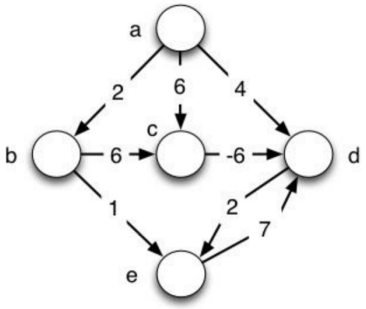
Problem 1: Floyd Warshall

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 via a

	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^2

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

$\mathbf{D}^3$

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

$\mathbf{D}^4$

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

$\mathbf{D}^5$

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

Problem 2: Parallel Floyd-Warshall

(a) **Design a parallel version of this algorithm using spawn, sync, and/or parallel for as appropriate.**

Algorithm 1 Parallel-Floyd-Warshall(W)

```
1: n = W.rows
2: create n × n arrays
3: parallel for i = 1 to n
4:     parallel for j = 1 to n
5:         D[i,j] = W[i,j]
6: for k = 1 to n
7:     parallel for i = 1 to n
8:         parallel for j = 1 to n
9:             D[i,j] = min(D[i, j] + D[i, k] + D[k, j])
10: return D
```

Explanation

- Modification for Line 3-4 and 7-8
  Parallel can be done to index i and j as this is where the memory location accessed inside each thread will be partition by each step of i and j. Line 5 accesses a unique location for each (i,j) and line 9 with a fixed k accesses a unique (i,j).
- No Modification for Line 6
  At line 6 unlike the previous nested for loops, we do not parallel it because this is where the bottom-up dynamic programming happens. Since bottom up method solves the smaller problems before the larger problems, there is no need to modify term k.

(b) **Analyze the asymptotic runtime of your algorithm in terms of its work, span, and parallelism**
The overall runtime will be because of the following reasons

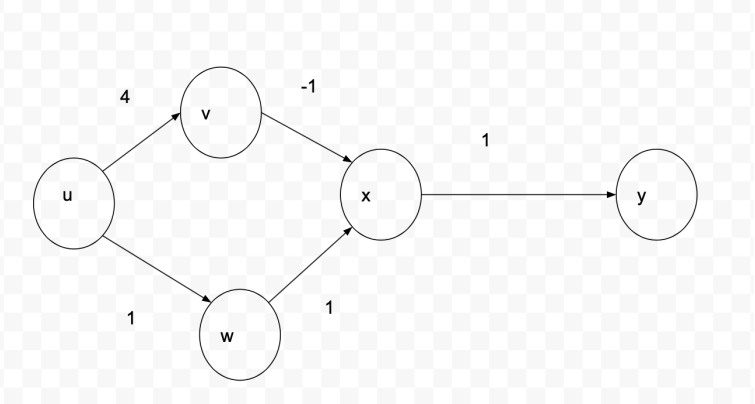
- From line 6 to line 8 it will take $\Theta(n^3)$
- For span, we use the divide and conquer method found in the CLRS method it talks about for the implementation in parallel for, which has a recursion depth of $\Theta(lgn)$. In the second set of the nested loop the overall runtime is $\Theta(n \lg n)$.
- $T_1 = \Theta(n^3)$
- $T_\infty = n \lg n$
- Then $\frac{n^3}{n \lg n} = \frac{n^2}{\lg n}$

Therefore, we have $\Theta(\frac{(n^2}{\lg n})$

Problem 3: 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.

To prove this we will provide a counter example.



Edges $(u, w), (w, x)$ and (x, y) are relaxed first.
However, (x,y) is relaxed out of order as the shortest path can be taken via v.

Problem 4: Vertex Capacities

(a) **Given a flow f computed on G, describe how you would construct a flow f* on G***
The flow f^* can be described by $\forall(u_2, v_1) \in$ G that were used to construct $(u, v) \in G^*$ then

$$f^*(u, v) = f(u_2, v_1)$$

(b) **Show that when flows computed on G are converted to flows in G*, edge capacities c*(u,v) in G* are respected.**
We know that every edge found in G has a corresponding edge in G^*. Then it follows that there is a corresponding capacity of the following:

$$c^*(u, v) = c(u_2, v_1)$$

In part (a) it was stated that $f^*(u, v) = f(u_2, v_1)$ and that from the flow and capacity rule that

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

then for each edge such that

$$f(u_2, v_1) \leq c(u_2, v_1)$$

then it implies that

$$f^*(u, v) \leq c^*(u, v)$$

Hence the edge capacities in G^* is respected.

(c) **Show that when flows computed on G are converted to flows in G*, vertex capacities c^*(v) in G* are respected**
Edge capacities in G are followed by f such that $f(v_1, v_2) \leq c(v_1, v_2)$. However, $f(v_1, v_2)$ is the flow $f^*(v)$ that is assigned to vertex v, and therefore,

$$c(v_1, v_2) = c^*(v)$$

then

$$f^*(v) \leq c^*(v)$$

(d) **Show that when flows computed on G are converted to flows in G*, conservation constraints are respected.**
From the CLRS and lecture all incoming and outgoing flows are equal then let us assume that f holds the conservation constraints at all vertices in G.
For each vertex v $\in G^* - \{s, t\}$ consider $v_1$and$v_2 \in G$.
From the conservation of flow f $\in$ G, the $\sum f(u, v_1)$(which is $\in G = \sum f(v_1, x)$. However, from the construction of the graph there is only one $x = v_2$ so that the sum $\forall$ flows in f(u, v_1) in G is equal to $f(v_1, v_2)$.
Furthermore, from the conservation flow rule then $\sum f(y, v_2)$ in G is equal $\sum f(v_2, w)$. Similarly in the previous paragraph, then there is only one y = v_1 so the flow $f(v_1, v_2)$ is equal to $\sum f(v_2, w)$ in G.
Hence, the sum of all flows $f(u_1, v) \in G$ is equal to the sum of all the flows in $f(v_2, w) \in$

G. These are the flows that f^* assigns to the edges incoming to v and outgoing from v $\in G^*$. Therefore, the sum of all flows $f^*(u, v) \in G$ is equal to the sum of all flows $f^*(v, w) \in G^*$. Therefore, the conservation property in G^* is held.

(e) **Show flow equality $|f| = |f^*|$: a flow in G has the same value as a flow in G^***
From our previous proof, the flow in G is defined as the sum of flows on edges coming out of source s_1 which is

$$|f| = \sum_{v \in V} f(s_1, v)$$

However, this is same to the flow on the single edge (s_1, s_2) since only one goes out from s_1, therefore,

$$\sum_{v \in V} f(s_1, v) = f(s_1, s_2)$$

Also, from conservation this is the same flows as the sum of flow on edges going out of s in G^*.

$$f(s_1, s_2) = \sum_{v \in V^*} f^*(s, v) = |f^*|$$

(f) **Finally, building on the above, show that the translated flow f^* is maximal**
From the previous proof it was established that there is only one edge coming out of s_1 such that

$$f(s_1, s_2) = |f^*|$$

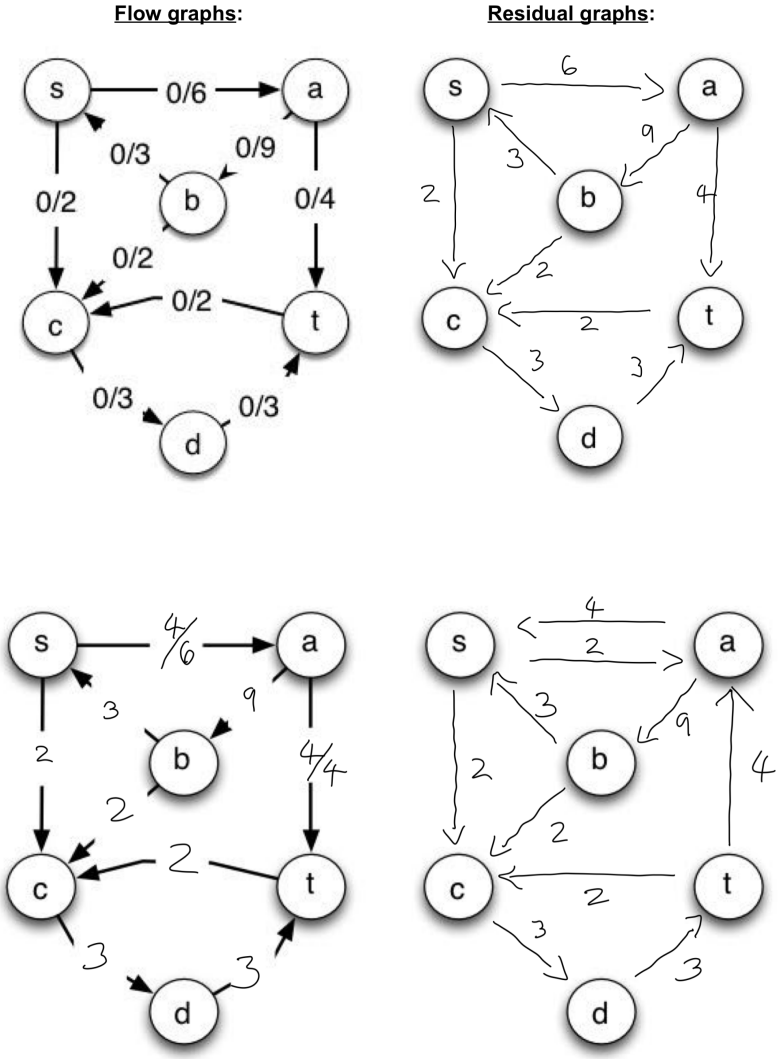
then there is only one path such that

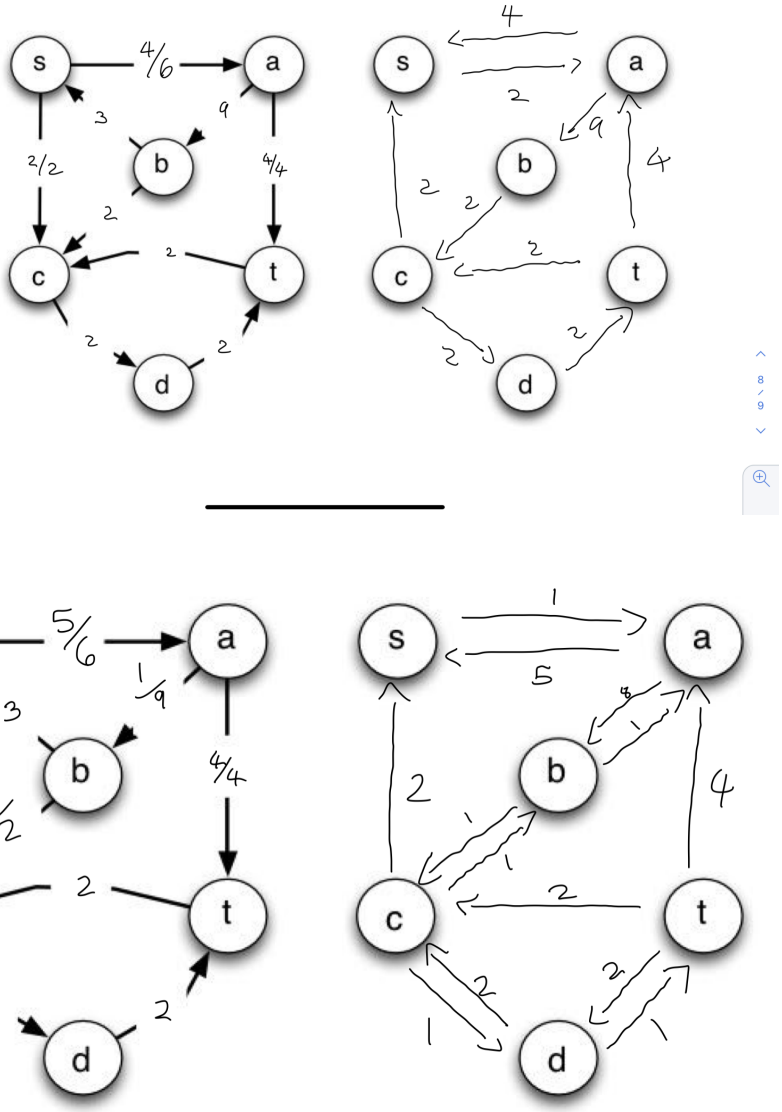
$$|f^*| = c(S, T)$$

for some cut in S and T. Then from the Max-Min Cut Theorem the statement is equivalent to f^* being a maximum flow.
Hence, f^* has a maximal flow.

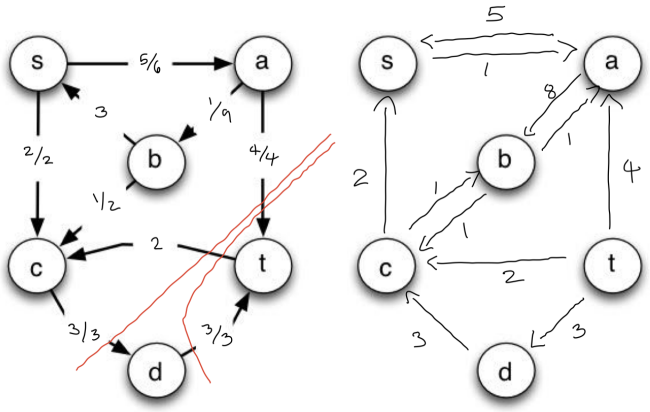
Problem 5: Tracing Edmund-Karp

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





(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**


s
a
b
c
t
d
5/6
3
1/9
2/2
4/4
1/2
2
3/3
3/3
s
a
b
c
t
d
5
1
8
1
2
1
4
1
2
3
3