CS25100 Homework 2: Spring 2017

Due Monday, April 17, 2017, before 11:59 PM. Please edit directly this document to insert your answers. You can use any remaining slip days for his homework. Submit your answers on Vocareum.

1. True/False Questions (18 pts)

1. _F_ The reverse postorder of a digraph's reverse is the same as the postorder of the digraph.
2. _T_ Adding a constant to every edge weight does not change the solution to the single-source shortest-paths problem.
3. _T_ An optimization problem is a good candidate for dynamic programming if the best overall solution can be defined in terms of optimal solutions to subproblems, which are not independent.
4. _F_ If we modify the Kosaraju algorithm to run first depth-first search in the digraph G (instead of the reverse digraph G^R) and the second depth-first search in G^R (instead of G), the algorithm will find the strong components.
5. _F_ If you insert keys in increasing order into a red-black BST, the tree height is monotonically increasing.
6. _T_ A good hash function should be deterministic, i.e., equal keys produce the same hash value.
7. _T_ In the situation where all keys hash to the same index, using hashing with linear probing will result in *O(n)* search time for a random key.
8. _F_ Hashing is preferable to BSTs if you need support for ordered symbol table operations.
9. _T In an adjacency list representation of an undirected graph, *v* is in *w*'s list if and only if *w* is in *v*'s list.
10. _F_ Every directed, acyclic graph has a unique topological ordering.
11. _F_ Preorder traversal is used to topologically sort a directed acyclic graph.
12. _T_ MSD string sort is a good choice of sorting algorithm for random strings, since it examines $N \log_R N$ characters on average (where R is the size of the alphabet).
13. _T_ The shape of a TST is independent of the order of key insertion and deletion, thus there is a unique TST for any given set of keys.
14. _T_ In a priority queue implemented with heaps, *N* insertions and *N* removeMin operations take *O(N log N)*.

15. _F_ An array sorted in decreasing order is a max-oriented heap.

16. _T If a symbol table will not have many insert operations, an ordered array implementation is sufficient.

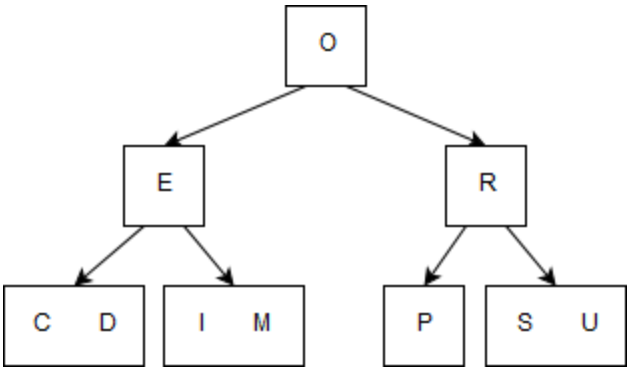
17. _F_ The floor operation returns the smallest key in a symbol table that is greater than or equal to a given key.

18. _T_ The root node in a tree is always an internal node.

2. Questions on Tracing the Operation of Algorithms (30 pts)

1. (4 pts) Draw the 2-3 tree that results when you insert the following keys (in order) into an initially empty tree:

   P U R D U E C O M P S C I



2. (5 pts) Give the contents of the hash table that results when you insert the following keys into an initially empty table of $M = 5$ lists, using separate chaining with unordered lists. Use the hash function $11k \bmod M$ to transform the k-th letter of the alphabet into a table index, *e.g.*, $hash(I) = hash(9) = 99 \% 5 = 4$. Use the conventions from Chapter 3.4 (new key-value pairs are inserted at the beginning of the list).

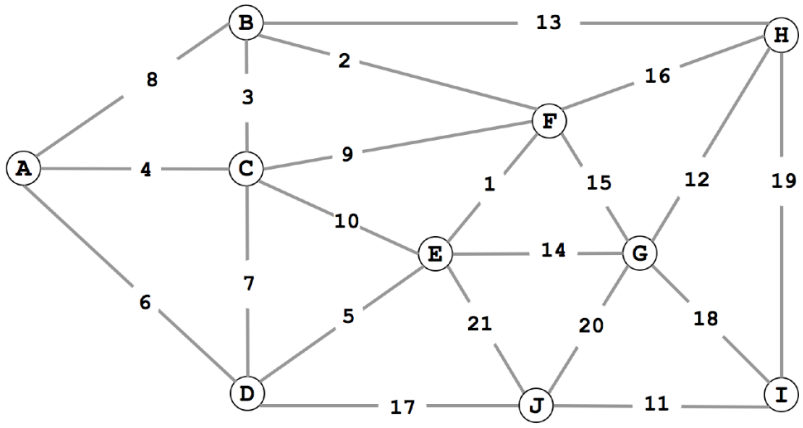
   M I T C H D A N I E L S

0	A				
1	L				
2	H	C	M		
3	S	I	N	D	I
4	E	T			

3. (4 pts) List the vertices in the order in which they are visited (for the first time) in DFS for the following undirected graph, starting from vertex 0. For simplicity, assume that the Graph implementation **always iterates through the neighbors of a vertex in increasing order**. The graph contains the following edges:

```
0-1 1-2 1-7 2-0 2-4 3-2 3-4 4-5 4-6 4-7 5-3 5-6 7-8 8-6
    3, 6
```

4. (7 pts) Consider the following weighted graph with 10 vertices and 21 edges. Note that the edge weights are distinct integers between 1 and 21. Since all edge weights are distinct, identify



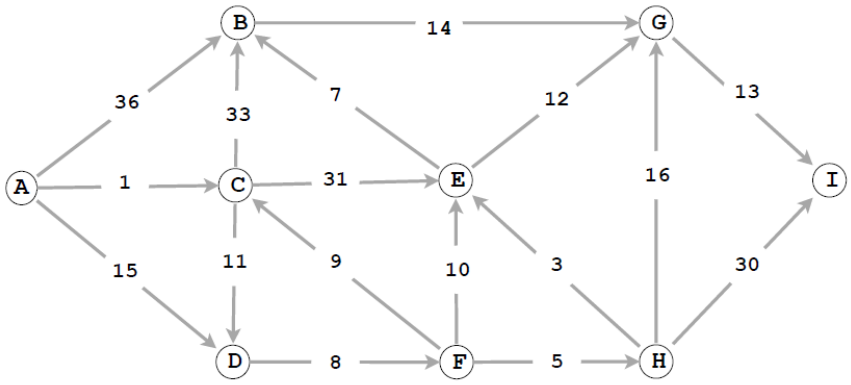
(a) List the sequence of edges in the MST in the order that Kruskal's algorithm includes them (starting with 1).

E-F, F-B, B-C, C-A, E-D, I-J, H-G, H-B, J-D

(b) List the sequence of edges in the MST in the order that Prim's algorithm includes them. Start Prim's algorithm from vertex A.

A-C, C-B, B-F, F-E, E-D, D-J, J-I, I-G

5. (5 pts) Consider the following weighted digraph and consider how Dijkstra's algorithm will proceed starting from vertex A. List the vertices in the order in which the vertices are dequeued (for the first time) from the priority queue and give the length of the shortest path from A to each vertex.



Vertex	A	C	D__	F__	H__	E__	B__	_G_	I_
Distance	0	1	12_	23_	28_	32_	36_	44_	58_

6. (5 pts) Sort the 12 names below using LSD string sort. Show the result (by listing the 12 full words) at each of the four stages of the sort:

John, Jane, Alex, Eric, Will, Nick, Jada, Jake, Nish, Luke, Yuan, Emma

Jada	Yuan	Jada	Alex
Emma	Nick	Jake	Emma
Eric	Jada	Jane	Eric
Jane	Alex	Nick	Jada
Jake	John	Will	Jake
Luke	Eric	Nish	Jane
Nish	Jake	Alex	John
Nick	Luke	Emma	Luke
Will	Will	John	Nick
Yuan	Emma	Eric	Nish
John	Jane	Yuan	Will
Alex	Nish	Luke	Yuan

3. The Right Data Structure (8 pts)

Indicate, for each of the problems below, the best data structure from the following options: binary search tree, hash table, linked list, heap. Provide a brief justification for each answer.

1. Find the k^{th} smallest element.

   Following the left edges of a BST yields the smallest elements, thus traversing it from the leftmost bottom node upwards yields the k^{th} smallest element.

2. Find the last element inserted.

   A heap provides constant time access to the last element inserted because it will be inserted as the last element in the heap. Accessing that element is as easy heap[length – 1].

3. Find the first element inserted

   Retrieving the first element in a Linked List can be done by accessing the head of the list.

4. Guarantee constant time access to any element.

   A heap provides constant time access because there is no traversing to find the element, you simply access it as if it was a regular array.

4. Design/Programming Questions (34 pts)

1. (7 pts) Give the pseudocode or Java code for a linear-time algorithm to count the parallel edges in an undirected graph.

   Assuming a sorted adjacency list *list* of size N

```
ST[] marked = ST[N]
int count = 0
for i:0->N
      int prevVert, currVert
      Boolean lastMatched = false
      for j:0->N
            if j is 0
                  prevVert = list[i, j]
                  currVert = prevVert
            else
                  prevVert = currVert
                  currVert = list[i, j]
                  if currVert is prevVert
                        if !marked[currVert].contains(i)
                              count += 1
                              marked[currVert].insert(i)
                              lastMatched = true
                        else if marked[currVert].contains(i) and
lastMatched
                              count += 1
                  else
                        lastMatched = false
return count
```

2. (7 pts) Given an MST for an edge-weighted graph G and a new edge e, describe how to find an MST of the new graph in time proportional to V .

   At the new vertex given by e, iterate through its edges and add the smallest-weight edge to the MST. Since the MST already does not contain any cycles, adding a single edge will result in no cycles. Finding the smallest-weight edge should take O(edges/V) where edges is the amount of edges given by the vertex.

3. (10 pts) Design a data type that supports:

   • insert in logarithmic time,
   • find the median in constant time,
   • remove the median in logarithmic time.

   Give pseudocode of your algorithms. Discuss the complexities of the three methods. Your answer will be graded on correctness, efficiency, and clarity.

```
PriorityQueue<Integer> leftSide             //max oriented heap
PriorityQueue<Integer> rightSide            //min oriented heap
int numOfElements

insert(int x)
      if numOfElements % 2 is 0
            leftSide.add(x)
            if rightSide.size is 0
                  numOfElements += 1
                  return
            else if leftSide.peek() > rightSide.peek()
                  int leftSideRoot = leftSide.pop()
                  int rightSideRoot = rightSide.pop()
                  leftSide.add(rightSideRoot)
                  rightSide.add(leftSideRoot)
      else
            rightSide.add(x)
      numOfElements += 1
find()
      if numOfElements % 2 is 0
            return (leftSide.peek() + rightSide.peek()) / 2
      else
            return rightSide.peek()
remove()
      if numOfElements % 2 is 0
            leftSide.pop()
      else
            rightSide.pop()
      numOfElements -= 1
```

The `find()` operation takes constant time because the `peek()` method only needs to return the root of the queue. `insert()` takes logarithmic time because `add()` may require needing to look through all elements in the queue before adding. `remove()` is similar in this regard that it takes logarithmic time to recreate the heap after removing it's root with `pop()`.

4. (10 pts) The 1D nearest neighbor data structure has the following API:
   - `constructor`: create an empty data structure.
   - `insert(x)`: insert the real number x into the data structure.
   - `query(y)`: return the real number in the data structure that is closest to y (or null if no such number).

   Design a data structure that performs each operation in logarithmic time in the worst- case. Your answer will be graded on correctness, efficiency, clarity, and succinctness. You may use any of the data structures discussed in the course provided you clearly specify it.

   1D should be implemented using an 18 way trie with -9 through -1 and 1 through 9 being the base keys. Insert should treat the real number as an array of numbers 0-9 and a decimal point. For example, `insert(5.25)` should result in **5**→`.`→`2`→`5`. An R-way trie guarantees an insert time of O(L) which satisfies the logarithmic requirement. A query should act the same as search in a R-way trie with a few exceptions. `query(0)` should always return 0. A search miss will return null if there are no numbers attached to the base key. A search miss on a base key with children will return the number up to the point of the miss. For example, `query(5.251)` **5**→`.`→`2`→`5`→`NULL` returns `5.25`. A search miss in a R-way trie is the worst-case scenario for a search, but only takes $O(\log_R(N))$. A search success takes only O(L). Both search scenarios satisfy the logarithmic requirement.