A - Frog 1

记 f_i 表示青蛙到达石头 i 的最小费用，则有状态转移方程：

$$f_i = \min(f_{i-1} + |h_i - h_{i-1}|, f_{i-2} + |h_i - h_{i-2}|)$$

时间复杂度 $\mathcal{O}(n)$。

```cpp
#include <iostream>

const int maxN = 1e5;
const int maxH = 1e4;
const int inf = 1e9;

int n;
int h[maxN + 10];
int f[maxN + 10];

int main() {
    std::cin >> n;
    for (int i = 1; i <= n; i++) std::cin >> h[i];
    std::fill(f, f + sizeof(f) / 4, inf);
    f[1] = 0;
    for (int i = 2; i <= n; i++) f[i] = std::min(f[i - 1] + std::abs(h[i] - h[i - 1]),
    std::cout << f[n] << '\n';
    return 0;
}
```

B - Frog 2

与上一题不同的是青蛙的步长边了，通过枚举步长来实现转移。

记 f_i 表示青蛙到达石头 i 的最小费用，则有状态转移方程：

$$f_i = \min_{1 \le j \le k} f_{i-j} + |h_i - h_{i-j}|$$

时间复杂度 $\mathcal{O}(nk)$。

```cpp
#include <iostream>

const int maxN = 1e5;
const int maxH = 1e4;
const int inf = 1e9;

int n, K;
int h[maxN + 10];
int f[maxN + 10];

int main() {
    std::cin >> n >> K;
    for (int i = 1; i <= n; i++) std::cin >> h[i];
    std::fill(f, f + sizeof(f) / 4, inf);
    f[1] = 0;
    for (int i = 2; i <= n; i++) {
        for (int k = 1; k <= K; k++) {
            int j = i - k;
            if (j < 1) break;
            f[i] = std::min(f[i], f[j] + std::abs(h[i] - h[j]));
        }
    }
    std::cout << f[n] << '\n';
    return 0;
}
```

C - Vacation

记 f_i 表示前 i 天的状态，发现不能处理连续两天进行同一活动的情况。

尝试更新状态表示，记 $f_{i,j}$ 表示前 i 且第 i 天进行活动 j 的状态，其中 $j \in 1,2,3$ 分别与游泳、捉虫、写作业对应。

关于状态转移，第 i 天只能从第 $i-1$ 天转移得到，于是枚举第 i 天和第 $i-1$ 分别进行的活动进行转移，需要保证第 i 天与第 $i-1$ 天进行的活动不同。

时间复杂度 $\mathcal{O}(n)$，枚举活动可视作常数忽略。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, x[100005][3];
ll dp[100005][3];
int main() {
    cin >> n;

    for (int i = 1; i <= n; i++)
        rep(j, 3) cin >> x[i][j];

    rep(j, 3) dp[1][j] = x[1][j];

    for (int i = 2; i <= n; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                if (j == k)
                    continue;

                dp[i][j] = max(dp[i][j], dp[i - 1][k] + x[i][j]);
            }
        }
    }

    ll ans = 0;
    rep(j, 3)ans = max(ans, dp[n][j]);
    cout << ans << endl;
}
```

D - Knapsack 1

其实就是背包问题。

注意题目要求选取的物品 $\sum w_i \leq W$，需要对所有 $1 \leq i \leq W$ 的 f_i 取最大值。

时间复杂度 $\mathcal{O}(nw)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
ll dp[100005];
int n, w[105], ww;
ll v[105];
int main() {
    cin >> n >> ww;

    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];

    for (int i = 0; i < 100005; i++)
        dp[i] = -1e18;

    dp[0] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = ww; j >= w[i]; j--) {
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
        }
    }

    ll ans = 0;

    for (int i = 0; i <= ww; i++)
        ans = max(ans, dp[i]);

    cout << ans << endl;
}
```

E - Knapsack 2

其实还是背包问题。

但是数据范围来到了 $1 \leq W \leq 10^9$，时空都无法接受。

注意到这次 $1 \leq v_i \leq 10^3$，那么我们更新状态表示，f_i 表示选取物品总价值为 i 时的最小空间，状态转移类似。

答案取最大的使得 $f_i \leq W$ 的 i。

时间复杂度 $\mathcal{O}(nv)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
ll dp[100005];
int n, v[105], ww;
ll w[105];
int main() {
    cin >> n >> ww;

    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];

    for (int i = 0; i < 100005; i++)
        dp[i] = 1e18;

    dp[0] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 100000; j >= v[i]; j--) {
            dp[j] = min(dp[j], dp[j - v[i]] + w[i]);
        }
    }

    ll ans = 0;

    for (int i = 100000; i >= 0; i--)
        if (dp[i] <= ww)
            ans = max(ans, 1LL * i);

    cout << ans << endl;
}
```

F - LCS

最长公共子序列问题，要求输出方案。

记 $n = |s|, m = |t|$。

可以 $\mathcal{O}(nm)$ 解决。

关于输出方案，在状态转移的时候记录一下来源，然后递归输出。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
string s, t;
int dp[3005][3005];
char prre[3005][3005];
P pre[3005][3005];
int main() {
    cin >> s >> t;

    for (int i = 0; i <= s.size(); i++) {
        for (int j = 0; j <= t.size(); j++) {
            if (i) {
                if (dp[i][j] < dp[i - 1][j]) {
                    dp[i][j] = max(dp[i][j], dp[i - 1][j]);
                    pre[i][j] = mp(i - 1, j);
                }
            }

            if (j) {
                if (dp[i][j] < dp[i][j - 1]) {
                    dp[i][j] = max(dp[i][j], dp[i][j - 1]);
                    pre[i][j] = mp(i, j - 1);
                }
            }

            if (i && j) {
                if (dp[i][j] < dp[i - 1][j - 1]) {
                    dp[i][j] = max(dp[i][j], dp[i - 1][j - 1]);
                    pre[i][j] = mp(i - 1, j - 1);
                }
            }
```

```
            if (i && j && s[i - 1] == t[j - 1]) {
                if (dp[i][j] < dp[i - 1][j - 1] + 1) {
                    dp[i][j] = max(dp[i][j], dp[i - 1][j - 1] + 1);
                    pre[i][j] = mp(i - 1, j - 1);
                    prre[i][j] = s[i - 1];
                }
            }
        }
    }

    int ans = -1;
    P cur;

    for (int i = 0; i <= s.size(); i++) {
        for (int j = 0; j <= t.size(); j++) {
            if (ans < dp[i][j]) {
                ans = dp[i][j];
                cur = mp(i, j);
            }
        }
    }

    string ret = "";

    while (cur != mp(0, 0)) {
        int x = pre[cur.fi][cur.sc].fi;
        int y = pre[cur.fi][cur.sc].sc;

        if ('a' <= prre[cur.fi][cur.sc] && prre[cur.fi][cur.sc] <= 'z') {
            ret.pb(prre[cur.fi][cur.sc]);
        }

        cur = mp(x, y);
    }

    reverse(ret.begin(), ret.end());
    cout << ret << endl;
}
```

G - Longest Path

求 DAG 最长路，考虑 DAG 上 DP。

状态按照拓扑序转移，时间复杂度 $\mathcal{O}(n+m)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, m;
vector<int>edge[100005];
int dp[100005];
int rec(int x) {
    if (dp[x] >= 0)
        return dp[x];

    if (edge[x].size() == 0)
        return dp[x] = 0;

    for (int i = 0; i < edge[x].size(); i++) {
        int to = edge[x][i];
        dp[x] = max(dp[x], rec(to) + 1);
    }

    return dp[x];
}
int main() {
    cin >> n >> m;

    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        edge[a].pb(b);
    }

    for (int i = 1; i <= n; i++)
        dp[i] = -1;

    for (int i = 1; i <= n; i++) {
```

```
        rec(i);
    }

    int ans = 0;
    repn(i, n)ans = max(ans, dp[i]);
    cout << ans << endl;
}
```

H - Grid 1

记 $f_{i,j}$ 表示从 $(1,1)$ 走到 (i,j) 的方案数，初始时 $f_{1,1}=1$。

$$f_{i,j}=f_{i-1,j}+f_{i,j-1}$$

从左上至右下依次转移即可。

时间复杂度 $\mathcal{O}(hw)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int h, w;
string f[1005];
ll dp[1005][1005];
ll rec(int x, int y) {
    if (dp[x][y] >= 0)
        return dp[x][y];

    if (x == h - 1 && y == w - 1)
        return dp[x][y] = 1;

    dp[x][y] = 0;

    if (x != h - 1 && f[x + 1][y] == '.')
        dp[x][y] += rec(x + 1, y);

    if (y != w - 1 && f[x][y + 1] == '.')
        dp[x][y] += rec(x, y + 1);

    return dp[x][y] = (dp[x][y] % mod + mod) % mod;
}
int main() {
    cin >> h >> w;

    for (int i = 0; i < h; i++)
        cin >> f[i];

    rep(i, h)rep(j, w) dp[i][j] = -1;
    cout << rec(0, 0) << endl;
}
```

I - Coins

“正面朝上的银币数比反面朝上的银币数多” 这一事件包含了很多样本点，尝试统计每个样本点的概率后求和。

记 $f_{i,j}$ 表示 i 次正面向上，j 次反面向上的概率，初始时 $f_{0,0}=1$。

$$f_{i,j}=p_{i+j}\times f_{i-1,j}+(1-p_{i+j})\times f_{i,j-1}$$

答案为 $\sum_{i+j=n,i>j} f_{i,j}$。

还有另一种方法，记 $f_{i,j}$ 表示前 i 枚硬币有 j 次朝上的概率，初始时 $f_{0,0}=1$。

$$f_{i,j}=p_{i}\times f_{i-1,j-1}+(1-p_{i})\times f_{i-1,j}$$

答案为 $\sum_{j>n-j} f_{n,j}$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n;
double p[3005];
double dp[3005][3005];
int main() {
    scanf("%d", &n);
    repn(i, n) scanf("%lf", &p[i]);
    dp[0][0] = 1;

    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= i - 1; j++) {
            dp[i][j + 1] += dp[i - 1][j] * p[i];
            dp[i][j] += dp[i - 1][j] * (1.0 - p[i]);
        }
    }

    double ans = 0.0;

    for (int i = 0; i <= n; i++)
        if (i > n - i)
            ans += dp[n][i];

    printf("%.12f\n", ans);
}
```

J - Sushi

一个局面可以用各个盘子里的寿司个数表示，可是总局面数有 3^n 个，无法接受。

其实我们只关心盘子里的寿司个数而不在乎盘子具体在哪个位置。

考虑合并同类状态，将所有寿司个数相同的盘子归为一类，那么一共有 4 类，这样状态数是 n^4。

再次考虑减少状态，四类盘子的总数为 n，那么只需要记录三类盘子的数量就可以推出另一类盘子的数量，此时状态数为 n^3。

记 $f_{i,j,k}$ 表示有 1 个寿司的盘子个数为 i，有 2 个寿司的盘子个数为 j，有 3 个寿司的盘子个数为 k 的状态。

转移时一次考虑选择的是哪一类盘子，不要遗漏选中空盘的情况。

这种多维度的状态转移可以使用循环，也可以使用记忆化搜索的形式。

时间复杂度 $\mathcal{O}(n^3)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
double dp[305][305][305];
int n, a[3];
double rec(int a, int b, int c) {
    if (dp[a][b][c] >= -1e16)
        return dp[a][b][c];

    if (a + b + c == 0)
        return 0.0;

    double sum = 0;

    if (a)
        sum += (double)(a) / (double)(a + b + c) * rec(a - 1, b, c);

    if (b)
        sum += (double)(b) / (double)(a + b + c) * rec(a + 1, b - 1, c);

    if (c)
        sum += (double)(c) / (double)(a + b + c) * rec(a, b + 1, c - 1);

    sum += 1.0 * (double)(n) / (double)(a + b + c);
    return dp[a][b][c] = sum;
}
int main() {
    cin >> n;

    for (int i = 0; i < 305; i++)
        for (int j = 0; j < 305; j++)
            for (int k = 0; k < 305; k++) {
                dp[i][j][k] = -1e18;
```

```
            }

    for (int i = 0; i < 305; i++)
        for (int j = 0; j < 305; j++)
            for (int k = 0; k < 305; k++) {
                if (i + j + k > n)
                    continue;

                dp[i][j][k] = rec(i, j, k);
            }

    rep(i, n) {
        int x;
        cin >> x;
        a[x - 1]++;
    }
    printf("%.12f\n", dp[a[0]][a[1]][a[2]]);
}
```

K - Stones

博弈论。

- 一个状态存在后继状态为必败态时该状态必胜。
- 一个状态任意后继状态为必胜态是该状态必败。

记 f_i 表示有 i 个石子时该状态是否先手必胜。

从小到大枚举石子个数，依次处理每种决策的后继状态对当前状态的影响。

时间复杂度 $\mathcal{O}(nk)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, k, a[105], g[100005];
int main() {
    cin >> n >> k;

    for (int i = 1; i <= n; i++)
        cin >> a[i];

    for (int i = 1; i <= k; i++) {
        for (int j = 1; j <= n; j++) {
            if (i >= a[j] && g[i - a[j]] == 0) {
                g[i] = 1;
                break;
            }
        }
    }

    puts(g[k] ? "First" : "Second");
}
```

L - Deque

任意局面下队列中的元素一定是原队列的一个区间，考虑区间 DP。

记 $f_{i,j}$ 表示当前区间对应原队列中 $[i,j]$ 时的 $x-y$。

- 先手希望最大化 x，即最大化 $x-y$。
- 后手希望最大化 y，即最小化 $x-y$。

当前局面由先手取数还是后手可以根据区间长度奇偶性判断。

由于只能从两段取，$f_{i,j}$ 只能从 $f_{i+1,j}, f_{i,j-1}$ 转移过来。

转移时根据先后手取最大最小值。

时间复杂度 $\mathcal{O}(n^2)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
ll dp[3005][3005][2], a[3005];
int n;
ll rec(int L, int R, int p) {
    if (dp[L][R][p] >= -1e15)
        return dp[L][R][p];

    if (L > R)
        return 0LL;

    if (p == 0) {
        ll x = -1e18;
        x = max(x, a[L] + rec(L + 1, R, 1 - p));
        x = max(x, a[R] + rec(L, R - 1, 1 - p));
        return dp[L][R][p] = x;
    } else {
        ll x = 1e18;
        x = min(x, -a[L] + rec(L + 1, R, 1 - p));
        x = min(x, -a[R] + rec(L, R - 1, 1 - p));
        return dp[L][R][p] = x;
    }
}
int main() {
    cin >> n;

    for (int i = 1; i <= n; i++)
        cin >> a[i];

    rep(i, 3005)rep(j, 3005)rep(k, 2) dp[i][j][k] = -1e18;
    cout << rec(1, n, 0) << endl;
}
```

M - Candies

记 $f_{i,j}$ 表示前 i 个人共分得 j 颗糖果的方案数。

$$f_{i,j}=\sum_{0\le k\le a_i} f_{i-1,j-k}$$

一次转移转移需要 $\mathcal{O}(k)$，总时间 $\mathcal{O}(nk^2)$，考虑优化。

不难发现对于一个 j，其求和的对象为 $[j-a_i,j]$，且这段区间属于上一层状态，这样静态的区间信息可以使用前缀和优化。

时间复杂度 $\mathcal{O}(nk)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, k, a[105];
ll dp[105][100005];
int main() {
    cin >> n >> k;

    for (int i = 1; i <= n; i++)
        cin >> a[i];

    dp[0][0] = 1;

    for (int i = 1; i <= n; i++) {
        ll pre = 0;

        for (int j = 0; j <= k; j++) {
            //dp[i][j] <- dp[i-1][j-a[i]]....dp[i-1][j]
            pre += dp[i - 1][j];

            if (j - a[i] - 1 >= 0)
                pre -= dp[i - 1][j - a[i] - 1];

            pre = (pre % mod + mod) % mod;
            dp[i][j] = pre;
        }
    }

    cout << dp[n][k] << endl;
}
```

N - Slimes

石子合并问题，直接区间 DP。

记 $f_{i,j}$ 表示将 $[i,j]$ 合并成一堆的最小代价。

$$f_{i,j} = \min_{i \le k < j} f_{i,k} + f_{k+1,j}$$

时间复杂度 $\mathcal{O}(n^3)$。

数据范围再大点可以使用决策单调性（四边形不等式）优化。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, a[405];
ll dp[405][405];
ll rui[405];
int main() {
    cin >> n;

    for (int i = 1; i <= n; i++)
        cin >> a[i];

    for (int i = 1; i <= n; i++)
        rui[i] = rui[i - 1] + a[i];

    for (int i = 1; i <= n; i++)
        dp[i][i] = 0;

    for (int a = 2; a <= n; a++) {
        for (int i = 1; i + a - 1 <= n; i++) {
            //dp[i][i+a-1]
            dp[i][i + a - 1] = 1e18;

            for (int j = i; j < i + a - 1; j++) {
                dp[i][i + a - 1] = min(dp[i][i + a - 1], dp[i][j] + dp[j + 1][i + a - 1
            }
        }
    }

    cout << dp[1][n] << endl;
}
```

O - Matching

$n \leq 21$ 考虑状压。

单压一个集合就有 $2^n = 2097152$ 个状态，两个集合都压肯定压不下。

由于求完备匹配数，一个元素就算现在不匹配早晚都得匹配。

于是记 $f_{i,j}$ 表示一个集合用前 i 个元素匹配了另一个集合的元素，且另一个集合已匹配的元素压缩状态表示为 j 的方案数。

枚举 i, j 需要 $\mathcal{O}(n2^n)$，一次转移需要 $\mathcal{O}(n)$，总时间复杂度 $\mathcal{O}(n^2 2^n)$，考虑优化。

当且仅当 $i = \operatorname{popcount}(j)$ 时一个状态是合法的。

- 若先枚举 i，则只需枚举 $\operatorname{popcount}(j) = i$ 的 j，符合条件的状态 j 可以预处理。
- 若先枚举 j，则无需枚举 i，令 $i = \operatorname{popcount}(j)$ 即可。

时间复杂度 $\mathcal{O}(n2^n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, a[21][21];
ll dp[(1 << 21)];
int x[21];
int main() {
    cin >> n;
    rep(i, n) {
        rep(j, n) {
            cin >> a[i][j];
            x[i] += a[i][j] * (1 << j);
        }
    }
    dp[0] = 1;
    rep(i, n) {
        for (int j = 0; j < (1 << n); j++) {
            if (__builtin_popcount(j) != i)
                continue;

            for (int xx = 0; xx < n; xx++) {
                if (a[i][xx] && !((j >> xx) & 1)) {
                    dp[j + (1 << xx)] += dp[j];

                    if (dp[j + (1 << xx)] >= mod)
                        dp[j + (1 << xx)] -= mod;
                }
            }
        }
    }
    cout << dp[(1 << n) - 1] % mod << endl;
}
```

P - Independent Set

树上独立集计数问题。

$f_{u,0/1}$ 表示 u 的子树内 u 染为白/黑的合法方案数，初识时 $f_{u,0/1}=1$。

$$f_{u,0}=\prod_{v\in son_u}(f_{v,0}+f_{v,1})$$
$$f_{u,1}=\prod_{v\in son_u}f_{v,0}$$

时间复杂度 $\mathcal{O}(n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n;
ll dp[100005][2];
vector<int>edge[100005];
void dfs(int v, int u) {
    dp[v][0] = 1;
    dp[v][1] = 1;

    for (int i = 0; i < edge[v].size(); i++) {
        int to = edge[v][i];

        if (u == to)
            continue;

        dfs(to, v);
        dp[v][0] = dp[v][0] * (dp[to][0] + dp[to][1]) % mod;
        dp[v][1] = dp[v][1] * dp[to][0] % mod;
    }
}
int main() {
    cin >> n;
    rep(i, n - 1) {
        int a, b;
        cin >> a >> b;
        edge[a].pb(b);
        edge[b].pb(a);
    }
    dfs(1, -1);
    ll ans = dp[1][0] + dp[1][1];
    cout << (ans % mod + mod) % mod << endl;
```

```
}
```

Q - Flowers

记 f_i 表示考虑前 i 枝花，且第 i 枝花不拿走的最大权值。

由于第 i 枝花不拿走，需要保证前一枝不拿走的花 j 满足 $h_j < h_i$。

直接枚举的话需要 $\mathcal{O}(n^2)$。

对于一枝花 i，符合条件的 j 的 h_j 在权值序列上为一个前缀，因此可以开一棵权值树状数组或权值线段树维护。

时间复杂度 $\mathcal{O}(n \log n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, h[200005];
ll a[200005];;
struct RMQ {
#define s (1<<20)
    ll seg[s];
    void init() {
        fill(seg, seg + s, -1e18);
    }
    void update(int k, ll a) {
        k += s / 2 - 1;
        seg[k] = a;

        while (k > 0) {
            k = (k - 1) / 2;
            seg[k] = max(seg[k * 2 + 1], seg[k * 2 + 2]);
        }
    }
    ll query(int a, int b, int k, int l, int r) {
        if (r < a || b < l)
            return -1e18;

        if (a <= l && r <= b)
            return seg[k];
        else {
            ll vl = query(a, b, k * 2 + 1, l, (l + r) / 2);
            ll vr = query(a, b, k * 2 + 2, (l + r) / 2 + 1, r);
            return max(vl, vr);
        }
    }
} rmq;
```

```
int main() {
    cin >> n;

    for (int i = 1; i <= n; i++)
        scanf("%d", &h[i]);

    for (int i = 1; i <= n; i++)
        scanf("%lld", &a[i]);

    rmq.init();
    rmq.update(0, 0);
    ll ans = 0;

    for (int i = 1; i <= n; i++) {
        ll v = rmq.query(0, h[i] - 1, 0, 0, s / 2 - 1);
        rmq.update(h[i], v + a[i]);
        ans = max(ans, v + a[i]);
    }

    cout << ans << endl;
}
```

R - Walk

记 $f_{i,u}$ 表示长度为 i 终点为 u 的路径数量。

$$f_{i,v} = \sum_{(u,v)\in E} f_{i-1,u}$$

答案为 $\sum_{u=1}^{n} f_{k,u}$。

时间复杂度 $\mathcal{O}(nk)$，但是 $k \leq 10^{18}$ 直接爆炸。

虽然 k 很大但是 n 很小只有 50，可以使用矩阵加速。

将邻接矩阵作为转移矩阵作矩阵快速幂，然后用 f_0 与之作向量乘矩阵得到答案。

其实 f_0 初值为 1 可以省略的。

时间复杂度 $\mathcal{O}(n^3 \log k)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
ll dp[55][55][65];
int a[55][55], n;
ll v;
ll ans[55][65];
int main() {
    cin >> n >> v;
    rep(i, n) rep(j, n) {
        cin >> a[i][j];

        if (a[i][j] == 1)
            dp[i][j][0] = 1;
    }

    for (int x = 0; x < 60; x++) {
        rep(i, n)rep(j, n)rep(k, n) {
            dp[i][j][x + 1] += dp[i][k][x] * dp[k][j][x] % mod;

            if (dp[i][j][x + 1] >= mod)
                dp[i][j][x + 1] %= mod ;
        }
    }

    rep(i, n)ans[i][0] = 1;
    int cur = 0;
    rep(x, 61) {
        if (((v >> x) & 1LL)) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++) {
                    ans[j][cur + 1] += ans[i][cur] * dp[i][j][x] % mod;
                    ans[j][cur + 1] %= mod;
```

```
                }

            cur++;
        }
    }
    ll ret = 0;
    rep(i, n) {
        ret += ans[i][cur];
        ret %= mod;
    }
    cout << (ret % mod + mod) % mod << endl;
}
```

S - Digit Sum

统计一定范围内符合条件的数字数量显然用数位 DP，但是要求数字是 d 的倍数不太好搞。

于是扩展状态表示，把模 d 的余数也加进去就行了。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
string str;
int d;
ll dp[10005][105][2];
int main() {
    cin >> str;
    cin >> d;
    dp[0][0][0] = 1;

    for (int i = 0; i < str.size(); i++) {
        int pos = str[i] - '0';

        for (int j = 0; j < d; j++) {
            for (int a = 0; a <= pos; a++) {
                if (a == pos)
                    dp[i + 1][(j + a) % d][0] += dp[i][j][0];
                else
                    dp[i + 1][(j + a) % d][1] += dp[i][j][0];
            }

            for (int a = 0; a <= 9; a++) {
                dp[i + 1][(j + a) % d][1] += dp[i][j][1];
            }
        }

        rep(j, d)rep(k, 2)dp[i + 1][j][k] %= mod;
    }

    ll ans = -1LL + dp[str.size()][0][0] + dp[str.size()][0][1];
    cout << (ans % mod + mod) % mod << endl;
}
```

T - Permutation

只关心元素的大小关系而不关心元素具体是多少，因此可以记录元素在子集内的相对大小。

$f_{i,j}$ 表示 i 在前 i 个数中相对大小为 j 的方案数。

- 若 $p_{i-1} < p_i$，$f_{i,j} = \sum_{k=1}^{j-1} f_{i-1,k}$。
- 若 $p_{i-1} > p_i$，$f_{i,j} = \sum_{k=j+1}^{i-1} f_{i-1,k}$。

使用前缀和优化 $\mathcal{O}(n^3)$ 到 $\mathcal{O}(n^2)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n;
string str;
ll dp[3005][3005];
ll rui[3005][3005];
int main() {
    cin >> n >> str;
    dp[1][1] = 1;

    for (int i = 1; i <= n; i++)
        rui[1][i] = 1;

    for (int i = 2; i <= n; i++) {
        if (str[i - 2] == '<') {
            for (int j = 1; j <= i; j++) {
                //dp[i][j]
                dp[i][j] = rui[i - 1][j - 1];
                //  for(int x=1;x<j;x++) dp[i][j] += dp[i-1][x];
                //  dp[i][j]%=mod;
            }
        } else {
            for (int j = 1; j <= i; j++) {
                //dp[i][j]
                dp[i][j] = rui[i - 1][i - 1] - rui[i - 1][j - 1];
                dp[i][j] = (dp[i][j] % mod + mod) % mod;
                //for(int x=j;x<=i-1;x++) dp[i][j] += dp[i-1][x];
                //dp[i][j]%=mod;
            }
        }

        for (int x = 1; x <= i; x++)
```

```
            rui[i][x] = (rui[i][x - 1] + dp[i][x]) % mod;
    }

    ll ans = 0;

    for (int i = 1; i <= n; i++)
        ans += dp[n][i];

    cout << (ans % mod + mod) % mod << endl;
}
```

U - Grouping

n 只有 16，考虑状压。

预处理出所有分组状态的得分。

f_i 表示 i 中元素分为若干组的最大得分。

枚举已分组状态和新分组状态，若两者交集为空则可以合并。

直接合并需要 $\mathcal{O}(4^n)$，考虑枚举当前状态补集的子集，总共需要 $\mathcal{O}(3^n)$。

时间复杂度 $\mathcal{O}(2^n n^2 + 3^n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, a[16][16];
ll sum[(1 << 16)];
ll dp[(1 << 16)];
int main() {
    cin >> n;
    rep(i, n)rep(j, n)cin >> a[i][j];

    for (int i = 0; i < (1 << n); i++) {

        rep(x, n) for (int y = x + 1; y < n; y++) {
            if (((i >> x) & 1) + ((i >> y) & 1) == 2) {
                sum[i] += a[x][y];
            }
        }
    }

    rep(i, (1 << n)) dp[i] = -1e18;
    dp[0] = 0;

    for (int i = 1; i < (1 << n); i++) {
        int M = i;
        int cur = (i - 1)&M;

        while (1) {
            dp[i] = max(dp[i], dp[cur] + sum[i - cur]);

            if (cur == 0)
                break;

            cur = (cur - 1)&M;
```

```
            }
        }

        cout << dp[(1 << n) - 1] << endl;
    }
```

V - Subtree

只考虑一个根的情况下怎么做。

记 f_i 表示 i 的子树内方案数。

$$f_u = \prod_{v \in son_u} (f[v] + 1)$$

考虑以其它点为根的情况，n 次 DP 一共需要 $\mathcal{O}(n^2)$，考虑换根 DP。

在以 u 为根时，u 子树内的情况是不变的，但是要考虑 u 子树外的情况。

记 g_u 表示不考虑 u 子树时 u 必选的方案数，则有 $ans_u = f_u \times g_u$，考虑如何换根求 g_u。

对于一个点 u，首先可以 g_u 继承 g_{fa_u} 再考虑 u 的兄弟节点。

$$g_u = g_{fa_u} \times \prod_{v \in son(fa_u), v \neq u} (f_v + 1) + 1$$

时间复杂度 $\mathcal{O}(n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
//#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n;
ll mod;
ll dp[100005];
vector<int>edge[100005];
ll ans[100005];
void dfs(int v, int u) {
    dp[v] = 1;

    for (int i = 0; i < edge[v].size(); i++) {
        int to = edge[v][i];

        if (to == u)
            continue;

        dfs(to, v);
        dp[v] = dp[v] * dp[to] % mod;
    }

    dp[v] = (dp[v] + 1LL) % mod;
}
void dfs2(int v, int u, ll um) {
    ans[v] = (dp[v] + mod - 1) % mod * um % mod;
    vector<ll>vec, vec2;
    ll x = 1LL;

    for (int i = 0; i < edge[v].size(); i++) {
        int to = edge[v][i];

        if (to == u)
            continue;
```

```
        x = x * dp[to] % mod;
        vec.pb(x);
    }

    x = 1LL;

    for (int i = 0; i < edge[v].size(); i++) {
        int to = edge[v][edge[v].size() - 1 - i];

        if (to == u)
            continue;

        x = x * dp[to] % mod;
        vec2.pb(x);
    }

    int num = 0;

    for (int i = 0; i < edge[v].size(); i++) {
        int to = edge[v][i];

        if (to == u)
            continue;

        ll vvv = um;

        if (num && num <= vec.size())
            vvv = vvv * vec[num - 1] % mod;

        int sz = vec2.size();

        if (sz - 2 - num >= 0)
            vvv = vvv * vec2[sz - 2 - num] % mod;

        vvv = (vvv + 1LL) % mod;
        dfs2(to, v, vvv);
        num++;
    }
}
int main() {
    scanf("%d%lld", &n, &mod);

    for (int i = 1; i < n; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        edge[a].pb(b);
        edge[b].pb(a);
    }

    dfs(1, -1);
```

```
    dfs2(1, -1, 1);

    for (int i = 1; i <= n; i++)
        printf("%lld\n", (ans[i] % mod + mod) % mod);
}
```

W - Intervals

记 $f_{i,j}$ 表示前 i 个位置，上一个 1 在 j 的最大得分。

$$f_{i,j} = \begin{cases} \max_{k<i} f_{i-1,k} & j = i \\ f_{i-1,j} + \sum_{l_k \le j, r_k \, i} a_k & j < i \end{cases}$$

分层 DP 可以压掉一维，但时间还是 $\mathcal{O}(n^2)$。

- 对于 $j = i$ 就是区间最大值
- 对于 $j < i$ 就是区间加

可以使用线段树维护。

时间复杂度 $\mathcal{O}(n \log n)$。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n, m;
vector<P1>query[200005];
class segtree {
public:
#define s (1<<18)
    ll seg[s * 2];
    ll lazy[s * 2];
    void lazy_evaluate(int k) {
        if (k * 2 + 2 >= s * 2)
            return ;

        lazy[k * 2 + 2] += lazy[k];
        lazy[k * 2 + 1] += lazy[k];
        seg[k * 2 + 2] += lazy[k];
        seg[k * 2 + 1] += lazy[k];
        lazy[k] = 0;
    }
    ll update(int beg, int end, int idx, int lb, int ub, ll num) {
        if (ub < beg || end < lb) {
            return seg[idx];
        }

        if (beg <= lb && ub <= end) {
            lazy[idx] += num;
            seg[idx] += num;
            return seg[idx];
        }

        if (lazy[idx]) {
            lazy_evaluate(idx);
```

```
        }

        return seg[idx] = max(update(beg, end, idx * 2 + 1, lb, (lb + ub) / 2, num), upd
                              (lb + ub) / 2 + 1, ub, num));
    }
    ll query(int beg, int end, int idx, int lb, int ub) {
        if (ub < beg || end < lb) {
            return -1000000000000000000LL;
        }

        if (beg <= lb && ub <= end) {
            return seg[idx];
        }

        if (lazy[idx]) {
            lazy_evaluate(idx);
        }

        return max(query(beg, end, idx * 2 + 1, lb, (lb + ub) / 2), query(beg, end, idx
                   ub));
    }
} kaede;
//Kaede Takagaki is my wife!
int main() {
    scanf("%d%d", &n, &m);

    for (int i = 0; i < m; i++) {
        int L, R, a;
        scanf("%d%d%d", &L, &R, &a);
        query[L].pb(mp(1, mp(R, a)));
        query[R + 1].pb(mp(-1, mp(L, a)));
    }

    ll ans = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 0; j < query[i].size(); j++) {
            int ty = query[i][j].fi;
            int za = query[i][j].sc.fi;
            int cs = query[i][j].sc.sc;

            if (ty == 1) {
                kaede.update(0, i - 1, 0, 0, s - 1, 1LL * cs);
            } else {
                kaede.update(0, za - 1, 0, 0, s - 1, -1LL * cs);
            }
        }

        ll v = kaede.query(0, i - 1, 0, 0, s - 1);
        ans = max(ans, v);
        kaede.update(i, i, 0, 0, s - 1, v);
```

```
    }

    cout << ans << endl;
}
```

X - Tower

只有一个堆，考虑贪心。

对于 i, j：

- 如果 i 放 j 上面，那么 i 上面还能放 $s_j - w_i$。
- 如果 j 放 i 上面，那么 j 上面还能放 $s_i - w_j$。

综上，如果 i 放 j 上面，应有 $s_j - w_i > s_i - w_j$，移项得 $s_i + w_i < s_j + w_j$。

因此按 $s_i + w_i$ 排序作 01 背包。

$$f_{j+w_i} = \max(f_{j+w_i}, f_j + v_i)$$

其中 $0 \le j \le s_i$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int n;
vector<pair<P, int>>vec;
ll dp[1005][20005];
bool cmp(const pair<P, int> &a, const pair<P, int> &b) {
    return a.fi.fi + a.fi.sc < b.fi.fi + b.fi.sc;
}
ll v[1005];
int main() {
    cin >> n;

    for (int i = 0; i < n; i++) {
        int a, b;
        cin >> a >> b >> v[i];
        vec.pb(mp(mp(a, b), i));
    }

    sort(vec.begin(), vec.end(), cmp);
    rep(a, 1005)rep(b, 20005)dp[a][b] = -1e18;
    dp[0][0] = 0;
    ll ans = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < 20005 - vec[i].fi.fi; j++) {
            dp[i + 1][j] = max(dp[i + 1][j], dp[i][j]);
            ans = max(ans, dp[i + 1][j]);

            if (j > vec[i].fi.sc)
                continue;

            dp[i + 1][j + vec[i].fi.fi] = max(dp[i + 1][j + vec[i].fi.fi], dp[i][j] + v
```

```
            ans = max(ans, dp[i + 1][j + vec[i].fi.fi]);
        }
    }

    cout << ans << endl;
}
```

Y - Grid 2

h, w 很大，不能再 $\mathcal{O}(hw)$ DP 了。

其实在不考虑不能经过的点的情况下，从 (x_1, y_1) 走到 (x_2, y_2) 的方案数就是 $calc(x_1, y_1, x_2, y_2) = \mathcal{C}_{|x_1-x_2|+|y_1-y_2|}^{|x_1-x_2|}$。

再将不能经过的点考虑进来，就是所有方案减去经过不能经过的点的方案。

从 n 入手，f_i 表示从 $(1, 1)$ 出发不经过其它不能经过的点到达一个不能经过的点 (x_i, y_i) 的方案数。

$f_i = calc(1, 1, x_i, y_i) - \sum_{j=1}^{i-1} [x_j \le x_i][y_j \le y_i] \times f_j \times calc(x_j, y_j, x_i, y_i)$

答案为 $calc(1, 1, h, 2) - \sum_{i=1^n} f_i \times calc(x_i, x_j, h, w)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
typedef pair<int, P> P1;
typedef pair<P, P> P2;
#define pu push
#define pb push_back
#define mp make_pair
#define eps 1e-7
#define INF 1000000000
#define mod 1000000007
#define fi first
#define sc second
#define rep(i,x) for(int i=0;i<x;i++)
#define repn(i,x) for(int i=1;i<=x;i++)
#define SORT(x) sort(x.begin(),x.end())
#define ERASE(x) x.erase(unique(x.begin(),x.end()),x.end())
#define POSL(x,v) (lower_bound(x.begin(),x.end(),v)-x.begin())
#define POSU(x,v) (upper_bound(x.begin(),x.end(),v)-x.begin())
int h, w, n;
vector<P>vec;
ll dp[3005];
ll modpow(ll x, ll n) {
    ll res = 1;

    while (n > 0) {
        if (n & 1)
            res = res * x % mod;

        x = x * x % mod;
        n >>= 1;
    }

    return res;
}
ll F[200005], R[200005];
void make() {
    F[0] = 1;

    for (int i = 1; i < 200005; i++)
        F[i] = F[i - 1] * i % mod;

    for (int i = 0; i < 200005; i++)
        R[i] = modpow(F[i], mod - 2);
}
ll C(int a, int b) {
    return F[a] * R[b] % mod * R[a - b] % mod;
}
int main() {
```

```
    cin >> h >> w >> n;
    rep(i, n) {
        int x, y;
        cin >> x >> y;
        vec.pb(mp(x, y));
    }
    SORT(vec);
    make();
    ll ans = C(h - 1 + w - 1, w - 1);
    rep(i, n) {
        int x = vec[i].fi, y = vec[i].sc;
        dp[i] = C(x - 1 + y - 1, y - 1);
        rep(j, i) {
            int xx = vec[j].fi, yy = vec[j].sc;

            if (xx <= x && yy <= y) {
                dp[i] -= dp[j] * C(x - xx + y - yy, y - yy) % mod;
            }
        }
        dp[i] %= mod;
        ans -= dp[i] * C(h - x + w - y, w - y) % mod;
    }
    cout << (ans % mod + mod) % mod << endl;
}
```

Z - Frog 3

记 f_i 表示跳到第 i 块石头的最小花费。

$$
\begin{aligned}
f_i &= f_j + (h_i - h_j)^2 + c \\
f_i &= f_j + h_i^2 + h_j^2 - 2h_ih_j + c \\
f_j + h_j^2 &= 2h_ih_j + f_i - h_i^2 - c
\end{aligned}
$$

令 $x_i = h_i,\ y_i = f_i + h_i^2,\ k_i = 2 \times h_i,\ b_i = f_i - h_i^2 - c,$ 使用斜率优化。

```cpp
#include <iostream>
#include <deque>

typedef long long lxl;
typedef double dbl;
typedef std::deque<int> diq;

const int maxN = 2e5;

lxl n, c;
lxl h[maxN + 10];
lxl f[maxN + 10];
diq q;

lxl x(int i) {
    return h[i];
}

lxl y(int i) {
    return f[i] + h[i] * h[i];
}

dbl slope(int a, int b) {
    dbl dx = x(b) - x(a);
    dbl dy = y(b) - y(a);
    return dy / dx;
}

int main() {
    q.push_back(1);
    std::cin >> n >> c;
    for (int i = 1; i <= n; i++) std::cin >> h[i];
    for (int i = 2; i <= n; i++) {
        int l = 0, r = q.size() - 1;
        while (l < r) {
            int mid = (l + r) / 2;
            if (slope(q[mid], q[mid + 1]) >= 2 * h[i]) r = mid;
            else l = mid + 1;
        }
        int j = q[l];
        f[i] = f[j] + (h[i] - h[j]) * (h[i] - h[j]) + c;
        while (q.size() > 1 && slope(q[q.size() - 2], q[q.size() - 1]) >= slope(q[q.siz
        q.push_back(i);
    }
    std::cout << f[n] << '\n';
    return 0;
}
```