BREADTH-FIRST SEARCH

- ALGORITHM DESCRIPTION
- ALGORITHM EXAMPLE
- ALGORITHM COMPLEXITY
- ALGORITHM APPLICATIONS
- PRACTICE: PATH WITH MINIMAL TRANSFERS

Author: prof. Yevhenii Borodavka

ALGORITHM DESCRIPTION

Traversal means visiting all the nodes of a graph. Breadth First Traversal or Breadth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.

A standard BFS implementation puts each vertex of the graph into one of two categories:

- Visited
- Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

ALGORITHM DESCRIPTION

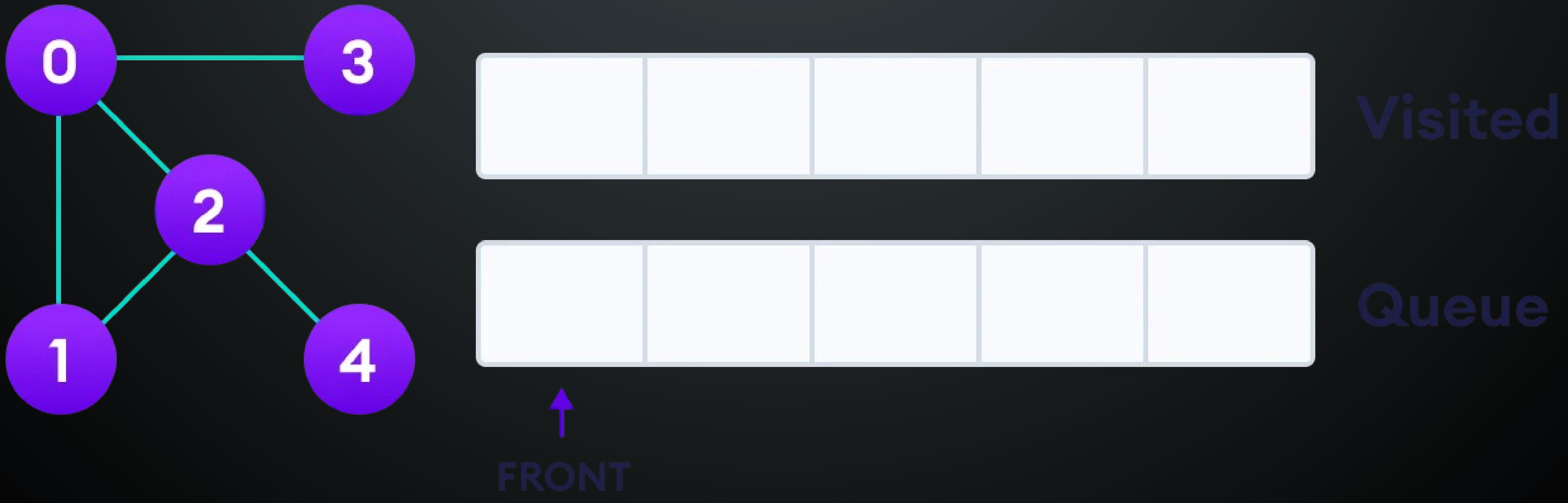
The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can run the BFS algorithm on every node.

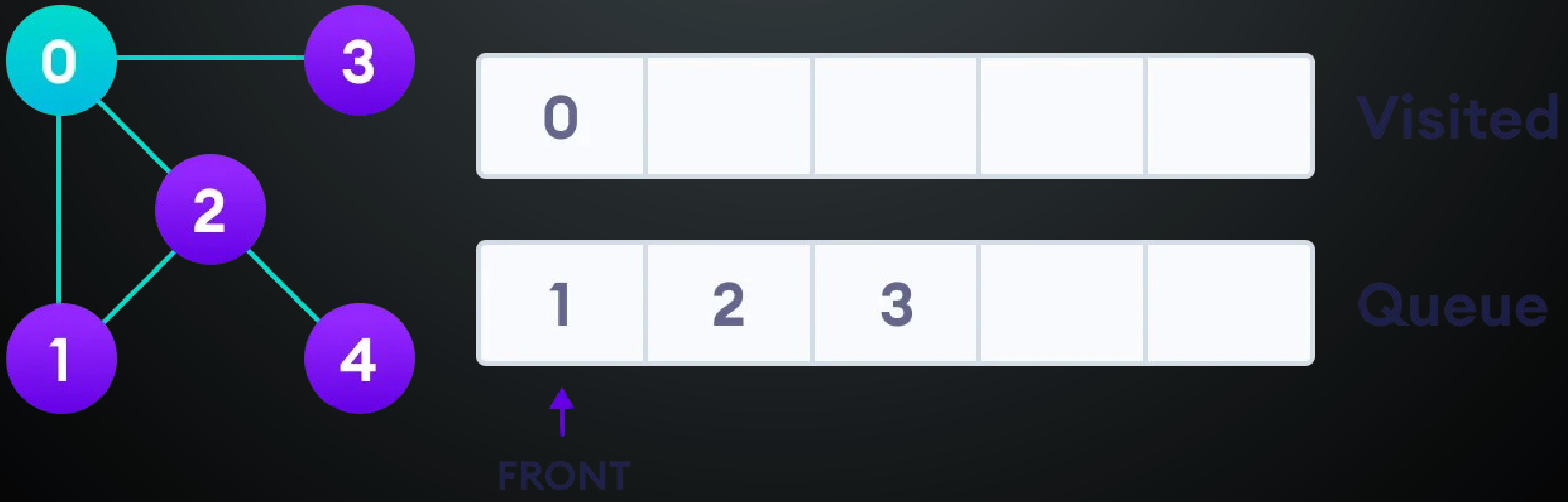
ALGORITHM EXAMPLE

Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



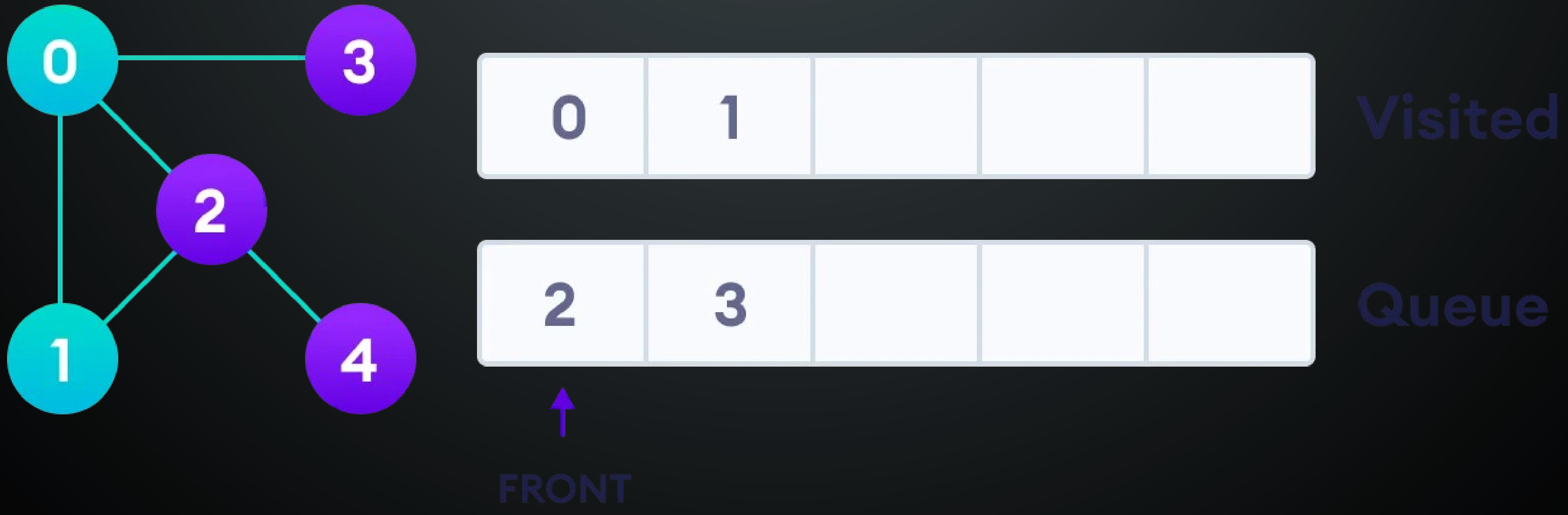
ALGORITHM EXAMPLE

We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



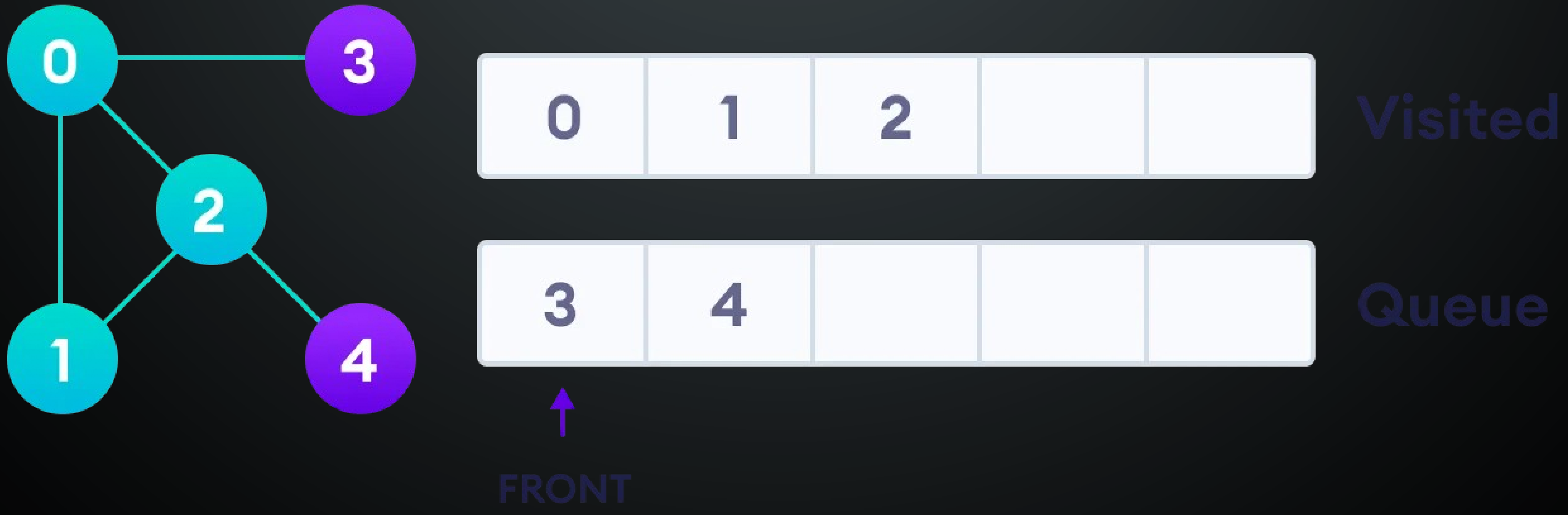
ALGORITHM EXAMPLE

Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.



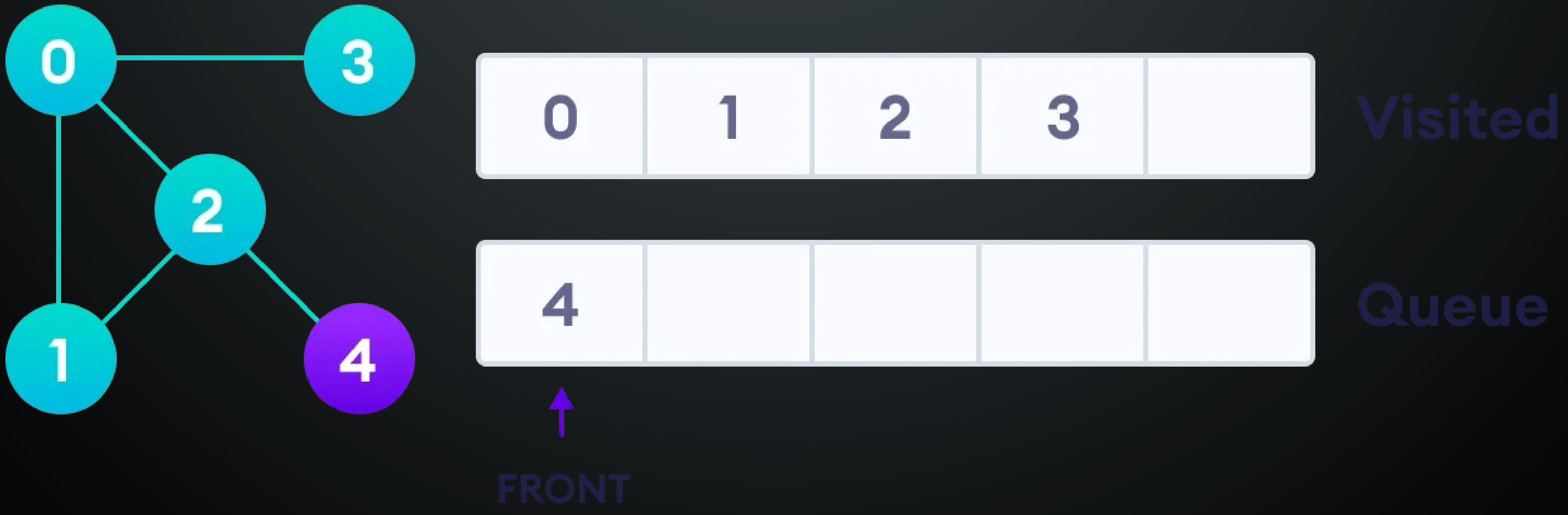
ALGORITHM EXAMPLE

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.



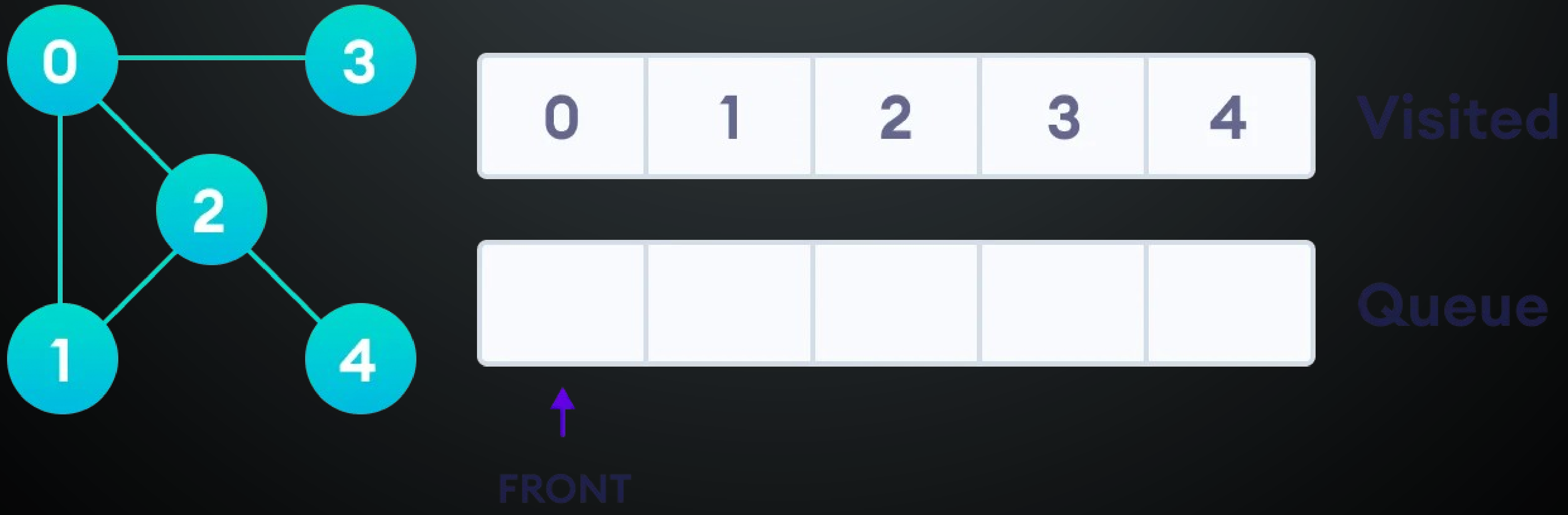
ALGORITHM EXAMPLE

Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.



ALGORITHM EXAMPLE

Since the queue is empty, we have completed the Breadth First Traversal of the graph.



ALGORITHM COMPLEXITY

The time complexity of the BFS algorithm is represented in the form of **O**(**V** + **E**), where **V** is the number of nodes and **E** is the number of edges.

The space complexity of the algorithm is **O**(**V**).

The algorithm pseudocode:

```
create a queue Q
mark v as visited and put v into Q
while Q is non-empty
    remove the head u of Q
    mark and enqueue all (unvisited) neighbours of u
```

ALGORITHM APPLICATIONS

1. To build index by search index
2. For GPS navigation
3. Path finding algorithms
4. In Ford-Fulkerson algorithm to find maximum flow in a network
5. Cycle detection in an undirected graph
6. In minimum spanning tree

PRACTICE: PATH WITH MINIMAL TRANSFERS

Problem. You have **N** cities (1<=**N**<=100) and **K** air races (1<=**K**<=5000) between the cities. You need fly from the city **A** to the city **B** with minimal number of transfers.

Task. Create a program using C/C++/Python to solve this problem.

Input. Two strings: the fist one contains 4 numbers **N**, **K**, **A**, and **B** divided by spaces; the second one contains **K** pairs of numbers – indexes of connected cities divided by spaces.

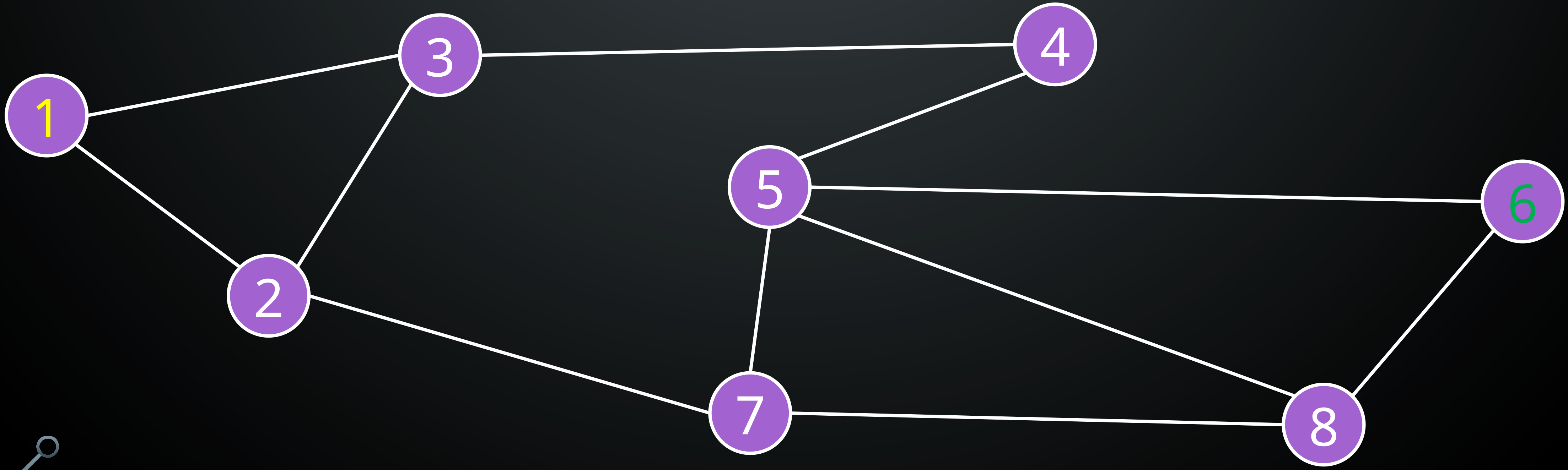
Output. Transfers count for the shortest path or number **-1** if path doesn't exist.

PRACTICE: PATH WITH MINIMAL TRANSFERS

Example. We have **N**=8 and **K**=10, **A**=1 and **B**=6. Input:

8 11 1 6

1 3 1 2 4 3 2 3 4 5 6 5 7 5 8 5 2 7 7 8 8 6

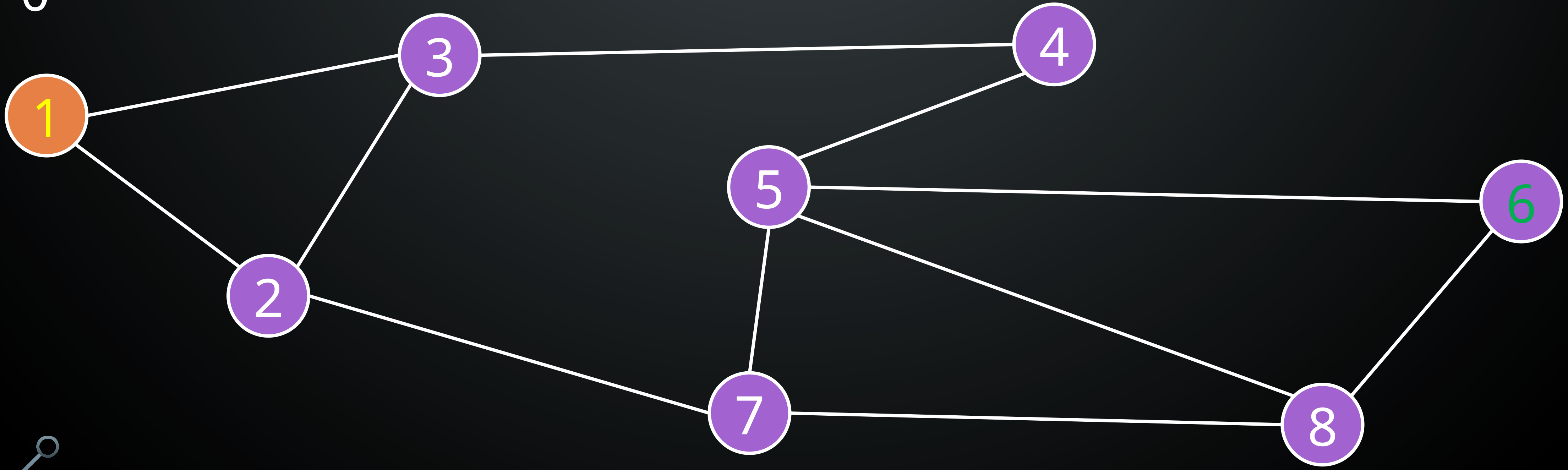


PRACTICE: PATH WITH MINIMAL TRANSFERS

Step 1. Push node **1** to queue and mark it as visited.

Minimal transfers: 0

1								Visited
1								Queue

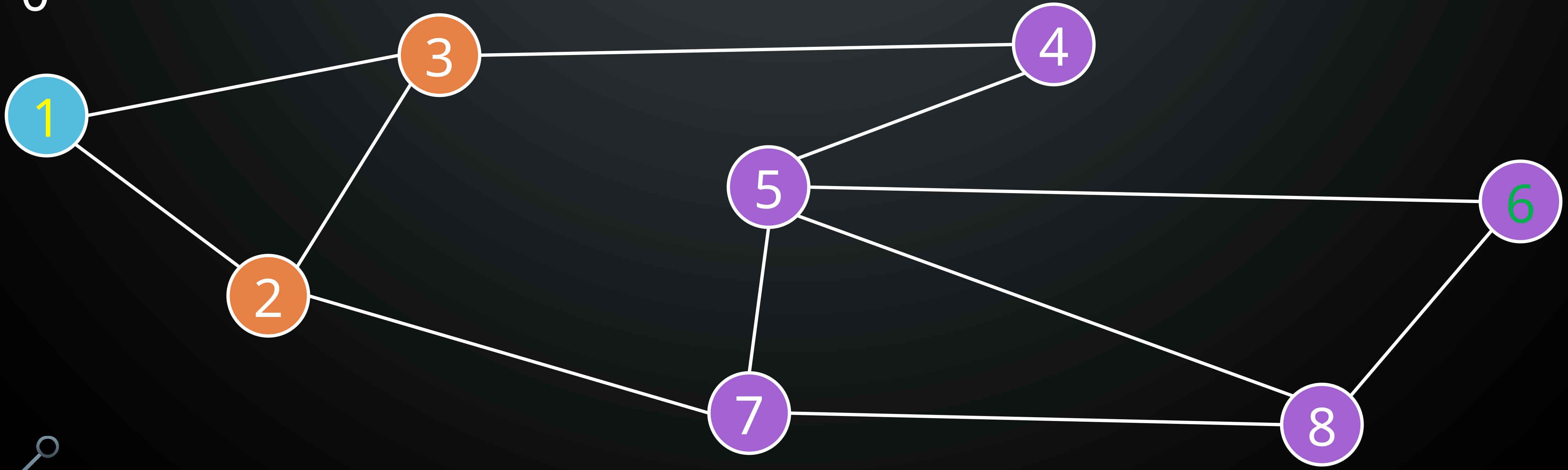


PRACTICE: PATH WITH MINIMAL TRANSFERS

Step 2. Pop node **1**, push it neighbor to queue and mark them as visited.

Minimal transfers: 0

1	2	3						Visited
	2	3						Queue

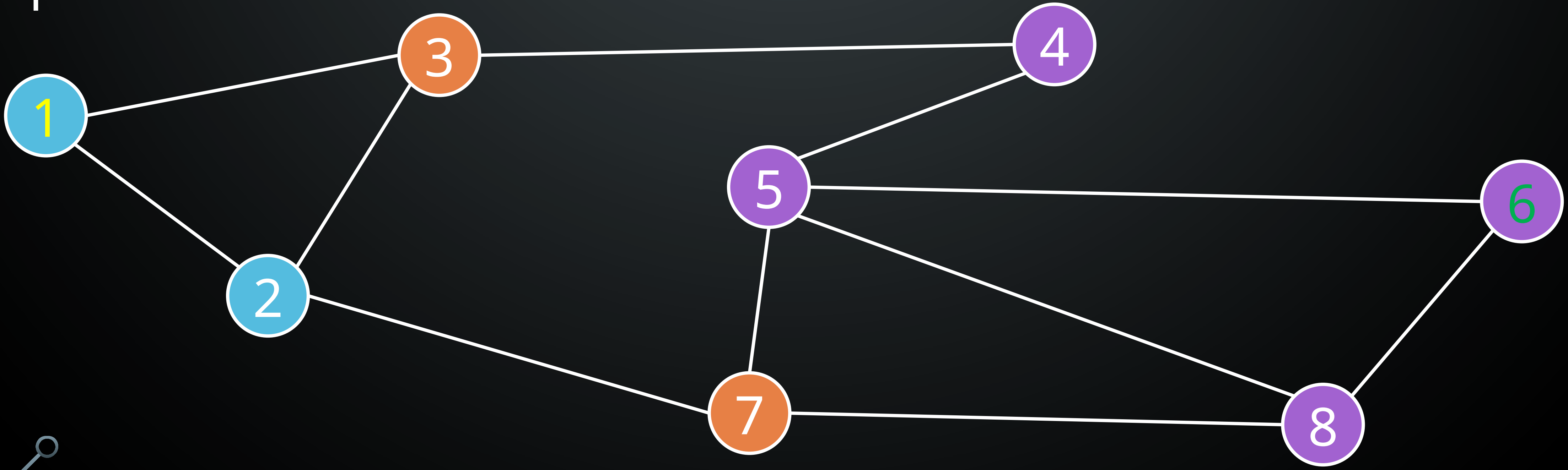


PRACTICE: PATH WITH MINIMAL TRANSFERS

Step 3. Pop node **2**, push it neighbors to queue and mark them as visited.

Minimal transfers: 1

1	2	3				7		Visited
		3	7					Queue

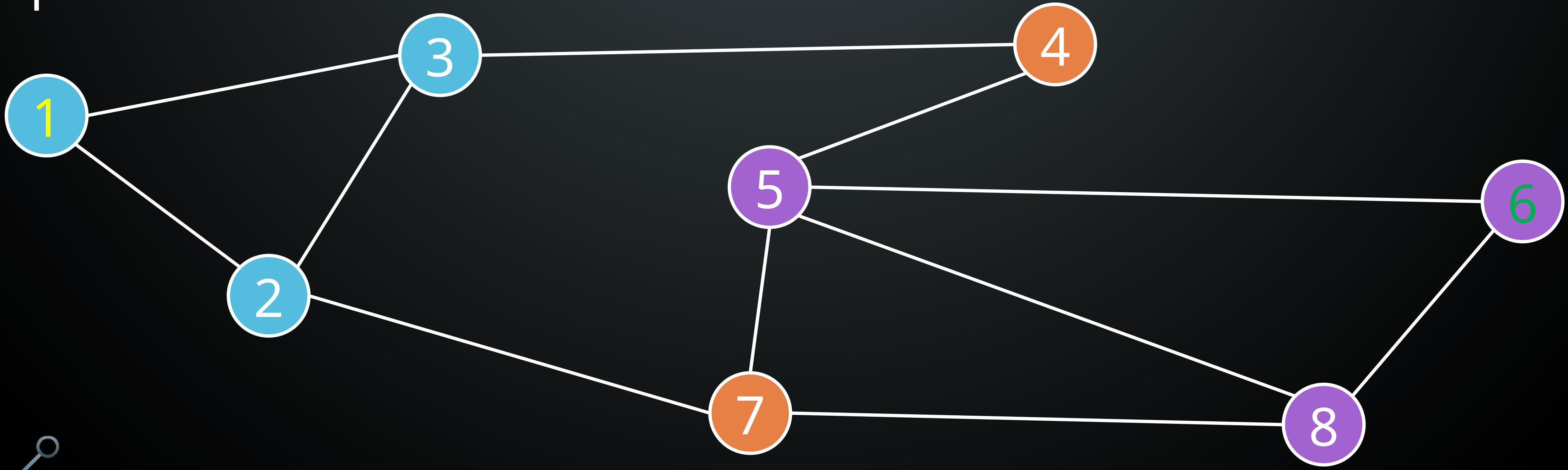


PRACTICE: PATH WITH MINIMAL TRANSFERS

Step 4. Pop node **3**, push it neighbors to queue and mark them as visited.

Minimal transfers: 1

1	2	3	4			7		Visited
			7	4				Queue

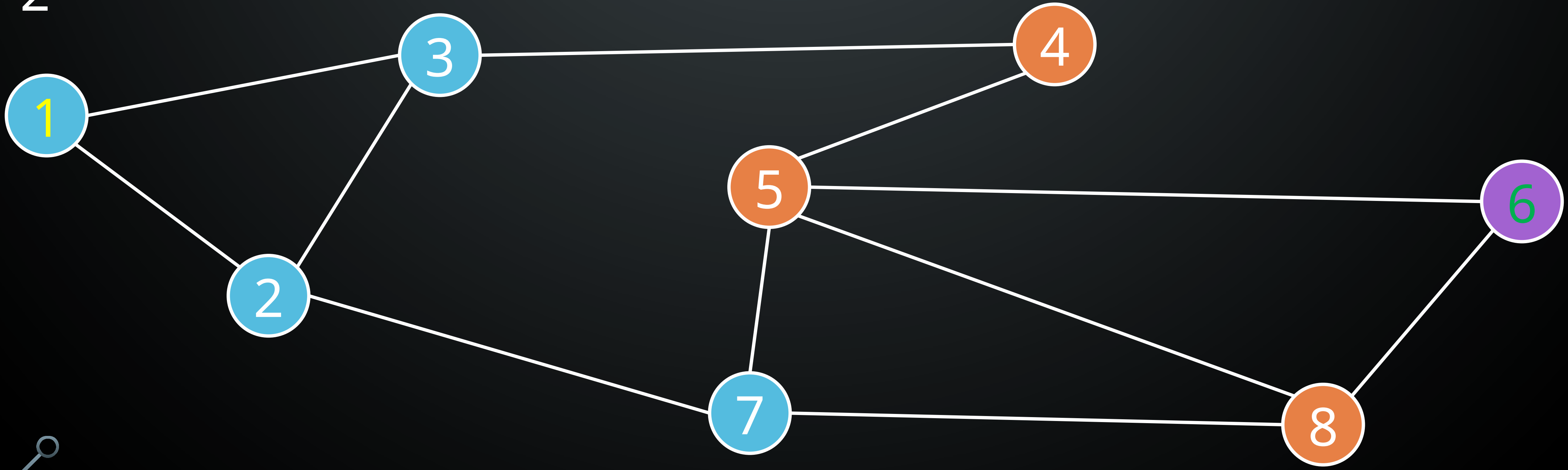


PRACTICE: PATH WITH MINIMAL TRANSFERS

Step 5. Pop node **7**, push it neighbors to queue and mark them as visited.

Minimal transfers: 2

1	2	3	4	5		7	8	Visited
				4	5	8		Queue

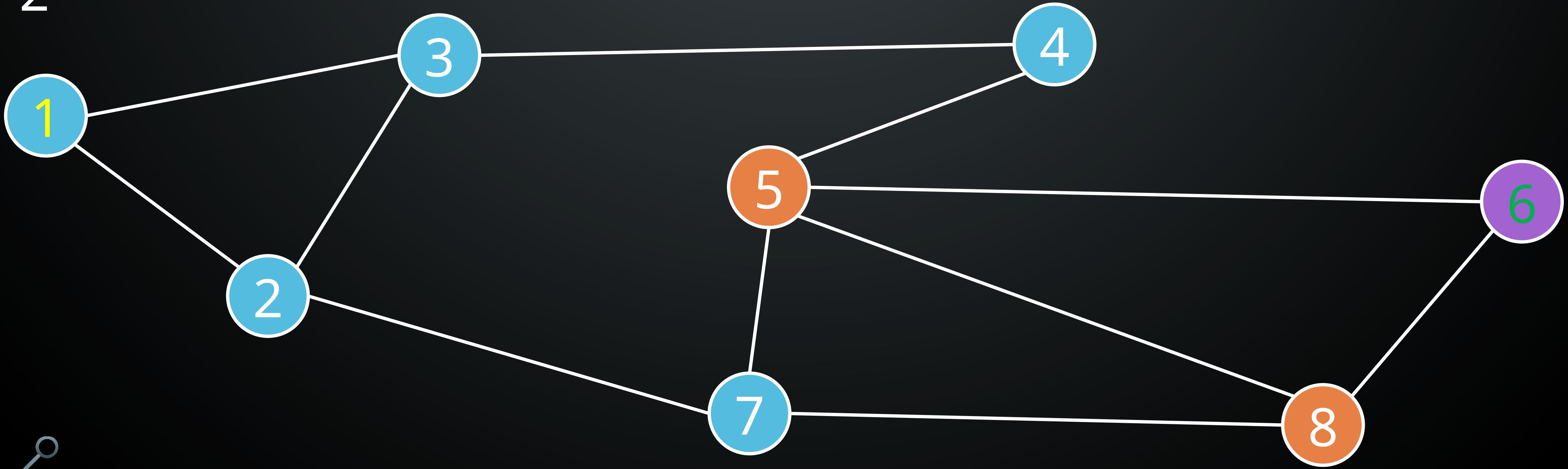


PRACTICE: PATH WITH MINIMAL TRANSFERS

Step 6. Pop node **4**. All it neighbor already visited so nothing to push.

Minimal transfers: 2

1	2	3	4	5		7	8	Visited
					5	8		Queue

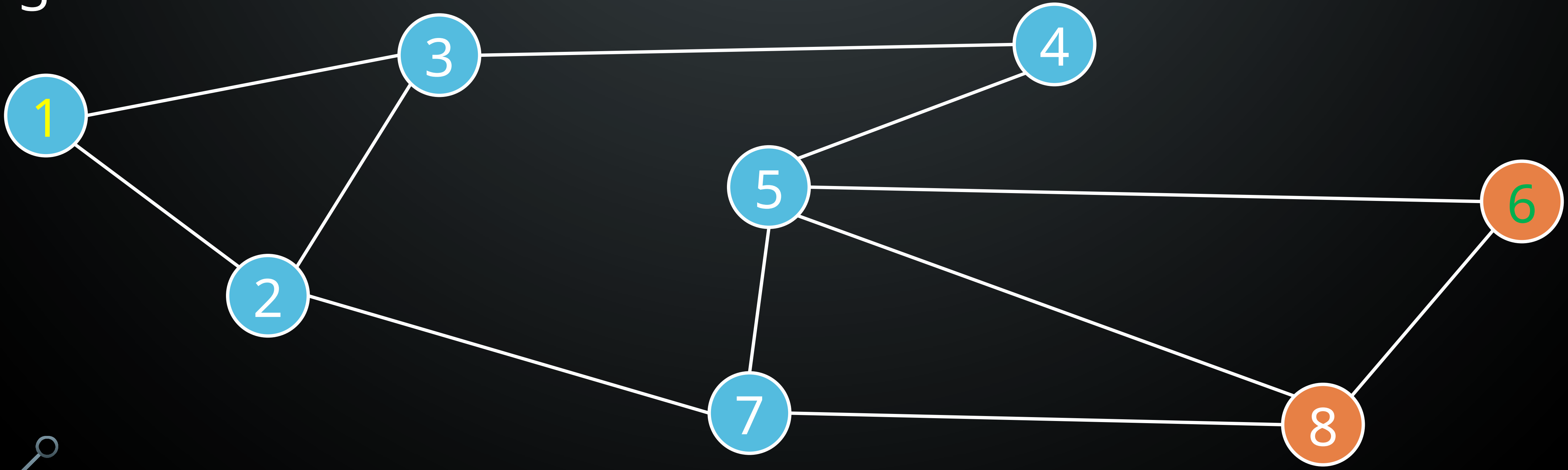


PRACTICE: PATH WITH MINIMAL TRANSFERS

Step 7. Pop node **5**. The destination node **6** is neighbor of **5**, so se finish.

Minimal transfers: 3

1	2	3	4	5	6	7	8	Visited
						8	6	Queue



PRACTICE: PATH WITH MINIMAL TRANSFERS

Code example:

```
#include <iostream>
#define MAX 101
int N, K, A, B, R, D[MAX], V[MAX], M[MAX][MAX];
int main()
{
    cin >> N >> K >> A >> B;
    int f, s;
    for(int i = 0; i < K; i++)
    {
        cin >> f >> s;
        M[f][s] = 1;
        M[s][f] = 1;
    }
    // Your code here
    cout << R;
    return 0;
}
```

THANK YOU!