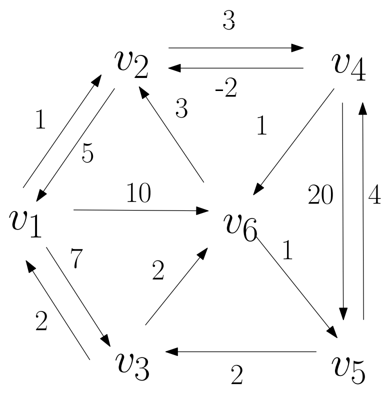
COMP 3711 – Design and Analysis of Algorithms
2019 Spring Semester – Written Assignment # 4
Distributed: April 26, 2019 – Updated: May 4, 2019
Due: May 10, 2019

Your solutions should contain (i) your name, (ii) your student ID #, and (iii) your email address
Notes:

- Follow the guidelines on doing your own work and avoiding plagiarism given on the class home page.
  In particular ***don't forget to acknowledge individuals who assisted you, or sources where you found solutions.*** Failure to do so will be considered plagiarism.
- Write clearly and follow the submission guidelines on the class web page. Use white, unwatermarked paper, e.g., no student society stationary.
  - If handwritten, solutions should be single sided, start a new page for every problem, be single column and leave space between consecutive lines and more space between paragraphs.
  - If typed, try to use an equation editor, e.g., latex , the equation editor in MS-WORD or whatever the equivalent is in whatever typesetting system you are using
- This assignment is due by 23:59 on May 10, 2019 in BOTH hard AND soft copy formats. A hard copy should be deposited in one of the two COMP3711 assignment collection boxes outside of room 4210. A soft copy for our records in PDF format should also be submitted via the online CASS system. See the Assignment 1 page in Canvas for information on how to submit online.
- The default base for logarithms will be 2, i.e., $\log n$ will mean $\log_2 n$. If another base is intended, it will be explicitly stated, e.g., $\log_3 n$.
- Correction. May 4, 2019. In the original distribution, the example on page 7 stated that $\{g, f\}$ is a *unique* smallest separating set. This was false. $\{g, a\}$ is also a smallest separating set. That error has now been fixed.

Problem 1: All-Pairs Shortest Path [25 pts]

Let $G = (V, E)$ be the directed weighted graph shown below with its associated weighted adjacency matrix.



$$W = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & \infty & \infty & 10 \\ \hline 5 & 0 & \infty & 3 & \infty & \infty \\ \hline 2 & \infty & 0 & \infty & \infty & 2 \\ \hline \infty & -2 & \infty & 0 & 20 & 1 \\ \hline \infty & \infty & 2 & 4 & 0 & \infty \\ \hline \infty & 3 & \infty & \infty & 1 & 0 \end{array}\right)$$

This problem asks you to solve the all-pairs shortest-path problem on this graph in two different ways.

1. Run the 2nd dynamic-programming solution from the slides on this graph. Recall that in the 2nd dynamic-programming solution

$$d_{ij}^{(2s)} = \min_{1 \le k \le n} \left\{ d_{ik}^{(s)} + d_{kj}^{(s)} \right\}$$

where $d_{ij}^{(1)} = w_{ij}$ so $D^{(1)} = W$ and, in general, $D^{(m)}$ is the matrix $\left[d_{ij}^{(m)} \right]$. For this problem you need to fill in the matrices $D^{(2)}$, $D^{(4)}$ and the final solution $D^{(8)} (= D^{(5)})$.

$$D^{(2)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & 11 & 9 \\ \hline 5 & 0 & 12 & 3 & 23 & 4 \\ \hline 2 & 3 & 0 & \infty & 3 & 2 \\ \hline 3 & -2 & 22 & 0 & 2 & 1 \\ \hline 4 & 2 & 2 & 4 & 0 & 4 \\ \hline 8 & 3 & 3 & 5 & 1 & 0 \end{array}\right) \qquad D^{(4)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & 6 & 5 \\ \hline 5 & 0 & 7 & 3 & 5 & 4 \\ \hline 2 & 3 & 0 & 6 & 3 & 2 \\ \hline 3 & -2 & 4 & 0 & 2 & 1 \\ \hline 4 & 2 & 2 & 4 & 0 & 4 \\ \hline 5 & 3 & 3 & 5 & 1 & 0 \end{array}\right)$$

$$D^{(8)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & 6 & 5 \\ \hline 5 & 0 & 7 & 3 & 5 & 4 \\ \hline 2 & 3 & 0 & 6 & 3 & 2 \\ \hline 3 & -2 & 4 & 0 & 2 & 1 \\ \hline 4 & 2 & 2 & 4 & 0 & 4 \\ \hline 5 & 3 & 3 & 5 & 1 & 0 \end{array}\right)$$

2. Now run the Floyd-Warshall algorithm on the graph. Recall that in the Floyd-Warshall algorithm

$$d_{ij}^{(k)} = \min\left\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$$

where $d_{ij}^{0} = w_{ij}$ so $D^{(0)} = W$, and $D^{(m)}$ is the matrix $\left[d_{ij}^{(m)}\right]$. For this problem you need to fill in the matrices $D^{(i)}$, where $i = 1, 2, 3, 4, 5, 6$ and $D^{(6)}$ is the final solution.

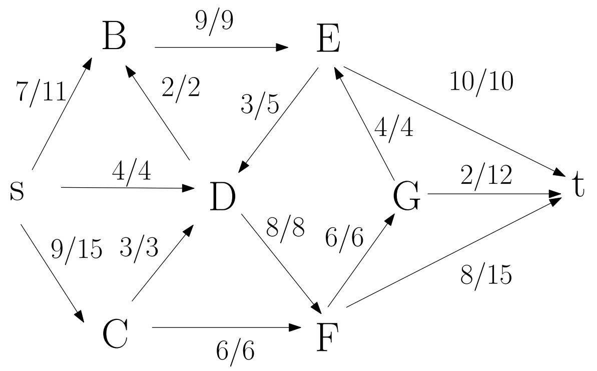
$$D^{(1)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & \infty & \infty & 10 \\ \hline 5 & 0 & 12 & 3 & \infty & 15 \\ \hline 2 & 3 & 0 & \infty & \infty & 2 \\ \hline \infty & -2 & \infty & 0 & 20 & 1 \\ \hline \infty & \infty & 2 & 4 & 0 & \infty \\ \hline \infty & 3 & \infty & \infty & 1 & 0 \end{array}\right) \qquad D^{(2)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & \infty & 10 \\ \hline 5 & 0 & 12 & 3 & \infty & 15 \\ \hline 2 & 3 & 0 & 6 & \infty & 2 \\ \hline 3 & -2 & 10 & 0 & 20 & 1 \\ \hline \infty & \infty & 2 & 4 & 0 & \infty \\ \hline 8 & 3 & 15 & 6 & 1 & 0 \end{array}\right)$$

$$D^{(3)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & \infty & 9 \\ \hline 5 & 0 & 12 & 3 & \infty & 14 \\ \hline 2 & 3 & 0 & 6 & \infty & 2 \\ \hline 3 & -2 & 10 & 0 & 20 & 1 \\ \hline 4 & 5 & 2 & 4 & 0 & 4 \\ \hline 8 & 3 & 15 & 6 & 1 & 0 \end{array}\right) \qquad D^{(4)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & 24 & 5 \\ \hline 5 & 0 & 12 & 3 & 23 & 4 \\ \hline 2 & 3 & 0 & 6 & 26 & 2 \\ \hline 3 & -2 & 10 & 0 & 20 & 1 \\ \hline 4 & 2 & 2 & 4 & 0 & 4 \\ \hline 8 & 3 & 15 & 6 & 1 & 0 \end{array}\right)$$

$$D^{(5)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & 24 & 5 \\ \hline 5 & 0 & 12 & 3 & 23 & 4 \\ \hline 2 & 3 & 0 & 6 & 26 & 2 \\ \hline 3 & -2 & 10 & 0 & 20 & 1 \\ \hline 4 & 2 & 2 & 4 & 0 & 4 \\ \hline 5 & 3 & 3 & 5 & 1 & 0 \end{array}\right) \qquad D^{(6)} = \left(\begin{array}{c|c|c|c|c|c} 0 & 1 & 7 & 4 & 6 & 5 \\ \hline 5 & 0 & 7 & 3 & 5 & 4 \\ \hline 2 & 3 & 0 & 6 & 3 & 2 \\ \hline 3 & -2 & 4 & 0 & 2 & 1 \\ \hline 4 & 2 & 2 & 4 & 0 & 4 \\ \hline 5 & 3 & 3 & 5 & 1 & 0 \end{array}\right)$$

Problem 2: The Ford-Fulkerson Algorithm [25 pts]

Consider the given flow in directed graph below. The labels on the edges are in the form f/c where c is the capacity of the edge and f is the flow value across the edge:



(a) Draw the residual network of the given graph and flow.

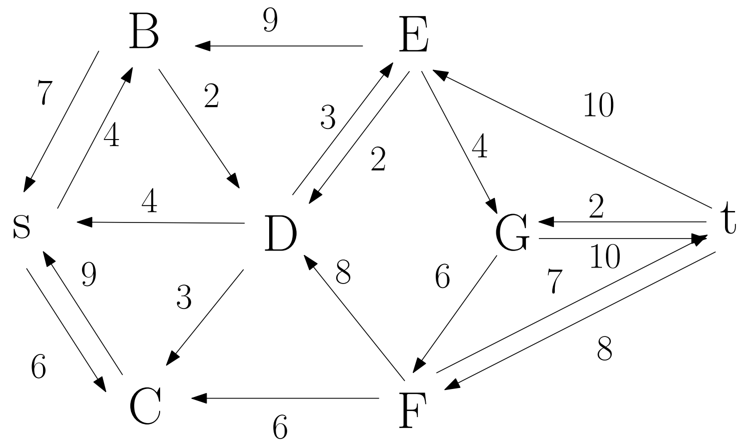
(b) Find an augmenting path p in the residual network and draw it. Explictly state the maximum flow $c_f(p)$ that can be pushed through p.

(c) Draw the new flow that results by adding the flow pushed through the augmenting path to the old flow. To answer this question you must redraw the graph. Relabel each edge in the format f'/c where f' is the new flow value.

(d) Is this new flow a maximum flow? PROVE your answer. (A proof that it is maximum would be a cut with capacity equal to the flow value. A proof that it isn't maximum would be a larger flow).

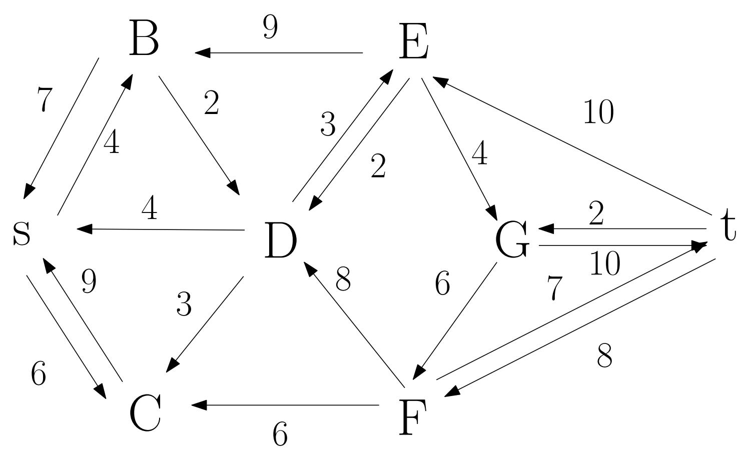
Solution:

a)

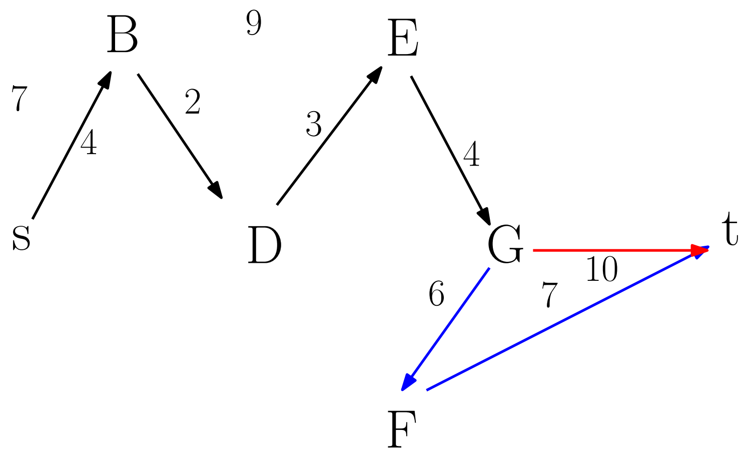


b)

The residual graph from the previous page is redrawn here

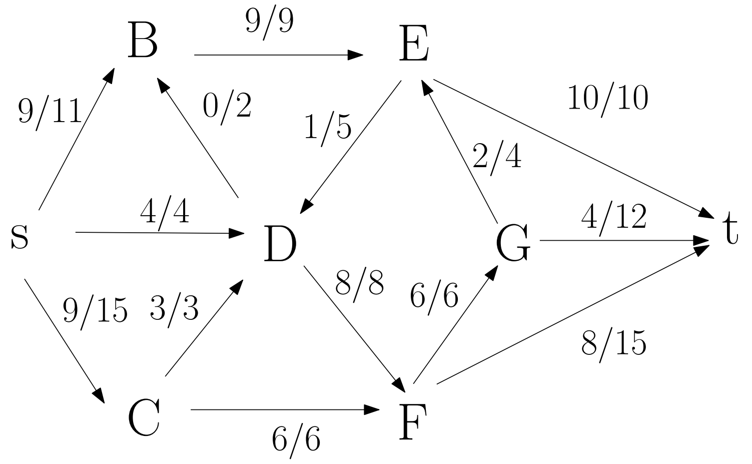


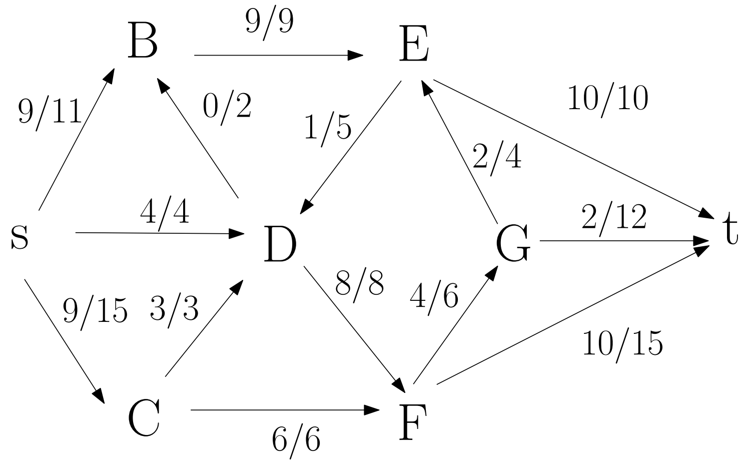
There are two possible augmenting paths:
$p_1 =< sBDEGt >$ *and*
$p_2 =< sBDEGFt >$.
They satisfy $c_f(p_1) = c_f(p_2) = 2$ *(the "2" comes from the edge BD).*



c)

The top flow results from adding $P-1$ to f; the bottom one from adding p_2.





d) Both flows created have flow value $= 22$.

To see that this is maximal let $S = \{s, B, C\}$. The cut S, T ($T = V - S$) has value

$$C(S,T) = c(sD) + c(BE) + c(CD) + c(CF) = 4 + 9 + 3 + 6 = 22.$$

Since this cut capacity equals the flow of 22, the max-flow min-cut theorem proves that the flow is maximal.

Problem 3: MST Redux [25 pts]

Let $G = (V, E)$ be a weighted undirected graph inputted as an adjacency list. Assume that all edges have distinct weights so the *Minimum Spanning Tree* T is unique and that you have already run a MST algorithm that has found and stored T.

Now arbitrarily *increase* the weight of any one edge $e = (u, v)$ in E. You may assume that, even after the increase, all edges still have distinct weights.

Let T' be the MST of the graph with the new weights. i.e, after replacing $w(u, v)$ with its new weight.

(a) Prove that after increasing one weight either $T' = T$ or that T and T' differ by at most one edge.

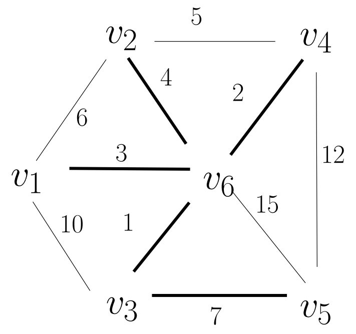
The only fact that you may use from class is the *Cut Lemma*. Any other fact must be proven from scratch.

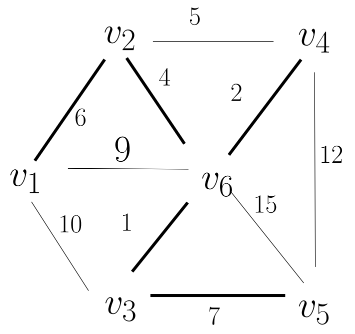
(b) Give an $O(|E|)$ time algorithm for finding T'. It is not necessary to give it in Pseudocode.

You may use any algorithm we taught in class or the tutorials as a subroutine. If you do use such an algorithm from the class/tutorials as a subroutine, you must explicitly state the name of that algorithm, its run time, why the input you are feeding into is of the correct type and the output of that algorithm.

It suffices to briefly describe what the algorithm does. Your description must explicitly justify why your algorithm is correct and why it is $O(|E|)$.

Example: below, in the graph on the left, the MST is the set of 5 heavy edges. If the weight of (v_1, v_6) *was increased from* 3 *to* 5.5*, the MST would stay the same. But, if the weight of* (v_1, v_6) *was increased from* 3 *to* 9 *the new MST would be* T'*, the 5 heavy edges on the right, i.e.,* T' *is* T *with the edge* (v_1, v_2) *being swapped for* (v_1, v_6)*.*





Hint: Consider the "cut lemma" and how it was used to prove the uniqueness of T *by showing that every edge in* T *must be in ALL MSTs. How can you modify* *that proof to show (a) above? For (b) consider WHY the cut lemma says that e is in T. If another edge needs to replace e, what property does that other edge need to satisfy?*

Solution:

MSTs: Let e' the edge whose weight is raised.
We call the original weights w and the new weights, after e' is raised, w'.
So $w'(e') > w(e')$ but for all $e \neq e'$, $w(e) = w'(e)$.
Also, let T be the MST for w and T' the MST for w'.

(a) *First recall the cut lemma from class and how we used it to prove that the MST is unique.*

The cut lemma was: **Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then every MST must contain e.**

In the proof that the MST T is unique, it was noted that
if e is any edge in T, then
(i) removing e from T cuts T into two unconnected parts, S_e and $V - S_e$ and
(ii) that e must be the min-cost edge connecting S_e and $V - S_e$, i.e., leaving S_e.
Thus, from the cut lemma, e must be in every MST.

In the proof of (a) there are now two possible cases
(1) $e' \notin T$:
Let $e \in T$ be any edge in the MST.

- *e was the min cost edge with exactly one endpoint in S_e for weights w so it is also the min cost edge with exactly one endpoint in S_e for w'*
  (Even if e' had exactly one endpoint in S_e, raising *its weight can't make it become the min-cost edge now if it wasn't previously the min cost edge).*
- *So, $e \in T'$.*

This is true for every $e \in T$ so $T \subseteq T'$.
Since $|T'| = |T| = |V| - 1$, this means $T' = T$.

(2) $e' \in T$.
In this case let $\bar{T} = T - \{e'\}$.
Let $e \in \bar{T}$ be any other edge in the MST T.
Then, after increasing the weight of e', e still is the smallest edge leaving S_e (since raising *the weight of e' can't change that fact). So e is still in every MST for the new weights. So $e \in T'$. This is true for every $e \in \bar{T}$ so $\bar{T} \subset T'$.*
Since $|\bar{T}| = |V| - 2$ and $|T'| = |V| - 1$ this automatically proves that T and T' can differ by at most one edge.

(b) *From part (a) we know that if $e' \notin T$ then $T' = T$, so return T.*

If $e' \in T$ then $\bar{T} = T' - \{e'\} \subset T'$ so we only need to find the one missing edge from T'.

Let $e' = (u', v')$. As noted in (a), removing e' from T splits G into two connected subgraphs. One contains one contains u' and the other containins v'. We will call these components $S_{u'}$ and $S_{v'}$. By construction, $\bar{T}$ contains no edge with exactly one endpoint in $S_{u'}$.

Let $\bar{e}$ be the edge with the smallest weight in w' with exactly one endpoint in $S_{u'}$, i.e., one endpoint in $S_{u'}$ and the other in $S_{v'}$ (Note that it is possible that $\bar{e} = e'$).

By the cut lemma, $\bar{e} \in T'$ so $T' = \bar{T} \cup \{\bar{e}\}$. To finish our construction we only need to find $\bar{e}$.

- *We are given a set of edges $\bar{T} = T - \{e'\}$ where $e' = (u', v')$.*
- *We are also given that the edges in $\bar{T}$ form two connected components; $S_{u'}$ containing u' and $S_{v'}$ containing v'.*
- *The problem is to find the min-cost edge with one endpoint in $S_{u'}$ and the other endpoint in $S_{v'}$.*
- *This can be done by first using BFS to find the two connected components, labelling the vertices in each to denote which component they belong to.*
- *Then run through all of the edges in the graph, finding the min-cost one with one vetex in $S_{u'}$ and the other in $S_{v'}$*

Here is a simple algorithm for doing that.

```
1. Start with $T$.
2. If $e' \notin T$     % Then $T' = T$
3.      Return($T$)
   else    % $T' = \bar{T} \cup \{\bar{e}\}$
4.      Set $\bar{T} = T - \{e' = (u', v')\}$
5.      Do a BFS in $\bar{T}$ starting from $u'$, labelling all the vertices
          found with a "1". These are $S_{u'}$.
6.      Do a BFS in $\bar{T}$ starting from $v'$, labelling all the vertices
          found with a "2". These are $S_{v'}$.
7.     % Find $\bar{e}$
        Run through all of the edges in $(u, v) \in E$ once
        Let $\bar{e}$ be minimum $w'$ weight edge that has
          one side labelled 1 and the side other labelled 2.
8.     Set $T' = \bar{T} \cup \{\bar{e}\}$. Return($T'$)
```

The correctness follows from the discussion above.

For the running time, note that

Lines (1),(2),(3),(4) require $O(|V|)$ time

The BFS in Lines (5) and (6) (3) takes $O(|E|)$ time (actually $O(|V|) = O(|E)|)$ because that's the running time of BFS.

Line (7) looks at every edge once, e.g., by running through the adjacency lists, so it takes $O(|E|)$ time.

Line (8) takes $O(|V|)$ time.

So, in total, the algorithm uses $O(|E|)$ time.

Marking Note:

(a) For full points it was necessary to explain why the OTHER edges in the MST are unaffected by the weight increase.

More specifically, let e be the edge whose weight was changed and T the MST. It was not enough to explain when e would be removed from T. You also needed to explicitly PROVE that all edges whose weights did NOT change would REMAIN in the MST. If you missed that point, marks were deducted.

(b) The solution to this problem could NOT use Kruskal's algorithm or a version of it. More explicitly, it could not use the Union-Find data structure as a subroutine. This is because U-F is NOT $O(1)$ time per operation. Thus, any solution using U-F would NOT be $O(E)$ time. Solutions that did use U-F had points deducted.

Part (b) also needed to explicitly explain HOW you found the two components. More explicitly, if you just said to look at the edges between the two components and take the minimum one, points were deducted. You needed to explain HOW you found the components, e.g., using BFS or DFS. If you said use BFS or DFS but did not provide any explanation at all as to HOW to use BFS or DFS to find components you also had some (but many fewer) points deducted.

Problem 4: Max-Flow and Graph Reliability [25 pts]

Let $G = (V, E)$ be a directed graph with two specified vertices $s, t \in V$. Assume that G contains at least one s-t path.

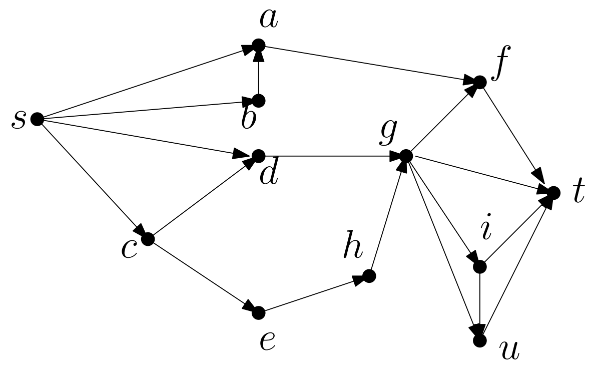
A subset of edges $E' \subseteq E$ is a *separating edge set* if, after removing E' from G, no s-t path exists.

A subset $V' \subseteq V - \{s, t\}$ of vertices is a *separating vertex set* if, after removing V' and all edges connected to V' from G, no s-t path exists. If $(s, t) \notin E$, a separating vertex set always exists.

E' is a *smallest separating edge set* if it contains the smallest number of edges among all separating edge sets. V' is a *smallest separating vertex set* if it contains the smallest number of vertices among all separating vertex sets.

Note that a smallest separating edge set E' might not be unique. A smallest separating vertex set V' also might not be unique.

In the graph below $\{(a, f), (d, g), (e, h)\}$ is a smallest separating edge set, but there are others as well. $\{g, f\}$ and $\{g, a\}$ are the only smallest separating vertex sets.



Smallest separating sets are indicators of failure points in a network and are therefore used in studies of network reliability.

In what follows you may use any facts or algorithms taught in class as long as you explicitly reference them (which means including their statement and where in the class or tutorial notes you found them).

(a) Give an $O(|E|^2)$ time algorithm for finding a smallest separating edge set for G.

(b) Assume $(s, t) \notin E$. Give an $O(|V||E|)$ time algorithm for finding a smallest separating vertex set for G.

For both (a) and (b) describe your algorithm clearly (code is not necessary) and prove its correctness and running time. Your proof of correctness must contain a section that *EXPLICITLY* justifies why your output is a smallest separating edge or vertex set.

Hints: For (a), consider the algorithm for finding the maximum number of edge disjoint s-t paths taught in class. How can you modify that to solve this problem?

For (b) Modify your graph as follows. For every vertex $v \in V$,
(i) remove v *and create two new vertices* v_ℓ, v_r,
(ii) create new edge (v_ℓ, v_r).
(iii) Replace every edge $(u, w) \in E$ *with edge* (u_r, w_ℓ).

Note that if P *is an* $s - t$ *path in* G *that passes through vertex* v, *the corresponding path in the modified graph must pass through edge* (v_ℓ, v_r). *Now appropriately modify the algorithm for part (a).*

Solution:

The concept of "separation" is actually called "connectivity" in the literature, e.g., "edge connectivity" and "vertex connectivity". Set

$$\begin{aligned} EC &= \text{size of a minimum separating edge set,} \\ VC &= \text{size of a minimum separating vertex set.} \end{aligned}$$

For both parts we wil need the following notation and facts.

Let $S \subset V$. Recall that (S,T) is a cut *if if $T = V - S$ $s \in S$ and $t \in T$.*

For a cut (S,T) define

$$E(S,T) = \{(u,v)\ u \in S\, v \in T\}.$$

$$C(S,T) = \sum_{(u,v)\in E(S,T)} c(u,v)$$

is the capacity between those edges. In both (a) and (b) we will be dealing with situations in which all of the edge from S to T have capacity 1. Note

$$\text{If } \forall u \in S, \text{ and } v \in T,\ c(u,v) = 1, \text{ then } |E(S,T)| = C(S,T). \tag{1}$$

Every path from s to t must include at least one edge in $E(S,T)$ so

$$\text{If } (S,T) \text{ is a cut, then } E(S,T) \text{ is a separating edge set.} \tag{2}$$

(a) The algorithm is simple

1. *Give every edge a capacity 1.*
2. *Run the Ford-Fulkerson Max-Flow Algorithm*
   Let f^* be the max flow found.
3. *Let (S', T') be the Min-Cut generated by the algorithm*
   Recall that S' is the set of vertices that are reachable from s in the residual graph G_{f^*}.
4. *Return $E(S', T')$, i.e., the edges crossing the cut.*

From Equation (2), every cut S,T defines an edge separating set $E(S, V-S)$ so,

$$\text{for EVERY cut } (S,T), \quad EC \le |E(S,T)|.$$

In particular

$$EC \le |E(S',T')| = C(S',T') = |f^*|. \tag{3}$$

Thus, to prove correctness of the algorithm we only need to prove that $|E(S',T')| \le EC$. This would imply that

$$|E(S',T')| = EC,$$

so $E(S',T')$ is a smallest edge separating set.

We provide TWO different proofs.

Proof 1 that $|E(S', T')| \leq EC$:

- *Let $\bar{E}$ be a minimum separating edge set and $\bar{G}$ be G with $\bar{E}$ removed. Define $\bar{S}$ to be all of the vertices reachable from s in $\bar{G}$. Since t is NOT reachable from s in $\bar{G}$, $(\bar{S}, \bar{T})$ with $\bar{T} = V - \bar{S}$ is a cut.*
- *If $(u, v) \in E(\bar{S}, \bar{T})$ then by definition $(u, v) \notin \bar{G}$. Thus*

  $$\forall (u, v) \in E(\bar{S}, \bar{T}),\ (u, v) \in \bar{E} \quad \Rightarrow \quad E(\bar{S}, \bar{T}) \subseteq \bar{E}.$$

  and thus
  $$|E(\bar{S}, \bar{T})| \leq |\bar{E}| = EC.$$
- *(S', T') is a* MINIMUM *cut so*

  $$|E(S', T')| = C(S', T')| \leq C(\bar{S}, \bar{T}) = |E(\bar{S}, \bar{T})| \leq EC$$

Proof 1 is finished. For completeness, we note that we have actually proven a bit more. That $E(S', T')| = EC = |f^*|$. This level of proof was not needed, though.

Proof 2 that $|E(S', T')| \leq EC$:

Let DP be the maximum number of edge disjoint $s - t$ paths. We showed in class that $DP = |f^*|$ so

$$DP = |f^*| = C(S', T') = |E(S', T')|.$$

Let $\bar{E}$ be a smallest separating edge set. $|\bar{E}| = EC$.

Now let $P_1, P_2, \ldots, P_{DP}$ be a maximum size set of edge disjoint $s - t$ paths. Each path P_i must contain at least one edge in $\bar{E}$ (otherwise $\bar{E}$ is not a separating set). The fact that the paths are edge disjoint immediately implies

$$DP \leq |\bar{E}| = EC.$$

This implies
$$|E(S', T')| = DP \leq EC$$
completing the second proof.

In the algorithm, steps 1,3,4 can all be implemented in $O(|E|)$ time. Step 1, trivially; Step 3, by running BFS from from s. In step 3, you can also label each vertex as being in S' or not. After doing that, Step 4 can be easily implemented in $O(|E|)$ time. Note that $|f^*| = O(|V|) = O(|E)$ so

Step 2 requires $O(|f^*||E|) = O(EC \cdot |E|) = O(|V||E|) = O(|E|^2)$ time.

Marking Note:

- **Some students gave the following "solution":**
  1. Use the algorithm from class to find $|f^*|$ edge disjoint paths.
  2. Then create an edge separating set by taking one edge from each path.

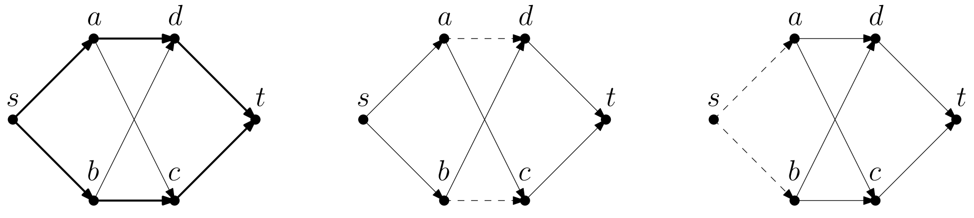
  This is wrong. This algorithm does not necessarily give you a separating edge set. While it IS possible to construct an edge separating set by taking one edge from each path not every edge will work.

  As an example, consider the graph depicted below.

  (A) shows two edge disjoint paths $< sadt >$ and $< sbct >$.

  (B) illustrates that the two edges $\{(a,d),(b,c)\}$ are NOT an separating edge set, even though the set contains one edge from each path. So, the algorithm of simply choosing one edge from each path does not work.

  (C) illustrates that the choice of $\{(s,a),(s,b)\}$ does yield an separating edge set with one edge on each path.



(A) (B) (C)

  If you wrote "choose one edge from each path" but did not specify HOW to choose the edge and why that leads to an edge separating set your answer would be wrong.

  You need to specify WHICH edge you chose from each path and what that works.

- **Some students gave the correct algorithm but did not prove correctness.**
  It is not hard to see that the $E(S',T')$ is the smallest edge separating set THAT IS DEFINED BY A CUT (by using the max-flow min-cut theorem).
  But, there are a lot of edge separating sets that are NOT cuts. For a correct answer you needed to explain why a smallest edge separating set is always defined by some cut (which is

what our two proofs were really doing). Only then is the max-flow min-cut theorem applicable.

– The instructions were very clear that you could only use facts that we taught. Anything else needed to be fully justified.

  So, for example, if you got a proof off of the internet that used Menger's theorem you would need to explicitly define Menger's theorem and prove it (or show that we had proven it already).

(b) Let G' be the graph created after making the modifications described and also removing the edges (s_ℓ, s_r) and (t_ℓ, t_r).

$G' = (E', V')$ contains two different types of edges:

$$\begin{aligned} E_1 &= \{(v_\ell, v_r) \,:\, v \in V - \{s,t\}\}, \\ E_2 &= \{(u_r, v_\ell) \,:\, (u,v) \in E\}. \end{aligned}$$

A $s-t$ path in the original graph then corresponds to a $s_r - t_\ell$ path in G'.

For $v \in V$, define its corresponding *corresponding edge* in E_1 as (v_ℓ, v_r) and vice-versa.

Removing vertex v from the original graph corresponds to removing its corresponding edge (v_ℓ, v_r) from E_1.

So, a separating vertex set in G corresponds to a separating edge set in G' in which all edges are in E_1 and vice-versa.

A first attempt at solving part (b) would be to just run the algorithm of part (a) on the new graph G' and then returning the min-cut. If all of the edges found were in E_1 this would solve the problem, since the set of these edges would correspond to a separating vertex set in the original G. The difficulty is that the cut might include edges from E_2.

The trick is to assign the edges in E_1 and E_2 different capacities. This will force the min-cut to only include E_1 edges.

Lemma: Assign all edges in E_1 capacity 1 and all edges in E_2 capacity 2. Run the Ford-Fulkerson Max-Flow Algorithm on G' using s_r as source and t_ℓ as sink. Let S', T' be the Min-Cut generated by the algorithm. Then

$$E(S', T') \subseteq E_1. \tag{4}$$

Proof: Recall that S' is the set of vertices in G' that are reachable from s_r in the residual graph G_f. Also that the max-flow f^* found by Ford-Fulkerson is integral.

- If $(u,v) \in E_1$ then, by the integrality, $f^*(u,v) = 1$ or $f^*(u,v) = 0$.
- If $(u,v) \in E_2$ then, since the flow through (u_r, u_ℓ) and through (v_r, v_ℓ) is either 0 or 1, conservation of flow requires $f^*(u,v)$ is 0 or 1. (note that if $u = s$ or $v = t$ those edges might not both exist. But since $(s,t) \notin E$, at least one of them must exist, so the statement is correct.)
- The proof of the Max-flow Min-Cut theorem showed that if an edge $(u,v) \in E(V', T')$, then $f^*(u,v) = c(u,v)$. But, if $(u,v) \in E_2$, $c(u,v) = 2$ while from the argument above, $f^*(u,v) \leq 1$. So if $(u.v) \in E(S', T')$ then $(u,v) \in E_1$, proving Equation (4).

The proof is not complete. Note that the value '2' was arbitrary. The lemma would remain correct if edges in E_2 had any capacity ≥ 2.

This now leads to a simple algorithm to solve (b):

1. Give every edge in E_1 a capacity 1 and every edge in E_2 a capacity 2.
2. Run the Ford-Fulkerson Max-Flow Algorithm on G' using s_r as source and t_ℓ as sink. Let f^* be the max flow found.
3. Let S', T' be the Min-Cut generated by the algorithm
   Recall that S' is the set of vertices in G' that are reachable from s_r in the residual graph G_f.
4. Return the set of vertices $V' \subset V$ corresponding to the edges in $E(S', T')$.

Note that Equation (4) from the Lemma implies that $E(S', T')$ is separating edge set in G' in which all edges are in E_1 so V' is in line 4 is well defined and is a separating vertex set in G.

Since for $(u, v) \in E_1$, $c(u, v) = 1$, applying Equation (1) shows

$$VC \leq |V'| = |E(S', T')| = C(S', T') = |f^*|. \tag{5}$$

Now let $\bar{V}$ be a smallest separating vertex set and $\bar{E} = \{(v_\ell, v_r) : v \in \bar{V}\}$. Note that $|\bar{E}| = |\bar{V}| = VC$.

Let $\bar{G}$ be G' with the edges in $\bar{E}$ removed and define $\bar{S}$ to be the set of all vertices in G' reachable from s_r in $\bar{G}$. Set $\bar{T} = V - \bar{S}$.

Similar to the proof in part (a) we now note that if $e \in E(\bar{S}, \bar{T})$ then $e \notin \bar{G}$ so $e \in \bar{E}$. In particular, this forces

$$E(\bar{S}, \bar{T}) \subseteq \bar{E}.$$

Since every $e \in \bar{E}$ had $c(e) = 1$ and by the max-flow min-cut theorem $C(S', T') \leq C(\bar{S}, \bar{T})$, combining with Equation (5) yields

$$|E(S', T')| = C(S', T') \leq C(\bar{S}, \bar{T}) = |E(\bar{S}, \bar{T})| \leq |\bar{E}| = VC.$$

We have just shown that $|E(S', T')| \leq VC$ and that $VC \leq |E(S', T')|$ so V' is a separating vertex set satisfying

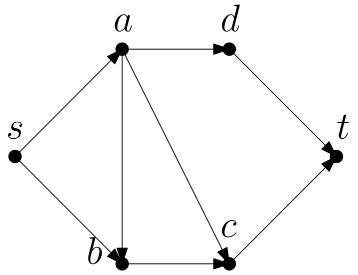
$$|V'| = |E(\bar{S}, \bar{T})| = VC,$$

i.e., V' is a minimum size separating vertex set.

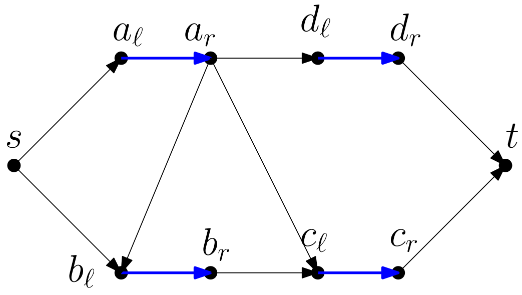
Everything in the algorithm runs in $O(|E|)$ time except for the max-flow algorithm which runs in time $O(|f^*||E|) = O(VC|E|) = O(|V||E|)$ so we are done.

Marking Note:

Consider the example below. The left side is the original graph G and the right side is the new G'. The thick blue edges in G' correspond to vertices in G. A separating vertex set in G corresponds to a separating blue edge set in G'.



(A)



(A)

- **A solution that said to simply run part (a) on G' would be wrong. In the example above would return the two edges $(s, a_\ell), (s, b_\ell)$ and neither of those edges is blue.**

  Note that giving the black edges capacity 2 and blue edges capacity 1 would find the min cut $(a_\ell, a_r), b_\ell, b_r)$ which are both blue and would correspond to the smallest separating vertex set $\}a, b\}$.
- **A correct solution would need to PROVE that a minimum cut (when running FF) in G' only contains blue edges. Without that proof, the answer is incomplete**
- **Note that even after doing the above it was ALSO still necessary to prove that there is a minimum separating blue edge set that is a CUT (otherwise it would be possible that FF would not give you the smallest solution).**