SC1007
Data Structures and Algorithms

Graph

Dr Liu Siyuan (syliu@ntu.edu.sg)

N4-02C-117a

Office Hour: Mon & Wed 4-5pm

Overview

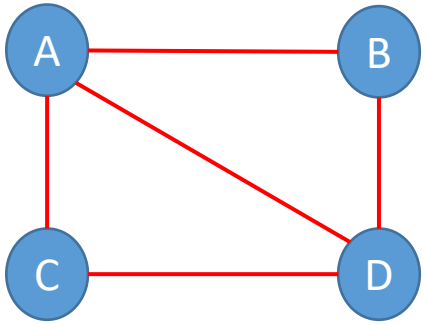
- Graph Terminology
- Graph Representation
  - Adjacency Matrix
  - Adjacency List
- Traversal of Graphs
  - Breadth-first Search
  - Depth-first Search

Graph Terminology

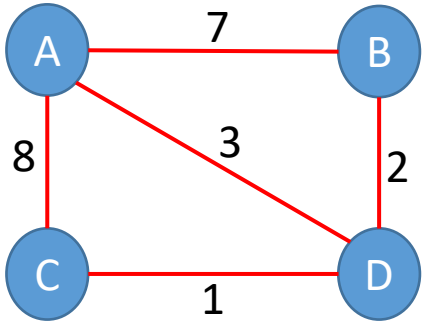
- A graph is a pair $G = (V,\ E)$, consisting of two finite sets:
  - A set V of vertices/ nodes
  - |V| is the number of vertices

  - A set E of edges/arcs/links that connect the vertices
  - $E = \{(x, y)|x, y \in V\}$
  - $|\ E\ |$ is the number of edges

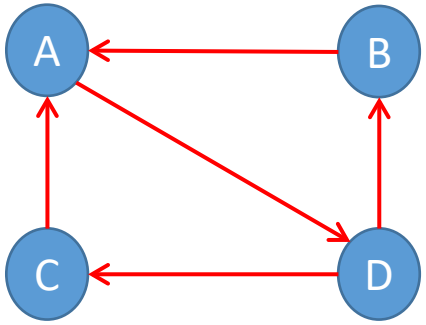
Graph Terminology



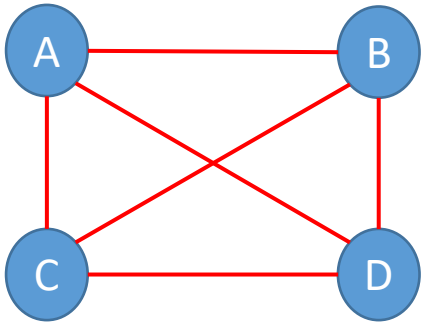
Undirected Graph



Weighted Graph



Directed Graph



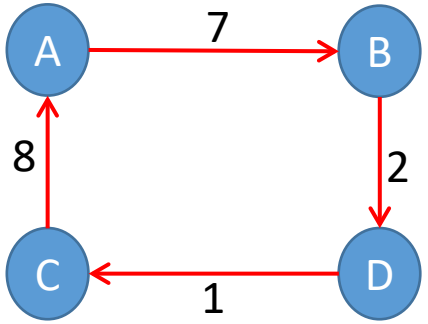
Complete Graph

Graph Terminology

- If E is unordered, then G is undirected; otherwise, G is a directed graph.
- If $e = (x, y)$ is an edge in an undirected graph, then e is incident with x and y; x is adjacent to y and vice versa.
- If $e = (x, y)$ is an edge in a directed graph, then y can be reached from x through one edge, so target y is adjacent to source x (but it doesn't mean x is adjacent to y).
- Degree of a vertex is the number of edges incident to it.

Graph Terminology

- A path is a sequence of nodes connected by edges. A simple path is a path that does not repeat any nodes. |V| = |E|+1
  - ABDC
- A path is a cycle if it starts and ends in the same node. A simple cycle is one containing at least three vertices and repeats only the first and last nodes. |V| = |E|
  - ABDCA

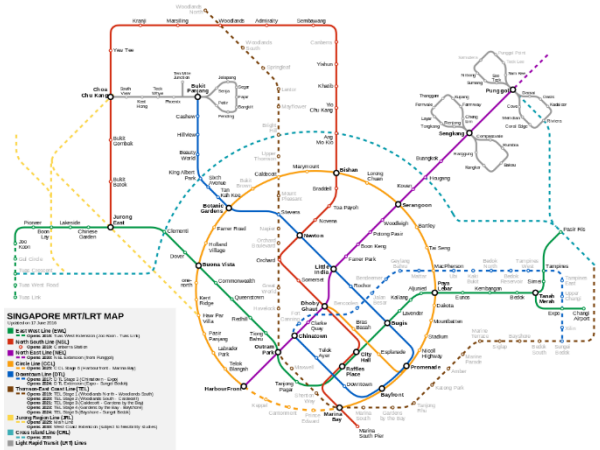


- A tree is a special graph with no cycle

Graph Terminology

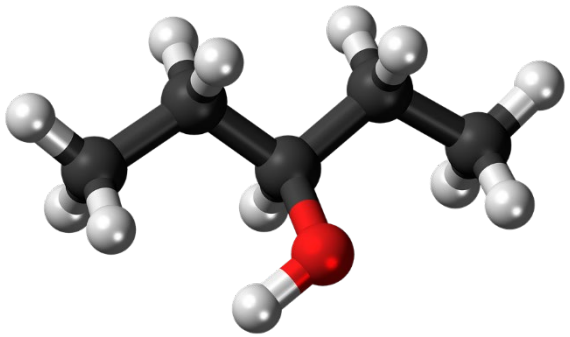
- An undirected graph is connected if there is a path from any vertex to any other vertex.
- A directed graph is strongly connected if there is a path from any vertex to any other vertex.
- A graph is cyclic if it contains one or more cycles; otherwise it is acyclic.
- A complete graph on n vertices is a simple undirected graph that contains exactly one edge between each pair of distinct vertices.

Graph Applications



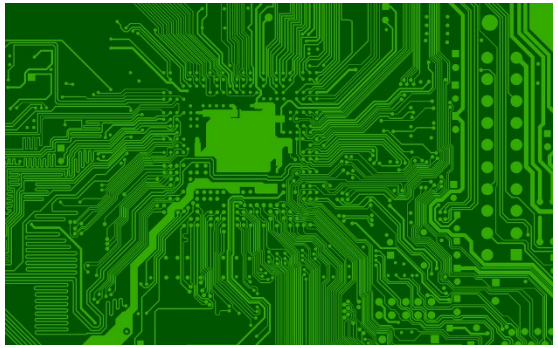
Maps

- V = {stations}
- E = {underground route}



Organic Chemistry

- V = {atoms}
- E = {bonds between atoms}



Electrical circuits

- V = {electrical devices}
- E = {linkage between devices}

Computer Networks

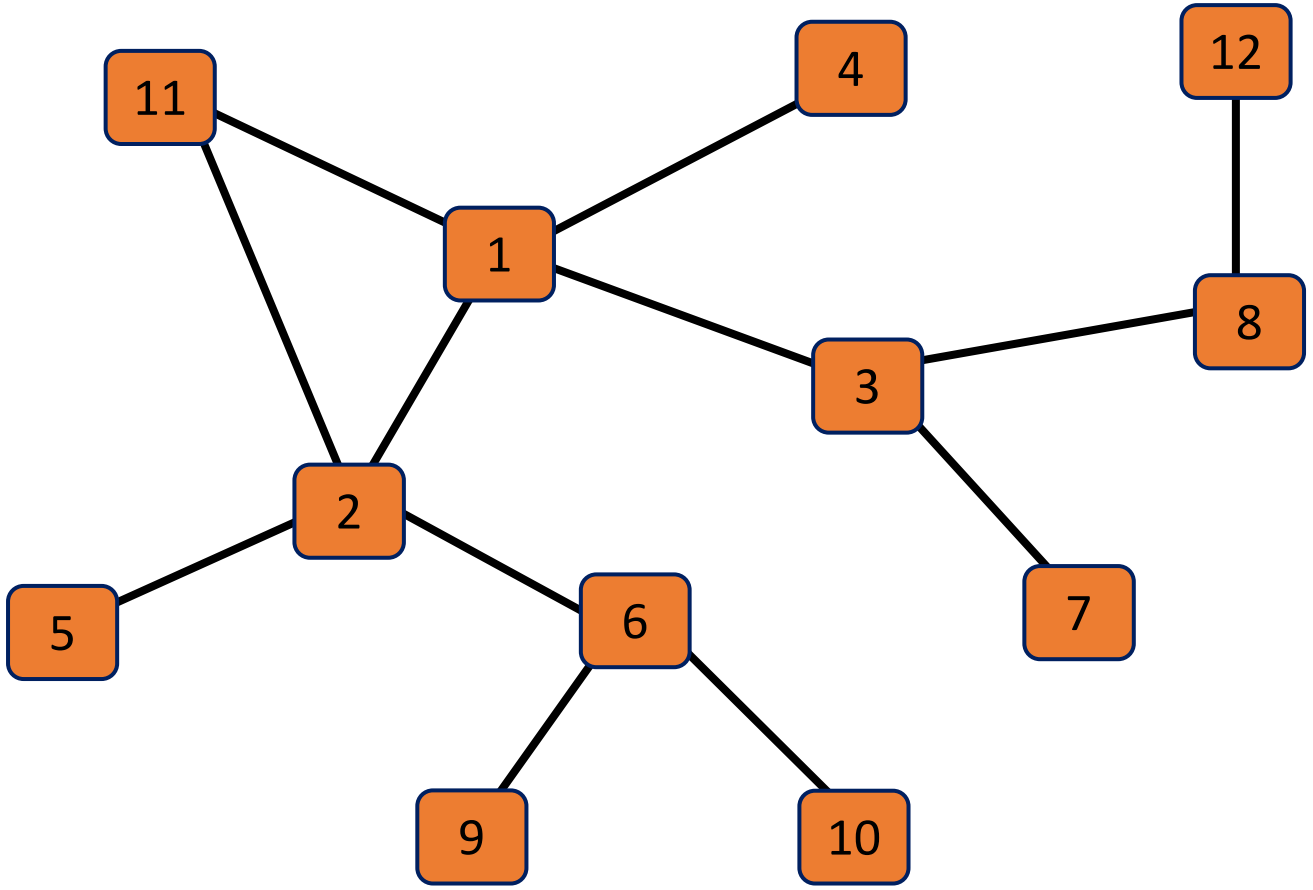
V = {computers}

E = {connections between computers}

- Aforl. (2014). A map of Singapore's Mass Rapid Transit (MRT) and Light Rail Transit (LRT) systems [Image]. Retrieved from https://commons.wikimedia.org/wiki/File:Singapore_MRT_and_LRT_System_Map.svg
- File: Electric circuit [Image]. (2013). Retrieved from https://pixabay.com/en/board-chip-circuit-electric-158973
- Chemistry-atoms [Image]. (2015). Retrieved from https://pixabay.com/en/pentanol-molecule-chemistry-atoms-867210/

Graph Representation

- Adjacency Matrix
- Adjacency List



Adjacency Matrix

- Use a matrix (2-D array) with size $|V| \times |V|$

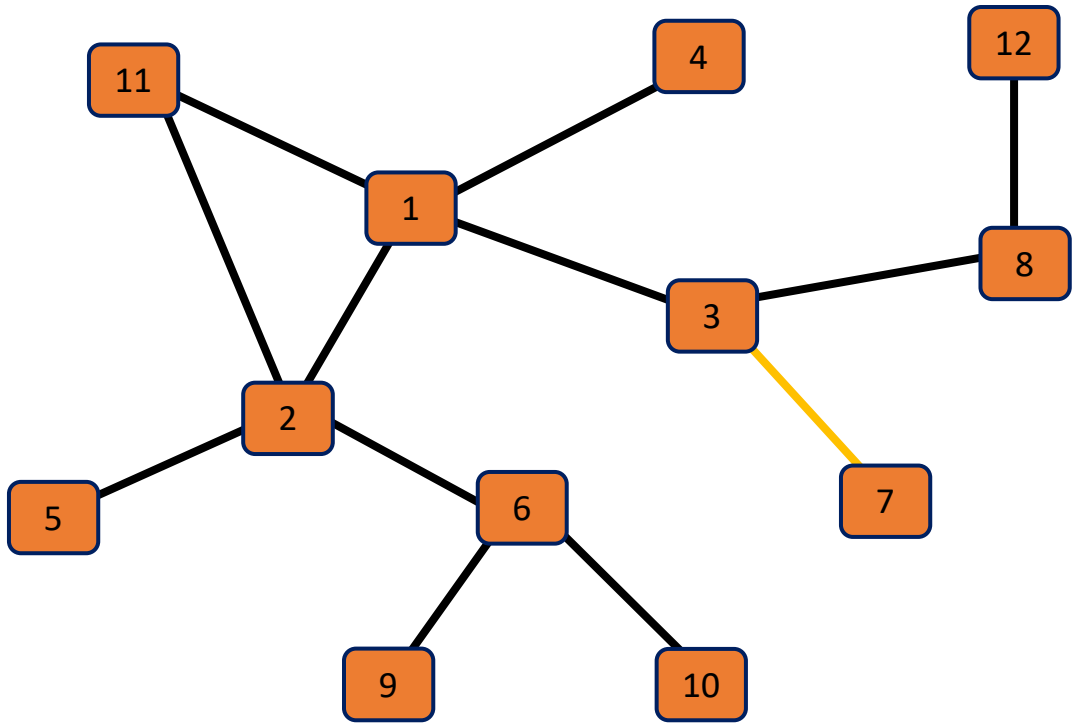
```
typedef struct _graph{
        int vSize;
        int eSize;
        int **AdjM;
}Graph;
```

- $(u, v) \in E$ implies AdjM[u][v] = 1; Otherwise AdjM[u][v] = 0.
- If a graph is undirected, then AdjM is symmetric
  - AdjM[u][v] = AdjM[v][u]
- If a graph is directed, then AdjM[u][v] = 1 iff $(u, v) \in E$ but it does not imply $(v, u) \in E$ and AdjM[v][u] = 1.

Adjacency Matrix

```
typedef struct _graph{
        int vSize;
        int eSize;
        int **AdjM;
}Graph;
```

- access time for AdjM[u][v] is constant
- when graph is sparsely connected, most of the entries in AdjM are zeros
- Space complexity is $\Theta(|V|^2)$, Access time for AdjM[u][v] is constant



	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	0	0	0	0	0	0	1	0
2	1	0	0	0	1	1	0	0	0	0	1	0
3	1	0	0	0	0	0	1	1	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	1	1	0	0
7	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	1	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	0	0	0	0
11	1	1	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	0

Adjacency List

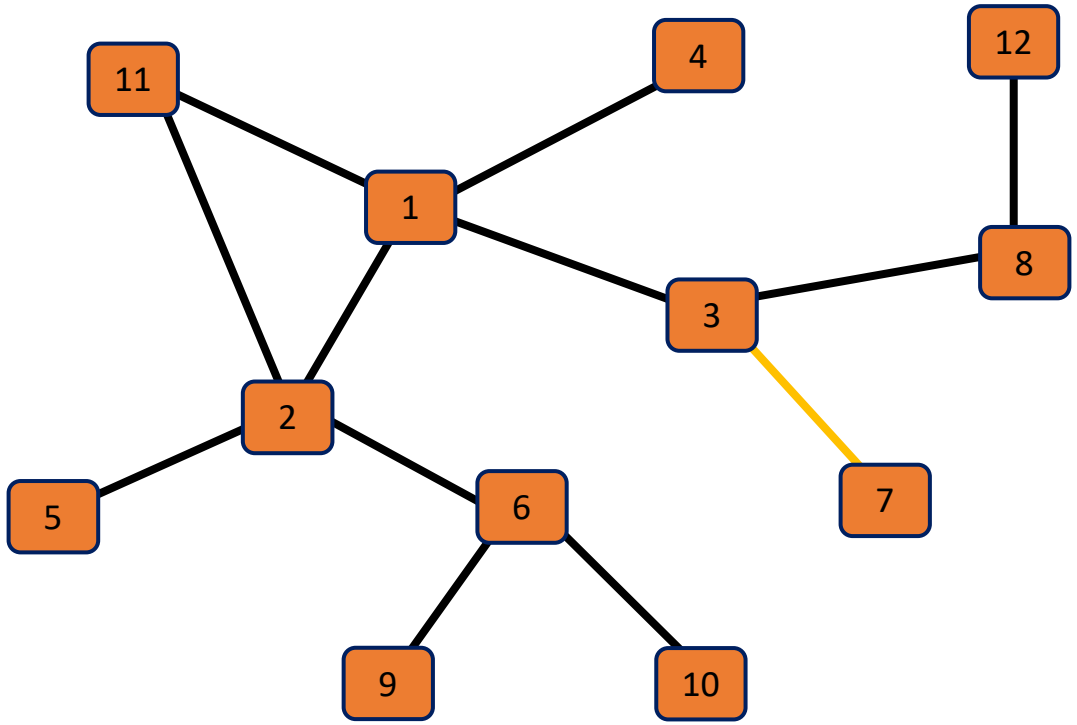
- Use an array to represent the vertices

- For each vertex, use a linked list to represent the connections to other vertices

```
struct _listnode
{
        int id; //or weight
        struct _listnode *next;
};
typedef struct _listnode ListNode;
typedef struct _graph{
       int vSize;
       int eSize;
       ListNode **AdjL;
}Graph;
```

Adjacency List

- Array size is |V|.
- For undirected graph, total number of nodes in link lists is 2|E|
- Space complexity is lower, Θ(|V|+|E|)
- Access time for AdjL[u][v] is linear



1	→ 2 → 3 → 4 → 11
2	→ 11 → 1 → 5 → 6
3	→ 1 → 8 → 7
4	→ 1
5	→ 2
6	→ 10 → 9 → 2
7	→ 3
8	→ 12 → 3
9	→ 6
10	→ 6
11	→ 2 → 1
12	→ 8

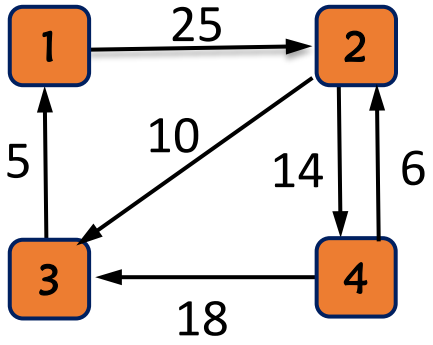
Represent Weighted Graphs

- In the array of adjacency lists, the weight can be stored as a data field in each list node
- In the adjacency matrices, the weight can be stored
  - The element at the *u*-th row and the v-th column can be defined as:

$$AdjM[u][v] = \begin{cases} W(u,v) & \text{if } (u,v) \in E \\ c & \text{otherwise} \end{cases}$$

- Constant c can be defined as 0 (weight as capacity) or some very large number ∞ (weight as cost)

Represent Weighted Graphs



	1	2	3	4
1	0	25	0	0
2	0	0	10	14
3	5	0	0	0
4	0	6	18	0

1	→ (2, 25)
2	→ (3, 10) → (4, 14)
3	→ (1, 5)
4	→ (2, 6) → (3, 18)

```
struct _listnode
{
        int id;
        int weight;
        struct _listnode *next;
};
typedef struct _listnode ListNode;
typedef struct _graph{
        int vSize;
        int eSize;
        ListNode **AdjL;
}Graph;
```

Traversal of Graphs

- To traverse a graph means to visit the vertices of the graph in some systematic order.

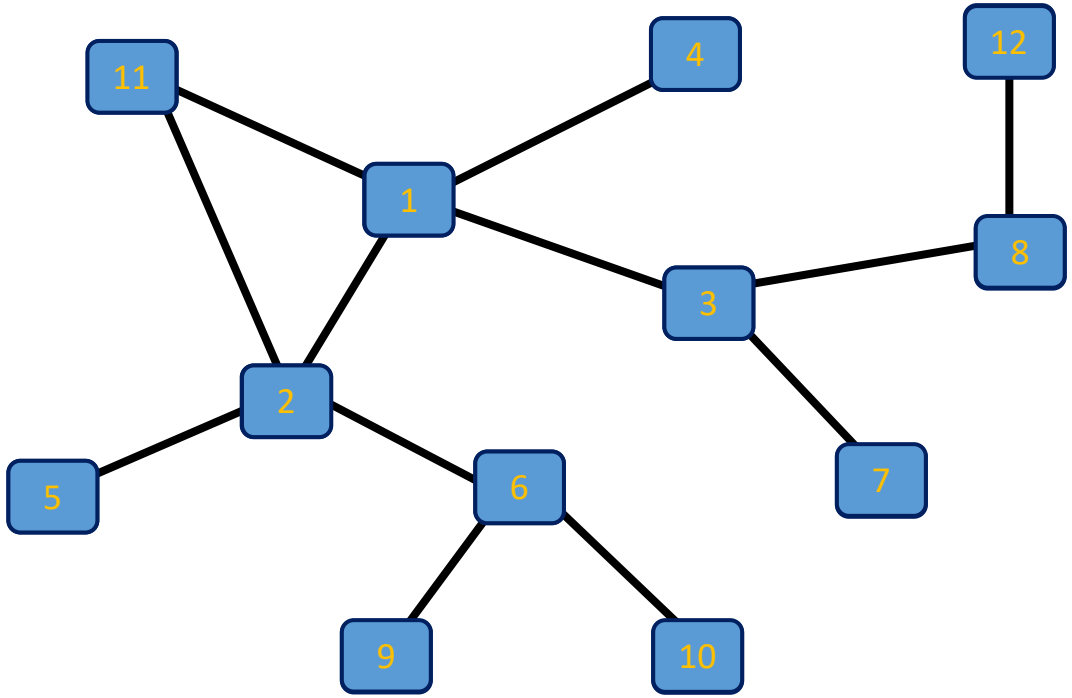
- In some applications, we may need to do some processing at every vertex of a graph.

- To visit each vertex and edge exactly once, we can apply:
  - Breadth-first Search

  - Depth-first Search

Breadth First Search (BFS)

- Work similar to level-order traversal of the trees
- BFS systematically explores the edges directly connected to a vertex before visiting vertices further away.

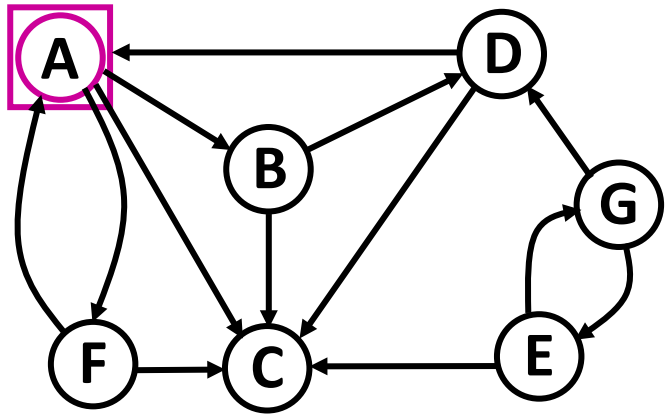


```
typedef struct _linkedlist{
     ListNode *head;
     int size;
} LinkedList;
```

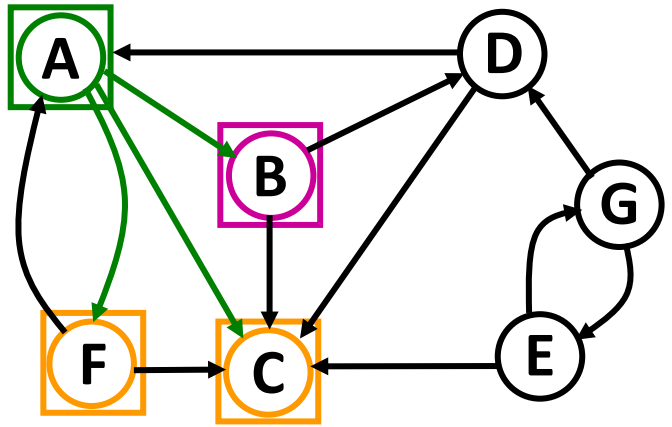
```
typedef ListNode QueueNode;
typedef struct _queue{
   int size;
   ListNode *head;
   ListNode *tail;
} Queue;
```

Breadth First Search (BFS)

- A queue is used to monitor which vertices to visit the next
- Action taken during visiting v_i depends on specific applications



Visited: A



Visited: ABCF

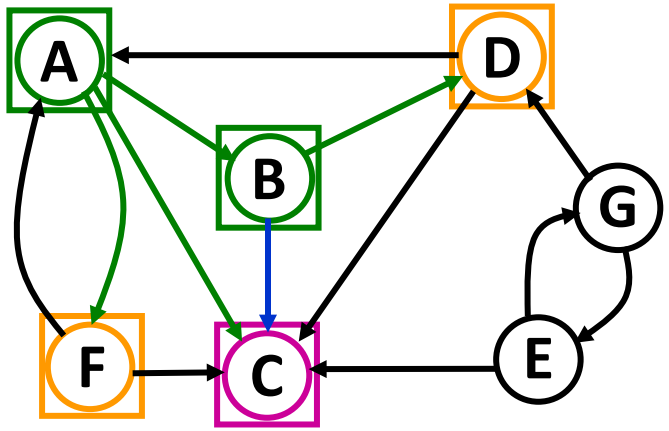
Visited: ABCF

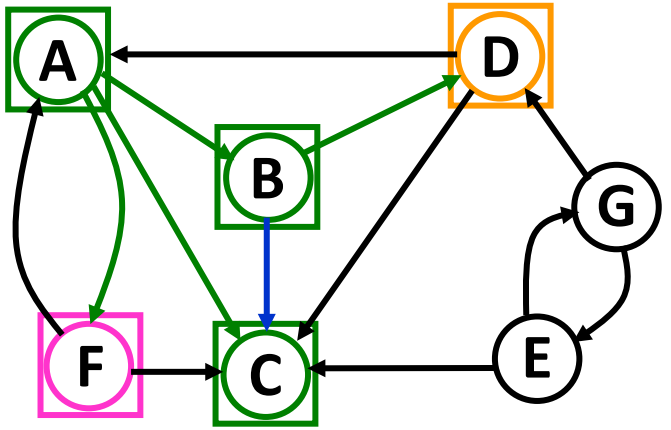
Visited: ABCFD

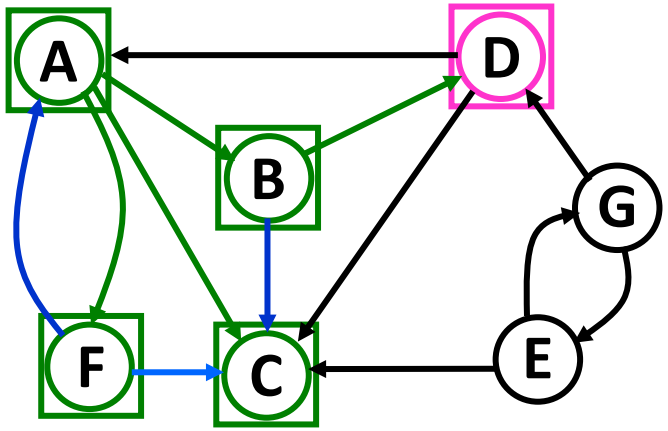
Visited: ABCFD

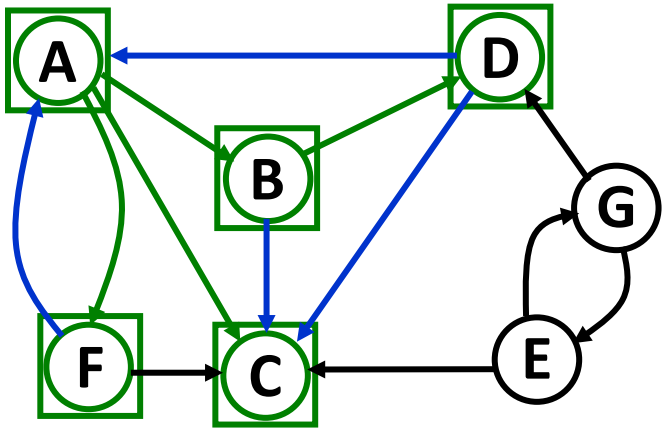
Visited: ABCFD

Visited: ABCFD

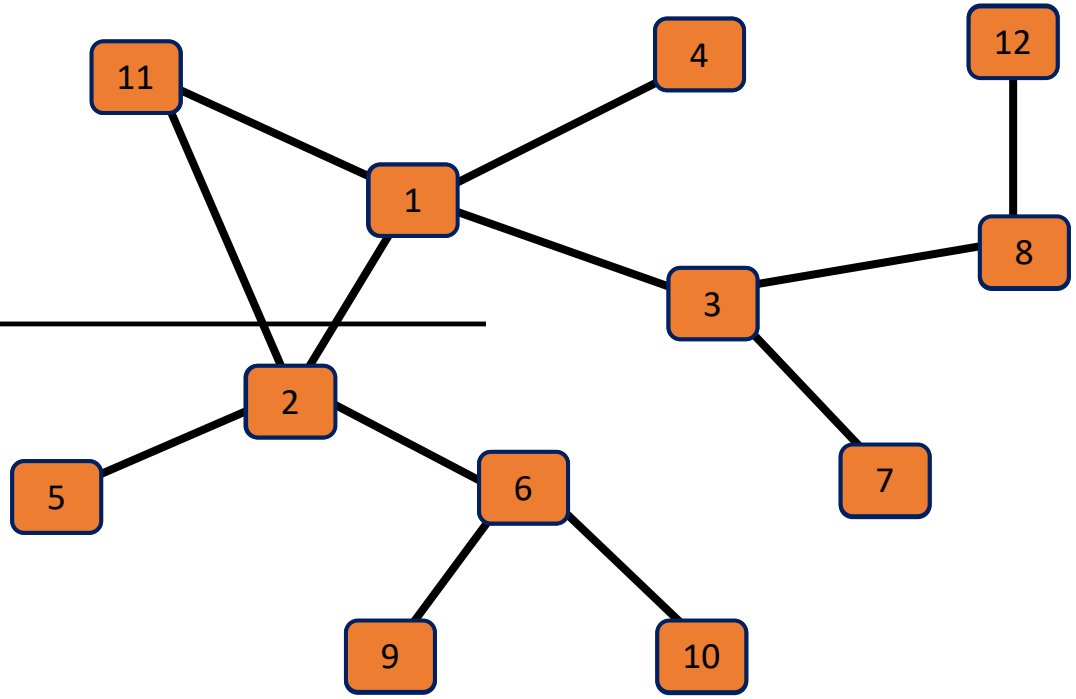








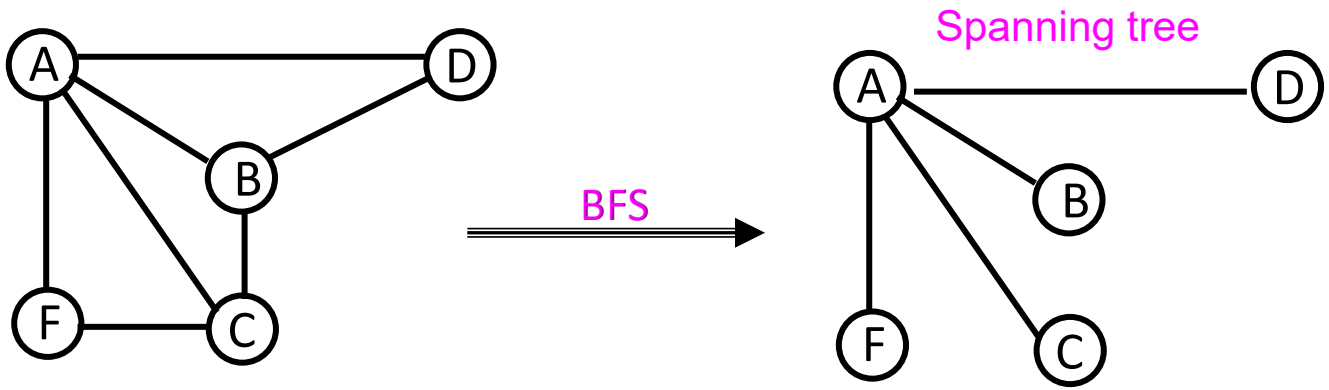
BFS Algorithm




```
function BFS(Graph G, Vertex v)
    create a Queue, Q
    enqueue v into Q
    mark v as visited
    while Q is not empty do
        dequeue a vertex denoted as w
        for each unvisited vertex u adjacent to w do
            mark u as visited
            enqueue u into Q
        end for
    end while
end function
```

Breadth First Search (BFS)

- If a vertex has several unmarked neighbours, it would be equally correct to visit them in any order.
- If the shortest path from s to any vertex v is defined as the path with the minimum number of edges, then BFS finds the shortest paths from s to all vertices reachable from s.
- The tree built by BFS is called the breadth first spanning tree (when graph G is connected), , i.e., a set of |V|-1 edges that connect all vertices of the graph.



Applications of BFS

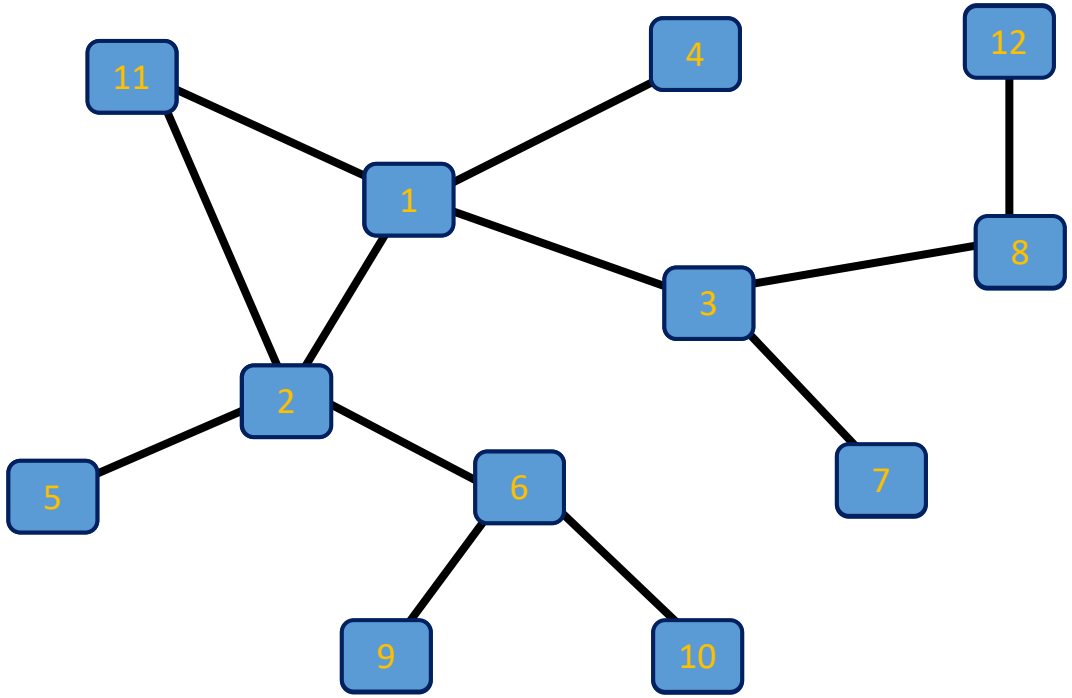
- Finding all connected components in a graph
- Finding all vertices within one connected component
- Finding the shortest path between two vertices

Time Complexity of BFS

- Each edge is processed once in the while loop for a total cost of $\Theta(|E|)$
- Each vertex is queued and dequeued once for a total cost of $\Theta(|V|)$
- The worst-case time complexity for BFS is
  - $\Theta(|V| + |E|)$ if graph is represented by adjacency lists
  - $\Theta(|V|^2)$ if graph is represented by an adjacency matrix
    - each vertex takes $\Theta(|V|)$ to scan for its neighbours

Depth First Search (DFS)

- Work similar to preorder traversal of the trees
- DFS systematically explores along a path from vertex v as deeply into the graph as possible before backing up.



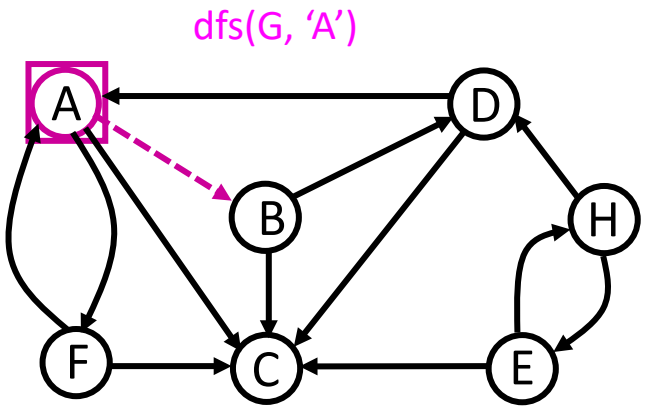
Depth First Search (DFS)

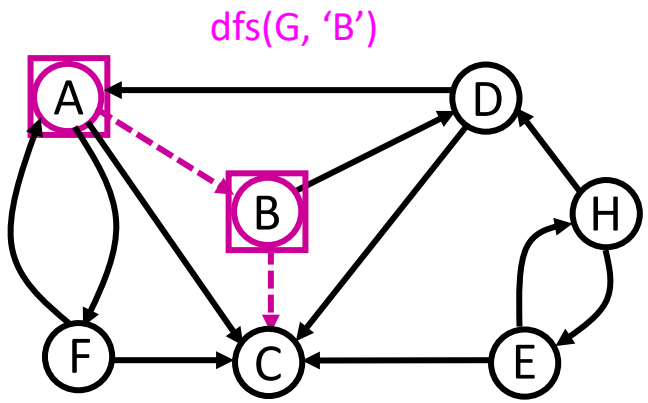
```
struct _listnode
{
      int item;
      struct _listnode *next;
} ListNode;
```

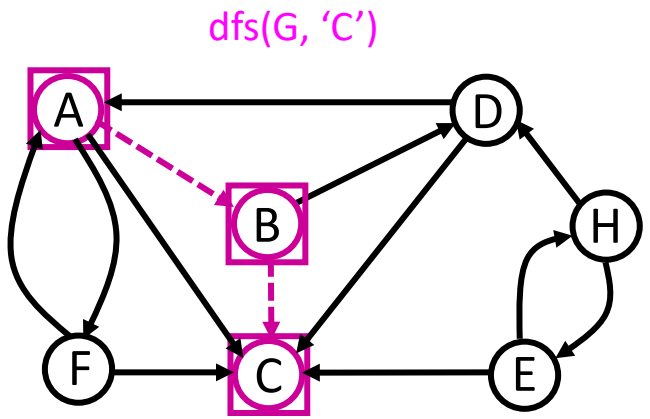
```
typedef struct _linkedlist{
      ListNode *head;
      int size;
} LinkedList;
```

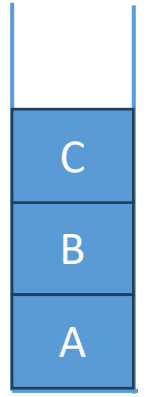
```
typedef ListNode StackNode;

typedef LinkedList Stack;
```

- A stack is used to monitor which vertices to visit the next
- Action taken during visiting v_i depends on specific applications

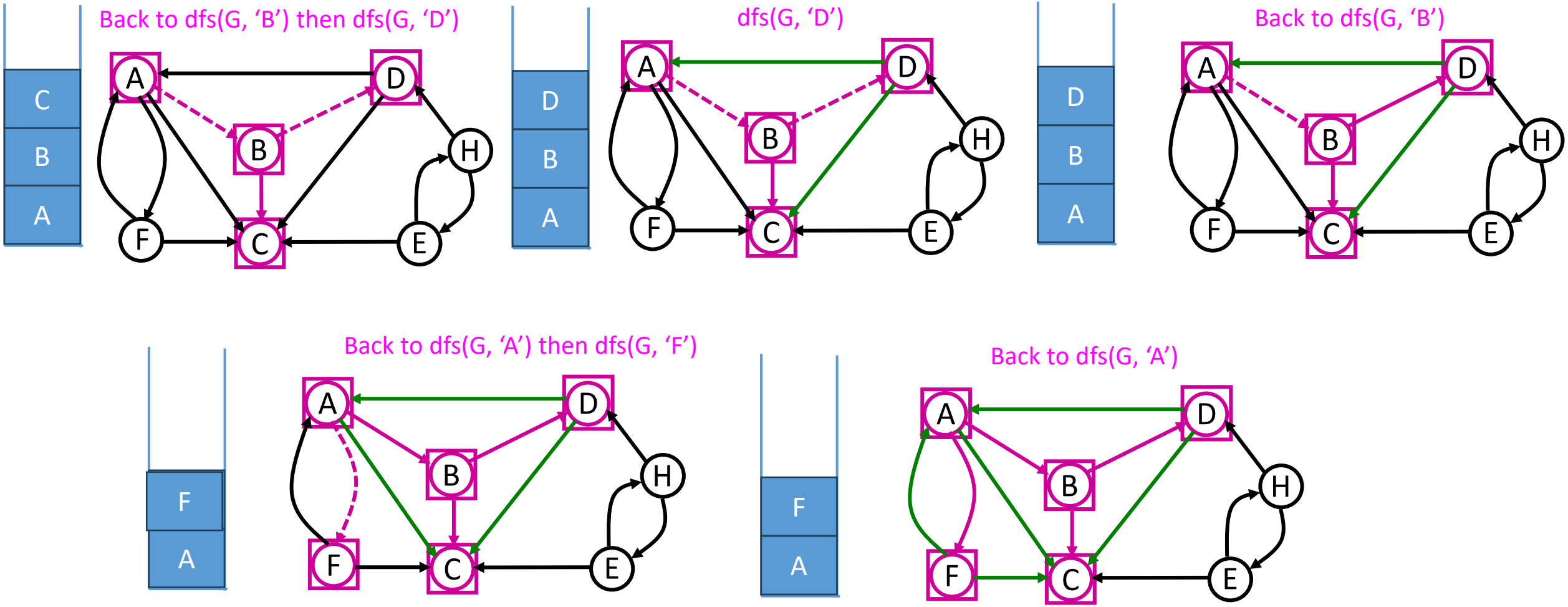






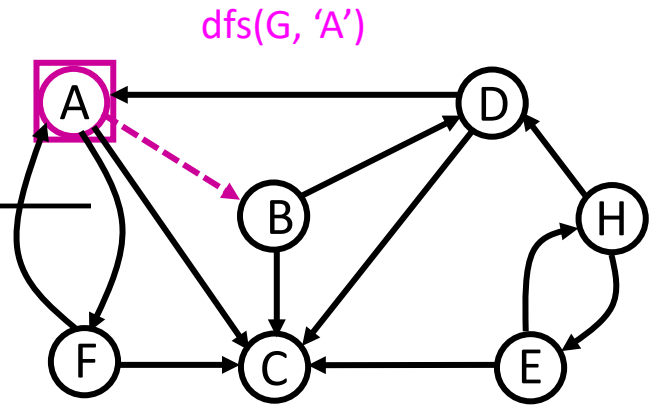


Depth First Search (DFS)



This directed graph is **not** strongly connected

DFS Algorithm

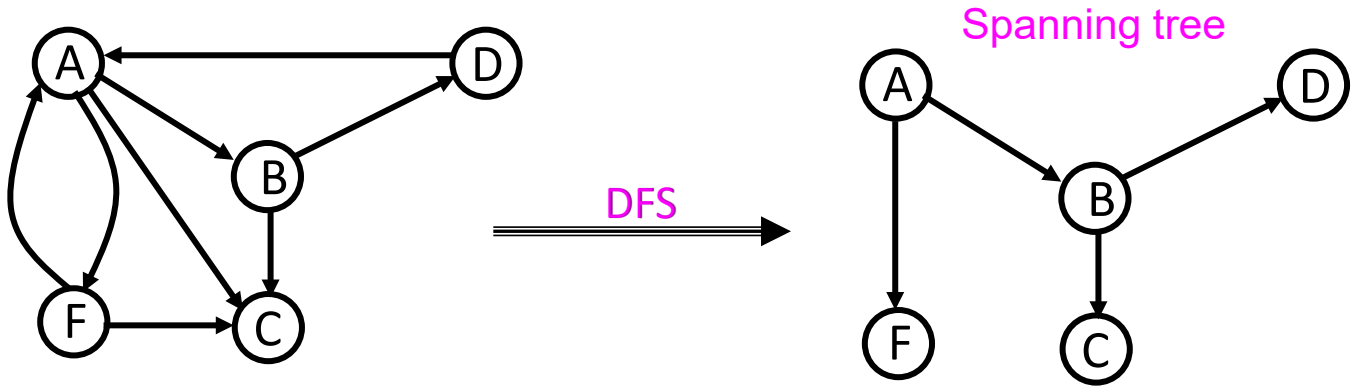



```
function DFS(Graph G, Vertex v)
    create a Stack, S
    push v into S
    mark v as visited
    while S is not empty do
        peek the stack and denote the vertex as w
        if no unvisited vertices are adjacent to w then
            pop a vertex from S
        else
            push an unvisited vertex u adjacent to w
            mark u as visited
        end if
    end while
end function
```

Depth First Search (DFS)

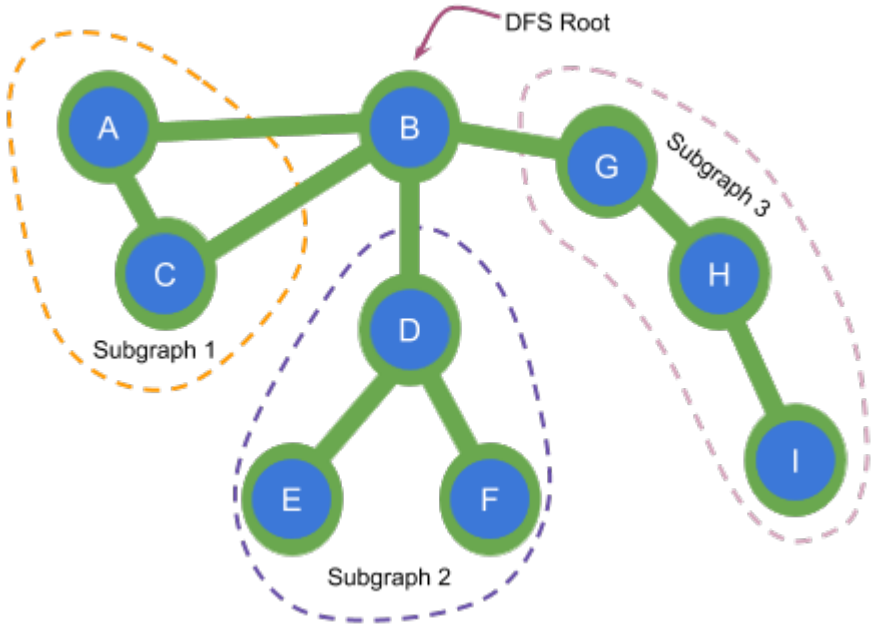
- If a vertex has several neighbours it would be equally correct to go through them in any order.

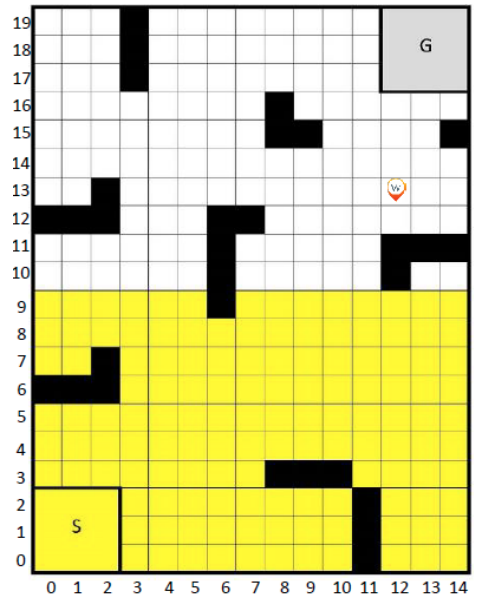
- If the graph is strongly connected, the tree T, constructed by the DFS algorithm is a spanning tree, T is called the **depth first search tree**.



Applications of DFS

- Finding connected components
- Finding strongly connected components
- Finding articulation points (cut vertices) of the graph
- Solving puzzles





Time Complexity of DFS

- The DFS algorithm visits each node exactly once; every edge is traversed once in forward direction (exploring) and once in backward direction (backtracking).

- Using adjacency-lists, time complexity of DFS is $O(|V| + |E|)$.
- Using adjacency matrix, time complexity of DFS is $O(|V|^2)$.

Summary

- Concepts and terminologies of graph, such as
  - A graph consists of a set of vertices and a set of edges
  - Directed vs. undirected graphs
  - The definitions of path and cycle, etc.
- Two data structures used to represent graphs:
  - Adjacency matrix
  - Array of adjacency lists
  - Their advantages and disadvantages for different applications

Summary

- Two elementary algorithms for graph traversal
  - Breadth-first search (BFS): Use queue
  - Depth-first search (DFS): Use stack
- Time complexity of BFS or DFS:
  - Using adjacency lists: $O(|V| + |E|)$
  - Using adjacency matrix: $O(|V|^2)$

Problem for you to think about

- You are given a grid with two dimensions. The grid cell values are 1 or 0 only, where 1 represents land and 0 represents water. An island is lands surrounded by water. Each island is formed by connecting nearby lands vertically and/or horizontally. Please give the number of island in the given grid.
- Hint: DFS

1	1	1
0	1	0
1	0	0
1	0	1