Practice Exam 2

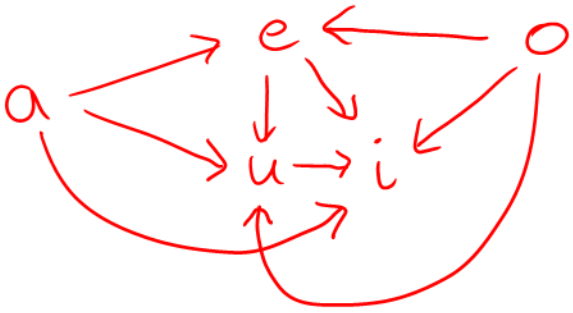
This is a collection of relevant problems I have given on previous exams; it does not correspond exactly to a one-hour test. This exam is open book and note; you may use DyKnow, Moodle, and a PDF reader, but not Kojo, or anything else on the internet. Please take some time to check your work. If you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

	a	e	i	o	u
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

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

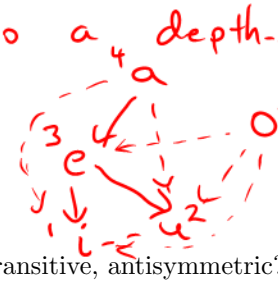
(a) Draw the graph of R:



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

acyclic; a topo. order is a o e u i (or o a e u i)

do a depth-first search:



no back edges

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

not reflexive ($a \not R\ a$)

not symmetric ($a\,R\,e$ but $e \not R\ a$)

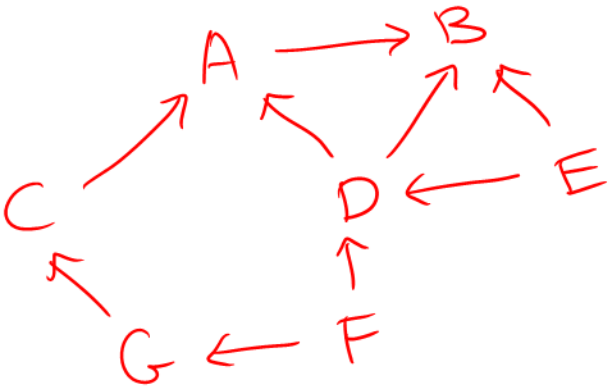
✓ transitive (all shortcuts present)

✓ antisymmetric (no pair x, y where both $x\,R\,y$ and $y\,R\,x$ at all)

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

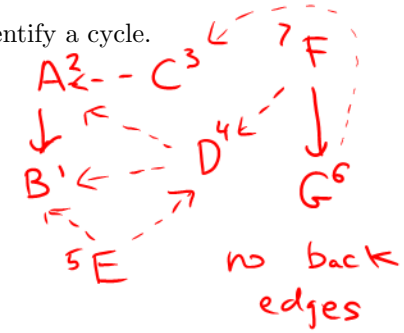
(a) Draw the graph.



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

acyclic; a topo. order is F G E D C A B (several others possible)

DFS:



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

F → G → C → A → B

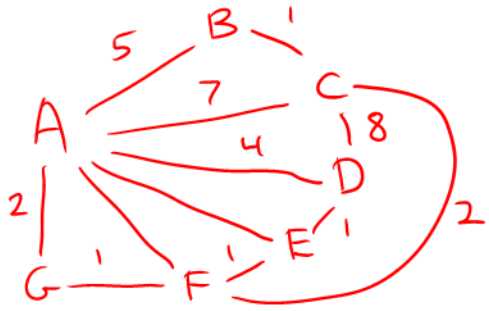
(d) Define the distance between two vertices as the length of the shortest path between them, or ∞ if there is no such path. What is the greatest non-infinite distance in this graph?

3: G → C → A → 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.

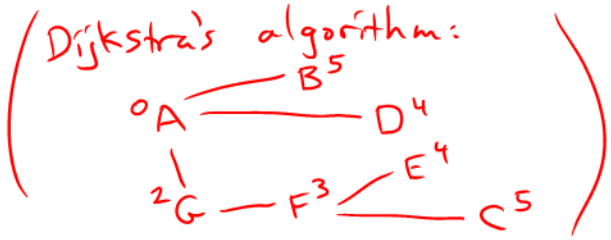


(b) Find the (weighted) shortest path from A to E.

A —2— G —1— F —1— E

weight = 4

(Dijkstra's algorithm:



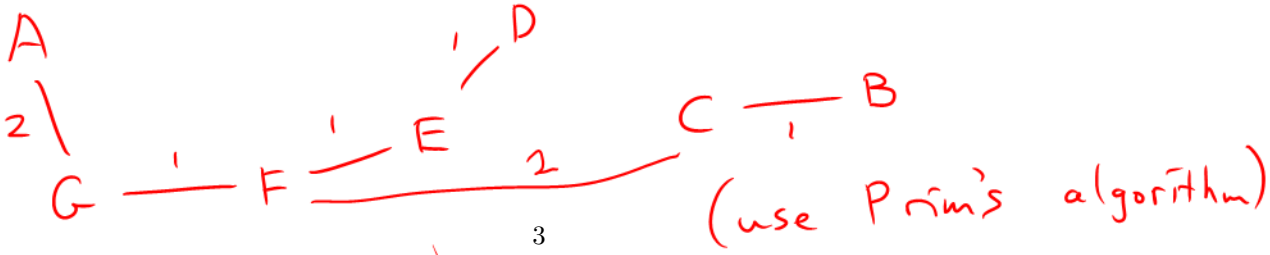
)

(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?

yes — could do A —5— B —-1— C —-1— B ... ... —-1— C —2— F —1— E for an arbitrarily low weight

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

total weight = 8



(use Prim's algorithm)

corrected 11/14/13

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

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

(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

$+2\cdot2-1$ $+2\cdot3-1$ $+2\cdot4-1$ ↑ guess

or solving:
$$\begin{aligned} T(N) &= T(N-1) + 2N - 1 \\ &= T(N-2) + 2(N-1) - 1 + 2N - 1 \\ &= T(N-k) + 2((N-k+1) + \ldots + (N-1) + N) - k \end{aligned}$$
at $k = N-1$:
$$\begin{aligned} &= T(1) + 2(2 + 3 + \ldots + N) - (N-1) \\ &= 2(1 + 2 + \ldots + N) - N \\ &= 2\,\frac{N(N+1)}{2} - N = N^2. \end{aligned}$$

(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 $N > 0$

then
$$\begin{aligned} T(N+1) &= T(N) + 2(N+1) - 1 \quad \text{by recurrence} \\ &= N^2 + 2(N+1) - 1 \quad \text{by ind. hyp.} \\ &= N^2 + 2N + 1 \\ &= (N+1)^2 \quad ✓ \end{aligned}$$

So $T(N) = N^2$ for all $N > 0$.

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 =>
```

insertAll(insert(t, head), tail) (i)

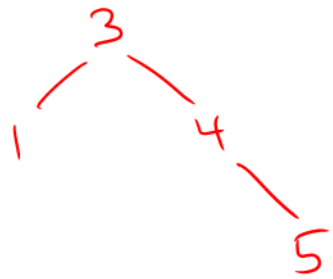
– or –

insert(insertAll(t, tail), head) (ii)

```
}
```

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

by (i):



by (ii):

(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"):

each insert is $O(\log N)$,

so insertAll is $O(N \log N)$

(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))`?

List(1, 3, 4, 5)

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

sorts and removes 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:

insertAll is $O(N \log N)$,
inorder is $O(N)$,
so total time is $O(N \log N)$.

6. (6 points) The following Scala code is one way to implement a state machine; the method `Test.start` takes a list of characters and returns `true` if the machine accepts that input:

```
object Test {
  def start(input: List[Char]): Boolean = input match {
    case Nil => false
    case 'a' :: rest => start(rest)
    case 'b' :: rest => state2(rest)
  }
  def state2(input: List[Char]): Boolean = input match {
    case Nil => true
    case 'a' :: rest => start(rest)
    case 'b' :: rest => state3(rest)
  }
  def state3(input: List[Char]): Boolean = input match {
    case Nil => false
    case 'a' :: rest => state3(rest)
    case 'b' :: rest => state3(rest)
  }
}
```

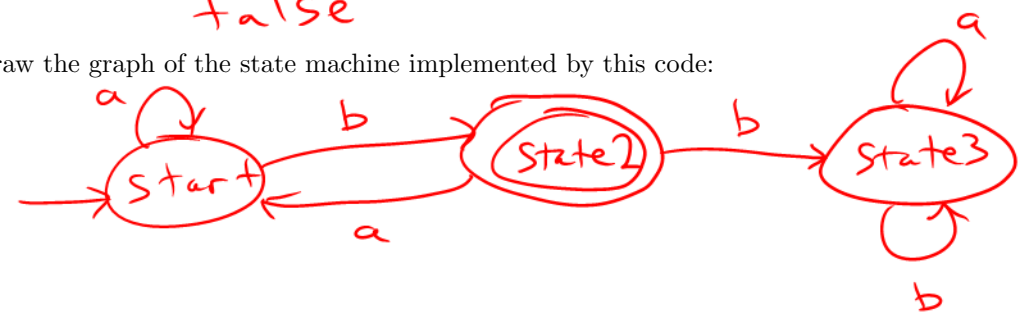
(a) What is the result of `Test.start(List('a', 'b', 'a', 'a', 'b'))`?

true

(b) What is the result of `Test.start(List('a', 'b', 'a', 'b', 'b'))`?

false

(c) Draw the graph of the state machine implemented by this code:



(d) Which of the following regular expressions, if any, match the same set of strings as this machine?

i) $(a^*b^*)^*b$ ✗ *ii)* $(a^*b)^*b$ ✗ *iii)* $(a^*ba)^*b$ ✓ *iv)* $(a \mid ba)^*b$ ✓

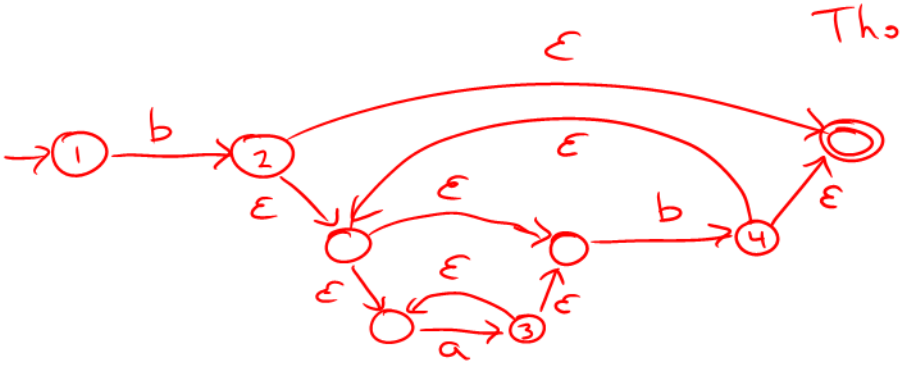
7. (7 points) Consider the regular expression $b(a^*b)^*$

(a) Describe the language matched by this regular expression in English, as simply as you can:

strings of a's and b's, starting and ending with b

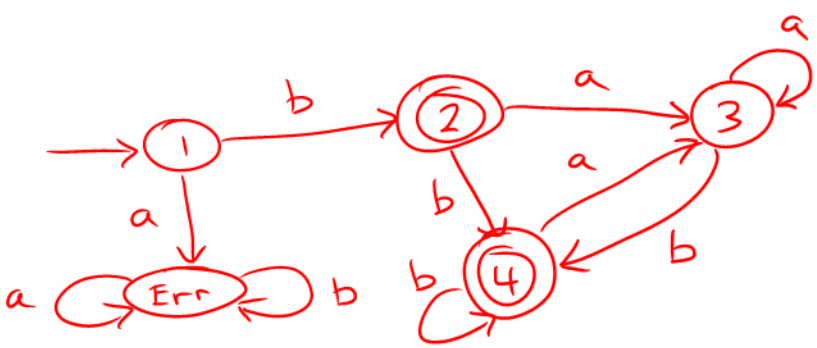
(b) Give a non-deterministic state machine that accepts this language (ϵ-transitions are OK):

Thompson's construction

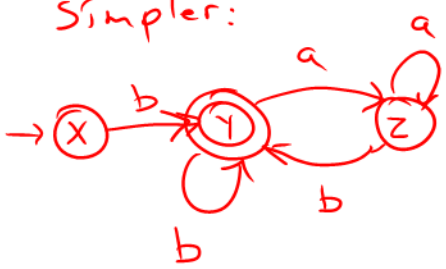


(c) Give a deterministic state machine that accepts this language:

Subset construction (+ ε-elimination)



Simpler:



(d) Give a context-free grammar that generates this language:

Start: $X \rightarrow bY$

$Y \rightarrow aZ \mid bY \mid \varepsilon$

$Z \rightarrow aZ \mid bY$