

UNIVERSITY COLLEGE DUBLIN

THE NATIONAL UNIVERSITY OF IRELAND, DUBLIN

An Colaiste Ollscoil, Baile Atha Cliath

Ollscoil na hÉireann, Baile Atha Cliath

SUMMER EXAMINATIONS 1999

M.Mangt.Sc. DEGREE EXAMINATION

MIS 406 : ALGORITHMS & DATA STRUCTURES : SOLUTIONS

Prof S. McClean

Dr A. Deegan

Dr Derek R. O'Connor

Answer any 4 questions

Time : 3 hours

1. (a) Define the abstract data type *Queue* and show how it can be implemented using a singly linked list. Write a Pascal procedure for *DeQueue*.
- (b) Write a Pascal procedure that deletes the last element of a singly-linked list that uses a header node. Check for special cases.
- (c) Write an algorithm that reads in a stream of words and outputs a stream of *distinct* words. Use a *hash table*.

SOLUTION (a) See notes

(b)

```

procedure DeleteLast( VAR L : listtype);
VAR
  p : nodeptrtype;
  IF NOT Empty(L) DO BEGIN
    p := L;
    while p^.link^.link <> NULL do begin
      p := p^.link;
    end; {WHILE}
    p^.link := NULL;
  end; {IF}
end; {PROC}

```

Note that the WHILE test stays two nodes ahead of p .

(c) Assume we have implemented a hash table with the operations $Member(x, T)$ and $Insert(x, T)$ the the following fragment does the job :

```

while NOT EoStream do
  word := GetWord(Stream)
  if NOT Member(word, Htable) then
    WriteOut(word)
    Insert(word, Htable)
  endif
endwhile

```

Note that we could use an array or a linked list to implement **Htable** but each would be very inefficient. A properly tuned hash table would make $Member(x, T)$ and $Insert(x, T)$ $O(1)$. (see notes)

2. (a) Define a Heap and write a Pascal procedure that implements the heap operation *SiftUp*. Derive the complexity of *SiftUp*.
- (b) Explain and write the procedure *MakeHeap* and show how it works on the set of numbers below.
- (c) Show how and why a heap can be used to sort in $O(n \log n)$ time. Illustrate your answer by sorting the following set of numbers :

40 91 21 45 23 89 55 40 21 62 33

SOLUTION (a) See notes

(b) see notes

(c) see notes

3. (a) Define the *Depth First Search* algorithm and show how it works on the graph in Figure 1. Assume it starts at node *A* and that the adjacency lists are in alphabetical order.
- (b) Write an *iterative* version of the DFS algorithm.
- (c) How could DFS be modified to test if there is a path between any two nodes (u, v) in an undirected graph.

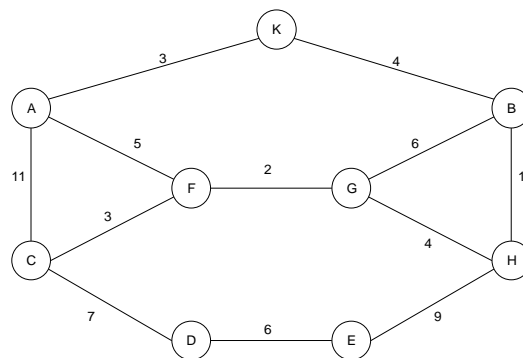


Figure 0.1. Graph for Questions 3 & 4.

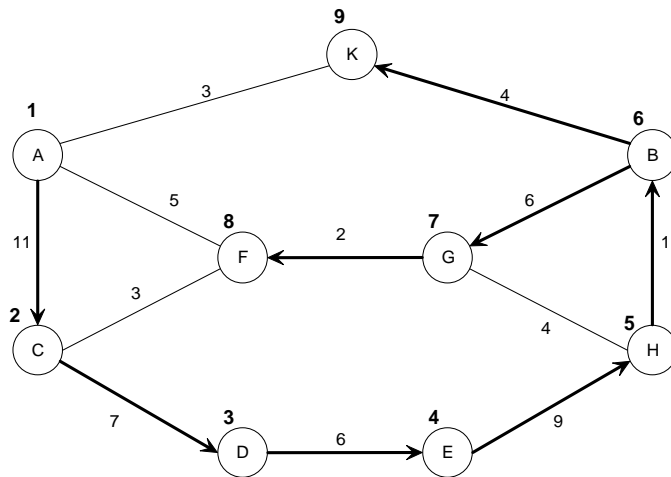


Figure 0.2. Solution Graph for Question 3 .

SOLUTION (a) See notes for definition

(b) This is the SearchG algorithm modified to use a stack.

```

algorithm DFSIter(r:node; p: array)
{ S is a Stack }
Create(S);
FOR each node u ∈ N DO visited[u] := false
Push(r,S)
visited[r] := true
WHILE NOT Empty(S) DO
  u := Pop(S)
  FOR each node v ∈ Adj(u) DO
    IF NOT visited[v] THEN
      p[v] := u
      visited[v] := true
      Push(v,S)
    ENDIF
  ENDFOR
ENDWHILE
endalg SearchG

```

(c) The simplest but not the most efficient is do a DFS from *u* and see does it reach *v*. Here it is :

```

function IsPath(u,v:node):BOOLEAN
-----
    DFS(u);
    IsPath := visited[v]
end FUN IsPath

```

We could make this more efficient by altering DFS(u) so that it stops if it reaches v . However, either algorithm would be $O(m + n)$.

4. (a) Define *Prim's* and *Dijkstra's* algorithms and show how Dijkstra's algorithm works on the graph in Figure 1.
 (b) Show how the algorithms above can be efficiently implemented, analyse the implementations, and show where they differ.
 (c) Another algorithm for the MST problem is as follows : delete one-by-one the most costly arcs whose deletion does not disconnect the graph. Show how this algorithm could be implemented and compare its complexity with Prim's algorithm.

5. (a) Define the abstract data type *Binary Search Tree* and write the Pascal procedures that implement the operations $Member(x, T)$ and $Insert(x, T)$.
 (b) Write a *recursive* Pascal function that counts the number of nodes in a binary tree.
 (c) What is the result of applying an *InOrder Traversal* on a binary search tree. Explain your answer by building a binary search tree from the following set of words :

[can be used heap on big with an idea to by tree that].

SOLUTION (a) See notes

(b) The number of nodes in a binary tree is either 0 or (1 + No. in left tree + No. in right tree). Hence we get the following Pascal function.

```

function CountBT(T : BinTreeType) : INTEGER
-----
begin
    if Empty(T) then CountBT := 0
    else
        CountBT := 1 + CountBT(T^LChild)+CountBT(T^RChild);
    end;

```

(c)

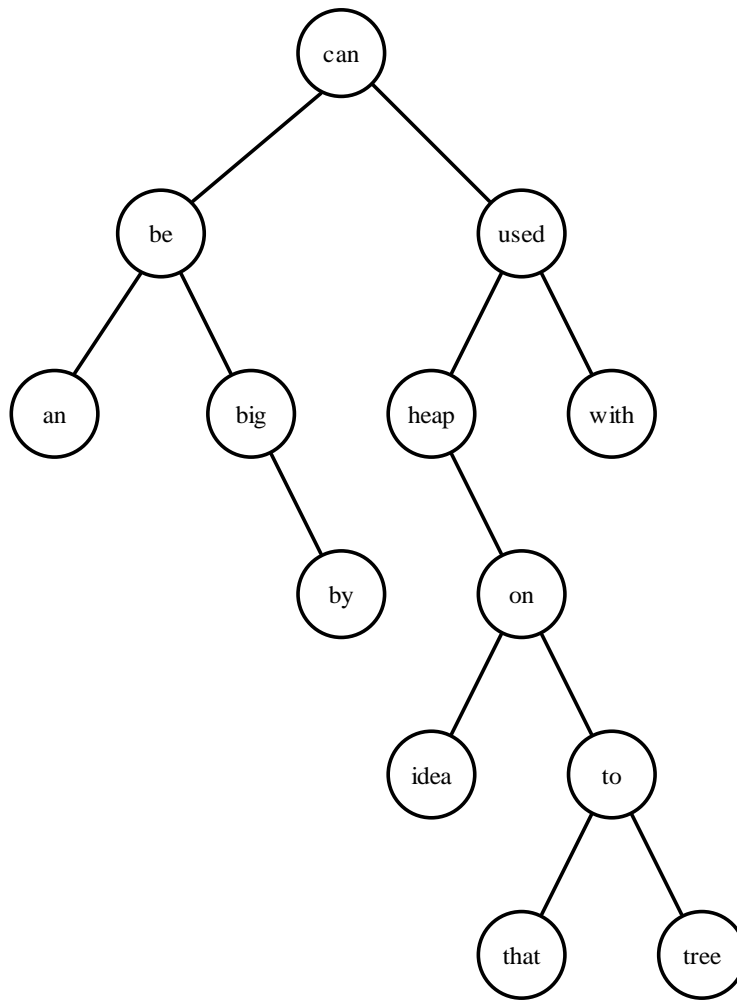


Figure 0.3. Binary Search Tree for Question 5(c).