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

```cpp
//
// 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;
}
```

```cpp
//
// There are N stones, numbered 1,2,…,N. For each i (1≤i≤N), the height of Stone i is h
// 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 one of the following: Stone i+1, i+2 ... i+K.
//      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(NK)
//

#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, K;
    cin >> n >> K;

    vector<int> heights(n);
    for (int i = 0; i < n; i++) {
        cin >> heights[i];
    }

    vector<int> dp(n);
    dp[0] = 0;
    for (int i = 1; i < n; i++) {
        int cur_ans = dp[i-1] + abs(heights[i] - heights[i-1]);
        for (int k = 2; k <= K; k++) {
            if (i - k >= 0) {
                cur_ans = min(cur_ans, dp[i-k] + abs(heights[i] - heights[i-k]));
            }
        }
        dp[i] = cur_ans;
    }

    cout << dp[n-1] << endl;
    return 0;
}
```

```cpp
//
// Taro's summer vacation starts tomorrow, and he has decided to make plans for it now.
// The vacation consists of N days. For each i (1≤i≤N),
// Taro will choose one of the following activities and do it on the i-th day:
//   A: Swim in the sea. Gain a[i] points of happiness.
//   B: Catch bugs in the mountains. Gain b[i] points of happiness.
//   C: Do homework at home. Gain c[i] points of happiness.
// As Taro gets bored easily, he cannot do the same activities for two or more consecutive days.
//
// Find the maximum possible total points of happiness that Taro gains.
//
//  Time Complexity: O(N)
//

#include <bits/stdc++.h>

using namespace std;

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

    vector<vector<int>> happy(n, vector<int>(3));
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < 3; k++) {
            scanf("%d", &happy[i][k]);
        }
    }

    vector<vector<int>> dp(n, vector<int>(3, 0));

    for (int k = 0; k < 3; k++) {
        dp[0][k] = happy[0][k];
    }

    for (int i = 1; i < n; i++) {
        for (int k = 0; k < 3; k++) {
            int ans = 0;

            for (int l = 0; l < 3; l++) {
                if (l != k) ans = max(ans, dp[i-1][l]);
            }

            dp[i][k] = happy[i][k] + ans;
        }
    }

    int ans = 0;
    for (int i = 0; i < 3; i++) {
        ans = max(ans, dp[n-1][i]);
    }

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

```cpp
//
//   There are N items, numbered 1,2, ... N. For each i (1 ≤ i ≤ N),
//   Item i has a weight of w[i] and a value of v[i]
//   Taro has decided to choose some of the N items and carry them home in a knapsack.
//   The capacity of the knapsack is W, which means that the sum of the weights
//   of items taken must be at most W.
//
//   Find the maximum possible sum of the values of items that Taro takes home.
//
//   Constraints:
//     1 ≤ N ≤ 100
//     1 ≤ W ≤ 10^5
//     1 ≤ w[i] ≤ W
//     1 ≤ v[i] ≤ 10^9
//
//   Time Complexity: O(NW)
//


#include <bits/stdc++.h>
#define ll long long

using namespace std;

int main() {
    int n, W;
    cin >> n >> W;

    vector<vector<ll>> dp(n+1, vector<ll>(W+1, 0));
    vector<int> weights(n+1), values(n+1);

    for (int i = 1; i ≤ n; i++) {
        cin >> weights[i] >> values[i];
    }

    for (int i = 0; i ≤ n; i++) {
        for (int w = 0; w ≤ W; w++) {
            if (i == 0 || w == 0) dp[i][w] = 0;
            else if (w - weights[i] ≥ 0) {
                // you can take that item
                dp[i][w] = max(dp[i-1][w], dp[i-1][w-weights[i]] + values[i]);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }

    cout << dp[n][W] << endl;
    return 0;
}
```

```cpp
//
//   There are N items, numbered 1,2, ... N. For each i (1 ≤ i ≤ N),
//   Item i has a weight of w[i] and a value of v[i]
//   Taro has decided to choose some of the N items and carry them home in a knapsack.
//   The capacity of the knapsack is W, which means that the sum of the weights
//   of items taken must be at most W.
//
//   Find the maximum possible sum of the values of items that Taro takes home.
//
//   Constraints:
//     1 ≤ N ≤ 100
//     1 ≤ W ≤ 10^9
//     1 ≤ w[i] ≤ W
//     1 ≤ v[i] ≤ 10^3
//
//   Time Complexity: O(N * max(v[i]))
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

int N, W;
vector<int> weights, values;
vector<vector<ll>> dp(101, vector<ll>(100001, -1));

// returns the weight
ll capacity(int i, int value) {
    if (value ≤ 0) return 0;
    if (i == N) return 1e12;
    if (dp[i][value] ≠ -1) return dp[i][value];

    dp[i][value] = min(
        capacity(i+1, value),
        capacity(i+1, value - values[i]) + weights[i]
    );

    return dp[i][value];
}

int main() {
    cin >> N >> W;

    weights.resize(N);
    values.resize(N);

    int sum_values = 0;
    for (int i = 0; i < N; i++) {
        cin >> weights[i] >> values[i];
        sum_values += values[i];
    }

    for (int value = sum_values; value ≥ 1; value--) {
        if (capacity(0, value) ≤ W) {
            cout << value << endl;
            return 0;
        }
    }

    cout << 0 << endl;
    return 0;
}
```

```cpp
//
//   You are given strings s and t.
//   Find one longest string that is a subsequence (non-continuous) of both s and t.
//
//   Time Complexity: O(NM)
//

#include <bits/stdc++.h>

using namespace std;

int main() {
    string s, t;
    cin >> s >> t;

    int n = (int)s.size(), m = (int)t.size();

    vector<vector<int>> dp(n, vector<int>(m, 0));

    dp[0][0] = (s[0] == t[0]);
    for (int i = 1; i < n; i++) dp[i][0] = max(dp[i-1][0], (int)(s[i] == t[0]));
    for (int i = 1; i < m; i++) dp[0][i] = max(dp[0][i-1], (int)(s[0] == t[i]));

    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {
            dp[i][j] = max({
                dp[i-1][j-1] + (s[i] == t[j]),
                dp[i-1][j],
                dp[i][j-1]
            });
        }
    }

    // this is the length of the LCS
    // cout << dp[n-1][m-1] << endl;

    if (dp[n-1][m-1] == 0) {
        cout << "" << endl;
        return 0;
    }

    int i = n-1;
    int j = m-1;

    string ans = "";
    while (i >= 0 && j >= 0) {
        if (s[i] == t[j]) {
            ans += s[i];
            i--; j--;
            continue;
        }

        if (i == 0) j--;
        else if (j == 0) i--;
        else {
            if (dp[i-1][j] > dp[i][j-1]) i--;
            else j--;
        }
    }

    reverse(ans.begin(), ans.end());
    cout << ans << endl;
    return 0;
}
```

```cpp
//
//   Given DAG, find the length of the longest path in G.
//
//   Time Complexity: O(N)
//

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> adj;
vector<int> dp;

int dfs(int node) {
    if (dp[node] != -1) return dp[node];

    int longest = 0;
    for (int i = 0; i < adj[node].size(); i++) {
        longest = max(longest, dfs(adj[node][i]) + 1);
    }

    dp[node] = longest;
    return longest;
}

int main() {
    int n, m;
    cin >> n >> m;

    adj.resize(n);
    dp.resize(n, -1);

    for (int i = 0; i < m; i++) {
        int x, y; cin >> x >> y; x--; y--;
        adj[y].push_back(x);
    }

    int ans = 0;

    // DFS from each vertex
    for (int i = 0; i < n; i++) {
        ans = max(ans, dfs(i));
    }

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

```cpp
//
//   Number of paths through a labyrinth.
//
//   Time Complexity: O(NM) = O(size of the matrix)
//

#include <bits/stdc++.h>

using namespace std;

const int MOD = 1e9 + 7;

vector<vector<bool>> board;
vector<vector<int>> dp;

int main() {
    int n, m;
    cin >> n >> m;

    board.resize(n, vector<bool>(m, true));
    dp.resize(n, vector<int>(m, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            char c; cin >> c;
            board[i][j] = (c == '.');
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (i == 0 && j == 0) dp[i][j] = 1;
            else if (i == 0) dp[i][j] = dp[i][j-1] * board[i][j];
            else if (j == 0) dp[i][j] = dp[i-1][j] * board[i][j];
            else {
                dp[i][j] = (dp[i-1][j] + dp[i][j-1]) * board[i][j] % MOD;
            }
        }
    }

    cout << dp[n-1][m-1] << endl;
    return 0;
}
```

```cpp
//
//  Let N be a positive odd number.
//  There are N coins, numbered 1,2,…,N. For each i (1≤i≤N), when Coin i is tossed,
//  it comes up heads with probability p[i] and tails with probability 1-p[i] tails.
//  Taro has tossed all the N coins. Find the probability of having more heads than tails.
//
//  Time Complexity: O(N^2)
//

#include <bits/stdc++.h>

using namespace std;

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

    // everything in this problem
    // is with 1-based indexing
    // for simplicity
    vector<double> p(n+1);
    for (int i = 1; i <= n; i++) {
        cin >> p[i];
    }

    vector<vector<double>> dp(n+1, vector<double>(n+1, 0));
    dp[1][0] = 1 - p[1];
    dp[1][1] = p[1];

    for (int i = 2; i <= n; i++) {
        for (int j = 0; j <= i; j++) {
            // dp[i][j] - the probability of
            //            getting j heads out of
            //            the coins to the i-th index

            if (j > 0)
                dp[i][j] += dp[i-1][j-1] * p[i];

            dp[i][j] += dp[i-1][j] * (1 - p[i]);
        }
    }

    double ans = 0;
    for (int i = n/2 + 1; i <= n; i++) {
        ans += dp[n][i];
    }

    cout << setprecision(16) << ans << endl;
    return 0;
}
```

```cpp
//
//  There are N dishes, numbered 1,2 ... N. Initially, for each i (1≤i≤N),
//  Dish i has a[i] (1 ⩽ a[i] ⩽ 3) pieces of sushi on it.
//  Taro will perform the following operation repeatedly until all the pieces of sushi are eaten:
//    - Roll a die that shows the numbers 1,2 ... N with equal probabilities, and let i be the outcome.
//      If there are some pieces of sushi on Dish i, eat one of them; if there is none, do nothing.
//  Find the expected number of times the operation is performed before all the pieces of sushi are eaten.
//
//     Time Complexity: O(N^3)
//

#include <bits/stdc++.h>

using namespace std;

int n;  // n is also the "sum of s[i]"
vector<int> mp(4, 0);

const int MAX = 310;
double dp[MAX][MAX][MAX];

double dp_f(int x, int y, int z) {
    if (dp[x][y][z] != -1) {
        return dp[x][y][z];
    }

    if (x == 0 && y == 0 && z == 0) return 0;

    double zero = n - x - y - z;
    double rolls = (n - zero) / n + zero * (2 * n - zero) / (n * (n - zero));
    double current_sum = rolls;

    if (x) {
        double weight = x / (n - zero);
        current_sum += weight * dp_f(x - 1, y, z);
    }

    if (y) {
        double weight = y / (n - zero);
        current_sum += weight * dp_f(x + 1, y - 1, z);
    }

    if (z) {
        double weight = z / (n - zero);
        current_sum += weight * dp_f(x, y + 1, z - 1);
    }

    dp[x][y][z] = current_sum;
    return current_sum;
}

int main() {
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            for (int k = 0; k < MAX; k++) {
                dp[i][j][k] = -1;
            }
        }
    }

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);
        mp[x]++;
    }

    cout << setprecision(16) << dp_f(mp[1], mp[2], mp[3]) << endl;
    return 0;
}
```

```cpp
//
// There is a set A = {a_1, a_2 ... }  consisting of N positive integers.
// Taro and Jiro will play the following game against each other.
// Initially, we have a pile consisting of K stones.
// The two players perform the following operation alternately, starting from Taro:
//    - Choose an element x in A, and remove exactly x stones from the pile.
// A player loses when he becomes unable to play.
//
// Assuming that both players play optimally, determine the winner.
// Time Complexity: O(NK)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

inline void umin(int& a, int b) { a = min(a, b); }
inline void umax(int& a, int b) { a = max(a, b); }

int main() {
    int n, k;
    cin >> n >> k;

    vector<int> a(n);
    for (int i = 0; i < n; i++) cin >> a[i];

    vector<vector<int>> dp(k+1, vector<int>({ 1000, -1000 }));
    // player 0 wants to minimize
    // player 1 wants to maximize

    for (int i = 0; i <= k; i++) {
        for (int j = 0; j < n; j++) {
            if (i - a[j] < 0) {
                umin(dp[i][0], 1);
                umax(dp[i][1], 0);
            } else {
                umin(dp[i][0], dp[i-a[j]][1]);
                umax(dp[i][1], dp[i-a[j]][0]);
            }
        }
    }

    cout << (dp[k][0] ? "Second" : "First") << endl;
    return 0;
}
```

```cpp
//
//  Taro and Jiro will play the following game against each other.
//  Initially, they are given a sequence a = (a[1], a[2], … , a[n])
//  Until a becomes empty, the two players perform the following operation alternately,
//  starting from Taro:
//    - Remove the element at the beginning or the end of a.
//      The player earns x points, where x is the removed element.
//
//  Let X and Y be Taro's and Jiro's total score at the end of the game, respectively.
//  Taro tries to maximize X-Y, while Jiro tries to minimize X-Y.
//  Assuming that the two players play optimally, find the resulting value of X-Y.
//
//     Time Complexity: O(N^2)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

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

    vector<ll> awards(n);
    vector<vector<ll>> dp(n, vector<ll>(n, 0));
    for (int i = 0; i < n; i++) {
        cin >> awards[i];

        // BASE CASE:
        dp[i][i] = awards[i];
    }

    // interval starting and ending at position i = score is awards[i]
    // solving for subintervals

    ///  _ _ _ _ _ _ _ _
    ///    ⟷
    ///  ⟷
    ///    ⟷

    for (int size = 1; size < n; size++) { // size of interval
        for (int start = 0; start < n - size; start++) { // starting position of interval
            if (size % 2 == 0) { // Taro (max)
                //  START - L
                //  START+SIZE = R
                dp[start][start+size] =
                    max(dp[start][start+size-1] + awards[start+size],
                        dp[start+1][start+size] + awards[start]);
            } else { // Jiro (min)
                dp[start][start+size] =
                    min(dp[start][start+size-1] - awards[start+size],
                        dp[start+1][start+size] - awards[start]);
            }
        }
    }

    // dp[0][n-1]
    cout << (n % 2 == 0 ? -1LL : 1LL) * dp[0][n-1] << endl;
    return 0;
}
```

```cpp
//
//  There are N children, numbered 1,2,…,N.
//  They have decided to share K candies among themselves.
//  Here, for each i (1≤i≤N), Child i must receive between 0 and a[i] candies (inclusive).
//  Also, no candies should be left over.
//
//  Find the number of ways for them to share candies, modulo 10^9 + 7.
//  Here, two ways are said to be different when there
//  exists a child who receives a different number of candies.
//
//    Time Complexity: O(NK)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

const int MOD = 1e9 + 7;

int main() {
    int n, k;
    cin >> n >> k;

    vector<int> a(n);
    for (int i = 0; i < n; i++) cin >> a[i];

    vector<vector<int>> dp(n, vector<int>(k+1, 0));
    for (int i = k; i >= k - a[0]; i--) dp[0][i] = 1;

    for (int i = 1; i < n; i++) {
        vector<int> pref(k+1);
        pref[k] = dp[i-1][k];
        for (int j = k-1; j >= 0; j--) {
            pref[j] = (pref[j+1] + dp[i-1][j]) % MOD;
        }

        for (int j = 0; j <= k; j++) {
            // dp[i][j] = dp[i-1][j+(0...a[i])]
            dp[i][j] = (pref[j] - (j+a[i]+1 <= k ? pref[j+a[i]+1] : 0) + MOD) % MOD;
        }
    }

    cout << dp[n-1][0] << endl;
    return 0;
}
```

```cpp
//
//  There are N slimes lining up in a row. Initially, the i-th slime from the left has a size of a[i].
//  Taro is trying to combine all the slimes into a larger slime.
//  He will perform the following operation repeatedly until there is only one slime:
//     - Choose two adjacent slimes, and combine them into a new slime.
//       The new slime has a size of x+y, where x and y are the sizes
//       of the slimes before combining them.
//       Here, a cost of x+y is incurred. The positional relationship
//       of the slimes does not change while combining slimes.
//  Find the minimum possible total cost incurred.
//
//    Time Complexity: O(N^3)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

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

    vector<int> s(n);
    vector<ll> prefix(n);
    for (int i = 0; i < n; i++) {
        cin >> s[i];
        prefix[i] = (i-1 >= 0 ? prefix[i-1]: 0LL) + s[i];
    }

    vector<vector<ll>> dp(n, vector<ll>(n, 0));

    for (int size = 1; size <= n; size++) {
        for (int start = 0; start < n - size; start++) {
            int L = start;
            int R = start + size;

            ll best = LLONG_MAX;
            for (int i = 0; i < R - L; i++) {
                best = min(best, dp[L][L+i] + dp[L+i+1][R]);
            }

            dp[L][R] = best + (prefix[R] - (L-1 >= 0 ? prefix[L-1] : 0));
        }
    }

    cout << dp[0][n-1] << endl;
    return 0;
}
```

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

#include <bits/stdc++.h>
#define ll long long

using namespace std;

const int MOD = 1e9 + 7;
const int maxSetCount = (1 << 21) + 1;

bool match[25][25];
int dp[maxSetCount][21];
int n;

int main() {
    cin >> n;

    // memset the dp
    for (int i = 0; i < maxSetCount; i++) {
        for (int j = 0; j < n; j++) {
            dp[i][j] = 0;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> match[i][j];
        }
    }

    int full_set = (1 << n) - 1;

    // base case
    for (int i = 0; i < n; i++) {
        dp[0][i] = 1;
    }

    for (int set = 0; set < (1 << n); set++) {
        int count = __builtin_popcount(set);
        for (int last = 0; last < n; last++) {
            if (((1 << last) & set) == 0 && match[last][count]) {
                int next_set = set ^ (1 << last);

                for (int i = 0; i < n; i++) {
                    dp[next_set][i] = (dp[next_set][i] + dp[set][last]) % MOD;
                }
            }
        }
    }

    cout << dp[full_set][0] << endl;
    return 0;
}
```

```cpp
//
//   There is a tree with N vertices, numbered 1,2, ... N.
//   For each i (1 ⩽ i ⩽ N-1), the i-th edge connects Vertex x[i] and y[i].
//   Taro has decided to paint each vertex in white or black.
//   Here, it is not allowed to paint two adjacent vertices both in black.
//
//   Find the number of ways in which the vertices can be painted, modulo 10^9 + 7.
//
//      Time Complexity: O(N)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

const int MOD = 1e9 + 7;

vector<vector<int>> adj;
vector<vector<int>> dp;

int dfs(int node, int black, int parent = -1) {
    if (dp[node][black] != -1) return dp[node][black];

    dp[node][black] = 1;
    for (int i = 0; i < (int)adj[node].size(); i++) {
        int next_node = adj[node][i];

        if (next_node == parent) continue;

        if (black) {
            dp[node][1] = ((ll)dp[node][1] * dfs(next_node, 0, node)) % MOD;
        } else {
            ll subtree_white = dfs(next_node, 0, node);
            ll subtree_black = dfs(next_node, 1, node);

            dp[node][0] = ((ll)dp[node][0] *
                ((subtree_white + subtree_black) % MOD)) % MOD;
        }
    }

    return dp[node][black];
}

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

    adj.resize(n);
    dp.resize(n, vector<int>(2, -1));

    for (int i = 0; i < n-1; i++) {
        int x, y;
        cin >> x >> y;
        x--; y--;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }

    cout << ((ll)dfs(0, 0) + dfs(0, 1)) % MOD << endl;
    return 0;
}
```

```cpp
//
//   There are N flowers arranged in a row. For each i (1≤i≤N), the height and the beauty of
//   the i-th flower from the left is h[i] and a[i], respectively.
//   Here, h[1], h[2], ... h[N] are all distinct.
//   Taro is pulling out some flowers so that the following condition is met:
//      - The heights of the remaining flowers are monotonically increasing from left to right.
//   Find the maximum possible sum of the beauties of the remaining flowers.
//
//   Constrains:
//     h[i] ⩽ N
//
//   Time Complexity: O(N * log(N))
//


#include <bits/stdc++.h>
#define ll long long

using namespace std;

// Max Fenwick Tree
struct FenwickTree {
    vector<ll> fwt;

    FenwickTree(int n) {
        fwt.resize(n, 0);
    }

    void maxFWT(int ind, ll val = 1) {
        for (ind++; ind < fwt.size(); ind+=ind&-ind)
            fwt[ind] = max(fwt[ind], val);
    }

    ll getFWT(int ind) {
        ll s = 0;
        for (ind++; ind > 0; ind-=ind&-ind)
            s = max(s, fwt[ind]);
        return s;
    }
};

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

    vector<ll> h(n), a(n);
    for (int i = 0; i < n; i++) scanf("%d", &h[i]);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);

    FenwickTree tree(n+10);

    for (int i = 0; i < n; i++) {
        // query best = max (1 ⇒ h[i] - 1)
        // update at h[i] = best + a[i];

        ll best = tree.getFWT(h[i]); // get maximum
        ll new_best = best + a[i];
        tree.maxFWT(h[i], new_best);
    }

    cout << tree.getFWT(n) << endl;
    return 0;
}
```

```cpp
//
//   There is a simple directed graph G with N vertices, numbered 1,2,…,N.
//   For each i and j (1 ⩽ i,j ⩽ N), you are given an integer a[i][j]
//   that represents whether there is a directed edge from Vertex i to j if a[i][j] is set.
//   Find the number of different directed paths of length K in G, modulo 10^9 + 7
//   We will also count a path that traverses the same edge multiple times.
//
//   Constraints:
//     N ⩽ 50
//     K ⩽ 10^18
//
//   Time Complexity: O(N^3 * log(K))
//

#include <bits/stdc++.h>
#define MX vector<vector<T>>
#define MOD 1000000007
#define ll long long

using namespace std;

template<typename T>
MX mult(MX a, MX b) {
    MX result(a.size(), vector<T>(a.size(), 0));
    for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < a.size(); j++) {
            for (int k = 0; k < a.size(); k++) {
                result[i][j] = (result[i][j] + (a[i][k] * b[k][j]) % MOD) % MOD;
            }
        }
    }

    return result;
}

template<typename T>
MX binary(MX a, ll n) {
    MX result(a.size(), vector<T>(a.size(), 0));
    for (int i = 0; i < a.size(); i++) result[i][i] = 1;

    while (n > 0) {
        if (n % 2 == 1) {
            result = mult(result, a);
            n--;
        }

        a = mult(a, a);
        n /= 2;
    }

    return result;
}

int main() {
    ll n, k;
    cin >> n >> k;

    vector<vector<long long>> mat(n, vector<long long>(n, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> mat[i][j];
        }
    }

    auto result = binary(mat, k);

    long long sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum = (sum + result[i][j]) % MOD;
        }
    }

    cout << sum << endl;
    return 0;
}
```

```cpp
//
//   Find the number of integers between 1 and K (inclusive)
//   satisfying the following condition, modulo 10^9 + 7:
//     - The sum of the digits in base ten is a multiple of D.
//
//     Time Complexity: O(ND)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

const int MOD = 1e9 + 7;
inline void uadd(int& a, int b) {
    a += b;
    if (a >= MOD) a -= MOD;
}

int main() {
    string s; cin >> s;
    int v; cin >> v;
    int n = s.size();

    if (n == 1) {
        int ans = 0;
        for (int i = 1; i <= (s[0] - '0'); i++) {
            if (i % v == 0) ans++;
        }
        cout << ans << endl;
        return 0;
    }

    vector<int> pref(n);
    pref[0] = (s[0] - '0') % v;
    for (int i = 1; i < n; i++) {
        pref[i] = (pref[i-1] + s[i] - '0') % v;
    }

    // backwards dynamic programming
    vector<vector<int>> bw(n, vector<int>(v, 0));
    int ans = 0;

    // Base case
    for (int d = 0; d < 10; d++) {
        bw[n-1][d%v] += 1;
        if (d <= s[n-1] - '0' && n-2 >= 0 && (pref[n-2] + d) % v == 0) ans++;
    }

    for (int i = n - 2; i >= 0; i--) {
        for (int d = 0; d <= (i == 0 ? s[0] - '0' - 1 : 9); d++) {
            for (int j = 0; j < v; j++) {
                uadd(bw[i][(j+d)%v], bw[i+1][j]);
            }
        }

        if (i-1 >= 0) {
            for (int d = 0; d < s[i] - '0'; d++) {
                uadd(ans, bw[i+1][(v - ((pref[i-1] + d + 100 * v) % v)) % v]);
            }
        }
    }

    cout << ((ll)ans + bw[0][0] - 1 + MOD) % MOD << endl;
    return 0;
}
```

```cpp
//
//   Let N be a positive integer. You are given a string s of length N-1, consisting of < and >.
//   Find the number of permutations (p[1], p[2], ... p[N]) of (1,2, ... N)
//   that satisfy the following condition, modulo 10^9 + 7
//   For each i (1 ≤ i ≤ N-1), p[i] < p[i+1] if the i-th character in s is '<'
//                             and p[i] > p[i+1] if the i-th character in s is '>'
//
//      Time Complexity: O(N^2)
//

#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int MOD = 1e9 + 7;

inline void uadd(int& a, int b) {
    a = (a + b);
    if (a >= MOD) a -= MOD;
}

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

    string s;
    cin >> s;
    s = " " + s;

    vector<vector<int>> dp(n, vector<int>(n, 0));
    dp[0][0] = 1;

    // dp[i][j] - number of ways to order the first i elements ending with item j
    for (int i = 1; i < n; i++) {
        int pref = 0;
        for (int j = 0; j <= i; j++) { // last number
            if (s[i] == '<') {
                uadd(dp[i][j], pref);
                uadd(pref, dp[i-1][j]);
            } else {
                uadd(dp[i][i-j], pref);
                uadd(pref, dp[i-1][i-j-1]);
            }
        }
    }

    int ans = 0;
    for (int i = 0; i < n; i++) {
        uadd(ans, dp[n-1][i]);
    }

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

```cpp
//
//   There are N rabbits, numbered 1,2, ... N.
//   For each i,j (1 ⩽ i,j ⩽ N), the compatibility of
//   Rabbit i and j is described by an integer a[i][j]
//     a[i][i] = 0
//     a[i][j] = a[j][i]
//
//   Taro is dividing the N rabbits into some number of groups.
//   Here, each rabbit must belong to exactly one group.
//   After grouping, for each i and j (1 ⩽ i < j ⩽ N),
//   Taro earns a[i][j] points if Rabbit i and j belong to the same group.
//
//   Find Taro's maximum possible total score.
//
//      Time Complexity: O(2^N * N^2 + 2^(2N))
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

const int mxn = (1 << 16) + 10;
ll dp[mxn];

int main() {
    for (int i = 0; i < mxn; i++) {
        dp[i] = 0;
    }

    int n;
    cin >> n;

    vector<vector<int>> a(n, vector<int>(n, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    for (int mask = 0; mask < (1 << n); mask++) {
        ll score = 0;
        for (int bit1 = 0; bit1 < n; bit1++) {
            for (int bit2 = bit1 + 1; bit2 < n; bit2++) {
                if (mask & (1 << bit1) && mask & (1 << bit2)) {
                    score += a[bit1][bit2];
                }
            }
        }

        dp[mask] = max(dp[mask], score);
    }

    for (int mask1 = 0; mask1 < (1 << n); mask1++) {
        for (int mask2 = mask1 + 1; mask2 < (1 << n); mask2++) {
            if ((mask1 & mask2) == 0) {
                dp[mask1 | mask2] = max(dp[mask1 | mask2], dp[mask1] + dp[mask2]);
            }
        }
    }

    cout << dp[(1 << n) - 1] << endl;
    return 0;
}
```

```cpp
//
//   There is a tree with N vertices, numbered 1,2, ... N.
//   For each i (1 ≤ i ≤ N-1), the i-th edge connects Vertex x[i] and y[i].
//   Taro has decided to paint each vertex in white or black,
//   so that any black vertex can be reached from any other black vertex
//   by passing through only black vertices.
//   You are given a positive integer M. For each v (1 ≤ v ≤ N), answer the following question:
//     - Assuming that Vertex v has to be black,
//       find the number of ways in which the vertices can be painted, modulo M.
//
//     Time Complexity: O(N+M)
//

#include <bits/stdc++.h>
#define ll long long
using namespace std;

int n;
ll MOD;

vector<ll> in, out, excepts;
vector<vector<int>> adj;

// ans[node] = in[node] * out[node]
int dfs_in(int node, int parent = -1) {
    int size = adj[node].size();
    vector<ll> ans_child(size, 1);
    vector<ll> dp_pref(size, 1), dp_suf(size, 1);
    for (int i = 0; i < size; i++) {
        int next_node = adj[node][i];
        if (next_node == parent) {
            dp_pref[i] = (i-1 >= 0 ? dp_pref[i-1] : 1);
            continue;
        }

        int black_ways = dfs_in(next_node, node);
        ans_child[i] = (black_ways + 1) % MOD;
        in[node] = (in[node] * (black_ways + 1) % MOD) % MOD;
        dp_pref[i] = ((i-1 >= 0 ? dp_pref[i-1] : 1) * (black_ways + 1) % MOD) % MOD;
    }

    if (size > 1) {
        // calculate the suffixes
        dp_suf[size-1] = ans_child[size-1];
        for (int i = size - 2; i >= 0; i--) {
            dp_suf[i] = (dp_suf[i+1] * ans_child[i]) % MOD;
        }

        // except the i-th child
        for (int i = 0; i < size; i++) {
            int child = adj[node][i];
            if (child == parent) continue;

            ll except =
                ((i-1 >= 0 ? dp_pref[i-1]: 1) *
                (i+1 < size ? dp_suf[i+1] : 1)) % MOD;

            excepts[child] = except;
        }
    }

    return in[node];
}

void dfs_out(int node, int parent = -1) {
    int size = adj[node].size();
    for (int i = 0; i < size; i++) {
        int child = adj[node][i];
        if (child == parent) continue;

        if (parent == -1) out[child] = (excepts[child] + 1) % MOD;
        else out[child] = ((excepts[child] * out[node] % MOD) + 1) % MOD;

        dfs_out(child, node);
    }
}

int main() {
    cin >> n >> MOD;
    in.resize(n, 1);
    out.resize(n, 1);
    adj.resize(n);
    excepts.resize(n, 1);
    for (int i = 0; i < n-1; i++) {
        int x, y;
        cin >> x >> y;
        x--; y--;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }

    dfs_in(0); dfs_out(0);
    // root node doesn't have out[root]
    cout << in[0] % MOD << "\n";
    for (int i = 1; i < n; i++) {
        cout << (in[i] * out[i]) % MOD << "\n";
    }
    return 0;
}
```

```cpp
//
//   Consider a string of length N consisting of 0 and 1.
//   The score for the string is calculated as follows:
//     - For each i (1 ⩽ i ⩽ M), a is added to the score
//       if the string contains 1 at least once
//       between the l-th and r-th characters (inclusive).
//   Find the maximum possible score of a string.
//
//      Time Complexity: O(N log N)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

inline void uadd(ll& a, ll b) { a += b; }
const ll NEG_INF = LLONG_MIN / 10;

struct LazySegmentTree {
    int N = -1;

    struct Node {
        ll value = NEG_INF; // default
        ll lazy = 0;
        Node operator+(const Node &other) {
            return {
                max(this->value, other.value)
            };
        }
    };

    vector<Node> seg;
    LazySegmentTree(int N) {
        this->N = N;
        int sz = 1 << 21;
        seg.resize(sz);
    }

    void push(int ind, int l, int r) {
        if (seg[ind].lazy == 0) return;
        if (seg[ind].value == NEG_INF) seg[ind].value = seg[ind].lazy;
        else uadd(seg[ind].value, seg[ind].lazy);
        if (r - l != 1) {
            uadd(seg[2*ind+1].lazy, seg[ind].lazy);
            uadd(seg[2*ind+2].lazy, seg[ind].lazy);
        }
        seg[ind].lazy = 0;
    }

    void updateRange(int b, int e, ll x, int ind = 0, int l = 0, int r = -1) {
        if (r == -1) r = N;
        push(ind, l, r);
        if (e⩽l || b⩾r) { return; }
        if (b⩽l && r⩽e) {
            uadd(seg[ind].lazy, (ll)x);
            push(ind, l, r);
            return;
        }

        int m = (l + r) / 2;
        updateRange(b, e, x, 2 * ind + 1, l, m);
        updateRange(b, e, x, 2 * ind + 2, m, r);
        seg[ind] = seg[2*ind+1] + seg[2*ind+2];
    }

    // result in [b,e) of node that covers [l,r)
    Node askLR(int b, int e, int ind = 0, int l = 0, int r = -1) {
        if (r == -1) r = N;
        push(ind, l, r);
        if (l⩾e || r⩽b) return {  }; // empty
        if (l⩾b && r⩽e) return seg[ind];
        int m = (l + r) / 2;
        return askLR(b, e, ind * 2 + 1, l, m) + askLR(b, e, ind * 2 + 2, m, r);
    }
};

struct query {
    ll l, r, v;
};
```

```cpp

int main() {
    int n, m;
    cin >> n >> m;

    vector<query> q(m);
    vector<vector<int>> events(n + 5);
    for (int i = 0; i < m; i++) {
        int l, r, v;
        cin >> l >> r >> v;
        q[i] = { l, r, v };
        events[l].push_back(i);
        events[r].push_back(~i);
    }

    // dp[i] - the best score of the prefix to index i if s[i] == 1
    // v[q] - value of the q-th query
    // dp[i] = max(dp[j] + sum(v[q])) for all j < i ,
    //         for all q if l[q] ⩽ i ⩽ r[q] && !(l[q] ⩽ j ⩽ r[q]))

    // when you meet a left edge  → tree.add( [ 0, l[q] - 1 ],  v[q] )
    // when you meet a right edge → tree.add( [ 0, l[q] - 1 ], -v[q] )

    // explanation:
    //  when you meed a left edge, this interval gets activated, so
    //  for every dp which is up to this point, we want to add this interval's value.
    //  when this interval dies out, we remove it from the dp tree, because other dps
    //  to the right will have this edge included if it was worth it.

    LazySegmentTree tree(n + 10);
    for (int i = 1; i ⩽ n; i++) {
        for (auto& event: events[i]) {
            if (event ⩾ 0) {
                tree.updateRange(0, q[event].l, q[event].v);
            }
        }

        ll new_dp = tree.askLR(0, i).value;
        tree.updateRange(i, i+1, new_dp);

        for (auto& event: events[i]) {
            if (event < 0) {
                tree.updateRange(0, q[~event].l, -q[~event].v);
            }
        }
    }

    cout << tree.askLR(0, n+1).value << endl;
    return 0;
}
```

```cpp
//
//   There are N blocks, numbered 1,2, ... N. For each i (1 ⩽ i ⩽ N),
//   Block i has a weight of w[i], a solidness of s[i] and a value of v[i].
//   Taro has decided to build a tower by choosing some of the N blocks
//   and stacking them vertically in some order.
//   Here, the tower must satisfy the following condition:
//      - For each Block i contained in the tower,
//        the sum of the weights of the blocks stacked above it is not greater than s[i].
//   Find the maximum possible sum of the values of the blocks contained in the tower.
//
//     Time Complexity: O(N * max(W))
//
//     ! Note: MXW HAS TO BE AT LEAST 2 * max(w[i]) !
//     ! Note: Memory usage can get too big quickly !
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

struct block {
    ll weight, solidness, value;

    bool operator<(const block& other) {
        return (weight + solidness) > (other.weight + other.solidness);
        // we want to exchange the blocks if
        //     w[top] > w[bottom]
        //  &  s[top] > s[bottom] ⇒
        //     w[top] + s[top] > w[bottom] + s[bottom]
    }
};

inline void umax(ll& a, ll b) {
    if (a < b) a = b;
}

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

    vector<block> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i].weight >> a[i].solidness >> a[i].value;
    }

    sort(a.begin(), a.end());

    const int mxw = 3e4 + 10;
    vector<vector<ll>> dp(n, vector<ll>(mxw, 0));
    // dp[i][j] - to the i-th box, if j weight remains

    ll ans = 0;

    // Base case
    for (int w = 0; w ⩽ a[0].solidness; w++) {
        umax(dp[0][w], a[0].value);
        umax(ans, dp[0][w]);
    }

    for (int i = 1; i < n; i++) {
        for (int j = 0; j < mxw; j++) {
            umax(dp[i][j], dp[i-1][j]);
            int weight_remains = min(j - a[i].weight, a[i].solidness);
            if (weight_remains ⩾ 0) {
                umax(dp[i][weight_remains], dp[i-1][j] + a[i].value);
                umax(ans, dp[i][weight_remains]);
            }
        }
    }

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

```cpp
#include <bits/stdc++.h>
#define ll long long

using namespace std;

const int MOD = 1e9 + 7;

struct Binomial {
    int mxn = -1;
    vector<int> inv, fact;

    // inverse = pow(a, mod - 2);
    int inverse(int a) {
        int m = MOD;
        int u = 0, v = 1;
        while (a != 0) {
            int t = m / a;
            m -= t * a;
            swap(a, m);
            u -= t * v;
            swap(u, v);
        }
        assert(m == 1);
        return (u + MOD) % MOD;
    }

    void preprocess() {
        fact[0] = 1;
        for (int i = 1; i < mxn; i++) {
            fact[i] = (ll)fact[i-1] * i % MOD;
        }
    }

    Binomial(int mxn = (int)2e5 + 10) {
        this->mxn = mxn;
        fact.resize(mxn);
        preprocess();
    }

    int C(int n, int k) {
        if (k > n) swap(k, n);
        return ((ll)fact[n] * inverse((ll)fact[k] * fact[n-k] % MOD)) % MOD;
    }
};

int main() {
    int w, h, n;
    cin >> w >> h >> n;

    n += 2;

    vector<pair<int, int>> points(n);
    for (int i = 1; i < n-1; i++) {
        int x, y;
        cin >> x >> y;
        x--; y--;
        points[i] = { x, y };
    }

    points[0] = { 0, 0 };
    points[n-1] = { w - 1, h - 1 };

    sort(points.begin(), points.end());

    Binomial b(2e5 + 10);
    vector<ll> f(n), g(n);
    g[0] = 0;

    for (int i = 1; i < n; i++) {
        int x1, y1;
        tie(x1, y1) = points[i];

        for (int j = i-1; j >= 0; j--) {
            int x2, y2;
            tie(x2, y2) = points[j];

            if (y2 > y1) continue;

            ll X = x1 - x2;
            ll Y = y1 - y2;

            f[i] = (f[i] + (g[j] * b.C(X + Y, Y)) % MOD) % MOD;
        }

        g[i] = (b.C(x1 + y1, y1) - f[i] + MOD) % MOD;
    }

    cout << g[n-1] << endl;
    return 0;
}
```

```cpp
//
//   There are N stones, numbered 1,2, ... N. For each i (1 ⩽ i ⩽ N), the height of Stone i is h[i].
//   Here h[1] < h[2] < [3] ... < h[N] holds.
//   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 one of the following:
//         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.
//   Find the minimum possible total cost incurred before the frog reaches Stone N.
//
//     Time Complexity: O(N)
//

#include <bits/stdc++.h>
#define ll long long

using namespace std;

vector<ll> h, dp;
int n;
ll C;

struct Line {
        ll k, m, p;
        ll eval(ll x) const { return k * x + m; }
};

struct CHT {
        deque<Line> hull;
        static const ll inf = LLONG_MAX;

        ll div(ll a, ll b) { // floored division
                return a / b - ((a ^ b) < 0 && a % b); }

        bool intersect(Line &x, const Line &y) {
                if (x.k == y.k) x.p = x.m > y.m ? inf : -inf;
                else x.p = div(y.m - x.m, x.k - y.k);
                return x.p >= y.p;
        }

    // Add a line with a form of kx + m
        void add(ll k, ll m) {
                Line L = {k, m, 0};
                while ((int) hull.size() >= 2 && (intersect(L, hull.back()),
                        intersect(hull.back(), hull[(int) hull.size() - 2]), L.p < hull.back().p))
                        hull.pop_back();
                hull.push_back(L);
        }

        // query at point x equivalent to cht(x)
        ll query(ll x) {
                while ((int) hull.size() >= 2 && hull[0].eval(x) >= hull[1].eval(x))
                        hull.pop_front();
                return hull[0].eval(x);
        }
};

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

    h.resize(n);
    dp.resize(n);
    for (int i = 0; i < n; i++) cin >> h[i];

    CHT cht;
    auto insert = [&](int i) {
        cht.add(-2LL * h[i], (dp[i] + (h[i] * h[i])));
    };

    dp[0] = 0; insert(0);
    for (int i = 1; i < n; i++) {
        ll x = cht.query(h[i]);
        dp[i] = C + (h[i] * h[i]) + x;

        insert(i);
    }

    cout << dp[n-1] << endl;
    return 0;
}
```