24.4 Summary

Dijktra's Algorithm and Single-Source Shortest Paths. Suppose we want to record the shortest paths from some source to every single other vertex (so that we can rapidly found a route from s to X, from s to Y, and so forth). We already know how to do this if we're only counting the number of edges, we just use BFS.

But if edges have weights (representing, for example road lengths), we have to do something else. It turns out that even considering edge weights, we can preprocess the shortest route from the source to every vertex very efficiently. We store the answer as a "shortest paths tree". Typically, a shortest paths tree is stored as an array of edgeTo[] values (and optionally distTo[] values if we want a constant time distTo() operation).

To find the SPT, we can use Dijkstra's algorithm, which is quite simple once you understand it. Essentially, we visit each vertex in order of its distance from the source, where each visit consists of relaxing every edge. Informally, relaxing an edge means using it if its better than the best known distance to the target vertex, otherwise ignoring it. Or in pseudocode:

```
Dijkstra(G, s):
    while not every vertex has been visited:
        visit(unmarked vertex v for which distTo(v) is minimized)
```

Where visit is given by the following pseudocode:

```
visit(v):
    mark(v)
    for each edge e of s:
        relax(e)
```

And finally, relax is given by:

```
relax(e):
    v = e.source
    w = e.target
    currentBestKnownWeight = distTo(w)
    possiblyBetterWeight = distTo(v) + e.weight
    if possiblyBetterWeight < currentBestKnownWeight
        Use e instead of whatever we were using before
```

Runtime is $O(V * logV + V * logV + E * logV)$, and since $E > V$ for any graph we'd run Dijkstra's algorithm on, this can be written as more simply O(E log V). See slides for runtime description.

A* Single-Target Shortest Paths. If we need only the path to a single target, then Dijkstra's is inefficient as it explores many many edges that we don't care about (e.g. when routing from Denver to NYC, we'd explore everything within more than a thousand miles in all directions before reaching NYC).

To fix this, we make a very minor change to Dijkstra's, where instead of visiting vertices in order of distance from the source, we visit them in order of distance from the source + h(v), where h(v) is some heuristic.

Or in pseudocode:

```
A*(G, s):
    while not every vertex has been visited:
        visit(unmarked vertex v for which distTo(v) + h(v) is minimized)
```

It turns out (but we did not prove), that as long as h(v) is less than the true distance from s to v, then the result of A* will always be correct.

Note: In the version in class, we did not use an explicit 'mark'. Instead, we tossed everything in the PQ, and we effectively considered a vertex marked if it had been removed from the PQ.

Previous
24.3 A* Algorithm

Next
24.5 Exercises

Last updated 1 year ago