CS2010 – Data Structures and Algorithms II

Lecture 07 – Mid-Semester Review +

Finding Shortest Way from Here to There, Part I

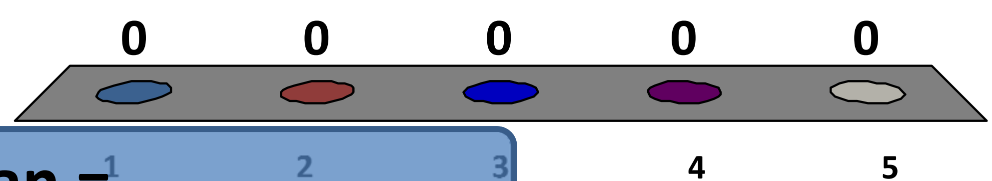
stevenhalim@gmail.com

National University of Singapore

School of Computing

Quiz 1: your opinion

1. Too hard
2. Hard
3. Just ok
4. Easy
5. Too easy

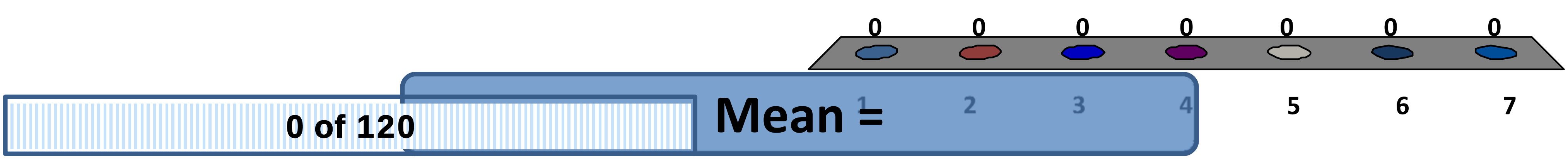


0 of 120

Mean =

Quiz 1: your prediction about your own score

1. [90..103]
2. [80..90)
3. [70..80)
4. [60..70)
5. [50..60)
6. [40..50)
7. [0..40)



Quiz 1 Solution Discussion

- Refer to Quiz1-sol.pdf

This is what we have covered so far…

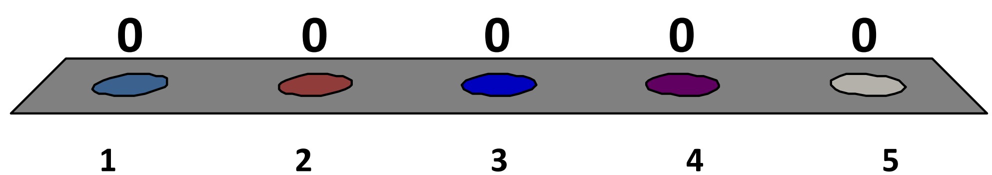
MID-SEMESTER REVIEW

Review of the First One-Third

- Trees
  - Binary Search Trees: Basic Concepts
  - Balanced BSTs: The importance of being Balanced, AVL
  - Priority Queues and Binary Heaps: Basic Concepts
- Notable Examples:
  - Census Problem ("Dictionary Problem")
  - PS1 Baby Names (also Dictionary Problem, involving **range**)
  - Air Traffic Controller
  - PS2 Scheduling Deliveries (a Priority Queue problem)

What is the right child of the root of BST if the following sequence of items are inserted to an initially empty BST: {8, 7, 2, 10, 4}

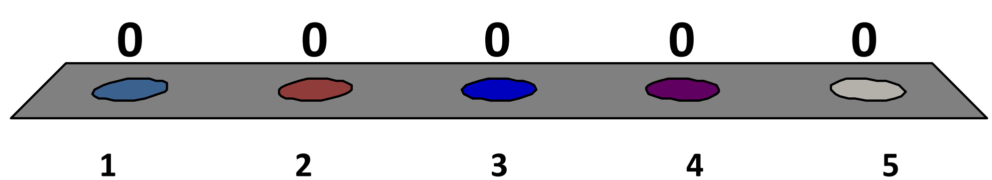
1. 2
2. 7
3. 8
4. 10
5. 4



0 of 120

Which Statement About Heap is False?

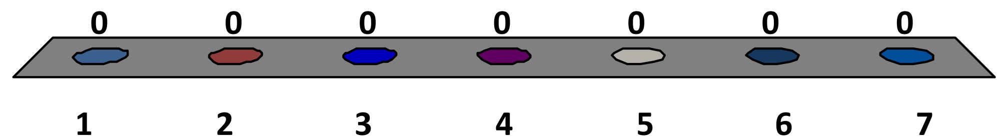
1. Heap must be a complete binary tree
2. Heap can be implemented with array
3. Heap can be used for sorting
4. Building a heap from an unsorted array can only be done in O(n log n)
5. Heap can be used in Prim's algorithm



0 of 120

CS1020 + a bit of CS2010 Review
Which sorting algorithm is the best?

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Merge Sort
5. Quick Sort
6. NEW: balanced BST Sort
7. NEW: Heap Sort



0 of 120

Review: Tree (+ Additional Stuffs)

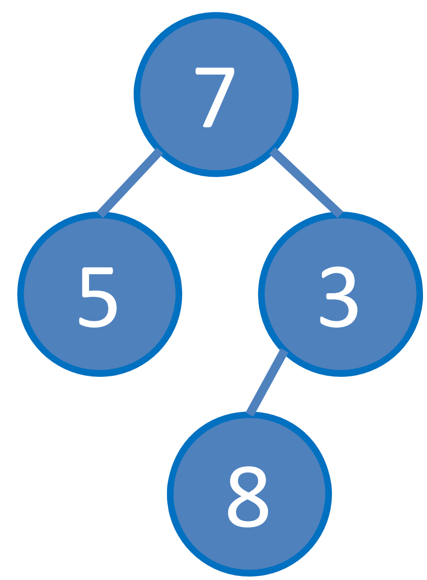
- The Concepts of Tree
  - Connected; E = V - 1; Unique path between two vertices
  - Cannot have cycle
  - Types: Binary, n-ary, Complete, Full
  - Terminologies: Root, Internal Nodes, Leaves, Sub-trees
  - Tree Traversal Algorithms: Pre-order, In-order, Post-order, Level-order (essentially BFS)

Size of a Binary Tree

- This is a naturally recursive algorithm

```
size(T)
  if (T = NULL) return 0;
  else return 1 + size(T->left) + size(T->right);
```

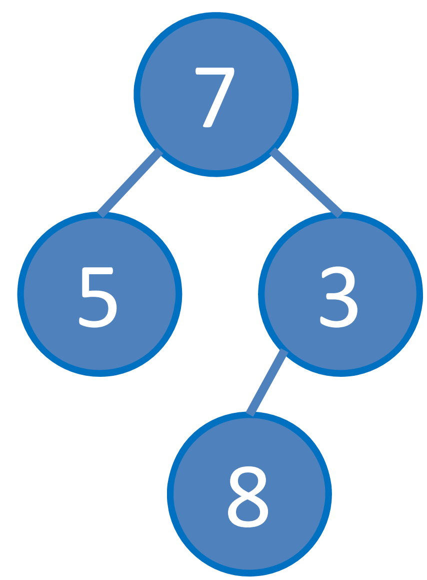
- Time Complexity: O(V)



Height of a Binary Tree

```
height(T)
  if (T = NULL) // empty tree
    return 0;
  else if (T->left = NULL and T->right = NULL) // leaf
    return 1;
  else // internal node
    return 1 + max(height(T->left), height(T->right));
```

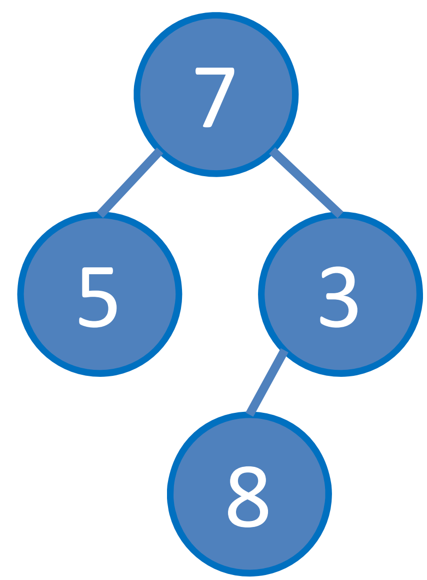
- Time Complexity: also O(V)



Max (or Min) Item of a Binary Tree

```
maxV(T) // minor adjustment for finding min
  if (T = NULL) return -INF;
  else return max(T->value,
    max(maxV(T->left), maxV(T->right));
```

- Time Complexity: again O(V)



Can You Generalize It?

- What if the tree is n-ary tree, not just binary tree?

Review: Second One-Third

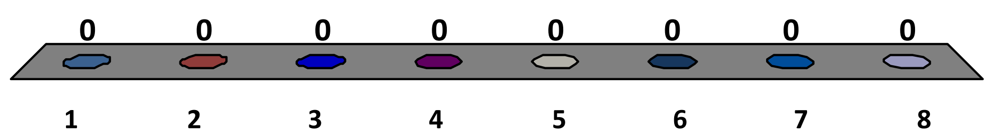
- Graphs
  - Graph Data Structures: AdjMat/List/EdgeList/Implicit Graph
  - Graph Traversal: DFS/BFS and its various applications
  - Minimum Spanning Trees: Prim's/Kruskal's
  - Single-Source Shortest Paths: Bellman Ford's/Dijkstra's (this and the next e-lecture)
- Notable Examples:
  - Hospital Tour (Connected Components++)
  - Pancake Sorting (BFS+++)
  - Out for a Walk (ongoing)
  - The Onset of Labor (soon)

PS Bonus (Pancake Sorting) Discussion

- Refer to psbonus-sol.pdf
- (Not written): Mention Steven's personal remarks for those who are struggling with CS2010 PSes so far…

Select graph terminologies that you know **now**… (can select up to 8/clicker)

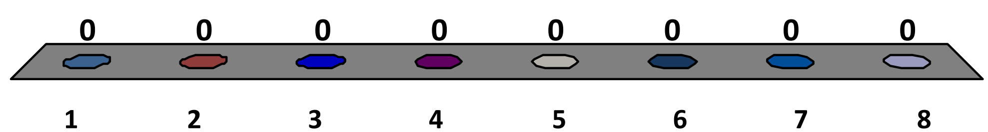
1. AdjMatrix/List/EdgeList
2. DFS/BFS
3. Topological Sort
4. MST/Prim's
5. MST/Kruskal's
6. SSSP/Bellman Ford's
7. SSSP/Dijkstra's
8. APSP/Floyd Warshall's



0 of 120

Select DS/algorithms that you have already **implement now**… (can select up to 8/clicker)

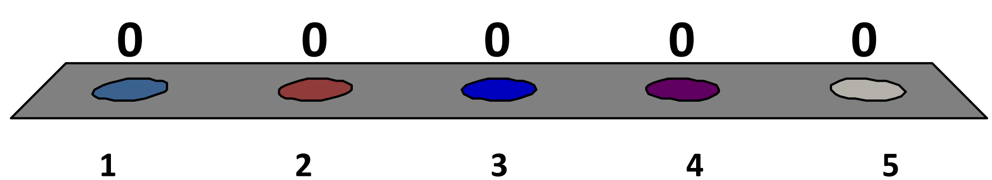
1. AdjMatrix/List/EdgeList
2. DFS/BFS
3. Topological Sort
4. MST/Prim’s
5. MST/Kruskal’s
6. SSSP/Bellman Ford’s
7. SSSP/Dijkstra’s
8. APSP/Floyd Warshall’s



0 of 120

DFS and BFS always run in $\Theta(V^2)$ on

1. Tree
2. Directed Acyclic Graph
3. Bipartite Graph
4. Complete Graph
5. Impossible, it is $O(V+E)$

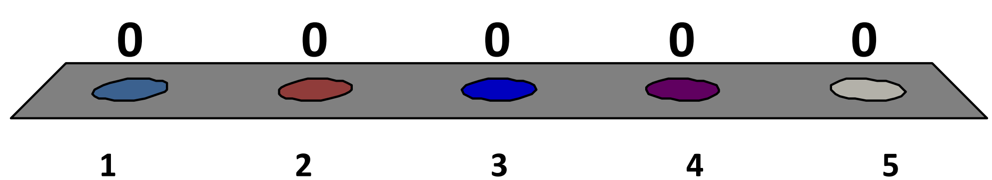


0 of 120

DFS and BFS can run in $O(V^2)$ on

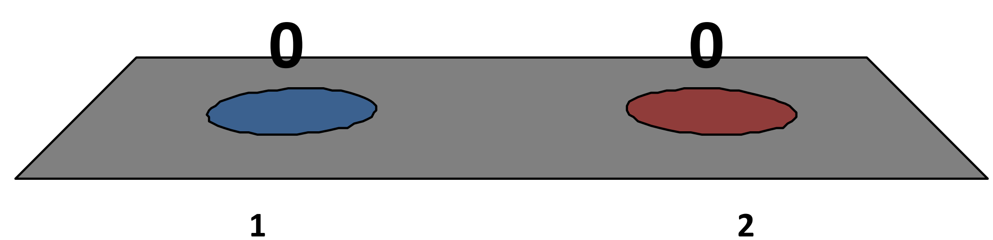
1. Tree
2. Directed Acyclic Graph
3. Bipartite Graph
4. Complete Graph
5. Impossible, it is O(V+E)

0 of 120



Hm… Can I use **standard BFS** (i.e. add one line like in modified DFS) to find toposort?

1. Yes, why not?
2. No, I think BFS will have a problem because ________



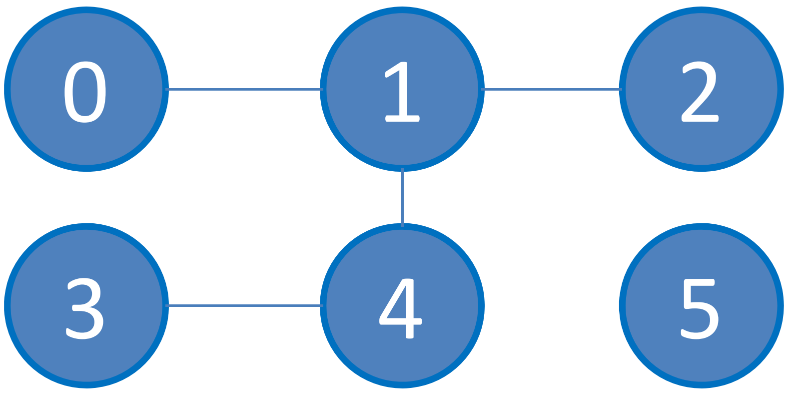
0 of 120

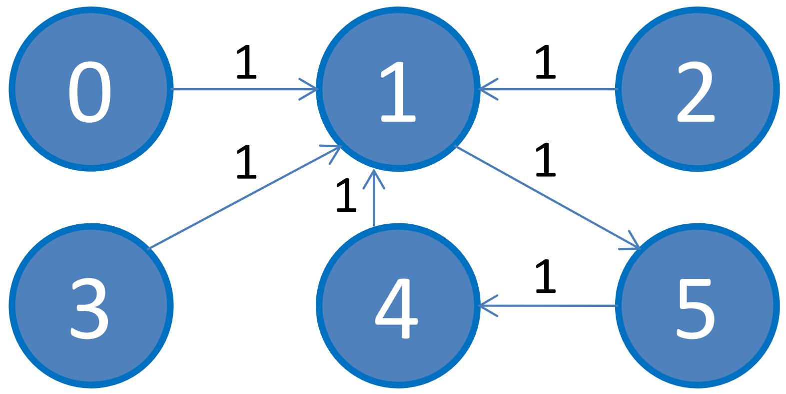
Graph Modeling
Find Real-Life Graphs Near You

- Find several real-life graphs around you (in this room) or around your life!
  - State what are the vertices, the edges
  - Find simple and meaningful graph problems (if any)
- Simple example:
  - Vertices: NUS modules
  - Edges: Module pre-requisites (it is a DAG!!)
  - Graph problem: I have taken a set of modules, can I take a certain future module given this pre-requisites DAG?

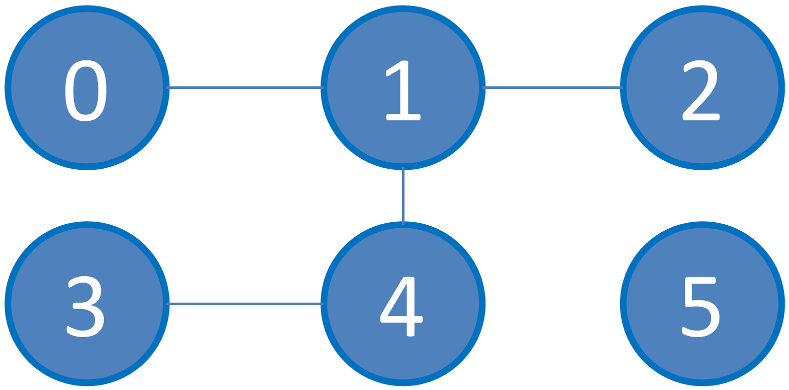
Review – Graph Properties

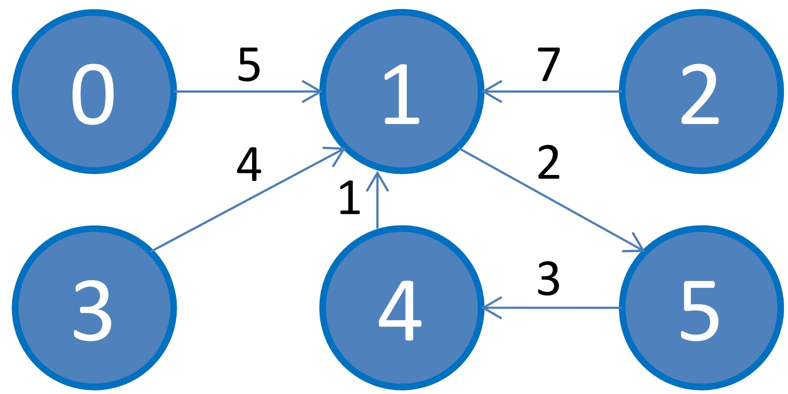
- Elaborate the properties of these two graphs
  - How many V? E? Components? Is it connected? Is it weighted? Is it directed? Is it acyclic? Is it a tree? Is it bipartite? etc…





Review – Graph DS: Adjacency Matrix, Adjacency List, and Edge List





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

0	→			
1	→			
2	→			
3	→			
4	→			
5	→			

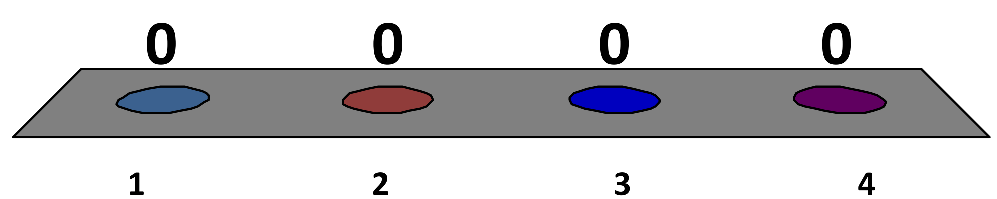
id	u	v	w
0			
1			
2			
3			
4			
5			

http://www.comp.nus.edu.sg/~stevenha/visualization/representation.html

Which One To Use? (1)

V = 10000, E = 10000

1. Adjacency Matrix
2. Adjacency List
3. Edge List
4. This is a trick question, the answer must be something else, which is __________________

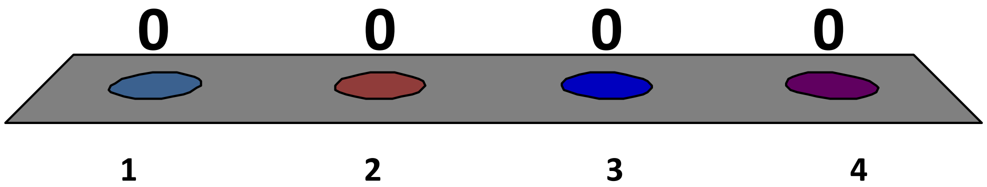


0 of 120

Which One To Use? (2)

V = 100, existence of edge(u, v) frequently asked

1. Adjacency Matrix
2. Adjacency List
3. Edge List
4. This is a trick question, the answer must be something else, which is _________________

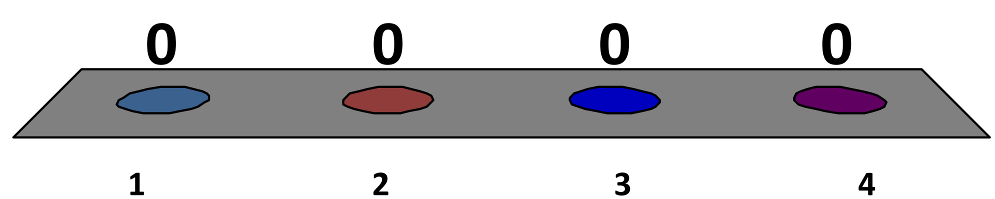


0 of 120

Which One To Use? (3)

V = 200, E = 19900, neighbors frequently enumerated

1. Adjacency Matrix
2. Adjacency List
3. Edge List
4. This is a trick question, the answer must be something else, which is _________________

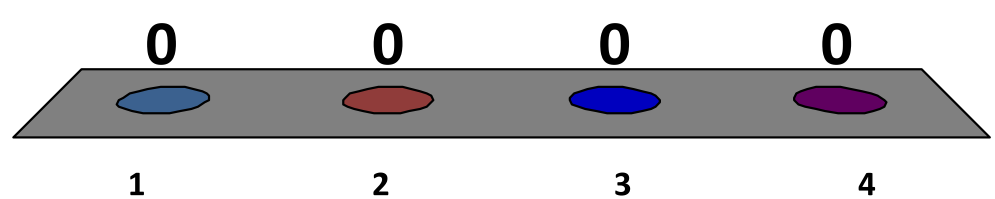


0 of 120

Which One To Use? (4)

V = 200, E = 19900, sort the edges based on weight

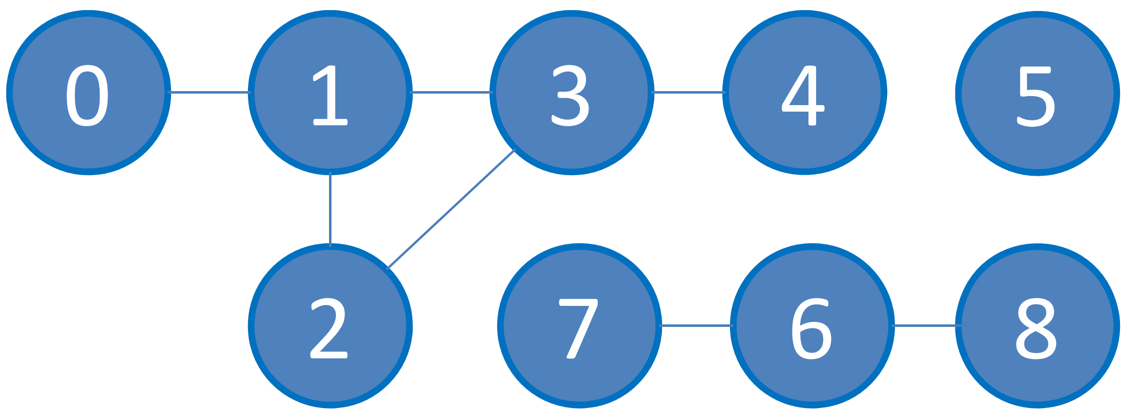
1. Adjacency Matrix
2. Adjacency List
3. Edge List
4. This is a trick question, the answer must be something else, which is ________________



0 of 120

Review – Graph Traversal

- Run BFS and then DFS from various source in the graph below



http://www.comp.nus.edu.sg/~stevenha/visualization/dfsbfs.html

Review: Path Reconstruction Algorithm (1)

```
// iterative version (will produce reversed output)
Output "(Reversed) Path:"
i ← t // start from end of path: suppose vertex t
while i != s
  Output i
  i ← p[i] // go back to predecessor of i
Output s



// try it on this array p, t = 4
// p = {-1, 0, 1, 2, 3, -1, -1, -1}
```

Review: Path Reconstruction Algorithm (2)

```
void backtrack(u)
  if (u == -1) // recall: predecessor of s is -1
    stop
  backtrack(p[u]) // go back to predecessor of u
  Output u // recursion will reverse the order

// in main method
// recursive version (normal path)
Output "Path:"
backtrack(t); // start from end of path (vertex t)
// try it on this array p, t = 4
// p = {-1, 0, 1, 2, 3, -1, -1, -1}
```

Hm… I prefer not to use recursion but I still want the correct path (from source to target), can I do that?

1. No, I have no choice but to use recursion to get the correct path
2. Possible, use this technique _______________

0 of 120

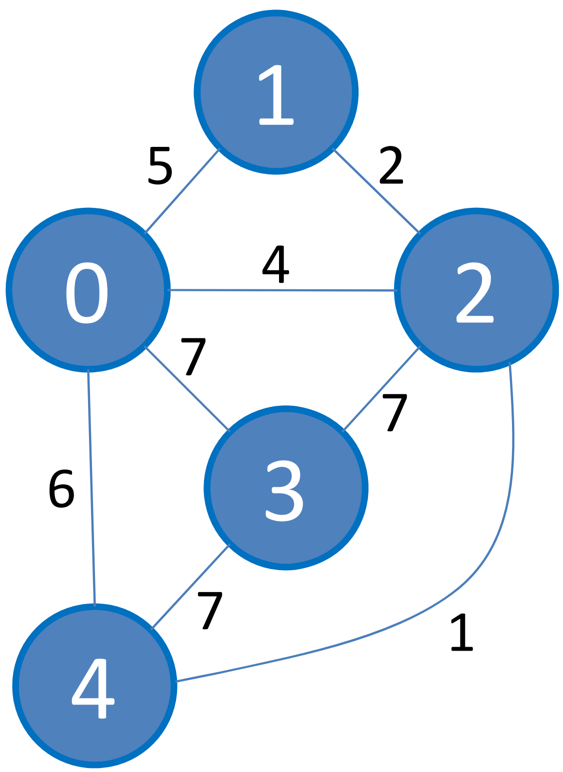
Review: What happen if we run toposort algorithm and the given graph is not a DAG?

1. There will be no topological order and the modified DFS algorithm(topoVisit) **will** be able to tell
2. There will be no topological order and the modified DFS algorithm (topoVisit) **will NOT** be able to tell

0 of 120

Quick Challenge

- Find MST of this connected weighted graph
  - Sort the edges and do the greedy strategy as shown earlier



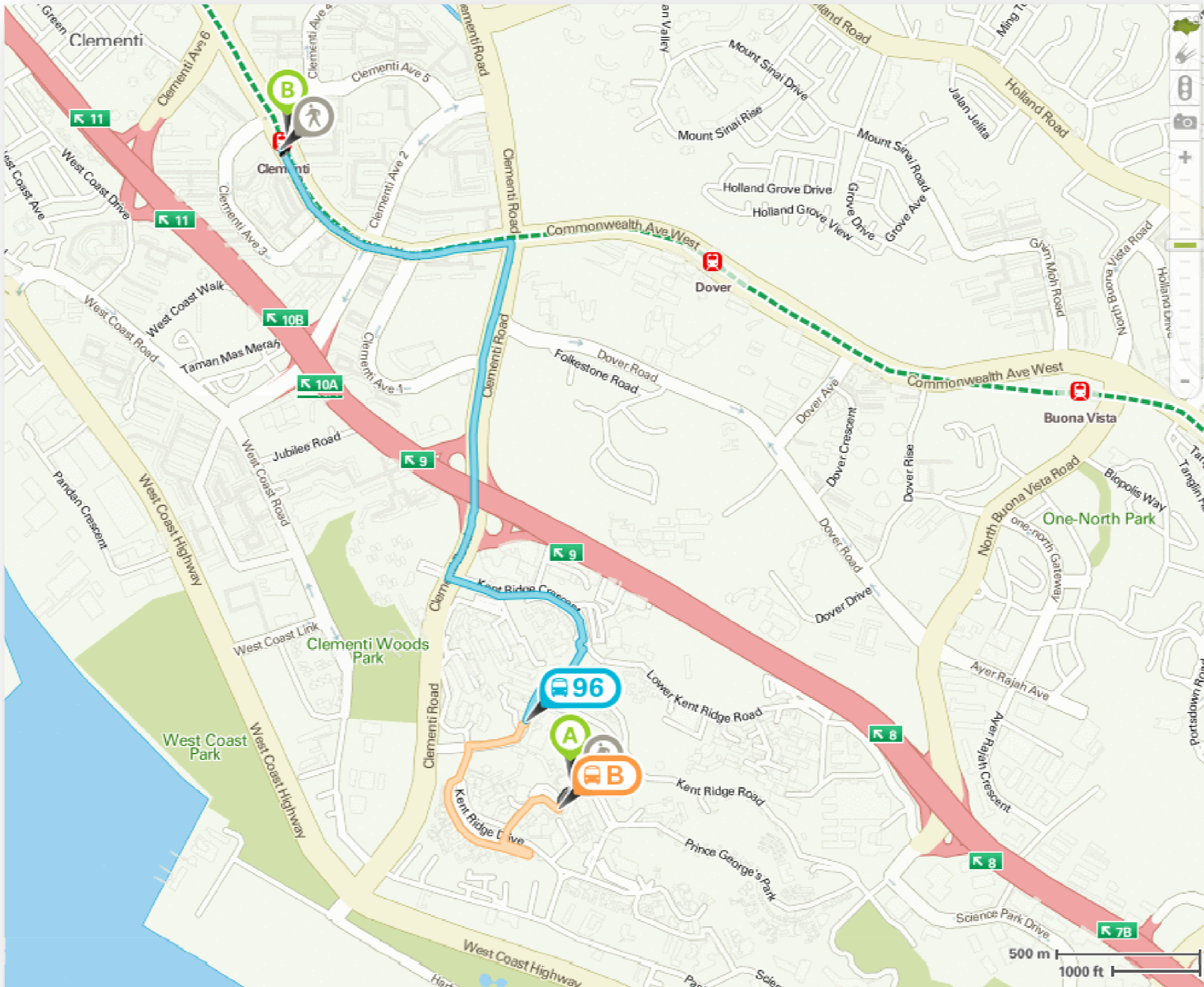
http://www.comp.nus.edu.sg/~stevenha/visualization/mst.html

5 minutes break

Outline (of the main lecture)

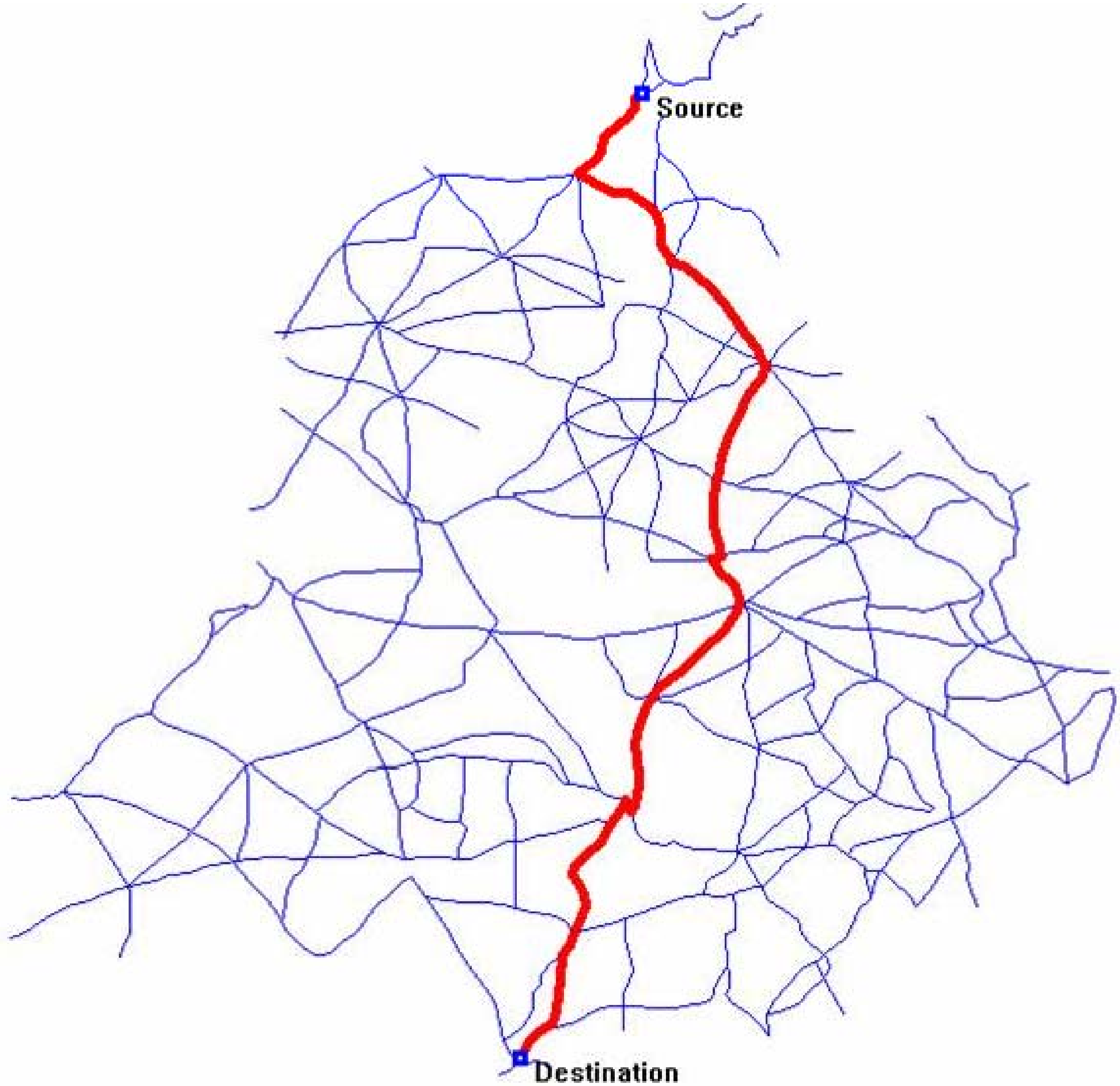
- What are we going to learn in this lecture?
  - Single Source Shortest Paths (SSSP) Problem
    - Motivating example
    - Some more definitions
    - Negative weight edges and cycles
  - Algorithms to Solve SSSP Problem (CP2.5 Section 4.4)
    - General SSSP Algorithm
    - Bellman Ford's Algorithm
      - Pseudo code, example animation, and later: Java implementation
      - Theorem, proof, and corollary about Bellman Ford's algorithm

Motivating Example



SINGLE SOURCE SHORTEST PATHS

(ON WEIGHTED GRAPHS)



More Definitions (1)

- **Weighted** Graph: **G(V, E), w(u, v): E→R**
- Vertex **V** (e.g. street intersections, houses, etc)
- Edge **E** (e.g. streets, roads, avenues, etc)
  - **Directed** (e.g. one way road, etc)
    - Note that we can simply use 2 edges (bi-directional) to model 1 undirected edge (e.g. two ways road, etc)
    - Recall that for MST problem discussed previously, we generally deal with **undirected weighted graph**
  - Weighted (e.g. distance, time, toll, etc)
- Weight function **w(u, v): E→R**
  - Sets the weight of edge from u to v

More Definitions (2)

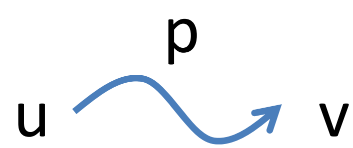
- **(Simple) Path** $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$
  - Where $(v_i, v_{i+1}) \in E, \forall_{0 \le i \le (k-1)}$
  - Simple = No repeated vertex!
- Shortcut notation: $v_0 \overset{p}{\rightsquigarrow} v_k$
  - Means that **p** is a path from v_0 to v_k
- **Path weight:** $PW(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$

More Definitions (3)

- **Shortest Path weight** from vertex u to v: **δ(u, v)**

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

If there exists such path



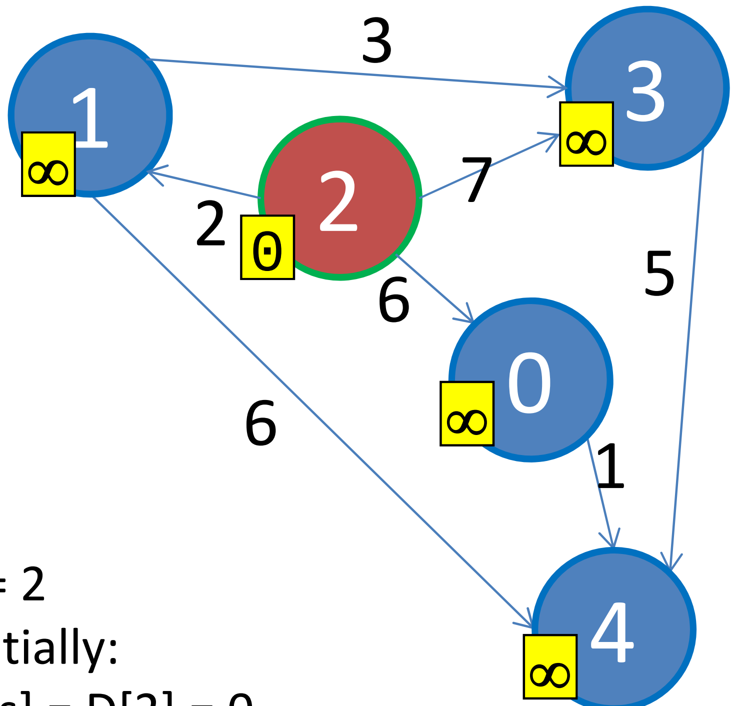
If **v** is unreachable from **u**

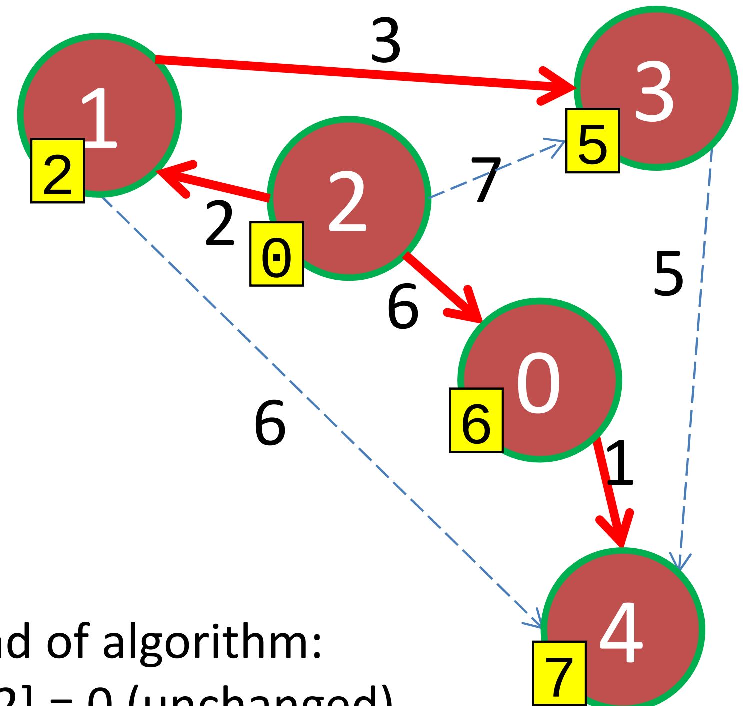
- <u>**Single Source**</u> **Shortest Path<u>s</u>** (SSSP) Problem:
  – Given **G(V, E)**, **w(u, v): E->R**, and a <u>**source vertex s**</u>
  – Find **δ(s, v)** (and the best paths) from **s** to each **v** ∈ **V**
    - i.e. From one source **to the rest**

More Definitions (4)

- **Additional Data Structures** to solve SSSP Problem:
  - An array/Vector **D** of size **V** (D stands for ‘distance’)
    - Initially, **D[v] = 0** if **v** = **s**; otherwise **D[v] =** $\propto$ (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 to v
    - **p[s]** = NULL (not defined, we can use a value like -1 for this)
    - Recall: The usage of this array/Vector p is already discussed in BFS/DFS Spanning Tree

Example



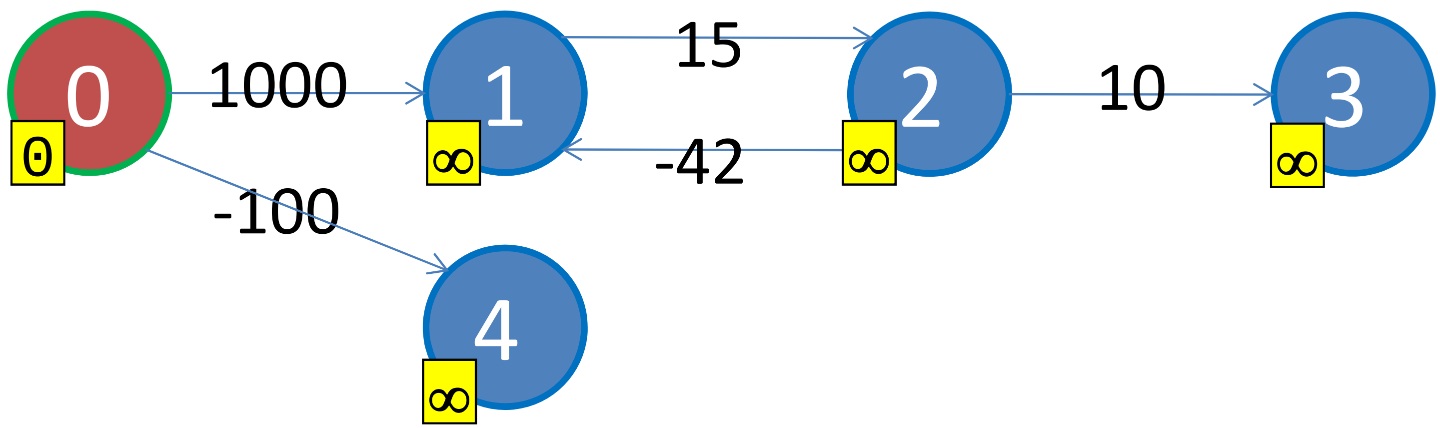


s = 2
Initially:
D[s] = D[2] = 0
D[v]= $\propto$ for the rest
Denoted as values in yellow boxes
p[s] = -1 (to say 'no predecessor')
p[v] = -1 for the rest
Denoted as **red arrows (none initially)**

s = 2
At the end of algorithm:
D[s] = D[2] = 0 (unchanged)
D[v] = δ(s, v) for the rest
e.g. D[0] = 6, D[4] = 7
p[s] = -1 (source has no predecessor)
p[v] = the origin of **red arrows** for the rest
e.g. p[0] = 2, p[4] = 0

Negative Weight Edges and Cycles

- They exist in some applications
  - Suppose you can travel back in time by passing through time tunnel (edges with negative weight)



- Shortest paths from 0 to {1, 2, 3} are **undefined**
  - One can take 1→2→1→2→1→… indefinitely to get -∝
- Shortest path from 0 to 4 is ok, with $\delta(0, 4) = -100$

SSSP Algorithms

- This SSSP problem is a **well-known** CS problem
- We will discuss three algorithms in this lecture:
  - O(?) "General" SSSP Algorithm
    - Introducing the "initSSSP" and "Relax" operations
  - O(VE) Bellman Ford's SSSP algorithm
    - Trick to ensure termination of the algorithm
    - Bonus: detecting negative weight cycle
    - *Note to self: Go slower… This one has low clicker score on Week05*
  - O(V+E) BFS fails for *general case* SSSP problem

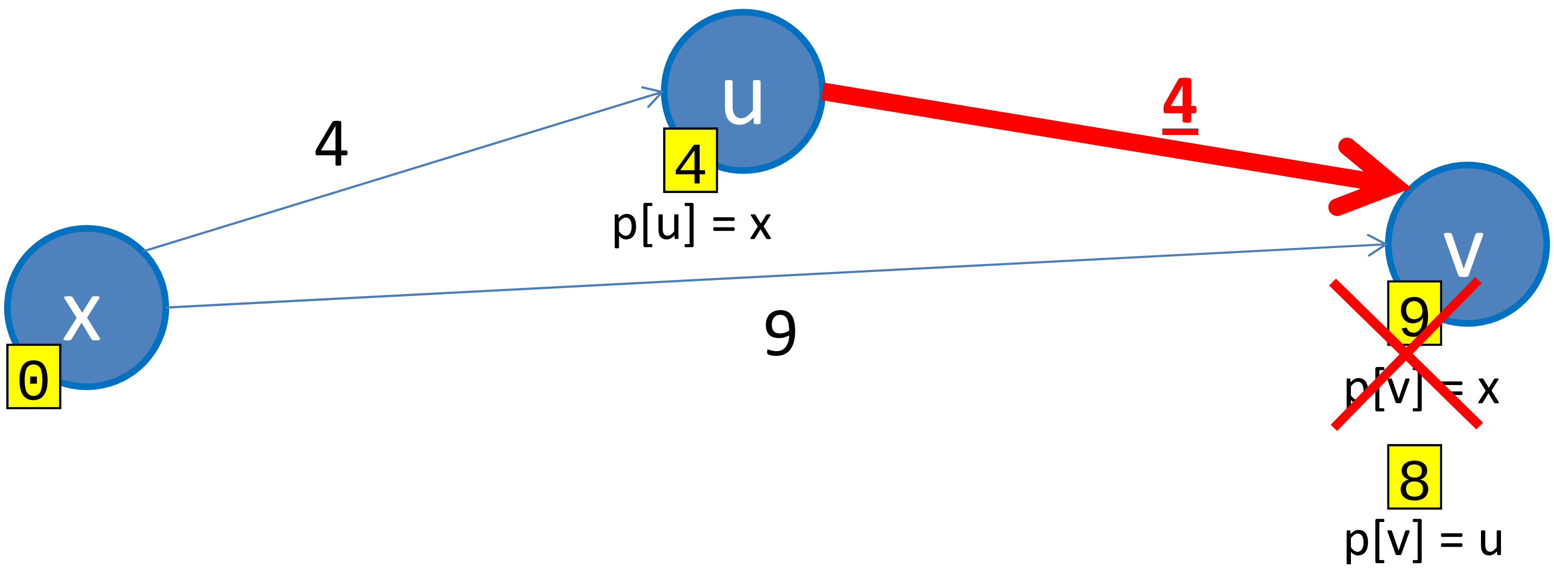
Initialization Step

- We will use this initialization step for all our SSSP algorithms

```
initSSSP(s)
  for each v ∈ V // initialization 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
```

“Relax” 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
```



General SSSP Algorithm

```
initSSSP(s) // as defined in previous two slides

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)
```

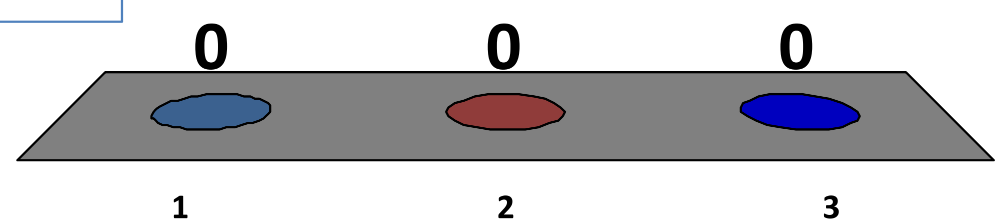
Q: Will this Algorithm Terminate?

(for now, use your feeling, the answer is in the next slide)

1. Yes, what is the problem?
2. No, because ______
3. Not always, because ________________

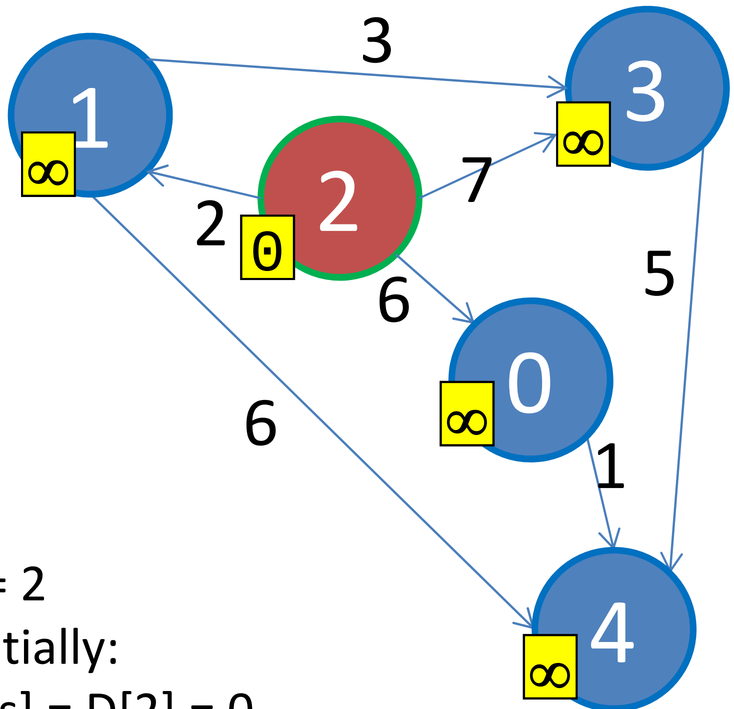
```
initSSSP(s) // as defined in previous two slides

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)
```

0 of 120

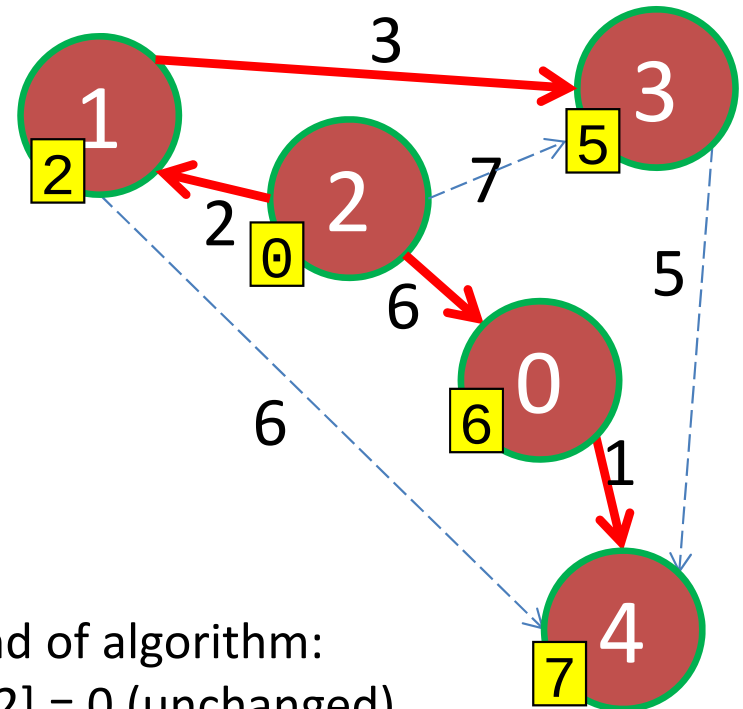


Example

(Revisited – Demo on Whiteboard)



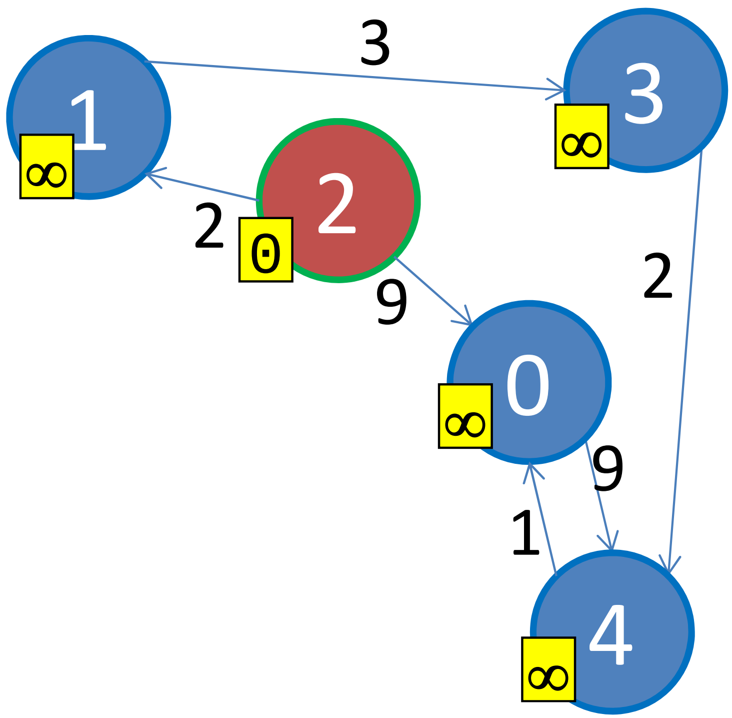
s = 2
Initially:
D[s] = D[2] = 0
D[v]= $\propto$ for the rest
Denoted as values in yellow boxes
p[s] = -1 (to say ‘no predecessor’)
p[v] = -1 for the rest
Denoted as **red arrows (none initially)**



s = 2
At the end of algorithm:
D[s] = D[2] = 0 (unchanged)
D[v] = δ(s, v) for the rest
e.g. D[0] = 6, D[4] = 7
p[s] = -1 (source has no predecessor)
p[v] = the origin of **red arrows** for the rest
e.g. p[0] = 2, p[4] = 0

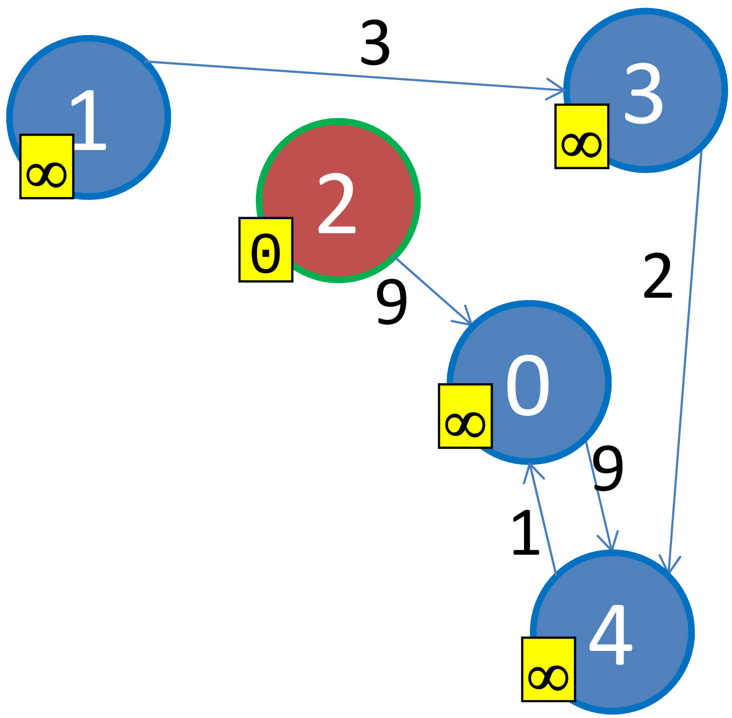
Quick Challenge (1)

- Find the shortest paths from s = 2 to the rest



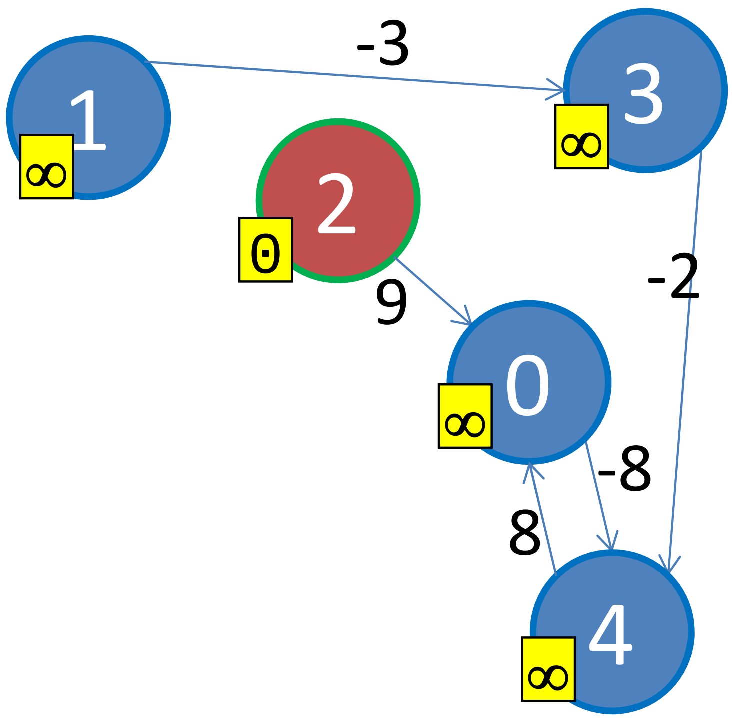
Quick Challenge (2)

- Find the shortest paths from s = 2 to the rest
  - This time, edge (2, 1) is removed



Quick Challenge (3)

- Find the shortest paths from s = 2 to the rest
  - This time, some edges are negative, but no negative cycle



Java Implementation (1)

- See GenericSSSP.java
  - Implemented using EdgeList to facilitate easier random-edge selection
    - This is the same as the one shown in MST lecture
  - With path reconstruction subroutine (if terminate)
    - This is the same as the one shown in BFS/DFS lecture
- Show performance on:
  - Small graph **without** negative weight cycle
    - OK
  - Small graph **with** negative weight cycle
    - Erm… the algorithm _____________ stop…
  - Small graph with some negative edges; no negative cycle
    - OK

Algorithm Analysis

- If given a graph without negative weight cycle, when will this Generic SSSP algorithm terminate?
  - A: Depends on your luck…
  - A: 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*!

Reference: CP2.5 Section 4.4 (especially Section 4.4.4)

http://www.comp.nus.edu.sg/~stevenha/visualization/sssp.html

BELLMAN FORD'S SSSP ALGORITHM

General SSSP Algorithm (Revisited)

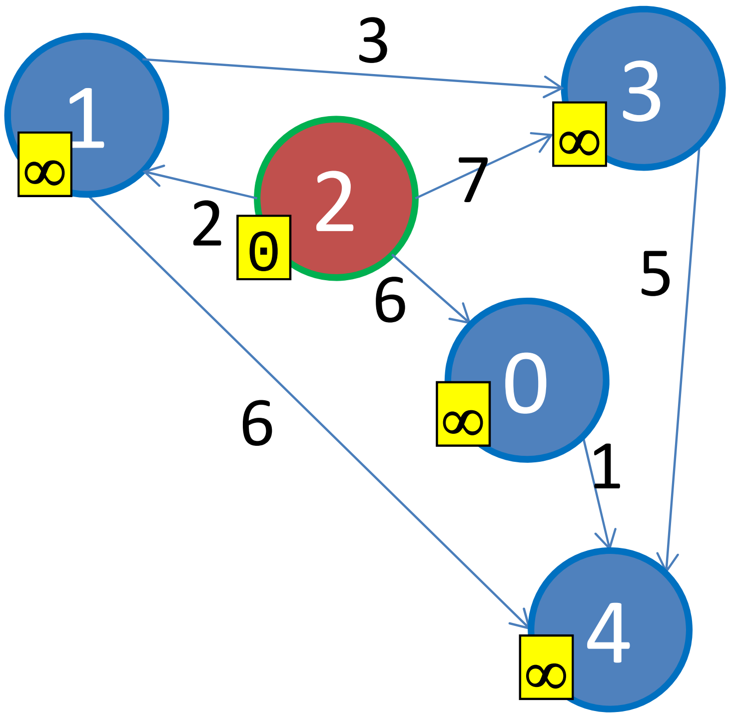
- What do we lack in the generic algorithm below?
  - An **"order"** of edge relaxation

```
initSSSP(s)

repeat
  select edge(u, v) ∈ E in arbitrary manner
  relax(u, v, w_u_v)
until all edges have D[v] <= D[u] + w(u, v)
```

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

// The chosen order is remarkably simple...
// "repeat relaxation on all edges V – 1 times"
// Question: Will it work?
```

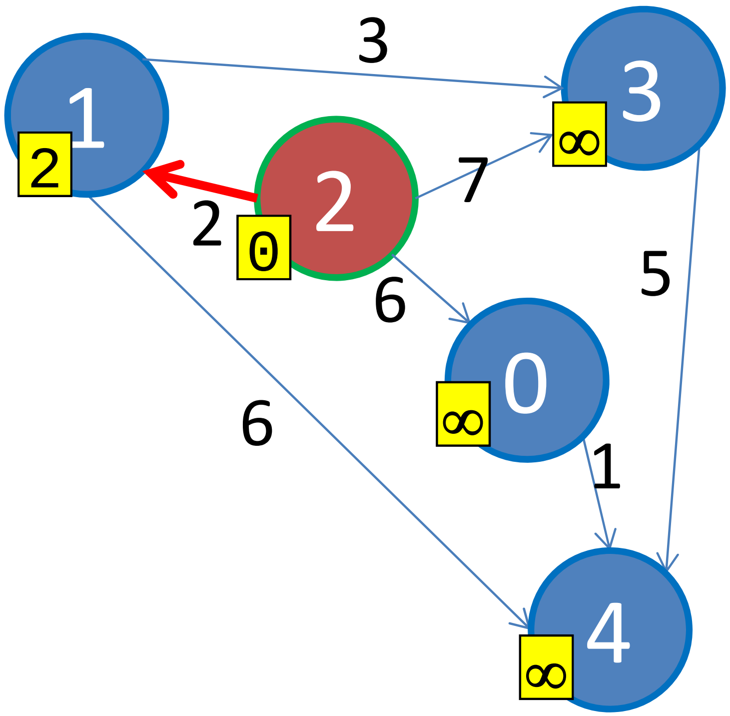
Bellman Ford's Animation (0)



Suppose the edges are stored in this order:

(1, 4), w = 6
(1, 3), w = 3
(2, 1), w = 2
(0, 4), w = 1
(2, 0), w = 6
(3, 4), w = 5
(2, 3), w = 7

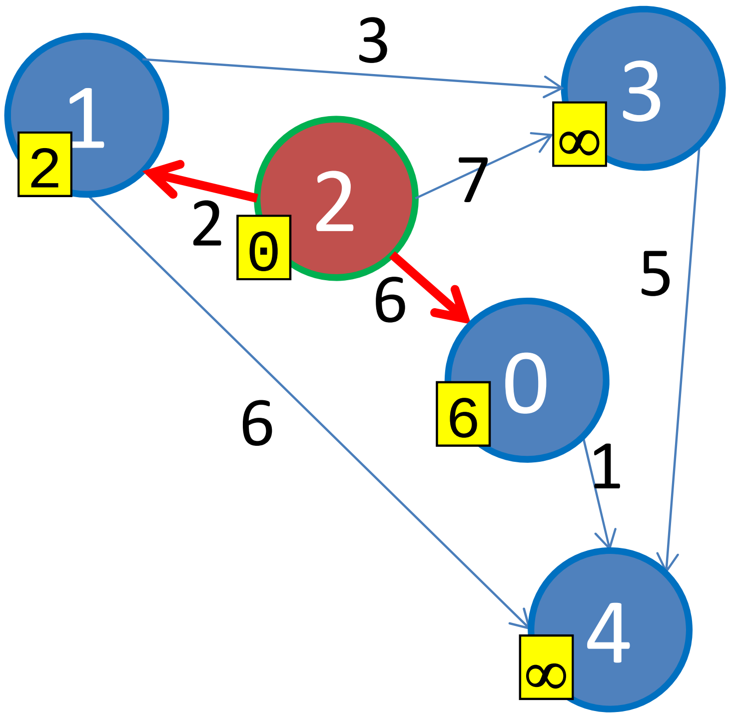
Bellman Ford's Animation (1a)



Suppose the edges are stored in this order:

(1, 4), w = 6
(1, 3), w = 3
→ (2, 1), w = 2
(0, 4), w = 1
(2, 0), w = 6
(3, 4), w = 5
(2, 3), w = 7

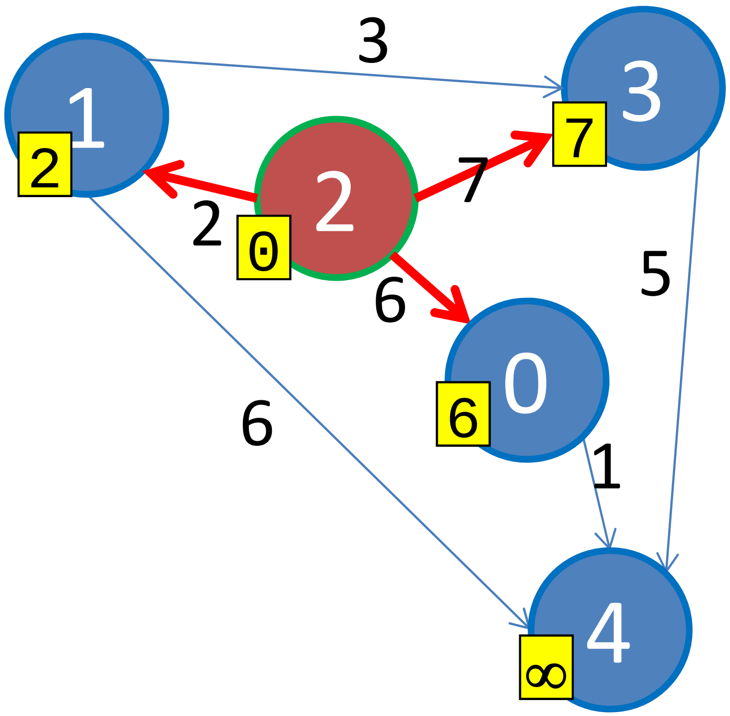
Bellman Ford's Animation (1b)



Suppose the edges are stored in this order:

(1, 4), w = 6
(1, 3), w = 3
(2, 1), w = 2
(0, 4), w = 1
→ (2, 0), w = 6
(3, 4), w = 5
(2, 3), w = 7

Bellman Ford's Animation (1c)

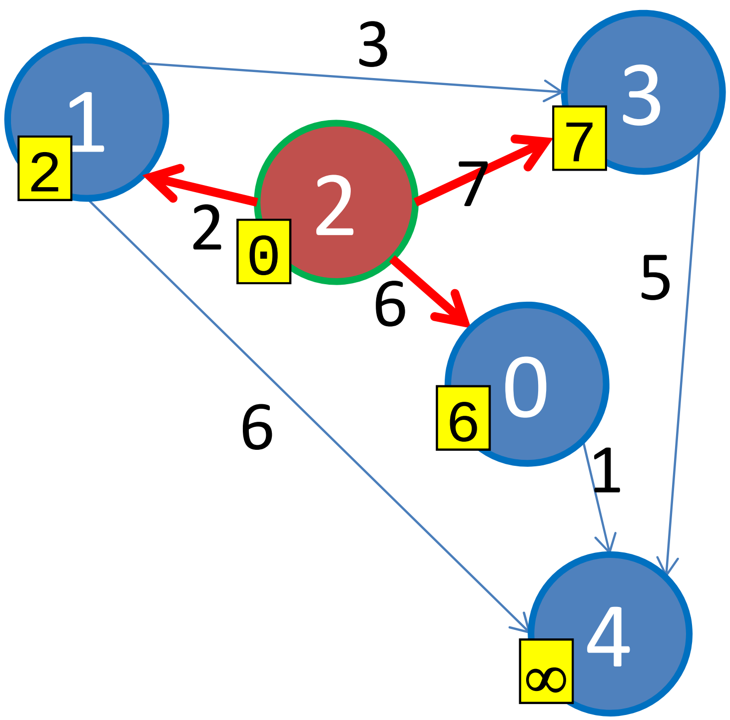


Suppose the edges are stored in this order:

(1, 4), w = 6
(1, 3), w = 3
(2, 1), w = 2
(0, 4), w = 1
(2, 0), w = 6
(3, 4), w = 5
→ (2, 3), w = 7

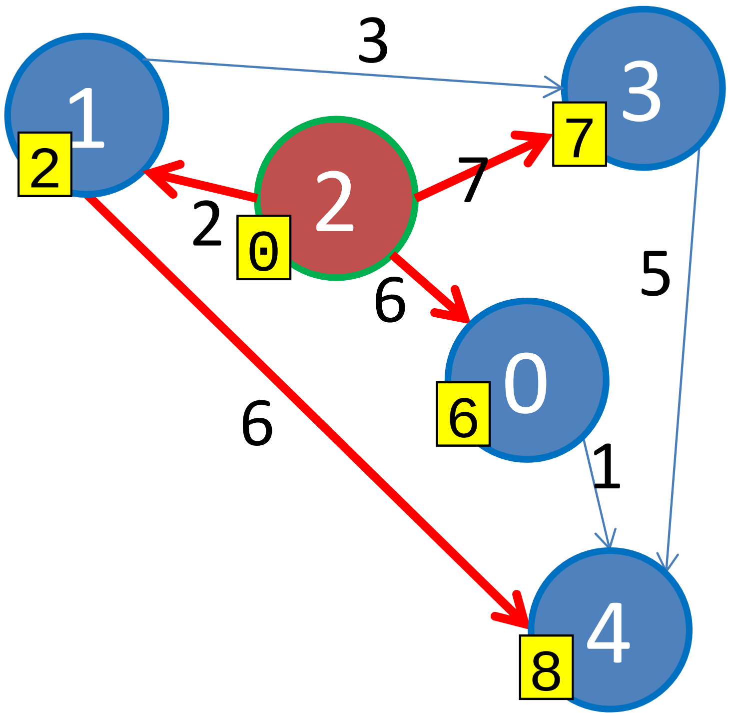
One pass through all edges is now done.
Is there any more edges that can be relaxed?

1. Yes, for example, edge(s) ___________
2. No more, we are done



0 of 120

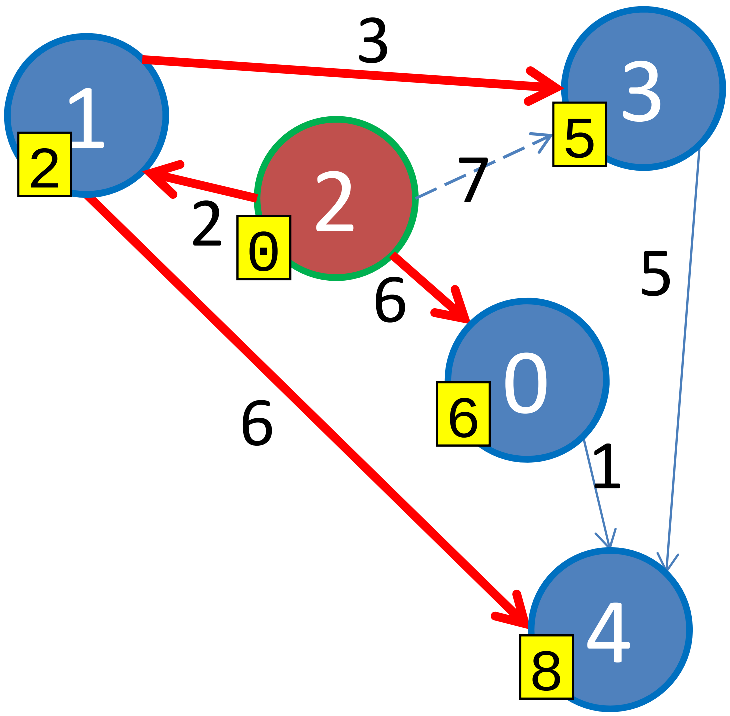
Bellman Ford's Animation (2a)



Suppose the edges are stored in this order:

→ (1, 4), w = 6
(1, 3), w = 3
(2, 1), w = 2
(0, 4), w = 1
(2, 0), w = 6
(3, 4), w = 5
(2, 3), w = 7

Bellman Ford's Animation (2b)

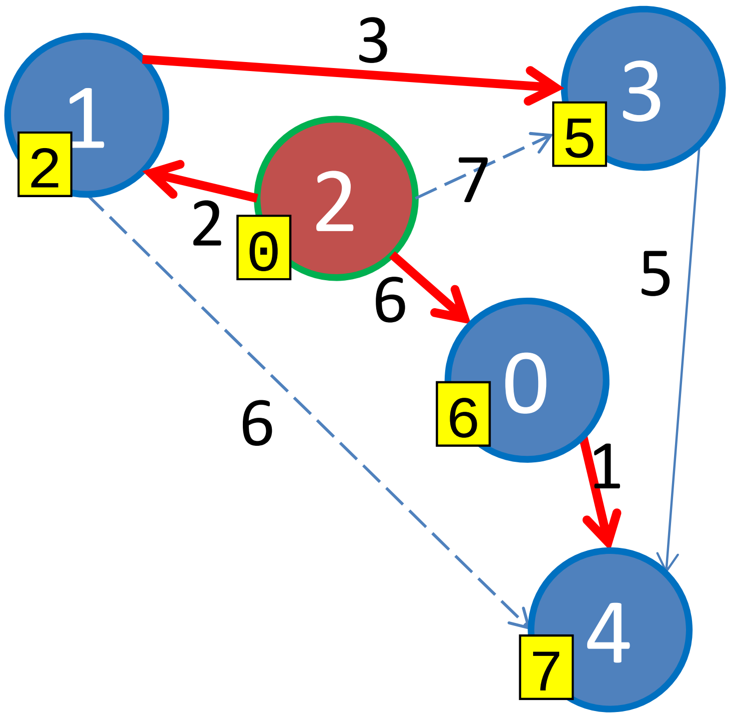


Suppose the edges are stored in this order:

(1, 4), w = 6
→ (1, 3), w = 3
(2, 1), w = 2
(0, 4), w = 1
(2, 0), w = 6
(3, 4), w = 5
(2, 3), w = 7

Observe that when we relax(1,3),
D[3] drops from 7 to 5
p[3] changes from 2 to 1

Bellman Ford's Animation (2c)

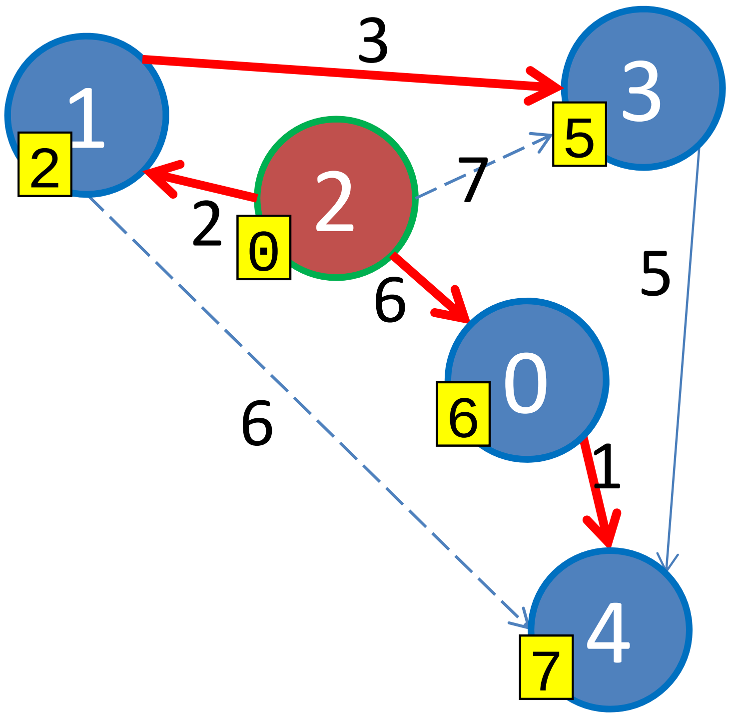


Suppose the edges are stored in this order:

(1, 4), w = 6
(1, 3), w = 3
(2, 1), w = 2
→ (0, 4), w = 1
(2, 0), w = 6
(3, 4), w = 5
(2, 3), w = 7

Observe that when we relax(0,4), D[4] drops from 8 to 7 and p[4] changes from 1 to 0

Bellman Ford's Animation (2d)



Suppose the edges are stored in this order:

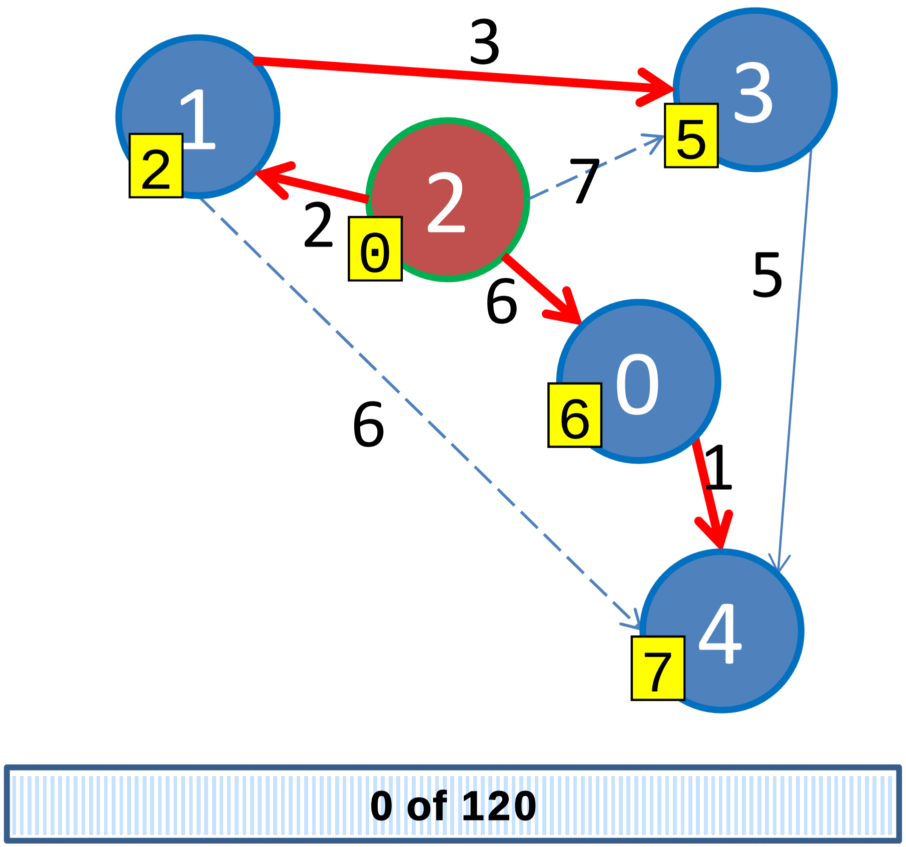
(1, 4), w = 6
(1, 3), w = 3
(2, 1), w = 2
(0, 4), w = 1
(2, 0), w = 6
(3, 4), w = 5
(2, 3), w = 7

Bellman Ford's will still go through all set of edges 2 more times, but with no further changes

We call the set of edges in red as the Shortest Paths (Spanning) Tree of the graph from source s

Now check. Does every D[v] = δ(s, v)?

1. Yes
2. No, because _______



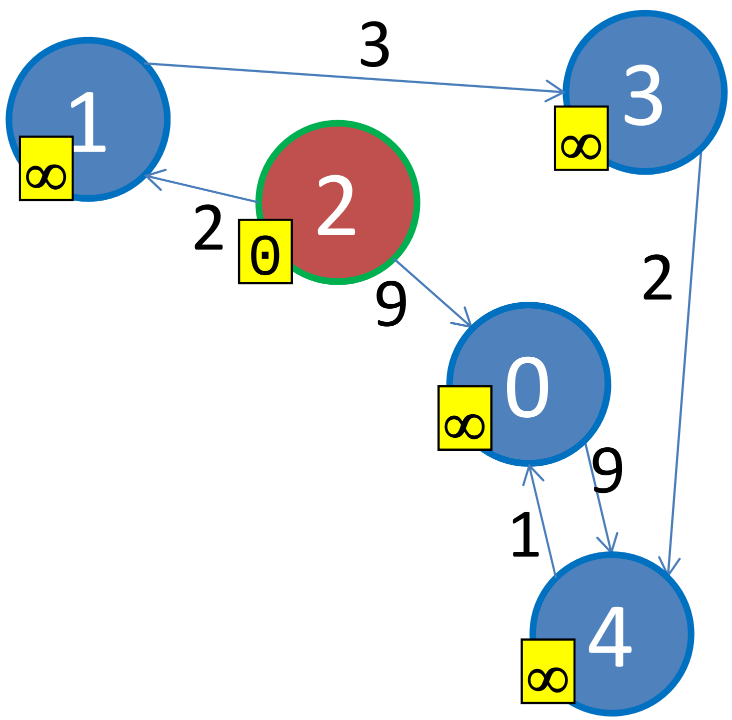
Visualization

- Let’s take a look at the SSSP visualization:

  www.comp.nus.edu.sg/~stevenha/visualization/sssp.html

Quick Challenge

- Run Bellman Ford's on this weighted graph (slide 58)
  - Do you get correct $D[v] = \delta(s, v), \forall v \in V$ again?



Suppose the edges are stored in this order:

(0, 4), w = 9
(4, 0), w = 1
(3, 4), w = 2
(1, 3), w = 3
(2, 1), w = 2
(2, 0), w = 9

Proof: Shortest Path on a graph without negative weight cycle is a Simple Path

- Theorem:
  - If **G = (V, E)** contains no negative weight cycle,
    then shortest path **p** from **s** to **v** is a **simple path**
- Proof by Contradiction:
  - Suppose **p** is not a simple path
  - Then **p** contains one (or more) cycle(s)
  - Suppose there is a cycle **c** in **p** with positive total weight
  - If we remove **c** from **p**, then we have a shorter shortest path than **p**
  - This contradicts the fact that **p** is a shortest path
  - Even if **c** is a cycle with zero total weight (it is possible!),
    we can still remove **c** from **p** without increasing the shortest path weight of **p**
  - So, **p** is (and can always be made into) a simple path
  - In another word, **p** has at most **|V|-1** edges from source **s** to the "furthest possible" vertex $\mathbf{v}_{|V|-1}$ in **G** (in terms of number of edges in the shortest path)

Correctness of Bellman Ford's

- Theorem:
  - If **G = (V, E)** contains no negative weight cycle, then after Bellman Ford's terminates $\mathbf{D[v] = \delta(s, v)}$, $\forall$ $\mathbf{v \in V}$
- Proof by Induction:
  - Consider shortest path **p** from **s** to $\mathbf{v_i}$ (**p** will have minimum number of edges)
    - $\mathbf{v_i}$ is defined as a vertex which shortest path requires *i* hops (number of edges) from s
  - Initially $\mathbf{D[v_0] = \delta(s, v_0) = 0}$, as $\mathbf{v_0}$ can be no other than **s**
    - It will not be changed since there is no negative cycle
  - After **1** pass through **E**, we have $\mathbf{D[v_1] = \delta(s, v_1)}$
  - After **2** passes through **E**, we have $\mathbf{D[v_2] = \delta(s, v_2)}$, …
  - After **k** passes through **E**, we have $\mathbf{D[v_k] = \delta(s, v_k)}$
  - When there is no negative weight cycle, shortest path **p** will be simple
    - At most **|V|-1** edges for the "longest" shortest path in terms of number of edges used
  - 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 in *worst possible order*

“Side Effect” of Bellman Ford’s

- 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 algorithm:

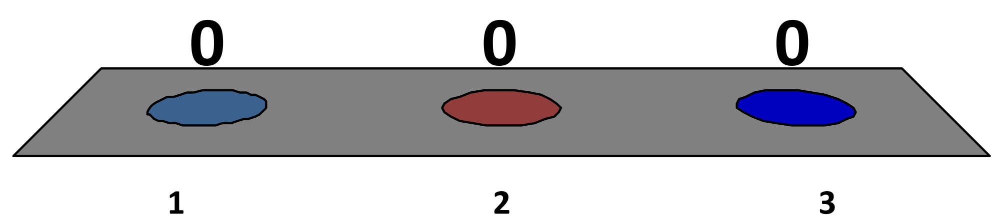
```
for each edge(u, v) ∈ E
  if D[v] > D[u] + w(u, v)
    report negative weight cycle exists in G
```

Java Implementation (2)

- See BellmanFordDemo.java
  - Now implemented using **AdjacencyList** ☺
    - You have a flexibility on choosing which graph data structure to use!
    - Both **AdjacencyList** and **EdgeList** can be used to have an O(VE) Bellman Ford's performance
- Show performance on:
  - Small graph without negative weight cycle
    - OK and time complexity is bounded by O(VE) steps
  - Small graph with negative weight cycle
    - Terminate and able to report that negative weight cycle exists
    - Time complexity is bounded by O(VE) steps
  - Small graph with some negative edges; no negative cycle
    - OK and time complexity is bounded by O(VE) steps

During Lecture 5, only minority of you said that you know/have implement Bellman Ford's algorithm before. Now…

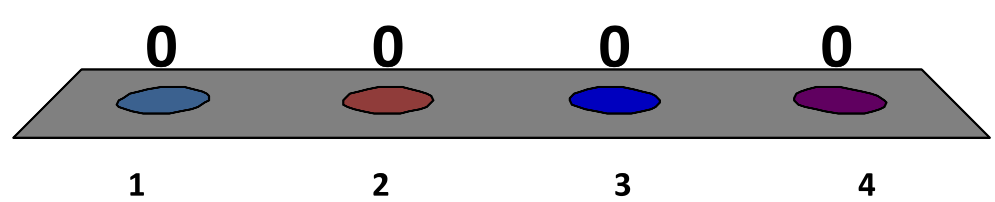
1. Bellman Ford's algorithm looks easy, I am now sure I can implement and use it to solve any SSSP problem
2. Bellman Ford's algorithm may be easy, but I know you can set hard SSSP question??
3. I think I still need more time to revise this lecture material… Still not sure how Bellman Ford's works



0 of 120

For next **e-Lecture**: What is your level of understanding of the other SSSP algorithm: Dijkstra's?

1. I have never heard about this algorithm before
2. This is a popular algorithm, I have heard about it but not the details
3. I know the algorithm details but have never implemented it before
4. I have implemented Dijkstra's algorithm to solve some SSSP problems before



0 of 120

Summary

- Mid-semester Review
- Introducing the SSSP problem
- Introducing the Generic SSSP algorithm
  - You can "forget" this algorithm after this lecture ☺
- Introducing the Bellman Ford's algorithm
  - This one solves SSSP for general weighted graph in O(VE)
  - Can also be used to detect the presence of -ve weight cycle
- Next week (Week08) is SoC e-learning week
  - It is not a holiday!
  - Please watch the e-lecture as if you attend the normal class