

Lecture 30: Graph Algorithms

Agenda:

- Single-source shortest paths
- Dijkstra's algorithm for non-negatively weighted case

Reading:

- Textbook pages 580 – 587, 595 – 601

Shortest path problems:

- BFS recall: outputs every s -to- v shortest path
 - s — start vertex
 - v — reachable vertex from s (residing in a same connected component)
 - shortest — $\#$ edges
 - running time $\Theta(n + m)$

- BFS solves the single-source-shortest-path problem on **undirected unweighted graphs**

Single-Source-Shortest-Path (SSSP) problem: given a source s , find out for all vertices their shortest paths from s

- Variants:
 - single source vs. all pairs
 - graphs: undirected vs. directed
 - edges: unweighted vs. weighted
 - edge weights: non-negative vs. may have negative weights
 - digraphs: acyclic vs. may have di-cycles

Note: if there is no path, the distance is set to ∞ ...

- 1. SSSP problem on non-negatively weighted digraphs
Dijkstra's algorithm (today)
 2. SSSP problem on weighted digraphs
Bellman-Ford's algorithm (next lecture)

Dijkstra's SSSP algorithm:

- $d[v]$ — weight of the shortest path from source s to v
if no such path, set to ∞
- Idea in Dijkstra's algorithm:
 - greedily grows an SSSP tree
 - ensures that when adding a vertex, its shortest path in the current (induced) subgraph is determined
 - records for every non-tree vertex v its best parent tree vertex $p[v]$

Note: very similar to Prim's MST algorithm (the min-priority queue implementation)

- Pseudocode (use $d[v]$ as the key):

```

procedure dijkstra( $G, w, s$ )           ** $G = (V, E)$ 

  for each  $v \in V(G)$  do                 **initialization
     $d[v] \leftarrow \infty$ 
     $p[v] \leftarrow \text{NIL}$ 
   $d[s] \leftarrow 0$ 
   $Q \leftarrow V(G)$ 
  while  $Q \neq \emptyset$  do
     $u \leftarrow \text{ExtractMin}(Q)$        ** $s$  dequeued first
    for each  $v \in \text{Adj}[u]$  do
      if  $d[u] + w(u, v) < d[v]$  then
        **update  $v$ , no matter if  $v \in Q$ 
         $p[v] \leftarrow u$ 
        decrease-key( $v, d[u] + w(u, v)$ )
        ** $d[v] \leftarrow d[u] + w(u, v)$ 

```

Dijkstra's SSSP algorithm vs. Prim's MST algorithm:

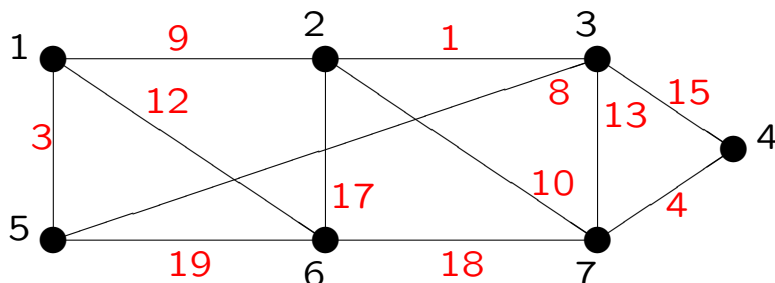
- procedure primMST(G, w, r) ** $G = (V, E)$

 for each $v \in V(G)$ do **initialization
 $key[v] \leftarrow \infty$
 $p[v] \leftarrow \text{NIL}$
 $key[r] \leftarrow 0$
 $Q \leftarrow V(G)$
 while $Q \neq \emptyset$ do
 $u \leftarrow \text{ExtractMin}(Q)$ ** r dequeued first
 for each $v \in \text{Adj}[u]$ do
 if $v \in Q$ && $w(u, v) < key[v]$ then
 **update v
 $p[v] \leftarrow u$
 decrease-key($v, w(u, v)$)
 ** $key[v] \leftarrow w(u, v)$
-
- procedure dijkstra(G, w, s) ** $G = (V, E)$

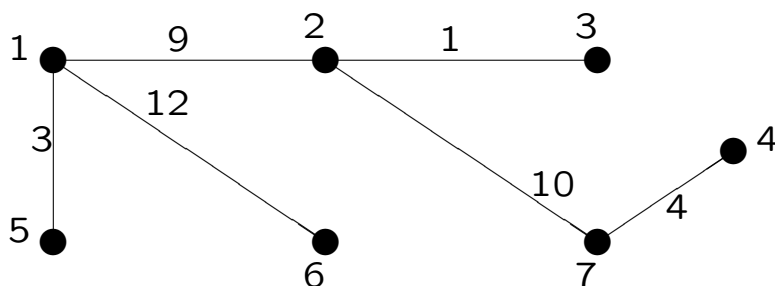
 for each $v \in V(G)$ do **initialization
 $d[v] \leftarrow \infty$
 $p[v] \leftarrow \text{NIL}$
 $d[s] \leftarrow 0$
 $Q \leftarrow V(G)$
 while $Q \neq \emptyset$ do
 $u \leftarrow \text{ExtractMin}(Q)$ ** s dequeued first
 for each $v \in \text{Adj}[u]$ do
 if $d[u] + w(u, v) < d[v]$ then
 **update v , no matter if $v \in Q$
 $p[v] \leftarrow u$
 decrease-key($v, d[u] + w(u, v)$)
 ** $d[v] \leftarrow d[u] + w(u, v)$

Dijkstra's SSSP algorithm — an example:

- Input graph G :



- $\text{dijkstra}(G, 1)$:



- $\text{dijkstra}(G, 1)$ trace:

$d[v]/p[v]$	1	2	3	4	5	6	7
	0/NIL	∞ /NIL	∞ /NIL	∞ /NIL	∞ /NIL	∞ /NIL	∞ /NIL
1 dequeued	0/NIL	9/1	∞ /NIL	∞ /NIL	3/1	12/1	∞ /NIL
5 dequeued	0/NIL	9/1	11/5	∞ /NIL	3/1	12/1	∞ /NIL
2 dequeued	0/NIL	9/1	10/2	∞ /NIL	3/1	12/1	19/2
3 dequeued	0/NIL	9/1	10/2	25/3	3/1	12/1	19/2
6 dequeued	0/NIL	9/1	10/2	25/3	3/1	12/1	19/2
7 dequeued	0/NIL	9/1	10/2	23/7	3/1	12/1	19/2

Dijkstra's SSSP algorithm — analysis:

- Applies to undirected graphs too

See the last example :-)

- Running time:

Same as the running time for Prim's MST algorithm

— $\Theta(m \log n)$, assuming adjacency list graph representation and min-priority queue implemented by a heap

- Correctness:

Let $S = V - Q$

(while) Loop Invariant: for every $v \in S$, $d[v]$ records the weight of the shortest path from s to v in graph G

Proof:

– initialization (S is empty):

– maintenance:

Exercise: fill in the detail

– termination: S becomes V , so LI implies that for every v , $d[v]$ records the weight of the shortest path from s to v in graph G

Dijkstra's SSSP algorithm — proof of maintenance:

- Maintenance (vertex u dequeued)

$dist[u]$ — weight of a shortest path from s to u in G :

— must show that at the end of loop body, $d[u] = dist[u]$

Let $P = (s, v_1, v_2, \dots, v_{k-1}, u)$ be any shortest path from s to u in graph G :

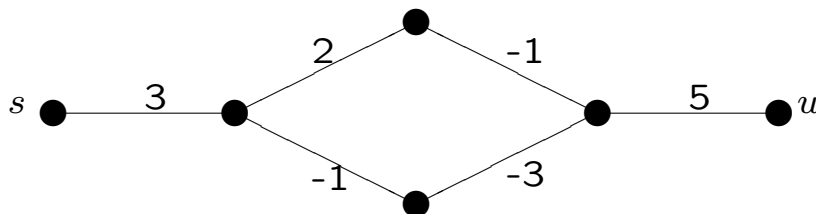
- y — first vertex in P but not in S
 - x — the vertex before y in P
 - $dist[y] \leq dist[u]$ — y on the path
 - $d[y] \geq d[u]$ — min-priority queue
 - $d[y] = dist[y]$ — since $x \in S$
 - conclusion: $d[u] \leq dist[u]$
- Question: why Dijkstra's algorithm does NOT apply to negative weights?

We fail to claim:

— $dist[y] \leq dist[u]$ — y on the path

- Another problem with negative weights:

Suppose there is a (direct/undirected) cycle with a negative weight



What is the weight of a shortest path from s to u ???

Lecture 30: Graph Algorithms

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	what is SSSP ???
☐	☐	☐	shortest path problem variants
☐	☐	☐	Dijkstra's algorithm: idea
☐	☐	☐	execution, correctness, & analysis