Trung Tran
CSCI 350

Assignment #5 (For steps 3, 4, 5, 6 and extra credit)

Step 3: I'll begin by providing the output of each search function used for the farmer problem, water jug and missionaries & cannibals.

Water Jug:

a) Breadth First Search:

```
Performing breadth first search on problem water jug.
#<SEARCH-STATISTICS {1006B4F163}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 239
  MAXIMUM-LENGTH-OF-NODE-LIST  = 114
  LENGTH-OF-SOLUTION           = 6
  MAXIMUM-DEPTH                = 7
%
#<NODE {1006A47E33}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<JUG-STATE {1006B66243}>
  PROBLEM    = #<PROBLEM {1005875D23}>
  PATH       = (DUMP-2 FILL-2-FROM-5 DUMP-2 FILL-2-FROM-5 DUMP-2 EMPTY-5-INTO-2)
  ANCESTORS  = NIL
```

b) DFS with duplicate node detection:

```
Performing depth first search with duplicate node detection on problem water jug.
#<SEARCH-STATISTICS {1002C93943}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 12
  MAXIMUM-LENGTH-OF-NODE-LIST  = 2
  LENGTH-OF-SOLUTION           = 6
  MAXIMUM-DEPTH                = 7
%
#<NODE {1002CFDA23}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<JUG-STATE {1002CFD983}>
  PROBLEM    = #<PROBLEM {1005875D23}>
  PATH       = (DUMP-2 FILL-2-FROM-5 DUMP-2 FILL-2-FROM-5 DUMP-2 EMPTY-5-INTO-2)
  ANCESTORS  = NIL
```

c) DFS with depth limit

```
Performing depth first search with depth limit on problem water jug.
#<SEARCH-STATISTICS {1002F06F83}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 13
  MAXIMUM-LENGTH-OF-NODE-LIST  = 4
  LENGTH-OF-SOLUTION           = 6
  MAXIMUM-DEPTH                = 7
%
#<NODE {1002F10B23}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<JUG-STATE {1002F10A83}>
  PROBLEM    = #<PROBLEM {1005875D23}>
  PATH       = (DUMP-2 FILL-2-FROM-5 DUMP-2 FILL-2-FROM-5 DUMP-2 EMPTY-5-INTO-2)
  ANCESTORS  = NIL
```

Farmer:

a) Breadth First Search:

```
Performing breadth first search on problem the farmer, the fox, the goose, and the
grain.
#<SEARCH-STATISTICS {1004752213}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 239
  MAXIMUM-LENGTH-OF-NODE-LIST  = 127
  LENGTH-OF-SOLUTION           = 7
  MAXIMUM-DEPTH                = 8
%
#<NODE {1004817623}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<FARMER-STATE {1004817563}>
  PROBLEM    = #<PROBLEM {10046C4333}>
  PATH       = (FARMER-TAKES-GOOSE FARMER-TAKES-SELF FARMER-TAKES-FOX..
  ANCESTORS  = NIL
```

b) DFS with duplicate node detection:

```
Performing depth first search with duplicate node detection on problem the farmer,
the fox, the goose, and the grain.
#<SEARCH-STATISTICS {100498D723}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 8
  MAXIMUM-LENGTH-OF-NODE-LIST  = 2
  LENGTH-OF-SOLUTION           = 7
  MAXIMUM-DEPTH                = 7
%
#<NODE {100498FD83}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<FARMER-STATE {100498FCC3}>
  PROBLEM    = #<PROBLEM {10046C4333}>
  PATH       = (FARMER-TAKES-GOOSE FARMER-TAKES-SELF FARMER-TAKES-FOX..
  ANCESTORS  = NIL
```

c) DFS with depth limit

```
Performing depth first search with depth limit on problem the farmer, the fox, the
goose, and the grain.
#<SEARCH-STATISTICS {1005E7B0E3}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 8
  MAXIMUM-LENGTH-OF-NODE-LIST  = 4
  LENGTH-OF-SOLUTION           = 7
  MAXIMUM-DEPTH                = 7
%
#<NODE {1005E7C253}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<FARMER-STATE {1005E7C193}>
  PROBLEM    = #<PROBLEM {10046C4333}>
  PATH       = (FARMER-TAKES-GOOSE FARMER-TAKES-SELF FARMER-TAKES-FOX..
  ANCESTORS  = NIL
```

Missionaries and Cannibals

a) Breadth First Search:

```
Performing breadth first search on problem missionaries-and-cannibals.
#<SEARCH-STATISTICS {1003A2E6D3}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 2622
  MAXIMUM-LENGTH-OF-NODE-LIST  = 1262
  LENGTH-OF-SOLUTION           = 11
  MAXIMUM-DEPTH                = 12
%
#<NODE {1003F64EB3}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<SIDE-STATE {1003F64DA3}>
  PROBLEM    = #<PROBLEM {100385D053}>
  PATH       = (TWO-CANN-BOAT-TO-GOAL ONE-CANN-BOAT-TO-START TWO-CANN-BOAT-TO-GOAL..
  ANCESTORS  = NIL
```

b) DFS with duplicate node detection:

```
Performing depth first search with duplicate node detection on problem missionaries-
and-cannibals.
#<SEARCH-STATISTICS {1004EA5F03}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 18
  MAXIMUM-LENGTH-OF-NODE-LIST  = 4
  LENGTH-OF-SOLUTION           = 11
  MAXIMUM-DEPTH                = 11
%
#<NODE {1004EBB3D3}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<SIDE-STATE {1004EBB2C3}>
  PROBLEM    = #<PROBLEM {100385D053}>
  PATH       = (TWO-CANN-BOAT-TO-GOAL ONE-CANN-BOAT-TO-START TWO-CANN-BOAT-TO-GOAL..
  ANCESTORS  = NIL
```

c) DFS with depth limit

```
Performing depth first search with depth limit on problem missionaries-and-cannibals.
#<SEARCH-STATISTICS {10050D7FE3}>
  [standard-object]

Slots with :INSTANCE allocation:
  NODES-VISITED                = 19
  MAXIMUM-LENGTH-OF-NODE-LIST  = 11
  LENGTH-OF-SOLUTION           = 11
  MAXIMUM-DEPTH                = 11
%
#<NODE {10050E4113}>
  [standard-object]

Slots with :INSTANCE allocation:
  STATE      = #<SIDE-STATE {10050E4003}>
  PROBLEM    = #<PROBLEM {100385D053}>
  PATH       = (TWO-CANN-BOAT-TO-GOAL ONE-CANN-BOAT-TO-START TWO-CANN-BOAT-TO-GOAL..
  ANCESTORS  = NIL
```

All 3 problems hang when using function depth-first-search.

Listed is a chart comparing results:

	BFS	**DFS**	**DFS(DND)**	**DFS(DL)**
Water Jug				
Success Running	Y	N	Y	Y
Max depth	7	N/A	7	7
Max Length of Node List	114	N/A	2	4
Nodes visited	239	N/A	12	13
Solution length	6	N/A	6	6
Farmer				
Success Running	Y	N	Y	Y
Max depth	8	N/A	7	7
Max Length of Node List	127	N/A	2	4
Nodes visited	239	N/A	8	8
Solution length	7	N/A	7	7
Missionaries				
Success Running	Y	N	Y	Y
Max depth	12	N/A	11	11
Max Length of Node List	1262	N/A	4	11
Nodes visited	2622	N/A	18	19
Solution length	11	N/A	11	11

As stated already, the general DFS function would hang for all three problems without providing a solution. I always had to kill the current buffer and restart. While I'm sure DFS could find the solution eventually, it would take an extremely long time since the tree is deep while solutions are rare. I had no problem however with the three other search functions, as they solved the problem within milliseconds. While breadth-first-search seemed to have solved the problem quickly, it expanded many more nodes than the depth-first-search-with-duplicate-node-detection and depth-first-search-with-depth-limit. Take for example the water jug problem; BFS expanded 239 nodes while DFSDND and DFSDL expanded 12 and 13 nodes, respectively. This difference is even more apparent with missionaries and cannibals, with BFS exploring 2622 nodes.

By pruning the search trees with a heuristic value for depth limiting or by preventing expansion of duplicate nodes, one can effectively decrease the number of searchable paths, diminishing the shortcomings in a basic algorithm, such as. DFS, while unwieldy on its own, could be improved immensely, resulting in the expansion of far fewer nodes than BFS to arrive at a solution path. This comparison demonstrates that while BFS works fine for these three simple problems, as search space grows, it might be better to consider the 2 other search functions.

Step 4:

Fringes: {S}, {a}, {b,d,e}, {b}, {G}
Expands:

S
S, a
S, a, b, d, e
S, a, b, c, d, G

Returns path s-a-d-G

Step 5:

X: {G, H, J, P, Q, R, S, T, W}
D: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
C:

C_1: $T + T = Q + 10 * X_1$
C_2: $10X_1 + S + S = J + 100 * X_2$
C_3: $100X_2 + R + R = T + 1000 * X_3$
C_4: $1000X_3 + Q + Q = H + 10000 * X_4$
C_5: $10000X_4 + P + W = G + 100000 * X_5$
C_6: <{W} = 1> → max carry is 1
C_7: <{Q} = EVEN> → result of T + T, whether T is even or odd, would be even

Step 6:

Extra Credit: As previously mentioned in step 5 within the constraints, it's important to note that W must be 1 and Q must be even. Because W must be 1, G must be 0, the reason being that

since there was a carryover involved and W is 1, the only possible number that P + W (and X_4) could add up to was 10. With all those in mind, I can list the following possibilities for each letter.

W = {1}
G = {0}
P = {8,9} //only viable numbers to obtain G (0)
Q = {2,4,6,8} //must be even
R, S, T, J, H = {2, 3, 4, 5, 6, 7, 8, 9}

Assuming P = 9, X_5 = 1
Q = {2, 4, 6}
H = {4, 5, 6, 7, 8, 9}
T = {2, 3, 6, 7}
R = {3, 6, 8}

Assuming Q = 2, X_4 = 0
H = {4, 5}
T = {3, 6}
R = {3, 6, 8}

Assuming T = 6, X_1 = 1
R = {3, 8}

Assuming R = 8, X_3 = 1, X_2 = 0
S = {3}
J = {7}
H = {5}

I was able to eventually solve the problem manually.

PQRST	92,836
+ WQRST	+ 12,836
WGHTJQ	105,672