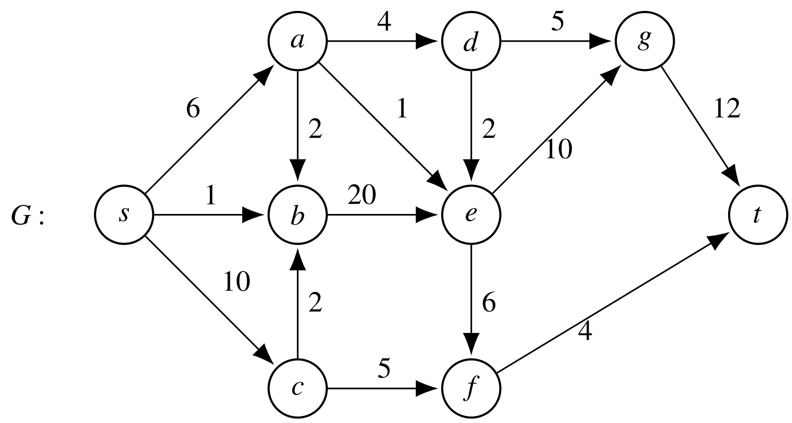
CMPSC 465 Data Structures & Algorithms
Fall 2025 Ke Chen and Yana Safonova

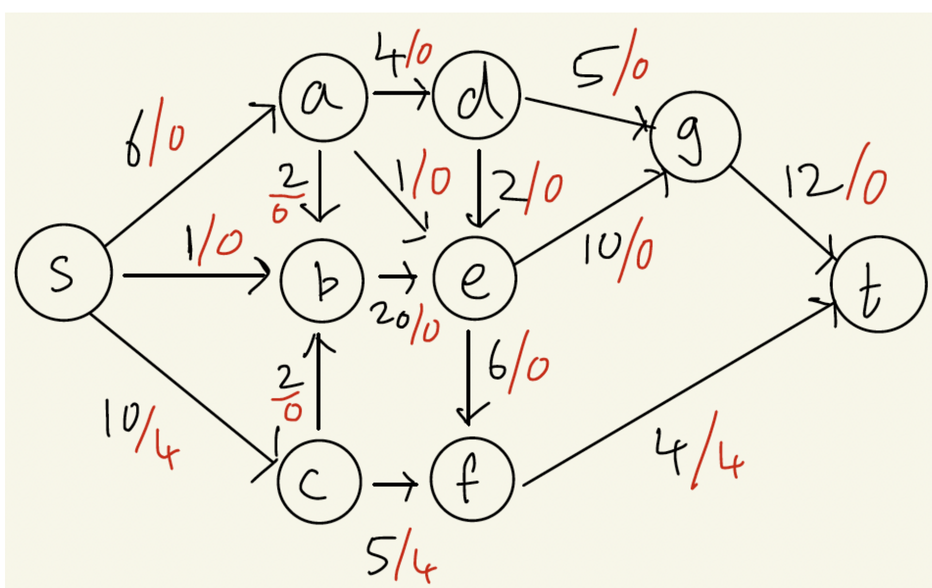
HW 7 Solution

1. (pts.) **Ford-Fulkerson Algorithm.** Use the Edmonds-Karp algorithm (namely, Ford-Fulkerson where each augmenting path is found by BFS) to find the maximum flow and the corresponding minimum cut in the given $s-t$ flow network. Process neighbors in alphabetical order during BFS. Show the augmenting path and draw the residual graph G_f for each step. Show the final maximum-flow and the minimum cut.

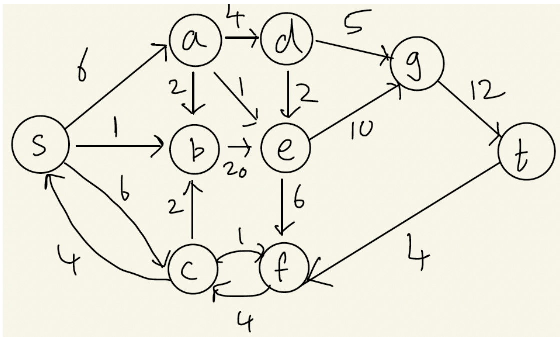


Answer:
The maximum flow is 13 units and the minimum (S,T)-cut is $(\{S,C,F\}, \{A,B,D,E,G,T\})$.

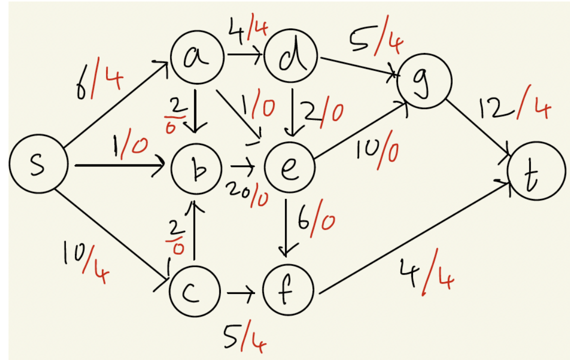
Using the Edmonds-Karp algorithm, the first $s-t$ path chosen for augmenting the flow is $s \to c \to f \to t$. The bottleneck edge is $f-t$ with a capacity of 4, so we augment the flow along this path by adding 4 units of flow to each edge. In the following figure, the flow values are written in red.



The new residual graph is shown below. Here, the $s-t$ path chosen is $s \to a \to d \to g \to t$. The bottleneck edge is $a-d$ with a capacity of 4, so we augment the flow along this path by adding 4 units.

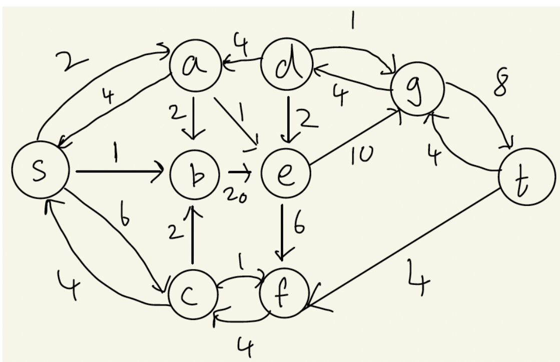


(a) Residual Graph

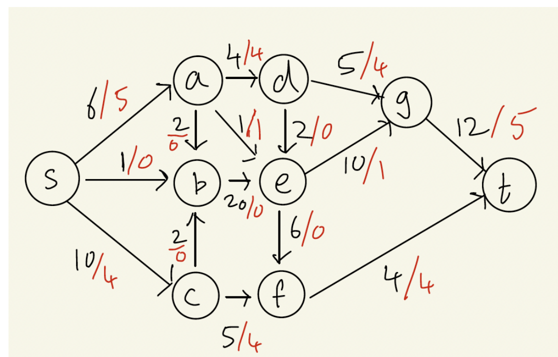


(b) Augmented Flow

After augmenting the flow, we get the following residual graph. In this iteration, the $s-t$ path chosen is $s \to a \to e \to g \to t$. The bottleneck edge is $a-e$ with a capacity of 1. So, we augment the flow along this path by adding 1 unit of flow.

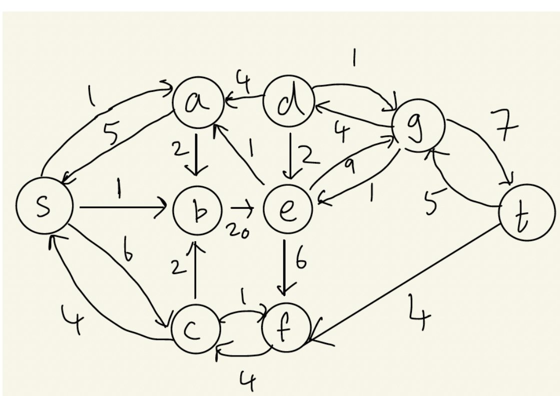


(a) Residual Graph

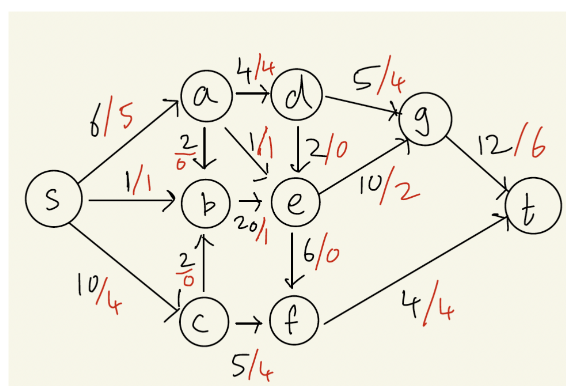


(b) Augmented Flow

The new residual graph is shown below. The $s-t$ path chosen is $s \to b \to e \to g \to t$. The bottleneck edge is $s-b$ with a capacity of 1. We augment the flow along this path by adding 1 unit of flow.

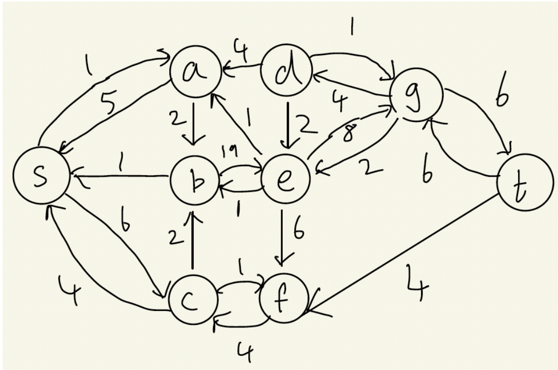


(a) Residual Graph

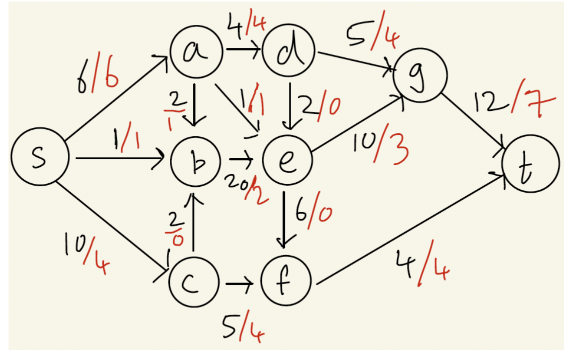


(b) Augmented Flow

The new residual graph is shown below. The $s-t$ path chosen is $s \to a \to b \to g \to e \to t$. The bottleneck edge is $s-a$ with a capacity of 1. We augment the flow along this path by adding 1 unit of flow.

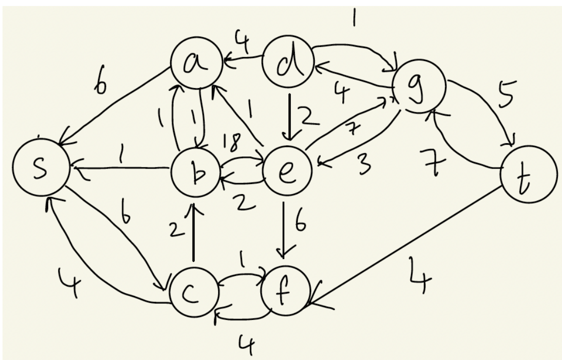


(a) Residual Graph

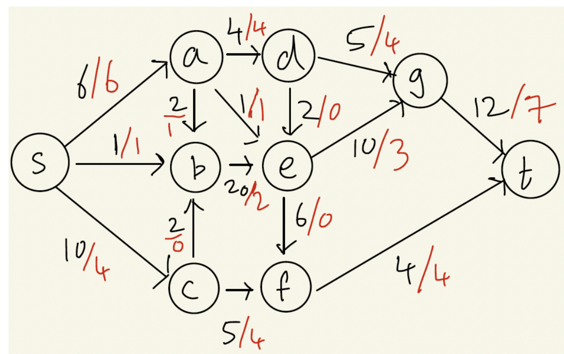


(b) Augmented Flow

The new residual graph is shown below. The $s-t$ path chosen is $s \to c \to b \to e \to g \to t$. The bottleneck edge is $c-b$ with a capacity of 2. We augment the flow along this path by adding 2 units of flow.



(a) Residual Graph

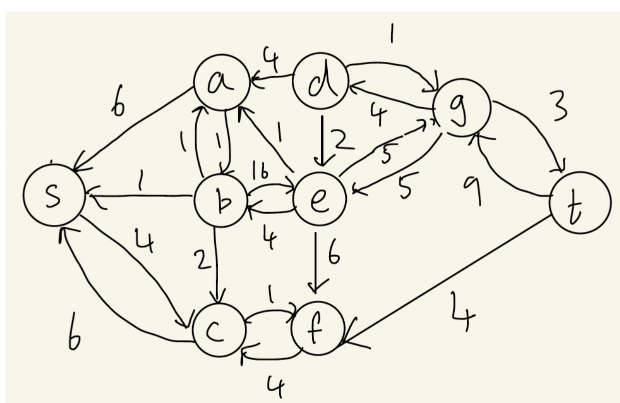


(b) Augmented Flow

The new residual graph is shown below. There is no $s-t$ path so the algorithm terminates. The maximum flow is 13. By definition, the minimum cut is (S^*, T^*) where:

$$S^* = \{v \mid v \text{ is reachable from s in } G_f^*\}$$

and $T^* = V \setminus S^*$.



2. (pts.) **Max Flow Formulation.** Consider a variation of the standard max-flow problem in a flow network. In addition to the existing edge capacities, each **vertex** has a capacity indicating the maximum amount of flow that is allowed to pass through. Show that this problem can be reduced to the standard max-flow problem and explain any pre-processing or additional data structures used. [Hint: transform this graph with both edge and vertex capacities into a regular flow network with only edge capacities by doubling the number of vertices.]

Answer:

Construct a new network G' by splitting each vertex v of G into two vertices: v_{in} and v_{out}. Make all edges going into v in G go into v_{in} and all edges leaving v in G leave v_{out} in G'. Finally, insert an edge from v_{in} to v_{out} with capacity equal to the capacity of v in G. Now it is sufficient to solve the maximum flow problem with edge capacities in G'. This works as every valid flow through v in G that satisfies v's capacity constraint can be made into a valid flow through v_{in} and v_{out} in G', and vice versa.

3. (pts.) **Critical Edge.** An edge of an $s-t$ flow network is called *critical* if decreasing the capacity of this edge results in a decrease in the $s-t$ maximum flow. Give an efficient algorithm that finds a critical edge in a network. Explain the correctness of your algorithm and analyze the running time.

Answer:

Let $f > 0$ be the value of the maximum $s-t$ flow (if $f = 0$, then there are no critical edges). By the max-flow min-cut theorem the capacity of any minimum capacity $s-t$ cut is also f. Fix a minimum $s-t$ cut $\mathscr{C} = \{S, T\}$, and let e be an edge of positive capacity c_e crossing this cut from S to T. Decreasing the capacity of this edge by any $\varepsilon > 0$ decreases the capacity of $\mathscr{C}$ by ε. Since $\mathscr{C}$ was a min-cut in the original graph, the min-cut in the new graph therefore has a strictly smaller min-cut and hence a strictly smaller max-flow (again, by weak duality). Thus, any positive capacity edge which crosses an $s-t$ min-cut is a *critical edge*.

To find such an edge, we first compute the $s-t$ max flow F of value f in $G = (V, E)$, and then construct the residual graph G'. If $f = 0$, then the network is disconnected (no path from s to t) hence there are no-critical edges, so we report this and exit. Otherwise, let S be the set of vertices reachable from s in G' (we know that there is no path from s to t in G', so $S \neq V$). Since there are no edges from S to $V - S$ in G', it follows that f is equal to the capacity of the cut $(S, V - S)$ in G. Thus, by the max-flow min-cut theorem, $(S, V - S)$ is a minimum cut, and from our preceding discussion, we can simply return a positive capacity edge in G that goes from S to $V - S$ (there exists such an edge since the capacity of the cut is $f > 0$).

Running time. We first do a max-flow computation. After this, constructing G' takes $O(|E| + |V|)$ time, and so does finding S (using BFS). Looking for an edge crossing $(S, V - S)$ can take a further $O(|V| + |E|)$ time, so the total running time is 1 max-flow computation $+ O(|V| + |E|)$ (for example, $O(|V||E|^2)$ using Edmonds-Karp).

The proof of correctness is given below:

Proof. Let $G = (V, E)$ be a flow network and let f^* be a maximum flow found using the Ford-Fulkerson algorithm. Let S_1 and T_2 be as defined above. We prove that the set of edges appear in every minimum (S, T)-cut is exactly

$$E_{\text{cut}} = \{(u, v) \in E \mid u \in S_1, v \in T_2\}.$$

First note that $(S_1, V - S_1)$ is the minimum cut associated with the maximum flow. Furthermore, $(V - T_2, T_2)$ is also a minimum cut because by the definition of T_2, all the edges from $V - T_2$ to T_2 must be fully saturated by f^*, and therefore $v(f^*) = c(V - T_2, T_2)$. The intersection of edges in $(S_1, V - S_1)$ and edges in $(V - T_2, T_2)$ is precisely E_{cut}. So E_{cut} must include all edges satisfying the requirement: Any edge not in E_{cut} is either not in the min-cut $(S_1, V - S_1)$ or not in the min-cut $(V - T_2, T_2)$.

Now we only need to show that each edge $(u, v) \in E_{\text{cut}}$ is indeed in all min-cuts. Consider an arbitrary min-cut (S, T). Since we know $s \in S$, and there is a path from s to u in G_{f^*} (by definition of S_1), if u is in T, then some edge along that path must be in the cut (S, T). But this edge is not fully saturated by f^* (otherwise we wouldn't be able to find this path in G_{f^*}), contradicting the assumption that (S, T) is a min-cut. Therefore, u must be in S. Similarly, we can show that v must be in T. Hence, the edge (u, v) is in the min-cut (S, T). □

4. (pts.) **Application of Network Flow.** Some programs in our university have a complicated set of graduation rules. Suppose a set $C = \{c_1, c_2, \ldots, c_n\}$ of n courses is offered. Each rule is specified by a subset $S \subseteq C$ of courses and an integer x, indicating that a student must take at least x courses from S in order to satisfy that rule. The subsets for different rules may overlap, but each course can be used to satisfy at most one rule. To graduate, a student must satisfy **all** of the rules.

Given a collection of m rules $\{(S_1,x_1),(S_2,x_2),\ldots,(S_m,x_m)\}$ and a set of ℓ courses a student has taken $\{c_{i_1}, c_{i_2}, \ldots, c_{i_\ell}\}$, describe an efficient algorithm to determine whether the student can graduate. Justify the correctness of your algorithm and analyze its running time.

For example, suppose there are 5 courses, $C = \{c_1, c_2, c_3, c_4, c_5\}$, and two rules:

- One must take at least $x_1 = 2$ courses from $S_1 = \{c_1, c_2, c_3\}$.
- One must take at least $x_2 = 2$ courses from $S_2 = \{c_3, c_4, c_5\}$.

Then a student who took $\{c_1, c_3, c_5\}$ cannot graduate, while a student who took $\{c_1, c_2, c_3, c_4\}$ can.

Answer: We build a flow network G as follows to represent the problem:

- One node for each course c_i [note: also OK to only include courses the student has taken], one node for each rule (S_j, x_j), a source node s, a sink node t.
- Connect s to each course c_i with capacity 1 if the student has taken the course, otherwise 0 [note: equivalently, we can ignore the edge].
- Connect each rule (S_j, x_j) to t with capacity x_j.
- For each course c_i, connect it to all the rules (S_j, x_j) where $c_i \in S_j$, with capacity 1.

We then run any maximum flow algorithm on G to obtain a flow f^*. If the value of the maximum flow, $v(f^*)$, equals the total requirement $\sum_{j=1}^{m} x_j$, we conclude that the student can graduate.

For correctness, we show that $v(f^*) = \sum_{j=1}^{m} x_j$ if and only if all graduation rules can be satisfied:

($\Longrightarrow$) Suppose there exists a flow f^* with $v(f^*) = \sum_{j=1}^{m} x_j$. Then all edges from the rule nodes to t must be saturated. Hence, for each rule node (S_j, x_j), the total inflow equals x_j. By construction, this means there are x_j courses sending one unit of flow each to (S_j, x_j), i.e., these courses are used to satisfy that rule. Because each course node has capacity 1 on its outgoing edges, no course can contribute to more than one rule. Therefore, all rules are satisfied without overlap.

($\Longleftarrow$) Conversely, suppose the student can graduate. Then there exist disjoint subsets $X_1, X_2, \ldots, X_m$ of the courses the student has taken such that $X_j \subseteq S_j$ and $|X_j| = x_j$ for all j. We can use this to construct a flow f on G: for each $c \in X_j$, send one unit of flow along the path $s \to c \to (S_j, x_j) \to t$. Because the X_j's are disjoint and satisfy the required cardinalities, f is a valid flow. Moreover, it saturates all edges into t, so its value is $v(f) = \sum_{j=1}^{m} x_j$, implying f is a maximum flow.

The flow network G has $O(n+m)$ nodes and $O(n+m+\sum_{j=1}^{n} x_j)$ edges, both linear in the input size. Thus, the running time is dominated by the maximum flow computation. Using the Edmonds-Karp algorithm, the running time is $O(|V||E|^2)$, which is $O(N^3)$ if N denotes the input size. [Note: it is OK to just say the running time is the same as a maximum flow algorithm without considering the input size of this problem.]

Rubric:

Problem 1, ? pts

?

Problem 2, ? pts

?

Problem 3, ? pts

?

Problem 4, ? pts

?

Problem 5, ? pts

?