Dire Wolf

题目大意：你是一个战士现在面对，一群狼，每只狼都有一定的主动攻击力和附带攻击力。你杀死一只狼。你会受到这只狼的（主动攻击力+旁边两只狼的附带攻击力）这么多伤害~现在问你如何选择杀狼的顺序使的杀完所有狼时，自己受到的伤害最小。（提醒，狼杀死后就消失，身边原本相隔的两只狼会变成相邻，而且不需要考虑狼围城环这种情况）

```
#include<iostream>
#include<cstring>
using namespace std;
const int size=1000;
int t,a[size],n,dp[size][size],b[size];//dp[i][j]表示区间i，j内的狼都杀死的最小伤害
int main()
{
    cin>>t;
    int h=1;
    while(t--)
    {

        cin>>n;
        for(int i=1;i<=n;i++) cin>>a[i];
        for(int i=1;i<=n;i++) cin>>b[i];
        for(int i=1;i<=n;i++)//初始化
        {
            for(int j=i;j<=n;j++)
            {
                dp[i][j]=0x3f3f3f3f;
            }
        }
        b[0]=b[n+1]=0;
        for(int len=1;len<=n;len++)//长度
        {
            for(int l=1;l<=n-len+1;l++)//左边
            {
                int r=len+l-1;//右边
                for(int k=l;k<=r;k++)//决策
                {
                    dp[l][r]=min(dp[l][r],dp[l][k-1]+a[k]+dp[k+1][r]+b[l-1]+b[r+1]);
                }
            }
        }
        cout<<"Case #"<<h<<": "<<dp[1][n]<<endl;
        h++;
    }
    return 0;
}
```

Brackets

题意：求最大括号匹配

```
#include<iostream>
#include<string>
#include<cstring>
using namespace std;
const int N=500;
string a;

int dp[N][N];
int main(){
    while(cin>>a){

        if(a[0]=='e') break;
        memset(dp,0,sizeof(dp));
        for(int l=1;l<=a.size();l++){
            for(int i=0;i+l-1<a.size();i++){
                int j=i+l-1;
                if((a[i]=='('&&a[j]==')')||(a[i]=='['&&a[j]==']'))
                    dp[i][j]=dp[i+1][j-1]+2;
                for(int k=i;k<j;k++)
                    dp[i][j]=max(dp[i][j],dp[i][k]+dp[k+1][j]);
            }
        }
        cout<<dp[0][a.size()-1]<<endl;
    }
    return 0;
}
```

Food Delivery

题目大意： 一个人他外卖员要去送外卖，他的店在X位置，他的速度为1/V(相当于一米v分钟)，然后现在有N个人要外卖，这N个 人的坐标分别为Xi，第 i 个人每等一分钟不满意度会增加Bi，他希望送完所有人让总的不满意度最少，求最少的不满意度是多少

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <algorithm>
using namespace std;
const int INF=0x3f3f3f3f;
struct node{
    int x,val;
    friend bool operator<(node a,node b){
        return a.x<b.x;
    }
}s[1005];
int sum[1005],dp[1005][1005][2];
int main(){
```

```
    int N,V,X,i,j,k,l;
    while(scanf("%d%d%d",&N,&V,&X)!=EOF){
        for(i=1;i<=N;i++)                          //dp[i][j][0]表示走完区间[i,j]停在左端点
        scanf("%d%d",&s[i].x,&s[i].val);           //dp[i][j][0]表示走完区间[i,j]停在右端点
        s[N+1].x=X,s[N+1].val=0;
        N++;
        sort(s+1,s+N+1);                           //一定是先走相邻的点费用最小,因此先排序
        memset(dp,INF,sizeof(dp));
        sum[0]=0;
        for(i=1;i<=N;i++){
        sum[i]=sum[i-1]+s[i].val;
        if(s[i].x==X&&s[i].val==0)
        dp[i][i][0]=dp[i][i][1]=0;
        }                                          //将起点初始化
        for(l=2;l<=N;l++){                         //遍历四种可能
            for(i=1;i<=N-l+1;i++){
                j=i+l-1;
                dp[i][j][0]=min(dp[i][j][0],dp[i+1][j][0]+(s[i+1].x-s[i].x)*(sum[i]+sum[N]-
sum[j]));
                dp[i][j][0]=min(dp[i][j][0],dp[i+1][j][1]+(s[j].x-s[i].x)*(sum[i]+sum[N]-
sum[j]));
                dp[i][j][1]=min(dp[i][j][1],dp[i][j-1][0]+(s[j].x-s[i].x)*(sum[i-1]+sum[N]-
sum[j-1]));
                dp[i][j][1]=min(dp[i][j][1],dp[i][j-1][1]+(s[j].x-s[j-1].x)*(sum[i-
1]+sum[N]-sum[j-1]));
            }
        }
        printf("%d\n",min(dp[1][N][0],dp[1][N][1])*V);
    }
    return 0;
}
```

*Sequence Swapping （我也觉得有点难啊）

大致题意：给你一些括号，有左括号有右括号，每一个括号对应一个数值vi。当左右括号i、j相邻并且左括号在左、右括号在右，你可以选择交换这两个括号的位置，并且产生一个vivj*的权值。交换次数不限，现在问你能够产生的最大权值和是多少。 首先，对于左括号来说，如果往右移了一位，即与某一个右括号交换了，那么就一定不会交换回来。这是一个很明显的无后效性，因此考虑dp。但是有另外一个问题，每一次的交换会对括号的序列发生改变，直接dp可能又会产生后效性。所以得从最后结果来考虑。 容易知道，由于交换一定是要左右括号配上之后才能够交换，所以左后的结果相当于是，所有左括号内部的相对位置不变，所有右括号内部的相对位置也不变。于是，我就可以不考虑中间的顺序，我直接考虑每个最后左括号相对所有右括号的位置。如果初始时某个左括号右边有i个右括号，最后又j个右括号，其中j<=i，那么产生的代价就是vi*(s[i]-s[j])，其中s表示右括号权值的后缀和。 如此一来，我们令dp[i][j]表示第i个左括号，最终的在第j个括号右边的最大权值和。于是可以得到转移方程dp[i][j]=max(dp[i-1][k]+s[pos[i]]-s[j+1])，其中pos[i]表示左括号i的初始位置，s表示右括号的后缀和。可以看到这个时间复杂度是$O(N^3)$的，但是显然，由于增加值s[pos[i]]-s[j+1]是固定的，每次转移只需要找最大的dp[i-1][k]即可，而这个最大值也是可以简单的维护的。总的复杂度可以做到$O(N^2)$。具体见代码：

```
#include<bits/stdc++.h>
```

```
#define INF 0x3f3f3f3f
#define LL long long
#define N 1010
using namespace std;

LL sum[N],v[N],w[N][N],dp[N][N];
//sum表示右括号的前缀和（从右往左），w[i][j]表示（从右往左）至少
//w[i][j]表示第i个‘（’至少放到第j个‘）’的最大值
//dp[i][j]表示第i个‘（’放到第j个‘）’的最大值
char s[N];

int main()
{

    int T; cin>>T;
    while(T--)
    {
        int m=0,n;
        cin>>n>>s+1;
        LL tot=0,t=0,ans=0;
        memset(sum,0,sizeof(sum));
        for(int i=1;i<=n;i++) cin>>v[i];
        for(int i=1;i<=n;i++)
            if (s[i]==')') sum[++tot]=v[i];
        for(int i=tot-1;i>=1;i--) sum[i]+=sum[i+1];//前缀和（右到左的‘）’括号）
        for(int i=0;i<=n;i++)
            for(int j=0;j<=tot+1;j++)
                dp[i][j]=w[i][j]=-INF;//初始化
        memset(w[0],0,sizeof(w[0]));
        for(int i=n;i>=1;i--)
        {
            if (s[i]==')') {t++;continue;}//t是目前出现‘）’括号个数，m是‘（’括号个数
            m++;
            for(int j=tot;j>=tot-t;j--)
            {
                dp[m][j]=w[m-1][j]+(sum[tot-t+1]-sum[j+1])*v[i];
                //第m个‘（’放到第‘j’个‘）’，需要他右边的‘（’都至少放到j之后最大值，再加上这一段‘）’的前缀和*自己的值

                ans=max(dp[m][j],ans);
            }
            for(int j=tot;j>=0;j--) w[m][j]=max(w[m][j+1],dp[m][j]);//更新
        }
        cout<<ans<<endl;
    }
}
```

B-number

题意：统计区间 [1,n] 中含有 '13' 且模 13 为 0 的数字有多少个。

```
#include<iostream>
#include<cstring>
using namespace std;
typedef long long ll;
int dp[25][25][5],a[25];
int dfs(int pos,int mod,int have,int limit)//位置，前面取余的数，前面的状态，是否上界
{
    if(pos==0)
        return mod==0&&have==2;
    if(!limit&&dp[pos][mod][have]!=-1)
        return dp[pos][mod][have];
    int up=limit?a[pos]:9;
    int ans=0;
    int _mod,_have;
    for(int i=0;i<=up;++i)
    {
        _mod=(mod*10+i)%13;
        _have=have;
        if(have==0&&i==1)
            _have=1;
        if(have==1&&i!=1)
            _have=0;
        if(have==1&&i==3)
            _have=2;
        ans+=dfs(pos-1,_mod,_have,limit&&i==up);
    }
    if(!limit)
        dp[pos][mod][have]=ans;
    return ans;
}
int solve(int num)
{
    memset(dp,-1,sizeof(dp));
    int top=0;
    while(num)
    {
        a[++top]=num%10;
        num/=10;
    }
    return dfs(top,0,0,1);
}

int main()
{
    int n;
    while(scanf("%d",&n)!=EOF)
    {
        solve(n);
        printf("%d\n",solve(n));
    }
    return 0;
}
```

Victor and World(floyd+状压dp)

题目大意：有n个城市，在n个城市之间有m条双向路，每条路有一个距离，现在问从1号城市去游览其它的2到n号城市最后回到1号城市的最短路径（保证1可以直接或间接到达2到n）。（n<=16）

```
#include <iostream>
#include <string.h>
#include <math.h>
#include <queue>
#include <algorithm>
#include <stdlib.h>
#include <map>
#include <set>
#include <stdio.h>
#include <time.h>
using namespace std;

const int mod=1e9+7;
const int INF=0x3f3f3f3f;
const double eqs=1e-9 ;
int Map[20][20] ;
int dp[200000][20] , dis[20] ;
int main() {
    int t,n,m;
    int w,l,r,min1;
    scanf("%d", &t) ;
    while(t--){
        scanf("%d%d",&n,&m) ;
        memset(Map,INF,sizeof(Map)) ;
        while(m--) {
            scanf("%d%d%d",&l,&r,&w) ;
            l--;r--;
            if(Map[l][r]>w)
            Map[l][r]=Map[r][l]=w;
        }
        for(int i=0;i<n;i++) Map[i][i] = 0 ;
        for(int k=0;k<n;k++) {
            for(int i=0;i<n;i++) {
                for(int j=0;j<n;j++) {
                    Map[i][j]=min(Map[i][j],Map[i][k]+Map[k][j]) ;
                }
            }
        }
        /*-----------------以上跑floyd，求点与点最短路---------------------------------------
----*/

        memset(dp,INF,sizeof(dp)) ;
        memset(dis,INF,sizeof(dis)) ;
        dp[1][0] = 0 ;//dp[i][j]表示在i状态最后（此时）在j点。
        dis[0] = 0 ;
```

```
        m = 1<<n ;//n位二进制，记录哪些点跑了为1，哪些点没跑为0。

        for(int i = 1 ; i < m ; i++) {//i是此时的二进制状态
            for(int j = 0 ; j < n ; j++) {//j是此时的点
                if( dp[i][j] == INF ) continue ;//该状态下在该点 不存在，没访问过
                for(int k = 0 ; k < n ; k++) {//k是想要到达的下一个点
                    if( (i&(1<<k)) || Map[j][k] == INF ) continue ;//如果k存在于i状态，表示已经到过k点||无法到达
                    if( dp[ i|(1<<k) ][k] > dp[i][j]+Map[j][k] ) {
                        dp[ i|(1<<k) ][k] = dp[i][j]+Map[j][k] ;//dp[i状态加上k点][最后处于k点]>dp[i状态][最后处于j点]+j到k最短路;
                        dis[k] = min(dis[k],dp[ i|(1<<k) ][k]) ;//到k点最短距离
                    }
                }
            }
        }
        min1 = INF ;
        for(int i = 0 ; i < n ; i++) {
            min1 = min(min1,dp[m-1][i]+dis[i]) ;
        }
        printf("%d\n", min1) ;
    }
    return 0 ;
}
```

A Simple Task

题意：给定一个简单图，求图中有多少环?

第一行两个整数n,m，表示简单图的点的个数和边的个数。(1<=n<=19, 0<=m) 接下来m行，每行两个整数x,y，表示x点，y点间有一条无向边。(1<=x,y<=n,a!=b)。

```
#include<bits/stdc++.h>
using namespace std;
#define clr(a, x) memset(a, x, sizeof(a))
#define mp(x, y) make_pair(x, y)
#define pb(x) push_back(x)
#define X first
#define Y second
#define fastin                      \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);
#define legal(a,b) a&b

typedef long long LL;
typedef pair<int, int> PII;
typedef vector<int> VI;
const int INF=0x3f3f3f3f;
int G[21][21];
```

```
LL dp[21][1<<20];//dp[i][state] 以第一个1出现的位置作为开头，以i结尾的路径的数量
int num[1<<21];//num[i]表示i的二进制里有几个1
int n,m;
//dp[i][s|(1<<i)] += dp[e][s]
int solve(int state)
{
    int ans=0;
    for(int i=0; i<=20; i++)
        if(state &(1<<i)) ans++;
    return ans;
}
int main()
{
    for(int i=0; i<(1<<21); i++)   num[i] = solve(i);

    while(cin>>n>>m)
    {
        clr(G,0);
        for(int i=0,x,y; i<m; i++)
        {
            cin>>x>>y;
            x--;  y--;
            G[x][y]=G[y][x]=1;
        }
        clr(dp,0);
        //初始化
        for(int i=0; i<n; i++)
            dp[i][1<<i] = 1;
        int total=1<<n;
        LL ans=0;
        for(int s=1; s<total; s++)
        {
            //找开头
            int st=0;
            for(int i=0; i<n; i++)
                if(s&(1<<i)) st=i,i=n;//st为第一个1位置
            for(int i=st; i<n; i++) //枚举状态中的结尾
                if(s&(1<<i)&&(dp[i][s]))
                        for(int j=st; j<n; j++) //枚举新状态的结尾
                        {0
                            if(s&(1<<j)) continue;//j已经存在于s中
                            if(G[i][j]==0) continue;
                            dp[j][s|(1<<j)] += dp[i][s];//有s状态，i连j
                            if(G[st][j] && num[s|(1<<j)]>=3)//如果st和j相连，且该新状态点数大于3
                                ans+=dp[i][s];
                        }

        }
        cout<<ans/2<<endl;
    }
    return 0;
}
```

大家闲暇时间可自学dp优化——四边形不等式，斜率优化，单调队列优化，数据结构优化等

可以自行结合书本习题学习