Paper Number(s): E1.8
E2.7A

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES

Tuesday 25th May 2004 2:00pm

There are THREE questions on this paper.

Answer TWO questions.

This exam is **open book**

Corrected Copy

Time allowed: 1:30 hours.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible:

First Marker(s): Shanahan, M.P.
Second Marker(s): Demiris, Y K.

Information for Invigilators:

Students may bring any written or printed aids into the exam.

Information for Candidates:

Marks may be deducted for answers that use unnecessarily complicated algorithms.

The Questions

1. Assume the existence of the following data types, `TArray` and `TList`, and assume that `TList` has the standard set of access procedures `Empty`, `First`, `Rest`, and `Add`.

```
type
  TList = ^TLink;
  TLink =
  record
    First : integer;
    Rest : TList;
  end;

type TArray = array[1..N] of integer;
```

(a) Write a function with the following header that takes two arrays and returns a linked list of all integers that occur in both arrays.

```
function Matches(A1 : TArray; A2 : TArray): TList
```

[12]

Ensure that the list returned does not contain duplicates.

(b) In general, how many integer comparisons will the procedure perform in the best case? When does the best case occur? Explain your answers. [8]

2. (a) An amoeba reproduces asexually, so each individual has only one parent. Define a Pascal data type `TFamily` that can represent the family tree of an amoeba. Each node in the tree should contain the name of a parent, and have potentially any number of sub-nodes for children. [6]

(b) Write a function with the following header that takes the family tree of an amoeba and two names and returns `True` if they are *siblings* (ie: have the same parent) and `False` otherwise. You may assume that every name in the tree is unique.

```
function Siblings(Family : TFamily;
  Name1 : string; Name2 : string): boolean
```

[10]

(c) Describe in words how you modify your data structure to allow for two parents. [4]

3. Figure 3.1 depicts a binary tree of characters. The tree is not ordered.

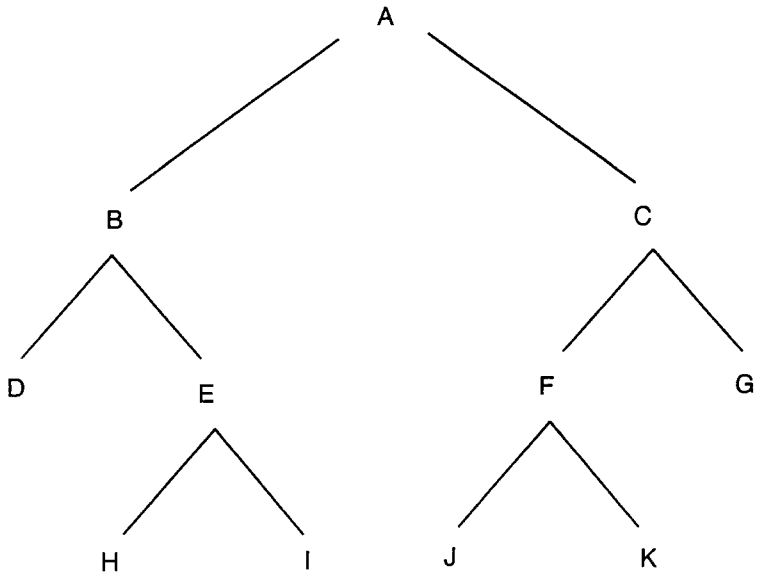


Figure 3.1

(a) Write out the sequence of nodes that would be visited by a procedure that traversed the tree in left-root-right order. [5]

(b) Draw an *ordered* binary tree with the same contents as the tree in Figure 3.1. [5]

(c) If a pointer takes up two bytes in memory, what is the storage requirement for the tree in Figure 3.1, assuming it is represented as a dynamic data structure? Explain your answer. [5]

(d) Draw a sketch showing how the left-half of the tree in Figure 3.1 might be represented in an array rather than using pointers. Explain your answer. [5]

Model Answers

1. (a) [New theoretical application]

```
function Matches(A1 : TArray; A2 : TArray): TList;
var X, Y : integer;
begin
  Ans := EmptyList;
  for X := 1 to N do
    for Y := 1 to N do
      if A1[X] = A2[Y]
      then Ans := AddND(A1[X],Ans);
  return Ans;
end;
```

If the student gets above right but omits to check for duplicates, they should get half the total marks.

```
function AddND(Z : integer; List : TList): TList;
{ Add with check for duplicates }
var Ptr : TList;
begin
  Ptr := List;
  while (Ptr <> EmptyList) and (First(Ptr) <> Z) do
    Ptr := Rest(Ptr);
  if Ptr = EmptyList
  then return Add(Z,List)
  else return List;
end;
```

(b) [New theoretical application]

The best case is when the two arrays have no elements in common. Then the calls to `AddND` will not require any integer comparisons and the total number is N^2 – once for each call to `AddND`. There will be N^2 calls because the invocation is embedded in two nested for loops, each of which carries out N iterations.

(If the student's answer allows for early exit of the inner for loop, then this will be reduced to N comparisons)

The worst case is where the two arrays are identical. Then the total number of comparisons is . We have the same N^2 comparisons as before, plus the comparisons carried out by the calls to `AddND`. The outer loop executes N times, and the i^{th} iteration of the inner loop requires $i-1$ comparisons, because the list of common elements will have length $i-1$.

2. (a) [New theoretical application]

```
type
  TFamily = ^TNode;
  TNode =
  record
    Parent : string;
    Kids : TList;
  end;

  TList = ^TLink;
  TLink =
  record
    First : TFamily;
    Rest : TList;
  end;
```

(b) [New theoretical application]

```
function Siblings(Family : TFamily;
  Name1 : string; Name2 : string): boolean;
var Kids : TList; Found : boolean;
begin
  if Family = nil
  then return False
  else begin
    Kids := Family^.Kids;
    if Member(Name1,Kids) and Member(Name2,Kids)
    then return True
    else begin
      Found := False;
      while Kids <> nil and not Found do
      begin
        if Siblings(Kids^.First,Name1,Name2)
        then Found := True
        else Kids := Kids^.Rest;
      end;
      return Found;
    end;
  end;
```

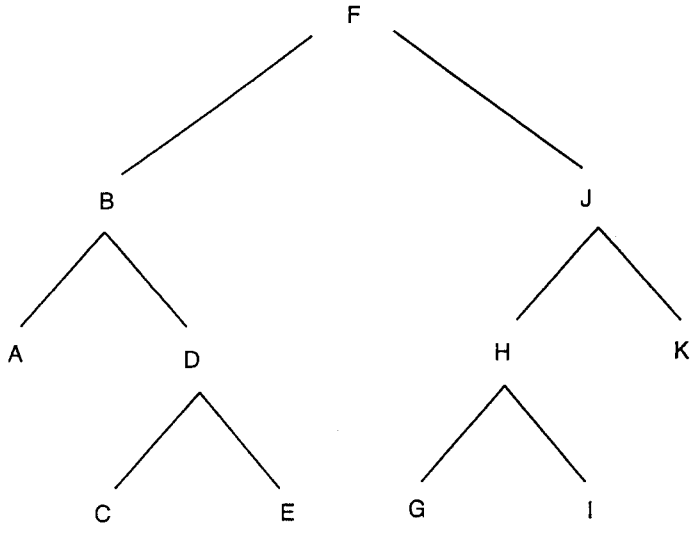
(c) [New theoretical application]

The `TNode` type definition would have to include a field for each parent. But it would not be possible to encode every hereditary relationship in a single tree. To do this, we would require multiple interconnected trees with different root nodes (a forest).

3. (a) [New theoretical application]

D B H E I A J F K C G

(b) [New theoretical application]



(c) [New theoretical application]

Each node *including the leaves* requires two bytes of storage per pointer plus one for the character. So the total is 55 bytes.

(d) [New theoretical application]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
B	4	7	D	0	0	E	10	13	H	0	0	I	0	0	

Each node takes up three elements of the array. The first element is the character, the second is the index of the left sub-node, and the third is the index of the third sub-node. If a node has no sub-node, then a 0 goes in the appropriate location (analogous to the nil pointer).