1. "The algorithm does not always find the optimal solution" is an almost-equivalent, but a more ambiguous statement of "the algorithm could return an incorrect solution" (I should have used the later for Question 1 in the practice midterm). Note that "incorrect" means that "not optimal" or "suboptimal" here. So for Question 1 of practice midterm, the given incorrect graph search algorithm never checked whether the node is in *closed* (i.e. explored before), thus it is effectively doing tree search. Tree search could not return an "suboptimal solution" – when tree search returns any solution it will be the best optimal solution; however, it could possibly (e.g. depth-first tree search) not return any solution at all if stuck in infinite loops. Comparing to tree search, graph search will not only eliminate redundant paths but also avoid infinite loops.

   Recall that, a search algorithm is **complete**, if whenever there is at least one solution, the algorithm is guaranteed to find it withing a finite amount of time. A search algorithm is **optimal** if when it finds a solution, it is guaranteed to be the best one (e.g. the least cost).

2. The correct implementation of generic graph search and A* graph search looks as follows. By "generic", it means that for depth-first (a stack - last in first out), breadth-first (a queue - last in last out), uniform cost (a priority queue), and A* tree search (a priority queue; also need heuristics), the only difference is what you use to implement the fringe; A* search in addition considers heuristics.

```
function GRAPH-SEARCH(problem, fringe)
    closed ← an empty set,
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem,STATE[node]) then
            return node
        end if
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
        end if
    end loop
end function

function A*-GRAPH-SEARCH(problem, fringe, Heuristic)
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem,STATE[node]) then
            return node
        end if
        if STATE[node] IS NOT IN closed then
            ADD STATE[node] TO closed
            for successor IN GETSUCCESSORS(problem, STATE[node]) do
                h ← Heuristic(successor, problem)
                fringe ← INSERT(MAKE-NODE(successor, h), fringe)
            end for
        end if
    end loop
end function
```