Graphs

A **graph** is a collection of vertices and edges

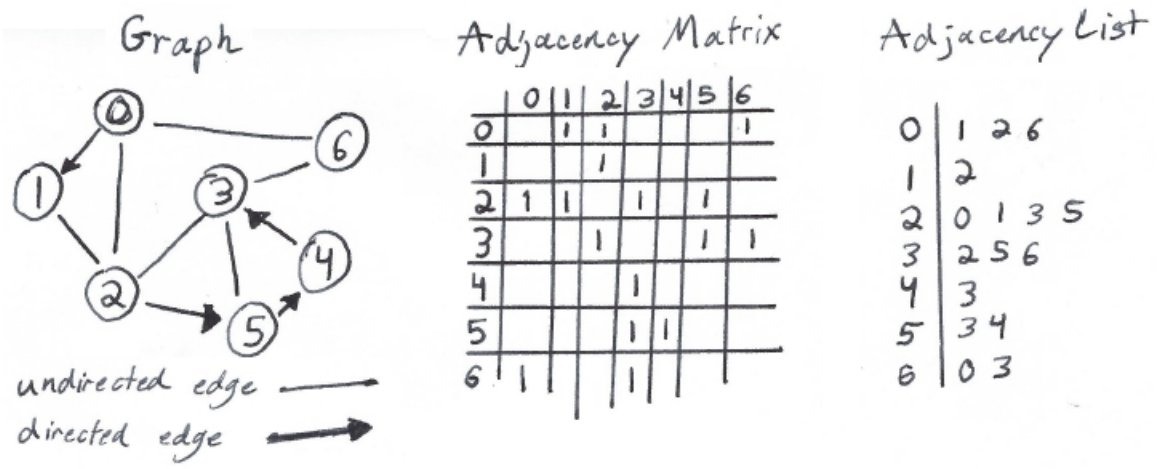
G =(V,E)

We may assume that a graph has |V| vertices and |E| edges, although sometimes we (lazily) refer to the number of vertices as V and the number of edges as E.

There are two main representations of graphs:

1. Adjacency List
2. Adjacency Matrix

The following shows an example of a graph, it's **adjacency matrix** as well as it's **adjacency list** representation.



The storage requirements for the representations are the following

adjacency matrix O V^2 storage
adjacency list O V+E storage

Graphs can be used to represent data that has some relationship, examples include

a) roads (edges) and intersections (vertices) in a mapping service
b) friends of friends on social media
c) genetic relations between species

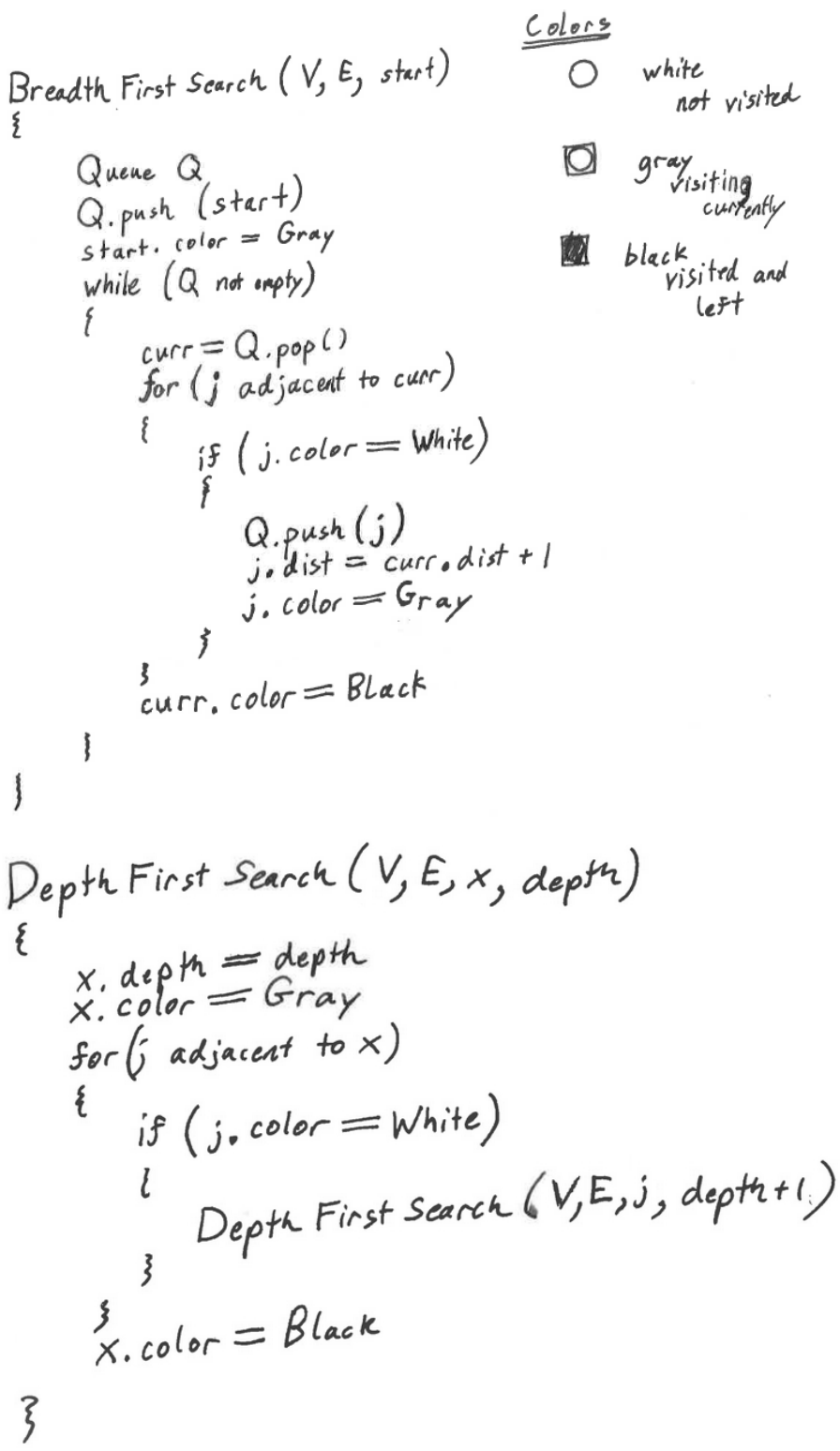
The "shortest path" can be determined by using a breadth first search. Breadth first search will determine the distance from a source vertex to all other vertices connected by a simple path, and as such

In order to traverse a graph, there are two primary kinds of traversals

BreadthFirstSearch BFS
DepthFirstSearch DFS

BFS is usually implemented iteratively using a Queue, whereas DFS is usually implemented recursively. Of course any recursive function can also be implemented iteratively using a stack.

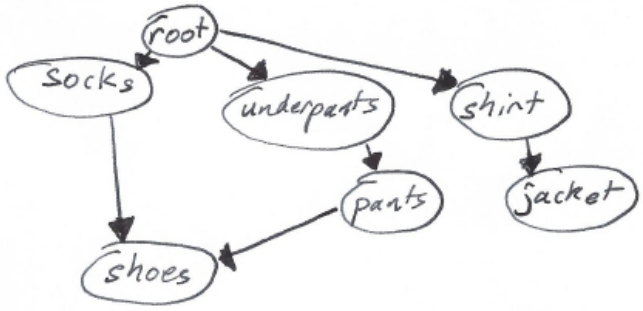
The pseudocode for BFS and DFS is as follows,



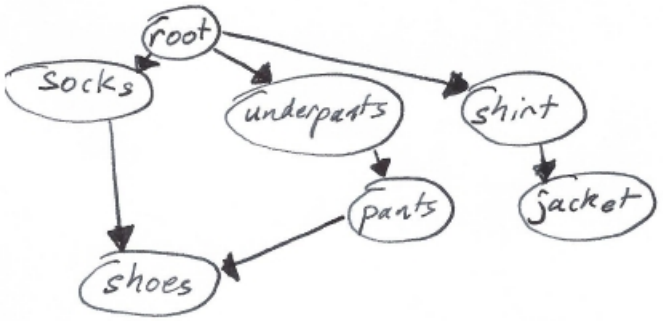
Topological Sort for **directed acyclic graphs** involves printing all ancestors before any children are printed. This is useful for planning workflows, where the order of operations matters.

Reverse Topological Sort can be accomplished using DFS. We modify DFS to add an additional Print statement at the end of the function as follows. We then show an example for a workflow plan of getting dressed in the morning.

```
Print Reverse Topological (V, E, x, depth)
{
    x.depth = depth
    x.color = Gray
    for (j adjacent to x)
    {
      if (j.color = white)
      {
        Print Reverse Topological (V, E, x, depth)
      }
    }
    x.color = Black
    Print (x)          // we print in reverse topological order
}
```



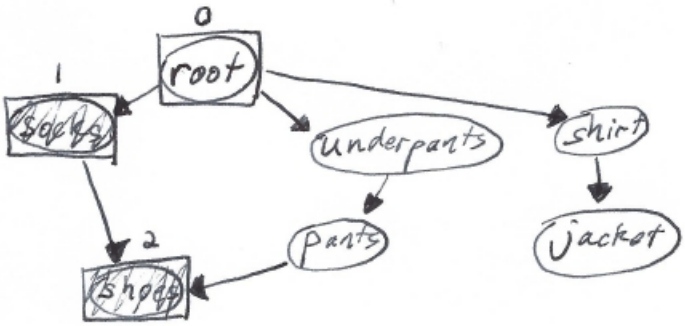
Print the reverse topological sort of the following



Path 1

0: Print shoes

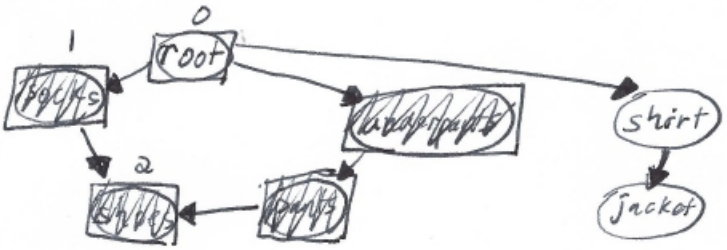
1: Print socks



Path 2

2: Print pants

3: Print under pants

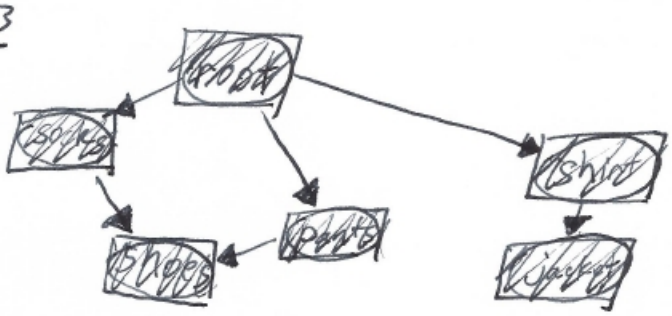


Path 3

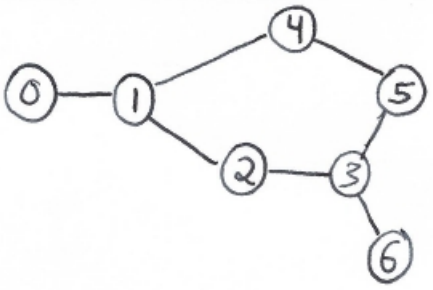
4: print jacket

5: print shirt

6: print root



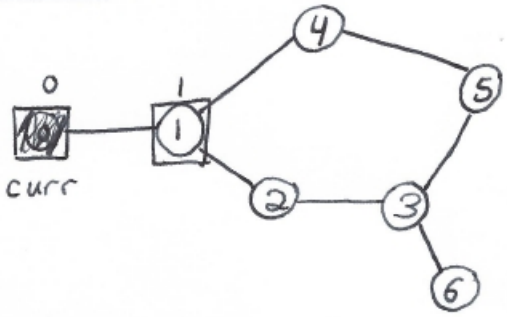
Given the following Graph



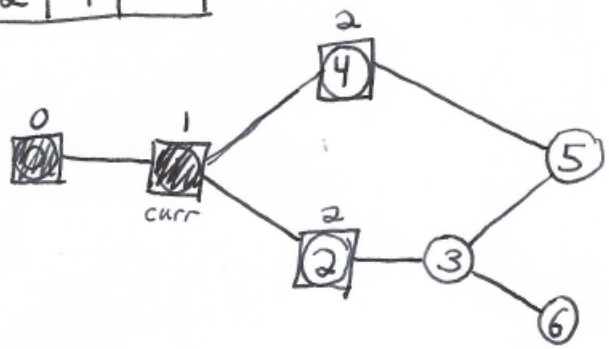
0	1
1	0 2 4
2	1 3
3	2 5 6
4	1 5
5	3 4
6	3

Breadth First Search (V, E, 0)

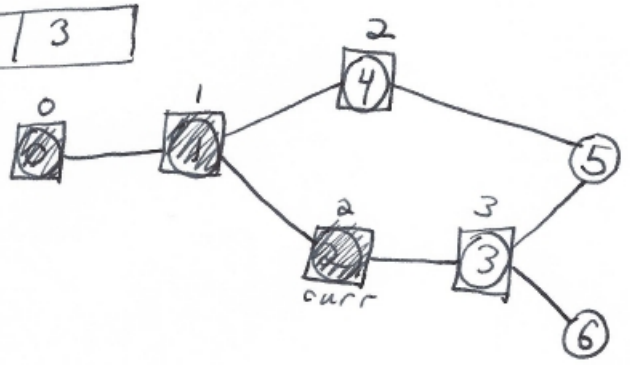
Q | 1 |

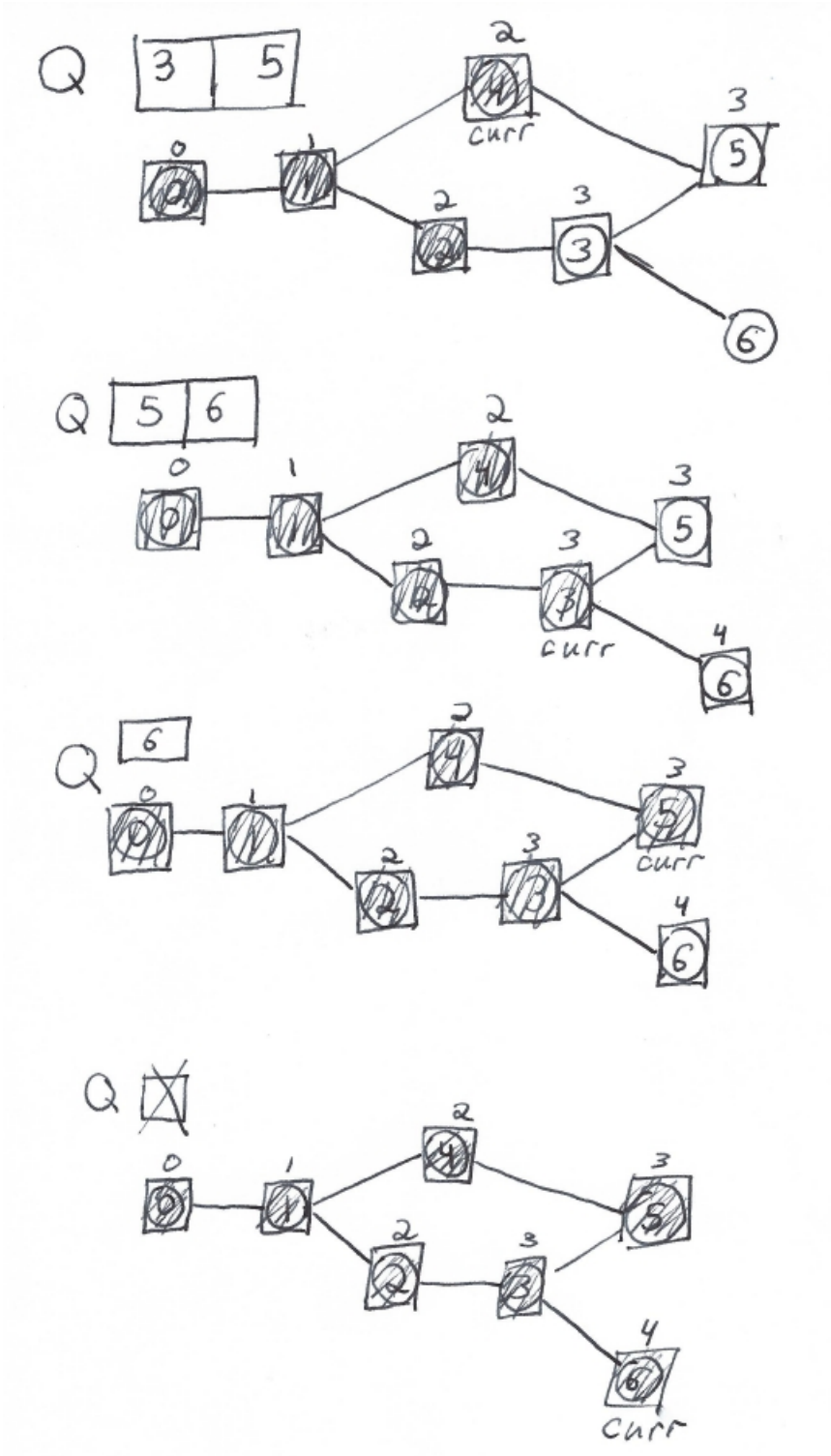


Q | 2 | 4 |

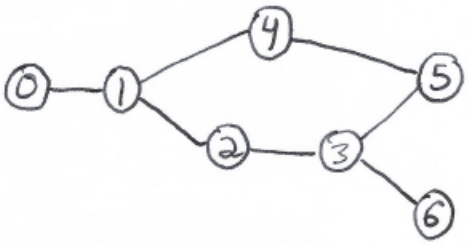


Q | 4 | 3 |



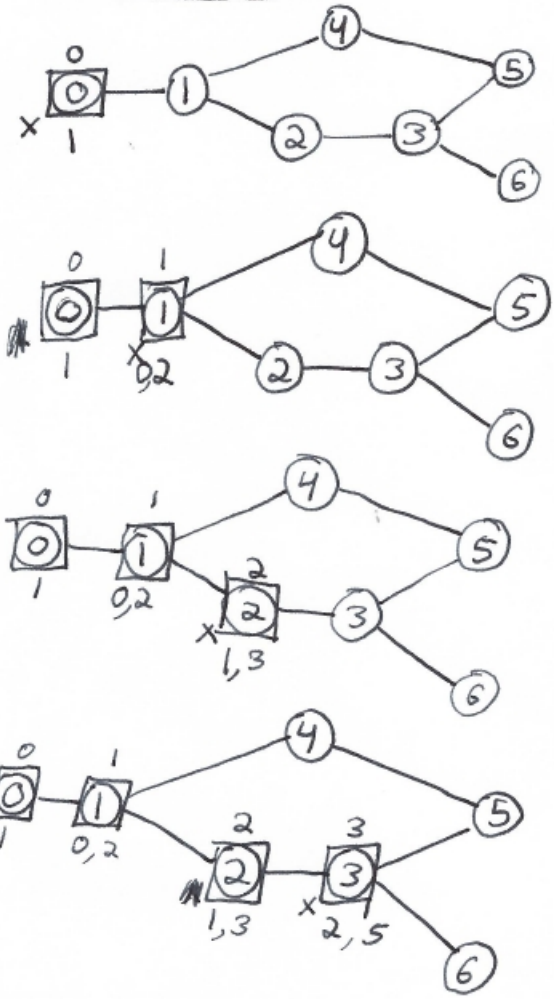

Q
3
5
curr
Q
5
6
curr
Q
6
curr
Q
curr

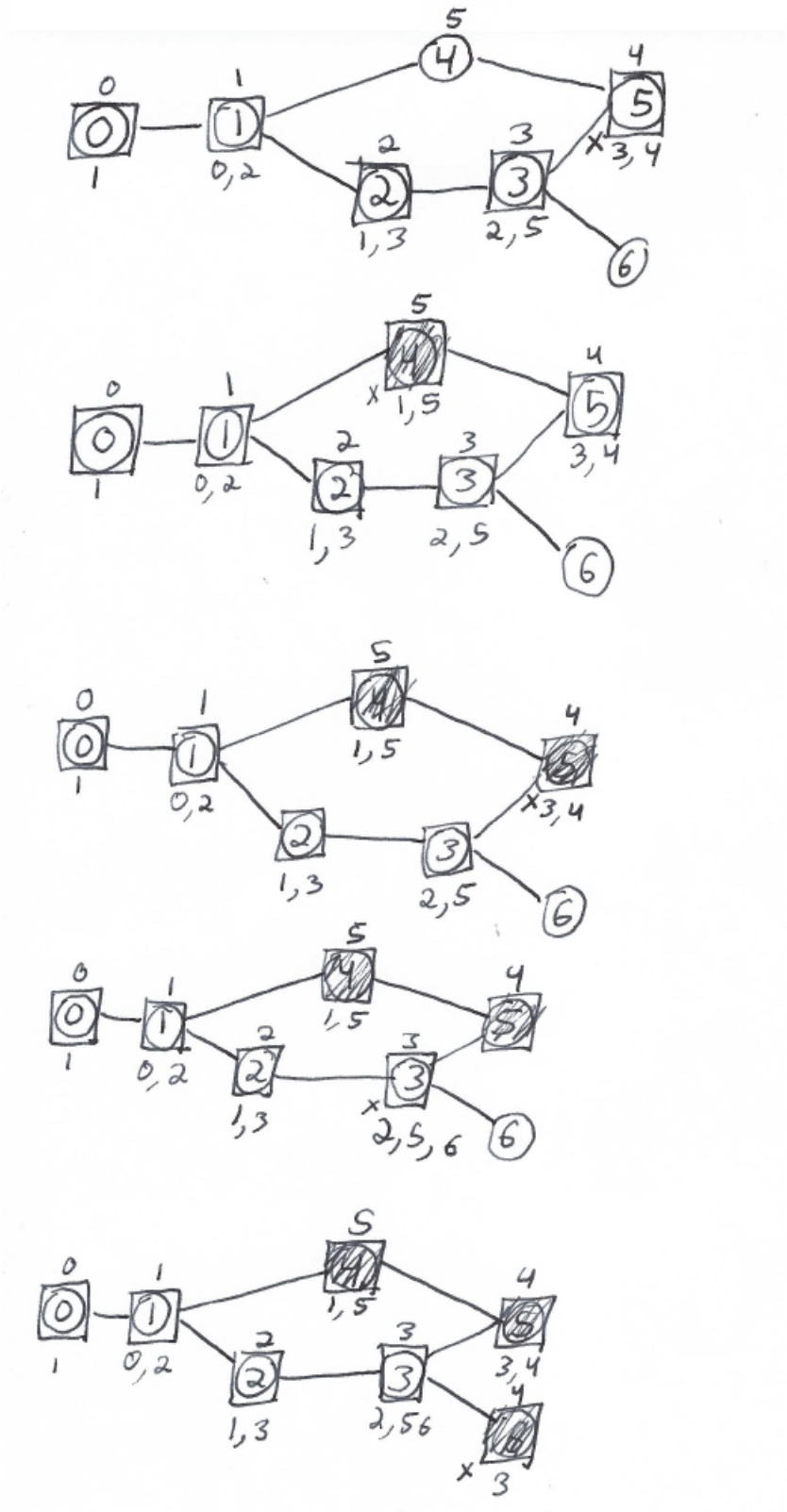
Given the following graph

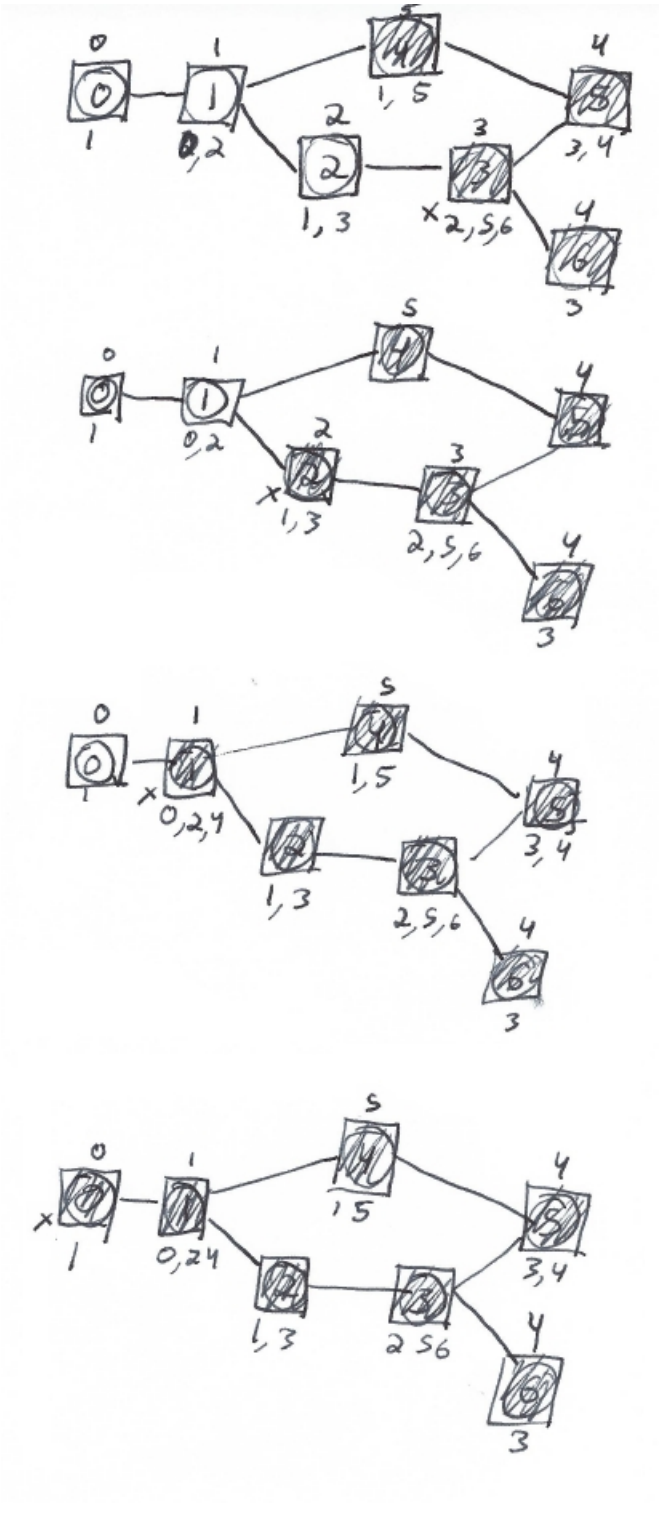


0	1
1	0 2 4
2	1 3
3	2 5 6
4	1 5
5	3 4
6	3

Depth first search (V, E, 0, 0)




0
1
1
0,2
2
1,3
3
2,5
5
4
4
5
x
3,4
6


0
1
5
4
1
0,2
2
3
1, 5
3, 4
1, 3
×2,5,6
4
3
5
0,2
×1,3
2,5,6
0,2,4
1,5
3,4
1,3
2,5,6
15
0,24
1,3
2 56