Shortest Paths

- Given an edge-weighted digraph find the shortest path from s to t

Which vertices?

- source-sink: from one vertex to another
- single source: from one vertex to every other
- all pairs: between all pairs of vertices

Restrictions on edge weights?

- non negative weights
- arbitrary weights
- euclidean weights

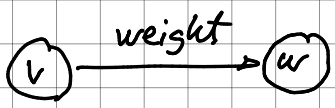
Cycles?

- no directed cycles
- no negative cycles

Simplifying assumption: shortest paths from s to each vertex v exists

class DirectedEdge

- int from()
- int to()
- double weight()



* EdgeWeightedDigraph class
  - allow self-loops and parallel edges

class ShortestPath

- double distTo(int v)
- Iterable<DirectedEdge> pathTo(int v)

* find shortest path from s to every other vertex

Shortest Paths Properties

I) A shortest-paths tree (SPT) solution exists

* can represent the SPT with two vertex-indexed arrays

- distTo[v] is the length of shortest path from s to v
- edgeTo[v] is the last edge on the shortest path from s to v

Edge Relaxation

- relax edge $e = v \to w$
- distTo[v] is length of shortest path from s to v
- distTo[w] is length of shortest path from s to w
- edgeTo[w] is last edge on shortest known path from s to w
- if $e = v \to w$ gives a shorter path to w through v => update both { distTo[w], edgeTo[w] }

Shortest-paths optimality conditions

=> distTo[] are the shortest path distances from (s) iff:

- distTo[s] = 0
- for each vertex v => distTo[v] is the length of some path from s to v
- for each edge $\boxed{e = v \rightarrow w}$ =>

distTo[w] ≤ distTo[v] + weight

Generic SPT

- Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices

=> repeat until optimality conditions are satisfied: relax any edge

Dijktra's Algorithm

- consider vertices in increasing order of distance from s (non-tree vertex with the lowest distTo[] value)
- add vertex to tree and relax all edges pointing from that vertex.

Proof

- every edge is relaxed exactly once ($e = v \to w$) when v is relaxed

=> leaving $distTo[w] \leq distTo[v] + e.weight()$

- inequality holds until algorithm terminates because:
  - distTo[w] cannot increase
  - distTo[v] will not change

* Prim's & Dijkstra's algorithms are essentially the same

* Distinction: rule used to choose next vertex for the tree
  - prim: closest vertex to the tree (via an undirected edge)
  - dijkstra's: closest vertex to the source (via a directed path)

dense graphs ⟶ array implementation

sparse graphs ⟶ binary heap

Edge-weighted DAGs

DAG = Directed Acyclic Graph

* acydic shortest paths:
  - consider vertices in topological order
  - relax all edges pointing from that

vertex

Proposition

• Topological sort algorithm computes SPT in any edge-weighted DAG in time proportional to E+V

* edge weights can be negative

Applications: seam carving

– resize an image without distortion for display on cell phones and web browsers

Content Aware Resizing

- grid DAG
- vertex = pixel
- edge = from pixel to 3 downward neighbours
- weight of pixel = energy function of 8 neighbouring pixels

- seam = shortest path from top to bottom

* to remove vertical seam
  - delete pixels on seam (one from each row)

Longest paths in edge-weighted DAGs:

- negate all weights
- find shortest paths
- negate weights in result

* parallel job scheduling
* Critical path method
  - source and sink vertices
  - 2 vertices: begin and end for each job
  - 3 edges for each job:
    1) begin to end (weighted by duration)

- Source to begin (0 weight)
- end to sink (0 weight)

- one edge for each precedence constraint (0 weight) —> end to start

=> solution = longest path from the source to schedule each job

Negative weights

a) Dijkstra —> doesn't work

! Re-weighting doesn't work => need a different algorithm

=> Negative cycles

Def: A negative cycle is a directed cycle whose sum of edge weights is negative

Proposition) SPT exists iff no negative cycles