1. (a) No, in the given CFG every node is attached to an edge. If this edge is covered the node must then also be covered. Because of this we know that when all edges are covered, all nodes are covered.

   (b) No, not for this CFG. To prove this we will assume we have a test set that covers all nodes and prove that it covers all of the edges

   1-2 Because our test set covers all the nodes there must be a path that contains the node 2. All paths start at the entry node 1, hence this path contains a path from 1 to 2. If we look at the graph we see that there is only one way to get from 1 to 2 and that is by taking the edge 1-2, hence 1-2 is covered.

   1-4 Because our test set covers all the nodes there must be a path that contains the node 4. All paths start at the entry node 1, hence this path contains a path from 1 to 4. If we look at the graph we see that there is only one way to get from 1 to 4 and that is by taking the edge 1-4, hence 1-4 is covered.

   4-5 Because our test set covers all the nodes there must be a path that contains the node 4. All paths end at the exit node 5, hence this path contains a path from 4 to 5. If we look at the graph we see that there is only one way to get from 4 to 5 and that is by taking the edge 1-4, hence 1-4 is covered.

   2-5 Because our test set covers all the nodes there must be a path that contains the node 2. All paths end at the exit node 5, hence this path contains a path from 2 to 5. If we look at the graph we see that to get from 2 to 5 we need to pass the edge 2-5, hence 2-5 is covered.

   2-3 Because our test set covers all the nodes there must be a path that contains the node 3. All paths start at the entry node 1, hence this path contains a path from 1 to 3. If we look at the graph we see that to get from 1 to 3 we need to pass the edge 2-3, hence 2-3 is covered.

   3-2 Because our test set covers all the nodes there must be a path that contains the node 3. All paths end at the exit node 5, hence this path contains a path from 3 to 5. If we look at the graph we see that to get from 3 to 5 we need to pass the edge 3-2, hence 3-2 is covered.

   We see that all edges are covered when all nodes are covered. Because of this we can conclude that there can not be a test set that covers all nodes, but not all edges.

   (c) Yes, consider the following test set

   ```
   {[1,2,3,2,5], [1,4,5]}
   ```

   It should be clear that this test set covers all edges, but it does not cover the edge pair (1-2,2-5).

(d) The minimum test set that covers all pairs of edges is of size 3. Observe that the following test set covers all pairs of edges.

`{[1,4,5],[1,2,5],[1,2,3,2,3,2,5]}`

Because of this we know that the minimum size of a test set is smaller or equal to 3. Now we only need to show that minimum size is greater or equal to 3 or in other words greater than 2.

Assume we have test set that covers all edge-pairs. This means that (1-4,4-5) is covered. Because 1 has no in-going edges and 5 has no out-going edge we can conclude that [1,4,5] must be a path in our test set. Also (1-2,2-5) is covered. Because 1 has no in-going edges and 5 has no out-going edge we can conclude that [1,2,5] must be a path in our test set. We see that a test set that covers all edge pairs must contains [1,2,5] and [1,4,5]. At the same time we see that

`{[1,4,5],[1,2,5]}`

does not cover (1-2,2-3) for example. Therefore a cover of all pairs of edges must be bigger than

`{[1,4,5],[1,2,5]}`

and therefore have at least 3 paths. Because we have already proven that the minimum cover of all pair edges has less or equal than 3 paths, we know that this must be exactly 3.

2. (a) No, the length of a path denotes the amount of edges, so for example length([1,2]) = 1. Therefore we see that $[1,2] \notin \{[x] | x \in \text{nodes}(G)\}$ and $[1,2] \in \{s | s \in \text{paths}(G) \text{ and } \text{length}(s) \leq 1\}$, hence they are not equivalent

   (b) There are two: [1,2,4,5,6,1] [1,2,4,6,1]

   (c) There are four: [2,3,2] [3,2,3] [3,2,4,6,1,7] [3,2,4,5,6,1,7]

   (d) There are fifteen: [1,2,4,5,6,1] [1,2,4,6,1] [2,3,2] [3,2,3] [3,2,4,6,1,7] [3,2,4,5,6,1,7] [2,4,5,6,1,7] [2,4,6,1,2] [2,4,5,6,1,2] [4,6,1,2,3] [4,5,6,1,2,3] [4,6,1,2,4] [4,5,6,1,2,4] [5,6,1,2,4,5] [6,1,2,4,6] [6,1,2,4,5,6]

3. The pre-condition will always be true in this case. We will denote a condition that is always true as True. For the post-condition we know that it will return x if and only if $x \leq y$ and will return y if and only if $y \leq x$.

```
{*True*}
int MIN(int x, int y)
{*(retval = x ⟺ x≤y) ∧ (retval = y ⟺ x≥y)*}
```

4. The pre-condition will always be true in this case. We will denote a condition that is always true as True. For the post-condition we know that it will return true if and only if and only if m is smaller or equal to both x and y and m is equal to x or y.

```
{*True*}
int MIN(int x, int y)
{*retval = true ⟺ (m≤x ∧ m ≤ y ∧ (m = x ∨ m = y))*}
```

5. (a) direct tour, because [0,1,2,0] is a subpath. side trip, because [0,1,2,0] has the edges in the same order. It is also a detour, because it is a subsequence

   (b) side trip, because [0,1,2,0] has the edges in the same order. It is also a detour, because it is a subsequence

   (c) detour, because [0,1,2,0] is a subsequence.