ICS 311 Fall 2020, Problem Set 10, Topics 18-20 & 22

Firstname Lastname <uhumail@hawaii.edu> Section 1

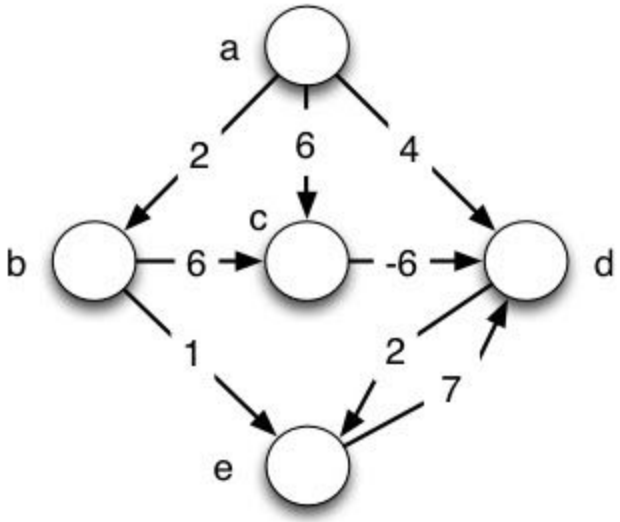
Due by midnight Tuesday December 8th. 40 points.

This is a combined problem set, replacing the former problem sets 10 and 11 so that TAs can catch up on grading. The denominator of points for the course will be adjusted accordingly. In order to combine assignments we have removed some of the algorithm tracing problems. You have already traced Dikjstra's algorithm in class, and the optional problem from the same class suggests that you trace Johnson's algorithm, which includes Bellman-Ford and Dijkstra. If you do this and want to see the solution, ask. The remaining algorithms to trace are Floyd-Warshall and Edmunds-Karp.

---

1. (5 pts) Floyd-Warshall

Recall that CLRS initially presents Floyd-Warshall as constructing a series of matrices $D^{(k)}$, and we subsequently showed it can just modify one matrix D. The $D^{(k)}$ are the states of D after processing each k in the main loop.



Blow we show the matrix $D^{(0)}$ for the above graph (which is different from the one used in class). Since Floyd-Warshall assumes that vertices are indexed by integers, we map them as follows: a is vertex 1, b is vertex 2, etc. **Run Floyd-Warshall, showing the matrix $D^{(k)}$ for each value of *k*.** The final matrix should have values for all start vertices.

$D^{(0)}$

	1 (a)	2 (b)	3 (c)	4 (d)	5 (e)
1 (a)	0	2	6	4	∞
2 (b)	∞	0	6	∞	1
3 (c)	∞	∞	0	-6	∞
4 (d)	∞	∞	∞	0	2
5 (e)	∞	∞	∞	7	0

$D^{(1)}$

	1 (a)	2 (b)	3 (c)	4 (d)	5 (e)
1 (a)					
2 (b)					
3 (c)					
4 (d)					
5 (e)					

$D^{(2)}$

	1 (a)	2 (b)	3 (c)	4 (d)	5 (e)
1 (a)					
2 (b)					
3 (c)					
4 (d)					
5 (e)					

$D^{(3)}$

	1 (a)	2 (b)	3 (c)	4 (d)	5 (e)
1 (a)					
2 (b)					
3 (c)					
4 (d)					
5 (e)					

$D^{(4)}$

	1 (a)	2 (b)	3 (c)	4 (d)	5 (e)
1 (a)					
2 (b)					
3 (c)					
4 (d)					
5 (e)					

$D^{(5)}$

	1 (a)	2 (b)	3 (c)	4 (d)	5 (e)
1 (a)					
2 (b)					
3 (c)					
4 (d)					
5 (e)					

2. (10 pts) Parallel Floyd-Warshall

In this problem we will parallelize the Floyd-Warshall algorithm. Use the following pseudocode as your starting point. (The CLRS version had `D = W` for line 2; we replace this with lines 2-5 to make the loops needed for array assignment explicit.)

```
Floyd-Warshall(W)
1  n = W.rows
2  create n x n array D
3  for i = 1 to n
4      for j = 1 to n
5          D[i,j] = W[i,j]
6  for k = 1 to n
7      for i = 1 to n
8          for j = 1 to n
9              D[i,j] = min(D[i,j], D[i,k] + D[k,j])
10 return D
```

(a) Design a parallel version of this algorithm using spawn, sync, and/or parallel for as appropriate. (Copy and modify the pseudocode.) Think carefully about what can be parallelized and what can't, and **explain your choices**.

(b) Analyze the asymptotic runtime of your algorithm in terms of its work, span, and parallelism (all three).

3. (5 pts) An Alternative Proof of Correctness

If we can show that Dijkstra's algorithm relaxes the edges of every shortest path in a directed graph in the order in which they appear on the path, then the path relaxation property applies to every vertex reachable from the source, and we have an alternative proof that Dijkstra's algorithm is correct. Either show that Dijkstra's algorithm must relax the edges of every shortest path in a directed graph in the order in which they appear

on the path, or provide a counter-example directed graph in which the edges of a shortest path could be relaxed out of order and explain how that happens.

---

4. (10 pts) Vertex Capacities

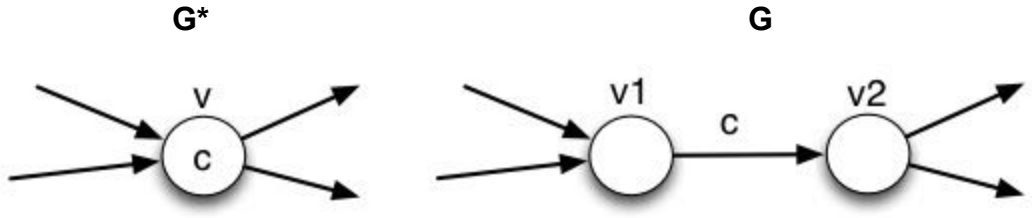
This is the challenge problem you started in class, but you will complete the proof.

Suppose that, in addition to edge capacities, a flow network G^* has **vertex capacities**. We will extend the capacity function c to work on vertices as well as edges: $c^*(u,v)$ gives the usual edge capacity and $c^*(v)$ gives the amount of flow that can pass through v. But we don't want to write a new algorithm.

We want to transform a flow network $G^*=(V^*,E^*)$ with c^* that defines both edge and vertex capacities into an equivalent ordinary flow network $G=(V,E)$ and c that is defined only on edges (without vertex capacities) such that a maximum flow in G has the same value as a maximum flow in G^*. Then we can run the algorithms we already have.

In class we identified this transformation: given $G^*=(V^*,E^*)$ and c^*, compute $G=(V,E)$ and c as follows:

> For each vertex $v \in G^*$ with capacity c, make vertices v_1 and $v_2 \in G$ and a new edge (v_1, v_2) with capacity c (the same as v). Then rewire the graph so that all edges into v in G^* now go to v_1 in G, and all edges out of v in G^* now come out of v_2 in G (that is, replace (u, v) with (u, v_1) and (v, u) with (v_2, u)). Finally, make s_1 and t_2 be the new source and target vertices of G.



<u>Prove that this solution is correct; that is, a solution computed on G will be a correct solution for G^*, as follows</u> (your proofs should use formal notation and algebra to be precise, not just English):

(a) Given a flow *f* computed on G, **describe how you would construct a flow *f** on G***.

(b) Show that when flows computed on G are converted to flows in G*, **edge capacities *c**(u,v) in G* are respected**.

(c) Show that when flows computed on G are converted to flows in G*, **vertex capacities *c**(*v*) in G* are respected**.

(d) Show that when flows computed on G are converted to flows in G*, **conservation constraints are respected**.

(e) Show flow equality |f| = |f*|: a flow in G has the same value as a flow in G*.

(f) Finally, building on the above, show that the translated flow f* is maximal. Hint: Proof by contradiction.

---

5. (10 pts) Tracing Edmund-Karp

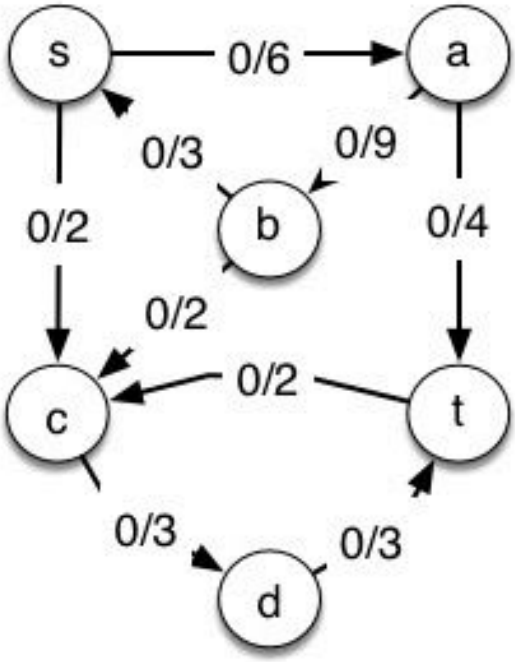
(a) Run Edmunds-Karp on the following graph with **s** as the source and **t** as the sink.

- Draw the flow graphs on the left hand side of the page with current flow indicated, and the residual graphs on the right hand side of the page.
- Mark the shortest augmenting path in the residual graph with thicker lines, and use it to update flow in the flow graph.
- Repeat, redrawing both updated graphs on a new line, until you can't find an augmenting path.

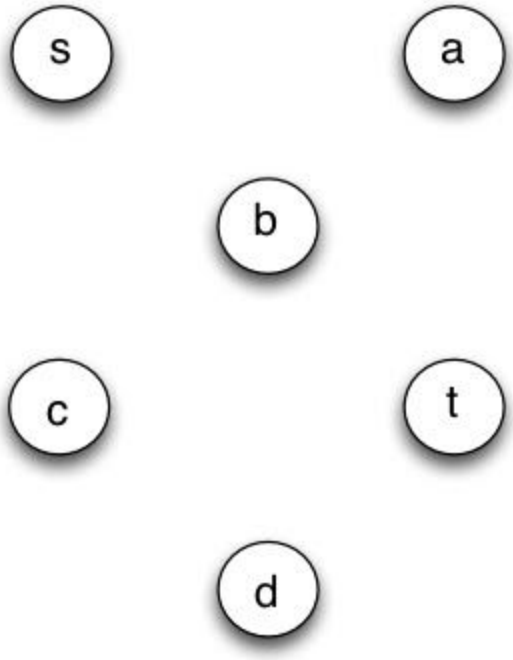
(b) When you can't update the graph any more, write the value of the flow |*f*| that was achieved, and draw a line in the final graph showing a min cut that corresponds to this max flow.

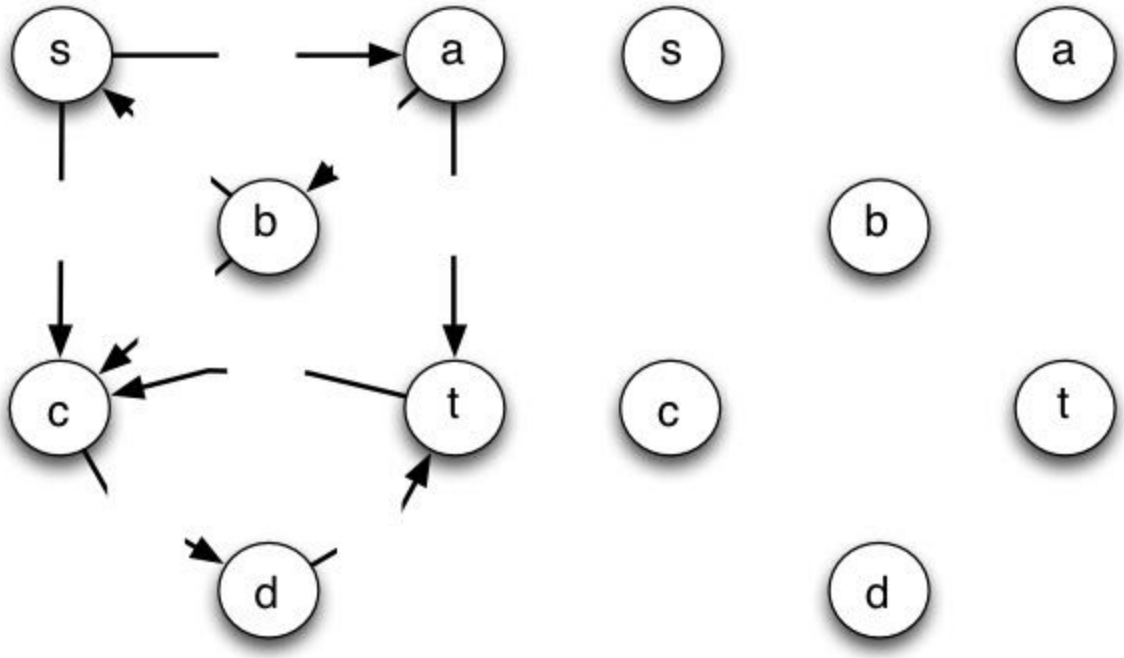
The graph and a printable template are on the next page for those doing this on paper. Google Drawing, Omnigraffle, and Visio templates are provided: follow this layout.

Flow graphs:



Residual graphs:




s
a
b
c
t
d
s
a
b
c
t
d