EECS 3101 A: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101A`
Also on Moodle

Definitions

- Shortest path = a path of the minimum weight
- Applications: static/dynamic network routing, robot motion planning,map/route generation in traffic

Problems

- Unweighted shortest-paths – BFS
- Single-source, single-destination: Given two vertices, find a shortest path between them
- Single-source, all destinations: Find a shortest path from a given source (vertex s) to each of the vertices. [Solution to this problem solves the previous problem efficiently]. Greedy algorithm!
- All-pairs Shortest Paths: Find shortest-paths for every pair of vertices. Dynamic programming algorithm

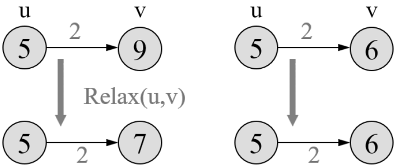
Optimal Substructure Property

- Theorem: subpaths of shortest paths are shortest paths
- Proof (cut and paste): if some subpath were not the shortest path, one could substitute the shorter subpath and create a shorter total path
- Suggests there are DP and greedy algorithms

Key operation: Relaxation

- For each vertex v in the graph, we maintain $d[v]$, the estimate of the shortest path from source s, initialized to ∞ at start
- Relaxing an edge (u, v) means testing whether we can improve the shortest path to v found so far by going through u

```
Relax (u,v,w)
if d[v] >
   d[u]+w(u,v) then
    d[v] ← d[u]+w(u,v)
    π[v] ← u
```



Dijkstra's Algorithm

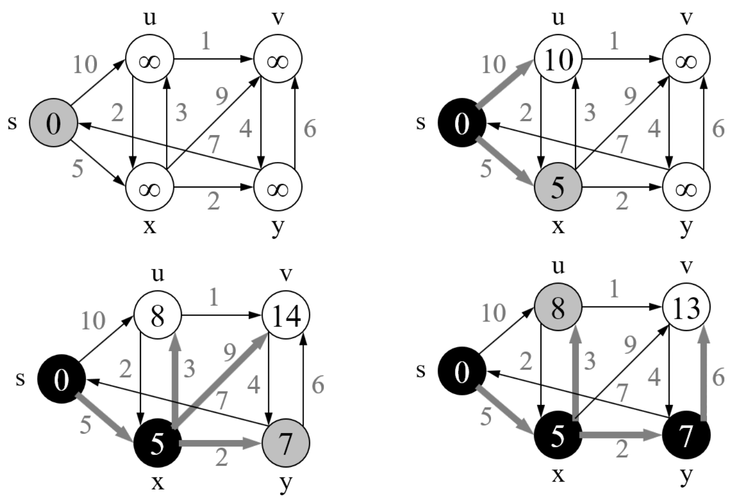
- Non-negative edge weights
- Greedy, similar to Prim's algorithm for MST
- Like breadth-first search (if all weights $= 1$, one can simply use BFS)
- Use priority queue Q keyed by $d[v]$ (BFS used FIFO queue, here we use a PQ, which is re-organized whenever some $d[]$ decreases)
- Basic idea
  - maintain a set S of solved vertices
  - at each step select "closest" vertex u, add it to S, and relax all edges from u

Dijkstra's Algorithm - pseudocode

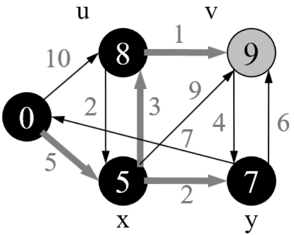
Graph G, weight function w, source s

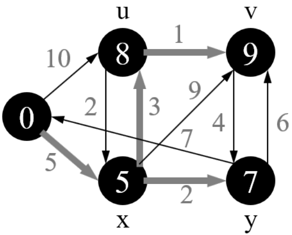
```
DIJKSTRA(G, w, s)
for each v ∈ V
        do d[v] ← ∞
d[s] ← 0
S ← ∅    ▷ Set of discovered nodes
Q ← V
while Q ≠ ∅
        do u ← EXTRACT-MIN(Q)
            S ← S ∪ {u}
            for each v ∈ Adj[u]
                  do if d[v] > d[u] + w(u, v)
                        then d[v] ← d[u] + w(u, v)
```

Dijkstra's Algorithm - example



Dijkstra's Algorithm - example





Correctness idea: a label $d[v]$ is set once, with the correct value of the shortest distance from s to v

Dijkstra's Algorithm - Running Time

- Extract-Min executed $|V|$ times
- Decrease-Key executed $|E|$ times
- $Time = |V| T_{Extract-Min} + |E| T_{Decrease-Key}$
- Time depends on different PQ implementations:
  Array-based: $\Theta(|V|^2)$
  Heap-based: $\Theta(|E| \log |V|)$
  Fibonacci Heap-based: $\Theta(|E| + |V| \log |V|)$

Bellman-Ford Algorithm

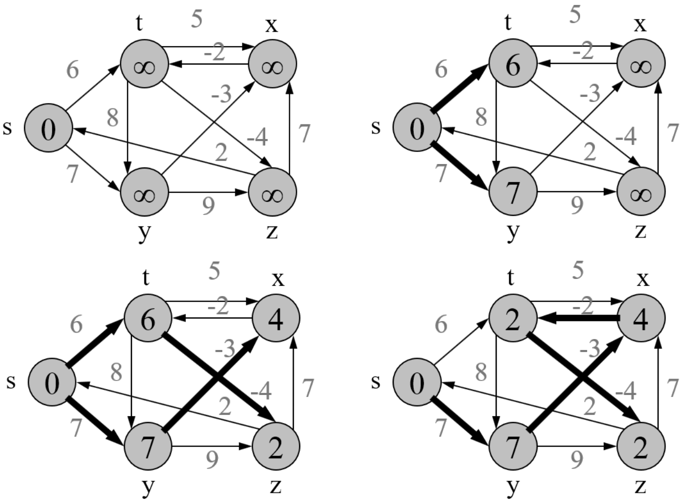
- Dijkstra's algorithm does not work when there are negative edges
- Intuition: we can not be greedy any more on the assumption that the lengths of paths will only increase in the future
- Bellman-Ford algorithm detects negative cycles (returns false) or returns the shortest path-tree

Bellman-Ford Algorithm - Pseudocode

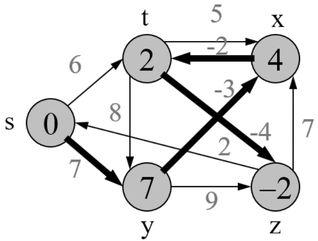
```
Bellman-Ford(G,w,s)
for each v ∈ V[G]
    d[v] ← ∞
d[s] ← 0
π[s] ← NIL
for i ← 1 to |V[G]|-1 do
     for each edge (u,v) ∈ E[G] do
         Relax (u,v,w)
for each edge (u,v) ∈ E[G] do
     if d[v] > d[u] + w(u,v) then return false
return true
```

Running time: $\Theta(|V||E|)$

Bellman-Ford Algorithm - Example



Bellman-Ford Algorithm - Example



Loop invariant: $d[]$ is the shortest path labels over paths that contain at most $i - 1$ edges

Shortest Paths in DAGs

- Topological sort the graph
- Relax all nodes in the topologically sorted order
- One round of relaxation instead of $|V| - 1$
- running time $O(|E|)$