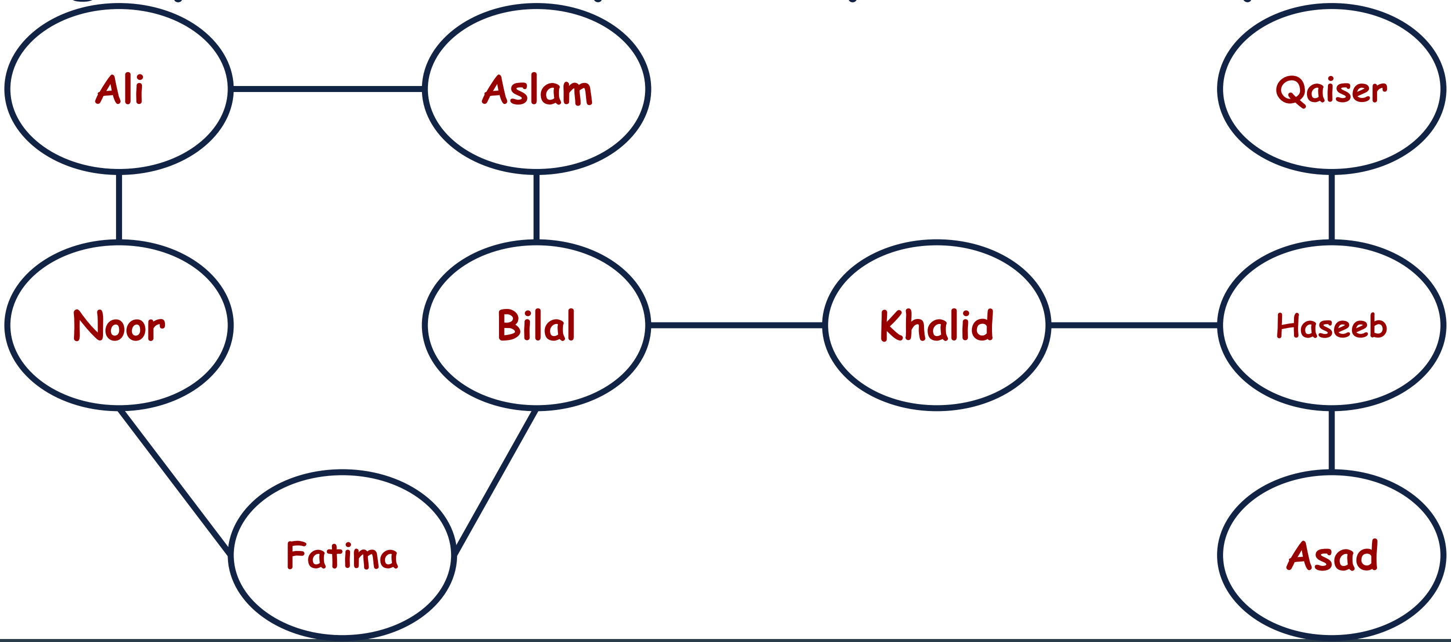
Graphs

Graphs: Review

Previously, we had seen graphs and the way to store these graphs efficiently in computer memory.

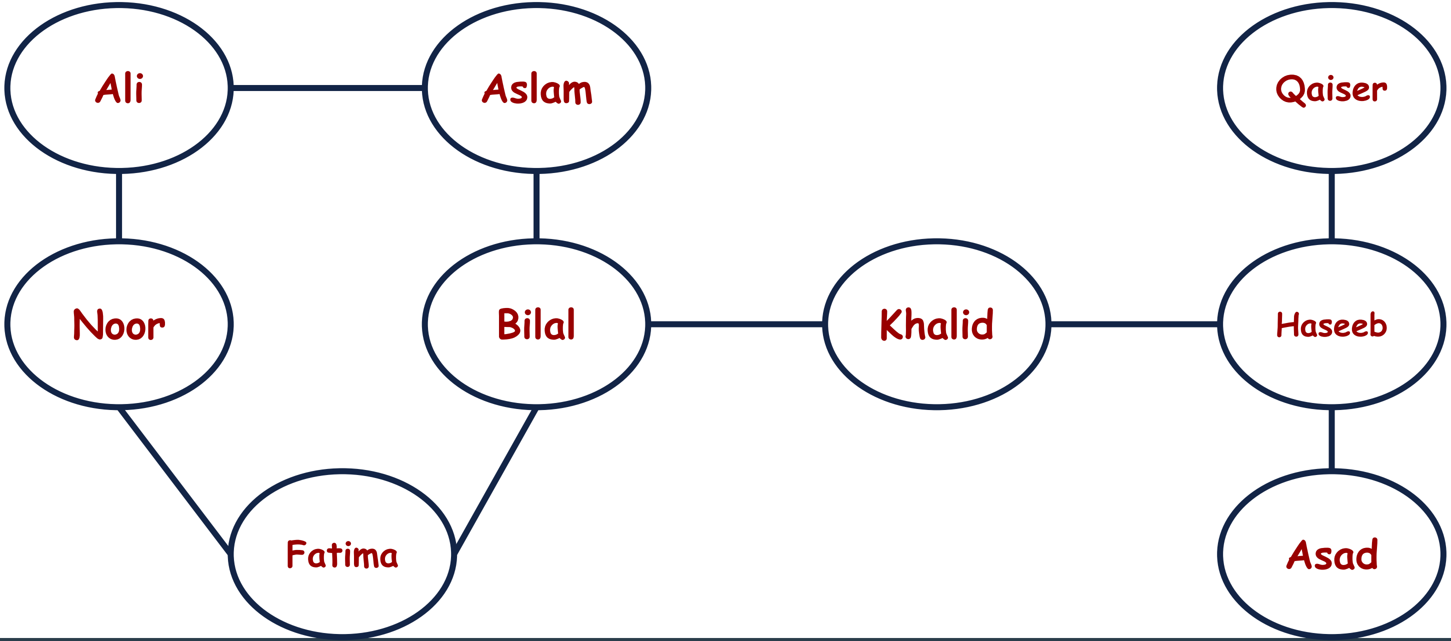


Graphs: Implementations

Representation	Time Complexity to find Adjacent nodes of a given node	Space Complexity
Edge List	O(|E|) where E = V^2	O(|V| + |E|)
Adjacency Matrix	O(|V|)	O(|V| + $\|V\|^2$)
Adjacency List	O(V) in case of Linked List O(1) in case of HashMaps	O(|V| + |E|)

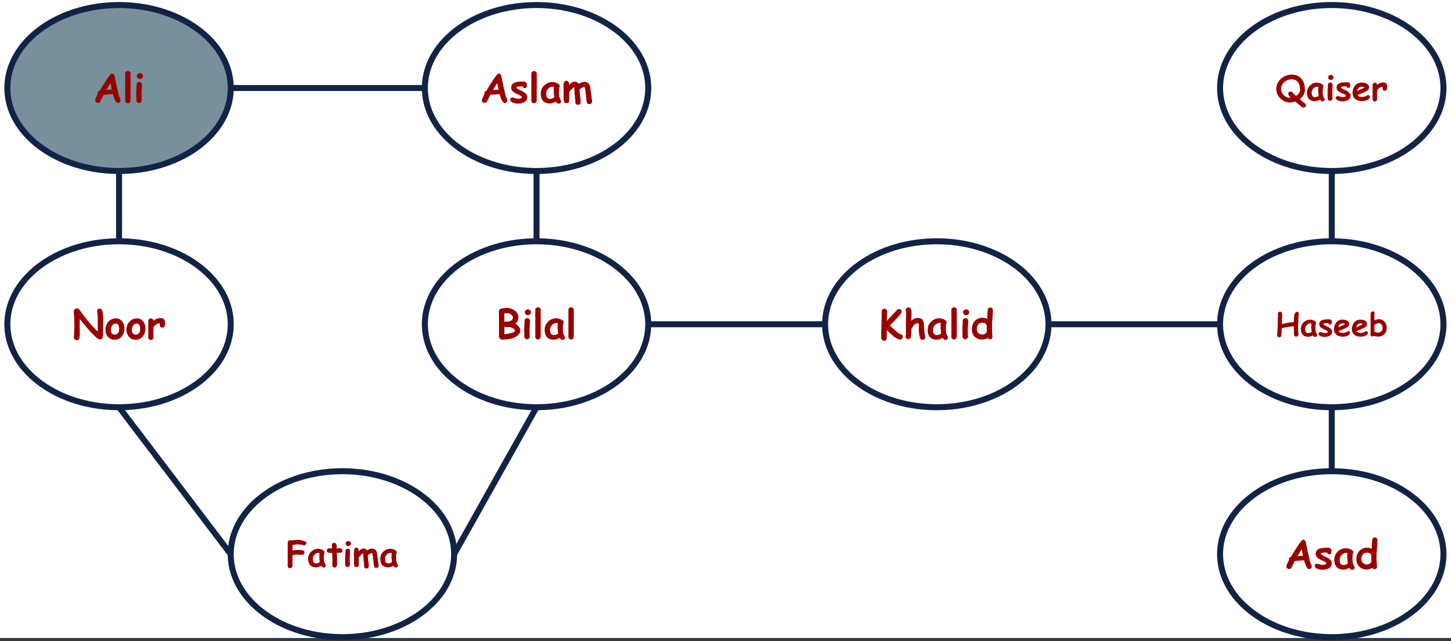
Graphs: Traversal

Now, we want to traverse the graph. How can we do that?



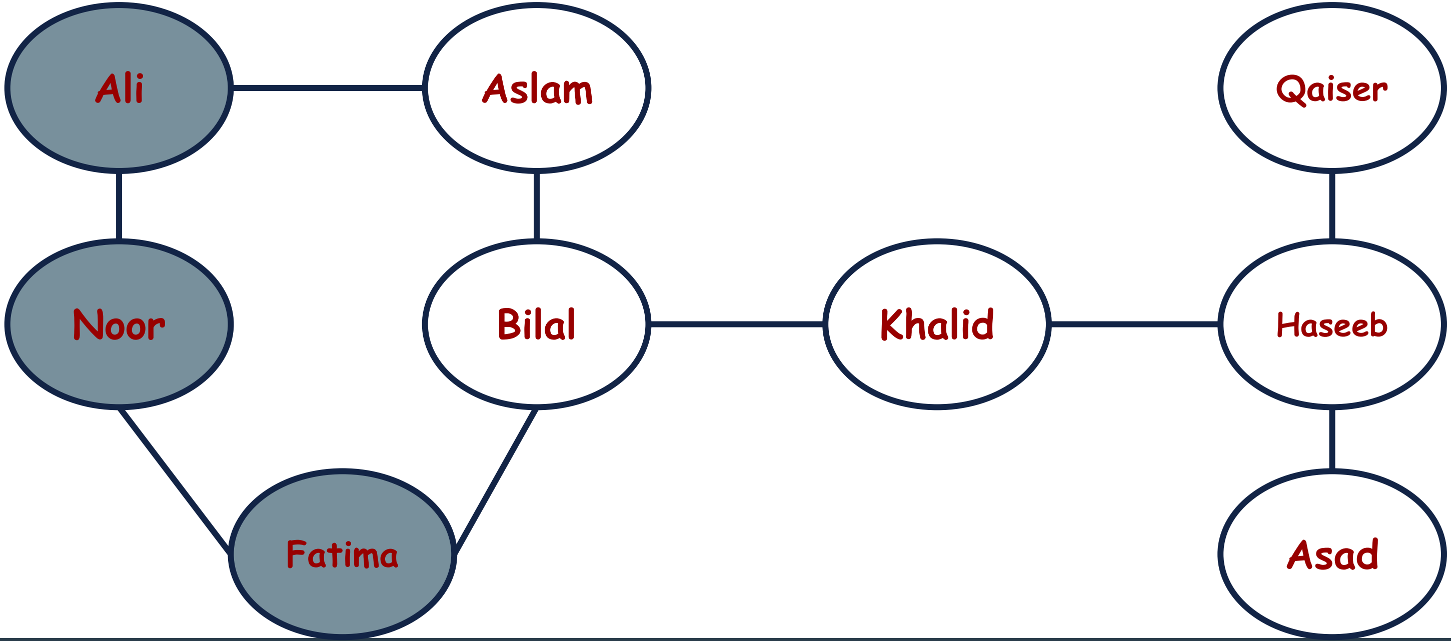
Graphs: Traversal

Let's say that the starting vertex is given which is Ali.



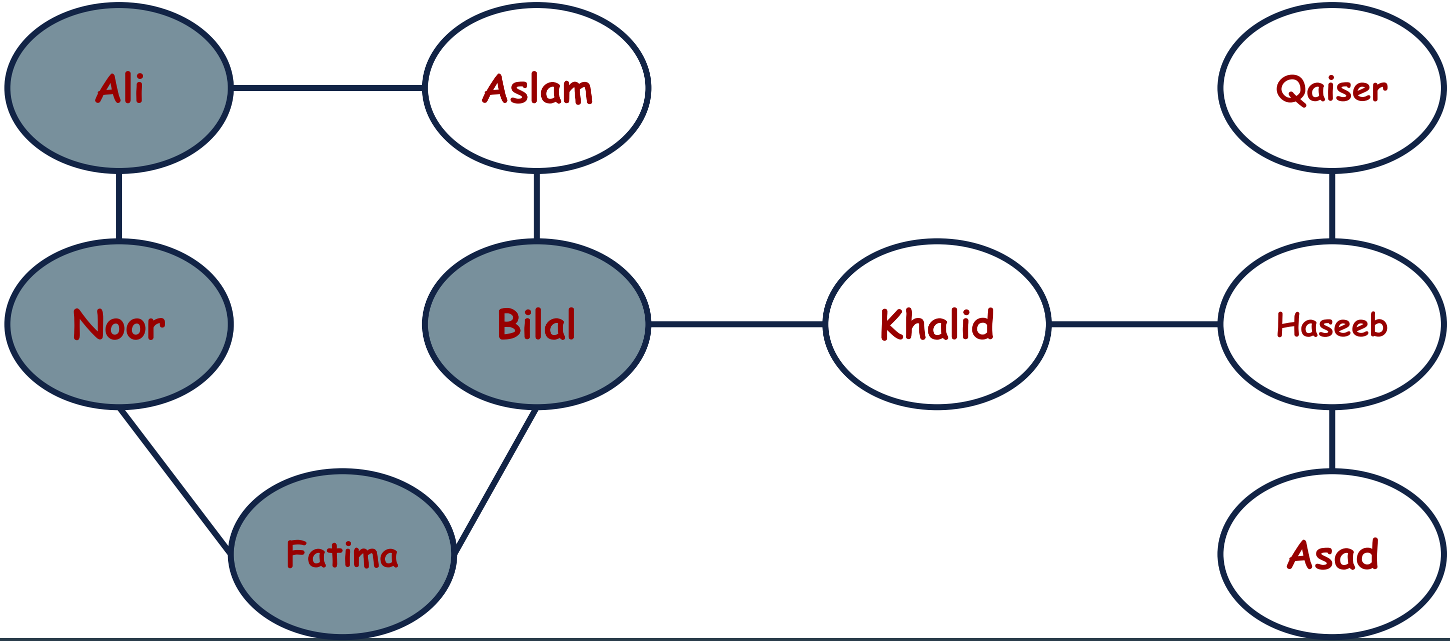
Graphs: Traversal

Now, we go to its adjacent vertex.



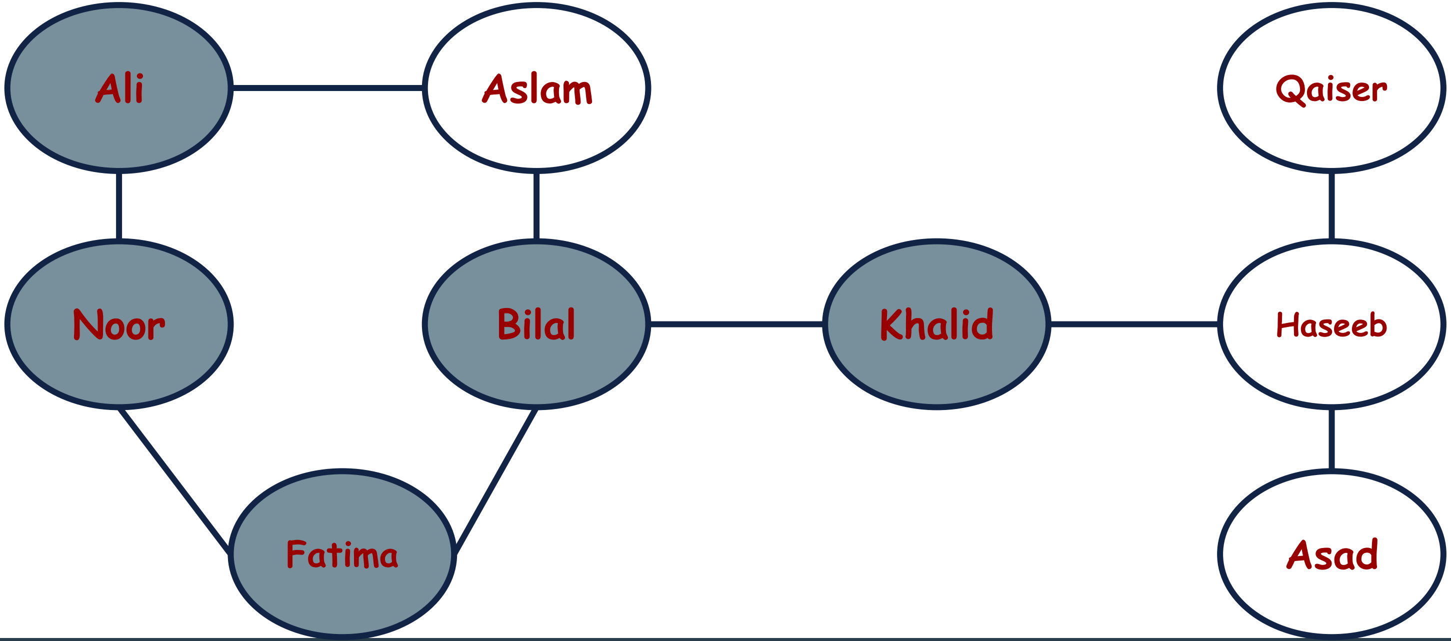
Graphs: Traversal

Now, we go to its adjacent vertex.



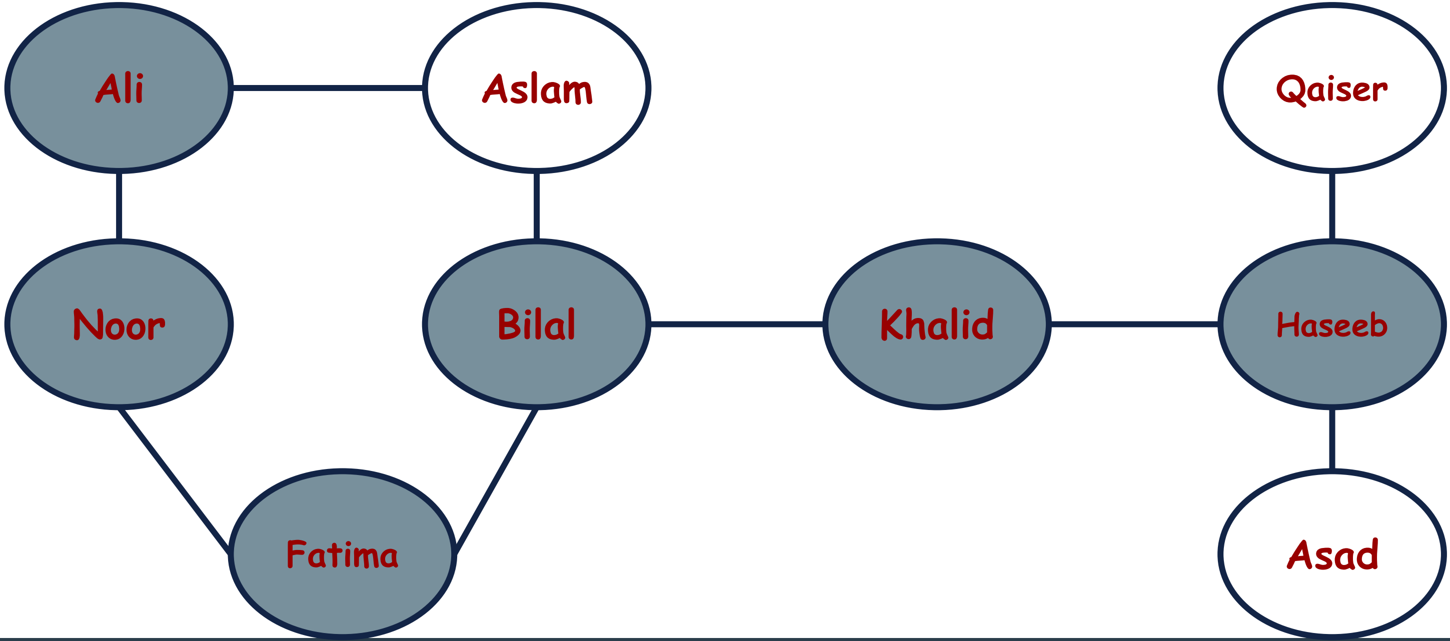
Graphs: Traversal

Now, we go to its adjacent vertex.



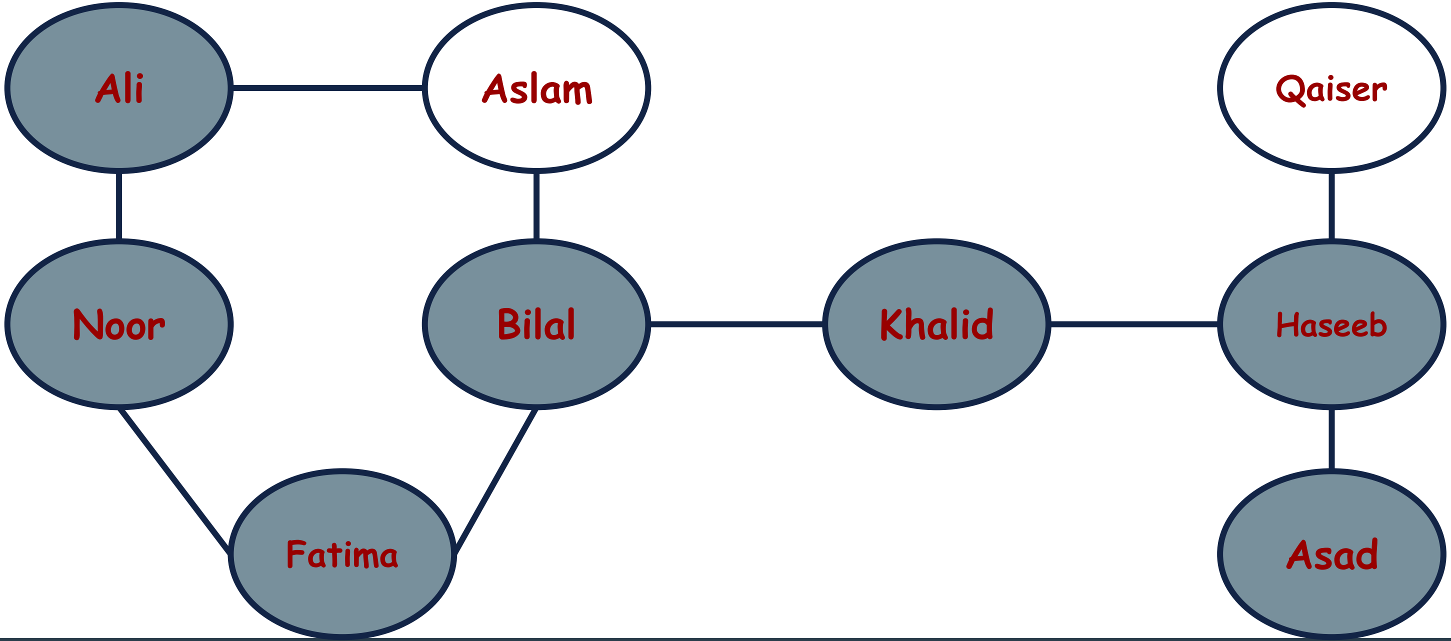
Graphs: Traversal

Now, we go to its adjacent vertex.



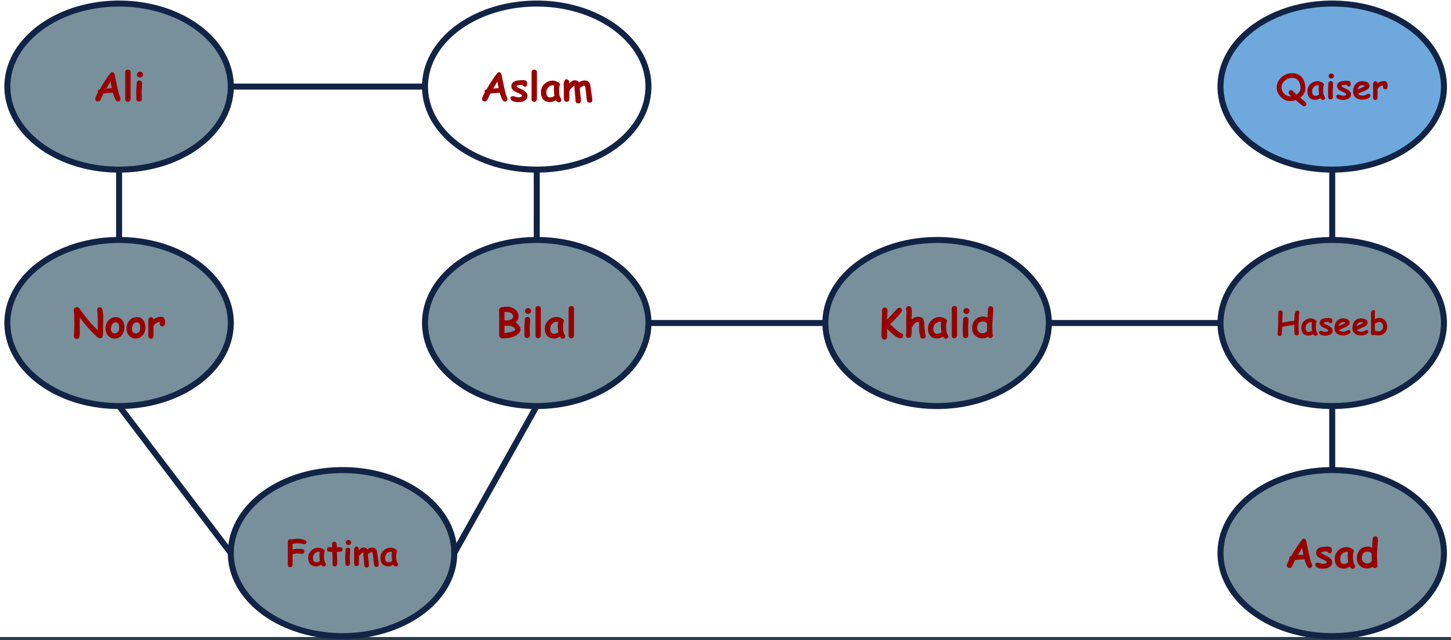
Graphs: Traversal

Now, we go to its adjacent vertex.



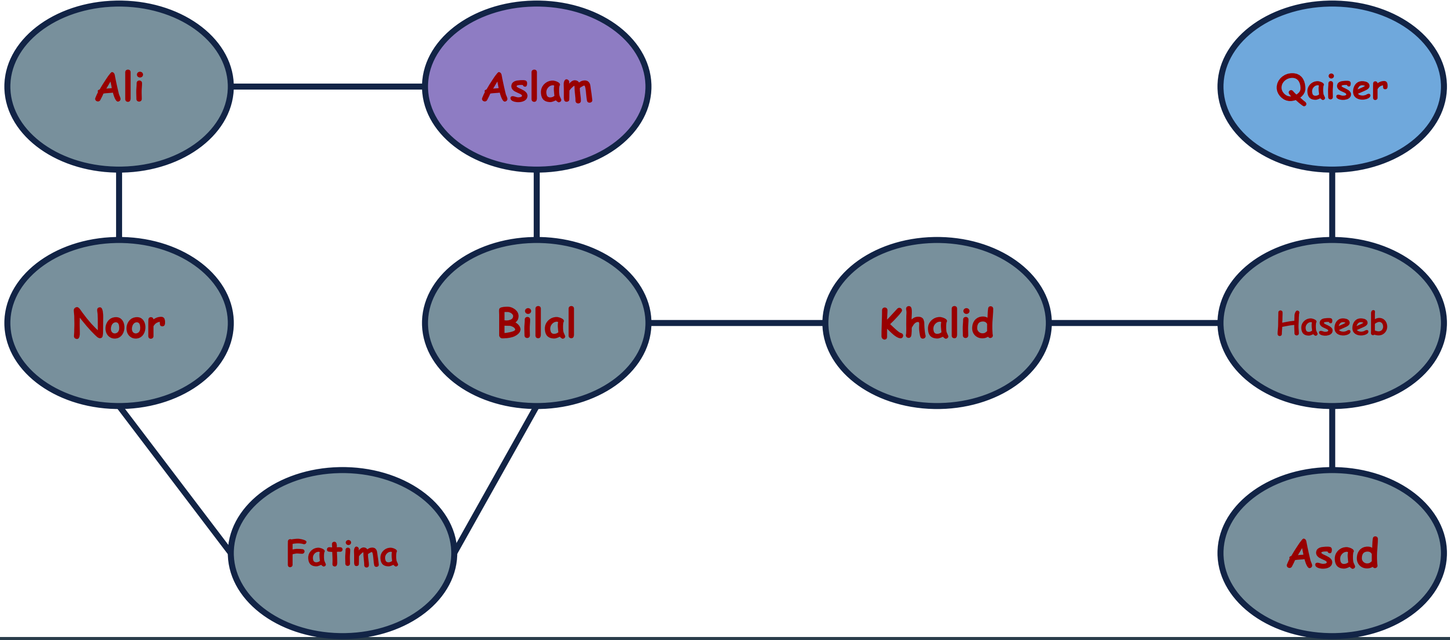
Graphs: Traversal

Now, we come back and traverse the remaining nodes of the previous nodes.



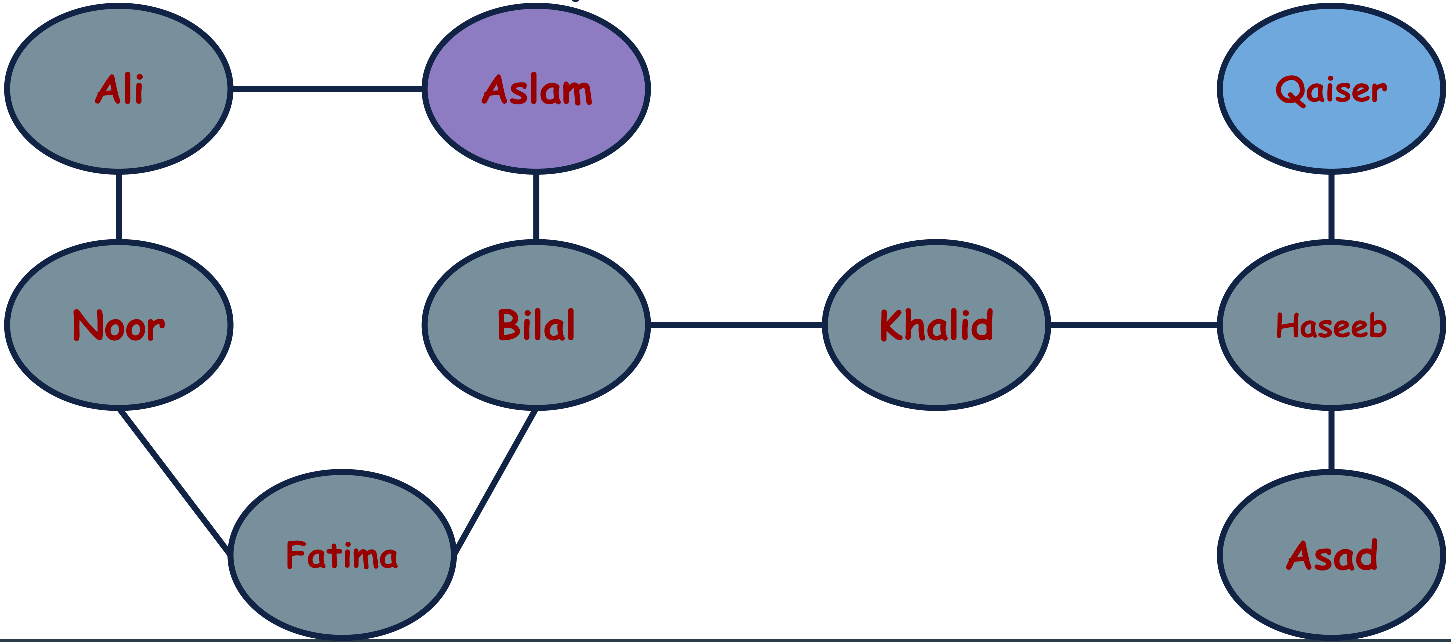
Graphs: Traversal

Now, we come back and traverse the remaining nodes of the previous nodes.



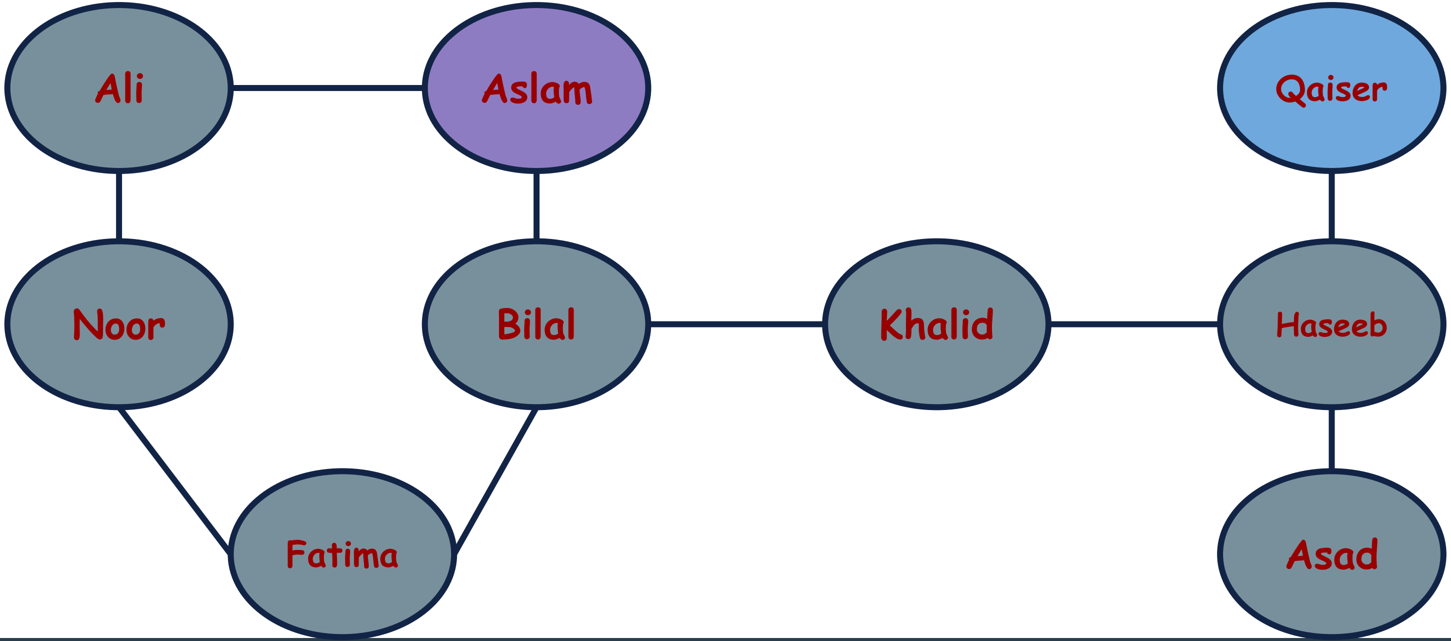
Graphs: Traversal

Here, the important thing to note is that we are visiting the vertices in the Depth First Order.



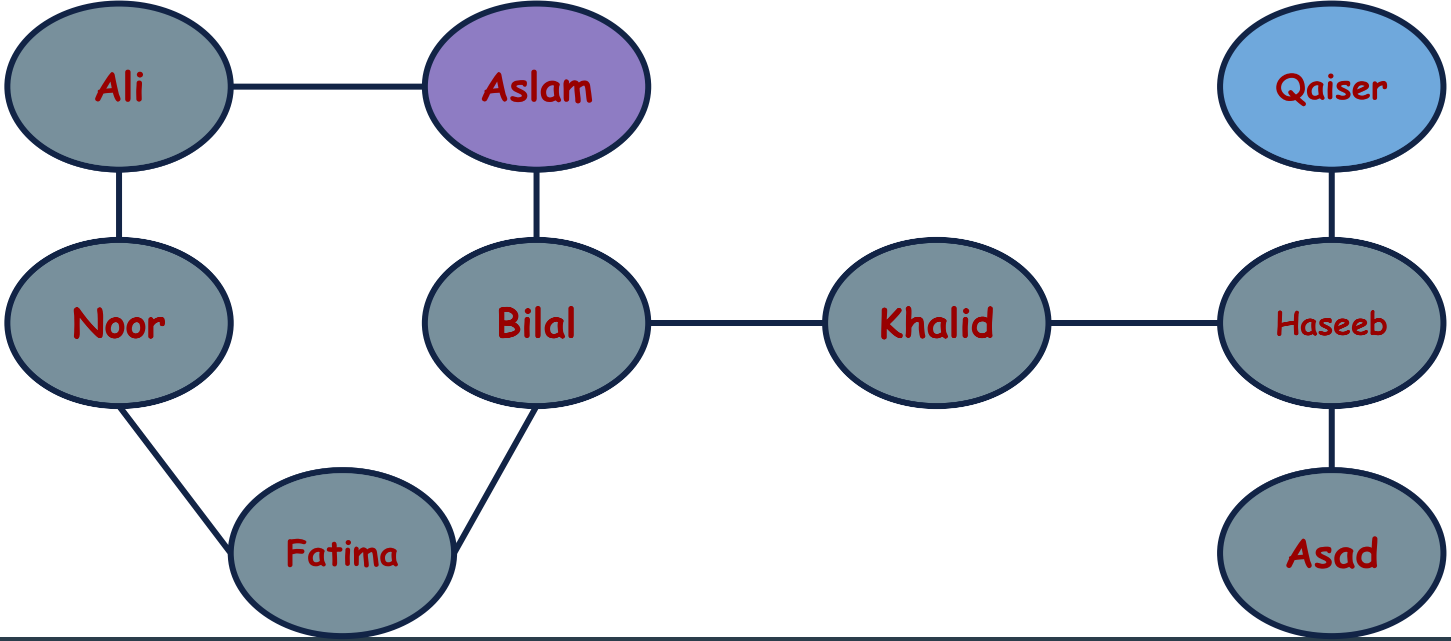
Graphs: Depth First Traversal

Therefore, this traversal is called as Depth First Traversal.



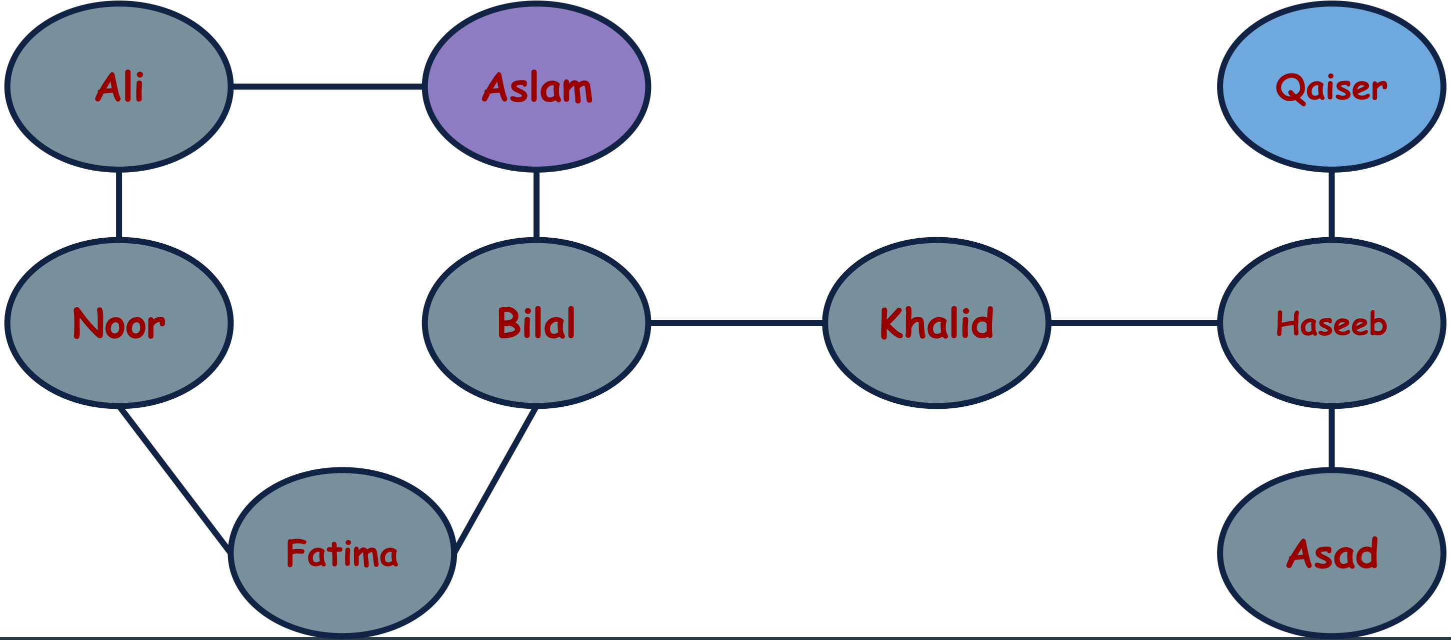
Graphs: Depth First Traversal

The depth first traversal algorithm of Graph is same as the depth first traversal of the Trees.



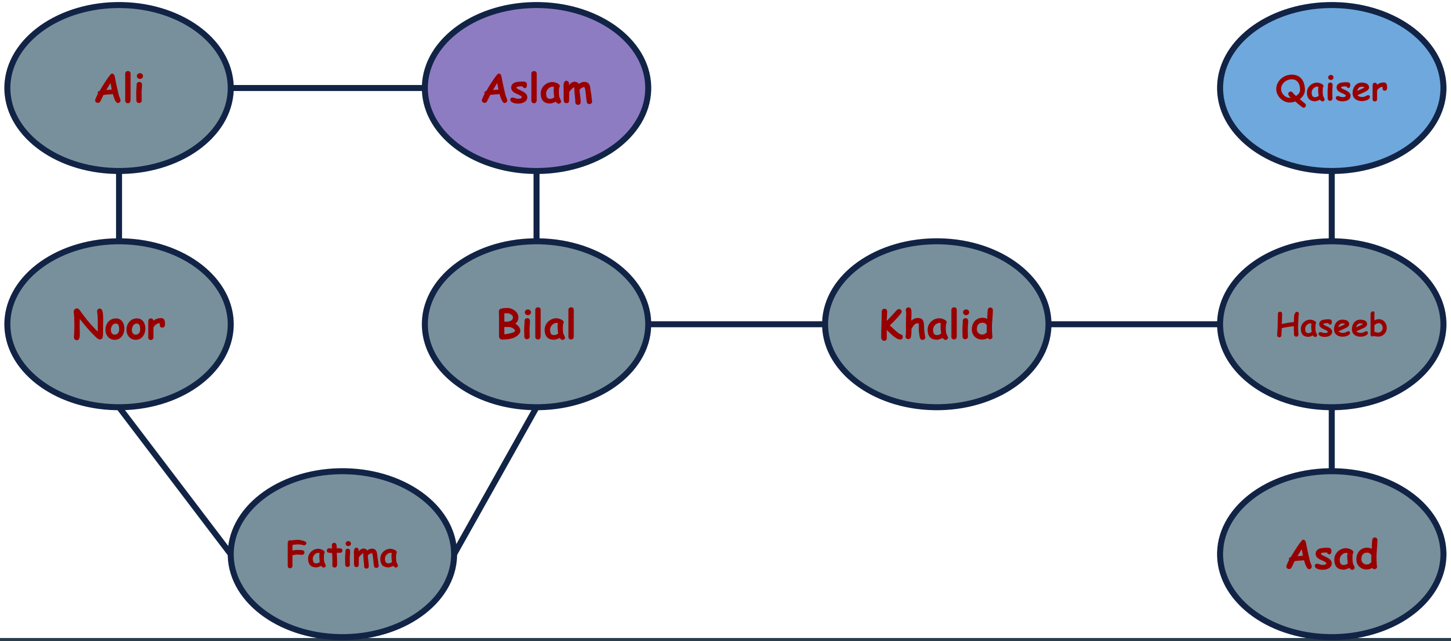
Graphs: Depth First Traversal

There is just one thing to make sure that you do not visit those vertices again that you had already visited.



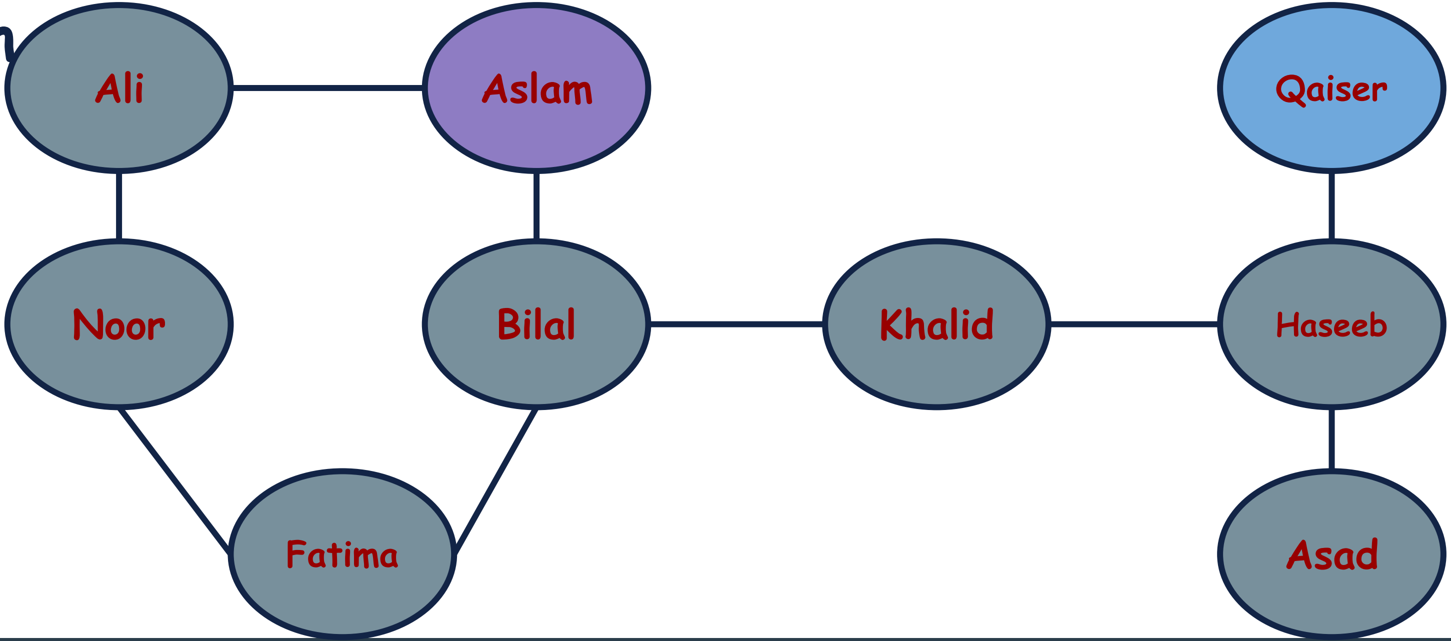
Graphs: Depth First Traversal

How can you do that?



Graphs: Depth First Traversal

You can make an array/vector/hashmap to store which vertices have been visited and which vertices have not been



Depth First Traversal: Implementation

Now, let's implement the depth first traversal for Graphs.

Hint: Do not forget the previous Data Structures that we have studied before.

Depth First Traversal: Implementation

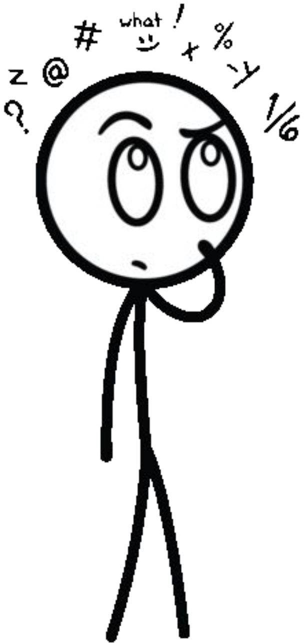
```
class Graph
{
    unordered_map<string, vector<string>> adjList;

public:
    void addEdge(string s, string d)
    {
        adjList[s].push_back(d);
        adjList[d].push_back(s);
    }
```

Depth First Traversal: Implementation

```
void DFS(string start)
{
    unordered_map<string, bool> visited;
    stack<string> s;
    s.push(start);
    visited[start] = true;
    while (!s.empty())
    {
        string current = s.top();
        s.pop();
        cout << current << " ";
        for (auto currentFriend : adjList.find(current)->second)
        {
            if (visited.find(currentFriend) == visited.end())
            {
                s.push(currentFriend);
                visited[currentFriend] = true;
            }
        }
    }
}
```

Depth First Traversal: Implementation

What is the Time Complexity of this algorithm?



Depth First Traversal: Implementation

We know that we have traversed all the Vertices of the graph.

For Looping all the Vertices time complexity is $O(|V|)$.

Depth First Traversal: Implementation

We know that we have traversed all the Vertices of the graph.
For Looping all the Vertices time complexity is $O(|V|)$.

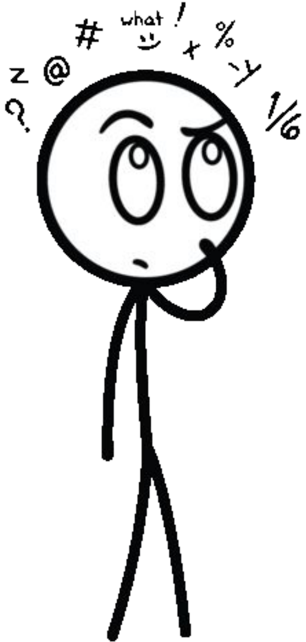
For each vertex in the HashMap we have to iterate all of its Edges.
For the undirected graph an edge is stored twice.
Therefore, for looping all the edges the time complexity is $O(2|E|)$

Depth First Traversal: Implementation

Overall Time Complexity is $O(|V| + |E|)$.

Depth First Traversal: Implementation

What is the Space Complexity of this algorithm?



Depth First Traversal: Implementation

We are declaring a stack as well as the array/hashMap to store the information of visited and unvisited vertices.

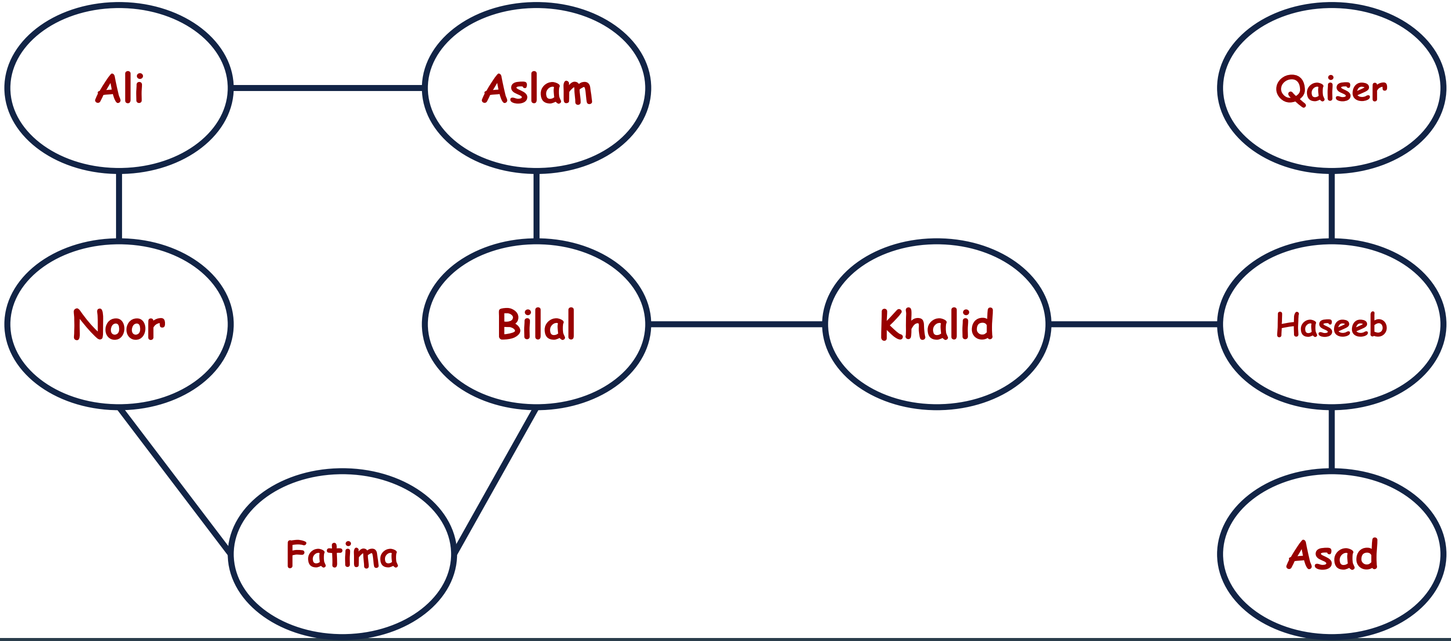
Worst Space Complexity is $O(|V|)$.

Graphs: Traversal Algorithm

Algorithm	Time Complexity	Space Complexity
Depth First Traversal	O(|V| + |E|)	O(|V|)

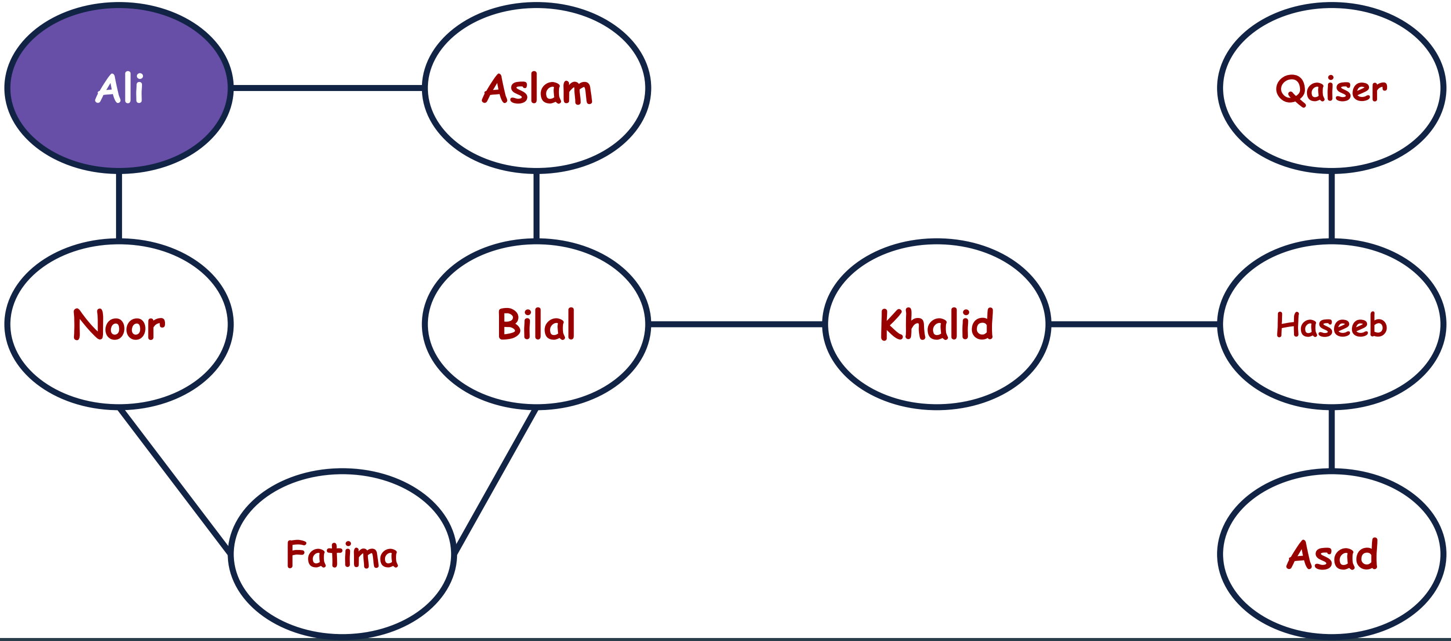
Graphs: Traversal

There are another way to traverse the graphs.



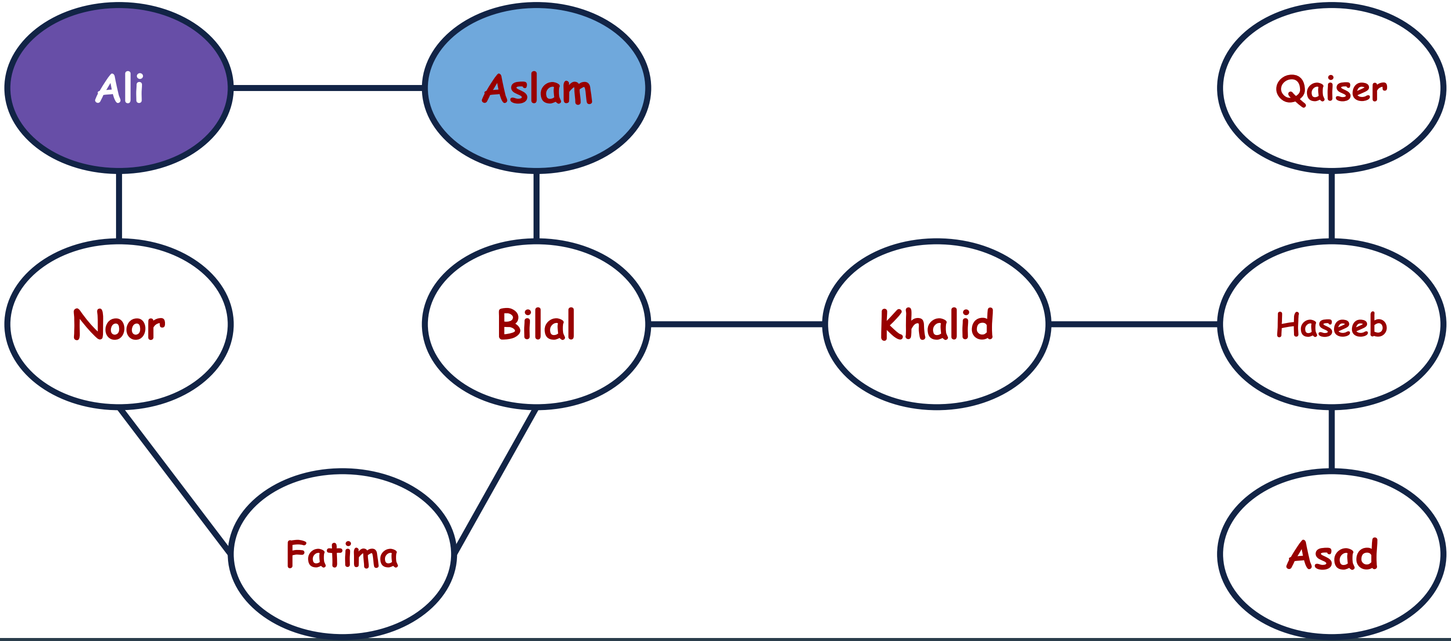
Graphs: Traversal

Let's say that the starting vertex is given which is Ali.



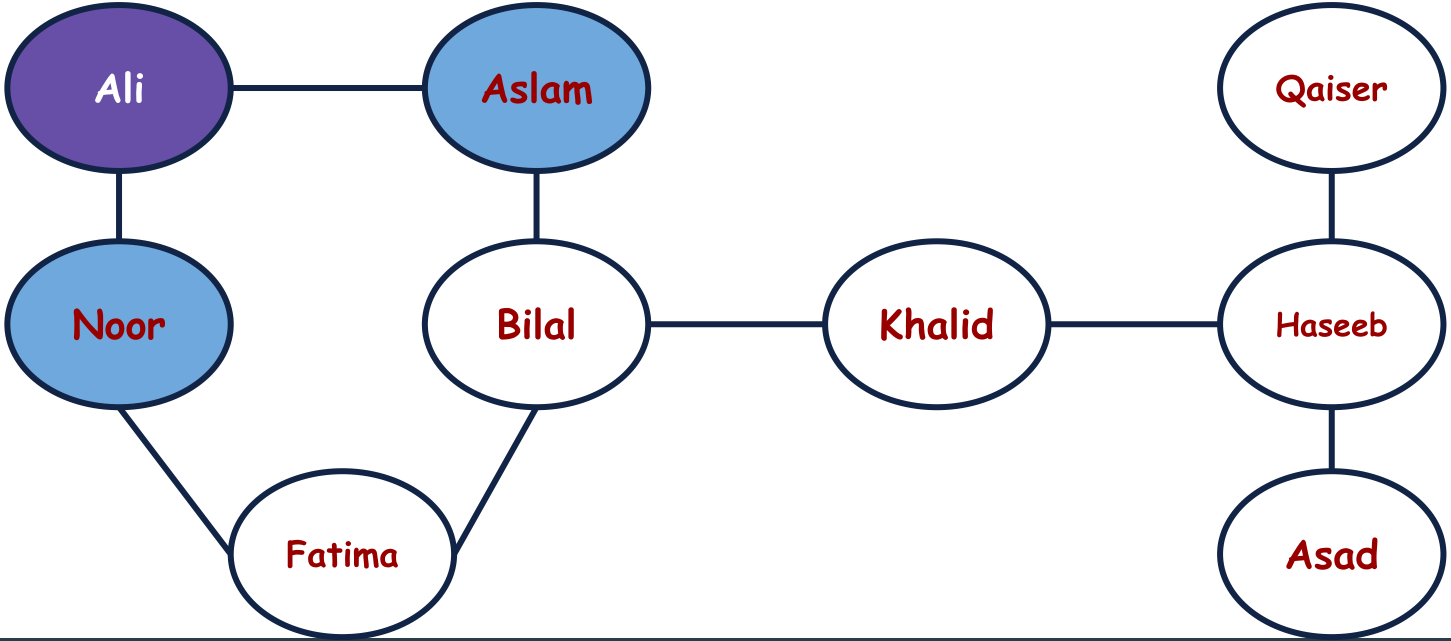
Graphs: Traversal

Now, first Visit all the adjacent nodes of Ali one by one.



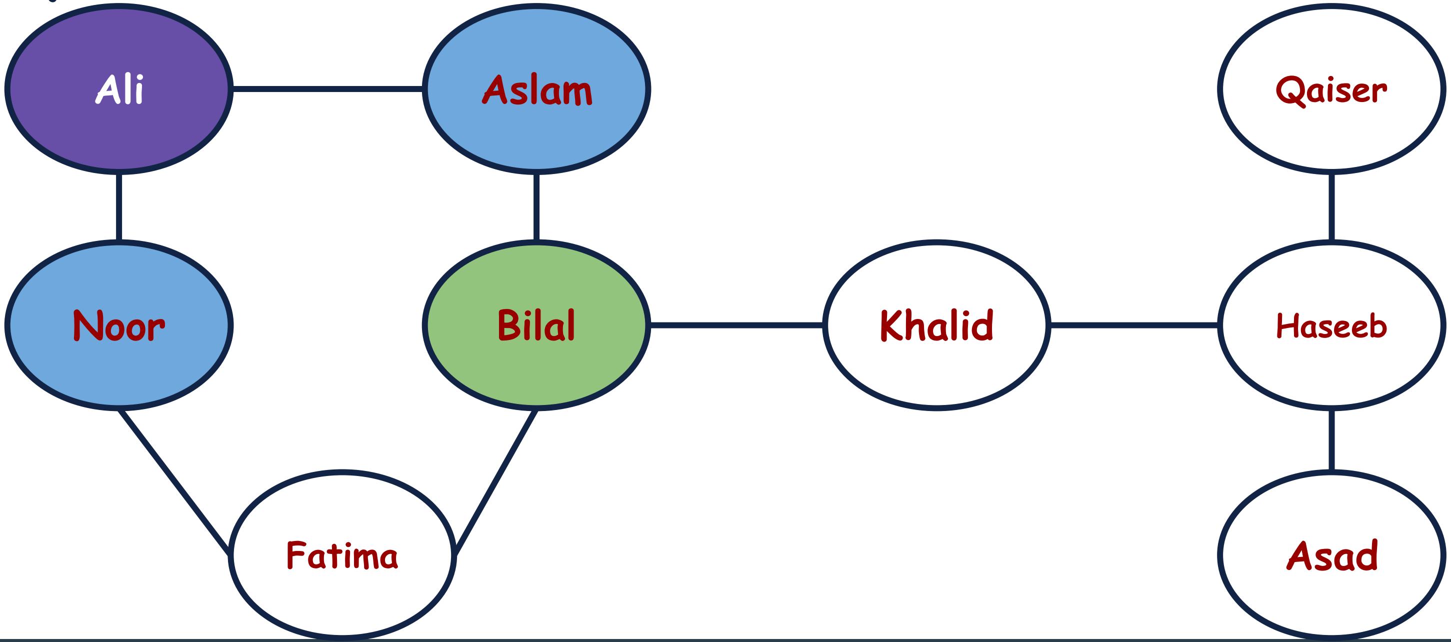
Graphs: Traversal

Now, first Visit all the adjacent nodes of Ali one by one.



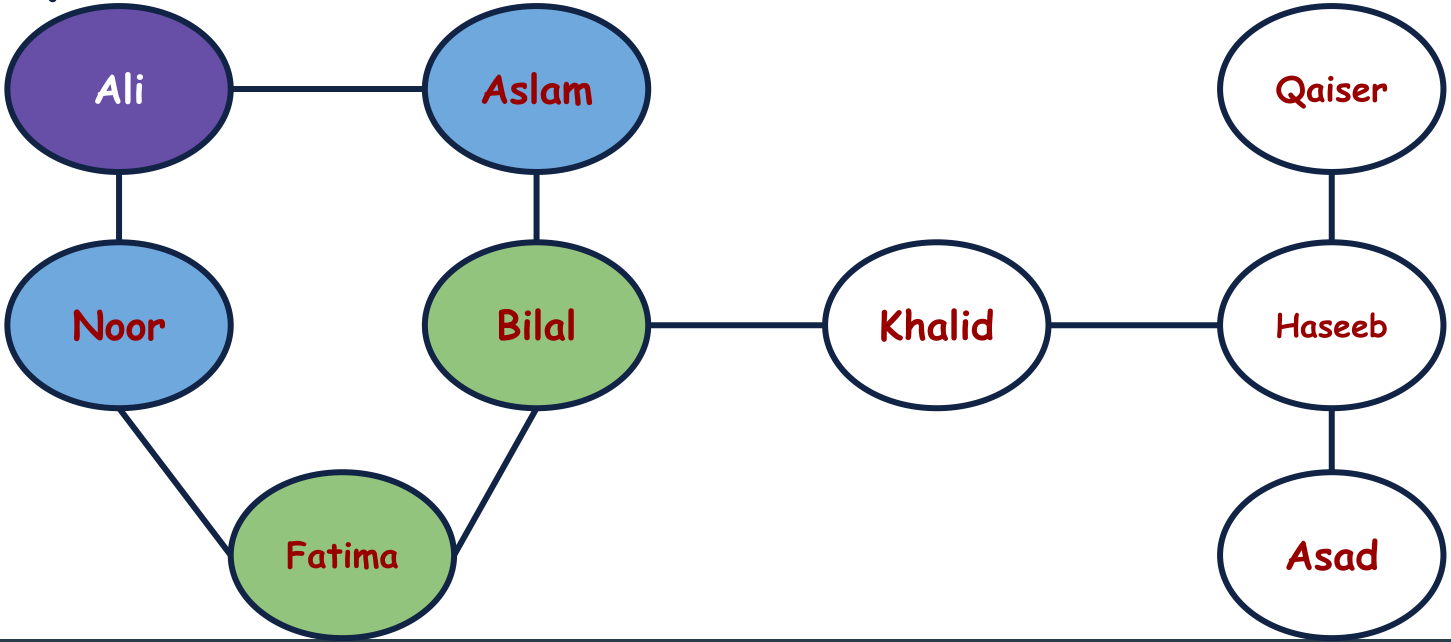
Graphs: Traversal

Now, Visit all the adjacent nodes of the visited nodes one by one.



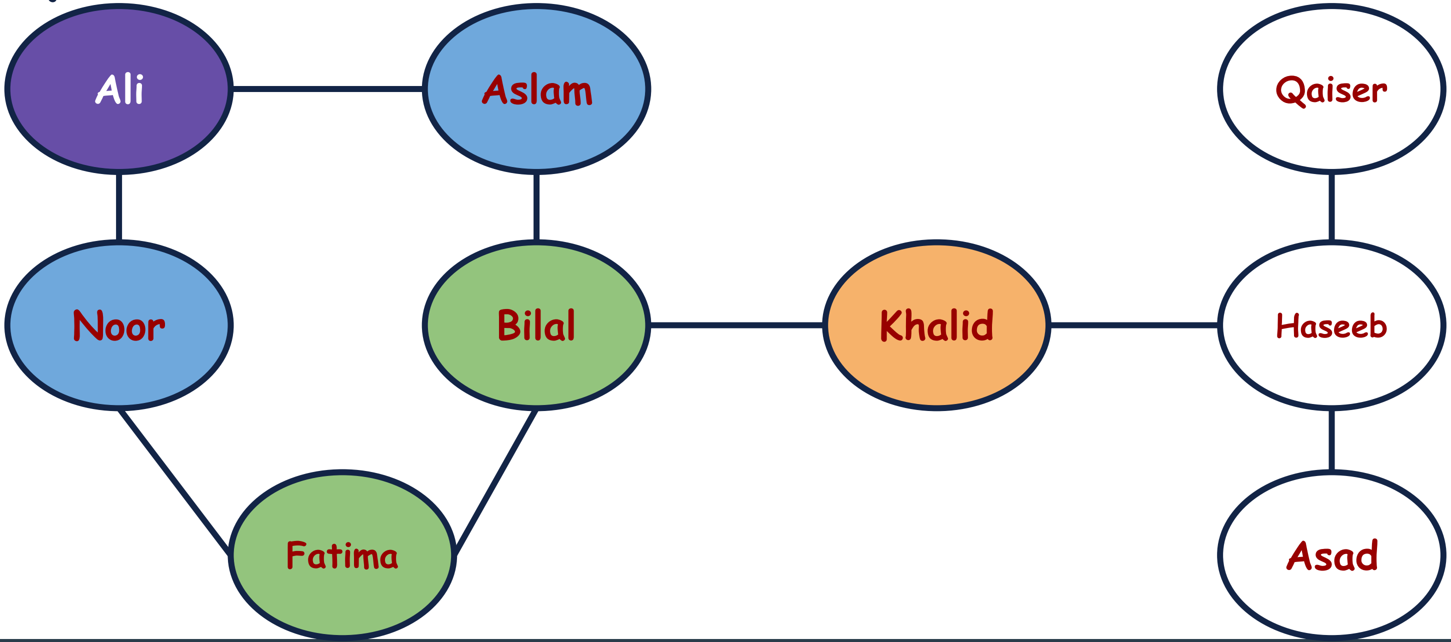
Graphs: Traversal

Now, Visit all the adjacent nodes of the visited nodes one by one.



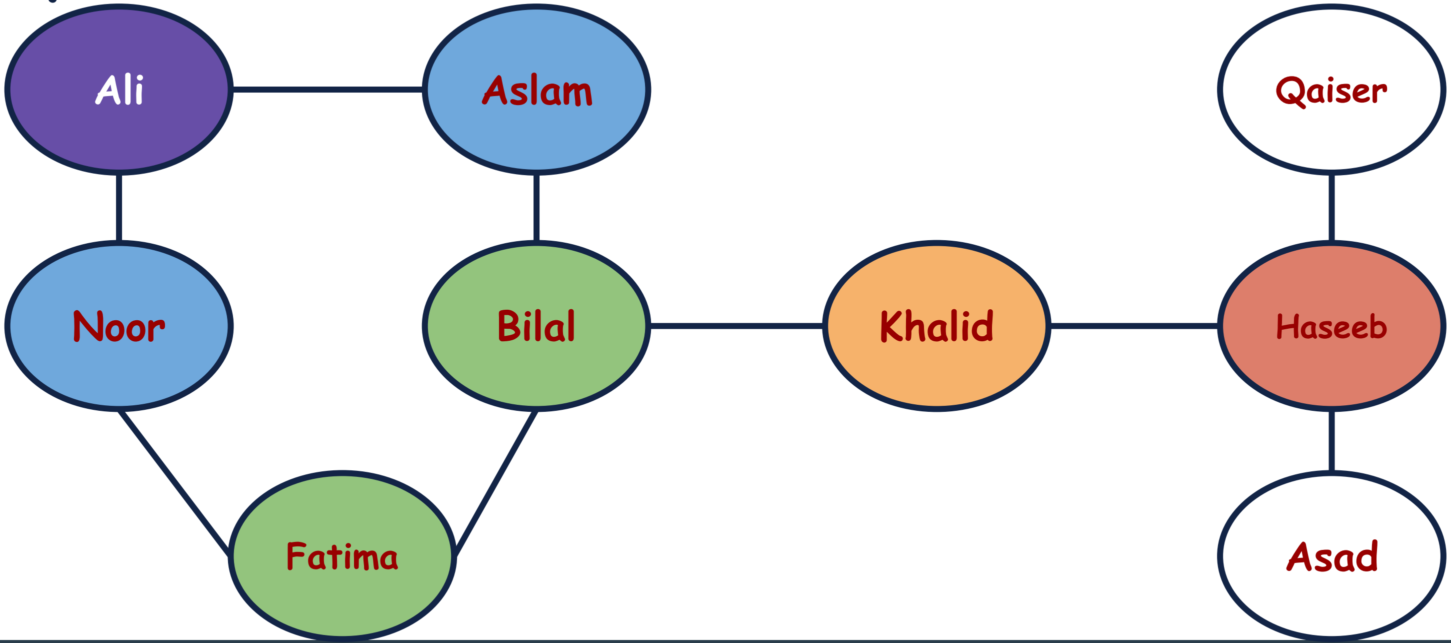
Graphs: Traversal

Now, Visit all the adjacent nodes of the visited nodes one by one.



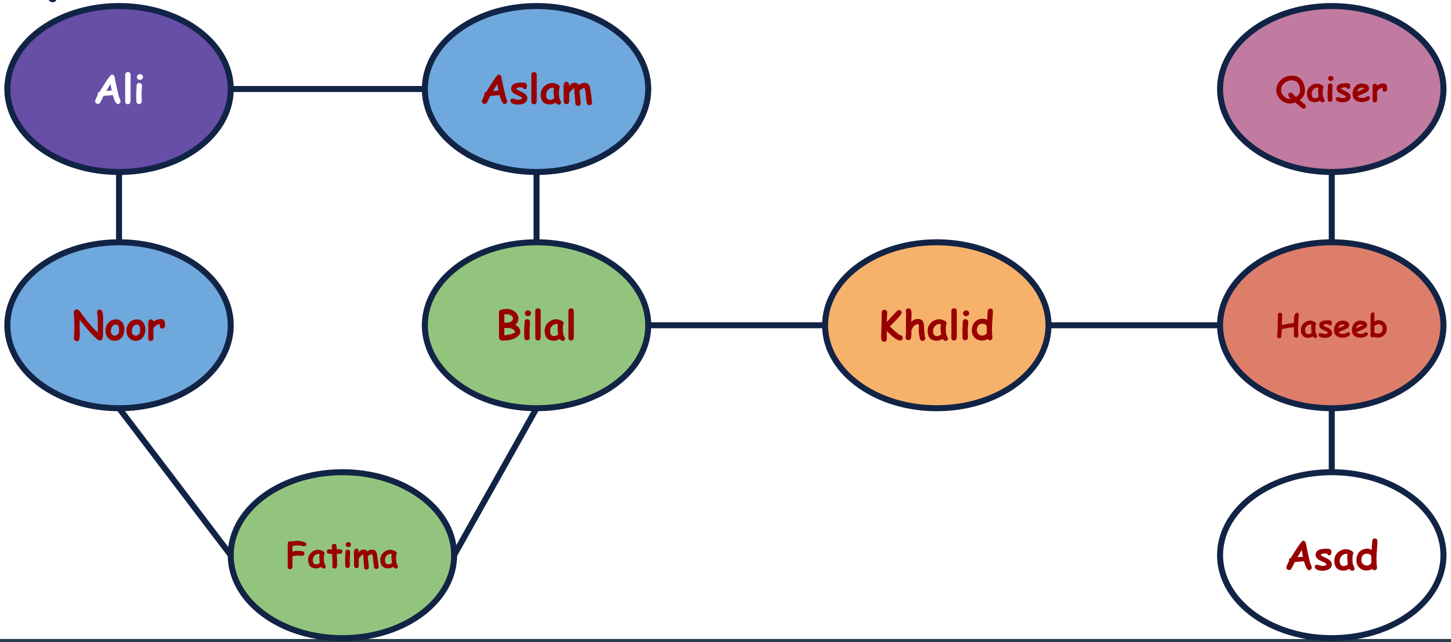
Graphs: Traversal

Now, Visit all the adjacent nodes of the visited nodes one by one.



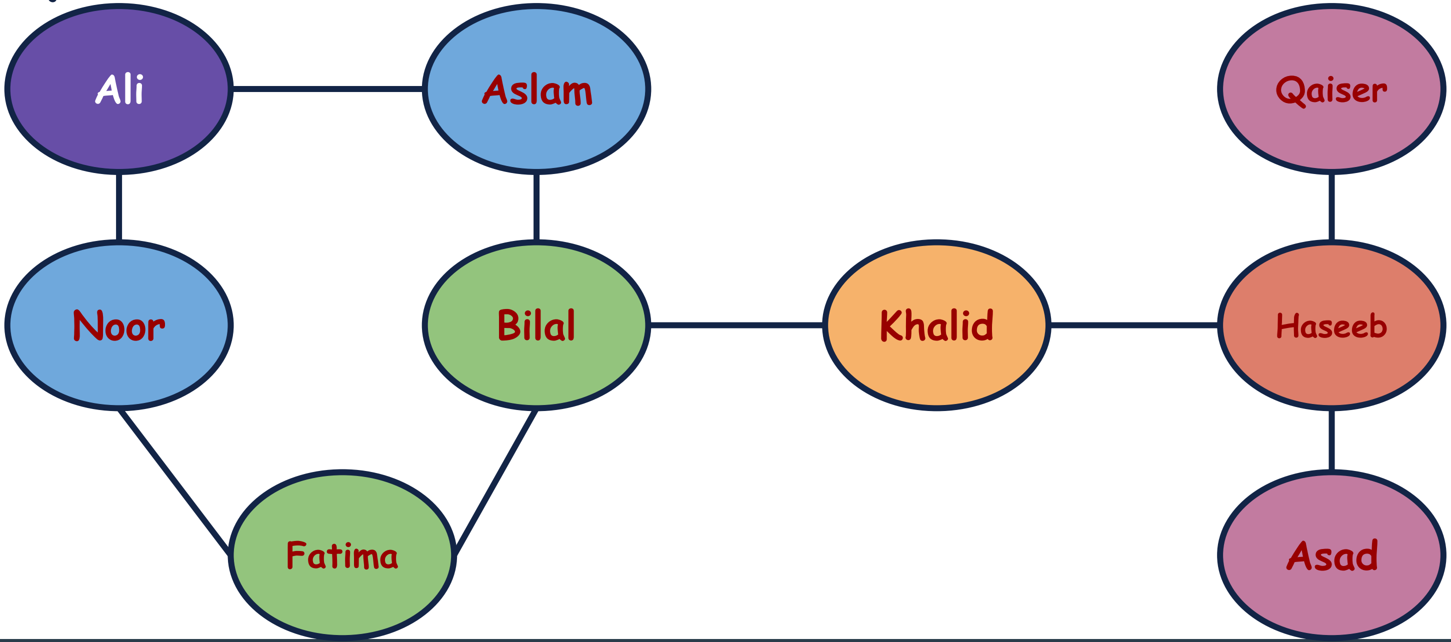
Graphs: Traversal

Now, Visit all the adjacent nodes of the visited nodes one by one.



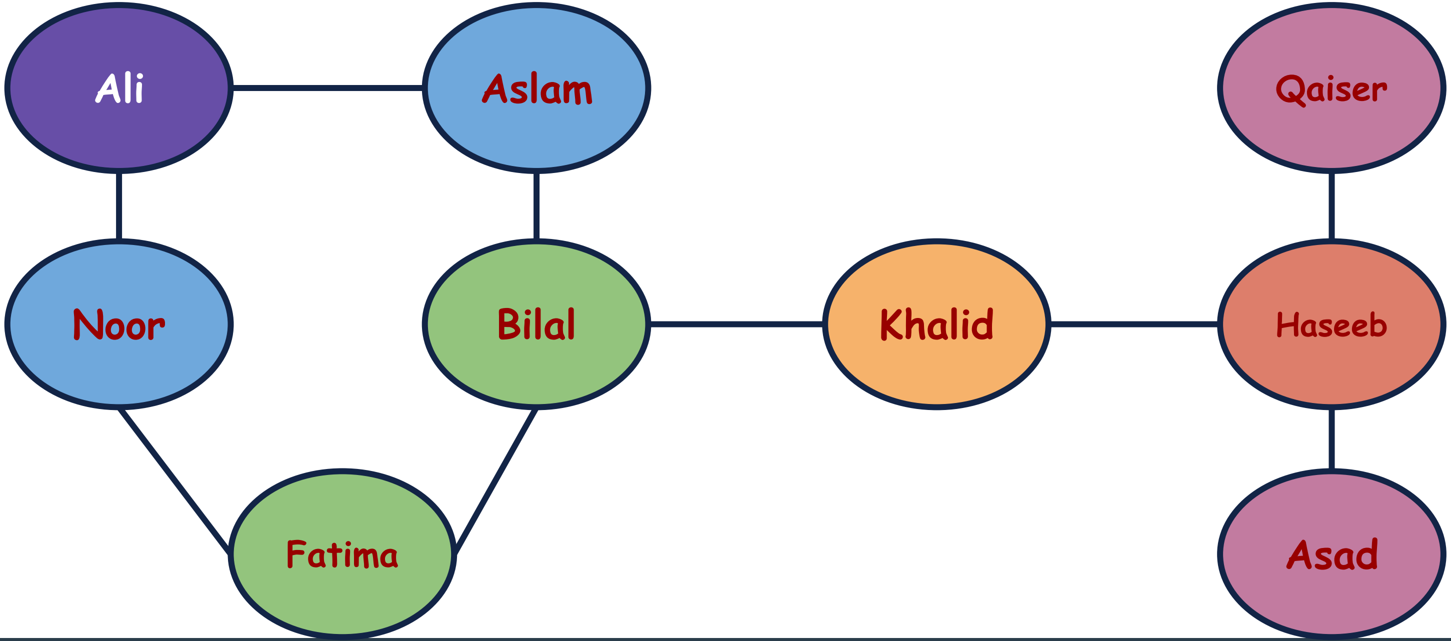
Graphs: Traversal

Now, Visit all the adjacent nodes of the visited nodes one by one.



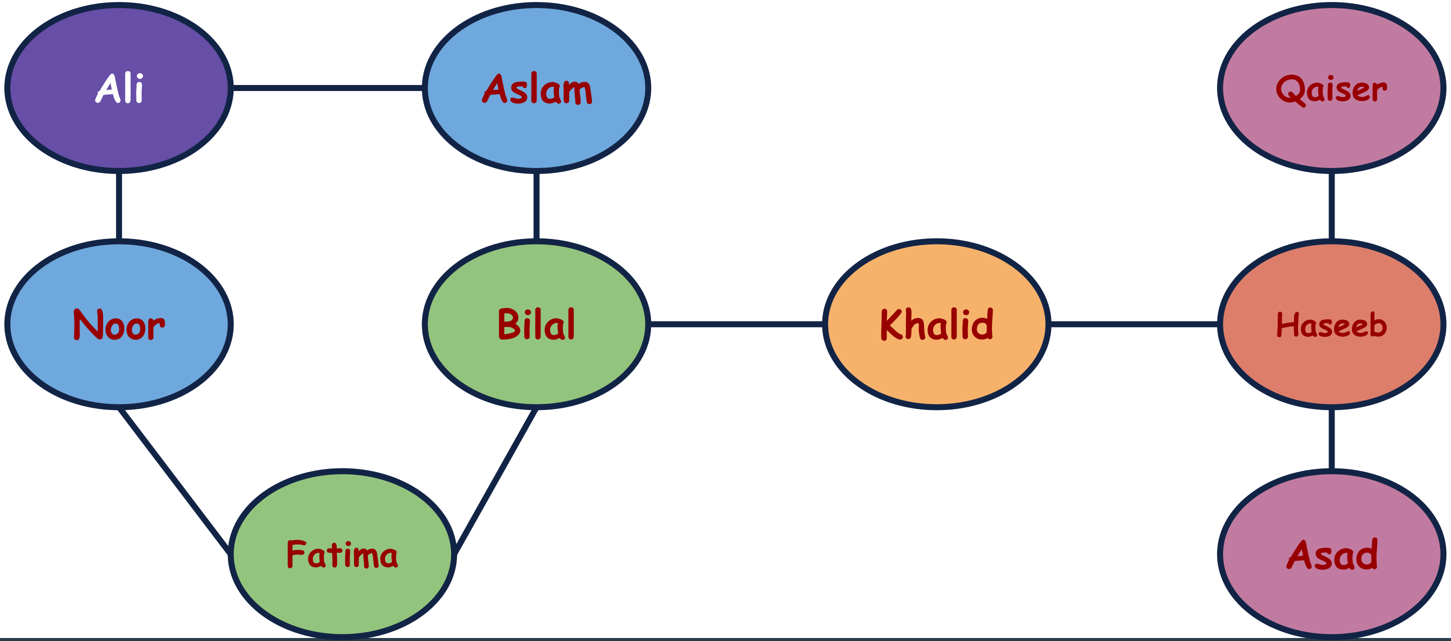
Graphs: Traversal

Here, the important thing to note is that we are visiting the vertices in the Breadth First Order.



Graphs: Traversal

Again we have to keep track of the visited and unvisited nodes.



Breadth First Traversal: Implementation

Now, let's implement the breath first traversal for Graphs.

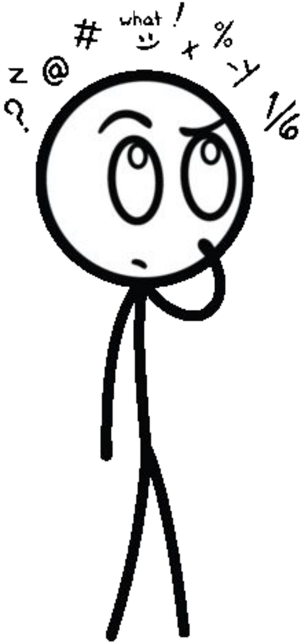
Hint: Do not forget the previous Data Structures that we have studied before.

Breadth First Traversal: Implementation

```
void BFS(string start)
    {
        unordered_map<string, bool> visited;
        queue<string> q;
        q.push(start);
        visited[start] = true;
        while (!q.empty())
        {
            string current = q.front();
            q.pop();
            cout << current << " ";
            for (auto currentFriend : adjList.find(current)->second)
            {
                if (visited.find(currentFriend) == visited.end())
                {
                    q.push(currentFriend);
                    visited[currentFriend] = true;
                }
            }
        }
    }
```

Breadth First Traversal: Implementation

What is the Time and Space Complexity of this algorithm?

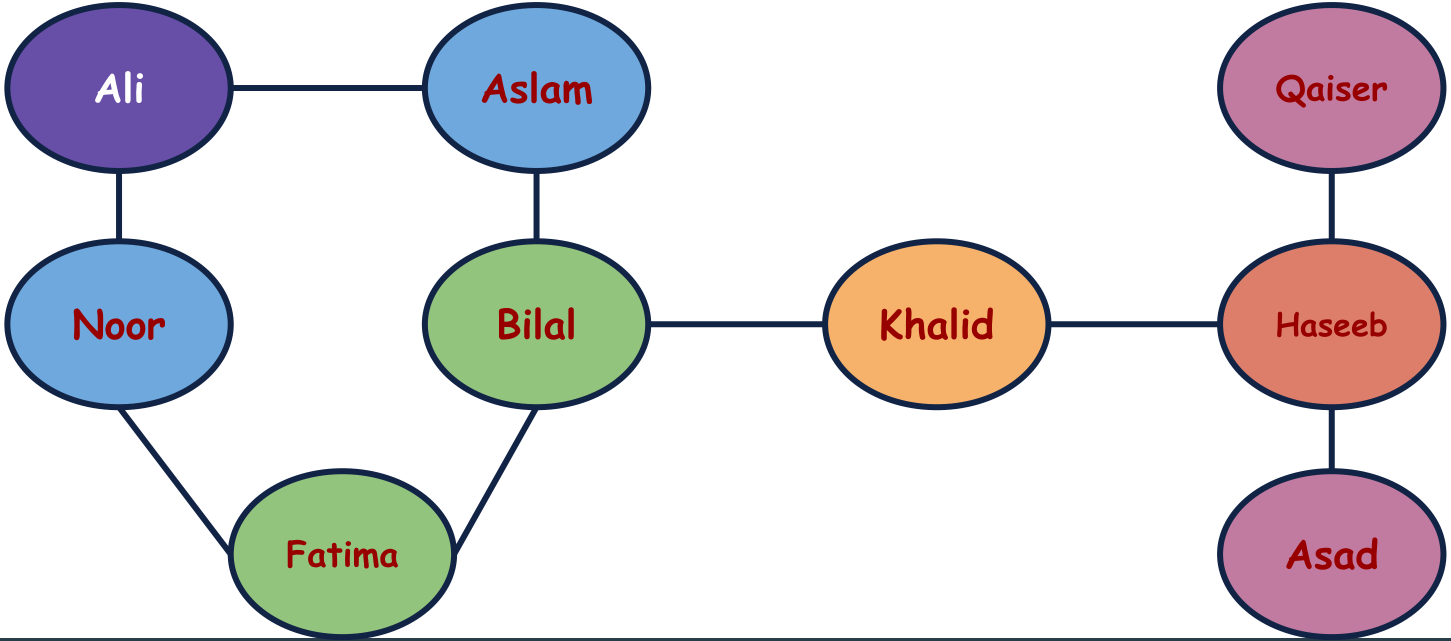


Graphs: Traversal Algorithm

Algorithm	Time Complexity	Space Complexity
Depth First Traversal	$O(\|V\| + \|E\|)$	$O(\|V\|)$
Breadth First Traversal	$O(\|V\| + \|E\|)$	$O(\|V\|)$

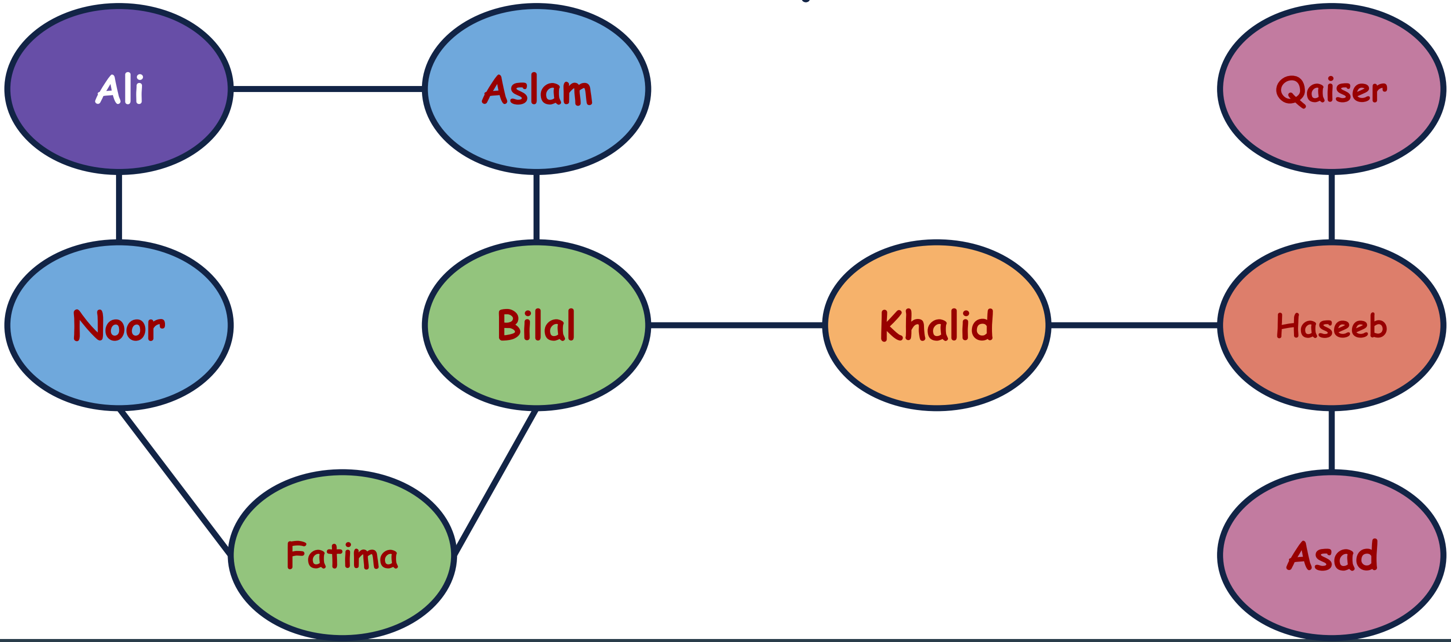
Graphs: Shortest Path

Now, i want to see what is the shortest path from Ali to Khalid. How can i do that?



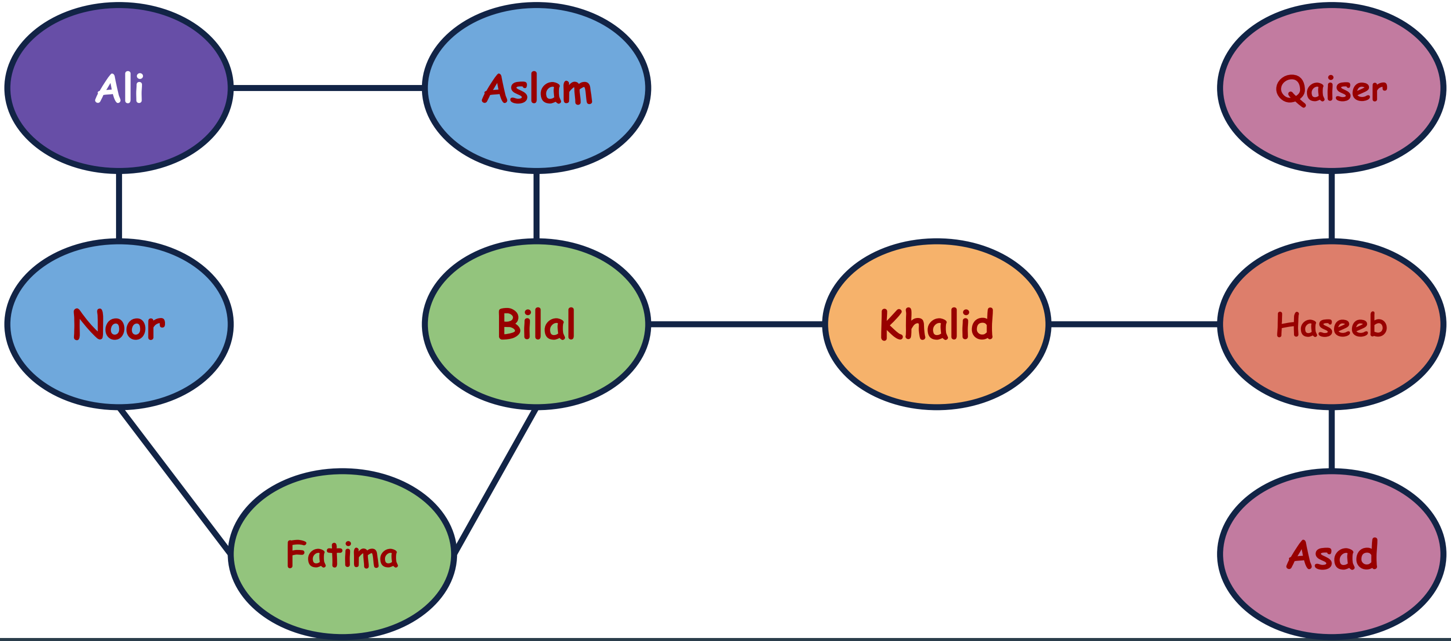
Graphs: Shortest Path

In case of undirected Graph, improvement of BFS can be used to find the shortest path between 2 vertices



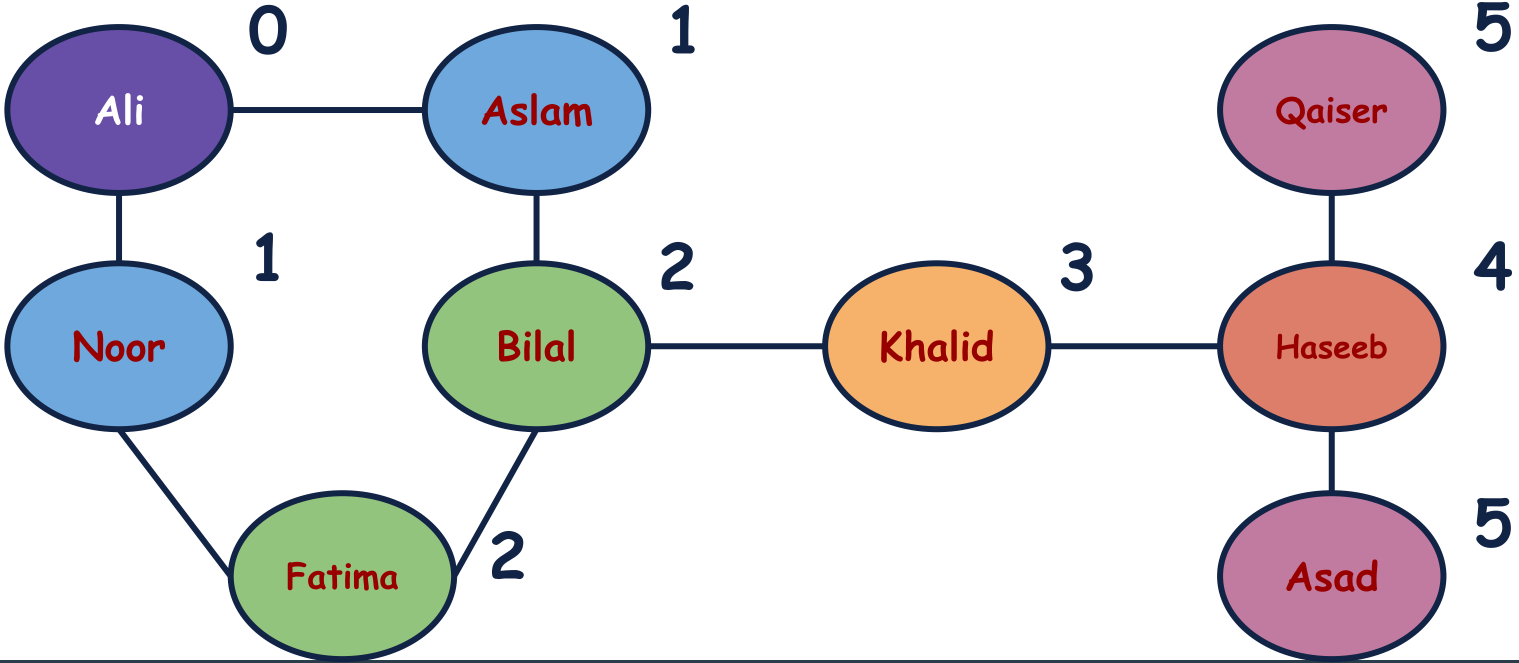
Graphs: Shortest Path

Let’s keep the cost to reach the next vertex.



Graphs: Shortest Path

Let's keep the cost to reach the next vertex.



Shortest Path (Undirected Graph): Implementation

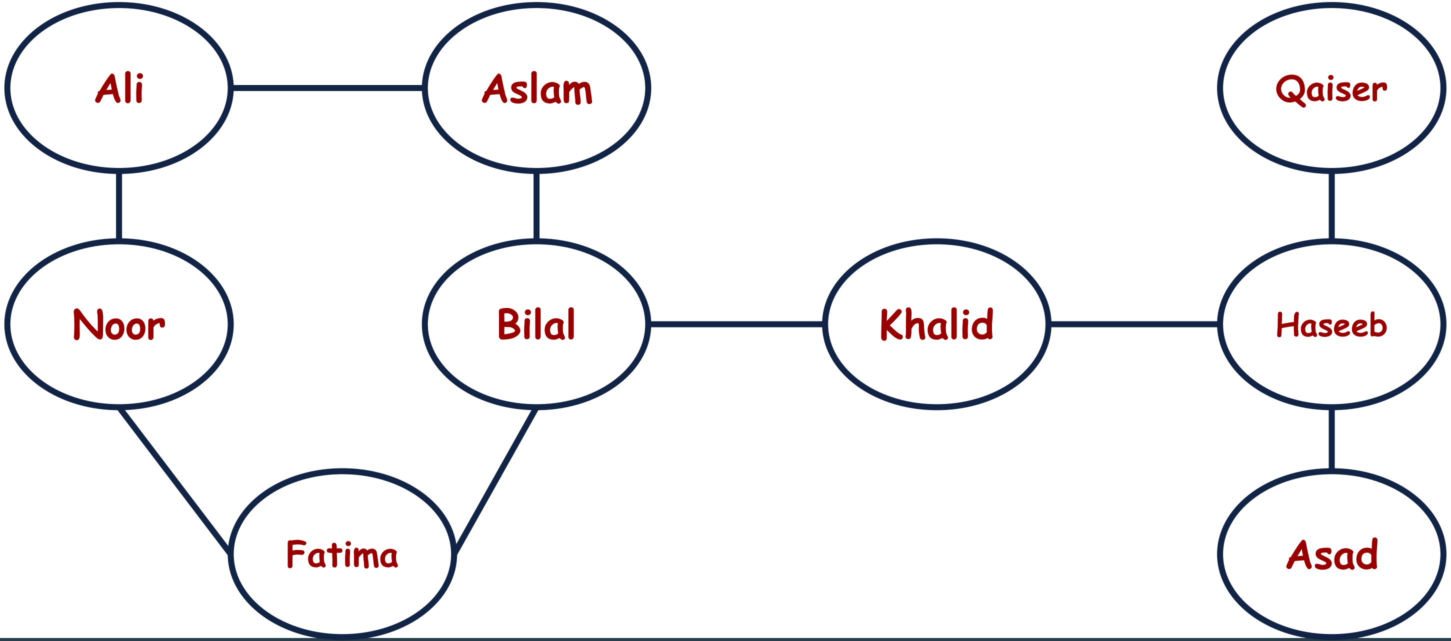
Now, let's implement the solution.

Shortest Path (Undirected Graph): Implementation

```
int shortestDistance(string source, string destination){
        unordered_map<string, pair<string, int>> visited;
        queue<string> q;
        q.push(source);
        visited[source] = {"", 0};
        while (!q.empty())
        {
            string current = q.front();
            q.pop();
            if (current == destination)
                break;
            for (auto currentFriend : adjList.find(current)->second)
            {
                if (visited.find(currentFriend) == visited.end())
                {
                    q.push(currentFriend);
                    visited[currentFriend] = {currentFriend, visited[current].second + 1};
                }
            }
        }
        return visited[destination].second;
    }
```

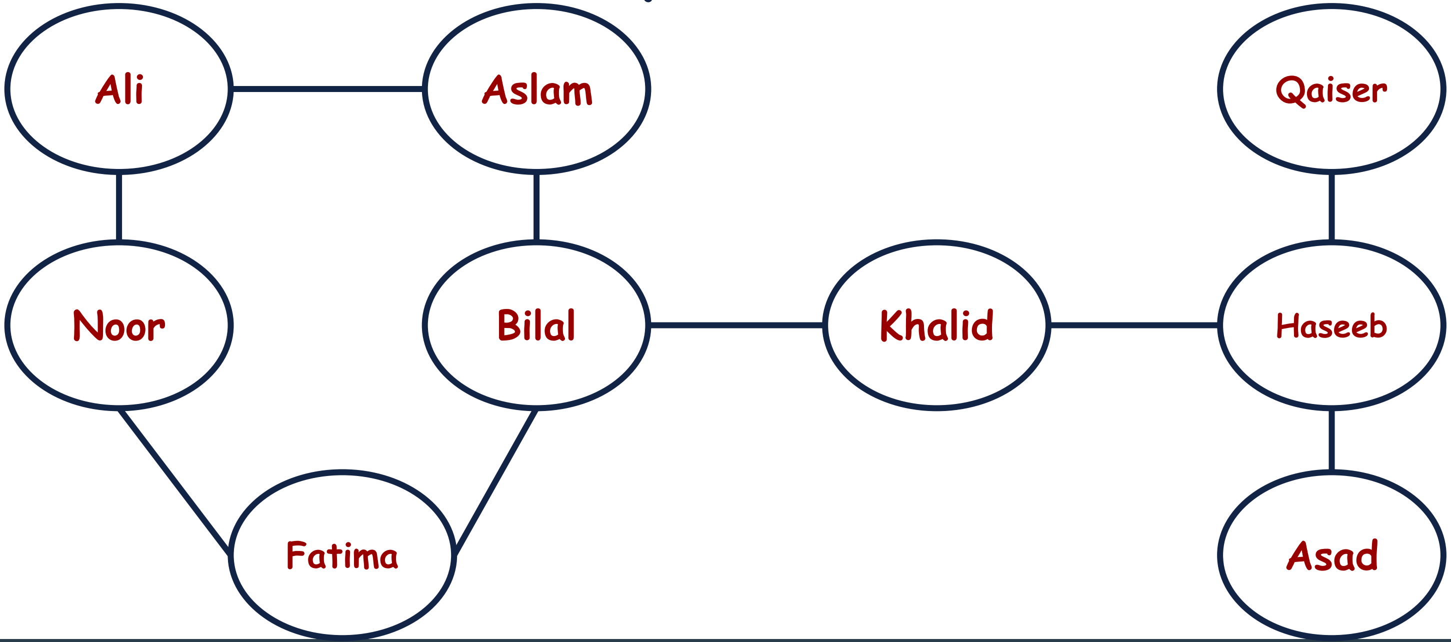
Graphs: Traversal

Now, I want to see if there exists a cycle in the graph. How can i do that?



Graphs: Is Cycle Exists

We can traverse the graph in DFS and if we reach at the node which is already visited then there is a cycle.



Cycle Detection (Undirected Graph): Implementation

Now, let's implement the solution.

```cpp
bool isCycleExist(string start)
    {
        unordered_map<string, string> visited;
        stack<string> s;
        s.push(start);
        visited[start] = "";
        while (!s.empty())
        {
            string current = s.top();
            s.pop();
            for (auto currentFriend : adjList.find(current)->second)
            {
                if (visited.find(currentFriend) == visited.end())
                {
                    s.push(currentFriend);
                    visited[currentFriend] = current;
                }
                else if (currentFriend != visited.find(current)->second)
                {
                    return true;
                }
            }
        }
        return false;
    }
```

Learning Objective

Students should be able to Traverse the graphs to solve real life problems.

Self Assessment

https://leetcode.com/problems/find-if-path-exists-in-graph/
https://leetcode.com/problems/keys-and-rooms/
https://leetcode.com/problems/all-paths-from-source-to-target/
https://leetcode.com/problems/find-all-possible-recipes-from-given-supplies/