Finding Shortest Paths

Shortest Path Problem

Shortest Path Problem

We are given a graph $G = (V, E)$ and an edge weight function $\omega \colon E \to \mathbb{R}$.

Length of a Path

The *length* or *weight* $\omega(P)$ of a path $P = \{v_1, v_2, \ldots, v_l\}$ with at least two vertices is

$$\omega(P) = \sum_{i=1}^{l-1} \omega(v_i v_{i+1})$$

If $|P| = 1$, $\omega(P) = 0$.

Shortest Path Problem

Shortest Path Problem

For two vertices u and v, the shortest path from u to v is the path P for which $\omega(P)$ is minimal. The *distance* $d(u, v)$ from u to v is the length of a shortest path from u to v.

Shortest Path Problem

Variants

- *Single Pair Shortest Path* (SPSP)
  Find a shortest path from a vertex u to some vertex v.
- *Single Source Shortest Path* (SSSP)
  Find shortest paths from a source vertex v to all other vertices in the graph.
- *All Pairs Shortest Path* (APSP)
  Find shortest paths fall vertex pairs u and v.

There is no algorithm for SPSP which is better in general than an algorithm for SSSP.

Shortest Path Properties

Theorem — Optimal Substructure Property

Each subpath of a shortest path is a shortest path.

Theorem — Triangle Inequality

For all vertices u, v, and w,

$$d(u, v) \leq d(u, w) + d(w, v).$$

Negative Weight Edges and Cycles

Negative Weight Edges

- Natural in some application
- Makes finding a shortest path harder

Theorem

If there is a path from u to v containing a vertex w and w is in a cycle C with $\omega(C) < 0$, then there is no shortest path from u to v.

Avoiding Cycles

- Only permit simple paths, i. e., no vertex twice
- Follows if graph has no negative cycles
- With negative cycles, shortest simple path problem equal to longest simple path problem
- Problem: loss of optimal substructure property

General Approach

General Approach

Store for each vertex v

- $\mathrm{dist}_s(v)$, length of currently best known path P from start vertex s to v
- $\mathrm{par}_s(v)$, parent of v in P

Relaxation

- Updates best known distance.

```
1 Procedure Relax(u, v)
2     If dist_s(v) > dist_s(u) + ω(uv) Then
3         Set par_s(v) := u and dist_s(v) := dist_s(u) + ω(uv).
```

General Approach

Initialization

- Set $\mathrm{par}_s(v) := \texttt{null}$ and $\mathrm{dist}_s(v) := \infty$ for each vertex v.
- Set $\mathrm{dist}_s(s) := 0$ for start vertex s.

Iteration

- Pick vertex pair u, v.
- Call Relax(u, v)
- Repeat

Open Questions

- How do we pick u and v?
- When do we stop the iteration?

Single Source Shortest Path

Bellman-Ford

Observation

- A shortest path has at most $|V| - 1$ edges.
- If we know all shortest path with k edges, we can compute all shortest paths with $k + 1$ edges by relaxing all edges once.

```
1 For Each v ∈ V
2     Set dist(v) := ∞ and par(v) = null.
3 Set dist(s) := 0.
4 For i := 1 To |V| − 1
5     For Each (u, v) ∈ E
6         Relax(u, v)
```

Bellman-Ford

Properties

- Runtime: $\mathcal{O}(|V||E|)$
- Works with negative weight edges
- Can detect negative cycles

Detecting negative cycles

- Negative cycle $\rightarrow$ There is always an edge (u, v) for which Relax(u, v) updates $\mathrm{dist}(v)$.
- If Relax(u, v) still updates $\mathrm{dist}(v)$ for $i \geq |V|$, then (u, v) is part of a negative cycle.

Dijkstra's Algorithm

Idea

- Let S be set of vertices where shortest path is known.
- Relax all outgoing edges (u, v), i. e., $u \in S$ and $v \notin S$.
- If $\mathrm{dist}(v)$ is minimal for all vertices not in S, then $\mathrm{dist}(v)$ is optimal.
- Add v to S and repeat.

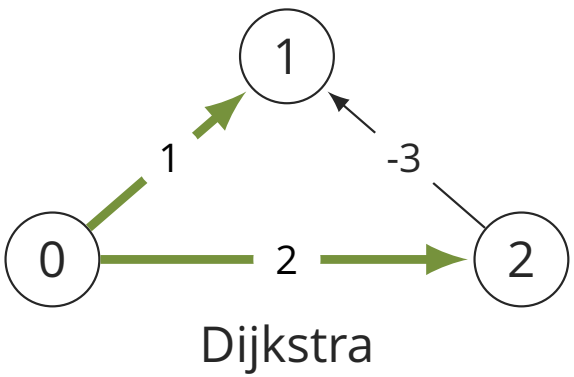
```
1 Initialize(G, s)
2 Create priority Q and add all vertices in V.
3 While Q is not empty
4     Remove v with minimal dist(v) from Q.
5     For Each (v, w) ∈ E
6         Relax(v,w)
```

Dijkstra's Algorithm

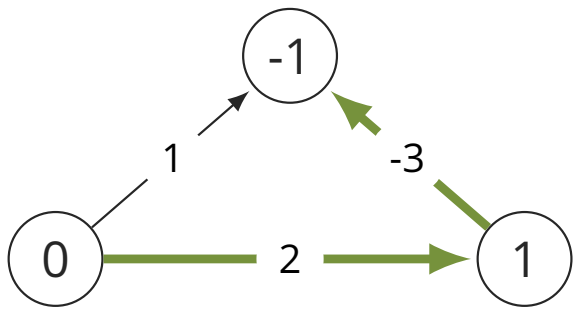
Properties

- Runtime: $\mathcal{O}(|E| \log |V|)$ with binary heaps and $\mathcal{O}(|V| \log |V| + |E|)$ with Fibonacci-Heaps
- Invariant: For all vertices in S, $\mathrm{dist}(s)$ is optimal.
- Requirement: No negative edges. The algorithm assumes that distances are always increasing.

What happens if there are negative edges?



Dijkstra

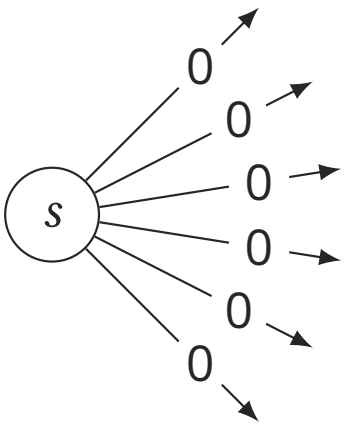


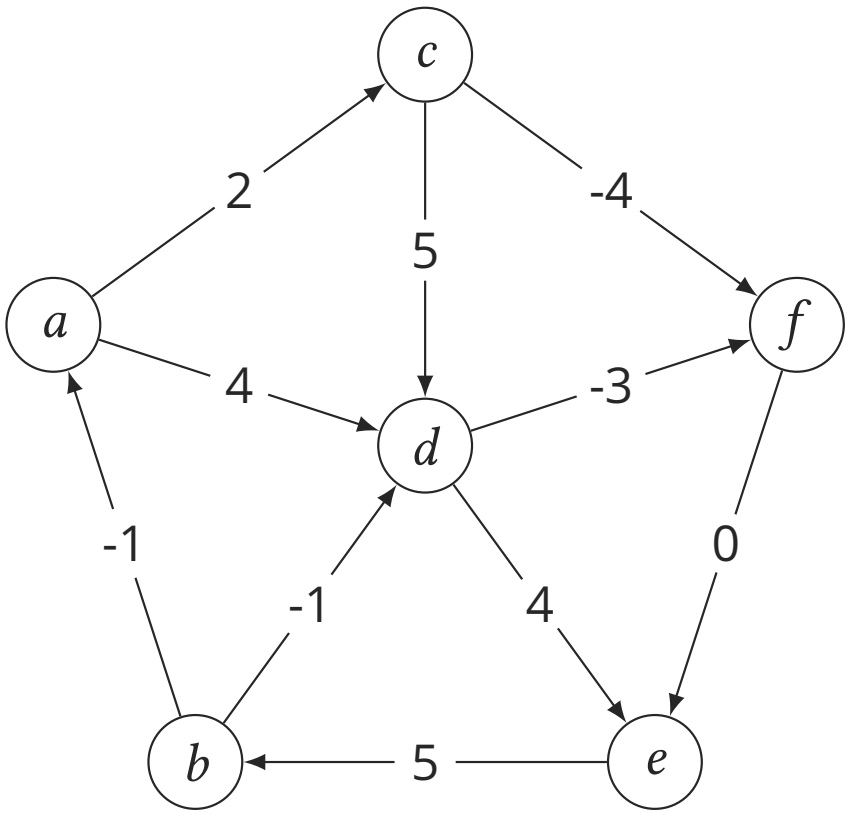
Bellman-Ford

Exercises

Exercises

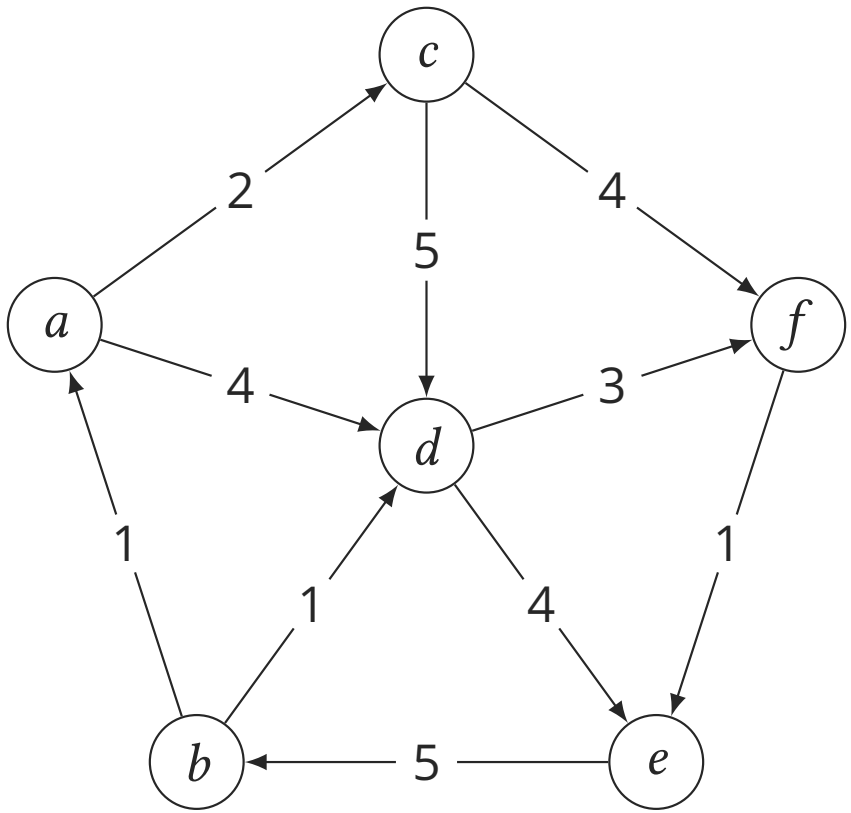
In the graph below, for each vertex $v \in \{a, b, \ldots, f\}$, there is an edge sv with weight 0. Run Bellman-Ford with start vertex s.





Exercises

Run Dijkstra's algorithm on the graph below with start vertex a.



Exercises – Uphill-Downhill Dijkstra

We say a sequence of numbers is *unimodal* if the sequence is non-decreasing until it reaches its maximum (ascent phase) and, then, it is non-increasing (descent phase). (We allow sequences that are monotone; either the ascent or the descent can have length zero.)

Examples

- Unimodal: $\langle 1, 5, 6, 2 \rangle$, $\langle 5, 7, 7, 9, 9, 9, 8, 3, 3, 1 \rangle$
- Not unimodal: $\langle 4, 3, 2, 5 \rangle$, $\langle 1, 2, 1, 2, 1 \rangle$

Consider a weighted digraph $G = (V, E)$ with non-negative edge weights and a source $s \in V$. Every vertex v has a given elevation $e(v)$. We say that a path is *unimodal* if the sequence of vertices along the path has unimodal elevations (first we go uphill and then downhill).

Find the minimum cost of unimodal paths from s to a given target vertex t in "Dijkstra time".

Exercises

For the the Uphill-Downhill Dijkstra problem (previous slide), assume that the elevation $e(v)$ is unique for each vertex v, i. e., $u \neq v$ if and only if $e(u) \neq e(v)$. Solve the problem in linear time.

Let $G = (V, E)$ be a directed weighted graph such that all the weights are positive. Let v and w be two vertices in G and $k \leq |V|$ be an integer. Design an algorithm to find the shortest path from v to w that contains exactly k edges. Note that the path does not need to be simple.

We are given a weighted directed graph G with a source vertex s. All weights are non-negative. We are also given a partition of the edges into "red" and "blue" edges. For all vertices v, find the minimum cost of reaching v from s along a path that uses at most one red edge. Your algorithm should run in "Dijkstra time".