I certify that the work submitted with this exam is mine and was generated in a manner consistent with this document, the course academic policy on the course website, and the Rutgers academic code. I also certify that I will not disclose this exam to others who are taking this course and who will take this course in the future without the authorization of the instructor.

Date: ____12/16/2022____________________

Name (please print): ____Dong Shu____________________

Signature: _____Dong Shu____________________

Name: ______________________ **Section:** ______________

Problem Number(s)	**Possible Points**	**Earned Points**
1	10	
2	10	
3	5	
4	10	
5	10	
6	10	
7	10	
8	10	
9	10	
10	15	
Total	100	

Exam Time: 15 hours, 10 problems (15 pages, including this page)

- Write your name on this page and the last page, put your initials on the rest of the pages.
- If needed, use the last page to write your answer.
- Show your work to get partial credits.
- Show your rational if asked. Just giving an answer can't give you full credits.
- You may use any algorithms (procedures) that we learned in the class.
- Keep the answers as brief and clear as possible.

Name: ________________________ **Section:** ______________

1. **(10 points) Circle True or False (no partial credit)**

   (1) [✓ T] F Given a directed graph G and a vertex v in the graph, breath first search (BFS) can be used to detect if v is part of a cycle in the graph.

   (2) T [✓ F] Let P be a shortest path from some vertex s to some other vertex t in a directed graph. If the weight of each edge in the graph is *decreased* by one, then P will still be a shortest path from s to t.

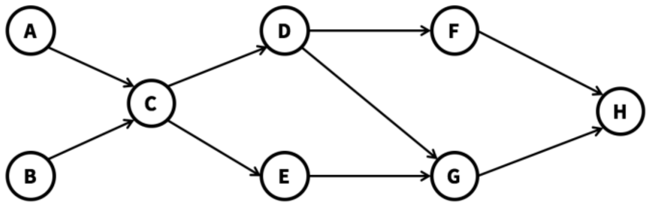
   (3) [✓ T] F Kruskal's algorithm is always correct even in graphs with negative edge weights.

   (4) T [✓ F] For any flow network, there is only one unique way to assign flow value to the edges so as to achieve the maximum flow for the network.

   (5) T [✓ F] NP problems are those problems that cannot be solved in polynomial time.

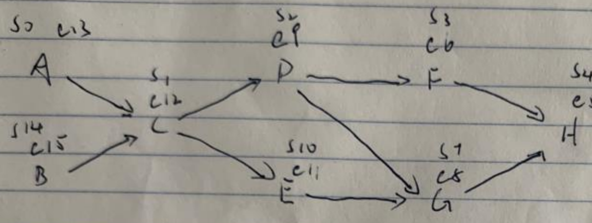
2. **(10 points) Graph Basics**

   (1) (5 points) DFS and its application: Run the DFS-based topological sorting algorithm on the following graph. Whenever there is a choice of vertices, choose the one that is alphabetically first. Give the resulting topological ordering of the vertices.



Name: ______________________ **Section:** ______________

2.

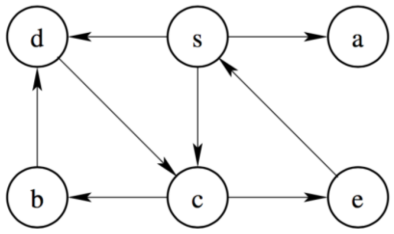


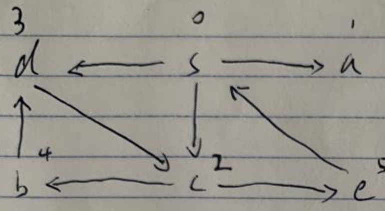
Reverse topological: H , F , G , D , E , C , A , B

topological : B , A , C , E , D , G , F , H

Name: ______________________ **Section:** ______________

(2) (5 points) BFS and its application: Run BFS algorithm on the following graph starting with vertex s. Whenever there is a choice of vertices, choose the one that is alphabetically first. What is the order that the vertices are visited? What is the shortest path from vertex s to vertex b?





L[0] : s
L[1] : a , c , d
L[2] : b , e

order: s, a, c, d, b, e

shortest path s to b : $s \to c \to b$ 2 steps

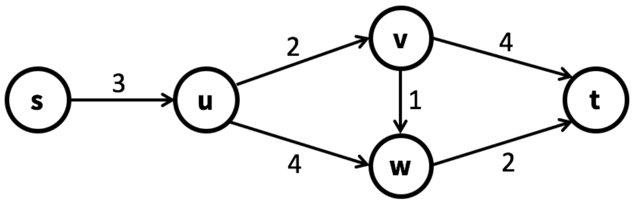
Name: ______________________ **Section:** ______________

3. **(5 points) Intractable Problems**
   Here is a statement about NP-complete problems: "Some NP-complete problems are polynomial-time solvable, and some NP-complete problems are not polynomial-time solvable." Decide if this statement if true or false and provide an explanation of your decision.

The statement is false, we can prove it by contradiction. If some NP-complete problems are polynomial-time solvable, and some NP-complete problems are not polynomial-time solvable, then that means some all of the NP problems can be solved in polynomial time, because NPC problems are harder than NP problems. Therefore, if we can solve some NPC problems in polynomial, then we can solve all of the NP problems in polynomial time. In this case, all of the NP problems will equal to P problems, because they can all solved in polynomial time. However, in the theory of P = NP, the graph also shows that P = NP = NPC. Therefore, if this statement wants to believe P = NP, then it should say "all of NPC problems are polynomial-time solvable". Because in P=NP theory, P=NP=NPC, which contradicts the statement.
On the other hand, if this statement want to believe P != NP theory, then the statement should say "all of NPC problem are not polynomial-time solvable" because some NP problems are not polynomial-time solvable and NPC problems are even harder than NP problems. Therefore, it also contradicts with the statement. So the statement is false.
Also, according to polynomial reduction, if a NPC problem is polynomial-time solvable, then that means we can solve all of the NPC problems in polynomial time. In this case, we will not have this statement, which further proved the statement is false.

Name: ______________________ **Section:** ______________

4. **(10 points) Dijkstra's Algorithm**
We have the following directed graph G, where the number on each edge is the cost of the edge.



(1) (8 points) Step through Dijkstra's Algorithm on the graph starting from vertex s, and complete the table below to show what the arrays *d* and *p* are at each step of the algorithm. For any vertex x, *d[x]* stores the current shortest distance from s to x, and *p[x]* stores the current parent vertex of x.

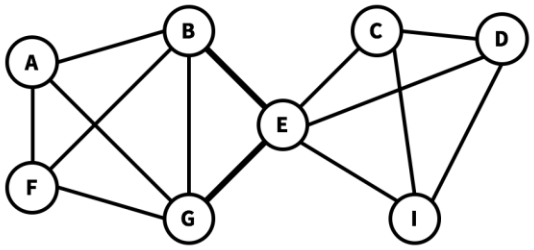
	d[s]	d[u]	d[v]	d[w]	d[t]	p[s]	p[u]	p[v]	p[w]	p[t]
Initialization	0	∞	∞	∞	∞	None	None	None	None	None
Immediately after iteration 1	0	3	∞	∞	∞	None	s	None	None	None
Immediately after iteration 2	0	3	5	7	∞	None	s	u	u	None
Immediately after iteration 3	0	3	5	6	9	None	s	u	v	v
Immediately after iteration 4	0	3	5	6	8	None	s	U	v	w
Final status	0	3	5	6	8	None	s	u	v	w

(2) (2 points) After you complete the table, provide the returned shortest path from s to t and the cost of the path.

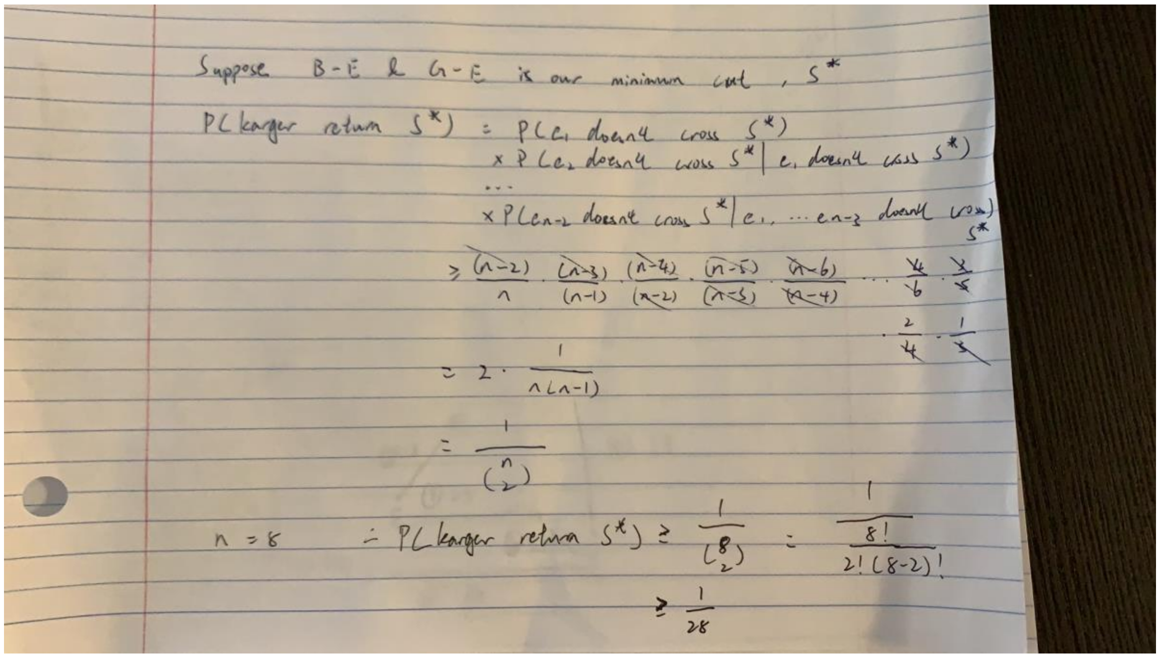
s -> u -> v -> w -> t
cost: 8

Name: ______________________ **Section:** ______________

5. **(10 points) Randomized Algorithm**
Suppose we have the following undirected graph, and we know that the two bolded edges (B-E and G-E) constitute the global minimum cut of the graph.



(1) (5 points) If we run the Karger's algorithm for just one time to find the global minimum cut, what is the probability for the algorithm to find the minimum cut correctly? Please show your reasoning process, just showing the final answer will get no point.



Suppose B-E & G-E is our minimum cut, S^*

$P($karger return $S^*) = P(e_1$ doesn't cross $S^*)$
$\times P(e_2$ doesn't cross $S^* \mid e_1$ doesn't cross $S^*)$
...
$\times P(e_{n-2}$ doesn't cross $S^* \mid e_1, \dots e_{n-3}$ doesn't cross $S^*)$

$$\geq \frac{\cancel{(n-2)}}{n} \cdot \frac{\cancel{(n-3)}}{(n-1)} \cdot \frac{\cancel{(n-4)}}{\cancel{(n-2)}} \cdot \frac{\cancel{(n-5)}}{\cancel{(n-3)}} \cdot \frac{\cancel{(n-6)}}{\cancel{(n-4)}} \cdots \frac{\cancel{4}}{\cancel{6}} \cdot \frac{\cancel{3}}{\cancel{5}} \cdot \frac{2}{\cancel{4}} \cdot \frac{1}{\cancel{3}}$$

$$= 2 \cdot \frac{1}{n(n-1)}$$

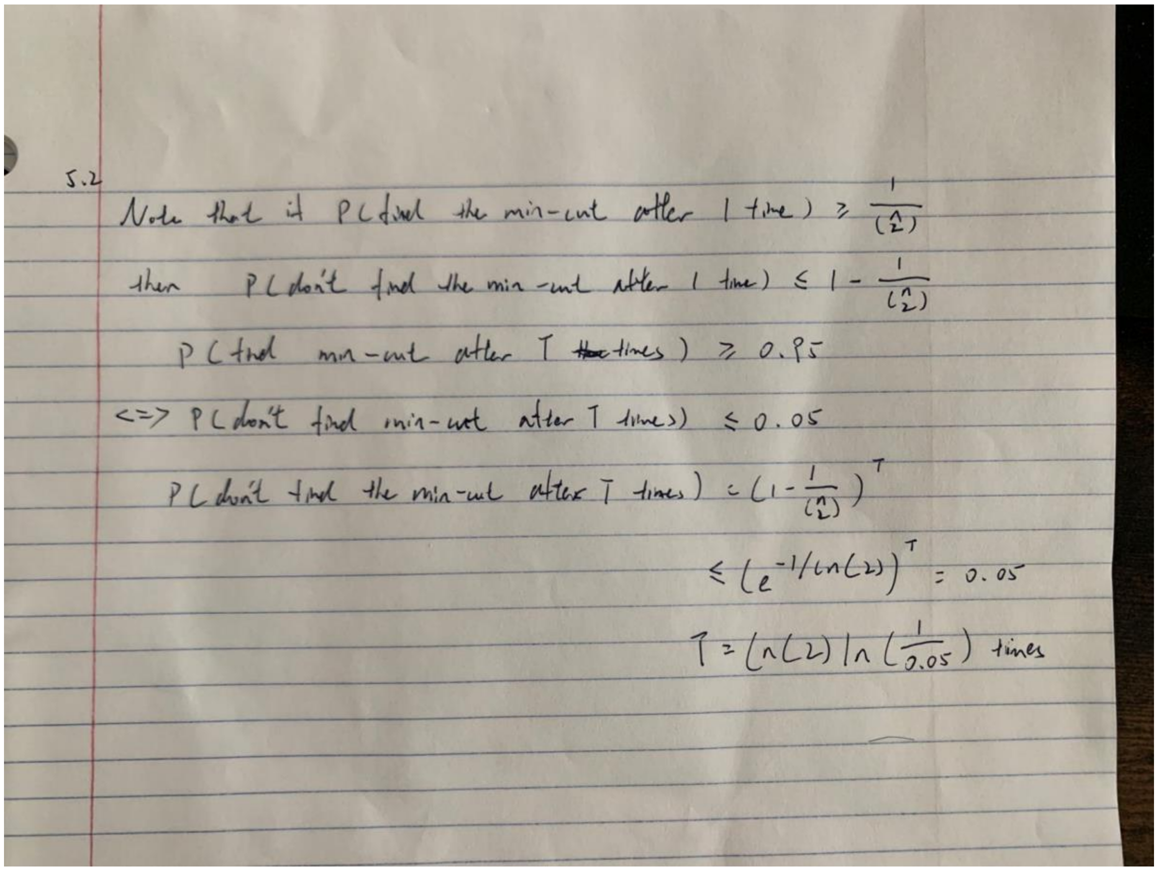
$$= \frac{1}{\binom{n}{2}}$$

$n = 8$ $\quad \therefore P($karger return $S^*) \geq \frac{1}{\binom{8}{2}} = \frac{1}{\frac{8!}{2!(8-2)!}}$

$$\geq \frac{1}{28}$$

Name: ______________________ **Section:** ______________

(2) (5 points) How many times do we need to run the Karger's algorithm if we want to guarantee that the probability of success is greater than or equal to 0.95, by "success" we mean that there is at least one time the Karger's algorithm correctly found the minimum cut. Please show your reasoning process, just showing the final answer will get no point. [You do not have to work out the exact value of a logarithm]



5.2

Note that if $P(\text{find the min-cut after 1 time}) \geq \frac{1}{\binom{n}{2}}$

then $P(\text{don't find the min-cut after 1 time}) \leq 1 - \frac{1}{\binom{n}{2}}$

$P(\text{find min-cut after T}$ ~~the~~ $\text{times}) \geq 0.95$

$\iff P(\text{don't find min-cut after T times}) \leq 0.05$

$P(\text{don't find the min-cut after T times}) = \left(1 - \frac{1}{\binom{n}{2}}\right)^T$

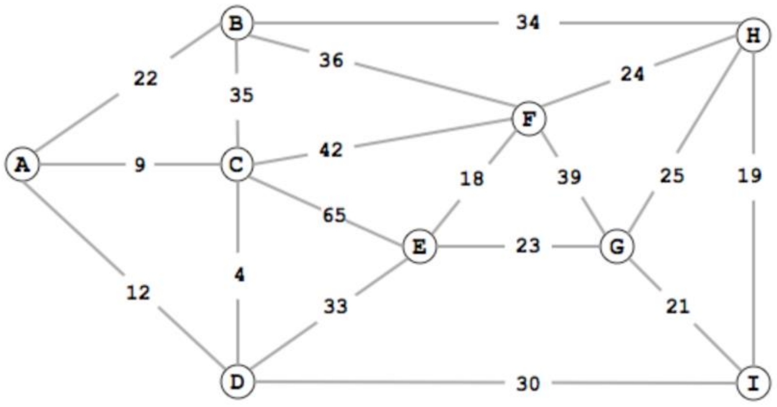
$\leq \left(e^{-1/\binom{n}{2}}\right)^T = 0.05$

$T = \binom{n}{2} \ln\left(\frac{1}{0.05}\right)$ times

6. **(10 points) Minimum Spanning Tree**

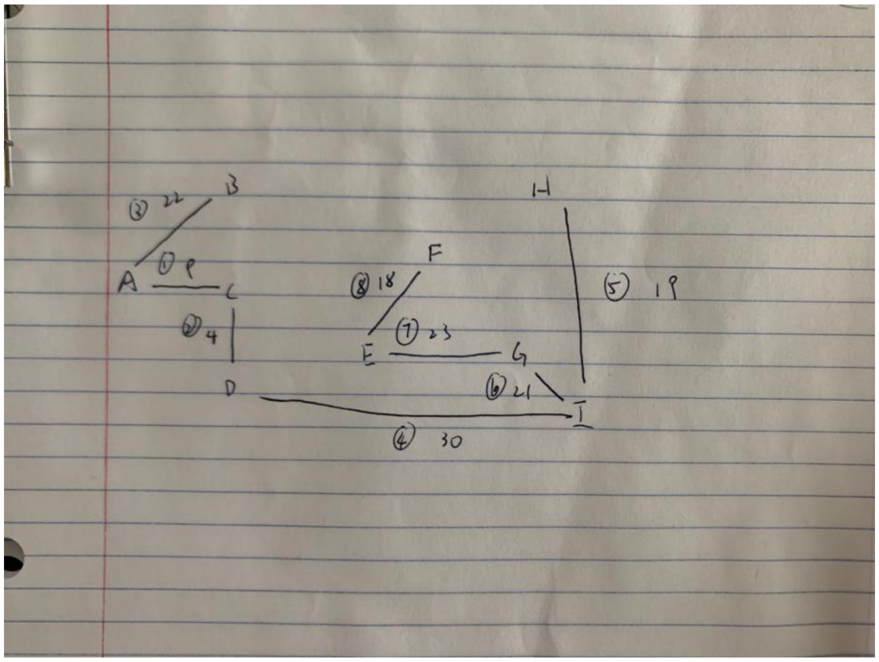
Consider the following weighted undirected graph:

Name: ______________________ **Section:** ______________



(1) (5 points) Assume we run Prim's MST algorithm starting at vertex A. List the edges that get added to the tree in the order in which the algorithm adds them. [You can denote an edge by its two adjacent vertices, e.g., (A, B)].

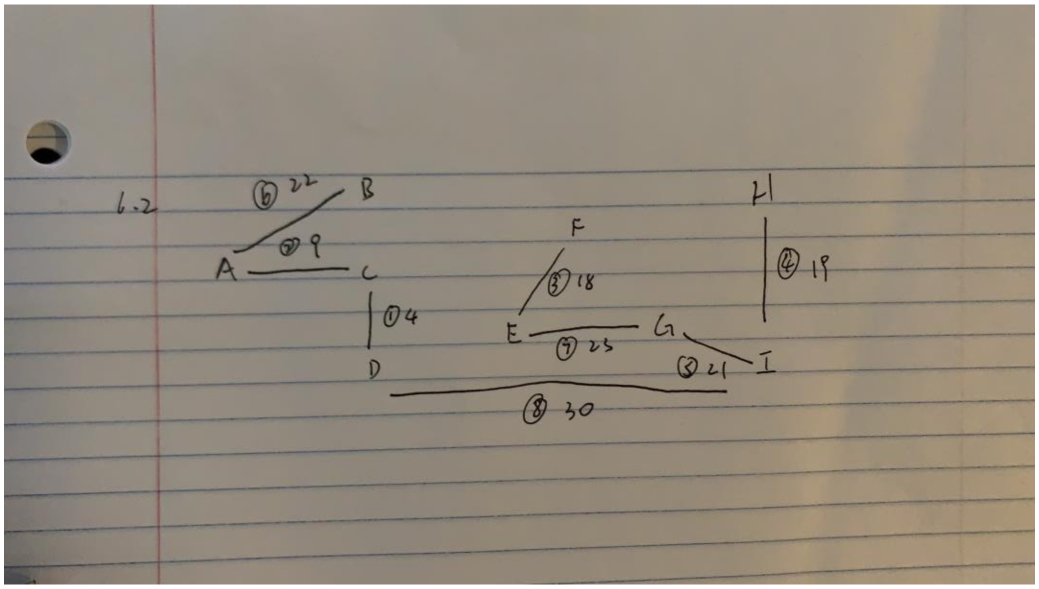
AC: 9
CD: 4
AB: 22
DI: 30
IH:19
IG: 21
GE: 23
EF: 18



Name: ______________________ **Section:** ______________

(2) (5 points) Now we use Kruskal's algorithm to find the MST, also, list the edges that get added to the tree in the order in which the algorithm adds them. [You can denote an edge by its two adjacent vertices, e.g., (A, B)].

CD: 4
AC: 9
EF: 18
HI: 19
GI: 21
AB: 22
EG: 23
DI: 30



7. **(10 points) Strongly Connected Components**
You are hired by a game design company and one of their most popular games is *The Journey*. The game has a ton of quests, and ***for a player to win, the player must finish all the quests***.

There are a total of *N* quests in the game. Here is how the game works: the player can *arbitrarily* pick one of the *N* quests to start from. Once the player completes

Name: ______________________ **Section:** ______________

a quest, they unlock some other quests. The player can then choose one of the unlocked quests and complete it, and so on.

For instance, let's say that this game had only 4 quests: A, B, C, and D. Let's say that after you complete

- quest A, you unlock quests [B, D].
- quest B, you unlock quests [C, D].
- quest C, you unlock nothing [ ].
- quest D, you unlock quest [C].

Is this game winnable? Yes, because of the following scenario:

The player picks quest A to start with. At the end of the quest A, the unlocked list contains [B, D]. Say that player chooses to do quest B, then the unlocked list will contain [C, D]. Say that player chooses to complete quest C, then the unlocked list will contain quest [D]. Finally, player finishes quest D.

Note that if the player had started with quest C instead of quest A, they would have lost this game, because they wouldn't have unlocked any quest and would be stuck. *But the game is still winnable, because there is* ***at least one*** *starting quest which makes the player win*.

The Journey has *N* quests, enumerated as $\{n_1, n_2, \ldots, n_N\}$. We construct a directed graph G for the game, where each node in G represents a quest. If completing quest n_i will unlock quest n_j, then there is a directed edge $<n_i, n_j>$ from n_i to n_j in the graph. Suppose the total number of edges in the graph is *M*.

(1) (6 points) We call a quest as a *lucky quest* if starting from this quest will allow the player to win the game. In the example above, quest A is a lucky quest. Show that (a) All lucky quests are in the same strongly connected component of G, and that (b) Every quest in that component is a lucky quest. [We are expecting a short but rigorous proof for both claims]

We know that strongly connected components means for all pairs of vertices u and v, there's a path from u to v and a path from v to u.

We can prove (a) all lucky quests are in the same strongly connected component of G by using contradiction. If not all lucky quests in G are in the same SCC, then that means there exist other lucky quests outside of this SCC. However, the SCC metagraph is a directed acyclic graph, which means the lucky quest that is outside of the SCC can either unlock the SCC or be unlocked by SCC, but not both, because there is no cycle path between the SCC and the outside lucky quest. Otherwise, they can merge into one SCC. In this case, it will leave us with two cases, the first one is the

Name: ________________________ **Section:** ________________

outside lucky quest can only unlock the SCC. In this case, the original SCC will no longer have lucky quest, because it violate the definition of lucky quest, which is that "if starting from this quest will allow the player to win the game". Since outside lucky quest can only unlock the SCC, then that means if we start at the SCC, we can not unlock the outside lucky quest. In this case, the outside lucky quest will form its own SCC, and this new SCC will have all the lucky quests. The other case is when the outside lucky quest can only be unlocked by the SCC. Then this outside lucky quest will not be a lucky quest, because it cannot unlock the SCC, which violates the definition of lucky quest. Therefore, if the outside lucky quest can only unlock the SCC, then outside lucky quest will form its own SCC. If the outside lucky quest can only be unlocked by the SCC, then the outside lucky quest will not be lucky quest. However, if a outside lucky quest can both unlock and be unlocked by SCC, then the outside lucky quest and SCC can collapse into one SCC. Therefore, we used contradiction to prove (a).

We can prove (b) every quest in that component is a lucky quest by contradiction. If not every quest in that SCC is lucky quest. Then it means that there exists not lucky quest in this SCC. In this case, this unlucky quest will not have a direct path from u to all other quest v in G. This means in this SCC, for all pairs of vertices u and v, there may not be a path from v to u. In this case, it violate the definition of SCC, which is for all pairs of vertices u and v, there's a path from u to v and a path from v to u. Therefore, this SCC should only contain lucky quest and unlucky quest should not be in this SCC. In this case, our contradiction proved (a) and (b)

Name: ______________________ **Section:** ______________

(2) (4 points) Suppose there is at least one lucky quest. Give an algorithm that runs in time O(N+M) and finds a lucky quest. You may use any algorithm we have seen in class as subroutine. [We are expecting pseudocode or a description of your algorithm, and a short justification of the runtime]

I will use Kosaraju's algorithm to find the lucky quest. We first use DFS to go through every node, and store the start and end time for each node. After this process, we will have a list of nodes from smallest end time to biggest end time. It will give us running time O(|N | + |M|). Then we reverse the graph and erase the status of the node, which will give us running time O(|N | + |M|). Then we do DFS again, and this time we will start at the node with highest end time, which is the last node of the list. Once we find a complete SCC, then that means we find the lucky quest, then we will return it. It will give us running time O(|N| + |M|).
Thus, we will have total running time of O(3(|N | + |M|)) = O(|N | + |M|).

```
cur_time = 0
list = []
algorithm dfs(u, cur_time):
    u.start_time = cur_time
    cur_time += 1
    u.status = 'in_progress'
    for v in u.neighbors:
        if v.status is 'unvisited':
            cur_time = dfs(v, curtime)
            cur_time += 1
    u.end_time = cur_time
    u.status = 'done'
    list.append(u)
    return cur_time

algorithm reverse_edge(G):
    all edge in G:
        reverse edge
    for all node in G:
        empty status
    return G
```

Name: ______________________ **Section:** ______________

```
scc = []
algorithm dfs_scc(u, G):
    u.status = 'in_progress'
    for v in u.neighbors:
        if v.status is 'unvisited':
            scc.append(dfs_scc(v, G))
    u.status = 'done'
    return scc

algorithm Kosaraju(G)
    u = Random_choose_node(G)
    dfs(u, cur_time)
    G = reverse_edge(G)
    last_node = list.length()
    return dfs_scc(list[last_node - 1], G)
```

8. **(10 points) Greedy Algorithms**
On the Christmas eve, Santa Claus will start working to deliver gifts to the lovely kids. Different from previous years that he secretly delivered the gifts, this year, he indeed wants to openly deliver the gifts to the kids and wish them a merry Christmas when delivering the gifts.

However, the kids each have very different sleep schedules. Kid i is awake only in a single closed time interval $[a_i, b_i]$. Santa doesn't want to disturb the sleeping kids, so each time he will find an awake kid, give the gift to the kid, wish the kid a merry Christmas, and talk with the kid until he/she falls asleep. However, Santa does want to meet and talk with as many kids as possible.

Name: ____________________ **Section:** ______________

Design a greedy algorithm which takes as input the number of kids n, and the n lists of intervals $[a_i, b_i]$, and outputs the maximum number of kids m that Santa can talk to. You are expected to write the pseudocode of your algorithm.

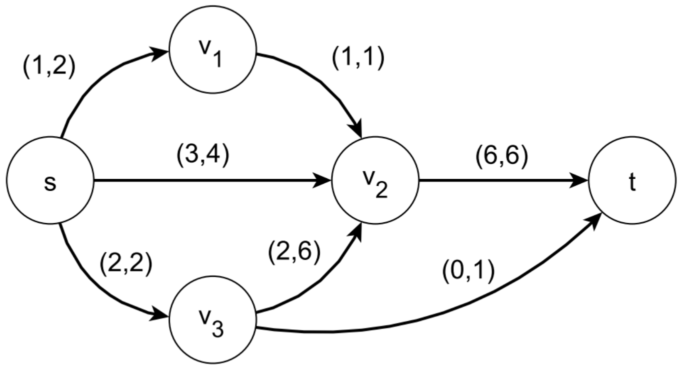
First, we need to sort the original kids array by their sleeping time b, from smallest to biggest. Then we will traverse this sorted array, and every time when we find a suitable kid, we will add one to M.

```
Greedy Algorithms(n):
    M = 0
    Sleep_time = 0
    n = sort all kids by their sleeping time, b, from smallest to biggest

    For I in n:
            If n[i].ai >= sleep_time:
                    M += 1
                    Sleep_time = n[i].bi
    Return M
```

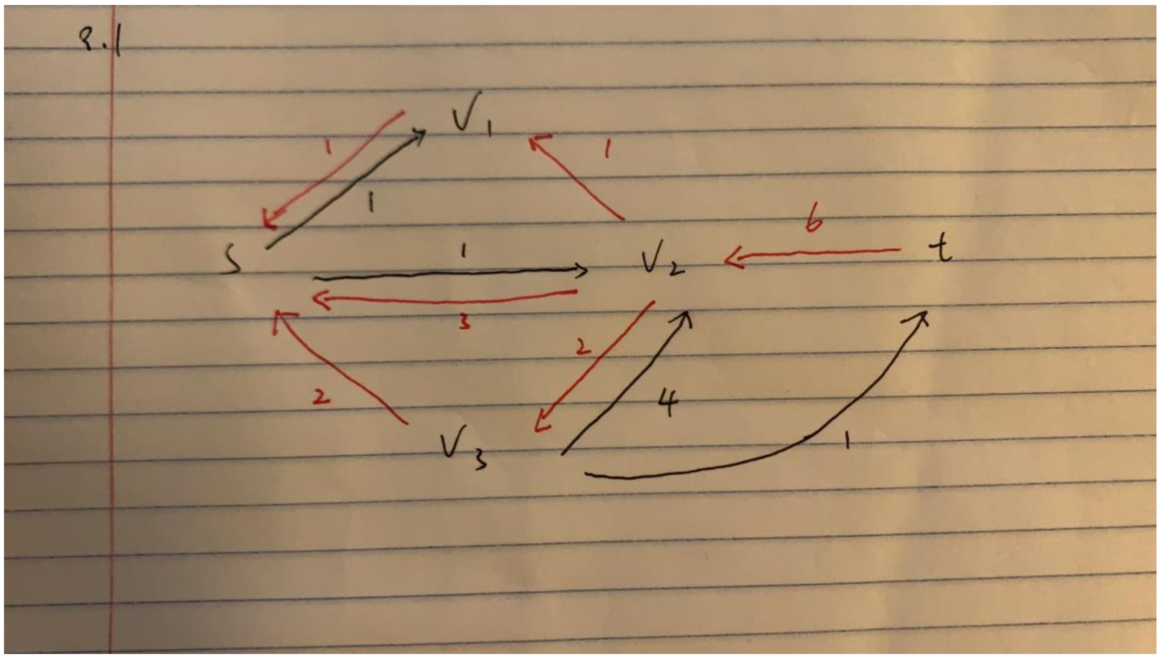
Name: ______________________ **Section:** ______________

9. **(10 points) Max-flow and Min-cut**
Consider the following flow network.



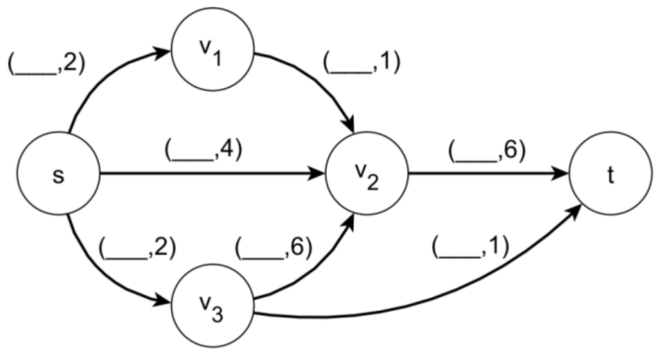
The figure describes a flow f and the capacity of the edges: if (x, y) appears next to an edge e, then the capacity of the edge c_e is y, and the flow f_e that goes through e in f is x. For example, if $e = (s, v_1)$, then $c_e = 2$ and $f_e = 1$.

(1) (4 points) Draw the residual network G_f of the above flow f. [Draw a graph containing all the nodes, edges, and the values on the edges]

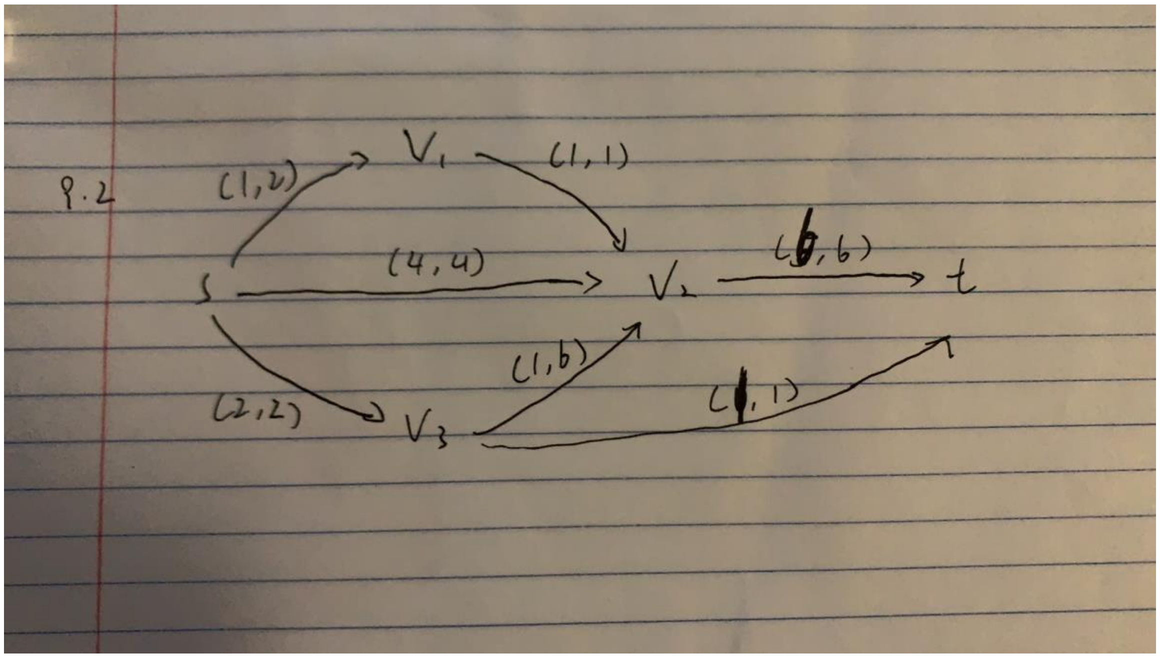


Name: ________________________ **Section:** ________________

(2) (3 points) Find an augmenting path that will increase the flow by 1. You only need to list the vertices in the path and indicate the resulting flow in the following figure (using the same notation as the above figure).



Name: ______________________ **Section:** ______________



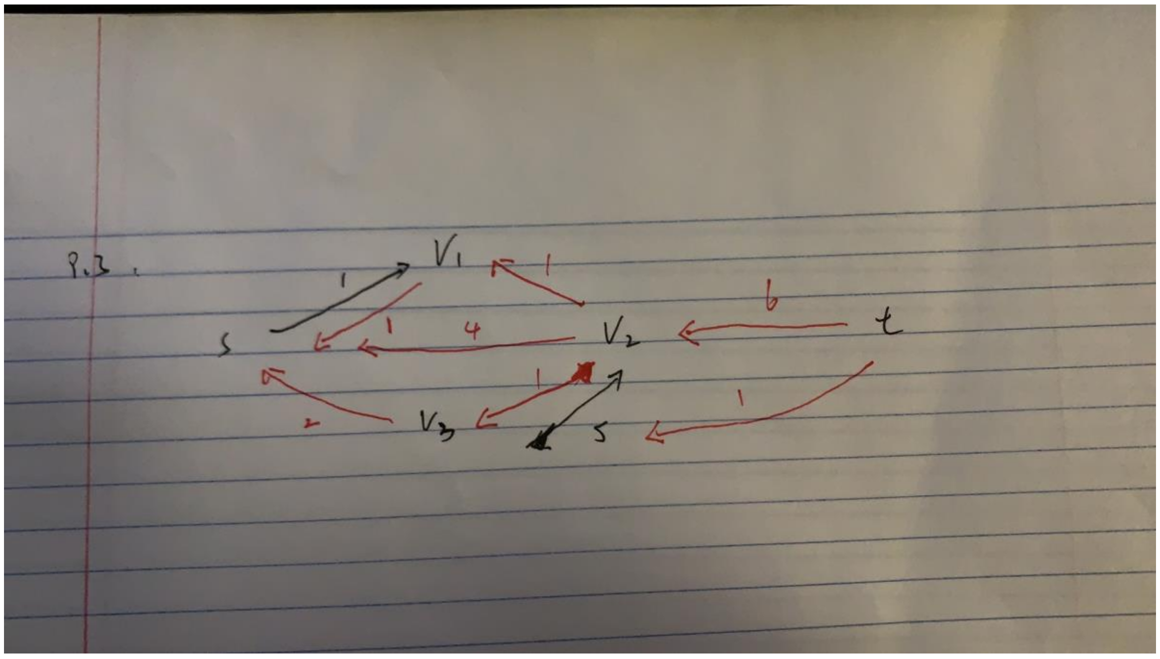
The path that I changed is S – v2 (4,4), v3 – v2 (1,6), v3 – t (1,1). The vertices are s, v2, v3, t

(3) (3 points) Find a minimum *s*-*t* cut in the graph (where the weight of an edge is its capacity). Briefly justify why the cut you found is a minimum cut.

We can only find the minimum cut in the 9.2 graph. We cannot find the minimum cut in the original graph, because the flow in the original graph is not maximum yet.

On the 9.2 graph, we can choose to cut (v2, t) and (v3, t). First, after we draw the residual network, we know that we cannot reach t from s and there is no more augmenting path. In this case, we know our current flow 7 is the maximum flow. Second, we know that our min cut also has flow 7. In this case, our max flow = min cut, and it meet s-t cut requirement. Thus, we know our cut is a min cut.

Name: ______________________ **Section:** ______________



10. **(15 points) Dynamic Programming**
A country has coins with k denominations $1 = d_1 < d_2 < \cdots < d_k$, and you want to make change for n cents using the smallest number of coins.

For example, in the United States we have $d_1 = 1,\ d_2 = 5,\ d_3 = 10,\ d_4 = 25$, and the change for 37 cents with the smallest number of coins is 1 quarter, 1 dime, and 2 pennies, which are a total of 4 coins.

To solve for the general case (change for n cents with k denominations $d_1 \dots d_k$), we refer to dynamic programming to design an algorithm.

Name: ______________________ **Section:** ______________

(1) (5 points) We will come up with sub-problems and recursive relationship for you. Let $C[n]$ be the minimum number of coins needed to make change for n cents, then we have:

$$C[n] = \begin{cases} \underset{\forall\, d_i \le n}{\text{minimum}}\{C[n-d_i]+1\}, if\ n > 0 \\ 0, \quad if\ n = 0 \end{cases}$$

Explain why the above recursive relationship is correct.
[Formal proof is not required]

For n = 0, we know that we need 0 coin, so our starting point is correct.

For n > 0, starting at n, at each step, we need to check for all coin, di <= n, which coin we want to choose. After we choose the coin, we will delete n with di value. Then we will add 1 to the number of coins and store in the C[n-di]. Since, in the end, we want the minimum number of coins. Therefore, at each step, we will choose the biggest di that <= n. In this case, we will guarantee to have minimum number of coins, because at each step we are using the minimum coin to subtract biggest value from n.

Name: ______________________ **Section:** ______________

(2) (5 points) Use the relationship above to design a dynamic programming algorithm, where the inputs include the k denominations $d_1 \dots d_k$ and the number of cents n to make changes for, and the output is the minimum number of coins needed to make change for n.
Provide the pseudocode of your algorithm and briefly justify the runtime of your algorithm using big-O notation.

The Coin algorithm will have O(nk) complexity, because as we can see, we have one while loop that loop when n > 0. Inside of the while loop, we need to loop through k to find the most suitable coin to select. Therefore, we will have O(nk) complexity.

```
Coin(n, k):
    If(n == 0){
        Return 0
    While n > 0
        C[n] = $\underset{\forall\, d_i \le n}{\text{minimum}}\{C[n - d_i] + 1\}$
        If C[n] changed:
            //we need to use the n after the subtraction
            //to continue finding the C[n]
            n = n - di
    return C[n]
```

Name: ______________________ **Section:** ______________

(3) (5 points) Adapt your algorithm above by tracking some useful information during the DP procedure, so that it returns the actual method (i.e., the number of coins for each denomination) to change for *n* cents, not just the minimum number of denominations.

```
Coin(n, k):
    Cent = []
    If(n == 0){
        Return 0
    While n > 0
        C[n] = $\underset{\forall\, d_i \le n}{\text{minimum}}\{C[n - d_i] + 1\}$
        If C[n] changed:
            n = n – di
            cent.append(di)

    //print out the number of coins for each denomination
    for I in cent:
        count = cent.count(i)
        print(“the number of coins”, I, “ are :” count)
    return C[n], cent
```

Name: ________________________ **Section:** ________________

(You may use this page to write answers if needed. Please mark the problem number clearly)