More Notes on Search for 16.410 and 16.413

Uniform Cost Search

Q is a priority queue sorted on the current cost from the start to the goal.

```
UNIFORM_COST(start, goal, cost)
  Q = init_queue()
  push Q, start
  while ! empty(Q)
    current = pop_min Q
    if (current contains goal)
      return current
    foreach child in children(current)
      push Q, child using value[current] + cost(current, child)
```

Space and time complexity: $O(b^{1+\lfloor C^\star/\epsilon \rfloor})$ where $C^\star$ is the cost of the optimal solution, ϵ is the minimum cost of any arc, and b is the branching factor.

Dijkstra's Algorithm

```
INIT_NODES(value, parent)
  for each state n
    value[n] = infinity
    parent[n] = null

DIJKSTRA(goal)
  INIT_NODES(value, parent)
  value[goal] = 0
  for each state
    push Q, state using value[state]
  while ! empty(Q)
    current = pop_min Q
    for each child in children(current)
      RELAX (current, child, value, cost, Q)

RELAX (current, child, value, parent, cost, Q)
  if (value[child] > value[current] + cost[current, child])
     value[child] = value[current] + cost[current, child]
     update Q, child using value[child]
     parent[child] = current
```

Complexity: $O(n^3)$ for n states if the graph is dense $O(e \log n)$ for n states and e edges if the graph is sparse

Informed Search - A* Search

In A* search, we order the search based on

$$f(state) = g(state) + h(state) \tag{1}$$

where $g(n)$ is the current cost from the start to n, and $h(n)$ is the current heuristic estimate of the cost from n to the goal. $f(state)$ is the estimate of the lowest-cost path that goes from start to the goal through n.

If we represent the search state as the path from the start to state n, then we can compute $g(n)$ directly from the search state, otherwise we have to follow parent pointers from n back to the start to compute $g(n)$, but in both cases we know $g(n)$ exactly.

A* search involves the notion of *admissibility*. An admissible heuristic $h(n)$ is a function that is never greater than the actual cost from n to the goal – that is, it never overestimates the cost.

`Q` is a priority queue sorted on the A* cost function $f(state)$.

```
INIT_NODES(value, parent)
  for each node n
    value[n] <- infinity
    parent[n] <- null

A*-COST(node)
  return value[node] + heuristic(node)

A*(start, goal, costs)
 INIT_NODES(value, parent)
 value[start] = 0
 push Q, start
 while ! empty(Q)
   current <- pop_min Q
   if (current contains goal)
     return current
   for each child in children(current)
       if value[child] > value[current] + cost(current, child)
          value[child] = value[current] + cost(current, child)
         push Q, children(current) using A*-COST(child)
```

Space and time complexity: $O(b^m)$ where b is the branching factor and m is the maximum depth of the search space.

Things you should know:

- Breadth-first, depth-first, iteratively-deepening and A* search
- Complexities in time and space
- Bi-directional search
- Relationship between dynamic programming and search

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?						
Time						
Space						
Optimal?						