香港中文大學（深圳）
The Chinese University of Hong Kong

CSC3100 Data Structures
Lecture 23: Graph shortest path

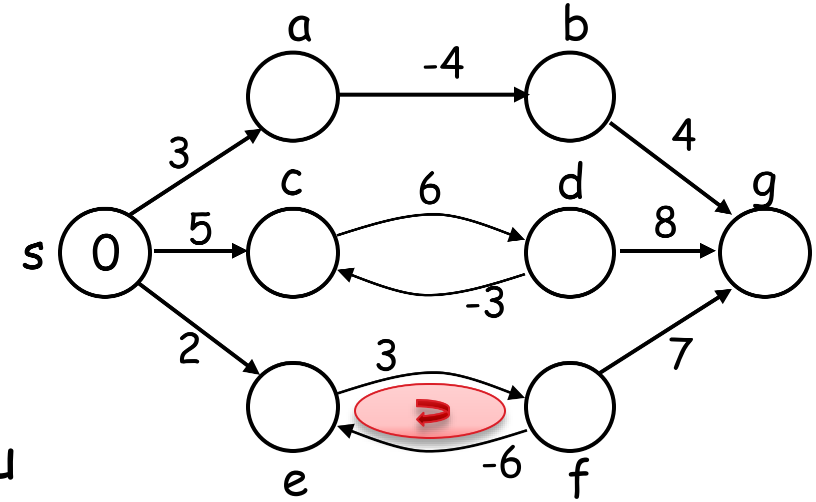
Li Jiang
School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen

Outline

- Shortest path problem
  - Graphs with non-negative weights
    - Single-Source Shortest Path: Dijkstra's algorithm
  - All-Pair Shortest Path: Floyd's algorithm
  - Graphs with negative weights
    - Bellman-Ford algorithm
- Topological sort

Negative-weight edges

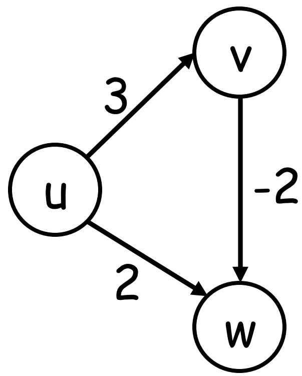
- Negative-weight edges may form negative-weight cycles
  - Negative weights are edge values that represent a cost or a loss, such as a penalty, a fee, or a discount
  - E.g., in a currency exchange graph, a negative weight could indicate a favorable exchange rate that gives you more money



- If such cycles are reachable from the source vertex s, then $\delta(s, v)$ is not properly defined!
  - Keep going around the cycle, and get $w(s, v) = -\infty$ for all v on the cycle

Is Dijkstra's algorithm still applicable for graphs with negative weights?

- Dijkstra's algorithm cannot handle a graph that has negative weights but no negative cycles



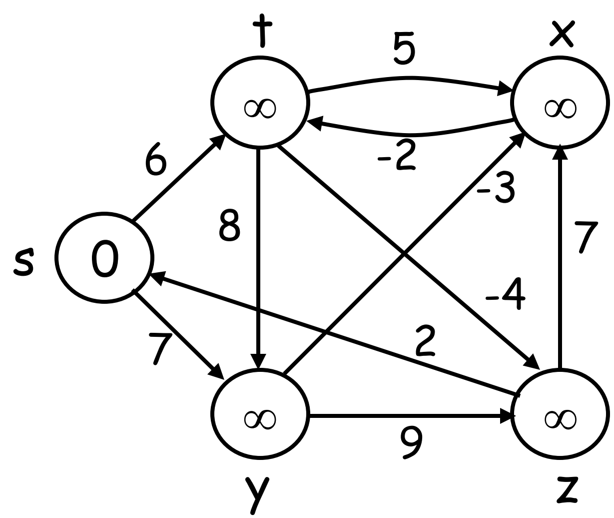
- How to handle a graph that has negative weights but no negative cycles?

Bellman-Ford algorithm

- Single-source shortest path problem
  - Computes $\delta(s, v)$ and $p[v]$ for all $v \in V$
- Allows negative edge weights - can detect negative cycles
  - Returns TRUE if no negative-weight cycles are reachable from the source s
  - Returns FALSE otherwise $\Rightarrow$ no solution exists

Bellman-Ford algorithm (cont'd)

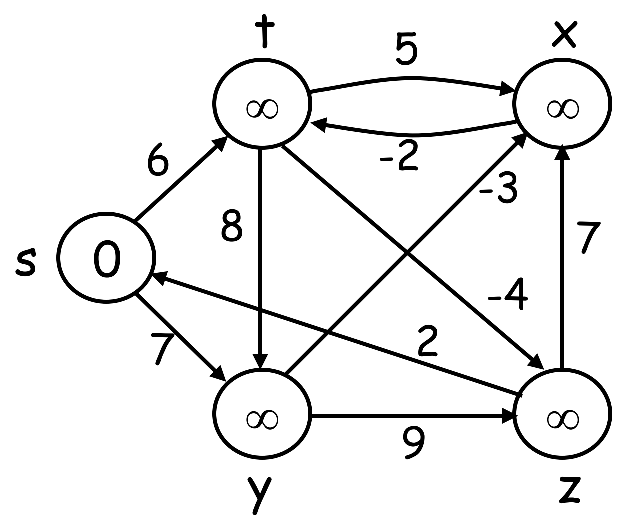
- Idea:
  - Each edge is relaxed |V| – 1 times by making |V| - 1 passes over the whole edge set
  - Any path will contain at most |V| - 1 edges

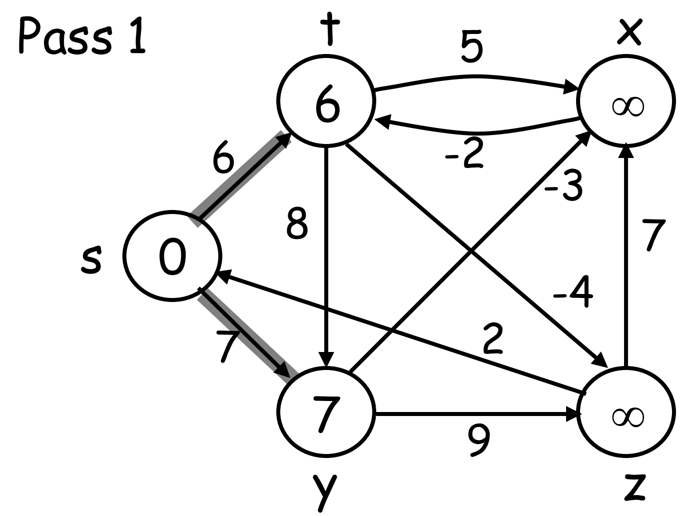


For each edge (u, v), do relaxation:
If d[v] > d[u] + w(u, v)
$\Rightarrow$ d[v] = d[u] + w(u, v)

Edge order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Bellman-Ford(V, E, w, s)

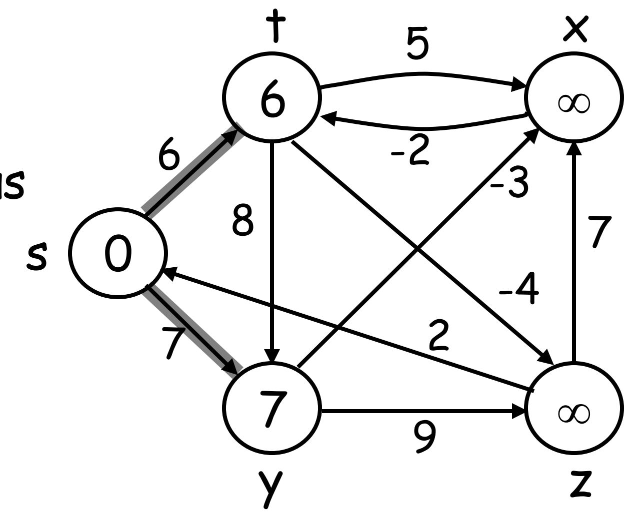




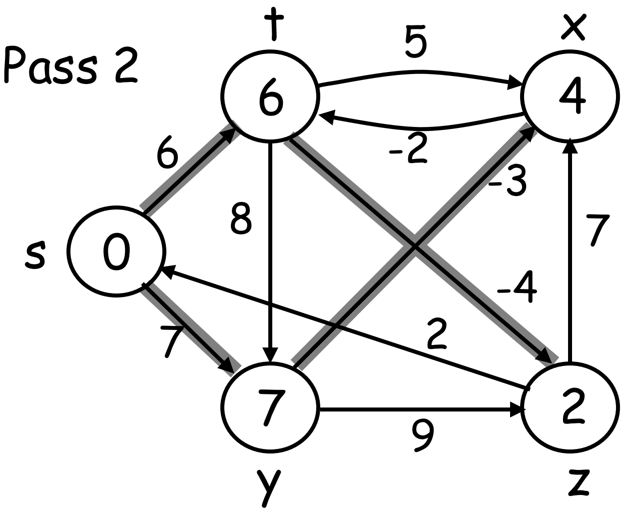
Edge order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Example

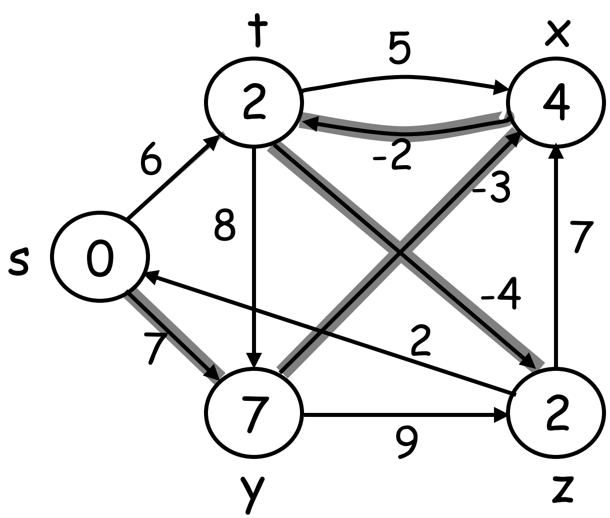
Pass 1 (from previous slide)



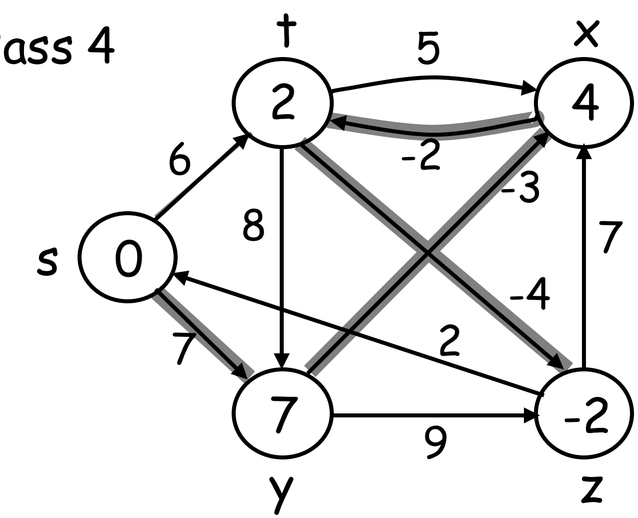
Pass 2



Pass 3



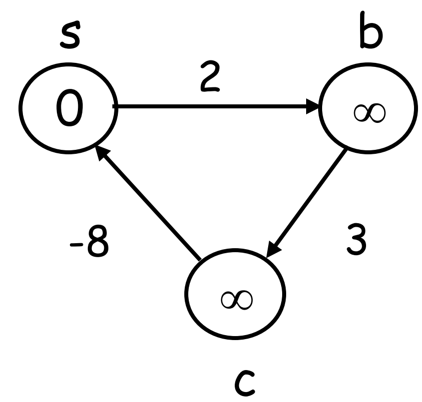
Pass 4



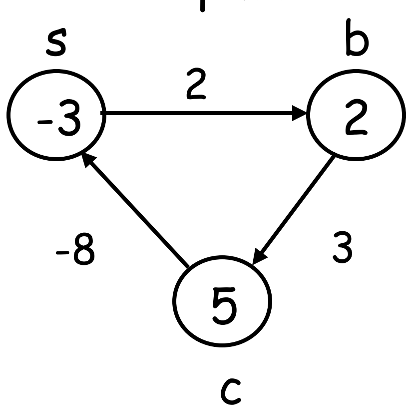
Edge order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Detecting negative cycles (perform extra test after |V| - 1 iterations)

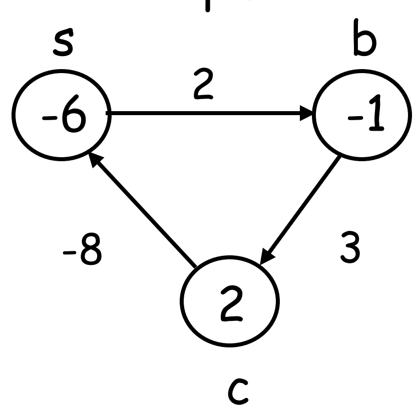
- **for** each edge $(u, v) \in E$
- **do if** $d[v] > d[u] + w(u, v)$
- **then return** FALSE
- **return** TRUE



1st pass



2nd pass



(s, b), (b, c), (c, s)

Look at edge (s, b):

$d[b] = -1$
$d[s] + w(s, b) = -4$

$\Rightarrow d[b] > d[s] + w(s, b)$

Bellman-Ford(V, E, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, s)            ← Θ(|V|)
for i ← 1 to |V| - 1                      ← O(|V|)
    do for each edge (u, v) ∈ E           ← O(|E|)      O(|V||E|)
            do RELAX(u, v, w)
for each edge (u, v) ∈ E                  ← O(|E|)
    do if d[v] > d[u] + w(u, v)
            then return FALSE
 return TRUE
```

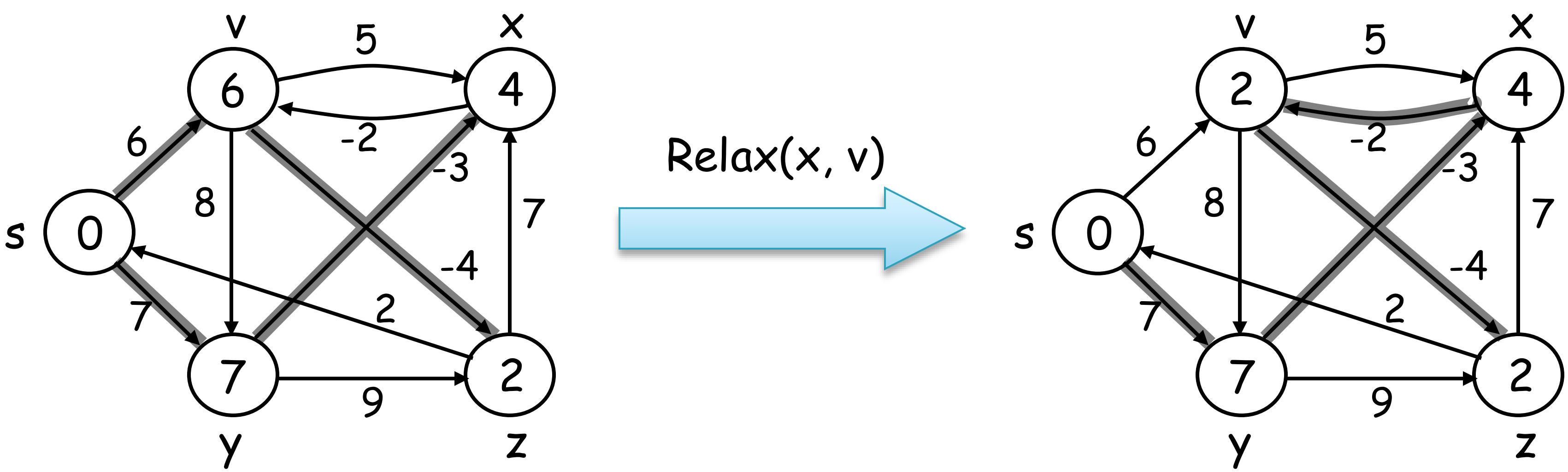
Running time: $O(|V||E|)$

Key points of Bellman-Ford

- If there is no negative cycle, then after |V| - 1 iterations, d values will not be updated or can't be lowered any more, and d values store the measure of the shortest path
  - Why? How to prove its correctness?

Shortest path properties

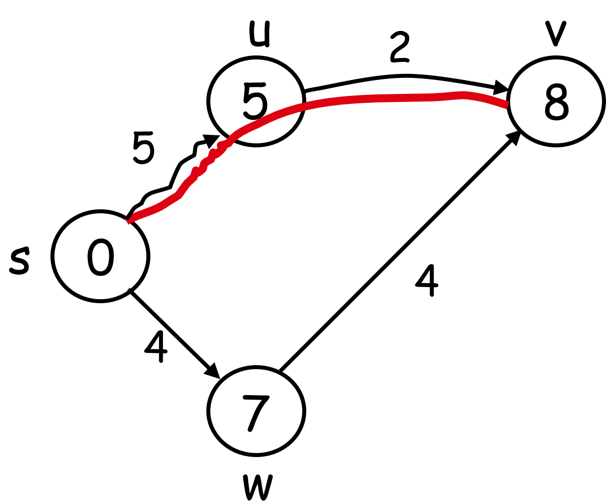
- **Upper-bound property**
  - We always have d[v] ≥ δ(s, v) for all v
  - The estimate never goes up – relaxation only lowers the estimate



Shortest path properties

- **Convergence property**

  If $s \rightsquigarrow u \rightarrow v$ is a shortest path, and if $d[u] = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $d[v] = \delta(s, v)$ at all times after relaxing (u, v)



- If $d[v] > \delta(s, v) \Rightarrow$ after relaxation:

  $d[v] = d[u] + w(u, v)$

  $d[v] = 5 + 2 = 7$

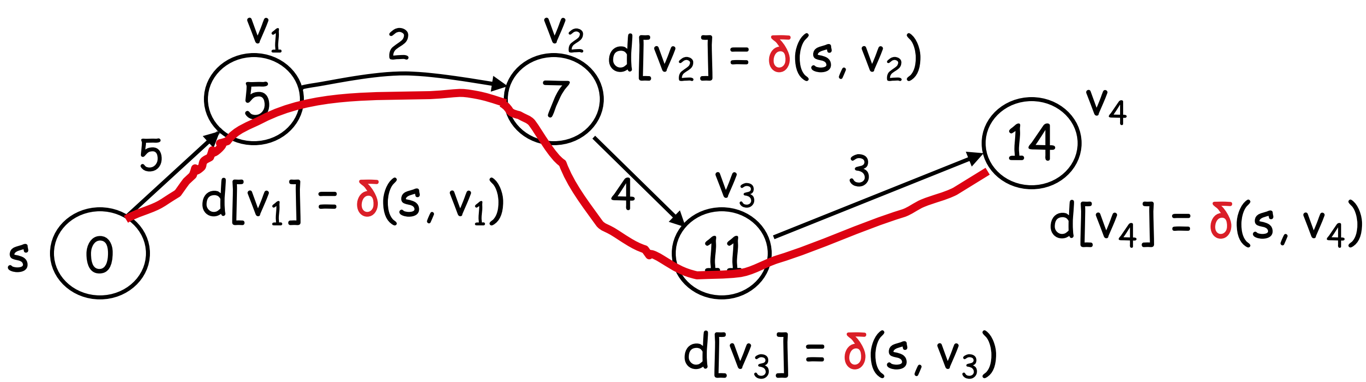
- Otherwise, the value remains unchanged, because it must have been the shortest path value

Shortest path properties

- **Path relaxation property**

  Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k

  If we relax, in order, $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, even intermixed with other relaxations, then $d[v_k] = \delta(s, v_k)$



Correctness of Bellman-Ford algorithm

- **Theorem:** Show that $d[v] = \delta(s, v)$, for every v, after $|V| - 1$ passes

- Case 1: G does not contain negative cycles which are reachable from s

  - Assume that the shortest path from s to v is
    $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $s = v_0$ and $v = v_k$, $k \le |V|-1$

  - Use mathematical induction on the number of passes i to show that:

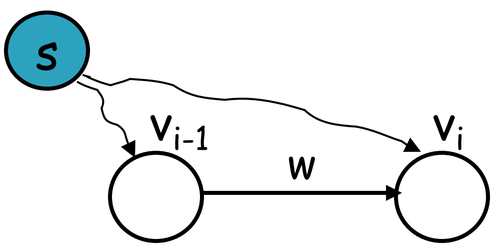
$$d[v_i] = \delta(s, v_i)\ ,\ i=0,1,\ldots,k$$

Correctness of Bellman-Ford algorithm

Base case: $i=0$, $d[v_0] = \delta(s, v_0) = \delta(s, s) = 0$

Inductive hypothesis: $d[v_{i-1}] = \delta(s, v_{i-1})$

Inductive step: $d[v_i] = \delta(s, v_i)$



$d[v_{i-1}] = \delta(s, v_{i-1})$

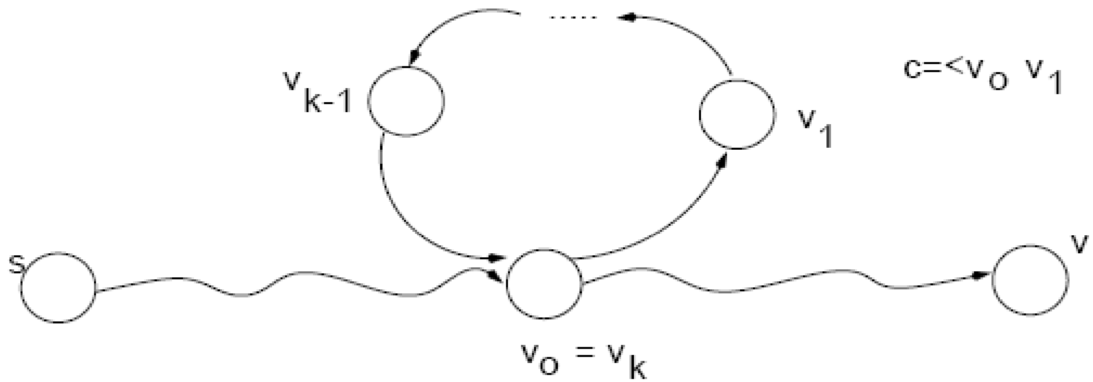
After relaxing (v_{i-1}, v_i) (convergence property) :
$d[v_i] \leq d[v_{i-1}] + w = \delta(s, v_{i-1}) + w = \delta(s, v_i)$

From the upper bound property: $d[v_i] \geq \delta(s, v_i)$

Therefore, $d[v_i] = \delta(s, v_i)$

Correctness of Bellman-Ford algorithm

- Case 2: G contains a negative cycle which is reachable from s



c=<v_o v_1 … v_k> is a negative cycle

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

Proof by Contradiction: suppose the algorithm returns a solution

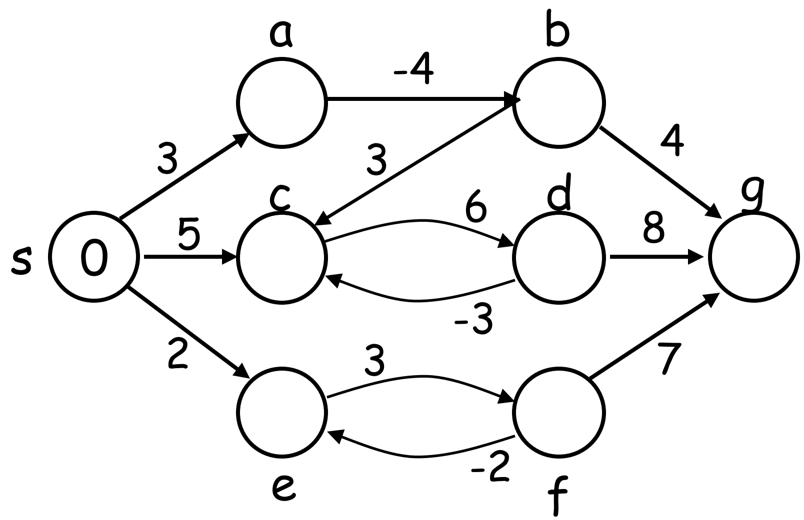
After relaxing (v_{i-1}, v_i): $dist[v_i] \le dist[v_{i-1}] + w(v_{i-1}, v_i)$

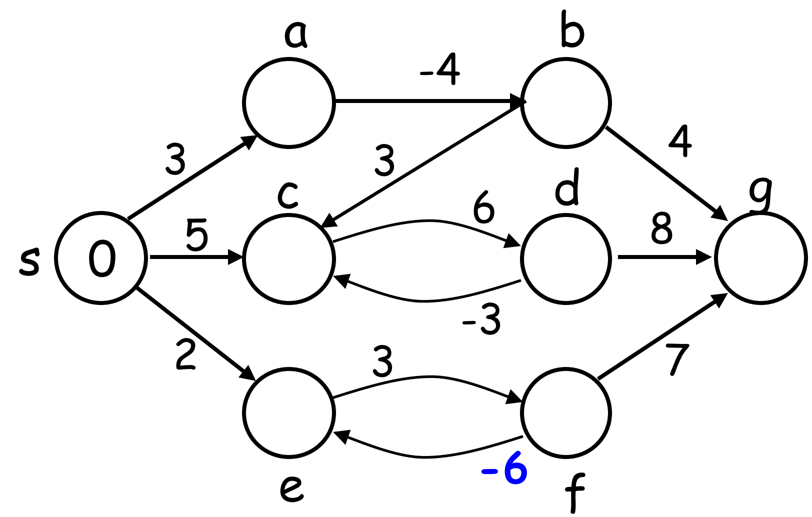
$$\Rightarrow \sum_{i=1}^{k} dist[v_i] \le \sum_{i=1}^{k} dist[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$\Rightarrow \sum_{i=1}^{k} w(v_{i-1}, v_i) \ge 0 \ (\sum_{i=1}^{k} dist[v_i] = \sum_{i=1}^{k} dist[v_{i-1}])$$

Contradiction!

Exercise 1

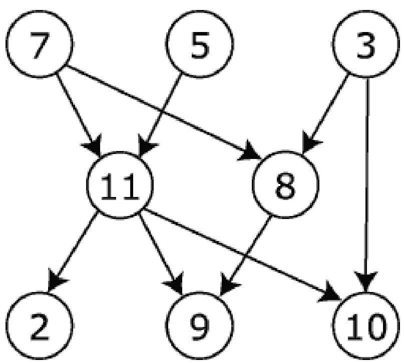


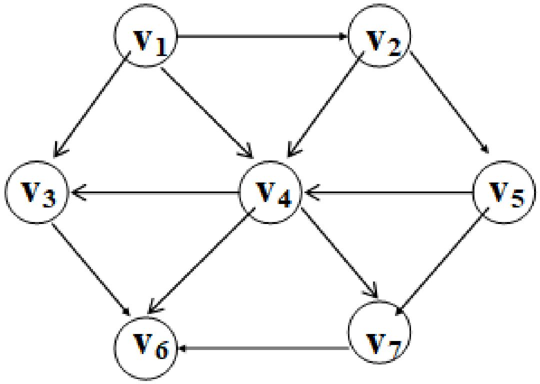


Topological sort

Topological sort

- An ordering of all vertices in a directed acyclic graph, such that if there is a path from v_i to v_j, then v_j appears after v_i in the ordering
- If there is no path between v_i and v_j, then any order between them is fine
- Applications: job scheduling, logistics planning, course selection, etc.





Topological sort

- Topological ordering is not possible if there is a cycle in the graph
- A DAG has at least one topological ordering
- A simple algorithm
  - Compute the indegree of all vertices from the adjacency information of the graph
  - Find any vertex with no incoming edges
  - Print this vertex, and remove it, and its edges
  - Apply this strategy to the rest of the graph

Topological sort

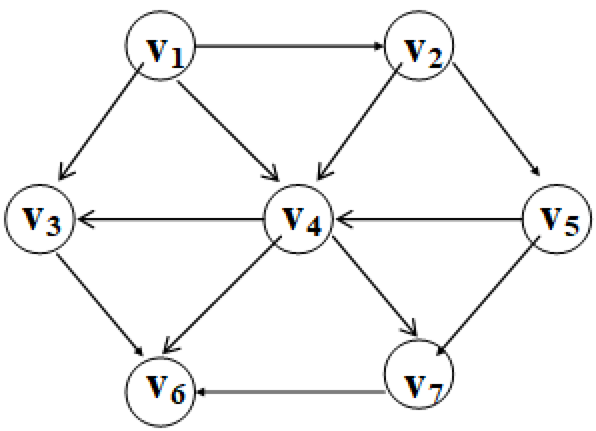
/ Assume that the graph is already read into an adjacency list and that the indegrees are computed and placed in an array */*

```
void topsort () {
        for (int counter = 0; counter < numVertex; counter++) {
                Vertex v = FindNewVertexOfInDegreeZero (); //check all vertices
                if (v == null) {
                        Error("Cycle Found"); return;
                }
                v.topNum = counter;
                for each Vertex w adjacent to v
                        w.indegree--;
        }
}
```

Running time is $O(|V|^2)$

Topological sort

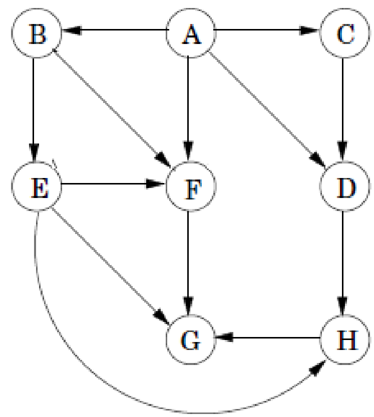
- An improved algorithm: $O(|E|+|V|)$ time
  - Keep all the unassigned vertices of indegree 0 in a queue
  - While queue is not empty,
    - Remove a vertex in the queue
    - Decrease the indegrees of all adjacent vertices
    - If the indegree of an adjacent vertex is 0, enqueue the vertex



	Indegree Before Dequeue #						
Vertex	1	2	3	4	5	6	7
v_1	0	0	0	0	0	0	0
v_2	1	0	0	0	0	0	0
v_3	2	1	1	1	0	0	0
v_4	3	2	1	0	0	0	0
v_5	1	1	0	0	0	0	0
v_6	3	3	3	3	2	1	0
v_7	2	2	2	1	0	0	0
Enqueue	v_1	v_2	v_5	v_4	v_3, v_7		v_6
Dequeue	v_1	v_2	v_5	v_4	v_3	v_7	v_6

Exercise

- Compute the topological sort of this graph



One topological sort result: A, B, C, D, E, F, H, G

Recommended reading

- Reading this week
  - Textbook Chapters 24-25
- Next lecture
  - DAG checking and SCC computation, Chapter 22