七种硬币，数量分别为 A【i】，价值已知，总容量 W；求最大价值；

```
int main(){
    int wt[]={0,3,5,2,6,11,8}; double v[]={0.1,0.25,0.5,1};
    int W,A[7];    double M[7][10005];

    for(int i=1; i<=6; i++)   cin >> A[i];
    memset(M, 0, sizeof(M));
//      状态：  coins 种类 + 容量；
    for ( int i=1; i<=6; i++)
        for ( int w=1; w<=W; w++){
            M[i][w] = 0 ;
            int n=min(A[i],w/wt[i]);//数据上限；
            for (int k=0; k<=n; k++){//状态转移；
                M[i][w]=
                max(M[i][w],k*v[i]+M[i-1][w-k*wt[i]]);
            }
        } cout <<"$"<<fixed<<setprecision(2)<<M[6][W]<<endl;
}
```

TSP， 起点 0， 终点 N-1， 计算最短路径；

```
#define inf 99999
using namespace std;

int x[25],y[25];
double dst[25][25], c[1<<19][25];
void dist(int a, int b){
    double dx=x[a]-x[b], dy=y[a]-y[b];
    dst[b][a]=dst[a][b]=sqrt(dx*dx+dy*dy);
}

int main(){
    memset(dst, 0, sizeof(dst));
    int n;  cin>>n;
//  读入各个城市坐标；
    for(int i=0; i<n; i++) cin>>x[i]>>y[i];

    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            dist(i,j);//求算距离；
    int nst=1<<(n-2); //已经遍历过的城市；
//  起：city1；屹：cityn，则只考虑之中的 n-2；
    for(int i=0; i<nst; i++){//n-2 座城，子集；
        for(int j=0; j<n; j++){
            c[i][j]=inf;//尚未启程；
        }
    }
    for(int j=0; j<n; j++)   c[0][j]=dst[0][j]; //0 城始发；
    for(int i=1; i<nst; i++){
        for(int j=1; j<n-1; j++){//n-2  cities；
            for(int k=1; k<n-1; k++){
                if((1<<(k-1))&i){//k 城已过；
//                  在已走的集合中选择一个端口连接下一站_j（含 city_n-1
                    c[i][j]=min(c[i][j], dst[j][k]+c[i-(1<<(k-1))][k]);

    double ans=inf;
    for(int i=1; i<n; i++){
//      为 city 0 找端口；
        ans=min(ans, dst[i][n-1]+c[nst-1][i]);
    }
    cout<<fixed<<setprecision(2)<<ans<<endl; }
```

N-queens：N 个皇后，所在不同的对角线，排，列；

```
using namespace std;

class Queen{          //创建一个 Queen 类
public:
    int count;
    Queen(int n){
        N = n;  //n queens;
        count = 0;
        memset(queens_N, 0, sizeof(queens_N));
    };
    bool Place(int k);
    void Backtrack(int t);  //回溯求解
```

```
private:
    int N;
    int queens_N[9];//第 n 个 queen 所在的排;
};
//寻找第 k 列（queen k，二者等）的位置，若无位置-false;
bool Queen::Place(int k){
//   检查放在 k 的皇后是否可行
    for (int j = 0; j < k; j++) {
        if (queens_N[k] == queens_N[j]
        || abs(k-j) == abs(queens_N[j]-queens_N[k]))
            return false;
    }
    return true;
}
//t 表示放入的第几个 queen;
void Queen::Backtrack(int t){
    if (t >= N) {
        count ++;
    }
    else{
        for(int i=0; i<N; i++){
            queens_N[t] = i;
            if (Place(t)) {
//            可以容纳，则下一个 queen;
              Backtrack(t+1);
            }
        }
    }
}
int main(int argc, const char * argv[]) {
    // insert code here...
    int N;
    while (cin >> N) {
        Queen queen(N);     //创建 N 黄后这个类
        queen.Backtrack(0);  //从第一个位置加上回溯
        cout << queen.count << endl;
    }
    return 0;
}
```

找子集，按顺序;

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

int n;
void pft(int s){
    for(int i=0; i<n; i++){
        if((1<<i)&s)      printf("%c",'A'+i);
    }printf("\n");
}

void subset(int entry, int s){
    pft(s);
//    先递归，后循环;
    for(int i=0; i<n; i++){
        if(entry&(1<<i)) continue;
        entry |= (1<<i);
        subset(entry, s|(1<<i));
    }
}

int main(){
    cin >> n;    subset(0, 0); }
```

找排列;

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

int main(){
    int n;  cin>>n;
    int a[n+5];
    for(int i=0; i<n; i++)    a[i] = i+1;
    do{printf("%d",a[0]);
        for(int i=1; i<n; i++){
            printf(" %d",a[i]);
        }printf("\n");
```

```
    }while(next_permutation(a, a+n));
    return 0;
}
```

能量项链，（基本 dp）

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

int a[205][205], e[205];
int dp(int i, int j){
    if(a[i][j]!=-1){
        return a[i][j];//记录;
    }else if(i==j){
        return a[i][j]=0;
    }

    int ans=0;
    for(int k=i; k<j; k++){
        ans=max(ans, dp(i,k)+dp(k+1,j)+e[i]*e[j+1]*e[k+1]);
        /*此处还是使用了递归，形式上而已;
        =>递归还是会将整个思路向精细化的方向发展，不同于递推;
        状态转移方程还是非常明晰;
        下一个状态与之前的若干个状态之间有紧密联系;
        另，
        递归的使用也并没有浪费太多的资源，
        由于合理地记录从而没有重复计算; */
    }
    return a[i][j]=ans;
}

int main(){
    int N;
    while(scanf("%d",&N)!=EOF){
        memset(a, -1, sizeof(a));
        for(int i=0; i<N; i++){
            int tmp;
            scanf("%d",&tmp);
            e[i]=e[i+N]=tmp;
        }
        int v=0;
        for(int i=0; i<N; i++){
            v=max(v, dp(i, i+N-1));
        }
        printf("%d\n",v);
    }
    return 0;
}
```

Subpath（初级 dp）

```
int main() {
    long long dp[35];
    dp[1] = dp[0] = 1;
    for (int i = 2; i <= 30; i++) {
        dp[i] = 0;
        for (int j = 0; j < i; j++)
            dp[i] += dp[j] * dp[i-1-j];
    }
    while (1) {
        int pos;  cin >> pos;
        if (pos == 0) break;
        cout << dp[pos] << endl;
    }
    return 0;
}
```

树结构

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

struct tr{
    tr *l, *r;
    int v;
    tr(){ l = r = NULL; }
};
```

```
inline void insert(tr *t, int v){
    tr *pin=t;
    if(v > pin->v){
        if(t->r == NULL){
            tr *p=new tr;
            p->v = v;
            pin->r = p;
        }else insert(pin->r, v);
    }else if(v <= pin->v){
        if(t->l == NULL){
            tr *p=new tr;
            p->v = v;
            pin->l = p;
        }else insert(pin->l, v);
    }
}

inline void pre(tr *t){
    printf(" %d",t->v);
    if(t->l != NULL)    pre(t->l);
    if(t->r != NULL)    pre(t->r);
}

inline void inn(tr *t){
    if(t->l != NULL)    inn(t->l);
    printf(" %d",t->v);
    if(t->r != NULL)    inn(t->r);
}
void pst(tr *t){
    if(t->l != NULL)    pst(t->l);
    if(t->r != NULL)    pst(t->r);
    printf(" %d",t->v);
}

int main(){
    int T;
    bool kg=false;
    while(cin >> T){
        int n; cin>>n;
        tr *t=new tr;
        t->v = n;
        for(int i=1; i<T; i++){
            cin>>n;    insert(t, n);
        }
        if(kg==true){
            printf("\n");
        }
        kg=true;
        printf("Inorder:");       inn(t);
        printf("\nPreorder:");    pre(t);
        printf("\nPostorder:");   pst(t);
        cout<<endl;
    }
    return 0;
}
```

9 进制转换;

```
int main(){
    string num;
    while(cin>>num&&num!="0"){
        int ans=0;
        for(int i=0;i<num.length();i++){
            ans*=9;   if(num[i]>'4')
            ans--;    ans+=num[i]-'0';
        }printf("%s: %d\n",num.c_str(),ans);
    }
}
```