1001 A+B format

题目的意思就是输入两个初始的参数，然后按照某种标准的形式进行输出。 将输入的两个数进行结果到字符串的转化一下，然后按照规则进行逗号找准位置插入到相对应的位置输出即可。

1002 A+B for Polynomials

多项式的加法，建立一个结构体存放指数和底数，然后建一个map，key为指数，value为底数，然后对应相加即可，最后输出，这里还需要注意的一点是，可能会有对应指数相加之后，它前面的系数变成了0，所以这个时候不用输出，需要特殊判断一下。

```
#include <bits/stdc++.h>
#include <cstdio>
using namespace std;

map<int, double> m;

int main()
{
    int k1;
    scanf("%d", &k1);

    for (int i = 0; i < k1; i++){
        int p;
        double e;
        scanf("%d %lf", &p, &e);
        m[p] += e;
    }

    scanf("%d", &k1);

    for (int i = 0; i < k1; i++){
        int p;
        double e;
        scanf("%d %lf", &p, &e);
        m[p] += e;
    }

    vector<pair<int, double>> vec;

    for (auto s : m){
        if (s.second != 0)  vec.push_back(make_pair(s.first, s.second));
    }

    cout << vec.size();
    sort(vec.begin(), vec.end(), [](pair<int, double>& a, pair<int,
double>& b){
        return a.first > b.first;
    });

    for (auto i : vec){
```

```
            cout << " " << i.first << " ";
            printf("%.1f", i.second);
        }
        return 0;
    }
```

1003 Emergency

变种狄杰斯科拉算法，判断一下distance如果相等的情况，这个时候需要使用weight再去判断一遍即可。 贴一个堆优化的狄杰斯科拉算法作为后续参考，主要是使用优先队列来进行寻找最小的cost的节点。

```
#include <iostream>
#include <set>
#include <cstdio>
#include <unordered_map>
#include <unordered_set>
#include <vector>
#include <algorithm>
#include <map>
#include <stack>
#include <cmath>
#include <string>
#include <cstring>
#include <sstream>
#include <queue>

using namespace std;

const int Max = 1000000;
const int maxn = 505;

struct node {
    int id, cost = 0, weight = 0, num = 1;
    bool operator< (const node& n) const{
        return cost > n.cost;
    }
};

struct edge {
    int to, c;
};

node N[maxn];
int visit[maxn];
int dis[maxn];
int weight[maxn];
vector<edge> Graph[maxn];
int n, m, c1, c2;

void djfunc(){
```

```
    priority_queue<node> node_queue;
    node_queue.push(N[c1]);

    while (!node_queue.empty()) {
        node n = node_queue.top(); node_queue.pop();
        if (visit[n.id]) continue;

        visit[n.id] = 1;

        for (int i = 0; i < Graph[n.id].size(); i++){
            node& adj = N[Graph[n.id][i].to];
            edge e = Graph[n.id][i];

            if (dis[n.id] + e.c < dis[e.to]){
                dis[e.to] = dis[n.id] + e.c;
                adj.cost = dis[e.to];
                adj.weight = N[n.id].weight + weight[adj.id];
                adj.num = N[n.id].num;
                node_queue.push(adj); // 记住要push这个节点...
            }else if (dis[n.id] + e.c == dis[e.to]){
                adj.num += N[n.id].num;
                if (N[n.id].weight + weight[adj.id] > adj.weight){
                    adj.weight = N[n.id].weight + weight[adj.id];
                }
                node_queue.push(adj);
            }
        }
    }
}

int main()
{
    cin >> n >> m >> c1 >> c2;

    for (int i = 0; i < n; i++) scanf("%d", &weight[i]);
    for (int i = 0; i < m; i++){
        int v1, v2, l;
        scanf("%d %d %d", &v1, &v2, &l);
        Graph[v1].push_back(edge{v2, l});
        N[v1].id = v1;
        Graph[v2].push_back(edge{v1, l});
        N[v2].id = v2;
    }

    // initial state
    std::fill(dis, dis + maxn, Max);
    dis[c1] = 0;
    N[c1].weight = weight[c1];

    djfunc();

    cout << N[c2].num<< " " << N[c2].weight;
    return 0;
```

```
}
```

1004 Counting leaves

这道题就是一个树的遍历，使用dfs或者bfs的方式遍历，然后在遍历的同时记录下最高的高度，和对应高度下的叶子节点的个数，当碰到一个叶子节点之后，就给对应的数组或者是使用map的value上给加一即可，最后循环输出打印结果。最主要的就是考察树的遍历操作了，注意插入到map当中的条件，是非叶节点，并且在遍历的同时就可以使用一个数来记录下最大高度就好了。

1005 Spell It Right

string接收一下，然后可能使用 long long 或者其他计算数字之和，然后输出英文版的发音即可。

1006 Sign In and Sign Out

这种有关于时间都可以转化到秒来做，然后简单的通过sign in或者sign out的时间来排序即可，就可以很容易的找到应该输出的人的ID.

1007 Maximum Subsequence Sum

这题主要是用来回顾最长字串和的算法，使用this_sum实时记录当前的字串和，当它小于0的时候置为0，如果超过已知的最大值，更新已知的最大值，这种算法是一个O(n)的计算算法。

```
int max_sum(vector<int>& rhs)
{
        int length = rhs.size();
        int this_sum = 0;
        int max_sum = min_value;

        for (int i = 0; i < length; i++)
        {
                this_sum += rhs[i];

                if (this_sum > max_sum)
                {
                        max_sum = this_sum;
                        max_position = i;
                }

                if(this_sum < 0)
                {
                        this_sum = 0;
                }
        }

        return max_sum;
}
```

同时这道题也要看清楚题目，需要输出的是对应位置上的数，而不是两个区间。同时算一个vector里面的最大值的时候，可以使用相对应的函数：

```
auto iter = max_element(vec.begin(), vec.end());
```

1008 Elevator

只需要计算总共需要上的次数和总共需要下的次数，然后算一下对应的时间就好了。 不过需要看清楚题目，就是第一个数一般来说会代表的是，后面总共有几个数出现。

1009 Product of Polynomials

跟之前那个差不多，就是注意这里是乘法，所以最后需要注意的是在map里面的键变成了加号。

```
poly_map[p1.exponent + p2.exponent] += (p1.coefficient * p2.coefficient);
```

1011 World Cup Betting

找个最大值，然后算一下输出就好了，非常的水题。

1012 The Best Rank

模拟题，事先计算好所有科目的平均分，然后对四门课程的分数分别进行排序，然后将最后的排名存储在结构体当中的对应位置。

1013 Battle Over Cities

使用dfs来判断图中有几个连通分量，最后输出cnt - 1即为最后的结果，输出就好，实现代码如下：

```
int n, m, k;
int G[1010][1010];
int visit[1010];

void dfs(int root){
    visit[root] = 1;

    for (int i = 1; i <= n; i++){
        if (visit[i] == 0 && G[root][i] == 1) dfs(i);
    }
}

int main()
{
    cin >> n >> m >> k;

    for (int i = 0; i < m; i++){
        int v1, v2;
```

```
        scanf("%d %d", &v1, &v2);
        G[v1][v2] = G[v2][v1] = 1;
    }

    for (int i = 0; i < k; i++){
        int q;
        scanf("%d", &q);

        std::fill(visit, visit + 1010, 0);

        visit[q] = 1;

        int cnt = 0;
        for (int i = 1; i <= n; i++){
            if (visit[i] == 0){
                cnt++;
                dfs(i);
            }
        }

        cout << cnt - 1 << endl;
    }
    return 0;
}
```

1015 Reversible Primes

这道题主要是考察的如何将对应进制下的数给表示出来，一般就是先取模然后使用除法的这种形式将数字表达成十进制的形式。

```
do {
    arr[len++] = n % d;
    n = n / d;
} while (n != 0);
```

1016 Phone Bills

这题和后面的那道停车的题目类似，都是先根据人名进行排序，然后再根据时间的先后次序来进行排序，这样排序的好处是可以很快速的排除掉那些不合理的数据，然后就看一些处理细节了。

1017 Queueing at Bank

这道题就是每个窗口都有一个对应的数值，指示什么时候结束当前人的服务。然后每有一个顾客进来的时候，都遍历一遍所有的窗口，找出最先完成服务的窗口的时间和对应的index，然后这个时候会有两种情况，一种情况是来早了，所以需要加上等待的时间，并且更新窗口的时间，如果是来晚的那种就可以直接服务，所以只要更新窗口的时间即可，具体的实现如下：

```
#include <iostream>
#include <set>
#include <cstdio>
#include <unordered_map>
#include <unordered_set>
#include <vector>
#include <algorithm>
#include <map>
#include <stack>
#include <cmath>
#include <string>
#include <cstring>
#include <sstream>
#include <queue>

using namespace std;

#define MAXN 61200
#define MINN 28800

typedef struct custom{
    int come;
    int serve;
} Custom;

int toSeconds(int h, int m, int s){
    return h * 3600 + m * 60 + s;
}

int main()
{
    int n, k;
    cin >> n >> k;

    vector<Custom> consumer;

    for(int i = 0; i < n; i++){
        int hh, mm, ss, time;
        scanf("%d:%d:%d", &hh, &mm, &ss);
        scanf("%d", &time);

        int come = toSeconds(hh, mm, ss);
        if(come > MAXN) continue;

        Custom cons {come, time * 60};
        consumer.push_back(cons);
    }

    sort(consumer.begin(), consumer.end(), [](Custom& a, Custom& b){
        return a.come < b.come;
    });

    vector<int> window(k, MINN);
```

```
    int result = 0;

    for(int i = 0; i < consumer.size(); i++){
        Custom c = consumer[i];
        int minindex = 0, mintime = window[0];

        /*find the first serve window*/
        for(int j = 0; j < k; j++){
            if(window[j] < mintime){
                minindex = j;
                mintime = window[j];
            }
        }

        if(c.come >= mintime){
            // update the corresponding window time
            window[minindex] = c.come + c.serve;
        }else{
            result += (mintime - c.come);
            window[minindex] += c.serve;
        }
    }

    if(consumer.empty()) cout << "0.0" << endl;
    else{
        printf("%.1f", result * 1.0 / 60 / consumer.size());
    }

    return 0;
}
```

1018 Public Bike Management

狄杰斯科拉+dfs路径回溯，这里主要坑的是，要到最后出去的时候，再去计算当前的back和need，并且车只能从上面带到下面，不能从下面带到上面，所以一开始我的那种方法就不行。

```
#include <bits/stdc++.h>
#include <cstdio>
using namespace std;

#define maxn 505

int cmax, n, sp, m;

int bikes[maxn];

struct node {
    int id, cost = 0;
    vector<int> prev;
    bool operator< (const node& a) const{
        return cost > a.cost;
```

```
    }
};

struct edge {
    int to, c;
};

vector<edge> G[maxn];
node N[maxn];
int visit[maxn];
int dis[maxn];

void djfunc()
{
    priority_queue<node> node_queue;
    node_queue.push(N[0]);

    while (!node_queue.empty()) {
        node n = node_queue.top(); node_queue.pop();
        if (visit[n.id]) continue;
        visit[n.id] = 1;

        for (int i = 0; i < G[n.id].size(); i++){
            edge e = G[n.id][i];
            int to = e.to;
            node& adj = N[to];
            node& thi = N[n.id];

            if (dis[thi.id] + e.c < dis[adj.id]){
                dis[adj.id] = dis[thi.id] + e.c;
                adj.cost = dis[adj.id];
                adj.prev.clear();
                adj.prev.push_back(thi.id);
                node_queue.push(adj);
            }else if (dis[thi.id] + e.c == dis[adj.id]){
                adj.prev.push_back(thi.id);
                node_queue.push(adj);
            }
        }
    }
}

vector<int> result, ans;
int minneed = 1000000, minback = 1000000;

void dfs(int root){
    if (root == 0){
        ans.push_back(root);
        int need = 0, back = 0;

        for (int i = ans.size() - 2; i >= 0; i--){
            int id = ans[i];
            int tmp = bikes[id] - (cmax / 2);
            if (tmp > 0){
```

```
                back += tmp; // 不能往回送
            } else{
                tmp = abs(tmp);
                if (back > tmp){
                    back -= tmp;
                }else{
                    need += (tmp - back);
                    back = 0;
                }
            }
        }
        if (need < minneed){
            minneed = need;
            minback = back;
            result = ans;
        }else if (need == minneed && back < minback){
            minback = back;
            result = ans;
        }
        ans.pop_back();
        return;
    }else{
        ans.push_back(root);
        for (int i = 0; i < N[root].prev.size(); i++){
            dfs(N[root].prev[i]);
        }
        ans.pop_back();
    }
}

int main()
{
    cin >> cmax >> n >> sp >> m;
    for (int i = 1; i <= n; i++) scanf("%d", &bikes[i]);
    for (int i = 0; i < m; i++){
        int v1, v2, c;
        scanf("%d %d %d", &v1, &v2, &c);
        N[v1].id = v1; N[v2].id = v2;
        G[v1].push_back(edge{v2, c}); G[v2].push_back(edge{v1, c});
    }

    fill(dis, dis + maxn, 1000000);
    dis[0] = 0;

    djfunc();

    dfs(sp);

    cout << minneed << " ";
    reverse(result.begin(), result.end());
    for (int i = 0; i < result.size(); i++){
        if (i == 0) cout << result[i];
        else cout << "->" << result[i];
    }
```

```
    cout << " " << minback;
    return 0;
}
```

1019 General Palindromic Number

也是求出在对应进制下的表示，然后看是不是对称的表示，比较简单。

1022 Digital Library

这道题就是用的各种map将信息聚集起来，然后去查询就好了。

1023 Have Fun with Numbers

这道题考察的是使用字符串进行两个数之间的加法，具体的做法就是将两个东西的位数补齐，然后逐位的进行相加。最后检查所有位置上，如果有超过9的东西，往前面的位置加上1，最后检查首位的数字是不是要比9要大，如果是的话，在最前面再补一个1上去。

```
string Add(string& add_1, string& add_2)
{
        string add_result = add_1;
        for (int i = add_1.size() - 1; i >= 0; i--)
                add_result[i] += (add_2[i] - '0');

        for (int i = add_result.size() - 1; i >= 1; i--)
        {
                if (add_result[i] > '9')
                {
                        add_result[i] -= 10;
                        add_result[i - 1]++;
                }
        }

        if (add_result[0] > '9')
        {
                add_result[0] -= 10;
                add_result.insert(add_result.begin(), '1');
        }

        return add_result;
}
```

1028 List Sorting

这道题主要是注意一些cstring的函数需要去关注一下，如strcmp(c1, c2) < 0代表的是c1比c2要小，同理其他的都是一样的。

1029 Median

这道题是求两个有序的序列当中的中位数，那么最开始的想法就是使用双指针的方式来做这道题，这样就是O(n)的时间复杂度完成。并且使用long long的精度来存储，就可以很快的扫描找到中位数。

```
#include <bits/stdc++.h>
#include <cstdio>
using namespace std;

typedef long long ll;

vector<ll> v1, v2;

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

    v1.resize(num);
    for (int i = 0; i < num; i++) scanf("%lld", &v1[i]);

    cin >> num;
    v2.resize(num);
    for (int i = 0; i < num; i++) scanf("%lld", &v2[i]);

    int pos1 = 0, pos2 = 0;
    ll pos;
    int total = v1.size() + v2.size();
    int median;
    if (total % 2 == 1) median = (total + 1) / 2;
    else median = total / 2;

    int cnt = 0;
    while (true) {
        for (; v1[pos1] <= v2[pos2] && pos1 < v1.size(); pos1++){
            cnt++;
            if (cnt == median){
                pos = v1[pos1];
                break;
            }
        }

        if (cnt == median) break;

        if (pos1 == v1.size()){
            pos = pos2 + (median - cnt - 1);
            pos = v2[pos];
            break;
        }else{
            for (; v2[pos2] <= v1[pos1] && pos2 < v2.size(); pos2++){
                cnt++;
                if (cnt == median){
                    pos = pos2;
                    pos = v2[pos2];
```

```
                    break;
                }
            }

            if (cnt == median) break;

            if (pos2 == v2.size()){
                pos = pos1 + (median - cnt - 1);
                pos = v1[pos];
                break;
            }
        }
    }

    cout << pos;
    return 0;
}
```

1030 Travel Plan

使用堆优化的狄杰斯科拉算法走一遍，不过要记得每次都要把对应的node给push到node_queue当中，然后最后把路径给打印出来。

1032 Sharing

这种链表题一般就是使用数组来存储键的值，然后给每一个node分配。在这道题里面只需要遍历一下第一个链表，将所有的key上打上flag，然后使用第二个链表来再次遍历，如果碰上刚好有一样的，那么OK就可以输出了，不过需要注意输出的格式，很多都是用`%5d`这种格式来输出的了，所以需要非常注意。 但是感觉这样，不会有点问题吗..... 如果是叉出来两条链表，就不存在common suffix了啊。

1033 To Fill or Not to Fill

注意贪心的思想和思路，先假设在当前加油站能到的最远的位置，如果有油价更低的加油站，就刚好加到那里，然后加满油。如果没有的话，那就先加满油，然后开到下一个油价相对最低的那个点去加油。

1034 Head of a Gang

使用dfs来判断图当中的联通分量，但是同时也要兼顾到联通分量当中有几个成员，以及最大的weight对应的head是哪位，所以这里考虑使用两步的dfs来做这个事情，一个dfs专门用来做一个联通分量当中的head和number的计算，出来之后再根据条件判断即可。

1038 Recover the Smallest Number

此题非常的经典，典型的使用贪心算法来做的一道题目。主要的思想就是根据两个字符串相加之后，如果更小的话，那么s1就应该放在s2的前面，这样才会满足所组成的字符串在最后是最小的那个字符串。因为首字母的0不输出，所以在输出index为0的时候，直接使用stoi函数处理一下就好来，其他的直接输出。

1040 Longest Symmetric String

这个首先需要注意的是，读取一行的是通过getline(cin, s)这种方式来进行一行内容的读取的。

1043 Is It a Binary Search Tree

学会根据前序来给二叉搜索树建树，然后知道二叉树的中序遍历是有序的，所以这样就能根据前序和中序或者是后序和中序来建树。

1051 Pop Sequence

模拟栈的push和pop，如果某个时刻压到栈里面的个数超过的栈的cap，说明就是不行的，如果能够全部完成，说明是yes的。

1053 Path of Equal Weight

树的遍历，可以采取dfs的形式进行遍历，然后需要注意的是，先对每个节点的childs进行排序，然后再dfs下去，如果刚好weight相同，并且它也是leaf，那就直接打印路径。然后弹出去再接着进行dfs的步骤。

1057 Stack

TODO: 树状数组，待看。

1059 Prime Factors

这道题主要考察的是素数表或者说是素数筛的建立，具体可以考虑下面这种实现的方式：

```
for(int i = 2; i * i < 50000; i++)
    for(int j = 2; j * i < 50000; j++)
        prime[j * i] = 0;
```

1064 Complete Binary Search Tree

完全二叉搜索树，它的中序遍历是有序的。所以可以根据这一点来寻找每个node它所对应的数组下标是多少，通过树的中序遍历去得到。然后就只需要对最后得到的那个tree循环输出一遍就好了，就是我们要得到的level order打印出来的结果。

1067 Sort with Swap(0, i)

贪心的思路，每次都选择最先的那个没有被排好序的位置，然后使用0不断的和正确位置上的数值交换，然后直到最后回到了0的位置上面，再去检查。如果还没换好，则把那个位置上的数和0做一个交换，然后直到最后所有的序列都被排好序就结束了。

1068 Find More Coins

排个序用dfs做把，虽然会有一个点超时，但是dp这个01背包暂时应该没时间看了orz.

1070 Mooncake

根据mooncake的单价进行排序，选择单价最高的先卖，然后再往后面一个一个卖。

1071 Speech Patterns

这道题主要就是考察的如何遍历一个字符串，将里面的一些子字符串给抽取出来，还有一些常用的函数可能会用到，解法如下所示：

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

map<string, int> word_map;

int main()
{
        string text;
        string word;

        getline(cin, text);
    // 转换大小写的函数
        transform(text.begin(), text.end(), text.begin(), ::tolower);

        for (int i = 0; i < text.length(); i++)
        {
                if (isalnum(text[i])) // 判断是字母或者是数字的函数
                {
                        word += text[i];
                }

                if (!isalnum(text[i]) || i == text.length() - 1)
                {
                        if (!word.empty())
                                word_map[word]++;

                        word.clear();
                }
        }

        int max_time = -1;
        auto final_iter = word_map.begin();
        for (auto iter = word_map.begin(); iter != word_map.end(); iter++)
        {
                if (iter->second > max_time)
                {
                        max_time = iter->second;
                        final_iter = iter;
                }
        }

        cout << final_iter->first + " " << final_iter->second << endl;
        //system("pause");
        return 0;
}
```

1074 Reversing Linked List

链表的题目很多都是先开好一个大数组存好，然后再塞到一个vector里面组成一个线性的链表，其实它的end_pos都不用特定的指明，最后输出的时候输出一下就好了，不过需要注意的就是最后一个节点，如果整条链表就只有一个节点的话，那么它的end_pos就是-1，直接输出-1就好了。

1075 PAT Judge

注意一下那些计算rank的时候的事情，一般来说就是先固定第一个位置的rank为1，然后从1开始循环，如果和前面的某些数值是一样的话，那么这个index的rank和前面的rank是一样的，否则更新当前index的rank为index + 1即可，完成排序。

```
vec[0].rank = 1;
vec[0].PrintInfo();
for(int i = 1; i < vec.size(); i++){
    if(vec[i].total == vec[i - 1].total) vec[i].rank = vec[i - 1].rank;
    else vec[i].rank = i + 1;
    vec[i].PrintInfo();
}
```

1076 Forwards on Weibo

这道题采用的是bfs的方法进行计算最大可能遍历到的节点，注意push进去的条件即可，一个是下一个节点没有被visit过，同时当前节点的level比给定的max_level要小，就push到queue当中就好了。bfs的方法的话，一般是会使用一个queue来存放下一个需要遍历的节点，并且是一个循环迭代的过程，比dfs会节省很多的时间的。

1078 Hashing

这题主要是针对的二次线性增长的hash出的一道题，非常的经典。首先是寻找下一个prime，然后注意的是发生碰撞之后应该怎么进行处理的这个步骤。

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<cstdio>
#include<vector>
#include<cmath>
using namespace std;

#define Max 20000
int HashTable[Max];
vector<int> pos_vec;

bool is_prime(int value)
{
        if (value <= 1) return false;
        else
        {
                for (int i = 2; i <= sqrt(value); i++)
```

```
                {
                        if (value % i == 0) return false;
                }
                return true;
        }
}

int next_prime(int value)
{
        while (1)
        {
                if (is_prime(value)) return value;
                value++;
        }
}

int main()
{
        int table_size, sequence_num;
        cin >> table_size >> sequence_num;
        int real_size = next_prime(table_size);

        for (int i = 0; i < sequence_num; i++)
        {
                int num;
                int steps = 0;
                int pos;
                scanf("%d", &num);

        // 当寻找的时候已经超过了hashtable的长度的时候,
        // 我们就可以认为它已经找不到自己的位置了，所以放进去的是-1
                for (int j = 0; j < real_size; j++, steps++)
                {
                        int hash_pos = num % real_size;
                        int real_pos = (hash_pos + j * j) % real_size;
                        if (HashTable[real_pos] == 0)
                        {
                                HashTable[real_pos] = num;
                                pos_vec.push_back(real_pos);
                                break;
                        }
                }

                if (steps == real_size) pos_vec.push_back(-1);
        }

        for (int i = 0; i < sequence_num; i++)
        {
                if (i == 0)
                {
                        if (pos_vec[i] != -1) cout << pos_vec[i];
                        else cout << "-";
                }
                else
```

```
                {
                        if (pos_vec[i] != -1) cout << " " << pos_vec[i];
                        else cout << " -";
                }
        }

        //system("pause");
        return 0;
}
```

1081 Rational Sum

Gcd算法，非常有用，在这里记录一下：

```
ll Gcd(ll a, ll b)
{
        return (b == 0) ? a : Gcd(b, a % b);
}
```

1086 Tree Traversals Again

二叉树，知道前序和中序，转换到后序进行输出。 TODO，待看，和前面有一个长的差不多，和知道中序和后序一起看一遍吧。

1087 All Roads Lead to Rome

狄杰斯科拉算法找到所有的最短路 + 路径的dfs寻找

1089 Insert or Merge

抓住插入排序和merge sort的区别，如果判断为插入排序的时候，再往后面排序一次就好了，如果是merge sort的话，就是模拟merge sort的方式，然后不停的判断当前的结果是不是与给出的排序结果相同，如果是就再往后面排序一次就好了。

```
#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;
}
```

1091 Acute Stroke

这道题主要要注意的就是，在bfs的时候，遍历的条件或者说怎么去遍历的时候，可以使用三个数组的形式去表示下一个方向，所以就像题解当中提到的这种形式：

```
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};
```

1093 Count PAT's

这道题就是以A为界限，它前面的P的个数再与它后面的T的个数相乘，每次都加上然后模一下余数。

```
#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;
}
```

1096 Consecutive Factors

这道题质数的情况或者是1的情况下，那么输出就肯定是本身这个数，否则的话，先把所有的因子列出来，然后遍历这个因子的列表，把这个最长的序列给找出来。

1098 Insertion or Heap Sort

这道题主要是考察的大顶堆的实现，就是DeleteMax这个操作的考察，先把最后一个节点空出来并且记录，然后从根节点开始，算左右child哪个更大，然后顶上去，如果最后一个数已经比这个还大的话，直接赋值过去就吼了。

1101 Quick Sort

这道题主要考察的是快速排序的枢纽点的选择，主要我们抓住的就是，枢纽点它的位置在排序前后是不变的，所以可以先排序一波，找到那些位置不变的点。同时也需要满足当前位置的数据比前面所有值的最大值要大才可以，所以在遍历的同时记录下前面序列的最大值就好了。

1102 Invert a Binary Tree

这道题需要中序和层序遍历一个inverted的二叉树，那么我们要做的就是在遍历的时候，把右儿子放在左儿子前面进行遍历即可了。详细的一些初始化和构造如下，可以参考：

```
typedef struct node
{
        int index = 0;
        struct node* Left = NULL;
        struct node* Right = NULL;
} Node;

typedef Node* node_ptr;

void initial(int node_num)
{
        for (int i = 0; i < node_num; i++)
        {
                node_ptr ptr = new Node;
                node_vector.push_back(ptr);
        }

        memset(is_root, false, sizeof(is_root));
}
```

还有就是那些没有出现在childs里面的那个node就是根节点，这个有的时候是需要我们自己去把这个跟节点给找出来的。

1107 Social Clusters

这题我的思路是把所有的hobby进行并查集，并且用一个vector保留下每个人的一个hobby，用来在后面进行叠加和输出，主要需要注意的就是这个并查集的使用了。

```
int S[1001];
vector<int> rem_first_num_vector;
map<int, int> result_map;

int Find(int root)
{
        if (S[root] <= -1)
                return root;
        else
                return S[root] = Find(S[root]);
}

void Union(int root1, int root2)
{
        int real_root1, real_root2;
        real_root1 = Find(root1);
        real_root2 = Find(root2);

        if (S[real_root1] <= S[real_root2]) //size root1 > size root2
        {
                S[real_root1] += S[real_root2];
                S[real_root2] = real_root1;
        }
        else
        {
                Union(root2, root1);
        }
}
```

1108 Finding Average

采用的是sscanf和sprintf的使用，具体过程是：

```
sscanf(a, "%lf", &tmp);
sprintf(b, ".2f", tmp);
```

1109 Group Photo

采用deque的方式，左右交替的将人给塞进去。这里就是要想到双向队列的使用。时刻提醒自己，一定要认真的读题，真的不能慌，一慌脑子就全乱套了！！！，今天这个模拟问题就很大！！！！

1110 Complete Binary Tree

根据计算出来的下标，因为使用数组来表示一棵树的时候，index * 2为左儿子，index * 2 + 1为右儿子，所以最后计算出来的index如果刚好是所有的总数，那么说明这个是一颗完全二叉树，如果超过了，那就不是了。 这里也是一个上午一直强调自己的点，能用string的，就用string去接，你为啥要用char呢。。这玩意能不用就不用，搞的都快怀疑自己了，浪费很多时间在这里，都是无效的时间的浪费！！！！

1111 Online Map

这道题用两个狄杰斯科拉+dfs做出来的。堆优化之后的时间降了很多，但是这里也出现了自己忽视的一个点，就是一开始的id没有给分配，导致后面一下就出现了问题。然后就是自定义的priority_queue的一个模板写法。

```
struct cmp1 {
    bool operator() (node& a, node& b){
        return a.cost > b.cost;
    }
};

struct cmp2 {
    bool operator() (node& a, node& b){
        return a.time > b.time;
    }
};

priority_queue<node, vector<node>, cmp1> node_queue;
```

1147 Heaps

针对堆的话，就是可以使用数组来模拟树，然后像上面这个一样把左右子树给串起来。然后检查最大堆最小堆的时候，就是递归的进行检查了。

```
typedef struct node
{
        int value;
        struct node* left = NULL;
        struct node* right = NULL;
} Node;

typedef Node* node_ptr;
node_ptr* node_vec = new node_ptr[Max];

// 这个node_vec里面的指针值需要重新的去动态获取地址一遍
void initial(int num)
{
        for (int i = 0; i <= num; i++)
                node_vec[i] = new Node;
}

// 递归检查
```

```
bool detect_max_heap(node_ptr root)
{
        if (root == NULL) return true;

        if (root->left != NULL)
        {
                if (root->value < root->left->value) return false;
        }

        if (root->right != NULL)
        {
                if (root->value < root->right->value) return false;
        }

        return (detect_max_heap(root->left) && detect_max_heap(root-
>right));
}
```

1115 Counting nodes in BST

主要考察的是如何给二叉搜索树递归建树，贴一下建树的代码和递归打level的代码：

```
node_ptr create_tree(node_ptr root, int value)
{
        if (root == NULL)
        {
                node_ptr new_node = new Node;
                new_node->value = value;
                return new_node;
        }

    // 递归建树
        if (value <= root->value)
                root->Left = create_tree(root->Left, value);
        else
                root->Right = create_tree(root->Right, value);

        return root;
}

void mark_level(node_ptr root)
{
        if (root == NULL) return;

        if (root->Left != NULL)
        {
                root->Left->level = root->level + 1;
                levels_vec.push_back(root->Left->level);
        }

        if (root->Right != NULL)
```

```
        {
                root->Right->level = root->level + 1;
                levels_vec.push_back(root->Right->level);
        }

        mark_level(root->Left);
        mark_level(root->Right);
}
```

1123 Is It a Complete AVL Tree

这道题考察的是AVL树的操作，就直接贴上代码看吧，主要是两个基本操作，左旋和右旋，然后LR的话，先对左子树做左旋，然后对自己做右旋，RL的话就是先对右子树做右旋，然后对自己做左旋：

```
struct node{
    int val;
    struct node* left = nullptr;
    struct node* right = nullptr;
};

node* RightRotate(node* root){
    node* temp = root->left;
    root->left = temp->right;
    temp->right = root;
    return temp;
}

node* LeftRotate(node* root){
    node* temp = root->right;
    root->right = temp->left;
    temp->left = root;
    return temp;
}

node* LeftRightRotate(node* root){
    root->left = LeftRotate(root->left);
    return RightRotate(root);
}

node* RightLeftRotate(node* root){
    root->right = RightRotate(root->right);
    return LeftRotate(root);
}

int getHeight(node* root){
    if (root == nullptr) return 0;
    int l = getHeight(root->left);
    int r = getHeight(root->right);
    return max(l, r) + 1;
}
```

```
node* Insert(node* root, int val){
    if (root == nullptr){
        root = new node();
        root->val = val;
        return root;
    }else{
        if (val < root->val){
            root->left = Insert(root->left, val);
            // judge the height
            int l = getHeight(root->left);
            int r = getHeight(root->right);
            if (l - r >= 2){
                if (val < root->left->val) root = RightRotate(root);
                else root = LeftRightRotate(root);
            }
        }else{
            root->right = Insert(root->right, val);
            int l = getHeight(root->left);
            int r = getHeight(root->right);
            if (r - l >= 2){
                if (val < root->right->val) root = RightLeftRotate(root);
                else root = LeftRotate(root);
            }
        }
        return root;
    }
}
```

1134 Vertex Cover

把自己做成一个set，然后遍历所有的边来进行查询。很多都是这样的= =

1135 Is It A Red-Black Tree

这题也被坑了挺久，就是红黑节点又是用正负号，所以直接用string接收，接收完了之后指示这个节点是红节点还是黑节点。然后根据它给的节点进行判断，这里有一个小技巧。因为要多次建树，所以一开始的那个node_ptr我们就用数组存好，后序操作遍历整棵树的时候也是非常有用的，就不用递归的去遍历整棵树了。然后就是看清楚题目！！！所有节点到子节点的路径，黑色节点相同。并且！！！，nullptr节点也算一个黑节点，一开始不以为然23333，吃亏了才发现好像不太对。然后就是综合这几个判断条件，去判断这棵树是不是符合规则的红黑树。

1136 A Delayed Palindrome

一个是string的加法，然后有一个就是怎么判断一些特例的错误，这里如果一开始就是Palindrome number的时候，就直接输出了，就是这个特例存在。

1137 Final Grading

这道题就是，好好看题。人家在题目当中说的是总分大于60即可，并不是期末考试... 所以看清楚题目非常重要，然后排序输出即可。

1138 Postorder Traversal

前序和中序建树即可。

1139 First Contact

这个就是用正负号来代表不同人的题，所以直接用string接收，因为可能会有+-0000的这种情况，你根本判断不了. 然后分别用一个map去记录下每个人的男生朋友和女生朋友，还有一个坑点在于，如果两个人的性别相同，则寻找的朋友并不能是对方，需要去特判一下这种情况。

1140 Look-and-say Sequence

这个就是一个很经典的模拟的题目了。对整个字符串进行模拟，用一个char指示现在已经到哪一个字母了，然后用cnt确定前面的出现的次数，如果碰到一个不一样的，那么塞到result里面，更新cnt到1，最后一个的话，那也是需要继续去更新的。最后将得到的result输出出去即可。记住有一个函数`to_string`来转到string里面的。

1141 PAT Ranking of Institutions

知道怎么去排序，然后知道怎么算rank就OK了这道题。最主要的还是审题需要非常清楚。复杂结构体的排序，针对student做了一个结构，然后再针对每个学校做了一个结构，最后对学校这个结构去排序即可得到最后的结果。

1142 Maximal Clique

就是普通的去判断就好了，先循环遍历输入的是不是能够构成一个clique. 然后再去所有的剩下的节点里去，一个个判断。用一个visit数组记录，是不是在输入的时候，已经放到我们的vector里面去了。

1143 LCA

这道题就是一个递归建树的过程，搞清楚中序的话，就是用来划分左右子树用的。然后左子树有多少个节点，右子树有多少个节点。并且范围划对，一般就没有什么问题。然后就是递归建树的过程了。

```
node_ptr create_tree(int pre_left_pos, int pre_right_pos, int in_left_pos,
int in_right_pos)
{
        if (pre_left_pos > pre_right_pos) return NULL;
        node_ptr new_node = new Node;
        new_node->value = pre_order[pre_left_pos];

        int k;
        for (k = in_left_pos; in_order[k] != new_node->value; k++)
continue;

        int left_subtree_num = k - in_left_pos;
        new_node->Left = create_tree(pre_left_pos + 1, pre_left_pos +
left_subtree_num, in_left_pos, k - 1);
        new_node->Right = create_tree(pre_left_pos + left_subtree_num + 1,
```

```
  pre_right_pos, k + 1, in_right_pos);
        if (new_node->Left != NULL) new_node->Left->parent = new_node;
        if (new_node->Right != NULL) new_node->Right->parent = new_node;

        node_map[new_node->value] = new_node;
        return new_node;
  }
```

后序和中序建树的思路就差不多了，就是找到根节点，然后递归建树。

1144 The Missing Number

开一个数组记录一下判断一下就好了，因为最小的正数肯定不会超过输入的n，所以做一个小小的trick就好了。

1145 Hashing - Average Search Time

这道题主要是考察的Hash的平方探测。注意的next prime这个东西，需要开大一点，因为10000以后的素数，肯定比10005要大了= =，吃亏了。唯一需要注意的是，最大的探测次数就是TableSize次，如果还是找不到，说明是插入不进去了。

1146 Topological Order

模拟拓扑排序，每次都去检查一下当前节点的InDegree是不是为0，如果是，更新其他相邻节点的InDegree，然后一直遍历下去模拟即可。

1147 Heaps

递归检查最大最小堆，没什么好说的。

1148 Werewolf - Simple Version

枚举做的题，先假设两个人是狼，然后再从两匹狼当中，再假设一个狼是liar，这个时候是在其他不是狼的人里面选一个liar，然后送到一个Check函数里面检查是否满足所有的state，如果满足了，就直接输出return，如果还能到下面，说明所有情况都不满足，就输出没有结果。这题真的，很难想到就直接枚举做出来的2333，所以大胆尝试最朴素的解法吧。

1149 Dangerous Goods Packaging

这道题和damn single那题是一样的，就是先给定一串不能放在一起的东西，可以用`map<int, vector<int>>`或者一个二维数组来存，然后把输入做成一个`unordered_set`来加快查询。主要就是要注意这一点，还有就是输出的时候，用`printf("%05d")`来做输出。

1150 Travelling Salesman Problem

旅行商问题，三个状态，主要就是用一个set保存所有不同节点的信息，如果首尾不同，就先判not，如果没有遍历所有节点，也是not，如果节点数太多了，就不是simple，否则就是simple。然后还有一个很重要的东西就是，判断这条路是不是能走通，不能走通也是not，有的图的题目也是，先要判断这个图是否是一个连通图，可以通过dfs或者并查集的方式来判断. 然后算最小的时候，要去掉not的那种情况，最后输出。

1151 LCA in a Binary Tree

首先是中序和前序建树，这个没什么好说的。然后用一个map存下value和对应的node_ptr的关系，后序查找方便。然后把所有节点的高度算一下，对于查找的两个节点，先让高度更下面的节点爬到相同的高度上，如果刚好相同，那就是ancestor了，如果不是的话，再同时往上走，知道两个节点的parent相同，就是LCA的那个节点了。

1152 Google Recruitment

这道题主要是去找连续的数字是质数的第一个那个数，就遍历一遍，找到就输出，没找到就输出404. 不过需要特别注意，打印的时候，是打印的这几个数，也就是打印字符串和直接打印数字的这两个区别，需要特别注意这一点，很多都是这样的 = =

1153 Decode Registration Card of PAT

模拟题，主要是怎么用map去维护各种各样的信息。首先题目时间限制这么小，那么肯定是用的char数组来当字符串，然后其他的信息都用map去维护。这道题的坑点在于输入可能是非法的，（下次得注意这一点），所以对于那些非法的输入，输出的是NA。还有就是type正确的时候，后面跟的那个数也可能是非法的，但是这个时候还是需要把这个东西给输出出来，所以第二个变量使用string来接收，然后直接打印，很多题都是这样，比如用正负号来代表两种人的题，务必要用`string`！！这道题可能是我想的偏了吧呜呜呜。

1154 Vertex Coloring

遍历一遍边就好了，没啥好说的。

1155 Heap Paths

这题主要考察堆的建立，使用数组的方式进行建立，空出第一个位置，然后`2 * i`是左孩子，`2 * i + 1`是右孩子。然后递归进行检查是不是maxheap或者minheap，对于路径打印，那就是`dfs`了！！！判断一下是不是到leaf节点了，如果到了，就将vector path里面的路径打印出来就吼了。

最后贴一个优先队列吧：

```
priority_queue<int> q // 默认大顶堆
priority_queue<int, vector<int>, greater<int> > q // 小顶堆

// 比如说求第k大的数，有一种方法就是使用的大顶堆来做的。
```

最后提醒一下，可以使用二分法来加快查找的速度。还有能用long long的，就用long long.

```
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;
    }
```