CSCI-570 Homework 6

Due Date: Tuesday, November 25th, 11:59pm

Name: Sakshi Kuldeep Dhariwal

USC ID: 1851490301

Problem 1. [Think through]
Consider a flow network $G = (V, E)$ with a source $s \in V$, a sink $t \in V$, capacities $c : E \to \mathbb{N}^+$ and a flow f. Prove or disprove the following statements:

(a) If f is a max-flow, then $f(e) = 0$ or $f(e) = c(e)$ on all $e \in E$ edges.

(b) There is a max-flow for which $f(e) = 0$ or $f(e) = c(e)$ on all $e \in E$ edges.

(c) If all capacities are different, then there is a unique min-cut.

(d) Adding a constant number δ to all capacities $c(e)$ does not change the min-cuts.

(e) Multiplying all capacities $c(e)$ by a constant number δ does not change the min-cuts.

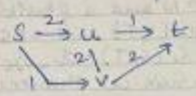
[5 points]

Q.1 Given: $G = (V, E)$ with source s, sink t, integer capacities $c: E \to N^+$ and a feasible flow f. A cut is any partition (S, T) with $s \in S$ and $t \in T$; its capacity is $C(S,T) = \Sigma_{(u,v) \in E, u \in S, v \in T} c(u,v)$. The max-flow min-cut theorem is useful as well.

(a) If f is a max-flow, then for every edge $e \in E$ either $f(e) = 0$ or $f(e) = c(e)$.

Intuition:
Maximum flow optimizes the total flow value, not each edge individually. It is possible and common that some edges are partially used in a max-flow. Flows can split, travel through parallel routes, or pass partially through an edge depending on the structure of the graph. Max flow is about pushing the maximum amount from s to t, not about filling every edge to its limit.

Counterexample:



- Flow path: $s \to u \to t$
  min = $\boxed{1}$ ∴ 1 unit of flow across s–u–t
  Residual edges for forward:
  $s \to u = 1$
  $u \to t = 0$
  Residual edges for backward:
  $u \to s = 1$
  $t \to u = 1$

- Flow path: $s \to u \to v \to t$
  min = $\boxed{1}$ ∴ 1 unit of flow across $s \to u \to v \to t$
  Residual edges for forward:
  $s \to u = 0$
  $u \to v = 1$
  $v \to t = 1$
  Residual edges for backward:
  ~~$u \to s = 2$~~
  $u \to s = 2$
  $v \to u = 1$
  $t \to v = 1$

- flow path: s – v – t min: $\boxed{1}$
  ∴ 1 unit of flow across s – v – t
  Residual edges of forward:
  s – v = 1
  v – t = 0
  Residual edges of backward:
  v – s = 1
  t – v = 2

∴ the maximum flow = 1+1+1 = 3
However, the edge $u \to v$ still has the value of $f(u, v) = 1$ even after finding the max flow.

Feasibility check for the obtained max-flow:

1. Capacity constraints: each $f(e) \leq c(e)$
   $s \to u = 2 \leq 2$
   $u \to t = 1 \leq 1$
   $s \to v = 1 \leq 1$
   $u \to v = 1 \leq 2$
   $v \to t = 2 \leq 2$
   ∴ all capacity constraints are satisfied.

2. Flow Conservation:

- At S: out $= 2+1 = 3$ (source) $\Rightarrow$ value $= 3$
- At u: in $= f(s,u) = 2$ out $= f(u,t) + f(u,v) = 1+1 = 2$ (Balanced)
- At v: $f(s,v) + f(u,v) = 1+1 = 2$ out $= f(v,t) = 2$ (Balanced)
- At t: in $= f(u,t) + f(v,t) = 1+2 = 3$

So f is a valid feasible flow of value 3

Consider the cut (S,T) with $S = \{s\}$ and $T = \{u,v,t\}$. The cut edges are $s \to u$ and $s \to v$. Their capacities sum to:

$$C(S,T) = c(s,u) + c(s,v) = 2+1 = 3$$

By the max-flow min cut theorem, no flow can have value greater than this cut capacity. Since our flow has value 3, it attains cut capacity & $\therefore$ is a maximum flow.

In the statement, it claimed that every edge in a max flow must be either 0 or full. But in the above example, the edge $u \to v$ has $f(u,v) = 1$ while $c(u,v) = 2$ and when max flow is achieved. That is a partially used edge in a max flow. $\therefore$ This statement is false.

b). Considering the same example from part (a)

$$\begin{tikzcd} s \arrow[r, "2"] \arrow[dr, "1"'] & u \arrow[r, "1"] \arrow[d, "2"] & t \\ & v \arrow[ur, "2"'] & \end{tikzcd}$$

There max-flow obtained was 3 and even after the max flow was obtained the edge $u \to v$ has the flow as 1.

The statement states that some maximum flow can always be chosen that only uses edges at either 0 or full capacity. But some network constraints force intermediate usage on some edges in every max flow.

Proof by contradiction:

- Case 1: Suppose $f(u,v) = 0$

Then the entire incoming flow to u (which is $f(s,u) \leq 2$) must leave it through $u \to t$. But $c(u,t) = 1$, so at most 1 unit can leave u to t.

Meanwhile s can send at most $2+1 = 3$ total, but to reach value 3 we need u to forward 2 units (if $f(s,u) = 2$); since $u \to t$ capacity is 1 we cannot push both units to t without using $u \to v$

A more direct contradiction : to attain total flow, $f(s,u) + f(s,v) = 3$; $f(s,u) \leq 2$ so either $f(s,u) = 2$, $f(s,v) = 1$ or vice versa. If $f(s,u) = 2$ and $f(u,v) = 0$ then must route 2 units to t via $u \to t$ but $u \to t$ supports only 1. So, $f(u,v) = 0$ cannot occur in a flow of value 3.

— Case 2 : Suppose $f(u,v) = 2$ (saturated)
Then the flow arriving at v is at least $f(u,v) = 2$, and $v \to t$ has capacity 2 so v can forward at most 2 units to t. But $s \to v$ may also send flow ; to get total flow 3 we need $f(s,v)$ possibly positive, and conservation might be violated : if $f(s,v) = 1$ and $f(u,v) = 2$, then v receives 3 units but can only send 2 to t, so net inflow at v would be positive (not allowed). ∴ to have value 3 with $f(u,v) = 2$, we must have $f(v,t) \geq 2$ and $f(s,v) = 0$ to avoid exceeding capacity out of v. Then $f(s,u)$ must be 3, but $s \to u$ capacity is 2 – impossible. Thus $f(u,v) = 2$ is incompatible with value 3.

∴ any max flow of value 3 must have $f(u,v) = 1$

— a strictly intermediate value. Hence no max-flow has every edge equal to 0 or fully saturated.

∴ the statement is false : there exist flow networks where every maximum flow must use some edge at a non-extreme value $0 < f(e) < c(e)$

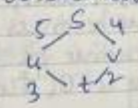
Q.1 c) The statement assumes that distinct edge capacities imply a unique min-cut. However, capacity of a cut is the sum of capacities of edges crossing the cut, not the individual capacities themselves. Even if all edges have distinct capacities, different subsets of edges may sum to the same total, producing multiple cut with the same minimum capacity.

Let E_1 and E_2 be 2 different sets of edges corresponding to two different cuts. Even if $c(e) \neq c(e')$ for all $e \neq e'$ it is possible that

$$\sum_{e \in E_1} c(e) = \sum_{e \in E_2} c(e)$$

This implies that 2 distinct cuts can have the same minimum capacity, so the min-cut is not necessarily unique.

Counterexample:



Edge capacities:
$s \to u = 5$, $s \to v = 4$, $u \to t = 3$, $v \to t = 2$
Compute capacities of possible cuts:

1) $S = \{s\}$, $T = \{u, v, t\}$ → cut edges $= 5 + 4 = 9$
2) $S = \{s, u\}$, $T = \{v, t\}$ → cut edges $= 3 + 4 = 7$
3) $S = \{s, v\}$, $T = \{u, t\}$ → cut edges $= 5 + 2 = 7$

Two cuts $S = \{s, u\}$ and $S = \{s, v\}$ both have same minimum cut capacity despite all edge capacities being distinct.

∴ the statement is False.

edge at a "non-extreme" value $0 < f(e) < c(e)$

(d) Intuition:

If you add δ to every edge, a cut that crosses k edges increases its capacity by $k \cdot \delta$. Cuts with more crossing edges get a larger additive increase; cuts with fewer crossing edges get a smaller increase. So relative order of cut capacities can change.

Consider two cuts with original capacities C_A and C_B and crossing edge counts k_A and k_B. After adding δ to each edge:

$$C_A' = C_A + k_A \delta \qquad C_B' = C_B + k_B \delta$$

If $C_A < C_B$ but $k_A > k_B$, then for sufficiently large δ we can have $C_A' > C_B'$ reversing the min-cut choice.

- Cut A has $C_A = 10$ with $k_A = 5$ crossing edges
- Cut B has $C_B = 12$ with $k_B = 2$ crossing edges

Originally A is the min-cut $(10 < 12)$: If we add $\delta = 5$ to every edge:

$$C_A' = 10 + 5 \cdot 5 = 35$$
$$C_B' = 12 + 2 \cdot 5 = 22$$

Now B has smaller capacity, so the min-cut changed. So in general adding a constant to all capacities can change which cut is minimum.

Only when all s-t cuts have the same number of crossing edges will adding δ preserve the min-cut; this is not true in general networks.

$\therefore$ adding a constant to every capacity may change the min-cuts, so the claim is false.

e) True for any positive $\delta > 0$

Intuition:
Multiplying all capacities by the same positive scalar scales every cut capacity by the same factor, so comparisons among cuts are preserved.

Proof:
Let $\delta > 0$. Define new capacities $c'(e) = \delta c(e)$ for every edge e. For any cut (S, T),

$$c'(S,T) = \sum_{\substack{(u,v): u \in S, \\ v \in T}} c'(u,v) = \sum_{\substack{(u,v) \in \\ \delta \text{crossing}}} \delta c(u,v) =$$

$$= \delta \sum_{(u,v) \text{ cross}} c(u,v) = \delta C(S,T)$$

Hence for any two cuts (S_1, T_1) and (S_2, T_2)
$C'(S_1, T_1) < C'(S_2, T_2) \Leftrightarrow \delta C(S_1, T_1) < \delta C(S_2, T_2)$
$\Leftrightarrow C(S_1, T_1) < C(S_2, T_2)$,
because multiplication by a positive scalar preserves order. Thus the set of cuts that minimize capacity is the same before and after scaling (and ties remains ties).

Edge Cases:
- If $\delta = 0$ then all capacities become 0 and the statement degenerates (all cuts have capacity 0).
- If $\delta < 0$ then capacities would no longer be valid (negative capacities) – we restrict to $\delta > 0$.

Proof by Contradiction:

Assume the opposite:
Suppose multiplying all capacities by $\delta > 0$ does change the min-cut.
Let $C_1 = (S_1, T_1)$ be a min-cut in the original network with capacity

$$c(C_1) = \sum_{e \in C_1} c(e)$$

Let $C_2 = (S_2, T_2)$ be some other cut such that

$$c(C_2) > c(C_1)$$

2) Multiply capacities by δ:

- The capacity of a cut after scaling:

$$c_\delta(C) = \sum_{e \in C} \delta\, c(e) = \delta \sum_{e \in C} c(e) = \delta \cdot c(C)$$

So, for our two cuts

$$c_\delta(C_1) = \delta \cdot c(C_1), \quad c_\delta(C_2) = \delta \cdot c(C_2)$$

Since $\delta > 0$ and $c(C_2) > c(C_1)$ we have

$$c_\delta(C_2) = \delta \cdot c(C_2) > \delta \cdot c(C_1) = c_\delta(C_1)$$

Our assumption was that scaling by δ changes the min-cut i.e. some other cut becomes smaller than C_1. But the inequality above shows that the order of capacities is preserved (C_1 still has the minimum value).

$\therefore$ there is a contradiction.

$\therefore$ multiplying all capacities by a positive constant δ does not change the min-cut.

$\therefore$ the statement is TRUE.

Problem 2.

Are max-flows and min-cuts unique?

(a) Give a simple example of a flow-network where the max-flow is unique, but it has several min-cuts.

(b) Give a simple example of a flow-network where the min-cut is unique, but is has several max-flows.

(c) Provide a polynomial-time algorithm which decides whether a flow-network has unique min-cut.

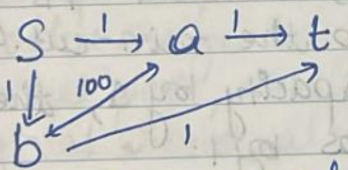
[Hint for (c): what can happen to the value of the max-flow if you change the capacities of *a* min-cut?]

[2+2+6 points]

Q.2 a) Vertices: $\{s, a, b, t\}$

Edges and capacities:

- $s \to a$ capacity: 1
- $s \to b$ capacity: 1
- $a \to t$ capacity: 1
- $b \to t$ capacity: 1
- $a \leftrightarrow b$ (both directions capacity: 100



Total flow out of s is: $1+1=2$

sum of edges into t is 2 $\therefore$ the max-flow value is 2.

- Because s only has 2 outgoing edges of unit capacity, any feasible flow of value 2 must saturate both s→a and s→b (each must carry 1). Likewise a→t and b→t must each carry 1. No other distribution is possible – the flow on these four edges is forced. Thus, the max-flow (the flow values on every edge) is unique.
- But consider cuts (partitions seperating s from t):

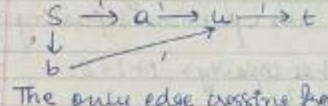
→ Cut S = {s} has capacity = 1+1 = 2.
  V = {a, b, t}

→ Cut S = {s, a} V = {b, t} has capacity
  s→b = 1 a→t = 1 ∴ 1+1 = 2.
→ Cut S = {s, b} V = {a, t} has capacity s→a (1) + b→t (1) = 2

So there are at least three different s-t cuts of capacity 2. Hence several min-cuts exist though the max-flow is unique.

∴ this network shows unique max-flow but multiple min-cuts.

b) Vertices: {s, a, b, u, t}
Edges and capacities:
s→a capacity 1
s→b capacity 1
a→u capacity 1
b→u capacity 1
u→t capacity 1



- The only edge crossing from the part that contains t (namely {u}) to t is u→t of capacity 1. Any cut that seperates t from the rest and isolates t will include this edge, cutting elsewhere gives larger capacity. ∴ the unique min cut value is 1, and the unique min-cut (set of edges crossing from the s-side to the t-side) is the singleton {u→t}. So min-cut is unique.
- The max flow value equals 1 (bottleneck u→t). However, this single unit of flow can be delivered to u either via s→a→u or via

$s \to b \to u$. That means there are multiple distinct maximum flows: one that uses the a-route, another that uses the b-route (or any convex combination if capacities allowed fractions — but we are integer here are at least two different integral max-flows). The edge $u \to t$ must carry 1 in all max-flows, but the choice of upstream path is not unique.

(c) Intuition:

- Run a max-flow algorithm and obtain a maximum flow f.
- Let G_f be the residual graph for f.
- Let R_s be the set of vertices reachable from s in G_f (do BFS/DFS from s using edges with positive residual capacity).
  - The cut $(R_s, V \setminus R_s)$ is a minimum s-t cut
- In general there may be many minimum cuts. However, the set of all S-sides of minimum cuts forms an interval: there is a unique minimal S-side (the intersection of all min-cut S sides) and a unique maximal S-side (the union of all min-cut S-sides).

- The minimal S-side is exactly R_s. The maximal s-side can be computed as $V \setminus R_t$, where R_t is the set of vertices from which t is reachable in the residual graph (equivalently perform BFS/DFS in G_f but on the graph with all edges reversed, started at t, or find all vertices that can reach t in G_f).
- All minimum cuts correspond exactly to sets S such that:

$$R_s \subseteq S \subseteq V \setminus R_t$$

$\therefore$ min-cut is unique iff the minimal and maximal S-sides coincide:

$$R_s = V \setminus R_t$$

Algorithm (polynomial-time):

1) Run a polynomial time max-flow algorithm such as Ford Fulkerson with capacity scaling to obtain a max-flow f and its residual graph G_f. (This step dominates time, call it $T_{maxflow}$)
2) Do a BFS/DFS in G_f from s using edges with positive residual capacity to compute R_s. Time $O(n+m)$
3) Do a BFS/DFS in G_f reversed (or equivalently

run BFS from t on the transpose) to compute the set R_t of vertices that can reach it in G_f. (Time $O(n+m)$) Call this set R_t.

4. Check whether:
$R_s \stackrel{?}{=} V \setminus R_t$
If equality holds → min-cut is unique else, there are multiple min cuts.

Overall time: $T_{maxflow} + O(n+m)$. If max-flow is given, then algorithm runs in $O(n+m)$)

Proof of Correctness:

1) R_s is the minimal s-side of any min-cut from a max-flow, the set of vertices reachable from s in the residual graph is the s-side of a min-cut; any other min-cut's s-side must contain R_s (otherwise there would be an augmenting path).

2) $V \setminus R_t$ is the maximal s-side of any min-cut Similarly, the set of vertices that cannot reach t in the residual graph must be included in every s-side of a min-cut; their complement is the maximal s-side

3) All min-cuts lie between these two extremes One can show (standard lattice property of minimum cuts) that any min-cut's s-sides satisfies $R_s \subseteq S \subseteq V \setminus R_t$. Every set in this interval corresponds to a min-cut

4) Uniqueness criterion
If $R_s = V \setminus R_t$, then there is only one possible S in that interval, so the min-cut is unique. If the two sets differ, pick any vertex x in the symmetric difference - you can change membership of x and obtain a different min-cut; hence non-unique.

Q-1 ~~We want to~~

Q-1

Runtime:
Max-Flow Computation: F
Check each edge in min-cut: $O(E)$ edges
Overall Runtime: $O(E \cdot F)$ If F = Ford-Fulkerson with capacity scaling then: $O(E \cdot E^2 \log C)$
$\Rightarrow O(E^3 \log C)$ where C is max flow.

Problem 3.
Let $G = (V, E)$ be a flow network with source s, sink t, and integer capacities. Suppose that we are given a maximum flow f in G.

(a) Suppose that we *increase* the capacity of a single edge $(u, v) \in E$ by 1.
Give an $O(n + m)$ (i.e. $O(V + E)$) runtime algorithm to update the maximum flow.

(b) Suppose that we *decrease* the capacity of a single edge $(u, v) \in E$ by 1.
Give an $O(n + m)$ (i.e. $O(V + E)$) runtime algorithm to update the maximum flow.

[Hint: one iteration of the Ford-Fulkerson algorithm and graph traversal algorithms each can be run in $O(n + m)$.]

[10 points]

Q3 We are given a flow network $G = (V, E)$ with integer capacities and a maximum flow f. A single edge (u, v) has its capacity changed by $+1$ or -1, and we want to update the maximum flow in $O(n+m)$ time.

Once a max-flow is known, adjusting capacity on one edge can only change the max-flow by at most 1 unit.
Therefore, we never need to recompute the whole max-flow; we only need to run one augmenting-path search in the residual graph.

Intuition:
When we increase the capacity of a single edge (u, v) by 1, the only possible effect on the maximum flow is that the value of max-flow might increase by at most 1 unit.

1. All capacities are integers, and flows are integral. The max-flow value always changes in whole units.
2. Increasing the capacity of one edge changes the network only slightly.
Before the change we had a maximum flow, so there was no augmenting path from s to t in the residual graph!
3. By adding 1 extra capacity to (u, v), we may create exactly one new residual path. This new unit of residual capacity might connect previously disconnected parts of the residual graph, allowing a new augmenting path to appear.
4. If an augmenting path exists, only 1 unit of additional flow can pass through it, because every edge in the residual graph has integral capacities and we only increased one edge by 1.

Thus increasing one edge's capacity cannot cause a large restructuring of the entire flow;

If the new residual edge allow a single new augmenting path from s to t, we push one extra unit, the max-flow value increases by 1, else the existing flow is still maximum.

∴ the flow may grow by at most 1, and testing this requires only one augmenting-path search

Algorithm:

1) Update residual graph
In the residual graph G_f, increase the residual capacity of edge (u, v) by 1. (This means adding 1 extra unit to the forward residual edge.)

2) Run one BFS/DFS search from s to t
Use BFS or DFS to see if there exist an augmenting path in the updated residual graph.
BFS/DFS runs in $O(n+m)$ time.

3) If an augmenting path exists:
- Augment by 1 unit along that path
- Update the flow values accordingly
- This gives the new maximum flow (value increases by exactly 1)

4) If no augmenting path exists:
- The previous flow is still a maximum flow
- No further change required.

Correctness:
Once capacity of a single edge increases:
- The flow value can increase by at most 1 (Max-flow Integrality Theorem).
- One BFS/DFS suffices to detect and perform that single augmentation.
- We never need more than one augmentation step.

Running Time:
- Residual update: $O(1)$
- BFS/DFS: $(O(n+m))$
Total: $O(n+m)$

Part (b):

When we reduce the capacity of one edge (u,v) by 1, the first question is:
if the edge (u,v) is previously full (saturated) in the max flow.

This leads to two cases:

Case 1: Edge was not saturated.
If $f(u,v) < c(u,v)$,
then after decreasing the capacity:

$$f(u,v) \leq c(u,v) - 1$$

so the flow is still feasible.

The edge had a slack, so lowering its capacity does not force us to change anything in the flow. A maximum that did not use this edge fully remains maximum.

Case 2: Edge was saturated
If $f(u,v) = c(u,v)$,
then after decreasing capacity, the current flow violates capacity constraints by exactly 1 unit on edge (u,v).
We must remove that extra flow unit.

a) We can reroute that 1 unit elsewhere:
If there exists a path in the residual graph that allows us to push that unit of flow around (u,v), then we can repair the flow without changing its total value.

We try to find another way to carry the same amount of flow from u to v without using the original edge.

If such a routing exists, the max-flow value does not change.

Possibility : No rerouting exists.
If no such alternative path exists, then the 1 unit of flow forced through (u, v) cannot be pushed anywhere else.
Hence we are forced to reduce the total s-t flow by 1.

The edge (u, v) was a bottleneck, and reducing its capacity shrinks the overall max-flow by exactly 1.
We cancel one unit of flow by following a path of flow and undoing it.

Algorithm :
1) Try to reroute the excess 1-unit of flow :
- In the residual graph, temporarily add +1 capacity to the reverse residual edge (u, v) (this represents reducing flow).
- Now look for an augmenting path from u to v in the residual graph (this path would push flow back backwards, cancelling the excess)
- Search takes $O(n+m)$

2) If such a u→v path exists
- Use it to send back 1 unit of flow
- The flow becomes feasible again without
- changing the total s→t flow value.
- The max-flow value remains the same.

3) If no such path exists :
- It is impossible to re-route the excess flow inside the interior of the graph.
- Therefore the overall max-flow must decrease by exactly 1.

To fix:

- Find an $s \to t$ path carrying 1 unit of flow that uses the edge (u,v)
- Cancel that 1 unit by sending one unit of flow back through the reverse edges (a DFS/BFS in residual graph).
- This decreases the total flow value by 1.

(b) Proof of Correctness.

We start with a maximum flow f in a directed graph $G = (V, E)$.

One edge (u,v) has its capacity decreased from $c(u,v)$ to $c(u,v)-1$

If $f(u,v) < c(u,v)$, the flow remains feasible and maximal.

So assume the only violated constraint is exactly

$$f(u,v) = c(u,v) > c(u,v)-1$$

Thus exactly one unit of excess needs to be removed from (u,v). Our algorithm tries to reroute this excess one unit of flow by checking if there is a path in the residual graph from u to v. If rerouting is possible → max-flow stays. If not → max-flow must decrease by 1.

We must show for proof of correctness:

(i) Feasibility

Case A: A $u \to v$ path exists in the residual graph.

The residual graph already contains an edge $v \to u$ of residual capacity 1 (representing reducing $f(u,v)$ by 1).

If the residual search finds a path

$u \to v$

in the residual graph, then:

- Send +1 unit along that path (forward on some edges, backward in others].

- Reduce flow on (u,v) by 1.

$\therefore f(u,v) = c(u,v) - 1$

which satisfies the new capacity.
All other edges maintain capacity constraints because the algorithm uses only residual capacity edges. $\therefore$ the new flow is feasible

Case B: No $u \to v$ residual path exists.
If no rerouting is possible, the algorithm selects an $s \to t$ flow path that used edge (u,v) and cancels one unit of flow along that path (i.e. sends 1 unit backward)

This reduces: $f(u,v) \leftarrow f(u,v) - 1 = c(u,v) - 1$
and because this cancels exactly one unit of flow across all edges on that path, the resulting flow satisfies:

- All capacity constraints
- All flow-conservation constraints

$\therefore$ new flow is feasible.

2) Optimality:

Case A: A $u \to v$ residual path exists.
Then:

- We reroute the excess flow u to v
- The value of the $s \to t$ flow remains unchanged

After rerouting, the residual graph becomes exactly same as original residual graph before capacity change.
Before the capacity decrease, (u,v) was saturated: its forward residual capacity was 0 & its reverse residual capacity was positive.
After decreasing the capacity & then rerouting one unit of flow from u to v we reduce the flow on (u,v) by 1. This restores the forward & reverse residual capacities of (u,v) to exactly the same values they had before the capacity change.
All other changes occur only along the $u \to v$ residual path, which already existed in the original residual graph of the maximum flow.
Since the original residual graph had no $s \to t$ augmenting path, the new residual

graph also has none. Thus the updated flow remains a maximum flow under new capacities

Case B: No $u \to v$ residual path exists.

Let R be the set of vertices reachable from s in the original residual graph.
Since f was maximum, $t \notin R$, and $(R, V \setminus R)$ is a min-cut of capacity $|f|$.
Consider the saturated edge (u, v):

- $u \in R$
- If $v \in R$, then there would be a residual path from $u \to v$. But we are in the case where no such path exists.
- So $v \notin R$

Thus (u, v) crosses the minimum cut.
Originally this edge contributed its full capacity $c(u, v)$ to the min-cut.
After decreasing its capacity by 1, the capacity of the min-cut drops by 1.
So the new min-cut capacity is:

$$|f| - 1.$$

By the Max-flow Min-cut theorem:

- No feasible flow can exceed $|f| - 1$
- So the max-flow must drop by 1

The algorithm cancels 1 unit of flow along an $s \to t$ flow path containing (u, v), producing a feasible flow of value:

$$|f| - 1$$

Since this equals the new min-cut capacity, the flow is maximum.

Runtime :

→ Every step involves only :

- one residual capacity update
- one BFS / DFS

All in $O(n+m)$ time.

Problem 4.

We define the *Escape Problem* as follows. We are given a directed graph $G = (V, E)$ (imagine a network of roads). A certain collection of nodes $X \subset V$ are designated as populated nodes, and a certain other collection $S \subset V$ are designated as safe nodes. (Assume that X and S are disjoint.) In case of an emergency, we want evacuation routes from the populated nodes to the safe nodes. A set of evacuation routes is defined as a set of paths in G so that (i) each node in X is the tail of one path, (ii) the last node on each path lies in S, and (iii) the paths do not share any edges. Such a set of paths gives a way for the occupants of the populated nodes to "escape" to S, without overly congesting any edge in G.

(a) Given G, X, and S, show how to decide in polynomial time whether such a set of evacuation routes exists.

(b) Suppose we have exactly the same problem as in (a), but we want to enforce an even stronger version of the "no congestion" condition (iii). Thus we change (iii) to say "the paths do not share any nodes." Provide a polynomial time algorithm to decide whether such a set of evacuation routes exists.

[Hint for (b): split vertices v into v_{in} and v_{out} and connect them with a unit capacity: $v_{in} \xrightarrow{1} v_{out}$]

[3+5 points]

Q.4 Given: A directed graph $G = (V, E)$, a set of populated vertices $X \subset V$ (sources of people who must evacuate), and a set of safe vertices $S \subset V$ (destinations). We want a collection of $|X|$ paths so that

(i) every $x \in X$ is the start (tail) of one path,
(ii) each path ends at some vertex in S, and
(iii) the paths are disjoint in the specified way (edge-disjoint in (a), node-disjoint in (b)).

(a) Edge-disjoint evacuation routes.

Goal: Decide whether there exist $|X|$ directed paths, one from each $x \in X$ to some vertex in S, such that no two paths share any edge.

Intuition:

i) Edge-disjoint paths = no two people use the same road. This happens in a flow network when each edge has capacity 1.

ii) Each populated node needs exactly one path. This means we want to send $|X|$ units of flow, one unit for each populated node.

iii) Any safe node can accept multiple evacuees. So the sink must allow many units to enter.

∴ now we need to decide if we can send $|X|$ units of flow from all populated nodes to any safe node, using edges only once.

Algorithm:

Construct a flow network G':

i) Original Structure:
- For each original edge $e \in E$, assign capacity 1

ii) Super-source
- Add a new node s^* which will act as the super source
- For every populated node $x \in X$, add edge $s^* \to x$ with capacity 1

iii) Super-sink:
- Add a new node t^* which will act as the super sink.

- For each safe node $s \in S$ add edge $s \to t^*$ with capacity $|X|$ (enough to allow multiple people to arrive)

iv) Compute max flow from s^* to t^* using any polynomial-time max flow algorithm such as Ford Fulkerson with capacity scaling [$O(m^2 \log C)$] or Edmonds-Karp algorithm

v) Decision:
- If the maximum flow value is $|X| \to$ Yes, evacuation possible
- Otherwise $\to$ NO, evacuation not possible.

Proof of Correctness:

The maximum $s^* - t^*$ flow in G' has value $|X|$ if and only if there exists $|X|$ directed paths in G such that
- Each path starts at a distinct $x \in X$
- Each path ends at some node of S
- No two paths share any original edge (edge-disjoint)

$\Rightarrow$ If there are $|X|$ edge-disjoint x to S paths in G', then maxflow $(G') \geq |X|$

Proof: Suppose $P = \{P_x : x \in X\}$ is a collection of $|X|$ directed edge-disjoint paths where each P_x starts at x & ends at some $s(P_x) \in S$. For each path P_x put one unit of flow along the corresponding sequence of edges in G': send 1 unit on edge $s^* \to x$ then along every original edge of P_x (each of those has capacity 1 & by edge-disjointness is used by at most one path), & finally on edge $s(P_x) \to t^*$. Because the paths are edge-disjoint, every original edge in G' is used by at most one unit of flow, so no capacity is violated, & because each $s^* \to x$ has capacity 1 & we use each exactly once, the flow is feasible. Total flow value is $|X|$. Thus max flow is atleast $|X|$.

$\Leftarrow$ If maxflow $(G') = |X|$, then there exists $|X|$ edge-disjoint x to S paths in G.

Step 1 – Integrality:

All capacities in G' are integers (1 on original edges, $|X|$ on edges to t^*)

By the Integrality theorem, there exists a maximum flow f in which every edge carries an integer amount of flow. $\therefore$ each original edge carries either 0 or 1 unit.

Step 2 – Every $s^* \to x$ is saturated:
The total capacity out of s^* is

$$\sum_{x \in X} cap(s^* \to x) = |X|$$

If the max flow value equals $|X|$, all this capacity must be used, so every edge $s^* \to x$ carries exactly 1 unit. Hence one full unit of flow originates from each populated node x.

Step 3 – Flow decomposition:
The integral flow f can be decomposed into unit $s^* \to t^*$ paths (and cycles, which we discard). Since the total flow is $|X|$, we obtain exactly $|X|$ such unit paths. Each path starts with a distinct $s^* \to x$ (because each is saturated with 1 unit) & ends at some $s \to t^*$. Remaining

the artificial start and end edges yields a directed path in G from that x to some $s \in S$.

Step 4: Edge-disjointness:
Every original edge has capacity 1, so the integral flow assigns at most 1 unit to it. Thus no 2 of the extracted paths can use the same original edge. Hence resulting x-to-S paths are edge disjoint.

Therefore we obtain $|X|$ directed edge-disjoint paths, each starting at a distinct $x \in X$ and ending at some $s \in S$.

Running Time:
Graph size grows by only $O(|V|)$
Max-flow runs in polynomial time (eg: $O(VE^2)$ for Edmonds-Karp or $O(E^2 \log C)$ for ford fulkerson with capacity scaling.

b) Node-Disjoint evacuation routes.

Intuition:
Split every node v into two nodes v_{in} and v_{out} with an edge
$v_{in} \to v_{out}$ (capacity = 1)
This edge acts as the node's capacity
We take capacity 1 because only one unit of flow can pass from v_{in} to v_{out}, so at most one path can use node v. This enforces node disjointness.

Suppose original graph has
$u \to v \to w$
After splitting:
$u_{in} \to u_{out} \to v_{in} \to v_{out} \to w_{in} \to w_{out}$
(cap = 1) (cap = 1) (cap = 1)

All original edges $u \to v$ become:
$u_{out} \to v_{in}$ (capacity = ∞)
We use capacity = ∞ on original edges because we only want to enforce node capacity, not restrict edges. If we put capacity 1 on all edges, we accidentally

disallow legitimate structures. The only bottleneck should be the node-capacity edges.

∴ Node-split edges
$v\_in \rightarrow v\_out$ (capacity = 1)
Original edges $(u \rightarrow v)$ become:
$u_{out} \rightarrow v_{in}$ (capacity = ∞)

∞ simply means the flow is not limited by this edge and it is only limited by node-capacity edges. Hence, we can use a large number like $|X|$ for implementation & design of algorithm.

Algorithm:

1) Split every vertex:
For each vertex $v \in V$
- Create two nodes v_{in} and v_{out}.
- Add a capacity-1 edge
$v_{in} \rightarrow v_{out}$
This enforces that at most one path can pass through v.

2) Transform original edges
For every original directed edge $(u \rightarrow v) \in E$
Add edge
$u_{out} \rightarrow v_{in}$
with capacity = ∞ (or equivalently, capacity = $|X|$)
This allows unlimited flow across original edges, because only nodes should enforce disjointness, not edges.

3) Add super source s^*
For every populated node $x \in X$:
Add edge $s^* \rightarrow x_{in}$ with capacity 1
This ensures exactly one flow unit originates from each populated node.

4) Add super sink t^*
For every safe node $s \in S$:
Add edge $s_{out} \rightarrow t^*$ with capacity 1
This ensures each safe node can be used by at most one evacuation path (node-disjointness).

5) Run a max flow algorithm such as

Edmonds Karp or Ford Fulkerson with capacity scaling) from s^* to t^* as these are polynomial time algorithms.

6) Decision
If max-flow = $|X|$, then Yes, node-disjoint evacuation path exists, otherwise, No the path does not exist.

Proof of Correctness:

1) Vertex splitting enforces node-disjointness:
- Each node v is split into $v_{in} \to v_{out}$ with capacity 1
- At most one path can pass through v, ensuring node-disjoint paths.

2) Edge transformation allows unrestricted flow:
- Original edges $u \to v$ become $u_{out} \to v_{in}$ with infinite capacity. Only nodes limit flow, not edges, so paths can freely traverse edges.

3) Super-source and super sink enforce path requirements.
- $s^* \to x_{in}$ with capacity $1 \to$ ensures one path per populated node
- $s_{out} \to t^*$ with capacity $1 \to$ ensures each safe node is used at most once.

4) Max-flow corresponds to evacuation paths.
- Each unit of flow = one path from x to s
- Capacity-1 edges guarantees node-disjointness
- If max-flow = $|X|$, all populated nodes have a disjoint evacuation path.
- If max-flow $< |X|$, it is impossible to route node-disjoint paths for all populated nodes.

$\therefore$ algorithm correctly decides whether node-disjoint evacuation path exists.

Runtime:
In general max-flow algorithms take $O(\text{max-flow} \times (M+N))$, where M and N are the number of edges and nodes, and max-flow = $|X|$.
If we apply Ford Fulkerson with capacity scaling runtime becomes: $O((M+N)^2 \log |X|)$ as runtime for Ford Fulkerson with capacity scaling is $O(E^2 \log C)$