Summary of the A* Algorithm

The A-Star algorithm maintains two sets, the OPEN list and the CLOSED list. The OPEN list keeps track of those nodes that *need to be* examined, while the CLOSED list keeps track of nodes that *have already been* examined. Initially, the OPEN list contains just the initial node, and the CLOSED list is empty. Each node n in the graph maintains the following additional information:

- `g(n) = the cost of getting from the initial node to n.`
- `h(n) = the estimate, according to the heuristic function, of the cost of getting from n to the goal node.`
- `f(n) = g(n) + h(n); intuitively, this is the estimate of the best solution that goes through n.`

Each node also maintains a pointer to its parent, so that later we can retrieve the best solution found, if one is found.

A-Star has a main loop that repeatedly gets the node, call it n, with the lowest f(n) value from the OPEN list (in other words, the node that we think is the most likely to contain the optimal solution). If n is the goal node, then we are done, and all that's left to do is return the solution by backtracking from n. Otherwise, we remove n from the OPEN list and add it to the CLOSED list. Next, we generate all the possible successor nodes of n (the action set U(n)). For each successor node n', if it is already in the CLOSED list and the copy there has an equal or lower f estimate, then we can safely discard the newly generated n' and move on (we can do this since a copy with a better estimate on the CLOSED list means we've already looked at it, and the new copy won't do any better). Similarly, if n' is already in the OPEN list and the copy there has an equal or lower f estimate, we can discard the newly generated n' and move on (we're going to be looking at a better version of n' later, so no need to keep this one around).

If no better version of n' exists on either the CLOSED or OPEN lists, we remove the inferior copies from the two lists and set n as the parent of n'. We also have to calculate the cost estimates for n' as follows: set g(n') to g(n) plus the cost of getting from n to n'; set h(n') to the heuristic estimate of getting from n' to the goal node; and set f(n') to g(n') plus h(n'). Lastly, add n' to the OPEN list and return to the beginning of the main loop

In pseudo-code, A-Star looks like this:

Algorithm A-Star

```
Initialize OPEN list (to the empty list)
Initialize CLOSED list (to the empty list)
Create goal node; call it node_goal
Create start node; call it node_start
Add node_start to the OPEN list

while the OPEN list is not empty
{
   Get node n off the OPEN list with the lowest f(n)
   Add n to the CLOSED list
   IF n is the same as node_goal
        we have found the solution; return Solution(n)
   ELSE: Generate each successor node n' of n

   for each successor node n' of n
      {
        Set the parent of n' to n
        Set h(n') to be the heuristically estimate distance to node_goal
        Set g(n') to be g(n) plus the cost to get to n' from n
        Set f(n') to be g(n') plus h(n')

        if n' is on the OPEN list and the existing one is as good or better
               then discard n' and continue
        if n' is on the CLOSED list and the existing one is as good or better
               then discard n' and continue
        Remove occurrences of n' from OPEN and CLOSED
        Add n' to the OPEN list
      }
}

return failure (if we reach this point, we’ve searched all reachable nodes and still
haven’t found the solution, therefore one doesn’t exist)
```

The Heuristic Function

The success of A-Star rests heavily on the heuristic function chosen. For any given problem that we may wish to apply A-Star to, there are good heuristics and bad heuristics. A good one will allow the algorithm to run quickly and find the optimal solution. A bad one may just increase the running time. Or it may be so bad that it misleads the algorithm into returning sub-optimal solutions or not find solutions at all.

To guarantee that we will find the optimal solution, if one exists, the heuristic must be “admissible.” To be admissible, a heuristic must never over-estimate the cost of getting from a state to the goal state. It’s easy to see why this is true: an over-estimating heuristic may think the cost of the optimal solution is higher than it really is, higher than some other solution’s estimated cost, and therefore make the algorithm pick a sub-optimal solution over the optimal one.

Another issue is how quickly the heuristic function can be computed. There’s almost always a trade-off between the accuracy of the heuristic and the time it takes to compute its estimates. It’s nice when the heuristic is very accurate, as it has been shown that when using a perfectly accurate heuristic, A-Star will expand exactly the nodes on the solution path and return the optimal solution. However, a perfect heuristic is almost never available, and to even come close requires significant additional computation. A lot of the time, a heuristic that only comes close to a perfect estimate, but runs very quickly, is superior to one that is perfect but takes forever to do so. For example, in the context of computer game path-finding, you don’t really need the absolute best path to some destination; you just want a good path quickly. When choosing the heuristic function for a specific implementation of A-Star, one should always think carefully about whether speed or accuracy is more valuable in the context of the problem.