Question 1

- Overall, performance on this question was good, with many students attaining full or close-to-full marks.
- Several students attempted a greedy approach to the problem, which either did not consider how to reallocate students if the faculty capacities were met (which is handled by augmenting paths in residual graphs in max flow), or attempted to reallocate without properly considering the extra complexity it introduces.
- Most max-flow solutions were well-done, though many students failed to mention how the students could be recovered from the resulting flow. It is important to re-read the question after writing a solution to ensure you have answered it completely.
- Some students stated that the flow was bound by m to justify complexity. It is expected that you explain what causes the max flow to be bound by m. E.g. the total flow leaving the source or entering the sink being bound by m.
- Some students interpreted the constraint that each student was between 1 and 4 societies as allowing a total of 4 flow through each student vertex, which then allowed them to represent multiple societies. By asking you to find the "m students" to attend the event if it can run, it is implied that each society sends a distinct representative. Generally, solutions which allowed multiple units of flow through a student vertex didn't properly consider the faculty diversity constraints, allowing the students to contribute up to 4 units of flow to those constraints.
- Several students attempted to construct a graph with three vertex sets (for students, societies, and faculties), but used societies (resp. faculties) as the middle group. This generally results in there being no way to ensure each unit of flow through a student also goes through their corresponding faculty (resp. society), so there was no way to guarantee that the students with saturated edges could all attend (since a student might contribute to the wrong faculty's diversity constraint, or be a representative of the wrong society). You can argue that the students can be matched properly if the flow is m still, but it is a much more involved argument, and you would need to provide an efficient algorithm to re-arrange the students to find the right allocation.
- Each u_i is unbounded, so you cannot use the faculty diversity constraint edges to constrain the maximum flow through the graph.
- It is expected that you clearly define the vertices, edges, capacities, and direction of all edges in your graph construction. For example, state explicitly that society vertices are connected **to** student vertices with an edge **of capacity 1** so that **each society has an edge to all students that are members** (or the other way around, depending on your graph construction).

Question 2

Overall, students performed well on this problem, most used the correct max flow approach, and many scored full or close to full marks accross the board.

2.1

- students mostly got the right solution here. The most common errors were lack of justification of correctness and failing to return an answer (as true or false rather than a flow output).
- some students constructed the graph incorrectly, here many students connected every island to the sink, every island to the source, or created a complete graph with every island connected to every other.
- students who did not use a max-flow approach at all recieved few if any marks in most cases, as it was very clearly a max flow assignment and none of the non-max flow approaches were correct.

2.2 Most students successfully solved this with an in-vertex, an out vertex and a weight of S_i between vertices. Some students were penalised for not fully explaining how to convert from the last graph to in and out vertices, but otherwise this was done well.
Some students either just stated vertex capacities, which was given no marks as these do not work with Ford Fulkerson or Edmonds-Karp. Others used a method of a "counter" for each vertex that decreased whenever it was used. This was penalised heavily, as it fails to account for how ford-fulkerson functions with the residual graph.

2.3 Most students got the correct solution for this. Many did not explain what was to be done in each step of the loop, stating th simply "traverse every island" but those that explicitly stated that each island should be set as the sink or connected to the sink (depending on their approach in 2.1) usually did well.

Question 3

For this question, very few students got full marks but a fair number of students scored more than 25/30. The main reason the students lost marks for this question is not explaining the process of recovering the Lab-Class assignments using the residual graph. However, note that this part was marked quite leniently. Therefore, be aware that a mark review request could adjust your mark downwards. However, if you strongly believe that your mark is incorrect, you are encouraged to submit a review request with a clear explanation of the reason for your review request.

3.1

- This part of the question was pretty straightforward. Most students managed to claim 3 - 4 marks.
- Given that this is a greedy solution, we expect (at least briefly) a correctness justification.

3.2

- For this part, most students managed to get 6 - 7 marks.
- Marks were taken away for not explaining the process of recovering the Lab-Class assignments using the residual graph.

3.3

- This part of the question was a bit tricky. However, most students managed to get full or near full marks.
- Putting careful thought before constructing the graph could have helped to come up with the correct answer.

3.4

- This part involved a bit of thinking as well. Note that array D indicates whether a tutor is available at a particular timeslot or not. It has no relevance to the lab or the class the tutor is going to take. So, students need to be careful in transferring this information into the graph that they construct.
- Those who correctly interpreted the aforesaid information, managed to score 6 - 7 marks for this part.

Question 4

We begin with an exemplar response that is imperfect but reflects the general expectations of this question. The main issue with the following is the monotonicity of the defined function in 4.2 is not well explained however responses of this quality were generally sufficient to achieve reasonably good marks.

Spy Escape

Agency X has sent n spies to Sydney for a secret mission. Before the mission begins, Agency X has prepared $m > n$ secret hideouts throughout the city, of which n of them contain a single emergency escape pod. The hideouts are connected via a network of tunnels, and each hideout can only accommodate one spy at a time. The emergency escape pods can also only accommodate one spy. Everyday, the spies can crawl through at most one tunnel to reach a new hideout.

The tunnels are represented by an adjacency matrix $T[1..m][1..m]$, where

$$T[i][j] = \begin{cases} \texttt{True} & \text{hideout } i \text{ has a tunnel to hideout } j \\ \texttt{False} & \text{otherwise} \end{cases}$$

4.1 A few days after the mission began, all spies have been compromised. The spies are currently scattered around the hideouts in Sydney and must make it to an emergency escape pod within D days to avoid capture.

Design an $O(nm^2D)$ algorithm which determines whether or not all spies can successfully escape.

Solution

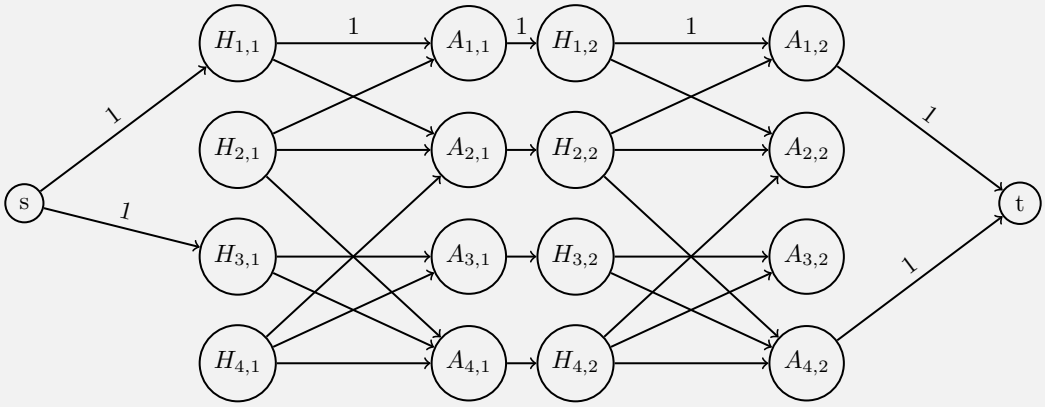
If $D = 0$, then all spies can escape if and only if every spy starts in a hideout with an escape pod.
Otherwise, if $D \geq 1$, construct a graph using an adjacency list with vertices:

- $H_{1,1}..H_{i,j}..H_{m,D}$ representing departures from the i^{th} hideout on the j^{th} day
- $A_{1,1}..A_{i,j}..A_{m,D}$ representing arrivals to the i^{th} hideout on the j^{th} day
- source vertex s and sink vertex t

and edges, all with capacity 1:

- From the source to every day 1 hideout depature vertex where a spy initially occupies that hideout.
  That is, $(s, H_{i,1})$ for every hideout i where a spy originates.
- From every arrival vertex to the departure vertex for the same hideout on the next day.
  That is, $(A_{i,j}, H_{i,j+1})$ for all $i \in [1..m]$ and $j \in [1..(D-1)]$.
- For every day, from a depature vertex to the same day's arrival vertex if a tunnel exists between the hideouts OR if the depature hideout and arrival hideout are the same.
  That is, $(H_{i,k}, A_{j,k})$ if $T[i][j] = \texttt{True}$ OR $i = j$ for all $k \in [1..D]$.
- From every day D arrival vertex with an escape pod to the sink.
  That is, $(A_{i,D}, t)$ if hideout i has an escape pod.

Such a graph should look something like:



Note that some capacities have been omitted for clarity.

We can then compute the max flow, F, of this graph using the Edmonds-Karp algorithm.
If $F = n$, then all spies can successfully escape.
Otherwise, if $F \neq n$, not all spies can escape.

Justication of Correctness

Looking at flow in a graph as created above:

- Flow out of depature vertices represents spies leaving a hideout on some day.
- Flow into arrival vertices vertices represent spies arriving at a hideout on some day.
- Edges from departure to arrival vertices having capacity 1 means flow here can represent a spy travelling from a hideout to another on some day. The capacity here limits each direction of a tunnel to being used by one spy per day.
- Flow conservation at the first departure vertices ensures that each hideout where a spy starts exactly models one spy departing (outgoing flow) since the incoming flow is restricted to 1.
- The edge of capacity 1 between each arrival vertex and the corresponding departure vertex on the next day prevents more than one spy being in a hideout between days.
- By flow conservation, this also means that only one spy can arrive and depart from a hideout on any given day.
- The edges from the final day arrival vertices of hideouts with escape pods to the sink having capacity one means that only one spy can use each escape pod. That is, flow here represents a spy using an escape pod.

With that, all conditions of the problem are met by the flow network, so flow can be interpreted as a valid way for spies to traverse the hideouts and escape.

When we calculate the max flow F, if $F = n$ then n spies made it to an escape pod and thus all spies escaped.
If $F < n$, then not all spies made it to an escape pod.

Justication of Time Complexity

Looking at the upper bound on the number of vertices, number of edges, and flow we find that:

$$V = 2 + 2mD \qquad E = n + m^2D + m(D-1) + n \qquad O(|f|) = O(n)$$
$$O(V) = O(mD) \qquad O(E) = O(m^2D)$$

Note that the flow is limited by the escape pods and initial spy locations and also that $n < m$.
Hence, graph construction, taking $O(V+E)$ time, takes $O(m^2D)$ time and running EK takes $O(\min(VE^2, E|f|))$ time which simplifies to $O(\min(m^5D^3, nm^2D)) = O(nm^2D)$ time.

Therefore, the algorithm runs in $O(nm^2D)$ time.

4.2 Despite the compromise, Agency X still wants to wrap up some tasks in Sydney before the spies escape. The spies will die if they are not done in D days.

Design an $O(nm^2 D \log D)$ algorithm to determine the minimum number of days needed for all spies to successfully escape.

Solution

Define $s(d)$ as whether the spies are able to escape in d days. In other words, the return value of the algorithm from 4.1 with $D = d$ days to escape.

Firstly, if $s(D) = \texttt{False}$, then there is no way for the spies to escape successfully Additionally, if $s(0) = \texttt{True}$, then the minimum days for the spies to escape is 0.
Otherwise, perform a binary search for the value 1 on the function $f(d)$ for $d \in [1..D]$ where $f(d)$ is defined as:

$$f(d) = \begin{cases} 0 & s(d) = \texttt{False} \\ 1 & s(d) = \texttt{True} \text{ and } s(d-1) = \texttt{False} \\ 2 & s(d) = \texttt{True} \text{ and } s(d-1) = \texttt{True} \end{cases}$$

The value of d found by this search is the minimum days required to escape.

Justication of Correctness

The main difficulty with this algorithm is how to define $f(d)$ such that $f(d) = 1$ exactly when d is the minimum days to escape, $f(d) = 0$ for all previous d and $f(d) = 2$ for following d.

This comes from a simple lemma: that if $s(n)$ is true, then $s(d)$ is true for all $d \geq n$. That is, if it is possible to escape in n days, it is always possible to escape in $d \geq n$ days.
This is easy to see because if, after n days, all spies are at an escape pod, they can stay there for any number of days and still be at an escape pod after $d > n$ days.

This lemma means that $s(d)$ is false for all d before the minimum days required to escape and true for days after and including the minimum days needed to escape.
Therefore, the minimum number of days to escape will be exactly when $s(d)$ is true but $s(d-1)$ is false. This is exactly how $f(d)$ is defined.
The other requirements for $f(d)$ also trivially follow from this lemma. It is not possible to escape below the minimum number of days needed, so if $s(d)$ is false it must be because d is less than the minimum. Also, if $s(d)$ is true it must either be the minimum or after the minimum and if $s(d-1)$ is also true then there is clearly a fewer number of days that it is possible to escape in.

This means the definition of $f(n)$ given matches all the needed conditions for a binary of $f(n)$ for the value of 1 for $n \in [1..D]$ will be able to find the minimum days needed to escape assuming escape is possible at all.

The case where escape is not possible is trivially covered by checking if $s(D)$ is false. If it is, then we know D is less than the minimum from the above lemma and thus the spies could never escape.
The other edge case that needs to be covered is when $s(0) = \texttt{True}$ as, according to the lemma, this means $f(d) = 2$ for all $d \in [1..D]$ which would result in a failed search. In this case however, since $s(0) = \texttt{True}$, all spies can escape in 0 days which is the absolute minimum possible since spies cannot escape in a negative amount of days. Thus, the minimum days to escape in this case is 0.

Justication of Time Complexity

The algorithm makes use of $s(d)$ which essentially runs the algorithm from 4.1, taking $O(nm^2 D)$ every time it is computed.
The algorithm computes $s(d)$ twice to check $s(D)$ and $s(0)$ and twice for every layer of the binary search to find $s(d)$ and $s(d-1)$. Binary search is known to conclude in $O(\log N)$ layers where N is the size of the search space. In this case, the search space is of size $N = D$.
Therefore, the algorithm takes $O(nm^2 D)$ for the initial checks and $O(nm^2 D \log D)$ for the binary search, giving an overall complexity of $O(nm^2 D \log D)$

4.1

- Students seemed to struggle on this question. A sizeable proportion of students took approaches which did not work and some proportion of students took approaches which made little sense. The overall direction of the cohort seemed to be attempts to modify existing max-flow algorithms rather than constructing a graph to represent the problem. All tutorial problems however, simply construct a graph and make no modifications to max-flow algorithms. This is the norm and students who were unsatisfied with their performance are recommended to consider the tutorial problems in PS4.

We begin by examining a common incorrect approaches which made some sense.

- All-pairs-shortest-paths and then maximum bipartite matching.
  Many students attempted to do an All-pairs-shortest-paths by constructing a graph with each vertex representing a hideout and edges between them representing tunnels. They then either used a BFS/Dijkstra's for every vertex on which a spy starts in to determine which escape pods a spy can reach in D days.
  From there a bipartite matching can be formed, whereby spy vertices are connected to escape pod vertices if and only if the spy can reach that escape pod in D days. This however omits a critical constraint, that no 2 spies can use the same hideout on the same night. Consider a trivial example:
  spy A starts at v1, spy B starts at v2.
  v1- v3
  v2- v3
  v3- escape pod 1
  v3- escape pod 2
  Clearly they cannot escape in D = 2 days since they can't go into v3 together but the bipartite matching yields a max flow of 2 units.
  The relaxation of this constraint leads to a significantly easier problem and thus, responses like these generallly recieved no more than 5 marks.

- Constraining max flow to flow paths of length less than D or equivalent
  Some students construct a graph where vertices represent hideouts and starting spies are represented by a connection to a super-source and escape pods represented by a connection to the super-sink. Students then say to consider max flow but only allow for flow paths of less than length D. There were 2 critical issues with such a response.
  People simply did not elaborate on how you are supposed to modify FF or EK to only consider flows from paths of length less than D. This is entirely non-trivial and really constitutes the bulk of the algorithm for an attempt like this as otherwise, such responses are basically restating the question as "allow spies to roam tunnels for D days and see maximum number of people who can reach escape pods". I do not know of any polynomial time algorithm to find the max-flow of a given network constraining flow paths to have only D arcs when D is greater than 3 and when flows must be integers. If you can come up with one and prove it's correctness, you would most likely be making advances in algorithm theory.
  Another issue is how are people to account for the constraint of only one spy being allowed in a hideout per day. Vertex capacities do not work as any constraint would only limit the number of times a vertex is used and have no relevance to the original objective.
  Such responses were generally awarded less than 3 marks.

Now onto attempts that did not make much sense at all.

- Many people suggest we can iteratively run FF or EK somehow. Some suggest we can run one iteration of the EK algorithm on a graph where vertices represent hideouts and edges represent the tunnels between them with super-source for spies and super-sink for escape pods.
  Some suggest that we can somehow extract the position of the remaining spies after 1 iteration

and update the graph to remove the used escape pods and the new starting position of remaining spies. Aside from the fact that the extraction of such information is entirely non-trivial, there is no reason that the EK algorithm would somehow yield the optimal movement of spies assuming they had D days to maximise the number of spies which escape. As such, these responses made little to no sense and if interpreted very, very generously, would only constitute an algorithm equivalent to making spies go to their nearest escape pod. As such, these responses were generally given 0.

Lastly, correct responses

- Many people fail to justify the correctness of their algorithms. We expect you to justify how your graph construction reflects the constraints of the problem e.g.: - The vertex capacity of 1 limits the number of spies in any particular hideout on any particular day to be 1. - The capacity of 1 from the source vertex to all hideouts in the first layer which begin with a spy reflects how each unit of flow represents the path of a spy.

  We also expect you to explicitly state how to interpret the max flow with respect to the original question and justify your interpretation. E.g. as each unit of flow represents possible paths of ALL spies, the maximum flow from the source to the sink represents the number of possible concurrent paths and therefore the total number of spies that can escape.

  Failure to do so generally meant that the maximum mark attainable was 7.

4.2 People generally did well on this sub-question if they did well on the previous one. Most students either had approaches that had no relevance to the problem, were entirely correct, or lacked critical justification. The key points were as follows:

- You must justify the monotonicty of your defined function/underlying function of your search procedure. A simple way to do so would be to explain that if spies can escape in E days where E is less than D, they can also escape in D days but waiting in their hideouts after reaching them in E days. Then with respect to your function definition, trivially show that it is monotonic.
- If a function that accounted for the fact that we were searching for a boundary point was not clearly defined, then we expected the procedure provided to make explicit modifications and explanation for how we find a boundary point.