NATIONAL INSTITUTE OF TECHNOLOGY, CALICUT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS2005 - DATA STRUCTURES AND ALGORITHMS

FINAL EXAM

NAME:	ROLL NUMBER:

MAXIMUM MARKS: 40 TIME : 2 HRS 30 MIN DATE 1/5/2014

GENERAL INSTRUCTIONS:

- There are four sections in the paper.
- The marks division among the sections is. 15+9+10+6
- All questions in a section have to be answered together in a sequence. Answers appearing elsewhere will not be evaluated.
- Sections A and B have to be answered in the question paper itself. Section C and D have to be answered in the main answer sheet provided. Additional answer sheets are only for rough work.
- Conversing, exchanging documents and gadgets and all other forms of suspicable behavior would be appropriately penalized. Academic Integrity violations would lead to a zero for the exam.

2

SECTION A

1

1.Mark the following statements as true/false 1 Mark

a. A given preorder traversal of a binary tree uniquely determines the tree ✗
b. A given inorder traversal of a binary tree uniquely determines the tree ✗
c. A given preorder traversal of a binary search tree uniquely determines the tree ✓
d. A given inorder traversal of a binary search tree uniquely determines the tree ✗

1

2. The depth first search of a directed graph classifies edges as 1 Mark

a. Tree and forward edges
b. Tree, forward, back and cross edges ✓
c. Tree and back edges
d. Tree, back, and cross edges

3. Write the result of executing the Bellman Ford Algorithm on the following graph, with source 1. The edge list order can be taken to be row order of the weight matrix. (No partial marks for this question. Marks[2/0])

$$\begin{matrix} 0 & -4 & 2 \\ 2 & 0 & 2 \\ -2 & 2 & 0 \end{matrix}$$

Π[1]= 3 Π[2]= 1 Π[3]= 2 1.d= -8 2.d= -8 3.d= -6 Return value= false

3.5

4. What does the following code, executed on a binary tree node pointer x, do? The answer has to explicitly mention the information returned, verbal explanations of the code are not required. Marks [2/0]

```
code(x)
if (x==NIL) return(0)
    else return ( 1 EX_OR (code(x.left) EX_OR code(x.right))      # EX_OR means exclusive OR
```

Returns 1 (true) if no: of nodes is odd, 0 (false) otherwise

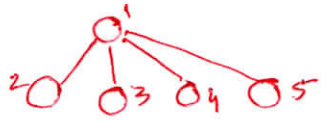
1.5

5. In a given application involving sorting, the input sequence has the following probability distribution: Sorted: 0.001 Reverse sorted: 0.00001: All others: equal probability: 2 Marks

The average case complexity of the following algorithms would then be:

Quicksort $\Theta(n^2)$ Insertion sort $\Theta(n^2)$ Heapsort $\Theta(n \log n)$ Mergesort $\Theta(n \log n)$

6. Draw an undirected connected graph with 5 nodes for which the BFS and DFS from the same source node would result in the same predecessor graph. 1 Mark



7. How many different undirected graphs can be constructed with a given set of n vertices? 1 Mark

a) 2^n

b) $2^{n(n-1)/2}$

c) n(n-1)/2

d) n!

1

8. What does the following code find(T) do given a binary tree T? 1 Mark

```
find(T)
print(mystery(T.root))
```

```
mystery(x)
if x==nil return 0
   else  if x.left != nil or x.right != nil
            return(mystery(x.left) + mystery(x.right))
            else return(1)
```

a) prints the number of nodes in T b) prints the number of leaves in T

c) prints the number of internal nodes in T d) prints the total degree of nodes in T

1

9. Let f(n) and g(n) be respectively the average case and worst case running times of an algorithm on an input size of n. Then which of the following statements is/are correct? 1 Mark

a) $f(n) = \Omega(g(n))$ b) $f(n) = O(g(n))$ c) $f(n) = \theta(g(n))$ d) $f(n) = o(g(n))$

1 10. In a Max-heap in which all the elements are distinct, where might the smallest element reside? 1 Mark

a) Amongst the leaf nodes b) amongst the internal nodes c) leftmost node d) rightmost node

1 11. A 3-ary heap is a heap such that all the non-leaf nodes of the heap have three children. Which of the following is the height of 3-ary heap with 124 nodes? 1 Mark

a) 3 b) 4 c) 5 d) 6

1 12. A single array A[1... N] is used to implement two stacks. The two stacks grow from opposite sides of the array, and top1 and top2(top1<top2) represent the tops of the stacks 1 and 2 respectively. If the space is to be used efficiently the condition for "stack full" is? 1 Mark

a) top2 = top1-1

b) top1=N/2 and top2 =N/2+1

c) top1 +top2 = N

d) top1 = N/2 or top2 =N

SECTION B

2 1. Show by induction that the number of nodes with degree 2 in a binary tree is one less than the number of leaves. 2 Marks

Proof by induction

Let n be the no: of nodes, l be the no: of leaves and d be the number of 2 degree nodes

Base case: For $n=3$, $d=0$ or $d=1$. If $d=0$, then $l=1$ ☹. If $d=1$, $l=2$ ☺

Hence $l = d+1$.

Inductive step: Consider $l = d+1$.

If true for n-node tree the true for n+1 node tree.

for n node tree $l_n = d_n + 1$. Add a new node. It can be either the child of a current leaf, or the child of current 1-degree node.

In Case I, $l_{new} = l_n$. $d_{new} = d_n$. Hence $l_{new} = d_{new} + 1$

In Case II, $d_{new} = d_n + 2$. $l_{new} = l_n + 1$. Hence

$l_{new} = l_n + 1 = d_n + 1 + 1 = d_n + 2 = d_{new} + 1$

Hence proved

2. The internal path length of a full binary tree is defined as the sum, taken over all internal nodes of the tree, of the depth of each node. The external path length is defined as the sum, taken over all the leaves of the tree, of the depth of each leaf. Consider a full binary tree with internal path length I, external path length e, and n internal nodes. Prove that e=i+2n. 2 Marks

In a full binary tree, the no: of internal nodes is one less than the no: of leaves, Let n = no: of internal nodes and l be no: of leaf nodes

Then no: of "incident" edges = $n + l - 1$ (i.e except root all have an edge leading to it, according to parent → child relation).

No: of "leaving" edges = $2n$

(considering parent → child relation) ↓ as no node has 0 leaving edge.

Hence $n + l - 1 = 2n \Rightarrow n = l - 1$

Inductive proof for $e = i + 2n$

Base case : 3 node full tree  $e=2, i=0, n=1$
$\Rightarrow e = i + 2n$
1 node full tree  $e=0, i=0, n=0$

Inductive step Assume it is true for all full node trees upto level $L-1$

Consider a level L tree, with a root node r having two subtrees, each of level $\leq L-1$

Let n_1, l_1, e_1, i_1 & n_2, l_2, e_2, i_2 be the no: of internal nodes, no: of leaf nodes, ~~no: of~~ external path length and internal path length for left & right subtrees of r and n_r, l_r, e_r, l_r be corresponding values for the new tree

$e_{new} = e_1 + l_1 + e_2 + l_2$ (as the path length of each leaf increases by 1)

$= i_1 + 2n_1 + l_1 + i_2 + 2n_2 + l_2 = (i_1 + i_2 + n_1 + n_2) + (l_1 + l_2 - 2) + 2(n_1 + n_2) + 2 - (n_1 + n_2)$

$= i_r + (n_1 + n_2) + 2(n_1 + n_2 - 1) - (n_1 + n_2)$ $[\because L_1 = n_1 - 1 \text{ & } L_2 = n_2 - 1]$

✓ path length
no: of internal nodes including r (which is 0)

$= i_r + 2(n_r)$ Hence proved.

2 3. A directed graph G = <V, E> is such that there is a vertex v_0 in V from which every node is reachable. Consider the undirected graph G' obtained by converting each edge of E into an undirected edge.

2 Marks

a. Is G' connected? Prove your answer True

b. Does G' form a tree? Prove your answer. False.

a. Yes

Consider any 2 nodes u & v in G'. The path $v_0 \overset{P_1}{\rightsquigarrow} u$ in G must have had some edges $(v_0 v_1) \cdots (v_k, u)$, as u is reachable. Similarly the path $v_0 \overset{P_2}{\rightsquigarrow} v$ in G must have had some edges $(v_0 v_1')(v_1' v_2') \cdots (v_s' v)$ as v is reachable

Consider the undirected edges in G'.

The path $(v v_s')(v_s' v_{s-1}') \cdots (v_1' v_0)(v_0 v_1) \cdots (v_k u)$ consists of existing edges in G'. Hence v & u are connected.

As v & u are generic vertices, it is true for

all vertex pairs, so G' is connected.

b) G' need not be a tree.

Counter example:

Let G = 

All nodes are reachable from V_0, but 

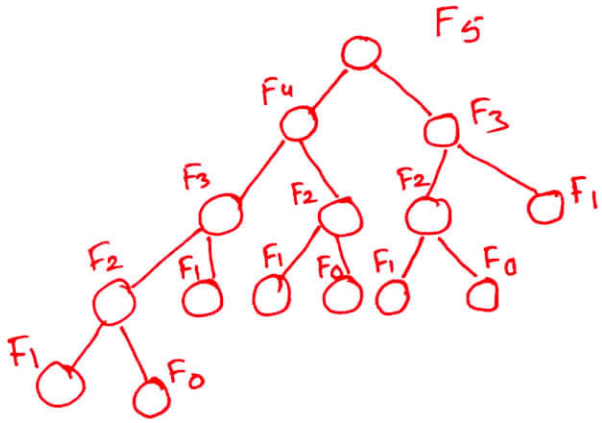
is not a tree.

2 4. Write a function for counting the number of nodes in a singly linked list given a pointer to the head of the list.(Use CLRS pseudo code notation) 2 marks

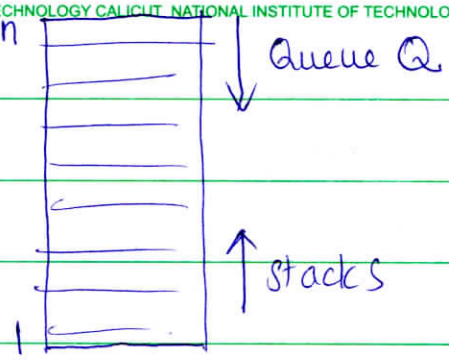
```
count-nodes (x)
1. count = 0
2. while (x != null)
3.         count = count + 1
4.         x = x.next
5. return (count)
```

1 5. Define the Fibonacci binary tree of order n as follows: If n=0 or n=1, the tree consists of a single node. If n>1, the tree consists of a root, with a Fibonacci tree of order n-1 as the left subtree and a Fibonacii tree of order n-2 as the right subtree.

Give an example for a Fibonacci binary tree of order 5. 1 Mark



.5 2 2 2 1.5 .5



Initialize (S, Q)

1. S.top = 1
2. Q.head = N
3. Q.tail = N

~~pop~~ Isempty (S)

1. if S.top == 1 then true
   else false

Isempty (Q)

1. if Q.head == Q.tail then true
   else false

Is_full (S)

1. if ((Q.head = = N)
   and (S.top == Q.tail + 1)) then true
2. else false.

Is full (Q)

1. if ((Q.head == N) and (S.top == Q.tail + 1)) then true
   else false

pop (S)

1. If isempty (S) return null
2. else ~~data~~ S.top = S.top - 1
   return (A [S.top])
3

dequeue (Q)

1. If isempty (Q) then return null
2. e else ~~Q.tail = Q.tail~~
   Q.head = Q.head - 1
   return (A [Q.head + 1])

push (S, x)

1. If isfull (S) return "stackfull".
2. else if (S.top ≠ Q.tail + 1)
3. S.top = S.top + 1
4. A[S.top - 1] = X
5. else. i = Q.head
6. while (i > Q.tail)
7. A[N - Q.head + i] = A[i]
8. Q.tail = N - Q.head + i
9. Q.head = N.

Enqueue (Q, x).

if isfull (Q) return (queuefull)

else if (Q.head ≠ N)

i = Q.head

while (i > Q.tail)

A[N - Q.head + i] = A[i]

Q.tail = N - Q.head + i

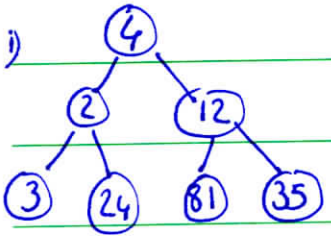
Q.head = N

Q2

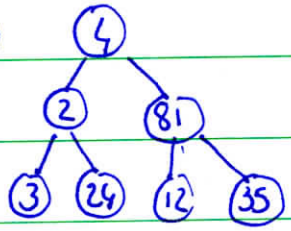
```
delete (x, k)
if (x.key == k)
        p = x
        while (p.key ≠ k)
                p = p.next
else p = x
start = p.
~~while (p.next.key ≠ k) (p != null)~~
q = p.next        ~~p = p.next while q = p.next~~
while (q != null)        ~~while (q.key ≠ k)~~
    if q.key == k
            p.next = q.next
            q = q.next
    else p = q
            q = q.next
return (start)
```

Q3

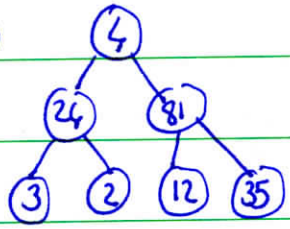
i)



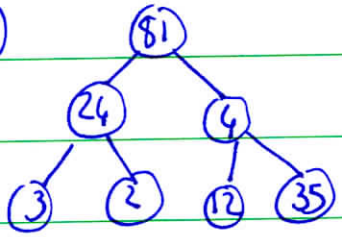
ii)



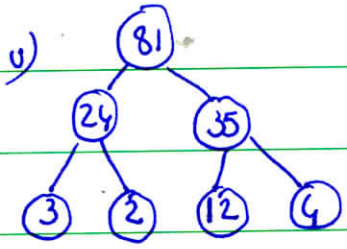
iii)



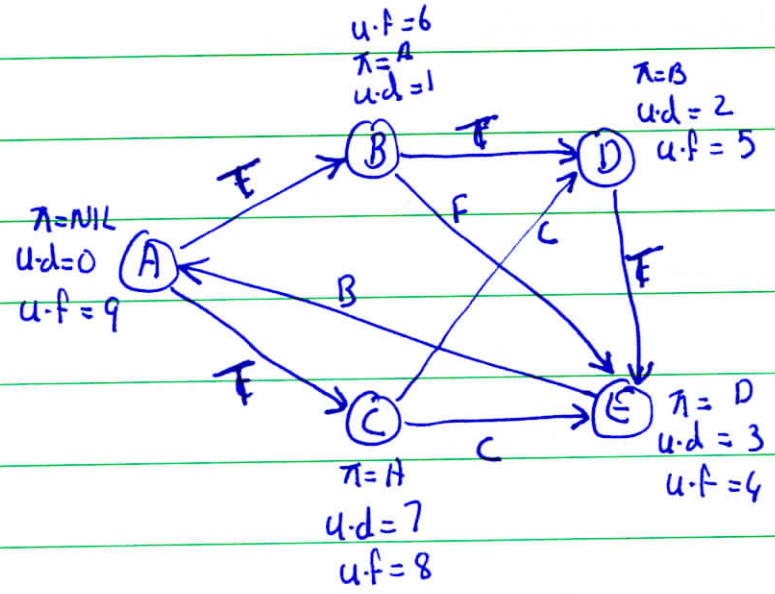
iv)



v)



Q4



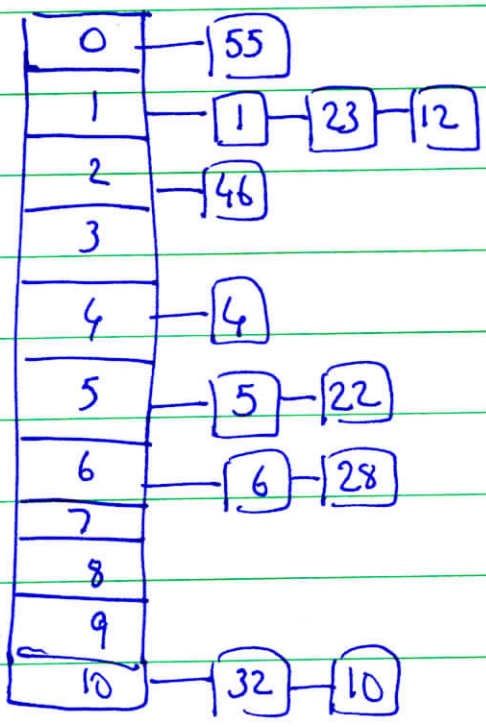
T → Tree Edge

C → Cross Edge

B → Back Edge

F - Forward Edge

Q5



Section - D

1. Level-order (root)

```
1.    Q = ∅          # Initialize Q.
2.    Enqueue (Q, root)
3.    while (Q ≠ ∅)
4.        x = dequeue (Q).
5.        for each u in x.adj
6.            do enqueue (Q, u)
```

2. Modified-Dijkstra (G, w, s)

```
1.  for each u ∈ G.V
2.      do  u.π = nil
3.          u.d = ∞
4.  s.d = 0
5.  Initialize array A[0.. w(|V|-1)] of pointers to nodes
    with null.
6.  A[0] = s.
7.  d = 0          # look at each weight
8.  c = 0          # count nodes whose d is set to δ.
9.  while((d ≤ w(|V|-1) and (c ≤ |V|))
10.     if (A[d] = nil)  d = d + 1
11.     else u = deleted first-node (A[d])
12.         c = c + 1
13.         for each v in Adj[u]
14.             new-relax(u, v, w)
```

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT NATIONAL INSTITUTE OF TECHNOLOGY CALICUT NATIONAL INSTITUTE OF TECHNOLOGY CALICUT NATIONAL INSTITUTE OF TECHNOLOGY CALICUT

Q3 i

new-relax (u, v, w).

1. if (v.d > u.d + W(u, v))
2. K = v.d
3. v.d = u.d + W(u, v)
4. v.π = u
5. ~~Delete (A[K], v)~~
6. Insert (A[v.d], v)
7. If K ≤ W(|V|-1) delete (A[K], v)

delete-first-node (i)

1. P = A[i]
2. A[i] = A[i] → next
3. p → next = null
4. return (p)

Q4

Q5