

Educational DP Contest @ AtCoder

Link to the contest: <https://atcoder.jp/contests/dp>

- ✓ A - Frog 1
- ✓ B - Frog 2
- ✓ C - Vacation
- ✓ D - Knapsack 1
- ✓ E - Knapsack 2
- ✓ F - LCS - (O(n^2) - minimum Levensteins distance)
- ✓ G - Longest Path - O(n)
- ✓ H - Grid 1 - O(n^2) - Pascal Triangle
- ✓ I - Coins - O(n^2)
- ✓ J - Sushi - 3 dimensional DP with compression
- ✓ K - Stones - Binary player tracking
- ✓ L - Deque - Interval DP
- ✓ M - Candies - Keeping prefix sums
- ✓ N - Slimes - Interval DP O(n^3) with prefix sums
- ✓ O - Matching - O($2^n * n$)
- ✓ P - Independent Set - Tree DP
- ✓ Q - Flowers - Segment Tree DP
- ✓ R - Walk - Matrix Expo (my fav)
- ✓ S - Digit Sum - Digit DP
- ✓ T - Permutation - Keeping prefix sums
- ✓ U - Grouping - DP on masks
- ✓ V - Subtree - DFS In and Out
- ✓ W - Intervals
- ✓ X - Tower - Exchange Principle
- ✓ Y - Grid 2 - Binomial Coefficients
- ✓ Z - Frog 3 - Convex Hull Trick

```
//  
// There are N stones, numbered 1,2,...,N. For each i (1≤i≤N), the height of Stone i is h[i]  
// There is a frog who is initially on Stone 1. He will repeat the following action  
// some number of times to reach Stone N:  
//   - If the frog is currently on Stone i, jump to Stone i+1 or Stone i+2.  
//     Here, a cost of | h[i] - h[j] | is incurred, where j is the stone to land on.  
// Find the minimum possible total cost incurred before the frog reaches Stone N.  
//  
// Time Complexity: O(N)  
  
#include <bits/stdc++.h>  
  
using namespace std;  
  
int main() {  
    int n;  
    cin >> n;  
  
    vector<int> heights(n);  
    for (int i = 0; i < n; i++) {  
        cin >> heights[i];  
    }  
  
    vector<int> dp(n); // dp[i] = minimum cost of getting to i-th stone from the first  
    dp[0] = 0;  
  
    // O(n)  
    for (int i = 1; i < n; i++) {  
        // what is the minimum cost of getting to the previous stone + abs(diff)  
        int cur_ans = dp[i-1] + abs(heights[i] - heights[i-1]);  
        if (i - 2 ≥ 0) {  
            cur_ans = min(cur_ans, dp[i-2] + abs(heights[i] - heights[i-2]));  
        }  
  
        dp[i] = cur_ans;  
    }  
  
    cout << dp[n-1] << endl;  
    return 0;  
}
```

```
1 //  
2 // There are N stones, numbered 1,2,...,N. For each i (1≤i≤N), the height of Stone i is h  
3 // There is a frog who is initially on Stone 1. He will repeat the following action  
4 // some number of times to reach Stone N:  
5 //     - If the frog is currently on Stone i, jump to one of the following: Stone i+1, i+2 ... i+K.  
6 //         Here, a cost of | h[i] - h[j] | is incurred, where j is the stone to land on.  
7 // Find the minimum possible total cost incurred before the frog reaches Stone N.  
8 //  
9 // Time Complexity: O(NK)  
10 //  
11  
12 #include <bits/stdc++.h>  
13 using namespace std;  
14  
15 int main() {  
16     int n, K;  
17     cin >> n >> K;  
18  
19     vector<int> heights(n);  
20     for (int i = 0; i < n; i++) {  
21         cin >> heights[i];  
22     }  
23  
24     vector<int> dp(n);  
25     dp[0] = 0;  
26     for (int i = 1; i < n; i++) {  
27         int cur_ans = dp[i-1] + abs(heights[i] - heights[i-1]);  
28         for (int k = 2; k ≤ K; k++) {  
29             if (i - k ≥ 0) {  
30                 cur_ans = min(cur_ans, dp[i-k] + abs(heights[i] - heights[i-k]));  
31             }  
32         }  
33         dp[i] = cur_ans;  
34     }  
35  
36     cout << dp[n-1] << endl;  
37     return 0;  
38 }
```

```
1 //  
2 // Taro's summer vacation starts tomorrow, and he has decided to make plans for it now.  
3 // The vacation consists of N days. For each i (1≤i≤N),  
4 // Taro will choose one of the following activities and do it on the i-th day:  
5 //   A: Swim in the sea. Gain a[i] points of happiness.  
6 //   B: Catch bugs in the mountains. Gain b[i] points of happiness.  
7 //   C: Do homework at home. Gain c[i] points of happiness.  
8 // As Taro gets bored easily, he cannot do the same activities for two or more consecutive days.  
9 //  
10 // Find the maximum possible total points of happiness that Taro gains.  
11 //  
12 // Time Complexity: O(N)  
13 //  
14  
15 #include <bits/stdc++.h>  
16  
17 using namespace std;  
18  
19 int main() {  
20     int n;  
21     cin >> n;  
22  
23     vector<vector<int>> happy(n, vector<int>(3));  
24     for (int i = 0; i < n; i++) {  
25         for (int k = 0; k < 3; k++) {  
26             scanf("%d", &happy[i][k]);  
27         }  
28     }  
29  
30     vector<vector<int>> dp(n, vector<int>(3, 0));  
31  
32     for (int k = 0; k < 3; k++) {  
33         dp[0][k] = happy[0][k];  
34     }  
35  
36     for (int i = 1; i < n; i++) {  
37         for (int k = 0; k < 3; k++) {  
38             int ans = 0;  
39  
40             for (int l = 0; l < 3; l++) {  
41                 if (l ≠ k) ans = max(ans, dp[i-1][l]);  
42             }  
43  
44             dp[i][k] = happy[i][k] + ans;  
45         }  
46     }  
47  
48     int ans = 0;  
49     for (int i = 0; i < 3; i++) {  
50         ans = max(ans, dp[n-1][i]);  
51     }  
52  
53     cout << ans << endl;  
54     return 0;  
55 }
```

```
1 //  
2 // There are N items, numbered 1,2, ... N. For each i (1 ≤ i ≤ N),  
3 // Item i has a weight of w[i] and a value of v[i]  
4 // Taro has decided to choose some of the N items and carry them home in a knapsack.  
5 // The capacity of the knapsack is W, which means that the sum of the weights  
6 // of items taken must be at most W.  
7 //  
8 // Find the maximum possible sum of the values of items that Taro takes home.  
9 //  
10 // Constraints:  
11 // 1 ≤ N ≤ 100  
12 // 1 ≤ W ≤ 10^5  
13 // 1 ≤ w[i] ≤ W  
14 // 1 ≤ v[i] ≤ 10^9  
15 //  
16 // Time Complexity: O(NW)  
17 //  
18  
19  
20 #include <bits/stdc++.h>  
21 #define ll long long  
22  
23 using namespace std;  
24  
25 int main() {  
26     int n, W;  
27     cin >> n >> W;  
28  
29     vector<vector<ll>> dp(n+1, vector<ll>(W+1, 0));  
30     vector<int> weights(n+1), values(n+1);  
31  
32     for (int i = 1; i ≤ n; i++) {  
33         cin >> weights[i] >> values[i];  
34     }  
35  
36     for (int i = 0; i ≤ n; i++) {  
37         for (int w = 0; w ≤ W; w++) {  
38             if (i == 0 || w == 0) dp[i][w] = 0;  
39             else if (w - weights[i] ≥ 0) {  
40                 // you can take that item  
41                 dp[i][w] = max(dp[i-1][w], dp[i-1][w-weights[i]] + values[i]);  
42             } else {  
43                 dp[i][w] = dp[i-1][w];  
44             }  
45         }  
46     }  
47  
48     cout << dp[n][W] << endl;  
49     return 0;  
50 }
```

```
1 //  
2 // There are N items, numbered 1,2, ... N. For each i (1 ≤ i ≤ N),  
3 // Item i has a weight of w[i] and a value of v[i]  
4 // Taro has decided to choose some of the N items and carry them home in a knapsack.  
5 // The capacity of the knapsack is W, which means that the sum of the weights  
6 // of items taken must be at most W.  
7 //  
8 // Find the maximum possible sum of the values of items that Taro takes home.  
9 //  
10 // Constraints:  
11 // 1 ≤ N ≤ 100  
12 // 1 ≤ W ≤ 10^9  
13 // 1 ≤ w[i] ≤ W  
14 // 1 ≤ v[i] ≤ 10^3  
15 //  
16 // Time Complexity: O(N * max(v[i]))  
17 //  
18 //  
19 #include <bits/stdc++.h>  
20 #define ll long long  
21  
22 using namespace std;  
23  
24 int N, W;  
25 vector<int> weights, values;  
26 vector<vector<ll>> dp(101, vector<ll>(100001, -1));  
27  
28 // returns the weight  
29 ll capacity(int i, int value) {  
30     if (value ≤ 0) return 0;  
31     if (i == N) return 1e12;  
32     if (dp[i][value] ≠ -1) return dp[i][value];  
33  
34     dp[i][value] = min(  
35         capacity(i+1, value),  
36         capacity(i+1, value - values[i]) + weights[i]  
37     );  
38  
39     return dp[i][value];  
40 }  
41  
42 int main() {  
43     cin >> N >> W;  
44  
45     weights.resize(N);  
46     values.resize(N);  
47  
48     int sum_values = 0;  
49     for (int i = 0; i < N; i++) {  
50         cin >> weights[i] >> values[i];  
51         sum_values += values[i];  
52     }  
53  
54     for (int value = sum_values; value ≥ 1; value--) {  
55         if (capacity(0, value) ≤ W) {  
56             cout << value << endl;  
57             return 0;  
58         }  
59     }  
60  
61     cout << 0 << endl;  
62     return 0;  
63 }
```

```
1 //  
2 // You are given strings s and t.  
3 // Find one longest string that is a subsequence (non-continuous) of both s and t.  
4 //  
5 // Time Complexity: O(NM)  
6 //  
7  
8 #include <bits/stdc++.h>  
9  
10 using namespace std;  
11  
12 int main() {  
13     string s, t;  
14     cin >> s >> t;  
15  
16     int n = (int)s.size(), m = (int)t.size();  
17  
18     vector<vector<int>> dp(n, vector<int>(m, 0));  
19  
20     dp[0][0] = (s[0] == t[0]);  
21     for (int i = 1; i < n; i++) dp[i][0] = max(dp[i-1][0], (int)(s[i] == t[0]));  
22     for (int i = 1; i < m; i++) dp[0][i] = max(dp[0][i-1], (int)(s[0] == t[i]));  
23  
24     for (int i = 1; i < n; i++) {  
25         for (int j = 1; j < m; j++) {  
26             dp[i][j] = max({  
27                 dp[i-1][j-1] + (s[i] == t[j]),  
28                 dp[i-1][j],  
29                 dp[i][j-1]  
30             });  
31         }  
32     }  
33  
34     // this is the length of the LCS  
35     // cout << dp[n-1][m-1] << endl;  
36  
37     if (dp[n-1][m-1] == 0) {  
38         cout << "" << endl;  
39         return 0;  
40     }  
41  
42     int i = n-1;  
43     int j = m-1;  
44  
45     string ans = "";  
46     while (i >= 0 && j >= 0) {  
47         if (s[i] == t[j]) {  
48             ans += s[i];  
49             i--; j--;  
50             continue;  
51         }  
52  
53         if (i == 0) j--;  
54         else if (j == 0) i--;  
55         else {  
56             if (dp[i-1][j] > dp[i][j-1]) i--;  
57             else j--;  
58         }  
59     }  
60  
61     reverse(ans.begin(), ans.end());  
62     cout << ans << endl;  
63     return 0;  
64 }
```

```
1 //  
2 // Given DAG, find the length of the longest path in G.  
3 //  
4 // Time Complexity: O(N)  
5 //  
6  
7 #include <bits/stdc++.h>  
8  
9 using namespace std;  
10  
11 vector<vector<int>> adj;  
12 vector<int> dp;  
13  
14 int dfs(int node) {  
15     if (dp[node] != -1) return dp[node];  
16  
17     int longest = 0;  
18     for (int i = 0; i < adj[node].size(); i++) {  
19         longest = max(longest, dfs(adj[node][i]) + 1);  
20     }  
21  
22     dp[node] = longest;  
23     return longest;  
24 }  
25  
26 int main() {  
27     int n, m;  
28     cin >> n >> m;  
29  
30     adj.resize(n);  
31     dp.resize(n, -1);  
32  
33     for (int i = 0; i < m; i++) {  
34         int x, y; cin >> x >> y; x-- ; y-- ;  
35         adj[y].push_back(x);  
36     }  
37  
38     int ans = 0;  
39  
40     // DFS from each vertex  
41     for (int i = 0; i < n; i++) {  
42         ans = max(ans, dfs(i));  
43     }  
44  
45     cout << ans << endl;  
46     return 0;  
47 }
```

```
1 //  
2 // Number of paths through a labyrinth.  
3 //  
4 // Time Complexity: O(NM) = O(size of the matrix)  
5 //  
6  
7 #include <bits/stdc++.h>  
8  
9 using namespace std;  
10  
11 const int MOD = 1e9 + 7;  
12  
13 vector<vector<bool>> board;  
14 vector<vector<int>> dp;  
15  
16 int main() {  
17     int n, m;  
18     cin >> n >> m;  
19  
20     board.resize(n, vector<bool>(m, true));  
21     dp.resize(n, vector<int>(m, 0));  
22  
23     for (int i = 0; i < n; i++) {  
24         for (int j = 0; j < m; j++) {  
25             char c; cin >> c;  
26             board[i][j] = (c == '.');  
27         }  
28     }  
29  
30     for (int i = 0; i < n; i++) {  
31         for (int j = 0; j < m; j++) {  
32             if (i == 0 && j == 0) dp[i][j] = 1;  
33             else if (i == 0) dp[i][j] = dp[i][j-1] * board[i][j];  
34             else if (j == 0) dp[i][j] = dp[i-1][j] * board[i][j];  
35             else {  
36                 dp[i][j] = (dp[i-1][j] + dp[i][j-1]) * board[i][j] % MOD;  
37             }  
38         }  
39     }  
40  
41     cout << dp[n-1][m-1] << endl;  
42     return 0;  
43 }
```

```
1 //  
2 // Let N be a positive odd number.  
3 // There are N coins, numbered 1,2,...,N. For each i (1≤i≤N), when Coin i is tossed,  
4 // it comes up heads with probability p[i] and tails with probability 1-p[i] tails.  
5 // Taro has tossed all the N coins. Find the probability of having more heads than tails.  
6 //  
7 // Time Complexity: O(N^2)  
8 //  
9  
10 #include <bits/stdc++.h>  
11  
12 using namespace std;  
13  
14 int main() {  
15     int n;  
16     cin >> n;  
17  
18     // everything in this problem  
19     // is with 1-based indexing  
20     // for simplicity  
21     vector<double> p(n+1);  
22     for (int i = 1; i ≤ n; i++) {  
23         cin >> p[i];  
24     }  
25  
26     vector<vector<double>> dp(n+1, vector<double>(n+1, 0));  
27     dp[1][0] = 1 - p[1];  
28     dp[1][1] = p[1];  
29  
30     for (int i = 2; i ≤ n; i++) {  
31         for (int j = 0; j ≤ i; j++) {  
32             // dp[i][j] - the probability of  
33             // getting j heads out of  
34             // the coins to the i-th index  
35  
36             if (j > 0)  
37                 dp[i][j] += dp[i-1][j-1] * p[i];  
38  
39             dp[i][j] += dp[i-1][j] * (1 - p[i]);  
40         }  
41     }  
42  
43     double ans = 0;  
44     for (int i = n/2 + 1; i ≤ n; i++) {  
45         ans += dp[n][i];  
46     }  
47  
48     cout << setprecision(16) << ans << endl;  
49     return 0;  
50 }
```

```

1 // 
2 // There are N dishes, numbered 1,2 ... N. Initially, for each i (1≤i≤N),
3 // Dish i has a[i] (1 ≤ a[i] ≤ 3) pieces of sushi on it.
4 // Taro will perform the following operation repeatedly until all the pieces of sushi are eaten:
5 //     - Roll a die that shows the numbers 1,2 ... N with equal probabilities, and let i be the outcome.
6 //     If there are some pieces of sushi on Dish i, eat one of them; if there is none, do nothing.
7 // Find the expected number of times the operation is performed before all the pieces of sushi are eaten.
8 //
9 // Time Complexity: O(N^3)
10 //
11
12 #include <bits/stdc++.h>
13
14 using namespace std;
15
16 int n; // n is also the "sum of s[i]"
17 vector<int> mp(4, 0);
18
19 const int MAX = 310;
20 double dp[MAX][MAX][MAX];
21
22 double dp_f(int x, int y, int z) {
23     if (dp[x][y][z] != -1) {
24         return dp[x][y][z];
25     }
26
27     if (x == 0 && y == 0 && z == 0) return 0;
28
29     double zero = n - x - y - z;
30     double rolls = (n - zero) / n + zero * (2 * n - zero) / (n * (n - zero));
31     double current_sum = rolls;
32
33     if (x) {
34         double weight = x / (n - zero);
35         current_sum += weight * dp_f(x - 1, y, z);
36     }
37
38     if (y) {
39         double weight = y / (n - zero);
40         current_sum += weight * dp_f(x + 1, y - 1, z);
41     }
42
43     if (z) {
44         double weight = z / (n - zero);
45         current_sum += weight * dp_f(x, y + 1, z - 1);
46     }
47
48     dp[x][y][z] = current_sum;
49     return current_sum;
50 }
51
52 int main() {
53     for (int i = 0; i < MAX; i++) {
54         for (int j = 0; j < MAX; j++) {
55             for (int k = 0; k < MAX; k++) {
56                 dp[i][j][k] = -1;
57             }
58         }
59     }
60
61     scanf("%d", &n);
62
63     for (int i = 0; i < n; i++) {
64         int x;
65         scanf("%d", &x);
66         mp[x]++;
67     }
68
69     cout << setprecision(16) << dp_f(mp[1], mp[2], mp[3]) << endl;
70     return 0;
71 }

```

```
1 //  
2 // There is a set A = {a_1, a_2 ... } consisting of N positive integers.  
3 // Taro and Jiro will play the following game against each other.  
4 // Initially, we have a pile consisting of K stones.  
5 // The two players perform the following operation alternately, starting from Taro:  
6 // - Choose an element x in A, and remove exactly x stones from the pile.  
7 // A player loses when he becomes unable to play.  
8 //  
9 // Assuming that both players play optimally, determine the winner.  
10 // Time Complexity: O(NK)  
11 //  
12  
13 #include <bits/stdc++.h>  
14 #define ll long long  
15  
16 using namespace std;  
17  
18 inline void umin(int& a, int b) { a = min(a, b); }  
19 inline void umax(int& a, int b) { a = max(a, b); }  
20  
21 int main() {  
22     int n, k;  
23     cin >> n >> k;  
24  
25     vector<int> a(n);  
26     for (int i = 0; i < n; i++) cin >> a[i];  
27  
28     vector<vector<int>> dp(k+1, vector<int>({ 1000, -1000 }));  
29     // player 0 wants to minimize  
30     // player 1 wants to maximize  
31  
32     for (int i = 0; i ≤ k; i++) {  
33         for (int j = 0; j < n; j++) {  
34             if (i - a[j] < 0) {  
35                 umin(dp[i][0], 1);  
36                 umax(dp[i][1], 0);  
37             } else {  
38                 umin(dp[i][0], dp[i-a[j]][1]);  
39                 umax(dp[i][1], dp[i-a[j]][0]);  
40             }  
41         }  
42     }  
43  
44     cout << (dp[k][0] ? "Second" : "First") << endl;  
45     return 0;  
46 }
```

```

1 // 
2 // Taro and Jiro will play the following game against each other.
3 // Initially, they are given a sequence a = (a[1], a[2], ..., a[n])
4 // Until a becomes empty, the two players perform the following operation alternately,
5 // starting from Taro:
6 //   - Remove the element at the beginning or the end of a.
7 //   The player earns x points, where x is the removed element.
8 //
9 // Let X and Y be Taro's and Jiro's total score at the end of the game, respectively.
10 // Taro tries to maximize X-Y, while Jiro tries to minimize X-Y.
11 // Assuming that the two players play optimally, find the resulting value of X-Y.
12 //
13 // Time Complexity: O(N^2)
14 //
15
16 #include <bits/stdc++.h>
17 #define ll long long
18
19 using namespace std;
20
21 int main() {
22     int n;
23     cin >> n;
24
25     vector<ll> awards(n);
26     vector<vector<ll>> dp(n, vector<ll>(n, 0));
27     for (int i = 0; i < n; i++) {
28         cin >> awards[i];
29
30         // BASE CASE:
31         dp[i][i] = awards[i];
32     }
33
34     // interval starting and ending at position i = score is awards[i]
35     // solving for subintervals
36
37     // - - - - -
38     //   ←→
39     //   ←→
40     //   ←→
41
42     for (int size = 1; size < n; size++) { // size of interval
43         for (int start = 0; start < n - size; start++) { // starting position of interval
44             if (size % 2 == 0) { // Taro (max)
45                 // START - L
46                 // START+SIZE = R
47                 dp[start][start+size] =
48                     max(dp[start][start+size-1] + awards[start+size],
49                         dp[start+1][start+size] + awards[start]);
50             } else { // Jiro (min)
51                 dp[start][start+size] =
52                     min(dp[start][start+size-1] - awards[start+size],
53                         dp[start+1][start+size] - awards[start]);
54             }
55         }
56     }
57
58     // dp[0][n-1]
59     cout << (n % 2 == 0 ? -1LL : 1LL) * dp[0][n-1] << endl;
60     return 0;
61 }
```

```
1 //  
2 // There are N children, numbered 1,2,...,N.  
3 // They have decided to share K candies among themselves.  
4 // Here, for each i (1≤i≤N), Child i must receive between 0 and a[i] candies (inclusive).  
5 // Also, no candies should be left over.  
6 //  
7 // Find the number of ways for them to share candies, modulo 10^9 + 7.  
8 // Here, two ways are said to be different when there  
9 // exists a child who receives a different number of candies.  
10 //  
11 // Time Complexity: O(NK)  
12 //  
13  
14 #include <bits/stdc++.h>  
15 #define ll long long  
16  
17 using namespace std;  
18  
19 const int MOD = 1e9 + 7;  
20  
21 int main() {  
22     int n, k;  
23     cin >> n >> k;  
24  
25     vector<int> a(n);  
26     for (int i = 0; i < n; i++) cin >> a[i];  
27  
28     vector<vector<int>> dp(n, vector<int>(k+1, 0));  
29     for (int i = k; i ≥ k - a[0]; i--) dp[0][i] = 1;  
30  
31     for (int i = 1; i < n; i++) {  
32         vector<int> pref(k+1);  
33         pref[k] = dp[i-1][k];  
34         for (int j = k-1; j ≥ 0; j--) {  
35             pref[j] = (pref[j+1] + dp[i-1][j]) % MOD;  
36         }  
37  
38         for (int j = 0; j ≤ k; j++) {  
39             // dp[i][j] = dp[i-1][j+(0...a[i])]  
40             dp[i][j] = (pref[j] - (j+a[i]+1) ≤ k ? pref[j+a[i]+1] : 0) + MOD) % MOD;  
41         }  
42     }  
43  
44     cout << dp[n-1][0] << endl;  
45     return 0;  
46 }
```

```
1 //  
2 // There are N slimes lining up in a row. Initially, the i-th slime from the left has a size of a[i].  
3 // Taro is trying to combine all the slimes into a larger slime.  
4 // He will perform the following operation repeatedly until there is only one slime:  
5 // - Choose two adjacent slimes, and combine them into a new slime.  
6 // The new slime has a size of x+y, where x and y are the sizes  
7 // of the slimes before combining them.  
8 // Here, a cost of x+y is incurred. The positional relationship  
9 // of the slimes does not change while combining slimes.  
10 // Find the minimum possible total cost incurred.  
11 //  
12 // Time Complexity: O(N^3)  
13 //  
14  
15 #include <bits/stdc++.h>  
16 #define ll long long  
17  
18 using namespace std;  
19  
20 int main() {  
21     int n;  
22     cin >> n;  
23  
24     vector<int> s(n);  
25     vector<ll> prefix(n);  
26     for (int i = 0; i < n; i++) {  
27         cin >> s[i];  
28         prefix[i] = (i-1 >= 0 ? prefix[i-1] : 0LL) + s[i];  
29     }  
30  
31     vector<vector<ll>> dp(n, vector<ll>(n, 0));  
32  
33     for (int size = 1; size <= n; size++) {  
34         for (int start = 0; start < n - size; start++) {  
35             int L = start;  
36             int R = start + size;  
37  
38             ll best = LLONG_MAX;  
39             for (int i = 0; i < R - L; i++) {  
40                 best = min(best, dp[L][L+i] + dp[L+i+1][R]);  
41             }  
42  
43             dp[L][R] = best + (prefix[R] - (L-1 >= 0 ? prefix[L-1] : 0));  
44         }  
45     }  
46  
47     cout << dp[0][n-1] << endl;  
48     return 0;  
49 }
```

```
1 //  
2 // There are N men and N women, both numbered 1,2,...,N.  
3 // For each i,j (1≤i,j≤N), the compatibility of Man i and Woman j is given as an integer a[i][j]  
4 // If a[i][j] = 1, Man i and Woman j are compatible; otherwise not.  
5 // Taro is trying to make N pairs, each consisting of a man and a woman who are compatible.  
6 // Here, each man and each woman must belong to exactly one pair.  
7 // Find the number of ways in which Taro can make N pairs, modulo 10^9 + 7  
8 //  
9 // Time Complexity: O(2^N * N^2)  
10 //
```

```
11  
12 #include <bits/stdc++.h>  
13 #define ll long long  
14  
15 using namespace std;  
16  
17 const int MOD = 1e9 + 7;  
18 const int maxSetCount = (1 << 21) + 1;  
19  
20 bool match[25][25];  
21 int dp[maxSetCount][21];  
22 int n;  
23  
24 int main() {  
25     cin >> n;  
26  
27     // memset the dp  
28     for (int i = 0; i < maxSetCount; i++) {  
29         for (int j = 0; j < n; j++) {  
30             dp[i][j] = 0;  
31         }  
32     }  
33  
34     for (int i = 0; i < n; i++) {  
35         for (int j = 0; j < n; j++) {  
36             cin >> match[i][j];  
37         }  
38     }  
39  
40     int full_set = (1 << n) - 1;  
41  
42     // base case  
43     for (int i = 0; i < n; i++) {  
44         dp[0][i] = 1;  
45     }  
46  
47     for (int set = 0; set < (1 << n); set++) {  
48         int count = __builtin_popcount(set);  
49         for (int last = 0; last < n; last++) {  
50             if (((1 << last) & set) == 0 && match[last][count]) {  
51                 int next_set = set ^ (1 << last);  
52  
53                 for (int i = 0; i < n; i++) {  
54                     dp[next_set][i] = (dp[next_set][i] + dp[set][last]) % MOD;  
55                 }  
56             }  
57         }  
58     }  
59  
60     cout << dp[full_set][0] << endl;  
61     return 0;  
62 }
```

```

1 // 
2 // There is a tree with N vertices, numbered 1,2, ... N.
3 // For each i (1 ≤ i ≤ N-1), the i-th edge connects Vertex x[i] and y[i].
4 // Taro has decided to paint each vertex in white or black.
5 // Here, it is not allowed to paint two adjacent vertices both in black.
6 //
7 // Find the number of ways in which the vertices can be painted, modulo 10^9 + 7.
8 //
9 // Time Complexity: O(N)
10 //
11
12 #include <bits/stdc++.h>
13 #define ll long long
14
15 using namespace std;
16
17 const int MOD = 1e9 + 7;
18
19 vector<vector<int>> adj;
20 vector<vector<int>> dp;
21
22 int dfs(int node, int black, int parent = -1) {
23     if (dp[node][black] != -1) return dp[node][black];
24
25     dp[node][black] = 1;
26     for (int i = 0; i < (int)adj[node].size(); i++) {
27         int next_node = adj[node][i];
28
29         if (next_node == parent) continue;
30
31         if (black) {
32             dp[node][1] = ((ll)dp[node][1] * dfs(next_node, 0, node)) % MOD;
33         } else {
34             ll subtree_white = dfs(next_node, 0, node);
35             ll subtree_black = dfs(next_node, 1, node);
36
37             dp[node][0] = ((ll)dp[node][0] *
38                             ((subtree_white + subtree_black) % MOD)) % MOD;
39         }
40     }
41
42     return dp[node][black];
43 }
44
45 int main() {
46     int n;
47     cin >> n;
48
49     adj.resize(n);
50     dp.resize(n, vector<int>(2, -1));
51
52     for (int i = 0; i < n-1; i++) {
53         int x, y;
54         cin >> x >> y;
55         x--; y--;
56         adj[x].push_back(y);
57         adj[y].push_back(x);
58     }
59
60     cout << ((ll)dfs(0, 0) + dfs(0, 1)) % MOD << endl;
61     return 0;
62 }
```

```

1 // 
2 // There are N flowers arranged in a row. For each i (1≤i≤N), the height and the beauty of
3 // the i-th flower from the left is h[i] and a[i], respectively.
4 // Here, h[1], h[2], ... h[N] are all distinct.
5 // Taro is pulling out some flowers so that the following condition is met:
6 // - The heights of the remaining flowers are monotonically increasing from left to right.
7 // Find the maximum possible sum of the beauties of the remaining flowers.
8 //
9 // Constrains:
10 // h[i] ≤ N
11 //
12 // Time Complexity: O(N * log(N))
13 //
14
15
16 #include <bits/stdc++.h>
17 #define ll long long
18
19 using namespace std;
20
21 // Max Fenwick Tree
22 struct FenwickTree {
23     vector<ll> fwt;
24
25     FenwickTree(int n) {
26         fwt.resize(n, 0);
27     }
28
29     void maxFWT(int ind, ll val = 1) {
30         for (ind++; ind < fwt.size(); ind+=ind&-ind)
31             fwt[ind] = max(fwt[ind], val);
32     }
33
34     ll getFWT(int ind) {
35         ll s = 0;
36         for (ind++; ind > 0; ind-=ind&-ind)
37             s = max(s, fwt[ind]);
38         return s;
39     }
40 };
41
42 int main() {
43     int n;
44     cin >> n;
45
46     vector<ll> h(n), a(n);
47     for (int i = 0; i < n; i++) scanf("%d", &h[i]);
48     for (int i = 0; i < n; i++) scanf("%d", &a[i]);
49
50     FenwickTree tree(n+10);
51
52     for (int i = 0; i < n; i++) {
53         // query best = max (1 ⇒ h[i] - 1)
54         // update at h[i] = best + a[i];
55
56         ll best = tree.getFWT(h[i]); // get maximum
57         ll new_best = best + a[i];
58         tree.maxFWT(h[i], new_best);
59     }
60
61     cout << tree.getFWT(n) << endl;
62     return 0;
63 }

```

```

1 // There is a simple directed graph G with N vertices, numbered 1,2,...,N.
2 // For each i and j ( $1 \leq i, j \leq N$ ), you are given an integer  $a[i][j]$ 
3 // that represents whether there is a directed edge from Vertex i to j if  $a[i][j]$  is set.
4 // Find the number of different directed paths of length K in G, modulo  $10^9 + 7$ 
5 // We will also count a path that traverses the same edge multiple times.
6 //
7 //
8 // Constraints:
9 //    $N \leq 50$ 
10 //    $K \leq 10^{18}$ 
11 //
12 // Time Complexity:  $O(N^3 * \log(K))$ 
13 //
14
15 #include <bits/stdc++.h>
16 #define MX vector<vector<T>>
17 #define MOD 1000000007
18 #define ll long long
19
20 using namespace std;
21
22 template<typename T>
23 MX mult(MX a, MX b) {
24     MX result(a.size(), vector<T>(a.size(), 0));
25     for (int i = 0; i < a.size(); i++) {
26         for (int j = 0; j < a.size(); j++) {
27             for (int k = 0; k < a.size(); k++) {
28                 result[i][j] = (result[i][j] + (a[i][k] * b[k][j]) % MOD) % MOD;
29             }
30         }
31     }
32
33     return result;
34 }
35
36 template<typename T>
37 MX binary(MX a, ll n) {
38     MX result(a.size(), vector<T>(a.size(), 0));
39     for (int i = 0; i < a.size(); i++) result[i][i] = 1;
40
41     while (n > 0) {
42         if (n % 2 == 1) {
43             result = mult(result, a);
44             n--;
45         }
46
47         a = mult(a, a);
48         n /= 2;
49     }
50
51     return result;
52 }
53
54 int main() {
55     ll n, k;
56     cin >> n >> k;
57
58     vector<vector<long long>> mat(n, vector<long long>(n, 0));
59
60     for (int i = 0; i < n; i++) {
61         for (int j = 0; j < n; j++) {
62             cin >> mat[i][j];
63         }
64     }
65
66     auto result = binary(mat, k);
67
68     long long sum = 0;
69     for (int i = 0; i < n; i++) {
70         for (int j = 0; j < n; j++) {
71             sum = (sum + result[i][j]) % MOD;
72         }
73     }
74
75     cout << sum << endl;
76     return 0;
77 }

```

```

1 // 
2 //   Find the number of integers between 1 and K (inclusive)
3 //   satisfying the following condition, modulo 10^9 + 7:
4 //     - The sum of the digits in base ten is a multiple of D.
5 //
6 //   Time Complexity: O(ND)
7 //
8
9 #include <bits/stdc++.h>
10 #define ll long long
11
12 using namespace std;
13
14 const int MOD = 1e9 + 7;
15 inline void uadd(int& a, int b) {
16     a += b;
17     if (a >= MOD) a -= MOD;
18 }
19
20 int main() {
21     string s; cin >> s;
22     int v; cin >> v;
23     int n = s.size();
24
25     if (n == 1) {
26         int ans = 0;
27         for (int i = 1; i <= (s[0] - '0'); i++) {
28             if (i % v == 0) ans++;
29         }
30         cout << ans << endl;
31         return 0;
32     }
33
34     vector<int> pref(n);
35     pref[0] = (s[0] - '0') % v;
36     for (int i = 1; i < n; i++) {
37         pref[i] = (pref[i-1] + s[i] - '0') % v;
38     }
39
40     // backwards dynamic programming
41     vector<vector<int>> bw(n, vector<int>(v, 0));
42     int ans = 0;
43
44     // Base case
45     for (int d = 0; d < 10; d++) {
46         bw[n-1][d%v] += 1;
47         if (d <= s[n-1] - '0' && n-2 >= 0 && (pref[n-2] + d) % v == 0) ans++;
48     }
49
50     for (int i = n - 2; i >= 0; i--) {
51         for (int d = 0; d <= (i == 0 ? s[0] - '0' - 1 : 9); d++) {
52             for (int j = 0; j < v; j++) {
53                 uadd(bw[i][(j+d)%v], bw[i+1][j]);
54             }
55         }
56
57         if (i-1 >= 0) {
58             for (int d = 0; d < s[i] - '0'; d++) {
59                 uadd(ans, bw[i+1][(v - ((pref[i-1] + d + 100 * v) % v)) % v]);
60             }
61         }
62     }
63
64     cout << ((ll)ans + bw[0][0] - 1 + MOD) % MOD << endl;
65     return 0;
66 }

```

```

1 // 
2 // Let N be a positive integer. You are given a string s of length N-1, consisting of < and >.
3 // Find the number of permutations (p[1], p[2], ... p[N]) of (1,2, ... N)
4 // that satisfy the following condition, modulo 10^9 + 7
5 // For each i (1 ≤ i ≤ N-1), p[i] < p[i+1] if the i-th character in s is '<'
6 //                                and p[i] > p[i+1] if the i-th character in s is '>'
7 //
8 //      Time Complexity: O(N^2)
9 //
10
11 #include <bits/stdc++.h>
12 #define ll long long
13 using namespace std;
14 const int MOD = 1e9 + 7;
15
16 inline void uadd(int& a, int b) {
17     a = (a + b);
18     if (a ≥ MOD) a -= MOD;
19 }
20
21 int main() {
22     int n;
23     cin >> n;
24
25     string s;
26     cin >> s;
27     s = " " + s;
28
29     vector<vector<int>> dp(n, vector<int>(n, 0));
30     dp[0][0] = 1;
31
32     // dp[i][j] - number of ways to order the first i elements ending with item j
33     for (int i = 1; i < n; i++) {
34         int pref = 0;
35         for (int j = 0; j ≤ i; j++) { // last number
36             if (s[i] == '<') {
37                 uadd(dp[i][j], pref);
38                 uadd(pref, dp[i-1][j]);
39             } else {
40                 uadd(dp[i][i-j], pref);
41                 uadd(pref, dp[i-1][i-j-1]);
42             }
43         }
44     }
45
46     int ans = 0;
47     for (int i = 0; i < n; i++) {
48         uadd(ans, dp[n-1][i]);
49     }
50
51     cout << ans << endl;
52     return 0;
53 }

```

```
1 //  
2 // There are N rabbits, numbered 1,2, ... N.  
3 // For each i,j (1 ≤ i,j ≤ N), the compatibility of  
4 // Rabbit i and j is described by an integer a[i][j]  
5 // a[i][i] = 0  
6 // a[i][j] = a[j][i]  
7 //  
8 // Taro is dividing the N rabbits into some number of groups.  
9 // Here, each rabbit must belong to exactly one group.  
10 // After grouping, for each i and j (1 ≤ i < j ≤ N),  
11 // Taro earns a[i][j] points if Rabbit i and j belong to the same group.  
12 //  
13 // Find Taro's maximum possible total score.  
14 //  
15 // Time Complexity: O(2^N * N^2 + 2^(2N))  
16 //  
17  
18 #include <bits/stdc++.h>  
19 #define ll long long  
20  
21 using namespace std;  
22  
23 const int mxn = (1 << 16) + 10;  
24 ll dp[mxn];  
25  
26 int main() {  
27     for (int i = 0; i < mxn; i++) {  
28         dp[i] = 0;  
29     }  
30  
31     int n;  
32     cin >> n;  
33  
34     vector<vector<int>> a(n, vector<int>(n, 0));  
35     for (int i = 0; i < n; i++) {  
36         for (int j = 0; j < n; j++) {  
37             scanf("%d", &a[i][j]);  
38         }  
39     }  
40  
41     for (int mask = 0; mask < (1 << n); mask++) {  
42         ll score = 0;  
43         for (int bit1 = 0; bit1 < n; bit1++) {  
44             for (int bit2 = bit1 + 1; bit2 < n; bit2++) {  
45                 if (mask & (1 << bit1) && mask & (1 << bit2)) {  
46                     score += a[bit1][bit2];  
47                 }  
48             }  
49         }  
50  
51         dp[mask] = max(dp[mask], score);  
52     }  
53  
54     for (int mask1 = 0; mask1 < (1 << n); mask1++) {  
55         for (int mask2 = mask1 + 1; mask2 < (1 << n); mask2++) {  
56             if ((mask1 & mask2) == 0) {  
57                 dp[mask1 | mask2] = max(dp[mask1 | mask2], dp[mask1] + dp[mask2]);  
58             }  
59         }  
60     }  
61  
62     cout << dp[(1 << n) - 1] << endl;  
63     return 0;  
64 }
```

```

1 // 
2 // There is a tree with N vertices, numbered 1,2, ... N.
3 // For each i (1 ≤ i ≤ N-1), the i-th edge connects Vertex x[i] and y[i].
4 // Taro has decided to paint each vertex in white or black,
5 // so that any black vertex can be reached from any other black vertex
6 // by passing through only black vertices.
7 // You are given a positive integer M. For each v (1 ≤ v ≤ N), answer the following question:
8 //   - Assuming that Vertex v has to be black,
9 //     find the number of ways in which the vertices can be painted, modulo M.
10 //
11 // Time Complexity: O(N+M)
12 //
13
14 #include <bits/stdc++.h>
15 #define ll long long
16 using namespace std;
17
18 int n;
19 ll MOD;
20
21 vector<ll> in, out, excepts;
22 vector<vector<int>> adj;
23
24 // ans[node] = in[node] * out[node]
25 int dfs_in(int node, int parent = -1) {
26     int size = adj[node].size();
27     vector<ll> ans_child(size, 1);
28     vector<ll> dp_pref(size, 1), dp_suf(size, 1);
29     for (int i = 0; i < size; i++) {
30         int next_node = adj[node][i];
31         if (next_node == parent) {
32             dp_pref[i] = (i-1 ≥ 0 ? dp_pref[i-1] : 1);
33             continue;
34         }
35
36         int black_ways = dfs_in(next_node, node);
37         ans_child[i] = (black_ways + 1) % MOD;
38         in[node] = (in[node] * (black_ways + 1) % MOD) % MOD;
39         dp_pref[i] = ((i-1 ≥ 0 ? dp_pref[i-1] : 1) * (black_ways + 1) % MOD) % MOD;
40     }
41
42     if (size > 1) {
43         // calculate the suffixes
44         dp_suf[size-1] = ans_child[size-1];
45         for (int i = size - 2; i ≥ 0; i--) {
46             dp_suf[i] = (dp_suf[i+1] * ans_child[i]) % MOD;
47         }
48
49         // except the i-th child
50         for (int i = 0; i < size; i++) {
51             int child = adj[node][i];
52             if (child == parent) continue;
53
54             ll except =
55                 ((i-1 ≥ 0 ? dp_pref[i-1]: 1) *
56                 (i+1 < size ? dp_suf[i+1] : 1)) % MOD;
57
58             excepts[child] = except;
59         }
60     }
61
62     return in[node];
63 }
64
65 void dfs_out(int node, int parent = -1) {
66     int size = adj[node].size();
67     for (int i = 0; i < size; i++) {
68         int child = adj[node][i];
69         if (child == parent) continue;
70
71         if (parent == -1) out[child] = (excepts[child] + 1) % MOD;
72         else out[child] = ((excepts[child] * out[node] % MOD) + 1) % MOD;
73
74         dfs_out(child, node);
75     }
76 }
77
78 int main() {
79     cin >> n >> MOD;
80     in.resize(n, 1);
81     out.resize(n, 1);
82     adj.resize(n);
83     excepts.resize(n, 1);
84     for (int i = 0; i < n-1; i++) {
85         int x, y;
86         cin >> x >> y;
87         x--; y--;
88         adj[x].push_back(y);
89         adj[y].push_back(x);
90     }
91
92     dfs_in(0); dfs_out(0);
93     // root node doesn't have out[root]
94     cout << in[0] % MOD << "\n";
95     for (int i = 1; i < n; i++) {
96         cout << (in[i] * out[i]) % MOD << "\n";
97     }
98     return 0;
99 }

```

```

1 // Consider a string of length N consisting of 0 and 1.
2 // The score for the string is calculated as follows:
3 //   - For each i (1 ≤ i ≤ M), a is added to the score
4 //     if the string contains 1 at least once
5 //     between the l-th and r-th characters (inclusive).
6 //   Find the maximum possible score of a string.
7 //
8 // Time Complexity: O(N log N)
9 //
10 //
11
12 #include <bits/stdc++.h>
13 #define ll long long
14
15 using namespace std;
16
17 inline void uadd(ll& a, ll b) { a += b; }
18 const ll NEG_INF = LLONG_MIN / 10;
19
20 struct LazySegmentTree {
21     int N = -1;
22
23     struct Node {
24         ll value = NEG_INF; // default
25         ll lazy = 0;
26         Node operator+(const Node &other) {
27             return {
28                 max(this->value, other.value)
29             };
30         }
31     };
32
33     vector<Node> seg;
34     LazySegmentTree(int N) {
35         this->N = N;
36         int sz = 1 << 21;
37         seg.resize(sz);
38     }
39
40     void push(int ind, int l, int r) {
41         if (seg[ind].lazy == 0) return;
42         if (seg[ind].value == NEG_INF) seg[ind].value = seg[ind].lazy;
43         else uadd(seg[ind].value, seg[ind].lazy);
44         if (r - l != 1) {
45             uadd(seg[2*ind+1].lazy, seg[ind].lazy);
46             uadd(seg[2*ind+2].lazy, seg[ind].lazy);
47         }
48         seg[ind].lazy = 0;
49     }
50
51     void updateRange(int b, int e, ll x, int ind = 0, int l = 0, int r = -1) {
52         if (r == -1) r = N;
53         push(ind, l, r);
54         if (e <= l || b >= r) { return; }
55         if (b <= l && r <= e) {
56             uadd(seg[ind].lazy, (ll)x);
57             push(ind, l, r);
58             return;
59         }
60
61         int m = (l + r) / 2;
62         updateRange(b, e, x, 2 * ind + 1, l, m);
63         updateRange(b, e, x, 2 * ind + 2, m, r);
64         seg[ind] = seg[2*ind+1] + seg[2*ind+2];
65     }
66
67     // result in [b,e) of node that covers [l,r)
68     Node askLR(int b, int e, int ind = 0, int l = 0, int r = -1) {
69         if (r == -1) r = N;
70         push(ind, l, r);
71         if (l >= e || r <= b) return { }; // empty
72         if (l >= b && r <= e) return seg[ind];
73         int m = (l + r) / 2;
74         return askLR(b, e, ind * 2 + 1, l, m) + askLR(b, e, ind * 2 + 2, m, r);
75     }
76 };
77
78 struct query {
79     ll l, r, v;
80 };

```

```

81 | int main() {
82 |     int n, m;
83 |     cin >> n >> m;
84 |
85 |
86 |     vector<query> q(m);
87 |     vector<vector<int>> events(n + 5);
88 |     for (int i = 0; i < m; i++) {
89 |         int l, r, v;
90 |         cin >> l >> r >> v;
91 |         q[i] = {l, r, v};
92 |         events[l].push_back(i);
93 |         events[r].push_back(~i);
94 |     }
95 |
96 |     // dp[i] - the best score of the prefix to index i if s[i] = 1
97 |     // v[q] - value of the q-th query
98 |     // dp[i] = max(dp[j] + sum(v[q])) for all j < i ,
99 |     //           for all q if l[q] ≤ i ≤ r[q] && !(l[q] ≤ j ≤ r[q]))
100 |    // when you meet a left edge → tree.add( [ 0, l[q] - 1 ], v[q] )
101 |    // when you meet a right edge → tree.add( [ 0, l[q] - 1 ], -v[q] )
102 |
103 |    // explanation:
104 |    // when you meet a left edge, this interval gets activated, so
105 |    // for every dp which is up to this point, we want to add this interval's value.
106 |    // when this interval dies out, we remove it from the dp tree, because other dps
107 |    // to the right will have this edge included if it was worth it.
108 |
109 |    LazySegmentTree tree(n + 10);
110 |    for (int i = 1; i ≤ n; i++) {
111 |        for (auto& event: events[i]) {
112 |            if (event ≥ 0) {
113 |                tree.updateRange(0, q[event].l, q[event].v);
114 |            }
115 |        }
116 |
117 |
118 |        ll new_dp = tree.askLR(0, i).value;
119 |        tree.updateRange(i, i+1, new_dp);
120 |
121 |        for (auto& event: events[i]) {
122 |            if (event < 0) {
123 |                tree.updateRange(0, q[~event].l, -q[~event].v);
124 |            }
125 |        }
126 |
127 |
128 |        cout << tree.askLR(0, n+1).value << endl;
129 |        return 0;
130 |    }

```

```

1 // 
2 // There are N blocks, numbered 1,2, ... N. For each i (1 ≤ i ≤ N),
3 // Block i has a weight of w[i], a solidness of s[i] and a value of v[i].
4 // Taro has decided to build a tower by choosing some of the N blocks
5 // and stacking them vertically in some order.
6 // Here, the tower must satisfy the following condition:
7 //     - For each Block i contained in the tower,
8 //         the sum of the weights of the blocks stacked above it is not greater than s[i].
9 // Find the maximum possible sum of the values of the blocks contained in the tower.
10 //
11 // Time Complexity: O(N * max(W))
12 //
13 // ! Note: MXW HAS TO BE AT LEAST 2 * max(w[i]) !
14 // ! Note: Memory usage can get too big quickly !
15 //
16
17 #include <bits/stdc++.h>
18 #define ll long long
19
20 using namespace std;
21
22 struct block {
23     ll weight, solidness, value;
24
25     bool operator<(const block& other) {
26         return (weight + solidness) > (other.weight + other.solidness);
27         // we want to exchange the blocks if
28         //     w[top] > w[bottom]
29         // & s[top] > s[bottom] ⇒
30         //     w[top] + s[top] > w[bottom] + s[bottom]
31     }
32 };
33
34 inline void umax(ll& a, ll b) {
35     if (a < b) a = b;
36 }
37
38 int main() {
39     int n;
40     cin >> n;
41
42     vector<block> a(n);
43     for (int i = 0; i < n; i++) {
44         cin >> a[i].weight >> a[i].solidness >> a[i].value;
45     }
46
47     sort(a.begin(), a.end());
48
49     const int mxw = 3e4 + 10;
50     vector<vector<ll>> dp(n, vector<ll>(mxw, 0));
51     // dp[i][j] - to the i-th box, if j weight remains
52
53     ll ans = 0;
54
55     // Base case
56     for (int w = 0; w ≤ a[0].solidness; w++) {
57         umax(dp[0][w], a[0].value);
58         umax(ans, dp[0][w]);
59     }
60
61     for (int i = 1; i < n; i++) {
62         for (int j = 0; j < mxw; j++) {
63             umax(dp[i][j], dp[i-1][j]);
64             int weight_remains = min(j - a[i].weight, a[i].solidness);
65             if (weight_remains ≥ 0) {
66                 umax(dp[i][weight_remains], dp[i-1][j] + a[i].value);
67                 umax(ans, dp[i][weight_remains]);
68             }
69         }
70     }
71
72     cout << ans << endl;
73     return 0;
74 }

```

```

1 #include <bits/stdc++.h>
2 #define ll long long
3
4 using namespace std;
5
6 const int MOD = 1e9 + 7;
7
8 struct Binomial {
9     int mxn = -1;
10    vector<int> inv, fact;
11
12    // inverse = pow(a, mod - 2);
13    int inverse(int a) {
14        int m = MOD;
15        int u = 0, v = 1;
16        while (a != 0) {
17            int t = m / a;
18            m -= t * a;
19            swap(a, m);
20            u -= t * v;
21            swap(u, v);
22        }
23        assert(m == 1);
24        return (u + MOD) % MOD;
25    }
26
27    void preprocess() {
28        fact[0] = 1;
29        for (int i = 1; i < mxn; i++) {
30            fact[i] = ((ll)fact[i-1] * i) % MOD;
31        }
32    }
33
34    Binomial(int mxn = (int)2e5 + 10) {
35        this->mxn = mxn;
36        fact.resize(mxn);
37        preprocess();
38    }
39
40    int C(int n, int k) {
41        if (k > n) swap(k, n);
42        return (((ll)fact[n] * inverse((ll)fact[k] * fact[n-k] % MOD)) % MOD;
43    }
44};
45
46 int main() {
47    int w, h, n;
48    cin >> w >> h >> n;
49
50    n += 2;
51
52    vector<pair<int, int>> points(n);
53    for (int i = 1; i < n-1; i++) {
54        int x, y;
55        cin >> x >> y;
56        x--; y--;
57        points[i] = {x, y};
58    }
59
60    points[0] = {0, 0};
61    points[n-1] = {w - 1, h - 1};
62
63    sort(points.begin(), points.end());
64
65    Binomial b(2e5 + 10);
66    vector<ll> f(n), g(n);
67    g[0] = 0;
68
69    for (int i = 1; i < n; i++) {
70        int x1, y1;
71        tie(x1, y1) = points[i];
72
73        for (int j = i-1; j >= 0; j--) {
74            int x2, y2;
75            tie(x2, y2) = points[j];
76
77            if (y2 > y1) continue;
78
79            ll X = x1 - x2;
80            ll Y = y1 - y2;
81
82            f[i] = (f[i] + (g[j] * b.C(X + Y, Y)) % MOD) % MOD;
83        }
84
85        g[i] = (b.C(x1 + y1, y1) - f[i] + MOD) % MOD;
86    }
87
88    cout << g[n-1] << endl;
89    return 0;
90}

```

```

1 // There are N stones, numbered 1,2, ... N. For each i (1 ≤ i ≤ N), the height of Stone i is h[i].
2 // Here h[1] < h[2] < [3] ... < h[N] holds.
3 // There is a frog who is initially on Stone 1.
4 // He will repeat the following action some number of times to reach Stone N:
5 // - If the frog is currently on Stone i, jump to one of the following:
6 //     Stone i+1,i+2, ... N. Here, a cost of (h[i] - h[j])^2 + C is incurred, where j is the stone to land on.
7 // Find the minimum possible total cost incurred before the frog reaches Stone N.
8 //
9 // Time Complexity: O(N)
10 //
11 //
12
13 #include <bits/stdc++.h>
14 #define ll long long
15
16 using namespace std;
17
18 vector<ll> h, dp;
19 int n;
20 ll C;
21
22 struct Line {
23     ll k, m, p;
24     ll eval(ll x) const { return k * x + m; }
25 };
26
27 struct CHT {
28     deque<Line> hull;
29     static const ll inf = LLONG_MAX;
30
31     ll div(ll a, ll b) { // floored division
32         return a / b - ((a ^ b) < 0 && a % b); }
33
34     bool intersect(Line &x, const Line &y) {
35         if (x.k == y.k) x.p = x.m > y.m ? inf : -inf;
36         else x.p = div(y.m - x.m, x.k - y.k);
37         return x.p ≥ y.p;
38     }
39
40     // Add a line with a form of kx + m
41     void add(ll k, ll m) {
42         Line L = {k, m, 0};
43         while ((int)hull.size() ≥ 2 && (intersect(L, hull.back()),
44             intersect(hull.back(), hull[(int)hull.size() - 2]), L.p < hull.back().p))
45             hull.pop_back();
46         hull.push_back(L);
47     }
48
49     // query at point x equivalent to cht(x)
50     ll query(ll x) {
51         while ((int)hull.size() ≥ 2 && hull[0].eval(x) ≥ hull[1].eval(x))
52             hull.pop_front();
53         return hull[0].eval(x);
54     }
55 };
56
57 int main() {
58     cin >> n >> C;
59
60     h.resize(n);
61     dp.resize(n);
62     for (int i = 0; i < n; i++) cin >> h[i];
63
64     CHT cht;
65     auto insert = [&](int i) {
66         cht.add(-2LL * h[i], (dp[i] + (h[i] * h[i])));
67     };
68
69     dp[0] = 0; insert(0);
70     for (int i = 1; i < n; i++) {
71         ll x = cht.query(h[i]);
72         dp[i] = C + (h[i] * h[i]) + x;
73
74         insert(i);
75     }
76
77     cout << dp[n-1] << endl;
78     return 0;
79 }

```