A^* Search Algorithm

28/07/19

- informed, guided search
- uses heuristics
- selects path with least cost at each iteration

$$f(n) = g(n) + h(n)$$

- $f(n)$: total cost of path if node "n" is chosen
- $g(n)$: cost from starting node to node "n"
- $h(n)$: heuristic function

Objective: $\underset{n}{\operatorname{argmin}}\ f(n)$

- A^* returns least cost path iff heuristic is admissable (never overestimates actual cost to get to goal)
- implemented using priority queue
- Dijkstra can be seen as a special case of A^* where $h(n) = 0\ \forall$ nodes
- time complexity is polynomial iff:
  - there is a single goal state
  - search space is a tree
  - $|h(x) - h^*(x)| = O(\log h^*(x))$
    {h^* is the optimal heuristic}

Algorithm

```
procedure A*(source, destination, h):
    pqueue := {}  // priority queue
    predecessor := nil  // mapping from node to node

    g := {∞ for all nodes}
    g[source] = 0
    f := {∞ for all nodes}
    f[start] = h(start)

    while pqueue is non empty:
        current = pqueue.extract_min()
        if current == goal:
            return reconstruct_path(predecessor,
                                    current)

        pqueue.remove(current)
        for each neighbour of current:
            g' := g[current] + d(current, neighbour)

            if g' < g[current]:
                predecessor[neighbour] = current
                g[neighbour] = g'
```

```
            f[neighbour] = g' + h(neighbour)

            if neighbour not in pqueue:
                pqueue.add(neighbour)

    return error

procedure reconstruct_path(predecessor, current):
    path := {current}

    while current in predecessor:
        current = predecessor[current]
        path = [current] + path

    return path
```