1. True or False? Explain your answer in one line.

   • (4 points) (clarification: $i$ does not need to be an integer)
     \[ \forall S \subseteq \{1, 2, \ldots, 10\}, \sum_{i \in S} i \leq \sum_{i:2i \in S} i. \]

     Solution: False. Consider a counterexample. Let $S = \{1, 2, 3, 4\}$. Then,
     \[ \sum_{i \in S} i = 10 \]
     and
     \[ \sum_{i:2i \in S} i = \sum_{i \in \{5, 1, 1.5, 2\}} i = 5 \]

   • (4 points) $\log_2(2^5n) = 5 \log_2 n$.

     Solution: False.
     $\log_2(2^5n) = \log_2(2^5) + \log_2(n) = 5 \log_2(2) + \log_2(n) = 5 + \log_2(n)$
     Assuming that $n$ is a real number, the equality in question ($5 + \log_2(n) = 5 \log_2 n$) holds only for $n = 2^{5/4}$, and it is not true in general for all values of $n$.

   • (4 points) Let $G = (V, E)$ be an undirected graph. If $\forall u \in V \exists uv \in E$, then $G$ is connected.

     Solution: False. Below is a counterexample where each node has an edge but the graph is disconnected.

      \[ u \]
      \[ v \]
      \[ x \]
      \[ y \]
• (4 points) Multiplying the capacities of all edges in a flow network by 2, multiplies the value of the maximum flow by 2.

**Solution:** True. We know that value of max-flow is same as capacity of min-cut.

\[ v(f) = c(A, B) = \sum_{e \text{ out of } A} c_e \]

And we know that for a min-cut \((A, B)\), \(c(A, B) \leq c(X, Y)\), for any cut \((X, Y)\). Therefore, it is easy to see that after multiplying all capacities by 2, \((A, B)\) would be a min-cut in updated network as well, and its capacity and hence max-flow would be doubled.

• (4 points) Consider a flow network with exactly one minimum cut \((A, B)\). Suppose that there are \(k\) edges going from \(A\) to \(B\). If we increase the capacities of every edge in the network by 1, then the value of the minimum cut will increase by \(k\).

**Solution:** False. Below is a counterexample where the minimum cut before increasing the capacities has a value of 3, and passes through 3 edges, and after increasing the capacities, the minimum cut only increases to 5.

![Counterexample](image)

2. (15 points) Construct a flow network with \(n\) nodes that has \(2^{n-2}\) minimum cuts. (This shows that no efficient algorithm can output all the minimum cuts, as simply there might be too many of them).

**Solution:** Consider the following graph, that can be generalized for any value of \(n\). The capacity of an edge from \(s\) to any one of the internal node \(x\) must have the same capacity of the edge from \(x\) to \(t\).

![Diagram](image)

3. (15 Points) Consider the following two algorithms for finding the maximum flow:
Algorithm 1: Scaling max-flow
- Initially set \( f(e) := 0 \) for all edges \( e \).
- Set \( \Delta \) to be \( \max c_e \) rounded down to a power of 2.
- While \( \Delta \geq 1 \):
  - While there is an \( s,t \)-path \( P \) in \( G_f(\Delta) \):
    - Augment the flow using \( P \) and update \( G_f(\Delta) \).
  - Endwhile.
- Set \( \Delta := \Delta/2 \).
- Endwhile.
- Output \( f \).

Algorithm 2: The fattest path algorithm
- Initially set \( f(e) := 0 \) for all edges \( e \).
- While there exists an \( s,t \)-path in \( G_f \):
  - Augment the flow using the fattest \( s,t \)-path \( P \) in \( G_f \).
  - Update \( G_f \).
- Endwhile.
- Output \( f \).

In the second algorithm the fattest means the largest bottleneck. From the class we know that the number of augmentations in Algorithm 1 is at most \( 2m \lceil \log_2 K \rceil \), where \( K \) is the maximum capacity of an edge. Deduce from this that the number of augmentations in Algorithm 2 is also at most \( 2m \lceil \log_2 K \rceil \).

Solution: At every iteration of the inner loop of Algorithm 1, the fattest path must be one of the possible paths that can be augmented, otherwise there would exist a path \( P \) with a bottleneck of at least \( \Delta \), which would be larger than the fattest path’s bottleneck (contradiction). Hence, Algorithm 1 could always augment the fattest path, in which case it would behave exactly as Algorithm 2. Therefore, because the upper bound on the number of augmentations is valid for any execution of Algorithm 1, it must be valid also for Algorithm 2.

4. (a) (15 points) Show that for every flow network, there exists an execution of the Ford-Fulkerson algorithm that finds the maximum flow after at most \( m \) augmentations. Here \( m \) is the number of the edges of \( G \).

Solution: To prove this, we apply the following algorithm to decompose a maximum flow \( f \) into a set of flow-paths:
- Repeat the following two steps until there is no such \( s,t \)-path:
  - Find an \( s,t \)-path \( P \) with positive flow, i.e., \( f(e) > 0 \) for all \( e \in P \).
  - Let \( \Delta \) be the minimum value of the flow \( f \) on edges of \( P \). Decrease the flow \( f \) on each edge \( e \in P \) by \( \Delta \).

In every iteration the value of \( f \) on at least one edge becomes 0. Hence the number of iterations is at most \( m \). Using these paths as augmenting paths (with augmenting value \( \Delta \)) leads to the desired result.

(b) (10 points) On the other hand, construct examples of flow networks that require \( \Omega(m) \) augmentations no matter how we choose the augmenting paths. (If you are not familiar with the notation, see the definition of big Omega in Chapter 2.2 of the textbook).
5. Suppose that we have solved the Max Flow problem on a flow network \((G = (V, E), s, t, \{c_e\}_{e \in E})\), and found the flow \(f\) with the largest value.

(a) (10 Points) Someone increases the capacity of one of the edges by 1. Can we update the value of the Max-Flow in \(O(m)\)? (Note that to achieve this running time, we cannot afford to run FF from scratch).

**Solution:** Let \(f^*\) be a max-flow in network \(G\). Let \(G'\) be a new network same as \(G\) but with increased capacity for one edge: \(c'(u, v) = c(u, v) + 1\). It is clear that every flow in \(G\) is also a flow in \(G'\) since capacity and conservation conditions are satisfied. Also, for a max-flow \(f'\) in \(G'\), \(v(f') \leq v(f^*)\).

Let \(G'_f\) be the residual network of \(G'\) for a flow \(f\). Then we know that \(f\) is a max-flow if and only if there is no augmenting path in \(G'_f\). Therefore we can update max-flow by finding an augmenting path as follows:

i. Obtain residual network \(G'_f\) from \(G\) with respect to original max-flow \(f^*\).

ii. Find an augmenting path \(P\) in \(G'_f\).

iii. If an augmenting path \(P\) was found, max-flow \(f' = \text{augment}(f^*, P)\).

Else if there is no augmenting path, \(f' = f^*\) is max-flow for \(G'\).

We can see that the step (iii) above (finding an augmenting path) needs to be performed only once because capacity of the augmenting path \(P\) has to be 1. If it is greater than 1, the original max-flow \(f^*\) could not have been a max-flow for \(G\) (because it could have been further augmented to obtain a greater flow). Thereafter augmenting with the path \(P\), \(G'_f\) does not have any more augmenting paths with capacity greater than 0, and \(f'\) must be a max-flow after step (iii).

Running time for constructing \(G'_f\) is \(O(m + n)\) and for finding one augmenting path it is \(O(m + n)\) using breadth-first search. Therefore total time is \(O(m + n) = O(m)\), considering \(m \geq n/2\).

(b) (15 Points) Someone decreases the capacity of an edge that is adjacent to \(s\) by 1. Can we update the value of the Max-Flow in \(O(m)\)? (Note that to achieve this running time, we cannot afford to run FF from scratch).

**Solution:** Let \(f^*\) be a max-flow in network \(G\). Let \(G'\) be a new network same as \(G\) but with decreased capacity for one edge from source: \(c'(s, u) = c(s, u) - 1\). Note that \(f^*\) might not be a valid flow for \(G'\) due to the capacity condition on the edge \(su\). The idea is to first modify \(f^*\) to make it a valid flow by possibly decreasing its value by 1, and then running one round of FF algorithm to turn it into a max-flow from \(G'\).

Then for a max-flow \(f'\) in \(G'\), \(v(f') \leq v(f^*)\).

There are two cases to consider according to value of flow \(f^*(s, u)\) for the edge \((s, u)\) in \(G\):

i. If \(f^*(s, u) < c(s, u)\) in \(G\), then \(f^*\) is also a max-flow in \(G'\) because a decrease of 1 in capacity in this case does not violate the conditions for \(f^*\) to be a flow.

ii. If \(f^*(s, u) = c(s, u)\) in \(G\), then:
   A. Obtain residual network \(G'_f\) from \(G'\) with respect to original max-flow \(f^*\).
   B. Since flow along edge \((s, u)\) is 1 less than its capacity \(c'(s, u)\) in \(G'\), we need to decrease flow along the path \(u \rightarrow t\). To do this, we find a simple path from \(t \rightarrow u\).
in residual network $G'_f$ using breadth-first search. Such a find is guaranteed to exist in $G'_f$ because the path $s \to u \to t$ has positive flow in $G'$. We add 1 unit of flow along the path $t \to u \to s$ in $G'_f$; i.e. adding 1 unit of flow on backward edges in residual network would effectively reduce flow along the path $s \to u \to t$ in $G'$. This results in flow $f$ such that $v(f) = v(f^*) - 1$

C. Now, we can find if there is an augmenting path $P$ in residual network $G'_f$.

D. If an augmenting path $P$ was found, max-flow $f' = \text{augment}(f, P)$.

Else if there is no augmenting path, $f' = f$, is max-flow for $G'$.

Similar arguments can be made as in question 5(a) to show that finding only one augmenting path (step C) is enough to get a max-flow. Running time for constructing $G'_f$ is $O(m + n)$, and $O(m + n)$ for two breadth-first searches in steps B & C. Therefore total time is $O(m + n) = O(m)$, again considering $m \geq n/2$