Deep Patel
Intro to AI - HW 2
9/30/18

Problem 1

BFS (FIFO) New nodes at end of Queue, Duplicate nodes removed

- ➔ q = [A]
- ➔ A, q = [B, C, D]
- ➔ A, B, q = [C, D, F]
- ➔ A, B, C, q = [D, E, F]
- ➔ A, B, C, D, q = [E, F]
- ➔ A, B, C, D, E, q = [F]
- ➔ A, B, C, D, E, F, q = []

DFS (LIFO) New nodes at front of Queue, Duplicate nodes removed

- ➔ q = [A]
- ➔ A, q = [D, C, B]
- ➔ A, D, q = [E, C, B]
- ➔ A, D, E, q = [F, C, B]
- ➔ A, D, E, F, q = [C, B]

Uniform Cost (priority queue with smallest cost at front)

- ➔ Q = [A(0)]
- ➔ A, q = [B(3), D(4), C(5)]
- ➔ A, B, q = [C(4), D(4), C(5), F(9)]
- ➔ A, B, C, q = [D(4), ~~C(5),~~ E(6), F(9)]
- ➔ A, B, C, D, q = [~~C(5),~~ E(6), ~~E(7),~~ F(9)]
- ➔ A, B, C, D, E, q = [~~C(5),~~ ~~E(7),~~ F(9)]
- ➔ A, B, C, D, E, F, q = [~~C(5),~~ ~~E(7)~~]

A*

- ➔ Q = [A(0+6)]
- ➔ A, q = [B(3+4), C(5+3), D(4+4)]
- ➔ A, B, q = [C(4+3), C(5+3), D(4+4), F(9+0)]
- ➔ A, B, C, q = [E(6+1), C(5+3), D(4+4), F(9+0)]
- ➔ A, B, C, E, q = [F(8+0), C(5+3), D(4+4), F(9+0)]
- ➔ A, B, C, E, F, q = [C(5+3), D(4+4), F(9+0)]

Problem 2

Cycle:

1. BFS Tree Search: O(n) because BFS adds the neighbors of x_1, one on each direction (i.e. x_{n-1} and x_2). Then it pops them from the queue and added their neighbors, which are on either direction of x1 (i.e. x_{n-2} and x_3). BFS continues to explore vertices one by one until the 2k nodes are explored, k on either direction of x_1. In the worst case, k will be n/2, so n/2 +n/2 = n vertices will be visited.
2. BFS Graph Search: O(n). This is the same as BFS tree search because having a history of what vertices are explored does not help because there are no intermediate cycles in the graph, besides the cycle as a whole.
3. DFS Tree Search: O(∞) because DFS will search in one direction from x1 and keep adding vertices until a vertex has no neighbors. In the case of tree search, we have no history of what neighbors are already visited, so DFS will continue to traverse the cycle forever because it is trying to go as deep as possible.
4. DFS Graph Search: O(n). Unlike in tree search, in graph search we have a history of what neighbors have been visited. So, similarly as before, DFS will traverse the entire cycle, but will stop when it comes back to x_1. Thus, the maximum number of vertices that will be processed before reaching x_1 is n.

Infinite Diamond Strip (IDS):

5. BFS Tree Search: $O(2^{2m})$ because in the first level, 4 neighbors of vertex x_1 will be added to the queue. In the next level, each of the 4 vertices will add 1 neighbor each, but because the graph is diamond shaped, it is really just 2 instances of the same 2 vertices. This happens in tree search because no history of the visited vertices is kept. Following this pattern, the number of vertices visited doubles at every other level. To reach a distance of 2m in the right direction, the number of vertices added to the queue is 2^{2m-1}.
6. BFS Graph Search: O(m) because unlike in tree search, the number of vertices does not double every other level, because each vertex is only added once to the queue because we can see that it is visited. This, at every odd level, 4 vertices will be added and at every even level, 2 vertices will be added. So, for every 2 levels, a total of 6 vertices will be added and to reach a distance of 2m, 6/2*2m = 6m vertices will be added to the queue.
7. DFS Tree Search: O(∞) because the graph is infinite in either direction and DFS will continue to search until the deepest level is found, which is neverending
8. DFS Graph Search: O(∞). Same logic as tree search. Having memory of visited vertices does not help, because the graph is infinite and the algorithm will always find new vertices.

Problem 3

Assume $h(x) \leq c(x,a,x') + h(x')$ and $h(xg) = 0$
Prove that $h(x) \leq C^*x$ where C^*x is optimal cost from x to x_g

Proof:
$h(n) \leq h(n_1) + c(n, a, n_1)$
$h(n) \leq h(n_2) + c(n_1,a,n_2) + c(n,a,n_2)$
.
.
.
$h(n) \leq c(n, a, n_2) + c(n_1, a, n_2) + a(n_2, a, n_3) + \ldots + c(n_{n-1},a,n_n)$
$h(n) \leq C^*x$
Thus, this proves that if a heuristic is consistent, it is also admissible.

Problem 4

1. For a heuristic to be admissible, $h(n) \leq h^*(n)$, which is the actual cost to goal, i.e. the heuristic should never be an overestimation compared to the actual cost.
2. For a heuristic to be consistent, $h(n) \leq c(n,a,n') + h(n')$, i.e. the heuristic from a point to the goal should never be more than the cost to an intermediate point and the heuristic from that point to the goal.

i) $h_{min}(n) = min\{h_1(n), h_2(n)\}$

If h_1 and h_2 and admissible then **h_{min} is admissible**, since
$h_1(n) \leq h^*(n)$ or $h_2(n) \leq h^*(n)$ and we can deduce this to $\min(h_1(n), h_2(n)) \leq h^*(n)$

If h_1 and h_2 are consistent then **h_{min} is also consistent**, since:
$h_{min}(n) = \min\{h_1(n), h_2(n)\}$
$\leq \min\{c(n,a,n') + h_1(n'), c(n,a,n') + h_2(n')\}$
$\leq c(n,a,n') + \min\{h_1(n'), h_2(n')\}$
$\leq c(n,a,n') + h_{min}(n')$

ii) $h_{max}(n) = max\{h_1(n), h_2(n)\}$

If h_1 and h_2 and admissible then **h_{max} is admissible**, since
$h_1(n) \leq h^*(n)$ and $h_2(n) \leq h^*(n)$ and we deduce this to $\max(h_1(n), h_2(n)) \leq h^*(n)$

If h_1 and h_2 are consistent then **h_{min} is also consistent**, since:
$h_{max}(n) = \max\{h_1(n), h_2(n)\}$

$\leq \max\{c(n,a,n') + h_1(n'), c(n,a,n') + h_2(n')\}$
$\leq c(n,a,n') + \max\{h_1(n'), h_2(n')\}$
$\leq c(n,a,n') + h_{max}(n')$

iii) $h_{lin}(n) = wh_1(n) + (1-w)h_2(n)$, $0 \leq w \leq 1$

We know that:
$h_1(n) \leq h^*(n)$ and,
$h_2(n) \leq h^*(n)$
Then,
$h_{lin}(n) = wh_1(n) + (1-w)h_2(n)$
$\leq wh^*(n) + (1-w)h^*(n)$
$\leq (w+1-w)h^*(n)$
$\leq h^*(n)$
Thus, **$h_{lin}(n)$ is also admissible**

In the same way, we can prove that $h_{lin}(n)$ is consistent if $h_1(n)$ and $h_2(n)$ are consistent:
We know that:
$h_1(n) \leq c(n,a,n') + h_1(n')$
$h_2(n) \leq c(n,a,n') + h_2(n')$
Then,
$h_{lin}(n) = wh_1(n) + (1-w)h_2(n)$
$\leq w[c(n,a,n') + h_1(n')] + (1-w)[c(n,a,n') + h_2(n')]$
$\leq (w+1-w)[c(n,a,n')] + wh_1(n') + (1-w)h_2(n')$
$\leq c(n,a,n') + h_{lin}(n')$
Thus, **$h_{lin}(n)$ is also consistent.**

Problem 5

If h(n) is admissible, the given function is proven to be optimal when
f(n) =(2−w)[g(n) + w*h(n) / (2−w)]
Which behaves like A* with a heuristic
f(n) = g(n) + w*h(n) / (2−w)]
So, to be optimal, w/(2-w) <= 1
Meaning for **w <= 1,** the algorithm is guaranteed to be optimal when h is admissible
For **w = 0**: f(n) = 2*g(n) ---> **Uniform-cost search**
For **w = 1**: f(n) = g(n) + h(n) ---> **A***
For **w = 2**: f(n) = 2*h(n) ---> **Greedy Best Search**

Problem 6

Iteration 0: # of attacks = 4. Lowest value is 3, so move leftmost Queen up 1.

3	Q	3	3
Q	3	5	3
3	5	3	Q
3	3	Q	3

Iteration 1: # of attacks = 3. Lowest value is 1, so move rightmost queen up 1.

Q	Q	4	3
4	2	4	1
3	3	4	Q
3	2	Q	3

Iteration 2: # of attacks = 1. Lowest value is 0, so move leftmost queen down by 2 to the 0.

Q	Q	4	3
3	2	3	Q
0	1	3	3
1	2	Q	3

Iteration 3: # of attacks = 0. This is the solution.

1	Q	2	1
3	2	2	Q
Q	2	2	3
1	2	Q	1

Problem 7 :

h_1 is the # of misplaced tiles. If a tile is misplaced, then it needs to be moved atleast 1 time. Thus, the number of tiles misplaced will never be less than the number of moves needed to complete the problem.

h_2 is the Total Manhattan Distance. Tiles are only allowed to move up/down or left/right and not diagonally. Thus, the Manhattan distance from start to target location represents the minimum number of moves a tile must perform before it and all the other tiles reach their target location. Thus, the Manhattan Distance will always be upper bounded by the actual number of moves, making it a valid heuristic.