目录

今年训练联盟的第一场周赛，CTU Open Contest 2019。CTU，是Czech Technical University，从1997年开始，在ICPC Central Europe Contest之前，在布拉格的Czech Technical University都要组织CTU Open Contest。CTU Open Contest试题的质量比较高，每届比赛，都公布数据、标程、解析。比赛网址：https://www.jisuanke.com/contest/7321主持比赛的学校：烟台大学

Beer Barrels

题目大意

给出四个整数：A，B，K，C，A,B,C 都是大于 0 的个位数，问在所有仅由 A 或 B 组成的 K 位数中(K 位数的每一位都是 A 或 B)，数字 C 的个数有多少。

问题分析一

分析样例：1 2 3 2
根据题目我们推出样例中所有的 3 位数为：

111	共 0 个 2
211 121 112	共 3 个 2
221 212 122	共 6 个 2
222	共 3 个 2

所以本题答案为 0+3+6+3=12。
考虑除 111 的其他数字，第二行的数中每个数都包含一个二，也就是说，第二行的数都是从 3 位数的其中一位将其设为 2，很明显可以联想到组合数，就是从三个位置选一个位置为 2 的所有情况，这种数的个数 C(3，1)，然后每个数都有一个 2，所以答案为 C(3，1)*1，所以第三行为 C(3，2)*2，第三行为 C(3，3)*3。

所以可以推出，最终答案为ΣC(K，i)*i，i∈[1，K]。

求组合数的公式为：

$$C_n^m = \frac{P_n^m}{P_m} = \frac{n!}{m!\,(n-m)!}, C_n^0 = 1.$$

但是此题要求答案要对 1e9+7 取模，每个阶乘部分都是被模过，所以分母部分要用快速幂求逆元。

参考代码

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using namespace std;
typedef long long ll;
const ll mod=1e9+7;
```

```
ll qpow(ll a,ll b,ll mod)//快速幂
{
	ll ans=1%mod;
	while(b)
	{
		if(b&1) ans=(ans%mod*a%mod)%mod;
		a=(a%mod*a%mod)%mod;
		b>>=1;
	}
	return ans;
}
int main()
{
	ll f[1007];
	f[0]=f[1]=1;
	for(int i=2;i<=1000;i++)f[i]=(f[i-1]*i)%mod;//打表获得阶乘
	ll a,b,k,c;
	ll tmp,ans=0;
	cin>>a>>b>>k>>c;
	if(a!=c&&b!=c)cout<<0<<endl;
	else if(a==b)cout<<k<<endl;
	else
	{
		for(int i=1;i<=k;i++)
		{

tmp=(i%mod*f[k]%mod*qpow(f[k-i],mod-2,mod)%mod*qpow(f[i],mod-2,mod)%mod)%mod;//i*组
合数
			ans=(ans%mod+tmp%mod)%mod;
		}
		cout<<ans<<endl;
	}
	return 0;
}
```

问题分析二

生成 K 位数的每一位上只能是 A 或 B，这一特点与 2 进制的 1 和 0 类似。因此可以仿照 2 进制数码表列出 A 与 B 组成的所有数，然后找规律。

K 位 2 进制数表有 K 列，2^K 行。表中每一列 1 和 0 出现的次数是相等的，都是 2^(K-1)。因此 1 和 0 出现的次数为列数*每列上出现的次数，即 K*2^(K-1)。当 A=B 时，只有 1 个数码，各位都相同，此时生成的数唯一。注意到 C 的取值是任意的，当 C≠A 且 C≠B 时，C 在 A、B 组成的数中出现次数为 0。当 C=A 或 C=B 时，出现的次数与 K 有关，当 K=0 时，0 位没有组成任何数，因此 C 的出现次数是 0；如果 K=1 只有 1 位，C 的出现次数为 1。K＞1 时 C 的出现次数为 K*2^(K-1)。因为 K 的取值范围是[0,1000]。当 K=1000 时，2^1000 计算需要快速幂运算，最后根据题意数据取模即可。

总结规律：

0 ,K=0 || (C≠A && C≠B)

1, K=1
K*2^(K-1)，A≠B
K, A=B
样例分析：

代码略。

Beer Bill

题目大意

计算字符串的价格。给多个字符串，每个串占一行。字符串分两种，一种字符串名为 Raked Line 只含有 C 个'|'字符,这种字符串的价格定义为 42*C。另一种字符串名为 Priced Line，格式是以数字 price 开头、中间用两个字符“，-“连接，结尾是连续 C 个‘|’。,这种字符串的价格定义为 price*C，若结尾没有‘|’出现，则 C 默认为 1 个。计算所有字符串的总价，总价向上取整到 10 的倍数。

问题分析

两种字符串的价格很容易计算，可以直接统计‘|’个数再乘 42。第二种 Priced Line 可以先将开头的 price 字符串转整数，然后记录‘|’的个数，二者相乘得这一字符串的价格。注意到当‘|’不出现时需要特殊处理一下。

思路：ch 逐个读取字符，根据字符串的开头判断是哪一种字符串并计算 price 的值。在每行的行尾进行一次累加。最后向上取整输出各行的累加结果。

参考代码

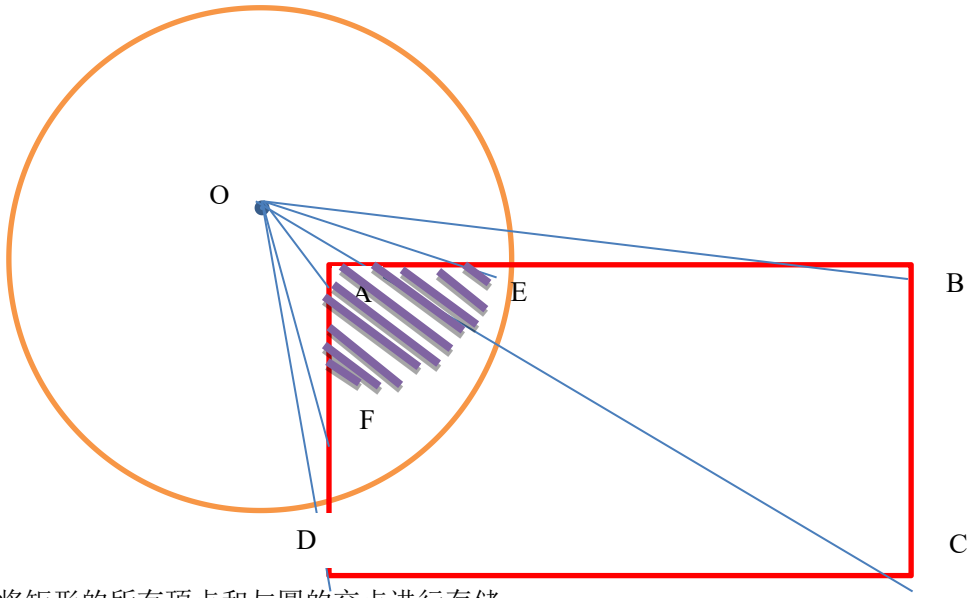
```
#include<stdio.h>
#include<ctype.h>

int main()
{
    int price=42,c=0,res=0;//默认价格为 42
    char ch,flag=1;//flag=1 代表这一行是 raked line
    while(~scanf("%c",&ch))//逐个读取每个字符
    {
        if(ch=='\n')//换行时把这一行的总价累加起来
        {
            if(c==0 && price!=42)//特判 priced line 中'|'个数为 0 的情况
                res = price; //'|'个数为 0 则价格为 price
            else
                res+=price*c;//'|'个数不为 0,直接累加
            c=0; //恢复默认价格
            price=42;
            flag=1;//重置 raked line  标记
            continue;
        }

        if(price==42 && ch=='|')//如果是 raked line 则进行 raked line 计数
            c++;
        else //不是 raked line ,进行 priced line 计数
        {
            if(flag==1)flag=0,price=0;//设置 flag 标记为 priced line 标记

            if(isdigit(ch))//字符串转整数
                price = price * 10 +    ch - '0';
            else if(ch=='|')//计数
                c++;
        }
    }
    if(res%10)res+=10-res%10; //向上取整到 10 的倍数
    printf("%d,-\n",res);
    return 0;
}
```

Beer Coaster

题目大意

给定一个圆的圆心和一个矩形的左下方坐标和右上方坐标，除圆的半径之外的 6 个数据均在-1000 到 1000 范围内，圆的半径在 1 到 1000 范围。求圆与矩形相交的面积

问题分析



如图所示，将矩形的所有顶点和与圆的交点进行存储。

此时重叠部分面积 **S=-S(△OAE)- S(△OBE)+ S(扇形 OBC)+S(扇形 OCD)-S(扇形 ODF)-S(△OFA);**

从 A 起点开始围绕矩形将每一段与 o 相连 形成的三角形和扇形的面积相加减形成了重叠部分面积。

参考代码

```
#include<iostream>
#include<string.h>
#include<stdio.h>
#include<cmath>
#include<algorithm>
#include<vector>
```

```
#include<queue>
#include<set>
#include<stack>
using namespace std;
#define LD long double
#define Zero (1e-10)
const int inf=100010;
struct PT{ LD x,y;}pt[20];   //point 存储点坐标，pt 数组存储矩形顶点坐标和其与圆形相交点坐标
struct CI{ PT p; LD r;}c;     //circle 存储圆的圆心 p,半径 r
struct LN{ PT s,e;}ln[4];     //存储矩形的四条边 s:star 起点，e:end 终点
bool inside[20];                    //判断 pt 数组的点是否在圆外 是：1 否：0
int cnt=0;                                  //记录 pt 数组的长度
void getPoint(LN a,int f){    //获取圆形和矩形的交点 存入 pt
    PT tmp;
    LD t;
    switch(f){
            case 0:                                                                 //上边
                    if(fabs(c.p.y-a.s.y)>c.r) return;
                    t=sqrt(fabs(c.r*c.r-pow(c.p.y-a.s.y,2)));
                    tmp={c.p.x-t,a.s.y};
                    if(a.s.x<tmp.x&&tmp.x<a.e.x)
                    pt[cnt++]=tmp;
                    tmp.x=c.p.x+t;
                    if(a.s.x<tmp.x&&tmp.x<a.e.x)
                    pt[cnt++]=tmp;
                    break;
            case 1:                                                         //右边
                    if(fabs(c.p.x-a.s.x)>c.r) return;
                    t=sqrt(fabs(c.r*c.r-pow(c.p.x-a.s.x,2)));
                    tmp={a.s.x,c.p.y-t};
                    if(a.e.y<tmp.y&&tmp.y<a.s.y)
                    pt[cnt++]=tmp;
                    tmp.y=c.p.y+t;
                    if(a.e.y<tmp.y&&tmp.y<a.s.y)
                    pt[cnt++]=tmp;
                    break;
            case 2:                                                 //下边
                    if(fabs(c.p.y-a.s.y)>c.r) return;
                    t=sqrt(fabs(c.r*c.r-pow(c.p.y-a.s.y,2)));
                    tmp={c.p.x-t,a.s.y};
                    if(a.e.x<tmp.x&&tmp.x<a.s.x)
                    pt[cnt++]=tmp;
                    tmp.x=c.p.x+t;
                    if(a.e.x<tmp.x&&tmp.x<a.s.x)
                    pt[cnt++]=tmp;
                    break;
            case 3:                                                 //左边
                    if(fabs(c.p.x-a.s.x)>c.r) return;
                    t=sqrt(fabs(c.r*c.r-pow(c.p.x-a.s.x,2)));
                    tmp={a.s.x,c.p.y-t};
                    if(a.s.y<tmp.y&&tmp.y<a.e.y)
```

```
                    pt[cnt++]=tmp;
                    tmp.y=c.p.y+t;
                    if(a.s.y<tmp.y&&tmp.y<a.e.y)
                    pt[cnt++]=tmp;
                    break;
    }
}
void initPoint(LD x1,LD y1,LD x2,LD y2){          //初始化矩形四个顶端存入 ln，获取所有
的交点
    ln[0]={{x1,y2},{x2,y2}};
    ln[1]={{x2,y2},{x2,y1}};
      ln[2]={{x2,y1},{x1,y1}};
    ln[3]={{x1,y1},{x1,y2}};
    for(int i=0;i<4;i++){
            pt[cnt++]=ln[i].s;
            getPoint(ln[i],i);
    }
}
LD Xp(PT a,PT b){return (a.x-c.p.x)*(b.y-c.p.y)-(a.y-c.p.y)*(b.x-c.p.x);} //向量的叉积
LD Dp(PT a,PT b){return (a.x-c.p.x)*(b.x-c.p.x)+(a.y-c.p.y)*(b.y-c.p.y);} //向量的点积
LD dis(PT a,PT b){return sqrt(pow(a.x-b.x,2)+pow(a.y-b.y,2));}                    //  求
a,b  两点的距离
LD dis2(PT a,PT b){return pow(a.x-b.x,2)+pow(a.y-b.y,2);}                    //a,b   两
点距离的平方
void getInside(){
                            //获取 inside 数组
    for(int i=0;i<cnt;i++){
            inside[i]=dis(pt[i],c.p)>(c.r+Zero);
    }
}
LD tri(PT a,PT b){return Xp(a,b)/2;}                    //获取三角形部
分面积,当圆心为(0,0)时  s=(x1*y2-x2*y1)/2;

LD sec(PT a,PT b){
    //获取扇形部分面积，s=(a*r^2)/2 a 为弧度制数值
    LD da=dis2(a,c.p);
    LD db=dis2(b,c.p);
    LD cj=Xp(a,b);
    LD dj=Dp(a,b);
    LD D=cj/sqrt(da*db);
    LD A=asin(D);
    if(dj<Zero) A>0?M_PI-A:-M_PI-A;
    return A*c.r*c.r/2;
}
LD getRes(){                                    //获取结果
    getInside();
    LD res=0;
    for(int i=0;i<cnt;i++){
            if(!inside[i]&&!inside[(i+1)%cnt]){
                    res+=tri(pt[i],pt[(i+1)%cnt]);
            }else{
                    res+=sec(pt[i],pt[(i+1)%cnt]);
```

```
                }
        }
        return fabs(res);
}
int main()
{
        LD x1,y1,x2,y2;
        scanf("%llf %llf %llf %llf %llf %llf %llf",&c.p.x,&c.p.y,&c.r,&x1,&y1,&x2,&y2);
        if(c.r<Zero){
                printf("%.12lf",0);
                return 0;
        }
        if(x1>x2)swap(x1,x2);
        if(y1>y2)swap(y1,y2);
        initPoint(x1,y1,x2,y2);
        printf("%.12lf",(double)getRes());
}
```

Beer Flood System

题目大意

给定一个图，要求在保证从源点可以到达任意点，从任意点可以到达汇点的情况下，最多能删掉几条多余的边。

其中，源点是图中入度为零的点，汇点是图中出度为零的点。

问题分析

把图中每个点分成两部分，出点和入点，每条边连接一个出点一个入点，用入点集和出点集构造二分图，求出最大流，也就是二分图最大匹配。二分图最大匹配等于最小点覆盖。在这个基础上判断原图，源点是否能到达其他任意点，其他任意点是否能到达汇点，也就是使除源点和汇点外的每个点入度和出度都不为零，不满足的把相应的边加上。

具体步骤：

首先，构造并建立二分图，将题目中描述的普通关系图转化为一个二分图。由一个 vector 容器保存每个点的所有邻接边的信息，然后设置 4010 和 4011 两个点分别为二分图的源点和汇点，根据建好的邻接表，对二分图加边。

然后，求出此二分图的最大流，也就是最大匹配。

最后，根据源点入度为零，汇点出度为零，确定出原图真正的源点和汇点。在求出的最大匹配的基础上，循环每条边，若此边入点出度为零或者出点入度为零，则需要加上这条边，把边数进行累加。

参考代码

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <sstream>
#include <map>
#include <set>
#include <queue>
#include <vector>
using namespace std;
typedef pair<int, int> pii;
#define INF 100000007
struct edge
{
	int from, to, resid, origr, antip, isresid;
};
int n, m, from, to, res;
vector<int> g[2014];
vector<pair<int, int> > flowg[4014];
int p[4014], pe[4014], u[4014], od[4014], id[4014];
vector<edge> e;
void add_edge (int from, int to, int cap)
{
	flowg[from].push_back(make_pair(to, e.size()));
	edge x;
	x.from = from;
	x.to = to;
	x.resid = x.origr = cap;
	x.antip = e.size() + 1;
	x.isresid = 0;
	e.push_back(x);
```

```
        flowg[to].push_back(make_pair(from, e.size()));
        x.from = to;
        x.to = from;
        x.resid = x.origr = 0;
        x.antip = e.size() - 1;
        x.isresid = 1;
        e.push_back(x);
}
int find_path (int x, int rc)
{
        int i;
        u[x] = 1;
        if (x == 4011)
                return rc;
        for(i=0; i<(int)flowg[x].size(); i++)
        {
                if (!u[flowg[x][i].first] && e[flowg[x][i].second].resid)
                {
                        int nrc = min(rc, e[flowg[x][i].second].resid);
                        p[flowg[x][i].first] = x;
                        pe[flowg[x][i].first] = flowg[x][i].second;
                        int ret = find_path(flowg[x][i].first, nrc);
                        if (ret)
                                return ret;
                }
        }
        return 0;
}
int max_flow_ff (void)
{
        int i;
        int rc, res = 0;
        while (1)
        {
                for(i=0; i<4014; i++)
                {
                        p[i] = pe[i] = -1;
                        u[i] = 0;
                }
                rc = find_path(4010, INF);
                if (!rc)
                        break;
                int cx = 4011;
                while (p[cx] != -1)
```

```
            {
                e[pe[cx]].resid -= rc;
                e[e[pe[cx]].antip].resid += rc;
                cx = p[cx];
            }
            res += rc;
        }
        return res;
}
int main ()
{
        int i,j;
        scanf("%d%d", &n, &m);
        for(i=0; i<m; i++)
        {
            scanf("%d%d", &from, &to);
            --from;
            --to;
            g[from].push_back(to);
        }
        //hrany zdroj->vrchol, vrchol->stok
        for(i=0; i<n; i++)
        {
            add_edge(4010, i, 1);
            add_edge(n + i, 4011, 1);
        }
        //hrany mezi vrcholy
        for(i=0; i<n; i++)
            for(j=0; j<(int)g[i].size(); j++)
                add_edge(i, n + g[i][j], 1);
        res = max_flow_ff();
        memset(id,0,sizeof(id));
        memset(od,0,sizeof(od));
        for(i=0; i<(int)e.size(); i++)
            if (!e[i].isresid && e[i].from != 4010 && e[i].to != 4011)
            {
                int fls = e[i].origr - e[i].resid;
                if (fls)
                {
                    ++od[e[i].from];
                    ++id[e[i].to];
                    //printf("Pairing has edge (%d, %d)\n", e[i].from + 1, e[i].to - n + 1);
                }
            }
```

```
    for(i=0; i<(int)e.size(); i++)
        if (!e[i].isresid && e[i].from != 4010 && e[i].to != 4011)
        {
            int fls = e[i].origr - e[i].resid;
            if (!fls && (!od[e[i].from] || !id[e[i].to]))
            {
                e[i].origr = 1;
                e[i].resid = 0;
                ++od[e[i].from];
                ++id[e[i].to];
                //printf("Adding edge back to graph (%d, %d)\n", e[i].from + 1,
e[i].to - n + 1);
                ++res;
            }
        }
    printf("%d\n", m - res);
    return 0;
}
```

Beer Can Game

问题分析

将数字对应的位置放上固定个数个'?'这个问题就可以转化成比较经典的求字符串距离的问题。对于这个问题只需要二维 dp，f[i][j]表示 s 串前 i 个字符到 t 串前 j 个字符的距离 dp 转移即可。时间复杂度空间复杂度均为 O(n*m)。

参考代码

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
char sch[10005],tch[1005];
int s[11005],t[2005],slen,tlen,f[11005][2005],ans;
int main()
{
```

```
	scanf("%s%s",sch,tch);
	for(int i=0;sch[i];i++)
	{
		if(sch[i]>='a'&&sch[i]<='z')
			s[++slen]=sch[i]-'a';
		else
		{
			for(int j=1;j<=sch[i]-'0';j++)
				s[++slen]=-1;
			ans++;
		}
	}

	for(int i=0;tch[i];i++)
	{
		if(tch[i]>='a'&&tch[i]<='z')
			t[++tlen]=tch[i]-'a';
		else
		{
			for(int j=1;j<=tch[i]-'0';j++)
				t[++tlen]=-1;
			ans++;
		}
	}
	memset(f,0x3f,sizeof f);
	f[0][0]=0;
	for(int i=0;i<=slen;i++)
	for(int j=0;j<=tlen;j++)
	{
		if(s[i+1]==t[j+1]||s[i+1]==-1||t[j+1]==-1)
			f[i+1][j+1]=min(f[i][j],f[i+1][j+1]);
		f[i+1][j]=min(f[i][j]+1,f[i+1][j]);
		f[i][j+1]=min(f[i][j]+1,f[i][j+1]);
	}
	printf("%d",ans+f[slen][tlen]);
}
```

Beer Marathon

题目大意

在一条直线上有 N 个啤酒摊，这些啤酒摊的位置在直线上随机摆放，题目要求是任何两个连续啤酒摊位之间的距离应完全相同，并等于指定的特定值 K。问每个啤酒摊最少移动多少米可以使他们之间的距离是 K 值（尽量减少所有啤酒摊位移动的总米数

问题分析

题目的最优解会对应一个起点，该起点最终存在，取区间范围 [-1e14,1e14] 为较大的边界，定义 low、high 分别初始化为-1e14 和 1e14 在该范围内用类似三分查找，查找该起点，每次查找计算两个起点对应的移动所需距离，左起点为 low+(high-low)，右起点为 low-(high-low)。如果左起点对应所需移动的距离小于右起点，则右起点 high 左移， 左起点 low 不变，同时向右移动减少，向左移动变大。如果右小于左则相反。取每次计算 结果的最小值，循环 100 次为 1e8 时间复杂度，3 的 100 次方的精准度。

参考代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<math.h>
using  namespace  std;
int  a[10000001]=  {0};
long  long  int  l=-1000000000007,r=1000000000007;
long  long  int  er(long  long  int  w,int  n,int  k)
{
        long  long  int  ans=0;
        int  i;
        for(i=0;  i<n;  i++)
        {
                ans+=abs(a[i]-w);
                w+=k;
        }
        return  ans;
}
int  main()
{
        #ifdef  ONLINE_JUDGE
        #else
                freopen("in.txt","r",stdin);
        #endif
        int  n,k;
        int  i,j;
        long  long  int  mid=0,L,R,ans1=0,ans2=0,ans=1000000000000000000;
        scanf("%d  %d",&n,&k);
```

```
        for(i=0;  i<n;  i++)
        {
                scanf("%d",&a[i]);
        }
        sort(a,a+n);
        for(i=0;  i<100;  i++)
        {
//                mid=(l+r)/2;
//                L=(mid+l)/2;
//                R=(mid+r)/2;
                L=l+(r-l)/3;
                R=r-(r-l)/3;
                ans1=er(L,n,k);
                ans2=er(R,n,k);
                if(ans1<ans2)
                {
                        r=R;
                }
                else
                {
                        l=L;
                }
                ans=min(ans,min(ans1,ans2));
        }
        printf("%lld\n",ans);
        return  0;
}
```

Beer Mugs

题目大意

给你一个字符串,让你求最长的连续的一段重新改变顺序可构造成回文串的子串。

问题分析

由于 n<=30w,所以暴力肯定不行,那我们可以用异或前缀和来做,我们可以先将 a-t 表示成 2^0-2^19 这样的二进制的形式,然后从左到右扫一遍该字符串,求出从该字符串开头到每个位置的异或和(也就是每一个状态),并将每一个状态第一次出现的的位置进行标记。若该状态在之前已经出现过,根据异或的性质,第一次出现该状态的位置到该状态的位置的距离即为一个可构造成回文串的子串,当然有可能会出现奇数字符的子串可构造成回文串(像是 abcab 这种可能不会被统计到),所以我们可以在每次循环的时候对 a 到 t 进行循环,然后对他们所表示的数进行单独异或(排除单独一个字符的影响,目的是补一个字符以避免该状态不被统计到)。

参考代码:

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=300007;
int n;
unordered_map<int,int>p;
char str[maxn];
int main()
{
	scanf("%d",&n);
	scanf("%s",str+1);
	int now=0,ans=0;
	p[0]=0;
	for(int i=1;i<=n;i++)
	{
		now^=(1<<(str[i]-'a'));
		if(p.count(now))
		ans=max(ans,i-p[now]);
		else
		p[now]=i;
		for(int j=0;j<20;j++)
		{
			int y=1<<j;
			if(p.count(now^y))
			ans=max(i-p[now^y],ans);
		}
	}
	printf("%d\n",ans);
```

```
    return 0;
}
```

Screamers In The Strom

题目大意

分析题意得，前半部分全是废话。

行动规则：
狼向东走，在到最东部后会跳至最西面，然后接着向东走，周遭往复。羊向南走，到头后和狼同理。
场地规则：
地上本是秃地，记作“.”三回合后会长草，草地记作“#”，羊会吃草，吃掉三回合后草会重新长，狼会吃羊，羊死后会标记为“*”
如果羊 5 回合没吃草会被饿死，狼则为 10 回合，死后都会被记作“*”标记为“*”的砖块不会再长草。

问题分析

明知题意，我们首先定义一个计数器 growps 记录轮数，eaten 记录羊吃草后的轮数，然后写入数据，分别判断遍历到狼和羊时的操作，另外利用两个计数器判定羊和狼超过对应轮数是否会饿死。
最后将判断后的数据输出。（注意因为当“*”上面有羊和狼时要优先输出羊和狼再进行判断是否死掉）

参考代码

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

const int imax = 22;

int t, m, n, growps[imax][imax], eaten[imax][imax], oldeat[imax][imax];
char mymap[imax][imax], oldmap[imax][imax];
```

```
void myswop(int x, int y, int xx, int yy)
{
    if (mymap[xx][yy] == '.')
    {
        mymap[xx][yy] = oldmap[x][y];
        eaten[xx][yy] = oldeat[x][y];
        oldeat[x][y] = 0;
    }
    else
    {
        mymap[xx][yy] = 'W';
        eaten[xx][yy] = -1;
        growps[xx][yy] = t + 1;
    }
}
int main()
{
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);
    cin >> t >> m >> n;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            growps[i][j] = 0;
            eaten[i][j] = 0;
            cin >> mymap[i][j];
            if (mymap[i][j] == '#') growps[i][j] = -5;
        }
    }


    for (int wa = 1; wa <= t; wa++)
    {
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                oldmap[i][j] = mymap[i][j];
```

```
            oldeat[i][j] = eaten[i][j];
            mymap[i][j] = '.';
            eaten[i][j] = 0;
        }
    }
    for (int i = 0; i < m; i++)///判断羊狼时操作
    {
        for (int j = 0; j < n; j++)
        {
            if (oldmap[i][j] == 'W')
            {
                int next = j < (n - 1) ? j + 1 : 0;
                myswop(i, j, i, next);
            }
            else if (oldmap[i][j] == 'S')
            {
                int next = i < (m - 1) ? i + 1 : 0;
                myswop(i, j, next, j);
            }
        }
    }

    for (int i = 0; i < m; i++)///判断轮数 5,10 后羊狼生死
    {
        for (int j = 0; j < n; j++)
        {
            if (mymap[i][j] == '.')
                continue;
            if (mymap[i][j] == 'S')
            {
                if ((wa - growps[i][j]) > 3)
                {
                    eaten[i][j] = 0;
                    growps[i][j] = wa;
                }
                else
                    eaten[i][j] ++;
                if (eaten[i][j] == 5)
                {
                    mymap[i][j] = '.';
                    eaten[i][j] = 0;
```

```
                    growps[i][j] = t + 1;
                }
            }
            if (mymap[i][j] == 'W')
            {
                if (eaten[i][j] == 9)
                {
                    mymap[i][j] = '.';
                    eaten[i][j] = 0;
                    growps[i][j] = t + 1;
                }
                else
                    eaten[i][j]++;
            }
        }
    }
}
for (int i = 0; i < m; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (mymap[i][j] == 'W' || mymap[i][j] == 'S')
        {
            cout << mymap[i][j];
            continue;
        }
        if (growps[i][j] == t + 1)
        {
            cout << "*";
            continue;
        }
        if (mymap[i][j] == '.')
        {
            if (t - growps[i][j] >= 3)
                cout << "#";
            else if (t - growps[i][j] > 0)
                cout << ".";
        }
    }
    cout << endl;
}
```

```
    return 0;
}
```

Sixpack

题目大意：

已知一个 2*N（3≤N≤10^5）的网格，每相邻的三列称为一个 sixpack，要求一个 sixpack 中各个网格的和等于一个定值 K（0≤K≤100），当然网格中的数值可以为 0。某些网格中的数值是已知的，也就是题目中会给定 M（0≤M≤2·10^5）个网格的固定值，以下 m 行中的每一列包含三个整数 C（0<=C<=N-1）、R（0≤R≤1）和 V（0≤V≤9）。C 和 R 指定网格中单元格的列和行，V 是该单元格中的预定义值。网格中每个单元格的值最多指定一次。
求一个杂志阅读器可以获得的最大数量的六包。把这个数字按模 1000000007 打印出来。

问题分析：

assert 宏的原型定义在< assert.h>中，其作用是如果它的条件返回错误，则终止程序执行。

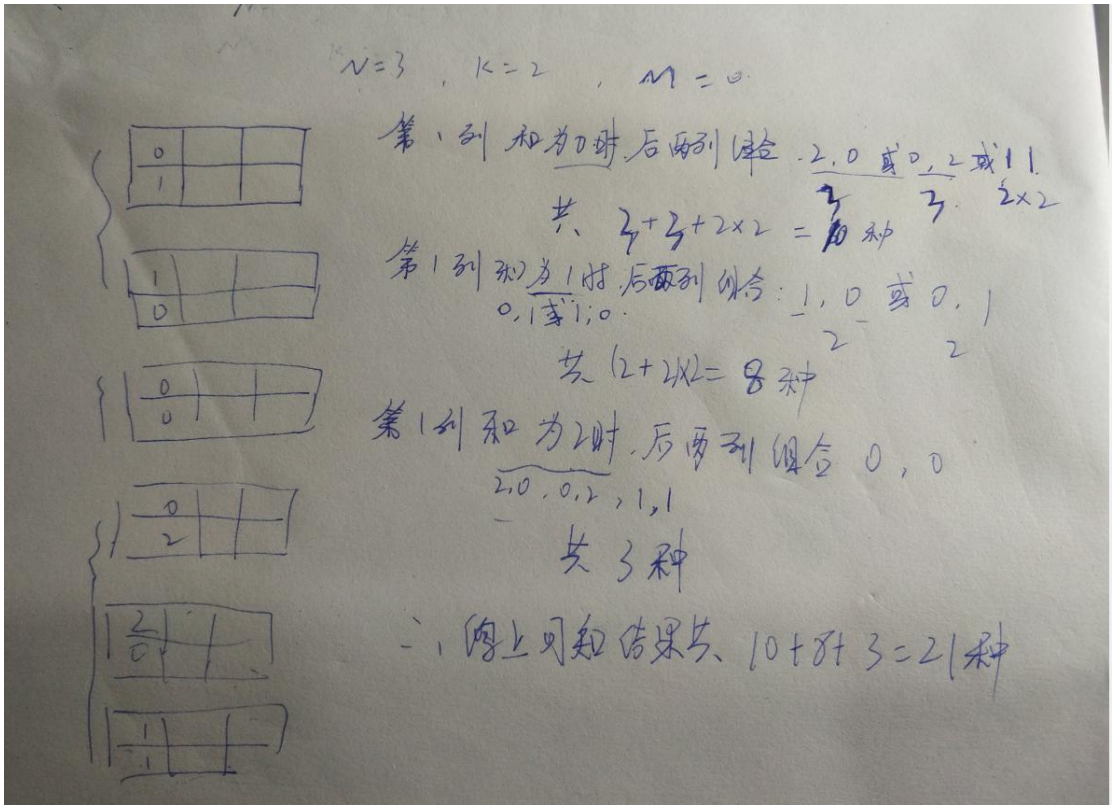
array 是一个固定大小的顺序容器，不能动态改变大小。

还得就是 1，4 列，2，5 列，3，6 列等等往后的和得相等。

第三列的和是由第一列跟第二列决定的，然后第四列又是由第二列和第三列决定的，依次这样，知道前面的就能决定后边的.故而得出和看前三列就行。

所以将前三列和的情况枚举出来即可，每种情况一旦确定，此后每一列的和分成两个数相加，求其方案数，往后全乘起来，那么这种和的排列的总方案数就求出来了，其他前三列和的情况与此类似，最后求和 mod1000000007 即可。

附上样例一的计算过程，具体实现代码注释详解。



参考代码

```
#include <bits/stdc++.h>
using namespace std;
using DPVal = int;
const int N = 1e5;
const int K = 100;
const int MAX_CASE_CNT = 1000;
const int UNKNOWN = -1;
const int MOD = 1e9+7;
map<int,int> ways[3]; // combine[how_many_nums][sum]
int filledin[N][2];//filledin 这里把题意中固定的两行记录成固定的两列
// fixed sums mod 3
uint32_t solve(int n, int k, array<int,3> sums/*, const vector<int,array<int,2>> & filledin*/)
{
    uint64_t res = 1;
    int currentSum = 0, nums = 0;
    for (int c = 0; c < n; ++c)
    {
        if (filledin[c][0] == UNKNOWN && filledin[c][1] == UNKNOWN)//两个网格的数都未知，与和为 sums[c%3]方案数相乘
            res = (res * ways[2][sums[c % 3]]) % MOD;
        else if (filledin[c][0] != UNKNOWN && filledin[c][1] != UNKNOWN)//某列两个网格数字全部给定
```

```
        {
            if (filledin[c][0] + filledin[c][1] != sums[c % 3])//这两个网格的和不等于 sums[c%3] 此种排列方案没有
                res = 0;
        }
        else if (filledin[c][0] != UNKNOWN)//某一列上面的数是已知的
            res = (res * ways[1][sums[c % 3] - filledin[c][0]]) % MOD;//剩下一个网格的数字确定为 sums[c % 3] - filledin[c][0]，方案数为 ways[1][sums[c % 3] - filledin[c][0]]
        else if (filledin[c][1] != UNKNOWN)//某一列下面的数是已知的
            res = (res * ways[1][sums[c % 3] - filledin[c][1]]) % MOD;//剩下一个网格的数字确定为 sums[c % 3] - filledin[c][1]，方案数为 ways[1][sums[c % 3] - filledin[c][1]]
    }
    return res;
}

int main()
{
    ios::sync_with_stdio(0);
    for (int i1 = 0; i1 < 10; ++i1)
    {
        for (int i2 = 0; i2 < 10; ++i2)
            ++ways[2][i1+i2];//初始化两个方格和为 i1+i2 的方案数
        ++ways[1][i1];//初始化一个方格和为 i1 的方案数
    }
    memset(filledin, -1, sizeof(filledin));
    int n, k, m, r, c, val;
    assert(cin >> n >> k >> m);
    assert(3 <= n && n <= N);
    assert(0 <= k && k <= K);
    assert(0 <= m && m <= 2 * n);
    for (int i = 0; i < m; ++i)
    {
        assert(cin >> c >> r >> val);
        assert(0 <= c && c < n);
        assert(0 <= r && r < 2);
        assert(0 <= val && val < 10);
        assert(filledin[c][r] == UNKNOWN);//只能规定网格的数字一次
        filledin[c][r] = val;
    }
    int res = 0;
    for (int i1 = 0; i1 <= min(k, 2 * 9); ++i1)//枚举三列的和，以三列的和决定某种方案
    {
        for (int i2 = 0; i2 <= min(k - i1, 2 * 9); ++i2)
        {
            int i3 = k - i1 - i2;
            if ( i3 >= 0 && i3 <= 2 * 9)
                res = (res + solve(n, k, { i1, i2, i3 })) % MOD;
        }
    }
    cout << res << endl;
    return 0;
}
```

Beer Vision

问题分析

题意可转化为对于所有的集合中的点，满足+x，+y 或-x，-y 至少有一个点也在集合中。问可以选出多少个这样不同的 x 和 y。对每对点做差，满足要求的 x，y 一定在差值中至少出现了 n/2（上取整）次。这样的 x,y 显然最多有 n 个。枚举判断是否合法即可，复杂度 O(n^2)。

参考代码

```
#include<cstdio>
using namespace std;
int n,a[4005][4005],c[4005][4005],x[1005],y[1005],ans;
int main()
{
	scanf("%d",&n);
	for(int i=1;i<=n;i++)
	{
		scanf("%d%d",&x[i],&y[i]);
		x[i]+=1000;y[i]+=1000;
		a[x[i]][y[i]]=1;
	}
	for(int i=1;i<=n;i++)
	for(int j=1;j<=n;j++)
	if(i!=j)
	{
		c[x[i]-x[j]+2000][y[i]-y[j]+2000]++;
	}
	for(int X=0;X<=4000;X++)
	for(int Y=0;Y<=4000;Y++)
	{
		if(c[X][Y]>=(n+1)/2)
		{
//			printf("%d %d\n",X,Y);
			for(int i=1;i<=n;i++)
			{

	if((x[i]-X+2000<0||y[i]-Y+2000<0||a[x[i]-X+2000][y[i]-Y+2000]==0)&&(x[i]+X-2000<0||y[
i]+Y-2000<0||a[x[i]+X-2000][y[i]+Y-2000]==0))
					break;
//				printf("%d->\n",i);
				if(i==n)
					ans++;
			}
		}
```

```
        }
        printf("%d",ans);
}
```