PSO 11

Dijkstra, Bellman-Ford, Union Find

https://juhttps://justin-zhang.com/teaching/CS251

Midterm

Time to lock in

Question 1

(Dijkstra's algorithm)

1. Give a simple example of a directed graph with negative-weighted edges for which Dijkstra's algorithm produces an incorrect answer.

2. Given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, but all other edge weights are non-negative, and there are no negative-weighted cycles. Can the Dijkstra's algorithm correctly find all the shortest paths from s in this graph?

3. Your classmate claims that Dijkstra's algorithm relaxes the edges of every shortest path in the graph in the order in which they appear on the path. Show that him/her is mistaken by constructing a directed graph for which the Dijkstra's algorithm could relax the edges of a shortest path out of order.

Question 2

(Bellman-Ford algorithm)

For the Bellman-Ford algorithm, explain

1. why it only requires $|V| - 1$ passes?
2. why the last pass ($|V| - 1$) through the edges will determine if there are negative weight cycles or not?

Question 3

(Union find)

1. Suppose that we implemented Union-Find data structure with quick-union. The current state of the data-structure is defined in the following table.

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

List each disjoint set along with its canonical element (Hint: It may help to draw the corresponding trees).

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union$(6, 5)$. What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

3. Assume that k_1 and k_2 are integers such that $1 \leq k_1 \leq k_2$. Show that

$$\log_2(k_1 + k_2) \geq \max\{1 + \log_2 k_1, \log_2 k_2\}.$$

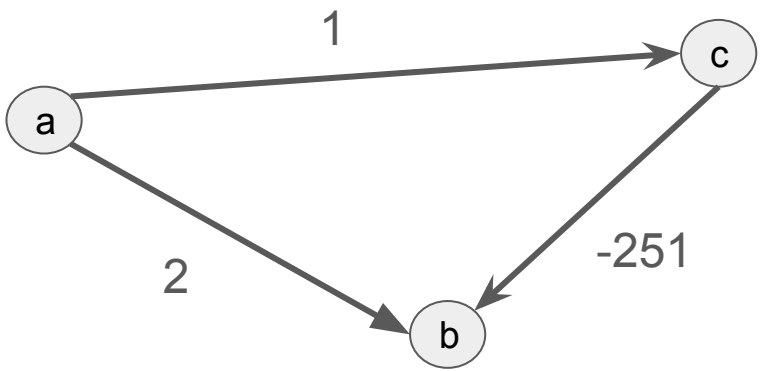
4. (Union–By–Weight) Suppose that we implement our Union–Find data structure with the union-by-weight optimization. Use an inductive argument to show that the maximum height of any tree in the union find data-structure is at most $\log_2 n$.

Question 1

(Dijkstra's algorithm)

1. Give a simple example of a directed graph with negative-weighted edges for which Dijkstra's algorithm produces an incorrect answer.

Note: We do not relax nodes we have already visited in the pQ



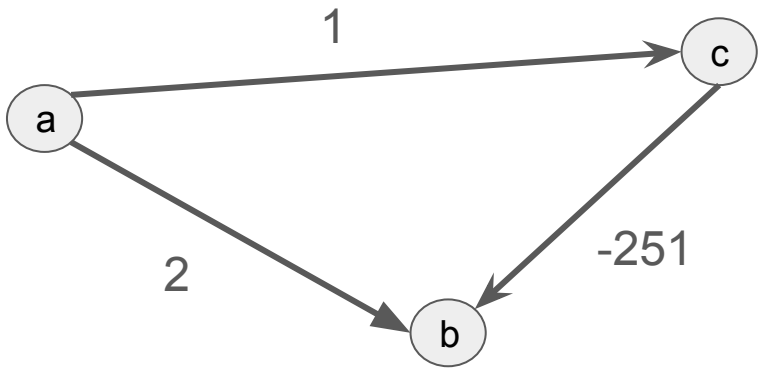
pQ

1.
2.
3.

	A	B	C
dist			
prev			

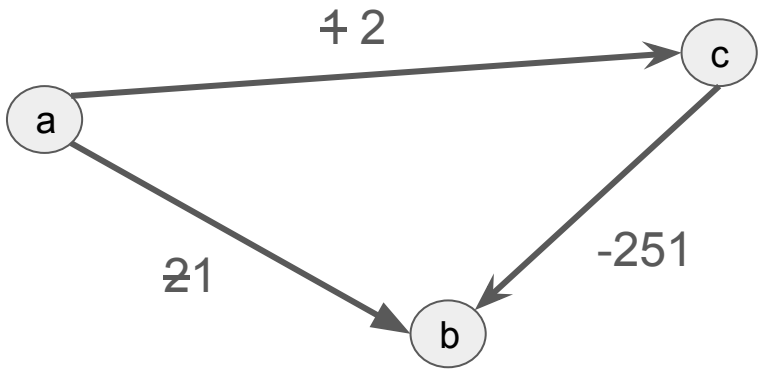
2. Given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, but all other edge weights are non-negative, and there are no negative-weighted cycles. Can the Dijkstra's algorithm correctly find all the shortest paths from s in this graph?

This seems plausible… but the previous example was false



2. Given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, but all other edge weights are non-negative, and there are no negative-weighted cycles. Can the Dijkstra's algorithm correctly find all the shortest paths from s in this graph?

This seems plausible… but the previous example was false



This works! (Exercise)

Question 2

(Bellman-Ford algorithm)

For the Bellman-Ford algorithm, explain

1. why it only requires $|V| - 1$ passes?
2. why the last pass ($|V| - 1$) through the edges will determine if there are negative weight cycles or not?

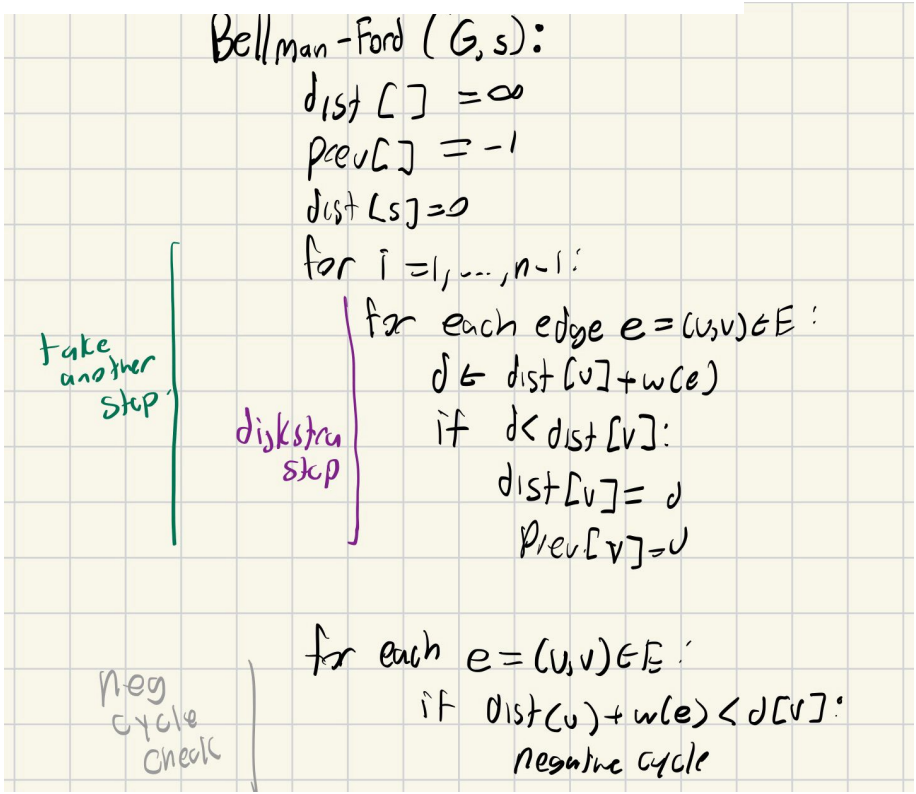
Intuition:

I am running dijkstra |V| - 1 times (w/o priority)

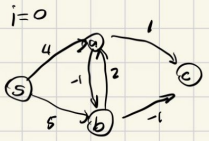
I might miss a shortest path

I can only miss it |V| - 1 times i.e.

my paths “improve” only |V| - 1 times



i=0



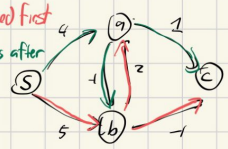
S	A	B	C
0	∞	∞	∞

i=1

I've only looked at each edge once

Looked at red first

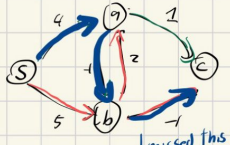
Green edges after



S	A	B	C
0	4	3	4

i=1

I've only looked at each edge once

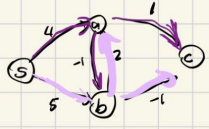


S	A	B	C
0	4	3	4

↑ not shortest yet.

I missed this path going out of vertex a!

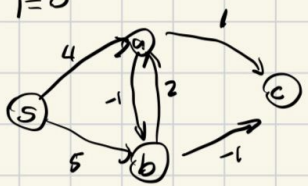
i=2



S	A	B	C
0	4	3	4
			2

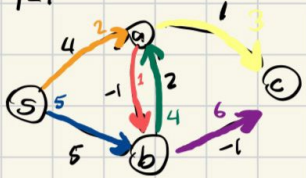
Another edge ordering

i=0



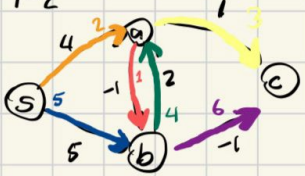
S	A	B	C
0	∞	∞	∞

i=1



S	A	B	C
0	~~∞~~	~~∞~~	~~∞~~
	4	5	11

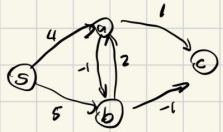
i=2



S	A	B	C
0	4	~~5~~	~~11~~
		3	5
			2

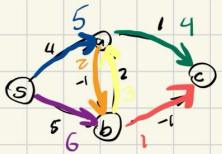
Another edge ordering, taking the full $|V|-1 = 3$ iterations

i=0



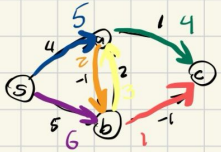
S	A	B	C
0	∞	2	2

i=1



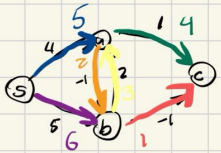
S	A	B	C
0	~~∞~~ 4	~~2~~ 5	2

i=2



S	A	B	C
0	4	~~5~~ 3	~~2~~ 5

i=3



S	A	B	C
0	4	3	~~5~~ -2

Question 3

(Union find)

1. Suppose that we implemented Union-Find data structure with quick-union. The current state of the data-structure is defined in the following table.

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

List each disjoint set along with its canonical element (Hint: It may help to draw the corresponding trees).

What is quick union?

How do the trees look?

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

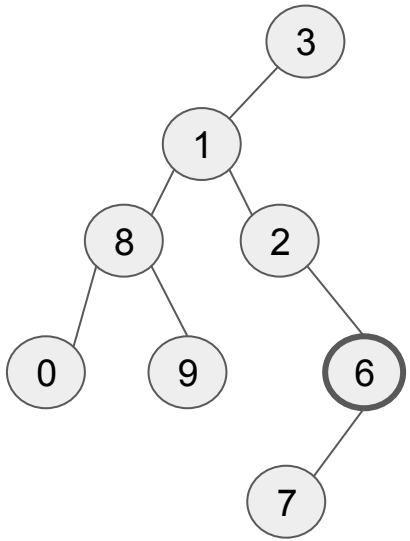
2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

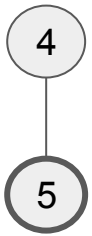
Path compression:

Union by weight:

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

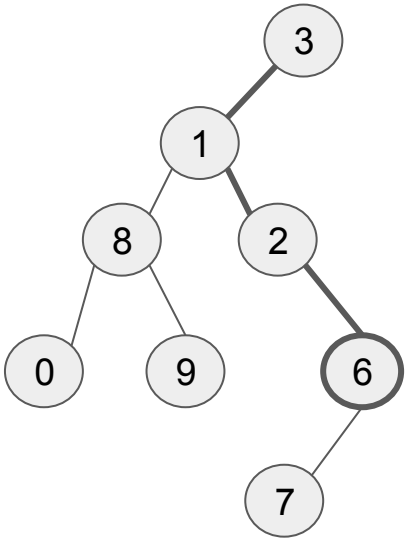




Union(5,6)

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

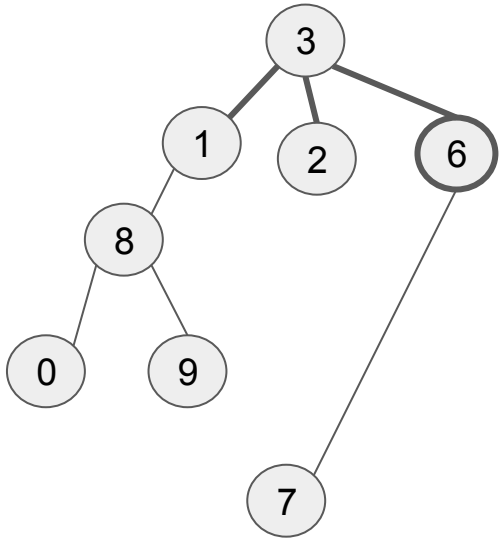
2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)



Union(5,6)
Step 1: find their roots by traversing up the tree

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

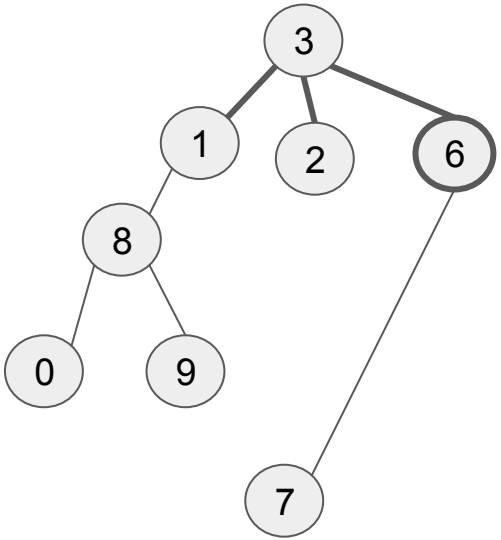
2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)



Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

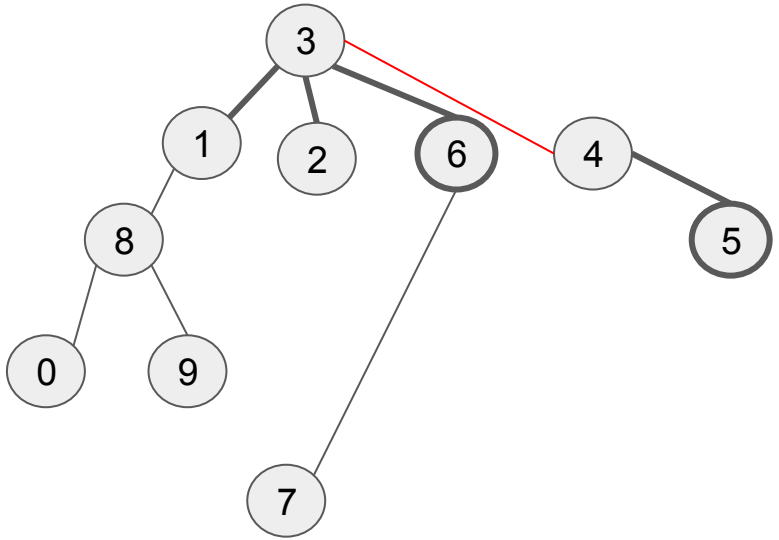


Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

Step 2: connect roots
Union by weight
(minimize suffering!)

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

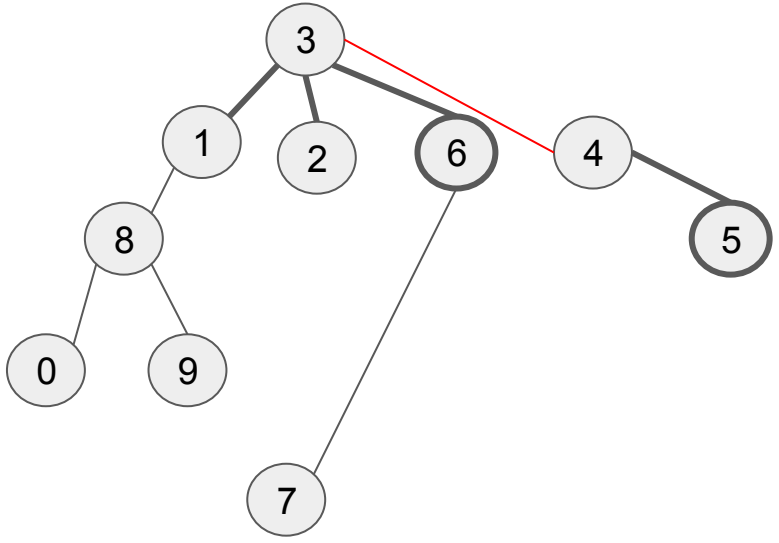


Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

Step 2: connect roots
Union by weight
(minimize suffering!)

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)



Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

Step 2: connect roots
Union by weight
(minimize suffering!)
Step 3: Update the table

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

3. Assume that k_1 and k_2 are integers such that $1 \leq k_1 \leq k_2$. Show that

$$\log_2(k_1 + k_2) \geq \max\{1 + \log_2 k_1, \log_2 k_2\}.$$

$\log(k_1 + k_2) \geq \log k_2$ because…

$\log(k_1 + k_2) \geq 1 + \log k_1$ because…

Hence, we are done.

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

4. (Union–By–Weight) Suppose that we implement our Union–Find data structure with the union-by-weight optimization. Use an inductive argument to show that the maximum height of any tree in the union find data-structure is at most $\log_2 n$.

Recall we know

$$\log_2(k_1 + k_2) \geq \max\{1 + \log_2 k_1, \log_2 k_2\}.$$

BC: n = 1

i	0	1	2	3	4	5	6	7	8	9
Id[i]	8	3	1	3	4	4	2	6	1	8

4. (Union–By–Weight) Suppose that we implement our Union–Find data structure with the union-by-weight optimization. Use an inductive argument to show that the maximum height of any tree in the union find data-structure is at most $\log_2 n$.

Recall we know

$$\log_2(k_1 + k_2) \geq \max\{1 + \log_2 k_1, \log_2 k_2\}.$$

IS: Assume for all n’ < n,