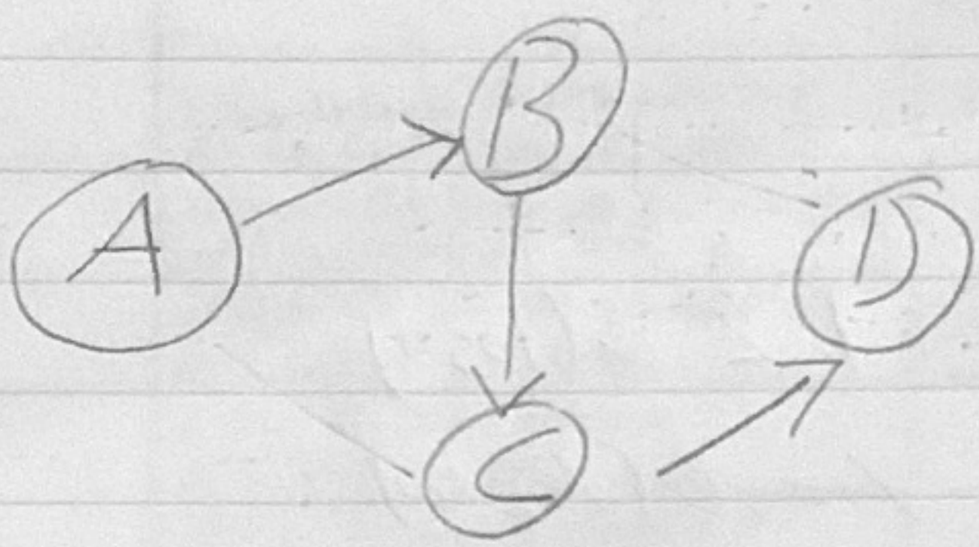
HW3

1)



1-

DFS: A, B, C, D
BFS: A, B, C, D

2. A → B → D
   ↘ C

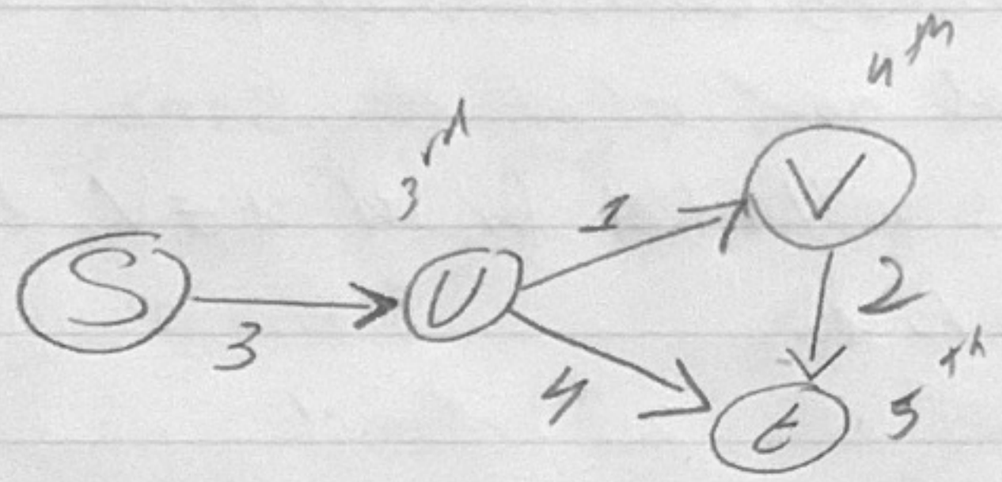
DFS: A, B, D, C
BFS: A, B, C, D

2)

	d[s]	d[u]	d[v]	d[t]	P[s]	P[u]	P[v]	P[t]
When entering the first while loop for the first time, the state is	0	∞	∞	∞	None	None	None	None
Immediately after the first element of D is added, the state is:	0	3	∞	∞	None	S	None	None
After the second element of D is added, the state is:	0	3	4	7	None	S	u	u
After the third element of D is added the state is:	0	3	4	6	None	S	u	v
After the fourth element of D is added, the state is:	0	3	4	6	None	S	u	v

The path should return s->u->v->t

d[t]=6



3) Also:
~~Any~~ Build the complete graph with weights that are equal to the less ~~G(V,E)~~ G(V,E,w). V= $C_0, \dots C_n$ E= (C_i, C_j): i≠j, and $w(C_i, C_j) = f_{ij}$. Use Dijkstras algorithm from C_s; return shortest $C_s \to C_t$ path.

Correctness: Any path from $C_s \to C_t$ in G indicates a sequence of exchanges and that weight total is the sum of the fees needed to perform these exchanges. This makes it ideal to find path $C_s \to C_t$ in min weight f_{ij}s are positive, so this is a valid input for Dijkstras alg.

Running time: $O(n^2)$ total. All path pairs are possible we have $m = \binom{n}{2} = \Theta(n^2)$ edges - this is also how long it takes to build G. Dijkstras also takes $O(n^2 + n\log n) = O(n^2)$ if we use the Fibonacci heap implementation.

4)

a. if v and v' are influential, there is a path from v to v' from v' to v. Thus v and v's are the same and strongly connected component. Also if v is influential and v' is the same SCC as v, then v' is influential for all $u \in V$, there is a path from v' to v to u.

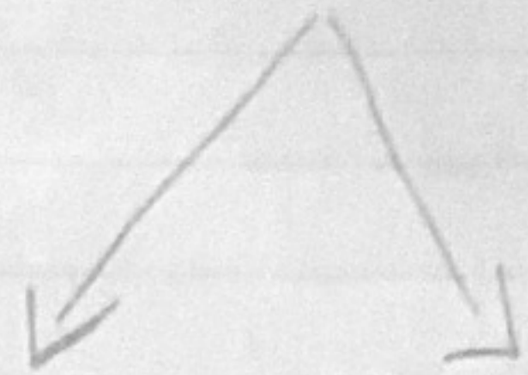
b.

```
findInfluentialPerson(G):
    L = V
    while L is not empty:
        v = any vertex in L
        VISTED = []
        DFS(VISTED, L, v)
        L.pop(v)
    return v
```

```
DFS(VISTED, L, S):
    if S == NULL or S
        is not in L:
            return
    VISTED.push(S)
    for v in neighbor:
        DFS(Visted, L, v)
```

5)

J_0	J_1	J_2	J_3
2	2	3	3
60	40	100	80



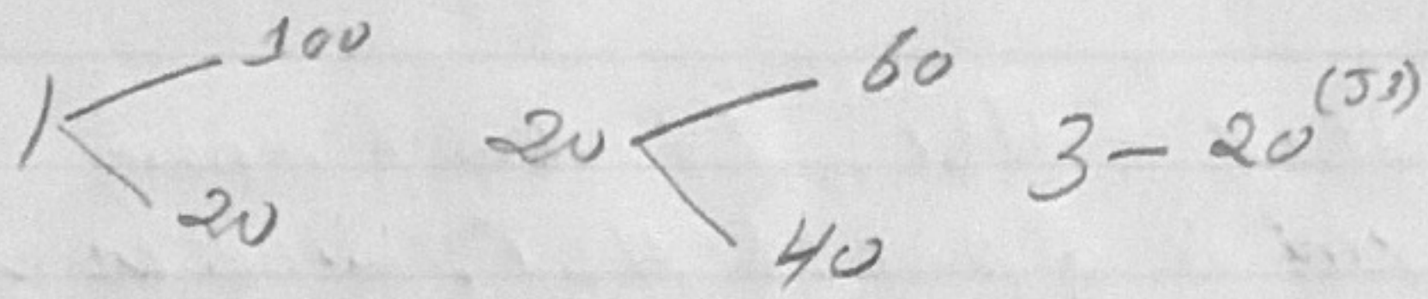
J_6 J_0

(4)

150

x

J_1	J_2	J_3	J_4	J_5
2	0	3	2	0
60	100	20	40	20

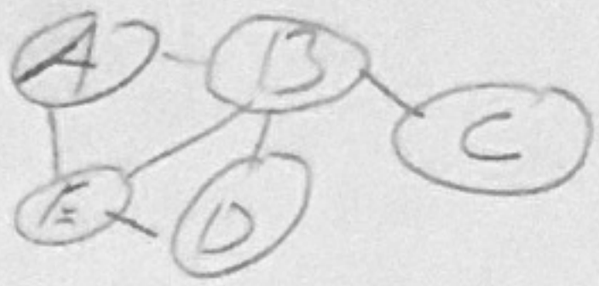


1st Sec → J_1 100, J_5 20

2nd Sec → J_1 60, 40

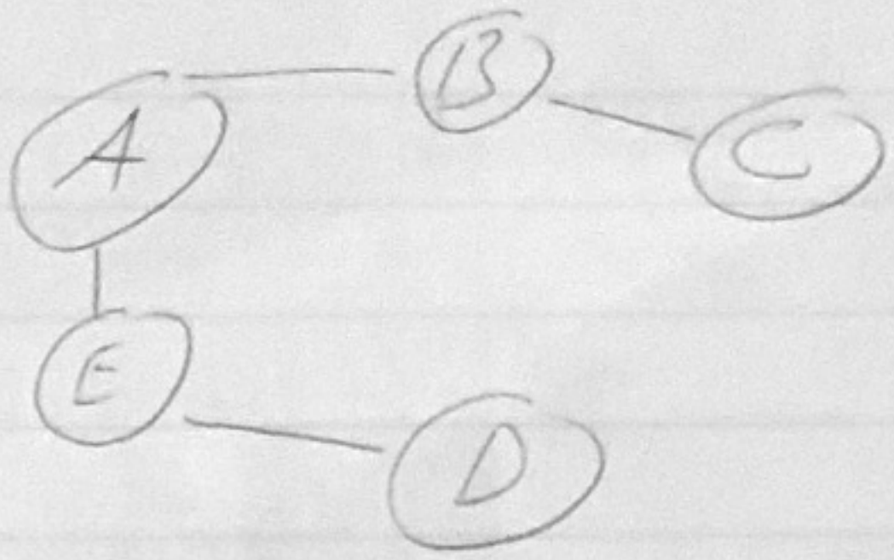
3rd Sec → J_3 20

$J_2 \rightarrow J_1 \rightarrow J_3$



If I remove
B-D
B-E
it still

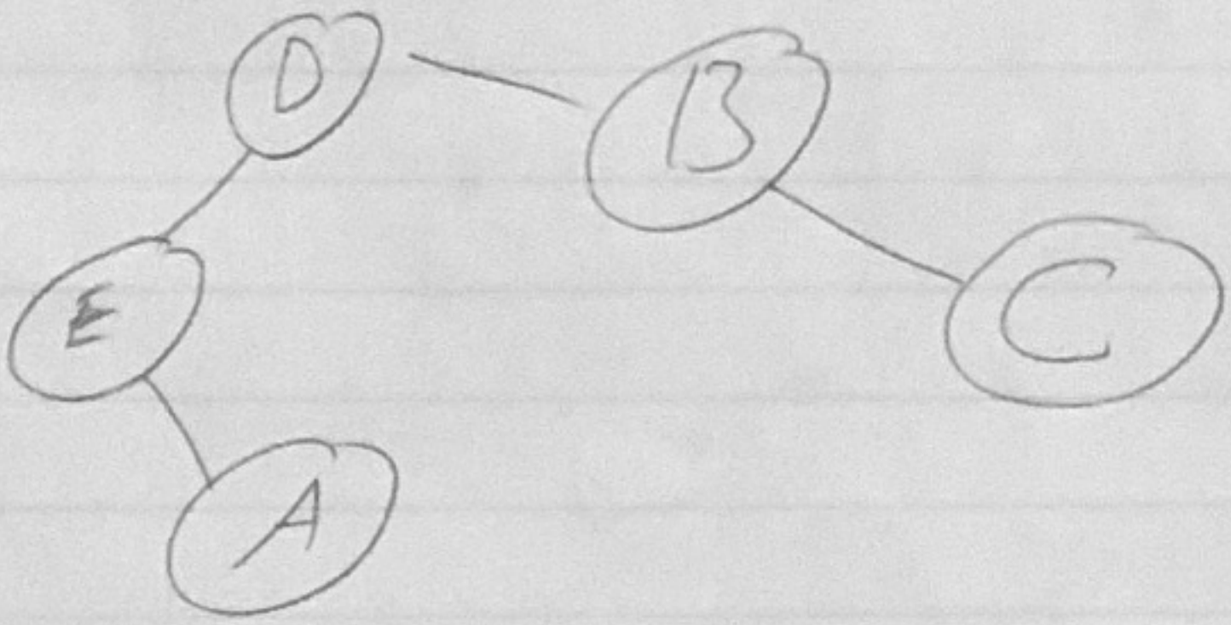
6)



A -> B ✓
A - E ✓
B -> E X
E -> D ✓
B - D X
B - C ✓

a- yes MST

b- No MST



a) A does return a MST. It doesn't remove an edge that is a part of MST. Since we don't remove edges in an increasing order, the weight of every edge on the cycle must be less than or equal of e.

b) It doesn't return a MST, It doesn't store the edges in a array.