**LAB # 05**

**ACTIVITY 1**Consider the following graph. If there is ever a decision between multiple neighbor nodes in the

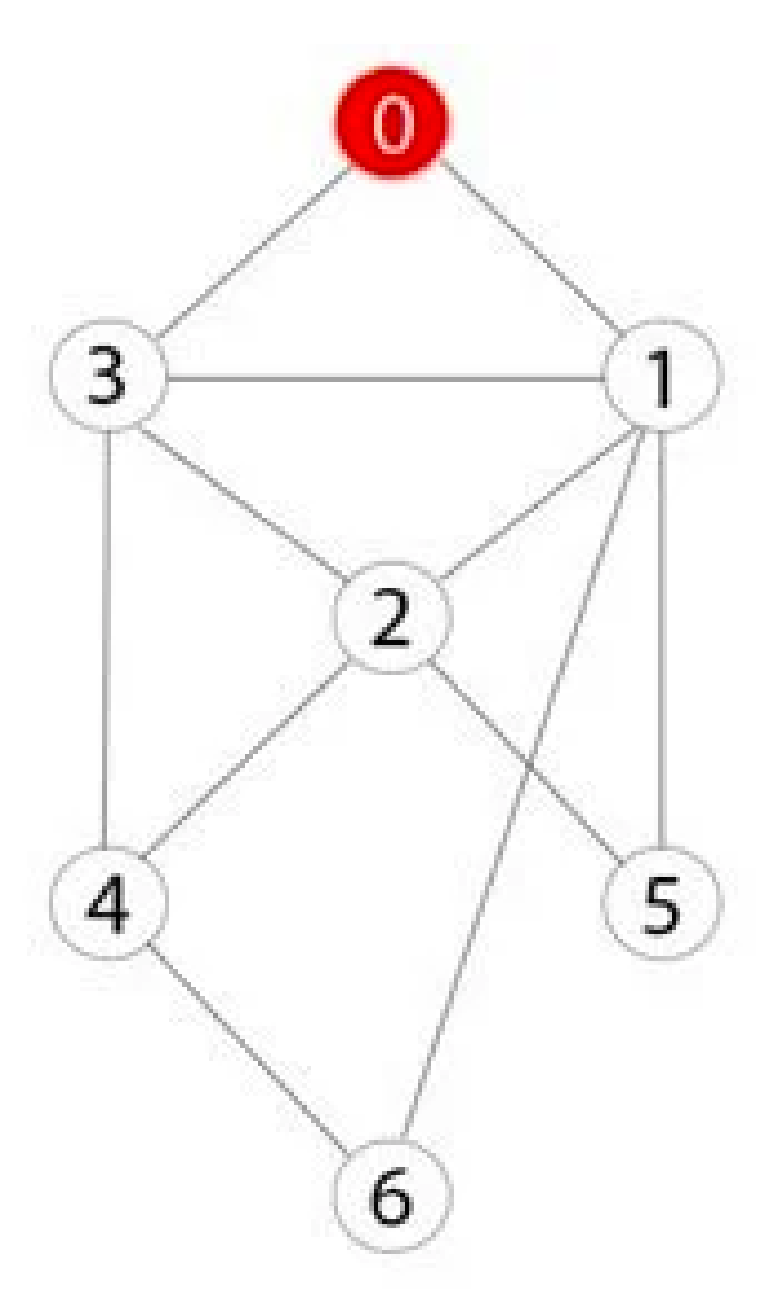
BFS algorithm, assume we always choose the letter closest to the beginning of the alphabet first.

A connected graph with 7 nodes and 7 edges. The edges are undirected and unweight. Distance

between two nodes will be measured based on the number of edges separating two vertices.

Represent a graph with adjacency list using dictionaries. The keys of the dictionary represent

nodes; the values have a list of neighbors.



Define function name ‗connected\_component‘, this function keep track of all the visited nodes

with BFS, is as simple as implementing the steps of the algorithm and assign ‗queue‘ variable

already has a node to be checked, i.e., the starting vertex that is used as an entry point to explore

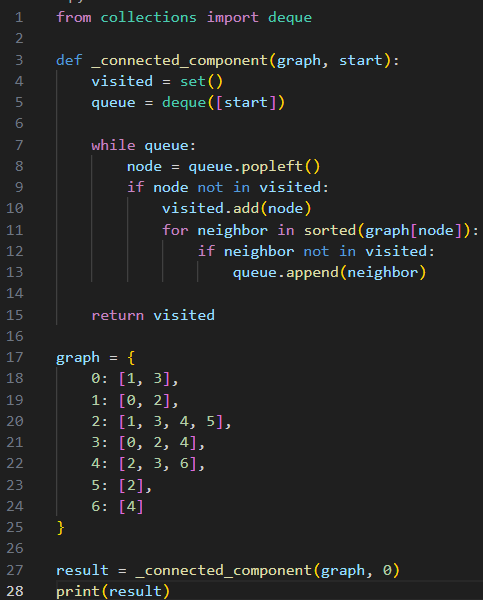
the graph. The next step is to implement a loop that keeps cycling until queue is empty. At each

iteration of the loop, a node is checked. If this wasn‘t visited already, its neighbors are added to

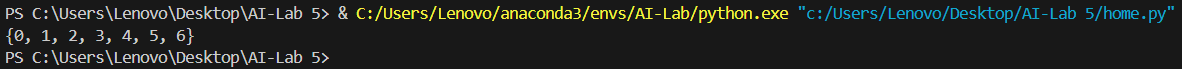
queue. Once the loop is exited, the function (connected\_component) returns all of the visited

nodes.

**INPUT**

****

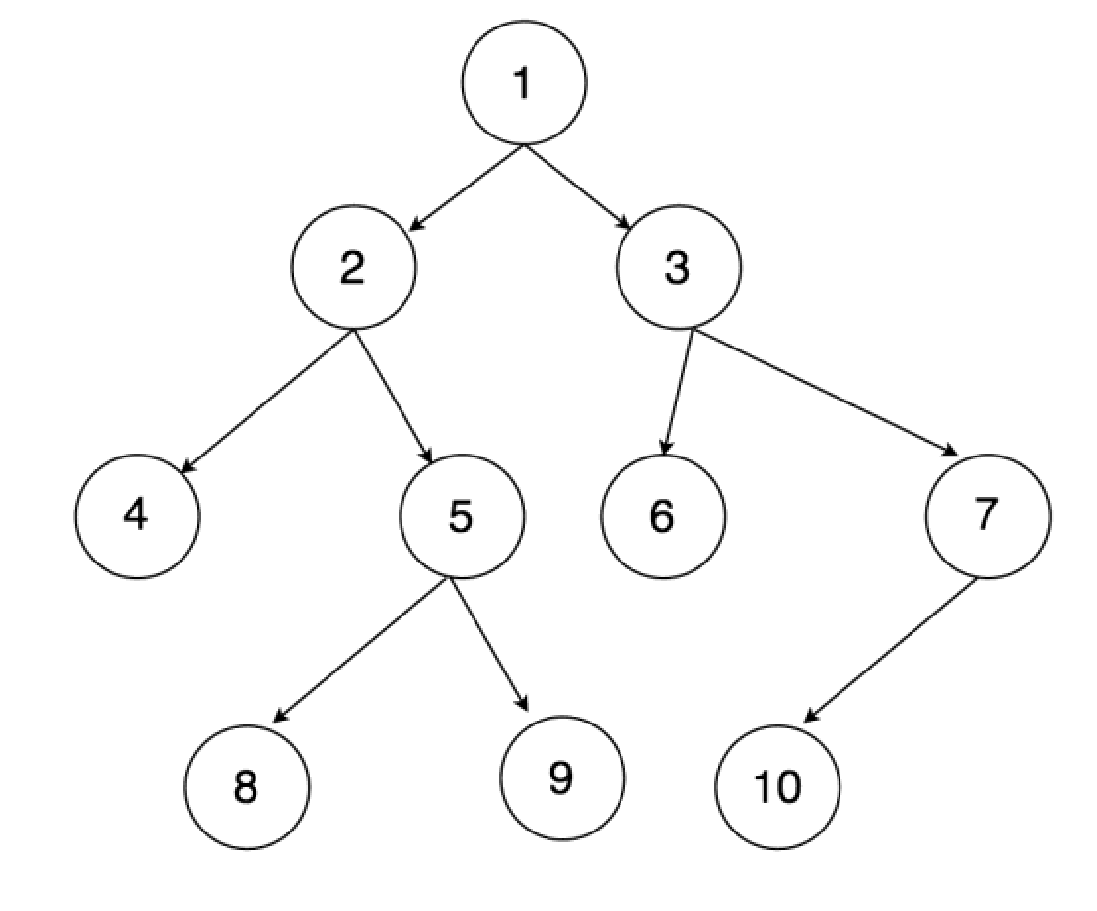
**OUTPUT**

****

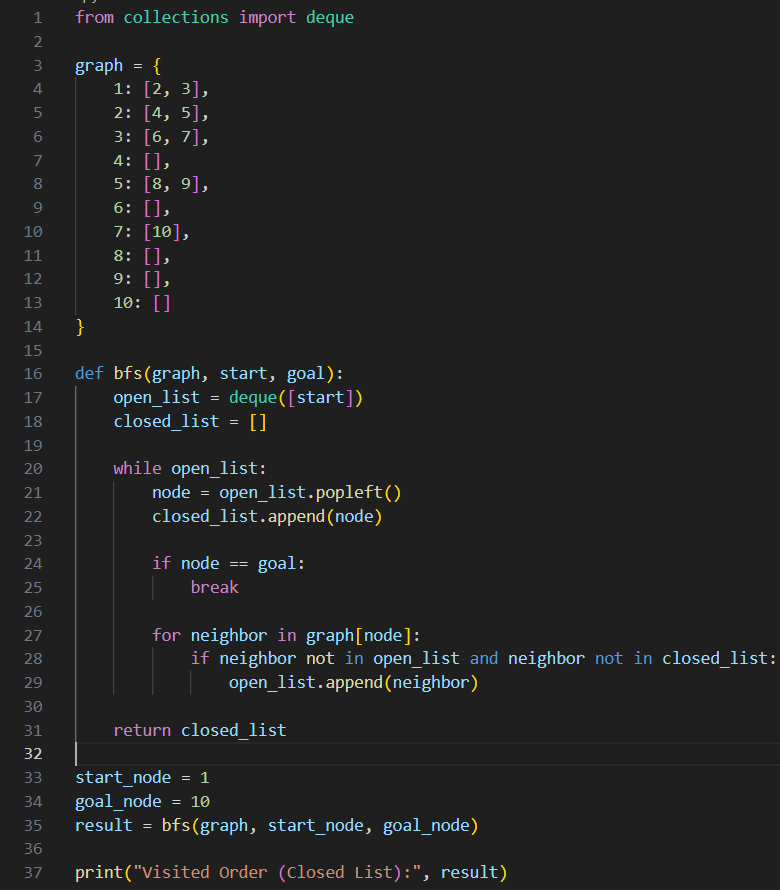
**ACTIVITY 2**

Apply Breadth First Search on following graph considering the initial state is 1 and final state is

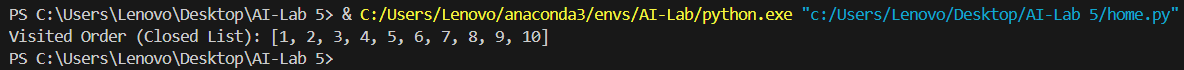
10. Show results in form of open and closed list. Also evaluate it manually.



**INPUT**

****

**OUTPUT**

****

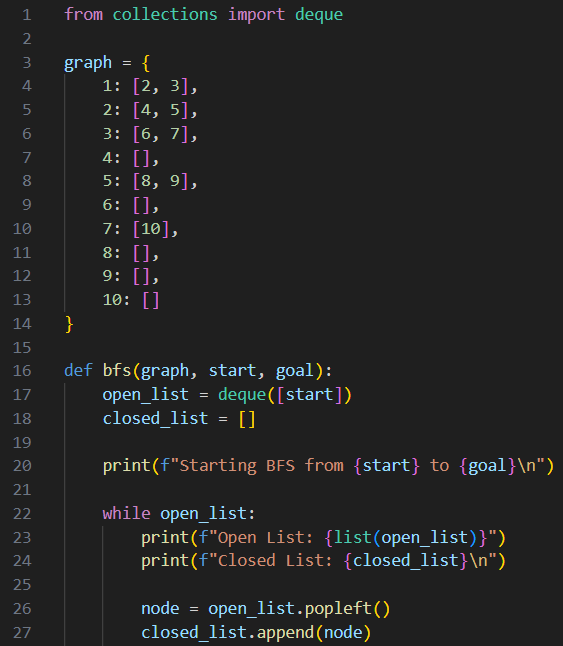
**MANUAL EVALUATION**

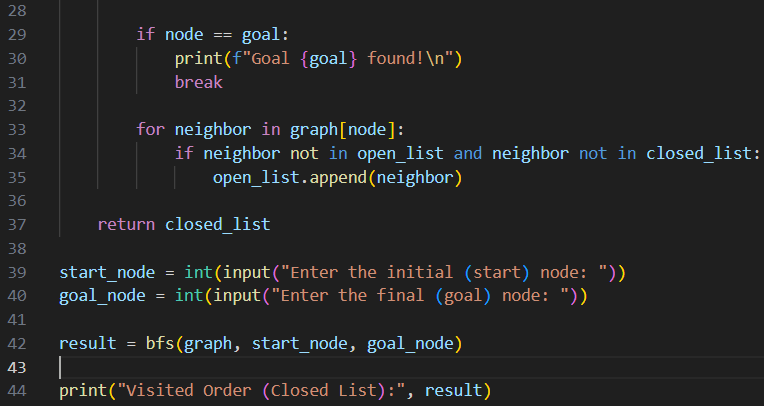
| **Step** | **Open** | **Closed** | **Action** |
| --- | --- | --- | --- |
| 0 | [1] | [] | Start from 1 |
| 1 | [2, 3] | [1] | Expand 1, add neighbors 2, 3 |
| 2 | [3, 4, 5] | [1, 2] | Expand 2, add neighbors 4, 5 |
| 3 | [4, 5, 6, 7] | [1, 2, 3] | Expand 3, add neighbors 6, 7 |
| 4 | [5, 6, 7] | [1, 2, 3, 4] | Expand 4, no neighbors |
| 5 | [6, 7, 8, 9] | [1, 2, 3, 4, 5] | Expand 5, add neighbors 8, 9 |
| 6 | [7, 8, 9] | [1, 2, 3, 4, 5, 6] | Expand 6, no neighbors |
| 7 | [8, 9, 10] | [1, 2, 3, 4, 5, 6, 7] | Expand 7, add neighbor 10 |
| 8 | [9, 10] | [1, 2, 3, 4, 5, 6, 7, 8] | Expand 8, no neighbors |
| 9 | [10] | [1, 2, 3, 4, 5, 6, 7, 8, 9] | Expand 9, no neighbors |
| 10 | [] | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | Goal found at 10! |

**ACTIVITY 3**

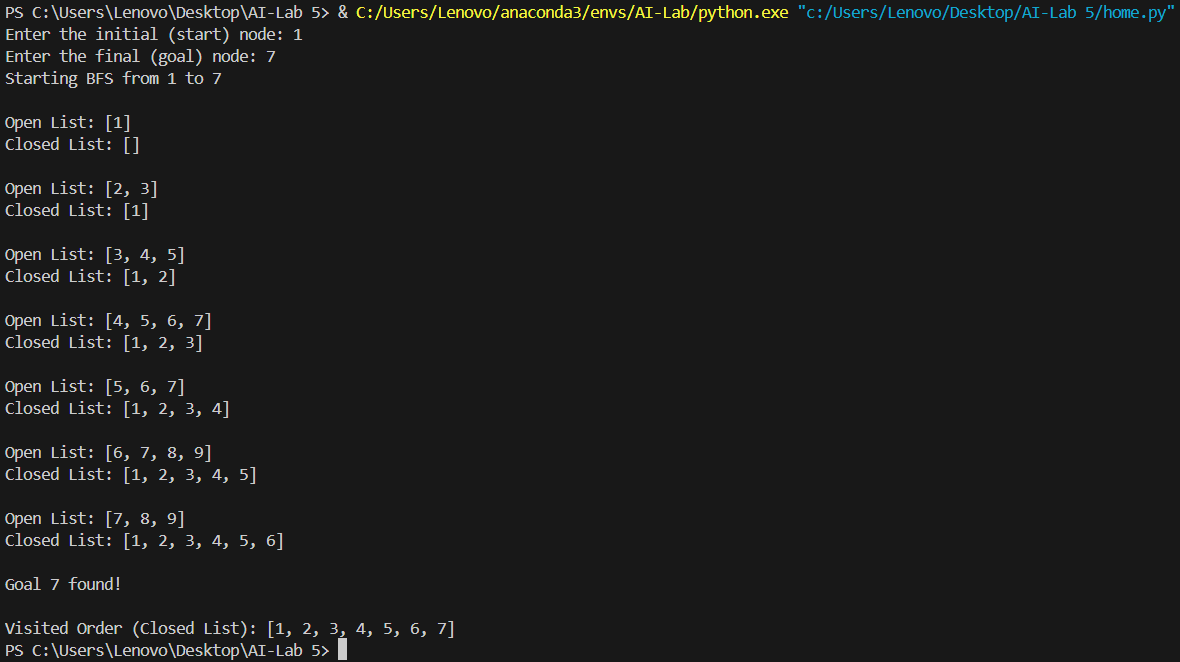
Repeat Activity-2, apply BFS by taking initial and final state as user input. Show results in form of open and closed list. Also evaluate it manually.

**INPUT**





**OUTPUT**

****

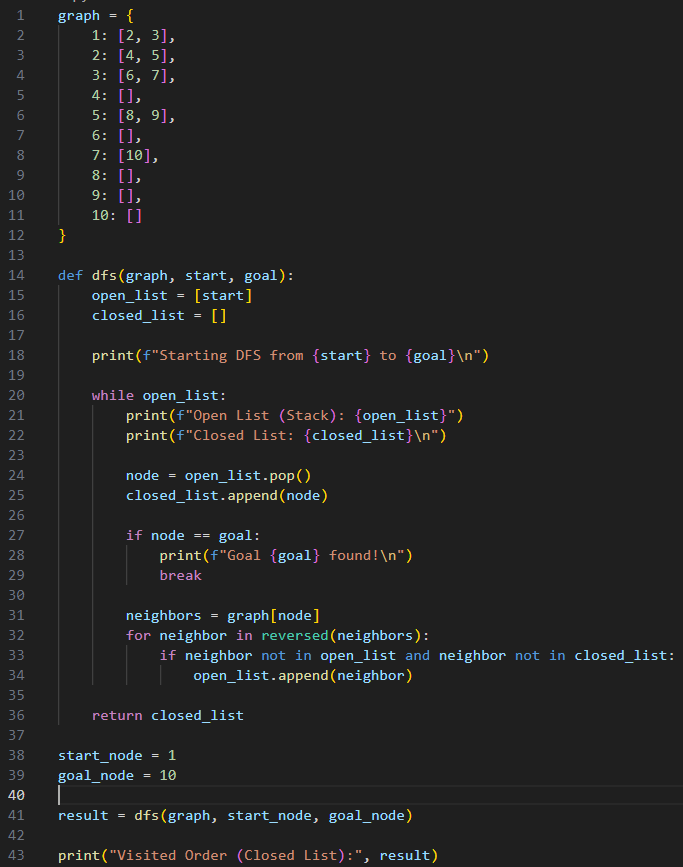
**MANUAL EVALUATION**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Step** | **Open List** | **Closed List** | **Current Node** | **Action** |
| 0 | [1] | [] | 1 | Remove 1, add its neighbors [2, 3] |
| 1 | [2, 3] | [1] | 2 | Remove 2, add its neighbors [4, 5] |
| 2 | [3, 4, 5] | [1, 2] | 3 | Remove 3, add its neighbors [6, 7] |
| 3 | [4, 5, 6, 7] | [1, 2, 3] | 4 | Remove 4, no neighbors to add |
| 4 | [5, 6, 7] | [1, 2, 3, 4] | 5 | Remove 5, add its neighbors [8, 9] |
| 5 | [6, 7, 8, 9] | [1, 2, 3, 4, 5] | 6 | Remove 6, no neighbors to add |
| 6 | [7, 8, 9] | [1, 2, 3, 4, 5, 6] | 7 | Remove 7, **Goal Found!** |

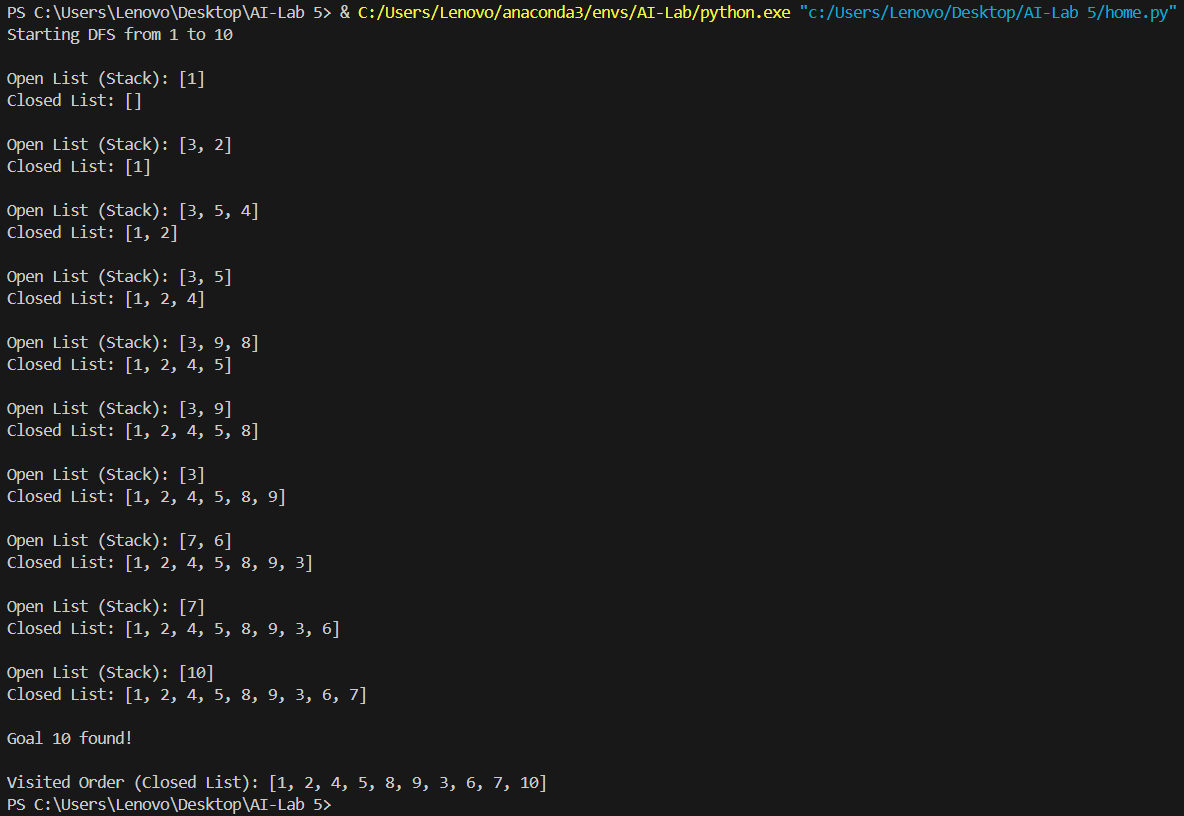
**ACTIVITY 4**

Apply Depth First Search on Activity-2 graph considering the initial state is 1 and final state is 10. Show results in form of open and closed list. Also evaluate it manually.

**INPUT**

****

**OUTPUT**

****

**MANUAL EVALUATION**

| **Step** | **Open List (Stack)** | **Closed List** | **Current Node** | **Action** |
| --- | --- | --- | --- | --- |
| 0 | [1] | [] | 1 | Pop 1, push 3 and 2 |
| 1 | [3, 2] | [1] | 2 | Pop 2, push 5 and 4 |
| 2 | [3, 5, 4] | [1, 2] | 4 | Pop 4, no neighbors |
| 3 | [3, 5] | [1, 2, 4] | 5 | Pop 5, push 9 and 8 |
| 4 | [3, 9, 8] | [1, 2, 4, 5] | 8 | Pop 8, no neighbors |
| 5 | [3, 9] | [1, 2, 4, 5, 8] | 9 | Pop 9, no neighbors |
| 6 | [3] | [1, 2, 4, 5, 8, 9] | 3 | Pop 3, push 7 and 6 |
| 7 | [7, 6] | [1, 2, 4, 5, 8, 9, 3] | 6 | Pop 6, no neighbors |
| 8 | [7] | [1, 2, 4, 5, 8, 9, 3, 6] | 7 | Pop 7, push 10 |
| 9 | [10] | [1, 2, 4, 5, 8, 9, 3, 6, 7] | 10 | Pop 10, **Goal found!** |

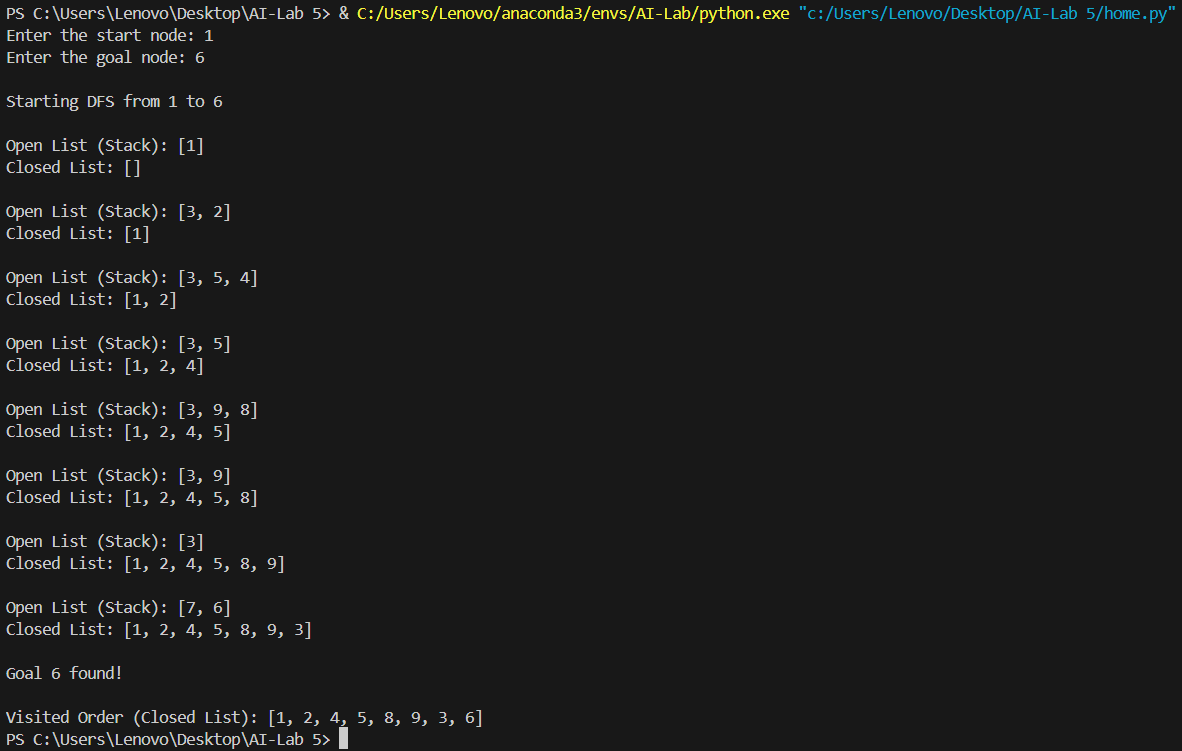
**ACTIVITY 5**

Repeat Activity-4, apply DFS by taking initial and final state as user input. Show results in form of open and closed list. Also evaluate it manually.

**INPUT**

****

**OUTPUT**

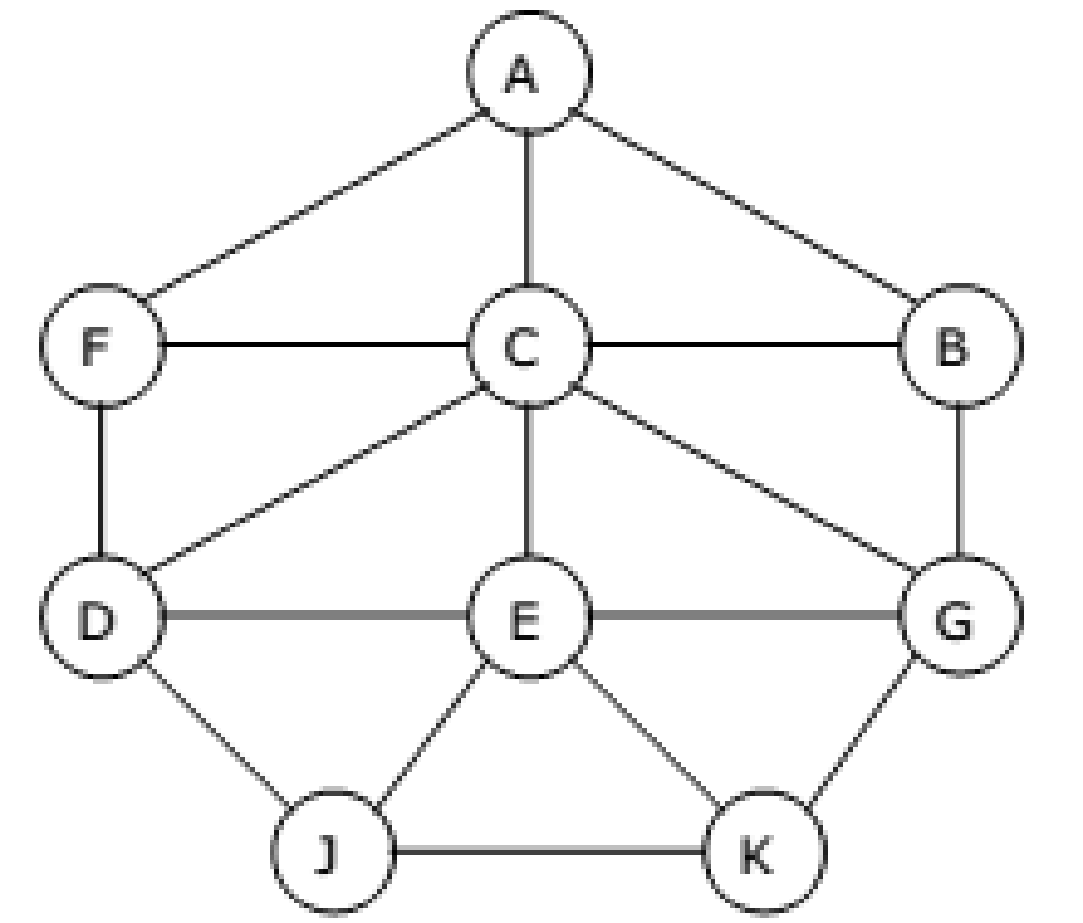
****

**MANUAL EVALUATION**

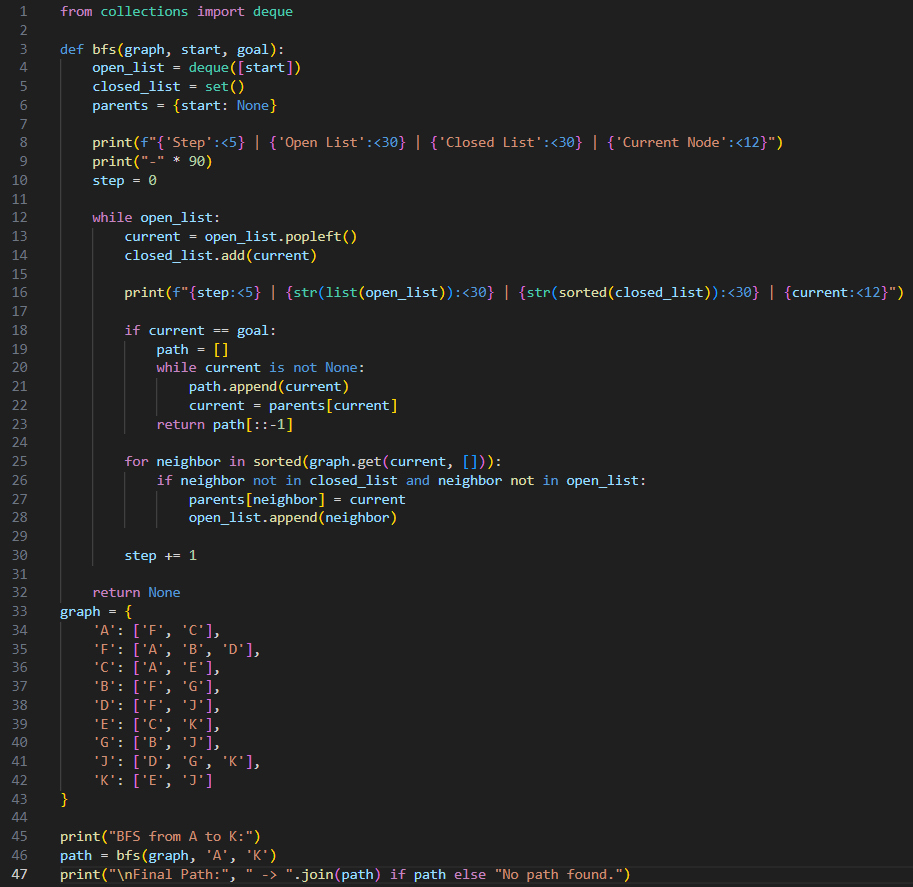
| **Step** | **Open List (Stack)** | **Closed List** | **Current Node** | **Action** |
| --- | --- | --- | --- | --- |
| 0 | [1] | [] | 1 | Pop 1, push 3 and 2 |
| 1 | [3, 2] | [1] | 2 | Pop 2, push 5 and 4 |
| 2 | [3, 5, 4] | [1, 2] | 4 | Pop 4, no neighbors |
| 3 | [3, 5] | [1, 2, 4] | 5 | Pop 5, push 9 and 8 |
| 4 | [3, 9, 8] | [1, 2, 4, 5] | 8 | Pop 8, no neighbors |
| 5 | [3, 9] | [1, 2, 4, 5, 8] | 9 | Pop 9, no neighbors |
| 6 | [3] | [1, 2, 4, 5, 8, 9] | 3 | Pop 3, push 7 and 6 |
| 7 | [7, 6] | [1, 2, 4, 5, 8, 9, 3] | 6 | Pop 6, **Goal Found!** |

**ACTIVITY 6**

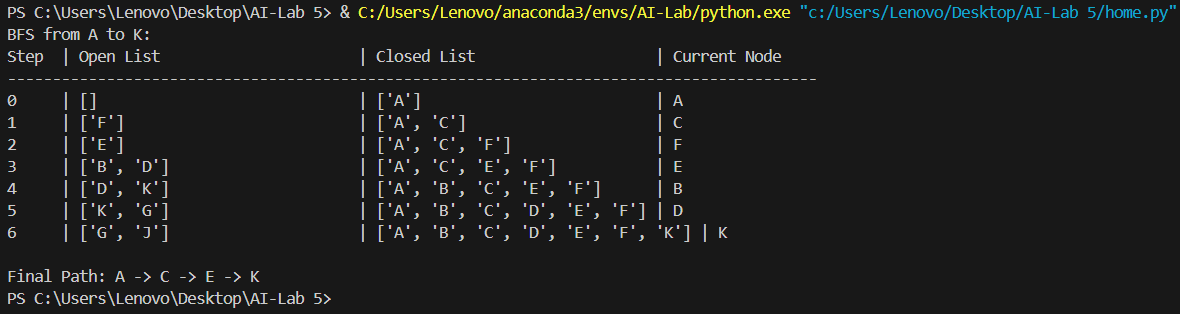
Represent a graph with adjacency list using dictionaries. The keys of the dictionary represent nodes; the values have a list of neighbors. Apply Breadth First Search on following graph considering the initial state is A and final state is K. Also evaluate it manually.



**INPUT**

****

**OUPUT**

****

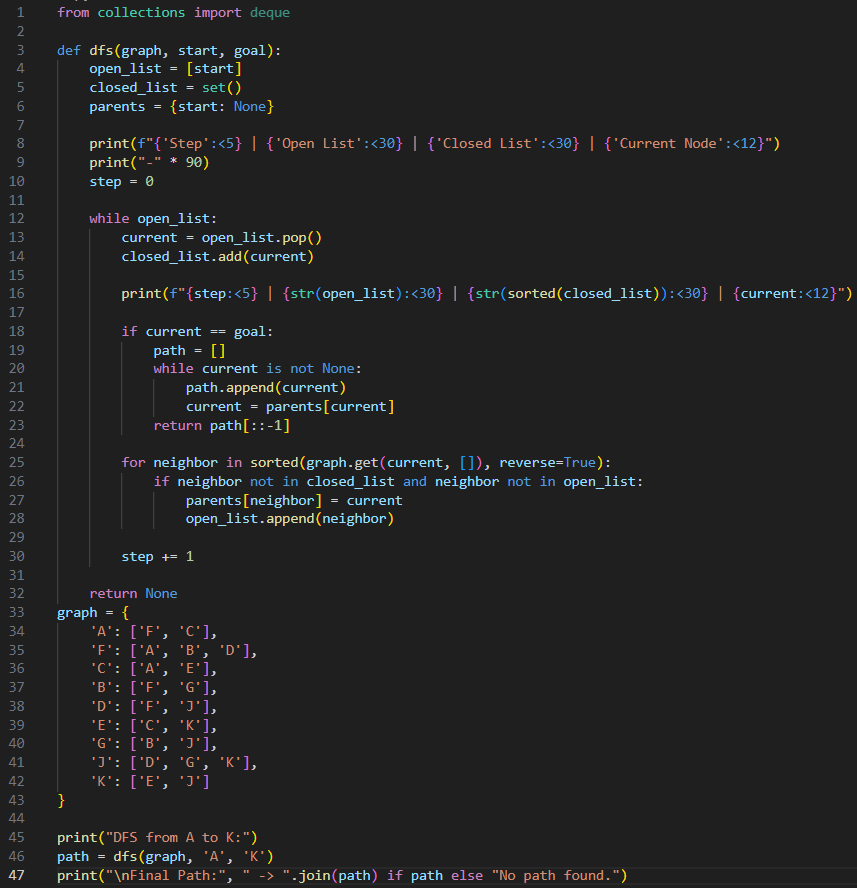
**MANUAL EVALUATION**

| **Step** | **Open List** | **Closed List** | **Current Node** |
| --- | --- | --- | --- |
| 0 | [F, C] | [A] | A |
| 1 | [C, B, D] | [A, F] | F |
| 2 | [B, D, E] | [A, C, F] | C |
| 3 | [D, E, G] | [A, B, C, F] | B |
| 4 | [E, G, J] | [A, B, C, D, F] | D |
| 5 | [G, J, K] | [A, B, C, D, E, F] | E |
| 6 | [J, K] | [A, B, C, D, E, F, G] | G |
| 7 | [K] | [A, B, C, D, E, F, G, J] | J |
| 8 | [] | [A, B, C, D, E, F, G, J, K] | K |

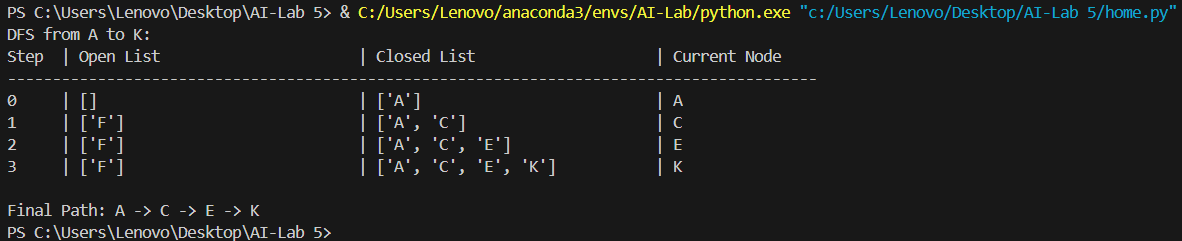
**ACTIVITY 7**

Represent a graph with adjacency list using dictionaries. The keys of the dictionary represent nodes; the values have a list of neighbors. Apply Depth First Search on Activity-6 graph considering the initial state is A and final state is K. Also evaluate it manually.

**INPUT**

****

**OUTPUT**

****

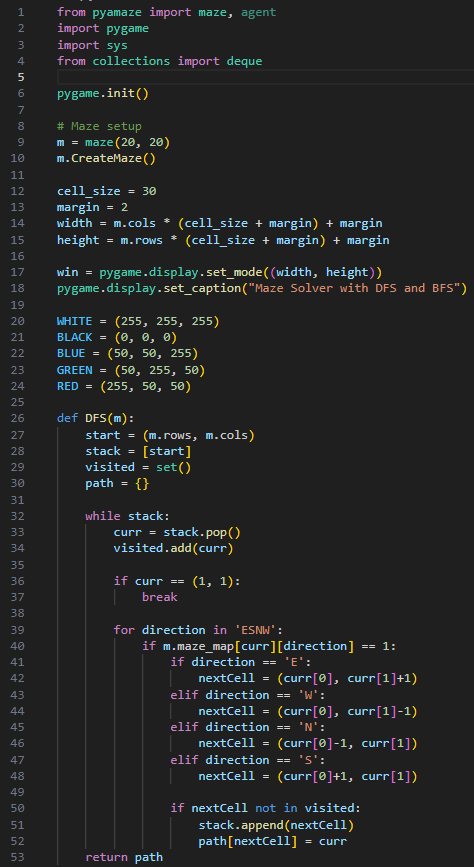
**MANUAL EVALUATION**

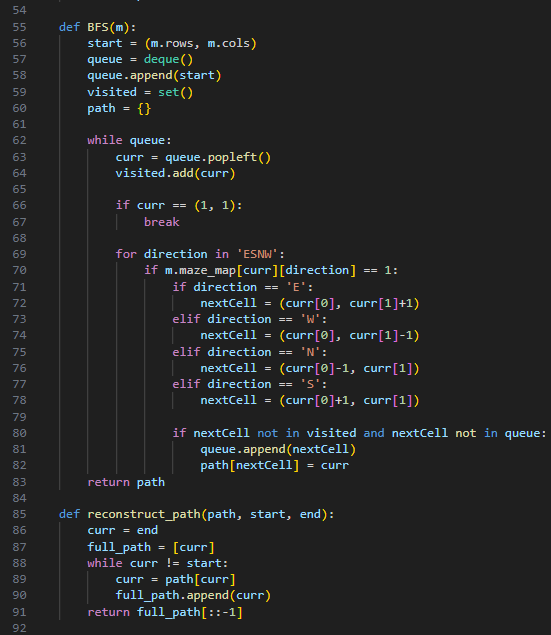
| **Step** | **Open List** | **Closed List** | **Current Node** |
| --- | --- | --- | --- |
| 0 | ['F', 'C'] | ['A'] | A |
| 1 | ['F', 'E'] | ['A', 'C'] | C |
| 2 | ['F', 'K'] | ['A', 'C', 'E'] | E |
| 3 | ['F'] | ['A', 'C', 'E', 'K'] | K |

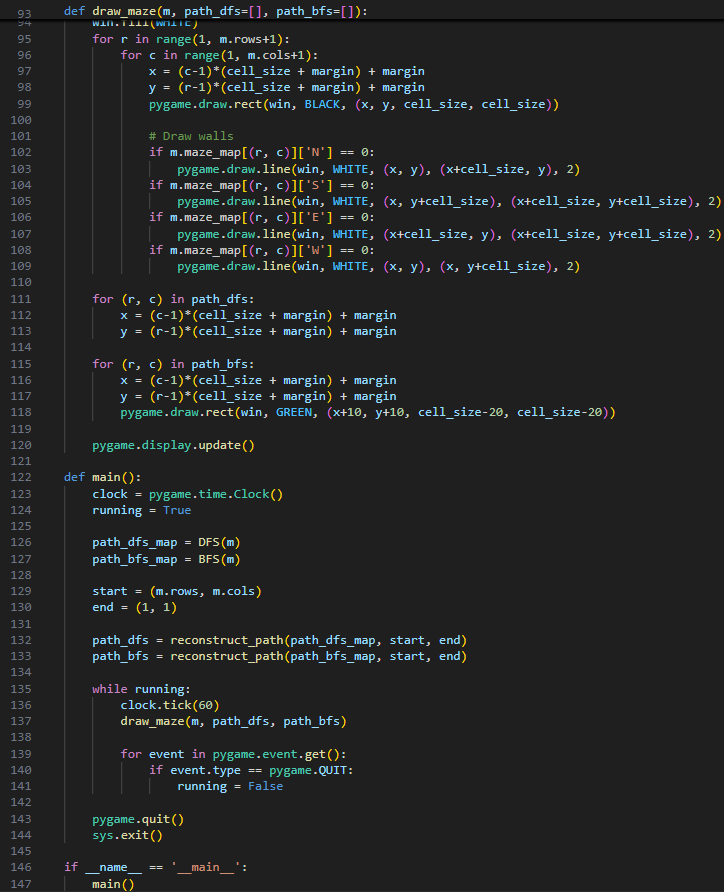
**ACTIVITY 8**

You need to implement DFS and BFS both using “Pygame” and “Pyamaze” library, generate the maze of (20,20) and find the path.

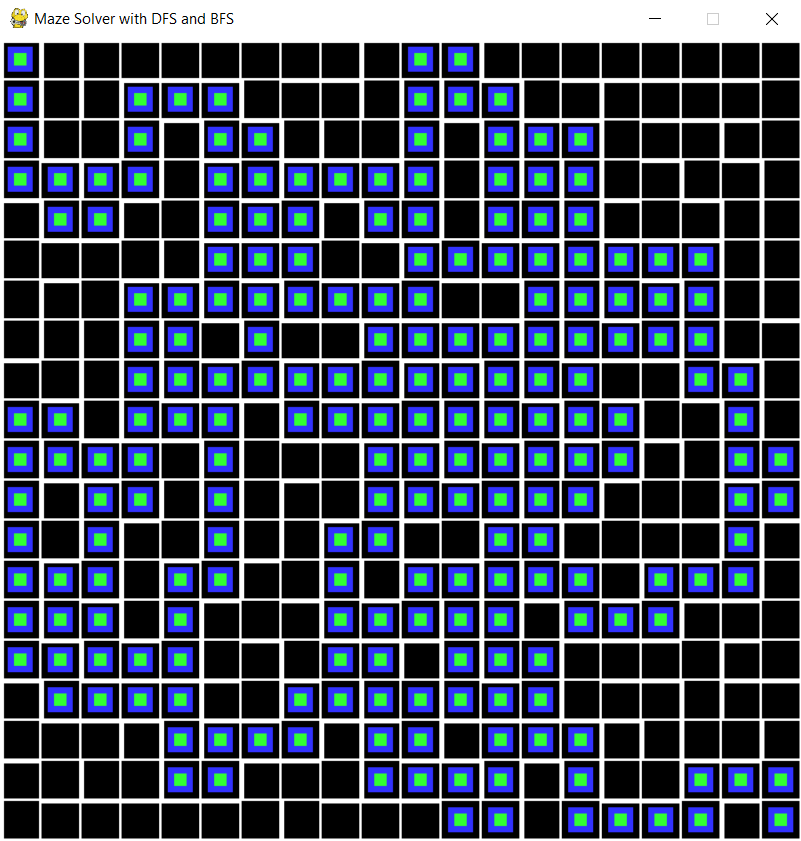
**INPUT**

****

****

****

**OUTPUT**

****