Please note that for max flow algorithms you must clearly detail any graphs you are constructing. This means describing all vertices, edges, capacities, sources, and sinks. Diagrams (either neatly hand-drawn or in LaTeX) are also encouraged as a supplement. In terms of quoting known max flow algorithms, you may only quote the ones taught in lectures (i.e. Ford-Fulkerson or Edmonds-Karp). To justify the correctness of a max flow algorithm, you must explain why your algorithm result satisfies all conditions imposed by the problem. Roughly 10% of the total marks will be specifically allocated towards the clarity and conciseness of your explanations.

This document gives model solutions to the assignment problems. Note that alternative solutions may exist.

Question 1 *Arc Competition*

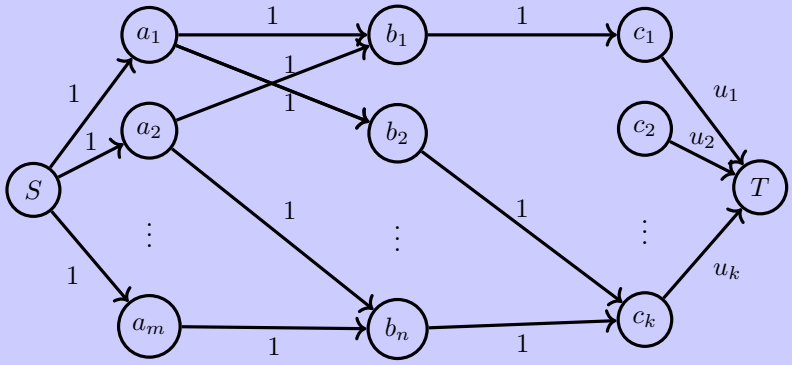
[20 marks] Arc is hosting a training session for the members of various societies at UNSW. There are n students who are Arc members, and $m < n$ registered societies. Each of the n students is a member of at least 1 and at most 4 of the m societies. Each of the m societies must send exactly 1 student to represent them at the training session.

In order to keep the crowd diverse, Arc would like to avoid all of the student representatives studying degrees from the same faculty. Each of the n students is enrolled in a single degree offered by 1 of the $k \leq n$ faculties at UNSW. For every faculty $i \in [1..k]$, at most u_i students belonging to it will be allowed to attend the event.

Design an $O(nm)$ algorithm that determines if it is possible to find a selection of students to attend the event such that all of these criteria are met. If it is possible, also identify which m students will attend.

Construct a flow network as follows:

- For each society $1 \leq x \leq m$, construct a vertex a_x.
- For each student $1 \leq y \leq n$, construct a vertex b_y.
- For each faculty $1 \leq i \leq k$, construct a vertex c_i.
- Construct a super source vertex S and a super sink vertex T.
- For each society x, construct an edge of capacity 1 from S to a_x. There are m edges here. This enforces that each society can only send 1 student.
- For each faculty i, construct an edge of capacity u_i from c_i to T. There are k edges here. This enforces the upper bound of students in each faculty.
- For each society x and student y, construct an edge of capacity 1 from a_x to b_y if student y is a member of society x. Since students can belong to at most 4 societies, there are at most $4n$ edges. This connects the societies to the students they could choose to represent them. The capacity here technically does not matter.
- For each student y and faculty i, construct an edge of capacity 1 from b_y to c_i if student y belongs to faculty i. There can only be 1 edge per student, for a maximum of n edges. This connects students to their faculties, and also enforces that a student can only represent 1 society since there is at most a capacity of 1 flowing from the student.



Now, the maximum flow of this network will correspond to the number of students who can attend the event. Hence, we can run Edmonds-Karp over this network and obtain the corresponding max flow F. If $F = m$, then the event can run. Otherwise if $F < m$, then Arc's conditions cannot be met and the event cannot run.

Assuming that the event can run, we can find the m students by looking at the final residual graph. The easiest way to do this is just to find all students who are reachable from the sink T in the final residual graph (e.g. with DFS/BFS), as this would require the edge between the student and their faculty to contain flow.

The time complexity of Edmonds-Karp is $O(E{\cdot}\min\{VE, |f|\}$. Here, the value of the maximum flow is at most m (capacity outgoing from the source node is m), and the total number of edges is at most $m+k+4n+n$. Since $m < n$ and $k \leq n$, the number of edges can be reduced to $O(n)$. Hence, the total time complexity is $O(nm)$.

Question 2 *Pirate Gathering*

[30 marks] Somewhere in the Caribbean there are V islands labeled $1..V$ and N pirate ships named $1..N$. Additionally there are E two-way routes, the i^{th} of which allows a ship to sail between island u_i and island v_i. However, the pirates are fiercely territorial and hence each direction of a route can only be used by one pirate ship. That is, if ship x has traveled from island u to island v, another ship y can still travel from island v to island u.

The pirate ships are initially on unique islands $\{h_1, h_2, \ldots, h_N\}$, and wish to gather on island $T \in [1, ..., V]$.

2.1 [18 marks] Design an $O(NE)$ algorithm which determines whether or not it is possible for every pirate to reach the designated island, T.

No marks will be deducted for an $O(V + NE)$ solution.

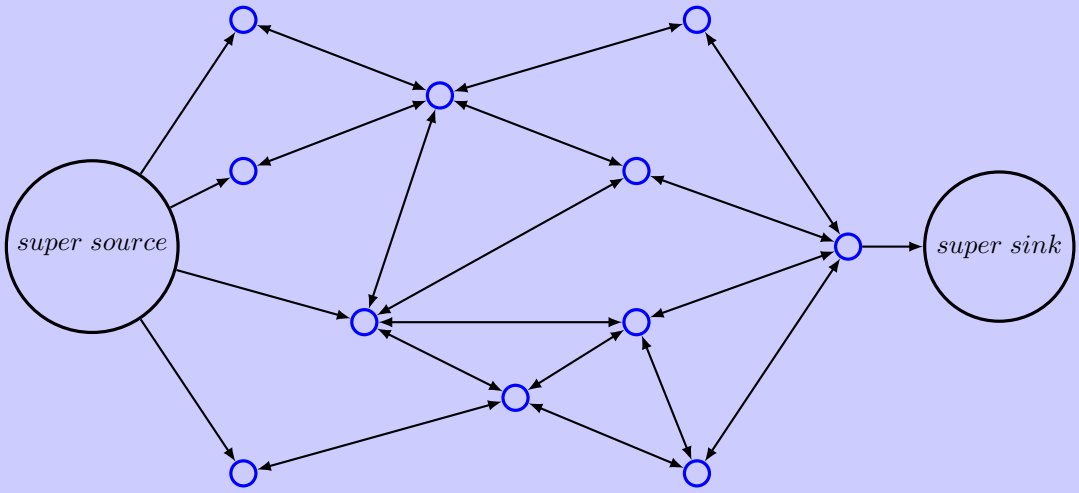
This subquestion is asking you to finding edge disjoint paths between designated start and end vertices.

We begin by constructing a flow graph G where vertices represent islands and flow edges represent routes. It is as follows:

- For each directed route from u_i to v_i, construct vertices u_i and v_i if they are not yet constructed. Then construct an edge of capacity 1 from u_i to v_i, and an edge of capacity 1 from v_i to u_i.
- A super source vertex.

- For each starting island h_i, construct an edge of capacity 1 from the super source vertex to vertex h_i.
- A super sink vertex.
- Construct an edge of capacity N from vertex T to the super sink vertex.

It takes $O(E)$ to construct our graph, because the number of constructed vertices is at most $2E$.



Now, a saturated path from the super source to the super sink is a saturated path from a starting island to island T. This represents the path that a pirate can take. Hence, we can run Ford-Fulkerson's over G, so that the total flow indicates the number of pirate ships which can reach reach the designated island, T. Thus, is it possible for every pirate to reach this island iff the total flow is equal to N. Since there are $O(E)$ edges and the flow is at most N, Ford-Fulkerson's runs in $O(NE)$ time.

In total, our time complexity is $O(E) + O(NE) = O(NE)$.

2.2 [6 marks] In an attempt to prevent the gathering, the government has now imposed a restriction that at most s_i pirate ships are allowed to depart from island i.

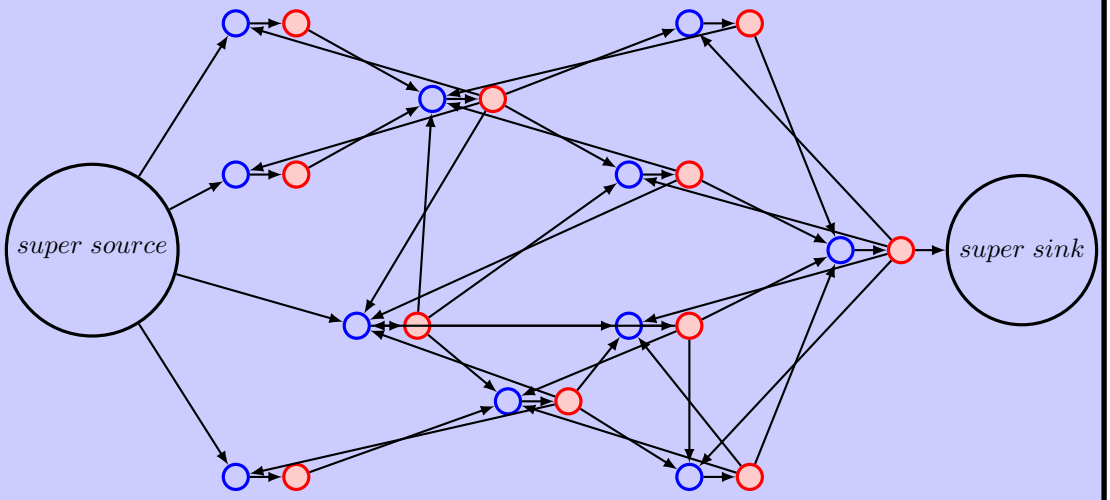
Design an $O(NE)$ algorithm which determines whether every pirate can still reach the designated island. You may reference your answer from Question 2.1 and only provide the details and justification of any modifications.

You may choose to skip Question 2.1 in which case your solution to Question 2.2 will be submitted for both parts.

No marks will be deducted for an $O(NV + NE)$ solution.

This subquestion is asking for the vth vertex to have a vertex capacity of s_v. For each original vertex v, this can be achieved by splitting it into v_{in} and v_{out}. Edges which were originally directed to this vertex will now be directed to v_{in}, while originally direction from this vertex will now be directed to v_{out}. Additionally, construct an edge of capacity s_v from v_{in} to v_{out}. Since the number of constructed vertices was initially at most $2E$, we have added at most $2E$ new vertices and $2E$ new edges. Our new graph has $O(E)$ vertices and $O(E)$ edges, so it

takes $O(E)$ to construct our graph.



As previoiusly mentioned, we can run Ford-Fulkerson's over G and check that the total flow is equal to N. We can show this is correct using a similar argument as before. There are $O(E)$ edges and the maximum flow is N, so Fold-Fulkerson's runs in $O(NE)$ time.

2.3 **[6 marks]** The location of the designated island T has been leaked to the government and it may no longer be safe. The pirates want to know if there any other options where all pirates can gather, given the same restrictions as Question 2.2. Design an $O(VEN)$ algorithm that counts the number of islands are suitable for the gathering (including island T).

You may reference your answer from Question 2.2 and only provide the details and justification of your modifications, which must also include an updated time complexity justification.

No marks will be deducted for an $O(NV^2 + VEN)$ solution.

From Question 2.2, we know how to test whether island T is suitable for gathering. Thus, we can simply try each vertex as the designated vertex T against our algorithm, and count the number of vertices that succeed.

The time complexity is N times the time complexity from Question 2.2. Hence, this is $O(VEN)$.

Question 3 *CSE Labs*

[30 marks] The School of CSE is scheduling some labs for its classes. There are n classes that need to be scheduled into one of the n available labs, and they would like all of the labs to run at the same time. A single class cannot occupy multiple labs, and multiple classes also cannot share a lab.

3.1 **[4 marks]** For a class to use a lab, there has to be at least 1 seat for each student. Given an array $C[1...n]$ where $C[i]$ indicates the number of students in class i, and an array $L[1...n]$ where $L[j]$ indicates the number of seats in lab j, design an $O(n \log n)$ time algorithm that assigns a lab room to each class. Your algorithm should also identify if an assignment is not possible.

We shall consider the following greedy method. Sort arrays C and L in non-increasing order via merge sort so that, for the rest of the solution, we may now assume that $C[i] \geq C[i+1]$ and $L[j] \geq L[j+1]$. Since we only can assign a lab to a class as long as the number of students in the class does not exceed the number of seats in the lab, we shall iterate over L and assign a lab $L[i]$ for each class $C[i]$.

We proceed as follows. Since C and L are sorted in non-increasing order, we assign lab $L[i]$ to $C[i]$ as long as $L[i] \geq C[i]$. If $L[i] < C[i]$ for any i, then no assignment is possible since the only possible way to assign a class to lab i is to assign a class j where $j < i$. If i is the first lab that cannot be assigned class i, then note that no such assignment is possible because every other class would have occupied a particular lab. Thus, an assignment is possible if and only if $L[i] \geq C[i]$ for every $1 \leq i \leq n$. This also gives the lab-class pairing in linear time.

To justify the time complexity, we observe that the two merge sorts take $O(2n \log n) = O(n \log n)$ time, while determining if such an assignment exists is linear in time complexity. Thus, the overall time complexity is $O(n \log n)$.

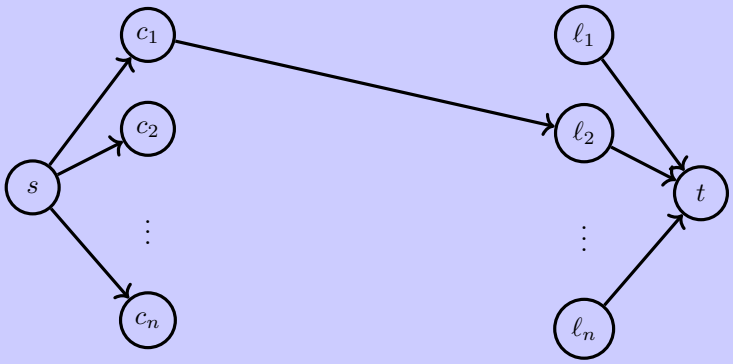
3.2 [7 marks] It turns out that classes are very picky about which labs they want. In particular, you are now provided an additional array $B[1..n][1..n]$ with

$$B[i][j] = \begin{cases} \texttt{True} & \text{class } i \text{ likes lab } j \\ \texttt{False} & \text{otherwise} \end{cases}$$

Given $C[1..n]$, $L[1..n]$ and $B[1..n][1..n]$, design an $O(n^3)$ algorithm to assign a lab room to each class. Your algorithm should also identify if an assignment is not possible.

This is a standard maximum bipartite matching problem. We construct the flow network as follows:

- For each class i, construct a vertex c_i.
- For each lab j, construct a vertex ℓ_j.
- For each class i and lab j, construct an edge of capacity 1 from c_i to ℓ_j if $B[i][j] = \texttt{True}$ and $C[i] \leq L[j]$.
- Construct a vertex s and a vertex t to represent the source and sink respectively.
- For each class i, construct an edge of capacity 1 from s to c_i.
- For each lab j, construct an edge of capacity 1 from ℓ_j to t.



This sets up the corresponding flow network. We can now run Edmonds-Karp on the flow network and the corresponding maximum flow L is the maximum number of class-lab pairs.

Note that, if $L < n$, then no assignment is possible because not all classes will have lab rooms available.

To see this, observe that the maximum possible flow is n since the maximum amount of flow coming into the flow network from the source is n; each edge from the source has capacity 1. Therefore, the amount of flow that can maximally arrive at the sink is n. In other words, we have that $L \leq n$. Now, if $L < n$, then there must exist two classes that point to the same lab. However, since the edge from c_i to ℓ_k has capacity 1, then it follows that only one such class can be assigned the lab by the flow conservation property of the flow network. In other words, not all classes will be assigned a lab. Thus, an assignment exists if and only if $L = n$.

To retrieve the assignment, observe that Edmonds-Karp produces the final residual graph. Within the final residual graph, we assign class c_i to lab ℓ_j if and only if the edge from ℓ_j to c_i exists in the residual graph.

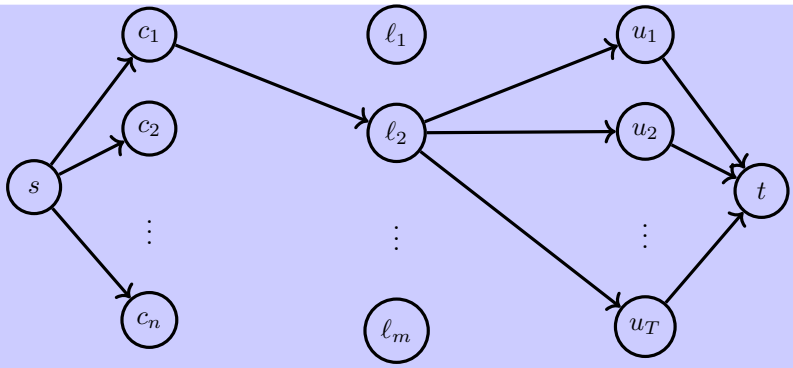
To show that such an algorithm runs in $O(n^3)$, first note that Edmonds-Karp runs in $O\left(\min\{E \cdot L, V \cdot E^2\}\right)$ time. Now, $L \leq n$. We also see that the number of edges (maximally) is $n + n^2 + n$ since every vertex c_i can be connected to every vertex ℓ_j. Thus, $E \cdot L = n\left(2n + n^2\right)$. On the other hand, $V \cdot E^2 = (2n+2) \cdot \left(2n + n^2\right)^2$. Clearly, the time complexity is $O(E \cdot L) = O(n^3)$.

If we use *Ford-Fulkerson* instead of Edmonds-Karp, we will obtain the same time complexity because the breadth-first search implementation of the augmenting path does not yield a better time complexity since the flow is bounded by n. Thus, using Ford-Fulkerson is just as good.

3.3 **[12 marks]** Unfortunately an issue with the building has affected the power and now there are only $m < n$ labs available. The school has no choice but to schedule some classes at different times. Given the arrays $C[1...n]$, $L[1...m]$, $B[1..n][1..m]$ defined as in the previous parts, design an $O(n^3)$ time algorithm that determines whether it is possible to schedule the classes with $T < N$ different time slots available.

We slightly modify the bipartite graph from 3.2 by adding an extra column to form a tripartite graph. Construct the flow network as follows:

- Construct the structure of the graph as in 3.2.
  - Each class i is a vertex c_i.
  - Each lab j is a vertex ℓ_j.
  - Construct an edge of capacity 1 from c_i to ℓ_j if $B[i][j] =$ `True` and $C[i] \leq L[j]$.
- For each time slot k, construct a vertex u_k.
- For each lab j and time slot k pair, construct an edge of capacity 1 from ℓ_j to u_k.
- Construct a source s and sink t.
- For each class i, construct an edge of capacity 1 from s to c_i.
- For each time slot k, construct an edge of capacity m from u_k to t.



We can now run Edmonds-Karp (or Ford-Fulkerson) on the flow network and the corresponding maximum flow L' is the maximum number of class-lab-time triplets. We now claim that an assignment exists if and only if $L' = n$. By a similar argument to 3.2, it is not hard to see that $L' \leq n$. It, therefore, suffices to show that, if $L' < n$, then no assignments are possible.

Suppose that $L' < n$. Since the maximum flow is less than n, then, from reading the final residual graph, there must exist a class i that is not assigned a lab. This implies that class i would either be not connected to any other lab j or connected to a lab that is already occupied by another class in the same timeslot. But, since each edge is of capacity 1, *exactly* one such class would be assigned the particular lab. Hence, class i would not be assigned any other lab and thus, it is impossible for class i to be assigned a lab which implies that no assignments are possible since an assignment requires every class to be assigned a lab. In other words, an assignment exists if and only if the maximum flow is exactly n.

To justify the time complexity, we observe that the number of edges in such a graph is (maximally) $n + n^2 + nT + n$ since every class can be connected to every lab, and every lab is connected to every time slot. Similar to 3.2, the maximum flow is bounded above by n. Thus, the time complexity is bounded by the time complexity of Ford-Fulkerson, giving us $O(E \cdot L') = O\big(\big(2n + n^2 + nT\big)\, n\big)$. Now, since $T < n$, we have that

$$2n + n^2 + nT < 2n + n^2 + n^2 = 2n + 2n^2 < 4n^2.$$

Thus, the time complexity is $O(4n^2 \cdot n) = O(n^3)$.

Alternative solution: Construct the flow network as in 3.2; however, each edge from ℓ_j to t has edge capacity T. Note that each class is still assigned a single lab. Since each time slot is unique, this implies that each lab can host (maximally) T many classes. Thus, the proof of correctness is exactly the same as the above solution, where assignments exist if and only if $L' = n$ after running either Edmonds-Karp or Ford-Fulkerson. The time complexity remains to be $O(n^3)$ since the only change that is made is the capacity of the outgoing edges of the labs (however, the flow is bounded above by n regardless of the change).

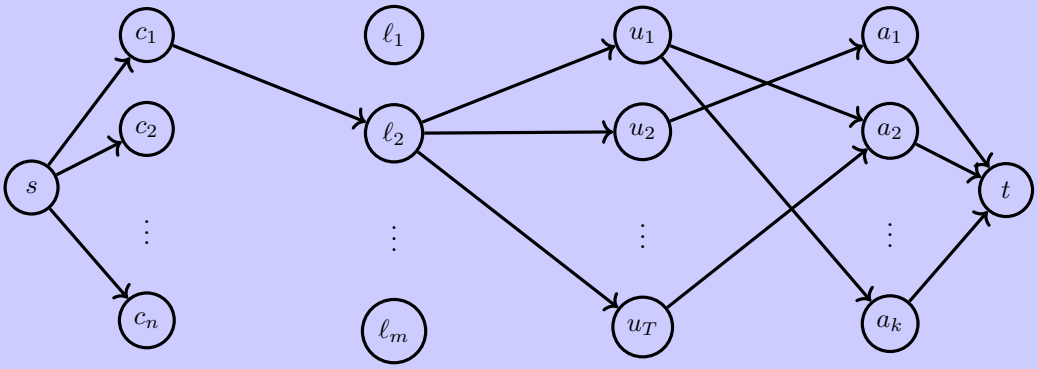
3.4 **[7 marks]** Each of the classes also requires one of $k \leq n$ tutors to run the lab. Each tutor is only allowed to run a maximum of 3 labs, and can only run 1 lab at a time. The tutors' availability is given as an array $D[1..k][1..T]$ with

$$D[i, j] = \begin{cases} 1 & \text{if the } i\text{th tutor is available at timeslot } j \\ 0 & \text{otherwise} \end{cases}$$

For some number of timeslots $T < n$, design an $O(n^3)$ algorithm that assigns a room, time, and tutor to each lab if a schedule is possible.

We make a further extension by introducing another column of vertices to the tripartite graph from 3.3. This gives us a 4-partite graph with the last column being the k tutors. Thus, we proceed as we do in 3.3.

- Construct the structure of the graph as in 3.3. We now demonstrate how we construct the last column of vertices.
- For each tutor q, construct a vertex a_q.
- For each time slot p and tutor q, construct an edge of capacity 1 from u_p to a_q if $D[q][p] = 1$; in other words, construct an edge from time slot p to tutor q if q is available to tutor in slot p.
- For each tutor q, construct an edge of capacity 3 from a_q to the sink t.



We can now run Edmonds-Karp (or Ford-Fulkerson) on the above flow network and the corresponding maximum flow L'' is the maximum number of class-lab-time-tutor quadruplets. We also claim that an assignment exists if and only if $L'' = n$, for which the proof follows almost immediately from the arguments expressed in earlier parts.

To retrieve the assignment, observe that Edmonds-Karp produces the final residual graph. Within the final residaul graph, we assign a class i to a lab j at time t if and only if there is an edge from ℓ_j to c_i and an edge from u_t to ℓ_j. The corresponding tutor x takes the class if and only if there is an edge from a_x to c_i passing through vertices u_t and ℓ_j.

To show that the capacity of 3 in the outgoing edges of u_i is correct, we observe that each tutor is allowed to teach a *maximum* of 3 labs. This corresponds to the capacity of each outgoing edge from a_i to t since each class-lab-timeslot accounts for each unique lab that tutor i can teach. Thus, having a capacity of 3 implies that each tutor can teach up to 3 unique classes. Since each timeslot has an outgoing capacity of 1 towards each tutor, this implies that each tutor can only teach one class in that particular timeslot. This satisfies all of the constraints, which shows that such a construction is correct.

To justify the time complexity, observe that the maximum flow is bounded above by n and the number of edges is $E = n + n^2 + nT + kT + k$. But observe that we have that

$$E \leq n + n^2 + n^2 + n^2 + n = 2n + 3n^2 \leq 5n^2.$$

Thus, the time complexity is $O(5n^2 \cdot n) = O(n^3)$.

Question 4 *Spy Escape*

[20 marks] 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 [14 marks] 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.

Try to first solve this for $D = 1$, and then consider how would you could extend this for $D = 2$ and greater.

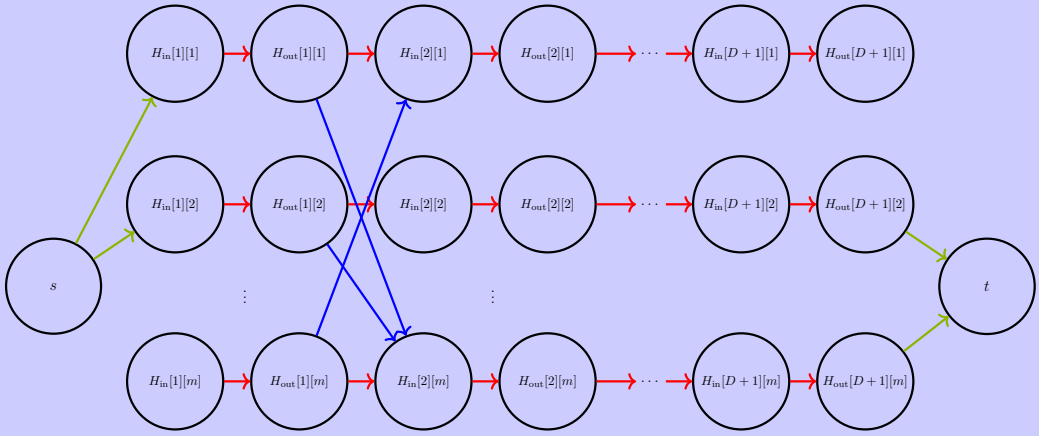
We start by constructing a flow graph for $D = 1$:

- We will have 2 layers; the first layer represents day 0 and the second layer represents day 1
- For both layers l, we have m hideouts and for each hideout $1 \leq i \leq m$, we construct two vertices $h_{in}[l][i]$ and $h_{out}[l][i]$, which are connected by an edge of capacity 1
- Construct an edge of capacity 1 from $h_{out}[1][i]$ to $h_{in}[2][j]$ if $T[i][j] =$ True, that is, if there is a tunnel between hideout i and hideout j
- For each hideout $1 \leq i \leq m$, construct an edge of capacity 1 from $h_{out}[1][i]$ to $h_{in}[2][i]$ for the case when a spy stays in the same hideout on day 1
- A source s which has an edge of capacity 1 to each of $h_{in}[1][i]$, where hideout i has spy on day 0 (n edges in total)
- A sink t which has an edge of capacity 1 to each of $h_{out}[2][i]$, where hideout i contains an emergency escape pod (n edges in total)

We can then generalise it and construct a flow graph as follows:

- Layers 1 to $D + 1$ represent days
- For each layer $1 \leq l \leq D + 1$, we have m hideouts and for each hideout $1 \leq i \leq m$, construct two vertices $h_{in}[l][i]$ and $h_{out}[l][i]$, which are connected by an edge of capacity 1. The edge of capacity 1 ensures that each hideout can only accommodate one spy at a time
- For each layer $1 \leq l \leq D$, construct an edge of capacity 1 from $h_{out}[l][i]$ to $h_{in}[l + 1][j]$ if $T[i][j] =$ True, that is, if there is a tunnel between hideout i and hideout j
- For each layer $1 \leq l \leq D$ and for each hideout $1 \leq i \leq m$, construct an edge of capacity 1 from $h_{out}[l][i]$ to $h_{in}[l+1][i]$ for the case when a spy stays in the same hideout on day l
- A source s which has an edge of capacity 1 to each of $h_{in}[1][i]$, where hideout i has spy on day 0(n edges in total), each unit flow represents the path of a spy

- A sink t which has an edge of capacity 1 to each of $h_{out}[D][i]$, where hideout i contains an emergency escape pod (n edges in total), capacity 1 reflects that each escape pod can only accommodate one spy



The paths from source s to sink t represent the paths of the spies over D days. If there is a flow through $h_{out}[l][i]$ and $h_{in}[l+1][j]$ where $1 \leq i \leq m$ and $1 \leq j \leq m$, then it means a spy crawl from hideout i to hideout j on day l. The unit flow on each edge ensures that only 1 spy can go on that path at a time.

The unit flow from source s to each of $h_{in}[1][i]$ ensures that each of the n spies begins with an initial hideout, and the maximum flow $F \leq n$ of this network corresponds to the number of spies that end up at a hideout with an emergency escape pod after D days.

To determine whether it is possible for all spies to escape, we run Ford-Fulkerson's Algorithm over this network to find the maximum flow F. If $F = n$ then it is possible, otherwise if $F < n$, then it is not possible and only F spies can escape.

The total number of edges E is at most $2n + Dm + Dm^2 < 2m + Dm + Dm^2$ and F is bounded by n. Since the time complexity of Ford-Fulkerson's Algorithm is $O(|F|E)$, the time complexity to run on this network is $O(nm^2D)$

4.2 [6 marks] 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 $O(nm^2D \log D)$ algorithm to determine the minimum number of days needed for all spies to successfully escape.

We define our algorithm in 4.1 $F(x)$, where

$$F(x) = \begin{cases} \texttt{True} & \text{if all spies can escape in } x \text{ days} \\ \texttt{False} & \text{otherwise} \end{cases}$$

Notice that if all spies can escape in x days, then it is also possible for them to escape in y days for all $y \geq x$. This is because all of them can follow the same path as the path calculated by $F(x)$, then remain in the same hideout from day x to day y. This means that $F(x)$ is a monotone function, that is, we can use binary search over the range $[1..D]$ to find the smallest x such that $F(x)$ is true.

The binary search works as follows:
It will return `impossible` if it we cannot escape in D days and we keep track of the minimum

number of days in variable `ans`, where `ans = impossible` initially. Suppose that m is the midpoint of sub-array $[l..r]$, we repeat the following until $l > r$:

- If $F(m)$ is true, we update $ans = m$, and search over $[l..m-1]$
- If $F(m)$ is false, search over $[m+1..r]$

The binary search check over the range $[1..D]$, which means it will run at most $O(logD)$ times. Since each of the binary search run function $F(x)$, which has a time complexity of $O(nm^2D)$, the overall time complexity to find the minimum number of days is $O(nm^2D \log D)$.