EECS 4020 Algorithms

HW6

I. Minimum Spanning Tree

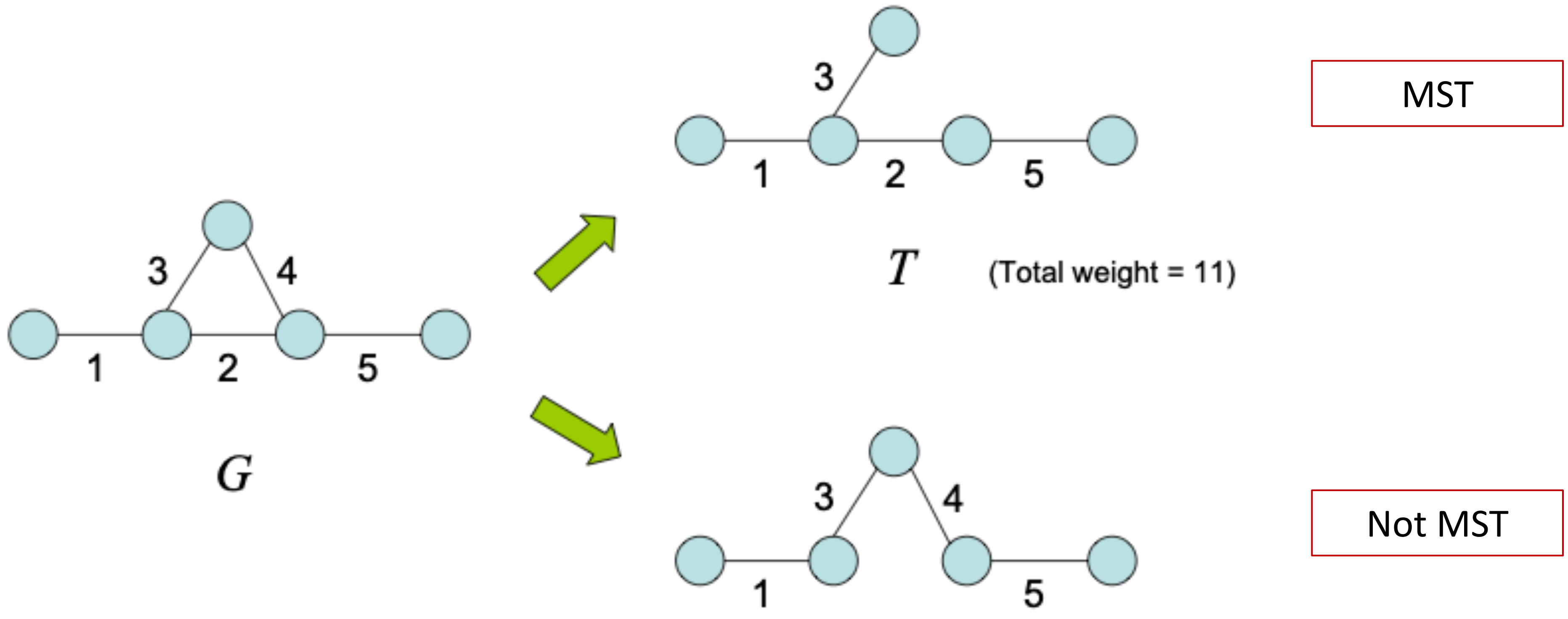
Q1

- Let G be a connected edge-weighted graph
- Bottleneck spanning tree of G is a spanning tree whose max weight edge has weight minimized

Is an MST always a bottleneck spanning tree?

Q1

Is a bottleneck spanning tree always an MST?



Q1

- The above implies that :

Bottleneck spanning tree can be found

by using any MST algorithm,

but it may be found by some faster way

Q1 [solution]

Can we find it in linear time ?

Key Observation :

If edges with weight < w can connect the graph

➔ Edges with weight ≥ w are not needed;

Else, edges with weight ≤ w are all needed

Q1 [solution]

Algorithm:

1. Find the median weight w
2. Check if edges with weight < w connect the graph
   - ➔ If so, remove edges with weight ≥ w ;
   - ➔ Else, pick all edges with weight ≤ w ;

     Contract each component into a node
3. Goto Step 1 if G has more than 1 node

Q1 [solution]

Running time :

1. Finding median takes $O(\ |E|\ )$ time
2. Contracting components takes $O(\ |E|\ )$ time
3. After Step 1 and Step 2, $|E|$ drops by half

   (that's why we chose w as median weight)

➔ Running time = $|E| + |E|/2 + |E|/4 + \ldots = O(|E|)$

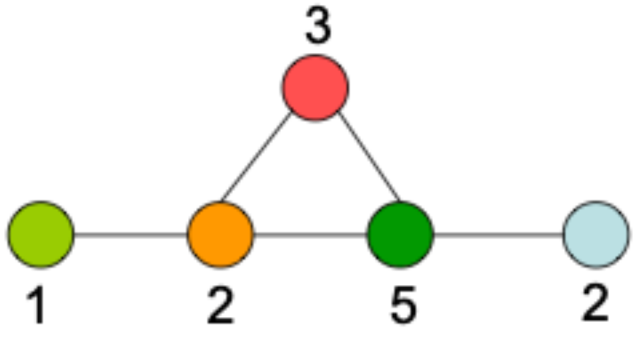
Q2

- Let G be a connected node-weighted graph
- Each node v has non-negative weight w(v)
- Target: Find a spanning tree T of G such that

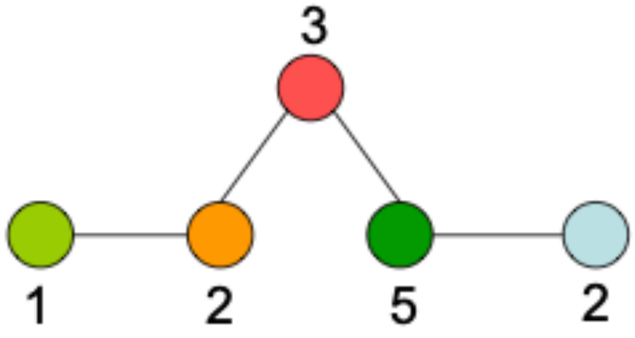
$$\sum_{v \in T} w(v) \times \deg_T(v)$$

is minimized

Q2 [Example: Weight of a spanning tree]



G



T

Weight of T

$= 1 * 1 + 2 * 2 + 3 * 2 + 5 * 2 + 2 * 1 = 23$

Q2 [solution]

How to solve the problem ?

Key Observation :

The cost of including an edge (u, v)
is equal to w(u) + w(v)

Q2 [solution]

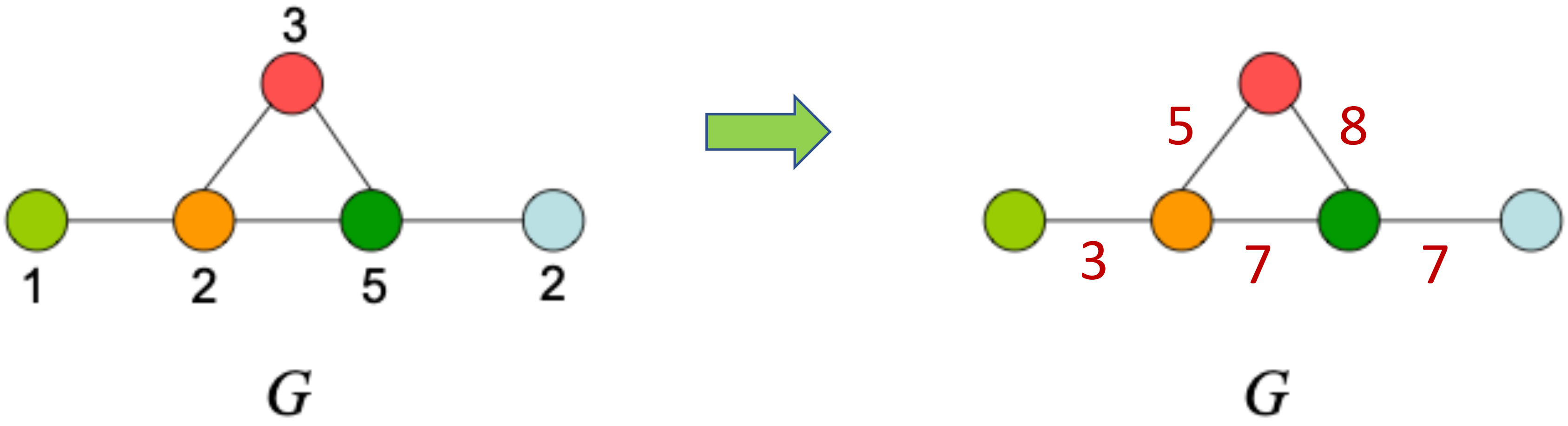
- The key observation allows us to transform G into an edge-weighted graph, with

$$\text{weight}(\text{edge}(u, v)) = w(u) + w(v)$$

so that the MST will be the desired solution

➔ Running time = $O(\,|E| + |V| \log |V|\,)$

Q2



II. Single-Source Shortest Path

Q1

- Let G be a DAG and s be a source vertex

How to count # paths from s to any vertex?

Key Idea: Topological Sort, then DP

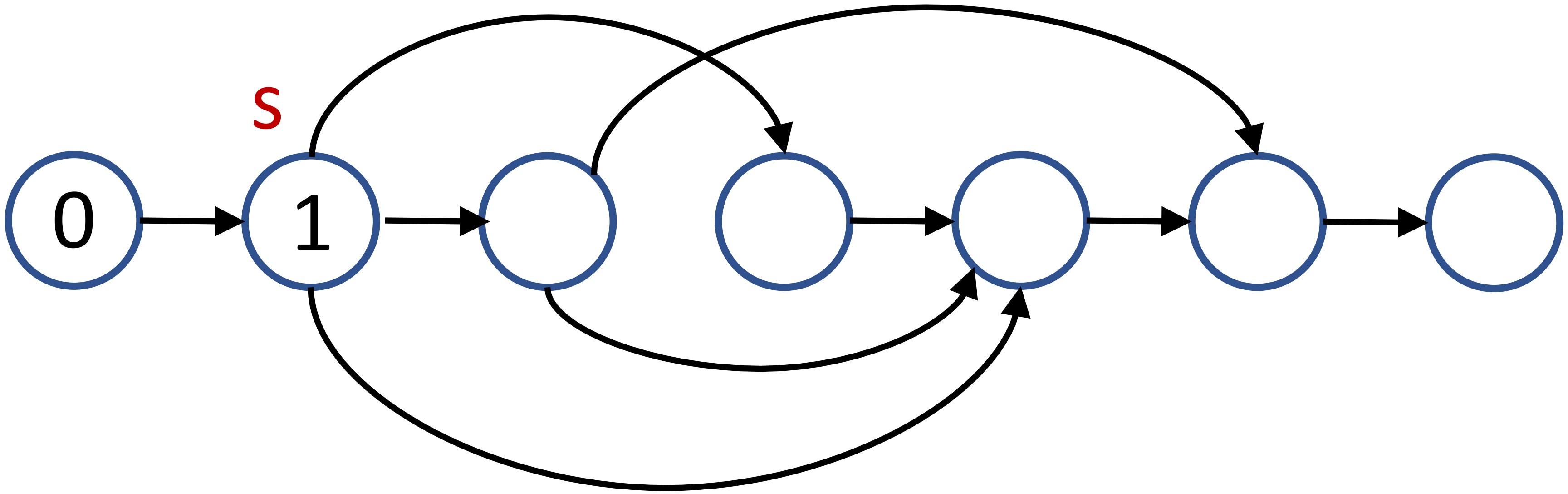
Q1 [solution]

1. Topological sort on G
2. For each vertex v: set #paths(v) = 0
3. For source s: set #paths(u) = 1
4. Process vertices v in topological-sort order:

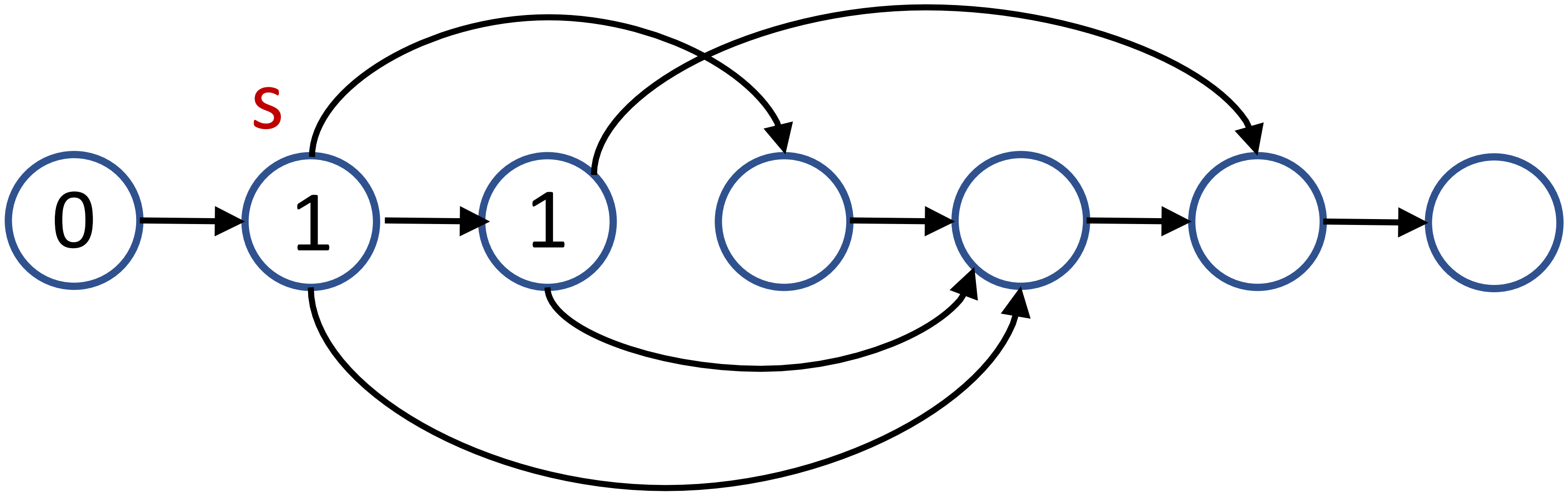
   For every in-neighbor u of v

   Increase #paths(v) by #paths(u) [relax]

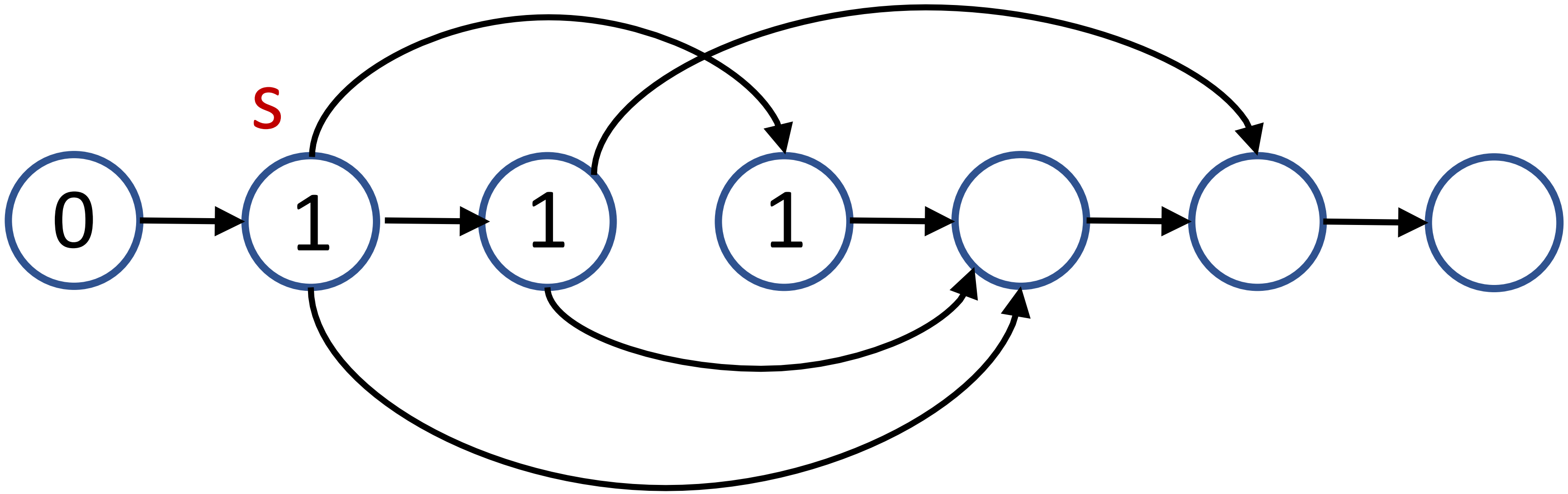
Q1 [example]



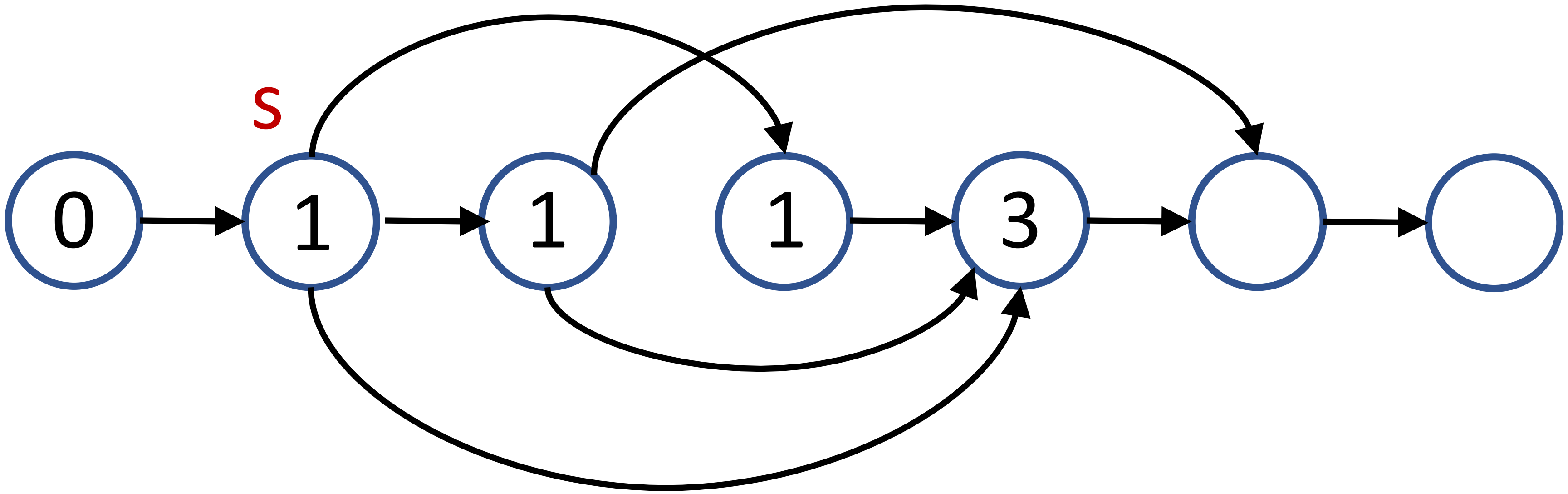
Q1 [example]



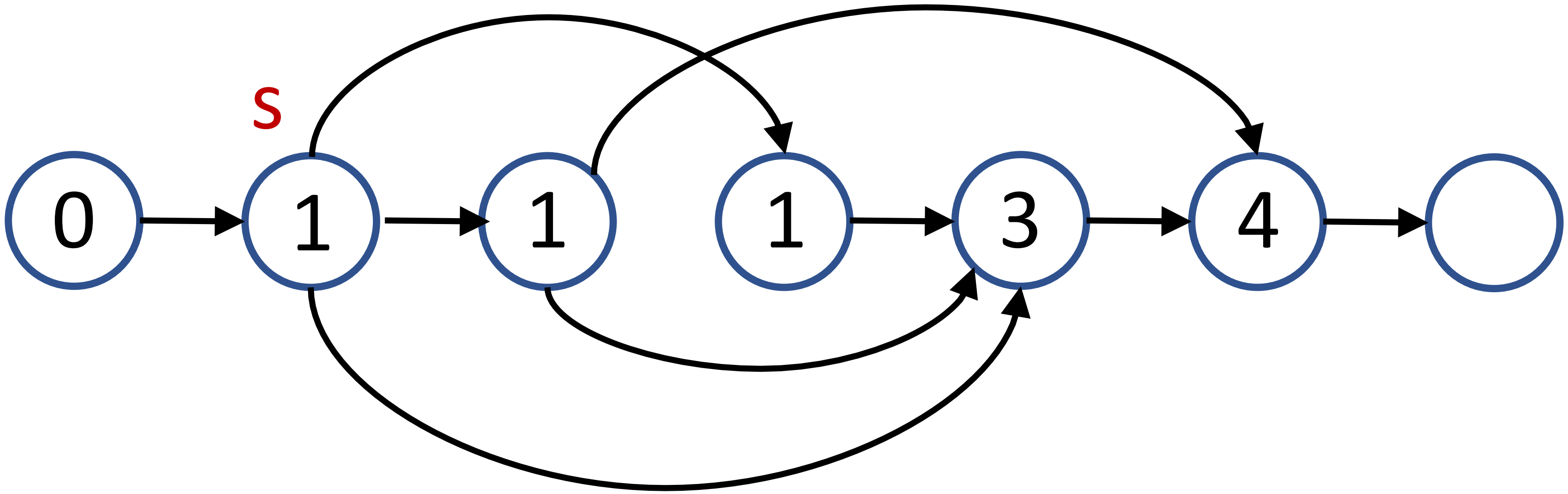
Q1 [example]



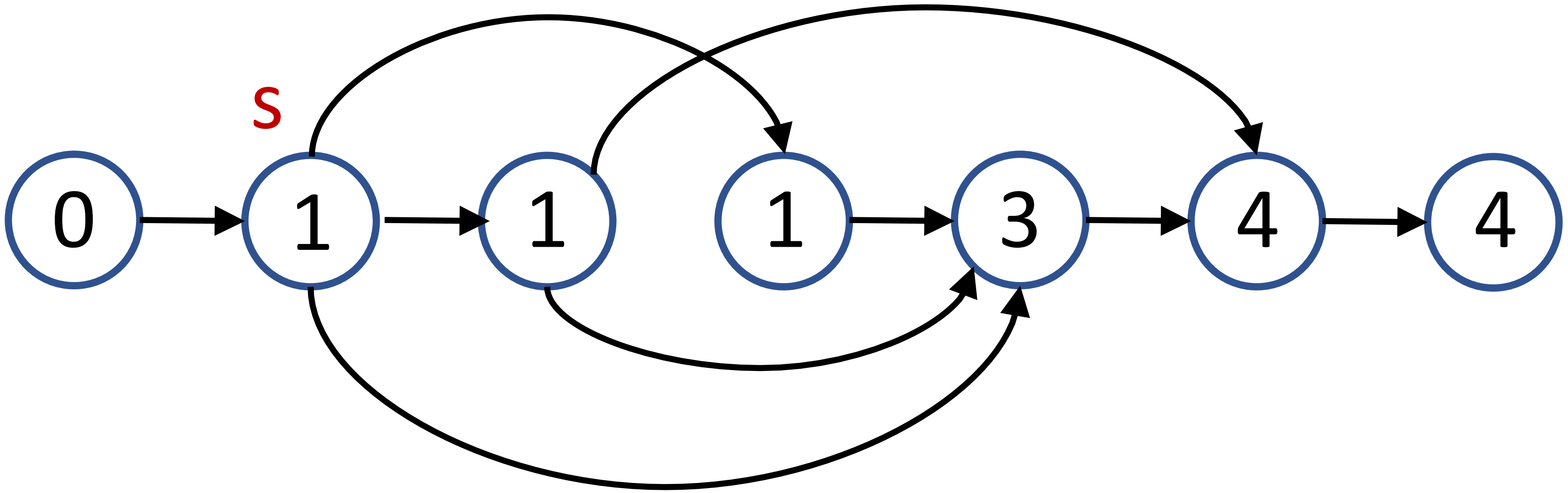
Q1 [example]



Q1 [example]



Q1 [example]



Q2

- Let $G = (V, E)$ be an edge-weighted graph
- Edge weight are integers from $\{ 1, 2, ..., W \}$

$$\boxed{\text{How to solve SSSP in } O(W|V| + |E|) \text{ time?}}$$

Key Idea:

All shortest distances are within $\{ 1, 2, ..., W|V| \}$

Q2 [Dial's algorithm]

- Let us solve a more general problem
- Suppose we know that all shortest distances are integers, and each does not exceed a value X

How to solve SSSP in $O(X + |E|)$ time?

Q2 [Dial's algorithm]

- Maintain $X + 1$ lists L_0, L_1, L_2, ..., L_X

  -- L_k keeps vertices with a known path of distance k from source (Initially, put source s to L_0)

- Process lists one by one :

  -- For each unvisited vertex v in L_k :

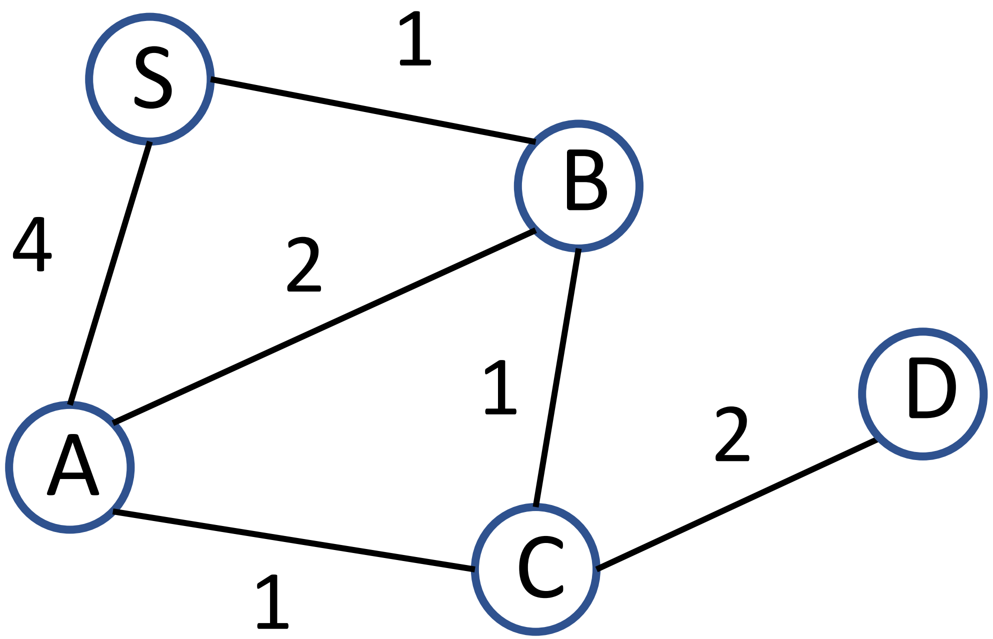
  Set v as visited and dist(v) = k ;

  Relax all neighbors and put them to corresponding list

Q2 [Dial's algorithm]

L_0
L_1
L_2
L_3
L_4

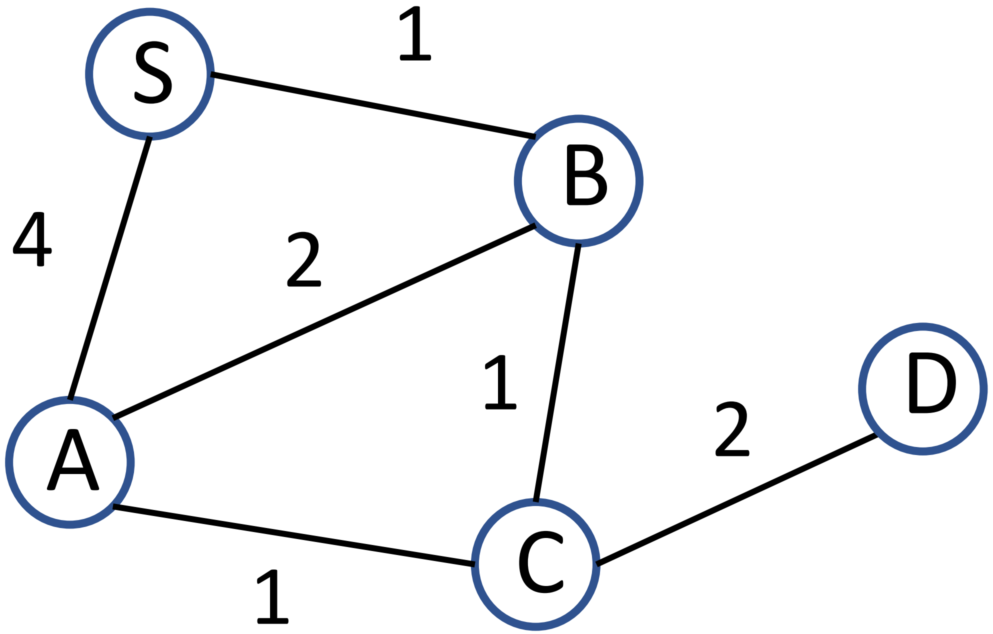
S



	S	A	B	C	D
dist	0				

Q2 [Dial's algorithm]

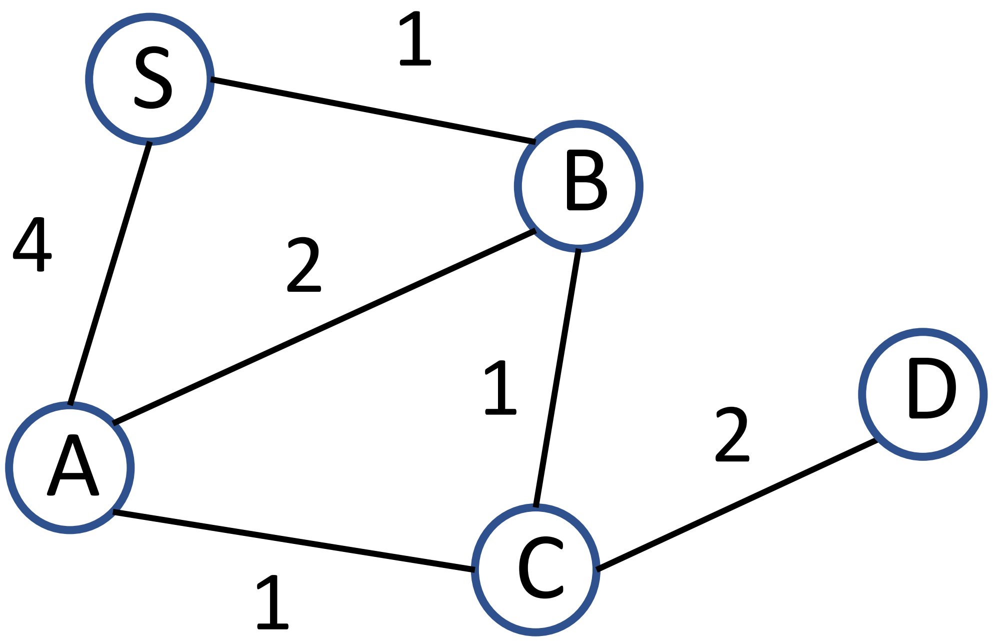
L_0	~~S~~
L_1	B
L_2	
L_3	
L_4	A



	S	A	B	C	D
dist	0				

Q2 [Dial's algorithm]

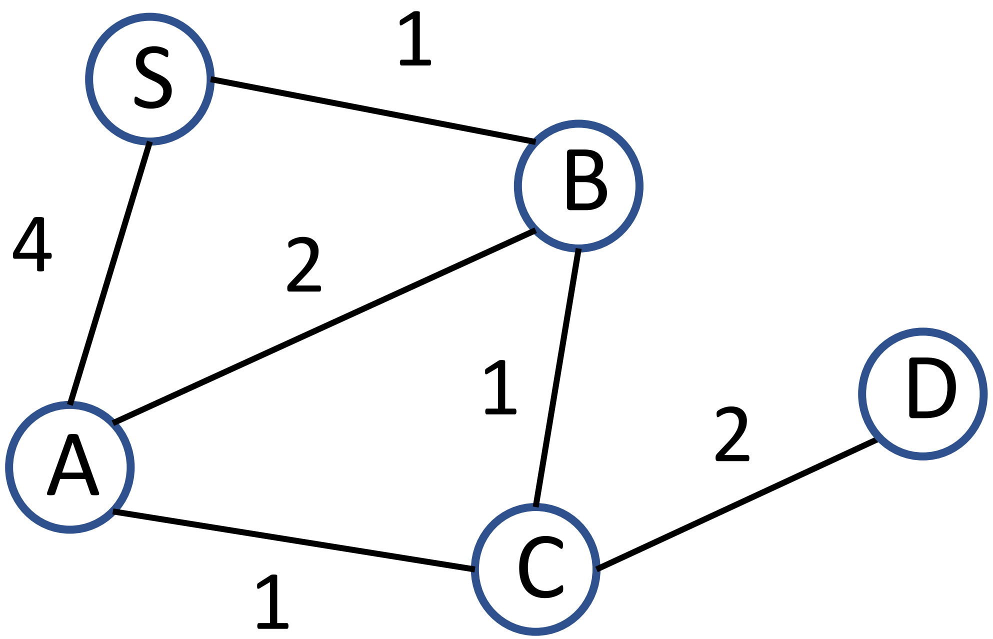
L_0	~~S~~
L_1	~~B~~
L_2	C
L_3	A
L_4	A



	S	A	B	C	D
dist	0		1		

Q2 [Dial's algorithm]

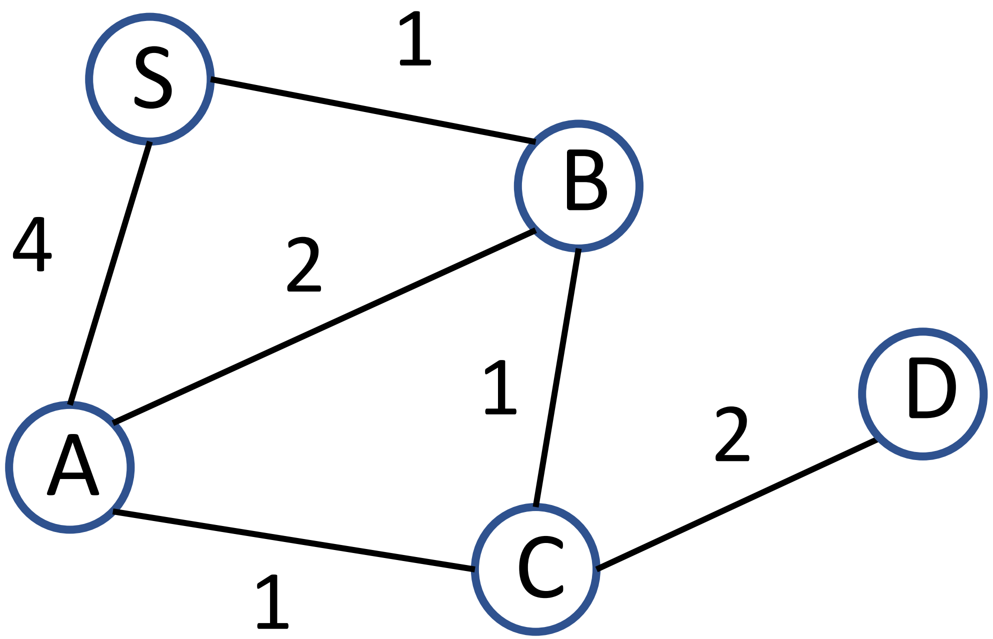
L_0	~~S~~	
L_1	~~B~~	
L_2	~~C~~	
L_3	A	A
L_4	A	D



	S	A	B	C	D
dist	0		1	2	

Q2 [Dial's algorithm]

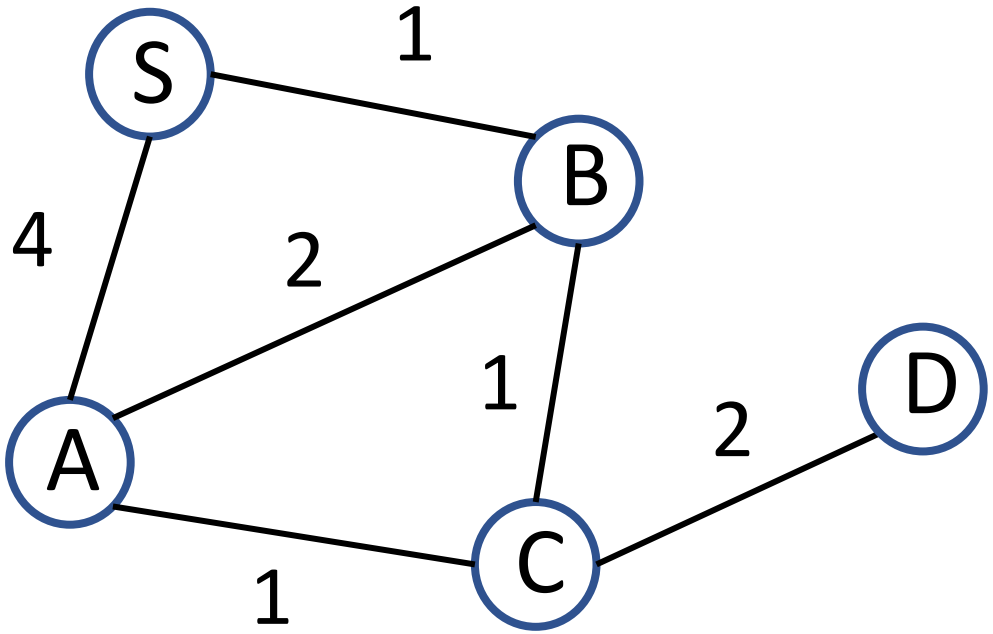
L_0	~~S~~	
L_1	~~B~~	
L_2	~~C~~	
L_3	~~A~~	A
L_4	A	D



	S	A	B	C	D
dist	0	3	1	2	

Q2 [Dial's algorithm]

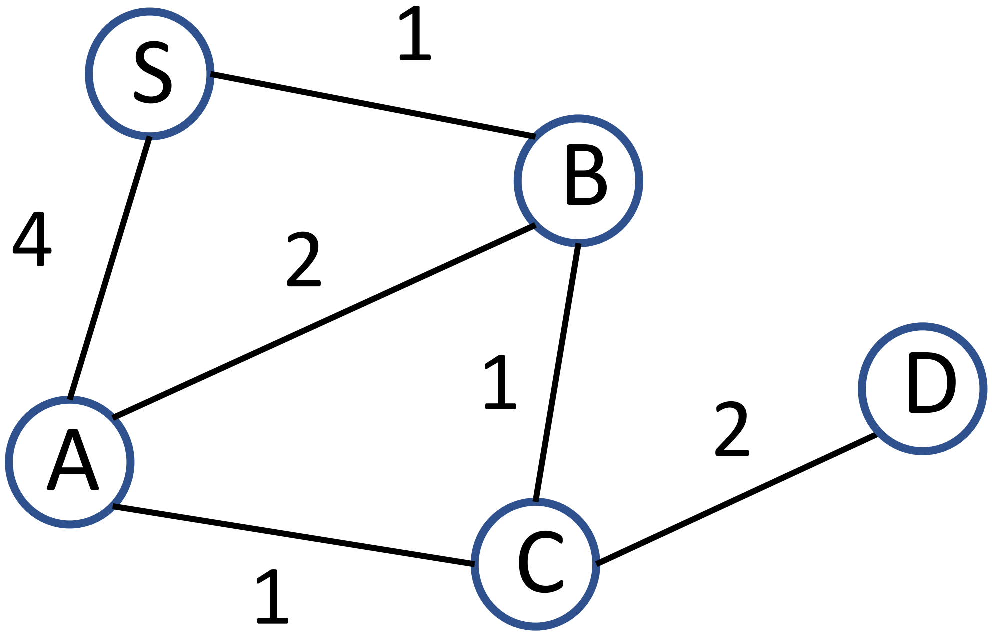
L_0	~~S~~	
L_1	~~B~~	
L_2	~~C~~	
L_3	~~A~~	~~A~~
L_4	A	D



	S	A	B	C	D
dist	0	3	1	2	

Q2 [Dial's algorithm]

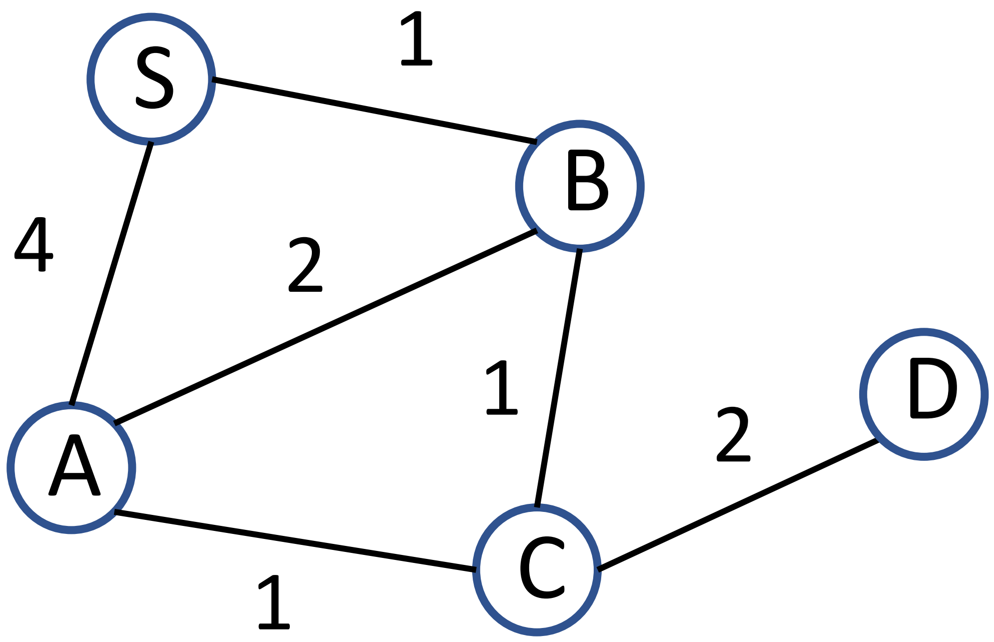
L_0	~~S~~	
L_1	~~B~~	
L_2	~~C~~	
L_3	~~A~~	~~A~~
L_4	~~A~~	D



	S	A	B	C	D
dist	0	3	1	2	

Q2 [Dial's algorithm]

L_0	~~S~~	
L_1	~~B~~	
L_2	~~C~~	
L_3	~~A~~	~~A~~
L_4	~~A~~	~~D~~



	S	A	B	C	D
dist	0	3	1	2	4

Q2 [Dial's algorithm]

Correctness: By induction

- All vertex with shortest distance at most k will be computed correctly

Running Time :

- Each edge is considered O(1) time, and there are O(X) lists ➔ O(X + |E|) time

Q2 [variation]

- Let $G = (V, E)$ be an edge-weighted graph
- Edge weight are integers from $\{ 1, 2, \ldots, W \}$

How to solve SSSP in $O(|E| \log W)$ time?

Key Idea: Use a balanced BST
to maintain non-empty lists

Q2 [variation]

- When we are processing L_k, only lists up to L_{k+W} can be non-empty (i.e., at most W lists)
  ➔ Relax an edge takes O(log W) time to find and update the corresponding list
- After processing the whole list L_k, we can find the next non-empty list in O(log W) time

Q2 [variation]

Running time :

- O(|E|) relax ➔ O(|E| log W) time
- O(|E|) lists can be non-empty ➔

  O(|E| log W) time to find next non-empty list

Total time : O(|E| log W)

Q3

- Let G = (V, E) be an edge-weighted graph
- Suppose G has no negative-weight cycle

If every shortest path from source s
takes at most m edges (but m is not known)
can we run Bellman-Ford faster?

Q3 [solution]

- Simple idea :

$\boxed{\text{Run Bellman-Ford until no update in a certain round of RelaxAll}}$

➔ # rounds $\leq$ m + 1 (why?)

➔ Total time: O(m |E|)