University of Western Australia

Data Structures 201

Final Examination

June 1994

This paper contains:

Section A (three questions);
Section B (three questions).
Five pages in total.

Time allowed: One hour
plus **reading time:** Ten minutes

Questions in Section A each carry 5 marks.
Questions in section B each carry 10 marks.

Marks for this paper total 35.

Candidates should answer all *three* questions from Section A, and *only two* questions from Section B.

Section A

Answer all three questions from this section.

A1 *(5 marks)*

(a) The diagram represents a Vuillemin priority queue, `v`.

Draw diagrams representing the results of evaluating the following expressions:

```
 i. enq(7,v)
ii. deq(v)
```

---

continued...

A2 *(5 marks)*

(a) The diagram represents an AVL search tree, `a`.

Draw diagrams representing the results of evaluating the following expressions

i. `insert(3, a)`
ii. `insert(13, a)`
iii. `insert(8,a)`

A3 *(5 marks)*

(a) The diagram shows a weighted, directed graph.

i. Which of the following lists of nodes are in topologically sorted order?
   A. `[a, b, c, d, e, f, g]`
   B. `[a, d, c, f, b, e, g]`
   C. `[a, c, d, f, b, e, g]`
   D. `[a, d, f, c, b, e, g]`
ii. Find a shortest path from `a` to `g`.
iii. What form of graph search would you use to find
   A. a topological sorting of the nodes,
   B. shortest path between two nodes?

---

continued...

Section B

Answer two questions from this section.

B1 *(10 marks)*

This question uses the following declaration of a type of binary trees:

```
datatype Tree = Lf | Nd of Tree * int * Tree
```

We assume that implementations of the following **heap operations** are provided

```
val insert   : int * Tree -> Tree
val remove   : Tree -> int * Tree
val downheap : Tree -> Tree
```

These operations satisfy the following properties:

- labels(`insert`$(a, t)) = a \cup$ labels(t)
- if t is a heap then `insert`(a, t) is a heap
- if $(a, t') =$ `remove`(t) then labels$(t) = a \cup$ labels(t')
  and if t is a heap, so is t'.
- labels(t) = labels(`downheap`(t))
- if the subtrees of t are heaps then so is `downheap`(t)

(a) *(3 marks)*
Draw a diagram of the result of the operation `downheap`(t), where t is the tree

```
Nd
 (Nd (Nd (Lf, 1, Lf), 4, Nd (Lf, 0, Lf)),
    3,
    Nd (Nd (Lf, 5, Lf), 8, Nd (Lf, 2, Lf)))
```

(b) *(3 marks)*
Give an ML signature `QueueSig` suitable for a *Priority Queue* of integers.

(c) *(4 marks)*
Write a functor with header

```
functor HPQ(
   datatype Tree = Lf | Nd of Tree * int *  Tree
   val insert : int * Tree -> Tree
   val remove : Tree -> int * Tree
   val downheap : Tree -> Tree
           ) : QueueSig
```

that uses these functions to implement an integer priority queue.

---

continued...

B2 (a) An implementation of Kruskal's algorithm (for finding a minimal-cost spanning-tree) uses two auxiliary datastructures: a priority queue of edges, and a partition of the vertices.

i. *(4 marks)*
Describe the operations on these datastructures that are used by Kruskal's algorithm. (Give the type of each operation, and briefly describe its effect. You are not asked to describe the implementation of these operations.)

ii. *(2 marks)*
What are the complexities of the priority queue operations, if the queue is implemented using a heap?

(b) Consider an alternative implementation that maintains a sorted list of the edges, removing edges from the head of the list as they are considered. (A datastructure representing a partition of the vertices is still required.)

- *(2 marks)*
What is the complexity of a suitable sorting algorithm?

(c) *(2 marks)*
Consider a graph, $G = (V, E)$. Compare the complexities (in terms of the sizes of the sets (V, E)) of the two implementations of Kruskal's algorithm. You may assume that the operations on a partition are $O(1)$, (this is not strictly correct, but it is a good approximation, adequate for all practical purposes).

---

continued...

B3 *(10 marks)*

(a) *(2 marks)*

What is an *efficient* hash function?

(b) *(8 marks)*

An implementation of sets of integers as **balanced search-trees** is "improved" by hashing with a hash table of size 117 and the hash function

```
fun hash n = n mod 117
```

(each entry in the hash table is itself a balanced search-tree).

For each of the following set operations, give the complexity (in terms of the size, N, of the set) for the original implementation, and say what speedup (or slowdown) you would expect to obtain from the introduction of hashing. (Assume that the hash function is efficient for the data encountered, and that the cost of computing the hash function may be ignored.)

i. `empty`
ii. `isEmpty`
iii. `member`
iv. `insert`
v. `delete`
vi. `union`
vii. `intersect`

The End