Algorithms overview

Pre-strike

- Prüfer code
- Cheapest Spanning Trees: Kruskal + Prim

What's left:

- Dijkstra's Algorithm for Shortest Path
- Traveling Salesperson Problem

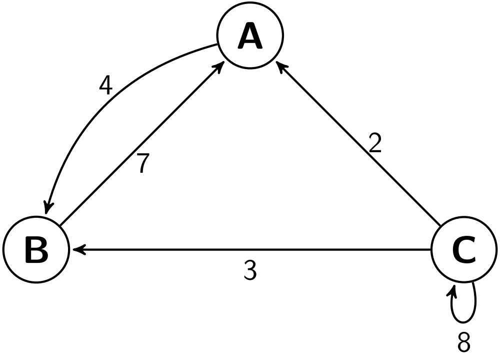
What we're going to skip:

Longest path + applications to scheduling (3.5 in notes)

Directed graphs

Definition

A *directed graph* is one where each edge has a chosen starting and ending point, usually indicated with arrows.



Walks in directed graphs:

You can only travel the edge in the direction the arrow shows.

Dijkstra's algorithm for shortest path

Input:

A weighted (possibly directed) graph G and starting vertex $v \in G$

Output:

For every vertex $w \in G$ a list of all shortest paths from v to w

Initialize:

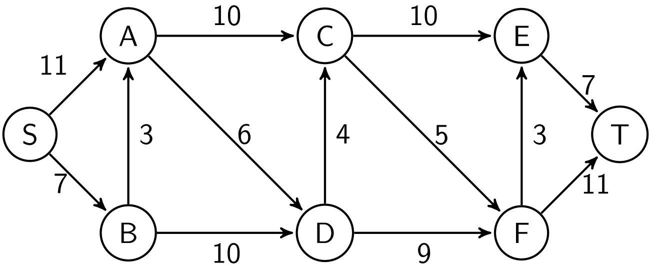
From starting vertex v list every edge out of v as a poeential shortest path to corresponding vertex w

Iterate:

- Choose w with cheapest potential shortest path and make these paths permanent
- Update list of potential paths by adding edges out of w to the shortest paths to w and checking if they're cheaper than known paths

Example graph from the 2008 Exam

Find all shortest paths from S to T.



Add on:

- Which edges if made a little longer would make the distance from S to T longer?
- Which edges if made a little shorter would make the distance from S to T shorter?

The main idea of why Dijkstra's works:

Suppose we're at the step were Dijkstra's decides the cheapest path to w.

Then:
A cheaper path to w would have to go through a vertex u we haven't found the cheapest path to yet.

But!
Even *getting* to u costs more than our cheapest path to w.

An observation:
Dijkstra's algorithm depends on the edge weights being non-negative!

Culture: performance of Dijkstra's algorithm

In a limited sense, Dijkstra's algorithm is optimal:

If *all we know* is that we have a weighted graph, then you can't do better than Dijkstra's algorithm.

In practise, often not very good:

When finding path from Sheffield to Edinburgh, Dijkstra's algorithm explores every street in London.

Real world maps have extra information:

It's easy to calculate the distance between two points as the crow flies, and we know the driving distance has to be at least that large.

The A* algorithm avoids searching London:

Supposes we have an easy to calculate "heuristic distance" $h(v, w)$, that is a lower bound for the actual distance $d(v, w)$.