ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2021

Homework 9

Due date: 23:59, December 5, 2021

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL Name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
5. When submitting, match your solutions to the according problem numbers correctly.
6. No late submission will be accepted.
7. Violations to any of above may result in zero score.

1: (2'+2'+2') Multiple Choices

Each question has **one or more** correct answer(s). Please answer the following questions **according to the definition specified in the lecture slides**.

Note: Write down your answers in the table below.

Question 1	Question 2	Question 3
ABC	BC	ABCD

Question 1. *Which of the following statements about **topological sort** is/are true?*

(A) Any sub-graph of a DAG has a topological sorting.

(B) Any directed tree has a topological sorting.

(C) Implementation of topological sort requires $O(|V|)$ extra space.

(D) Since we have to scan all vertices to find those with zero in-degree in each iteration, the run time of topological sort is $\Omega(|V|^2)$.

Question 2. *Which of the following statements about **Dijkstra's algorithm** is/are true?*

(A) Dijkstra's algorithm can find the shortest path in any DAG.

(B) If we use Dijkstra's algorithm, whether the graph is directed or undirected does not matter.

(C) If we implement Dijkstra's algorithm with a binary min-heap, we may change keys of internal nodes in the heap.

(D) We prefer Dijkstra's algorithm with binary heap implementation to the naive adjacency matrix implementation in a dense graph where $|E| = \Theta(|V|^2)$.

Question 3. *Which of the following statements about **Bellman-Ford's algorithm** is/are true?*

(A) Bellman-Ford's algorithm can find the shortest path for negative-weighted directed graphs without negative cycles while Dijkstra's algorithm may fail.

(B) The run time of Bellman-Ford's algorithm is $O(|V||E|)$, which is more time-consuming than Dijkstra's algorithm with heap implementation.

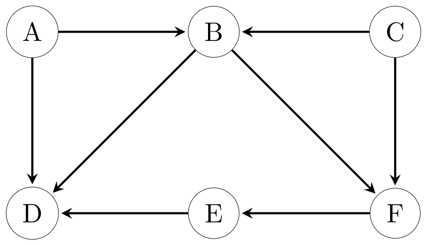
(C) Topological sort can be extended to determine whether a graph has a cycle while Bellman-Ford's algorithm can be extended to determine whether a graph has a negative cycle.

(D) Topological sort can find the critical path in a DAG while Bellman-Ford's algorithm can find the single-source shortest path in a DAG.

2: (5'+3') Topological Sort

Given the following DAG, run topological sort with a queue. Write down the vertex you select and update the in-degree `ind[i]` of all vertices in each iteration.

Note: When pushing several vertices into the queue at the same time, push them alphabetically. You are NOT required to show your queue at each step.



	vertex	`ind[A]`	`ind[B]`	`ind[C]`	`ind[D]`	`ind[E]`	`ind[F]`
initial	/	0	2	0	3	1	2
iteration 1	A	0	1	0	2	1	2
iteration 2	C	0	0	0	2	1	1
iteration 3	B	0	0	0	1	1	0
iteration 4	F	0	0	0	1	0	0
iteration 5	E	0	0	0	0	0	0
iteration 6	D	0	0	0	0	0	0

Question 4. *Fill in the table above. What is the topological sorting that you obtain?*

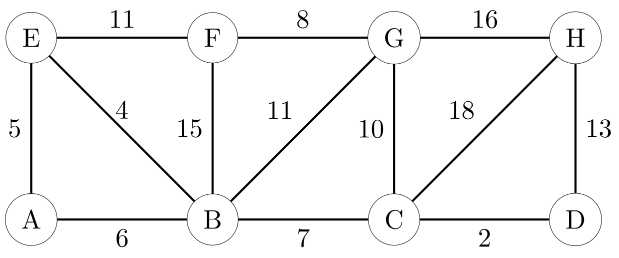
ACBFED

Question 5. *How many different topological sortings does this graph have? Write them down.*

2: ACBFED, CABFED

3: (5'+3') Dijkstra's Algorithm

Given the following weighted graph, run Dijkstra's algorithm by considering A as the source vertex. Write down the vertex you select and update current distance `dis[i]` of all vertices in each iteration.



Question 6. *Fill in the table below.*

	vertex	dis[A]	dis[B]	dis[C]	dis[D]	dis[E]	dis[F]	dis[G]	dis[H]
initial	/	0	∞	∞	∞	∞	∞	∞	∞
iteration 1	A	0	6	∞	∞	5	∞	∞	∞
iteration 2	E	0	6	∞	∞	5	16	∞	∞
iteration 3	B	0	6	13	∞	5	16	17	∞
iteration 4	C	0	6	13	15	5	16	17	31
iteration 5	D	0	6	13	15	5	16	17	28
iteration 6	F	0	6	13	15	5	16	17	28
iteration 7	G	0	6	13	15	5	16	17	28
iteration 8	H	0	6	13	15	5	16	17	28

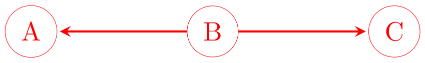
Question 7. *Fill in the table below for **time complexity analysis** of Dijkstra's algorithm with different implementation. Write down your answer as simplified as possible.*
Hint: Total Run Time = `pop()` Time $\times |V|$ + `relax()` Time $\times |E|$ for Dijkstra's algorithm with adjacency list implementation, where `pop()` is the operation to find the unvisited vertex with least `dis[i]` in one iteration and `relax()` is the operation to update `dis[v]` according to `dis[u]` and weight of edge (u, v). Heap implementation is based on adjacency list implementation.

Implementation	pop()	relax()	total
Adjacency Matrix	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
Adjacency List	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
Binary Heap	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|E\| \log \|V\|)$
Fibonacci Heap	$O(\log \|V\|)$	$O(1)$	$O(\|V\| \log \|V\| + \|E\|)$

4: (2'+3'+3')Providing Counterexamples

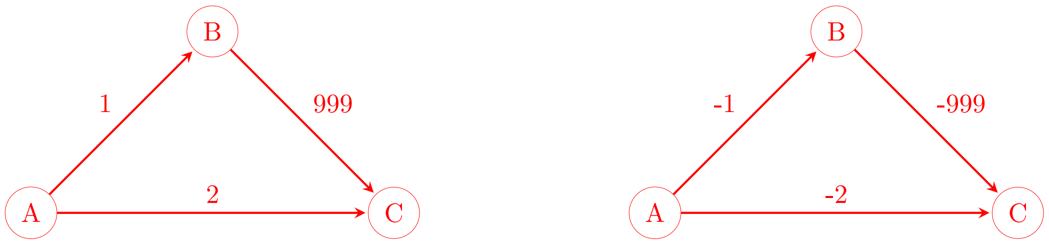
For each of the following statements, provide counterexamples by **drawing a graph and briefly explaining it** to illustrate that these three statements are incorrect.

Question 8. *Each DAG with $|V|$ vertices and $|V|-1$ edges has its unique topological sorting.*



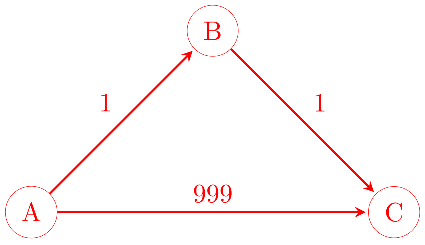
BAC and BCA are both topological sorting of the graph above.

Question 9. *Since we can find the* ***maximum spanning tree*** *in a graph by negating all edge weights by multiplying -1 and then running Prim's algorithm, similarly we can find the* ***longest path*** *in a positive-weighted graph by negating all weights and running Dijkstra's algorithm from every source node.*



By negating all weights in the left graph, it will become the right graph. We will visit A, C and then B by running Dijkstra's algorithm in the right graph. Hence `dis[C]` is fixed to -2 before visiting B so we cannot relax it any more by edge (B, C) with weight -999. Finally we will obtain a "longest" path of weight 2 $(A \to C)$ instead of 1000 $(A \to B \to C)$ in fact.

Question 10. *Assume that we implement Dijkstra's algorithm with a binary heap as the priority queue in a positive-weighted graph, then we can do the following operation: when the terminal vertex is pushed into the priority queue, we can stop the algorithm and return the correct shortest path from the source vertex to the terminal vertex, instead of keeping running the algorithm until popping the terminal vertex from the heap.*



Start with vertex A, and then we will push C into heap with `dis[C]`$=$ 999. Since the terminal vertex is pushed, we stop the algorithm and return the "shortest" path of weight 999 $(A \to C)$ instead of 2 $(A \to B \to C)$ in fact. (Only after popping B, updating `dis[C]` to 2 and then popping the terminal vertex C, will we obtain the correct shortest path.)