CSE 332: Data Abstractions

Review of Post-Midterm Material

December 4, 2014

1. Show how the Parallel Prefix Sum algorithm would work on the following array. Draw the tree and, for each node, show the left, right, sum and fromLeft values. Use a sequential cutoff of 2.

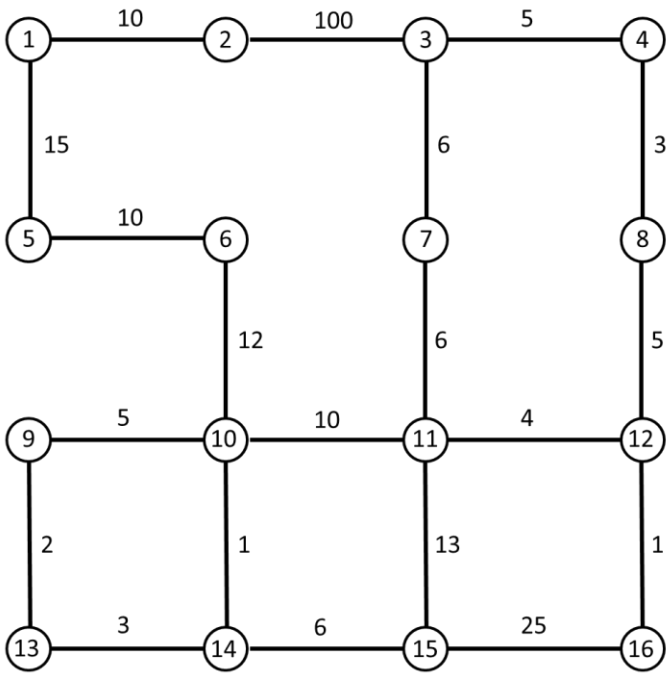
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	7	4	3	9	8	2	1	5	6	3	5	1	8	7	5	1

2. a. What is the problem with using Dijkstra's algorithm for a graph with a negative cost cycle? Give a small example graph and explain what goes wrong.

b. Give a small example graph showing where Dijkstra's algorithm can fail to find the correct least cost path when negative weights are present, even if there is no negative cycle.

c. Will Kruskal's algorithm work with negative edge weights? Why or why not? Do you have to modify the definition of minimum spanning tree given in lecture if there are non-positive edge weights so that it makes sense?

3. Consider the following graph G.



a. Use Kruskal's algorithm to find a minimum spanning tree in G. Each time an edge e is added to T, redraw the current forest T and the full collection of up-trees resulting from adding e to T. Each time an edge e is considered but not added to T, list the find operations that explain why e was not included in T. Use weighted unions (called union-by-size in Section 8.4 of the textbook) and path compression. Whenever you have a tie in the weighted union, that is, you are merging two up-trees that have exactly the same size, break such ties by making the root with the lower vertex number the parent of the root with the higher vertex number. For instance, if you are taking the union of two sets whose up-trees have roots 5 and 11, and each of these up-trees has 2 nodes in it, then break the tie by making 5 the parent of 11. What is the cost of the minimum spanning tree that you found?

b. Use Dijkstra's algorithm to find the least cost path (the path itself, not just its cost) from vertex 1 to every other vertex. Each time a vertex is removed from the heap, do the following: redraw the vertices (but not the edges) of the graph, circle the vertices that still remain in the heap, show the current value of w.dist next to each vertex w, and draw a directed edge **from** w **to** the current value of w.path for each vertex w for which $0 < w.dist < \infty$. You may stop when every vertex w has its final value w.dist.

c. Use breadth-first search to find the shortest path (the path itself, not just its cost) from vertex 1 to every other vertex, ignoring the edge weights. Each time a vertex is removed from the queue, do the following to show the state at the end of this iteration: redraw the vertices (but not the edges) of the graph, show the current value of w.dist next to each vertex w, draw a directed edge **from** w **to** the current value of w.path for each vertex w for which $0 < w.dist < \infty$, and show the current contents of the queue with oldest entries toward the left and newest toward the right. Assume that vertices appear in numerical order on each adjacency list; for example, v_4's adjacency list is given in the order (v_3,v_5,v_7). You may stop when every vertex w has a finite w.dist.

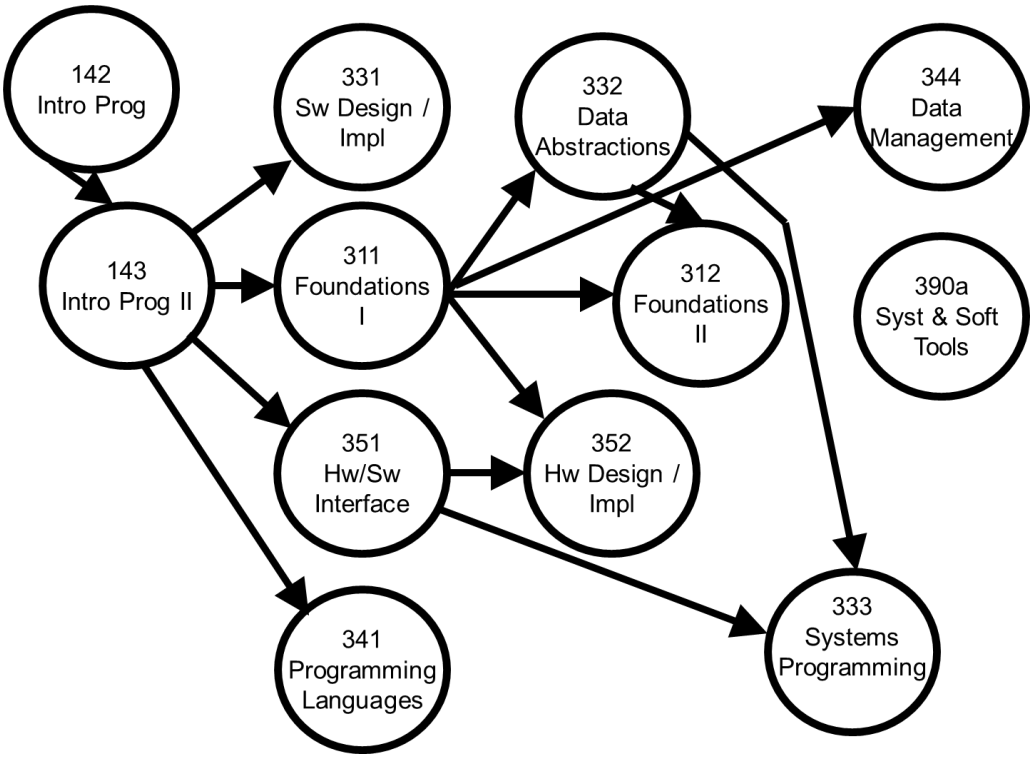
4. Using Java's ForkJoin framework, write a program to verify that an array of integers is a valid binary heap.

5. a. Starting with Figure 8.11 in the textbook, show the resulting collection of up-trees after each of the following operations. Use weighted union and path compression.

union(1,2), union(0,1), find(7), union(1,3), find(5)

b. Starting with Figure 8.15 in the textbook, show the up-tree after find(7), using path compression.

6. Simulate topological sort on the following prerequisite graph to find an order in which you can take these courses so that you will have completed all the prerequisites before taking each given course. Show your work.



7. Show your work in the following operations using the binary tree representation of a heap.

a. Use the linear time buildHeap method to construct a heap from the values: 4, 9, 15, 3, 10, 2, 23, 100, 1, 21.

b. Do 3 deleteMin operations on the resulting heap from part a.

c. Insert the values 92, 40, 3, 6 into the heap resulting from part b.

8. Draw the following graph:

$$V=\{a,b,c,d,e\},$$
$$E=\{(a,b):2,\ (a,c):5,\ (a,d):14,\ (b,c):4,\ (b,d):12,\ (b,e):1,\ (c,d):7,\ (c,e):1,\ (d,e):9\},$$

where (x,y):z represents an undirected edge between x and y with weight z.

a. Find a minimal spanning tree using Kruskal's algorithm. Each time an edge e is added to T, redraw the current forest T and the full collection of up-trees resulting from adding e to T. Each time an edge e is considered but not added to T, list the find operations that explain why e was not included in T. Use weighted unions (called union-by-size in Section 8.4 of the textbook) and path compression. Whenever you have a tie in the weighted union, that is, you are merging two up-trees that have exactly the same size, break such ties by making the root with the vertex letter earlier in the alphabet the parent of the other root. For instance, if you are taking the union of two sets whose up-trees have roots b and d, and each of these up-trees has 2 nodes in it, then break the tie by making b the parent of d. What is the cost of the minimum spanning tree that you found?

b. Use Dijkstra's algorithm to find the least cost path (the path itself, not just its cost) from vertex a to every other vertex. Each time a vertex is removed from the heap, do the following: redraw the vertices (but not the edges) of the graph, circle the vertices that still remain in the heap, show the current value of w.dist next to each vertex w, and draw a directed edge **from** w **to** the current value of w.path for each vertex w for which $0 < w.dist < \infty$. You may stop when every vertex w has its final value w.dist.

9. Intuition suggests that, by using more processors, we can always reduce the running time of a parallelizable algorithm, such as summing an array of integers. Explain at a high level – as though you were explaining to a non-computer-scientist – why we can't sum the entries in an array in constant time, no matter how many processors we throw at the problem.