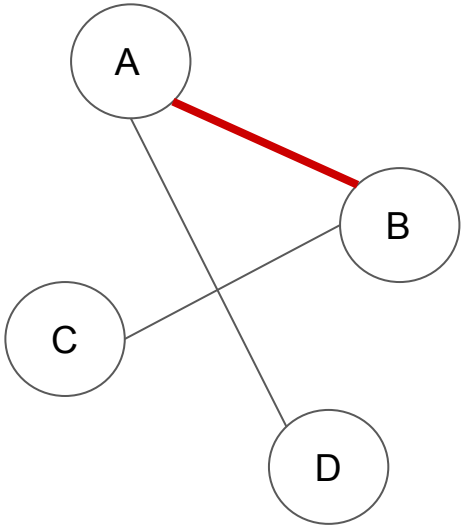
Announcements:

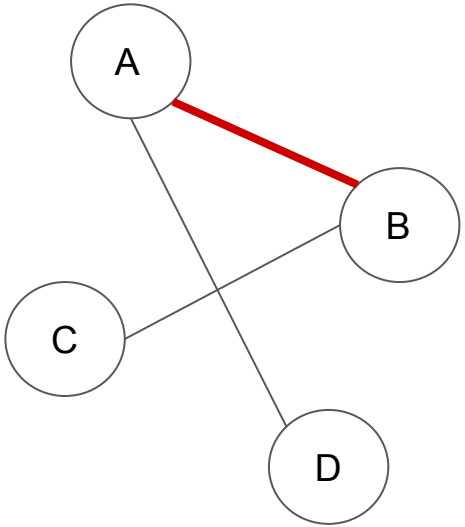
Submit all recitations from now on Moodle
Grade is not for attendance but only for submission



How do I represent it?

Adjacency Matrix

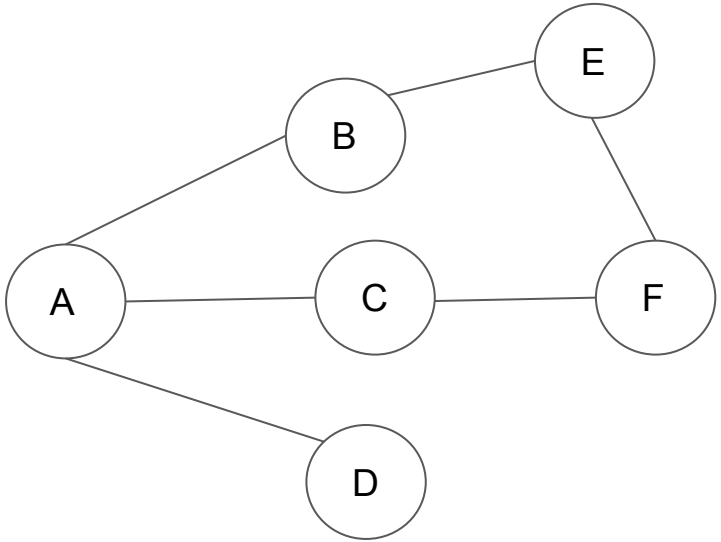
	A	B	C	D
A	0	**1**	0	1
B	1	0	1	0
C	0	1	0	0
D	1	0	0	0



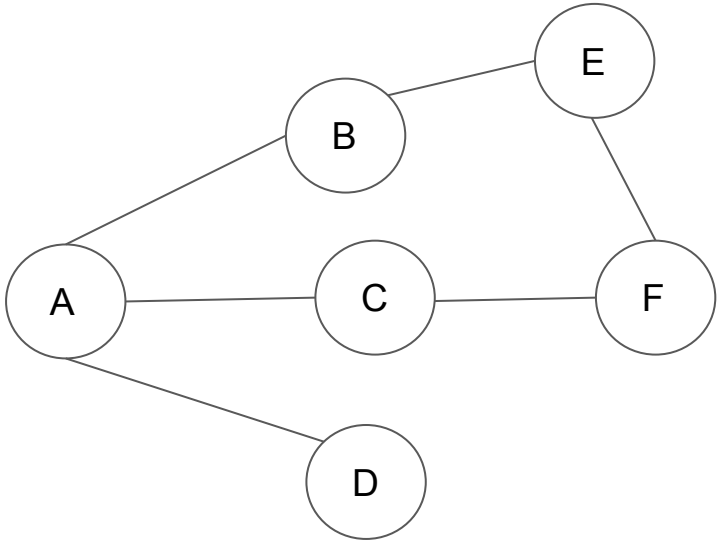
How do I represent it?

Adjacency List

A → **(B,** D)
B → **(A**, C)
C → (B)
D → (A

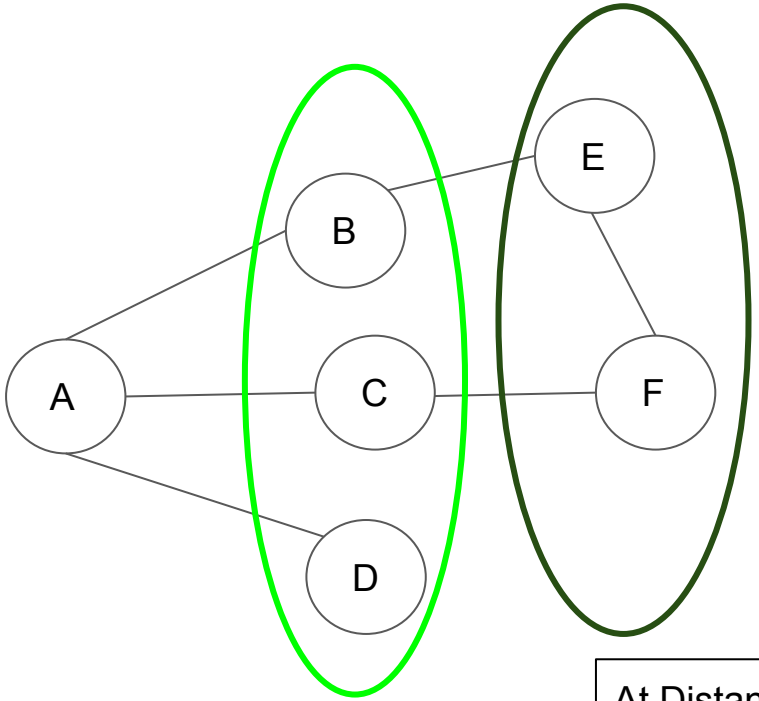


Possible paths from A to F:



Possible paths from A to F:

- A, B, E, F
- A,C,F - Shorter Path.

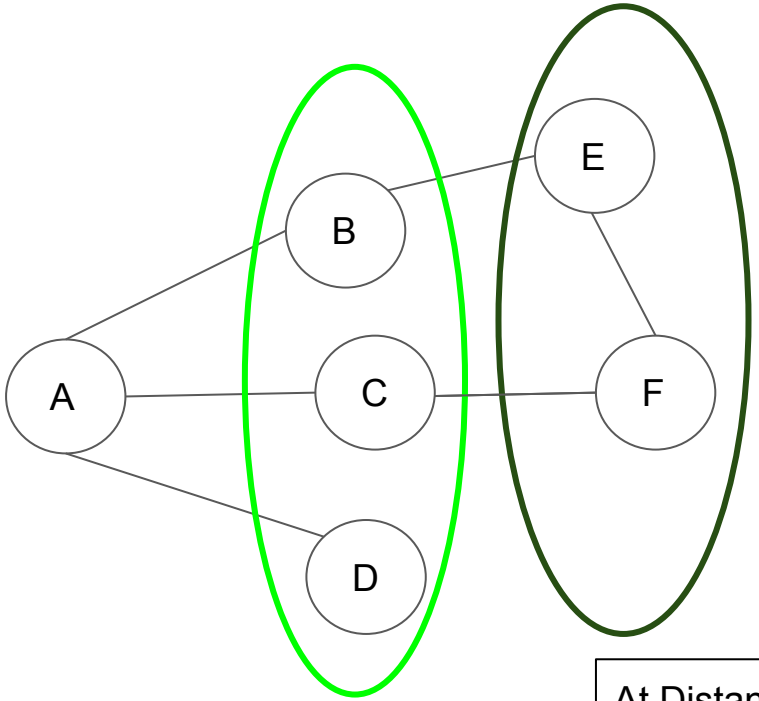


At Distance: 2

At Distance: 1

Possible paths from A to F:

- A, B, E, F
- A,C,F - Shorter Path.

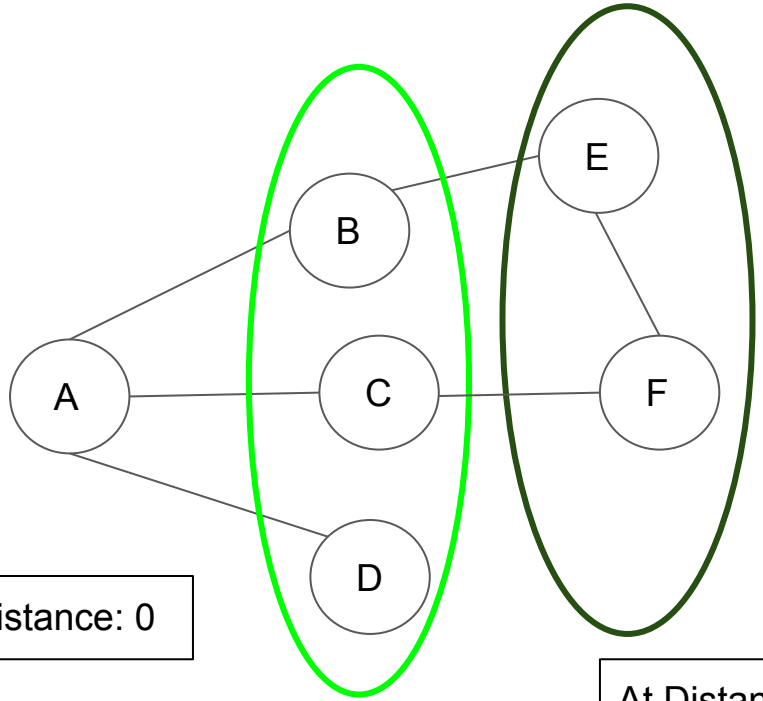


At Distance: 2

At Distance: 1

Possible paths from A to F:

- A, B, E, F
- A,C,F - Shorter Path.



At Distance: 0

At Distance: 1

At Distance: 2

Breadth First Traversal goes level by level

What is the output for this traversal?

```
BFS (G, s)
    let Q be queue.
    Q.enqueue( s )

    mark s as visited.
    while ( Q is not empty)
        v  =  Q.dequeue( ) # A

        for all neighbours w of v in Graph G
            if w is not visited
                    Q.enqueue( w )
                    mark w as visited.
```

Queue Initially:
(A)

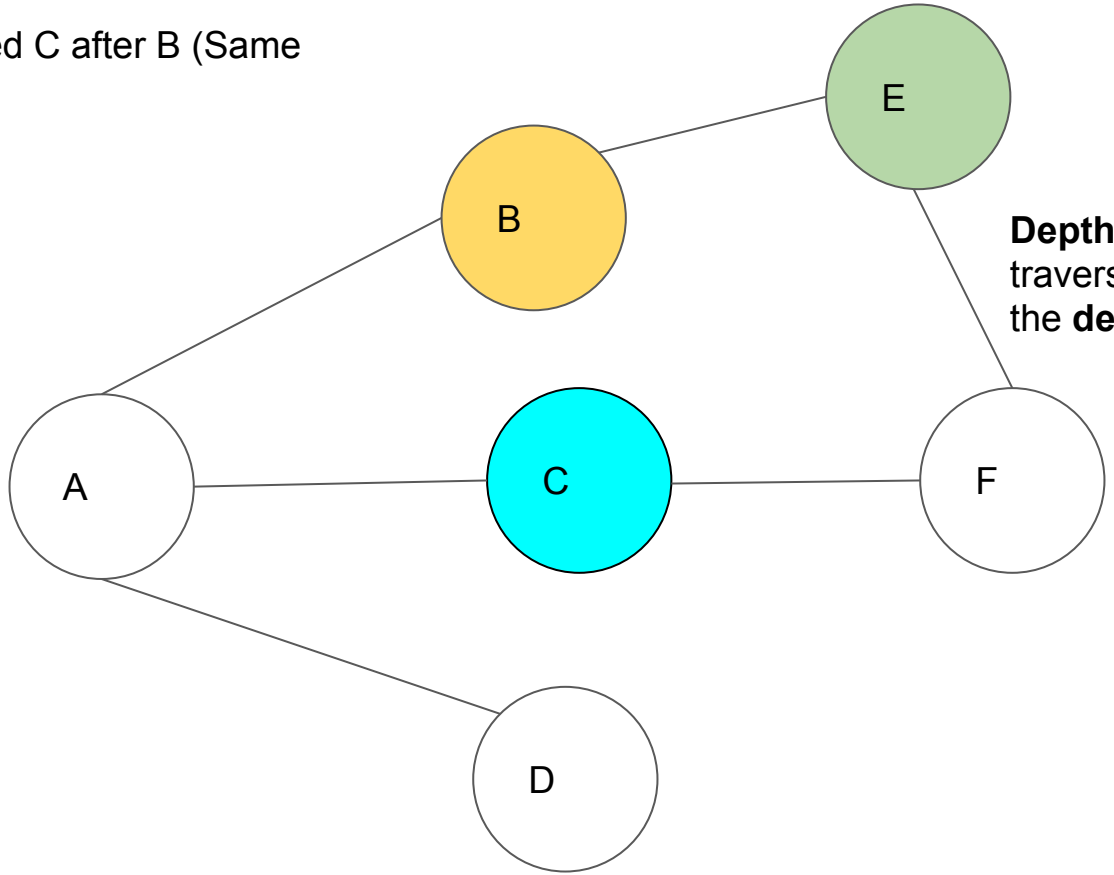
Queue After first iteration:
(B,C,D)

Source:
https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

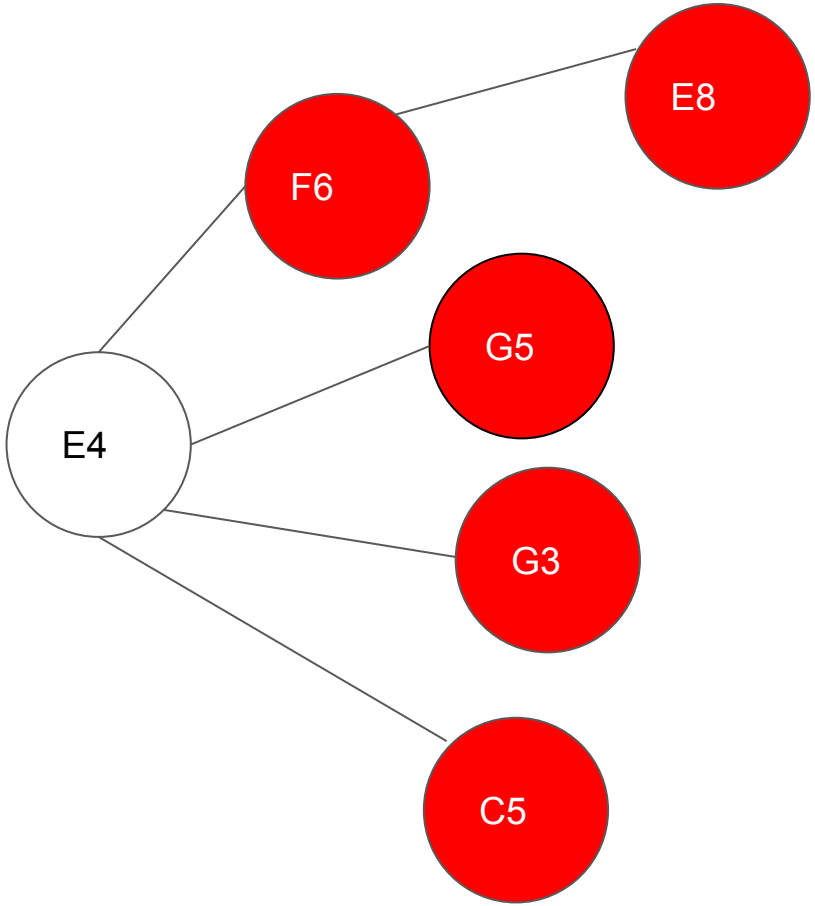
```
#ifndef GRAPH_H
#define GRAPH_H
#include<vector>
#include<iostream>

struct vertex;

struct adjVertex{
    vertex *v;
};

struct vertex{
    int key;
    bool visited = false;
    std::vector<adjVertex> adj;
};

class Graph
{
    public:
        void addEdge(int v1, int v2);
        void addVertex(int v);
        void printGraph();

    private:
        std::vector<vertex*> vertices;

};

#endif
```

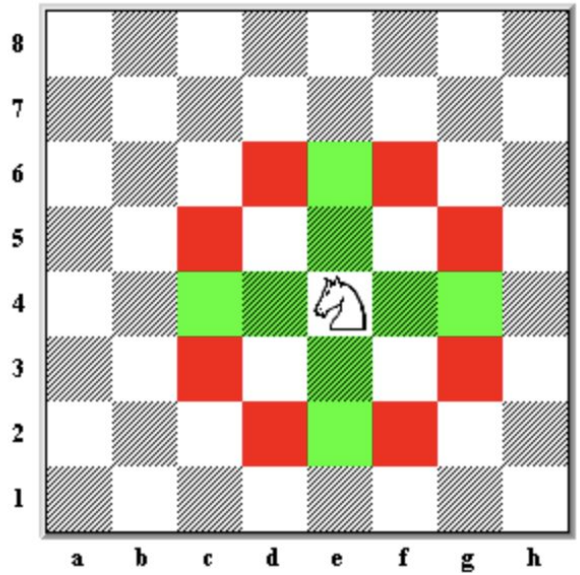
Vertices = {vertex(A),vertex(B),....}
vertex(A) -- key : A
Visited = false
Adjlist = {adjVertex(B)
adjVertex(C)
adjVertex(D)}

BFS traversed C after B (Same breadth)

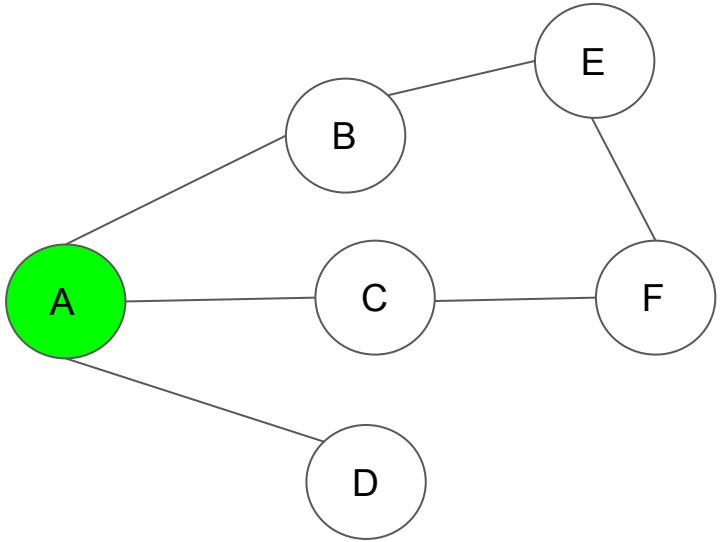


Depth First Search traverses E after B (along the **depth**) instead of C

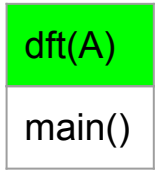

E8
F6
G5
E4
G3
C5


8
7
6
5
4
3
2
1
a
b
c
d
e
f
g
h

Depth First Search/Traversal

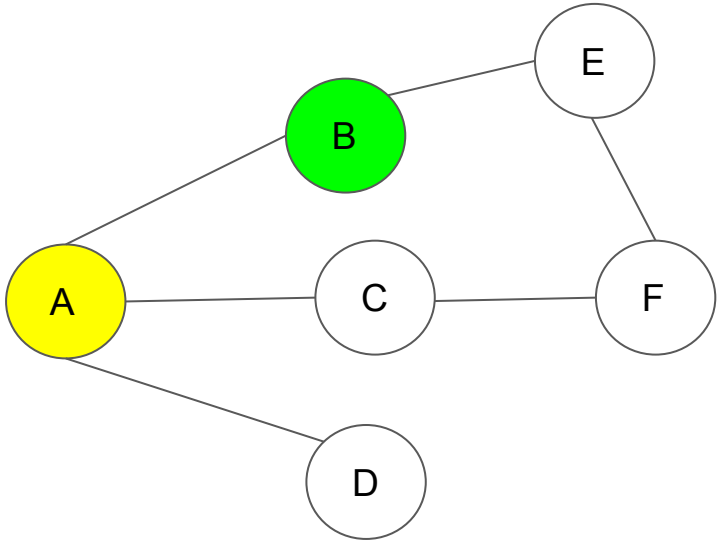


dft(A)




```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

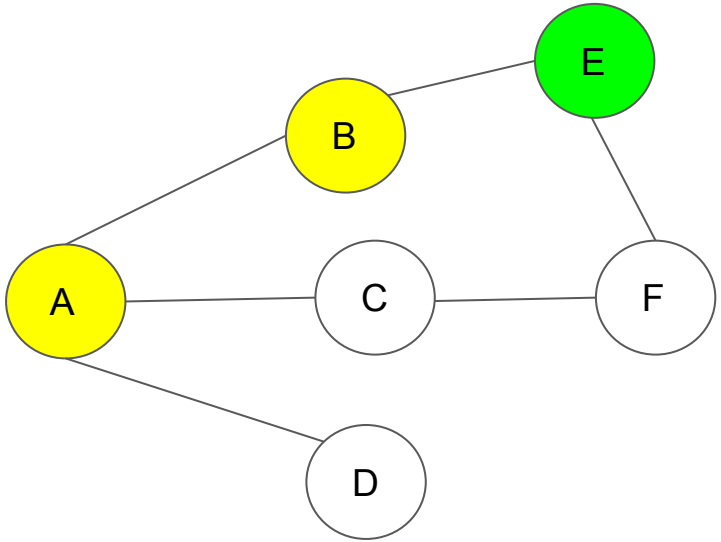
Depth First Search/Traversal



dft(B)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

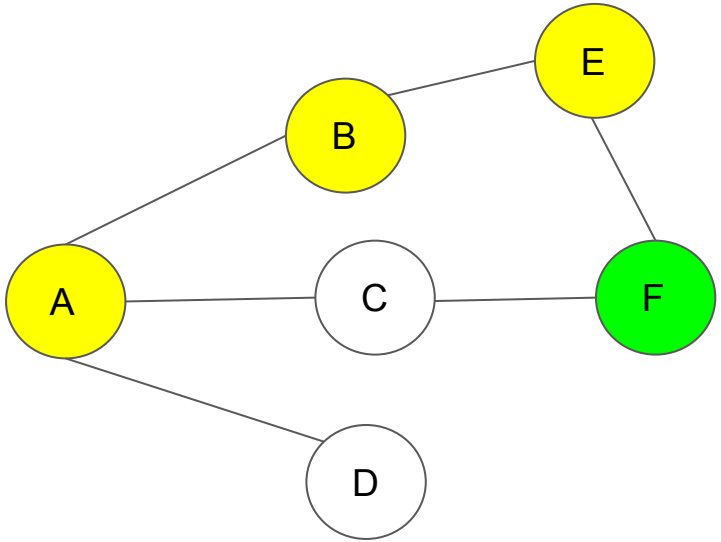
Depth First Search/Traversal



dft(E)
dft(B)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

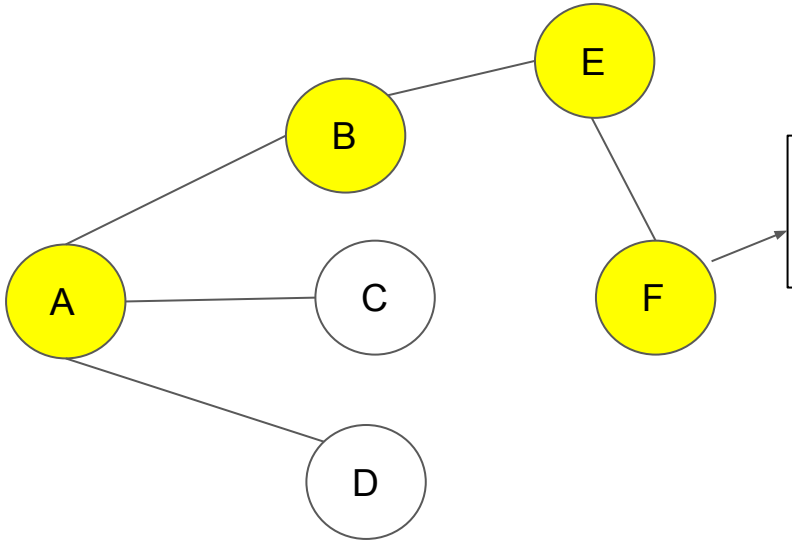
Depth First Search/Traversal



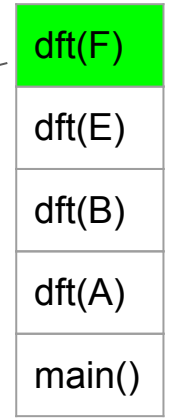
dft(F)
dft(E)
dft(B)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal

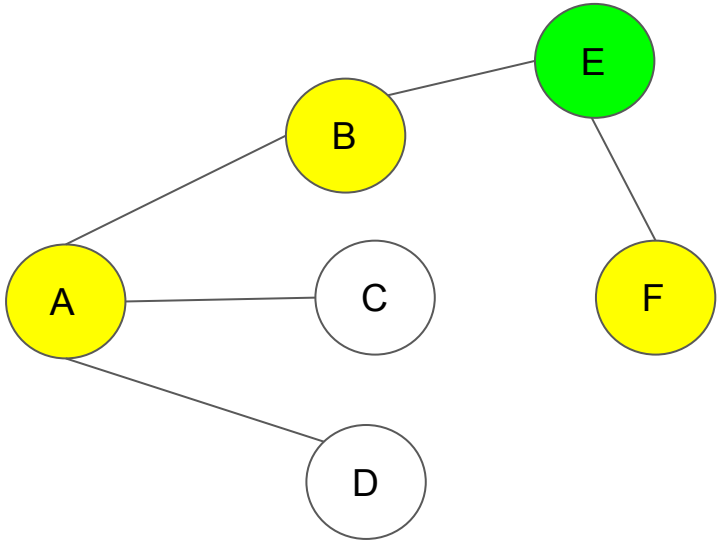


Since F doesn't have any unvisited adjacent vertices, it returns and pops from the stack




```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal

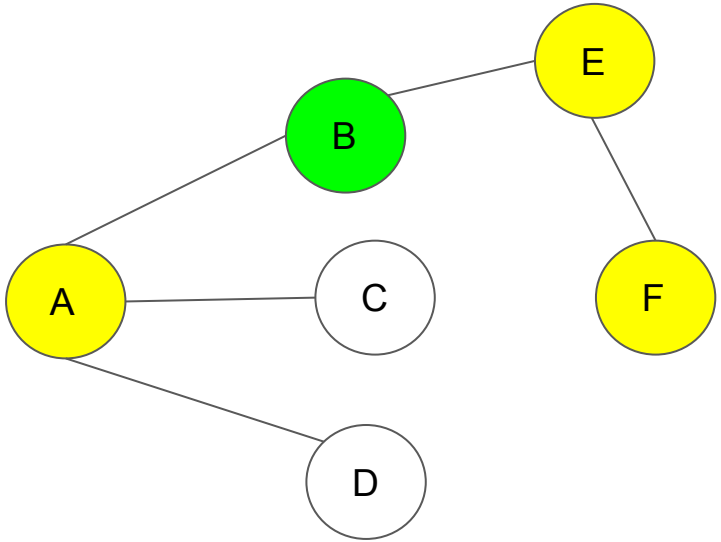


E pops from the stack as well

dft(E)
dft(B)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal

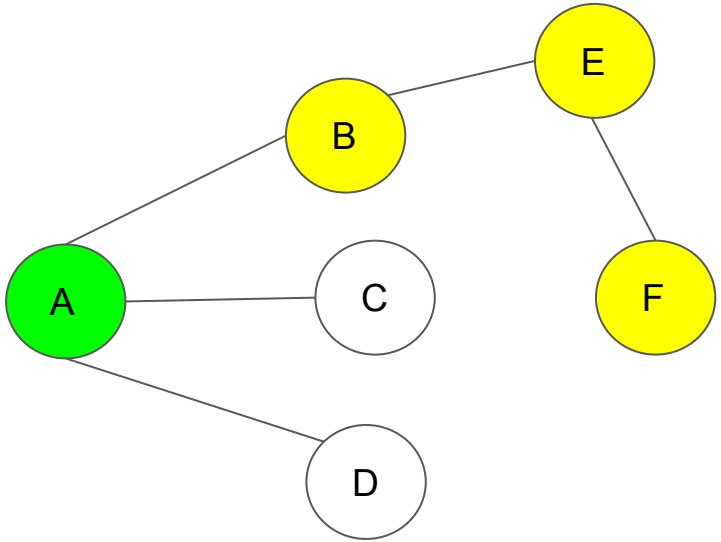


B pops from the stack as well

dft(B)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal



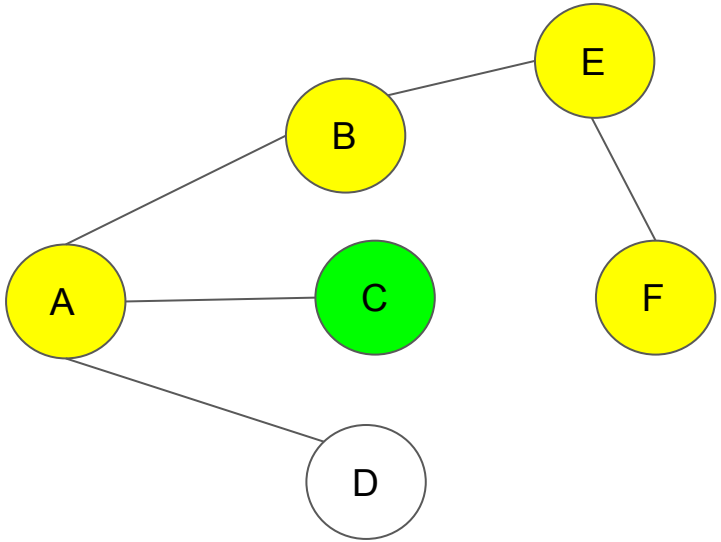
We are back at A. The next adjacent vertex is C

dft(A)

main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal

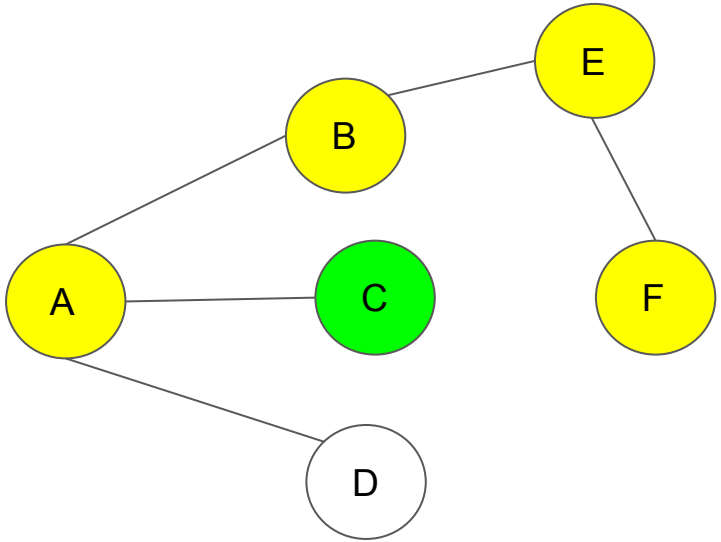


DFT is called on C

dft(C)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal

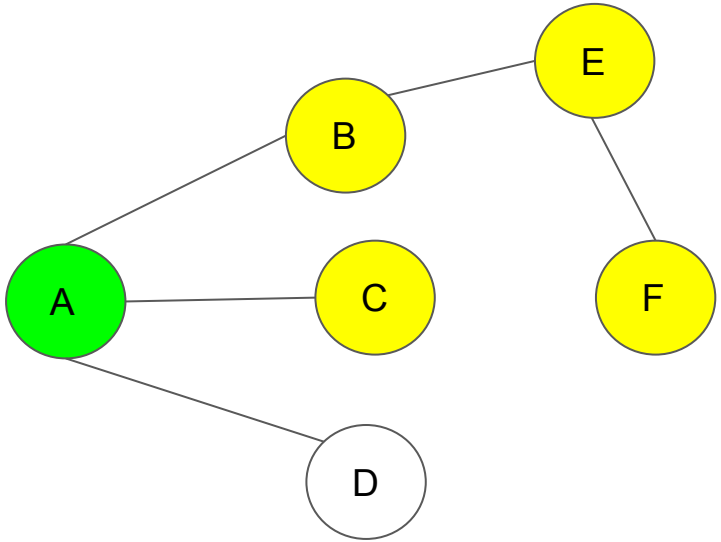


C has no unvisited adjacent vertices, hence it pops

dft(C)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal



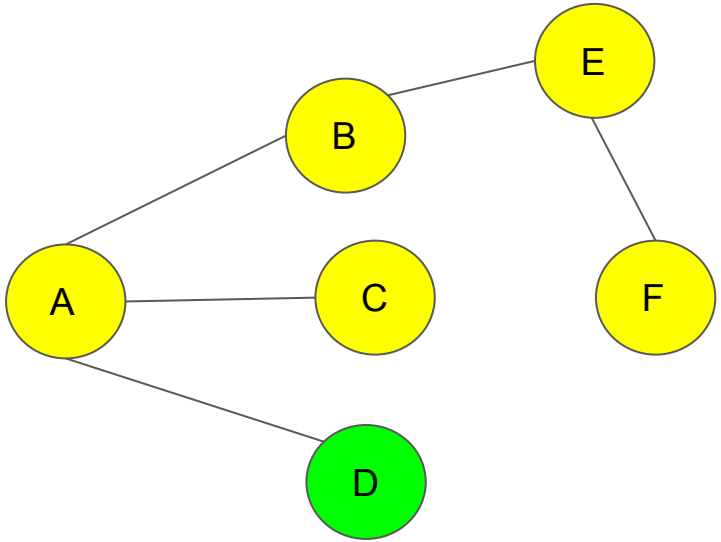
We are back at A. The next vertex is D.

dft(A)

main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal

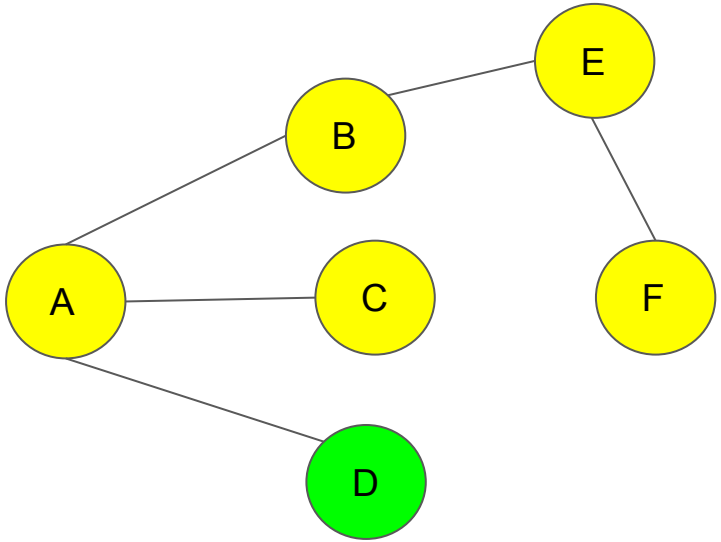


DFT is called on D

dft(D)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal

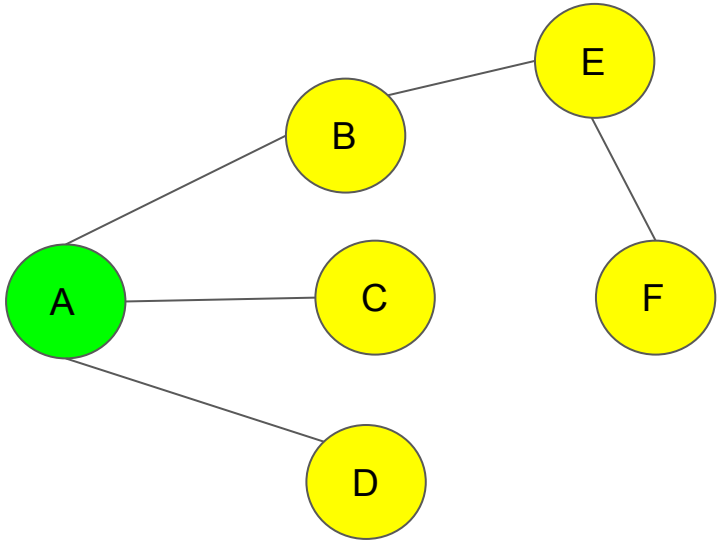


D is now popped since it has no unvisited adjacent vertices.

dft(D)
dft(A)
main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal



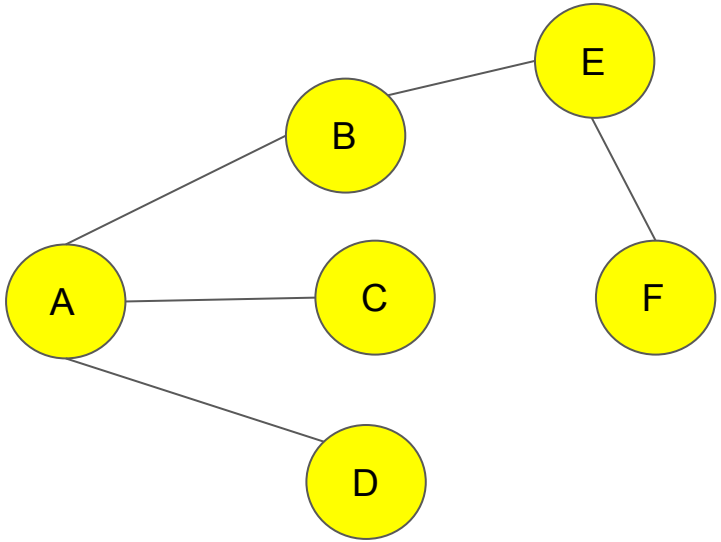
A is also popped from stack because it has no other unvisited adjacent vertices..

dft(A)

main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```

Depth First Search/Traversal



Main is popped from the stack now.

main()

```
dft (vertex *n)
{
        n->visited = true;
        for each adjacent vertex,v which is not visited:
                dft(v)
 }
```