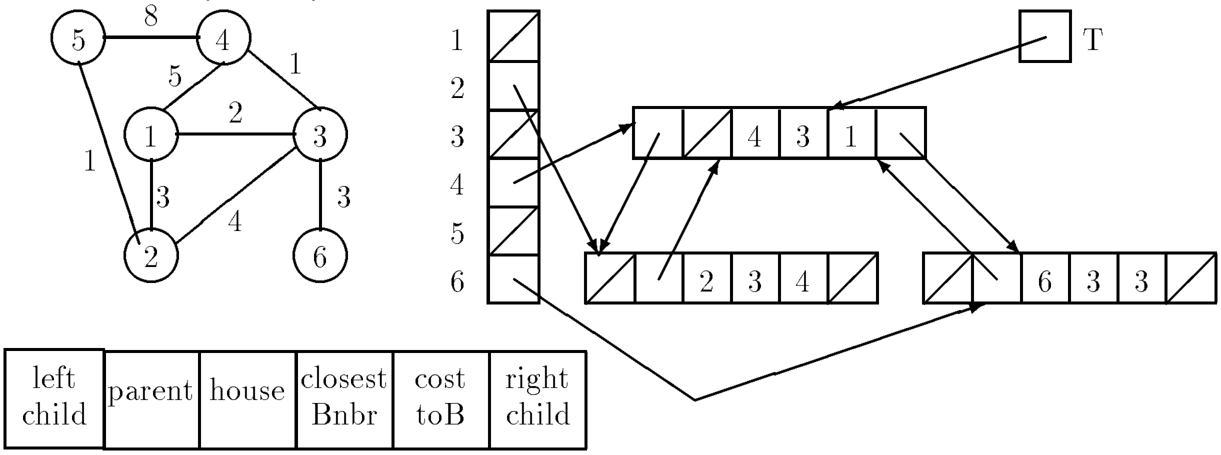
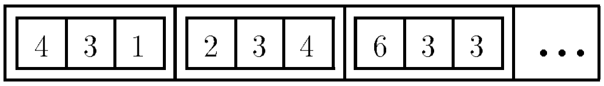
SOLUTIONS TO SECOND TERM TEST 148S 1997 FICH'S SECTION

1. (a) (4 marks) Recall the data structure used in part 4 of assignment 2 to represent the set A of houses. Draw a diagram of the data structure, assuming that $B = \{1,3\}$, $A = \{2,4,6\}$, and the input graph is the one depicted below.
Show and clearly identify all the fields.



(b) (2 marks) Suppose the binary tree in this data structure was represented implicitly using an array instead of explicitly using pointers. Draw a diagram of this modified part of the data structure for the example in part (a).



(c) (2 marks) Give two advantages of this modified data structure as compared to the original data structure.

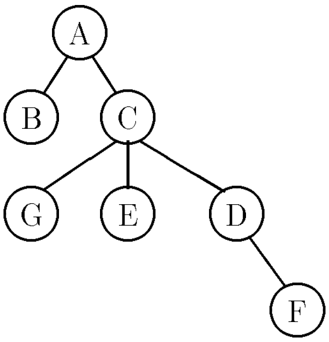
In the modified data structure, there is no need to store pointers to children and parents. This saves space. However, it is not true that the modified data structure always uses less space than the original data structure, since the array must have enough room to store $n-1$ elements in A, say, if there are road segments between house 1 and every other house.

In the modified data structure, it is much easier to compute the location of the rightmost leaf. It is just the ath record in the array.

2. An Eulerian tour of a tree is a path, starting and ending at the root, which follows every edge exactly once in each direction. For example,

A B A C G C E C D F D C A

is an Eulerian tour of the following tree.
Note that there are other Eulerian tours of this tree.



(a) (6 marks) Suppose the following type declaration has been made

```
type treenode :
    record
        NodeName : char
        LeftKid, MidKid, RightKid : ^treenode
    end record
```

Write a recursive procedure to print an Eulerian tour of a tree, given a pointer to its root.

```
procedure EulerianTour(t:^treenode)
%assumes t is not nil
put t->NodeName
if t->LeftKid <> nil then
     EulerianTour(t->LeftKid)
     put t->NodeName
endif
if t->MidKid <> nil then
     EulerianTour(t->MidKid)
     put t->NodeName
endif
if t->RightKid <> nil then
     EulerianTour(t->RightKid)
     put t->NodeName
endif
end EulerianTour
```

(b) (4 marks) Write a statement that expresses the correctness of your procedure in a form that can be used to prove it correct.
DO NOT PROVE YOUR PROCEDURE CORRECT.
$S(n)$: If EulerianTour is called with a tree of $n \geq 1$ nodes, then it prints an Eulerian tour of that tree and returns.

3. (9 marks) Recall the following type declaration and procedure from the first midterm:

```
type box :
    record
```

```
        stuff : int
        arrow : ^box
    end record

procedure P (var c : ^box)
% Deletes the third last element of the singly linked list pointed to by c,
% if c has at least 3 elements.
% Otherwise, the list pointed to by c remains unchanged.
var a, b : ^box
% a and b are temporary pointers into that list.
var i : int

% Decide if the list has fewer than 3 elements.
% If so, return without changing the list.
a := c
i := 0
loop
     if a = nil then
         return
     end if
     a := a -> arrow
     i := i + 1
     exit when i = 3
end loop

% If the list has exactly 3 elements, delete the first element.
if a = nil then
    c := c -> arrow
    return
end if
% Find the fourth last element in the list.
b := c

loop % SECOND LOOP
    % a points 3 elements farther along the list than b does.
    a := a -> arrow
    exit when a = nil
    b := b -> arrow
end loop
% b points to the fourth last element in the list.
% Delete the third last element in the list.
b -> arrow := b -> arrow -> arrow
end P
```

Which of the following are loop invariants for the SECOND loop? For each, briefly explain why or why not.

(a) b $\neq$ nil YES NO

This is a loop invariant. The SECOND loop is not reached unless the list `c` has at least 4 elements. Since `b := c` immediately before the loop is executed, the condition is true before the first iteration. Notice that `a` points 3 elements farther along the list than `b` does at the beginning of every iteration of the list and `b` is advanced only if `a` was advanced and did not fall off the end of the list. Thus `b` remains non-nil.

(b) `i = 3` YES NO

This is a loop invariant. The exit condition of the first loop is `i = 3`. The value of `i` is not changed thereafter. Thus the condition is true before and after every iteration of the second loop.

(c) `b->arrow->arrow = a` YES NO

This is NOT a loop invariant. It is not true after the last iteration of the loop, i.e. when the loop exits. For example, consider a list of length 4.

4. (3 marks) Assume that you have written a module for an interesting new data structure. Why is it important to BOTH test it and prove it correct?

They do complementary things. Proving a program correct makes sure that the program meets its specifications. Testing catches syntax errors. It also helps us determine whether the specifications really capture what was intended. It can sometimes help us find errors. But finding no errors does not mean that the program is correct.