电子科技大学
格拉斯哥学院
Glasgow College, UESTC

Kelvin Programming Competition
Coding Problem Handout

Track II: The VIP Track
December 7th, 2025

Contents

Background

In this competition, we will focus on one engineering challenge related to *health & well-being*. Specifically, it is about *sleep* - one of the most significant aspects of our physical health.

HypnosTech - The Sleep Architect

"Sleep is the golden chain that ties health and our bodies together."
— Thomas Dekker

In today's fast-paced life, sleep disorders have become a major problem affecting the physical and mental health of billions of people worldwide. Chronic lack of deep sleep can lead to weakened immunity, anxiety, and even cognitive impairment.

To address this crisis, *HypnosTech*, a health technology company founded by Glasgow College alumni, has developed a revolutionary smart sleep aid pillow. This device goes beyond passively monitoring sleep; it actively induces the brain to enter specific sleep stages by playing specific soundscapes—such as white noise, rain sounds, and forest wind sounds—through bone conduction technology.

However, the human brain's sleep state is a highly complex and random process. Incorrect audio guidance may wake the user, producing the opposite effect. Furthermore, neuroscientific research indicates that the brain requires a 'synchronization period' to adapt to a new soundscape. Frequent switching breaks this synchronization, creating a **'Cognitive Switching Penalty'.** Instead of inducing sleep, rapid changes force the brain into an alert state, allowing it to process new stimuli and effectively reducing the sleep quality score. Therefore, the algorithm must balance the immediate benefit of a better sound against the cost of disrupting the current state.

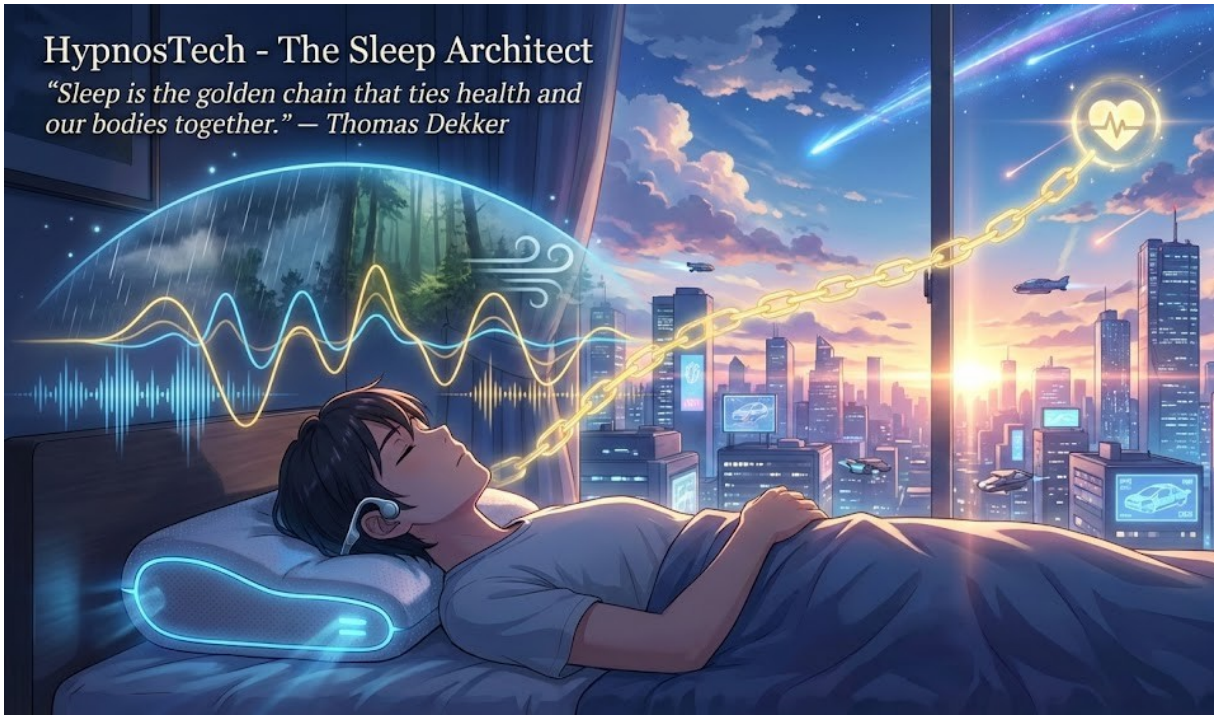
As the lead backend algorithm engineer at HypnosTech, your task is to design the core scheduling algorithm. You need to generate **an optimal "sleep aid score"** based on each user's **brainwave characteristics (state transition probabilities)** to maximize the user's **sleep recovery quality** within a limited time.

Objective

Your objective is to write a program that generates an **audio playback sequence** with a duration of **T minutes**, maximizing the total sleep integral produced by this sequence in terms of **mathematical expectation**. The program needs to automatically calculate the optimal action path based on the **input state transition matrix** and **reward function**. You may choose to use **C, C++** or **Python** to complete the challenge.

As the emphasis of this challenge is coding, we have provided all the mathematical knowledge you need to engineer the solution in *Mathematical Model*. The guidelines for the program input and output formats are given in *Input & Output*. Meanwhile, to help you understand how your solution will be evaluated, we also attach the detailed rubrics we use to grade your programs in *Rubrics*. In the *Appendix*, we include some example input & output and codes to help you write code more effectively.

Happy & Healthy Coding, Master Programmers!



Mathematical Model

3.1 Table of Notations

The following symbols are used throughout the model definition:

Symbol	Definition
T	Total duration of the sleep session (minutes)
N	Number of discrete sleep states (e.g., Awake, Deep Sleep)
M	Number of available soundscapes (actions)
S_t	The state of the brain at minute t ($S_t \in [S_0, \ldots, S_{N-1}]$)
a_t	The action selected at minute t ($a_t \in [a_0, \ldots, a_{M-1}]$)
L	The wandering cycle period for the time-variant reward system
μ	The system's average reward centroid
$R_{initial}(s)$	The initial base reward input for state s
$R(s,t)$	The calculated reward obtained for being in state s at time t
$P_{i,j}^{(a)}$	Probability of moving from state i to j under action a
$C_{i,j}$	Cost penalty when switching actions while moving from state i to j
$I(\cdot)$	The indicator function, which equals 1 if the condition is true, and 0 otherwise

3.2 States

The brain has N discrete states. Example with $N = 5$:

- S_0: Awake
- S_1: Light Sleep
- S_2: Deep Sleep
- S_3: REM
- S_4: Waked Up

3.3 Actions

The system supports M types of soundscapes (sound effects), e.g., A_0 (silence), A_1 (white noise), A_2 (thunderstorm sound).

3.4 Dynamics

3.4.1 Transition Matrix

For each action $a \in [a_0, \ldots, a_{M-1}]$, there is an $N \times N$ matrix $P^{(a)}$. $P^{(a)}_{i,j}$ represents the probability of transitioning to state j in the next minute when the current state is i and sound effect a is playing.

3.4.2 Generalized Cognitive Switching Penalty

To model the biological cost of changing neural states, the switching penalty is defined by a Cost Matrix $C_{N \times N}$. Let $C_{i,j}$ be the cost of switching soundscapes ($a_t \neq a_{t-1}$) when the brain transitions from State i to State j.

The penalty at time t is calculated as:

$$Penalty_t = I(a_t \neq a_{t-1}) \cdot C_{S_{t-1}, S_t} \tag{3.1}$$

For $t = 1$, assume $a_{t-1} = a_0 = 0$. Noted that in basic cases, this matrix simplifies to a uniform structure where all off-diagonal costs equal a constant 0. For example, $C_{i,j} = 0$ for all $i \neq j$.

3.4.3 Time-Variant Reward System (Value Wandering)

Rewards follow a ***Mean-Reverting Process***. They follow a Mean-Reverting Process with an invariant centroid, preventing the brain from "locking" into a single state. As time progresses, high-value states naturally drift towards the system average, while low-value states recover, periodically inverting their relative advantages. The reward for state s at time t is calculated as:

$$R(s,t) = \mu + (R_{initial}(s) - \mu) \cdot \cos\left(\frac{2\pi t}{L}\right) \tag{3.2}$$

Where:

- μ: The system's average reward centroid ($\mu = \frac{1}{N}\sum_{i=0}^{N-1} R_{initial}(s_i)$).
- L: The wandering cycle period. If $L = 0$, rewards remain static: $R(s,t) = R_{initial}(s)$.
- $R_{initial}(s)$: The base reward input for state $\boldsymbol{s}$.

Noted that if $L = 0$ in the input, the rewards remain static ($R(s,t) = R_{initial}(s)$).

3.4.4 Initial Condition

The system is in State 0 (S_0) at time $t = 0$ with a probability of 1.0.

- $P(S_0 = 0) = 1.0$
- $P(S_0 = k) = 0$ for all $k \neq 0$

3.4.5 Calculation Implications

- **First Transition:** The probability distribution of the state at $t = 1$ (S_1) depends entirely on the transition row for State 0 in the matrix of the chosen action a_1:

$$P(S_1 = j) = P_{0,j}^{(a_1)} \tag{3.3}$$

- **First Penalty:** For $t = 1$, the penalty calculation uses $S_0 = 0$.

$$Penalty_1 = I(a_1 \neq 0) \cdot C_{0,S_1} \tag{3.4}$$

3.5 Optimization Objective

Objective Function:

Let $A = [a_1, a_2, \ldots, a_T]$ be the sequence of actions chosen for each minute t.

The total expected score $J(A)$ is defined as:

$$J(A) = E\left[\sum_{t=1}^{T}(R(S_t, t) - Penalty_t)\right] \tag{3.5}$$

Where :

- S_t is the random variable representing the sleep state at time t.
- $I(\cdot)$ is the indicator function (1 if switching occurs, 0 otherwise).
- $E[\cdot]$ is the expectation taken over the stochastic transitions defined by matrices $P(a_t)$.

Optimization Problem:

Find the optimal sequence A^* such that:

$$A^* = \underset{a_1,\ldots,a_T}{\operatorname{argmax}}\, J(A) \tag{3.6}$$

Input & Output

All data is read through standard input (stdin).

4.1 Input Format

The input consists of the following structure:

```
T N M L Type
R0 R1 ... RN-1
[Switching Penalty Matrix]
[Matrix block Action 0]
[Matrix block Action 1]
...
[Matrix block Action M-1]
```

Line 1: Four integers

- T: Total duration (minutes).
- N: Number of states.
- M: Number of sound effects.
- L: Cycle period for reward wandering. If $L = 0$, $R(s, t) = R_{initial}(s)$.
- $Type$: Constraint mode identifier.

Line 2: Base Rewards

- N integers representing the initial base rewards $R_{initial}(s)$.

Switching Penalty Matrix

An $N \times N$ integer matrix block always follows, regardless of the $Type$ value.

- If $Type = 1$ (Generalized Matrix Penalty Mode): The value at row i, column j represents the specific switching penalty $C_{i,j}$.
- If $Type = 0$ (Uniform Penalty Mode): The matrix block is still provided to maintain input format consistency, but all values in this matrix will be 0. Your program should ignore the values in this matrix and assume the penalty is 0 (Static $K = 0$).

Matrix Blocks

The following M blocks each represent a transition matrix for a sound effect. Each block contains N rows, with N floating-point numbers (0.0 to 1.0) per row. The number in the i-th row and j-th column represents $P(State_i \to State_j)$. The sum of each row is guaranteed to be 1.0.

4.2 Output Format

Output a single line containing the sequence of sound effect IDs formatted as a JSON-style list:

$$[a_1, a_2, \ldots, a_T]$$

Each integer represents the sound effect ID selected for this minute (from 0 to $M-1$). Separators must strictly be a comma followed by a space.

Example Input:

For input examples, please refer to the Appendix *Raw Input.txt*.

Example Output:

```
[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 1, 1, 1, 0, 0]
```

Rubrics

5.1 Scoring Mechanism: God's-eye View and Time Penalty

The competition consists of 10 test cases. The total score is calculated using the following formulas:

$$Score^i_{base} = \min(Score^i_{raw}, Score^i_{standard}) \tag{5.1}$$

$$Score^i_{bonus} = \max(0, Score^i_{raw} - Score^i_{standard}) \times \frac{1}{1 + \log(1 + T_{use_i})} \tag{5.2}$$

$$Score_{final} = \sum_{i=1}^{10} (Score^i_{base} + Score^i_{bonus}) \tag{5.3}$$

Where:

- $Score^i_{raw}$: The expected value calculated by the evaluation system using precise matrix multiplication based on your output sequence.
- $Score^i_{standard}$: The qualification target score officially set by the Competition Committee.
- $Score^i_{base}$: Capped at the standard score.
- $Score^i_{bonus}$: Awarded only if the raw score exceeds the standard, scaled by computational efficiency.
- T^i_{use}: The algorithm's running time on the i-th test case.

Note: Illegal output results in a score of 0.

5.2 Test Cases

Basic cases (30%):

- Constraints: No penalty ($Type = 0$)
- States Reward: Constants ($L = 0$)
- Size: $T \leq 50, M = 10, N = 5$

Performance test cases (40%):

- Constraints: Switching Penalty Matrix ($Type = 1$)
- States Reward: Constants ($L = 0$)
- Size: $1000 \leq T \leq 3000, M = 80, N = 40$

Ultimate test cases (30%):

- Constraints: Switching Penalty Matrix ($Type = 1$)
- States Reward: Time-Variant Reward System ($L = T$)
- Size: $1000 \leq T \leq 3000, M = 80, N = 40$

5.3 Submission and Grading Policy

- **Single File Submission:** Upload exactly **one program file** (recommended name **"main"**).
- **Error Penalty:** Runtime errors, crashes, or formatting non-compliance result in a score of 0 for that test case.
- **Environment:** Python 3.12, Clang 12, GCC 10.
- **Notes:** We have provided sample code frameworks in C, C++, and Python for your reference, which also facilitates scoring and testing. You can view them in the Appendix.

Appendix

6.1 Example I/O

6.1.1 Example Input & Example Solution Code

Code 6.1: How an input file generated (C)

```cpp
#include <vector>
#include <iostream>

using namespace std;

// Part 1: Data Structure Definition (Schema)
struct InputData {
    // --- Header (Line 1) ---
    int T;        // Total Duration (e.g., 20 mins)
    int N;        // Number of States (e.g., 5 states)
    int M;        // Number of Actions (e.g., 3 sound effects)
    int L;        // Wandering Cycle (e.g., 20 mins, 0 = static)
    int Type;     // Mode: 0 = Static Penalty, 1 = Matrix Penalty

    // --- Base Rewards (Line 2) ---
    vector<int> BaseRewards; // Size: [N]

    // --- Penalty Configuration (Line 3) ---
    // If Type == 0: Only use StaticK
    // If Type == 1: Use PenaltyMatrix
    int StaticK;
    vector<vector<int>> PenaltyMatrix; // Size: [N][N]

    // --- Transition Probabilities (Block 4...) ---
    // Structure: [ActionID][FromState][ToState]
    // Size: [M][N][N]
    vector<vector<vector<double>>> Probabilities;
};

```

```
// Part 2: Example Data Instance (N=5, M=3)
// This corresponds to the input: "20 5 3 20 1 ..."
InputData ExampleCase = {
    // 1. Header: T=20, N=5, M=3, L=20, Type=1 (Matrix Mode)
    .T = 20,
    .N = 5,
    .M = 3,
    .L = 20,
    .Type = 1,

    // 2. Base Rewards (for States 0, 1, 2, 3, 4)
    .BaseRewards = { 0, 20, 50, 100, 0 },

    // 3. Penalty Matrix (Type=1 active)
    // Cost to switch FROM row_i TO col_j
    .PenaltyMatrix = {
        {  0,  10,  20,  50, 100 }, // From State 0
        { 10,   0,  10,  20, 100 }, // From State 1
        { 20,  10,   0,  10, 100 }, // From State 2
        { 50,  20,  10,   0, 100 }, // From State 3
        {  0,   0,   0,   0,   0 }  // From State 4 Waked Up - No cost to switch
    },

    // 4. Transition Matrices (3 Actions x 5x5 Matrix)
    .Probabilities = {
        // --- Action 0 (Silence / Initial) ---
        {
            {0.9, 0.1, 0.0, 0.0, 0.0}, // From S0: Mostly stays awake
            {0.5, 0.5, 0.0, 0.0, 0.0}, // From S1: Unstable
            {0.1, 0.1, 0.8, 0.0, 0.0}, // From S2: Returns to Deep Sleep
            {0.0, 0.0, 0.1, 0.9, 0.0}, // From S3: Good REM retention
            {1.0, 0.0, 0.0, 0.0, 0.0}  // From S4: Waked up stays awake
        },

        // --- Action 1 (White Noise) ---
        {
            {0.5, 0.5, 0.0, 0.0, 0.0}, // From S0: Helps fall asleep
            {0.1, 0.8, 0.1, 0.0, 0.0}, // From S1: Stabilizes Light Sleep
            {0.0, 0.2, 0.7, 0.1, 0.0}, // From S2: Deep sleep
```

```
        slightly disturbed
69              {0.0, 0.0, 0.2, 0.8, 0.0}, // From S3: REM OK
70              {1.0, 0.0, 0.0, 0.0, 0.0}  // From S4
71          },
72
73          // --- Action 2 (Thunderstorm) ---
74          {
75              {0.2, 0.2, 0.2, 0.2, 0.2}, // From S0: Chaos (High
        variance)
76              {0.2, 0.2, 0.2, 0.2, 0.2}, // From S1
77              {0.2, 0.2, 0.2, 0.2, 0.2}, // From S2
78              {0.2, 0.2, 0.2, 0.2, 0.2}, // From S3
79              {1.0, 0.0, 0.0, 0.0, 0.0}  // From S4
80          }
81      }
82 };
```

6.1.2 Example input file format

Note: Input will be in the form of a txt file. Here are the data inside *Raw Input.txt*

```
 1 20 5 3 20 1
 2 0 20 50 100 0
 3 0 10 20 50 0
 4 10 0 10 20 0
 5 20 10 0 10 0
 6 50 20 10 0 0
 7 0 0 0 0 0
 8 1.0 0.0 0.0 0.0 0.0
 9 0.9 0.1 0.0 0.0 0.0
10 0.5 0.5 0.0 0.0 0.0
11 0.1 0.1 0.8 0.0 0.0
12 0.0 0.0 0.1 0.9 0.0
13 0.5 0.5 0.0 0.0 0.0
14 0.1 0.8 0.1 0.0 0.0
15 0.0 0.2 0.7 0.1 0.0
16 0.0 0.0 0.2 0.8 0.0
17 1.0 0.0 0.0 0.0 0.0
18 0.2 0.2 0.2 0.2 0.2
19 0.2 0.2 0.2 0.2 0.2
20 0.2 0.2 0.2 0.2 0.2
21 0.2 0.2 0.2 0.2 0.2
22 1.0 0.0 0.0 0.0 0.0
```

6.1.3 Example output format

Format Specification: $[a_1, a_2, \ldots, a_T]$

- The output must start with [and end with].
- a_t: An integer representing the Sound Effect ID selected for the t^{th} minute ($0 \leq a_t < M$).
- **Separator:** Between any two integers, strictly use a comma followed by a space (,).

`Example Output: [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 1, 1, 1, 0, 0]`

6.1.4 Solution templates in C, C++, and Python

Code 6.2: Solution template (C) *main.c*

```c
/**
 * 2025 Lord Kelvin Programming Competition - Solution Template
 * Language: C (C11 Standard Recommended)
 * * Usage:
 * - Compile: gcc -O3 solution.c -o solution -lm
 * - Local Test: ./solution < input.txt
 * - CodeGrade: Automatically feeds data via stdin
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

// ==========================================
// 1. Constants & Data Structures
// ==========================================

// Maximum constraints defined based on "Ultimate test cases"
// Ref: Section 5.2 Test cases [cite: 145, 149]
#define MAX_T 3005
#define MAX_N 45
#define MAX_M 85

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

typedef struct {
    int T; // Total duration
    int N; // Number of states
    int M; // Number of sound effects
    int L; // Wandering cycle period
```

```c
    int Type; // 0 = No Penalty, 1 = Matrix Penalty
} Config;

typedef struct {
    Config cfg;

    // Initial base rewards R_initial(s)
    int initial_rewards[MAX_N];

    // System average reward centroid (mu)
    double mu;

    // Switching Penalty Matrix C_{i,j}
    // Access: penalty_matrix[from_state][to_state]
    int penalty_matrix[MAX_N][MAX_N];

    // Transition Matrices P^(a)
    // Access: transition_matrices[action_id][from_state][to_state]
    double transition_matrices[MAX_M][MAX_N][MAX_N];

} ProblemData;

// ==========================================
// 2. Logic Helpers (Math Model)
// ==========================================

/**
 * Calculate Reward R(s,t)
 * Implements the Time-Variant Reward System.
 * Ref: Section 3.4 [cite: 60, 61]
 */
double get_reward(const ProblemData* data, int state, int t) {
    // Case 1: Constant Rewards (L=0) [cite: 64]
    if (data->cfg.L == 0) {
        return (double)data->initial_rewards[state];
    }

    // Case 2: Wandering Rewards (L > 0) [cite: 60]
    double r_init = (double)data->initial_rewards[state];
    double cosine_term = cos((2.0 * M_PI * t) / (double)data->cfg.L);

    return data->mu + (r_init - data->mu) * cosine_term;
}

```

```c
/**
 * Calculate Switching Penalty
 * Ref: Section 3.4 [cite: 52, 109, 110]
 */
int get_penalty(const ProblemData* data, int prev_state, int curr_state, int prev_action, int curr_action) {
    // If actions are the same, cost is 0 [cite: 52]
    if (prev_action == curr_action) return 0;

    // If Type is 0 (Uniform/Static Mode), logic forces 0 penalty [cite: 110]
    if (data->cfg.Type == 0) return 0;

    // Otherwise, use the penalty matrix [cite: 109]
    return data->penalty_matrix[prev_state][curr_state];
}

// ==========================================
// 3. Input Parsing (TODO)
// ==========================================

void read_input(ProblemData* data) {
    // Initialize standard streams for speed if possible (less critical in C than C++)

    // ------------------------------------------------------------
    // TODO: Implement Data Reading Logic Here
    // ------------------------------------------------------------
    // Ref: Section 4.1 Input Format [cite: 93-112]
    // 1. Read Header: T, N, M, L, Type
    // 2. Read Initial Rewards (N integers)
    // 3. Read Penalty Matrix (N x N integers)
    // 4. Read Transition Matrices (M blocks of N x N doubles)
    // ... Write your scanf code here ...
}

// ==========================================
// 4. Core Algorithm (Solver)
// ==========================================

/**
 * Fill the result array with the sequence of actions.
 * Size of result array is data->cfg.T.
 */
void solve(const ProblemData* data, int* result_actions) {
    // ------------------------------------------------------------
```

```
    // TODO: Implement your Optimization Algorithm Here
    // ------------------------------------------------------------
    // Objective: Maximize expected score J(A) [cite: 79]
    // Starting state S_0 is always 0. [cite: 67]

    // Example Placeholder: Fill with 0s
    for (int t = 0; t < data->cfg.T; ++t) {
        result_actions[t] = 0;
    }
}

// ========================================
// 5. Output Formatting
// ========================================

void print_output(const int* actions, int T) {
    // Format specification: [a1, a2, ..., aT] [cite: 254]
    // Separator: Comma followed by a space [cite: 257]

    printf("[");
    for (int i = 0; i < T; ++i) {
        printf("%d", actions[i]);
        if (i < T - 1) {
            printf(", ");
        }
    }
    printf("]\n");
}

// ========================================
// Main Entry Point
// ========================================

int main() {
    // Allocate data on stack (static size allows this)
    // If stack size is an issue on some systems, make this static
        or global.
    static ProblemData data;

    // 1. Read Input
    read_input(&data);

    // 2. Solve
    if (data.cfg.T > 0 && data.cfg.T <= MAX_T) {
        // Allocate result array
        int result_actions[MAX_T];
```

```
166
167         solve(&data, result_actions);
168
169         // 3. Output
170         print_output(result_actions, data.cfg.T);
171     }
172
173     return 0;
174 }
```

Code 6.3: Solution template (C++) *main.cpp*

```
1  /**
2   * 2025 Lord Kelvin Programming Competition - Solution Template
3   * Language: C++20
4   * * Usage:
5   * - Compile: g++ -O3 main.cpp -o solution
6   * - Local Test: ./solution < input.txt
7   * - CodeGrade: Automatically feeds data via stdin
8   */
9
10 #include <iostream>
11 #include <vector>
12 #include <cmath>
13 #include <iomanip>
14 #include <numeric>
15
16 using namespace std;
17
18 // ==========================================
19 // 1. Data Structure Definitions
20 // ==========================================
21
22 struct Config {
23     int T; // Total duration [cite: 101]
24     int N; // Number of states [cite: 102]
25     int M; // Number of sound effects [cite: 103]
26     int L; // Wandering cycle period [cite: 104]
27     int Type; // 0 = Uniform Penalty, 1 = Generalized Matrix
          Penalty [cite: 106]
28 };
29
30 struct ProblemData {
31     Config cfg;
```

```
32
33     // Initial base rewards R_initial(s)
34     // Size: N
35     vector<int> initial_rewards;
36
37     // Switching Penalty Matrix C_{i,j}
38     // Size: N x N
39     // Note: Always read this matrix regardless of Type [cite:
           108]
40     vector<vector<int>> penalty_matrix;
41
42     // Transition Matrices P^(a)
43     // Structure: [Action_ID][From_State][To_State]
44     // Size: M x N x N
45     vector<vector<vector<double>>> transition_matrices;
46
47     // System average reward centroid (mu), calculated
           automatically
48     double mu;
49 };
50
51 // ========================================
52 // 2. Helper Functions (Math Model)
53 // ========================================
54
55 /**
56  * Calculate Reward R(s,t)
57  * Formula: R(s,t) = mu + (R_initial(s) - mu) * cos(2*pi*t / L) [
        cite: 60]
58  */
59 double get_reward(const ProblemData& data, int state, int t) {
60     if (data.cfg.L == 0) {
61         return (double)data.initial_rewards[state];
62     }
63
64     double r_init = data.initial_rewards[state];
65     // cos takes radians
66     double cosine_term = cos((2.0 * M_PI * t) / (double)data.cfg.L
           );
67     return data.mu + (r_init - data.mu) * cosine_term;
68 }
69
70 /**
71  * Calculate Switching Penalty
72  * Penalty = I(a_t != a_{t-1}) * C_{s_{t-1}, s_t} [cite: 52]
73  */
```

```cpp
int get_penalty(const ProblemData& data, int prev_state, int curr_state, int prev_action, int curr_action) {
    // If Type is 0, penalty is 0 (Uniform mode) [cite: 110]
    if (data.cfg.Type == 0) return 0;

    // No penalty if action doesn't change
    if (prev_action == curr_action) return 0;

    return data.penalty_matrix[prev_state][curr_state];
}

// ==========================================
// 3. Input Parsing (TODO)
// ==========================================

ProblemData read_input() {
    ProblemData data;

    // ------------------------------------------------------------
    // TODO: Implement Data Reading Logic Here
    // ------------------------------------------------------------
    // Refer to Section 4.1 Input Format in the Problem Statement.
    //
    // Step 1: Read Header (T, N, M, L, Type) [cite: 100]
    // Step 2: Read N integers for initial_rewards [cite: 107]
    // *Tip: Calculate 'mu' (average of rewards) here.
    // Step 3: Read Switching Penalty Matrix (N x N) [cite: 108]
    // Step 4: Read Transition Matrices (M blocks, each N x N) [cite: 111]

    // ... Write your cin code here ...

    return data;
}

// ==========================================
// 4. Core Algorithm (Solver)
// ==========================================

vector<int> solve(const ProblemData& data) {
    // Result vector to store action for each minute t=1 to T
    vector<int> actions(data.cfg.T);

    // ------------------------------------------------------------
    // TODO: Implement your Optimization Algorithm Here
    // ------------------------------------------------------------
```

```cpp
    // Objective: Maximize J(A) [cite: 80]
    // You can use Dynamic Programming (DP), Greedy, or MCTS.

    // Placeholder: This fills the array with 0s. Replace this logic.
    for (int t = 0; t < data.cfg.T; ++t) {
        actions[t] = 0;
    }

    return actions;
}

// ==========================================
// 5. Output Formatting
// ==========================================

void print_output(const vector<int>& actions) {
    // Format must be strictly: [a1, a2, ..., aT] [cite: 117]
    cout << "[";
    for (size_t i = 0; i < actions.size(); ++i) {
        cout << actions[i];
        if (i < actions.size() - 1) {
            cout << ", "; // Comma followed by a space
        }
    }
    cout << "]" << endl;
}

// ==========================================
// Main Entry Point
// ==========================================

int main() {
    // Optimize Standard I/O Speed
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // Read Data
    ProblemData data = read_input();

    // Solve
    if (data.cfg.T > 0) {
        vector<int> result = solve(data);
        print_output(result);
    }

```

```
return 0;
}
```

Code 6.4: Solution template (Python) *main.py*

```python
#!/usr/bin/env python3
import sys
import math
from typing import List, Optional

# ==========================================
# 1. Data Structure Definitions
# ==========================================

class Config:
    def __init__(self):
        self.T: int = 0 # Total duration (minutes)
        self.N: int = 0 # Number of states
        self.M: int = 0 # Number of sound effects (actions)
        self.L: int = 0 # Wandering cycle period (0 = Static)
        self.Type: int = 0 # 0 = No Penalty, 1 = Matrix Penalty

class ProblemData:
    def __init__(self):
        self.cfg = Config()

        # Initial base rewards R_initial(s)
        self.initial_rewards: List[int] = []

        # System average reward centroid (mu), calculated during input
        self.mu: float = 0.0

        # Switching Penalty Matrix C_{i,j}
        # Dimensions: [N][N] - Cost of switching from state i to j
        self.penalty_matrix: List[List[int]] = []

        # Transition Matrices P^(a)
        # Dimensions: [M][N][N] - P(from_state -> to_state) given Action
        self.transition_matrices: List[List[List[float]]] = []

# ==========================================
# 2. Logic Helpers (Handles Basic/Ult. Cases)
```

```python
# ==========================================

def get_reward(data: ProblemData, state: int, t: int) -> float:
    """
    Calculates R(s,t).
    Handles both 'Basic/Performance' (L=0) and 'Ultimate' (L>0) cases.
    Ref: Section 3.4 - Time-Variant Reward System
    """
    # Case 1: Static Rewards (L=0)
    if data.cfg.L == 0:
        return float(data.initial_rewards[state])

    # Case 2: Time-Variant Rewards (L > 0)
    # Formula: R(s,t) = mu + (R_initial(s) - mu) * cos(2*pi*t / L)
    r_init = data.initial_rewards[state]
    cosine_term = math.cos((2 * math.pi * t) / data.cfg.L)

    return data.mu + (r_init - data.mu) * cosine_term

def get_penalty(data: ProblemData, prev_state: int, curr_state: int, prev_action: int, curr_action: int) -> int:
    """
    Calculates Switching Penalty.
    Handles 'Basic' (Type=0) and 'Performance/Ultimate' (Type=1) cases.
    Ref: Section 3.4 - Dynamics
    """
    # No penalty if the action does not change
    if prev_action == curr_action:
        return 0

    # Case 1: Uniform/No Penalty Mode (Type=0)
    if data.cfg.Type == 0:
        return 0

    # Case 2: Generalized Matrix Penalty Mode (Type=1)
    # Penalty = C_{prev_state, curr_state}
    return data.penalty_matrix[prev_state][curr_state]

# ==========================================
# 3. Input Parsing (TODO)
# ==========================================

def read_input() -> Optional[ProblemData]:
    """
```

```
    Reads all data from stdin adhering to the competition Input
        Format.
    """
    try:
        # Read all tokens at once for performance (crucial for T
            =3000)
        input_data = sys.stdin.read().split()
        if not input_data:
            return None

        iterator = iter(input_data)
        data = ProblemData()

        # ------------------------------------------------------------
        # TODO: Implement Data Reading Logic Here
        # ------------------------------------------------------------
        # 1. Header: T, N, M, L, Type
        # 2. Base Rewards: N integers (Calculate data.mu here)
        # 3. Penalty Matrix: N*N integers (Always read it, even if
            Type=0)
        # 4. Transition Matrices: M blocks of N*N floats
        #
        # Use `val = next(iterator)` to fetch the next token.

        pass # Remove this pass when implementing

        return data

    except StopIteration:
        return None

# ==========================================
# 4. Core Algorithm (Solver)
# ==========================================

def solve(data: ProblemData) -> List[int]:
    """
    Generates the optimal action sequence [a_1, ..., a_T].
    """
    actions = [0] * data.cfg.T

    # ------------------------------------------------------------
    # TODO: Implement your Optimization Algorithm Here
    # ------------------------------------------------------------
    # NOTE:
    # - For Basic Cases (Small T, N, M)
```

```python
    # - For Performance/Ultimate (Large T)
    return actions

# ==========================================
# 5. Output Formatting
# ==========================================

def print_output(actions: List[int]):
    """
    Formats the output strictly as [a1, a2, ..., aT].
    """
    # Using specific formatting to ensure ", " separator
    formatted_str = "[" + ", ".join(map(str, actions)) + "]"
    print(formatted_str)

# ==========================================
# Main Entry Point
# ==========================================

if __name__ == "__main__":
    # Increase recursion depth just in case (optional)
    sys.setrecursionlimit(5000)

    data = read_input()

    if data and data.cfg.T > 0:
        result = solve(data)
        print_output(result)
```