日期格式化：

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

int main()
{
    int y, m, d;
    scanf("%d.%d.%d", &y, &m, &d);
    printf("%04d-%02d-%02d", y, m, d);
    return 0;
}
```

A除以B：

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

int main()
{
    int a, b;
    cin >> a >> b;
    if(b==0)
        printf("%d/%d=Error", a, b);
    else if(b<0)
        printf("%d/(%d)=%.2f", a, b, 1.0*a/b);
    else
        printf("%d/%d=%.2f", a, b, 1.0*a/b);
    return 0;
}
```

胎压监测：

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

int main()
{
    int a[4];
    int minx, maxx=0, yuzhi;
    for (int i = 0; i < 4; ++i)
    {//一边读入一边判断最大值
        cin >> a[i];
        if(a[i]>maxx) maxx = a[i];
    }
    cin >> minx >> yuzhi;//最小胎压，阈值
    int count = 0;//不正常轮胎数量
    int pos;//不正常轮胎编号
    for (int j = 0; j < 4; ++j) {
        if (((maxx-a[j])>yuzhi)||a[j]<minx)
```

```
        {
            count++;
            pos = j+1;//编号是下标加1
        }
    }
    if (count==0) printf("Normal");
    else if (count>1) printf("Warning: please check all the tires!");
    else if (count==1)printf("Warning: please check #%d!",pos);
    return 0;
}
```

敲笨钟：

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

int main()
{
    int n;
    cin >> n;
    getchar(); //注意getchar的使用
    string s;
    for(int i=0; i<n; i++)
    {
        getline(cin , s);
        if(s.find("ong,")!=string::npos && s.find("ong.")!=string::npos)
        {
            int j = s.size()-1;
            int cnt = 0;
            for( ; cnt<3; j--)
                if(s[j]==' ') cnt++;  //从后往前定位到第三个空格
            cout << s.substr(0, j+1); //输出不被替换的内容
            cout << " qiao ben zhong." << endl;
        }
        else
        {
            cout << "Skipped" << endl;
        }

    }
}
```

整除光棍：

用竖式除法的方法。若使用long long直接检测，不能满分。

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

int main()
{
    int x;
    cin >> x;
    int s = 1;
    int cnt = 1;
```

```
    while(s/x==0) //求出比x大的最小的11...111
    {
        s = s*10+1;
        cnt++;
    }
    while(s%x!=0)//当本次的11..111不能被x整除
    {//循环过程就是模拟竖式除法的过程
        cout << s/x; //输出商
        s = s%x*10 + 1;//余数后面补1，等下继续除。
        cnt++;
    }
    cout << s/x << " " << cnt;
}
```

互评成绩:

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

int main()
{
    int n, m, k;
    cin >> n >> k >> m;
    double a[10000];
    for(int i=0; i<n; i++)
    {
        double x, sum = 0;
        double min = 101, max = -1;
        for(int j=0; j<k; j++)
        {
            cin >> x;
            sum += x;
            if(min>x) min = x;
            if(max<x) max = x;
        }
        a[i] = (sum-min-max)/(k-2);
    }
    sort(a, a+n);
    for(int i=n-m; i<n; i++)
    {
        printf("%.3f", a[i]);
        if(i<n-1) cout << " ";
    }

    return 0;
}
```

月饼:

使用贪心算法，即使你不知道什么是贪心，但是这个方法你一定能想出来
把月饼按性价比排序，按照性价比非升序进行选择

```
//注意库存与售价是浮点数
#include<iostream>
#include<algorithm>
using namespace std;
```

```
struct moon
{
    double cnt, price, unit;
};

bool cmp(moon x, moon y)
{
    return x.unit > y.unit;
}

int main()
{
    int n, d, i;
    cin >> n >> d;
    moon a[1000];
    for(i=0; i<n; i++)
        cin >> a[i].cnt;
    for(i=0; i<n; i++)
    {
        cin >> a[i].price;
        a[i].unit = a[i].price/a[i].cnt;//性价比
    }
    sort(a, a+n, cmp);//按性价比非升序排序
    double sum = 0, rest = d;
    int k = 0;
    while(rest>0 && k<n)//需求量和月饼数量
    {
        if(a[k].cnt<=rest) sum+=a[k].price, rest-=a[k].cnt;
        else sum+=a[k].unit*rest, rest=0;
        k++;
    }
    printf("%.2lf", sum);
    return 0;
}
```

城市间紧急救援：

典型的最短路径题目。使用dfs，最后一个点超时。改用dijkstra就OK了

```
//注意：路径是双向的

#include<iostream>
#include<climits>
#include<vector>
using namespace std;

int dis[500][500] = {0};
int teams[500] = {0};

int min_len = INT_MAX;
int max_num = 0;
bool vis[500] = {0};
vector<int> path, bestpath;

int cnt = 1;
```

```
void dfs(int n, int s, int e, int len, int num);

int main()
{
    int n, m, s, e;
    int i, x, y, d;
    int len, num;

    cin >> n >> m >> s >> e;

    for (i=0; i<n; i++)
        cin >> teams[i];

    for (i=0; i<m; i++)
    {
        cin >> x >> y >> d;
        dis[x][y] = d;
        dis[y][x] = d;
    }

    vis[s] = 1;

    dfs(n, s, e, 0, teams[s]);

    cout << cnt << " " << max_num << endl;

    cout << s;
    for(auto x:bestpath) cout << " " << x;

    return 0;
}

void dfs(int n, int s, int e, int len, int num)
{
    int i, j, flag;

    if (s == e) //到终点了
    {
        if(len < min_len)//当前所走的路径长度更小
        {
            min_len = len; //更新最短长度
            max_num = num; //更新min_len对应路径收集的人数
            bestpath = path;
            cnt = 1;
        }
        else if(len==min_len)//找到一条同样短的路径
        {
            cnt++; //相同长度的路径数加1
            if(num > max_num) //收集的人数更多
            {
                max_num  = num;
                bestpath = path;
            }
        }
        return; //本层函数结束，返回上层调用
    }

    for(i=0; i<n; i++)
```

```
    {
        if(dis[s][i]>0 && len+dis[s][i]<=min_len)
        //s到i有路，且当前长度+dis[s][i]小于等于暂时的最小值
        {
            if(vis[i]==0)//顶点i未经过
            {
                vis[i] = 1;//设为经过
                path.push_back(i);
                dfs(n, i, e, len+dis[s][i], num+teams[i]);
                // 往下查找i出发，到e的路径
                //新的路径长度是len+dis[s][i]，对应新的收集人数是num+teams[i]，下一层是第t个顶点
                vis[i] = 0; //恢复vis[i]为未经过
                path.pop_back();
            }

        }
    }
}
```

```
//注意：路径是双向的
//注意:最短路径数目的更新：
//当发现更短路径时，数量更新为 id的最短路径数目
//当发现相同长度的路径时，数量 += id的最短路径数目
//测试数据不包含起点终点相同的情况

#include<iostream>
#include<climits>
#include<vector>
#include<stack>
using namespace std;

int dis[500][500] = {0};
int teams[500] = {0};

vector<int> p(500, -1);
vector<int> sum(500, 0);
vector<int> cnt(500, 0);

int n, m, s, e;

void dijkstra();

int main()
{
    int i, x, y, d;

    cin >> n >> m >> s >> e;

    for (i=0; i<n; i++)
        cin >> teams[i];

    for (i=0; i<m; i++)
    {
        cin >> x >> y >> d;
        dis[x][y] = dis[y][x] = d;
    }
```

```
    dijkstra();

    cout << cnt[e] << " " << sum[e] << endl;
    stack<int> t;
    i = e;
    while(i!=s)
    {
        t.push(i);
        i = p[i];
    }
    cout << s;
    while(!t.empty())
    {
        cout << " " << t.top();
        t.pop();
    }

    return 0;
}

void dijkstra()
{
    vector<int> res(n, INT_MAX);
    vector<bool> vis(n, false);

    int i, j;
    for(i=0; i<n; i++)//初始化
    {
        if(dis[s][i]>0)
        {
            res[i] = dis[s][i];
            p[i] = s;
            sum[i] = teams[s]+teams[i];
            cnt[i] = 1;
        }
    }
    vis[s] = true;

    int mins, id;
    for(i=0; i<n-1; i++)//循环n-1次
    {
        mins = INT_MAX;
        for(j=0; j<n; j++)//找出当前res中最小的路径长度
            if(!vis[j] && res[j]<mins) { mins = res[j]; id = j;}
        vis[id] = true;

        if(id==e) break;

        for(j=0; j<n; j++)//刷新res中的路径长度
        {
            if(!vis[j] && dis[id][j]>0)
            {
                if(res[id]+dis[id][j]<res[j])
                {
                    cnt[j] = cnt[id];
                    res[j] = res[id]+dis[id][j];
                    sum[j] = sum[id]+teams[j];
```

```
                    p[j] = id;

                }
                else if(res[id]+dis[id][j]==res[j])
                {
                    cnt[j] += cnt[id];
                    if(sum[id]+teams[j]>sum[j])
                    {
                        sum[j] = sum[id]+teams[j];
                        p[j] = id;
                    }
                }
            }
        }
    }
}
```

图着色问题：

简单啊。检查每一条边的两个邻接点是否相同颜色，如果有，那就NO。
注意，要所有顶点都有颜色。

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

int main()
{
    int v, e, k;
    cin >> v >> e >> k;
    pair<int, int> a[250000];
    for(int i=0; i<e; i++) cin >> a[i].first >> a[i].second;
    int n;
    cin >> n;
    for(int i=0; i<n; i++)
    {
        bool vi[501] = {0};
        int c[501] = {0};
        int cnt = 0;
        for(int j=0; j<v; j++)
        {
            int x;
            cin >> x;
            if(vi[x]==0) vi[x]=1, cnt++;
            c[j+1] = x;
        }
        if(cnt!=k)//着色顶点数不够
        {
            cout << "No\n";
            continue;
        }
        int flag = 1;
        for(int j=0; j<e; j++)
        {
            if(c[a[j].first]==c[a[j].second]) //检查每一条边的两个邻接点是否相同颜色
            {
```

```
                flag = 0;
                break;
            }
        }
        if(flag) cout << "Yes\n";
        else cout << "No\n";
    }
    return 0;
}
```

秀恩爱分得快：

不难
测试点4有时会超时，看运气。原因是计算了任两人的亲密度。 == ==改进： 只计算问题中那两人与别人的亲密度。

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

double res[1000][1000] = {0};//编号i与编号j的亲密度

int main()
{
    int n, m;
    cin >> n >> m;
    int sex[1000] = {0};//编号为i的性别
    for(int i=0; i<m; i++)
    {
        int k;
        cin >> k;
        vector<int> v;
        for(int j=0; j<k; j++)//读入数据
        {
            string x;
            cin >> x;
            if(x[0]=='-') sex[-stoi(x)]=1;//stoi(x):把字符串x转为整数
            v.push_back(abs(stoi(x)));
        }
        for(int j=0; j<k-1; j++)//计算这张照片里面每两个异性的亲密度并累计
            for(int t=j+1; t<k; t++)
            {
                if(sex[v[j]]==sex[v[t]]) continue;
                res[v[j]][v[t]] += 1.0/k;
                res[v[t]][v[j]] = res[v[j]][v[t]];
            }
    }
    string x, y;
    cin >> x >> y;
    int xx = abs(stoi(x)), yy = abs(stoi(y));
    double maxx = *max_element(res[xx], res[xx]+n);//max_element返回的是指针，所以要间接访问
    double maxy = *max_element(res[yy], res[yy]+n);
    if(res[xx][yy]==maxx && res[yy][xx]==maxy)
        cout << x << " " << y;
    else
```

```
    {
        for(int i=0; i<n; i++)
            if(res[xx][i]==maxx) cout << x << " " << (sex[i]==1?"-":"") << i <<
endl;
        for(int i=0; i<n; i++)
            if(res[yy][i]==maxy) cout << y << " " << (sex[i]==1?"-":"") << i <<
endl;
    }
    return 0;
}
```

彩虹瓶：

使用数组模拟栈，cnt是栈顶下标，now是当前要出货的号码
货架一直没满 && 货物全出（cnt最后为0）才能YES

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

int main()
{
    int n, m, k;
    cin >> n >> m >> k;
    while(k--)
    {
        int a[1001] = {0};
        int now = 1;//当前出货号
        int cnt = 0;//当前堆放数量
        int flag = 0;//能否完成任务
        int i;
        for(i=0; i<n; i++)
        {
            int x;
            cin >> x;
            if(x==now)
            {
                now++;
                while(cnt>0 && a[cnt]==now) cnt--, now++;
            }
            else if(cnt==m) flag = 1;//货架满，不能完成
            else a[++cnt] = x;
        }
        if(flag || cnt>0) cout << "NO\n";
        else cout << "YES\n";
    }
    return 0;
}
```

完全二叉树的层序遍历:

```
//思路：假设完全二叉树的层次遍历序列(b[])是：1, 2, 3, ..., n
//则其后序(a[])的第1个顶点的查找序列是：
// 1,2,4,8,...i,i*2, 直到i*2>n, b[i]=a[1]（满足i*2<=n的最大i）
//其后序(a[])的第2个顶点的查找序列是：没想清楚怎样表达。。。
#include<bits/stdc++.h>
using namespace std;

int res[31];
int n;
int cnt = 1;

void postorder(int a[], int t)
{
    if(t>n) return;
    postorder(a, t*2);//后序遍历左子树
    postorder(a, t*2+1);//后序遍历右子树
    res[t] = a[cnt++];//访问根结点（获得这些根结点的顺序就是后序遍历的顺序）
}

int main()
{
    cin >> n;
    int a[31];
    for(int i=1; i<=n; i++) cin >> a[i];
    postorder(a, 1);
    for(int i=1; i<=n; i++)
    {
        if(i>1) cout << " ";
        cout << res[i];
    }
    return 0;
}
```

病毒溯源:

典型的dfs，vector是可以比较大小的

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

vector<vector<int>> v;
vector<int> path, bpath;
int res=0;

void dfs(int x, int t)
{
    if(v[x].size()==0)
    {
        if(t>res || (t==res && path<bpath))
        {
            res = t;
            bpath = path;
```

```
        }
        return;
    }
    for(int i=0; i<v[x].size(); i++)
    {
        path.push_back(v[x][i]);
        dfs(v[x][i], t+1);
        path.pop_back();
    }
}

int main()
{
    int n;
    cin >> n;
    v.resize(n);

    vector<int> vis(n, 0);
    int k, x;
    for(int i=0; i<n; i++)
    {
        cin >> k;
        while(k--)
        {
            cin >> x;
            v[i].push_back(x);
            vis[x] = 1;
        }
    }
    int root = 0;
    while(vis[root]) root++;
    path.push_back(root);
    dfs(root, 1);
    cout << res << endl;
    for(int i=0; i<bpath.size(); i++)
    {
        if(i>0) cout << " ";
        cout << bpath[i];
    }
}
```