200 和整数对之间的情缘:给你一个有 N 个整型数字的序列 A，整数对(i,j)满足 Ai-Aj 是 200 的整数倍

```
int n,arr[int(2e5)],temp,ans = 0;
int main()
{
    cin >> n;
    for (int i = 0; i < n; n--){
        cin >> temp;
        arr[temp%200]++; //余数相同的数加在一起
    }
    for(int i = 0; i < 200; i++)
        ans += arr[i] * (arr[i] - 1); // C(num,2)组合数
    cout<< ans / 2;
```

模拟计算器:给出 n 个数，和 n - 1 个运算符（只含有加减乘号，不含除号，按顺序填入 n 个数之间），要求输出该式的答案。

```
int n,num[100];
char flag[100];
int main(){
    cin>>n;
    for(int i = 0; i < n; i++) cin>>num[i];
    for(int i = 1; i < n; i++) cin>>flag[i];
    long long n1 = num[0], n2 = num[1];
    char op = flag[1];
    for(int i = 2; i < n; i++){
        if(flag[i]=='*'){
            n2*=num[i];
        }else{
            n1 = (op == '-') ? n1-n2 : (op == '+') ? n1+n2 : n1*n2;
            op = flag[i];
            n2 = num[i];
        }
    }
    n1 = (op == '-') ? n1-n2 : (op == '+') ? n1+n2 : n1*n2;
    cout<<n1;
```

名望值排队:n 个人排队，每人有一名望值，进队规则如下：第一个人直接进队,新来的人发现队尾的人的名望值比自己大或者相等会选择离开,队伍最多 5 人，若队满且一人要进队时发现他的名望值比队首的人大，他会把队首的人挤掉而继续排在队尾。

```
int n,arr[5],temp,head=0,tail=4,last=0,cur=0;
int main(){
    cin>>n;
    for(int i = 0; i < n; i++){
        cin>>temp;
        if(temp>last){
            last = temp;
            if(cur<5){
                arr[cur++] = i+1;
            }
            else{
                arr[head] = i+1;
                head = (head+1)%5;
                tail = (head+1)%5;
            }
        }
    }
    int len = (n>5)?5:n;
    for(int i = 0; i < len; i++)
        if(arr[(head+i)%5]!=0)
            cout<<arr[(head+i)%5]<<" ";
```

台阶问题:有 n 级台阶，每次可向上迈最多 k 级，问到达第级台阶有多少种不同方式。

```
int n,k,ans[int(1e6)];
int main(){
    ans[0] = 1;
    cin>>n>>k;
    for(int i = 1; i <= n; i++){
        ans[i] = 0;
        for(int j = 1; j<=k; j++){
            ans[i] = (ans[i] + ans[i-j]) % 100003;
        }
    }
    cout<<ans[n];
```

(分治）**汉诺塔**:现在给你一个 n 片圆盘的汉诺塔，并从小到大编号为 1 至。请你输出搬动 n 个圆盘最少次数的全过程。

```
//整体法分析，将最底下一个盘子与其上面的多个盘子分为两部分；不用考虑上部的移动，细化为小问题交给递归处理；
void move(int n,char from,char to){
    cout<<"Move disk "<<n<<" from "<<from<<" to "<<to<<endl;
}
// a 是出发点，c 是目的地
void Hanoi(int n, char a, char b, char c){
    if(n==1) move(n,a,c);
    else{
        Hanoi(n-1,a,c,b);   //上部先移动到辅助塔
        move(n,a,c);        //下部移动到目的地
        Hanoi(n-1,b,a,c);}} //上部移动到目的地
int main(){
    int n;
    cin>>n;
    Hanoi(n,'A','B','C');}
```

逆序对:对于一个序列 a ，如果有且 ai > aj 且 i<j，则称 ai，aj 为一逆序对。求出给定序列中逆序对的数量（序列中可能存在重复数字）

```
const int maxn = 1e5 + 10;
int a[maxn], temp[maxn], n;
long long ans = 0;
void MergeSort(int l, int r){
    if (l >= r) return;
    int mid = (l + r) / 2; // 分为子问题
    MergeSort(l, mid);
    MergeSort(mid + 1, r);
    int tempstart = l, start1 = l, start2 = mid + 1, end1 = mid + 1, end2 = r;
    while (start1 < end1 && start2 <= end2){ // 分完后开始排序
        if (a[start1] <= a[start2]) temp[tempstart++] = a[start1++];
        else{
            temp[tempstart++] = a[start2++];
            ans += end1 - start1; // 逆序对个数计算
        }
    }
    while (start1 < end1)
        temp[tempstart++] = a[start1++]; // 防止 a[]中前面的数还没有移完
    while (start2 <= end2) temp[tempstart++] = a[start2++]; // 同上
    tempstart = l; start1 = l;
    while (tempstart <= r)
        a[start1++] = temp[tempstart++]; // 转移到 a[]中
}
int main(){
    while(scanf("%d", &n) != EOF){
        for(int i = 1; i <= n; i ++)
            scanf("%d", &a[i]);
        MergeSort(1, n);
        printf("%d\n", ans);
    }
```

给你三个整数 a,b,p，求（**a 的 b 次方余 p**）的值

```
long long a,b,p,t,ans;
int main(){
    cin>>t;
    while(t--){
        cin>>a>>b>>p;
        ans = 1;
        while(b){
            if(b&1)
                ans = (ans * a) % p;
            a = (a * a) % p;
            b/=2;
        }
        cout<<ans<<endl;
```

每组数据给出 n m k 表示有 n 个数，求**第 k 小**

```
long long arr[int(5e7)],n,m,k;
int quick_k(int start,int end, int _k){
    if(start == end) return arr[start];
    int base = arr[start]; // 以数组第 0 个元素为 base
    int left = start, right = end;
    while(left < right){
        while(left < right && arr[right]>=base) right--;
        arr[left] = arr[right];
        while(left < right && arr[left]<=base) left++;
        arr[right] = arr[left];
    }
    arr[left] = base;

    if(left - start + 1 == _k) return arr[left];
    else if(left - start >= _k) return quick_k(start, left-1, _k);
    else return quick_k(left+1, end, _k-(left-start+1));
}
int main(){
    cin>>n>>m>>k;
    arr[0] = m;
    for(int i = 1; i < n; i ++) arr[i] = 1LL * arr[i - 1] * m % int(1e9 + 7);
    cout << quick_k(0,n-1,k) << endl;
```

循环赛日程表（2 的倍数)n 个选手(编号 1~n)进行循环赛：每个选手必须与其他 n-1 个选手各赛一次；每个选手一天只能赛一次；循环赛一共进行 n-1 天；设计一个 n 行 n-1 列的表，第 i 行第 j 列填入第 i 个选手第 j 天的对手；

```
int arr[32][32],n;
void solution(int n)
{
    if(n==1) return;
    n/=2;
    solution(n);   // 分割直到 n=1 时开始操作
    // 右上块产生
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            arr[i][j+n] = arr[i][j]+n;
    // 右下块复制左上块
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            arr[i+n][j+n] = arr[i][j];
    // 左下块复制右上块
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            arr[i+n][j] = arr[i][j+n];
}
int main(){
    cin>>n;
    arr[1][1] = 1; // 设置初始块为第一位选手
    solution(n);
    for(int i = 1; i <= n; i++){
        for(int j = 2; j <= n; j++)
            cout<<arr[i][j]<<" ";
        cout<<endl;
```

整数位乘:以二进制形式给出两个数，求它们的乘积，也以二进制表示。

```
struct BigBinary{
    std::vector<int> x;    // 由低位到高位保存二进制位
    void check(){          // 检测格式
        x.push_back(0);
        for(int i = 0; i < x.size(); i++){
            x[i+1] += x[i] >> 1;
            x[i] &= 1;
```

```cpp
        }
        while(x[x.size()]>1){
            x.push_back(x[x.size()-1] >> 1);
            x[x.size()-1] &= 1;
        }
        while(!x.empty() && x.back() == 0)
            x.pop_back();
    }
    void Print(){
        check();
        for(int i = x.size()-1; i >= 0; i--) printf("%d", x[i]);
        if(x.empty()) printf("0");
        printf("\n");
    }

    BigBinary(){x.clear();}
    BigBinary(int start, int end, vector<int> _x){
        x.clear();
        for(int i = start; i <= end; i ++) x.push_back(_x[i]);
    }
    BigBinary(char buf[]){
        x.clear();
        for(int i = strlen(buf) - 1; i >= 0; i --) x.push_back(buf[i] == '1');
    }
};
BigBinary Add(const BigBinary &a, const BigBinary &b, int flag=1){
    BigBinary c;
    c.x = a.x;
    for(int i = 0; i < b.x.size(); i ++)
        c.x[i] += b.x[i] * flag;
    return c;
}
BigBinary Mul(const BigBinary &a, const BigBinary &b){
    BigBinary c;
    c.x.resize(a.x.size() * b.x.size() + 1);
    for(int i = 0; i < a.x.size(); i ++)
        for(int j = 0; j < b.x.size(); j ++)
            c.x[i + j] += a.x[i] * b.x[j];
    return c;
}
void MulN2(BigBinary &a, int n_2){
    int size = a.x.size();
    a.x.resize(size + n_2);
    for(int j = size - 1; j >= 0; j--)
        a.x[a.x.size() - size + j] = a.x[j];
    for(int i = n_2 - 1; i >= 0; i --)
        a.x[i] = 0;
}
BigBinary FasterMul(const BigBinary &a, const BigBinary &b){
    if(a.x.size() < 32) return Mul(a, b);
    int n_2 = a.x.size() >> 1;
    BigBinary A(n_2, a.x.size()-1, a.x);
    BigBinary B(0, n_2-1, a.x);
    BigBinary C(n_2, b.x.size()-1, b.x);
    BigBinary D(0, n_2-1, b.x);
    BigBinary A_C = FasterMul(A, C);
    BigBinary B_D = FasterMul(B, D);
    // AD+BC = (A+B)*(C+D)-AC-BD，该方式避免减法出现负数，恒有(A+B)*(C+D) >= AC+BD
    BigBinary ADpBC = Add(Add(FasterMul(Add(A, B), Add(C, D)), A_C , -1), B_D, -1);
    MulN2(A_C, n_2 << 1);
    MulN2(ADpBC, n_2);
    return Add(Add(A_C, ADpBC), B_D);
}
const int maxn = 1e5 + 10;
char buf[maxn];
int main(){
    while(scanf("%s", buf) != EOF)
    {
        BigBinary a(buf);
        scanf("%s", buf);
        BigBinary b(buf);

        // 提前设置 size
        if(a.x.size() < b.x.size()) a.x.resize(b.x.size(), 0);
        else b.x.resize(a.x.size(), 0);

        FasterMul(a, b).Print();
    }
```

(动态规划）**最大子串和**:找一具有最大和的连续子数组，返回其最大和。

```cpp
int n, a[int(1e5 + 10)], dp[int(1e5 + 10)],ans=-1e9;
int main(){
    cin>>n;
    for(int i = 1; i <= n; i ++){
        // 如果之前累加得到的是正数，则继续累加，累加得到负数，取消累加。
        cin>>a[i];
        dp[i] = max(dp[i-1]+a[i],a[i]);
        ans = max(dp[i],ans);
    }
    cout<<ans<<endl;
```

方块堆塔:有 n 种立方体，给出三条棱长，求最高叠多高(下方的底面长宽一定大于上方的).

```cpp
struct{int x,y,z;} cubes[350];
int ans,n,arr[310][310],dp[310];
int dfs(int i){
    if(dp[i]!=-1) return dp[i];
    dp[i] = cubes[i].z;
    for(int j = 0; j <= 3*n; j++)
        if(arr[i][j]) dp[i] = max(dp[i], dfs(j)+cubes[i].z);
    return dp[i];
}
int main(){
    while(cin>>n){
        ans = 0;
        for(int i = 0; i < 3*n; i+=3){
            //每一种立方体的底面有三种情况
            cin>>cubes[i].x>>cubes[i].y>>cubes[i].z;//1
            cubes[i+1].x=cubes[i].y;// 2
            cubes[i+1].y=cubes[i].z;
            cubes[i+1].z=cubes[i].x;
            cubes[i+2].x=cubes[i].z;// 3
            cubes[i+2].y=cubes[i].x;
            cubes[i+2].z=cubes[i].y;
        }
        for(int i = 0; i < 3*n; i++){
            dp[i] = -1; // 初始化 dp 数组
            for(int j = 0; j < 3*n; j++)
                if(cubes[i].x>cubes[j].x&&cubes[i].y>cubes[j].y ||
cubes[i].x>cubes[j].y&&cubes[i].y>cubes[j].x )//两种情况，长宽可换位
                    arr[i][j] = 1;// 标记下标为 j 的立方体能在下标为 i 的立方体上
                else arr[i][j] = 0;
        }
        for(int i = 0; i < 3*n; i++)
            ans=max(ans,dfs(i));
        cout<<ans<<endl;
```

矩阵链相乘的乘法次

```cpp
int n,arr[310],dp[310][310];
int main(){
    while(cin>>n){
        for(int i = 0; i <= n; cin>>arr[i++]);
        for(int i = 0; i <= n; i++)
            for(int j = 0; j <= n; j++)
                dp[i][j]=0;
        for(int r = 2; r <= n; r++){
            int j = r;
            for(int i = 1; i <= n-(r-1); i++){// i 为连乘起点位置，取值不能大于 n-r+1
                j = i+(r-1);// j 为连乘终点位置
                dp[i][j] = dp[i + 1][j] + arr[i - 1] * arr[i] * arr[j];//初始化值
                for(int k = i+1; k < j; k++){ //遍历由 i 到 k 的全部可能的划分点 k，计
算出最优的划分方案
                    int q = dp[i][k] + dp[k + 1][j] + arr[i - 1] * arr[k] * arr[j]; //
计算划分的代价,意思是看在哪套括号最好
                    if (q < dp[i][j])
                        dp[i][j] = q; //最优的值保存在 m[i][j]中
                }
            }
        }
        cout << dp[1][n] << endl;
```

括号匹配 II

```cpp
int dp[105][105];
string str;
bool judge(char a, char b){
    return ((a=='(' && b==')')||(a=='[' && b==']')||(a=='{' && b=='}'));
}
int main(){
    while(cin>>str){
        memset(dp,0,sizeof(dp));
        for(int i = 0; i < str.length(); i++) dp[i][i] = 1; // 自己对自己的符号缺失
数为 1
        for (int len = 1; len < str.length(); len++) // 从长度为 2 开始，记录每个长
度为 len 的括号补充数
            for (int i = 0; i < str.length() - len; i++)
            {
                int j = i + len;
                dp[i][j] = 1e9; // 初始化一个极大值，方便后面缩小
                // 检查当前子字符串是否是一个有效的括号序列，如果是，我们更新最少插入
次数为左右两边字符组成的子序列的最小插入次数。
                if (judge(str[i], str[j]))
                    dp[i][j] = min(dp[i][j], dp[i + 1][j - 1]);
                // 在当前子字符串计算各个分割点 k 处左右两边字符组成的子序列的最小插
入次数之和，取最小值作为当前子字符串的最少插入次数。
                for (int k = i; k < j; k++)
                    dp[i][j] = min(dp[i][j],dp[i][k] + dp[k + 1][j]);
            }
        if (dp[0][str.length() - 1] != 0)
            cout << dp[0][str.length() - 1] << endl;
        else
            cout << "SZTU_WOD YYDS!" << endl;
```

物品无限的背包问题（01 **背包**则把 dp[i][j-w[i]]+v[i]改为 dp[i-1][j-w[i]]+v[i])

```cpp
int n,b,w[1000],v[1000],dp[1001][10001];
int main(){
    cin>>n>>b;
    for(int i = 0; i <= n; i++) dp[i][0] = 0;
    for(int i = 1; i <= b; i++) dp[0][i] = 0;
    // i 为要考虑拿的第 i 物品，j 为背包容量
    for(int i = 1; i <= n; i++){
        cin>>w[i]>>v[i];
        for(int j = 1; j <= b; j++){
            dp[i][j] = (j>=w[i]) ? max(dp[i-1][j],dp[i][j-w[i]]+v[i]) : dp[i-1][j];
        }
    }
    cout<<dp[n][b]<<endl;
```

最长上升子序列（小规模）

```cpp
int main(){
    int n,ans = 0,arr[1000],dp[1000];
    cin>>n>>arr[1];
    for(int i = 2; i <= n; i ++) // 生成序列
        arr[i] = 1LL * (arr[i - 1] + 1) * (arr[i - 1] + 1) % int(1e9 + 7);
```

```
    for(int k = 1; k <= n; k++){
        dp[k] = 1; // 初始化计算长度，自身至少长为 1
        for(int j = 1; j < k; j++)
            if(arr[j] < arr[k]) dp[k] = max(dp[k],dp[j] + 1);
        ans = max(dp[k],ans);
    }
    cout<<ans;
```

最长上升子序列（大规模）

```
int n,dp[int(1e6)],arr[int(1e6)];
int main() {
    cin>>n>>arr[1];
    for(int i = 2; i <= n; i ++) // 生成序列
        arr[i] = 1LL * (arr[i - 1] + 1) * (arr[i - 1] + 1) % int(1e9 + 7);
    // dp 的内容不能作为最终字串结果，但能得到最长字串长度。
    int len = 0,l,r,mid;
    for(int i=1;i<=n;i++){
        l=0; r=len;
        while(l<r){
            mid = (l+r+1)>>1;
            if(dp[mid]<arr[i]) l=mid;
            else r=mid-1;
        }
        len=max(len,l+1);
        dp[l+1]=arr[i];
    }
    cout<<len<<endl;
```

最长公共子序列

```
int dp[1001][1001];
string a,b;
int main(){
    for (int i = 0; i < 1001; i++)
        dp[0][i] = dp[i][0] = 0;

    while(cin>>a>>b){
        for(int i = 1; i <= a.length(); i++)
            for(int j = 1; j <= b.length(); j++){
                if(a[i-1]==b[j-1])
                    dp[i][j] = dp[i-1][j-1]+1;
                else
                    dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
            }
        cout<<dp[a.length()][b.length()]<<endl;
```

写出下述线性规划的对偶

max $2x_1-x_2+3x_3$

s.t. $x_1+3x_2-2x_3 \le 5$

$-x_1-2x_2+x_3=8$

$x_1 \ge 0, x_2 \ge 0, x_3$ 任意

令$x_3=x_3'-x_3''$,

$A=B$等价于$A\le B$和$-A\le -B$,

max $2x_1-x_2+3x_3'-3x_3''$

s.t. $x_1+3x_2-2x_3'+2x_3''\le 5$

$-x_1-2x_2+x_3'-x_3''\le 8$

$x_1+2x_2-x_3'+x_3''\le -8$

$x_1\ge 0, x_2\ge 0, x_3'\ge 0, x_3''\ge 0$

对偶规划为

min $5y_1+8y_2'-8y_2''$

s.t. $y_1-y_2'+y_2''\ge 2$

$3y_1-2y_2'+2y_2''\ge -1$

$-2y_1+y_2'-y_2''\ge 3$ $2y_1-y_2'+y_2''\ge -3$

$y_1\ge 0, y_2'\ge 0, y_2''\ge 0$

令$y_2=y_2'-y_2''$, 合并后2个不等式

min $5y_1+8y_2$

s.t. $y_1-y_2\ge 2$

$3y_1-2y_2\ge -1$

$-2y_1+y_2=3$

$y_1\ge 0, y_2$任意

（贪心）活动选择

```
struct Time{int start; int end;} arr[10000];
int n, cur_end, ans;
bool comp(Time a, Time b){return a.end<b.end;}
int main(){
    cin>>n;
    for(int i = 0; i < n; i++)
        cin>>arr[i].start>>arr[i].end;
    sort(arr, arr+n, comp);
    ans = 1; cur_end = arr[0].end;
    for(int i = 1; i < n; i++){
        if(cur_end <= arr[i].start){
            cur_end = arr[i].end;
            ans += 1;
        }
    }
    cout<<ans;
```

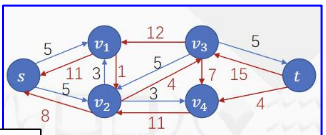
最少拦截系统

```
int t,arr[100],height,cnt,flag;
int main(){
    while(cin>>t){
        cnt = 0;arr[0] = 0;
        while(t--){
            cin>>height;
            flag = 0;
            for(int i = 0; i < cnt; i++)
                if(height<arr[i]){
                    arr[i] = height;
                    flag = 1;
                    break;
                }
            if(!flag) arr[cnt++] = height;
        }
        cout<<cnt<<endl;
```

3

"方法"而不是"算法"：Ford-Fulkerson有不同的实现方式

- 斜对称性：$f(u,v)=-f(v,u)$
  - u流向v的流量，可以抽象地看作v流向u的负流量
- 残存网络：$c_f(u,v)=c(u,v)-f(u,v)$



残存网络

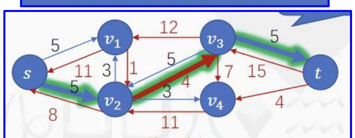
4

增广路：残存网络中s到t的简单路径

残存容量：增广路上能加推的最大流量

抵消操作：增广路加推流量时，部分边撤回的原流量

抵消操作就是"反悔"部分之前推的流
使算法不必担心推流的顺序



增广路径

（回溯）

枚举全排列

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

int n,arr[8],ans[8],vis[8];
void dfs(int cur){
    if(cur==n){
        for(int i = 0; i < n; i++)
            cout<<ans[i]<<" ";
        cout<<endl;
        return;
    }
    for(int i = 0; i < n; i++){
        if(!vis[i]){
            vis[i] = 1;
            ans[cur] = arr[i];
            dfs(cur+1);
            vis[i] = 0;
        }
    }
}
```

```
int main(){
    cin>>n;
    for(int i = 0; i < n; i++){
        cin>>arr[i];
        vis[i] = 0;
    }
    sort(arr,arr+n);
    dfs(0);
```

6.4.3 单纯形算法

$z = 2x_1 + x_2 + x_3$

$x_4 = 10 - x_1 - x_2 + 2x_3$

$x_5 = 10 - 2x_1 - 2x_2 - x_3$

$x_6 = 12 - 3x_1 + x_2 - 2x_3$

选择新的基变量，不断*Pivot*

观察目标函数，提升系数为正的变量，就能让z增加

x_1,x_2,x_3的系数都为正，选择系数最大的x_1

x_1提升的瓶颈是：

$x_4 = 10 - x_1 - x_2 + 2x_3$：$x_1$最大是10

$x_5 = 10 - 2x_1 - 2x_2 - x_3$：$x_1$最大是5

$x_6 = 12 - 3x_1 + x_2 - 2x_3$：$x_1$最大是4

考虑所有变量的非负约束

松弛变量x_6对x_1的约束最"紧"——x_1的增加至多会把x_6变成0

生成可重排列

```
int n,m,arr[8],ans[8],cnt[8];
void dfs(int cur){
    if(cur==n){
        for(int i = 0; i < n; i++)
            cout<<ans[i]<<" ";
        cout<<endl;
        return;
    }
    for(int i = 0; i < m; i++){
        if(cnt[i]){
            cnt[i]--;
            ans[cur] = arr[i];
            dfs(cur+1);
            cnt[i]++;
        }
    }
}
int main(){
    cin>>n;
    m = 0;
    for(int i = 0; i < n; i++){
        cin>>arr[i];
        cnt[i] = 0;
    }
    sort(arr,arr+n);
    for(int i = 0; i < n; i++){
        if(m==0 || arr[m-1] != arr[i])
            arr[m++] = arr[i];
        cnt[m-1]++;
    }// 利用 cnt 计算重复元素个数，避免同一层相同元素重复调用；
    dfs(0);
```

$z = -3x_1 + 2x_2 - x_3 + x_4$

$x_5 = -10 + x_1 + 3x_2 - 3x_3 + 3x_4$

$x_6 = -30 - 4x_1 + x_2 + 5x_3 - 5x_4$

$x_7 = 30 + 4x_1 - x_2 - 5x_3 + 5x_4$

$\xrightarrow{Pivot(5,1)}$

$z = -30 + 11x_2 - 10x_3 +$

$x_1 = 10 - 3x_2 + 3x_3 -$

$x_6 = -70 + 13x_2 - 7x_3 +$

$x_7 = 70 - 13x_2 + 7x_3 -$

转动（Pivot）操作：$Pivot(l,e)$

将非基本变量x_e（替入变量）替换x_l（替出变量）成为基本变量

以$Pivot(5,1)$为例：

$x_5 = -10 + x_1 + 3x_2 - 3x_3 + 3x_4 \Rightarrow x_1 = 10 - 3x_2 + 3x_3 - 3x_4 + x_5$

将所有其他x_1出现的地方用这个等式替换，x_5就成为了非基变量

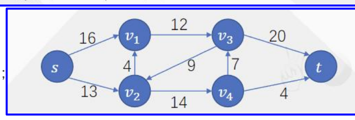
生成 r 子集

```
int n,r,arr[10];
bool sel[10];
void dfs(int cur, int rcnt){
    if(cur==-1){
        if(rcnt != r) return;
        for(int i = 0; i < n; i++)
            if(sel[i]) cout<<arr[i]<<" ";
        cout<<endl;
        return;
    }
    sel[cur] = false;
    dfs(cur-1,rcnt);
    sel[cur] = true;
    dfs(cur-1,rcnt+1);
}
int main(){
    while(cin>>n>>r){
        for(int i = 0; i < n; cin>>arr[i++]);
        dfs(n-1, 0);
```

1

网络（流网络，Flow Network）

- 有向图$G=(V,E)$
- 每条边$(u,v)\in E$有权值$c(u,v)$表示容量（物料的最大流速）
- $(u,v)\notin E$时$c(u,v)=0$
- 有两个特殊的点：源点$s\in V$，汇点$t\in V$



n 皇后问题

```
// 我的解法(模拟真棋盘，空间耗费多)
int n,cnt,qp[13][13];
bool check(int r, int c){
    for(int i = 0; i < r; i++)
        if( (qp[i][c]) || (c+(r-i)<n && qp[i][c+(r-i)]) || (c-(r-i)>=0 &&
qp[i][c-(r-i)]) ) return false;
    return true;
}
void dfs(int cur){ // 棋盘第 cur 行
    if(cur==n){ cnt++; return;}
    for(int i = 0; i < n; i++)
        if(!qp[cur][i] && check(cur,i)){
            qp[cur][i] = 1;
            dfs(cur+1);
            qp[cur][i] = 0;
        }
}
int main(){
    while(cin>>n){
        cnt = 0;
        dfs(0);
        cout<<cnt<<endl;
```

2

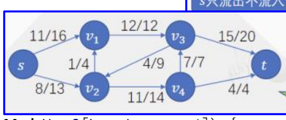
流：$f(u,v)$定义在$(u\in V, v\in V)$上的实数函数且满足：

- 容量限制：$f(u,v)\le c(u,v)$
- 流量守恒：$\forall u\in V-\{s,t\}$有$\sum_{v\in V} f(v,u)=\sum_{u\in V} f(u,v)$
  - 除源点s和汇点t外，流入节点的总量等于节点流出的总量
- "流"的流量：$|f|=\sum_{v\in V} f(s,v)-\sum_{v\in V} f(v,s)$

节点不"囤积"流量

s只流出不流入，$\sum_{v\in V} f(v,s)=0$

```
// 答案解法(数值处理直线、对角线)
int n, count;
bool column[13], diag1[25], diag2[25];
void backtrack(int k) {
    if (k == n) {count++; return;}
    for (int i = 0; i < n; i++)
        if (!column[i] && !diag1[k + i] && !diag2[k - i + n - 1]) {
            column[i] = diag1[k + i] = diag2[k - i + n - 1] = true;
            backtrack(k + 1);
            column[i] = diag1[k + i] = diag2[k - i + n - 1] = false;
        }
}
int main() {
    while (cin >> n) {
        for (int i = 0; i < n; i++) column[i] = false;
```



```
        for (int i = 0; i < 2 * n - 1; i++) diag1[i] = diag2[i] = false;
        count = 0;
        backtrack(0);
        cout << count << endl;
```

着色问题

```
int n,m,q,graph[11][11],color[11],r,c,cnt;
bool check(int cur, int col){
    for(int i = 0; i < n; i++)
        if(graph[cur][i] && color[i]==col) return false;
    return true;
}
void dfs(int cur){
    if(cur==n){cnt++;return;}
    for(int i = 1; i<=m; i++){
        if(check(cur,i)){
            color[cur] = i;
            dfs(cur+1);
            color[cur] = 0;
        }
    }
}
int main(){
    while(cin>>n>>m>>q){
        for(int i = 0; i < n; i++){
            color[i] = 0;
            for(int j = 0; j < n; j++)
                graph[i][j] = 0;
        }
        while(q--){
            cin>>r>>c;
            graph[r-1][c-1] = graph[c-1][r-1] = 1;
        }
        cnt = 0;
        dfs(0);
        cout<<cnt<<endl;
```

最佳安排

```
int n,arr[11][11],ans;
bool select[11];
void dfs(int cur, int sum){
    if(sum>=ans)return;
    if(cur == n){ans = min(sum, ans); return;}
    for(int i = 0; i < n; i++)
        if(!select[i]){
            select[i]=true;
            dfs(cur+1, sum + arr[cur][i]);
            select[i]=false;
        }
}
int main(){
    while(cin>>n){
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                cin>>arr[i][j];
        ans = 1e9;
        dfs(0, 0);
        cout<<ans<<endl;
```

合适的 01

```
string str;
int n,cnt;
void dfs(int cur, int sum){
    if(cur == n+1){
        if(sum>0) cnt++;
        return;
    }
    dfs(cur+1, sum); // 0 不用处理加减
    if(cur==0 || str[cur-1]=='+') dfs(cur+1,sum+1);
    else dfs(cur+1, sum-1);
}
int main(){
    while(cin>>str){
        cnt = 0;
        n = str.length();
        dfs(0,0);
        cout<<cnt<<endl;
```

最大团(分支界限)

完全图：如果无向图中的任何一对顶点之间都有一条边，这种无向图称为完全图。
完全子图(团)：给定无向图 G=(V,E)。如果 U⊆V，且对任意 u，v⊆U 有(u，v) ⊆ E，则称 U 是 G 的完全子图。V 是顶点集，E 是边集。
最大团：G 的最大团是指 G 中所含顶点数最多的团。
空子图：在一个图的子图中，边集为空集的子图。简单来说，空子图包含一些顶点，但这些顶点之间没有任何边相连。
独立集：对于给定无向图 G=(V,E)。如果顶点集合 V*⊆V，若 V*中任何两个顶点均不相邻，则称 V*为 G 的点独立集，或简称独立集。
最大独立集：G 中所含顶点数最多的独立集。

```
int g[36][36],vis[36],n,m,ans,r,c;
bool check(int cur){
    for(int i = 0; i < n; i++)
        if(vis[i] && !g[cur][i]) return false;
    return true;
}
void dfs(int cur,int cnt){
    if(cur==n){ans = max(ans, cnt); return;}
    // 分支限界：剩下的点都算上也不如当前的界则回溯
    if(cnt + n - cur + 1 <= ans)return;
    if(check(cur)){
        vis[cur] = 1;
        dfs(cur+1,cnt+1); // 加 cur
    }
    vis[cur] = 0;
    dfs(cur+1,cnt); // 不加 cur
}
int main(){
    while(cin>>n>>m){
        memset(g, 0, sizeof(g));
        memset(vis, 0, sizeof(vis));
        for(int i = 0; i < m; i++){
            cin>>r>>c;
            g[r-1][c-1] = g[c-1][r-1] = 1;
        }
        ans = 0;
        dfs(0, 0);
        cout<<ans<<endl;
```

旅行商

```
int g[13][13],vis[13],n,ans,r,c,v;
int path[13]; //记录路径
int min_dis[13]; //每个点出发的最小边长度
// 当前点 , 当前路径长度, 剩余最短路径长度
void dfs(int cur, int dis, int min_dis_sum)
{
    if (cur == n-1){ans = min(dis + g[path[cur]][0], ans); return;}
    if(dis + min_dis_sum >= ans) return; // 分支限界：已确认路长+剩下点最小边之和
    for(int i = 1; i < n; i ++) { //回路, 1 起点, 2 开始查找
        if(!vis[i]){
            vis[i] = true;
            path[cur+1] = i;
            dfs(cur + 1, dis + g[path[cur]][i], min_dis_sum - min_dis[i]);
            vis[i] = false;
        }
    }
}
int main(){
    while(cin>>n){
        memset(g,0,sizeof(g));
        memset(vis,0,sizeof(vis));
        memset(min_dis,1e9,sizeof(min_dis));
        // fill(min_dis, min_dis+13, 1e9);
        int m = n * (n - 1) / 2;
        for(int i = 0; i < m; i++){
            cin>>r>>c>>v;
            g[r-1][c-1] = g[c-1][r-1] = v;
            // 获取每个点最短路径长度
            min_dis[r-1] = min(min_dis[r-1],v);
            min_dis[c-1] = min(min_dis[c-1],v);
        }
        int min_dis_sum = 0; // 所有点最短路径长度之和
        for(int i = 0; i < n; i++) min_dis_sum += min_dis[i];
        ans = 1e9;
        path[0] = 0; //回路, 固定 1 为起点
        dfs(0, 0, min_dis_sum);
        cout << ans << endl;
```

圆排列最小宽度

```
int n,selected[12],arr[12];// 圆排列顺序
double ans,r[12],pos[12];// 圆的半径/圆心位置
double get_pos(double R, int cur){ // 计算当前圆的圆心位置
    // 第一个圆, 将圆心放置在半径处
    if (cur == 0) return R;
    // 遍历已排列的圆, 找到与当前圆相切的最右位置
    // 要考虑当前圆比较大, 碰到的不一定是左侧相邻的圆
    double d, cur_pos = R;
    for (int i = 0; i < cur; i++){
        d = sqrt(pow(r[arr[i]] + R, 2) - pow(r[arr[i]] - R, 2));
        cur_pos = max(cur_pos, pos[i] + d);
    }
    return cur_pos;
}
bool judge(int cur, double len) { // 分支界限
    double min_r = 1e11;
    for (int i = 0; i < n; i++) // 找到未排列圆中半径最小的圆
        if (!selected[i]&&r[i]<min_r) min_r = r[i];
    // 计算未排列最小圆的圆心位置
    double min_r_pos = get_pos(min_r, cur);
    // 设未排列最小圆为所有未排列圆的半径,计算全部加入后的最右位置,若比当前最优解差, 则剪枝
    double cur_len = max(len, min_r_pos + min_r * ((n - cur) * 2 - 1));
    return cur_len < ans;
}
void dfs(int cur, double len)
{
    if (cur == n){ans = min(len,ans); return;}
    if(!judge(cur,len)) return;
    for (int i = 0; i < n; i++){
        if (!selected[i]){
            selected[i] = true;
            double cur_pos = get_pos(r[i], cur);
            // 更新排列的最右端位置
            // 考虑到最右端的圆可能会比较小,所以最右边的位置不一定是最右端的圆+其半径
            double real_cur_len = max(len, cur_pos + r[i]);
            arr[cur] = i;
            pos[cur] = cur_pos;
            dfs(cur + 1, real_cur_len);
            selected[i] = 0;
        }
    }
}
int main(){
```

```
    cin >> n;
    for (int i = 0; i < n; cin >> r[i++]);
    memset(selected,0,sizeof(selected));
    ans = 1e9;
    dfs(0, 0);
    cout << fixed << setprecision(6) << ans;
```

线性规划例题

```
const double eps = 1e-7;   // 定义一个很小的数，用于比较浮点数的精度
const double inf = 1e20;   // 定义一个很大的数，用于表示无穷大
// 标准型： max Σcx, s.t. ax<=b, x>=0
class Simplex{
public:
    vector<double> b, c;      // b: 约束条件右端项向量,c: 目标函数系数向量
    vector<vector<double>> a;   // a: 约束条件系数矩阵
    double z;                  // 目标函数值
    int m, n;                  // m: 约束条件个数, n: 自由变量个数
    Simplex(int _m, int _n);
    void Pivot(int e, int l);
    double Solve();
};
```

依次输入

$$\begin{matrix} c_1 & c_2 & c_3 & \\ a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{matrix}$$

$$\begin{aligned} \min\ & c_1x_1 + c_2x_2 + c_3x_3 \\ s.t.\ & a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \le b_1 > 0 \\ & a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \le b_2 > 0 \\ & a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \le b_3 > 0 \\ & x_1, x_2, x_3 \ge 0 \end{aligned}$$

```
// 构造函数，初始化 Simplex 对象
Simplex::Simplex(int _m, int _n) : m(_m), n(_n), z(0){
    a.resize(m + 10, vector<double>(n + 10, 0));
    b.resize(m + 10, 0);
    c.resize(n + 10, 0);
}
// 枢轴操作函数，实现高斯消元法
void Simplex::Pivot(int e, int l){
    b[l] /= a[l][e];
    for (int j = 0; j < n; j++)
        if (j != e) a[l][j] /= a[l][e];
    a[l][e] = 1 / a[l][e]; // 将主元逆转
    for (int i = 0; i < m; i++){
        if (i == l || (a[i][e] > -eps && a[i][e] < eps)) continue; // 排除当前行和
近似为 0 的行
        b[i] -= a[i][e] * b[l];
        for (int j = 0; j < n; j++)
            if (j != e) a[i][j] -= a[i][e] * a[l][j];
        a[i][e] = -a[i][e] * a[l][e];
    }
    z += c[e] * b[l];                    // 更新目标函数值
    for (int j = 0; j < n; j++)
        if (j != e)                      // 排除主元列
            c[j] -= c[e] * a[l][j];        // 更新目标函数系数
    c[e] *= -a[l][e];                    // 更新主元系数
}
// 求解函数，使用单纯形法求解线性规划问题
double Simplex::Solve(){
    while (true){
        int e = -1, l = -1; // e: 入基变量索引, l: 出基变量索引
        double maxc = eps; // 初始化最大目标系数为很小的数
        // 找到最优解的入基变量,找目标函数里最大的正系数
        for (int j = 0; j < n; j++)
            if (c[j] > maxc){
                maxc = c[j];
                e = j; // 确认替入变量
            }
        if (e == -1) return z;// 如果没有入基变量，说明当前解是最优的
        // 有入基变量，则准备转动
        double minba = inf; // 初始化最小比值为无穷大
        // 找到最优解的出基变量,
        for (int i = 0; i < m; i++)
            if (a[i][e] > eps && minba > b[i] / a[i][e]){
                minba = b[i] / a[i][e]; // 找替入变量增加的瓶颈
                l = i; // 替出变量是卡住替入变量增长瓶颈那一行的松弛变量
            }
        if (l == -1) return inf;// 如果没有出基变量，说明无界解
        Pivot(e, l);
    }
}
```

j 种产品的售价，$0 \le c_j \le 1000$;

个数 $\{a_{ij}|j \in 1,2,\ldots,n\}$ 表示生产第 j 种产品对第 i 种原料的消耗量，第 $n+1$ 个数

```
int main(){
    Simplex spx(3, 3);
    cin >> spx.c[0] >> spx.c[1] >> spx.c[2];      // 输入目标函数系数
    for (int i = 0; i < 3; i++){
        // 转换为标准型：目标函数统一为 max
        spx.c[i] *= -1;
        // 输入约束条件
        for(int j = 0; j < 3; j++) cin >> spx.a[i][j];
        cin >> spx.b[i];
    }
    double res = spx.Solve();
    // 如果返回值接近无穷大，说明无解
    if (res >= inf - eps) cout<<"No solution"<<endl;
    else cout<<fixed<<setprecision(2)<<-res<<endl; // 输出最优解，并将其取反恢复原
来的符号
}
```

}用 m 种原料生产种产品，每种 n 产品需要消耗各类原料，同时有个固定售价，求各类产品最优生产数量，使全部售卖得到的收入最高.

- 第一行为$1 \le n \le 10$ 和 $1 \le m \le 100$;
- 第二行为 n 个数，$\{c_j|j \in 1,2,\ldots,n\}$ 表示第 j
- 接下来 m 行，每行 $n+1$ 个数，第 i 行前 n 个

  b_i 为第 i 种原料的储量，$0 \le a_{ij}, b_i \le 1000$.

```
int main(){
    int m,n;
    cin>>n>>m;
    Simplex spx(m, n);
    for(int i = 0; i < n; i++){
        cin >> spx.c[i]; // 输入目标函数系数
        // spx.c[i] *= -1;  // 转换为标准型：目标函数统一为 max
    }
    for (int i = 0; i < m; i++){
        // 输入约束条件
        for(int j = 0; j < n; j++) cin >> spx.a[i][j];
        cin >> spx.b[i];
    }
    double res = spx.Solve();
    // // 如果返回值接近无穷大，说明无解
    cout<<fixed<<setprecision(6)<<res<<endl;
```

防守战线

战线可以看作一个长度为 n 的序列，现在需要在这个序列上建塔来防守敌兵，

以建任意多的塔，费用累加计算。有 m 个区间 $[L_1,R_1],[L_2,R_2],\ldots,[L_m,R_m]$,

敌兵，在序列第 i 号位置上建一座塔的花费为 C_i，且一个位置可

$R_m]$，在第 i 个区间的范围内要建至少 D_i 座塔，求最少花费。

第一行为两个数 序列长度n(<1000)，区间个数m(<10000)

第二行 n 个数，描述序列 Ci(<10000)

接下来 m 行，每行有三个数 Li,Bi,Di，描述一个区间 (1≤L<B<n,Di<10000)

```
int main() {
    int n, m;
    cin>>n>>m;
    Simplex spx(n, m);
    vector<double> C(n);
    vector<int> L(m), R(m);
    vector<double> D(m);
    for(int i = 0; i < n; i ++
        cin>>C[i];
    for(int i = 0; i < m; i ++)
        cin>>L[i]>>R[i]>>D[i];
    // 对偶转换
    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < m; j ++)
            spx.a[i][j] = i >= L[j] - 1 && i <= R[j] - 1;
        spx.b[i] = C[i];
    }
    for(int j = 0; j < m; j ++) spx.c[j] = D[j];
    cout<<(int)spx.Solve()<<endl;}
```

志愿者招募

奥运将至，布布需要为奥运项目招募一批短期志愿者。经过估算，这

通过了解得知，一共有 m 类志愿者可以招募。其中第 i 类可以从第

招足够多的志愿者，请你帮他设计一种最优的招募方案。

这个项目需要 n 天才能完成，其中第 i 天至少需要 a_i 个志愿者。布布

s_i 天工作到 t_i 天，招募费用是每人 c_i 元。布布希望用尽量少的费用

```
int main(){
    int n, m;
    cin >> n >> m;
    Simplex spx(m, n)
    vector<int> S(m), T(m), C(m), A(n);
    for (int i = 0; i < n; i++)
        cin >> A[i];
    for (int i = 0; i < m; i++)
        cin >> S[i] >> T[i] >> C[i];
    // 对偶转换
    for (int i = 0; i < n; i++)
        spx.c[i] = A[i];
    for (int j = 0; j < m; j++){
        for (int i = 0; i < n; i++)
            spx.a[j][i] = (i + 1 >= S[j] && i + 1 <= T[j]);
        spx.b[j] = C[j];
    }
    cout << fixed << setprecision(0) << spx.Solve() << endl;
```

最大流问题

```
const int MAXN = 300; // 定义最大节点数为 300
int N,M,S,T; //点数量，边数量，源点，汇点
int C[MAXN][MAXN]; // 容量矩阵
int F[MAXN][MAXN]; // 流量矩阵，用于存储每条边当前的流量
int parent[MAXN];  // 父节点数组，用于存储增广路径中的父节点
bool bfs(){
    bool visited[MAXN] = {false};
    queue<int> queue;
    queue.push(S);// 源点访问
    visited[S] = true;
    while(!queue.empty()){
        int u = queue.front();
        queue.pop();
        for(int i = 0; i<N; i++)
            if(!visited[i]&&C[u][i]>F[u][i]){
                queue.push(i);
                visited[i] = true;
                parent[i] = u;
                if(i==T) return true;
            }
    }
    return false;
}
```

给一有向图，源点及汇点计算其最大流。网络中顶点号为1~N。

第一行给节点数N(1<=N<=200), 有向边数M(N<=M<=5000),源点序号S,汇点T。

接下来M行每行包括三个正整数ui，vi，wi，表示第 i条有向边从 ui出发，到达

vi，边权为 wi。0<wi<=10000。

```
int edmonds_karp(){
    int max_flow = 0;
    while(bfs()){
        int path_flow = 1e9; // 初始化路径流为无限大
        for(int i = T; i != S; i = parent[i]){ // 找到最小的路径流
            int u = parent[i];
            path_flow = min(path_flow, C[u][i] - F[u][i]);
        }
        for(int i = T; i != S; i = parent[i]){
            int u = parent[i];
            F[u][i] += path_flow;
            F[i][u] -= path_flow;
        }
        max_flow += path_flow; // 增加路径流到最大流
    }
    return max_flow;
}
int main(){
    // 初始化全局变量 C
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) {
            C[i][j] = 0;
        }
    cin>>N>>M>>S>>T;
    --S;--T;
    int u,v,w; //边起点，边终点，边容量
    for(int i = 0; i < M; i++){
        cin>>u>>v>>w;
        --u;--v;
```

```
        C[u][v] += w;
    }
    cout<<edmonds_karp()<<endl;
}
```

基本的二分图

```
int main() {
    int n, m, e; // 左右两侧顶点数和边数
    cin >> n >> m >> e;
    N = n + m + 2; // 总节点数，包括源点和汇点
    S = 0; // 源点
    T = N - 1; // 汇点
    // 初始化容量矩阵 C
    memset(C, 0, sizeof(C)
    memset(F, 0, sizeof(F)
    // 读入边并构建图
    for (int i = 0; i < e; ++i) {
        int u, v;
        cin >> u >> v;
        // 左侧点 u 连接到右侧点 v，u 在源点后的编号是 u，v 在右侧的编号是 v+n
        C[u][v + n] = 1; // 每条边的容量为 1
    }
    for (int i = 1; i <= n; ++i) { // 源点连接到所有左侧点
        C[S][i] = 1; // 容量为 1
    }
    for (int i = 1; i <= m; ++i) { // 所有右侧点连接到汇点
        C[i + n][T] = 1; // 容量为 1
    }
    cout << edmonds_karp() << endl;
}
```

给定二分图，左边 n 个点，右边 m 个点，之间有 e 个边，求最大匹配边数。
每组数据第一行为 1 ≤ n, m ≤ 500, 1 ≤ e ≤ 10000。
接下来 e 行每行两个数字，前一个为左边的点编号 1 ≤ ai ≤ n，后一个为右边点编号 1 ≤ bi ≤ m。

收费站建设

```
int main() {
    cin >> N >> M;
    S = 0; // 源点编号 (A 城
    T = 1; // 汇点编号 (B 城
    // 初始化容量矩阵 C 和流量矩阵 F
    memset(C, 0, sizeof(C));
    memset(F, 0, sizeof(F));
    // 读入边并构建图
    for (int i = 0; i < M; ++i) {
        int u, v, c;
        cin >> u >> v >> c;
        --u; --v; // 将节点编号转换为从 0 开始
        C[u][v] += c; // 容量
        C[v][u] += c; // 因为无向图，反向边也要设置容量
    }
    cout << edmonds_karp() << endl;
}
```

要选择一些公路段，在路上建设收费站，保证A城市无论通过什么路径到B城市都一定经过收费站。而收费站的建设成本与所在的公路段车流容量成正比，姑且看作等于其车流容量的数值。最少花费多少成本能够满足收费站的建设目的。

每组数据第一行 n, m 分别表示城市数量和公路段数量，其中 $1 \le n \le 50, 1 \le m \le 500$。

城市编号为 $1 \sim n$，A城市编号为 1，B城市编号为 2。

接下来 m 行，每行三个空格隔开的正整数 s, e, c 表示城市 s 与城市 e 之间的公路段车流容量为 c。

数据保证 A 能够到达 B。

较轻的小球

```
int n, m, cur;
int solu(int l, int r){
    cin>>cur;
    int right = ((r-l+1)&1) ? r-1 : r;
    if(cur == 2) return r;
    else if(cur == 1){
        if(l == right-1) return right;
        return solu((l+right)/2+1, right);
    }
    else if(cur == 0){
        if(l == right-1) return l;
        return solu(l, (l+right)/2);
    }
}
int main(){
    while(cin>>n>>m)
        cout<<solu(1, n)<<endl;
}
```

01 饭卡

```
int n,m,ans,min_bal,arr[1001],dp[1001][1001];
int main(){
    while(cin>>n && n!=0){
        memset(dp,0,sizeof(dp));
        for(int i = 1; i <= n; cin>>arr[i++]); //考虑到与 dp 做兼容, 从 1 开始.
        cin>>m;
        if(m<5) {cout<<m<<endl; continue;}
        m -= 5;

        sort(arr+1,arr+n+1); // 对价格排序
        int max_price = arr[n]; // 得最大价
        for(int i = 1; i < n; i++) // 考虑第 i 件物品,到第 n-1 件, 最后一件价格最大另
外处理
            for(int j = 0; j <= m; j++) //余额
                if(j>=arr[i])
                    dp[i][j] = max(dp[i-1][j], dp[i-1][j-arr[i]]+arr[i]);
                else
                    dp[i][j] = dp[i-1][j];
        ans = m+5-dp[n-1][m]-max_price;
        cout << ans << endl;
```

开机 I

```
int ans = 0, n, temp, arr[1010];
int main(){
    arr[0] = 0;
    cin>>n;
    for(int i = 1; i <= n; i++) cin>>arr[i];
    sort(arr,arr+n+1);
    for(int i = 1; i <= n; i++)
        ans += (temp += arr[i] + arr[i-1]);
    cout<<ans<<endl;
```

若 $n^{\log_m k} > f(n)$，则 $T(n) = \Theta(n^{\log_m k})$
若 $n^{\log_m k} = f(n)$，则 $T(n) = \Theta(f(n)\cdot\log n)$
若 $n^{\log_m k} < f(n)$，则 $T(n) = \Theta(f(n))$

合适的顺序

```
int calculateWeightSum() { // 计算当前排列的权值和
    int sum = 0;
    for (int i = 1; i <= 8; ++i)
        sum += arr[i-1] * arr[i] * arr[i + 1];
    return sum;
}
int main() {
    while(cin >> arr[1]){ // 读入 8 个数
        for (int i = 2; i <= 8; ++i) cin >> arr[i];
        arr[0]=arr[9]=1;
        // 对所有排列进行计算
        int maxWeightSum = 0;
        sort(arr+1, arr+9);
        do {
            int currentWeightSum = calculateWeightSum();
            if (currentWeightSum > maxWeightSum)
                maxWeightSum = currentWeightSum;
        } while (next_permutation(arr+1, arr+9));
        cout << maxWeightSum << endl;
```

$$T(n) = \begin{cases} O(1) & n = 1 \\ kT(\frac{n}{m}) + f(n) & n > 1 \end{cases}$$

常数阶 O(1)
对数阶 O(log2n)
线性阶 O(n)
线性对数阶 O(nlog2n)
平方阶 O(n^2)
立方阶 O(n^3)
k 次方阶 O(n^k)
指数阶 O(2^n) 低->

算法是一系列解决问题的清晰指令准确而完整的描述。**特性**：可行性：由若干有限时间结束的基本操作组成；有穷性：执行有限次基本操作后终止；确定性：同样的输入输出相同的结果；有输入；有输出。》**最坏时间复杂度**：最坏情况下的运算执行次数；**平均时间复杂度**：根据输入的概率分布计算平均所需运算次数。》用定义域为自然数集 N 的函数来定义算法的渐进运行时间》**程序**：是算法用某种程序设计语言的具体实现。**程序与算法的区别**：程序可以不满足算法的第四点性质即有限性。》**分治**：递归地调用自身解决紧密相关的若干子问题，**步骤**：1，分解原问题为若干不相交的子问题；2，解决这些子问题，递归进行；3，合并这些子问题的解为原问题的解。**优化策略**：用一部分子问题的解表达另一部分子问题，减少计算子问题的个数》**贪心**：每次选择局部最优；证明贪心有反证法、归纳法、交换论证法。

渐进上界符号 O：如果存在正的常数 C 和自然数 N0，使得当 N >= N0 时有 f(N)<=Cg(N)，则称函数 f(N) 当 N 充分大时上有界，且 g(N)是它的一个上界，记为 T(N)= O(g(N))。

渐进下界符号Ω：如果存在正的常数 C 和自然数 N0，使得当 N >= N0 时有 f(N)>= Cg(N)，则称函数 f(N) 当 N 充分大时下有界且 g(N) 是它的一个下界，记为 T(N)= Ω(g(N))。

符号θ：若存在 3 个正常数 C1，C2，N0，且 N>=N0，有 C1g(N)<=f(N)<=C2g(N)，T(N)= θ(g(N))

o 记号：f(n)=o(g(n))表示对任意正数 c 都存在常数 n，使得对一切 n≥n0 有 0≤f(n)< cg(n)

ω记号：f(n)=ω(g(n))表示对任意正数 c 都存在常数 n0 使得对一切 n≥n0 有 0≤cg(n)<f(n)

分治法的时间复杂性分析：将规模为 n 的问题分成 k 个规模为 n/m 的子问题来求解。设分解阈值 n0= 1，且用解最小子问题的算法求解规模为 1 的问题耗费 1 个单位时间。再设将原问题分解为 k 个子问题以及将 k 个子问题的解合并为原问题的解需用 f(n)个单位时间。用 T(n)表示该分治法解规模为 n 的问题所需的计算时间。

$T(n) = 4T(n/2) + n^2$
$n^{\log_2 4} = n^2 = f(n)$
⇓
$T(n) = \Theta(n^2 \cdot \log n)$

高到低排序

2^{2^n}， $n!$， $n2^n$， $\left(\frac{3}{2}\right)^n$， $(\log n)^{\log n} = n^{\log\log n}$

n^3， $\log(n!) = \Theta(n\log n)$， $n = 2^{\log n}$

$\log^2 n$， $\log n$， $\sqrt{\log n}$， $\log\log n$， $n^{1/\log n}$

• 3种原料配置2种产品，各配多少能有最大收益

	原料1	原料2	原料3	售价
产品A	0.25	0.50	0.25	12
产品B	0.50	0.50		15
存量	120	150	50	

设产品A和B分别配制x和y
目标：$max\, z = 12x + 15y$
$s.t.$ (subject to，约束条件)
$0.25x + 0.50y \le 120$
$0.50x + 0.50y \le 150$
$0.25x \le 50$
$x \ge 0, y \ge 0$

$min(或max)\ z = \sum_{j=1}^{n} c_j x_j$ 目标函数
$s.t. \sum_{j=1}^{n} a_{ij} x_j \le (或 =, \ge) b_i, i = 1,2,\dots,m$ 约束条件
$x_j \ge 0, j \in J \subseteq \{1,2,\dots,n\}$ 非负条件
$x_j 任意, j \in \{1,2,\dots,n\} - J$ 自由变量
可行解：变量满足约束条件和非负条件的取值
可行域：全体可行解
最优解：目标函数值最优的可行解
最优值：最优解的目标函数值

• 标准型

统一最大化

$max\, z = \sum_{j=1}^{n} c_j x_j$

$s.t.$

只有"≤"号

$\sum_{j=1}^{n} a_{ij} x_j \le b_i$

$x_j \ge 0, j = 1,2,\dots n$

所有变量都有非负约束

$min\, z = 3x_1 - 2x_2 + x_3$
$s.t.\quad x_1 + 3x_2 - 3x_3 \ge 10$
$4x_1 - x_2 - 5x_3 = -30$
$x_1 \ge 0, x_2 \ge 0, x_3 任意$

统一目标函数为求

$max\, z = -3x_1 + 2x_2 - x_3$
$s.t.\quad x_1 + 3x_2 - 3x_3 \ge 10$
$4x_1 - x_2 - 5x_3 = -30$
$x_1 \ge 0, x_2 \ge 0, x_3 任意$

$max\, z = -3x_1 + 2x_2 - (x_3' - x_3'')$
$s.t.\quad x_1 + 3x_2 - 3(x_3' - x_3'') \ge 10$
$4x_1 - x_2 - 5(x_3' - x_3'') = -30$
$x_1 \ge 0, x_2 \ge 0, x_3' \ge 0, x_3'' \ge 0$

自由变量转为有非负约束

"="和"≥"处理为"≤"

$max\, z = -3x_1 + 2x_2 - (x_3' - x_3'')$
$s.t.\quad -x_1 - 3x_2 + 3(x_3' - x_3'') \le -10$
$4x_1 - x_2 - 5(x_3' - x_3'') \le -30$
$-4x_1 + x_2 + 5(x_3' - x_3'') \le 30$
$x_1 \ge 0, x_2 \ge 0, x_3' \ge 0, x_3'' \ge 0$

为了进一步方便用算法解决，将"≤"约束转为有松弛变量的"="约束
对于$\sum_{j=1}^{n} a_{ij} x_j \le b_i$，引入$s$，做等价变换：
$s = b_i - \sum_{j=1}^{n} a_{ij} x_j$，$s \ge 0$

将松弛变量也编号，即每一行"x_{n+i}"代替"s"这个符号
将标准型转为松弛型：

标准型
$max\, z = -3x_1 + 2x_2 - x_3 + x_4$
$s.t.\quad -x_1 - 3x_2 + 3x_3 - 3x_4 \le -10$
$4x_1 - x_2 - 5x_3 + 5x_4 \le -30$
$-4x_1 + x_2 + 5x_3 - 5x_4 \le 30$
$x_1, x_2, x_3, x_4 \ge 0$

松弛型
$max\, z = -3x_1 + 2x_2 - x_3 + x_4$
$s.t.\quad x_5 = -10 + x_1 + 3x_2 - 3x_3 + 3x_4$
$x_6 = -30 - 4x_1 + x_2 + 5x_3 - 5x_4$
$x_7 = 30 + 4x_1 - x_2 - 5x_3 + 5x_4$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \ge 0$

投资组合问题

10亿元投资5个项目，预测年收益率(%)分别为
A: **8.1**，B: **10.5**，C: **6.4**，D: **7.5**，E: **5.0**
基于风险的考虑，要求投资组合满足下述条件：
- 每个项目不超过3亿元. ------ • $x_1 \le 3, x_2 \le 3, x_3 \le 3, x_4 \le 3, x_5 \le 3$
- A、B投资不超过总投资的一半，即5亿元. ------ • $x_1 + x_2 \le 5$
- B不超过AB总投资的一半. ------ • $x_2 \le 0.5(x_1 + x_2) \Rightarrow x_1 - x_2 \ge 0$
- E不少于CD总投资的40%. ------ • $x_5 \ge 0.4(x_3 + x_4) \Rightarrow 0.4x_3 + 0.4x_4 - x_5 \le 0$
- 试确定投资组合中各项目的投资额，使年收益率最大. • $x_1 + x_2 + x_3 + x_4 + x_5 = 10$
- • $x_1, x_2, x_3, x_4, x_5 \ge 0$

设投资A~B对应$x_1 \sim x_5$
$\max z = 8.1x_1 + 10.5x_2 + 6.4x_3 + 7.5x_4 + 5.0x_5$
$s.t.$

$z = -3x_1 + 2x_2 - x_3 + x_4$
$x_5 = -10 + x_1 + 3x_2 - 3x_3 + 3x_4$
$x_6 = -30 - 4x_1 + x_2 + 5x_3 - 5x_4$
$x_7 = 30 + 4x_1 - x_2 - 5x_3 + 5x_4$
基本变量 B 非基本变量 N

基本解：所有非基本变量N设为0，计算得到B后，所有变量的值
$(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \overline{x_5}, \overline{x_6}, \overline{x_7}) = (0,0,0,0,-10,-30,30)$
可行解：满足所有约束，包括非负约束
基本可行解：既是可行解又是基本解，

这个解是可行解吗？
如何获得一个初始的基本可行解？