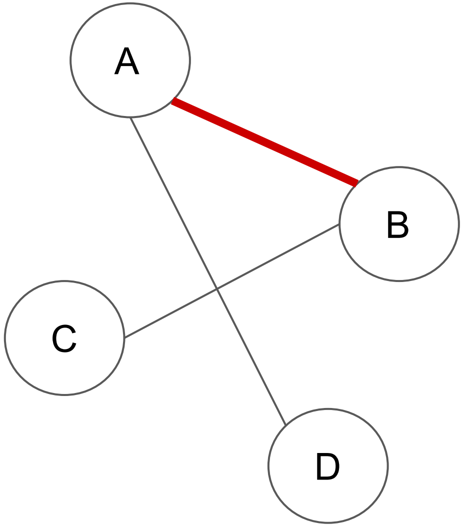
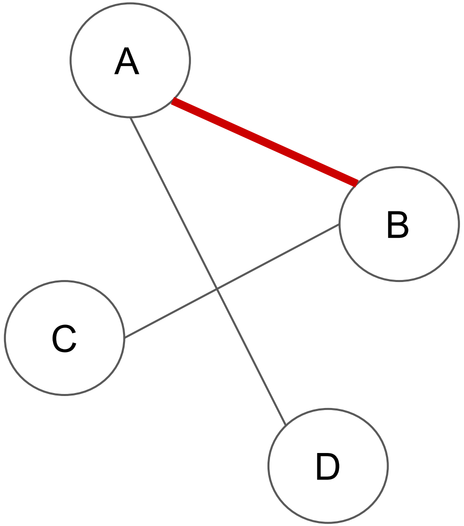
How do I represent it?

Adjacency Matrix



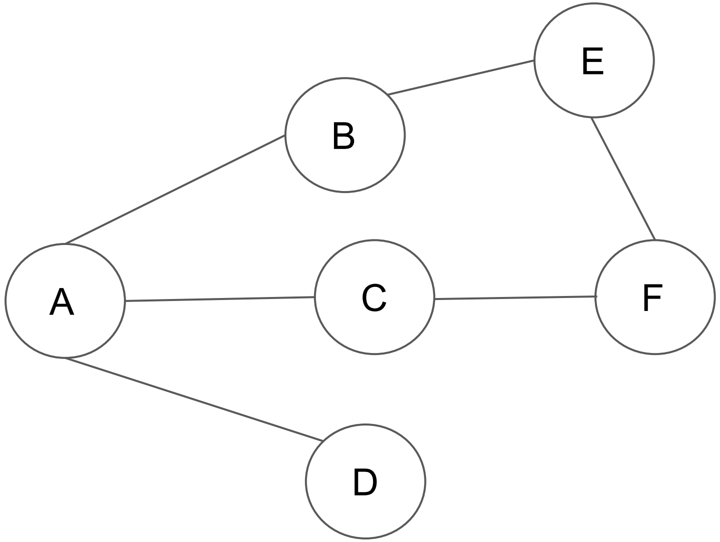
	A	B	C	D
A	0	1	0	1
B	1	0	1	0
C	0	1	0	0
D	1	0	0	0



How do I represent it?

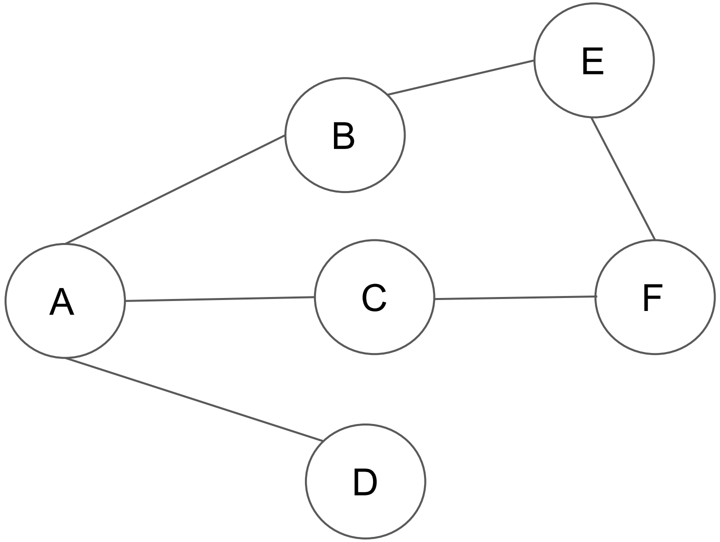
Adjacency List

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



Breadth First Traversal goes level by level

Output:
A,B,C,D,E,F



Depth First Traversal

Output:
A,B,E,F,C,D

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

BFS	DFS
BFS starts traversal from the root node and visits nodes in a level by level manner (i.e., visiting the ones closest to the root first).	DFS starts the traversal from the root node and visits nodes as far as possible from the root node (i.e., depth wise).
Usually implemented using a **queue** data structure.	Usually implemented using a **stack** data structure./ **recursion**
Optimal for finding the shortest distance.	Not optimal for finding the shortest distance.
Used for finding the shortest path between two nodes, testing if a graph is bipartite, finding all connected components in a graph, etc.	Used for topological sorting, solving problems that require graph backtracking, detecting cycles in a graph, finding paths between two nodes, etc.

Source: https://www.educative.io/edpresso/dfs-vs-bfs

```cpp
struct vertex;

struct adjVertex{
    vertex *v;
};

struct vertex{
    int key;
    bool visited = false;
    int distance = 0;
    std::vector<adjVertex> adj;
};
```

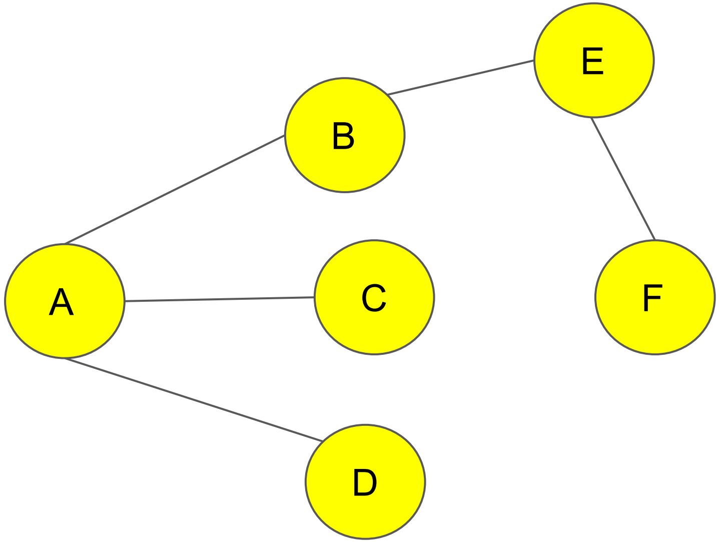
```cpp
class Graph
{
    public:
        void addEdge(int v1, int v2);
        void addVertex(int v);
        bool isBridge(int key1, int key2);
        void removeEdge(int key1, int key2);
        void DFTraversal(vertex *n);
        void setAllVerticesUnvisited();
        void printGraph();


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

};
```

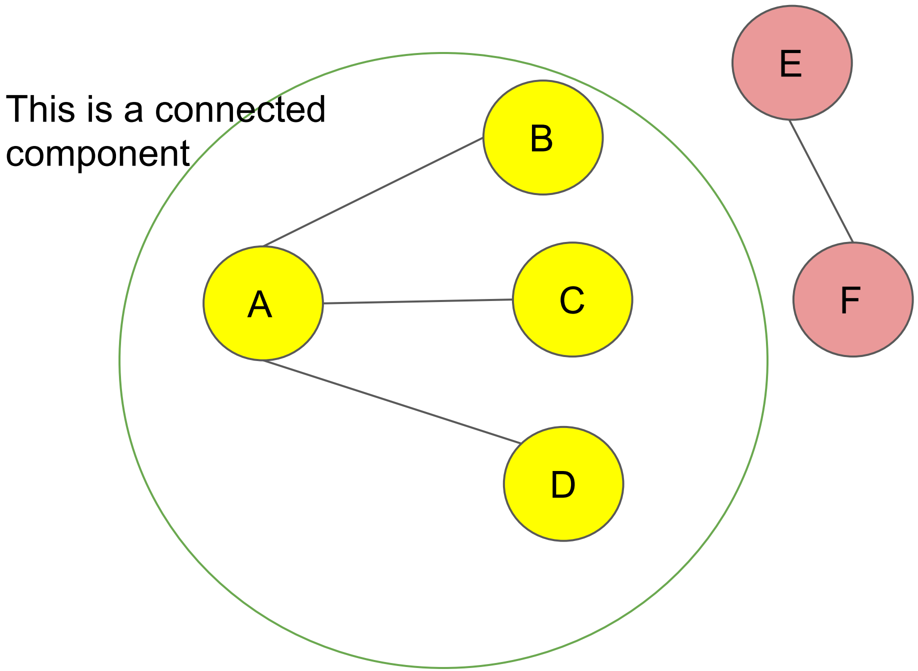
Functions to implement in assignment:

- addVertex
- addEdge
- displayEdges
- breadthFirstTraverse
- getConnectedComponents



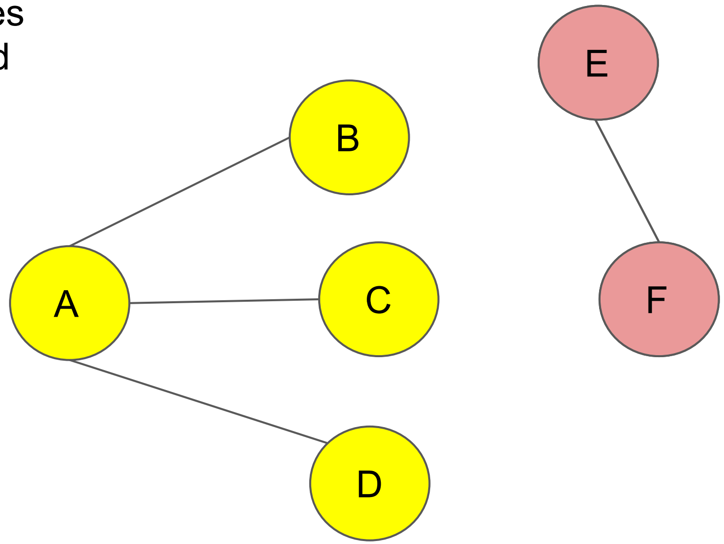
We traversed the entire graph using one call of the DFS

Connected Component

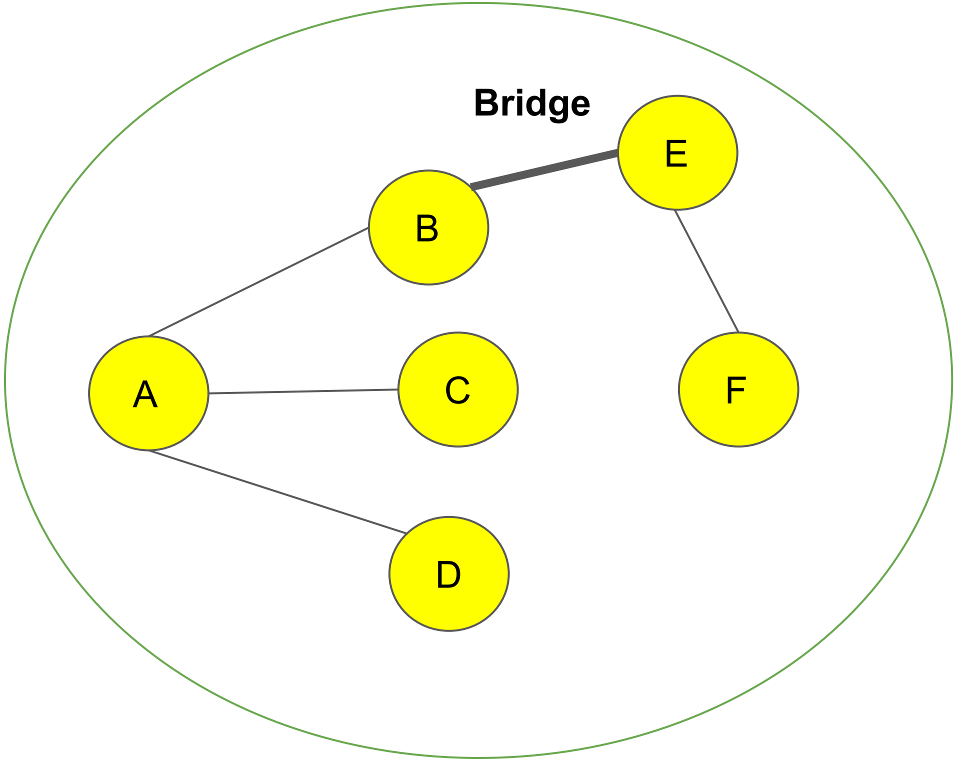


If edge between B and E was removed, will we still be able to traverse E and F through just one call of DFS?

All the yellow vertices constitute connected component - I

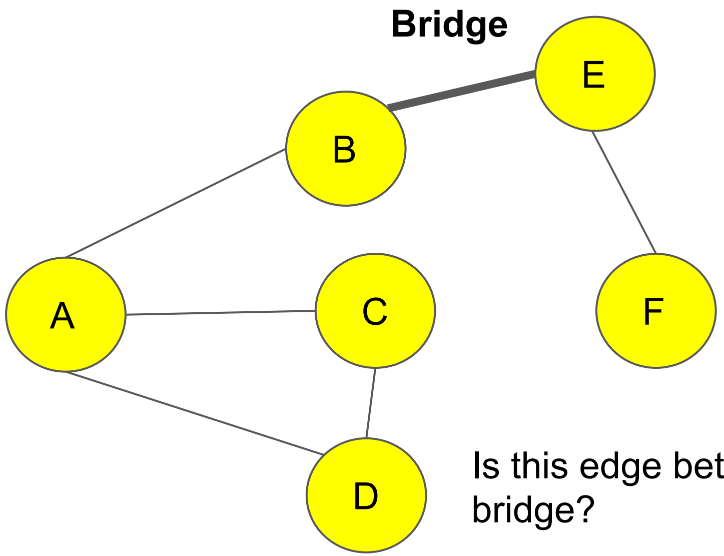


All the pink vertices constitute connected component - II



There is just one connected component now.

Removing a bridge, increases the number of connected components

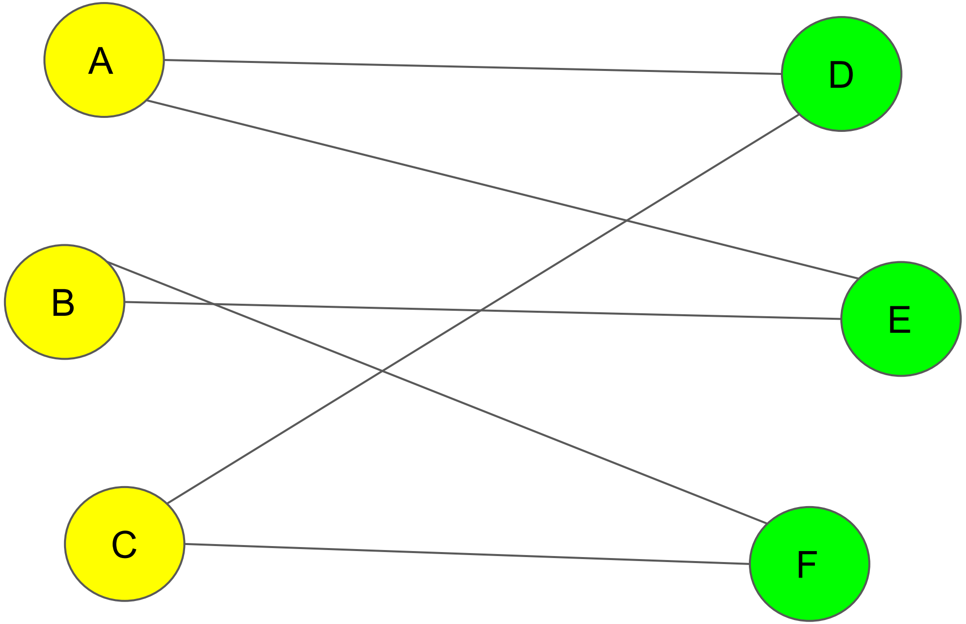


Is this edge between C and D a bridge?

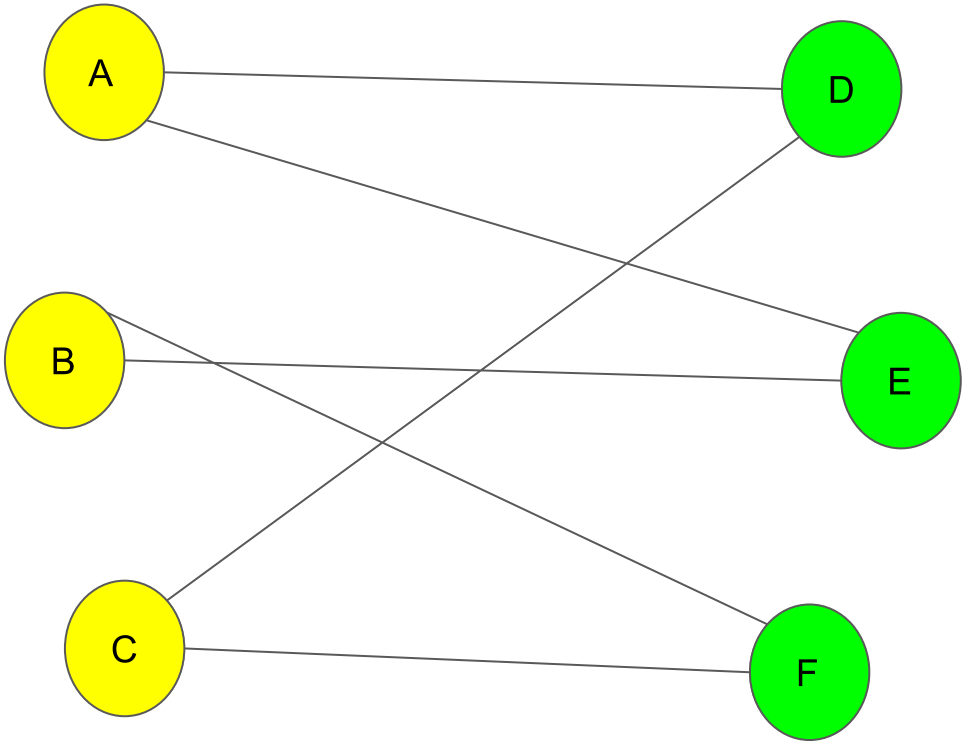
Check if edge is a bridge:

- Find connected components in graph.
- Remove edge between two vertices.
- Find connected components in the new graph(with edge removed).
- If the number of connected components increases, the edge is a bridge.

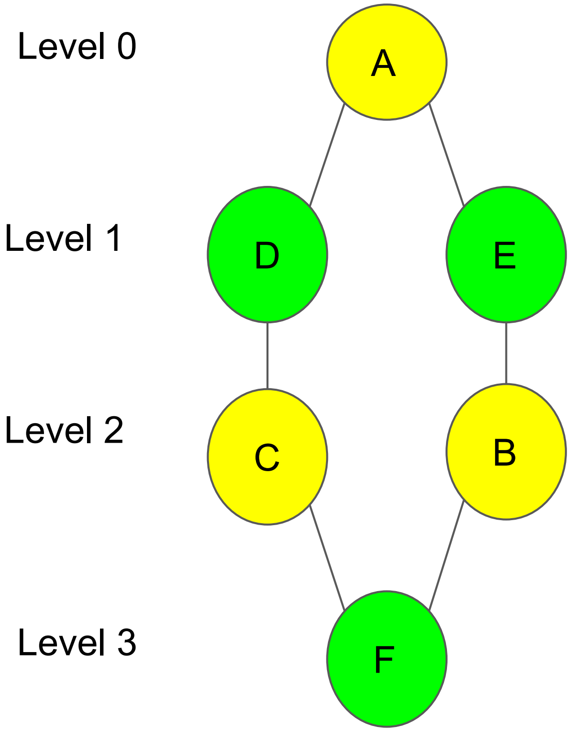
Bipartite graph



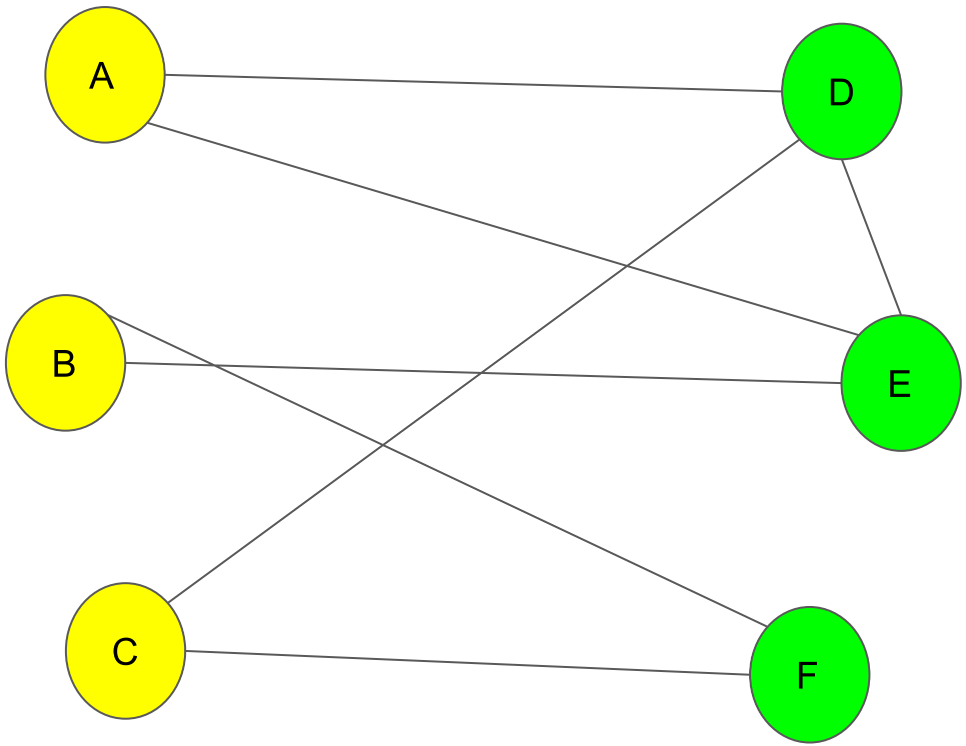
Start BFT from A



Breadth First Traversal(Level Order)



Start BFT from A



Breadth First Traversal(Level Order)