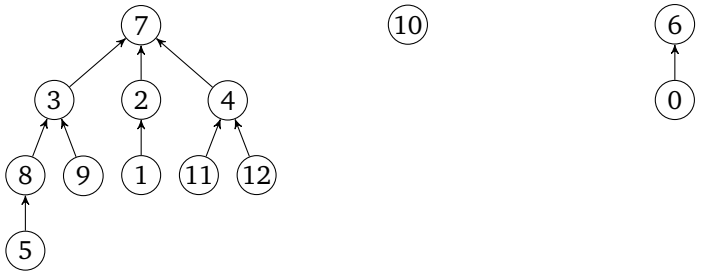
Section 07: Solutions

1. Disjoint Sets: Union and Find

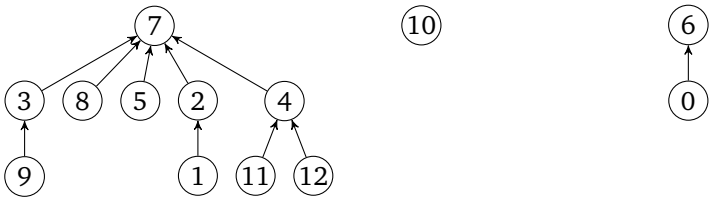
(a) Consider the following trees, which are a part of a disjoint set:



For these problems, use both the **union-by-size** and **path compression** optimizations.

(i) Draw the resulting tree(s) after calling `findSet(5)` on the above. What value does the method return?

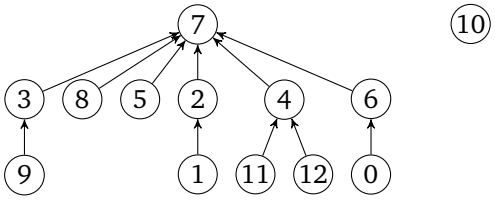
Solution:



The method returns 7, which is the root of the tree.

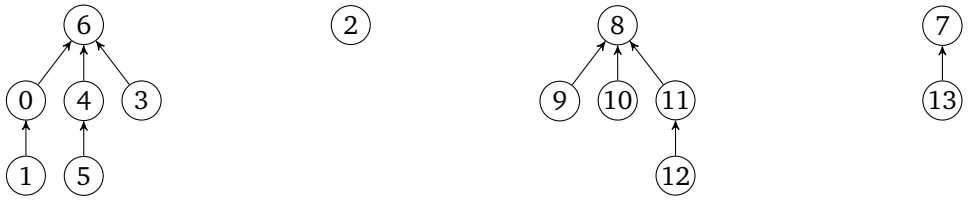
(ii) Draw the final result of calling `union(2,6)` on the result of part (i).

Solution:



We attach the second tree to the first since it has a smaller size.

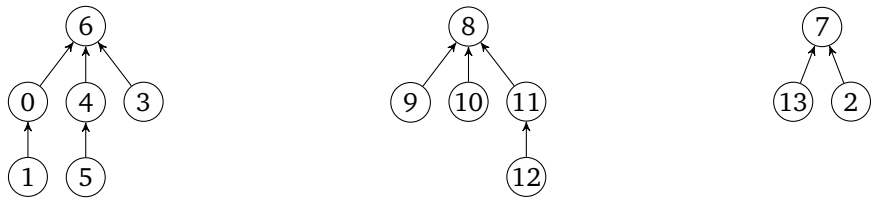
(b) Consider the disjoint-set shown below:



What would be the result of the following calls on union if we add the **union-by-size** and **path compression** optimizations? Draw the forest at each stage.

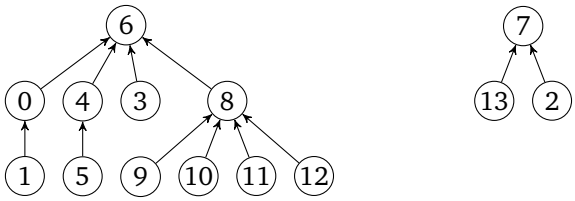
(i) `union(2, 13)`

Solution:



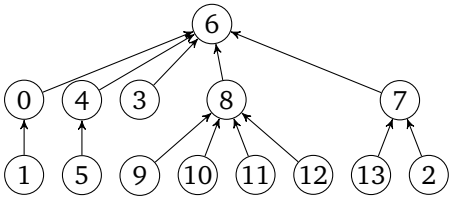
(ii) `union(4, 12)`

Solution:



(iii) `union(2, 8)`

Solution:



2. Graphs: True or False?

Answer each of these true/false questions about minimum spanning trees.

(a) A MST contains a cycle.

Solution:

False. Trees (including minimum spanning trees) never contain cycles.

(b) If we remove an edge from a MST, the resulting subgraph is still a MST.

Solution:

False, the set of edges we chose will no longer connect everything to everything else.

(c) If we add an edge to a MST, the resulting subgraph is still a MST.

Solution:

False, an MST on a graph with n vertices always has $n - 1$ edges.

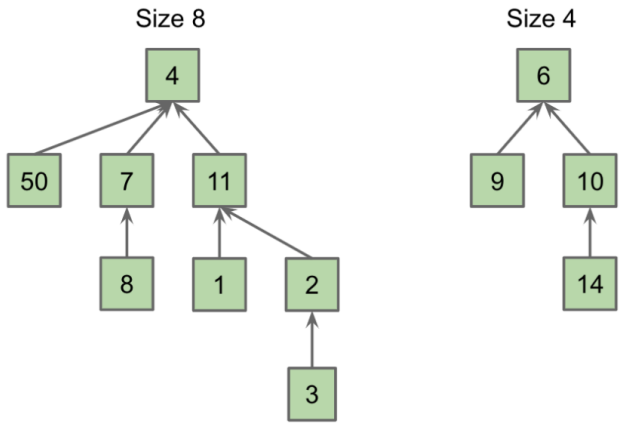
(d) If there are V vertices in a given graph, a MST of that graph contains $|V| - 1$ edges.

Solution:

This is true (assuming the initial graph is connected).

3. Disjoint Sets: Array Representation

Fill in the correct array representation for the following disjoint sets structure given a mapping of items to their indices. Use the table below, and assume we are using the implementation from **lecture**.



Items:	50	4	7	8	9	11	1	2	3	6	10	14
Index:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Value:												

Solution:

Items:	50	4	7	8	9	11	1	2	3	6	10	14
Index:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Value:	1	-8	1	2	9	1	5	5	7	-4	9	10

4. Disjoint Sets: Array Find Set

Using the disjoint sets **at the top of this page**, perform `findSet(2)` with path compression. Give the return value (i.e. the index of the representative), draw the updated array, and draw the resulting disjoint sets.

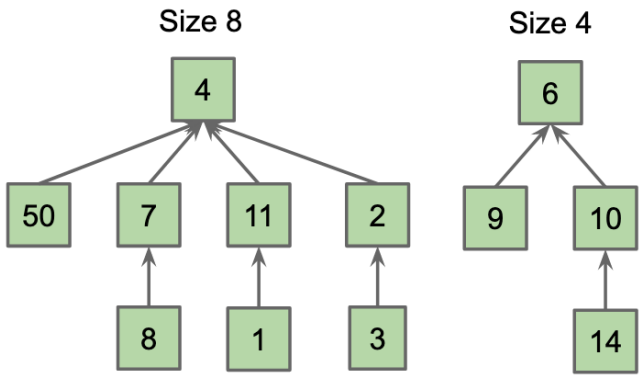
Solution:

Returned Representative Index: 1.

Updated Array:

Items:	50	4	7	8	9	11	1	2	3	6	10	14
Index:	0	1	2	3	4	5	6	7	8	9	10	11
Value:	1	-8	1	2	9	1	5	1	7	-4	9	10

Resulting Disjoint Sets:



5. Disjoint Sets: Array Union

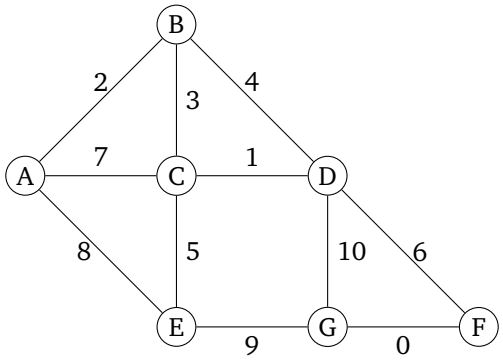
Using the disjoint sets **at the top of this page**, *instead* perform `union(3, 14)`. Use both the path compression and union-by-size optimizations. Draw the resulting array.

Solution:

See Section slides.

6. MSTs: Unique Minimum Spanning Trees

Consider the following graph:



(a) What happens if we run Prim's algorithm starting on node A? What are the final costs and edges selected? Give the set of edges in the resulting MST.

Solution:

Step	Components	Edge
1	{A} {B} {C} {D} {E} {F} {G}	(A,B)
2	{A,B} {C} {D} {E} {F} {G}	(B,C)
3	{A,B,C} {D} {E} {F} {G}	(C,D)
4	{A,B,C,D} {E} {F} {G}	(C,E)
5	{A,B,C,D,E} {F} {G}	(D,F)
6	{A,B,C,D,E,F} {G}	(F,G)

(b) What happens if we run Prim's algorithm starting on node E? What are the final cost and edges selected? Give the set of edges in the resulting MST.

Solution:

Step	Components	Edge
1	{A} {B} {C} {D} {E} {F} {G}	(E,C)
2	{C,E} {A} {B} {D} {F} {G}	(C,D)
3	{C,D,E} {A} {B} {F} {G}	(C,B)
4	{B,C,D,E} {A} {F} {G}	(B,A)
5	{A,B,C,D,E} {F} {G}	(D,F)
6	{A,B,C,D,E,F} {G}	(F,G)

(c) What happens if we run Prim's algorithm starting on *any* node? What are the final costs and edges selected? Give the set of edges in the resulting MST.

Solution:

The answer would be the same as the one we get above, since for each node, we always choose the smallest-weight edge that links to it.

(d) What happens if we run Kruskal's algorithm? Give the set of edges in the resulting MST.

Solution:

We'll use this table to keep track of components and edges we processed. The edges are listed in an order sorted by weight.

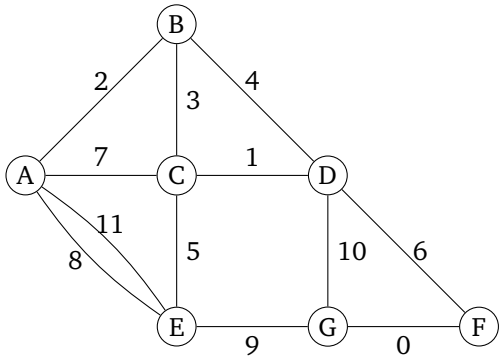
Step	Components	Edge	Include?
1		(F,G)	
2		(C,D)	
3		(A,B)	
4		(B,C)	
5		(B,D)	
6		(C,E)	
7		(D,F)	
8		(A,C)	
9		(A,E)	
10		(E,G)	
11		(D,G)	

After executing Kruskal's algorithm on the above graph, we get

Step	Components	Edge	Include?
1	{A} {B} {C} {D} {E} {F} {G}	(F,G)	Yes
2	{A} {B} {C} {D} {E} {F,G}	(C,D)	Yes
3	{A} {B} {C,D} {E} {F,G}	(A,B)	Yes
4	{A,B} {C,D} {E} {F,G}	(B,C)	Yes
5	{A,B,C,D} {E} {F,G}	(B,D)	No
6	{A,B,C,D} {E} {F,G}	(C,E)	Yes
7	{A,B,C,D,E} {F,G}	(D,F)	Yes
8	{A,B,C,D,E,F,G}	(A,C)	No
9	{A,B,C,D,E,F,G}	(A,E)	No
10	{A,B,C,D,E,F,G}	(E,G)	No
11	{A,B,C,D,E,F,G}	(D,G)	No

The resulting MST is a set of all edges marked as *Include* in the above table.

(e) Suppose we modify the graph above and add a heavier parallel edge between A and E, which would result in the graph shown below. Would your answers for above subparts (a, b, c, and d) be the same for this following graph as well?

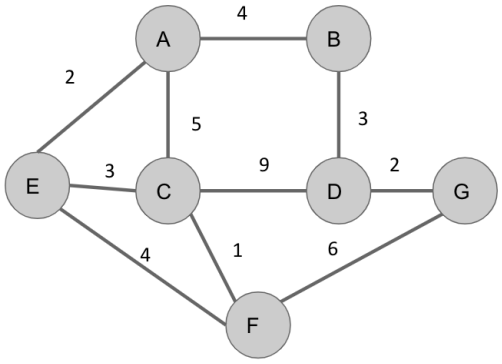


Solution:

The steps are exactly the same, since we don't consider the heavier edge when there are parallel edges. The reason is that the heavier edge would never be considered as the best edge when there is a lighter one (of weight 8) that can be added to the graph instead.

7. MSTs: Basic Kruskal's

What happens if we run Kruskal's algorithm on this graph? Give the set of edges in the resulting MST.



Solution:

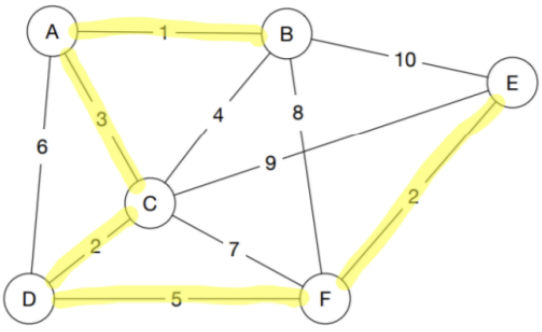
See Section slides.

8. MSTs: Kruskal's Algorithm

Answer these questions about Kruskal's algorithm.

(a) Execute Kruskal's algorithm on the following graph. Fill the table.

Solution:



Step	Components	Edge	Include?
1	{A} {B} {C} {D} {E} {F}	A, B	Yes
2	{A, B} {C} {D} {E} {F}	D, B, C	Yes
3	{A, B} {C, D} {E} {F}	E, F	Yes
4	{A, B} {C, D} {E, F}	A, C	Yes
5	{A, B, C, D} {E, F}	B, C	No
6	— " —	D, F	Yes
7	{A, B, C, D, E, F}	A, D	No
8	— " —	C, F	No
9	— " —	B, F	No
10	— " —	C, E	No
11	— " —	E, B	NO

(b) In this graph there are 6 vertices and 11 edges, and the for loop in the code for Kruskal's runs 11 times, a few more times after the MST is found. How would you optimize the pseudocode so the for loop terminates early, as soon as a valid MST is found.

Solution:

Use a counter to keep track of the number of edges added. When the number of edges reaches $|V| - 1$, exit the loop.