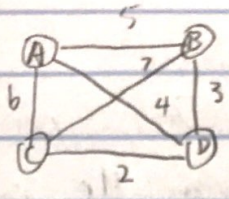
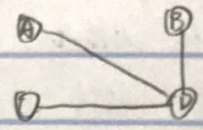
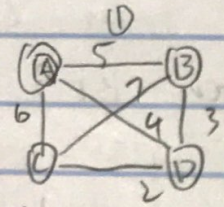
1. graph:



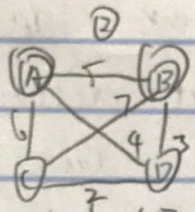
MST:



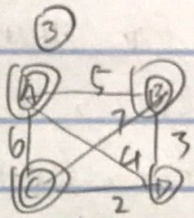
BFS:



output: A



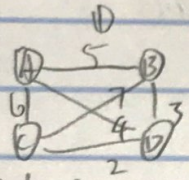
output: AB



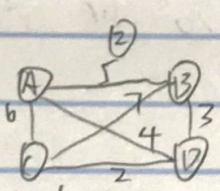
output: ABC

output: ABCD

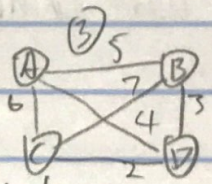
DFS:



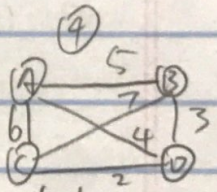
output: A



output: AD

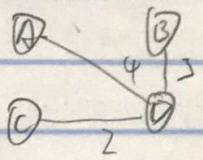


output: ADB

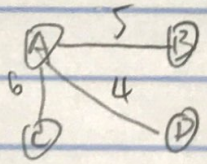


output: ADBC

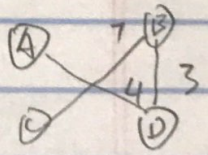
Final MST:



Final BFS:



Final DFS:



the DFS and BFS I got is the most similar to the original graph of MST. There are other ways to get the Final BFS and Final DFS but these ways are not same as MST as well.

2. First I will use Topological sort to sort DAG, then I have to make sure the DAG is a lattice. I will use BFS to see if the graph can reach each vertex from the start vertex which is the first vertex after sort. If it can goes through each vertex then the start vertex is a source, if not, then no source in DAG. After found the source, reverse the direction in DAG and do BFS again. If it can reach each vertex then start vertex is a sink, else no sink in DAG.

Pseudocode: S = stack( )

```
S = Topological sort (DAG)
first = peak(s)
if BFS(first) not all vertex black then
    return False
else
    reverse direction in DAG
    if BFS (last vertex in stack) not all vertex black then
        return False
    else
        return True
```

Time complexity of topological sorting is $O(V+E)$, after this use BFS twice and not inner loop $O(V+E)$, so by Maximum Therom $T(n) = O(V+E)$ for pseudocode.

3. For prove question 3, Assume theres a graph G is not a tree, then BFS and DFS spanning trees of a connected undirected graph from the same start vertex s are not equal to each other.

If G is not a tree, then there must be a cycle in G. For example, the cycle is from C_1 to C_n Assum C_k is the start vertex, then BFS must be $C_k, C_{k+1}, C_{k-1}, \cdots$. But for DFS is either C_k, C_{k+1}, C_{k+2} or C_k, C_{k-1}, C_{k-2} which these two are not same. So the statement is correct

4. First I will creat a 2D array like matrix and set all distance for the vertex itself to itself to 0 and others are big enough, then put all the weight of vertex in the 2D array. Then use three for loops to go through all the element in the array, in these three loops I will compare each one with other distance start and end with the as same vertex as the first loop picked. After that find shortest and put in the 2D array. Now the 2D array is complete and find out the maximum.

```
array[V][V]
for i to V
    for j to V
        if i=j then
            array[i][j]=0
        else
            array[i][j] = a number that big enough

for every edge in G
    add weight into array

for m to V do
    for s to V do
        for e to V do
            array[s][e] = min(array[s][m]+array[m][e], array[s][e])
        od
    od
od
```

① $O(V^2)$

② $O(V^3)$

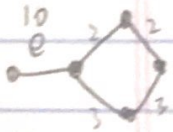
③ $O(V^2)$

```
max = array[0][0]
for i to V do
    for j to V do
        if max < array[i][j] & array[i][j] ≠ number that big enough
            max = array[i][j]
    od
od
```

correctness and time complexity:

in part ① the 2D array store data that we already know before the algorithm. The time complexity for ① is $O(V^2)$. In part ② is to find the value from s to e is larger than s to m and plus m to e then put into array. The time complexity for ② is $O(V^3)$. In part ③ is just find which value is the largest in the array, and time complexity for ③ is $O(V^2)$. So according maximum theorem $T(n) = O(V^3)$

5. (a) False.

ep: 

$|MST| = |V| - 1$, so there must have a cycle. If e is part of the non-cycle part the e must be inside of any MST, so the statement is False

(b) True.

e is a unique lightest means e is the lightest cross between a cut in somewhere. Suppose there's a e' in a MST called T'. So if e crosses a cut and e is not in T', so there must be a edge e' crosse the cut and $e' \in T'$. Then total(T) = total(T') - weight(e') + weight(e) < total(T') since weight(e) < weight(e') so T' can not be minimum spanning tree

(c) True

Since edge e is a maximal weight of a cycle of G, there must be other lighter edges in the cycle which contain all the vertex in the cycle. The weight of these edges must be less than the weight of edges include edge e.

(d) True

The order in which edges are added in Prim's algorithm do not change and the cut property also holds for negative weight edges.

6. To find out the shortest distance between the vetex that be selected and other vetex I will use Dijsktra's algorithm. So I will use the algorithm twice, first time it calculate the shortest distance between input vertex v to all others. After that, because the graph is directed, so I will reverse the direction and use Dijsktra's algorithm again to get the distance between all other vertex and input v. Then pair them up.

```
Dijsktra( G, vertex)
    Initialize S, d
    while S≠ V
        u ← V&S
        add u to S
        for v&S
            d[v] ← min(d[v], d[u]+w(u,v)
Path( G, vertex)
a1[] = Dijsktra(G, v)
    for each edge from every vertex a to vertex b
        reverse ((a,b) ← direction)
a2[] = Dijsktra(G, v)
    array [G←V][G←V]
    for i to G←V-1
        for j to G←V-1
            if i≠j then
                array[i][i] = a1[i] + a2[j]
```

The time complexity for Dijsktra is $O(E+V\log V)$ and thererse two loops one is inner in another $O(V^2)$, so by maximum Theorm $T(n) = O(V^2)$