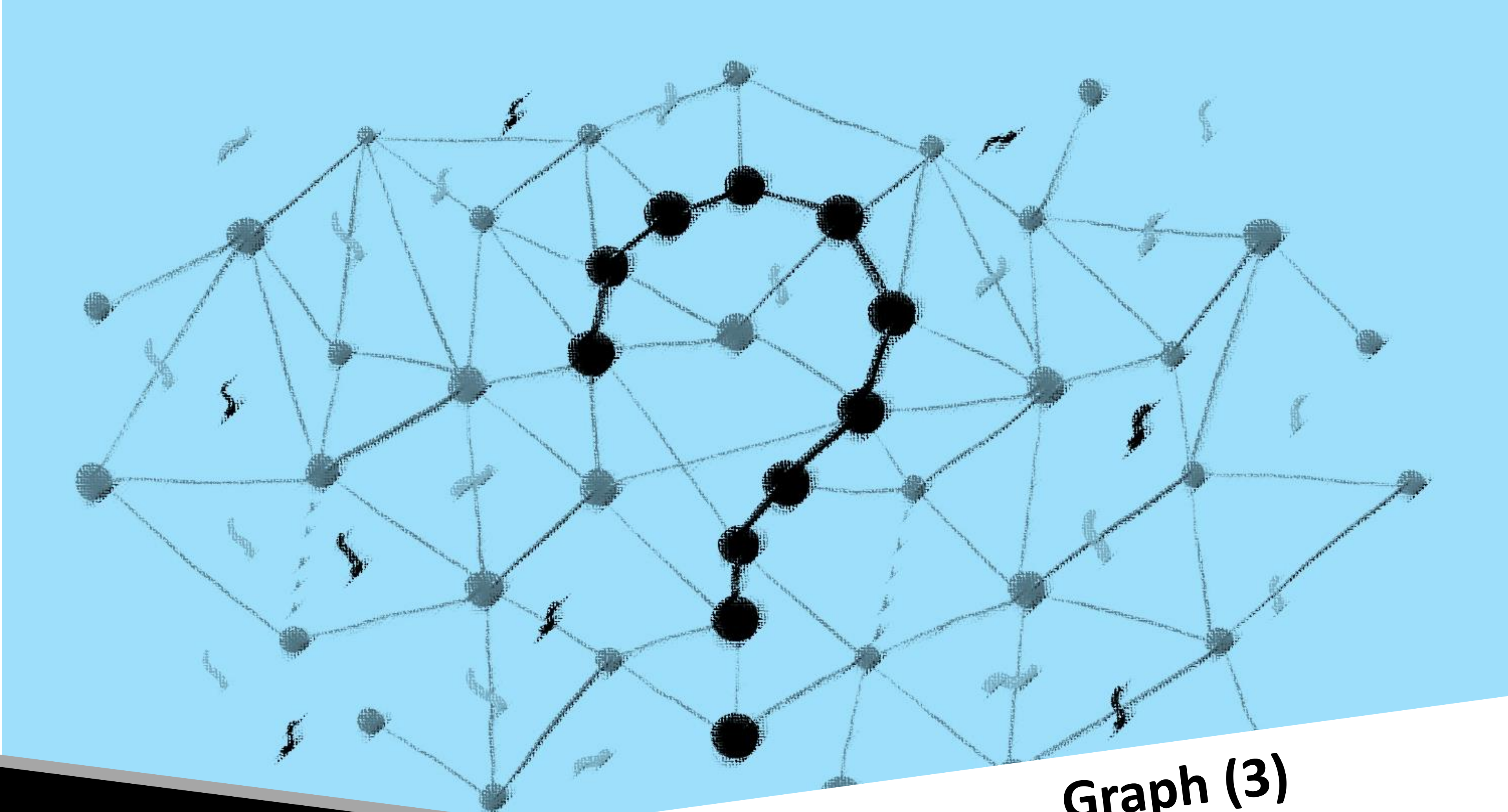


Graph (3)

Dec 6th, 2018

Algorithm Design and Analysis

Yun-Nung (Vivian) Chen http://ada.miulab.tw

國立臺灣大學
National Taiwan University

Slides credited from Hsueh-I Lu & Hsu-Chun Hsiao

Announcement

- Homework 3 released
  - Due on 12/13 (Thur) 14:20 (one week only)
- Mini-HW 9 released
  - Due on 12/13 (Thur) 14:20
- Homework 4 released
  - Due on 1/3 (Thur) 14:20 (four weeks later)

Frequently check the website for the updated information!

Mini-HW 9

Following is the implementation of a queue using 2 stacks. (assuming that the capacity of both stacks are unlimited)

```
enQueue(Q, x) {
   stack1.push(x);
}

deQueue(Q) {
   if ( stack2.empty() ) {
      while ( !stack1.empty() ) {
         stack2.push( stack1.top() );
         stack1.pop();
      }
   }
   ans = stack2.top();
   stack2.pop();
   return ans;
}
```

Please answer the following questions:

1. What is the exact cost of a single ***enQueue(Q, x)*** operation ? (10%)
2. What is the exact cost of a single ***deQueue(Q)*** operation ? (10%)
3. What is the amortized cost of Q, considering ***a sequence of n operations*** ? Please choose one of the methods mentioned in class (aggregate/accounting/potential) to show how you derive the answer. (80%)

Outline

- Single-Source Shortest Paths
  - Bellman-Ford Algorithm
  - Lawler Algorithm (SSSP in DAG)
  - Dijkstra Algorithm

5 Single-Source Shortest Paths

Textbook Chapter 24 – Single-Source Shortest Paths

Shortest Path Problem

- Input: a weighted, directed graph $G = (V, E)$
  - Weights can be arbitrary numbers, not necessarily distance
  - Weight function needs not satisfy triangle inequality
- Output: a minimal-cost path from s to t s.t. $\delta(s, t)$ is the minimum weight from s to t

- Problem Variants
  - Single-source shortest-path problem
  - Single-destination shortest-path problem
  - Single-pair shortest-path problem
  - All-pair shortest path problem

Cycles in Graph

- Can a shortest path contain a negative-weight edge?

  Yes.

- Can a shortest path contain a negative-weight cycle?

  Doesn't make sense.

- Can a shortest path contain a cycle?

  No.

Single-Source Shortest Path Problem

- Input: a weighted, directed graph $G = (V, E)$ and a source vertex s
- Output: a minimal-cost path from s to t, where $t \in V$

Shortest Path Tree

- Let $G = (V, E)$ be a weighted, directed graph with no negative-weight cycles reachable from s
- A shortest path tree $G' = (V', E')$ of s is a subgraph of G s.t.
  - V' is the set of vertices reachable from s in G
  - G' forms a rooted tree with root s
  - For all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G

Shortest Path Tree Problem

- Input: a weighted, directed graph $G = (V, E)$ and a vertex s
- Output: a tree T rooted at s s.t. the path from s to u of T is a shortest path from s to u in G

Problem Equivalence

- The shortest path tree problem is equivalent to finding the minimal cost $\delta(s, u)$ from s to each node u in G
  - The minimal cost from s to u in G is the length of any shortest path from s to u in G

"equivalence": a solution to either problem can be obtained from a solution to the other problem in linear time

Shortest Path Tree Problem = Single-Source Shortest Path Problem

12 Bellman-Ford Algorithm

Textbook Chapter 24.1 – The Bellman-Ford algorithm

Bellman and Ford

Richard Bellman, 1920~1984

- Norbert Wiener Prize in Applied Mathematics, 1970
- Dickson Prize, Carnegie-Mellon University, 1970
- John von Neumann Theory Award, 1976.
- IEEE Medal of Honor, 1979,
- Fellow of the American Academy of Arts and Sciences, 1975.
- Membership in the National Academy of Engineering, 1977

Lester R. Ford, Jr. 1927~2017

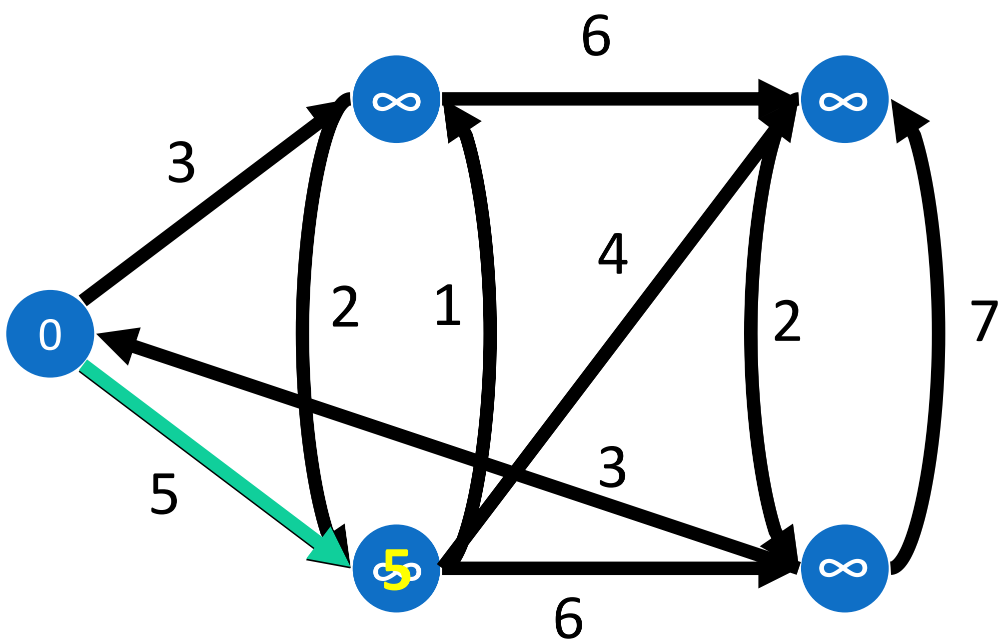
- Proved the algorithm before Bellman
- An important contributor to the theory of network flow.

Bellman-Ford Algorithm

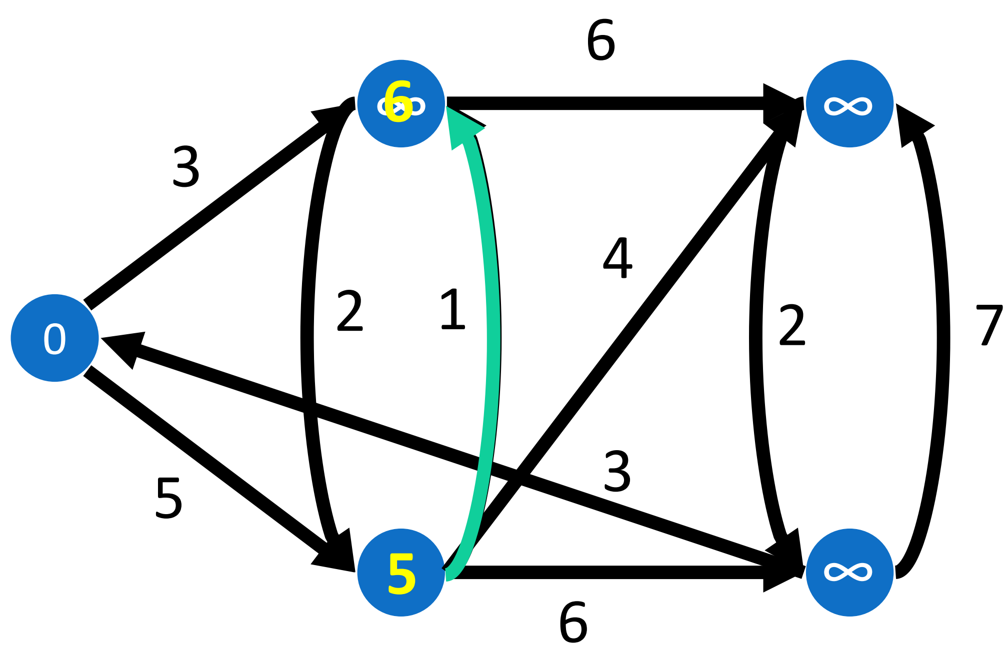
- Idea: estimate the value of $d[u]$ to approximate $\delta(s,u)$
- Initialization
  - Let $d[u] = \infty$ for $u \in G$
  - Let $d[s] = 0$
- Repeat the following step for sufficient number of phases
  - For each edge $(u,v) \in E$, relax edge (u,v)
  - Relaxing: If $d[v] > d[u] + w(u,v)$, let $d[v] = d[u] + w(u,v)$

→ improve the estimation of $d[u]$

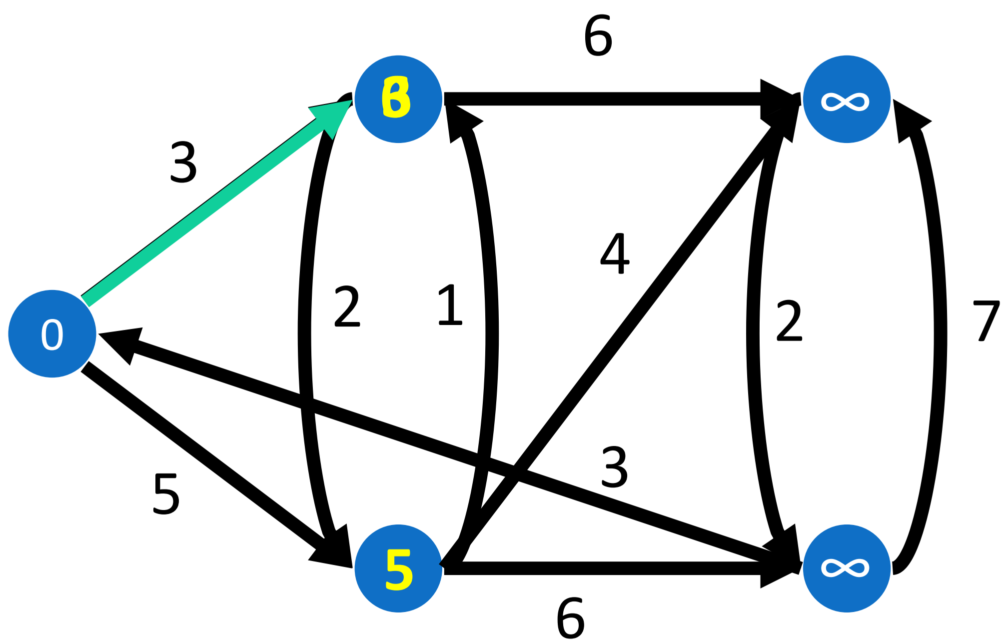
Bellman-Ford Algorithm Illustration



Bellman-Ford Algorithm Illustration

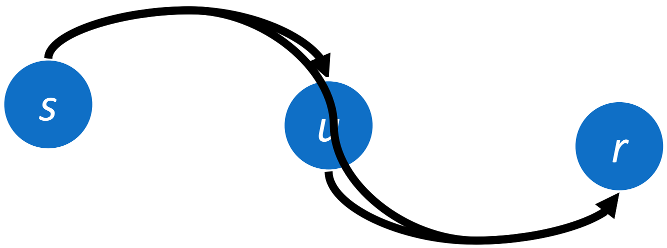


Bellman-Ford Algorithm Illustration



Bellman-Ford Algorithm Correctness

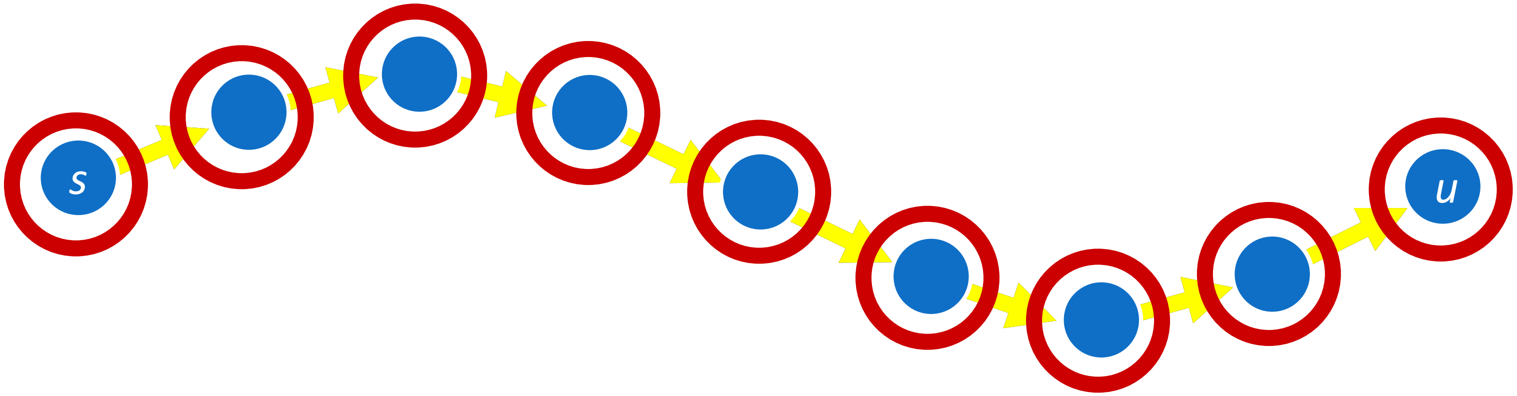
- Observation: let P be a shortest path from s to r
  - For any vertex u in P, the subpath of P from s to u has to be a shortest path from s to u → optimal substructure
  - For any edge (u, v) in P, if $d[u] = \delta(s, u)$, then $d[v] = \delta(s, v)$ also holds after relaxing edge (u, v)



- If G contains no negative cycles, then each node u has a shortest path from s to u that has at most $n - 1$ edges
- From observation, after the first i phases of improvement via relaxation, the estimation of $d[u]$ for the first $i + 1$ nodes u in the path is precise ($= \delta(s, u)$)

→ $n - 1$ phases

Bellman-Ford Algorithm Correctness



Bellman-Ford Time Complexity

```
BELLMAN-FORD(G, w, s)
  INITIALIZATION(G, s)
  for i = 1 to |G.V| - 1        n − 1 times
    for (u, v) in G.E           O(m)
      RELAX(u, v, w)
```

$n-1$ times; $O(m)$

```
INITIALIZATION(G, s)
  for v in G.V
    v.d = ∞
    v.π = NIL
  s.d = 0
```

$O(n)$

```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
    // DECREASE-KEY
    v.d = u.d + w(u, v)
    v.π = u
```

$O(1)$

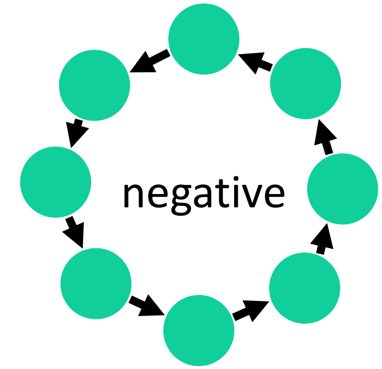
- Time complexity: $O(mn)$

How to do if there is a negative cycle in the graph?

Negative Cycle Detection

- Q: How do we know G has negative cycles?
- A: Using another phase of improvement via relaxation
  - Run another phase of improving the estimation of $d[u]$ for each vertex $u \in V$ via relaxing all edges E
  - If in the n-th phase, there are still some $d[u]$ being modified, we know that G has negative cycles

Negative Cycle Detection



If there exists a negative cycle in G, in the n-th phase, there are still some $d[u]$ being modified.

- Proof by contradiction
  - Let C be a negative cycle of k nodes $v_1, v_2, \dots, v_k$ $(v_{k+1} = v_1)$
  - Assume $d[v_i]$ for all $1 \le i \le k$ are not changed in a phase of improvement, then for $1 \le i \le k$

$$d[v_{i+1}] \le d[v_i] + w(v_i, v_{i+1})$$

  - Summing all k inequalities, the sum of edge weights of C is nonnegative

$$\sum_{i=1}^{k} d[v_{i+1}] \le \sum_{i=1}^{k} d[v_i] + \sum_{i=1}^{k} w(v_i, v_{i+1}) \Rightarrow 0 \le \sum_{i=1}^{k} w(v_i, v_{i+1})$$

Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
  INITIALIZATION(G, s)
  for i = 1 to |G.V| - 1        n − 1 times
    for (u, v) in G.E           O(m)
      RELAX(u, v, w)
  for (u, v) in G.E
    if v.d > u.d + w(u, v)
      return FALSE
  return TRUE
```

negative cycle detection

```
INITIALIZATION(G, s)
  for v in G.V
    v.d = ∞
    v.π = NIL
  s.d = 0
```

$O(n)$

```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
    // DECREASE-KEY
    v.d = u.d + w(u, v)
    v.π = u
```

$O(1)$

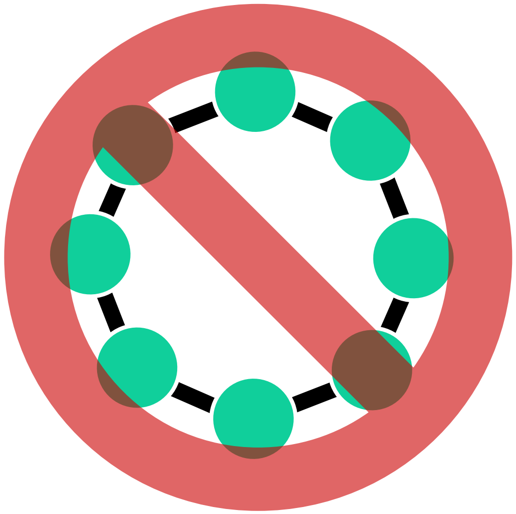
- Time complexity: $O(mn)$
- Finding a shortest-path tree of G: $O(mn) + O(m+n) = O(mn)$

24 Lawler Algorithm

Textbook Chapter 24.2 – Single-source shortest paths in directed acyclic graphs

Single-Source Shortest Path Problem

- Input: a weighted, directed, and acyclic graph $G = (V, E)$ and a source vertex s
- Output: a shortest-path distance from s to t, where $t \in V$

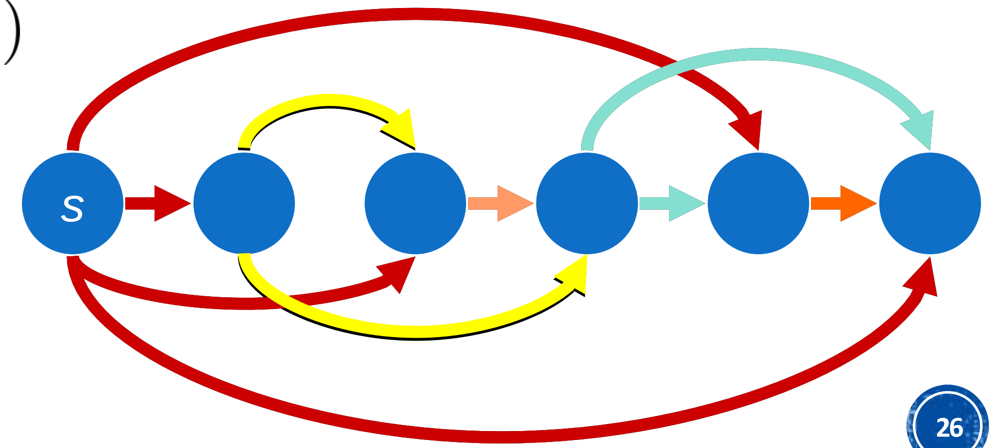


No negative cycle!

Lawler Algorithm

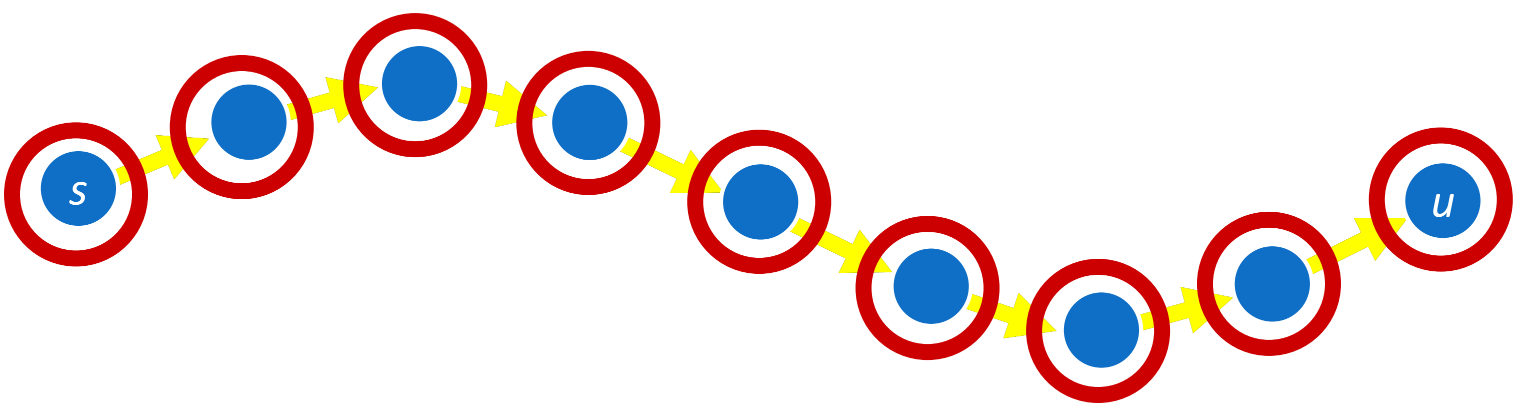
- Idea: one phase relaxation
- Perform a topological sort in linear time on the input DAG
- For $i = 1$ to n
  - Let v_i be the i-th node in the above order
  - Relax each outgoing edge (v_i, u) from v_i

Time complexity: $O(m + n)$



Lawler Algorithm Correctness

- Assume this is a shortest path from s to u
- If we follow the order from topological sort to relax the vertices' edges, in this shortest path, the left edge must be relaxed before the right edge
- One phase of improvement is enough

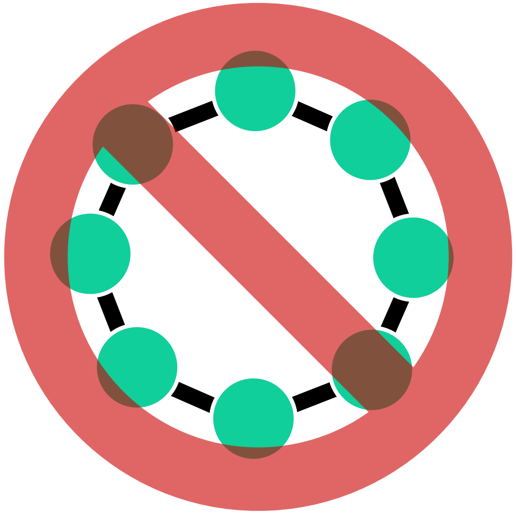


28 Dijkstra's Algorithm

Textbook Chapter 24.3 – Dijkstra's algorithm

Single-Source Shortest Path Problem

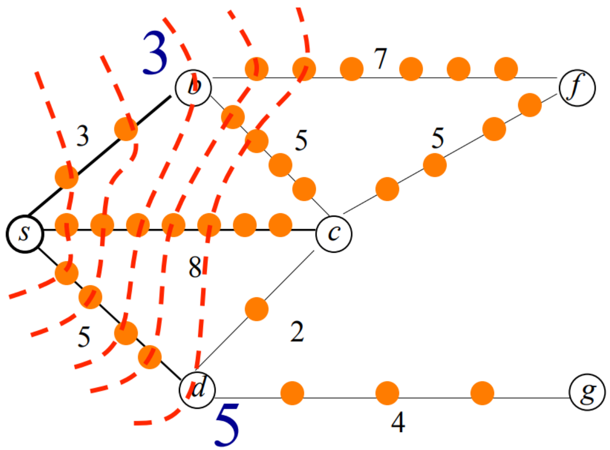
- Input: a non-negative weighted, directed, graph $G = (V, E)$ and a source vertex s
- Output: a shortest-path distance from s to t, where $t \in V$



No negative cycle!

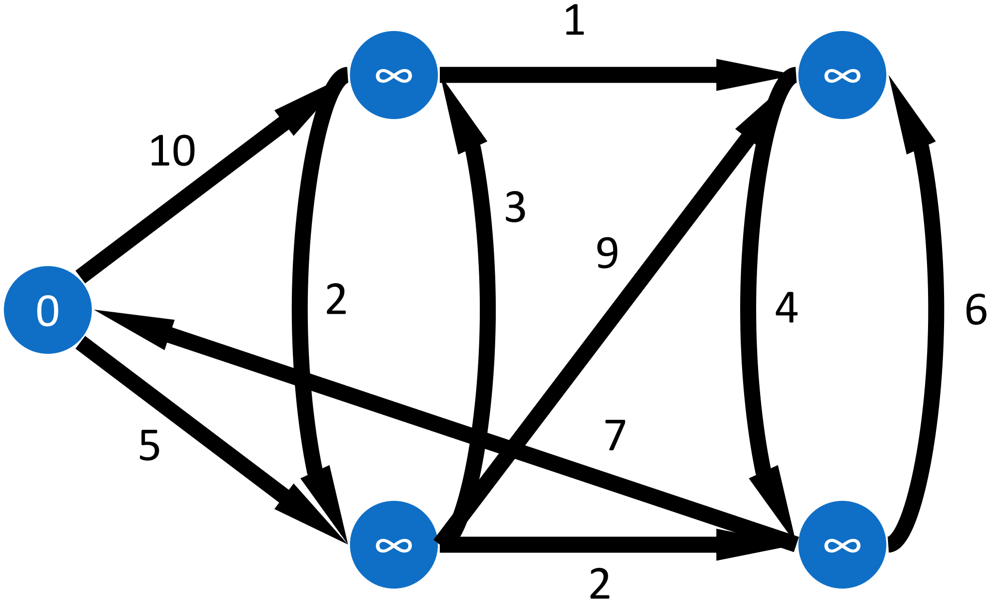
Dijkstra’s Algorithm

- Idea: BFS finds shortest paths on unweighted graph by expanding the search frontier

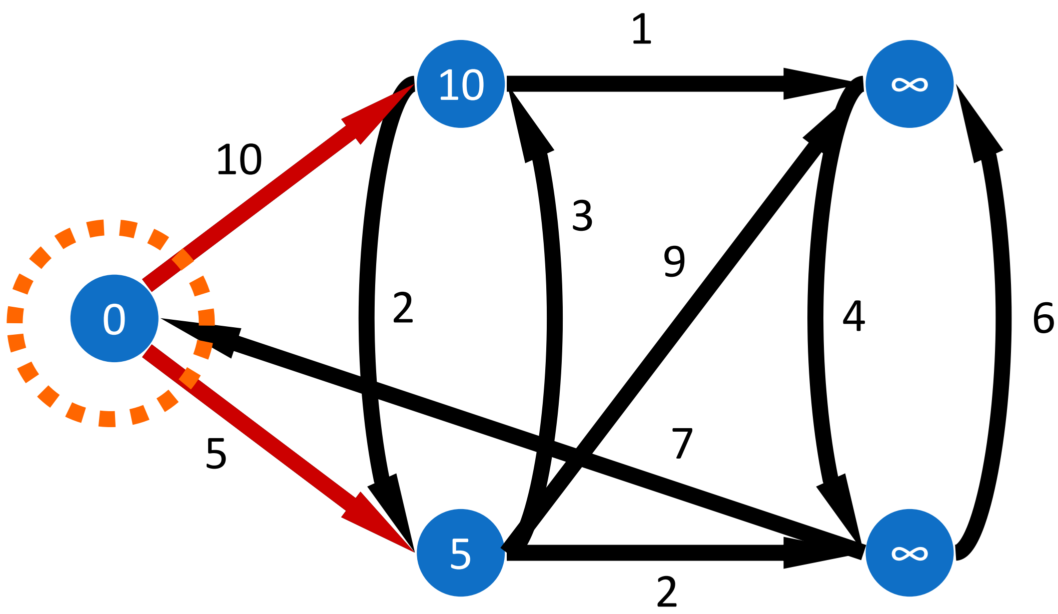


- Initialization
- Loops for n iterations, where each iteration
  - relax outgoing edges of an unprocessed node u with minimal $d[u]$
  - marks u as processed

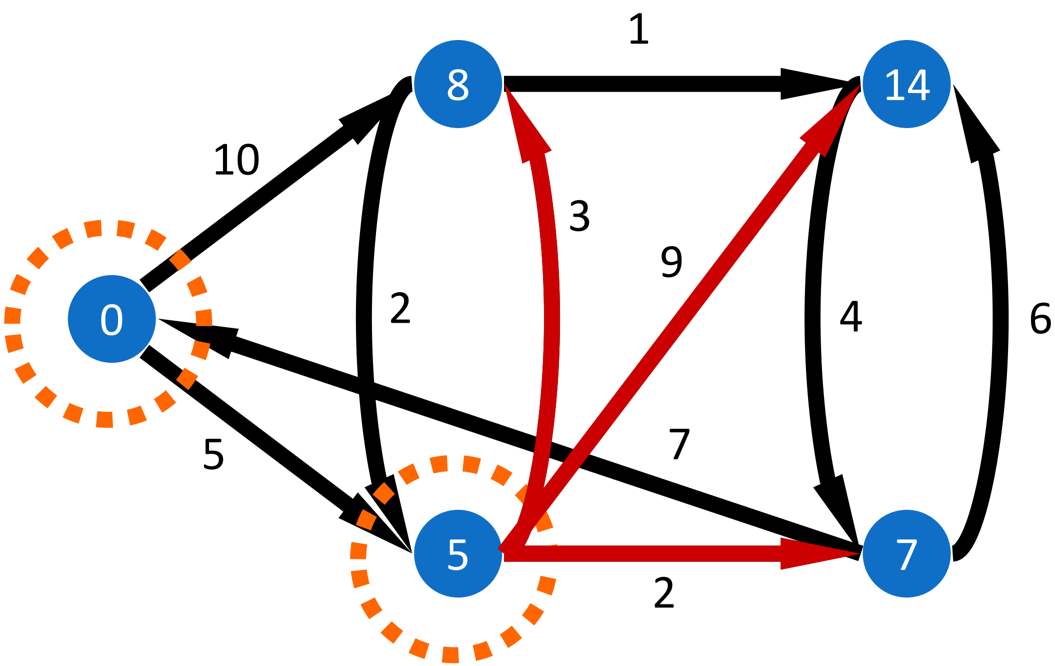
Dijkstra's Algorithm Illustration



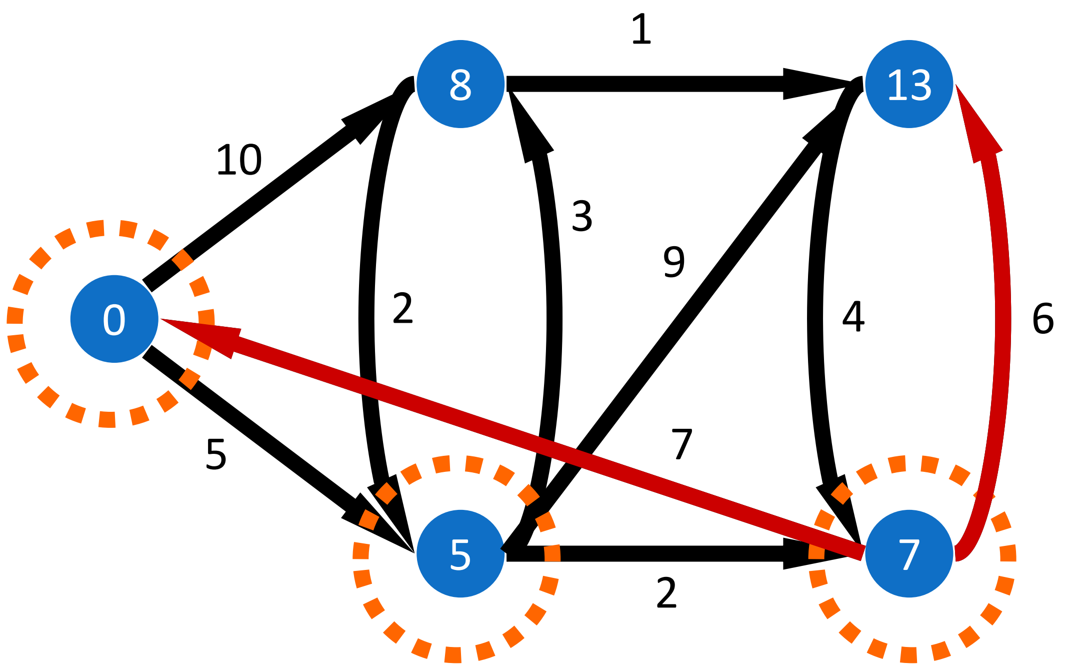
Dijkstra's Algorithm Illustration



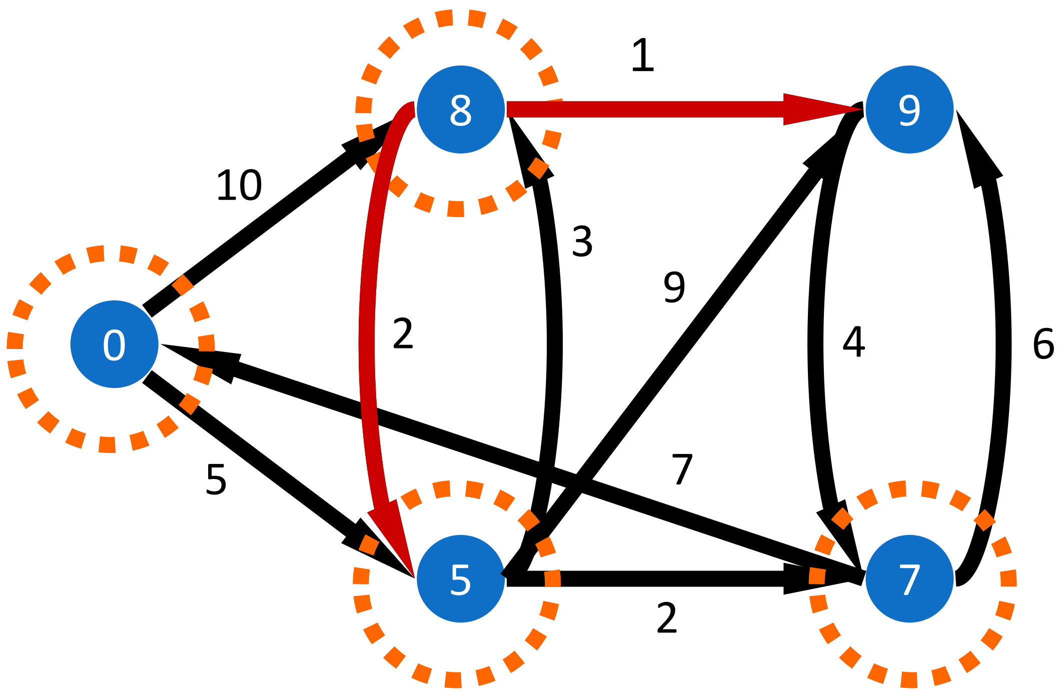
Dijkstra's Algorithm Illustration



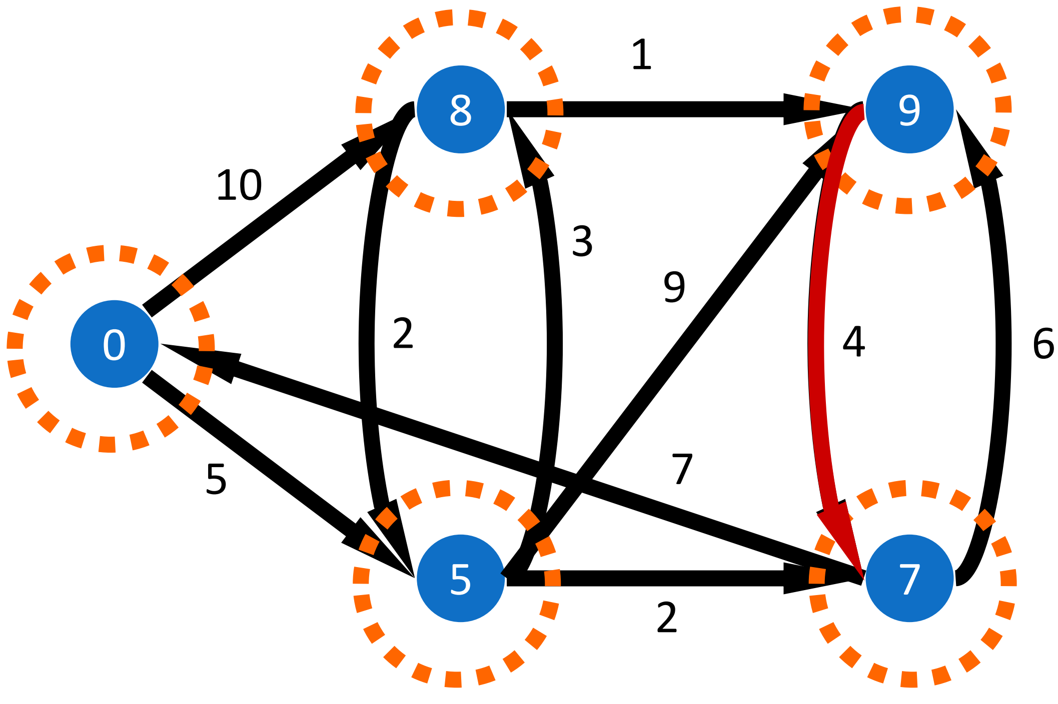
Dijkstra's Algorithm Illustration



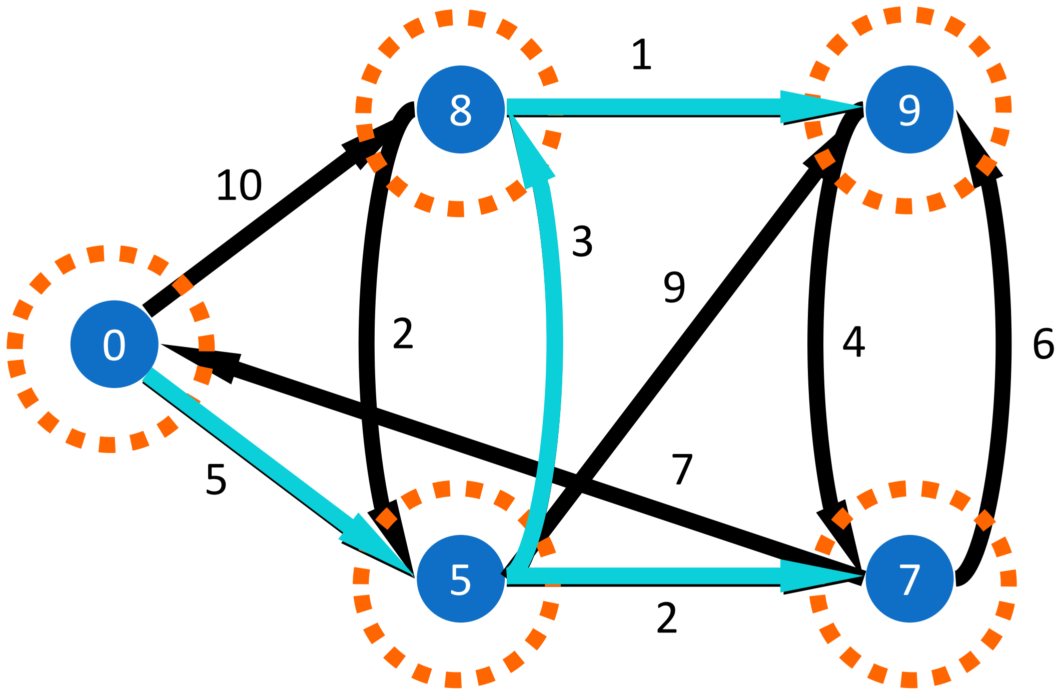
Dijkstra's Algorithm Illustration



Dijkstra’s Algorithm Illustration



Dijkstra's Algorithm Illustration



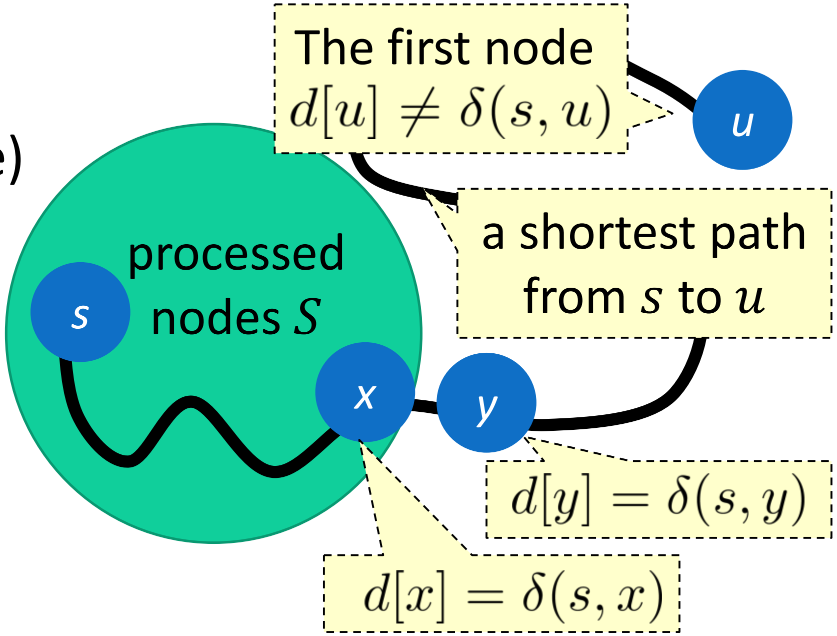
Dijkstra's Algorithm Correctness

The vertex selected by Dijkstra's algorithm into the processed set must precise estimation of its shortest path distance.

- Prove by contradiction
  - Assume u is the first vertex for being processed (minimal distance)
  - Let a shortest path P from s to u,
    - x is the last vertex in P from S
    - y is the first vertex in P not from S
  - $d[y] = \delta(s, y)$ because (x, y) is relaxed when putting x into S

$$d[u] > \delta(s,u) \geq \delta(s,y) = d[y]$$

  - y should be processed before u, contradiction.



Dijkstra's Time Complexity

```
DIJKSTRA(G, w, s)
  INITIALIZATION(G, s)
  S = empty
  Q = G.v // INSERT              n times
  while Q ≠ empty                n times
    u = EXTRACT-MIN(Q)           O(n)
    S = SU{u}
    for v in G.adj[u]            m times
      RELAX(u, v, w)
```

```
INITIALIZATION(G, s)
  for v in G.V      ┐
    v.d = ∞         │
    v.π = NIL       ├ O(n)
  s.d = 0           ┘
```

```
RELAX(u, v, w)                      O(1)
  if v.d > u.d + w(u, v)
    // DECREASE-KEY
    v.d = u.d + w(u, v)
    v.π = u
```

- Min-priority queue
  - INSERT: $O(1)$
  - EXTRACT-MIN: $O(n)$
  - DECREASE-KEY: $O(1)$
- Total complexity: $O(n^2 + m)$

Dijkstra's Time Complexity

```
DIJKSTRA(G, w, s)
  INITIALIZATION(G, s)
  S = empty
  Q = G.v // INSERT            O(n)
  while Q ≠ empty              n times
    u = EXTRACT-MIN(Q)         O(log n)
    S = S∪{u}
    for v in G.adj[u]          m times
      RELAX(u, v, w)
```

```
INITIALIZATION(G, s)
  for v in G.V     ┐
    v.d = ∞        │
    v.π = NIL      ├ O(n)
  s.d = 0          ┘
```

```
RELAX(u, v, w)                      O(1)
  if v.d > u.d + w(u, v)
    // DECREASE-KEY
    v.d = u.d + w(u, v)
    v.π = u
```

- Fibonacci heap (Textbook Ch. 19)
  - BUILD-MIN-HEAP: $O(n)$
  - EXTRACT-MIN: $O(\log n)$ (amortized)
  - DECREASE-KEY: $O(1)$ (amortized)
- Total complexity: $O(m + n \log n)$

Concluding Remarks

- Single-Source Shortest Paths
  - Bellman-Ford Algorithm (general graph and weights)
    - $O(mn)$ and detecting negative cycles
  - Lawler Algorithm (acyclic graph)
    - $O(m+n)$
  - Dijkstra Algorithm (non-negative weights)
    - $O(m+n\log n)$ with Fibonacci heap

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada.miulab.tw

Email: ada-ta@csie.ntu.edu.tw