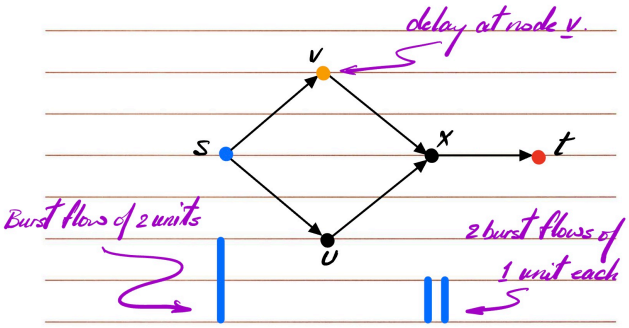
Network Flow

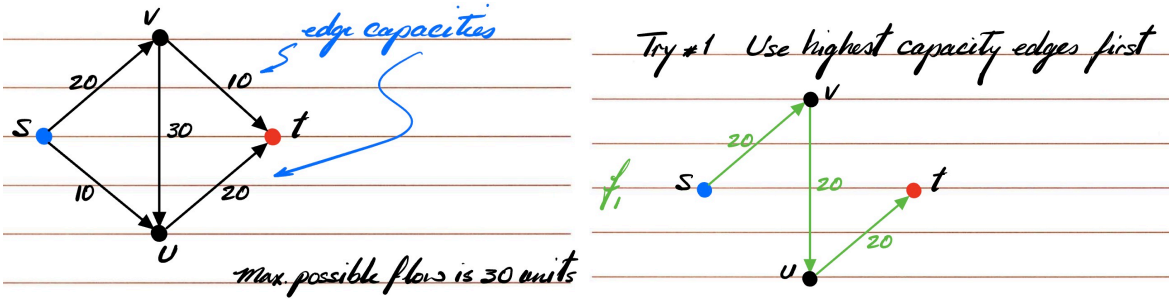
Julius April 8, 2024

1 Definitions

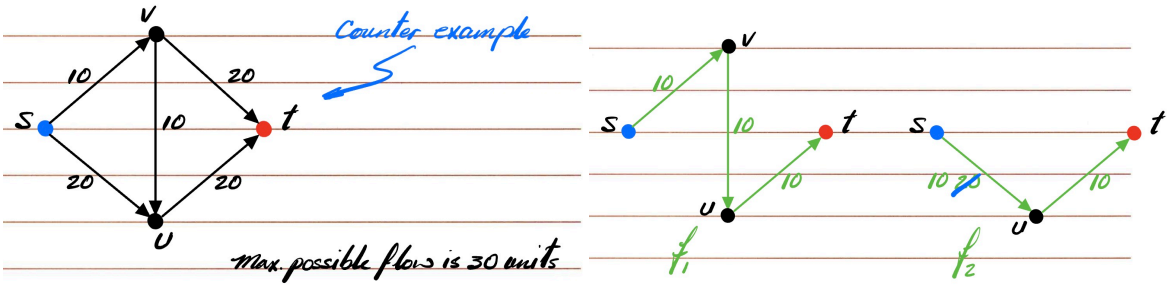
Flow Network: a directed graph $G = (V, E)$ with two special vertices: a source $s \in V$ and a sink $t \in V$. And Each edge $e = (u, v) \in E$ has a non-negative capacity $c_e \geq 0$.
Assumptions: No edges enter s or leave t; At least one edge is connected to each node. O(m+n) = O(m); All capacities are integers.
Flow f(e): the flow through edge e, with $0 \leq f(e) \leq c(e)$ and for all $v \in V \setminus \{s, t\}$, $\sum_{\text{e into v}} f(e) = \sum_{\text{e out of v}} f(e)$
Transient or Burst Flow: the flow velocity and pressure are changing with time.



Value of Flow: for a steady state flow (no velocity change in time), the value of the flow is the total flow out of the source, i.e. $v(f) = \sum_{\text{e out of s}} f(e)$
Max Flow Problem: Given a flow network G, find a s-t flow with maximum value.
Counter Example for Use Highest Capacity Edge First:



Counter Example for Use Lowest Capacity Edge First:



Obviously, the order of path selection matters, sometimes we need to undo the flow we have chosen.

2 Ford-Fulkerson Algorithm

2.1 Residual Graph

Residual Graph: Given a flow network G and a flow f, the residual network G_f is a directed graph with the same vertices as G. Consider all edges in flow f:

If $f(e) = 0, G_f$ will have 1 **Forward Edges** e with capacity c_e;
If $f(e) = c_e, G_f$ will have 1 **Backward Edges** e' in opposite direction of e with capacity c_e;
If $0 < f(e) < c_e, G_f$ will have 2 edges, one **Forward Edges** with capacity $c_e - f(e)$ and one **Backward Edges** with capacity $f(e)$
Bottle Neck: In G_f, a simple path P from s to t, $bottleneck(P)$ is the minimum residual capacity of any edge on P. Then given a flow f and a path P in G_f, we can augment the flow along P by $bottleneck(P)$

2.2 Flow Augment

$FlowAugment(f,\ G_f,\ P)$:

Let $b = bottleneck(P)$
Copy f to f'
For each edge $e \in P$:
    if $e = (u, v)$ is a forward edge in G_f: $f'(e) = f(e) + b$;
    else: let $e' = (v, u) \in f',\ f'(e') = f(e') - b$;
return f'

Theorem: If f is a valid flow, then f' is also a valid flow.

Proof. First, Check the Capacity Condition: for each edge $e \in E$, $0 \le f'(e) \le c_e$.
If e is a forward edge, then $bottleneck(P) \le c_e - f(e)$, so $f'(e) = f(e) + bottleneck(P) \le c_e$;
If e is a backward edge, then $bottleneck(P) \le f(e) \le c_e$, so $0 \le f(e) - bottleneck(P) = f'(e) \le c_e$.
Seconde, Check the Flow Conservation Condition:
Since f is a valid flow, for each vertex $v \in V \setminus \{s, t\}$, $\sum_{\text{e into v}} f(e) = \sum_{\text{e out of v}} f(e)$. Consider each node on the Augment Path P, say edges e_i into v and e_{i+1} out of v, $bottleneck(P) = b$:
If e_i, e_{i+1} are both Forward edges, then $f'(e_i) = f(e_i) + b, f'(e_{i+1}) = f(e_{i+1}) + b$, so

$$\sum_{\text{e into v}} f'(e) = \sum_{\text{e into v}} f(e) + b = \sum_{\text{e out of v}} f(e) + b = \sum_{\text{e out of v}} f'(e)$$

If e_i, e_{i+1} are both Backward edges, then $f'(e_i) = f(e_i) - b, f'(e_{i+1}) = f(e_{i+1}) - b$, so

$$\sum_{\text{e into v}} f'(e) = \sum_{\text{e into v}} f(e) - b = \sum_{\text{e out of v}} f(e) - b = \sum_{\text{e out of v}} f'(e)$$

If e_i is Forward edge, e_{i+1} is Backward edge, then say e'_{i+1} is the corresponding Forward Edge into v, then $f'(e_i) = f(e_i) + b, f'(e'_{i+1}) = f(e'_{i+1}) - b$, so

$$\sum_{\text{e into v}} f'(e) = \sum_{\text{e into v}} f(e) + b - b = \sum_{\text{e out of v}} f(e) = \sum_{\text{e out of v}} f'(e)$$

If e_i is Backward edge, e_{i+1} is Forward edge, then say e'_i is the corresponding Forward Edge out of v, then $f'(e'_i) = f(e'_i) - b, f'(e_{i+1}) = f(e_{i+1}) + b$, so

$$\sum_{\text{e into v}} f'(e) = \sum_{\text{e into v}} f(e) = \sum_{\text{e out of v}} f(e) + b - b = \sum_{\text{e out of v}} f'(e)$$

□

2.3 Algorithm Implementation and Proof of Correctness

`Ford-Fulkerson Algorithm: MaxFlow(G, s, t):`

Initially, $f(e) = 0$ for all $e \in G$
While there is a simple s-t path P from s to t in residual graph G_f:
    $f' = FlowAugment(f,\ G_f,\ P)$
    $f = f'$
    Update G_f
return f

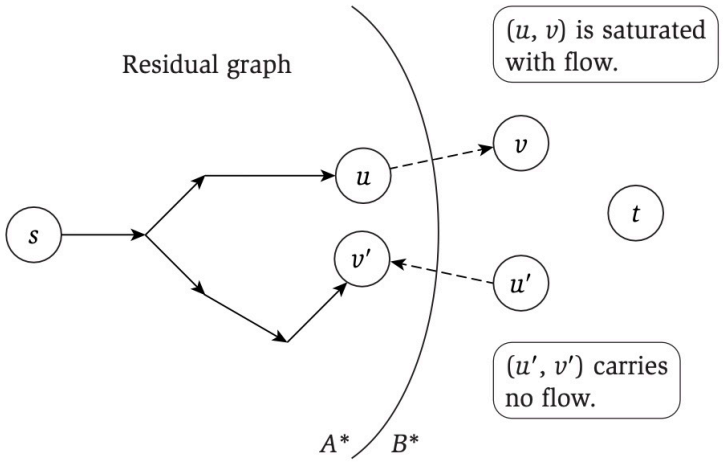
Proof. **While Loop in Ford-Fulkerson Algorithm will terminate.**
Because f is an integer flow, v(f) increases by at least 1 unit at each iteration, and $v(f) = \sum_{\text{e out of s}} f(e) \le \sum_{\text{e out of s}} c_e = C$, thus the while loop must terminate in at most C iterations. Easy proof by Mathematical Induction. □

Proof. **Ford-Fulkerson Algorithm returns a Max Flow.**
Define Cut: A cut (A, B) of a flow network G is a partition of V into two sets A and B such that $s \in A$ and $t \in B$.
Define Capacity Of Cut C(A, B): the total capacities of the edges from A to B, $C(A, B) = \sum_{\text{e out of A}} c_e$.

Fact $v(f) = f^{out}(A) - f^{in}(A)$, where f be any s-t flow and (A, B) be any S-T cut of G
With the above fact, we get:

$$v(f) \leq f^{out}(A) \leq \sum_{\text{e out of A}} c_e = C(A, B)$$

That means **Any s-t flow value is upper bounded by the capacity of any S-T Cut.**
Recall that Ford-Fulkerson Algorithm terminates when the flow of f has no S-T path in the residual graph G_f. We can proof at that terminate time, there is an S-T Cut (A^*, B^*), where $v(f) = C(A^*, B^*)$. **Consequently, f has the maximum value of any flow in G, and** (A^*, B^*) **has the minimum capacity of any S-T cut in G.**
Construct A^* as the set of all vertices reachable from s in G_f, and $B^* = V \setminus A^*$.
Consider all the edges in G from A^* to B^*, and from B^* to A^*:



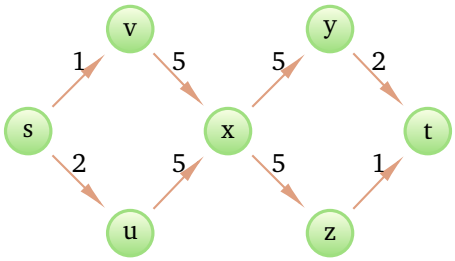
Because there is no S-T path in G_f, all (v, u) edges from A^* to B^* are saturated, and all (u', v') edges from B^* to A^* are empty, otherwise there should be corresponding Backward edge (v', u') from A^* to B^*, thus $f(e) = c_e, f(e') = 0$:

$$v(f) = f^{out}(A^*) - f^{in}(A^*) = \sum_{\text{e out of } A^*} c_e - 0 = C(A^*, B^*)$$

□

2.4 Minimum Cut may not be Unique

Solution 1: Ford-Fulkerson Algorithm terminate with the minimum cut (A^*, B^*) **nearest to the source.** So we can use above property to find the minimum cut closest to Source and Sink (flipping direction of all edges) in a network. If they are the same cut, then the minimum cut is unique.
Use any algorithm to find the maximum flow, then consider the residual graph G_f now, run BFS/DFS from s and say all the vertices reachable from s via an augmenting path $\cup\{s\}$ is A, then flip all edges and run BFS/DFS from t and all vertices that are reachable from t via an augmentation path $\cup\{t\}$ is B. Then both $(A, V - A)$ and $(B, V - B)$ are minimum cuts. There is a unique minimum cut iff $A = V - B$.



In the above example, $A = \{s\}, B = \{t\}$ are not the same, so both $(\{s\}, \{v, u, x, y, z, t\})$ and $(\{t\}, \{s, v, u, x, y, z\})$ are minimum cut, not unique.
So Given a maximum flow f in a flow network G with m edges, we can easily determine if G has a unique minimum cut in $O(m)$ **time by constructing the residual graph** G_f**.**

Solution 2: Use the Definition of **Critical Edge** and **Pseudo-Critical Edge**. For each edge e in the given minimum cut, increase its capacity and compute the value of maximum flow again. If the value does not increase, then it is a Pseudo-Critical Edge, i.e. there is anther minimum cut that does not contain this edge, so the given minimum cut is not unique. Otherwise, if for all edges the maximum flow increase, then all edges in the given minimum cut are Critical Edges, and the given minimum cut is unique.

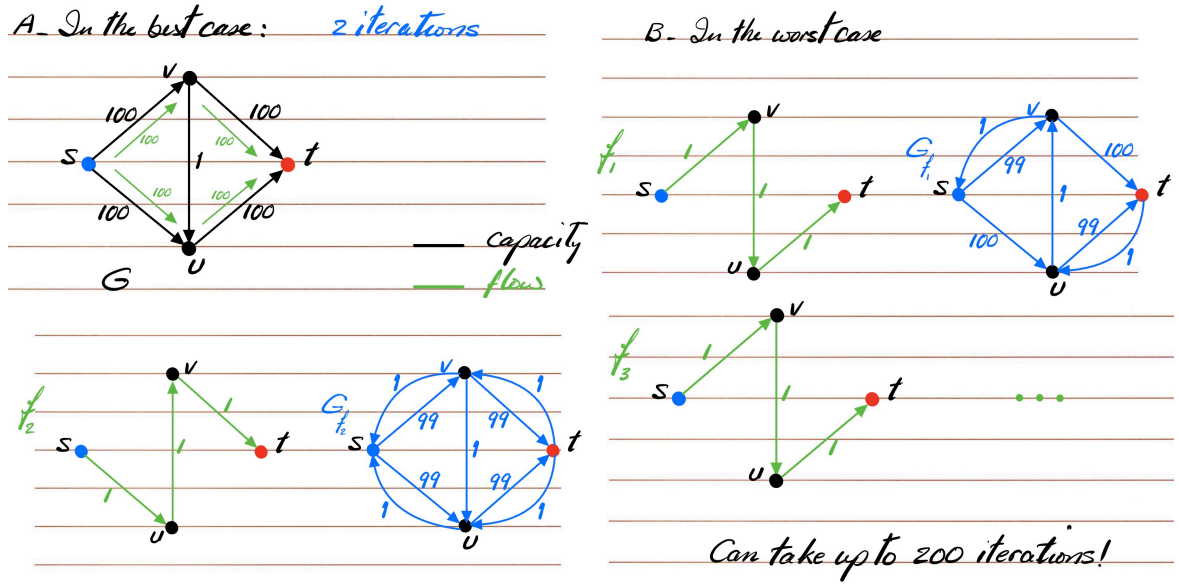
2.5 Complexity Analysis

The outer while loop for simple s-t path in the G_f runs at most $O(C)$ times, and each iteration takes $O(m)$ time to find a path, $O(m)$ time to update the flow and $O(m)$ time to update the G_f. Thus the time complexity of Ford-Fulkerson Algorithm is $O(C \cdot m)$, where $C = \sum_{\text{e out of s}} c_e$, and m is the number of edges. It's not efficient.

3 Scaled Version of Ford-Fulkerson Algorithm

3.1 Implementation

Observation: the order of path we find for the augment flow matters, in worst case we increase the flow by 1 at each iteration, and the number of iterations is $O(C)$.
Initially, set Δ is the largest power of 2 that is less than or equal to the largest capacity of edges out of Source. Let $G_f(\Delta)$ be the subset of the G_f consisting only of edges with residual capacity of at least Δ
Initially, f(e) = 0 for all e in G
While $\Delta > 0$:
  While there is simple s-t path P in $G_f(\Delta)$:
    $f' = FlowAugment(f,\ G_f(\Delta),\ P)$
    $f = f'$, Update $G_f(\Delta)$, filter all Forward Edge if updated capacity less than Δ
  $\Delta = \Delta/2$
return f

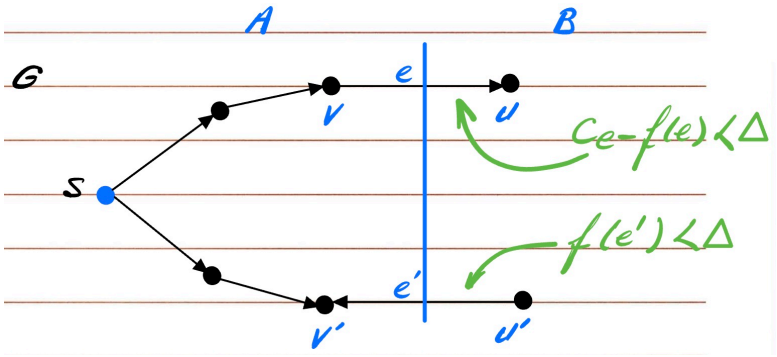


3.2 Time Complexity Analysis

Fact: During each outer while Δ_Scaling Iteration, each augmentation increase $v(f)$ by at least Δ.
Theorem: At the end of each Δ_Scaling Iteration, there is an s-t cut (A, B) in G for which $C(A, B) \leq v(f) + m\Delta$.

Proof. Similar to the proof of Ford-Fulkerson Algorithm, we can construct the set A as the set of all vertices reachable from s in $G_f(\Delta)$, and $B = V \setminus A$.
Since there are at most m edges across the cut (A, B), each with residual capacity less than Δ, we have $C(A, B) \leq v(f) + m\Delta$. □

Using the above theorem, If f^* is the final maximum flow, it must be $v(f^*) \leq C(A, B) \leq v(f) + m\Delta$. So in the next Δ_Scaling Iteration, we set $\Delta = \Delta/2$, the most number of iterations is $m\Delta/(\Delta/2) = 2m$.
Time Complexity: the outer while loop (number of Δ-scaling phases) runs at most $O(\log_2 C)$ times, and inner



while loop runs at most $O(2m)$ times. During each inner iteration, it takes $O(m)$ to find a path, $O(m)$ time to update the flow and $O(m)$ time to update the G_f. Thus the time complexity of Scaled Ford-Fulkerson Algorithm is $O(m^2 \log C)$. It is **weakly polynomial**, only requiring time proportional to **the number of bits** needed to specify the capacities in the input to the problem.

3.3 Strongly vs Weakly polynomial time

Strongly Polynomial Time: the running time is bounded by a polynomial in the number of integers in the input.
Weakly Polynomial Time: the running time is bounded by a polynomial in the number of bits (but not in the number of integers) in the input.
Pseudo Polynomial Time: its running time is bounded by a polynomial in the numeric value of the input, but exponential in the number of bits in the input.

4 Edmonds-Karp Algorithm

BFS: Given a flow network G and a flow f, we can use BFS to find the shortest s-t path, path with the fewest number of edges, in the unweighted residual graph G_f.

Edmonds-Karp Algorithm: MaxFlow(G, s, t) $O(m^2n)$:

Initially, $f(e) = 0$ for all $e \in G$
While there is a simple s-t path P from s to t in residual graph G_f:
  Use BFS to find the **shortest s-t path P** in G_f
  $f' = FlowAugment(f,\ G_f,\ P)$
  $f = f'$
  Update G_f
return f

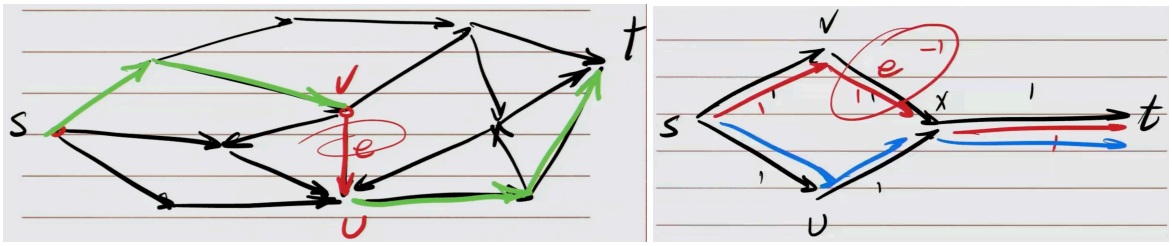
Algorithm	Time Complexity	Polynomial
Ford-Fulkerson	$O(Cm)$	Pseudo-Polynomial
Scaled Ford-Fulkerson	$O(m^2 log_2 C)$	Weakly Polynomial
Edmonds-Karp	$O(m^2 n)$	Strongly Polynomial
Orlin+KTP	$O(mn)$	Strongly Polynomial

More recently developed approximation methods solve the Maximum Flow in close to linear time with respect to the number of edges $O(m^{4/3})$.

5 Application of Network Flow

5.1 Given the maximum s-t flow f for a network G=(V, E), with integer edge capacities. How can you compute the maximum flow of G' that is identical to G except that the capacity of edge e is decreased by one in O(|V|+|E|) time?

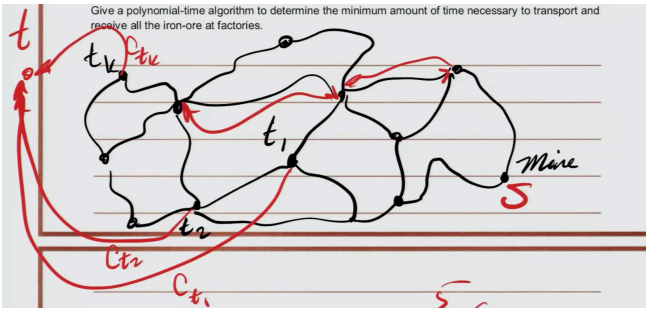
Case 1: If $f(e) < c_e$, f is also the maximum flow of G'.
Case 2: If $f(e) = c_e$, we need to take away one unit of flow from s to t by just finding the path from s to v and from u to t that contains at least one unit of flow, then decrease the flow f by one along the s-v-u-t path.
We just get a valid flow f' by the above decrease flow operation. Then we need to check $G_{f'}$ to see if there is a simple s-t path, if yes the bottleneck of this path must be 1, otherwise we would get a maximum flow greater



than f even after decreasing e by 1, that's impossible. Anyway, it's possible to find another way to push the flow from s to t.
Time Complexity: find a path from s to v and u to t takes $O(m)$ time, and decrease the flow f by one along the s-v-u-t path takes $O(m)$ time, construct the residual graph of f' takes $O(m)$ time, find a path from s to t in $G_{f'}$ takes $O(m)$, and potentially augment flow by one unit takes $O(m)$ time, thus the total time complexity is $O(m)$.

5.2 You are given a graph representing the road network of cities, with a list of k of its vertices $t_1, t_2, \ldots, t_k$ as factories and one vertex S as iron-ore mine. You are also given the Road Capacities for each road (edges) between cities, that's the amount of iron can be transported per minute. The amount of ore to be transported from the mine C and the amount of iron can be received per minute for each factory $t_1, \ldots, t_k$. Give a polynomial-time algorithm to compute the minimum amount of time necessary to transport and receive all the iron-ore at factories.

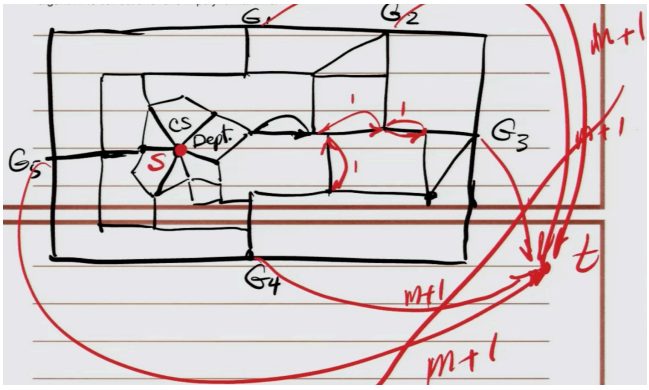
First, we need to bulid a flow network G=(V, E) with the given graph and capacities. Add a virtual sink vertex T with an edge from each factory to T **with capacity equal to the amount of iron can be received per minute of that factory**. Then **add two directed edges for each road** with the capacity of road.
Then we can use the Edmonds-Karp Algorithm to find the maximum flow f of G, then minimum amount of time necessary to transport and receive all the iron-ore at factories is C/v(f).



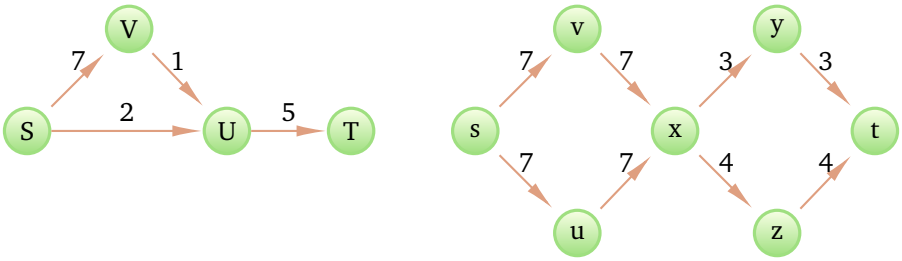
5.3 Think the USC campus as a graph $G = (V, E)$, n which the nodes are locations, and edges are pathways or corridors. One of the nodes is the burglar's starting point S, and several nodes $P \subset V$ are the escape USC gates. If the burglar reaches any one of gates, he is goen forever. Students and Staff can be placed to monitor the edges. Give an polynomial-time algorithm to compute the minimum number of students/stuff needed to ensure that the burglar cannot reach any escape points undetected.

First, we need to construct a flow network G=(V, E) with the given vertices and **two directed edges with capacity 1 for each road** because we only need one person to monitor one edge. Add a virtual sink vertex T with **a directed edge from each escape gate to T with capacity |E|+1** to ensure these edges are not in the minimum cut.

Then we can just run the Ford-Fulkerson Algorithm to find the maximum flow f of G in O(Cm) = O(nm) because all the edge capacity is one unit, then we can get the minimum cut in G_f, $C(A^*, B^*)$ is the minimum number of students/stuff needed.



5.4 Define a "Most Vital Edge" of a network as an edge whose deletion causes the largest decrease in the maximum s-t flow value. Let f be an arbitrary maximum s-t flow. The following is true or false?
(a) A most vital edge is an edge e with the maximum value of c_e;
(b) A most vital edge is an edge e with the maximum value of $f(e)$
(c) A most vital edge is an edge e with the maximum value of $f(e)$ among edges belonging to some minimum cut. (d) A edge that does not belong to any minimum cut cannot be a most vital edge.
(e) A network can contain only one most vital edge.



The left one is the counterexample for (a) (c) (d), the right one is the counterexample for (b) (e).

5.5 Define a "Critical Edge" of a network as an edge if it crosses every minimum cut in G.
Define a "Pseudo-Critical Edge" of a network as an edge if it cross at least one minimum cut in G.
Design an efficient algorithm to determine whether a given edge is Critical or Pseudo-Critical.

(a) Run Ford-Fulkerson Algorithm to find the maximum s-t flow f in G, denoted by f_1. **Increase the capacity of the given edge.** Then run Ford-Fulkerson Algorithm again to find the maximum s-t flow, denoted by f_2.
The given edge is a Critical Edge if and only if $v(f_1) < v(f_2)$, i.e. the capacity of "Critical Edge" is the most critical bottleneck of the maximum flow.
Consider the right graph in Q5.4, there is no "Critical Edge" beacuse only increase the capacity of edge (x, y) or (x, z) or (y, t) or (z, t) can't increase the maximum flow.

(b) Run Ford-Fulkerson Algorithm to find the maximum s-t flow f in G, denoted by f_1. **Decrease the capacity of the given edge.** Then run Ford-Fulkerson Algorithm again to find the maximum s-t flow, denoted by f_2.
The given edge is a Pseudo-Critical Edge if and only if $v(f_1) > v(f_2)$, i.e. the capacity of "Pseudo-Critical Edge" is one of the bottlenecks of the maximum flow.
Consider the right graph in Q5.4, both (x, y) (x, z) (y, t) (z, t) are "Pseudo-Critical Edge" because decrease the capacity of any of them will decrease the maximum flow.
Please note that the "**Critical Edge" does NOT necessarily exist**, but the "Pseudo-Critical Edge" must exist because there is always a minimum cut in the Max Flow Problem.
Given a minimum cut (A, B), if we strictly increase the capacity of every edge that crosses the (A,B) cut, then the value of a maximum flow of the network does NOT necessarily increase. There might be no "Critical Edge" in (A, B) cut so the flow value won't increase. See the below example where all three edges are minimum cut but increase one of them won't increase the maximum flow.