CS2040 – Data Structures and Algorithms

Lecture 14 – Finding Shortest Way from Here to There, Part I

axgopala@comp.nus.edu.sg

NUS
National University of Singapore
School of Computing

Outline

- Single-Source Shortest Paths (SSSP) Problem
- Motivating example
- Some more definitions
- Discussion of negative weight edges and cycles

- Algorithms to Solve SSSP Problem (CP4 Section 4.4)
- BFS algorithm (cannot be used for the general SSSP problem)
- Bellman-Ford's algorithm
  - Precursor
  - Pseudo code, example animation, and later: Java implementation
  - Theorem, proof, and corollary about Bellman-Ford's algorithm
- https://visualgo.net/sssp

Motivational Example

Definitions (yes, we had to have them!)

- **Path** (p): $<v_0, v_1, v_2, \ldots, v_k>$, where an edge exists between $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$
    - Usually a simple path (no repeated vertex), unless there is a negative cycle

- Shortcut notation: $v_0 \overset{p}{\rightsquigarrow} v_k$

- **Path weight**: $PW(p) = w(v_0, v_1) + \ldots + w(v_{k-1}, v_k)$

- **Important**: Edges are **Directed**

More definitions

- **Shortest Path weight** from vertex **a** to **b**: **δ(a, b)** (pronounced as 'delta')

$$\delta(a,b)=\begin{cases}\min(PW(p)) \\ \infty\end{cases}$$

If there exists such a path

If **b** is unreachable from **a**

Single Source Shortest Paths – Problem

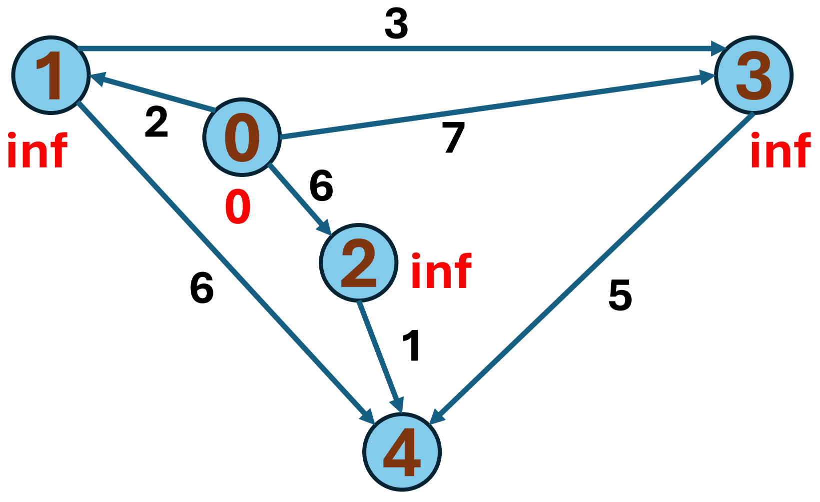
- Given **G(V, E), w(a, b): E->R**, and a source vertex s

- Find $\delta(s, b)$ from vertex **s** to each vertex **b** (in V) together with the corresponding shortest path
  - From **one** source to the **rest**

Some More Definitions

- **Additional Data Structures** to solve the SSSP problem:
  - An Array/Vector **D** of size **V** (**D** stands for 'distance')
    - Initially, **D[v] = 0** if **v** = **s**; otherwise **D[v] =** ∞ (a large number)
    - **D[v]** decreases as we find better paths
    - **D[v]** ≥ **δ(s, v)** throughout the execution of SSSP algorithm
    - **D[v] = δ(s, v)** at the end of SSSP algorithm
  - An Array/Vector **p** of size **V**
    - **p[v]** = the predecessor on best path from source **s** to **v**
    - **p[s]** = –1 (not defined)
    - Recall: The usage of this Array/Vector **p** is already discussed in BFS/DFS Spanning Tree

Example



s = 0

Initially:

D[s] = D[0] = 0

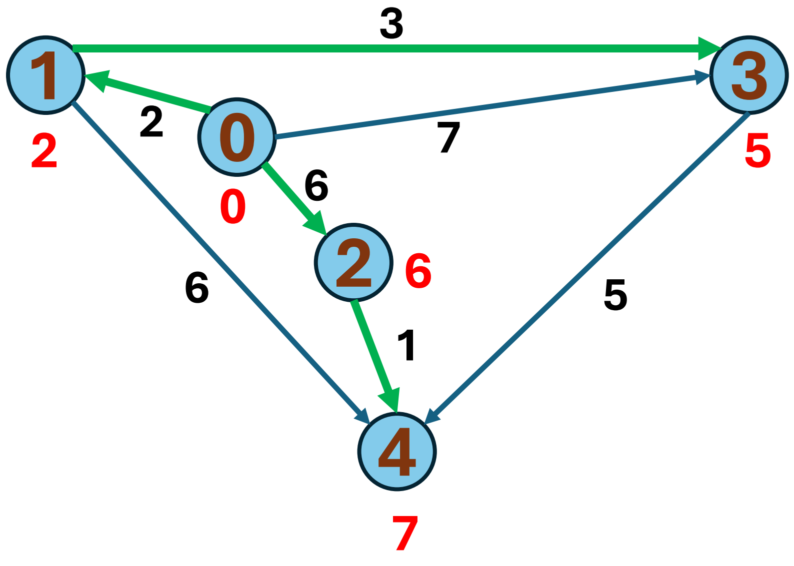
D[v] = inf for the rest

Denoted as values in **red font/vertex**

p[s] = –1 (to say ‘no predecessor’)

p[v] = –1 for the rest

Denoted as green edges (none initially)



s = 0

At the end of algorithm:

D[s] = D[0] = 0 (unchanged)

D[v] = δ(s, v) for the rest

e.g. D[2] = 6, D[4] = 7

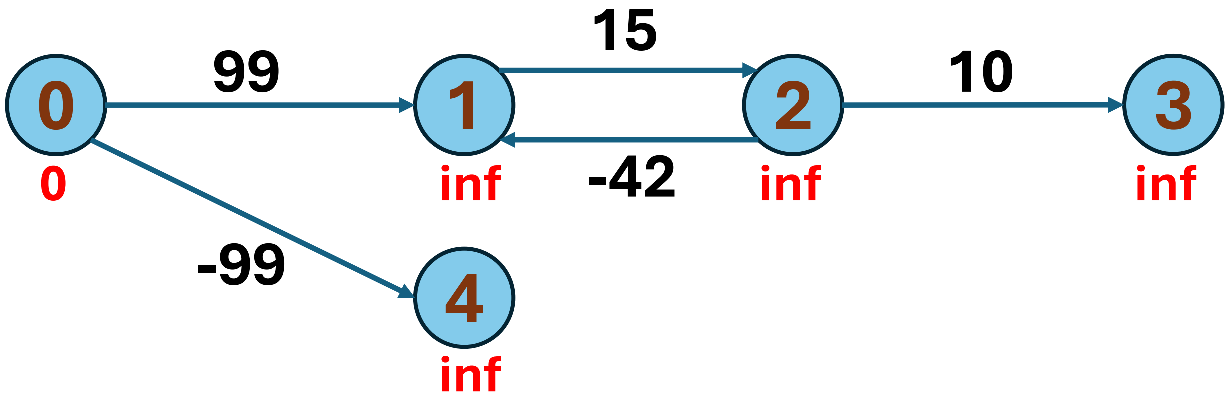
p[s] = –1 (source has no predecessor)

p[v] = the origin of green edges for the rest

e.g. p[2] = 0, p[4] = 2

Negative Edge Weights and Cycles

- Exists in some applications



- Shortest paths from 0 to {1, 2, 3} are **undefined**
  - 1 ➔ 2 ➔ 1 is a negative cycle as it has negative total path (cycle) weight
  - One can take 0 ➔ 1 ➔ 2 ➔ 1 ➔ 2 ... indefinitely to get $-\infty$
- Shortest path from 0 to 4 is ok, with $\delta(0, 4) = -99$

SSSP Algorithms

- SSSP problem is a(nother) **well-known** CS problem

- Three algorithms discussed in this topic

1. O(**V+E**) BFS which fails on *general case* of SSSP problem but useful for a special case
    - Introducing the “initSSSP” and “Relax” operations
2. General SSSP algorithm (pre-cursor to Bellman-Ford)
3. O(**VE**) Bellman-Ford’s SSSP algorithm
    - General idea of SSSP algorithm
    - Trick to ensure termination of the algorithm
    - Bonus: Detecting negative weight cycle

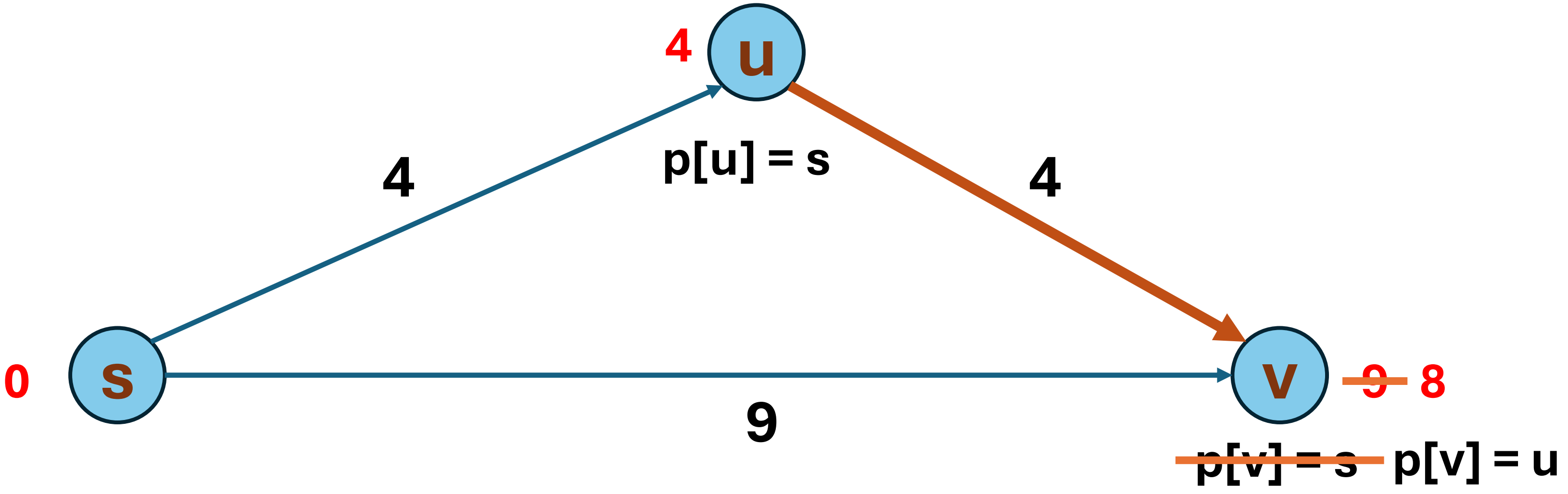
Initialisation Step

- Used in all the SSSP algorithms discussed

```
initSSSP(s)
  for each v ∈ V // initialisation phase
    D[v] ←    1000000000 // use 1B to represent INF
    p[v] ←    -1 // use -1 to represent NULL
  D[s] ← 0 // this is what we know so far
```

'Relaxation' Operation 🏝

```
relax(u, v, w(u,v))
  if D[v] > D[u] + w(u,v) // if SP can be shortened
    D[v] ← D[u] + w(u,v) // relax this edge
    p[v] ←u  // remember/update the predecessor
    // if necessary, update some data structure
```



BFS for SSSP

- When the graph is **unweighted/edges have same weight**, the SSSP can be viewed as a problem of finding the **least number of edges** traversed from source **s** to other vertices
- The O(**V**+**E**) Breadth First Search (BFS) traversal algorithm precisely measures this (BFS Spanning Tree = Shortest Paths Spanning Tree)

BFS Modifications

- Three simple modifications:

1. Replace **visited** with **D**
2. At the start of BFS, set **D[v] = INF** (say, 1 Billion) for all **v** in **G**, except **D[s] = 0**
3. Change this part (in the BFS loop) from:

```
if visited[v] = 0 // if v is not visited before
   visited[v] = 1; // set v as reachable from u
```

into:

```
if D[v] = INF // if v is not visited before
   D[v] = D[u]+1; // v is 1 step away from u ☺
```

Modified BFS Pseudo Code

```
for all v in V
  D[v] ← INF
  p[v] ← -1
Q ← {s} // start from s
D[s] ← 0
```

Initialization phase

```
while Q is not empty
  u ← Q.dequeue()
  for all v adjacent to u // order of neighbour
    if D[v] = INF //influences BFS
      D[v] ← D[u]+1 // visitation sequence
      p[v] ← u
      Q.enqueue(v)
```

Main loop

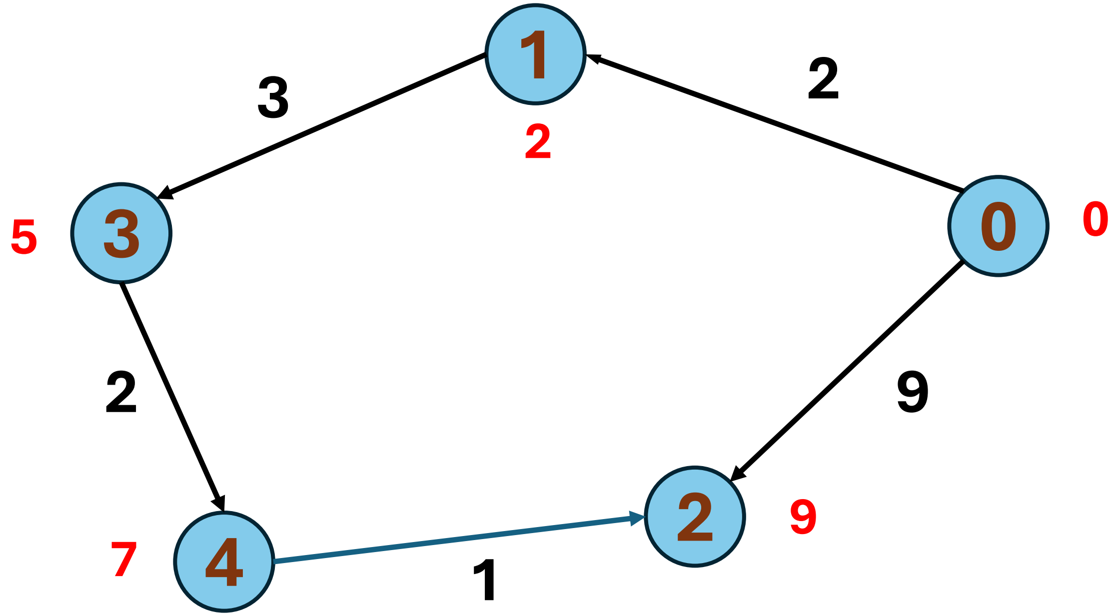
```
// we can then use information stored in D/p
```

But .....

- BFS does not always work on generic graphs ☹

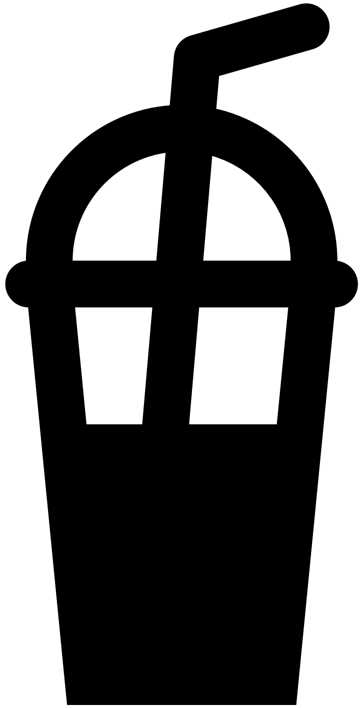
For Visualgo

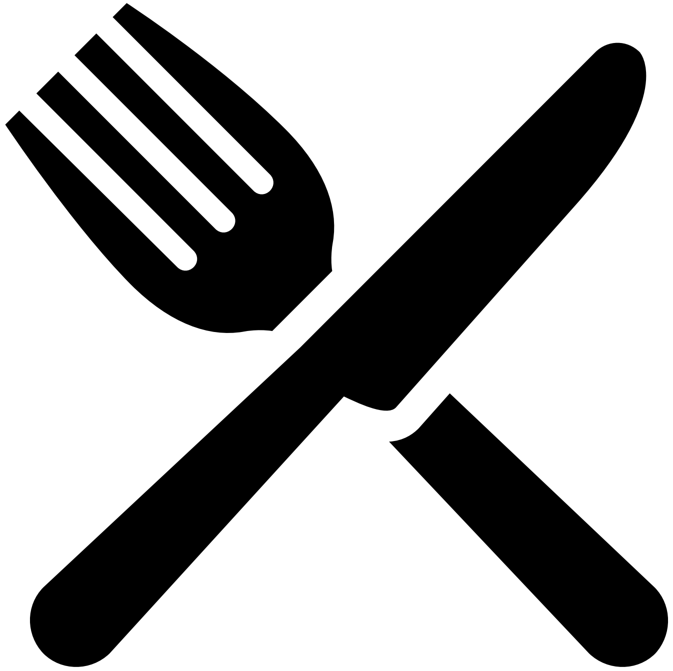
5 5
0 1 2
0 2 9
1 3 3
3 4 2
4 2 1



Generally:
If you know **for sure** that your graph is unweighted
- Use BFS O(**V + E**) to solve the SSSP problem – more efficient

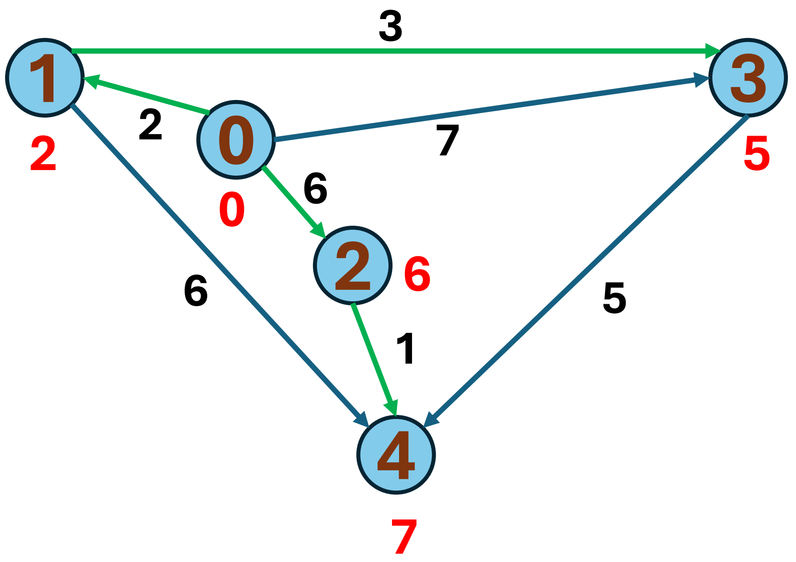
Take a Break





SSSP – Terminating Condition

- How do we determine when an algorithm has solved the SSSP?
    - When for all edges (u,v), D[v] <= D[u] + w(u,v) (i.e., no edge can be relaxed further)

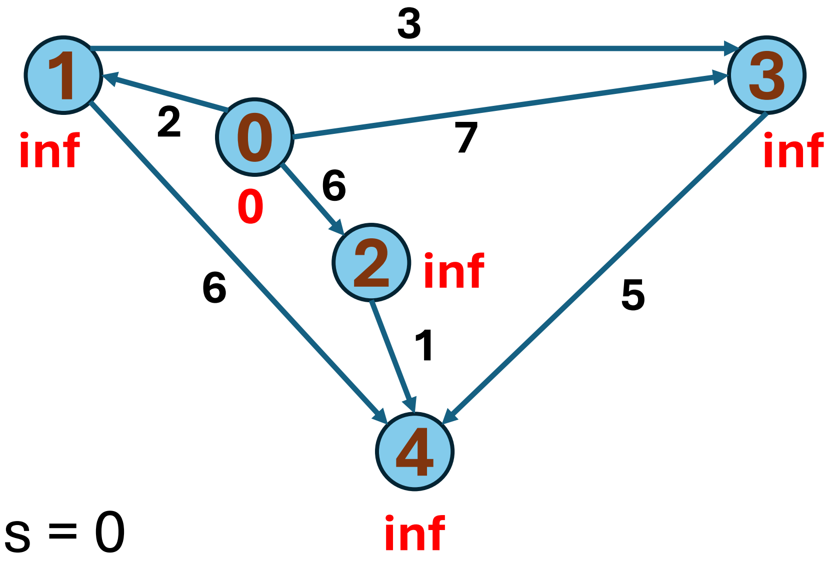


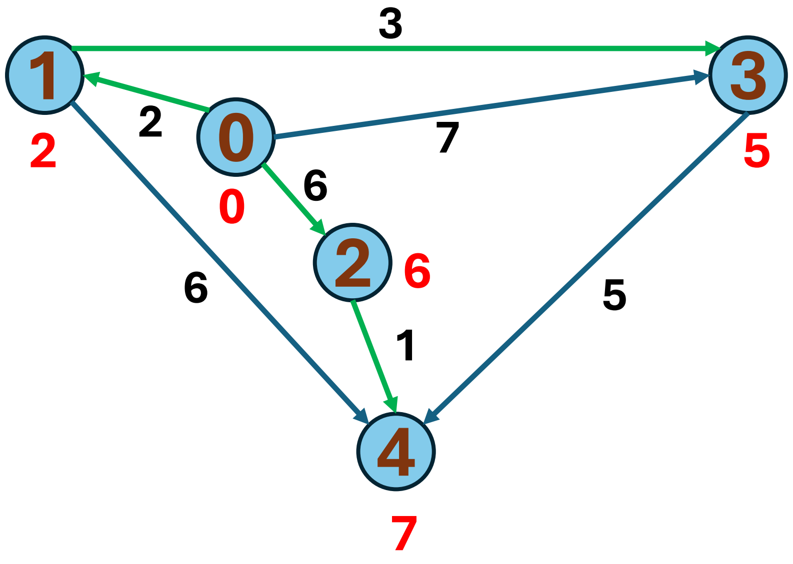
Simple Algorithm

```
initSSSP(s) // as defined earlier

repeat // main loop
  select edge(u, v) ∈ E in arbitrary manner
  relax(u, v, w(u, v)) // as defined in previous slide
until all edges have D[v] <= D[u] + w(u, v)
```

Recall: Example





s = 0

Initially:

D[s] = D[0] = 0

D[v] = inf for the rest

Denoted as values in **red font/vertex**

p[s] = –1 (to say ‘no predecessor’)

p[v] = –1 for the rest

Denoted as green edges (none initially)

s = 0

At the end of algorithm:

D[s] = D[0] = 0 (unchanged)

D[v] = δ(s, v) for the rest

e.g. D[2] = 6, D[4] = 7

p[s] = –1 (source has no predecessor)

p[v] = the origin of green edges for the rest

e.g. p[2] = 0, p[4] = 2

Simple SSSP Algorithm – Analysis

- If given a graph without negative weight cycle, when will this simple SSSP algorithm terminate?
  - Depends on your luck … ☹
  - Can be very slow …
- The main problem is in this line:

```
select edge(u, v) ∈ E in arbitrary manner
```

- Next, we will study **Bellman-Ford's** algorithm that do these relaxations in a *better order*!

Bellman-Ford’s Algorithm

```
initSSSP(s)

// Simple Bellman-Ford's algorithm runs in O(VE)
for i = 1 to |V|-1 // O(V) here
  for each edge(u, v) ∈  E // O(E) here
    relax(u, v, w(u,v)) // O(1) here

// At the end of Bellman-Ford's algorithm,
// D[v] = δ(s, v) if no negative weight cycle exist

// Q: Why “relaxing all edges V–1 times” works?
```

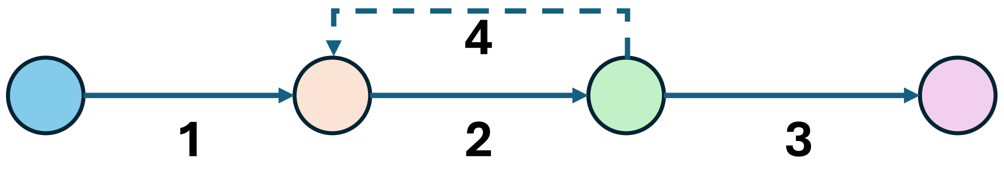
Theorem

- If **G = (V, E)** contains no negative weight cycle, then the shortest path **p** from **s** to **v** is a **simple path**

- Proof by contradiction (our favourite kind of proof 😎)

Proof (1/2) – By Contradiction!

1. Suppose the shortest path **p** is not a simple path
2. Then **p** contains one (or more) cycle(s)
3. Suppose there is a cycle **c** in **p** with positive weight
4. If we remove **c** from **p**, then we have a shorter ‘shortest path’ than **p**
5. This contradicts the fact that **p** is a shortest path

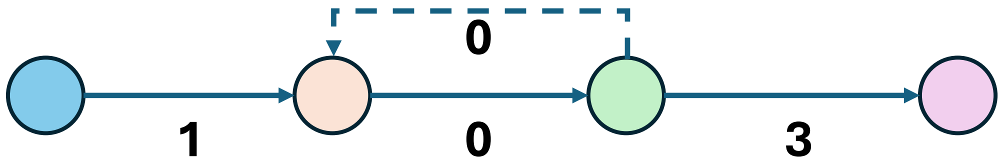


Proof (2/2) – By Contradiction!

6. Even if **c** is a cycle with zero total weight (it is possible!), we can still remove **c** from **p** without increasing the SP weight of **p**

7. So, **p** is a simple path (from point 5) or can always be made into a simple path (from point 6)

- In other words, path **p** has at most **|V|-1** edges from the source **s** to the "furthest possible" vertex **v** in **G** (in terms of number of edges in the shortest path)



Another Theorem!

- Theorem 2 : If **G = (V, E)** contains no negative weight cycle, then after Bellman-Ford's terminates **D[v] = δ(s, v), $\forall$v $\in$ V**

- Proof by Induction! 😯

Proof (1/2) – By Induction

1. Define $\mathbf{v_i}$ to be any vertex that has shortest path **p** requiring *i* number of edges from s

2. Initially $\mathbf{D[v_0] = \delta(s, v_0) = 0}$, as $\mathbf{v_0}$ is just **s**

3. After **1** pass through **E**, we have $\mathbf{D[v_1] = \delta(s, v_1)}$

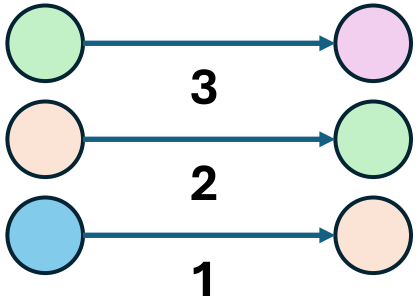
4. After **2** passes through **E**, we have $\mathbf{D[v_2] = \delta(s, v_2)}$, …

5. After **k** passes through **E**, we have $\mathbf{D[v_k] = \delta(s, v_k)}$

Proof (2/2)– By Induction!

6. When there is no negative weight cycle, the shortest path **p** will be simple (see the previous proof)

7. Thus, after **|V|-1** iterations, the "furthest" vertex $\mathbf{v_{|V|-1}}$ from **s** has $\mathbf{D[v_{|V|-1}] = \delta(s, v_{|V|-1})}$
   - Even if edges in **E** are processed in the *worst possible order*



A ‘Side Effect’

- Corollary: If a value **D[v]** *fails to converge* after **|V|-1** passes, then there exists a negative-weight cycle reachable from **s**

- Additional check after running Bellman-Ford’s

```
for each edge(u, v) ∈ E
    if (D[u] != INF && D[v] > D[u]+w(u, v))
        report negative weight cycle exists in G
```

Summary

- SSSP problem
- BFS algorithm for unweighted SSSP problem
  - But it fails on general case
- Bellman-Ford’s algorithm
  - Solves SSSP problem for general weighted graph in O(**VE**)
  - Can also be used to detect the presence of negative weight cycle

Next

- Special cases of the classical SSSP problem

Continuous Feedback

https://forms.office.com/r/KsNwmTUD0q