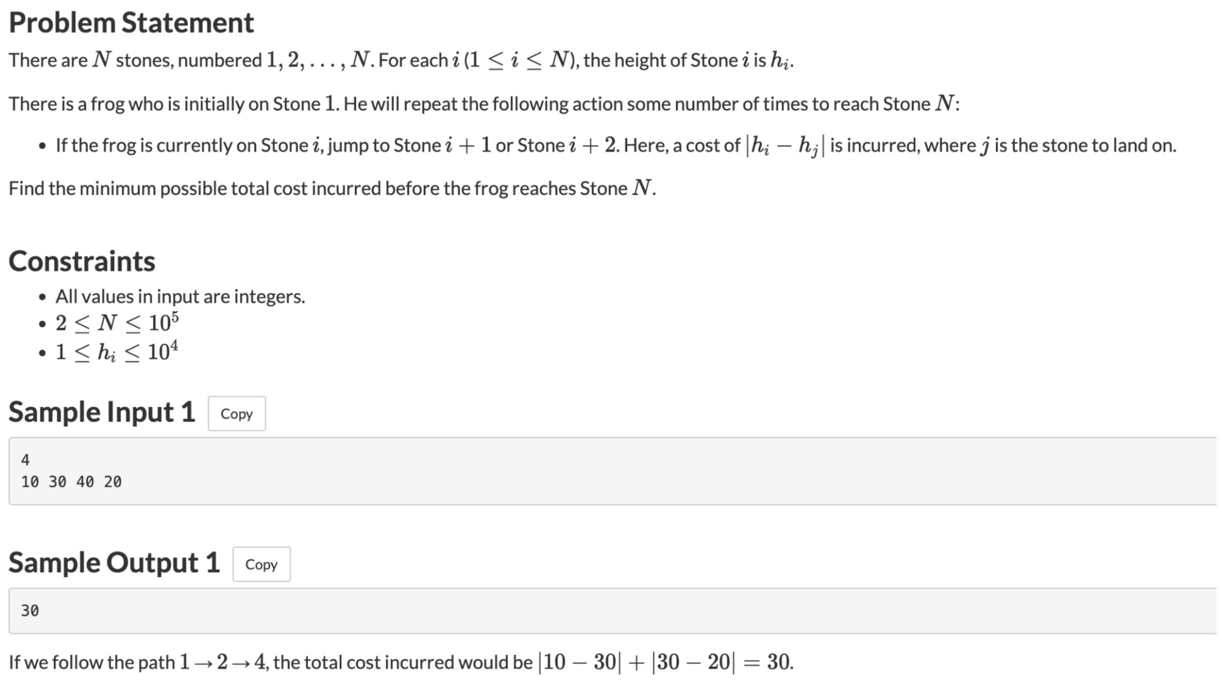
1. Frog1



Problem Statement

There are N stones, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), 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.

Constraints

- All values in input are integers.
- $2 \leq N \leq 10^5$
- $1 \leq h_i \leq 10^4$

Sample Input 1 Copy

```
4
10 30 40 20
```

Sample Output 1 Copy

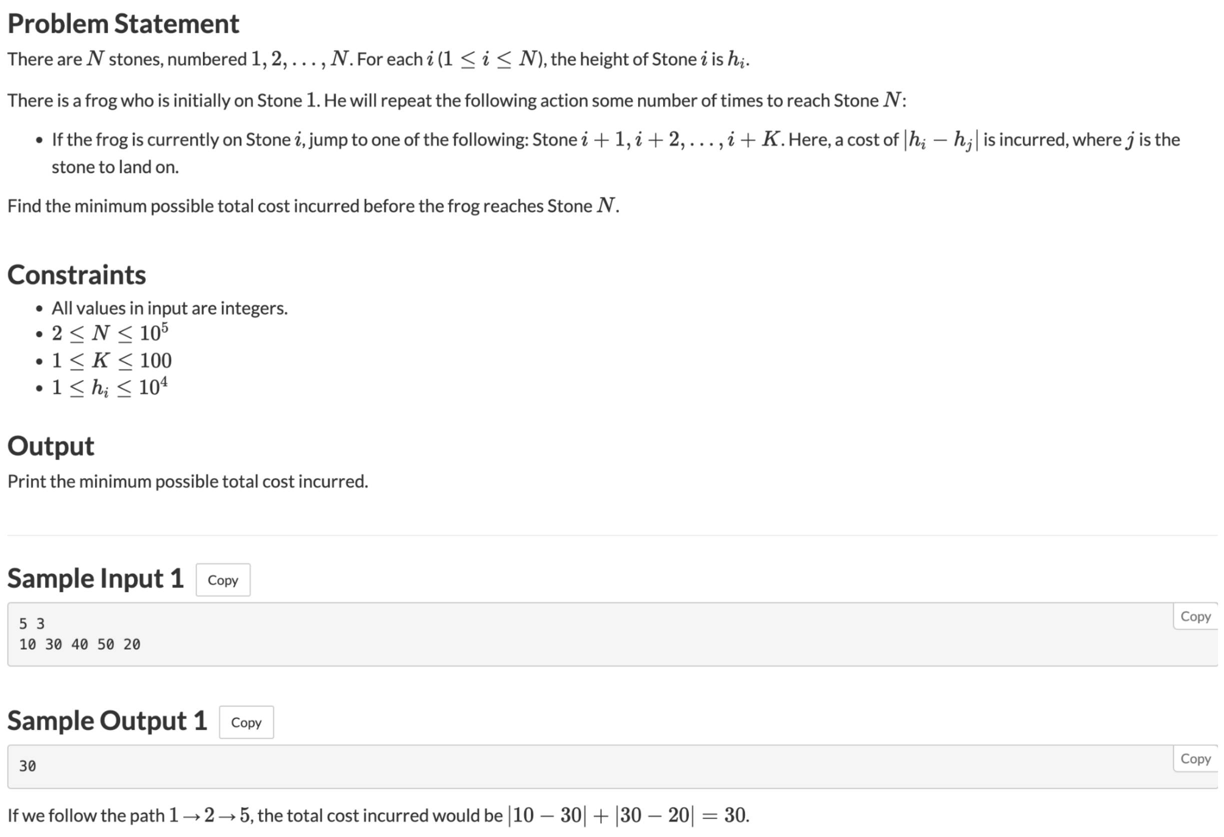
```
30
```

If we follow the path $1 \rightarrow 2 \rightarrow 4$, the total cost incurred would be $|10 - 30| + |30 - 20| = 30$.

SOL:

```
// dp[position] = the minimum cost at 'position'
int n = in.nextInt();
int[] h = new int[n+1];
for (int i = 1; i <= n; i++) h[i] = in.nextInt();

int[] dp = new int[n+1];
dp[0] = 1_000_000_000;
dp[1] = 0;
for (int i = 2; i <= n; i++) {
        dp[i] = Math.min(dp[i-1]+Math.abs(h[i]-h[i-1]),
                dp[i-2]+Math.abs(h[i]-h[i-2]));
}

System.out.println(dp[n]);
```

2. Frog B



Problem Statement

There are N stones, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), 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 one of the following: Stone $i+1, i+2, \ldots, i+K$. 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.

Constraints

- All values in input are integers.
- $2 \leq N \leq 10^5$
- $1 \leq K \leq 100$
- $1 \leq h_i \leq 10^4$

Output

Print the minimum possible total cost incurred.

Sample Input 1

Copy

```
5 3
10 30 40 50 20
```

Copy

Sample Output 1

Copy

```
30
```

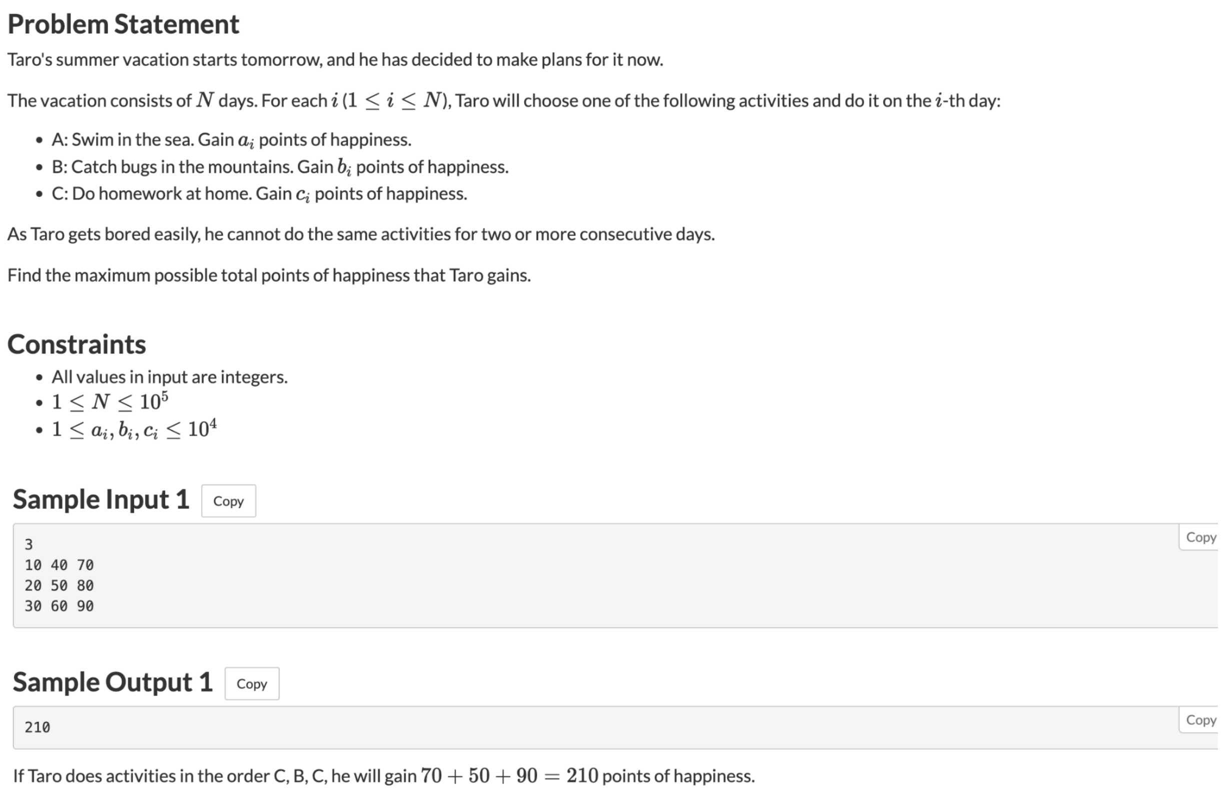
Copy

If we follow the path $1 \to 2 \to 5$, the total cost incurred would be $|10 - 30| + |30 - 20| = 30$.

Code:

```
public static int minCostToReachStoneN(int N, int K, int[] heights) {
      int[] dp = new int[N];
      dp[0] = 0;

      for (int i = 1; i < N; i++) {

         dp[i] = Integer.MAX_VALUE;

         for(int j = 1;j <= K && i-j >= 0;j++) {
            int jumpCost = dp[i-j] + Math.abs(heights[i] - heights[i - j]);
            dp[i] = Math.min(dp[i], jumpCost);
         }
      }

      return dp[N - 1];
   }
```

3. Vacation



Problem Statement

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 $(1 \leq i \leq N)$, 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.

Constraints

- All values in input are integers.
- $1 \leq N \leq 10^5$
- $1 \leq a_i, b_i, c_i \leq 10^4$

Sample Input 1

```
3
10 40 70
20 50 80
30 60 90
```

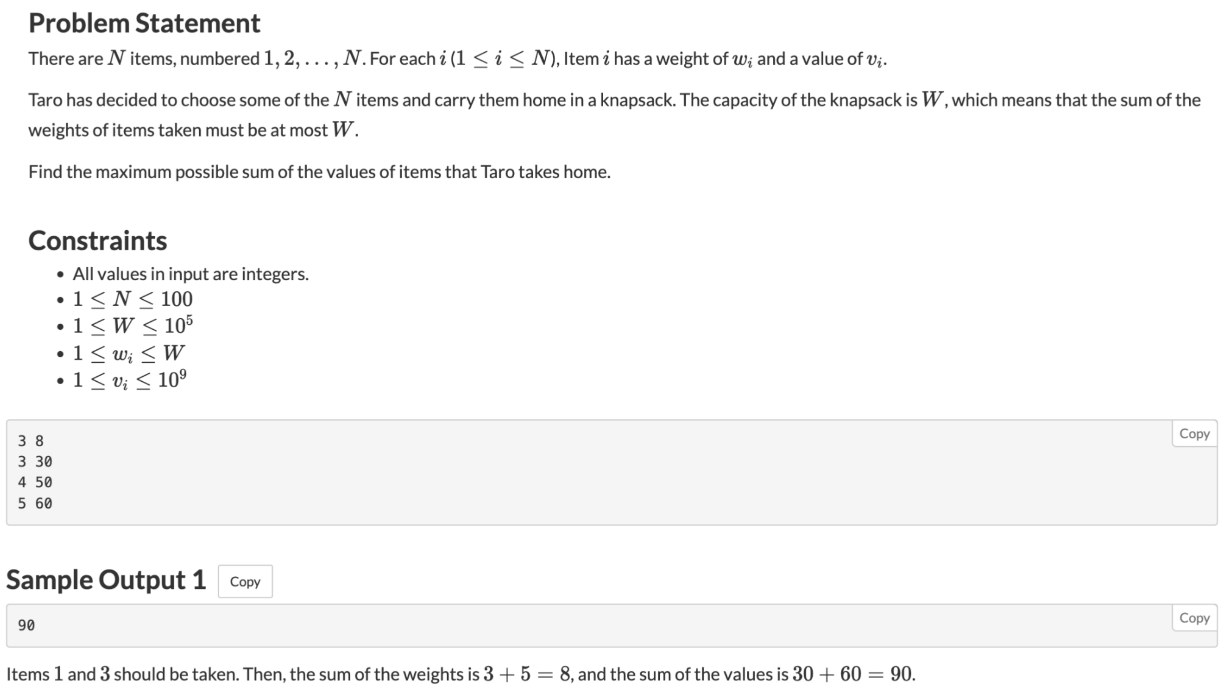
Sample Output 1

```
210
```

If Taro does activities in the order C, B, C, he will gain $70 + 50 + 90 = 210$ points of happiness.

```
        // dp[position] = the minimum cost at 'position'
        int n = in.nextInt();
        int[][] points = new int[n+1][4];
        for (int i = 1; i <= n; i++)
                for (int j = 1; j <= 3; j++)
                        points[i][j] = in.nextInt();
        // dp[day][last_activity] = max points
        int[][] dp = new int[n+1][4];
        dp[0][1] = dp[0][2] = dp[0][3] = 0;
        for (int i = 1; i <= n; i++)  {
                for (int j = 1; j <= 3; j++) {
                        for (int k = 1; k <= 3; k++) {
                                if (j == k) continue;
                                dp[i][j] = Math.max(dp[i][j], dp[i-1][k]);
                        }
                        dp[i][j] += points[i][j];
                }
        }

        int answer = Math.max(dp[n][1], Math.max(dp[n][2], dp[n][3]));
        out.println(answer);
}
```

4. Knapsack1



Problem Statement

There are N items, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), 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

- All values in input are integers.
- $1 \leq N \leq 100$
- $1 \leq W \leq 10^5$
- $1 \leq w_i \leq W$
- $1 \leq v_i \leq 10^9$

```
3 8
3 30
4 50
5 60
```

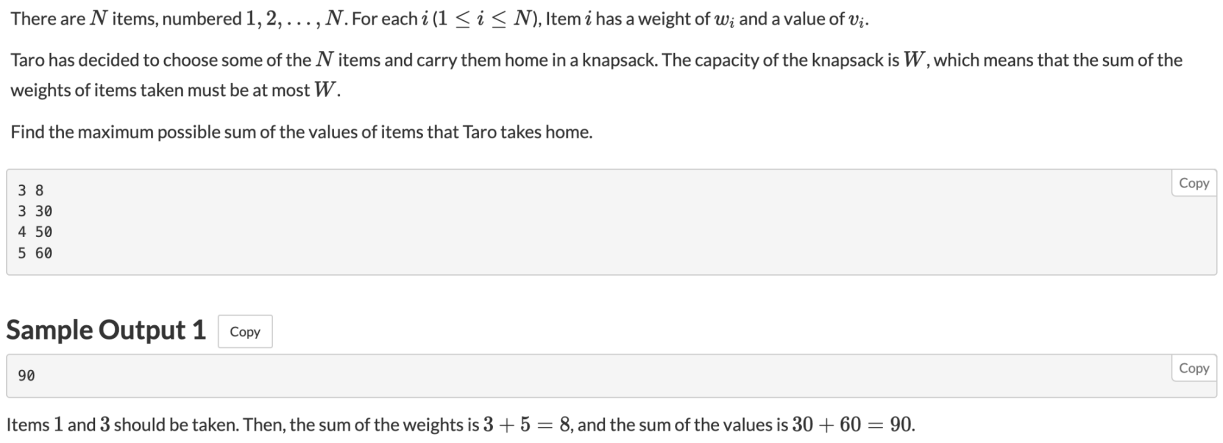
Sample Output 1

```
90
```

Items 1 and 3 should be taken. Then, the sum of the weights is $3 + 5 = 8$, and the sum of the values is $30 + 60 = 90$.

```
import java.util.Scanner;
public class Main {
  public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int W = sc.nextInt();
        long[] result = new long[W + 1];
        for (int i = 1; i <= N; i++) {
            int w = sc.nextInt();
            int v = sc.nextInt();
            for (int j = W; j >= w; j--) {
                result[j] = Math.max(result[j], result[j - w] + v);
            }
        }
        System.out.println(result[W]);
    }
}
```

5. Knapsack 2



There are N items, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), 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.

Copy

```
3 8
3 30
4 50
5 60
```

Sample Output 1 Copy

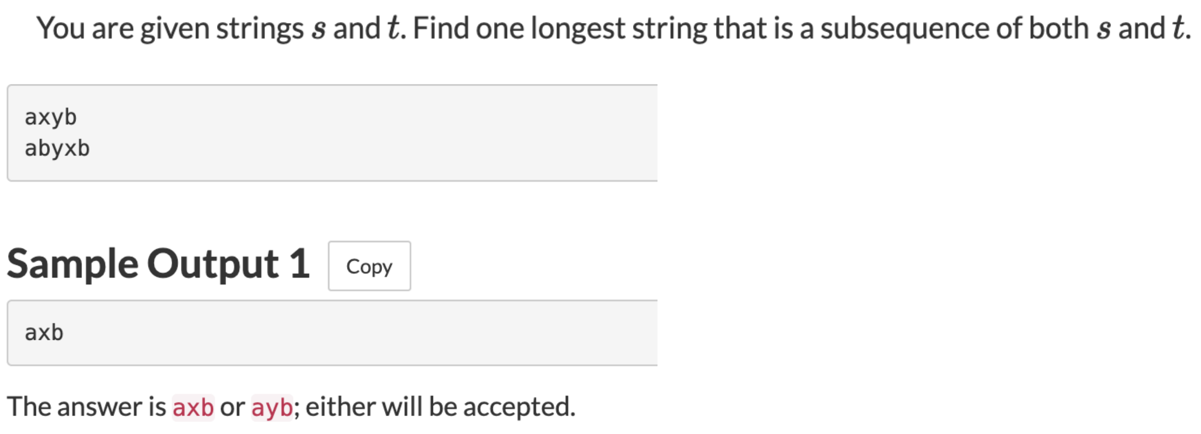
Copy

```
90
```

Items 1 and 3 should be taken. Then, the sum of the weights is $3 + 5 = 8$, and the sum of the values is $30 + 60 = 90$.

```
void solve() throws Exception {
        // dp[value] = min weight
        int n = in.nextInt(),
            W = in.nextInt();
        int[] w = new int[n],
             v = new int[n];
        for (int i = 0; i < n; i++) {
            w[i] = in.nextInt();
            v[i] = in.nextInt();
        }
        long[] dp = new long[MAXV+1];
        for (int i = 0; i <= MAXV; i++) dp[i] = INF;
        // iterate over items
        dp[0] = 0;
        for (int i = 0; i < n; i++) {
            // iterate over values
            for (int j = MAXV; j >= 0; j--) {

                // update dp
                if (j-v[i] >= 0)
                    dp[j] = Math.min(dp[j], dp[j-v[i]]+w[i]);
            }
        }
        long answer = 0;
        for (int i = MAXV; i >= 1; i--) {
            if (dp[i] <= W) {
                answer = i;
                break;
            }
        }
        out.println(answer);
    }
```

6.LCS



You are given strings s and t. Find one longest string that is a subsequence of both s and t.

```
axyb
abyxb
```

Sample Output 1 Copy

```
axb
```

The answer is `axb` or `ayb`; either will be accepted.

```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        String t = sc.nextLine();
        int n = s.length(), m = t.length();
        int[][] dp = new int[n+1][m+1];
        for(int i = 0; i < n; ++i) dp[i][0] = 0;
        for(int j = 0; j < m; ++j) dp[0][j] = 0;
        for(int i = 1; i <= n; ++i) {
            for(int j = 1; j <= m; ++j) {
                if(s.charAt(i-1) == t.charAt(j-1)) dp[i][j] = 1+ dp[i-1][j-1];
                else dp[i][j] = Math.max(dp[i-1][j], dp[i][j-1]);
            }
        }
        String lcs = "";
        int i = n, j = m;
        while(i > 0 && j > 0) {
            if(s.charAt(i-1) == t.charAt(j-1)) {
                lcs += s.charAt(i-1);
                i--;j--;
            }else if(dp[i-1][j] > dp[i][j-1]) {
                i--;
            }else {
                j--;
            }
        }
        lcs = new StringBuilder(lcs).reverse().toString();
        System.out.println(lcs);
    }
}
```

7.Longest Path

There is a directed graph G with N vertices and M edges. The vertices are numbered $1, 2, \ldots, N$, and for each i ($1 \leq i \leq M$), the i-th directed edge goes from Vertex x_i to y_i. G **does not contain directed cycles.**

Find the length of the longest directed path in G. Here, the length of a directed path is the number of edges in it.

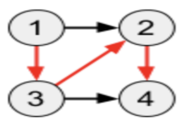
```
4 5
1 2
1 3
3 2
2 4
3 4
```

Sample Output 1

Copy

```
3
```

The red directed path in the following figure is the longest:



```
static ArrayList<Integer>[] g;
static int[] dp;

public static void main(String args[]) throws IOException {
        int n = readInt(), m = readInt();
        g = new ArrayList[n + 1];
        dp = new int[n + 1];
        Arrays.fill(dp, Integer.MIN_VALUE);
        for (int i = 0; i <= n; i++) g[i] = new ArrayList<>();
        for (int i = 0; i < m; i++) g[readInt()].add(readInt());
        for (int i = 1; i <= n; i++)
                if (dp[i] < 0) dfs(i);
        int ans = dp[1];
        for (int i = 2; i <= n; i++) ans = Math.max(ans, dp[i]);
        System.out.println(ans);
}
   static void dfs(int n) {
        for (int i : g[n]) {
                if (dp[i] < 0) dfs(i);
                dp[n] = Math.max(dp[n], dp[i] + 1);
                }
       dp[n] = Math.max(dp[n], 0);
   }
```

8. H Grid

There is a grid with H horizontal rows and W vertical columns. Let (i, j) denote the square at the i-th row from the top and the j-th column from the left.

For each i and j ($1 \leq i \leq H, 1 \leq j \leq W$), Square (i, j) is described by a character $a_{i,j}$. If $a_{i,j}$ is `.`, Square (i, j) is an empty square; if $a_{i,j}$ is `#`, Square (i, j) is a wall square. It is guaranteed that Squares $(1, 1)$ and (H, W) are empty squares.

Taro will start from Square $(1, 1)$ and reach (H, W) by repeatedly moving right or down to an adjacent empty square.

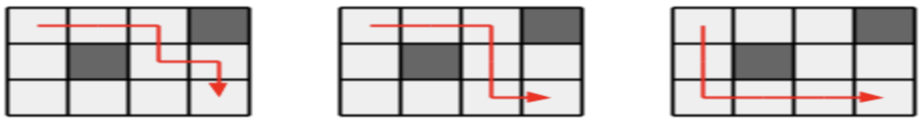
Find the number of Taro's paths from Square $(1, 1)$ to (H, W). As the answer can be extremely large, find the count modulo $10^9 + 7$.

```
3 4
...#
.#..
....
```

Sample Output 1 Copy

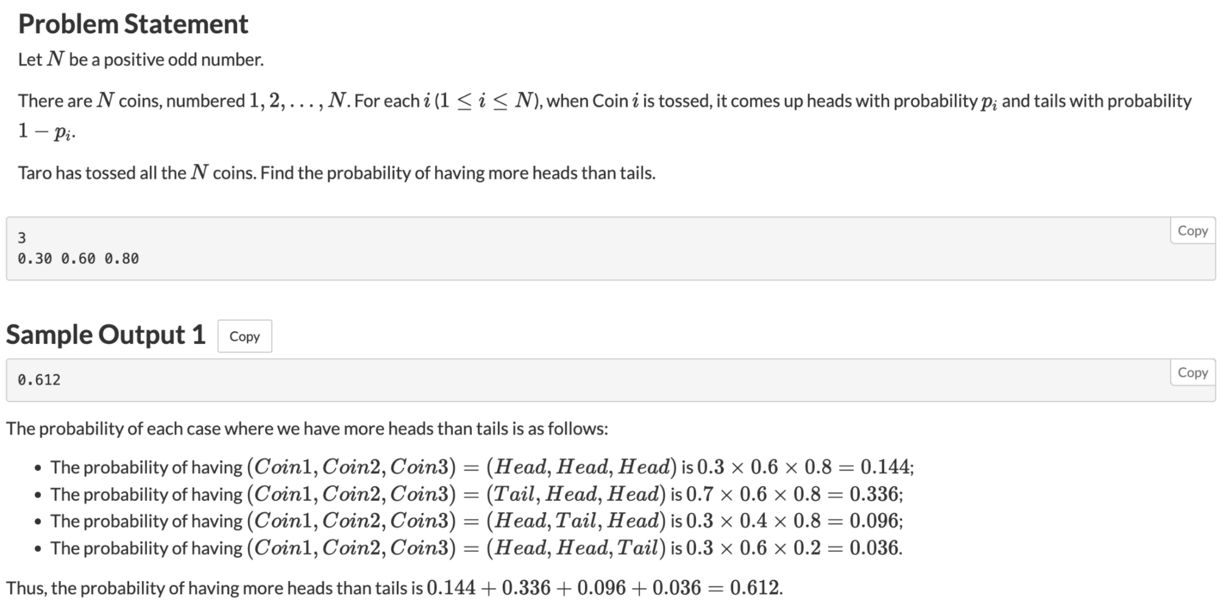
```
3
```

There are three paths as follows:



```
import java.util.Scanner;


public class Main {
    static boolean isValid(int i, int j, int n, int m) {
        return ((i >= 0 && i < n) && (j >= 0 && j < m));
    }

    static int getCost(int idx, int jdx, int n, int m, int arr[][], int dp[][]) {
        int mod = 1000000007;

        if (!isValid(idx, jdx, n, m)) {
            return 0;
        }
        if (idx == n-1 && jdx == m-1) {
            return 1;
        }

        if (arr[idx][jdx] == 1) {
            return 0;
        }

        if (dp[idx][jdx] != -1) {
            return dp[idx][jdx];
        }
        int d = getCost(idx, jdx + 1, n, m, arr, dp);
        int a = getCost(idx + 1, jdx, n, m, arr, dp);

        return dp[idx][jdx] = (a%mod + d%mod)%mod;
    }


    public static void main(String[] args) {
        int n, m, i, j;

        Scanner scan = new Scanner(System.in);
        n = scan.nextInt();
        m = scan.nextInt();

        int [][] arr = new int [n][m];
        int [][] dp = new int[n + 1][m + 1];

        for (i = 0; i < n; i++) {
            String tmp = scan.next();
            j = 0;
            for (char c : tmp.toCharArray()) {
                if (c == '.') {
                    arr[i][j] = 0;
                } else {
                    arr[i][j] = 1;
                }
                dp[i][j] = -1;
                j += 1;
            }
        }
        getCost(0, 0, n, m, arr, dp);
        int mod = 1000000007;
        System.out.println(dp[0][0]%mod);
    }
}
```

9. Coins



Problem Statement

Let N be a positive odd number.

There are N coins, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), when Coin i is tossed, it comes up heads with probability p_i and tails with probability $1 - p_i$.

Taro has tossed all the N coins. Find the probability of having more heads than tails.

```
3
0.30 0.60 0.80
```

Sample Output 1

```
0.612
```

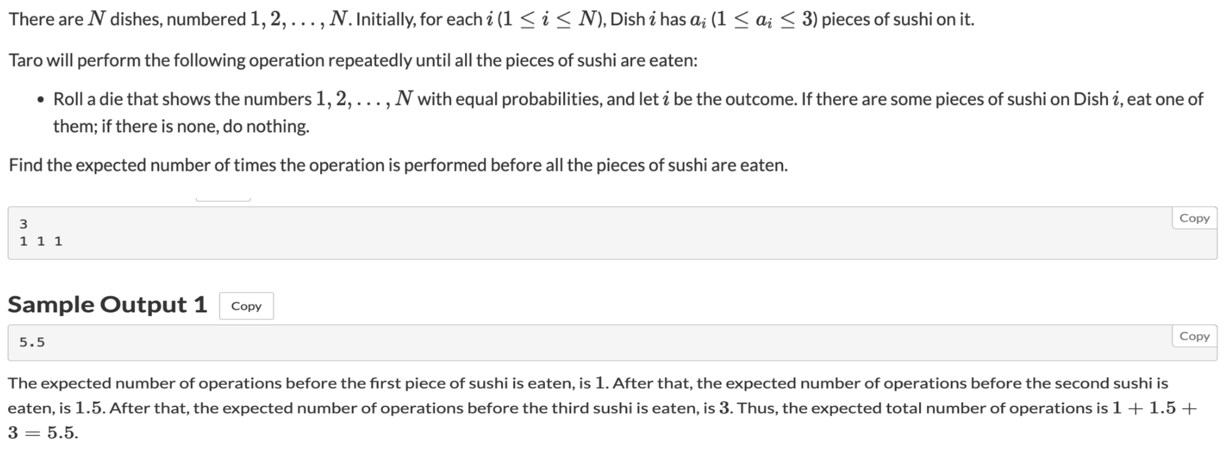
The probability of each case where we have more heads than tails is as follows:

- The probability of having $(Coin1, Coin2, Coin3) = (Head, Head, Head)$ is $0.3 \times 0.6 \times 0.8 = 0.144$;
- The probability of having $(Coin1, Coin2, Coin3) = (Tail, Head, Head)$ is $0.7 \times 0.6 \times 0.8 = 0.336$;
- The probability of having $(Coin1, Coin2, Coin3) = (Head, Tail, Head)$ is $0.3 \times 0.4 \times 0.8 = 0.096$;
- The probability of having $(Coin1, Coin2, Coin3) = (Head, Head, Tail)$ is $0.3 \times 0.6 \times 0.2 = 0.036$.

Thus, the probability of having more heads than tails is $0.144 + 0.336 + 0.096 + 0.036 = 0.612$.

```
public class Coins {
        public static void main(String[] args) throws IOException {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                String[] s = br.readLine().split(" ");
                int n = Integer.parseInt(s[0]);
                s = br.readLine().split(" ");
                double[] probs = new double[n + 1];
                double[][] dp = new double[n + 1][n + 1];
                for (int i = 1; i <= n; i++)
                        probs[i] = Double.parseDouble(s[i - 1]);
                dp[1][0] = 1 - probs[1];
                dp[1][1] = probs[1];
                for (int i = 2; i <= n; i++)
                        dp[i][0] = (1 - probs[i]) * dp[i - 1][0];
                for (int i = 2; i <= n; i++) {
                        for (int j = 1; j <= i; j++) {
                                dp[i][j] = (dp[i - 1][j - 1] * probs[i]) + (dp[i - 1][j] * (1 - probs[i]));

                        }
                }
                double sum = 0;
                for (int i = n; i >= (n + 1) / 2; i--)
                        sum += dp[n][i];

                System.out.println(sum);
        }
}
```

10. Sushi



There are N dishes, numbered $1, 2, \ldots, N$. Initially, for each i ($1 \leq i \leq N$), Dish i has a_i ($1 \leq a_i \leq 3$) 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 $1, 2, \ldots, N$ 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.

```
3
1 1 1
```

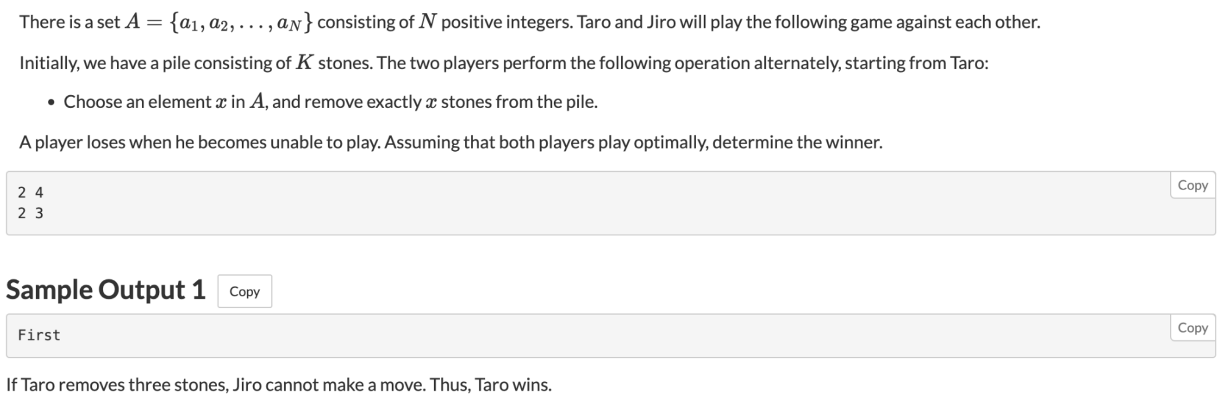
Sample Output 1

```
5.5
```

The expected number of operations before the first piece of sushi is eaten, is 1. After that, the expected number of operations before the second sushi is eaten, is 1.5. After that, the expected number of operations before the third sushi is eaten, is 3. Thus, the expected total number of operations is $1 + 1.5 + 3 = 5.5$.

```
public class Main {
  static Scanner scanner = new Scanner(System.in);
  public static void main(String[] args){
    int n = scanner.nextInt();
    int[] count = new int[4];
    for(int i = 0; i < n; i++){
      int c = scanner.nextInt();
      count[c]++;
    }
    double[][][] dp = new double[n + 1][n + 1][n + 1];
    for(int i = 0; i <= n; i++){
      for(int j = 0; j <= n; j++){
        for(int k = 0; k <= n; k++){
          if(i + j + k == 0) continue;
          if(i + j + k > n) continue;
          dp[i][j][k]++;
          if(i > 0 && j + 1 <= n){
            dp[i][j][k] += dp[i-1][j+1][k] * i / n;
          }
          if(j > 0 && k + 1 <= n){
            dp[i][j][k] += dp[i][j-1][k+1] * j / n;
          }
          if(k > 0){
            dp[i][j][k] += dp[i][j][k-1] * k / n;
          }
          dp[i][j][k] /= (double) (i+j+k) / n;
        }
      }
    }
    System.out.printf("%.14f\n", dp[count[3]][count[2]][count[1]]);
  }
}
```

11. Stones



There is a set $A = \{a_1, a_2, \ldots, a_N\}$ 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.

```
2 4
2 3
```

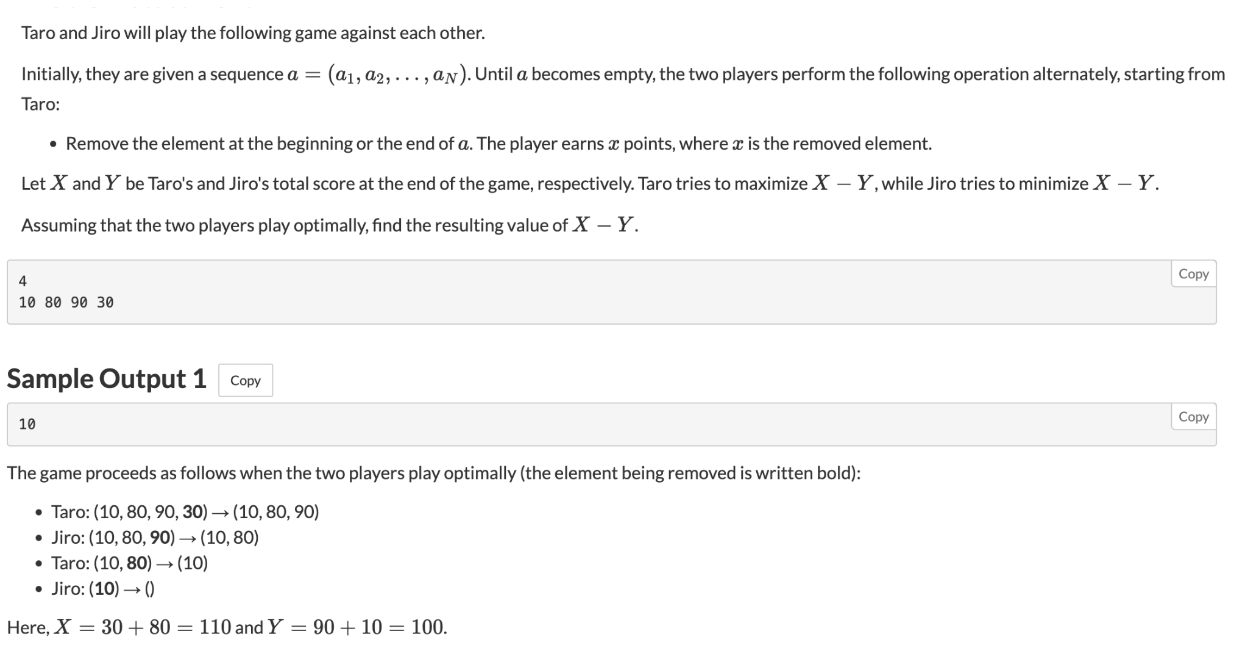
Sample Output 1

```
First
```

If Taro removes three stones, Jiro cannot make a move. Thus, Taro wins.

```
import java.util.*;
public class Main {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int k = sc.nextInt();
		int[] maunt = new int[n];
		boolean[] secWin = new boolean[k + 1];
		for(int i = 0;i < n;i++) {
			maunt[i]  =sc.nextInt();
		}secWin[0] = true;
		for(int i = 1;i <= k;i++) {
			boolean firWin = false;
			for(int j = 0;j < n;j++) {
				if(i - maunt[j] < 0)break;
				if(secWin[i - maunt[j]])firWin = true;
			}if(!firWin) {
				secWin[i] = true;
			}
		}System.out.print(secWin[k] ? "Second":"First");

	}

}
```

12. Deque



Taro and Jiro will play the following game against each other.

Initially, they are given a sequence $a = (a_1, a_2, \ldots, a_N)$. 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 $X - Y$.

```
4
10 80 90 30
```

Copy

Sample Output 1 Copy

```
10
```

Copy

The game proceeds as follows when the two players play optimally (the element being removed is written bold):

- Taro: (10, 80, 90, **30**) → (10, 80, 90)
- Jiro: (10, 80, **90**) → (10, 80)
- Taro: (10, **80**) → (10)
- Jiro: (**10**) → ()

Here, $X = 30 + 80 = 110$ and $Y = 90 + 10 = 100$.

```
import java.util.*;
import java.io.*;

public class Deque {

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int n = Integer.parseInt(br.readLine());
        String[] s = br.readLine().split(" ");
        int[] arr = new int[n + 1];
        long[][] dp = new long[n + 1][n + 1];

        for (int i = 1; i <= n; i++) {
            arr[i] = Integer.parseInt(s[i - 1]);
            dp[i][i] = arr[i];
        }

        for (int i = 2; i <= n; i++)
            for (int j = i - 1; j > 0; j--)
                dp[i][j] = Math.max(arr[j] - dp[i][j + 1], arr[i] - dp[i - 1][j]);

        System.out.println(dp[n][1]);
    }

}
```

13. Candies

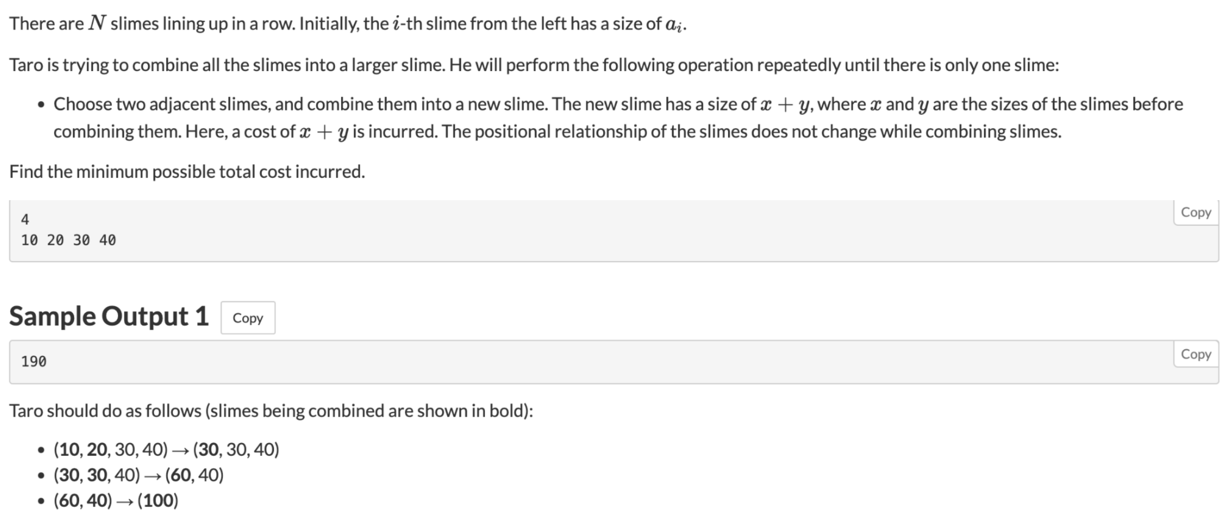
There are N children, numbered $1, 2, \ldots, N$.

They have decided to share K candies among themselves. Here, for each i ($1 \leq i \leq N$), 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.

```
import java.util.*;
import java.io.*;

public class Candies {

	public static final int mod = 1000000007;

	public static void main(String[] args) throws IOException {
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		String[] s = br.readLine().split(" ");
		int n = Integer.parseInt(s[0]);
		int k = Integer.parseInt(s[1]);
		int[] arr = new int[n + 1];
		int[][] dp = new int[n + 1][k + 1];
		int[] prefix = new int[k+1];
		s = br.readLine().split(" ");

		dp[0][0] = 1;
		for (int i = 1; i <= n; i++) {
			arr[i] = Integer.parseInt(s[i - 1]);
			dp[i][0] = 1;
		}
		int max = 0;
		for (int i = 1; i <= n; i++) {
			prefix[0] = dp[i-1][0];
			max += arr[i];
			for (int j = 1; j <= max && j <=k; j++) {
				prefix[j] = (dp[i-1][j]+ prefix[j-1]) % mod;
				if (j-arr[i]-1 < 0) {
					dp[i][j] = prefix[j];
				} else {
					dp[i][j] = (prefix[j]-prefix[j-arr[i]-1]+mod)%mod;
				}
			}
		}
		System.out.println(dp[n][k]);
	}
}
```

14. Slimes



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.

```
4
10 20 30 40
```

Sample Output 1

```
190
```

Taro should do as follows (slimes being combined are shown in bold):

- (**10**, **20**, 30, 40) → (**30**, 30, 40)
- (**30**, **30**, 40) → (**60**, 40)
- (**60**, **40**) → (**100**)

```
import java.util.*;
import java.io.*;

public class Slimes {

	public static void main(String[] args) throws IOException {
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		int n = Integer.parseInt(br.readLine());
		String[] s = br.readLine().split(" ");
		long[] prefix = new long[n + 1];
		for (int i = 0; i < n; i++)
			prefix[i + 1] = prefix[i] + Integer.parseInt(s[i]);

		long[][] dp = new long[n + 1][n + 1];

		for (int i = 2; i <= n; i++) {
			for (int j = i - 1; j > 0; j--) {
				dp[j][i] = Long.MAX_VALUE;
				for (int k = j; k < i; k++) {
					long val = dp[j][k] + dp[k + 1][i] + (prefix[k] - prefix[j - 1]) +
(prefix[i] - prefix[k]);
					if (dp[j][i] > val)
						dp[j][i] = val;
				}
			}
		}
		System.out.println(dp[1][n]);
	}

}
```

15. Matching

There are N men and N women, both numbered $1, 2, \ldots, N$.

For each i, j ($1 \leq i, j \leq N$), 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; if $a_{i,j} = 0$, they are 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 $10^9 + 7$.

```
public class Matching {
        public static final int mod = 1000000007;
        public static int count(int k) { //count number of ones in binary
                int c = 0;
                while (k > 0) {
                        k &= k-1;
                        c++;
                }
                return c;
        }
        public static void main(String[] args) throws IOException {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                int n = Integer.parseInt(br.readLine());
                String[] s;
                boolean[][] matches = new boolean[n][n];
                for (int i = 0; i < n; i++) {
                        s = br.readLine().split(" ");
                        for (int j = 0; j < n; j++)
                                matches[i][j] = Integer.parseInt(s[j]) == 1;
                }
                int[] dp = new int[(1 << n)];
                dp[0] = 1;
                for (int j = 0; j < (1 << n) - 1; j++) {
                        int taken = count(j);
                        for (int i = 0; i < n; i++) {
                                if (matches[taken][i] && !((j & (1<<i)) == 1)) {
                                        dp[j^(1<<i)] = (dp[j^(1<<i)] + dp[j]) % mod;
                                        System.out.printf("%-5s %d %d %s\n",
Integer.toBinaryString(j), taken, i, Integer.toBinaryString(j^(1<<i)) );
                                }
                        }
                }
                System.out.println(dp[(1<<n)-1]);

        }

}
```

16. Independent Sets

There is a tree with N vertices, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N-1$), 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.

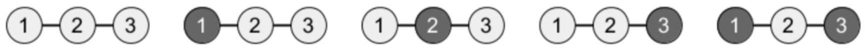
Find the number of ways in which the vertices can be painted, modulo $10^9 + 7$.

```
3
1 2
2 3
```

Sample Output 1

```
5
```

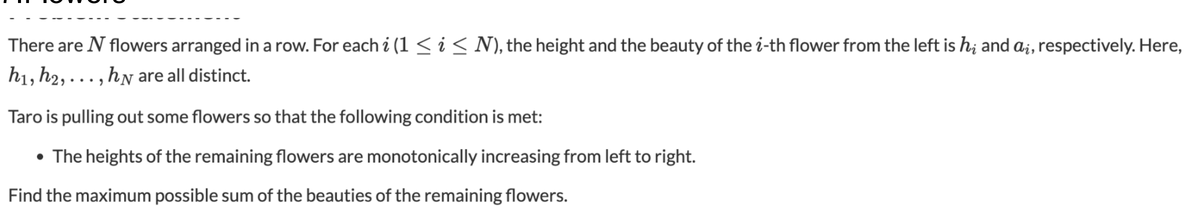
There are five ways to paint the vertices, as follows:



```
import java.util.*;
import java.io.*;

public class Independent_Set {

	public static Map<Integer, LinkedList<Integer>> graph = new HashMap<>();
	public static long[][] arr;
	public static int n;
	public static final long mod = 1000000007;

	public static long recurse(int cur, int parent, boolean black) {
		int col = black ? 1 : 0;
		if (arr[cur][col] != -1)
			return arr[cur][col];

		LinkedList<Integer> temp = graph.get(cur);
		long sum = 1;
		if (black) {
			for (int k : temp) {
				if (k != parent) {
					sum *= recurse(k, cur, false);
					sum = sum % mod;
				}
			}

			return arr[cur][1] = sum;
		} else {
			for (int k : temp) {
				if (k != parent) {
					sum *= recurse(k, cur, false) + recurse(k, cur, true);
					sum = sum % mod;
```

```
                }
            }
            return arr[cur][0] = sum;
        }
    }

    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        n = Integer.parseInt(br.readLine()); //Bottom up method not possible. Children
vary.
        if (n == 1) {
            System.out.println(2);
            return;
        }
        arr = new long[n + 1][2];
        for (int i = 0; i < n + 1; i++)
            Arrays.fill(arr[i], -1);
        String[] s;
        int a, b;
        for (int i = 0; i < n - 1; i++) {
            s = br.readLine().split(" ");
            a = Integer.parseInt(s[0]);
            b = Integer.parseInt(s[1]);
            if (!graph.containsKey(a))
                graph.put(a, new LinkedList<>());
            if (!graph.containsKey(b))
                graph.put(b, new LinkedList<>());
            graph.get(a).add(b);
            graph.get(b).add(a);
        }
        System.out.println((recurse(1, 0, true) + recurse(1, 0, false)) % mod);
    }

}
```

17.Flowers



There are N flowers arranged in a row. For each i ($1 \leq i \leq N$), the height and the beauty of the i-th flower from the left is h_i and a_i, respectively. Here, $h_1, h_2, \ldots, h_N$ 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.

```
public class Flowers {
    public static long[] tree;
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int n = Integer.parseInt(br.readLine());
        int[] height = new int[n];
        int[] beauty = new int[n];
        long[] dp = new long[n + 1];
        String[] s = br.readLine().split(" ");
        for (int i = 0; i < n; i++)
            height[i] = Integer.parseInt(s[i]);

        s = br.readLine().split(" ");
        for (int i = 0; i < n; i++)
            beauty[i] = Integer.parseInt(s[i]);
        int b = 1;
        while (b < n)
            b *= 2;
        tree = new long[b * 2];
        for (int i = 0; i < n; i++) {
            int k = b + height[i]-1;
            long max = 0;
            while (k > 1) {

                if (k % 2 == 1)
                    max = Math.max(max, tree[k - 1]);
                k /= 2;
            }
            dp[height[i]] = beauty[i] + max;
            k =b + height[i]-1;
            while (k >= 1) {
                tree[k] = Math.max(tree[k], dp[height[i]]);
                k /= 2;
            }


        }
        System.out.println(tree[1]);
```

18. Walk

There is a simple directed graph G with N vertices, numbered $1, 2, \ldots, N$.

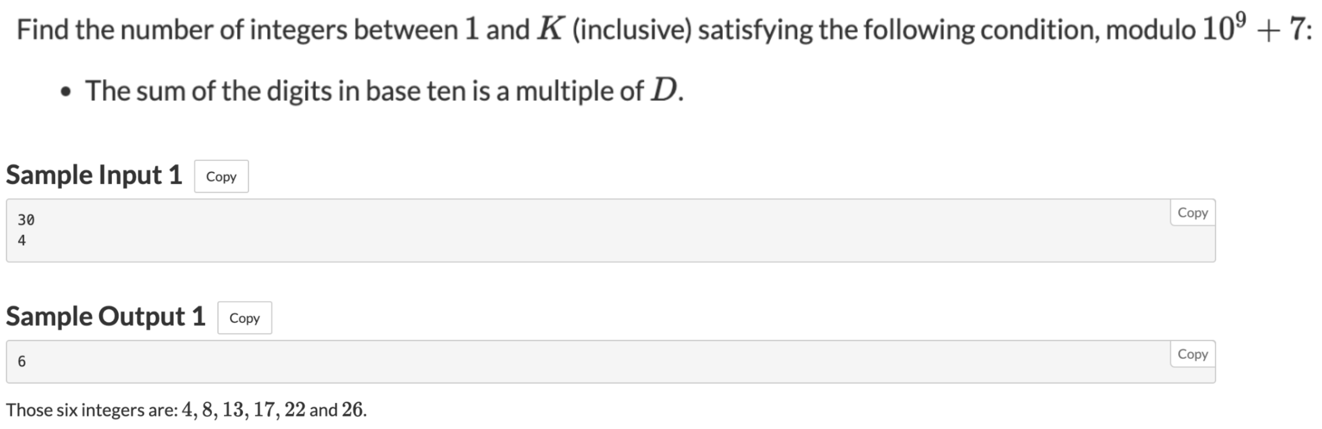
For each i and j ($1 \leq i, j \leq N$), you are given an integer $a_{i,j}$ that represents whether there is a directed edge from Vertex i to j. If $a_{i,j} = 1$, there is a directed edge from Vertex i to j; if $a_{i,j} = 0$, there is not.

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.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

class Main {
	static long mod = (long) (1e9+7);

	public static void main (String[] args) throws IOException {
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		StringTokenizer st = new StringTokenizer(br.readLine());

		int N = Integer.parseInt(st.nextToken());
		long k = Long.parseLong(st.nextToken());

		long[][] A = new long[N+1][N+1];

		for (int i = 1; i <= N; i++) {
			st = new StringTokenizer(br.readLine());
			for (int j = 1; j <= N; j++) {
				A[i][j] = Long.parseLong(st.nextToken());
			}
		}

		br.close();
		long[][] answer = pow(A, k);

		long r = 0;

		for (int i = 1; i <= N; i++) {
			for (int j = 1; j <= N; j++) {
				r += answer[i][j];
				r %= mod;

			}
		}

		System.out.println(r);
	}
```

```
        static long[][] multiply(long[][] A, long[][] B) {
                int n = A.length-1;
                long[][] C = new long[n+1][n+1];
                for (int i = 1; i <= n; i++) {
                        for (int j = 1; j <= n; j++) {
                                for (int k = 1; k <= n; k++) {
                                        C[i][j] += A[i][k] * B[k][j];
                                        C[i][j] %= mod;
                                }
                        }
                }
                return C;
        }
        static long[][] pow(long[][] A, long k) {
                long[][] res = new long[A.length][A.length];
                long[][] pow = new long[A.length][A.length];

                int n = A.length-1;
                for (int i = 1; i <= n; i++) {
                        res[i][i] = 1;
                }
                for (int i = 1; i <= n; i++) {
                        for (int j = 1; j <= n; j++) {
                                pow[i][j] = A[i][j];
                        }
                }
                while (k > 0) {
                        if (k%2 == 1) {
                                res = multiply(res, pow);
                        }
                        pow = multiply(pow, pow);
                        k /= 2;
                }
                return res;
        }
}
```

19.Digit Sum



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.

Sample Input 1 Copy

```
30
4
```

Sample Output 1 Copy

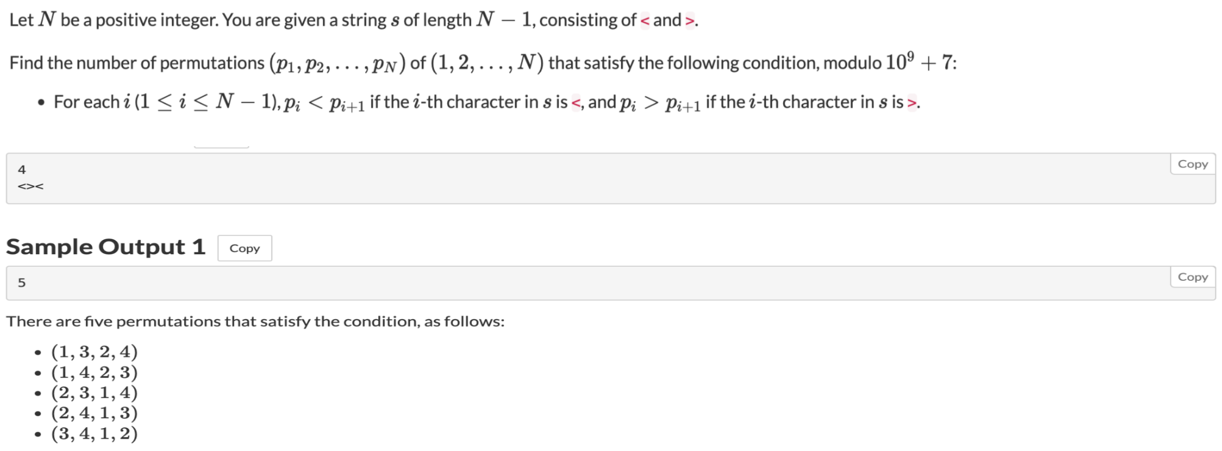
```
6
```

Those six integers are: $4, 8, 13, 17, 22$ and 26.

```
import java.io.*;
import java.util.*;
public class Main
{
   static long mod=1000000007;
         public static void main(String[] args) {
            Scanner sc=new Scanner(System.in);
                  int d;
                  String k;
                  k=sc.next();
                  d=sc.nextInt();
                  long[][][] dp = new long[10002][100][2];
      for (int i = 0; i <10001; i++)
         for (int j = 0; j < 100; j++)
            Arrays.fill(dp[i][j], -1);
                  System.out.println((sdigit(0,0,0,k,dp,d)+mod-1l)%mod);
         }
         public static long sdigit(int i,int sum,int f,String limit,long[][][] dp,long d)
         {
            sum%=d;
            if(i==limit.length())
            {
               if(sum==0)
                  return 1;
               else
                  return 0;
            }
            if(dp[i][sum][f]!=-1)
               return dp[i][sum][f];
            long val=0;
            if(f==1)
```

```
                {
                    for(int j=0;j<10;j++)
                    {
                        val+=sdigit(i+1,sum+j,1,limit,dp,d);
                        val%=mod;
                    }
                }
                else
                {
                    for(int j=0;j<=limit.charAt(i)-'0';j++)
                    {
                        if(j<limit.charAt(i)-'0')
                        {
                        val+=sdigit(i+1,sum+j,1,limit,dp,d);
                        }
                        else
                        {
                            val+=sdigit(i+1,sum+j,0,limit,dp,d);
                        }
                        val%=mod;
                    }
                }
                return dp[i][sum][f]=val%mod;
            }
      }
```

20. Permutation



Let N be a positive integer. You are given a string s of length $N-1$, consisting of < and >.

Find the number of permutations $(p_1, p_2, \ldots, p_N)$ of $(1, 2, \ldots, N)$ that satisfy the following condition, modulo 10^9+7:

- For each i $(1 \le i \le N-1)$, $p_i < p_{i+1}$ if the i-th character in s is <, and $p_i > p_{i+1}$ if the i-th character in s is >.

Copy

```
4
<><
```

Sample Output 1 Copy

Copy

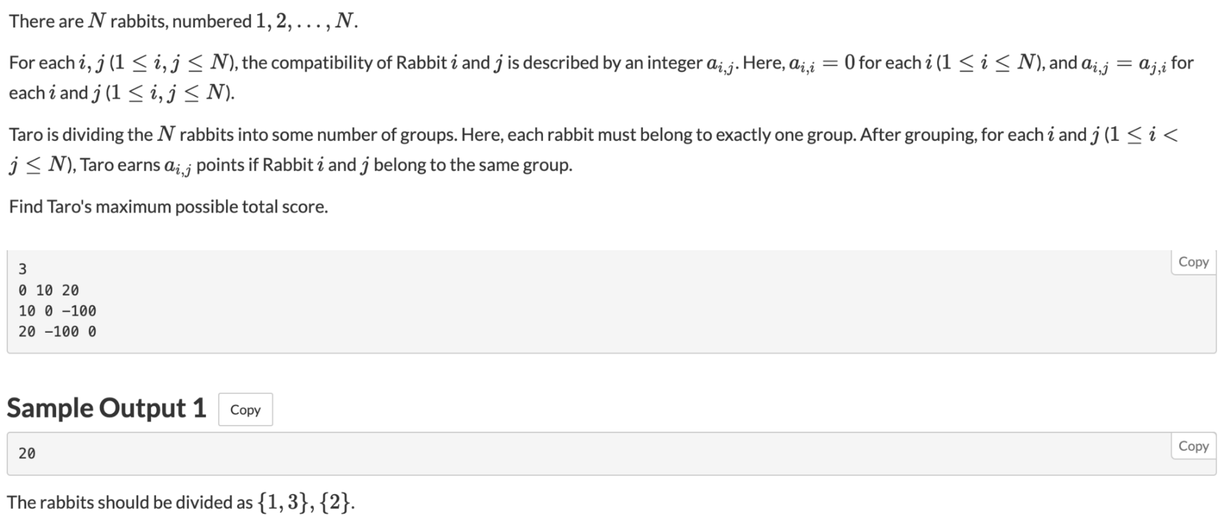
```
5
```

There are five permutations that satisfy the condition, as follows:

- $(1, 3, 2, 4)$
- $(1, 4, 2, 3)$
- $(2, 3, 1, 4)$
- $(2, 4, 1, 3)$
- $(3, 4, 1, 2)$

```
public class Main{
  public static long MOD = 1000000007L;
  public static void main(String[] args)throws IOException{
    BufferedReader f = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter out = new PrintWriter(System.out);
    int n = Integer.parseInt(f.readLine());
    char[] array = f.readLine().toCharArray();
    long[][] dp = new long[n][n];
    dp[0][0] = 1L;
    for(int k = 0; k < n-1; k++){
      long sum = 0L;
      if(array[k] == '<'){
        for(int j = k; j >= 0; j--){
          sum = (sum + dp[k][j] + MOD)%MOD;
          dp[k+1][j] = sum;
        }
      } else {
        for(int j = 1; j <= k+1; j++){
          sum = (sum + dp[k][j-1] + MOD)%MOD;
          dp[k+1][j] = sum;
        }
      }
    }
    long answer = 0L;
    for(int k = 0; k < n; k++){
      answer = (answer + dp[n-1][k] + MOD)%MOD;
    }
    out.println(answer);
    out.close();
  }
}
```

21. Grouping



There are N rabbits, numbered $1, 2, \ldots, N$.

For each i, j ($1 \leq i, j \leq N$), the compatibility of Rabbit i and j is described by an integer $a_{i,j}$. Here, $a_{i,i} = 0$ for each i ($1 \leq i \leq N$), and $a_{i,j} = a_{j,i}$ for each i and j ($1 \leq i, j \leq N$).

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 ($1 \leq i < j \leq N$), Taro earns $a_{i,j}$ points if Rabbit i and j belong to the same group.

Find Taro's maximum possible total score.

```
3
0 10 20
10 0 -100
20 -100 0
```

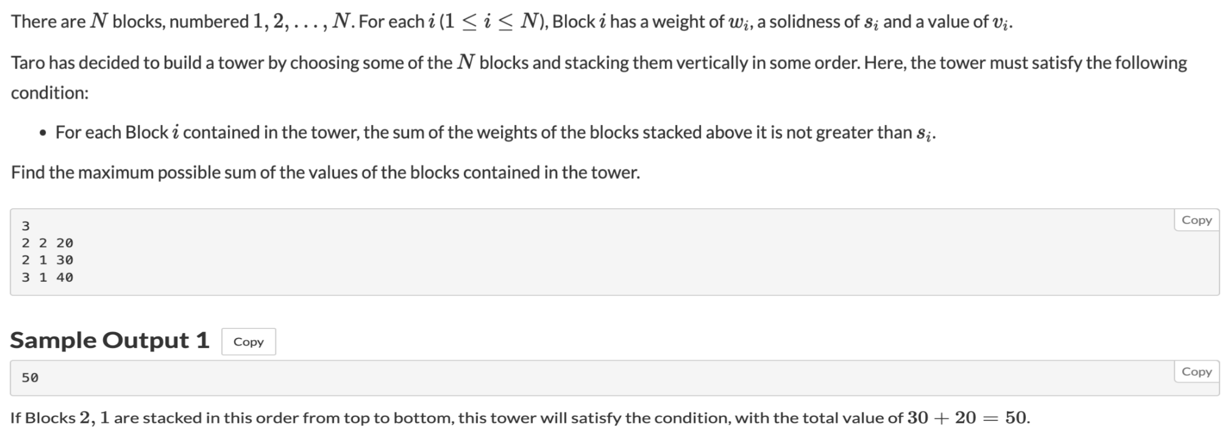
Sample Output 1

```
20
```

The rabbits should be divided as $\{1, 3\}, \{2\}$.

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), up = 1 << n;
        int[][] d = new int[n][n];
        long[] dp = new long[up];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                d[i][j] = sc.nextInt();
        for (int i = 0; i < up; i++)
            for (int j = 0; j < n; j++)
                if ((i >> j & 1) != 0)
                    for (int k = 0; k < j; k++)
                        if ((i >> k & 1) != 0)
                            dp[i] += d[j][k];
        for (int i = 1; i < up; i++)
            for (int j = (i - 1) & i; j != 0; j = (j - 1) & i)
                dp[i] = Math.max(dp[i], dp[j] + dp[i ^ j]);
        System.out.print(dp[up - 1]);
    }
}
```

24. Tower



There are N blocks, numbered $1, 2, \ldots, N$. For each i $(1 \leq i \leq N)$, 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.

```
3
2 2 20
2 1 30
3 1 40
```

Sample Output 1

```
50
```

If Blocks $2, 1$ are stacked in this order from top to bottom, this tower will satisfy the condition, with the total value of $30 + 20 = 50$.

```
class Main {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    int N;
    int[][] wsv;
    long[] dp;
    static int i(String s) { return Integer.parseInt(s); }
    void calc() throws Exception {
        N = i(br.readLine());
        wsv = new int[N][3];
        int maxS = 0;
        for (int i = 0; i < N; i++) {
            String[] wsvs = br.readLine().split(" ");
            for (int j = 0; j < 3; j++) wsv[i][j] = i(wsvs[j]);
            maxS = Math.max(maxS, wsv[i][0] + wsv[i][1]);
        }
        Arrays.sort(wsv, Comparator.comparingInt(d -> d[0] + d[1]));
        dp = new long[maxS + 1];
        for (int i = 0; i < N; i++) {
            for (int j = Math.min(maxS - wsv[i][0], wsv[i][1]); j >= 0; --j) {
                int w = j + wsv[i][0];
                dp[w] = Math.max(dp[w], dp[j] + wsv[i][2]);
            }
        }
        long max = 0;
        for (int i = 0; i <= maxS; i++)
            max = Math.max(max, dp[i]);
        System.out.println(max);
    }
    public static void main(String[] args) throws Exception {
        new Main().calc();
    }
}
```

26. Frog3

There are N stones, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), the height of Stone i is h_i. Here, $h_1 < h_2 < \cdots < h_N$ 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 $i+1, i+2, \ldots, N$. Here, a cost of $(h_j - h_i)^2 + C$ is incurred, where j is the stone to land on.

Find the minimum possible total cost incurred before the frog reaches Stone N.

```
import java.io.*;
import java.util.ArrayDeque;
import java.util.Deque;
class Main {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int N;
        long C;
        int[] h;
        long[] dp;
        CHT cht;
        static int i(String s) { return Integer.parseInt(s); }
        static long l(String s) { return Long.parseLong(s); }
        void calc() throws Exception {
                String[] nc = br.readLine().split(" ");
                N = i(nc[0]); C = l(nc[1]);
                h = new int[N];
                String[] hs = br.readLine().split(" ");
                for (int i = 0; i < N; i++) h[i] = i(hs[i]);
                dp = new long[N];
                cht = new CHT();
                for (int i = 1; i < N; i++) {
                        long a = -2L * h[i-1];
                        long b = dp[i-1] + (long)h[i-1] * h[i-1];
                        cht.add(a, b);
                        dp[i] = cht.min(h[i]) + (long)h[i] * h[i] + C;
                }
                System.out.println(dp[N-1]);
        }
        public static void main(String[] args) throws Exception {
                new Main().calc();
        }
}
class CHT {
        Deque<long[]> deq = new ArrayDeque<>();
        void add(long a1, long b1) {
                while (deq.size() > 1) {
                        long[] c2 = deq.pollLast();
```

```
            long[] c3 = deq.peekLast();
            if ((b1 - c2[1])*(c2[0] - c3[0]) < (c2[1] - c3[1])*(a1 - c2[0])) {
                deq.addLast(c2);
                break;
            }
        }
        deq.addLast(new long[] {a1, b1});
    }
    long min(int x) {
        long[] c1, c2;
        long v1, v2;
        c1 = deq.pollFirst();
        v1 = c1[0] * x + c1[1];
        while (!deq.isEmpty()) {
            c2 = deq.pollFirst();
            v2 = c2[0] * x + c2[1];
            if (v2 > v1) {
                deq.addFirst(c2);
                break;
            }
            c1 = c2; v1 = v2;
        }
        deq.addFirst(c1);
        return v1;
    }
}
```