对拍

一个保证正确的程序 n^2

一个需要检测正确性的程序 nlogn

一个生成测试数据的程序

一个自动运行以上三者并检测对错的程序

```
// brute.cpp 放一个可以保证正确性的代码
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    cin>>a>>b;
    cout<<a+b<<endl;
    return 0;
}
```

```
// code.cpp 放你要检测正确性的代码
#include <bits/stdc++.h>
using namespace std;

int main() {
    srand(time(0));
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d\n", a+b+(rand()%5==4)); // 故意制造一些错误
    return 0;
}
```

```
// 生成随机数据
#include <bits/stdc++.h>
using namespace std;
const int lim = 1e8;

int myrand(int L, int R) {
    int rnd = rand() << 15 | rand(); // 注意由于 windows 下 rand() 的返回值最大为
32767，为了获得 int 范围内的随机数需要这么做
    if(rnd < 0) rnd = -rnd;
    return rnd % (R - L + 1) + L;
}

int main() {
    srand(time(0));
    int a, b;
    a = myrand(1, lim);
    b = myrand(1, lim);
    printf("%d %d\n", a, b);
    return 0;
}
```

```
// 最终运行的对拍程序
#include <iostream>
using namespace std;
int main() {
    int rp = 0;
    while (!rp) {
        system("data.exe > input.txt"); // 生成数据，将结果存储在 input.txt中
        system("force.exe < input.txt > fout.txt"); // 调用暴力代码，将结果存储在
fout.txt 中
        system("std.exe < input.txt > sout.txt"); // 调用对比代码，将结果存储在
sout.txt 中
        rp = system("fc fout.txt sout.txt"); // 进行对比
        if (rp == 0)  cout << "AC" << endl;
        else cout << "WA" << endl;
    }
    return 0;
}
```

```
// 生成各种形态的随机数据
#include <iostream>
#include <cstdlib> // rand(), srand()
#include <ctime> // time()
#include <set>
#include <vector>
#include <algorithm> // shuffle
#include <utility> // pair
#include <bits/stdc++.h>
using namespace std;

int random(int n) {//返回0-n-1之间的随机整数
    // cout << rand() % n << '\n';
    return rand()%n;
}

void generateRandomArray() {//随机生成长度为n的绝对值在1e9之内的整数序列
    int n = random(1e5) + 1;
    int m = 1e9;
    for (int i = 1; i <= n; i++) {
        cout << random(2 * m + 1) - m << '\n';
    }
}

void generateIntervals() {//随机生成 m个[1,n]的子区间
    int m = 10, n = 100;
    for (int i = 1; i <= m; i++) {
        int l = random(n) + 1;
        int r = random(n) + 1;
        if (l > r) swap(l, r);
        cout << l << " " << r << '\n';
    }
}

void generateTree() {//随机生成一棵n个点的树，用n个点n-1条边的无向图的形式输出
    int n = 10;
```

```
    for (int i = 2; i <= n; i++) {//从2 ~ n之间的每个点i向 1 ~ i-1之间的点随机连一条边
        int fa = random(i - 1) + 1;
        int val = random(1e9) + 1;
        cout << fa << " " << i << " " << val << '\n';
    }
}

void generateGraph() {//随机生成一张n个点m条边的无向图，图中不存在重边、自环
    int n = 10, m = 6;
    set<pair<int, int>> edges;//防止重边
    for (int i = 1; i <= n; i++) {//先生成一棵树，保证连通
        int fa = random(i - 1) + 1;
        edges.insert({ fa, i + 1 });
        edges.insert({ i + 1, fa });
    }
    while (edges.size() < m) {//再生成剩余的 m-n+1 条边
        int x = random(n) + 1;
        int y = random(n) + 1;
        if (x != y) {
            edges.insert({ x, y });
            edges.insert({ y, x });
        }
    }
    // Shuffling and outputting
    vector<pair<int, int>> Edges(edges.begin(), edges.end());
    random_shuffle(Edges.begin(), Edges.end());
    for (auto& edge : Edges) {
        cout << edge.first << " " << edge.second << '\n';
    }
}

int main() {
    srand(time(0));
    /*随机生成*/
    return 0;
}
```

最后运行以下testlib那个例子

分数取模和质数计算

取模

加法乘法直接取模

res = (a+b)%MOD

res = (a*b)%MOD

减法先加一个再取模

res = (a-b+MOD)%MOD

除法参考以下分数取模

a/b % MOD 逆元

快速幂

O(b)

logb 时间求 a^b

```
int MOD = 1e9+7;
int ksm(int a,int b){ // a^b b=5 --> 0b101 --> a^5 a * a^4  a a^2 a^4
    int res = 1, temp = a;
    while(b){ 5 --> 0b101
        if(b&1) res = (res*temp)%MOD;
        temp = (temp*temp)%MOD;
        b >>= 1;
    }
    return res%MOD;
}
```

分数取模

考虑需要计算 ***(a / b) % M*** 的情况

```
int res = (a/b)%MOD = a * ksm(b, MOD - 2) % MOD; // a * b^(MOD-2) 费马小定理
998244353
```

组合数取模计算

C(n,m) = n! / (m!)* (n-m)!

P(n,m) = n! / （n-m）！

O(n)

m [L,R] C(n,m) O(n)O(n)

O(n) O(1)

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 100010, mod = 1e9 + 7;

typedef long long LL; //long long 缩写

int fact[N], infact[N]; // fact[i]:i的阶乘 infact[i]:i的阶乘逆元 可以理解成 1/fact[i]
对mod取模之后的值
// a/b --> a * (1/b)
// (1/b)--> x
// (a/b)%MOD --> (a*x)%MOD

int qmi(int a, int b) //快速幂模板
```

```
{
    int res = 1;
    while(b)
    {
        if(b & 1) res = (LL)res * a % mod;
        a = (LL)a * a % mod;
        b >>= 1;
    }

    return res;
}
int main()
{
    // 初始化阶乘和阶乘对mod取模的逆元
    fact[0] = infact[0] = 1; //初始阶乘为一

    for(int i = 1; i < N; i ++) //阶乘和逆元预处理 C(n,m)
    {
        fact[i] = (LL)fact[i - 1] * i % mod;
        infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2) % mod; // i^(MOD-2)
    } // 1/(i!)
    // fact[x]/infact[y] --> (fact[x] * infact[y])%MOD;

    int n;
    cin >> n;

    while(n --)
    {
        int a, b;
        cin >> a >> b;
        // C(a,b) = a! / (b!)*(a-b)!
        cout << (LL)fact[a] * infact[b] % mod * infact[a - b] % mod << endl; //组
合数公式
    }

    return 0;
}
```

质数计算

根号n --> n --> 2-根号n i

埃氏筛 n 1-n

时间复杂度 nlognlogn 空间复杂度 n 2e6 线性筛 O(n)

```
vector<int> prime; // 存放1-n的质数
bool is_prime[N]; // is_prime[i]表示i是否是质数 is_prime[x] = false i
// 1-N -->  [L-R]
void Eratosthenes(int n) {
  is_prime[0] = is_prime[1] = false;
  for (int i = 2; i <= n; ++i) is_prime[i] = true;

  for (int i = 2; i <= n; ++i) {
    if (is_prime[i]) { // 对于每个质数 遍历它的倍数并且标记为false
```

```
      prime.push_back(i);
      for (int j = 2*i; j <= n; j += i) n/i --> O(n)
        // 因为从 2 到 i - 1 的倍数我们之前筛过了，这里直接从 i
        // 的倍数开始，提高了运行速度
        is_prime[j] = false;  // 是 i 的倍数的均不是素数
    }
  }
}
// 1/1 + 1/2 + 1/3 + 。。。 + 1/n logn
```

区间筛

在区间[a,b)内有多少个素数

素数判定中，如果d是n的因子，那么n/d也是n的因子。且 min(d,n/d) <= $\sqrt{n}$

因此 b 以内的合数的最小质因数一定不超过 $\sqrt{b}$。 如果有 $\sqrt{b}$内的素数表的话，就可以把埃氏筛法应用到[a,b)上。

即如果 [a,b)的某个数是合数，它在最多遍历到根号b时就会被筛掉

先分别做好[2,$\sqrt{b}$)的表和[a,b)的表，然后从[2,$\sqrt{b}$)的表中筛得素数的同时，也将倍数从[a,b)的表划去，剩下的就是[a,b)内的素数了

```
typedef long long ll;

bool is_prime[MAXN];//is_prime[i-a]=true <=> i是素数
bool is_prime_small[MAXN];//[2, 根号b] 的表

//对区间[a,b)内的整数筛选素数。
void segment_sieve(ll a,ll b){
    for(int i=0;(ll)i*i<b;i++)  is_prime_small[i]=true;
    for(int i=0;i<b-a;i++)  is_prime[i]=true;

    for(int i=2;(ll)i*i<b;i++){
        if(is_prime_small[i]){ // 相当于用埃氏筛计算small表范围的素数的过程中 一并标记
[a,b)的表
            //筛[2,根号b)
            for(int j=2*i;(ll)j*j<b;j+=i)   is_prime_small[j]=false;
            //筛[a,b)
            for(int j=max(2LL,((a+i-1)/i)*i;j<b;j+=i)   is_prime[j-a]=false;
        }
    }
}
```

竞赛建图

n个顶点 m条边 的有向图 1e5 1e6 m --> n^2

n * n

nlogn 堆优化 n^2

```
vector<pair<int,int>> edge[100010]; // 取决于n的范围
```

```
vector<int> edge[100010]; // edge[i] --> vector<int> = {a,b,c}
struct edge{
    int next, to, weight;
}[100010]; // 链式前向星

int main(void)
{
    cin >> n >> m;
    for(int i=1;i<=m;i++){
        int u,v,w;
        cin >> u >> v >> w;
        edge[u].push_back(<v,w>);
    }

    for(int i=0;i<edge[now].size();i++){
        int from = now, to = edge[now][i].first, weight = edge[now][i].second;
    }

    return 0;
}
```

n个顶点 m条边的无向图

```
vector<pair<int,int>> edge[100010]; // 取决于n的范围

int main(void)
{
    cin >> n >> m;
    for(int i=1;i<=m;i++){
        int u,v,w;
        cin >> u >> v >> w;
        edge[u].push_back(<v,w>);
        edge[v].push_back(<u,w>);
    }

    for(int i=0;i<edge[now].size();i++){
        int from = now, to = edge[now][i].first, weight = edge[now][i].second;
    }

    return 0;
}
```

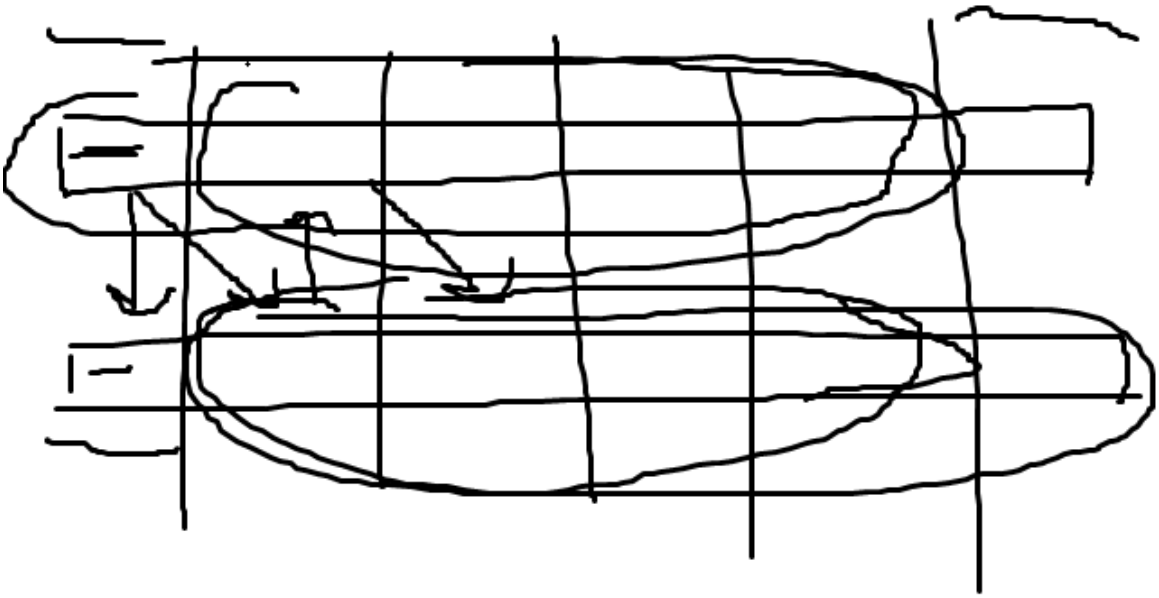
KMP应用的例子

https://www.luogu.com.cn/problem/P4391

有一个长串s1，已知它是由某个字符串s2多次重复而成（至少两次）。

现在只知道s1的一个子串，求s2的最短长度。

（画图解释下）



代码非常短，其实就是 L - next[n]

```
#include<cstdio>
using namespace std;
const int maxn=1111111;
int n,kmp[maxn];//kmp数组即next数组
char ss[maxn];
int main()
{
    scanf("%d%s",&n,ss+1);
    int j=0;
    for(int i=2;i<=n;++i)
    {
        while(j&&ss[i]!=ss[j+1]) j=kmp[j];
        if(ss[i]==ss[j+1]) ++j;
        kmp[i]=j;
    }
    printf("%d",n-kmp[n]);
    return 0;
}
```

KMP应用2

https://leetcode.cn/problems/shortest-palindrome/solutions/392561/zui-duan-hui-wen-chuan-by-leetcode-solution/

给定一个字符串 s，可以在前面添加任意前缀 s'，得到 s'+s。

问最短的 s' 的长度，使得 s'+s 是回文串。返回这个s'+s。

时间复杂度需要 O(n) 或 O(nlogn)

先考虑哈希做法：

（画个图）

只要枚举 s 每个长度的前缀，判断它是不是回文的。

注意奇数/偶数长度前缀的分类讨论

选择最长的前缀 i，需要添加的距离就是 |s| - i

添加的内容是 s 长度为 |s|-i 的后缀的逆序串

```
class Solution {
public:
    string shortestPalindrome(string s) {
        int n = s.size();
        int base = 131, mod = 1000000007;
        int left = 0, right = 0, mul = 1;
        int best = -1;
        for (int i = 0; i < n; ++i) { // 枚举s的所有前缀
            // left：左边高位 右边低位
            left = ((long long)left * base + s[i]) % mod;
            // right：左边低位 右边高位
            right = (right + (long long)mul * s[i]) % mod;
            if (left == right) {
                best = i;
            }
            // 计算right时的权重
            mul = (long long)mul * base % mod;
        }
        // best == n-1 表示s本身就是回文的
        string add = (best == n - 1 ? "" : s.substr(best + 1, n));
        reverse(add.begin(), add.end());
        return add + s;
    }
};
```

考虑KMP做法：

回忆next数组的作用，设 s 的逆序串为 s'。

如果s的长度为best的前缀是回文的，那么s长度为best的前缀 = s'长度为best的后缀

（设s_best为s的前缀，则s_best的逆序串为s'的后缀。由于s_best是回文串，则 s_best = 它的逆序串）

因此把 s 看作模式串，s'看作母串，计算 s 的next数组，在 s' 上从后往前匹配。

```
class Solution {
public:
    string shortestPalindrome(string s) {
        int n = s.size();
        // 计算next数组
        vector<int> fail(n, -1);
        for (int i = 1; i < n; ++i) {
            int j = fail[i - 1];
            while (j != -1 && s[j + 1] != s[i]) {
```

```
                j = fail[j];
            }
            if (s[j + 1] == s[i]) {
                fail[i] = j + 1;
            }
        }
        int best = -1;
        // 从后往前遍历s 模拟在s'上的匹配
        // best记录最长回文前缀的长度
        for (int i = n - 1; i >= 0; --i) {
            // 模式串s与母串s'失配 则在next数组上往前跳
            // 和模板题的匹配思路一致
            while (best != -1 && s[best + 1] != s[i]) {
                best = fail[best];
            }
            // 如果s[0:best+1] == s[i:n-1]
            if (s[best + 1] == s[i]) {
                ++best;
            }
        }
        // 此时遍历到s'的末尾（即s的开头）
        // 匹配到s中下标为best的字符
        // 表示s的前缀 = s'的后缀（这里再画个图）
        // best == n-1 表示s本身就是回文的
        string add = (best == n - 1 ? "" : s.substr(best + 1, n));
        reverse(add.begin(), add.end());
        return add + s;
    }
};
```

五、动态规划

2. 区间DP

最经典的石子合并 https://www.luogu.com.cn/problem/P1880

N堆石子，每堆a_i个。每次可以选相邻的合成一堆。并把新堆的石子数量记作该次合并的得分。问最终合并成一堆时的最小得分和最大得分。

和之前合并果子（可以用优先队列/堆写的那个）的区别：那个每次可以任选两堆。这题只能选相邻的。

dp[i] [j] [i,j]

[1,n] dp[1] [n] O(n^3)

dp[i] [j] = dp[i] [k] + dp[k+1] [j]

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int n;
int a[111] = {0};
int pre[111] = {0};
```

```
int dp[111][111] = {0}; // dp[i][j]表示[i,j]内石子的最大得分

int s(int x,int y){
    return pre[y] - pre[x-1];
}
signed main(void)
{
    cin >> n;
    for(int i=1;i<=n;i++) cin >> a[i];
    for(int i=1;i<=n;i++) pre[i] = pre[i-1] + a[i];

    for(int l=1;l<=n;l++){ // l 长度
        for(int i=1;i<=n;i++){ // 左端点
            int j = i+l-1;  // 右端点
            if(j>n) continue;
            if(l==1){
                dp[i][j] = 0; // dp[i][j] [i,j]
                continue;
            }
            dp[i][j] = 1e9;
            for(int mid=i;mid<j;mid++){ // [i, mid] [mid+1, j]
                dp[i][j] = min(dp[i][j], dp[i][mid] + dp[mid+1]
[j]+s(i,mid)+s(mid+1,j)); // s(i,j)
            }
        }
    }
    cout << dp[1][n] << '\n';
    memset(dp,0,sizeof(dp));
    for(int l=1;l<=n;l++){
        for(int i=1;i<=n;i++){
            int j = i+l-1;
            if(j>n) continue;
            if(l==1){
                dp[i][j] = 0;
                continue;
            }
            for(int mid=i;mid<j;mid++){
                dp[i][j] = max(dp[i][j], dp[i][mid] + dp[mid+1]
[j]+s(i,mid)+s(mid+1,j));
            }
        }
    }
    cout << dp[1][n];
    return 0;
}
```

同时，注意这题的石子是环形的，首尾两堆也算相邻。

1 - n n+1 - 2n dp[1]

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int n;
int a[222] = {0};
```

```
int pre[222] = {0};
int dp[222][222] = {0}; // dp[i][j]表示[i,j]内石子的最大得分

int s(int x,int y){
    return pre[y] - pre[x-1];
}
signed main(void)
{
    cin >> n;
    for(int i=1;i<=n;i++) cin >> a[i];
    for(int i=n+1;i<=2*n;i++) a[i] = a[i-n];
    for(int i=1;i<=2*n;i++) pre[i] = pre[i-1] + a[i];

    for(int l=1;l<=n;l++){
        for(int i=1;i<=2*n;i++){
            int j = i+l-1;
            if(j>2*n) continue;
            if(l==1){
                dp[i][j] = 0;
                continue;
            }
            dp[i][j] = 1e9;
            for(int mid=i;mid<j;mid++){
                dp[i][j] = min(dp[i][j], dp[i][mid] + dp[mid+1]
[j]+s(i,mid)+s(mid+1,j));
            }
        }
    }
    int res = 2e9;
    for(int i=1;i<=n;i++) res = min(res, dp[i][i+n-1]);
    cout << res << '\n';
    memset(dp,0,sizeof(dp));
    for(int l=1;l<=n;l++){
        for(int i=1;i<=2*n;i++){
            int j = i+l-1;
            if(j>2*n) continue;
            if(l==1){
                dp[i][j] = 0;
                continue;
            }
            for(int mid=i;mid<j;mid++){
                dp[i][j] = max(dp[i][j], dp[i][mid] + dp[mid+1]
[j]+s(i,mid)+s(mid+1,j));
            }
        }
    }
    res = 0;
    for(int i=1;i<=n;i++) res = max(res, dp[i][i+n-1]); //dp[1][2n]
    cout << res << '\n';
    return 0;
}
```

合并相邻相同数字 https://www.luogu.com.cn/problem/P3146

如果两个x相邻，可以把它们换成一个x+1。游戏的目标是最大化最终序列中的最大数值。

dp[i] [j]

```
#include <bits/stdc++.h>
using namespace std;
int f[255][255] = {0}; // dp数组

    int main() {
        int n, ans = 0;
        cin >> n;
        for(int i = 1; i <= n; i++) { // f[i][i]
            cin >> f[i][i];
            ans = max(ans, f[i][i]);
        }
        // 注意ans的初始值 全局max
        for(int len = 2; len <= n; len++)
            for(int l = 1; l +len -1 <= n; l++) {
                int r = l + len - 1;
                if(r > n) continue;
                for(int k = l; k < r; k++)  // 枚举mid f[i][j] f[i][mid] f[mid+1]
[j]
                // 注意这里的f[l][k]>0
                // 如果没有这句 会发生: f[l][k] = f[k+1][r] = 0时 f[l][r] = 1的情况
                // 这种更新是错误的（每个数的范围在 1…40 之间）
                    if(f[l][k] == f[k + 1][r] && f[l][k]>0) {
                        f[l][r] = max(f[l][r], f[l][k] + 1);
                        ans = max(ans, f[l][r]); // 答案不一定是dp[1][n]
                    }
            }

        cout << ans << '\n';
        return 0;
    }
```

3. 状压DP

糖果 https://www.luogu.com.cn/problem/P8687

M种口味，每包糖k颗，口味都已知。最少买几包可以吃到所有口味?

```
#include<bits/stdc++.h>
using namespace std;
const int N=20;
int n,m,k,dp[1<<20],v[1<<20];//状压的数组需要开2^n，因为表示的是状态
// 2^n *
// dp[i]: 集齐状态i的口味最少需要几包糖
// M: 1111....1111
// 000..000
// dp[11111...111] --> 2^M
int main(){
    scanf("%d%d%d",&n,&m,&k);
    memset(dp,0x3f3f3f3f,sizeof(dp)); // 最大化
    for(int i=1;i<=n;++i){
        int h=0,p;// 1000010
```

```
        for(int j=1;j<=k;++j){ // 每包糖对应的状态h
            scanf("%d",&p);p--;
            h=h|(1<<p);//这里不能用+，一包糖里可能有多种同一口味的
        } 1|1 --> 1
        // h --> 第i包糖里面包含的口味
        dp[h]=1;//这些口味都可以用一包糖解决
        v[i]=h;//记录糖的状态 i --> h
    }

    for(int i=0;i<(1<<m);++i){//i枚举的是状态，即0～1...11111（m个1）
        for(int j=1;j<=n;++j){ // 枚举所有可以买的糖的组合
            // dp[i] = dp[i-1]...

            dp[i|v[j]]=min(dp[i|v[j]],dp[i]+1);// | 拼上去
            // dp[i|v[j]]
        }
    }


  // m个1
    if(dp[(1<<m)-1]==0x3f3f3f3f) cout<<-1;//搭配不出来
    else cout<<dp[(1<<m)-1];//搭配出来
    return 0;
}
```

炮兵阵地 https://www.luogu.com.cn/problem/P2704

NxM的网格，有的是山地（H）有的是平原（P）。

平原可以放军队，山地不能放。每个军队攻击范围十字形，如图所示。

需要放尽可能多的军队格数，但它们不能互相攻击到（攻击范围可以重叠）

求最多可以放的军队数量?

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,m,k;
int s[1005],g[1005];
int f[102][1005][1005],ans;
// f[i][j][k]:第i行，上一行状态为j，本行状态为k的方案数
char ma[103];
int map[103];
int get(int x)//计算某一状态含有多少个1(即有多少个炮兵)  用builtin_popcount也可以
{
    int e=0;
    while(x>0)
    {
        ++e;
        x-=x&(-x);
    }
    return e;
}
int main()
```

```
{
    cin>>n>>m;
    for(int i=1;i<=n;++i)//读入地图，将山地（不能放兵）的地方设为1
    {
        scanf("%s",ma);
        for(int j=0;j<m;++j)
          if(ma[j]=='H') map[i]+=1<<j;
    }
    // ------------------------
    // 先考虑一行中 哪些状态是合法的
    for(int i=0;i<=(1<<m)-1;++i)//枚举一行中所有的状态
        // 如果第i列放了军队 [i-2,i+2]就不能放了
      if(((i&(i<<1))==0)&&((i&(i<<2))==0)&&((i&(i>>1))==0)&&((i&(i>>2))==0))
        //判断每个1左右各两个是否有1
        {
        // 状态i存在
            ++k;
            s[k]=i; // 存放所有合法状态的数组 s[k]表示第k种合法的二进制状态
            g[k]=get(i); // 这种状态中1的数量
            // 如果状态i合法 且都是平原 放上去
            if((i&map[1])==0) f[1][0][k]=g[k];//初始化第一行
        }

    // ------------------------
    //初始化第二行
    for(int i=1;i<=k;++i)//枚举第一行状态
        for(int j=1;j<=k;++j)//枚举第二行状态
            if(((s[i]&s[j])==0)&&((s[j]&map[2])==0))
                f[2][i][j]=max(f[2][i][j],f[1][0][i]+g[j]);
                // 判断是否与地形和第一行冲突 以及是否与山地冲突
                // 写法比较巧妙：考虑什么时候第一行的状态s[i]与第二行状态s[j]冲突
                // 只有[1][x]和[2][x]同时放了军队的情况
                // (s[i]&s[j])==0 表示这两个状态中，没有同一个bit位是1

    // ---------------------
    // 其他行
    for(int i=3;i<=n;++i)//枚举当前行数
        for(int j=1;j<=k;++j)//枚举当前行的状态
            if((map[i]&s[j])==0)//不与地形冲突
                for(int p=1;p<=k;++p)//枚举前一行状态
                    if((s[p]&s[j])==0)//当前行状态不与前一行冲突
                        for(int q=1;q<=k;++q)//枚举前两行
                        //不与前两行冲突，且前两行自身不冲突
                            if(((s[q]&s[p])==0)&&((s[q]&s[j])==0))
                                f[i][p][j]=max(f[i][p][j],f[i-1][q][p]+g[j]);

    for(int i=1;i<=k;++i)//枚举最后两行为结尾的情况，统计答案
        for(int j=1;j<=k;++j)
            ans=max(f[n][i][j],ans);
    cout<<ans;      //输出
    return 0;
}
```

国王问题 https://www.luogu.com.cn/problem/P1896

NxN的棋盘放K个国王，不能互相攻击，有多少种方案?

国王能攻击到它上下左右，以及左上左下右上右下八个方向上附近的各一个格子，共 8 个格子。（八连通）

多了国王数量这个限制。

```
#include<bits/stdc++.h>
using namespace std;
const int M=1<<9;
long long g[M],h[M],f[10][M][82],n,k,tot=0;
// f[i][j][k]:到第i行，状态为j，总共放k个国王的状态有多少种
int main(){
    cin>>n>>k;
    memset(f,0,sizeof(f));
    // 预处理所有状态 把该行不合法的处理出来（国王相邻）
    for(int x=0;x<(1<<n);x++){
        if(!(x&(x>>1))&&!(x&(x<<1)))g[x]=1; // 合法状态
        int w=x;
        while(w){
            if(w%2)h[x]++; // 计数
            w/=2;
        }
        if(g[x])f[1][x][h[x]]=1; // 合法可以放在第一行
    }
    // 对每一行 枚举它的上一行的状态
    for(int x=2;x<=n;x++){
        for(int y=0;y<(1<<n);y++){ // 枚举第x-1行状态
            if(g[y]){ // 如果是合法状态
                for(int z=0;z<(1<<n);z++){ // 枚举第x行状态
                    if(g[z]&&!(y&z)&&!(y&(z>>1))&&!(y&(z<<1))){ // 合法状态 + 不冲
突
                        for(int w=0;w+h[z]<=k;w++) // 枚举之前放的国王个数 总个数不能
超过k
                            // 前x行，第x行状态为z，共有w+h[z]个国王 的方案数量
                            f[x][z][w+h[z]]+=f[x-1][y][w];
                    }
                }
            }
        }
    }
    // 统计总方案数 对于最后一行的所有状态y求和
    for(int y=0;y<(1<<n);y++)tot+=f[n][y][k];
    cout<<tot;
    return 0;
}
```