 999.

Now given a set of registration card numbers and the scores of the card owners, you are supposed to output the various statistics according to the given queries.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers N (≤10

4

) and M (≤100), the numbers of cards and the queries, respectively.

Then N lines follow, each gives a card number and the owner's score (integer in [0,100]), separated by a space.

After the info of testees, there are M lines, each gives a query in the format Type Term, where

Type being 1 means to output all the testees on a given level, in non-increasing order of their scores. The corresponding Term will be the letter which specifies the level;

Type being 2 means to output the total number of testees together with their total scores in a given site. The corresponding Term will then be the site number;

Type being 3 means to output the total number of testees of every site for a given test date. The corresponding Term will then be the date, given in the same format as in the registration card.

Output Specification:

For each query, first print in a line Case #: input, where # is the index of the query case, starting from 1; and input is a copy of the corresponding input query. Then output as requested:

for a type 1 query, the output format is the same as in input, that is, CardNumber Score. If there is a tie of the scores, output in increasing alphabetical order of their card numbers (uniqueness of the card numbers is guaranteed);

for a type 2 query, output in the format Nt Ns where Nt is the total number of testees and Ns is their total score;

for a type 3 query, output in the format Site Nt where Site is the site number and Nt is the total number of testees at Site. The output must be in non-increasing order of Nt's, or in increasing order of site numbers if there is a tie of Nt.

If the result of a query is empty, simply print NA.

Sample Input:

8 4

B123180908127 99

B102180908003 86

A112180318002 98

T107150310127 62

A107180908108 100

T123180908010 78

B112160918035 88

A107180908021 98

1 A

2 107

3 180908

2 999

Sample Output:

Case 1: 1 A

A107180908108 100

A107180908021 98

A112180318002 98

Case 2: 2 107

3 260

Case 3: 3 180908

107 2

123 2

102 1

Case 4: 2 999

NA

题目大意：给出一组学生的准考证号和成绩，准考证号包含了等级(乙甲顶)，考场号，日期，和个人编号信息，并有三种查询方式

查询一：给出考试等级，找出该等级的考生，按照成绩降序，准考证升序排序

查询二：给出考场号，统计该考场的考生数量和总得分

查询三：给出考试日期，查询改日期下所有考场的考试人数，按照人数降序，考场号升序排序

分析：先把所有考生的准考证和分数记录下来～

1、按照等级查询，枚举选取匹配的学生，然后排序即可

2、按照考场查询，枚举选取匹配的学生，然后计数、求和

3、按日期查询每个考场人数，用unordered_map存储，最后排序汇总～

注意：1、第三个用map存储会超时，用unordered_map就不会超时啦～

2、排序传参建议用引用传参，这样更快，虽然有时候不用引用传参也能通过，但还是尽量用，养成好习惯～

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;
struct node {
    string t;
    int value;
};
bool cmp(const node &a, const node &b) {
    return a.value != b.value ? a.value > b.value : a.t < b.t;
}
int main() {
    int n, k, num;
    string s;
    cin >> n >> k;
    vector<node> v(n);
    for (int i = 0; i < n; i++)
        cin >> v[i].t >> v[i].value;
    for (int i = 1; i <= k; i++) {
        cin >> num >> s;
        printf("Case %d: %d %s\n", i, num, s.c_str());
        vector<node> ans;
        int cnt = 0, sum = 0;
        if (num == 1) {
            for (int j = 0; j < n; j++)
                if (v[j].t[0] == s[0]) ans.push_back(v[j]);
        } else if (num == 2) {
            for (int j = 0; j < n; j++) {
                if (v[j].t.substr(1, 3) == s) {
                    cnt++;
                    sum += v[j].value;
                }
```

```
            }
            if (cnt != 0) printf("%d %d\n", cnt, sum);
        } else if (num == 3) {
            unordered_map<string, int> m;
            for (int j = 0; j < n; j++)
                if (v[j].t.substr(4, 6) == s) m[v[j].t.substr(1, 3)]++;
            for (auto it : m) ans.push_back({it.first, it.second});
        }
        sort(ans.begin(), ans.end(),cmp);
        for (int j = 0; j < ans.size(); j++)
            printf("%s %d\n", ans[j].t.c_str(), ans[j].value);
        if (((num == 1 || num == 3) && ans.size() == 0) || (num == 2 && cnt ==
0)) printf("NA\n");
    }
    return 0;
}
```

1154. Vertex Coloring (25) [set,hash]

A proper vertex coloring is a labeling of the graph's vertices with colors such that no two vertices sharing the same edge have the same color. A coloring using at most k colors is called a (proper) k-coloring.

Now you are supposed to tell if a given coloring is a proper k-coloring.

Input Specification:

Each input file contains one test case. For each case, the first line gives two positive integers N and M (both no more than 10

4

), being the total numbers of vertices and edges, respectively. Then M lines follow, each describes an edge by giving the indices (from 0 to N−1) of the two ends of the edge.

After the graph, a positive integer K (≤ 100) is given, which is the number of colorings you are supposed to check. Then K lines follow, each contains N colors which are represented by non-negative integers in the range of int. The i-th color is the color of the i-th vertex.

Output Specification:

For each coloring, print in a line k-coloring if it is a proper k-coloring for some positive k, or No if not.

Sample Input:

10 11

8 7

6 8

4 5

8 4

8 1

1 2

1 4

9 8

9 1

1 0

2 4

4

0 1 0 1 4 1 0 1 3 0

0 1 0 1 4 1 0 1 0 0

8 1 0 1 4 1 0 5 3 0

1 2 3 4 5 6 7 8 8 9

Sample Output:

4-coloring

No

6-coloring

No

题目大意：给出一个图（先给出所有边，后给出每个点的颜色），问是否满足：所有的边的两个点的颜色不相同

分析：把所有边存起来，把所有点的颜色存起来（存的过程中放入set统计颜色个数），枚举所有边，检查是否每条边的两点个颜色是否相同，若全不相同，则输出颜色个数，否则输出No～

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;
struct node {int t1, t2;};
int main() {
    int n, m, k;
    cin >> n >> m;
    vector<node> v(m);
    for (int i = 0; i < m; i++)
        scanf("%d %d", &v[i].t1, &v[i].t2);
    cin >> k;
    while (k--) {
        int a[10009] = {0};
        bool flag = true;
        set<int> se;
        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i]);
            se.insert(a[i]);
```

```
            }
            for (int i = 0; i < m; i++) {
                if (a[v[i].t1] == a[v[i].t2]) {
                    flag = false;
                    break;
                }
            }
            if (flag)
                printf("%d-coloring\n", se.size());
            else
                printf("No\n");
        }
        return 0;
    }
```

1155. Heap Paths (30) [深搜回溯，堆]

In computer science, a heap is a specialized tree-based data structure that satisfies the heap property: if P is a parent node of C, then the key (the value) of P is either greater than or equal to (in a max heap) or less than or equal to (in a min heap) the key of C. A common implementation of a heap is the binary heap, in which the tree is a complete binary tree. (Quoted from Wikipedia at https://en.wikipedia.org/wiki/Heap_ (data_structure))

One thing for sure is that all the keys along any path from the root to a leaf in a max/min heap must be in non-increasing/non-decreasing order.

Your job is to check every path in a given complete binary tree, in order to tell if it is a heap or not.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer N (1<N≤1,000), the number of keys in the tree. Then the next line contains N distinct integer keys (all in the range of int), which gives the level order traversal sequence of a complete binary tree.

Output Specification:

For each given tree, first print all the paths from the root to the leaves. Each path occupies a line, with all the numbers separated by a space, and no extra space at the beginning or the end of the line. The paths must be printed in the following order: for each node in the tree, all the paths in its right subtree must be printed before those in its left subtree.

Finally print in a line Max Heap if it is a max heap, or Min Heap for a min heap, or Not Heap if it is not a heap at all.

Sample Input 1:

8

98 72 86 60 65 12 23 50

Sample Output 1:

98 86 23

98 86 12

98 72 65

98 72 60 50

Max Heap

Sample Input 2:

8

8 38 25 58 52 82 70 60

Sample Output 2:

8 25 70

8 25 82

8 38 52

8 38 58 60

Min Heap

Sample Input 3:

8

10 28 15 12 34 9 8 56

Sample Output 3:

10 15 8

10 15 9

10 28 34

10 28 12 56

Not Heap

题目大意：给出一颗完全二叉树，打印出从根节点到所有叶节点的路径，打印顺序先右后左，即先序遍历的镜像。然后判断该树是大顶堆、小顶堆或者不是堆～

分析：1.深搜打印出所有路径（从右往左，即先序的镜像），vector保存一路上的节点，通过push和pop回溯，维护路径，index <= n是对只有左叶节点没有右叶节点的点特判

2.判断是否为堆：从第二个节点开始遍历，如果比父节点小，就不是小顶堆，如果比父节点大，就不是大顶堆～

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> v;
int a[1009], n, isMin = 1, isMax = 1;
void dfs(int index) {
    if (index * 2 > n && index * 2 + 1 > n) {
        if (index <= n) {
```

```
            for (int i = 0; i < v.size(); i++)
                printf("%d%s", v[i], i != v.size() - 1 ? " " : "\n");
        }
    } else {
        v.push_back(a[index * 2 + 1]);
        dfs(index * 2 + 1);
        v.pop_back();
        v.push_back(a[index * 2]);
        dfs(index * 2);
        v.pop_back();
    }
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    v.push_back(a[1]);
    dfs(1);
    for (int i = 2; i <= n; i++) {
        if (a[i/2] > a[i]) isMin = 0;
        if (a[i/2] < a[i]) isMax = 0;
    }
    if (isMin == 1)
        printf("Min Heap");
    else
        printf("%s", isMax == 1 ? "Max Heap" : "Not Heap");
    return 0;
}
```