

Lecture 31: Graph Algorithms

Agenda:

- Single-source shortest paths
- Bellman-Ford's algorithm
- Floyd-Warshall's algorithm

Reading:

- Textbook pages 588 – 592, 629-632

Dijkstra's SSSP algorithm (recall):

- $d[v]$ — 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( $G, w, s$ )           **digraph  $G = (V, E)$ 

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( $v, d[u] + w(u, v)$ )
            ** $d[v] \leftarrow d[u] + w(u, v)$ 

```

Bellman-Ford's SSSP algorithm for the general case:

- General case — edge weights could be negative
- Output:
 1. if there is a negative weight cycle, report it
 2. otherwise report all the $dist[u]$ values and associated paths
- [Idea](#) in the algorithm:

If there is no negative weight cycle reachable from s , then every s -to- u shortest path contains at most $n - 1$ edges; also is true that when all s -to- u shortest paths are discovered, for every edge (u, v) there must be $d[v] \leq d[u] + w(u, v)$.

It follows that $d[v]$ can be reduced $n - 1$ times in order to have value $dist[v]$, but no more.

- Pseudocode:

```

procedure bellman-ford( $G, w, s$ )           **digraph  $G = (V, E)$ 

  for each  $v \in V(G)$  do                     **initialization
     $d[v] \leftarrow \infty$ 
     $p[v] \leftarrow \text{NIL}$ 
   $d[s] \leftarrow 0$ 

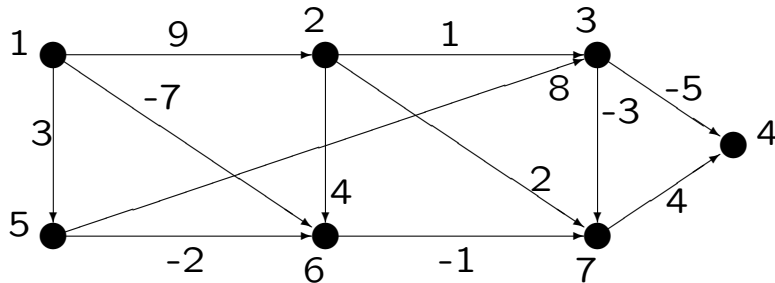
  for  $i \leftarrow 1$  to  $n - 1$                  ** $n = |V(G)|$ 
    for each edge  $(u, v) \in E(G)$  do
      if  $d[u] + w(u, v) < d[v]$  then          **update  $d[v]$ 
         $p[v] \leftarrow u$ 
         $d[v] \leftarrow d[u] + w(u, v)$ 

  for each edge  $(u, v) \in E(G)$  do
    if  $d[u] + w(u, v) < d[v]$  then          **there is a negative cycle
      return FALSE
  return TRUE

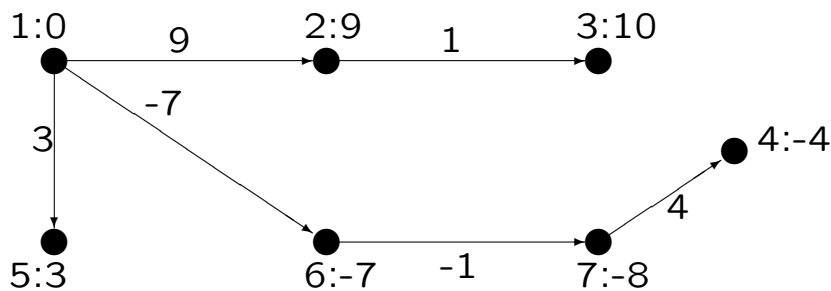
```

Bellman-Ford's SSSP algorithm — analysis:

- An example:



`bellman-ford( $G, 1$ ):`



- Correctness: textbook pages 589 – 591
- Running time:
 1. initialization: $\Theta(n)$
 2. updating $d[v]$: $\Theta(n \times m)$
 3. checking existence of negative cycles: $\Theta(m)$

Conclusion: $\Theta(nm)$ time (assuming adjacency list graph representation)

Floyd-Warshall's algorithm for All-Pairs-Shortest path:

- General case (weights can be negative but no negative cycle);
- Output: shortest path between “every” pair u, v of vertices.
- Idea: Use dynamic programming.
Define $d[i, j, k]$ to be the length of shortest path from i to j for which all intermediate vertices are in $\{1, \dots, k\}$, for every $1 \leq i, j \leq n$ and $0 \leq k \leq n$.
- When $k = 0 \implies$ no intermediate vertex, so: $d[i, j, 0] = w(i, j)$.
- For general $k \geq 1$:
 - If the path does not contain k , inter. vertices only from $\{1, \dots, k-1\}$: $d[i, j, k-1]$.
 - If the path contains k , it has two parts: one goes from i to k with intermediate only from $\{1, \dots, k-1\}$, followed by a path from k to j using only from $\{1, \dots, k-1\}$: $d[i, k, k-1] + d[k, j, k-1]$.
- Recurrence:

$$d[i, j, k] = \min \begin{cases} w(i, j) & k = 0 \\ d[i, k, k-1] + d[k, j, k-1] & k \geq 1 \end{cases}$$
- We compute the table bottom-up, starting from smaller values of k to larger values.

Floyd-Warshall's algorithm:

- Pseudocode:

```

procedure Floyd-Warshall( $G$ )
  for  $i \leftarrow 1$  to  $n$  do
    for  $j \leftarrow 1$  to  $n$  do
       $d[i, j, 0] = w(i, j)$ 
       $b[i, j] = 0$           **  $b[i, j]$  keeps the break point vertex
  for  $k \leftarrow 1$  to  $n$  do
    for  $i \leftarrow 1$  to  $n$  do
      for  $j \leftarrow 1$  to  $n$  do
        if  $d[i, k, k-1] + d[k, j, k-1] < d[i, j, k-1]$  then
           $d[i, j, k] = d[i, k, k-1] + d[k, j, k-1]$ 
           $b[i, j] = k$ 
        else
           $d[i, j, k] = d[i, j, k-1]$ 

```

- To print the actual path between i and j :

```

procedure Print-FW( $i, j$ )
  if  $b[i, j] = 0$  then
    Print 'edge',  $i$ , 'to',  $j$ 
  else
    Print-FW( $i, b[i, j]$ )
    Print-FW( $b[i, j], j$ )

```

- Running time: $\Theta(n^3)$, better than n run of Bellman-Ford or Dijkstra (one for every vertex as a source).

Lecture 31: Graph Algorithms

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	SSSP problem
☐	☐	☐	shortest path problem variants
☐	☐	☐	Dijkstra's algorithm: idea
☐	☐	☐	where Dijkstra's fails? why?
☐	☐	☐	Bellman-Ford's algorithm
☐	☐	☐	Floyd-Warshall's algorithm