ALGORITHMS AND DATA STRUCTURES II

Lecture 5
Shortest Paths,
Dijkstra's algorithm,
Bellman-Ford algorithm

Lecturer: K. Markov
markov@u-aizu.ac.jp

1/26

SHORTEST PATH

- Google Maps – Directions
- How to find the shortest path from **A** to **B**?

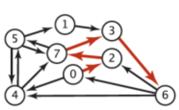
2/26

SHORTEST PATH

- Given an edge-weighted digraph, find the shortest (directed) path from *s to t*.

edge-weighted digraph

4->5	0.35
5->4	0.35
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



shortest path from 0 to 6

0->2	0.26
2->7	0.34
7->3	0.39
3->6	0.52

3/26

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".

4/26

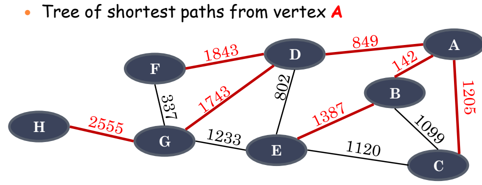
SHORTEST PATH

- 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.

5/26

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**



6/26

SHORTEST PATH

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

$$V_\pi = \{v \in V : v.\pi \neq NIL\} \cup \{s\}$$

$$E_\pi = \{(v.\pi, v) \in E : v \in V - \{s\}\}$$

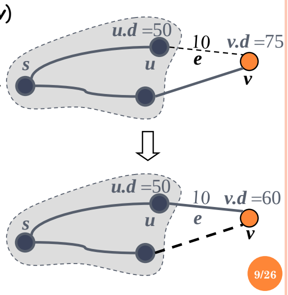
SHORTEST PATH

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

EDGE RELAXATION

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

  v.d =*min*{***v.d***,***u.d*** +***weight(e)***}

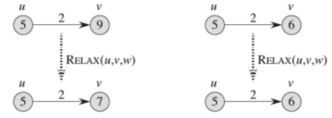


SHORTEST PATH

- Initialization and relaxation algorithms.

```
def INIT-SS (G, s):
    // Initializes 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
```



SHORTEST PATH

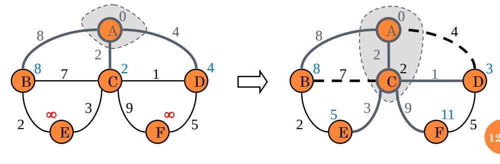
- Generic algorithm for finding shortest path tree (SPT).

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

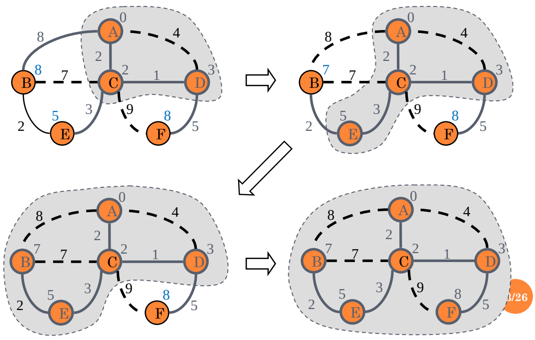
- How to choose which edge to relax?
  - Dijkstra's algorithm (nonnegative weights).
  - Topological sort algorithm (no directed cycles).
  - Bellman-Ford algorithm (no negative cycles).

DIJKSTRA'S ALGORITHM

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

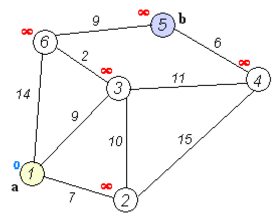


DIJKSTRA'S ALGORITHM



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

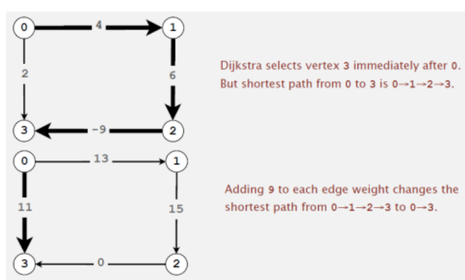
- Time complexity:
  - Label operations
    - We set/get the distance and parents of vertex ***v*** $O(deg(z))$ times. Each such operation takes $O(1)$ time.
  - Priority queue operations
    - Each vertex is inserted once into and removed once from the priority queue, where each insertion or removal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

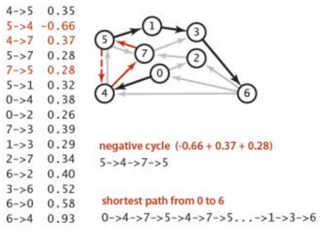
SHORTEST PATH - NEGATIVE WEIGHTS

- Dijkstra. Doesn't work with negative weights!



SHORTEST PATH - NEGATIVE WEIGHTS

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



- SPT exists if there are no negative cycles!

19/26

SHORTEST PATH - NEGATIVE WEIGHTS

- SPT algorithm:
  - Step 1. Initialize.
  - Step 2. Relax all edges *V* times.

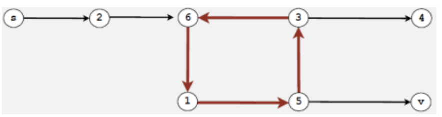
```
def BELLMAN-FORD (G, s, w):
    // Bellman-Ford algorithm, no negative cycles.
    // Input: Graph G, start node s, weight matrix w.
    INIT-SS (G, s)
    for i=1 to |G.V|-1:
        for each edge (u,v) ∈ G.E:
            RELAX (u, v, w)
```

  - The running time is proportional to |E|·|V| in the worst case, so **O(EV)**.

20/26

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.

21/26

NEGATIVE CYCLE

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

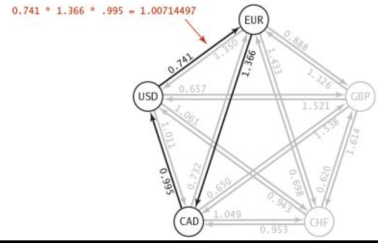
	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.011
EUR	1.350	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.943	0.698	0.620	1	0.953
CAD	0.995	0.732	0.650	1.049	1

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

22/26

NEGATIVE CYCLE

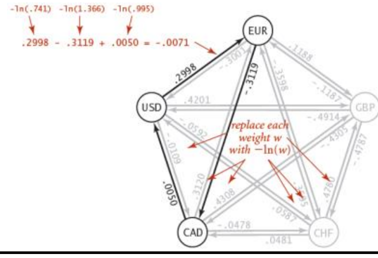
- Currency exchange graph
  - Vertex -> currency, Edge -> transaction, Weight -> exchange rate
  - Find directed cycle with weight product > 1.



23/26

NEGATIVE CYCLE

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



24/26

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.

25/26

THAT'S ALL FOR TODAY!

26/26