

Assignment 1

Wen Dong #110057395

3.8 (a)

Answer: because if there are arbitrary negative path costs, any unvisited node, which might hold a value negative enough, can bring advantage to a search path to be the optimal path by appending the negative-valued node itself to the search path.

39 (a)

Initial state – three missionaries and three cannibals on one bank of a river along with a boat

Actions – ship one or two of the missionaries and cannibals from one to the other side of the river

Transition model – returns the resulting state as per the given state and actions

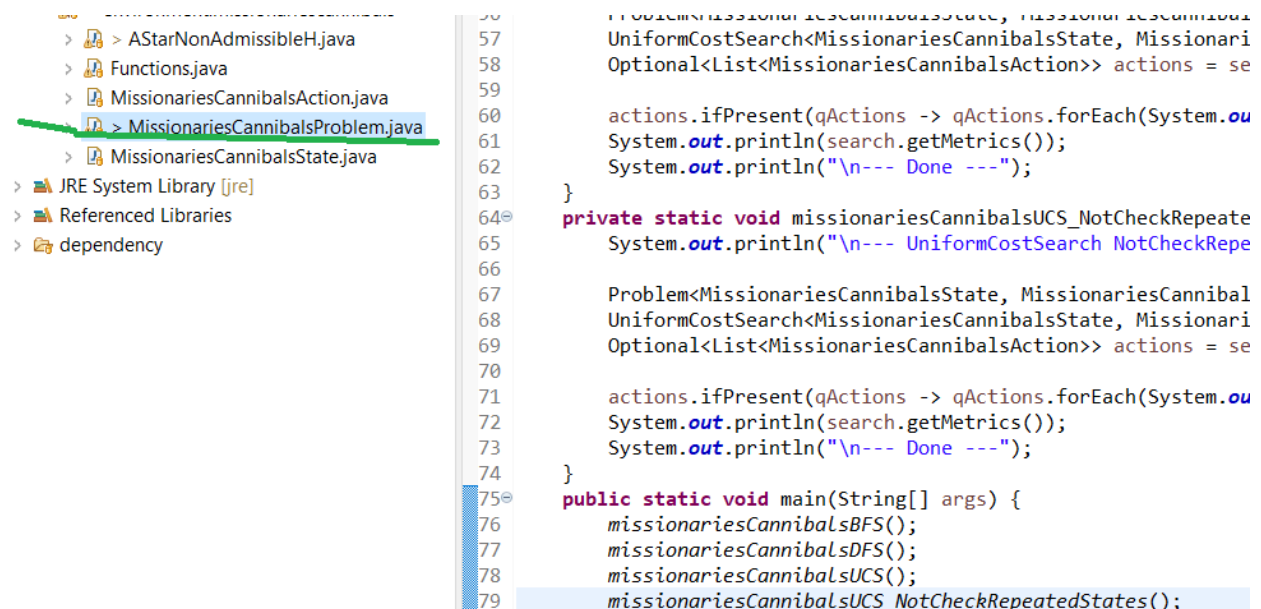
Goal test – checks whether all the missionaries and cannibals are on the other bank of the river.

Path cost – the cost is the number of shipments where each shipment is of cost 1

39 (b) Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

Firstly, yes it is a good idea to check repeated states to avoid repeated node visit and expanding, the advantage is obvious, with repeated states check, it usually finds the optimal solution with dozens of node expanding, while if not check, it will conduct thousands of expanding.

Source code of my implementation in 4 algorithms:



```
57 Problem<MissionariesCannibalsState, MissionariesCannibal
58 UniformCostSearch<MissionariesCannibalsState, Missionari
59 Optional<List<MissionariesCannibalsAction>> actions = se
60 actions.ifPresent(qActions -> qActions.forEach(System.out
61 System.out.println(search.getMetrics());
62 System.out.println("\n--- Done ---");
63 }
64 private static void missionariesCannibalsUCS_NotCheckRepeate
65 System.out.println("\n--- UniformCostSearch NotCheckRepe
66
67 Problem<MissionariesCannibalsState, MissionariesCannibal
68 UniformCostSearch<MissionariesCannibalsState, Missionari
69 Optional<List<MissionariesCannibalsAction>> actions = se
70
71 actions.ifPresent(qActions -> qActions.forEach(System.out
72 System.out.println(search.getMetrics());
73 System.out.println("\n--- Done ---");
74 }
75 public static void main(String[] args) {
76     missionariesCannibalsBFS();
77     missionariesCannibalsDFS();
78     missionariesCannibalsUCS();
79     missionariesCannibalsUCS_NotCheckRepeatedStates();
```

Run the `MissionariesCannibalsProblem.java` file to see the results:

```
BreadthFirstSearch - {maxQueueSize=3, nodesExpanded=13, pathCost=11.0, queueSize=2}
DepthFirstSearch   - {maxQueueSize=5, nodesExpanded=11, pathCost=11.0, queueSize=4}
UniformCostSearch  - {maxQueueSize=3, nodesExpanded=14, pathCost=11.0, queueSize=1}
UniformCostSearch NotCheckRepeatedStates
                    - {maxQueueSize=14520, nodesExpanded=11445, pathCost=11.0,
                      queueSize=14519}
```

Performance comparison:

As shown above, with graph search checking repeated states, BFS, DFS, and UniformCostSearch performs well to find the optimal path, but with tree search during UniformCostSearch which does not check repeated states, it expands more than ten thousands of node expanding owing to repeated states visits.

3.13

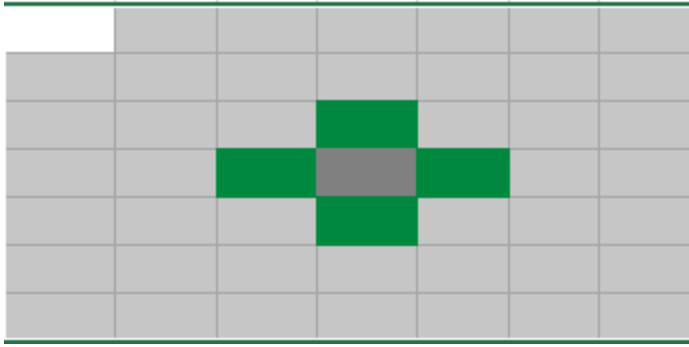
Illustrated step by step with GRAPH SEARCH

Green denotes frontier, Gray denotes closed, White denotes unexplored.

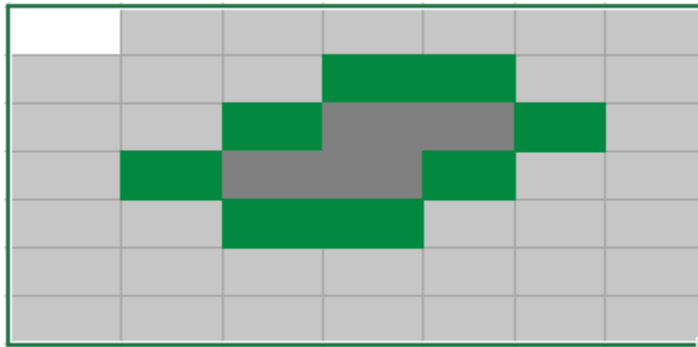
1. Initial state – complies with the rule



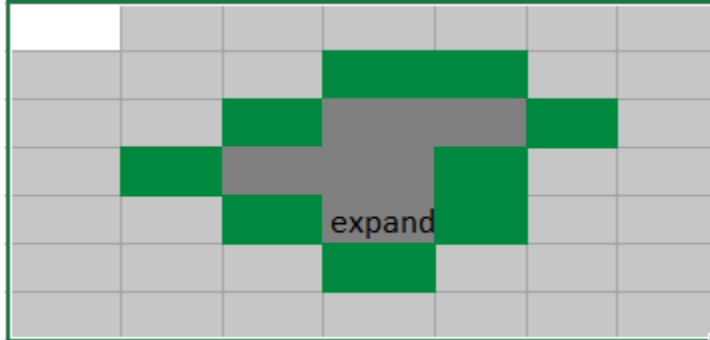
2. After Source expanded – complies with the rule



3. Suppose it hold the property before applying the expanding as below.

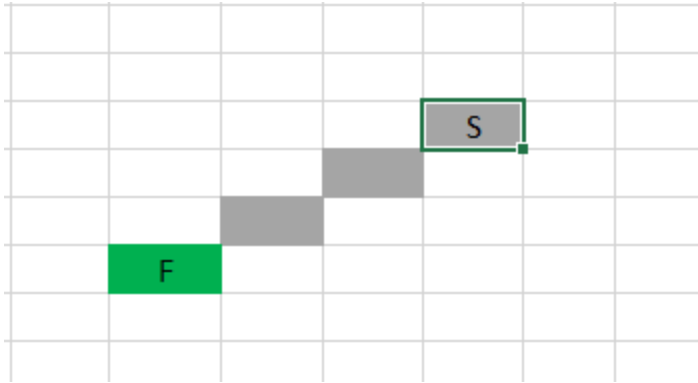


4. We can see expanding of any frontier results a state compliant with the rule



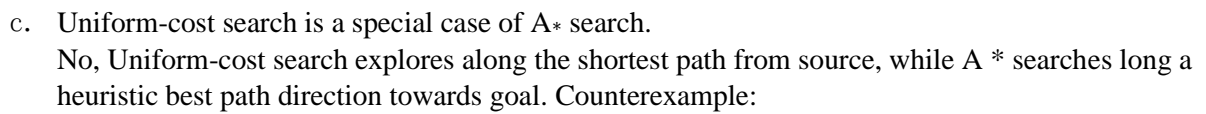
We can safely deduce that GRAPH SEARCH satisfies the graph separation property illustrated in Figure 3.9.

Depth First Search algorithm violates the rule, as it expands nodes along depth direction, there is no frontier separating the explored and the unexplored.



3.21 Prove each of the following statements, or give a counterexample:

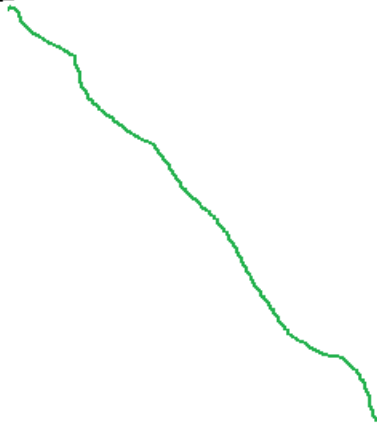
- a. Breadth-first search is a special case of uniform-cost search.
 Yes, when all path costs are equal, uniform-cost search is Breadth-first search. Because UCS chooses frontier node with least cost to source as the next visit, so it must explore along breadth direction.
- b. Depth-first search is a special case of best-first tree search.
 No, Depth-first search is an uninformed search, while Best-First tree search is an informed search. Depth-First searches along depth as below black line, while Best-First tree searches along the best heuristic path towards goal as below green line.



Start

A * search

Start



Goal

Create a small example of your own and follow the A* algorithm on non-admissible heuristic to see what the output is. Is the output optimal?

Yes, still optimal the output. Although the heuristic is definitely greater than true cost for some cases.

```
public static double getHeuristicDistance(Node<EightPuzzleBoard, Action> node)
{
    EightPuzzleBoard currState = node.getState();
    int result = 0;
    for (int val = 1; val <= 8; val++) {
        XYLocation locCurr = currState.getLocationOf(val);
        XYLocation locGoal = GOAL_STATE.getLocationOf(val);
        result += Math.sqrt(Math.abs(locGoal.getXCoOrdinate() +
locCurr.getXCoOrdinate()));
        result += Math.sqrt(Math.abs(locGoal.getYCoOrdinate() +
locCurr.getYCoOrdinate()));
    }
    return result;
}
```

Implementation as below file, output is:

```
{maxQueueSize=29743, nodesExpanded=125642, pathCost=23.0, queueSize=28901}
```

```
8700W [uwinds2 master]
src
  environment.missionariescannibals
    AStarNonAdmissibleH.java
    Functions.java
    MissionariesCannibalsAction.java
    MissionariesCannibalsProblem.java
    MissionariesCannibalsState.java
```

Note question 3.9 is in fact a programming question, although the original question asks you to implement a solution using one algorithm, you should try at 3 different search algorithms and compare their performance. In your submission, you should report the algorithms implemented and show the performance comparison.