1. **(3 points)** Assume we run $\alpha - \beta$ pruning expanding successors from left to right on a game with tree as shown in Figure 1 (a). Then we have that:

(a) *(true or false)* For some choice of pay-off values, no pruning will be achieved (shown in Figure 1 (a)).

(b) *(true or false)* For some choice of pay-off values, the pruning shown in Figure 1 (b) will be achieved.

(c) *(true or false)* For some choice of pay-off values, the pruning shown in Figure 1 (c) will be achieved.

(d) *(true or false)* For some choice of pay-off values, the pruning shown in Figure 1 (d) will be achieved.

(e) *(true or false)* For some choice of pay-off values, the pruning shown in Figure 1 (e) will be achieved.

(f) *(true or false)* For some choice of pay-off values, the pruning shown in Figure 1 (f) will be achieved.

![Figure 1: Game trees.](image)
2. The following implementation of graph search may be incorrect. Circle all the problems with the code.

```java
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
    add STATE[node] to closed
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
end loop
end function
```

(a) Nodes may be expanded twice.
(b) The algorithm is no longer complete.
(c) The algorithm could return an incorrect solution.
(d) None of the above.

3. (2 points) The following implementation of A* graph search may be incorrect. You may assume that the algorithm is being run with a consistent heuristic. Circle all the problems with the code.

```java
function A*-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 STATE[node] is not in closed then
        add STATE[node] to closed
        for successor in GETSUCCESSORS(problem, STATE[node]) do
            fringe ← INSERT(MAKE-NODE(successor), fringe)
            if GOAL-TEST(problem, successor) then
                return successor
            end if
        end for
    end if
end loop
end function
```

(a) Nodes may be expanded twice.
(b) The algorithm is no longer complete.
(c) The algorithm could return an incorrect solution.
(d) None of the above.