CS2040 Lab 9
Graph Traversal

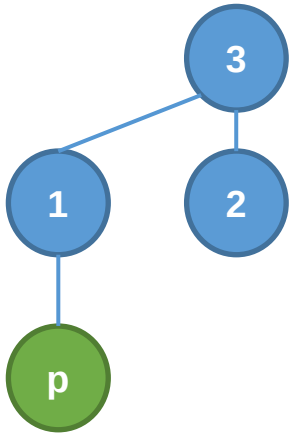
Take Home Assignment 3 – Almost Union Find

- To address operation 3 (ie. outputting the number/sum of items), each set will need to keep track of the total number/sum of items
- This can be done by creating 2 additional arrays (for number, and sum of items), and updating these arrays when performing operations 1 (union) and 2 (move)
- Similar to UFDS tutorial (Tutorial 6), the number/sum arrays will only need to be updated at the representative item; other items may store incorrect/invalid results, but this is fine as these values will not be used in any further computations

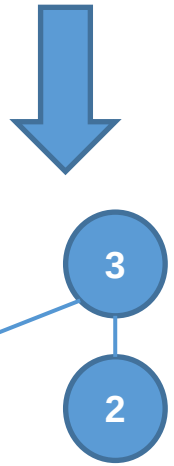
Take Home Assignment 3 – Almost Union Find

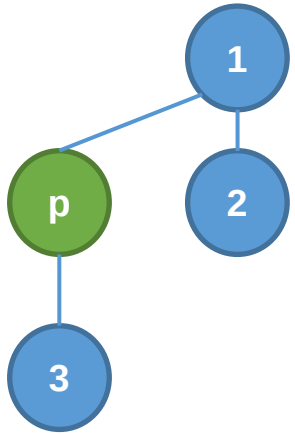
- To address operation 2 (move), consider this to be split into 2 separate steps: removing p from the old set, and adding p to the new set
- Attempting to actually remove p from the old set may be more difficult than expected however (too many cases to work with; examples on the next slide)

Take Home Assignment 3 – Almost Union Find



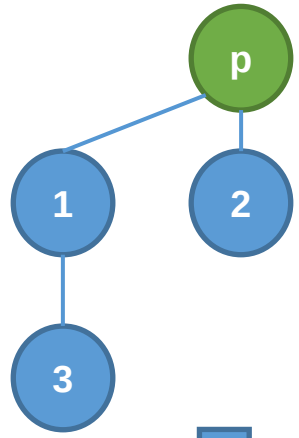
no children





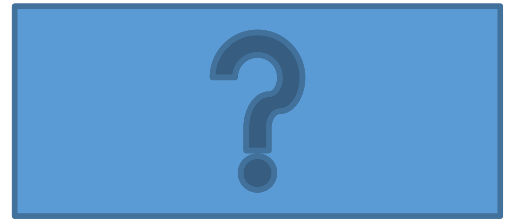
with children, but not representative item

unclear final result



representative item

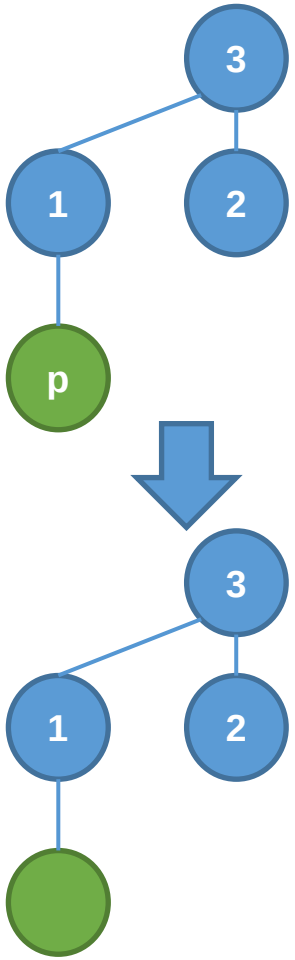
unclear final result



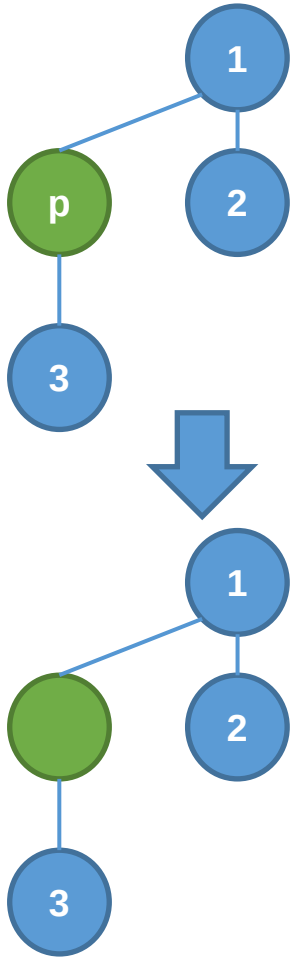
Take Home Assignment 3 – Almost Union Find

- Additionally, there is no easy way to find children of a particular item in standard UFDS
  - Updating all children to point to a different item would therefore take too long, and likely TLE
- Therefore, instead of actually trying to remove the item from the set and updating the parent(s) accordingly, we can leave the node inside, but avoid associating that node with the item p

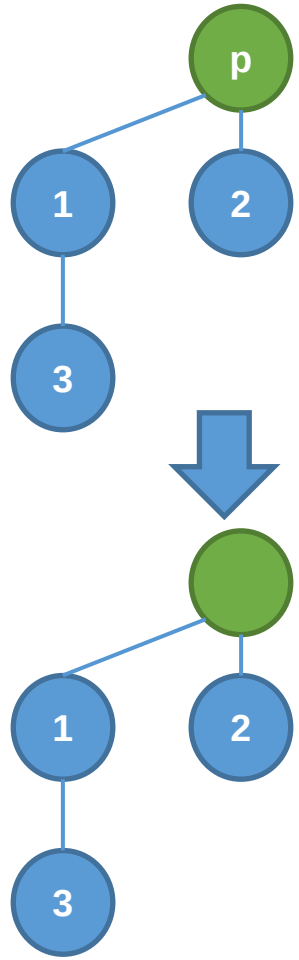
Take Home Assignment 3 – Almost Union Find



no children



with children, but not representative item



representative item

Take Home Assignment 3 – Almost Union Find

- Such an approach would require separating out nodes and items (which typically mean the same thing in a standard UFDS)
- Initially, each item is represented by a node with the same index
- When we move an item, we leave the node representing that item where it is, and instead assign that item to a node in the other set
  - The representative item of the old set should still be updated, to remove 1 from the number of items in the set, and p from the sum of items in the set

Take Home Assignment 3 – Almost Union Find

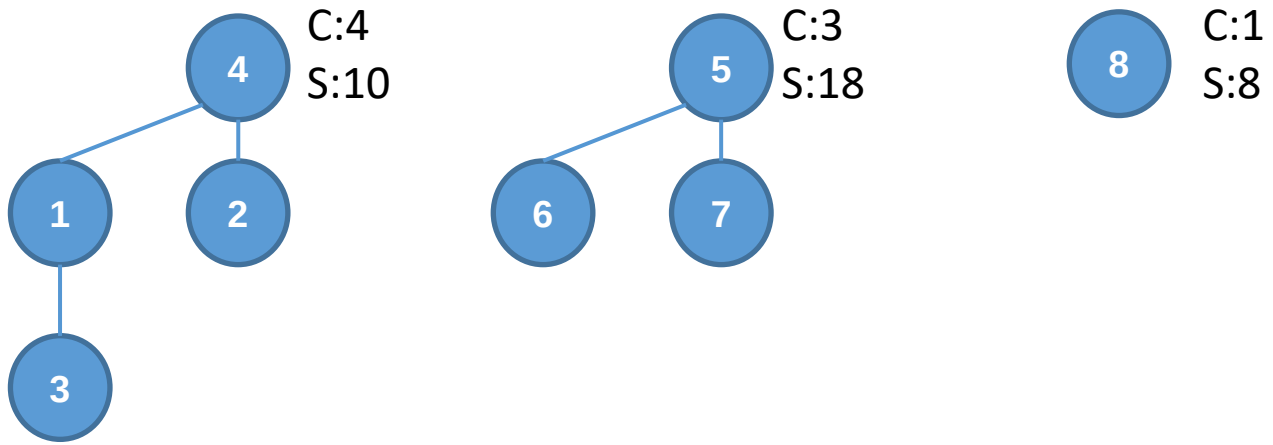
- Idea behind separating items and nodes is similar to Q2 of Tutorial 6
- Warlords (items) are initially represented by a node with the same number
- As warlords get conquered, the nodes representing each item may change
  - Warlords that are conquered are now represented by an invalid node (value of -1)
  - Warlords that are victorious may have their node changed to the "representative node" of the set

Take Home Assignment 3 – Almost Union Find

- Each item would therefore need to keep track of which node it is actually using
- All operations must then be modified to use the node representing that item (and not use the item number directly)

Take Home Assignment 3 – Almost Union Find

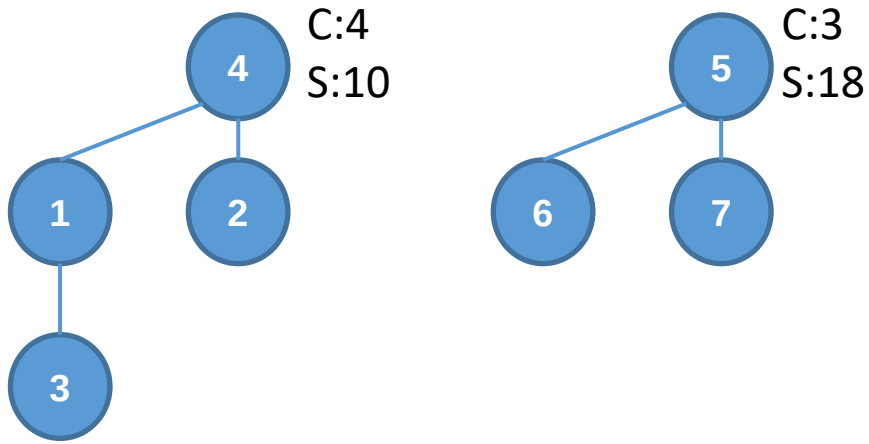
- As an example, suppose we have reached the following resulting UFDS for n = 8 via only operation 1 (the union operation, so no moves have occurred yet)
- All numbers shown here are the node numbers



Take Home Assignment 3 – Almost Union Find

- Additionally, a separate array needs to be maintained to map item number to node number
- Without any move operations, currently all of them have item == node

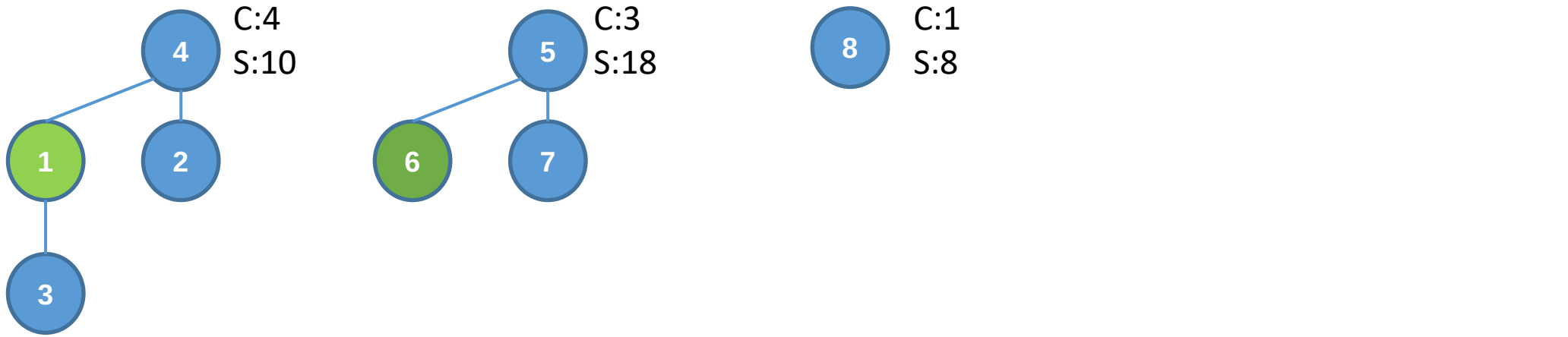
Item	1	2	3	4	5	6	7	8
Node	1	2	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- Suppose we now move(1, 6)
- We need to refer to the nodes representing these 2 items, which is currently still (1, 6)

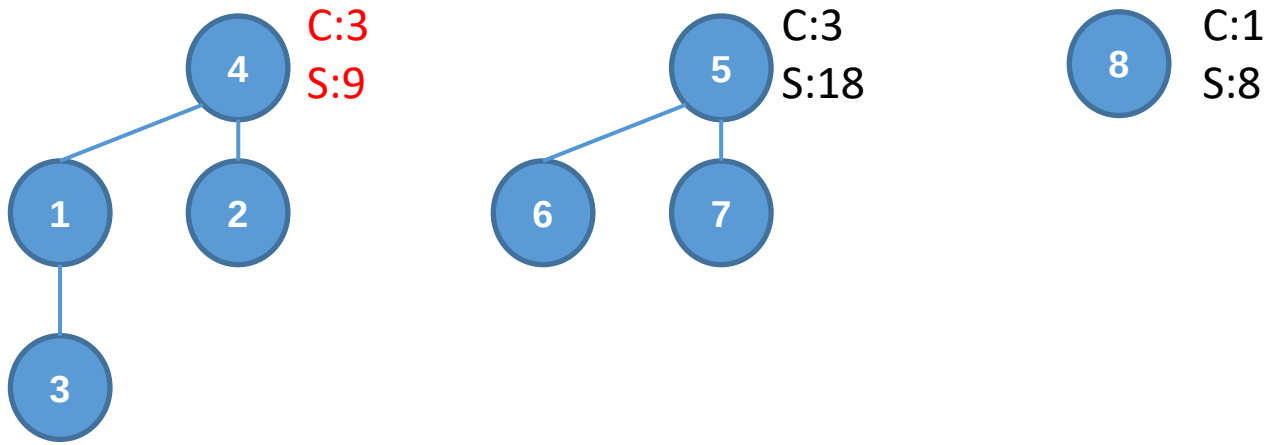
Item	1	2	3	4	5	6	7	8
Node	1	2	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- Since they are in different sets, we will need to move item 1 to the set containing item 6
- We first remove item 1 from the old set by updating the count/sum values

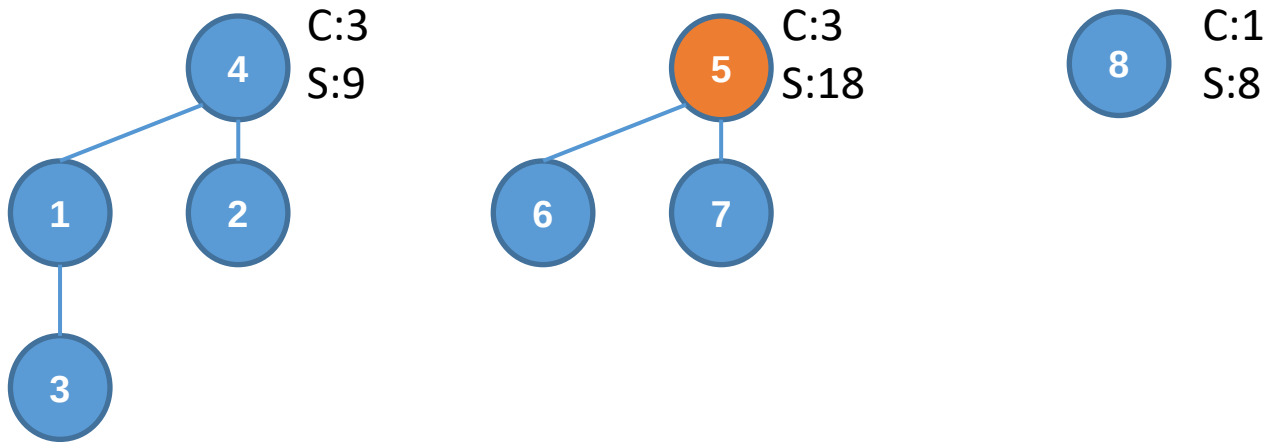
Item	1	2	3	4	5	6	7	8
Node	1	2	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- We then point item 1 to node 5 (the representative node of the set containing item 6)
- In theory, pointing item 1 to any node in the set containing item 6 is fine, but for efficiency reasons we'll use the representative node

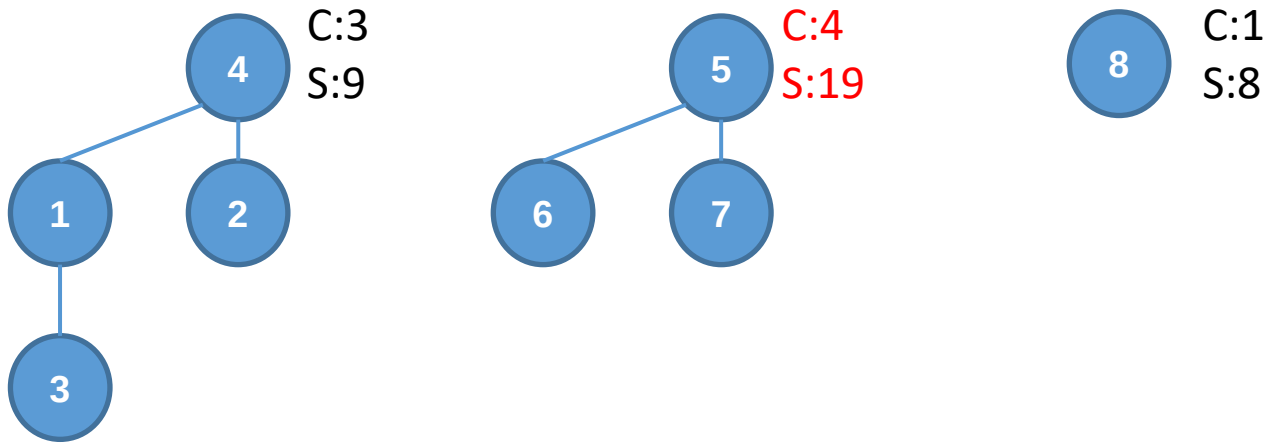
Item	1	2	3	4	5	6	7	8
Node	5	2	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- The count/sum values of the representative node of the new set must be updated as well

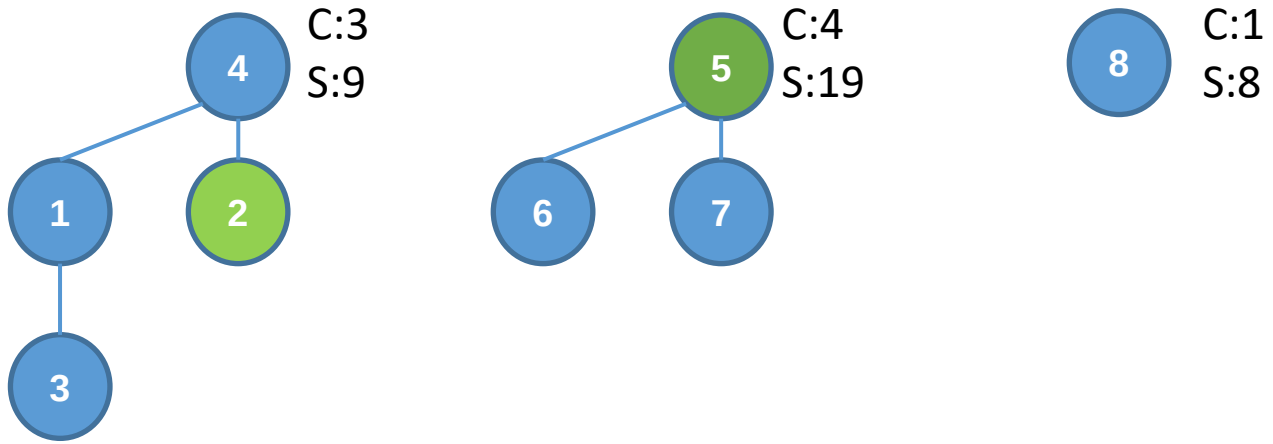
Item	1	2	3	4	5	6	7	8
Node	5	2	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- We will now perform move(2, 1)
- Note that even though the nodes 2 and 1 are in the same set, items 2 and 1 are actually in different sets (node 2, and node 5)

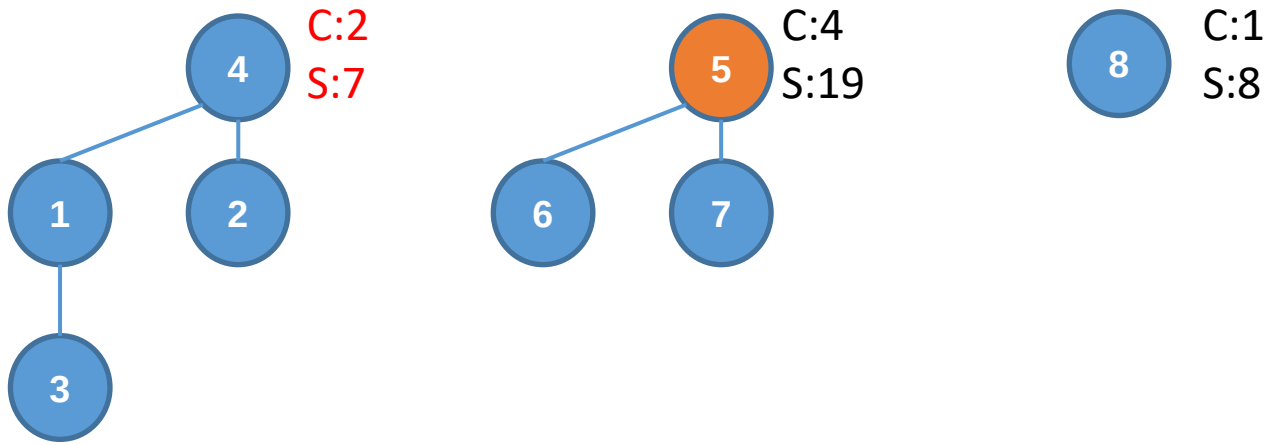
Item	1	2	3	4	5	6	7	8
Node	5	2	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- Once again, we have to update C/S values of the old set
- And point item 2 to the representative node of the set containing item 1

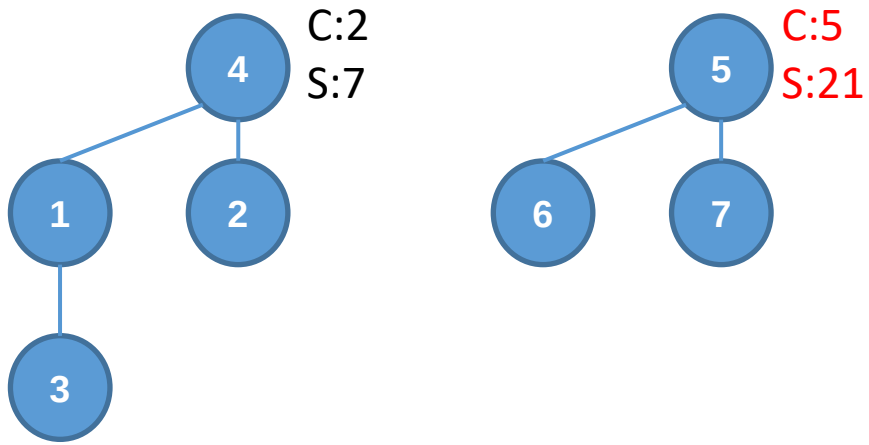
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- Update the C/S values of the new set

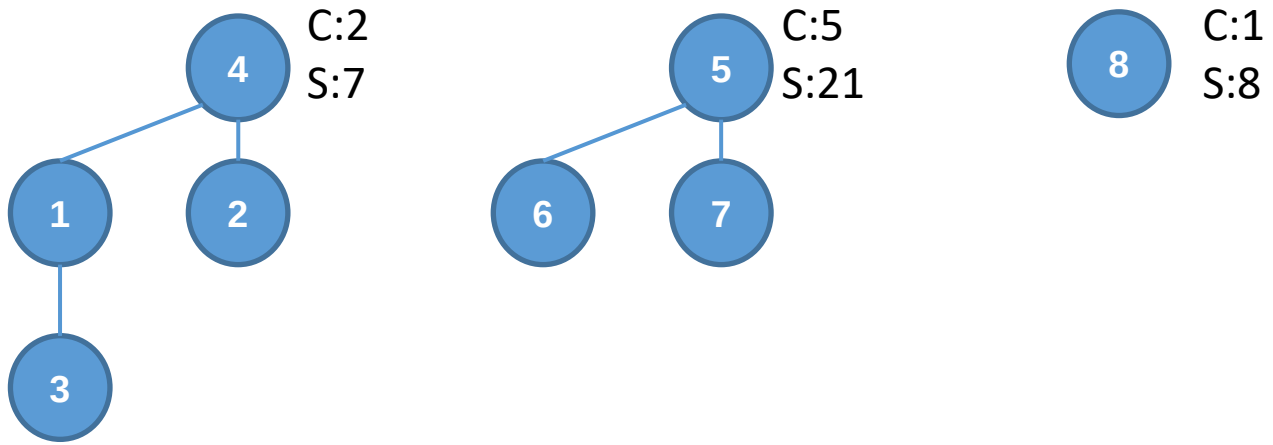
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- Suppose we now do "3 3" (query the set containing item 3)

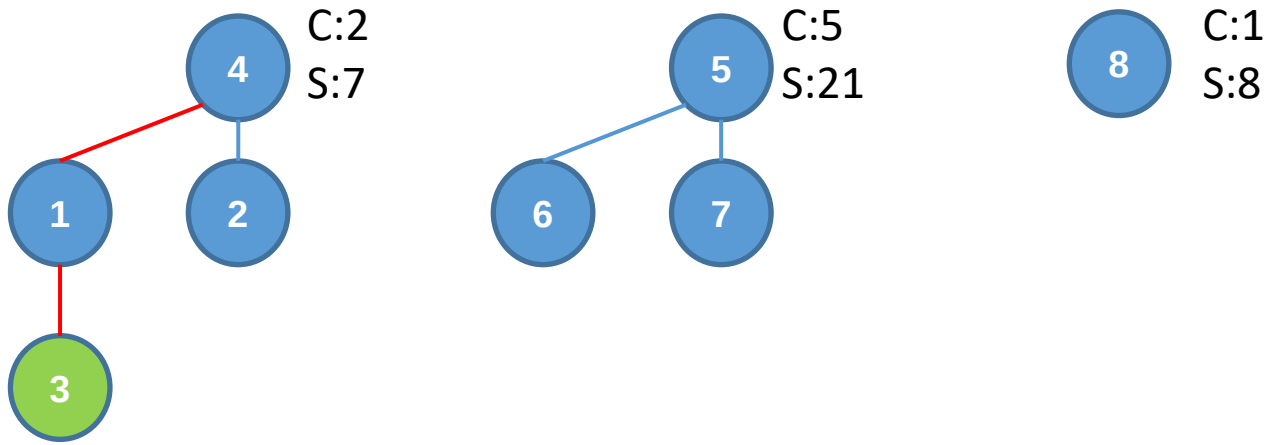
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- From the array, item 3 is mapped to node 3, which has a parent node 1
- Notice that by not modifying node 1, node 3 is still able to correctly determine that its representative node is node 4

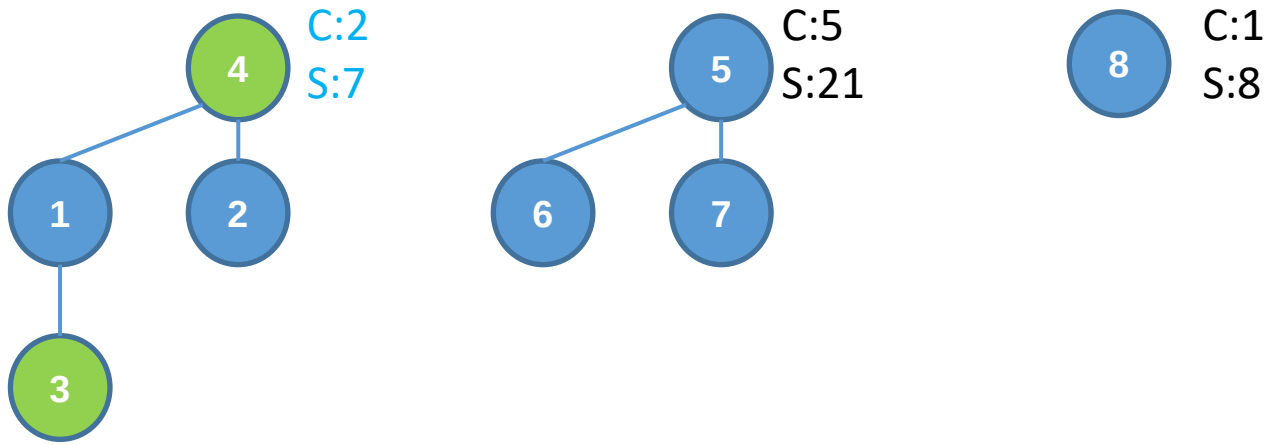
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- Output the C/S values of the representative node 4, which is "2 7"
- This is consistent with our array, which indicates that only items 3 and 4 are in this set

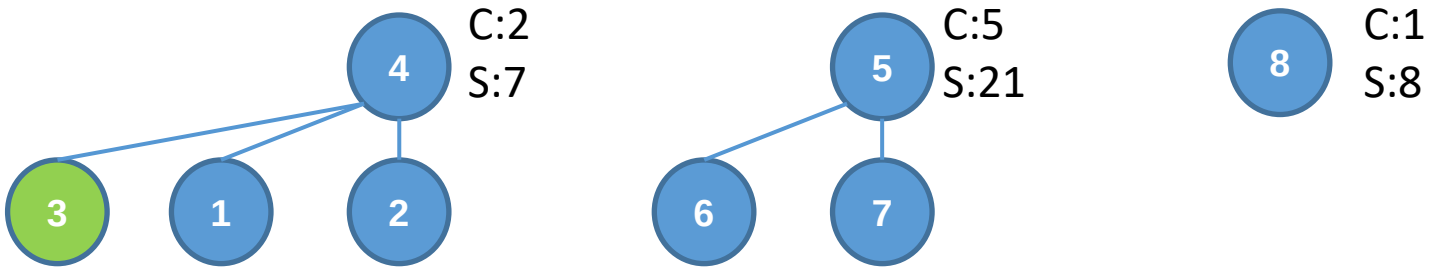
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- Node 3 will also be shifted up as well if performing path compression, as a result of an implicit call to findSet()

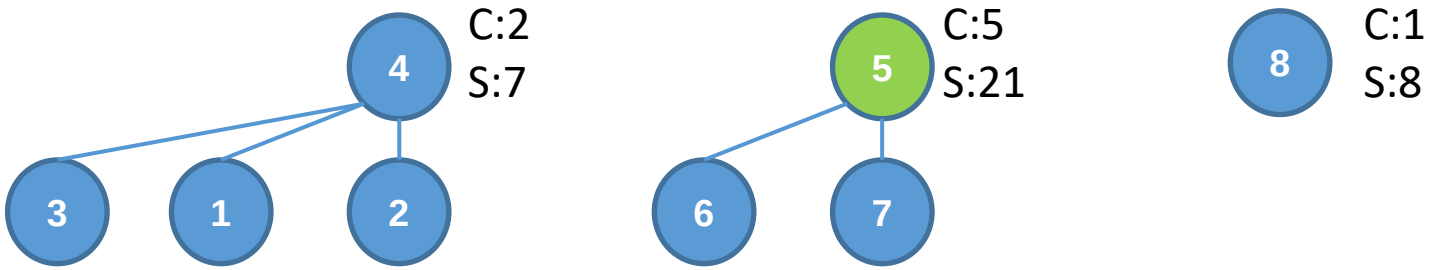
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- If we run "3 1" now (query the set containing item 1)
- Notice that item 1 is represented by node 5

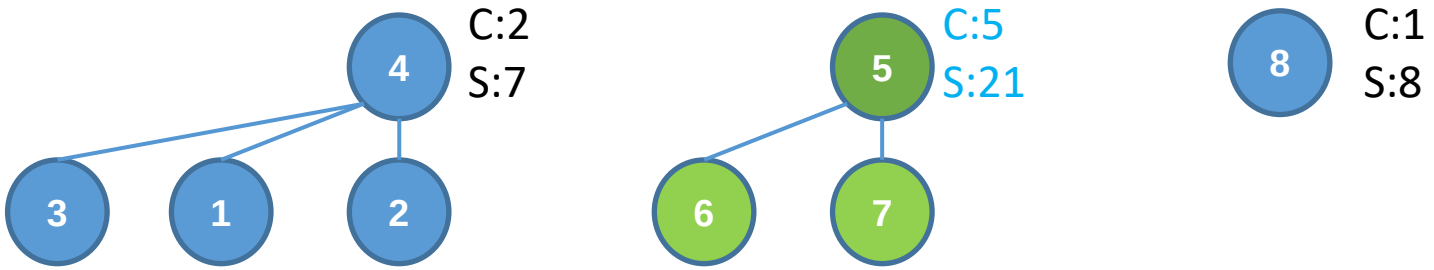
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- The representative node of the set is node 5, which has C/S values "5 21"
- This is consistent with our array, which indicates that items 1, 2, 5, 6 and 7 are in this set

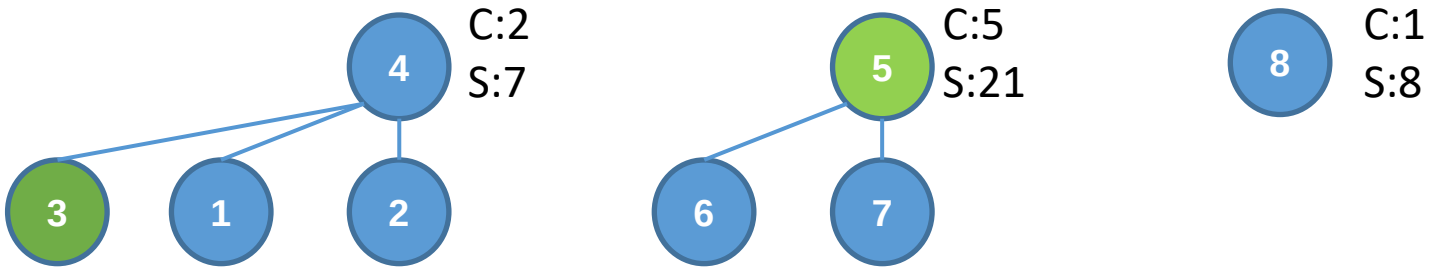
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- We now attempt to perform move(5, 3)
- These are mapped to the nodes 3 and 5 respectively, which are in different sets

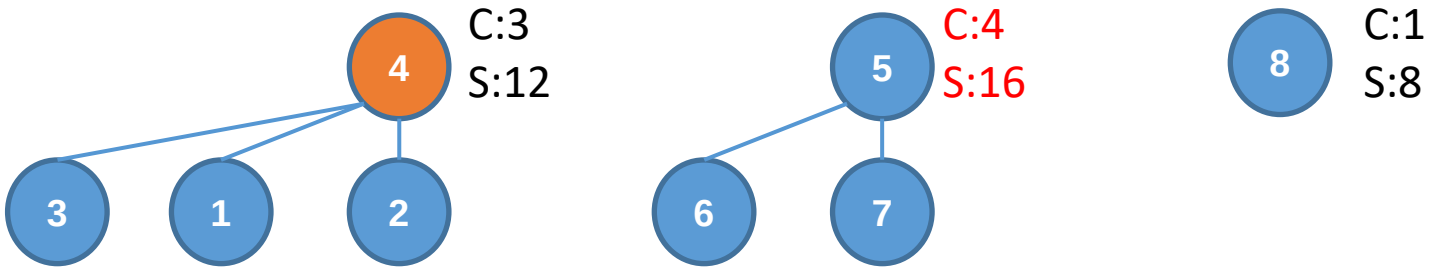
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	5	6	7	8



Take Home Assignment 3 – Almost Union Find

- We update the C/S values of the old and new set, and have item 5 point to node 4 (representative node of item 3)

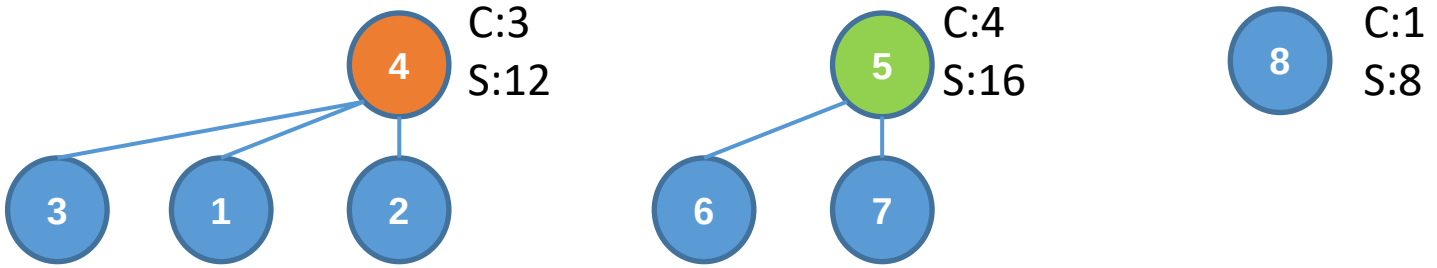
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	4	6	7	8



Take Home Assignment 3 – Almost Union Find

- Notice that despite moving item 5 to a different set, items 1 and 2 (which also point to node 5 before the operation) have not been moved

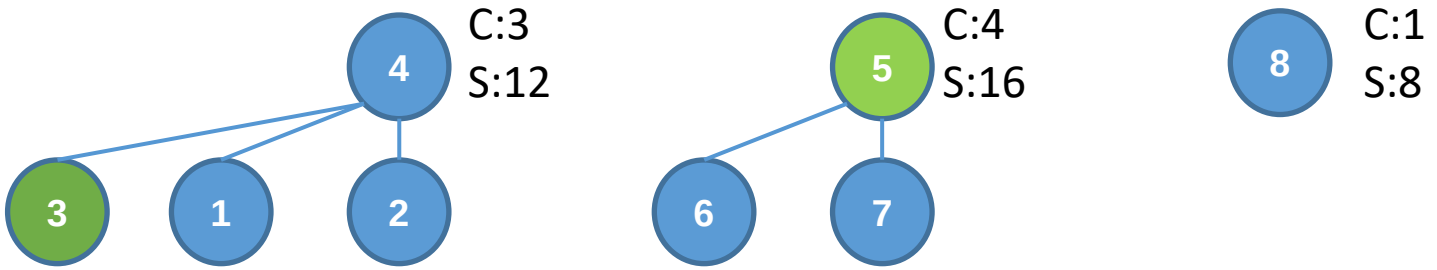
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	4	6	7	8



Take Home Assignment 3 – Almost Union Find

- Finally, suppose we union (1, 3)
- These items are mapped to nodes 5 and 3 respectively, which are in different sets, with representative nodes 5 and 4

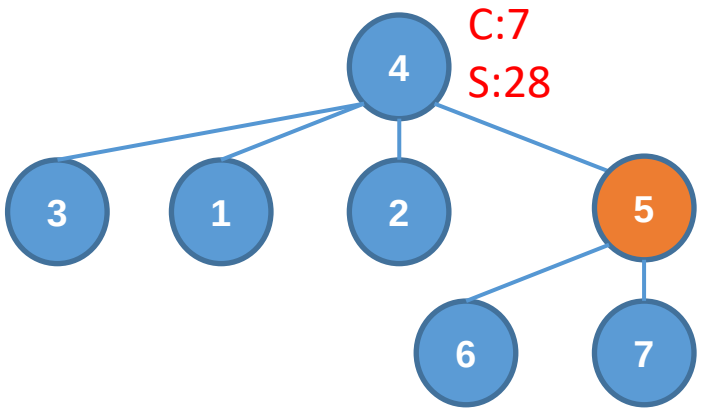
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	4	6	7	8



Take Home Assignment 3 – Almost Union Find

- For union operations, there is no need to modify the item -> node mapping
- After performing the union, update C/S values (as usual)

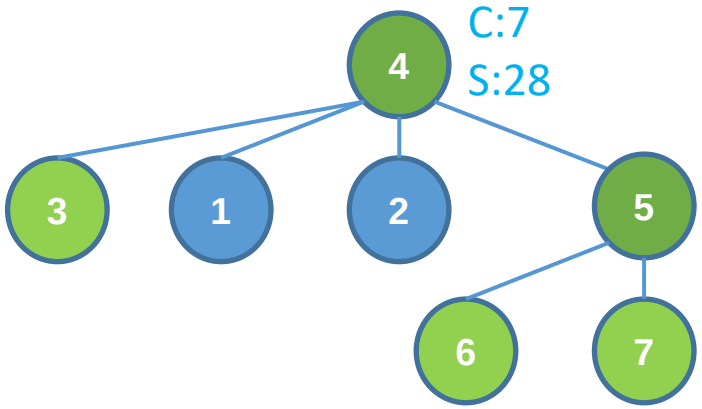
Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	4	6	7	8



Take Home Assignment 3 – Almost Union Find

- Querying any of the items from 1 to 7 should now correctly give the result "7 28", as stored in the representative node 4 of the set

Item	1	2	3	4	5	6	7	8
Node	5	5	3	4	4	6	7	8



Take Home Assignment 3 – Nicknames

- Write a custom AVL class
- Each node in the AVL needs to store a name in lexicographical order
- When we find names matching a particular nickname (suppose the query is for "koel"), we can answer this query by finding how many names lie in the range:
  - ["koel", "koelzzzzzz"] (ie. just append 'z's to the back until the length is 10 or more); or
  - ["koel", "koelzzzzzzz") (ie. just append 'z's to the back until the length is 11 or more)

Take Home Assignment 3 – Nicknames

- When answering a query (again assuming "koel"), we need to find the number of names which are (lexicographically) smaller than "koel", and the number of names smaller than "koelzzzzzzz" (the 11 char version)
- The first query finds all names from ["a", "koel"), and the second query finds all names from ["a", "koelzzzzzzz")
- By subtracting the result of the first query from the second, we can find the number of names in the range ["koel", "koelzzzzzzz")

Take Home Assignment 3 – Nicknames

- To find the number of names smaller than a particular name ("koel" again), notice that all such names must appear to the left of "koel" in the tree
- We can use the rank(x) method covered in Tutorial 7 to find the rank of a particular node
  - This will require each node to store the size of its subtree
  - The number of names smaller than the query x is simply rank(x) - 1
- Note that the string we are attempting to find the rank for may not exist; you can modify rank(x) to account for this, or add, perform the query, and then remove the string after the query

Lab 9 – Graph Traversal

- Two different forms of graph traversal are covered: Breadth First Search (BFS) and Depth First Search (DFS)
  - BFS tends to employ the use of a queue, while DFS uses an explicit stack or implicit stack via recursion
- Pseudocode for both can be found in the lecture slides
- Important note for BFS: mark a vertex as visited when you add it to the queue, as opposed to when you remove the vertex from the queue. Otherwise, you may end up enqueueing the same vertex multiple times, which will result in a slow program

One Day Assignment 8 – Islands

- Count the minimum number of islands present in the graph
- ‘L’ cells – definitely land
- ‘W’ cells – definitely water
- ‘C’ cells – could be either land or water

One Day Assignment 8 – Examples

- CCCCC
- CCCCC
- CCCCC
- CCCCC
- Assume all the clouds are water, answer is 0
- LW
- CC
- WL
- Assume the 2 clouds are land, answer is 1

One Day Assignment 8 – Notes

- In these 2 examples all the clouds are water or all the clouds are land, however clouds need not be assigned graph-wide to all water or all clouds (ie. there can be both clouds that are water, and clouds that are land in the same graph) eg:
- LCCL
- WWWW
- CCCC
- While the graph is not given in the form of any of the DSes we have taught (Adj Matrix, Adj List, Edge List), it may be easier to leave the graph as it is, and modify your algorithm to account for the grid structure, instead of explicitly converting it to a graph DS you have learnt

Extra – Implicit Graphs

Sometimes, you do not always need to explicitly “create” a graph, but we can still make use of graph concepts to solve problems. Try to solve the following problem **without creating a graph**.

I have a n x n grid, and I start at some position (a, b). I can only move in L shape steps, just like a knight in chess.

E.g. If I am at (a, b), I can only move in 1 step to (a + 2, b – 1), (a + 2, b + 1), (a - 2, b – 1), (a - 2, b + 1), (a + 1, b + 2), (a – 1, b + 2), (a + 1, b - 2), (a – 1, b - 2),

List all the positions in the grid that I can reach from my starting position in any order.