Practice Exam 2 Solutions

This is a collection of relevant problems I have given on previous exams; it does not correspond exactly to a one-hour test. This exam is open book and note; you may use DyKnow, Moodle, and a PDF reader, but not Kojo, or anything else on the internet. Please take some time to check your work. If you need extra space, write on the back. You must show your work to receive any partial credit.

1. Let A be the set $\{a, e, i, o, u\}$, and consider the relation R on A whose graph is given by the following adjacency matrix:

$$\begin{array}{c|ccccc} & a & e & i & o & u \\ \hline a & 0 & 1 & 1 & 0 & 1 \\ e & 0 & 0 & 1 & 0 & 1 \\ i & 0 & 0 & 0 & 0 & 0 \\ o & 0 & 1 & 1 & 0 & 1 \\ u & 0 & 0 & 1 & 0 & 0 \end{array}$$

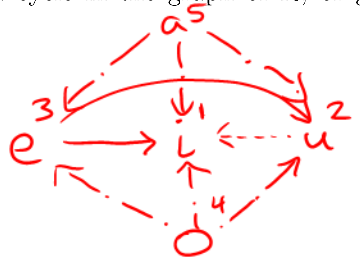
(Recall that the convention is that the cell at row x, column y is 1 if $x\,R\,y$.)

(a) Draw the graph of R:

$$\begin{tikzcd} & a \arrow[dl] \arrow[d] \arrow[dr] & \\ e \arrow[r] \arrow[rr, bend left] & i & u \arrow[l] \\ & o \arrow[ul] \arrow[u] \arrow[ur] & \end{tikzcd}$$

(b) Either identify a cycle in the graph of R, or give a topological ordering of the elements of A according to R:

dfs: (arb. start at e)



tree: $\longrightarrow$
fwd: $\cdots\rightarrow$
cross: $-\cdot\rightarrow$

no back arcs, so acyclic; one topo. order is a, o, e, u, i (reverse of postorder)

only other topo. order is o, a, e, u, i

(c) Which of the following properties apply to R: reflexive, symmetric, transitive, antisymmetric?

ignore

2. Here is an adjacency list representation of a directed graph:

Node	Successors
A	B
B	(none)
C	A
D	A, B
E	B, D
F	D, G
G	C

(a) Draw the graph.

$$\begin{tikzcd}
C \arrow[rr] & & A \arrow[rr] & & B & \\
& G \arrow[ul] & & D \arrow[ul] \arrow[ur] & & E \arrow[ul] \arrow[ll] \\
& & F \arrow[ul] \arrow[ur] & & &
\end{tikzcd}$$

(b) If the graph is acyclic, give a topological ordering of its nodes; otherwise, identify a cycle.

Can do dfs, or notice there are no "left pointing" arcs in this redrawing:
(which also shows a topo. order)

$$\begin{tikzcd}
E \arrow[rr, bend left] \arrow[rrrrrr, bend right] & F \arrow[r] \arrow[rr, bend left] & D \arrow[rrr, bend right] \arrow[rrrr, bend left] & G \arrow[r] & C \arrow[r] & A \arrow[r] & B
\end{tikzcd}$$

(c) What is the longest path in the graph that never revisits a node?

by inspection: $F \rightarrow G \rightarrow C \rightarrow A \rightarrow B$

(d) Define the distance between two vertices as the length of the shortest path between them, or ∞ if there is no such path. What is the greatest non-infinite distance in this graph?

Can run Floyd's algorithm, or

by inspection: G is 3 steps from B
$(G \rightarrow C \rightarrow A \rightarrow B)$

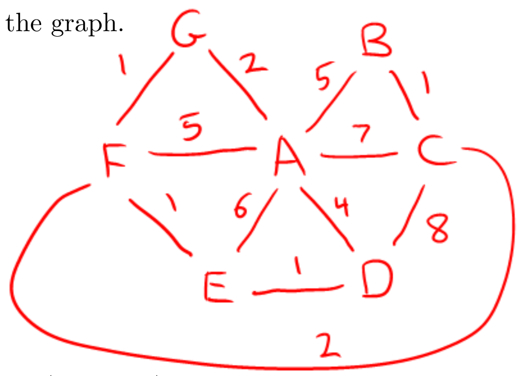
and nothing is further

(F to B via C is 4 steps in (c), but there is a 2-step path $F \rightarrow D \rightarrow B$)

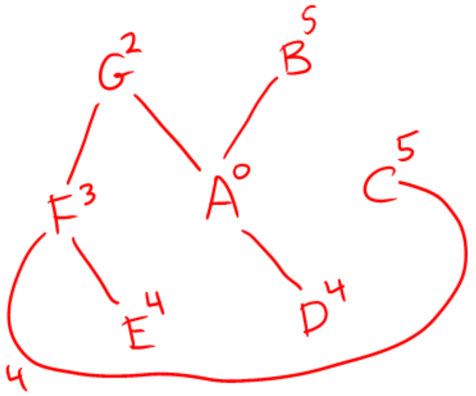
3. Here is an adjacency matrix representation of an undirected weighted graph (a weight of ∞ means there is no edge between those vertices):

	A	B	C	D	E	F	G
A	0	5	7	4	6	5	2
B	5	0	1	∞	∞	∞	∞
C	7	1	0	8	∞	2	∞
D	4	∞	8	0	1	∞	∞
E	6	∞	∞	1	0	1	∞
F	5	∞	2	∞	1	0	1
G	2	∞	∞	∞	∞	1	0

(a) Draw the graph.



(b) Find the (weighted) shortest path from A to E.



Using Dijkstra's Algorithm:

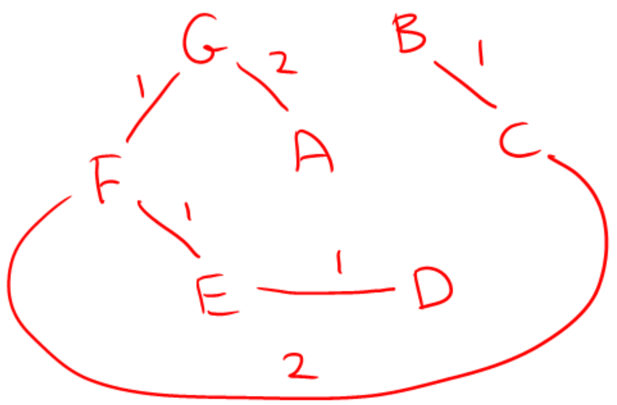
A – G – F – E, length 4

(c) Would the answer to the previous question change if the weight of the edge between B and C were changed to −1? Why or why not?

no, it would not change distance from A to E. The shortest distances A to B and A to C would become 4, but that is not enough to improve the A to E cost.

(d) Find a minimum spanning tree for the graph.

Using Prim's Algorithm:



4. (5 points) Suppose the running time $T(N)$ of some algorithm is given by the following recurrence:

$$\begin{cases} T(1) = 1 \\ T(N) = T(N-1) + 2N - 1, & (N > 1) \end{cases}$$

(a) Fill in the following table of values. For the last entry, give a closed-form expression for $T(N)$, either by solving the recurrence or by guessing:

$T(1)$	$T(2)$	$T(3)$	$T(4)$	$T(N)$
1	4	9	16	N^2

Solving:

$$\begin{aligned} T(N) &= T(N-1) + 2N - 1 \\ &= T(N-2) + 2(N-1) + 2N - 2 \\ &= \dots = T(N-k) + 2((N-k+1) + \dots + (N-1) + N) - k \\ &= T(1) + 2(2+3+\dots+(N-1)+N) - (N-1) \\ &= 2(1+2+3+\dots+N) - N = (N^2+N) - N \boxed{= N^2} \end{aligned}$$

(b) Prove by induction that your closed-form expression for $T(N)$ is correct.

Base Case: $T(1) = 1 = 1^2$ ✓

Ind. Step: Suppose $T(N) = N^2$ for some $N \geq 1$

Then $T(N+1) = T(N) + 2(N+1) - 1$

$= N^2 + 2N + 2 - 1$

$= (N+1)^2$ ✓

So true for all $N \geq 1$.

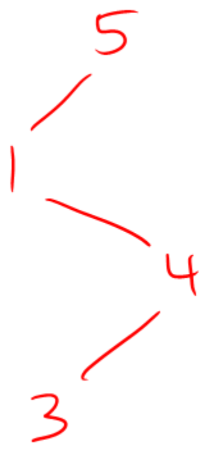
5. (12 points) Here is our Scala code for inserting a value in a binary search tree:

```
trait Tree
case object Empty extends Tree
case class Node(left: Tree, value: Int, right: Tree) extends Tree

def insert(t: Tree, n: Int): Tree = t match {
  case Empty => Node(Empty, n, Empty)
  case Node(l, v, r) =>
    if (n == v)  // No change -- already in tree
      t
    else if (n < v)
      Node(insert(l, n), v, r)
    else  // n > v
      Node(l, v, insert(r, n))
}
```

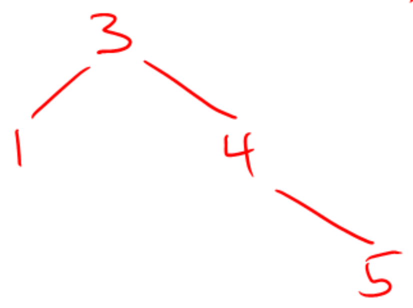
(a) Complete the following skeleton to define a function `insertAll` which takes a tree and a list of numbers and returns a new tree with all of the numbers inserted into the original tree:

```
def insertAll(t: Tree, nums: List[Int]): Tree = nums match {
  case Nil =>
                t

  case head :: tail =>
            insert(insertAll(t, tail), head)

}   [or,   insertAll(insert(t, head), tail)]
```

(b) Show the tree which results from evaluating `insertAll(Empty, List(3, 1, 4, 1, 5))`:



[or,



]

(continued)

(c) Give a tight big-oh upper bound on the average running time of `insertAll` in terms of the size of the list, N (assume that the initial tree is empty, and that the resulting tree is "balanced"):

insert into tree with k nodes is $O(\log k)$ on average, so insertAll is

$$O(\log 1 + \log 2 + \ldots + \log(N-1)) = O(N \log N)$$

(d) Here is a version of inorder traversal which returns the visited items in a list (the `:::` operator concatenates two lists; assume for this problem that this can be done in constant time):

```
def inorder(t: Tree): List[Int] = t match {
  case Empty => Nil
  case Node(l, v, r) => inorder(l) ::: List(v) ::: inorder(r)
}
```

Now we may define the following function:

```
def doSomething(nums: List[Int]): List[Int] = inorder(insertAll(Empty, nums))
```

What is the result of `doSomething(List(3, 1, 4, 1, 5))`?

inorder traversal of a BST lists all values in order:

List(1, 3, 4, 5)

(e) Describe the effect of `doSomething(nums)` on an arbitrary list `nums` of type `List[Int]`:

Sorts the list nums, removing duplicates

(f) Give a tight big-oh upper bound on the average running time of `doSomething` in terms of the size of its argument, N:

$$O(\underbrace{N \log N}_{\text{insertAll}} + \underbrace{N}_{\text{inorder}}) = O(N \log N)$$

This is called "tree sort"