Candidate Number: 2031B
Paper 1
Question 9

a. s -> a -> t = (3, 3)
   s -> a -> b -> t = (2, 3)
   s -> b -> t = (4, 1)

b. Proof by contradiction. Assume that v0 -> v1 -> … -> vk-1 is has costs (a,b) and is not Pareto efficient. Therefore there exists some other path v0 -> u1 -> u2 -> … -> vk-1 for which both costs are lower, say (a', b'). Let (c, d) be the costs of the edge vk-1 -> vk. Therefore the path v0 -> u1 -> u2 -> … -> vk-1 -> vk has costs (a'+c, b'+d). The original path v0 -> v1 -> … -> vk has costs (a+c, b+d). Since a' < a, a'+c < a+c. Since b' < b, b'+d < b+d. Therefore both costs of v0 -> u1 -> u2 -> … -> vk-1 -> vk are less than the costs of v0 -> v1 -> … -> vk and so the latter is not Pareto efficient, which is a contradiction. Therefore v0 -> v1 -> … -> vk-1 must be Pareto efficient.

c. We have assumed that some Pareto efficient path v0 -> … -> vk with costs(ca, cb) exists. We need to make this assumption because the question does not specify that the graph has to be connected (i.e. there might be some v0 and vk for which no path between them exists at all).

   Proof by contradiction that that this path has <= V-1 edges:
   Assume that the number of edges is >= V. Therefore the number of visited vertices >= V+1. By the pigeonhole principle, this means that at least on vertex must have been visited more than once. Therefore the path contains a cycle. Since all costs are >= zero, both costs of this path will be either decreased or remain the same by removing this cycle from the path. Therefore, the path is not Pareto efficient, which is a contradiction.

   Therefore this path has <= V-1 edges

Candidate Number: 2031B
Paper 1
Question 9

```
def all_paths_of_length(s, graph, l):
    if l == 0:
        return [[], 0, 0]
    all_paths = []
    for v0, v1, c1, c2 in graph.edges:
        if v0 == s:
            paths = all_paths_of_length(v1, graph, l-1)
            for p, cost1, cost2 in paths:
                all_paths.append(([s]+p, cost1+c1, cost2+c2))
    return all_paths

def all_pareto_costs(s, graph):
    # graph.vertices = e.g., [v0, v1, ...]
    # graph.edges = e.g., [(v0, v1, c1, c2), ...]
    all_paths = all_paths_of_length(s, graph, graph.vertices.length)
    pareto_costs = []
    for path, c1, c2 in all_paths:
        found = False
        for other_path, oc1, oc2 in all_paths:
            if oc1 < c1 and oc2 < c2:
                found = True
                break
        if not found:
            pareto_costs.append((c1, c2))

    return pareto_costs
```

d.

e. First: * proof that all_paths_of_length(s, graph, l) returns every sequence of nodes starting from s which is of length l (as well as the costs).

When l equal 0, the base case triggers, returning [[], 0, 0] which is indeed the correct answer, as the only path of length 0 is the path with 0 nodes (the empty list), with costs 0 and 0.

Assuming the * holds for l=k, I will prove that * holds for l=k+1.

When l=k+1, each edge from s to some other node v1 is considered. The function is called recursively which by assumption returns all the paths of length k from v1. Prepending s to each of these, and adding on the costs from the current edge gives the list of all edges of length k+1 which start s -> v1. Therefore, accumulating these over all edges from v1 gives all paths of length k+1 which start at s.

By induction, * holds for all l >= 0.

Next: ** proof that all_pareto_costs is correct.
Any path of length > V where V is the number of vertices in the graph must by the pigeonhole principle visit at least one node more than once. Therefore this path contains a

cycle. Since all costs are >= zero, both costs of this path will be either decreased or remain the same by removing this cycle from the path. Therefore, the path is not Pareto efficient. Therefore, all Pareto efficient paths must have length <= V

It follows that the list returned by the call to all_paths_of_length(s, graph, graph.vertices.length) must contain all Pareto efficient paths.

For each of them, we check if there exists any other path in that list for which both costs are less than its own (we do not need to worry about checking the path against itself, because by definition this test will fail). If this test passes, the path being tested is not a Pareto efficient path. Otherwise, it is a Pareto efficient path, and we can add its cost to the list of Pareto efficient costs.

Therefore, after this iteration, that list will contain all possible Pareto costs from s.