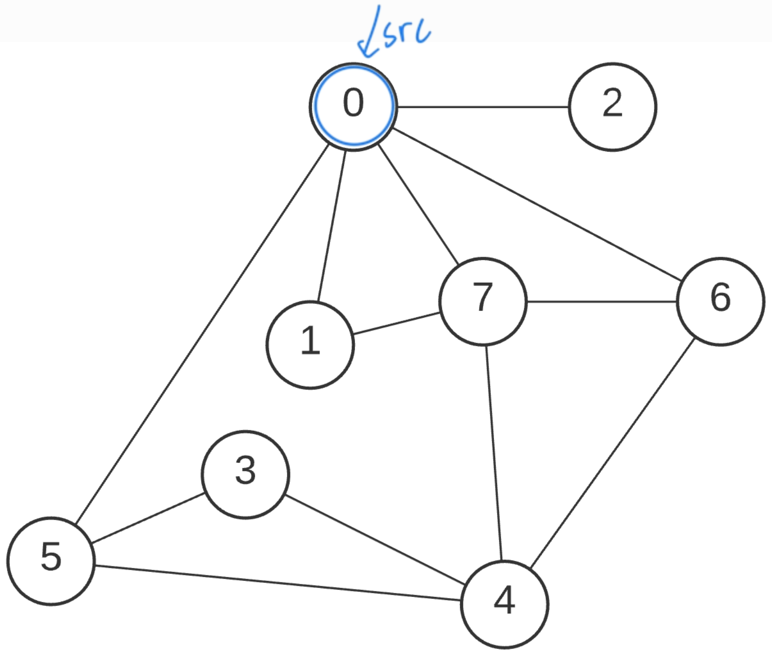
BFS DFS


```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```



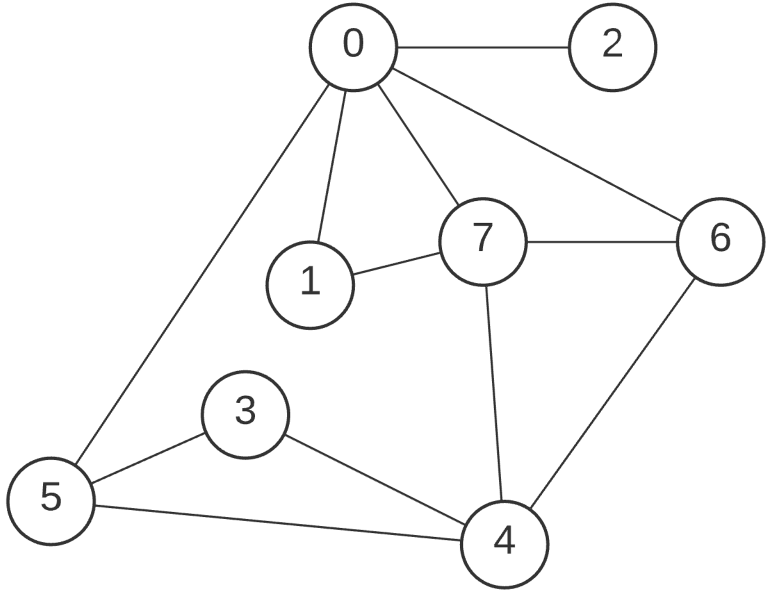
BFS DFS

T	F	F	F	F	F	F	F

-1	-1	-1	-1	-1	-1	-1	-1

Q = { 0 }

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```



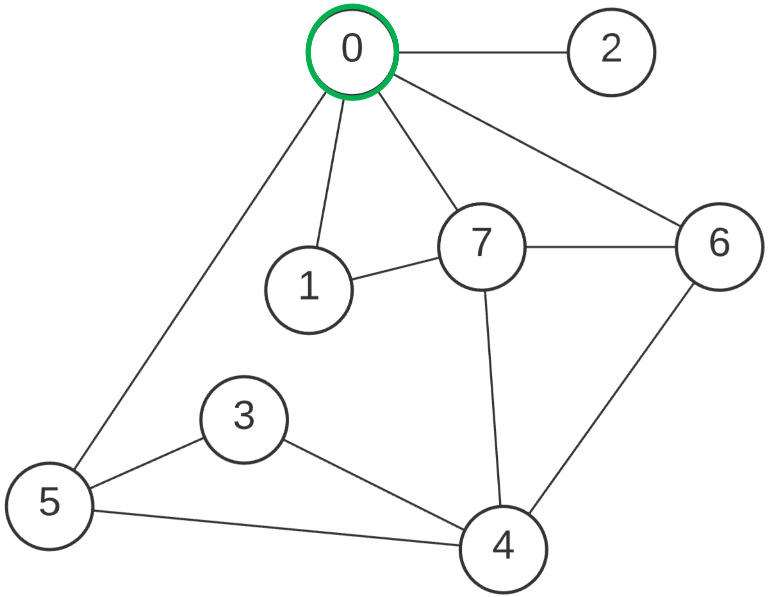
BFS DFS

T	T	T	F	F	T	T	T

-1	0	0	-1	-1	0	0	0

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```

Q = { $\underline{1}$, 2, 5, 6, 7 }



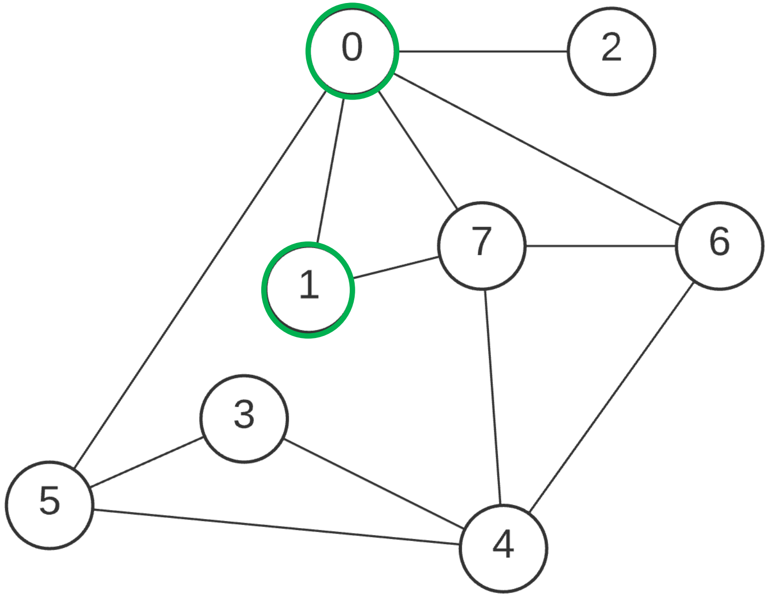
BFS DFS

T	T	T	F	F	T	T	T

-1	0	0	-1	-1	0	0	0

Q = { 2, 5, 6, 7 }

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to −1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```

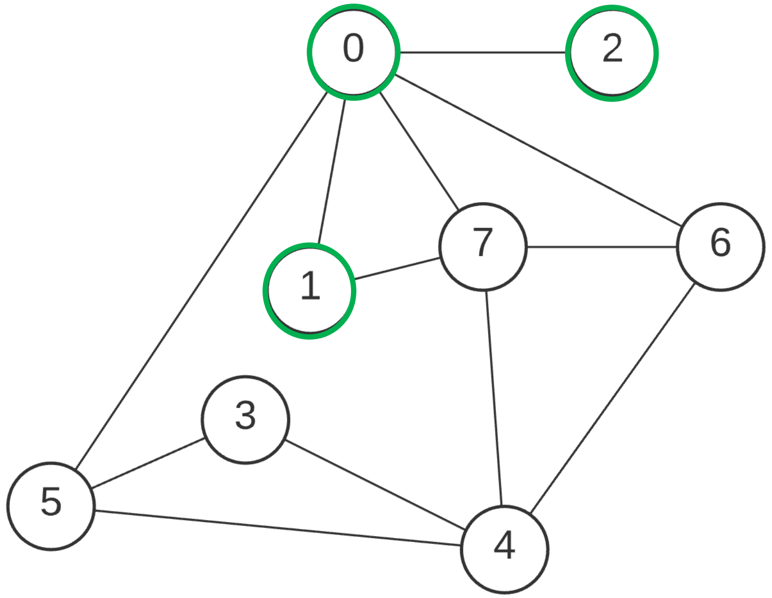


BFS DFS

T	T	T	F	F	T	T	T
-1	0	0	-1	-1	0	0	0

$Q = \{ \underline{5}, 6, 7 \}$

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to −1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```



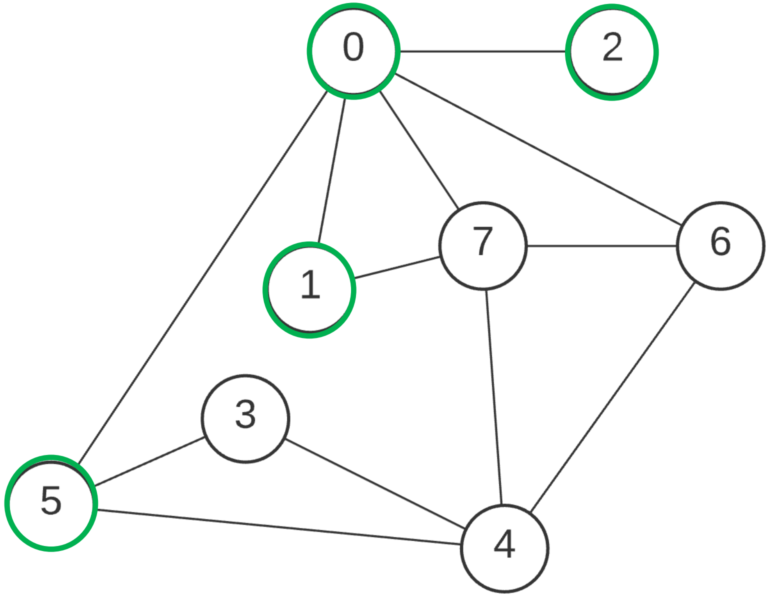
BFS DFS

T	T	T	T	T	T	T	T

-1	0	0	5	5	0	0	0

Q = { 6, 7, 3, 4 }

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to −1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```



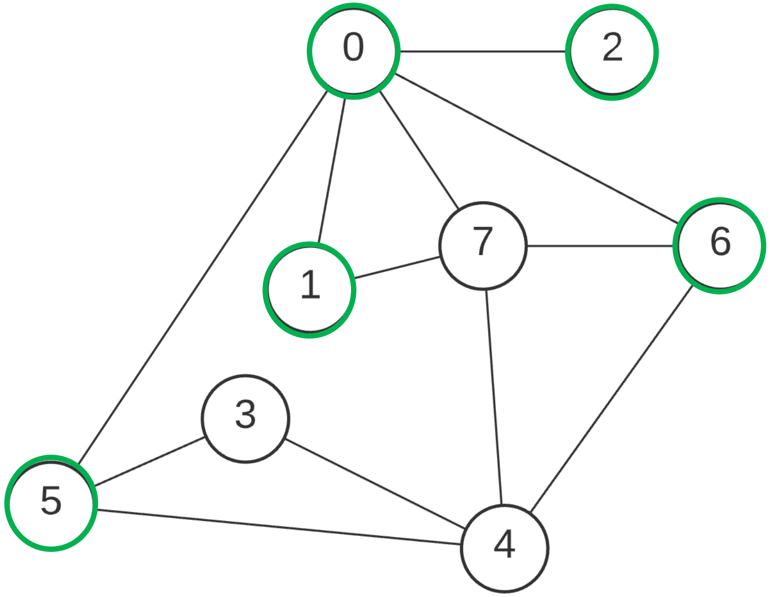
BFS DFS

T	T	T	T	T	T	T	T

-1	0	0	5	5	0	0	0

Q = { $\underline{7}$, 3, 4 }

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to −1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```



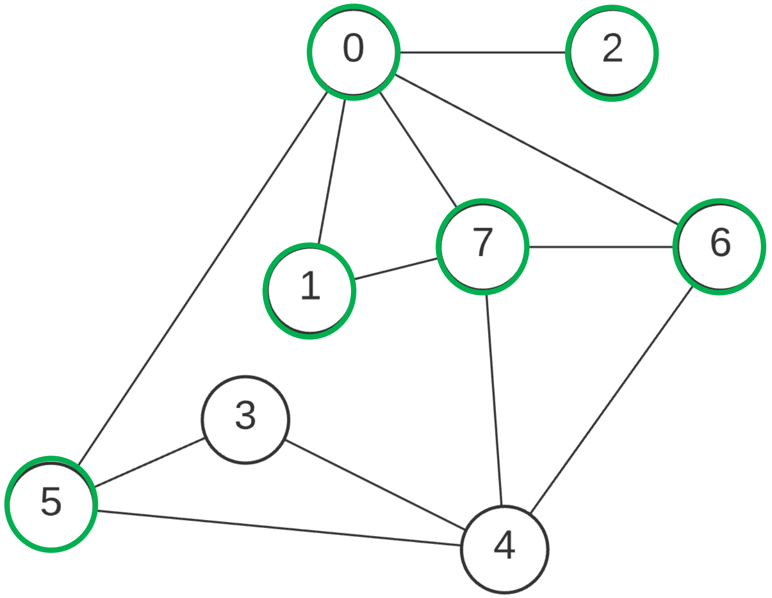
BFS DFS

T	T	T	T	T	T	T	T

-1	0	0	5	5	0	0	0

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```

Q = { 3, 4 }



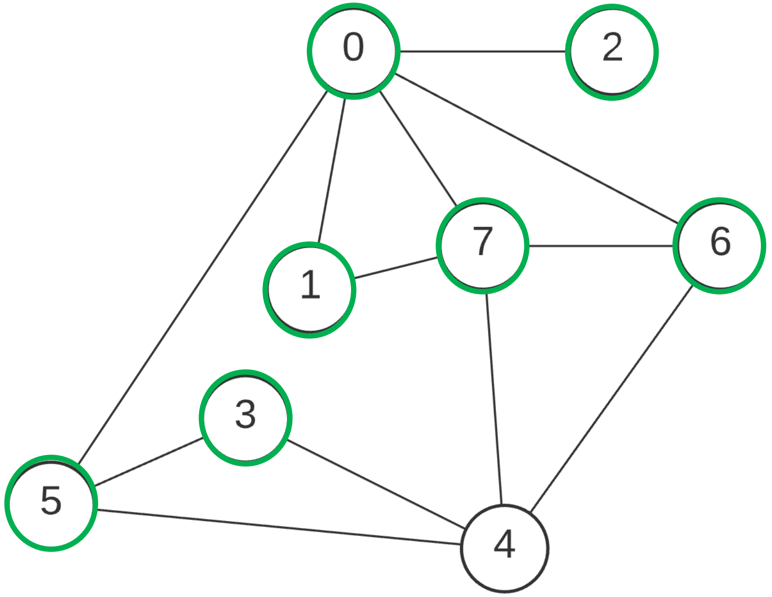
BFS DFS

T	T	T	T	T	T	T	T

-1	0	0	5	5	0	0	0

Q = { 4 }

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to −1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```



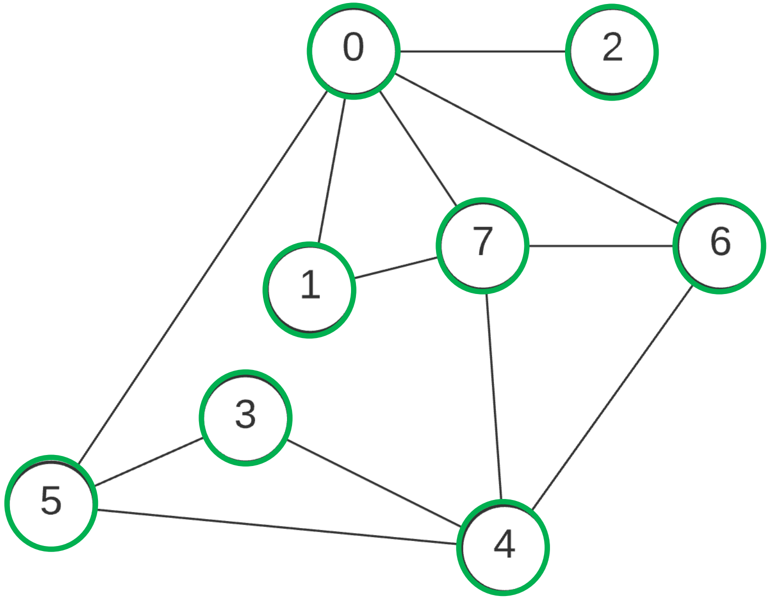
BFS DFS

T	T	T	T	T	T	T	T

-1	0	0	5	5	0	0	0

Q = { }

```
bfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create queue Q

    mark src as visited
    enqueue src into Q

    while Q is not empty:
        dequeue v from Q

        for each neighbour w of v:
            if w has not been visited:
                mark w as visited
                set predecessor of w to v
                enqueue w into Q
```



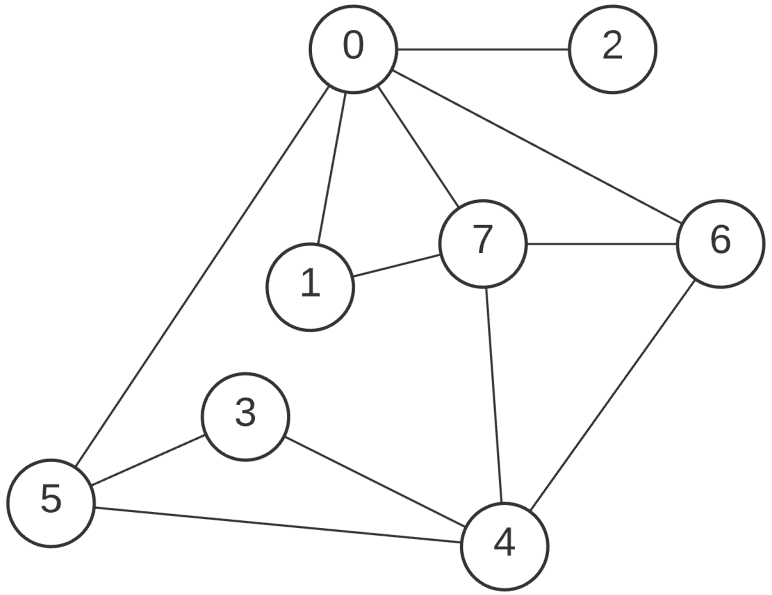
BFS DFS

F	F	F	F	F	F	F	F

-1	-1	-1	-1	-1	-1	-1	-1

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to −1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```

S = { 0 }



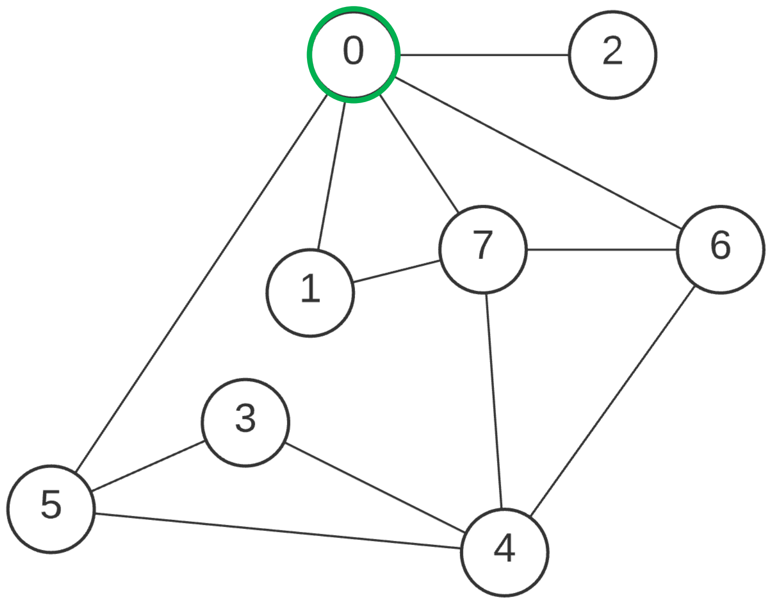
BFS DFS

T	F	F	F	F	F	F	F

-1	0	0	-1	-1	0	0	0

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```

S = { 7, 6, 5, 2, 1 }



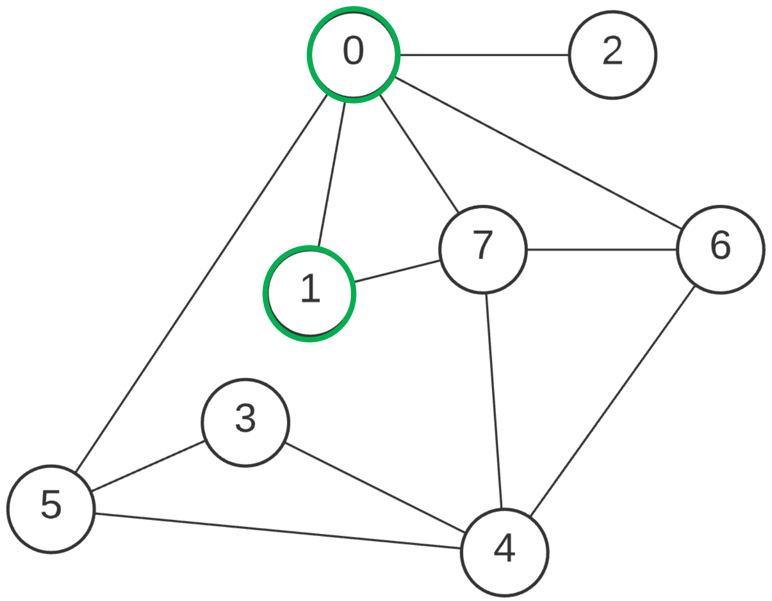
BFS DFS

T	T	F	F	F	F	F	F

-1	0	0	-1	-1	0	0	1

S = { 7, 6, 5, 2, 7 }

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```

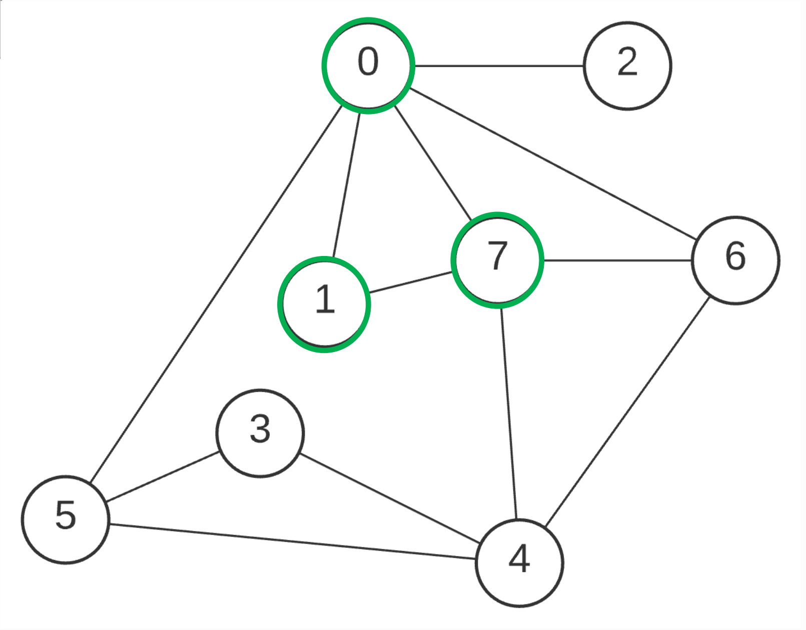


BFS DFS

T	T	F	F	F	F	F	T
-1	0	0	-1	7	0	7	1

S = { 7, 6, 5, 2, 6, 4 }

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```



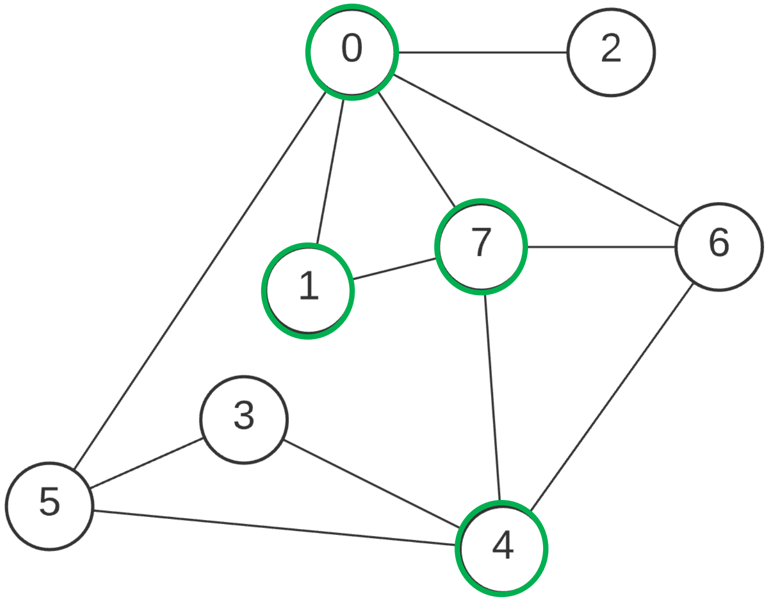
BFS DFS

T	T	F	F	T	F	F	T

-1	0	0	4	7	4	4	1

S = { 7, 6, 5, 2, 6, 6, 5, 3 }

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```

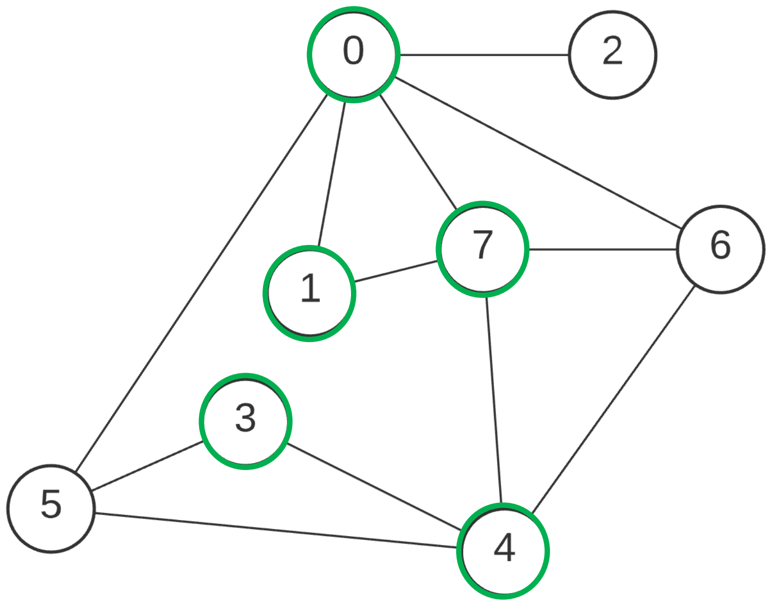


BFS DFS

T	T	F	T	T	F	F	T
-1	0	0	4	7	3	4	1

S = { 7, 6, 5, 2, 6, 6, $\underline{5}$ }

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```



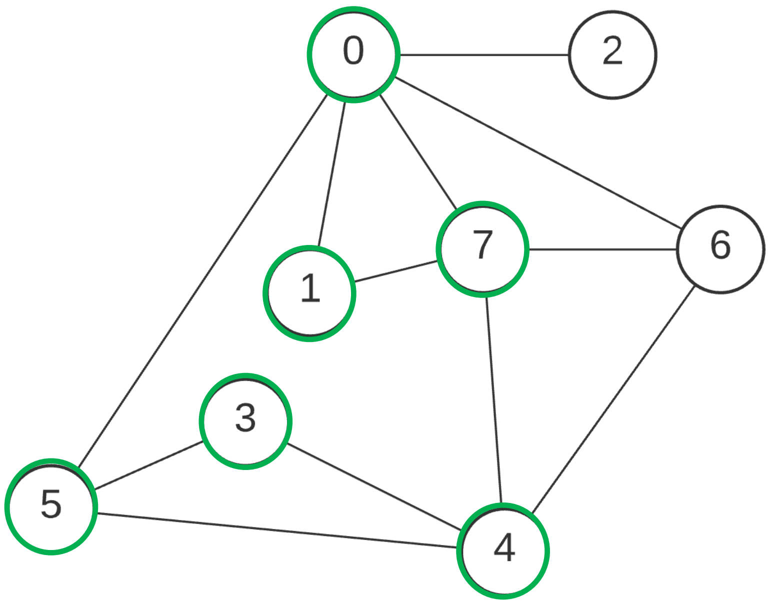
BFS DFS

T	T	F	T	T	T	F	T

-1	0	0	4	7	3	4	1

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```

S = { 7, 6, 5, 2, 6, 6 }



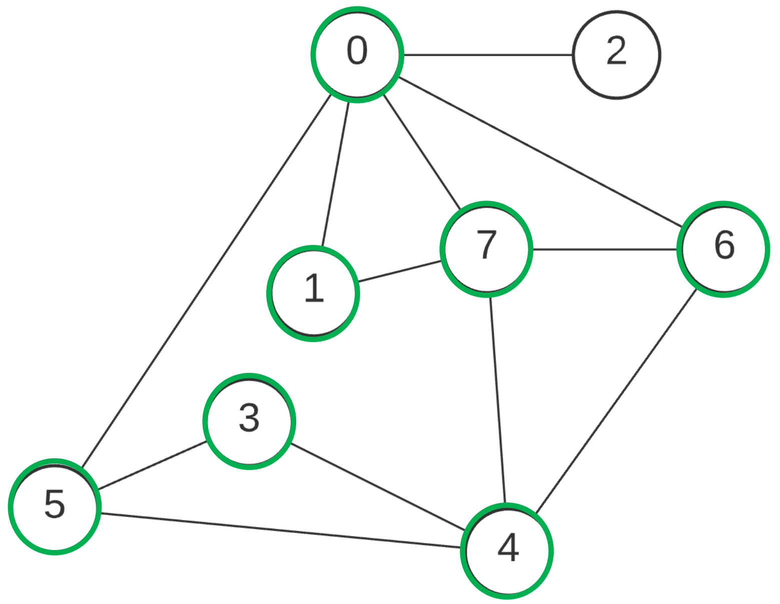
BFS DFS

T	T	F	T	T	T	T	T

-1	0	0	4	7	3	4	1

S = { 7, 6, 5, 2, 6 }

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```

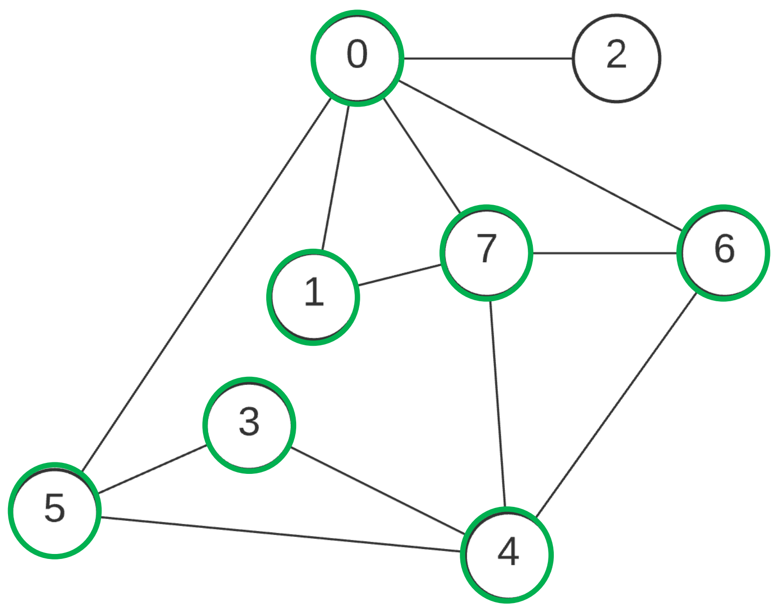


BFS DFS

T	T	F	T	T	T	T	T
-1	0	0	4	7	3	4	1

S = { 7, 6, 5, 2 }

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```



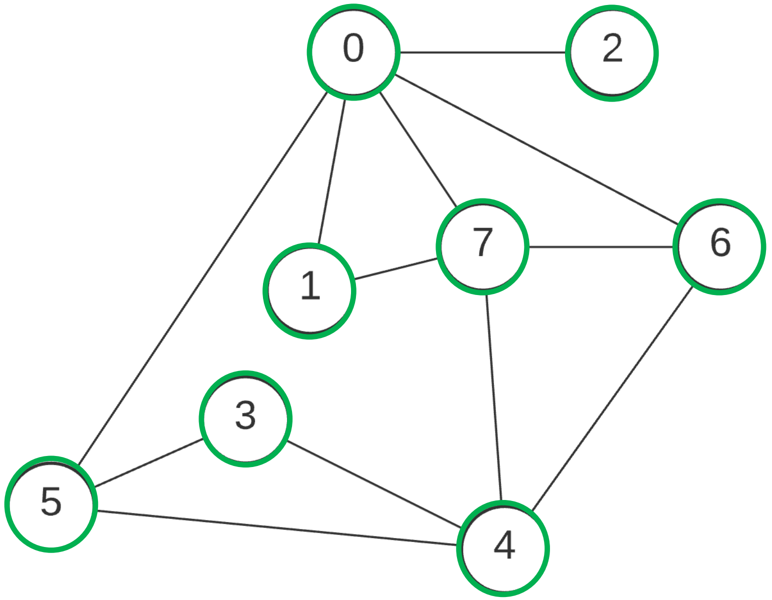
BFS DFS

T	T	T	T	T	T	T	T

-1	0	0	4	7	3	4	1

S = { 7, 6, <u>5</u> }

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```



BFS DFS

T	T	T	T	T	T	T	T
-1	0	0	4	7	3	4	1

```
dfs(G, src):
    Input: graph G, vertex src

    create visited array, initialised to false
    create predecessor array, initialised to -1
    create stack S

    push src onto S

    while S is not empty:
        pop v from S
        if v has been visited:
            continue

        mark v as visited

        for each neighbour w of v:
            if w has not been visited:
                set predecessor of w to v
                push w onto S
```

S = { }