A*

One pair, single-source, vs all pairs shortest path problem

One pair: We need the shortest path from s to f

Single source: We need the shortest path from s to all other nodes

All pairs: We need the shortest path between all pairs of nodes.

Dijkstra's algorithm solves the single source problem.

- We can use it to solve the all-pairs problem by repeating it for each node — but there are better methods.
- We can use it to solve the one pair problem.
  - $*$ Start at s . Stop as soon as f is black.

More information

Now suppose

- nodes represent points on a 2-D plane —e.g. locations on a map— or in a 3D space (e.g. rooms in a building)
- the edge weights are distances.

These edge weights need not be Euclidean (as the crow flies) distances ($\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$) (Perhaps the road is not straight).

No worm holes. We can assume however that $t(u,v) \geq h(u,v)$ where

$$h(u,v) = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$$

and $t(u,v)$ is the length of the shortest path from u to v.

Dijkstra's algorithm's search spreads out in all directions equally.

- However it seems intuitively sensible to search first nodes that are "in the right direction".
- E.g. if I use Dijkstra's algorithm to find the shortest driving route from Toronto to Montreal, it will visit Waterloo before Kingston.

A* is a variation on Dijkstra's that makes use of the heuristic "try nodes that seem to be in the right direction earlier".

A*

The problem: find a shortest path from s to f in a graph $G = (V, E)$ using a heuristic function h that estimates the distance to f. I.e. $h(v)$ is an estimate of the distance from v to f.

Triangle inequality: We require that for any edge (u,v),

$$h(u) \leq w(u,v) + h(v) \tag{1}$$

Suppose there is a path p from u to v with a cost of $w(p)$. By induction on path length we have

$$h(u) \leq w(p) + h(v) \tag{2}$$

For the driving application, the Euclidean distance makes a suitable heuristic.

Recall that $t(u)$ is the minimal cost of travelling from s to u.

We just use Dijkstra's algorithm, except:

- We use $t(u) + h(u)$ in place of $t(u)$.
- Thus $d(u)$ approaches $t(u) + h(u)$.
  - $*$ We only turn a node u black when $d(v) = t(v)+h(v)$.
- We stop when f is black and hence (if $h(f) = 0$) $d(v) = t(v)$.

Thus the invariants (replacing DI1 and DI2) are

- A*I1 : For each black node, v, $d(v) = t(v) + h(v)$.
- A*I2 : For each grey node, v, $d(v) - h(v)$ is the length of some path from s to v.

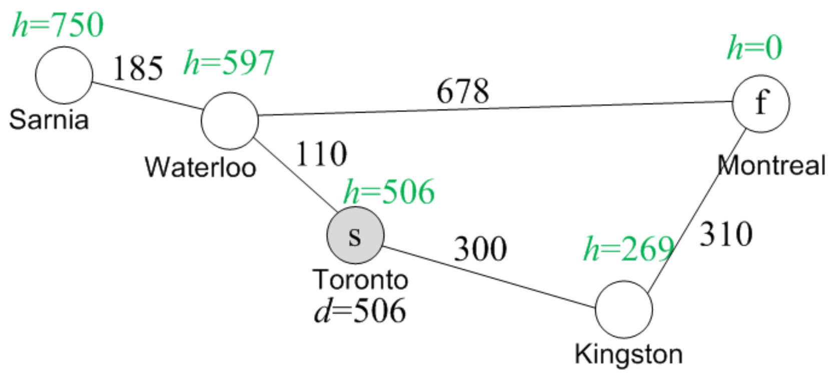
```
for $v \leftarrow V$ do
    $\pi(v) := \mathrm{null}$      $\mathrm{colour}(v) := \mathrm{white}$      $d(v) := \infty$
end for
var $PQ : \mathrm{PriorityQueue} := \mathrm{new\ PriorityQueue}$
$PQ.\mathrm{add}(s, \underline{h(s)})$      $\mathrm{colour}(s) := \mathrm{grey}$      $\underline{d(s) := h(s)}$
while $\neg PQ.\mathrm{empty}()$ and $\mathrm{colour}(f) \neq \mathrm{black}$ do
    val $u := PQ.\mathrm{removeLeast}()$
    { $u$ has the smallest $d$ value of all grey nodes }
    $\mathrm{colour}(u) := \mathrm{black}$
    for $v \mid u \rightarrow v$ do
        $\underline{\text{val } d := d(u) - h(u) + w(u, v) + h(v)}$
        if $\underline{d(v) > d}$ then
            { $v$ is not black }
            $\underline{d(v) := d}$      $PQ.\mathrm{add}(v, d)$      $\pi(v) := u$
            $\mathrm{colour}(v) := \mathrm{grey}$ end if end for end while
```

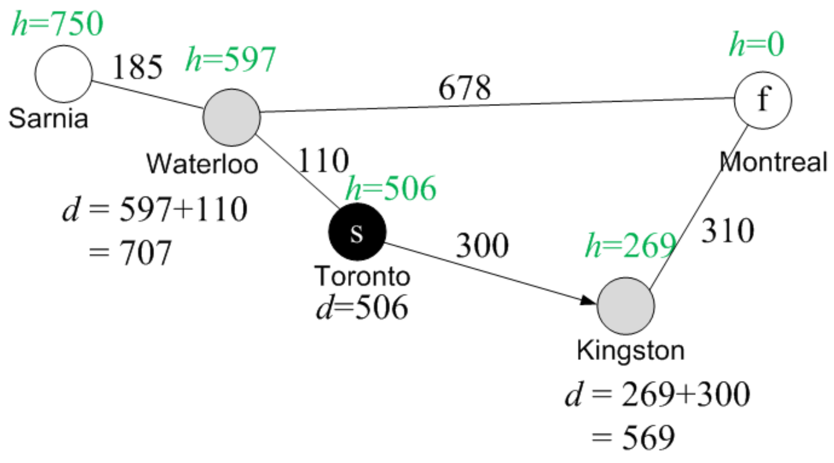
At termination: If $\mathrm{colour}(f) = \mathrm{black}$, $d(f) = t(f) + h(f) = t(f)$ is the length of a shortest path from s to f and

$$s, \ldots, \pi(\pi(f)), \pi(f), f$$

is such a path. If $\mathrm{colour}(f) \neq \mathrm{black}$ then there is no path from s to f.

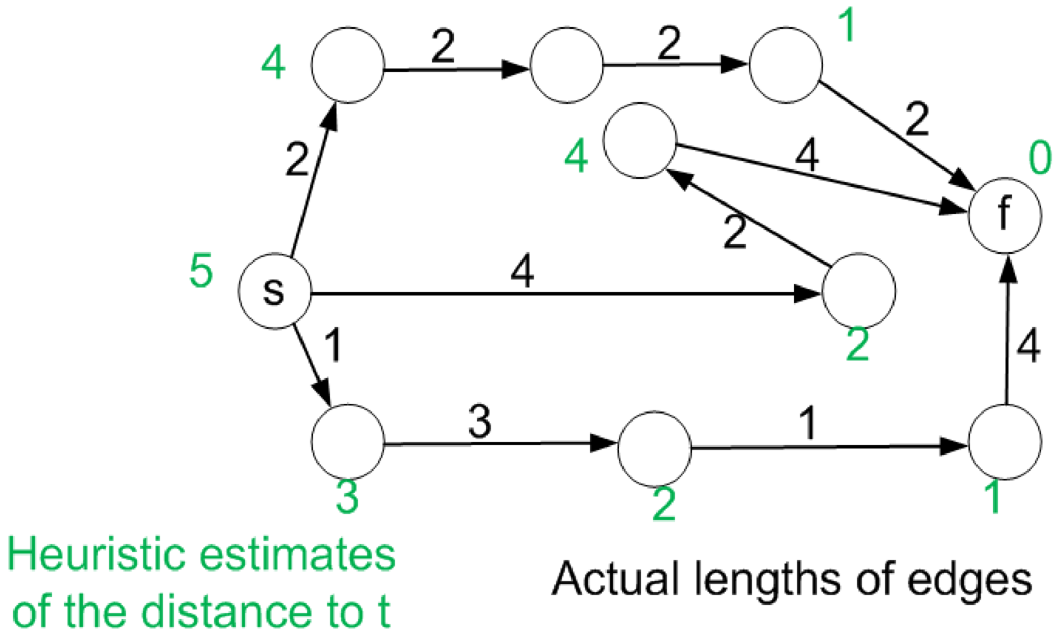
Here is a simple example. Dijkstra's algorithm visits: Toronto, Waterloo, Sarnia, Kingston, Montreal. It must visit Sarnia before Kingston, since there could be a Sarnia-Montreal edge of length 1km that would beat any path through Kingston.



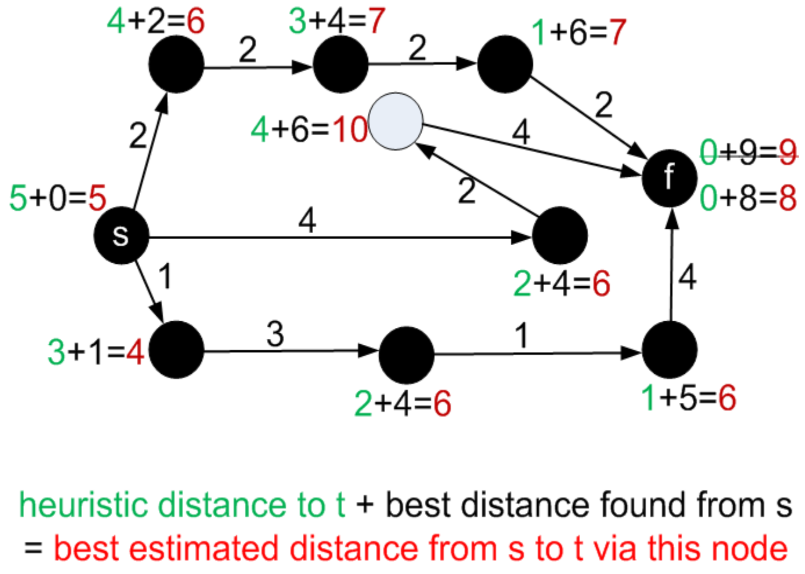


A* visits Kingston, 2nd. At that time Montreal and Waterloo are grey. The d value for Montreal is 610 and for Waterloo 707. So Montreal is visited third and the algorithm stops.

Here is an example. Try hand simulating the algorithm on it.



Here is what I got.


4+2=6
2
3+4=7
2
1+6=7
2
4+6=10
4
~~0+9=9~~
0+8=8
f
2
5+0=5
s
4
2+4=6
4
1
3+1=4
3
1
2+4=6
1+5=6
heuristic distance to t + best distance found from s
= best estimated distance from s to t via this node

- A*I1 : For each black node, v, $d(v) = t(v) + h(v)$.
- A*I2 : For each grey node, v, $d(v) - h(v)$ is the length of some path from s to v.

Lemma: If A*I1 and A*I2 hold, then, for any w and any optimal path from s to w, the first grey node v on the path (if any) has $d(v) = t(v) + h(v)$.

Proof. If v is s, then $d(v) = h(s) = t(s) + h(s)$.

If v is not s, the predecesor u of v on the path is black and so (by A*I1) $d(u) = t(u) + h(u)$.

Furthermore, when u was visited, the edge from u to v would have been considered and so $d(v) \leq d(u) - h(u) + w(u, v) + h(v)$, or $d(v) \leq t(u) + w(u, v) + h(v)$.

Since (u, v) is on an optimal path, $t(v) = t(u) + w(u, v)$. So $d(v) \leq t(v) + h(v)$.

By AI*2, $d(v) - h(v)$ is the length of some actual path, so $d(v) - h(v) \leq t(v)$, and, thus, $d(v) \geq t(v) + h(v)$.

So $d(v) = t(v) + h(v)$.

- A*I1 : For each black node, v, $d(v) = t(v) + h(v)$.
- A*I2 : For each grey node, v, $d(v) - h(v)$ is the length of some path from s to v.
- **Lemma**: If A*I1 and A*I2 hold, then, for any w and any optimal path from s to w, the first grey node v on the path (if any) has $d(v) = t(v) + h(v)$.

A*I1 is preserved. Suppose that A*I1 and A*I2 hold, but, at line $\text{colour}(u) := \text{black}$, $d(u)$ does not reflect a shortest path ($t(u) + h(u) < d(u)$) i.e. A*I1 is about to be broken.

Consider the first grey node u' on an optimal path p from s to u.

By the lemma, it must be that $d(u') = t(u') + h(u')$

Consider the part of p that runs from u' to u; call it p'; by the optimality of p, we have, $t(u) = t(u') + w(p')$, or $w(p') = t(u) - t(u')$.

By triangle inequality (2), $h(u') \le w(p') + h(u)$, so $h(u') \le t(u) - t(u') + h(u)$, or $h(u') + t(u') \le t(u) + h(u)$.

All together we have.

$$d(u') = t(u') + h(u') \le t(u) + h(u) < d(u)$$

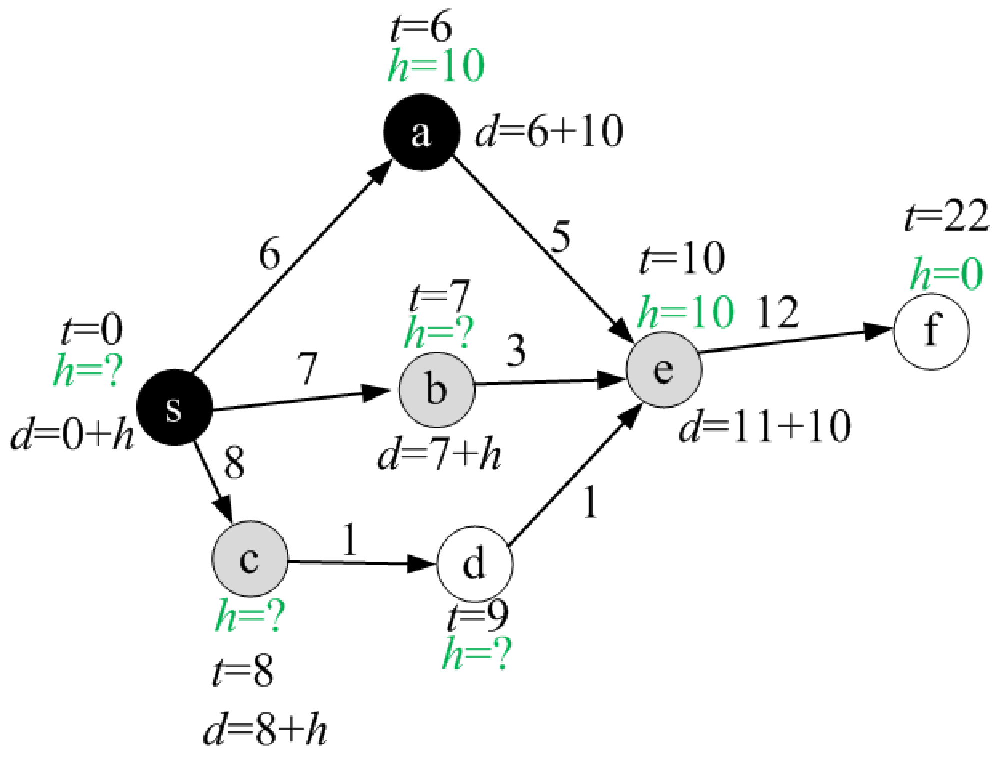
But this is impossible since u' would have priority over u and so u wouldn't have been picked on the previous line.

A*I2 is preserved. Exercise.

Why A* works.

We will try to pick h values that break the algorithm.

Suppose the algorithm is in the following state and about to turn e black, even though the best route to it has not yet been considered. I.e. e is ahead (or tied with) b and c in the p.q.



From the constraint on the heuristic (1), we know $h(\mathrm{b}) \le w(\mathrm{b},\mathrm{e}) + h(\mathrm{e})$ and so $h(\mathrm{b}) \le 13$.

But then $d(b) \le 20$ and so b will be ahead of e in the p.q. Contradiction.

From the constraint on the heuristic (1), we know $h(\mathrm{c}) \le w(\mathrm{c},\mathrm{d}) + w(\mathrm{d},\mathrm{e}) + h(\mathrm{e})$ and so $h(\mathrm{c}) \le 12$.

But then $d(\mathrm{c}) \le 20$ and so c will be ahead of e in the p.q. Contradiction, again.

The constraint

$$h(u) \leq w(u, v) + h(v)$$

on h makes it impossible to pick h values that cause nodes to turn black before a best route to them has been found. Thus when f turns black we have found the best path to it.

If the heuristic is not conservative (does not obey the constraint), A* can still be used, but it may fail to find the shortest path.

E.g. if we want to minimize gas consumption, a good heuristic is great-circle distance * average gas consumption per km. However this may fail to find a route that is all down-hill. (A conservative heuristic would be great-circle distance * minimum gas consumption per km)

At an extreme if the h values are so large that they swamp out the weights, the algorithm degenerates to a "best first search".

- Best first $d(v) = h(v)$
- Dijkstra $d(v)$ = the weight of the best path from s to v found so far.
- A* $d(v) = h(v)+$ the weight of the best path from s to v found so far.