Searching paths whose link lengths vary

Task: Find shortest route to the goal.

A good method: Follow route we think is shortest

This is the strategy of A*:

- At any N along the route, guess total path length by adding:
  - Length of shortest route found so far from start point to N.
  - A heuristic distance function $h$ for distance from N to goal.

- Require that $h(N)$ is never greater than the real distance, so that we won’t ignore good paths because of pessimistic estimates.

- Continue searching even after finding the goal, as long as our estimates suggest that other paths are better than the ones found so far.
Examples of $h$ for driving distance

- Straight-line distance on a map.

- The distance on a distance table (your distance may be worse if you don’t know the roads very well).

- Estimate mileage by how long your dare-devil roommate says each route should take.

Other criteria for cost: danger, views, time (which may not match distance)
Sketch of $A^*$

Assume we are given a function $h(n)$ to estimate cost to the goal.

We will maintain:

- An array $u(n)$ giving the cost to reach $n$ from the start state.
- An array $p(n)$ recording a pointer to the best (cheapest) node from which it was reached.
- A set OPEN of pending nodes
- A set CLOSED of nodes already reached
Search process

1. Set current element \( N \) to initial state

2. Is \( N \) goal? If so
   
   - If for each open node, the estimated cost of proceeding through that node is greater than the cost of the shortest path you’ve found to goal, stop.

3. Generate successors \( S_i \) of \( N \). For each \( S_i \),
   
   - If \( S_i \) is not already in OPEN or CLOSED,
     
     \[
     u(S_i) \leftarrow u(N) + \text{distance}(N, S_i).
     \]
     
     Set \( p(S_i) \) to point to \( N \).

   - If \( S_i \) is already on the OPEN or CLOSED list,
     
     \[
     u(S_i) \leftarrow \min\{u(N) + \text{distance}(N, S_i), u(S_i)\}.
     \]
     
     If new path to \( S_i \) is cheaper, update \( p(S_i) \).

     Replace \( N \) with its successors on OPEN. Put \( N \) in CLOSED.

4. As the next \( N \), take the open node \( S \) with the lowest \( u(S) + h(S) \). Go to 2
A* to find the shortest route Bloomington-Indy

Initial state:

Open nodes:

<table>
<thead>
<tr>
<th>City</th>
<th>Tot. dist. B→city</th>
<th>Est. dist city →Indy</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloomington</td>
<td>0</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

Pick the node with the best total: Bloomington. Replace it with successors not yet visited: Louisville, Bedford, Spencer, Martinsville.
Step 2:

Open nodes:

<table>
<thead>
<tr>
<th>City</th>
<th>Tot. dist. B→city</th>
<th>Est. dist. city →Indy</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Louisville</td>
<td>120</td>
<td>160</td>
<td>280</td>
</tr>
<tr>
<td>Bedford</td>
<td>25</td>
<td>45</td>
<td>70</td>
</tr>
<tr>
<td>Spencer</td>
<td>25</td>
<td>60</td>
<td>85</td>
</tr>
<tr>
<td>Martinsville</td>
<td>20</td>
<td>27</td>
<td>47</td>
</tr>
</tbody>
</table>

Pick node with the best total: Martinsville.

Replace it with successors not yet visited: Chicago, Mooresville
Step 3:

Open nodes:

<table>
<thead>
<tr>
<th>City</th>
<th>Tot. dist. B→city</th>
<th>Est. dist. city →Indy</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>280</td>
<td>200</td>
<td>480</td>
</tr>
<tr>
<td>Mooresville</td>
<td>30</td>
<td>18</td>
<td>48</td>
</tr>
<tr>
<td>Louisville</td>
<td>120</td>
<td>160</td>
<td>280</td>
</tr>
<tr>
<td>Bedford</td>
<td>25</td>
<td>45</td>
<td>70</td>
</tr>
<tr>
<td>Spencer</td>
<td>25</td>
<td>60</td>
<td>85</td>
</tr>
</tbody>
</table>

Pick node with the best total: Mooresville.

Replace it with successors not yet visited: Indianapolis.
Step 4:

Open nodes:

<table>
<thead>
<tr>
<th>City</th>
<th>Tot. dist. B→city</th>
<th>Est. dist city →Indy</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indianapolis</td>
<td>50</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>Chicago</td>
<td>280</td>
<td>200</td>
<td>480</td>
</tr>
<tr>
<td>Louisville</td>
<td>120</td>
<td>160</td>
<td>280</td>
</tr>
<tr>
<td>Bedford</td>
<td>25</td>
<td>45</td>
<td>70</td>
</tr>
<tr>
<td>Spencer</td>
<td>25</td>
<td>60</td>
<td>85</td>
</tr>
</tbody>
</table>

Pick node with the best total: Indianapolis.

No estimates are cheaper: done!