CS101 Algorithms and Data Structures
Fall 2022
Homework 9

Due date: 23:59, November 27th, 2022

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 problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (?? points) Multiple Choices

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

(a)	(b)	(c)	(d)
ACD	AC	ABD	ABCD

(a) (3') Which of the following statements about **topological sort** is/are true?

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

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

C. Any sub-graph of a DAG has a topological sorting.

D. Any directed tree has a topological sorting.

(b) (3') Which of the following statements about **Dijkstra's algorithm** is/are true?

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

B. Dijkstra's algorithm can find the shortest path in any DAG.

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

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

(c) (3') Which of the following statements about **Dijkstra's algorithm** is/are true?

A. Dijkstra's algorithm with a binary heap could run in time $O((|V| + |E|) \log |V|)$.

B. Dijkstra's algorithm with a Fibonacci heap could run in time $O(|E| + |V| \log |V|)$.

C. Dijkstra's algorithm on a tree with a binary heap could run in time $O(|E|+|V|)$.

D. Dijkstra's algorithm with an adjacency list could run in time $O(|V|^2 + |E|)$.

(d) (3') Which of the following statements about **Bellman-Ford's algorithm** is/are true?

A. 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.

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

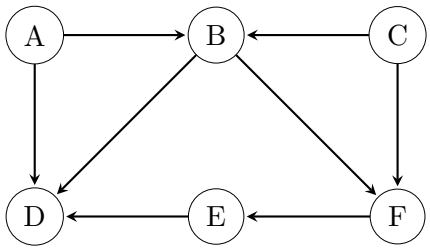
C. Topological sort can find the critical path in a DAG while Bellman-Ford's algorithm can find the single-source shortest path in a DAG.

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

2. (?? points) 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

(a) (5') I. Fill in the table above. II. What is the topological sorting that you obtain?

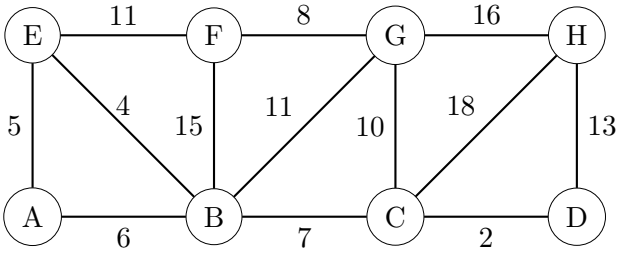
Solution:
A, C, B, F, E, D

(b) (3') How many different topological sortings does this graph have? Write them down.

Solution:
2
A, C, B, F, E, D
C, A, B, F, E, D

3. (?? points) Dijkstra

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



Fill in the table below.

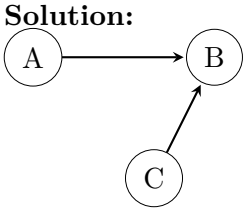
	vertex	`dis[A]`	`dis[B]`	`dis[C]`	`dis[D]`	`dis[E]`	`dis[F]`	`dis[G]`	`dis[H]`
initial	/	0	∞	∞	∞	∞	∞	∞	∞
iteration 1	A	0	6	∞	∞	5	∞	∞	∞
iteration 2	E	0	6	∞	∞	5	16	∞	∞
iteration 3	B	0	6	13	∞	5	16	17	∞
iteration 4	C	0	6	13	15	5	16	17	31
iteration 5	D	0	6	13	15	5	16	17	28
iteration 6	F	0	6	13	15	5	16	17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

4. (?? points) 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.

(a) (3') Each DAG with $|V|$ vertices and $|V|-1$ edges has its unique topological sorting.

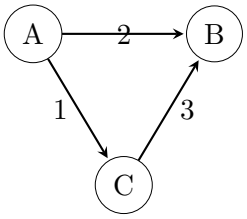
Solution:



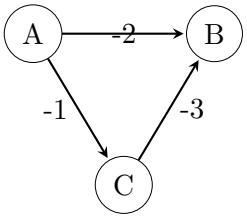
it has two topological sorting results:
A, C, B
C, A, B

(b) (3') Since we can find a **maximum spanning tree** in a graph by multiplying all edge weights by -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.

Solution:



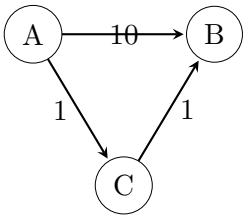
if we set the source node be A, the terminal node be B, then after nagating all weights, the graph become



And when running Dijkstra's algorithm on the new graph from source node A, end at terminal node B.
after the first time, node B has the weight -2, which is less than C's -1, so we will mark node B as visited, and the final weight from A to B become -2, however, the shortest path on the new graph from A to B should be -4. so it is incorrect.
i.e. after negating all weights and running Dijkstra's algorithm, we get the longest path 2, however, it should be 4.

(c) (3') 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.

Solution:



if we set the source node be A, the terminal node be B, then
if stop the algorithm when pushed the terminal node into the priority queue, then the result is :
the path is $A \to B$, and the total weight is 10.
but if stop the algorithm when popping the terminal node into the priority queue, then the result is :
the path is $A \to C \to B$, and the total weight is 2. which is better than the previous one.