# COS 485 — Homework 7

Samuel Barton Benjamin Montgomery

April 24th, 2017

## Problem 1

In this problem we categorize the following sorting algorithms into the various genres of algorithms discussed in this course.

| **Algorithm** | **Technique** | **Justification** |
|---|---|---|
| Insertion sort | Iterative Improvement | Go through each element starting at position 0. Shift all of the elements greater than the current one to the left. |
| Selection sort | Greedy | Find the minimum value, swap it with the item at position 0. |
| Bubble sort | Greedy | If the next value is smaller than the current one, swap them. |
| Quicksort | Divide and Conquer | Divide the problem size recursively, sort at the bottom, then sort on your way up. Note that Quicksort *can* be implemented with a randomized partition. |
| Merge sort | Divide and Conquer | Divide the problem size recursively, sort at the bottom, then sort on your way up. |
| Heap sort | Greedy | Larger values go higher (maxheap), or smaller values go higher (minheap) |

# Problem 2

In this problem we are asked to explain why this flow is valid, although not a maximum flow.

This flow is not optimal as we have not saturated any of the forward edges,

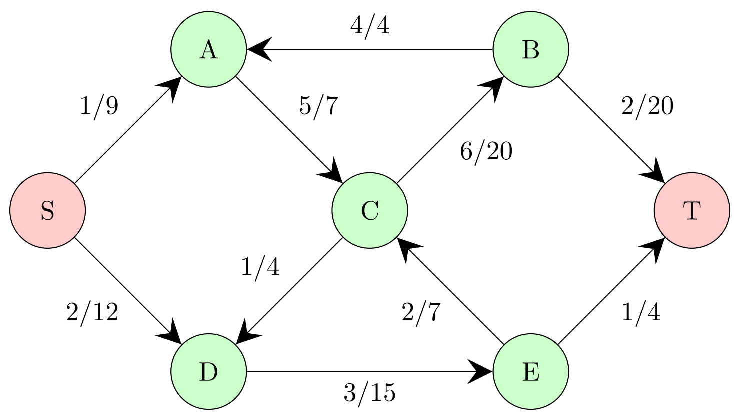



Figure 1: Flow graph with non-maximal flow

and there is a backwards edge, namely **A** — **B**, which has been saturated even though it should not have any flow whatsoever.

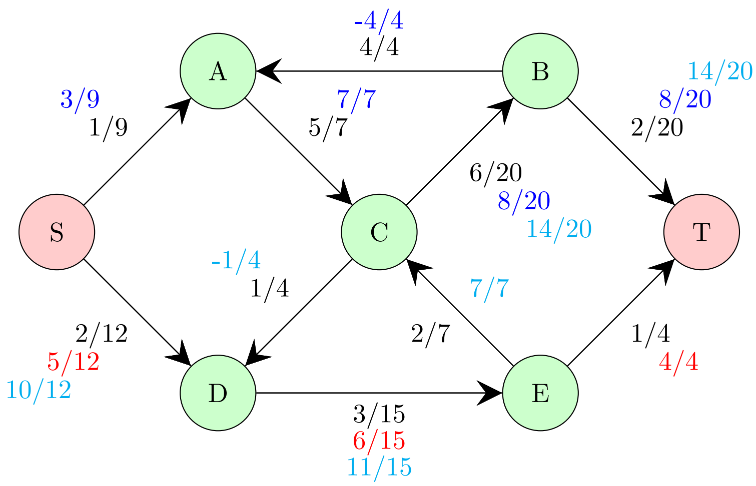
## Problem 3

In this problem we are to find a set of augmenting paths which will make the flow in Figure 1 maximal.

We consider the following set of augmenting paths:

| Path | Flow Added |
|---|---|
| **S — D — E — T** | 3 |
| **S — A — C — B — T** | 6 |
| **S — D — E — C — B — T** | 6 |

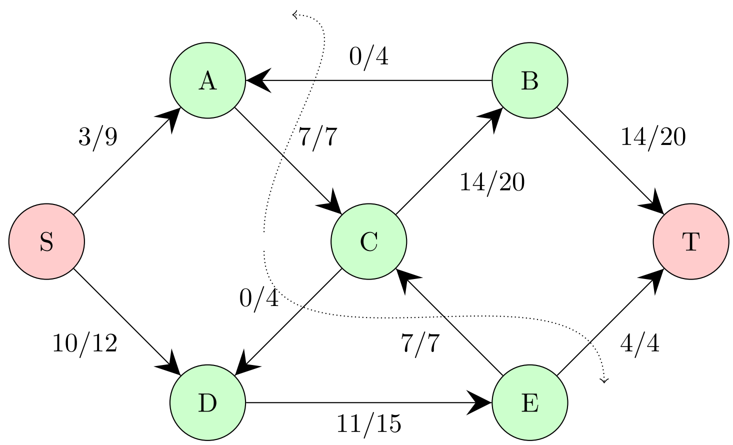
These paths correspond to the following changes to the flow of the graph.




The flow of this graph is now 18 as noted by the flows through **E** — **T** and **B** — **T**.

# Problem 4

The flow of the graph from Problem 3 is maximal as domonstrated by the following minimum cut of forward paths. **A** — **C**, **E** — **C**, and **E** — **T** are all saturated forward paths. These paths span the graph and thus form a minimum cut. The sum of the flow of these three paths is 18, and the flow of the graph is now 18 as well. Below is the final flow of the graph.




As demonstrated above, the minimum possible cut is 18.

# Problem 5

In this problem we are asked to find an O(V + E) algorithm to determine if some graph is bipartite.

First, we assert, as validated by Wikipedia (https://en.wikipedia.org/wiki/Bipartite_graph), that a graph is bipartite if and only if it has no odd length cycles.

Using this we now propose an algorithm to determine if a graph is bipartite:

1. Take one step of depth-first search
2. Look at the weight of the parent to the current node in the search space tree. If the node has no parent, then set its weight to 1. Otherwise set its weight to its parent's weight plus 1.
3. If we find a back edge, then take the difference between the weight of the current node and the node the back edge leads to. If the difference is even, then there are an odd number of nodes in the cycle and we do not have a bipartite graph.
4. repeat 1-3 until we have gone through all nodes in the graph.

This algorithm will have $T_n \in \Theta(V + E)$ assuming that our graph is represented as an adjacency list.

## Problem 6

In this problem we are asked to find the largest set of disjoint paths between two verticies in a graph.

The algorithm we propose is as follows:

1. Give each path in the graph a weight of 1
2. Use Dijkstra's algorithm to find the shortest path between the two target verticies
3. Remove the edges that consititute the path found in 2, and add the path to the set of disjoint paths between the two verticies
4. Repeat 1-3 until a path between the two verticies cannot be found

Now in the worst case, which occurs when the graph is completely connected, there will be $V-1$ paths between the two target verticies. Since Dijkstra's algorithm takes $T_n \in \Theta(V\ \lg E)$ time at each iteration, and there will be $V-1$ iterations to find all the paths, then this algorithm will take

$$T_n = (V-1) \cdot V\ \lg E\ =\ (V^2 - V)\ \lg E = V^2\ \lg E - V\ \lg E$$

Thus in the worst case the algorithm's time complexity is

$$T_n \in \Theta(V^2\ \lg E)$$