ALGORITHMS AND DATA STRUCTURES II

Lecture 5

Shortest Paths,

Dijkstra's algorithm,

Bellman-Ford algorithm

Lecturer: K. Markov

markov@u-aizu.ac.jp

SHORTEST PATH

- Given a weighted directed graph, one common problem is finding the shortest path between two given vertices.

- Recall that in a weighted graph, the length of a path is the <u>sum of the weights</u> of each of the edges in that path.

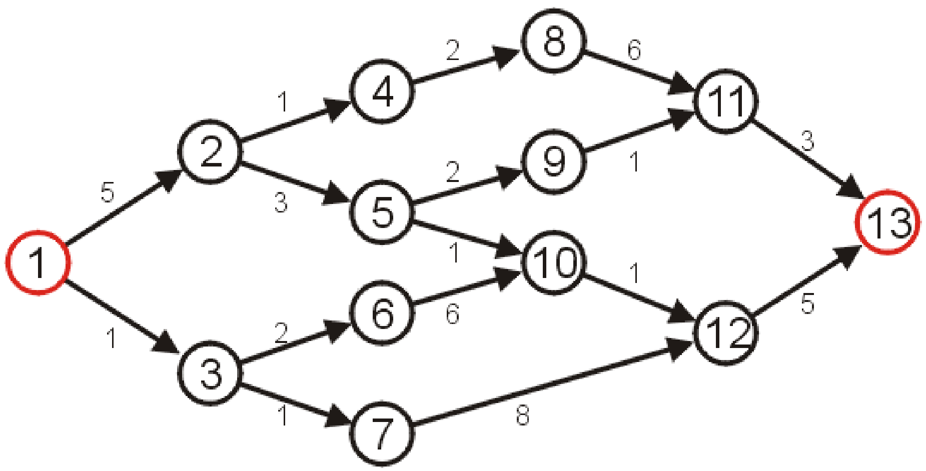
SHORTEST PATH

- Google Maps – Directions

- How to find the shortest path from **A** to **B**?

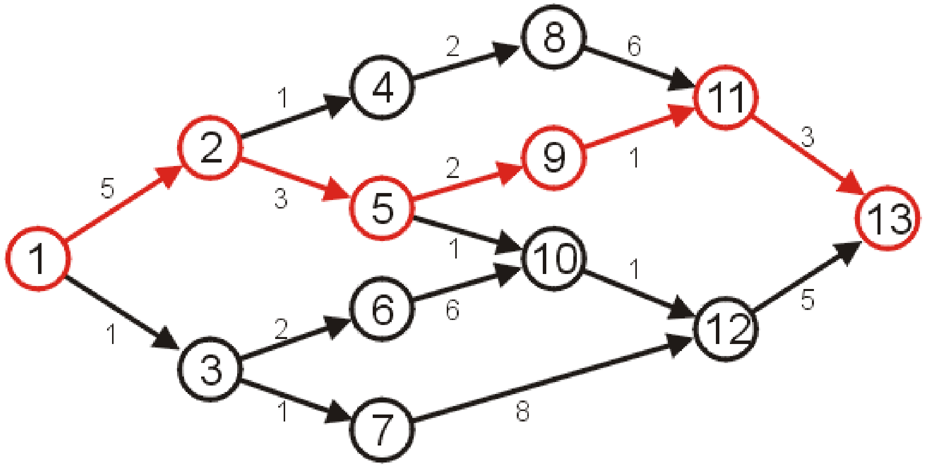
SHORTEST PATH

- Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13



SHORTEST PATH

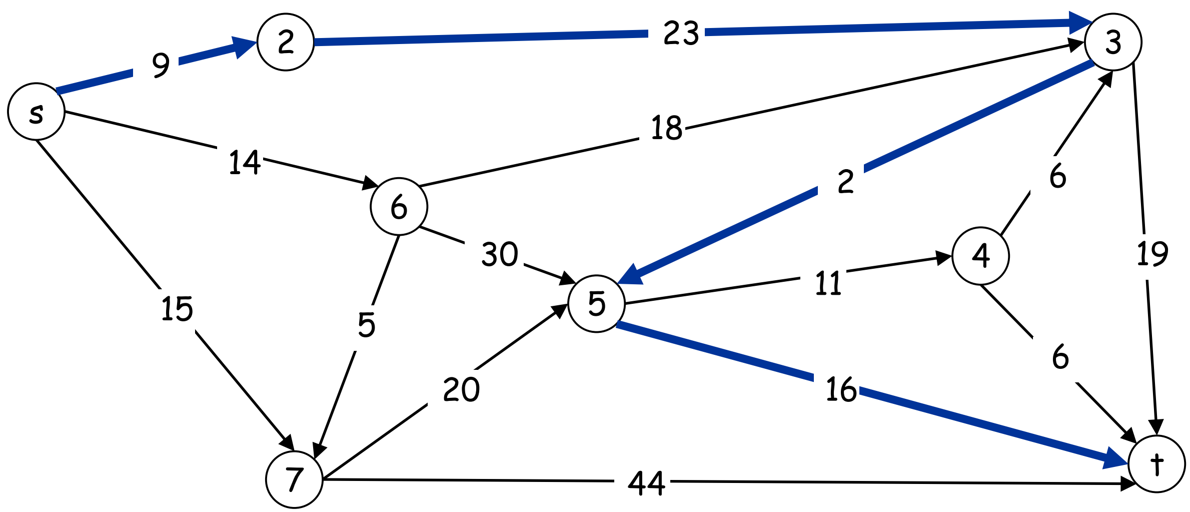
- After some consideration, we may determine that the shortest path is as follows, with length 14



- Other paths exists, but they are longer.

SHORTEST PATH EXAMPLE

- Given:
  - Weighted Directed graph G = (V, E).
  - Source s, destination t.
- Find shortest directed path from s to t.



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

SHORTEST PATH

- Which vertices?
  - **Source-sink**: from one vertex to another.
  - **Single source**: from one vertex to every other.
  - **All pairs**: between all pairs of vertices.
- Restriction on edge weights?
  - Nonnegative weights.
  - Arbitrary weights.
  - No weights / equal weights.
- Cycles?
  - No cycles.
  - No “negative cycles”.

SHORTEST PATH

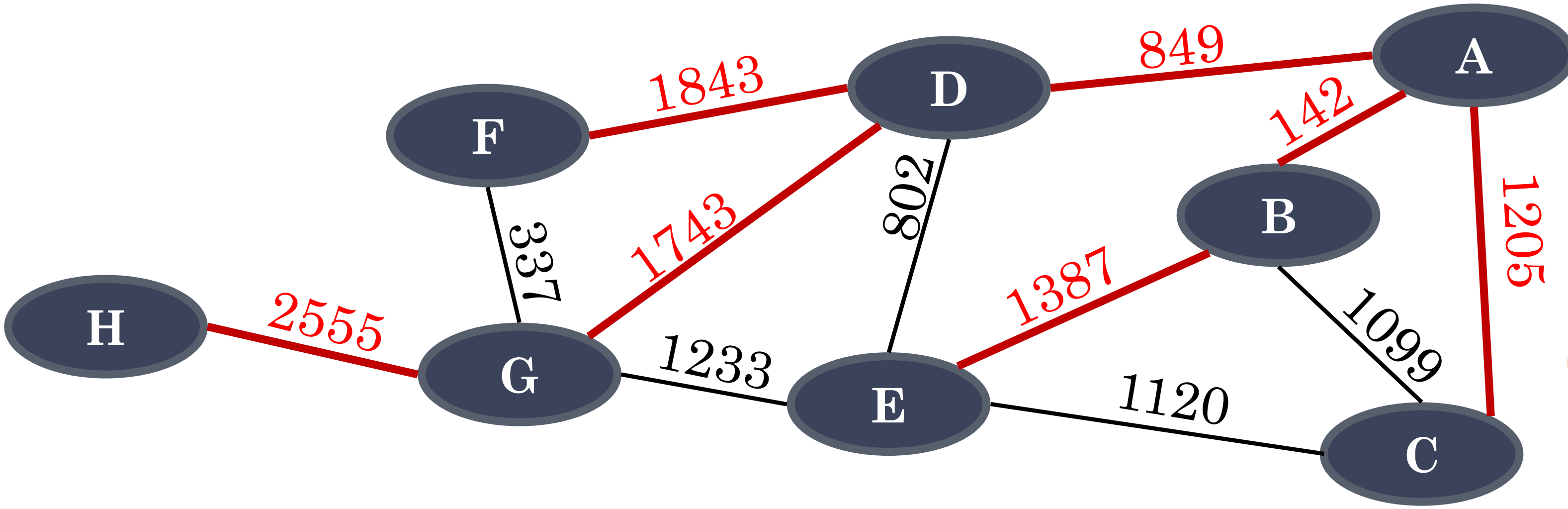
- Applications:
  - Map routing.
  - Robot navigation.
  - Texture mapping.
  - Typesetting in TeX.
  - Urban traffic planning.
  - Optimal pipelining of VLSI chips.
  - Routing of telecommunication messages.
  - Network routing protocols - OSPF, BGP, RIP.
  - Exploiting arbitrage opportunities in currency exchange.

SHORTEST PATH

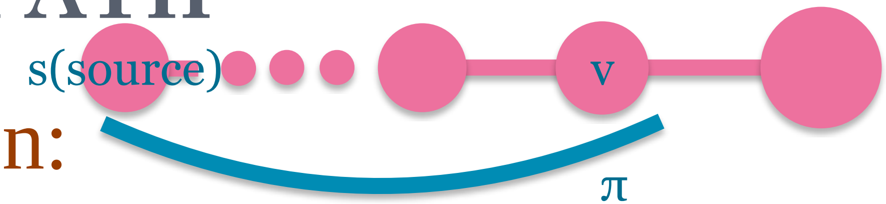
- A key observation is that if the shortest path between vertices s and t contains vertex v, then:
  - It will contain v once, as any cycles will only add to the length.
  - The shortest path from s to v must be the shortest path to v from s.
  - The shortest path from v to t must be the shortest path to t from v.
- If we can determine the shortest path to all other vertices that are incident to the target vertex we can easily compute the shortest path.

SHORTEST PATH

- Properties:
  - A sub-path of a shortest path is itself a shortest path.
  - There is a tree of shortest paths from a start vertex to all the other vertices.
- Example:
  - Tree of shortest paths from vertex **A**.



SHORTEST PATH



Representation:

- Given a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, for each vertex $\boldsymbol{v} \in \boldsymbol{V}$ we maintain a **predecessor $\boldsymbol{v}$**. π that is either another vertex or $\boldsymbol{NIL}$.
- Shortest path algorithms set the π attribute so that the chain of predecessors originating at a vertex $\boldsymbol{v}$ runs backwards along a shortest path from $\boldsymbol{s}$ (the source node) to $\boldsymbol{v}$.
- We are interested in the predecessor sub-graph $\boldsymbol{G_\pi} = (\boldsymbol{V_\pi}, \boldsymbol{E_\pi})$ induced by the π values:

$$V_\pi = \{v \in V : v.\pi \neq NIL\} \cup \{s\}$$
$$E_\pi = \{(\mathrm{v}.\pi, v) \in E : v \in V - \{s\}\}$$

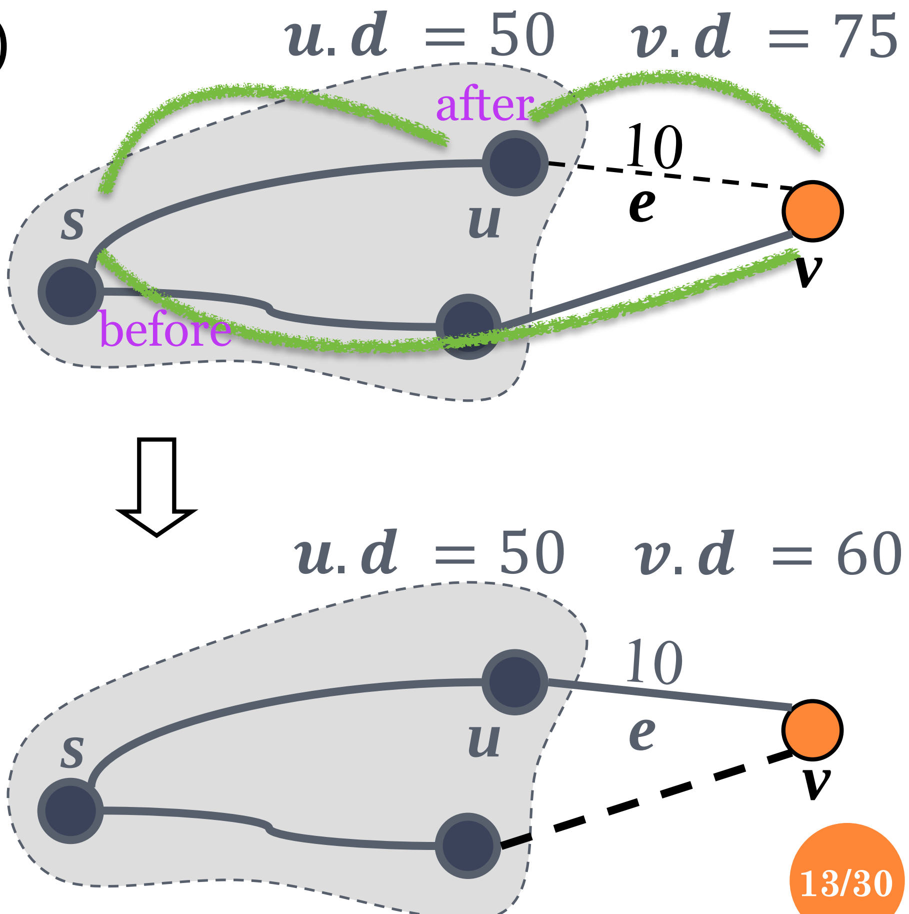
SHORTEST PATH

- Edge relaxation.
  - Shortest path algorithms are based on the technique called **relaxation**.
  - For each vertex we maintain attribute $\boldsymbol{v.d}$ which is an upper bound on the weight of a shortest path from source $\boldsymbol{s}$ to $\boldsymbol{v}$.
  - Relaxing the edge $(\boldsymbol{u}, \boldsymbol{v})$ consists of testing whether we can improve the shortest path to $\boldsymbol{v}$ found so far by going through $\boldsymbol{u}$, and if so, updating $\boldsymbol{v}.\pi$ and $\boldsymbol{v.d}$.

EDGE RELAXATION

- Consider an edge $\boldsymbol{e} = (\boldsymbol{u}, \boldsymbol{v})$ such that
  - $\boldsymbol{u}$ is the vertex most recently added to the tree.
  - $\boldsymbol{v}$ is not in the tree.
- The relaxation of edge $\boldsymbol{e}$ updates distance $\boldsymbol{v.d}$ as follows:

  $\boldsymbol{v.d} = \min\{\boldsymbol{v.d}, \boldsymbol{u.d} + \boldsymbol{weight}(\boldsymbol{e})\}$



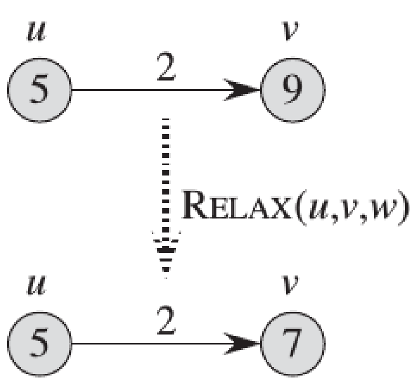
SHORTEST PATH

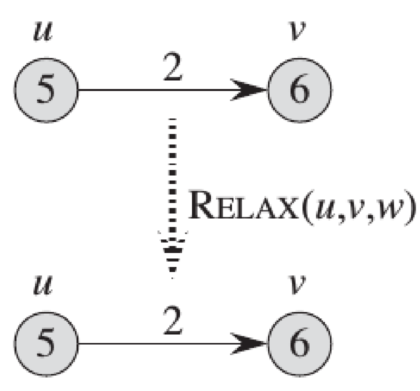
- Initialization and relaxation algorithms.

```
def INIT-SS (G, s):
    // Initialises shortest
    // single source tree
    // Input: Graph G
    // source vertex s.
    for each u ∈ G.V:
        u.d = ∞
        u.π = NIL
    s.d = 0
```

```
def RELAX (u, v, w):
    // Performs relaxation
    // step on edge (u, v)
    // Input: nodes u, v and
    // weight matrix w.
    if v.d > u.d + w (u, v):
        v.d = u.d + w (u, v)
        v. π = u
```

weight





SHORTEST PATH

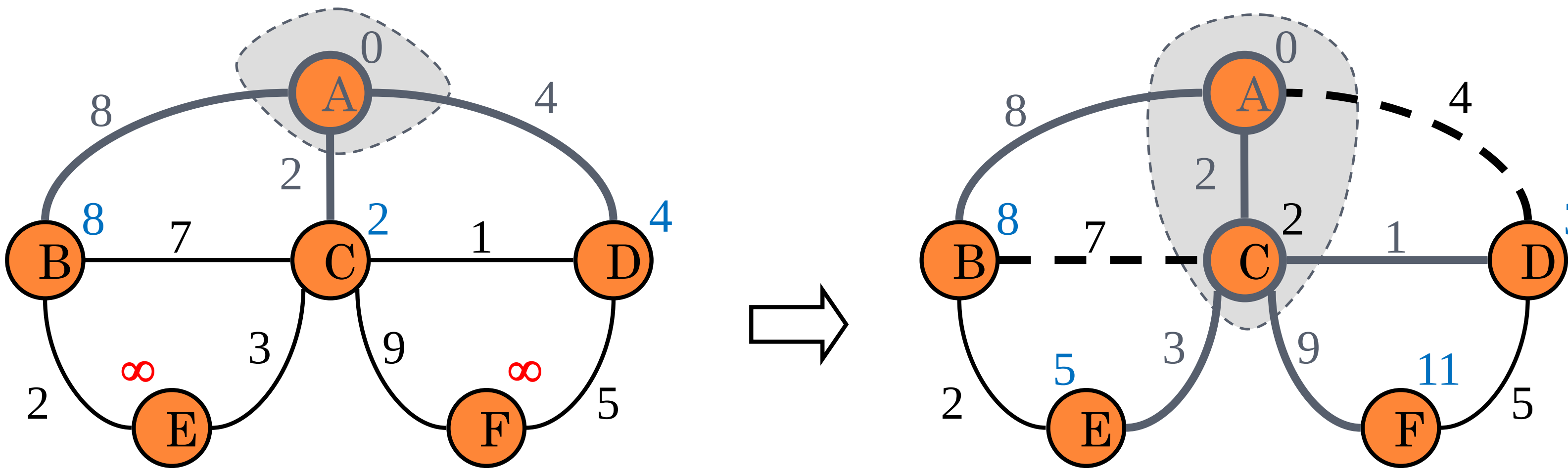
- Generic algorithm for finding shortest path tree (SPT).

```
def SPT (G, s, w):
    // Generic algorithm
    INIT-SS (G, s)
    while SPT not ready:
        choose edge (u, v)
        RELAX (u, v, w)
```

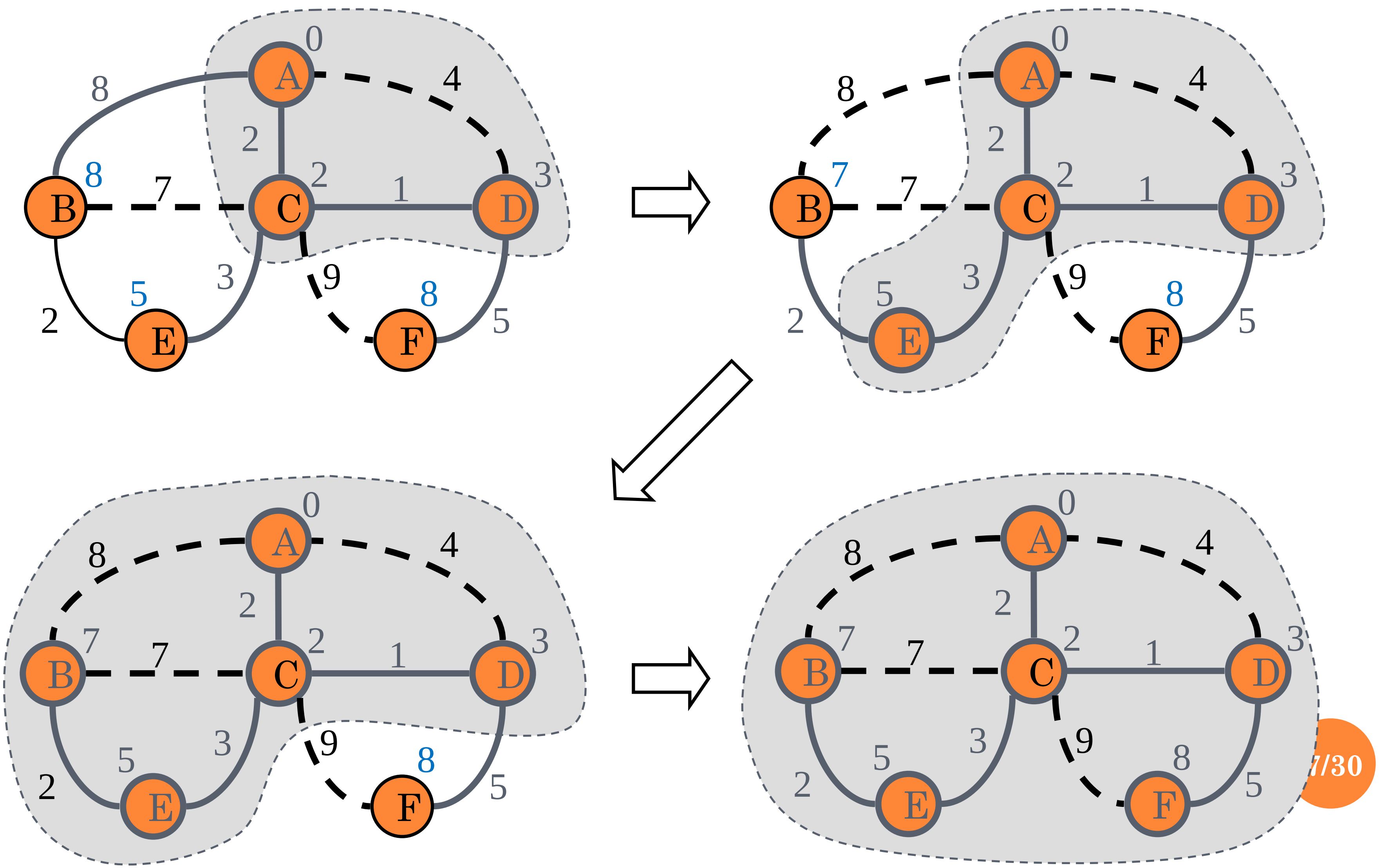
- How to choose which edge to relax?
  - Dijkstra's algorithm (nonnegative weights).
  - Topological sort algorithm (no directed cycles).
  - Bellman-Ford algorithm (no negative cycles).

DIJKSTRA'S ALGORITHM

- Consider vertices in increasing order of distance from $\boldsymbol{s}$ (non-tree vertex with the lowest $\boldsymbol{v.d}$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

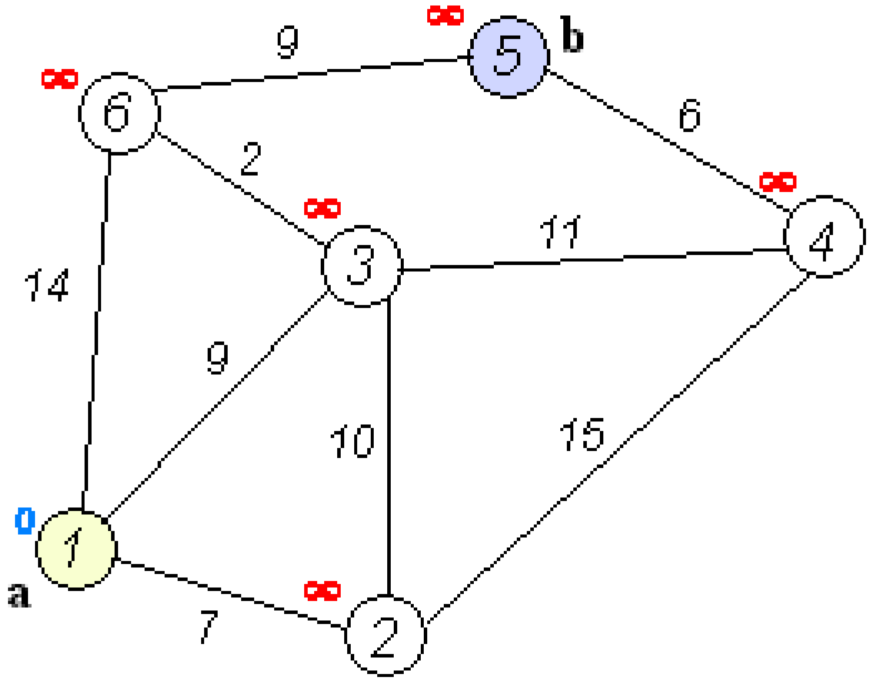


DIJKSTRA'S ALGORITHM



DIJKSTRA'S ALGORITHM

- Animated example of finding shortest path between vertex ***a*** and vertex ***b***.



DIJKSTRA'S ALGORITHM

- Implementation:
  - Uses min-priority queue of vertices keyed by their ***d*** values.

```
def DIJKSTRA (G, s, w):
    // Dijkstra's algorithm based on min-priority queue
    // Input: Graph G, start node s, weight matrix w.
    INIT-SS (G, s)
    Q = MIN-PRIORITY-QUEUE (G.V)
    while  Q ≠ ∅:
        u = EXTRACT-MIN (Q)
        for each v ∈ G.Adj[u]:
            RELAX (u, v, w)
```

DIJKSTRA'S ALGORITHM

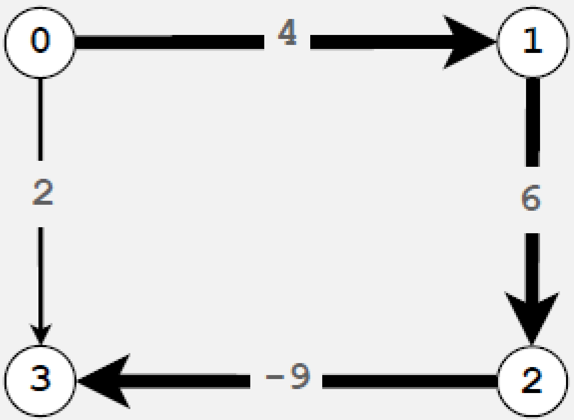
- Time complexity:
  - Label operations
    - We set/get the distance and parents of vertex $\boldsymbol{v}$ $\boldsymbol{O}(\deg(\boldsymbol{z}))$ times. Each such operation takes $\boldsymbol{O}(1)$ time.
  - Priority queue operations
    - Each vertex is inserted once into and removed once from the priority queue, where each insertion or removal takes $\boldsymbol{O}(\log \boldsymbol{V})$ time. In total - $\boldsymbol{V} \cdot \boldsymbol{O}(\log \boldsymbol{V})$
    - The key of a vertex in the priority queue is modified at most $\deg(\boldsymbol{w})$ times, where each key change takes $\boldsymbol{O}(\log \boldsymbol{V})$ time. In total - $\boldsymbol{E} \cdot \boldsymbol{O}(\log \boldsymbol{V})$
  - Dijkstra's algorithm runs in $\boldsymbol{O}((\boldsymbol{V} + \boldsymbol{E}) \log \boldsymbol{V})$ time provided the graph is represented by the adjacency list structure.

PRIORITY-FIRST SEARCH

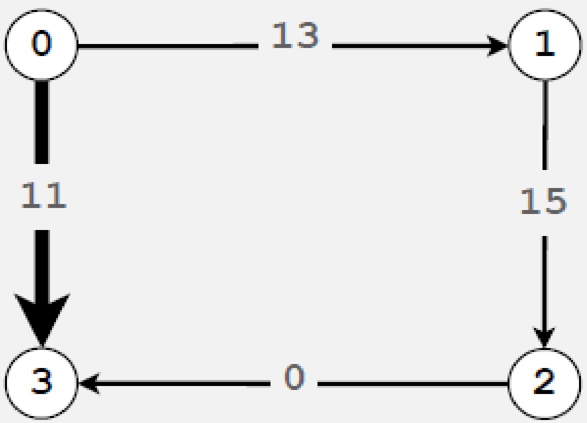
- Observation: Four of our graph-search methods are the same algorithm!
  - Maintain a set of explored vertices ***S***.
  - Grow ***S*** *by exploring edges with exactly one endpoint leaving* ***S***.
- DFS. Take edge from vertex which was discovered **most** recently.
- BFS. Take edge from vertex which was discovered **least** recently.
- Prim. Take edge of **minimum** weight.
- Dijkstra. Take edge to vertex that is **closest** to ***S***.

SHORTEST PATH - NEGATIVE WEIGHTS

- Dijkstra. Doesn't work with negative weights!



Dijkstra selects vertex 3 immediately after 0.
But shortest path from 0 to 3 is 0→1→2→3.

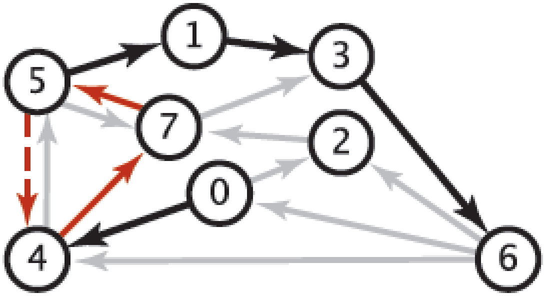


Adding 9 to each edge weight changes the shortest path from 0→1→2→3 to 0→3.

SHORTEST PATH - NEGATIVE CYCLE

- Negative cycle - directed cycle whose sum of edge weights is negative.

4->5	0.35
5->4	-0.66
4->7	0.37
5->7	0.28
7->5	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93



negative cycle (-0.66 + 0.37 + 0.28)

5->4->7->5

shortest path from 0 to 6

0->4->7->5->4->7->5...->1->3->6

- SPT exists if there are no negative cycles!

SHORTEST PATH - NEGATIVE WEIGHTS

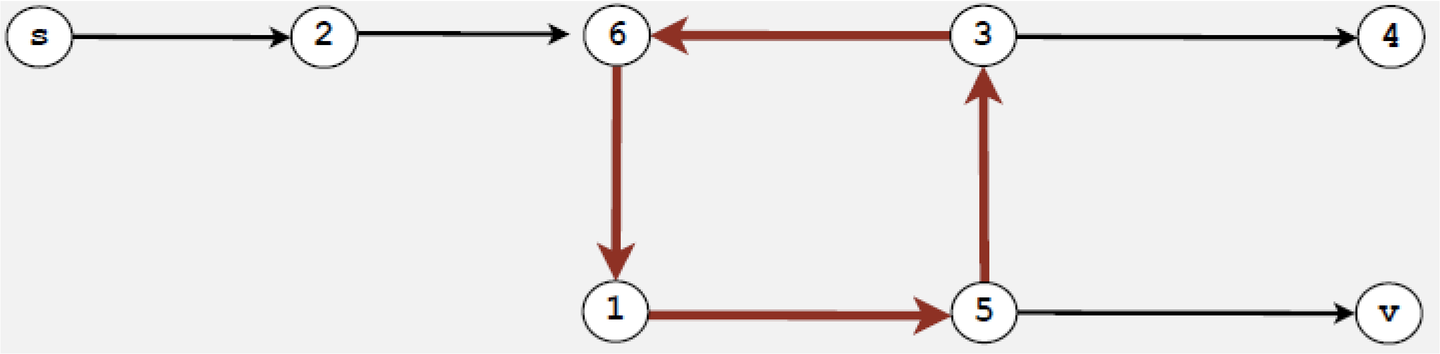
- SPT algorithm:
  - Step 1. Initialize.
  - Step 2. Relax all edges $\boldsymbol{V-1}$ times.

```
def BELLMAN-FORD (G, s, w):
    // Bellman-Ford algorithm, no negative cycles.
    // Input: Graph G, start node s, weight matrix w.
    INIT-SS (G, s)
    for i=1 to |G.V|-1:
        for each edge (u,v) ∈ G.E:
            RELAX (u, v, w)
```

  - The running time is proportional to $|\boldsymbol{E}| \cdot |\boldsymbol{V}|$ in the worst case, so $\boldsymbol{O(EV)}$.

FINDING A NEGATIVE CYCLE

- If there is a negative cycle, Bellman-Ford gets stuck in loop, updating ***d*** and ***π*** attributes of vertices in the cycle.



- If any vertex ***v*** is updated at ***V****th* loop, there exists a negative cycle.

NEGATIVE CYCLE

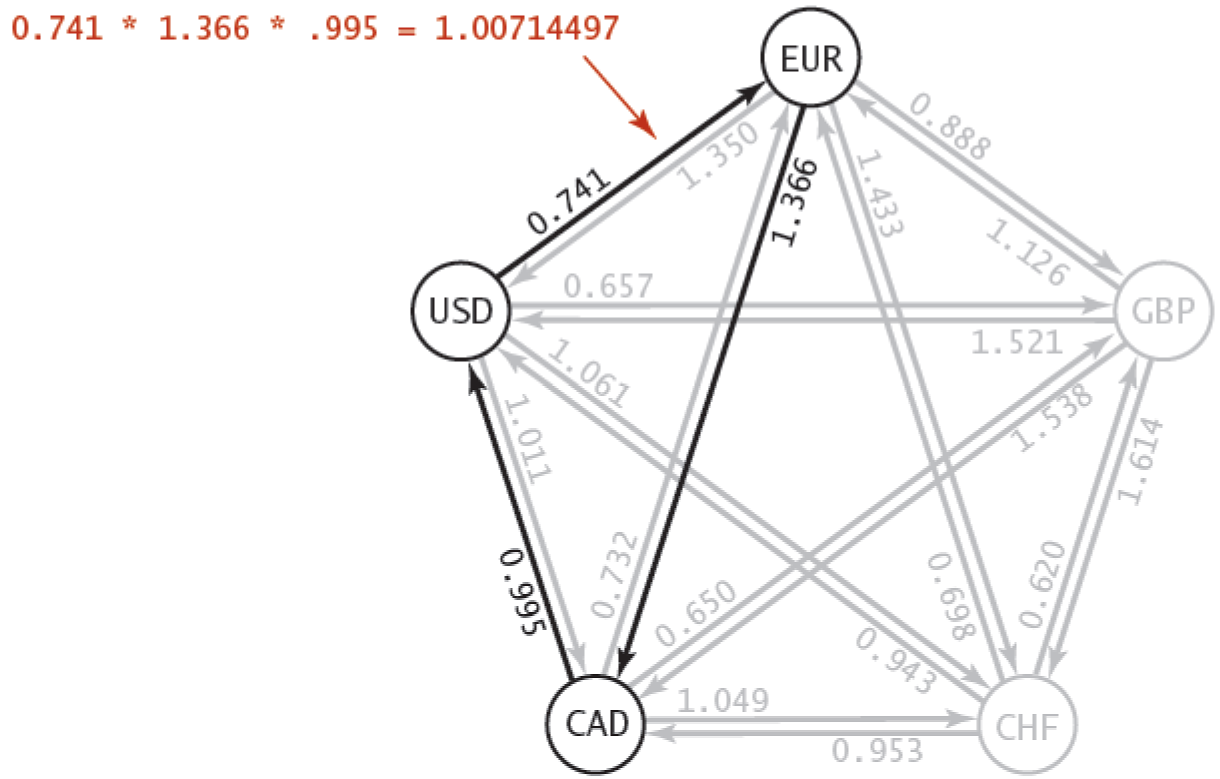
- Application: Given table of exchange rates, is there an arbitrage opportunity?

	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.011
EUR	1.350	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.943	0.698	0.620	1	0.953
CAD	0.995	0.732	0.650	1.049	1

- Ex. \$1,000 $\Rightarrow$ 741 Euros $\Rightarrow$ 1,012.206 Canadian dollars $\Rightarrow$ \$1,007.14497. $(1000 \times 0.741 \times 1.366 \times 0.995 = 1007.14497)$

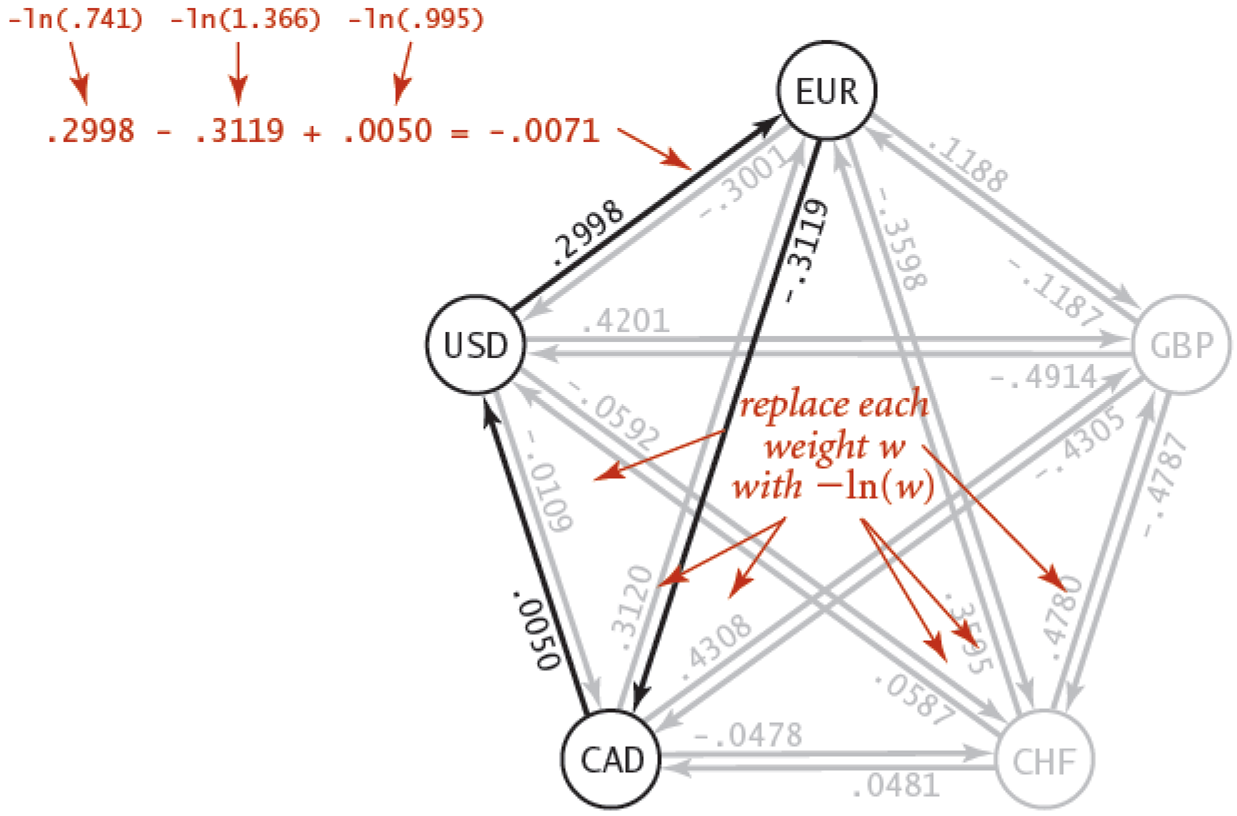
NEGATIVE CYCLE

- Currency exchange graph
  - Vertex -> currency, Edge -> transaction, Weight -> exchange rate
  - Find directed cycle with weight product > 1.



NEGATIVE CYCLE

- Currency exchange graph
  - Take log of the weights: w -> -ln(w).
  - Look for negative cycle.



SHORTEST PATH SUMMARY

- Dijkstra's algorithm.
  - Nearly linear-time when weights are nonnegative.
  - Generalization encompasses DFS, BFS, and Prim.
- Negative weights and negative cycles.
  - Arise in applications.
  - If there are no negative cycles, we can find shortest paths via Bellman-Ford.
  - If negative cycles exist, we can find one via Bellman-Ford.
- Shortest-paths is a broadly useful problem-solving model.

That's All for Today!