A 保安抓小偷

根据题意，使用一个整形变量 ans 储存分钟数，再用一个二维字符数组 mp 储存整张地图。地图边长为 10，所以 mp 空间至少要开 10* 10，但是为了更好的判断越界情况，我们可以开一个 12* 12 的数组，然后把边框全部变为'* '，这样相当于将边框变为了障碍物，判断更加方便。同时题中说初始方向为正北，我们就可以将初始方向北设为 0（初始），顺时针依次将东、南、西设为 1，2，3。最后通过生成专属值的方法来判断保安追小偷是否已经陷入了一个循环，也就是无法相遇，专属值=保安的 x 坐标+保安的 y 坐标* 10+小偷的 x 坐标* 100+小偷的 y 坐标* 1000+保安的方向* 10000+小偷的方向* 40000（方向最多为 4）

```
#include<bits/stdc++.h>
using namespace std;
char mp[12][12];//地图
//保安位置(sx,xy)及朝向 sd，小偷位置(tx,ty)即朝向 ty，答案 ans，专属值 tdz
int sx,sy,sd,tx,ty,td,ans,tdz;
bool zt[160005];//记录专属值是否出现
void move(int x,int y,int d,int h){//移动函数,d 表示方向，h 代表保安还是小偷
    if (d==0){
        if (mp[x-1][y]=='*') if (h==0) sd=1; else td=1;
        else if (h==0) sx--; else tx--;
    }else if (d==1){
        if (mp[x][y+1]=='*') if (h==0) sd=2; else td=2;
        else if (h==0) sy++; else ty++;
    }else if (d==2){
        if (mp[x+1][y]=='*') if (h==0) sd=3; else td=3;
        else if (h==0) sx++; else tx++;
    }else{
        if (mp[x][y-1]=='*') if (h==0) sd=0; else td=0;
        else if (h==0) sy--; else ty--;
    }
}
bool pd(){ //判断循环终止条件：如果保安坐标与小偷坐标相等，则他们重叠，返回 0，退出循环
    if (sx==tx&&sy==ty) return 0;
    else return 1;
}
int main(){
    //由于边界也是障碍，所以直接将数组的 0 和 11 下标的元素设为障碍
    for (int i=0;i<=11;i++) mp[i][0]='*',mp[i][11]='*';
    for (int i=1;i<=11;i++) mp[0][i]='*',mp[11][i]='*';
    for (int i=1;i<=10;i++){
    for (int j=1;j<=10;j++){
        cin>>mp[i][j];
        if (mp[i][j]=='S') sx=i,sy=j;
        if (mp[i][j]=='T') tx=i,ty=j;
```

```
            }
        }
        while (pd()){//模拟每分钟
            tdz=sx+sy*10+tx*100+ty*1000+sd*10000+td*40000;
            if (zt[tdz]){//进入同一个坐标并且方向相同就输出 0 并结束程序
                cout<<0<<endl;
                return 0;
            }
            zt[tdz]=1;//标记
            move(sx,sy,sd,0);
            move(tx,ty,td,1);//依次移动农夫和奶牛
            ans++;//分钟数加一
        }
        cout<<ans<<endl;//输出
        return 0;
}
```

B PIPI 逛超市

根据题意，要求购买所有物品的最少花费。把所有的物品都看作节点，购买所有物品所花费的最小值就应该是把所有点连接起来的最小代价也就是该图的最小生成树。如果两个物品之间存在关系，那么可以有优惠，就把两个点连起来，增加一条边，边权值为优惠代价。在建好图之后，直接用 kruskal 算法求解。

```
#include<bits/stdc++.h>
using namespace std;
struct node
{
    int u,v,w;//买完 u 再买 v，v 只要 u 元
}e[250000];
//tot 记录最小生成树的边数,k 记录图的边数+1
int a,b,k=0,tot=0,ans,fa[505];
bool cmp(node x,node y)
{
    return x.w<y.w;
}
int find(int x)
{
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
void kruskal()
{
    int j=0;
    while(j<k&&tot<b){
```

```
            if(find(e[j].u)!=find(e[j].v)){
                tot++;
                ans+=e[j].w;
                int xx=find(e[j].u);
                 int yy=find(e[j].v);
                 if(xx!=yy) fa[xx]=yy;
            }
            j++;
        }
}
int main()
{
    scanf("%d%d",&a,&b);
    for(int i=1;i<=b;i++){
        for(int j=1;j<=b;j++){
            int x;
            scanf("%d",&x);
            if(i<j&&x!=0) {//对称矩阵，千万记得没有优惠是 0，不建边
                e[k].u=i;
                e[k].v=j;
                e[k].w=x;
                k++;
            }
        }
    }
    //从 0 向各点连边权为 a 的边
    for(int i=1;i<=b;i++) {
        e[k].u=0;
        e[k].v=i;
        e[k].w=a;
        k++;
    };
    for(int i=0;i<=b;i++) fa[i]=i;
    sort(e,e+k,cmp);
    kruskal();
    printf("%d\n",ans);
    return 0;
}
```

C 回文子串数量

根据题意，要求出字符串中子串是回文串的数量。如果直接枚举所有子串，需要 O(n^2)枚举区间，然后用 O(n)复杂度遍历子串来判断是否回文，一共是 O(n^3)复杂度。根据数据规

模 1000 不难判断一定会超时，因此要使用更优的方法。注意到回文串的特性，两端的字符相等，去掉两个端点后依然是回文串，例如回文串 abba 去掉两端后 bb 依然是回文串。因此可以考虑使用动态规划来求解。定义 dp[i][j]为从 i 到 j 的子串是否为回文串，则转移方程为：

$$dp[i][j] = dp[i+1][j-1] \&\& s[i]==s[j];$$

按区间从小到大枚举，更新 dp 数组即可，注意边界情况的处理。

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1005;
char s[N];
bool dp[N][N];
int main()
{
    for(int i=1;i<N;++i){
        for(int j=0;j<i;++j){
            dp[i][j] = 1;//对边界进行特殊处理
        }
    }
    while(~scanf("%s",s+1)){//从 1 开始存储更加方便
        int n = strlen(s+1);//求出字符串长度
        int ans = 0;//记录答案
        for(int len=1;len<=n;++len){//从小到大枚举区间
            for(int i=1;i+len<=n+1;++i){//枚举左端点
                int j = i+len-1;//右端点
                dp[i][j] = dp[i+1][j-1] && s[i]==s[j];//状态转移
                if(dp[i][j])ans++;
            }
        }
        printf("%d\n",ans);
    }
}
```

D 收银员 PIPI

根据题意，要用最少的钱数来凑齐目标的 t 元。对于这类求最小数量的题，可以考虑使用 BFS 求解，关键就是怎样将题目转化成图的模型。每一步选择一种纸币，总金额等于目标值时就相当于到达了终点，该问题就可以转化为求最短路径长度的问题。对于相同数值的金额，没有必要重复访问，因此可以使用一个数组或者哈希表来标记访问过的金额，避免重复访问。由于遍历的对象包含金额和纸币数两个属性，因此可以用结构体封装一下。

```
#include<bits/stdc++.h>
using namespace std;
struct node{
    int x,c;//金额，数量
}p;
```

```
int main()
{
    int t,n;
    while(~scanf("%d%d",&t,&n)){
        vector<int>a(n);
        for(int i=0;i<n;++i)scanf("%d",&a[i]);
        unordered_map<int,bool>vis;//用哈希表当标记数组
        queue<node>q;//队列
        q.push({0,0});
        while(!q.empty()){//BFS
            p = q.front();q.pop();
            if(p.x==t)break;
            for(int &x:a){//遍历 a 数组，x 是 a 中的元素
                int y = p.x+x;
                if(y<=t&&!vis[y]){
                    q.push({y,p.c+1});
                    vis[y] = 1;//入队的时候标记已访问的金额
                }
            }
        }
        if(p.x==t)printf("%d\n",p.c);
        else puts("-1");
    }
}
```

E 分饼干

根据题意，要求使得所有小朋友手上饼干数相同所移动的最少饼干数，显然每个小朋友最终持有的饼干数等于总数除以人数。我们规定一个方向，每个人给左手边的人 xi 块饼干（xi 可以小于 0）。设每个人初始有 a_i 块饼干，对于第 1 个人，他给了第 n 个人 x1 块饼干，并且第 2 个人给了他 x_2 块饼干，因此他还剩 $a_1-x_1+x_2$，设平均值是 m，则 $a_1-x_1+x_2=m$。
同理，对于第 2 个人，有 a2-x2+x3=m...对于第 n 个人有 $a_n-x_n+x_1=m$。这 n 个方程实际上秩只有 n-1，不能接触所有的变量，但是可以把 x_i 都用 x1 来表示。对于第 1 个方程 $x_2=m-a_1+x_1=x1-C_1(C_1=a_1-m)$，对于第 2 个方程 $x_3=m-a_2+x_2=m-a_2+x_1-C_1=x_1-C_2$。同理可以推出所有的 xi 的表达式。那么传递数量就等于$|x_1-0|+|x_1-C_1|+|x_1-C_2|+...+|x_1-C_{n-1}|$，这个式子的最小值求法其实在高中学过，利用$|x_1-C_i|$的几何意义为数轴上 x1 到 Ci 的距离可以得出 x1 应该是这些点的中位数，记 C0=0，则求出 Ci 的中位数，计算答案即可，计算中位数可以使用复杂度为 O(n)的快速选择算法，c++中有内置的函数，至此问题就被解决了。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int main(){
```

```
    int i,n;
    while(~scanf("%d",&n)){
        vector<ll>a(n);
        for(i=0;i<n;++i)scanf("%lld",&a[i]);
        ll m = accumulate(a.begin(),a.end(),0)/n;//求出平均值
        vector<ll>b(1,0);
        for(i=0;i<n-1;++i)b.push_back(b.back()+a[i]-m);//计算常数 C
        nth_element(b.begin(),b.begin()+n/2,b.end());//快速选择算法求中位数
        ll sum = 0, x = b[n/2];
        for(i=0;i<n;++i)sum += abs(b[i]-x);//计算传递总数
        printf("%lld\n",sum);
    }
    return 0;
}
```