Code

C Auto

```c
#include <stdbool.h>
#include <stdlib.h>

typedef struct {
    int* data;
    int front, rear, size, capacity;
} Queue;

// Queue functions
Queue* createQueue(int capacity) {
    Queue* q = (Queue*)malloc(sizeof(Queue));
    q->data = (int*)malloc(sizeof(int) * capacity);
    q->front = q->rear = q->size = 0;
    q->capacity = capacity;
    return q;
}

void enqueue(Queue* q, int val) {
    if (q->size == q->capacity) return;
    q->data[q->rear] = val;
    q->rear = (q->rear + 1) % q->capacity;
    q->size++;
}

int dequeue(Queue* q) {
    if (q->size == 0) return -1;
    int val = q->data[q->front];
    q->front = (q->front + 1) % q->capacity;
    q->size--;
    return val;
}

bool isEmpty(Queue* q) {
    return q->size == 0;
}

// Graph using adjacency list
typedef struct Node {
    int val;
    struct Node* next;
} Node;
```

Saved

Ln 1, Col 1

Testcase | Test Result

Code

C Auto

```c
    int val;
    struct Node* next;
} Node;

Node** createGraph(int n) {
    Node** graph = (Node**)malloc(sizeof(Node*) * n);
    for (int i = 0; i < n; i++)
        graph[i] = NULL;
    return graph;
}

void addEdge(Node** graph, int u, int v) {
    Node* node1 = (Node*)malloc(sizeof(Node));
    node1->val = v;
    node1->next = graph[u];
    graph[u] = node1;

    Node* node2 = (Node*)malloc(sizeof(Node));
    node2->val = u;
    node2->next = graph[v];
    graph[v] = node2;
}

bool validPath(int n, int** edges, int edgesSize, int* edgesColSize, int source, int destination) {
    // Build graph
    Node** graph = createGraph(n);
    for (int i = 0; i < edgesSize; i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        addEdge(graph, u, v);
    }

    // BFS
    bool* visited = (bool*)calloc(n, sizeof(bool));
    Queue* q = createQueue(n);
    enqueue(q, source);
    visited[source] = true;

    while (!isEmpty(q)) {
        int node = dequeue(q);
        if (node == destination) return true;
```

Saved

Ln 83, Col 20

Testcase | Test Result

Code

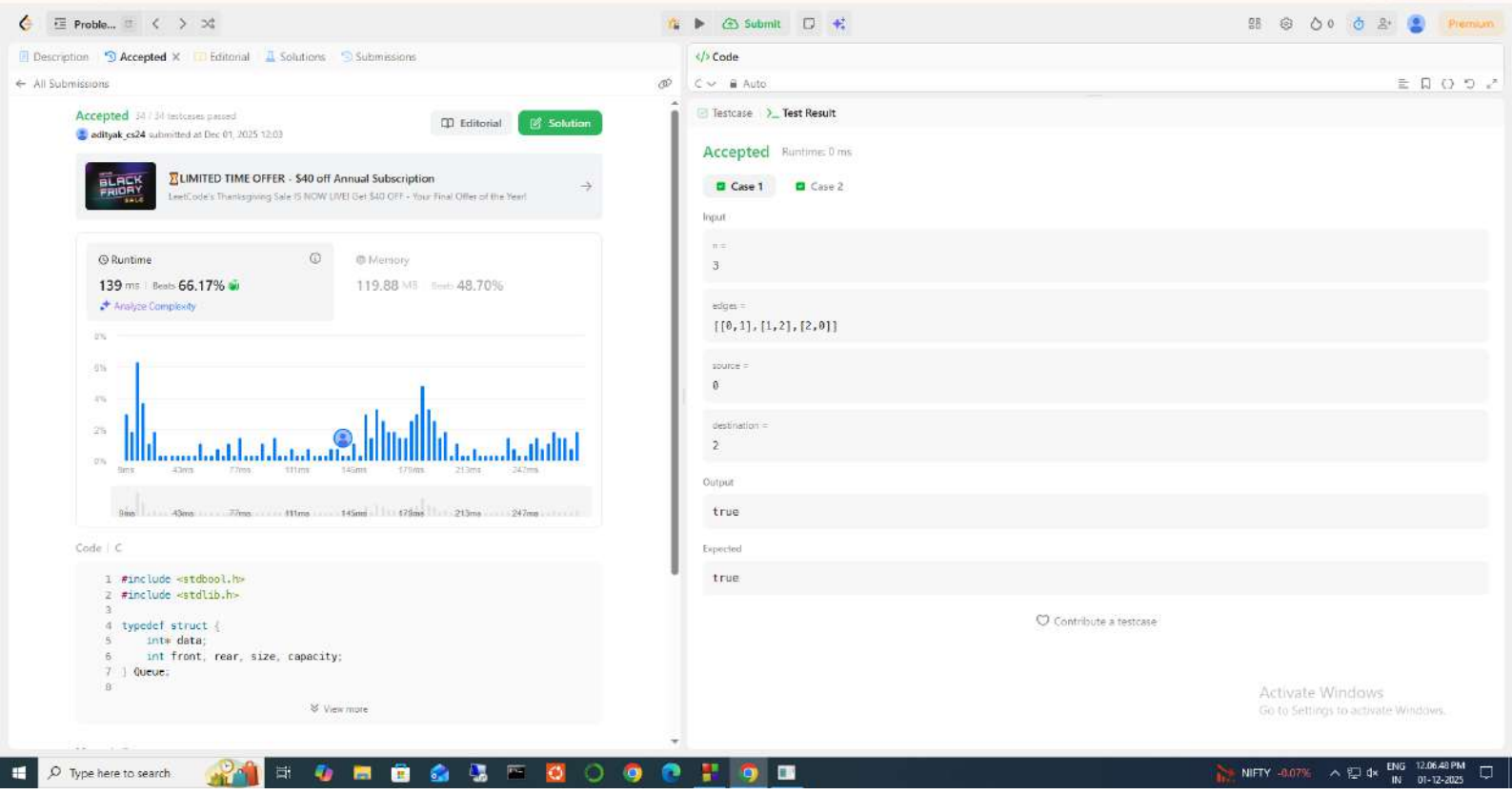
C Auto

```
    node2->val = u;
    node2->next = graph[v];
    graph[v] = node2;
}

bool validPath(int n, int** edges, int edgesSize, int* edgesColSize, int source, int destination) {
    // Build graph
    Node** graph = createGraph(n);
    for (int i = 0; i < edgesSize; i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        addEdge(graph, u, v);
    }

    // BFS
    bool* visited = (bool*)calloc(n, sizeof(bool));
    Queue* q = createQueue(n);
    enqueue(q, source);
    visited[source] = true;

    while (!isEmpty(q)) {
        int node = dequeue(q);
        if (node == destination) return true;

        Node* neighbor = graph[node];
        while (neighbor) {
            if (!visited[neighbor->val]) {
                visited[neighbor->val] = true;
                enqueue(q, neighbor->val);
            }
            neighbor = neighbor->next;
        }
    }

    return false;
}
```

Saved

Ln 83, Col 20

Testcase | Test Result


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 34 / 34 testcases passed
adityak_cs24 submitted at Dec 01, 2025 12:03
Editorial
Solution
LIMITED TIME OFFER - $40 off Annual Subscription
Runtime
139 ms | Beats 66.17%
Analyze Complexity
Memory
119.88 MB | Beats 48.70%
Code | C
#include <stdbool.h>
#include <stdlib.h>
typedef struct {
int* data;
int front, rear, size, capacity;
} Queue;
View more
Code
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
n =
3
edges =
[[0,1],[1,2],[2,0]]
source =
0
destination =
2
Output
true
Expected
true
Contribute a testcase
Activate Windows
Go to Settings to activate Windows.