Problem 1

We must find the number of shortest paths from v-w.

Result:

Run BFS like normal, marking nodes as visited, starting at node v.
- In addition of adding the node to the queue, also add the number of incoming paths
- If node has been visited, ignore it
- If node is already in queue, increment path count with current value
- Propagate counts on queue when adding new nodes
- When reach w, number that is stored with it is number of shortest paths.

Analysis: This is very similar to just finding the shortest path in a graph using BFS. Except as we go along, we have a count to go along with each node.

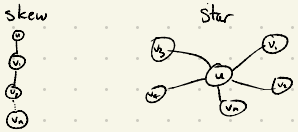
Runtime: The worst case here will be the desired node as the last iteration of our search. Worst case will go through all edges and vertices, meaning the run time is $O(m+n)$.

Proof: There is a node m that is between v and w. There must be multiple edges on m if there are more than 1 optimal path. If m has been encountered before it will not be beneficial in finding the shortest path. This means we can encounter each node a maximum of once. So, a BFS will suffice in creating an algorithm.

Problem 2

Prove $G = T$ if the DFS and BFS trees of G are the same.

Proof: for the DFS and BFS trees to be the same, G must be a skew tree or star topology



Assume G contains edge (p, q) that is not contained in T.

The DFS tree says p must be a direct ancestor of q
The BFS tree says distance between u and p q must be at maximum 1.

The vertex p must be an ancestor of q, and must be a direct parent. (p, q) must be contained in T.

The graph G has no edges that are not in T. By contradiction, $G = T$.

Problem 3

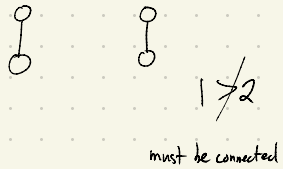
Proving or not

> *Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least n/2, then G is connected.*

Proof: This is true.

i) Max degree of node is $n-1$ for graph G with n vertices

By contradiction, graph G with N nodes is disconnected
There are two disconnected node sets of N_1 and N_2. $N_1 + N_2 = N$.
Max degrees of each set are $N_1 - 1$ and $N_2 - 1$ respectively. From i)
The degrees are all $\geq N/2$.



So we get the equations $N_1 - 1 \geq N/2$ and $N_2 - 1 \geq N/2$.

Added together: $N_1 - 1 + N_2 - 1 \geq N/2 + N/2$

$N_1 + N_2 - 2 \geq$

$N_1 + N_2 \geq N + 2$ ×

We know from before that $N_1 + N_2 = N$, so the statement above cannot be true.

It is then proven by contradiction that graph G must be connected if every node of G has at least a degree of $N/2$.

Problem 4 Ex. 3

Have to prove that the current algorithm is the best with trucks and weights

Proof: Consider that there is an optimal solution O that does not send packages by how fast they arrive.

Our solution will order the boxes based on how they come in, $b_{p_1}, b_{p_2}, \ldots, b_{p_k}$
The other solution, will fit the boxes differently, in set O. $b_{q_1}, b_{q_2}, \ldots, b_{q_m}$
We will prove that ours is just as ideal through induction ($m \leq k$).

i) For each $T = 1, 2, \ldots, m$, we have $b_{k_T} \geq b_{m_T}$

Base case: $T = 1$. (trivial)
Will pack as many boxes until the only truck can't carry anymore.

Inductive case: $T > 1$. Assume claim is true for all $i < T$.

$$b_{q_T} - b_{q_{T-1}} \leq W$$

Since O is valid set of boxes

$$b_{q_T} - b_{p_{T-1}} \leq b_{p_T} - b_{q_{T-1}}$$

since $b_{T-1} \geq b_{q_{T-1}}$ by induction hypothesis. Combine both inequalities.

$$b_{q_T} - b_{p_{T-1}} \leq W$$

This means that the company has the option of packing $b_{p_{T-1}}$ to b_{q_T} in one truck; the box b_{p_T} that they pack can only be better packed than b_{q_T}.
Greedy algorithm stays ahead, at each step, the choice made by the other solution is one of its valid options.

Greedy algorithm produces valid set of boxes.

Problem 5

Finding an algorithm for the best way to send people out. Finding the earliest completion time.

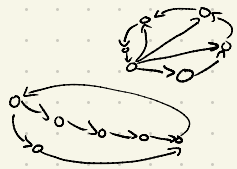
Result: Let s_i, b_i, and r_i represent swimming, biking, and running times respectively.

Arrange students in order of decreasing b_i and r_i.
Send them out in this order

Analysis & Proof: Suppose an optimal solution uses an ordering different.
Student j is sent before k and $b_j + r_j > b_k + r_k$. Student k will get in the pool as soon as j gets out. However, the completion time will not be lesser than ours as $b_j + r_j > b_k + r_k$. Because k's bike and run times are lesser than j's, this evens out j being sent in. Our solution is optimal.

Complexity: We perform one sort, by decreasing bike plus running times, so the run time here is only: $O(n \log n)$.

Problem 6

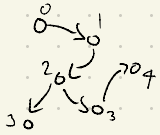


a) We must find longest path in a directed graph.

Result: You cannot find an algorithm that uses each edge at most once. The Longest Path Problem is classified as NP-Complete.

b) Finding longest path in a DAG (Directed acyclic graph)

Result: Given graph G and n vertices
Create distances[n], and distances[s] is 0, the source vertex. All the rest is negative infinity.
Do a topological sort of the graph G from s. Put in order V.
For every vertex u in V
  For every vertex v adjacent to u.
    if distances[v] < distances[u] + 1)
      distances[v] = distances[u] + 1
Return max of distances[].



Analysis: First we find a topological sorting of G, to create a linear ordering of the vertices in the graph. The distances are stored in the array. We calculate the distances by iterating through the ordering, adding up the distance as it goes on.

Proof: For every given DAG G, the topological sort will create a linear ordering of every vertex. This means there cannot be a longer path than what is in the ordering. We will go through all the vertices and find the longest path. By contradiction, there cannot be a linear ordering of G that does not contain the longest path.

Runtime: First, the sort is O(m+n). The next loop goes over every vertex and adjacent vertex. This inner loop takes O(m+n) as well. So, overall, the time complex. is O(m+n).