2026 Team Formation Test Day 2

Task Overview

ID	Name	Time Limit	Memory Limit	Subtasks
T2621	Chained Together	2.000 s	1024 MB	6 + 7 + 10 + 18 + 21 + 15 + 16 + 7
T2622	Chess Tournament	1.000 s	1024 MB	30 + 70
T2623	Peacock Aviary	3.000 s	1024 MB	5 + 7 + 16 + 9 + 14 + 23 + 22 + 4

Notice:

- All tasks are divided into subtasks. You need to pass all test cases in a subtask to get points.
- There is an attachment package that you can download from the contest system, containing sample graders, sample implementations, example test cases, and compile and run scripts.
- When testing your programs with the sample grader, your input should match the format and constraints from the task statement, otherwise, unspecified behaviors may occur.
- In sample grader inputs, every two consecutive tokens on a line are separated by a single space, unless another format is explicitly specified.
- The task statements specify signatures using generic type names such as `void`, `bool`, `string`, `int`, `int64`, `pair`, `int[]` (array) and `int[][]` (array of array).
- In C++, the graders use appropriate data types or implementations, as listed below:

void	**bool**	**string**	**int**	**int64**
`void`	`bool`	`std::string`	`int`	`long long`

pair(int, bool)	**int[]**	**int[][]**	**length of array a**
`std::pair<int, bool>`	`std::vector<int>`	`std::vector<std::vector<int>>`	`a.size()`

T2621 - CHAINED TOGETHER

Time Limit: 2.000 s / Memory Limit: 1024 MB

It's school picnic day again! This year, your entire class, together with your class teacher, will be visiting a country park which can be modelled as a tree with N nodes, labelled $0, 1, \ldots, N-1$. The i-th edge of the tree (where $0 \leq i \leq N-2$) connects node $U[i]$ and node $V[i]$. For nodes $0 \leq u, v \leq N-1$, we denote the distance between two nodes u and v as the number of edges on the simple path connecting nodes u and v. Also, we say that node v is a neighbour of node u if and only if there is an edge directly connecting them.

Enjoying the cheerful and energetic vibes, your entire class decides to play a collaborative game, with your class teacher assigning the class a series of missions.

Your class number is 0, and your remaining $M-1$ classmates each have a distinct class number between 1 and $M-1$ inclusive. Your class teacher first assigns an initial position for every student: for every $0 \leq j \leq M-1$, the student with class number j is assigned to stand at node $A[j]$ of the tree initially. It is possible for multiple students to stand on the same node.

After all M students have settled into their initial positions, the class teacher then chains pairs of students together using chains of length K: specifically, for every $0 \leq j \leq M-2$, your class teacher adds a chain of length K between the student with class number j and the student with class number $j+1$.

Once a chain is added between two students, their distance between each other during any point of the game must never exceed K. The distance between two students is defined to be the distance of the two nodes where they are currently located. It is guaranteed that for all $0 \leq j \leq M-2$, the distance between node $A[j]$ and node $A[j+1]$ is at most K, so that all chains can be properly added.

After putting on $M-1$ chains, your class teacher will then announce missions one by one, for a total of Q missions. The k-th mission (where $0 \leq k \leq Q-1$) contains a single node $X[k]$. This indicates that the target of the k-th mission is for you (with class number 0) to go and stand at node $X[k]$. To complete the mission, you can perform instructions. In an instruction, you can instruct any one of your M classmates, including yourself, to move to a neighbour of the node that the instructed classmate is currently standing at.

Since you have to shout each instruction out loud, you realise such a process could make you lose your voice rather quickly. Therefore, you want to minimise the number of instructions to complete each mission. Remember that your instructions must never allow any two students with consecutive class numbers to have a distance larger than K at any time.

Note that the Q missions are persistent: all M students stay at their new positions and start the next mission (if any) directly. They do not go back to their initial positions. It can be proven that it is always possible to complete the current mission, and that the optimal positions after each mission are always uniquely determined.

Your task is simple: for each mission, compute the minimum number of instructions that you have to shout to complete it, under the constraint that no two students with consecutive class numbers can have a distance larger than K at any time.

IMPLEMENTATION DETAILS

You should implement the following procedures:

```
void init(int N, int[] U, int[] V, int M, int[] A, int K)
```

- This procedure is called exactly once per run, before any calls to `minimum_instructions`.
- N: The number of nodes of the tree.
- U, V: Arrays of size $N-1$, representing the edges of the tree. The i-th edge (where $0 \leq i \leq N-2$) connects between node $U[i]$ and node $V[i]$.
- M: The number of students in the class.
- A: Array of size M, representing the initial position of each student.
- K: The length of each chain.

`int64 minimum_instructions(int X)`

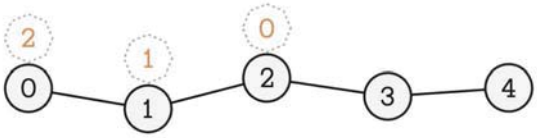
- This procedure is called exactly Q times.
- X: the node that you have to stand at, in order to complete the current mission. If this is the i-th call (where $0 \leq i \leq Q-1$) to the procedure `minimum_instructions`, then this represents $X[i]$.
- This procedure should return an integer, the minimum number of instructions required to complete the current mission.

EXAMPLE

Consider the following call:

`init(5, [0, 1, 2, 3], [1, 2, 3, 4], 3, [2, 1, 0], 1)`

The initial state is illustrated in the following figure, where the dotted circles indicate where students are currently standing at:

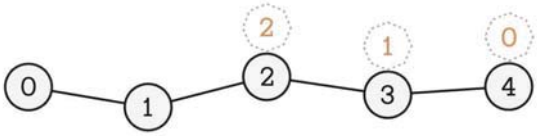


After initialisation has been done, consider the following call:

`minimum_instructions(4)`

A possible sequence involving 6 instructions is as follows:

- The student with class number 2 moves from node 0 to node 1.
- Then, the student with class number 2 moves from node 1 to node 2.
- Then, the student with class number 1 moves from node 1 to node 2.
- Then, the student with class number 1 moves from node 2 to node 3.
- Then, you (with class number 0) move from node 2 to node 3.
- Then, you move from node 3 to node 4.



You have moved to node $X[0] = 4$ while ensuring all pairs of students with consecutive class numbers having distance at most $K = 1$ after every move. It can be proven that it is impossible for you to move to node 4 in fewer than 6 instructions without breaking the requirement. It can also be proven that this is the only valid list of resulting locations where students will stand, under the assumption that you use at most 6 instructions to reach node 4. Therefore, the procedure should return `6`.

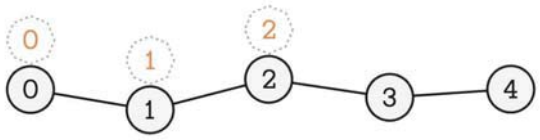
Consider the following subsequent call:

`minimum_instructions(0)`

A possible sequence involving 6 instructions is as follows:

- You move from node 4 to node 3.
- Then, you move from node 3 to node 2.
- Then, the student with class number 1 moves from node 3 to node 2.

- Then, the student with class number 1 moves from node 2 to node 1.
- Then, you move from node 2 to node 1.
- Then, you move from node 1 to node 0.



It can be proven that it is impossible for you to move to node $X[1] = 0$ in fewer than 6 instructions. So, the procedure should return `6`.

SAMPLE GRADER

The sample grader reads input in the following format:

- The first line contains three integers, N, M, and K.
- The $(i+1)$-th of the next $N-1$ lines contains two integers, $U[i]$ and $V[i]$.
- The $(N+1)$-th line contains M integers, $A[0], A[1], \ldots, A[M-1]$.
- The $(N+2)$-th line contains one integer, Q.
- The $(i+1)$-th of the next Q lines contains one integer, $X[i]$.

The sample grader prints your answers in the following format:

- The $(i+1)$-th of Q lines contains a single integer, the returned integer of the i-th call to `minimum_instructions`.

SAMPLE TESTS

	Input	Output
1	5 3 1 0 1 1 2 2 3 3 4 2 1 0 2 4 0	6 6
2	5 3 2 0 1 0 2 3 1 4 1 3 4 0 2 2 3	4 3

SUBTASKS

For all cases:
$1 \leq K < N \leq 2 \times 10^5$
$2 \leq M \leq 2 \times 10^5$
$0 \leq U[i], V[i] \leq N-1$ for all $0 \leq i \leq N-2$
$U[i] \neq V[i]$ for all $0 \leq i \leq N-2$
The $N-1$ edges form a tree
$0 \leq A[i] \leq N-1$ for all $0 \leq i \leq M-1$
The distance between node $A[i]$ and node $A[i+1]$ is at most K for all $0 \leq i \leq M-2$
$1 \leq Q \leq 4 \times 10^5$
$0 \leq X[i] \leq N-1$ for all $0 \leq i \leq Q-1$

	Points	Constraints
1	6	$N, M, Q \leq 100$ $U[i] = i$ and $V[i] = i+1$ for all $0 \leq i \leq N-2$
2	7	$N, M, Q \leq 500$
3	10	$N, M, Q \leq 2500$
4	18	$N, M, Q \leq 10^5$ $U[i] = i$ and $V[i] = i+1$ for all $0 \leq i \leq N-2$
5	21	$N, M, Q \leq 10^5$ $K = 1$ $X[0]$ is a neighbour of $A[0]$ $X[i]$ is a neighbour of $X[i-1]$ for all $1 \leq i \leq Q-1$
6	15	$N, M, Q \leq 10^5$ $X[0]$ is a neighbour of $A[0]$ $X[i]$ is a neighbour of $X[i-1]$ for all $1 \leq i \leq Q-1$
7	16	$N, M, Q \leq 10^5$
8	7	No additional constraints

HINT

Be aware of the potentially tight time limit.

T2622 - CHESS TOURNAMENT

Time Limit: 1.000 s / Memory Limit: 1024 MB

A total of N players have joined the Grand Chess Tournament of Hackerland! The players are labelled as player 0, player 1, till player $N-1$. As the organiser of the tournament, you have decided this year, prizes will be given to the top R players. Since you want this tournament to be fair, you want to distribute prizes such that the strongest player gets the best prize, second strongest player gets the second best prize, etc.

Each player has a unique rating. Thus, we can rank the players in descending order of rating as player $A[0]$, player $A[1]$, till player $A[N-1]$. For example, suppose $A = [2, 5, 4, 0, 6, 7, 1, 3]$, then player 2 is the strongest player, player 5 is the second strongest player, etc, and player 3 is the weakest player out of all. When two player play a *game* of chess together, the player who has a higher rating always wins. That is, whenever player $A[x]$ plays against player $A[y]$ where $x < y$, player $A[x]$ always wins.

Each *round* of the tournament consists of several games. The games within the same round are **played simultaneously in parallel**, so no player can play two games at the same time. On the other hand, it is perfectly fine for some players not to play in some rounds. After all the games in the round ends, the results of the games are sent back to you, in which you can use them to determine the games for the next round, if necessary.

You want to find the top R players, **in the correct order**, using as few rounds as possible, since, well, more rounds mean more work to do! Write a program to help you schedule the rounds and also determine the top players.

IMPLEMENTATION DETAILS

You should implement the procedure `chess_tournament`:

```
int[] chess_tournament(int N, int R)
```

- This procedure will be called exactly once per run. You should implement your strategy here.
- N: The number of players.
- R: The number of top players you have to find, in descending order of rating.
- You should return an array of length R, indicating the labels of the top R players in descending order of rating.

The above procedure can make calls to the following procedure:

```
int[] play_round(pair(int, int)[] games)
```

- This procedure simulate a round of games between players.
- $games$: An array of length K, describing the players in the games. The i-th game ($0 \le i < K$) is represented by $games[i] = (x[i], y[i])$, which means the game is played between player $x[i]$ and player $y[i]$.
- The indices $x[i]$ and $y[i]$ must satisfy $0 \le x[i], y[i] < N$ and $x[i] \neq y[i]$.
- Each player must appear at most once in $games$.
- The function returns an array of length K, the winner of each game. The i-th ($0 \le i < K$) element is either $x[i]$ or $y[i]$, the player index of the winner in game i.
- You may only call this function **at most 150 times**.

Important Notice

In some test cases the behavior of the grader is **adaptive**. This means that in these test cases the grader does not have a fixed order of the player ratings. Instead, the answers given by the grader may depend on the questions asked by your solution.

It is guaranteed that the grader answers in such a way that after each answer there is at least one order of player ratings consistent with all the answers given so far.

EXAMPLE

Suppose $N = 8$, $R = 2$, and the players in descending order of ratings are $A = [2, 5, 4, 0, 6, 7, 1, 3]$. The procedure `chess_tournament` is called as

```
chess_tournament(8, 2)
```

Let's say the in the first round, you want to have games between

- player 0 vs player 1,
- player 2 vs player 3,
- player 4 vs player 5, and
- player 6 vs player 7.

Then you should call

```
play_round([(0, 1), (2, 3), (4, 5), (6, 7)])
```

Which it would return $[0, 2, 5, 6]$ as the winner of the games.

Suppose in the second round you want to have games between

- player 2 vs player 5, and
- player 4 vs player 6.

Then you should call

```
play_round([(2, 5), (4, 6)])
```

Which it would return $[2, 4]$ as the winner of the games.

Suppose you deduce that the strongest and second strongest player out of all the players are player 2 and player 5. Then the `chess_tournament` procedure should return $[2, 5]$. Please note that the order matters; for example, $[5, 2]$ is not accepted.

SAMPLE GRADER

The sample grader reads input in the following format:

- The first line consists of two integers, N and R.
- The next lines consists of N integers, $A[0], A[1], \ldots, A[N-1]$.

For the run, if your program is judged as incorrect, the type of the error will be outputted (`Wrong Answer: Reason`).

Otherwise, the program outputs `Accepted` followed by the value of Q, the number of times `play_round` is called.

Please note that the sample grader does not check whether the input is valid or not. Unexpected behaviour might occur if the input is invalid.

SAMPLE TESTS

	Input	Output
1	8 2 2 5 4 0 6 7 1 3	Accepted: 4

SUBTASKS

For all cases:
$0 \le A[i] < N$ for all $0 \le i < N$
$A[i] \ne A[j]$ for all $0 \le i < j < N$

	Points	Constraints
1	30	$N = 8$, $R = 2$
2	70	$N = 4096$, $R = 8$

SCORING

For a test case, you will receive zero score if your program does not exit correctly, does not determine the top R players correctly, or violates any requirements in the task description. Otherwise, let K be the maximum number of calls to the function `play_round` over all test cases. Your score will be calculated as follows.

In subtask 1, your score is calculated using the following formula:

$$score = \begin{cases} 30, & \text{if } K \le 4 \\ 15 + 5 \times (7 - K), & \text{if } 4 < K \le 7 \\ 7, & \text{if } 7 < K \le 13 \\ 0, & \text{if } K > 13 \end{cases}$$

In subtask 2, your score is calculated using the following formula:

$$score = \begin{cases} 70, & \text{if } K \le 19 \\ 61 + 3 \times (22 - K), & \text{if } 19 < K \le 22 \\ 45 + 0.8 \times (42 - K), & \text{if } 22 < K \le 42 \\ 26 + 0.5 \times (80 - K), & \text{if } 42 < K \le 80 \\ 12 + 0.2 \times (150 - K), & \text{if } 80 < K \le 150 \\ 0, & \text{if } K > 150 \end{cases}$$

T2623 - PEACOCK AVIARY

Time Limit: 3.000 s / Memory Limit: 1024 MB

Alice and Bob are visiting a peacock aviary consisting of N areas, numbered from 0 to $N-1$. There are K types of peacocks, numbered from 0 to $K-1$. For $0 \le i \le N-1$, area i contains a peacock of type $T[i]$ on the 0-th day. The N areas are connected by $N-1$ **directed** roads. For $1 \le i \le N-1$, the i-th road goes from area $P[i]$ to area i, where $P[i] < i$. We also assume $P[0] = -1$.

Alice and Bob go to the aviary for $Q+1$ days. On each day, they start their visit at area 0. At any moment, if they are at area u, they will perform the following operations:

1. Observe the peacock at area u.
2. If there are no areas they can move to from area u, their visit ends.
3. Otherwise, one of them chooses a road starting at area u, and they move to the destination of the chosen road together. If $C[u] = 0$, Alice decides which road to choose; if $C[u] = 1$, Bob decides which road to choose.

The aviary will be updated daily from Day 1 to Day Q. For $0 \le i \le Q-1$, the type of peacock at area $X[i]$ changes to type $Y[i]$ at the beginning of day $i+1$. These updates are persistent, meaning each change remains in effect for all following days.

Before starting the travel, Alice picks a peacock type as her favorite. She wants to choose a type such that, regardless of Bob's decisions, she can guarantee that she will observe that peacock type at least once by making optimal decisions. For each day, please find the number of peacock types that could possibly be Alice's favorite.

IMPLEMENTATION DETAILS

You should implement the following procedure:

```
int[] count_peacocks(int N, int K, int Q, int[] P, int[] C, int[] T, int[] X, int[] Y)
```

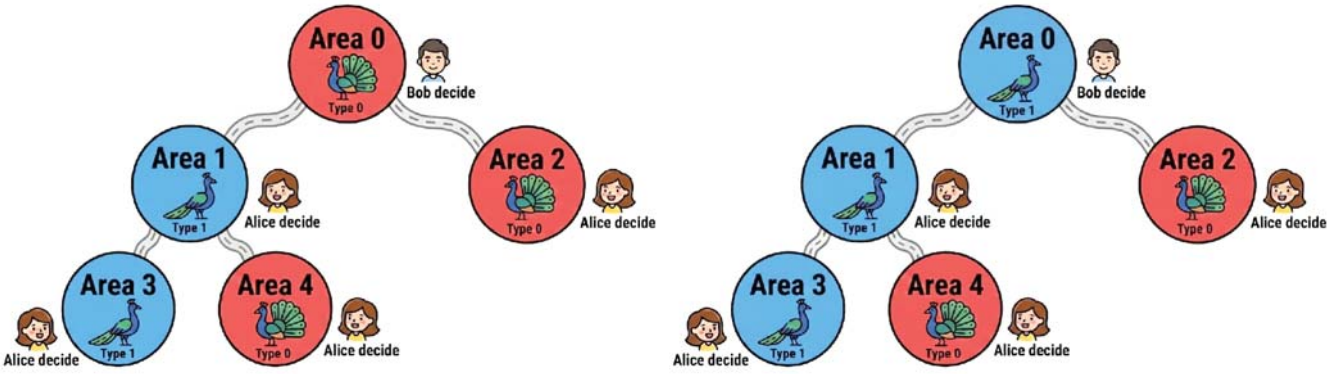
- This procedure is called exactly once per test case.
- N: The number of areas in the aviary.
- K: The number of peacock types.
- Q: The number of updates.
- P: Array of length N, where $P[0] = -1$ and the remaining $N-1$ elements represent the roads in the aviary. The i-th road (where $1 \le i \le N-1$) goes from area $P[i]$ to area i.
- C: Array of length N, representing the people making the decision at each area.
- T: Array of length N, representing the peacock types at each area.
- X, Y: Arrays of length Q, representing updates to the aviary. At the beginning of day $i+1$ (where $0 \le i \le Q-1$), the type of peacock at area $X[i]$ changes to type $Y[i]$.
- The procedure should return an array of length $Q+1$, representing the number of peacock types that could possibly be Alice's favorite for each day.

EXAMPLE

Consider the following call:

```
count_peacocks(5, 2, 1, [-1, 0, 0, 1, 1], [1, 0, 0, 0, 0], [0, 1, 0, 1, 0], [0], [1])
```

The map of the aviary on each day is visualised as follows:



Day 0 Day 1

On day 0:

- **For peacocks of type 0:** Alice observes the peacock of type 0 at area 0. Therefore, type 0 is a possible favorite.
- **For peacocks of type 1:** If Bob decides to move to area 2 after area 0, Alice will not observe any peacock of type 1. Therefore, type 1 cannot be Alice's favorite.

In total, one peacock type could possibly be Alice's favorite.

On Day 1:

- **For peacocks of type 0:**
  - If Bob decides to move to area 1 after area 0, Alice can move to area 4 and observe the peacock of type 0 at area 4.
  - If Bob decides to move to area 2 after area 0, Alice will observe the peacock of type 0 at area 2.

  Type 0 could be Alice's favorite.
- **For peacocks of type 1:** Alice will observe the peacock of type 1 at area 0. Therefore, type 1 could be Alice's favorite.

In total, two peacock types could possibly be Alice's favorite.

To report the answers, `count_peacocks` should return `[1, 2]`.

SAMPLE GRADER

The sample grader reads the input in the following format:

- The first line consists of three integers: N, K, Q.
- The next line consists of $N-1$ integers: $P[1], P[2], \ldots, P[N-1]$.
- The next line consists of N integers: $C[0], C[1], \ldots, C[N-1]$.
- The next line consists of N integers: $T[0], T[1], \ldots, T[N-1]$.
- The next Q lines consist of two integers: $X[i]$ and $Y[i]$.

Note that the second line of the input contains **only $N-1$ integers**, as the sample grader does not read the value of $P[0]$.

The sample grader outputs one integer on each line, the elements of the returned array.

SAMPLE TESTS

	Input	Output
1	5 2 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1	1 2
2	10 10 10 0 0 0 2 2 4 1 7 5 0 1 0 1 0 1 1 0 1 0 7 1 7 3 1 8 7 6 7 0 0 7 9 2 5 5 0 8 4 2 4 9 5 5 3 1 5 1 4 6	6 6 6 6 7 7 8 8 7 6 5

SUBTASKS

For all cases:
$2 \le N, K \le 2 \times 10^5$
$0 \le Q \le 2 \times 10^5$
$P[0] = -1$
$0 \le P[i] < i$ for all $1 \le i \le N-1$
$0 \le C[i] \le 1$ for all $0 \le i \le N-1$
$0 \le T[i] \le K-1$ for all $0 \le i \le N-1$
$0 \le X[i] \le N-1$ for all $0 \le i \le Q-1$
$0 \le Y[i] \le K-1$ for all $0 \le i \le Q-1$

	Points	Constraints
1	5	$N, K, Q \le 5000$ $P[i] = 0$ for all $1 \le i \le N-1$
2	7	$N, K, Q \le 5 \times 10^4$ $P[i] = 0$ for all $1 \le i \le N-1$
3	16	$N, K \le 5000$ $Q = 0$
4	9	$N, K, Q \le 5000$
5	14	$N, K, Q \le 5 \times 10^4$ $P[i] = \left\lfloor \frac{i-1}{2} \right\rfloor$ for all $1 \le i \le N-1$
6	23	$N, K \le 5 \times 10^4$ $Q = 0$
7	22	$N, K, Q \le 5 \times 10^4$
8	4	No additional constraints