

Re: Python Programming Contest

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2005-07/msg02335.html>

- *From:* Tom Anderson <twic@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Sun, 17 Jul 2005 01:16:43 +0100
-

On Sat, 16 Jul 2005, George Sakkis wrote:

"Tom Anderson" <twic@xxxxxxxxxxxxxxxxxxx> wrote:

On Sat, 16 Jul 2005, Joseph Garvin wrote:

Someone correct me if I'm wrong -- but
isn't this the Shortest Path problem?

Dang! I was just about to point that out.

[snipped]

But yes, this is basically about who can write the
fastest implementation of Dijkstra's algorithm.
I've got one somewhere - i have a half-finished
journey planner for the London Underground! - so
maybe i should enter ...

Hmm. Actually, Dijkstra's algorithm isn't always
the fastest way to find the shortest path. The
thing is, it's fully general, so it works on
absolutely any graph; if the graph you're
traversing has particular properties, you might be
able to leverage those to find a solution faster.
[snipped]

Yes, that's right. Moreover, Dijkstra's computes the shortest
path from a given start to **all** nodes in the network, which
is usually an overkill if all you want is the shortest path to

Re: Python Programming Contest

one (or a few) node(s).

Actually, it only computes shortest paths from the source to every node which is closer than the destination. Unless you keep computing after you've found your route!

I can't immediately see any properties of this network that could be exploited, but that doesn't mean there aren't any.

Hints:

- You have to take exactly one decision (flight or stop) every single day until you reach the destination; no more, no less.
- There is no time machine; days pass in one direction only, one at a time.

Ah, but in my approach, those rules are encoded in the structure of the graph; i don't think they leave you with a graph with any obvious exploitable properties.

To expand on that, rather than making a graph in which vertices are cities and edges are services, which then leaves me with the headache of finding a route in the face of shifting availability and weight of edges, i make each vertex a spacetime, rather than a purely spatial, location - 'Frankfurt on friday', for example, would be a different vertex to 'Frankfurt on monday'. Flights are then simply edges which link the origin city on the day they depart to the destination city on the next day; their weight is the cost of the flight. City X on day N is always linked to City X on day N+1 by an edge representing the option of staying in a hostel. Vertices corresponding to the destination city on any day are all linked to a special goal vertex by length-0 edges, so if you can make it to the destination at any time, you win. You could put nonzero weights on those edges if you liked, to capture your preferences about when you wanted to arrive (if you were willing to pay 15 pounds more to get there on tuesday, you'd set the weight of links from tuesday to the goal to -15, for example). You'd probably also want a special start node linked to your home airport on every day by length-0 edges.

I can't see any properties of the resulting graph that are particularly interesting. The fact that flight prices are not correlated with physical distance makes an A*-like approach impossible.

Actually, there is one thing - the number of edges you need to traverse to get from any vertex to any other (apart from the start and goal

Re: Python Programming Contest

vertices) is always the same, regardless of the route taken, since each edge corresponds to one day's actions, and any given pair of vertices are a certain number of days apart. Ah, is that what you were hinting at? I'm still not sure how you could use this, though!

tom

--

I do not think we will have to wait for very long.

.