Mitchell Chan 10753135
Peter Chung 38777124

CPSC 322
Assignment 1

1.1 DFS

a) a, b, c, d, e, f, g, h, i, z
b) a → b → c → d → e → f → g → h → i → z
c) 64

1.2 BFS

a) a, b, e, c, f, g, d, f, g, h, i, z
b) a → e → g → z
c) 16

1.3 A*

a) a, e, b, g, f, c, d, z
b) a → e → g → z
c) 16

1.4 B&B

a) a, b, c, d, e, f, g, h, i, z, i, z, z, g, z, f, e, f, g, h, z, g, h, i, z
b) a → e → g → z
c) 16

1.5

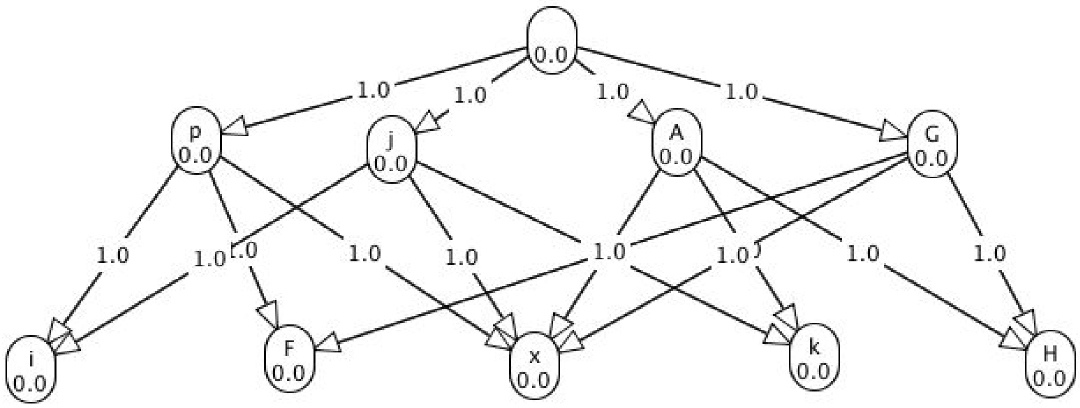
a) Yes
b) Yes, but take different paths to get to optimal solution. If cycle exists, B&B not able to get out of cycle but BFS can
c) No it didn't expand fewer nodes than A*. In B&B, a variant of DFS, we would first go all the way to the goal node in alphabetical order, meaning that we'd have to expand a, b, c, d, e, f, g, h, i, z first (a total of 10 expansions already). These expansions are already greater than the total expansions done in A*.

Mitchell Chan 10753135
Peter Chung 38777124

2.1

a) A node is current board state. Root node is initial board state with the only empty hole being at x.
b) Goal node is to have only one peg left on the board occupying x.
c) Arcs are all valid moves that are one move away from the current node. In this case, it would be a set of valid locations on the board where a peg could be removed from the current node
d) 2^{33}

2.2



a)
b) Length of solution is 32 because we are eliminating a single peg with each move. There are 33 spots on the board and the goal state is having a single peg in the middle. We start out with one empty spot.

2.3

a) BFS. We know solution will be found at depth = 32. Worst case DFS will have to search every path, but for BFS will find first solution at d=32 and stop.
b) No. Cycle checking not required because since we are removing a peg each turn, there are less moves available and not the same number of pegs that can be used.
c) No because the solution path is limited to depth=32. Therefore, there are no shorter paths that can be taken that will lead to the same state.

Mitchell Chan 10753135
Peter Chung 38777124

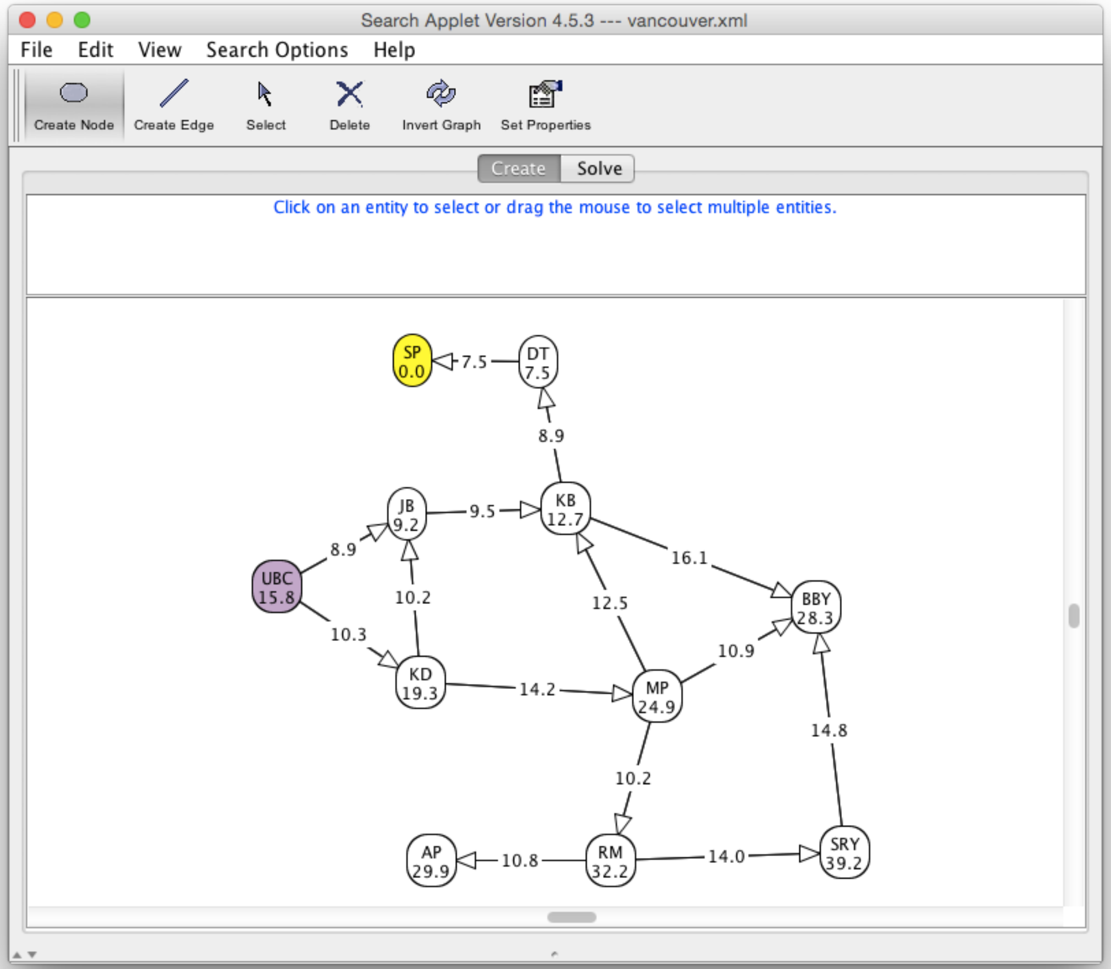
3.1

a) A node is the current board state. Root node is the initial board state.
b) Goal node is to having all home cells occupied with all the cards in proper order of rank, from lowest to highest
c) Arcs would be all valid moves that are one move away from current node.

3.2 An admissible heuristic is where we don't have to place cards in the home cells in order. If a card is of a valid suit and can be moved to its appropriate pile, then we can move it straight to the home cell.

Mitchell Chan 10753135
Peter Chung 38777124

4



Click on an entity to select or drag the mouse to select multiple entities.

Nodes expanded: 7
Optimal path: UBC → JB → KB → DT → SP
Path cost: 34.8

4.1

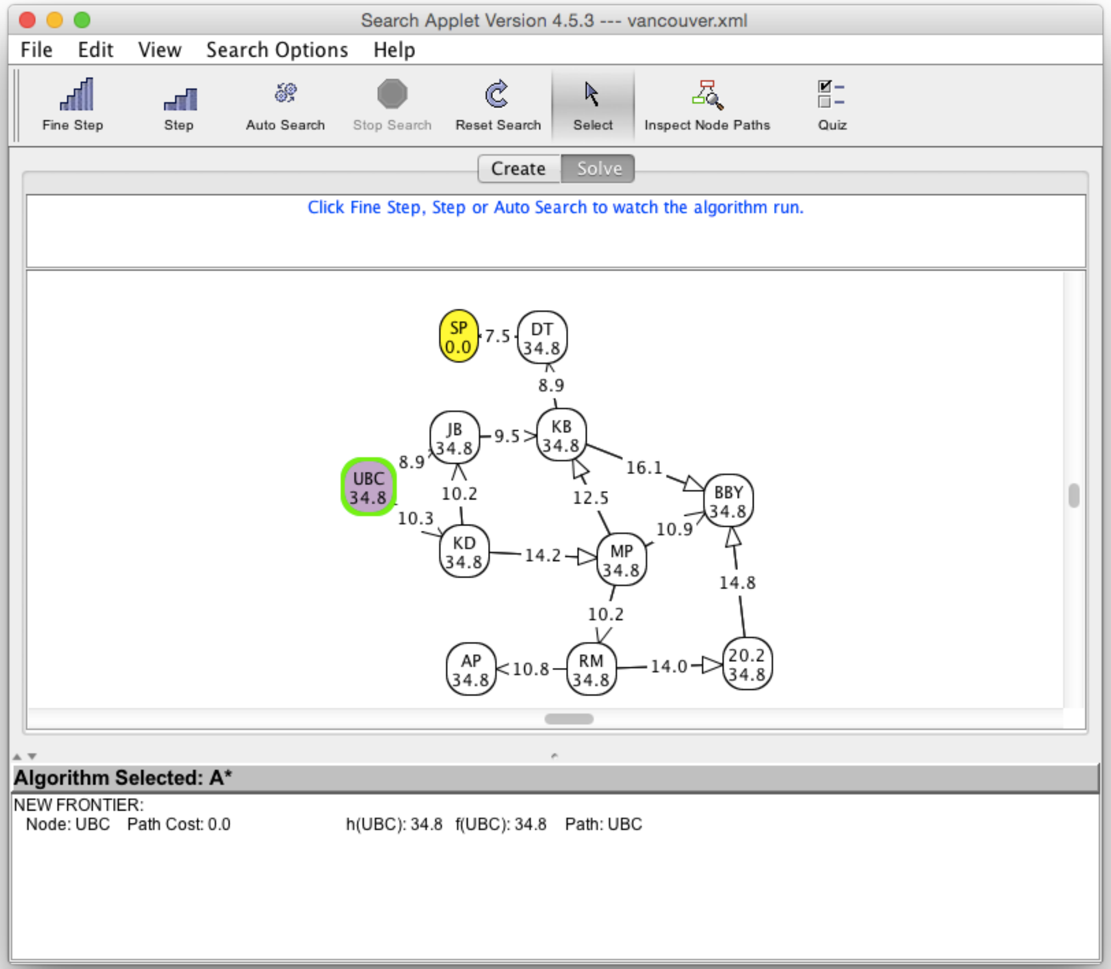
- Subtract from all (-7):
  i) Nodes expanded: 7
  ii) Optimal Path: UBC → JB → KB → DT → SP
  iii) Path Cost: 34.8
- Reduce on path (-7):
  i) Nodes expanded: 7
  ii) Optimal Path: UBC → JB → KB → DT → SP
  iii) Path Cost: 34.8

Mitchell Chan 10753135
Peter Chung 38777124

- Reduce ones not on path (-19):
  i) Nodes expanded: 8 (UBC>KD>JB>JB>MP>KB>DT>SP)
  ii) Optimal Path: UBC → JB → KB → DT → SP
  iii) Path Cost: 34.8

a. A* finds the optimal path in all of the situations.
b. Efficiency is reduced when the h(n) of the nodes not on the solution path are reduced. This forces the algorithm to expand more nodes.
c. A* will be optimal so long as h(n) is an underestimate and non-negative. A* expands the minimal number of paths so it is optimally efficient.

4.2



Click Fine Step, Step or Auto Search to watch the algorithm run.

Algorithm Selected: A*

```
NEW FRONTIER:
  Node: UBC   Path Cost: 0.0            h(UBC): 34.8   f(UBC): 34.8   Path: UBC
```

Nodes expanded: 8 (UBC>JB>KD>KB>JB>MP>DT>SP)
Optimal Path: UBC → JB → KB → DT → SP
Path Cost: 34.8

Mitchell Chan 10753135
Peter Chung 38777124

a) Yes, A* still finds the optimal path
b) Efficiency is decreased. In the case where we leave the goal node equal to 0, efficiency is only decreased slightly.
c) If all heuristics are the same except for the goal, then it is an admissible heuristic. Optimality is maintained as h(n) is admissible. Optimally efficient as A* still explores minimal number of paths - with h(n)'s being equal, the only thing that matters is path cost

4.3

- Add to all (+20):
    - iv) Nodes expanded: 7
    - v) Optimal Path: UBC → JB → KB → DT → SP
    - vi) Path Cost: 34.8
- Add on path (+20):
    - i) Nodes expanded: 8
    - ii) Optimal Path: UBC → JB → KB → DT → SP
    - iii) Path Cost: 34.8
- Add ones not on path (+20):
    - i) Nodes expanded: 5 (UBC>JB>KB>DT>SP
    - ii) Optimal Path: UBC → JB → KB → DT → SP
    - iii) Path Cost: 34.8

a) A* finds the optimal path in all conditions
b) A* efficiency is improved when we increase the h(n) of nodes not on the solution path by a large constant. By doing so, this disqualifies all these paths from being explored by A* because their f-values are larger than that of the nodes on the optimal path. Efficiency is decreased when increase the h(n) of nodes on the solution path. The f-value of some nodes may be greater than those not on the solution path so A* checks those additional paths.
c) Optimal efficiency is maintained in all conditions as A* only expands the minimal number of paths. Any solution returned by the algorithm is guaranteed to be optimal for the constraints presented to it.