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.
   The reverse postorder of a digraph reverse is used in Kosaraju's algorithm to prescribe the order in which strongly connected components are determined via DFS.
2. _**F**_ Adding a constant to every edge weight does not change the solution to the single-source shortest-paths problem.
   Counter example: Consider A-B (weight 2), B-C (weight 1), and A-C (weight 4). SPT from A: A (0), B (2, A-B), C(3, A-B-C). This SPT changes if we add 2 to each weight.
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.
   The interdependency of subproblems is what makes dynamic programming an effective strategy.
4. _**T**_ 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.
   G and GR have the same strongly connected components hence we can exchange their roles
5. _**T**_ If you insert keys in increasing order into a red-black BST, the tree height is monotonically increasing. cf. lecture on balanced trees.
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.
   No support for ordered traversal in hash tables
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. By definition
10. _**F**_ Every directed, acyclic graph has a unique topological ordering.
    If A ->C and B->C and A and B are independent A, B, C and B, A, C are both valid topological orderings

11. _**F**_ Preorder traversal is used to topologically sort a directed acyclic graph.

Topological sort uses *post*-order traversal

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).

Only a small number of characters per string need to be considered in MSD.

13. _**F**_ 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.

The first character of the first key is always going to be the root hence the order affects the TST.

14. _**T**_ In a priority queue implemented with heaps, N insertions and N removeMin operations take $O(N \log N)$. cf. lecture on priority queues.

15. _**T**_ An array sorted in decreasing order is a max-oriented heap.

The sorted array satisfies the requirement that A[n] > A[2*n] and A[n] > A[2*n+1]

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

The cost of reordering the array after each insertion will be negligible and search will have log N complexity

17. _**F**_ The floor operation returns the smallest key in a symbol table that is greater than or equal to a given key. That is the definition of the ceiling.

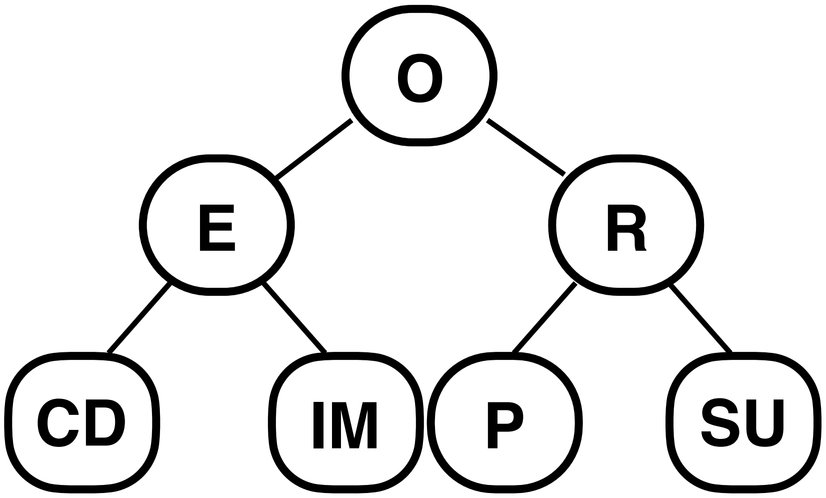
18. _**F**_ The root node in a tree is always an internal node.

Not if the root is the only node in which case it is a leaf.

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
```

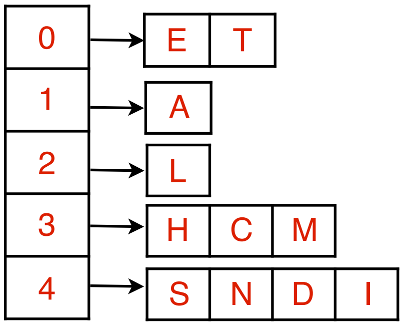


2. (5 pts) Give the contents of the hash table that results when you insert the following keys into an initially empty table of *M = 5* lists, using separate chaining with unordered lists. Use

   the hash function *11k mod 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
```

hash(M) = hash(13) = 143 % 5 = 3
hash(I) = hash(9) = 99 % 5 = 4
hash(T) = hash(20).= 220 % 5 = 0
hash(C) = hash(3) = 33 % 5 = 3
hash(H) = hash(8) = 88 % 5 = 3
hash(D) = hash(4) = 44 % 5 = 4
hash(A) = hash(1) = 11 % 5 = 1
hash(N) = hash(14) = 154 % 5 = 4
hash(E) = hash(5) = 55 % 5 = 0
hash(L) = hash(12) = 132 % 5 = 2
hash(S) = hash(19) = 209 % 5 = 4

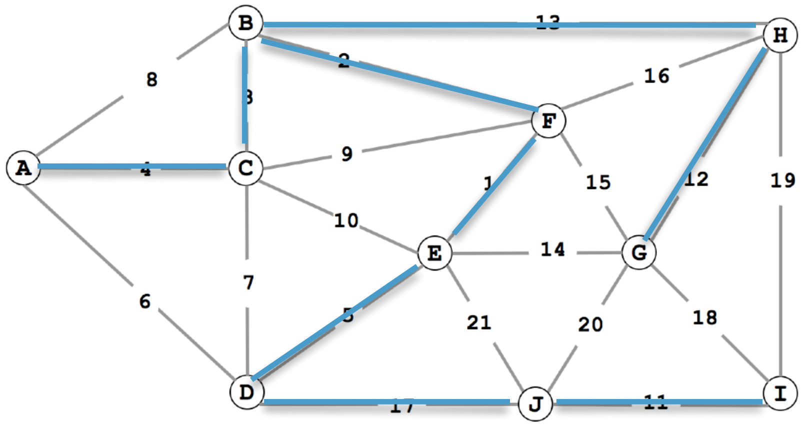


3. (4 pts) List the vertices in the order in which they are visited (for the first time) in DFS for the following undirected graph, starting from vertex 0. For simplicity, assume that the Graph implementation **always iterates through the neighbors of a vertex in increasing order**. The graph contains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

0 1 2 3 4 5 6 8 7

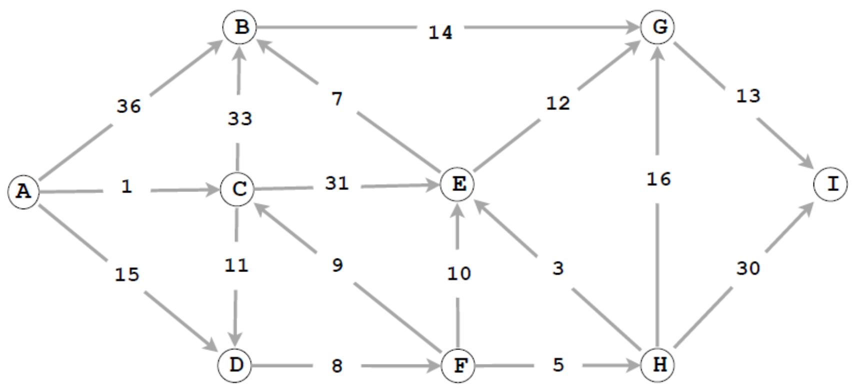
4. (7 pts) Consider the following weighted graph with 10 vertices and 21 edges. Note that the edge weights are distinct integers between 1 and 21. Since all edge weights are distinct, identify each edge by its weight (instead of its endpoints).



(a) List the sequence of edges in the MST in the order that Kruskal's algorithm includes them (starting with 1). **1 2 3 4 5 11 12 13 17**

(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. **4 3 2 1 5 13 12 17 11**

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.



edgeTo:		A-C	C-D	D-F	F-H	H-E	C-B	E-G	G-I
Vertex	A	C	D	F	H	E	B	G	I
Distance	0	1	**12**	**20**	**25**	**28**	**34**	**40**	**53**

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`

John	Jada	Yuan	Jada	Alex
Jane	Emma	Nick	Jake	Emma
Alex	Eric	Jada	Jane	Eric
Eric	Jane	Alex	Nick	Jada
Will	Jake	John	Will	Jake
Nick	Luke	Eric	Nish	Jane
Jada	Nish	Jake	Alex	John
Jake	Nick	Luke	Emma	Luke
Nish	Will	Will	John	Nick
Luke	John	Emma	Eric	Nish
Yuan	Yuan	Jane	Yuan	Will
Emma	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 kth smallest element. Binary search tree. Support in-order traversal and faster than linked list.

2. Find the last element inserted. Linked list. Last element will be at the front of the list.

3. Find the first element inserted. Binary search tree. The first element inserted will be the root of the BST.

4. Guarantee constant time access to any element. Hash table. Probabilistic guarantee of constant time search cost.

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.

```
int n=0;
class Edge {
    int u, v;
    public Edge(int a, int b) { u = min(a,b); v = max(a,b); }
    public int hashCode() {
       int h=17; h = 31*h + u;
       return 31*h + v;
    }
}
Hash<Edge,int> seen;
for (int v=0; v<G.V(); v++) {
   for (int w : G.adj(v)) {
      if (w>v) { // avoid counting edges twice
         Edge e(v,w);
         if (seen.get(e) != null) seen[e].value = seen[e].value+1;
         else seen[e] = 1;
      }
   }
}
int n=0;
for (value in seen.values()) { if (value > 1) n += value; }
return n;
```

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 .

The MST will only change if the new edge `e: v-w` is part of the MST.

Adding `e` to the existing MST creates a cycle by construction. If `e` belongs to the new MST, there must be an edge with higher weight on that cycle. To find this cycle, run DFS on MST from `v`: if a vertex has been checked already when reached by DFS a cycle is found. A `edgeTo` array gives us the list of edges that comprise the cycle, from which we now remove the one with highest weight.

The complexity of this solution is the complexity of the DFS run on the MST, which corresponds to the number of edges in the MST. Since there are `V-1` edges on a MST of `V` vertices, we obtain a linear time complexity in `V`.

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.

Use a **`maxPQ`** for lower half of values, **`minPQ`** for upper half. Assuming binary heap implementation of PQ.

```
insert(value): // preserve order:
if (value > minPQ.min()), minPQ.insert(value) else if (value< maxPQ.max())
maxPQ.insert(value);
// preserve balance:
if (minPQ.size() > maxPQ.size()+1) maxPQ.insert(minPQ.delMin());
else if (maxPQ.size() > minPQ.size()+1) minPQ.size().insert(maxPQ.delMax());

find():if (maxPQ.size()==minPQ.size()) return 0.5*(minPQ.min()+maxPQ.max());
else if (maxPQ.size()==minPQ.size()+1) return maxPQ.max(); else return
minPQ.min().

remove: if (maxPQ.size()==minPQ.size()+1) maxPQ.delMax(); else return
minPQ.min())
```

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.

We note that the nearest neighbor is the closest number of ceiling and floor values; We saw in class how floor and ceiling can be computed in BST. To ensure logarithmic cost, we need a balanced BST.

```
constructor: construct LLRBBST.
insert: Insert in LLRBBST
query: compute floor.
    if floor == value return floor;
    else compute ceiling.
        if ceiling == value return ceiling;
        else if (value-floor > ceiling-floor) return ceiling;
        else return floor;
```