CSCE 411: Design and Analysis of Algorithms

Lecture 14: Single-Source Shortest Path

Date: March 6 *Nate Veldt, updated by Samson Zhou*

Course Logistics

- Reading from Chapter 24 this week, especially sections 24.1, and 24.3
- Homework 5 due Friday (tomorrow)

1 The Single-Source (weighted) Shortest Paths Problem

Consider a directed and weighted graph $G = (V, E, w)$, where $w \colon E \to \mathbb{R}$ maps edges to weights.

Given a path $p = \{v_0, v_1, \ldots, v_k\}$ from $i = v_0$ to $j = v_k$,

the weight of the path p is given by:

Let $\mathcal{P}_{ij}$ be the set of paths from nodes i to j. The *shortest path weight* from i to j is:

$$\delta(i,j) = \begin{cases} & \\ \infty & \end{cases}$$

A ________________________ from u to v is a path p such that $\delta(u, v) = w(p)$.

Definition Given a *source* node $s \in V$, the single-source shortest paths problem (SSSP) seeks to find the shortest path weight $\delta(s, u)$ for every $u \in V$.

1.1 Shortest Path Properties

Triangle Inequality For any set of three nodes s, u, and v,

$$\delta(s,u) \leq \delta(s,v) + \delta(v,u)$$

Optimal Substructure

Lemma 1.1. *If $p = \{v_0, v_1, \ldots, v_k\}$ is a shortest path from v_0 to v_k, then all of its subpaths* ____

Cycles and shortest paths

Lemma 1.2. *: If $G = (V, E, w)$ has no negative edges, then*

1.2 Negative edges

Question 1. *Assume $G = (V, E, w)$ contains negative edges, and that it is strongly connected. If G contains a cycle where the sum of edge weights is negative, which of the following is true?*

A *The shortest path weight between every pair of nodes will be a finite negative number*

B *The shortest path weight between some (but not necessarily all nodes) will be a finite negative number*

C *The shortest path weight can be $-\infty$ for some pairs of nodes, but not necessarily all pairs of nodes*

D *The shortest path weight will be $-\infty$ between all pairs*

1.3 SSSP Algorithm Basics

Let s denote the starting node of an SSSP problem. In shortest path algorithms, we maintain two attributes for each $v \in V$:

Attribute	**Explanation**	**Initialization**	**Invariant**
$u.d$			
$u.\pi$			

Given values for these attributes, we can construct a *predecessor* graph:

- $V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$

If $v.d = \delta(s, v)$ and $v.\pi$ gives the predecessor of v in a shortest path from s to v, then $G_\pi = (V_\pi, E_\pi)$ is called a ______________________________.

1.4 Relaxation and Generic Algorithm

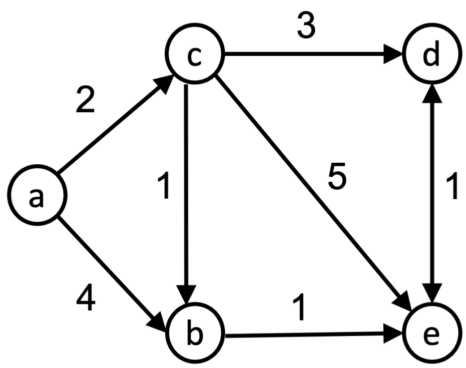
The key idea behind shortest path algorithms is to successively "relax" edges.

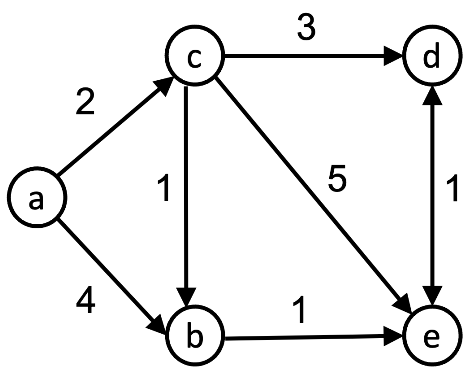
RELAX(u, v, w)
  if $v.d > u.d + w(u, v)$ **then**

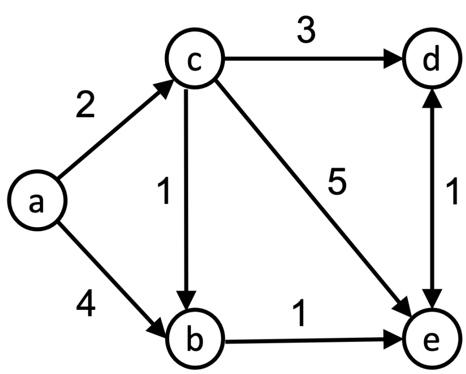
  end if

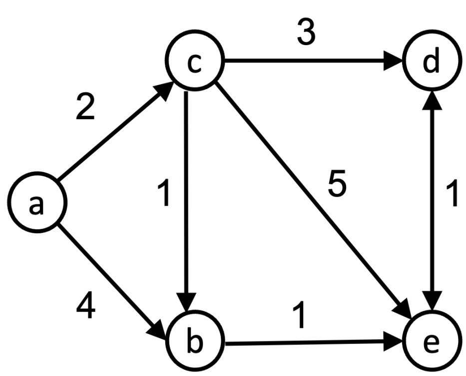
GENERICSSSP(G, w, s)
  INITIALIZE-SSSP(G, w, s)
  while There is some edge (u, v) with $v.d > u.d + w(u, v)$ **do**

  end while


3
2
1
5
1
4
1
a
b
c
d
e


3
2
1
5
1
4
1
a
b
c
d
e


3
2
1
5
1
4
1
a
b
c
d
e


3
2
1
5
1
4
1
a
b
c
d
e

2 The Bellman Ford Algorithm

Let's do a quick recap of what we have learned so far about the SSSP problem.

Cycles Let p be a cycle in a directed graph.

- Case 1: $w(p) \geq 0$: then I can get rid of the cycle and I have a better path
- Case 2: $w(p) < 0$: then all paths are of length $-\infty$ because we can traverse p as many times as we want to decrease the path weight.

If Case 2 is true, we would prefer to be aware of this, in which case we state that all shortest paths have weight $-\infty$ and we're done.
Otherwise, we do not seek out paths with cycles when solving the SSSP problem.

RELAX(u, v, w)
  if $v.d > u.d + w(u, v)$ **then**
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
  end if

GENERICSSSP(G, w, s)
  INITIALIZE-SSSP(G, w, s)
  while There is some edge (u, v) with $v.d > u.d + w(u, v)$ **do**
    RELAX(u, v, w)
  end while

Relaxations and Generic Algorithm Now we are ready to find a more concrete implementation of this approach, by answering the question:

2.1 Bellman-Ford Algorithm Pseudocode

```
BELLMAN-FORD(G, w, s)
  for v ∈ V do
      v.d = ∞
      v.π = NIL
  end for
  s.d = 0

  for i = 1 to |V| − 1 do
      for (u, v) ∈ E do
          RELAX(u, v, w)
      end for
  end for

  for (u, v) ∈ E do
      if v.d > u.d + w(u, v) then
          Return "the graph has a negative cycle"
      end if
  end for

  V_π = {v ∈ V : v.π ≠ NIL} ∪ {s}
  E_π = {(v.π, v): v ∈ V_π − {s}}
  Return (V_π, E_π) and {v.d: v ∈ V}
```

Question 2. *What is the runtime complexity of running the* BELLMAN-FORD *Algorithms?*

A $O(V+E)$

B $O(E \log V)$

C $O(E + V \log V)$

D $O(EV)$

BELLMAN-FORD(G, w, s)

  INITIALIZESSSP(G, w, s)

  for $i = 1$ to $|V| - 1$ **do**

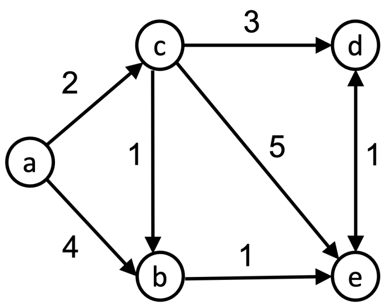
    for $(u, v) \in E$ **do**

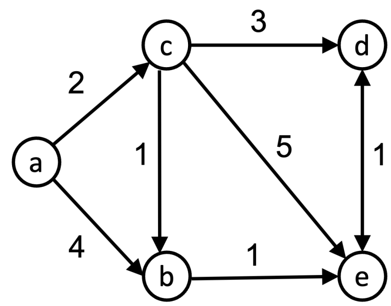
      RELAX(u, v, w)

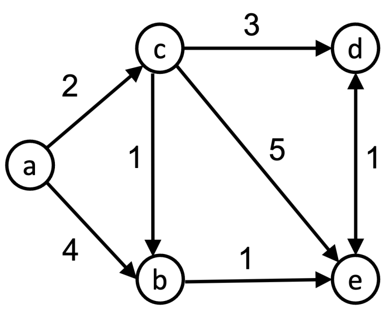
    end for

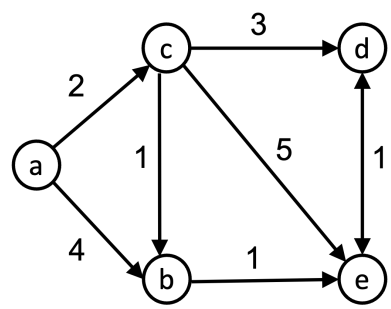
  end for

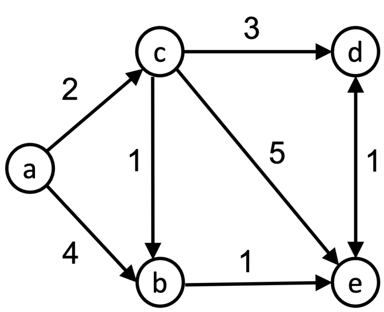
  Return $\{v.d\}$

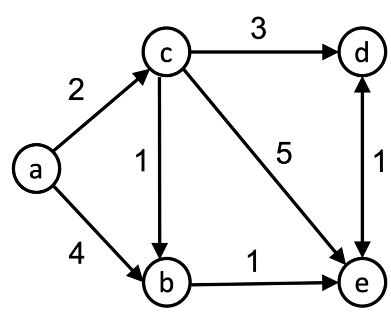


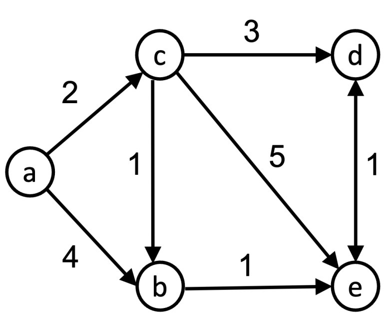


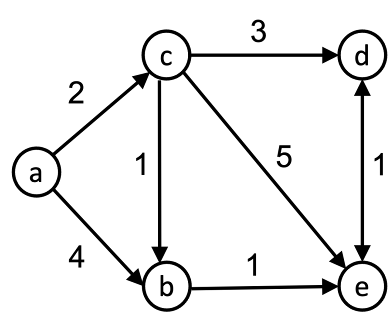












2.2 Correctness

Theorem 2.1. *Assuming that* $G = (V, E, w)$ *has no negative edges, when* BELLMAN-FORD *terminates,* $v.d$ *will equal* $\delta(s, v)$ *for every* $v \in V$*.*

2.3 Catching Negative Cycles

BELLMAN-FORD (Negative cycle check)

```
Do everything else...
for (u, v) ∈ E do
    if v.d > u.d + w(u, v) then
        Return an error, the graph has a negative cycle
    end if
end for
```

Theorem 2.2. *If $G = (V, E, w)$ has a negative weight cycle that is reachable from s, then the code above will produce an error*

3 Dijkstra's Algorithm

```
DIJKSTRA(G, w, s)
  INITIALIZE-SSSP(G, w, s)
  S = ∅
  Q = V
  while Q ≠ ∅ do
      u = EXTRACTMIN(Q)
      S = S ∪ {u}
      for v ∈ Adj[u] do
          RELAX(u, v, w)
      end for
  end while
```

This algorithm maintains:

- S = set of vertices for which $v.d = \delta(s, d)$
- Q = min-priority queue storing $v.d$ for all vertices in $V - S$

Question 3. *How many times does the while loop in Dijkstra's algorithm iterate?*

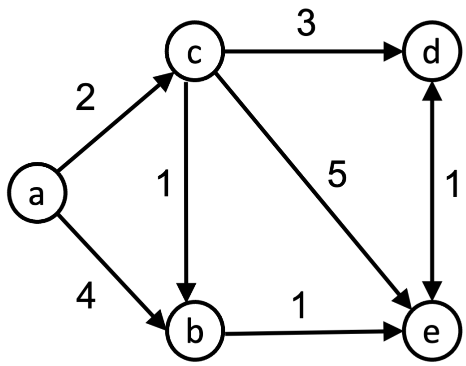
A $O(V)$ *times*

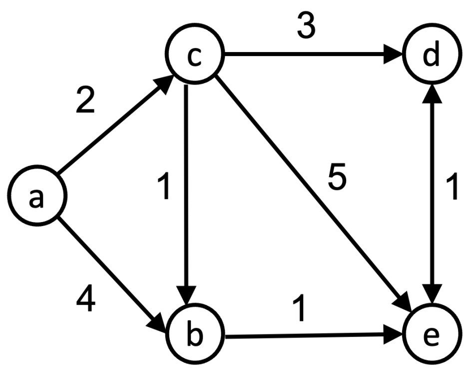
B $O(E)$ *times*

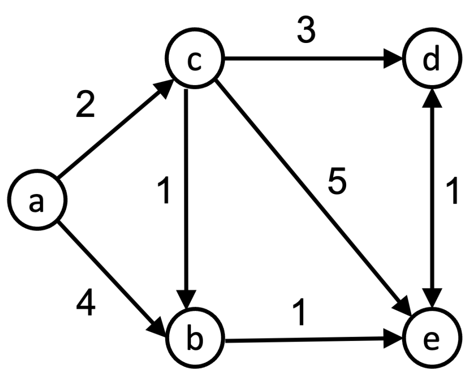
C $O(EV)$ *times*

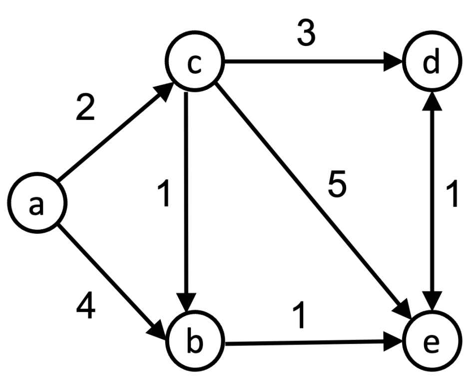
D *Depends on the graph*

3.1 Example









3.2 Correctness

```
DIJKSTRA(G, w, s)
  INITIALIZE-SSSP(G, w, s)
  S = ∅
  Q = V
  while Q ≠ ∅ do
      u = EXTRACTMIN(Q)
      S = S ∪ {u}
      for v ∈ Adj[u] do
          RELAX(u, v, w)
      end for
  end while
```

We will take the following properties as given.

Upper bound property For each $v \in V$, the property $v.d \geq \delta(s, v)$ is maintained throughout the algorithm.

Convergence property Let $p = \{s, \ldots, u, v\}$ be a shortest path involving an edge $(u, v) \in E$. If $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ after relaxing (u, v).

No path property If there is no path from s to v, then $v.d = \delta(s, v) = \infty$ throughout the algorithm.

Theorem 3.1. *Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E, w)$ with positive weights and source $s \in V$, terminates with $u.d = \delta(s, u)$ for all $u \in V$*

Proof. We maintain the following invariant throughout.

Invariant: At the start of the while loop, the node u added to S satisfies $u.d = \delta(s, u)$.

If this invariant is true, the theorem is shown. Why?

Assume this invariant is *not* true and we will show a contradiction.

Define u to be the first node added to S where $u.d \neq \delta(s,u)$.

1. We know $u \neq s$.

2. We know there is a path from s to u.

3. Let p be a shortest path from s to u, which goes from S to $V - S$, let y be the first node in p from $V - S$ in this path, and x be its predecessor.

4. We know $x.d = \delta(s,x)$

5. We then know $y.d = \delta(s,y)$

6. We know that $\delta(s,y) \leq \delta(s,u)$

7. And we know $\delta(s,u) \leq u.d$

8. We know also that $u.d \leq y.d$

9. The last steps show $y.d = \delta(s,y) \leq \delta(s,u) \leq u.d \leq y.d$. Why is this a contradiction?

3.3 Runtime Analysis

```
INITIALIZE-SSSP(G, w, s)
S = ∅; Q = V
while Q ≠ ∅ do
    u = EXTRACTMIN(Q)
    S = S ∪ {u}
    for v ∈ Adj[u] do
        if v.d > u.d + w(u, v) then
            v.d = u.d + w(u, v)
            v.π = u
        end if
    end for
end while
```

Question 4. *Assuming that all nodes are reachable from s, what runtime we can obtain for Dijkstra's algorithm? (Warning: there may be more answer that you can argue is valid! Don't just select an answer, think about a specific reasoning for the runtime analysis you select).*

A $O(V + E)$

B $O(V^2)$

C $O(VE)$

D $O(E \log V)$

E $O(E + V \log V)$

3.4 Class Activity: another example