Shortest path algorithms

Data Structures and Algorithms for Computational Linguistics III
(ISCL-BA-07)

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

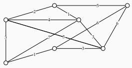
University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2021/22

version: a9be53e @2021-12-08

Weighted graphs

- A *weighted graph* is a graph, where each edge is associated with a weight
- Weights can be any numeric value, but for some algorithms require
  - Non-negative weights
  - 'Euclidean' weights: weights that are proper distance metrics
- Weights often indicate distance or cost, but they can also represent positive relations (e.g., affinity between nodes)
- Weight of a path is the sum of wights of the edges on the path



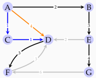
Shortest path

- Finding shortest paths on a weighted (directed) graph is one of the most common problems in many fields
- Applications include
  - Navigation
  - Routing in computer networks
  - Optimal construction of electronic circuits, VLSI chips
  - Robotics, transportation, finance, …

Shortest paths on unweighted graphs

BFS

- A BFS search tree gives the shortest path from the source node to all other nodes
- The BFS is not enough on weighted graphs
- Shortest-cost path may be longer in terms of nodes visited



Shortest paths on weighted graphs

variation of the problem

- Different versions of the problem:
  - Single source shortest path: find shortest path from a source node to all others
  - Single target (sometimes called sink) shortest path: find shortest path from all nodes to a target node
  - Source to target: from a particular source node to a particular target node
  - All pairs: shortest paths between all pairs of nodes
- Restrictions on weights:
  - Euclidean weights
  - Non-negative weights
  - Arbitrary weights

Dijkstra's algorithm

intro

- Dijkstra's algorithm is a 'weighted' version of the BFS
- The algorithm finds shortest path from a single source node to all connected nodes
- Weights has to be non-negative
- It is a greedy algorithm that grows a 'cloud' of nodes for which we know the shortest paths from the source node
- The new nodes are included in the cloud in order of their shortest paths from the source node

Dijkstra's algorithm

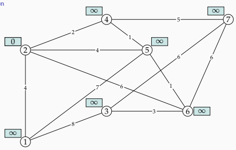
the algorithm

- We maintain a list D of minimum know distances to each node
- At each step
  - we take closest node out of Q
  - update the distances of all nodes
- Can be more efficient if Q is implemented using a (adaptable) priority queue

```
D[s] ← 0
for each node v ≠ s do
    D[v] ← ∞
Q ← nodes
while Q is not empty do
    Remove node u with min D[u] from Q
    for each edge (u, v) do
        if D[u] + w(u, v) < D[v] then
            D[v] ← D[u] + w(u, v)
D contains the shortest distances from s
```

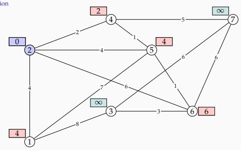

Dijkstra's algorithm

demonstration



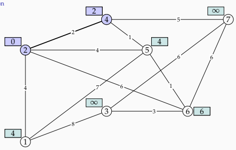
Dijkstra's algorithm

demonstration



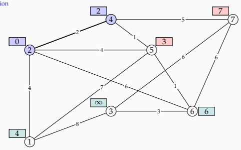
Dijkstra's algorithm

demonstration



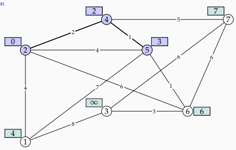
Dijkstra's algorithm

demonstration

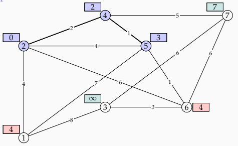


Dijkstra's algorithm

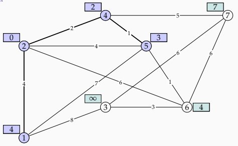
demonstration



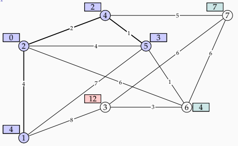
Dijkstra's algorithm
demonstration



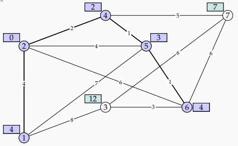
Dijkstra's algorithm
demonstration



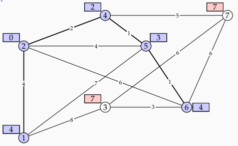
Dijkstra's algorithm
demonstration



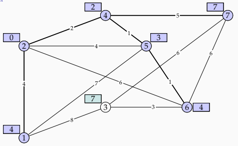
Dijkstra's algorithm
demonstration



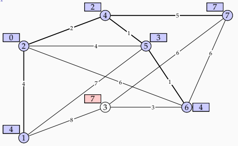
Dijkstra's algorithm
demonstration



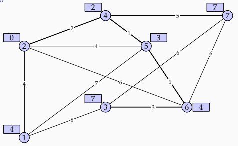
Dijkstra's algorithm
demonstration



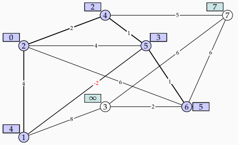
Dijkstra's algorithm
demonstration



Dijkstra's algorithm
demonstration



Dijkstra's algorithm and negative weights



Dijkstra's algorithm
the algorithm

- In general, complexity is $O(t_{\text{find\_min}} n + t_{\text{update\_key}} m)$
- With list-based implementation of Q: $O(m + n^2) = O(n^2)$
- With a priority queue: $O((m + n) \log n)$

$D[s] \leftarrow 0$
for each node $v \neq s$ **do**
  $D[v] \leftarrow \infty$
$Q \leftarrow$ nodes
while Q is not empty **do**
  Remove node u with min $D[u]$ from Q
  for each edge (u, v) **do**
    if $D[u] + w(u, v) < D[v]$ **then**
      $D[v] \leftarrow D[u] + w(u, v)$
D contains the shortest distances from s

Shortest-path tree

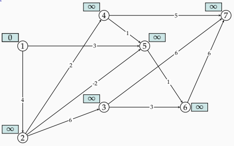
- The way we introduced, the Dijkstra's algorithm does not give the shortest-path tree
- Similar to traversal algorithms, we can extract it from distances D
- Running time is $O(n^2)$ (or $O(n + m)$)

$T \leftarrow \varnothing$
for $u \in D - \{s\}$ **do**
  for each edge (v, u) **do**
    if $D[v] == D[u] + w(v, u)$ **then**
      $T \leftarrow T \cup (v, u)$

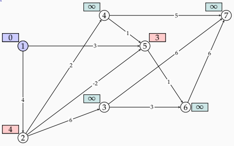
Shortest-paths on DAGs

- The shortest path can be found more efficiently, if the graph is a DAG
- The algorithm is similar to Dijkstra's, but simpler and faster
- Only difference is we follow a topological order
- The algorithm will also work with negative edge weights

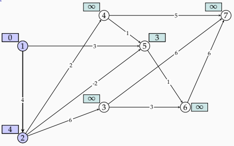
Shortest-paths on DAGs
demonstration



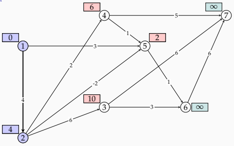
Shortest-paths on DAGs
demonstration



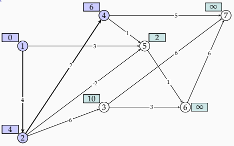
Shortest-paths on DAGs
demonstration



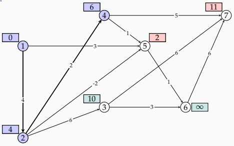
Shortest-paths on DAGs
demonstration



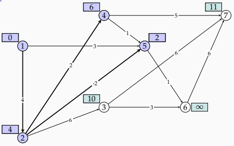
Shortest-paths on DAGs
demonstration



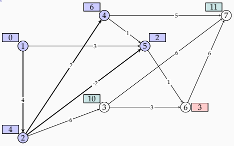
Shortest-paths on DAGs
demonstration



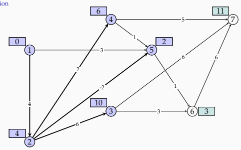
Shortest-paths on DAGs
demonstration



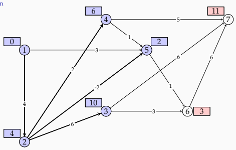
Shortest-paths on DAGs
demonstration



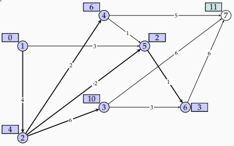
Shortest-paths on DAGs
demonstration



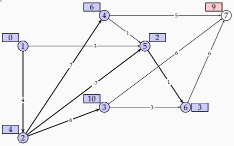
Shortest-paths on DAGs
demonstration



Shortest-paths on DAGs
demonstration

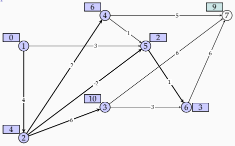


Shortest-paths on DAGs
demonstration



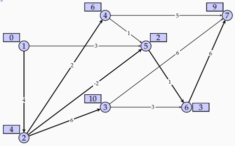
Shortest-paths on DAGs

demonstration



Shortest-paths on DAGs

demonstration



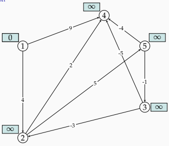
Shortest-paths on directed graphs

with negative wights – without negative cycles

- Single-source shortest path problem can also be solved efficiently for any directed graph
  - including cycles (no DAG requirement)
  - including negative weights
  - *excluding* negative cycles
- The algorithms is known as Bellman-Ford algorithm
  - Similar to earlier algorithms, initialize $D[s] = 0$, $D[v] = \infty$
  - Make n passes over the edges
    - Update distances for each edge (relax edges)
    - Stop if there were no changes at the end of a pass

Bellman-Ford algorithm

demonstration

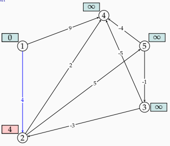


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

demonstration

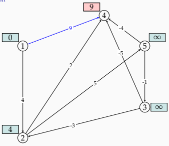


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

demonstration

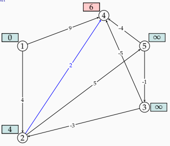


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

demonstration

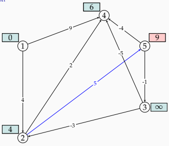


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

demonstration

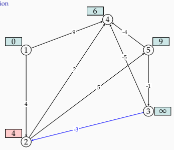


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

demonstration

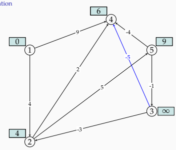


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

demonstration

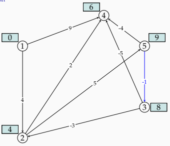


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

demonstration

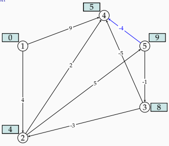


edges

$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm

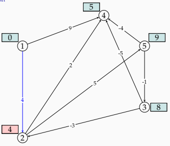
demonstration



edges

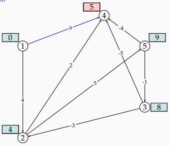
$1 \to 2$
$1 \to 4$
$2 \to 4$
$2 \to 5$
$3 \to 2$
$3 \to 4$
$5 \to 3$
$5 \to 4$

Bellman-Ford algorithm
demonstration



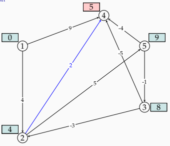
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



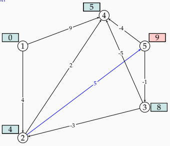
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



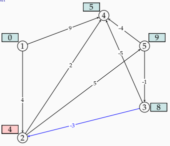
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



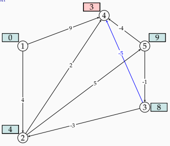
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



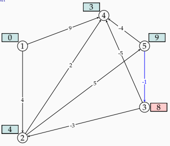
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



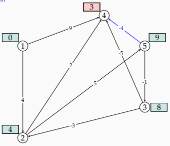
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



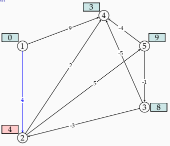
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



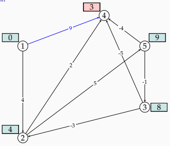
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



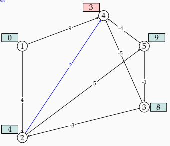
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



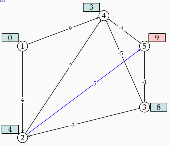
edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm
demonstration



edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

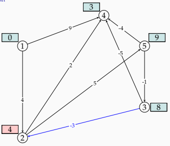
Bellman-Ford algorithm
demonstration



edges
1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Bellman-Ford algorithm

demonstration



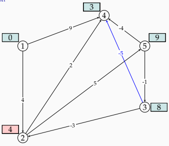
edges

1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Introduction/motivation Dijkstra SP on DAGs SP with negative weights

Bellman-Ford algorithm

demonstration



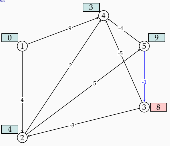
edges

1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Introduction/motivation Dijkstra SP on DAGs SP with negative weights

Bellman-Ford algorithm

demonstration



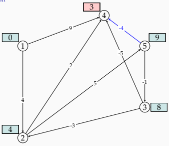
edges

1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Introduction/motivation Dijkstra SP on DAGs SP with negative weights

Bellman-Ford algorithm

demonstration



edges

1 → 2
1 → 4
2 → 4
2 → 5
3 → 2
3 → 4
5 → 3
5 → 4

Introduction/motivation Dijkstra SP on DAGs SP with negative weights

Summary

- Shortest path algorithms are one of the most applied graph algorithms
- We revised three algorithms
  - Dijkstra's: non-negative weights, general algorithm
  - For DAGs: unrestricted weights, following topological order
  - Bellman-Ford: no negative cycles, digraphs
- Reading: Goodrich, Tamassia, and Goldwasser (2013, chapter 14)

Next:

- Minimum spanning trees
- Reading: Goodrich, Tamassia, and Goldwasser (2013, chapter 14)

Acknowledgments, credits, references

Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser (2013). *Data Structures and Algorithms in Python*. John Wiley & Sons, Incorporated. ISBN: 9781118476734.

blank

blank

blank

blank