```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
using namespace std;

struct Node {
    string name;
    string path;
    int clicks;

    Node(string name, string path, int clicks) : name(name), path(path),
clicks(clicks) {}
};

void bfs(const unordered_map<string, vector<string>>& filesystem, const string&
target) {
    queue<Node> q;
    q.push(Node("C:", "C:", 0));

    while (!q.empty()) {
        Node curr = q.front();
        q.pop();

        if (curr.name == target) {
            cout << curr.path << endl;
            cout << curr.clicks << endl;
            return;
        }

        if (filesystem.find(curr.name) != filesystem.end()) {
            for (const string& child : filesystem.at(curr.name)) {
                string childPath = curr.path + "\\" + child;
                int childClicks = curr.clicks + 2;
                q.push(Node(child, childPath, childClicks));
            }
        }
    }

    cout << "File not found" << endl;
    cout << -1 << endl;
}

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

    unordered_map<string, vector<string>> filesystem;

    for (int i = 0; i < e; i++) {
        string parent, child;
        cin >> parent >> child;
        filesystem[parent].push_back(child);
    }

    int q;
    cin >> q;

    for (int i = 0; i < q; i++) {
        string target;
        cin >> target;
        bfs(filesystem, target);
    }

    return 0;
}
```

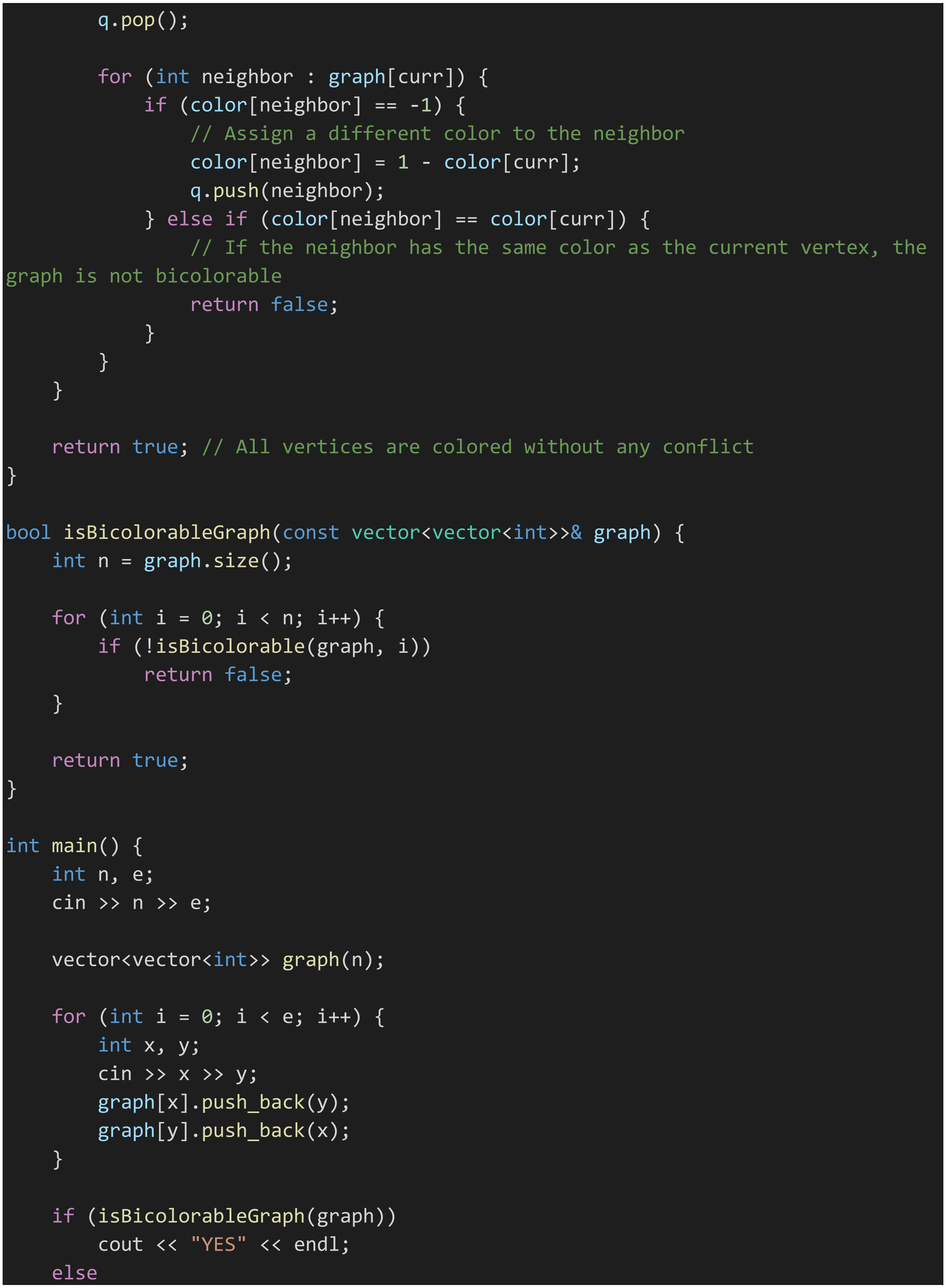
BFS SWAPNIL SIR 2

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

bool isBicolorable(const vector<vector<int>>& graph, int start) {
    int n = graph.size();
    vector<int> color(n, -1); // Initialize all vertices with no color
    color[start] = 0; // Assign the start vertex color 0

    queue<int> q;
    q.push(start);

    while (!q.empty()) {
        int curr = q.front();
```



```
        q.pop();

        for (int neighbor : graph[curr]) {
            if (color[neighbor] == -1) {
                // Assign a different color to the neighbor
                color[neighbor] = 1 - color[curr];
                q.push(neighbor);
            } else if (color[neighbor] == color[curr]) {
                // If the neighbor has the same color as the current vertex, the graph is not bicolorable
                return false;
            }
        }
    }

    return true; // All vertices are colored without any conflict
}

bool isBicolorableGraph(const vector<vector<int>>& graph) {
    int n = graph.size();

    for (int i = 0; i < n; i++) {
        if (!isBicolorable(graph, i))
            return false;
    }

    return true;
}

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

    vector<vector<int>> graph(n);

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

    if (isBicolorableGraph(graph))
        cout << "YES" << endl;
    else
```

```
        cout << "NO" << endl;

    return 0;
}
```

BFS SWAPNIL SIR 3

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
using namespace std;

// Structure to represent a chess cell
struct Cell {
    int row;
    int col;

    Cell(int r, int c) : row(r), col(c) {}
};

// Function to check if a cell is valid and within the chessboard limits
bool isValidCell(int row, int col) {
    return (row >= 0 && row < 8 && col >= 0 && col < 8);
}

// Function to get the minimum number of moves required for the knight to reach the destination
int getMinMoves(Cell start, Cell dest) {
    // Offsets for the knight's possible moves
    int rowOffsets[] = {-2, -2, -1, -1, 1, 1, 2, 2};
    int colOffsets[] = {-1, 1, -2, 2, -2, 2, -1, 1};

    // Create a visited map to keep track of visited cells
    unordered_map<int, unordered_map<int, bool>> visited;

    // Create a map to store the parent cell for each visited cell
    unordered_map<int, unordered_map<int, Cell>> parent;

    // Create a queue for BFS traversal
    queue<Cell> q;
    // Initialize the starting cell as visited
    visited[start.row][start.col] = true;

    // Enqueue the starting cell
    q.push(start);

    // Perform BFS traversal
    while (!q.empty()) {
        Cell curr = q.front();
        q.pop();

        // If the current cell is the destination, return the minimum number of moves
        if (curr.row == dest.row && curr.col == dest.col) {
            return parent[curr.row][curr.col].row;
        }

        // Explore all possible moves from the current cell
        for (int i = 0; i < 8; i++) {
            int newRow = curr.row + rowOffsets[i];
            int newCol = curr.col + colOffsets[i];

            // Check if the new cell is valid and not visited
            if (isValidCell(newRow, newCol) && !visited[newRow][newCol]) {
                visited[newRow][newCol] = true;
                parent[newRow][newCol] = curr;
                q.push(Cell(newRow, newCol));
            }
        }
    }

    // If the destination cell cannot be reached, return -1
    return -1;
}

// Function to print the path from the starting cell to the destination cell
void printPath(Cell start, Cell dest) {
    vector<Cell> path;
    Cell curr = dest;

    // Traverse the parent map to build the path
    while (curr.row != start.row || curr.col != start.col) {
        path.push_back(curr);
        curr = parent[curr.row][curr.col];
    }
    // Print the minimum number of moves
    cout << path.size() << endl;

    // Print the path in reverse order
    for (int i = path.size() - 1; i >= 0; i--) {
        cout << static_cast<char>('A' + path[i].row) << path[i].col + 1;

        if (i > 0) {
            cout << "->";
        }
    }

    cout << endl;
}

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

    for (int i = 0; i < q; i++) {
        string startCell, destCell;
        cin >> startCell >> destCell;

        // Convert the cell positions to row and column numbers
        Cell start(startCell[0] - 'A', startCell[1] - '1');
        Cell dest(destCell[0] - 'A', destCell[1] - '1');

        // Get the minimum number of moves and print the path
        int minMoves = getMinMoves(start, dest);
        printPath(start, dest);
    }

    return 0;
}
```

BFS SWAPNIL SIR 4

```
#include <iostream>
#include <queue>
using namespace std;

// Node structure for BST
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

// Function to insert a value into the BST
Node* insertNode(Node* root, int value) {
    if (root == nullptr)
        return new Node(value);

    if (value < root->data)
        root->left = insertNode(root->left, value);
    else
        root->right = insertNode(root->right, value);

    return root;
}

// Function to perform level-wise traversal of the BST
void levelOrderTraversal(Node* root) {
    if (root == nullptr)
        return;

    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        int size = q.size();

        for (int i = 0; i < size; i++) {
            Node* current = q.front();
            q.pop();

            cout << current->data << " ";

            if (current->left)
                q.push(current->left);

            if (current->right)
                q.push(current->right);
        }
        cout << endl; // Print newline after each level
    }
}

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

    Node* root = nullptr;

    for (int i = 0; i < n; i++) {
        int value;
        cin >> value;
        root = insertNode(root, value);
    }

    levelOrderTraversal(root);

    return 0;
}
```

BFS SWAPNIL SIR 5

```
#include <iostream>
#include <queue>
#include <string>
using namespace std;

// Function to generate binary strings of length up to n
void generateBinaryStrings(int n) {
    queue<string> q;
    q.push("0");
    q.push("1");

    for (int i = 1; i <= n; i++) {
        cout << "Length " << i << ": ";

        int size = q.size();
        for (int j = 0; j < size; j++) {
            string curr = q.front();
            q.pop();
            cout << curr << " ";

            q.push(curr + "0");
            q.push(curr + "1");
        }

        cout << endl;
    }
}

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

    generateBinaryStrings(n);

    return 0;
}
```

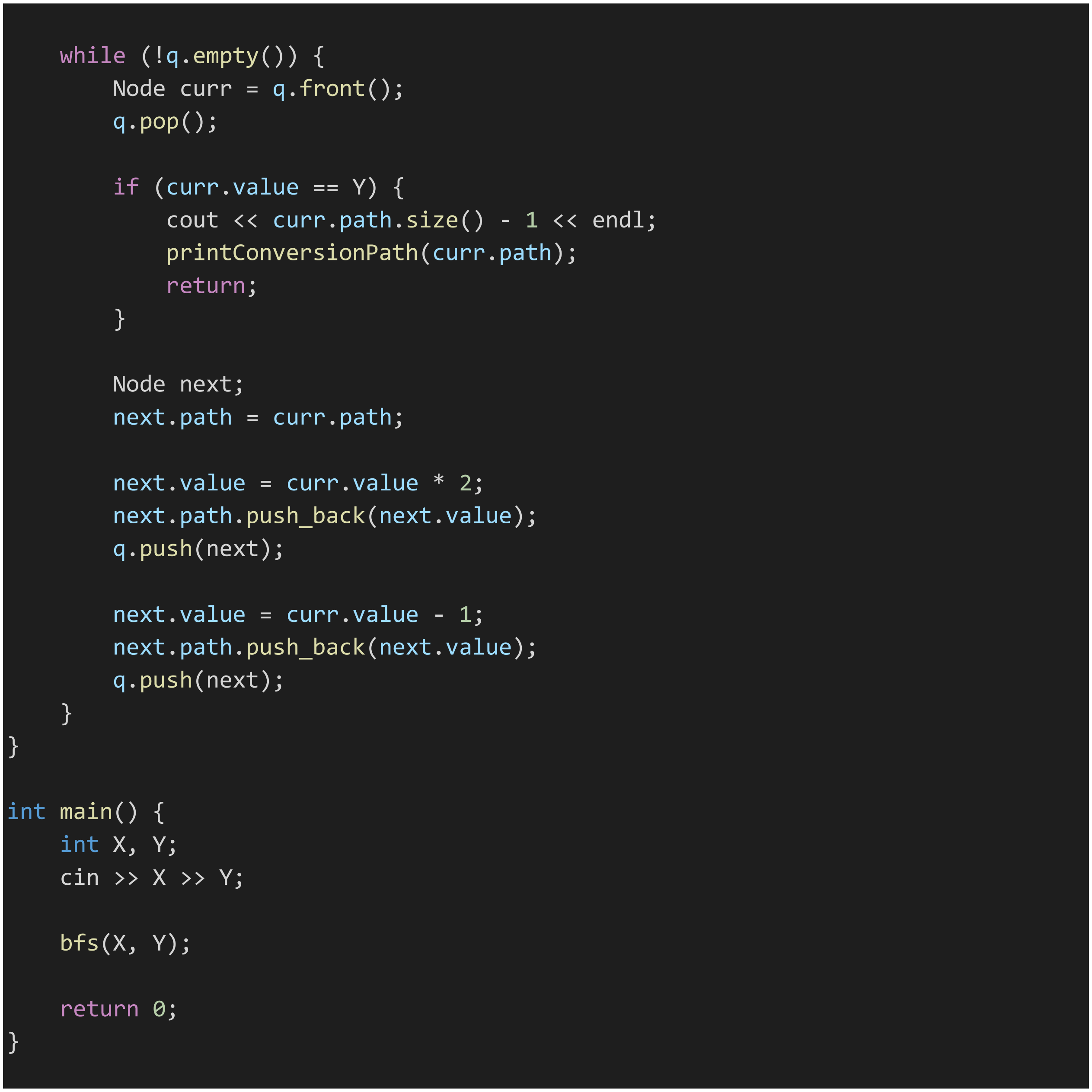
BFS SWAPNIL SIR 6

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct Node {
    int value;
    vector<int> path;
};

void printConversionPath(const vector<int>& path) {
    int n = path.size();
    for (int i = 0; i < n; i++) {
        cout << path[i];
        if (i != n - 1)
            cout << " -> ";
    }
    cout << endl;
}

void bfs(int X, int Y) {
    queue<Node> q;
    Node start;
    start.value = X;
    start.path.push_back(X);
    q.push(start);
```



```
    while (!q.empty()) {
        Node curr = q.front();
        q.pop();

        if (curr.value == Y) {
            cout << curr.path.size() - 1 << endl;
            printConversionPath(curr.path);
            return;
        }

        Node next;
        next.path = curr.path;

        next.value = curr.value * 2;
        next.path.push_back(next.value);
        q.push(next);

        next.value = curr.value - 1;
        next.path.push_back(next.value);
        q.push(next);
    }
}

int main() {
    int X, Y;
    cin >> X >> Y;

    bfs(X, Y);

    return 0;
}
```