Writeup

(a) The possible paths taken taken to achieve the expected outcome, that is the first path that reaches node Z, is basically dependent on the alphabetical order of the nodes in the tree and also on the what level of the tree(depth) node Z is on. It's dependent on the alphabetical order a little since the algorithm will find new junctions to travel to in the alphabetical order so it may lead to vertices that ultimately lead to a dead end.

(b) Depending on the location of the node Z and the depth of the tree, the complexity of the search can vary. As a result, in many cases it may take several nodes(up to all possible nodes in the tree except Z) to get to node Z finally. Since this algorithm also takes a trial and error approach at finding node Z, depending on the size of the tree, it can take a while compute to right path(s). As a result, the computation intensity can vary.

***Notes about output from infer analyzer.**

When "make infer" is run, the infer analyzer gives an error about dead storage because a pointer passed in to the stack_pop() function and the pointer isn't used for anything except for being assigned to value that is popped. However, I decided to follow the implementation shown by the API which recommends doing that.

```
stack.c:65: error: DEAD_STORE
  The value written to &item (type unsigned int*) is never used.
  63.     } else {
  64.       s->top -= 1;              // Decrement stack pointer.
  65. >     item = &s->items[s->top]; // pop element to pointer passed.
  66.       return true;              // Return true if element popped successfully.
  67.     }
```