ENGINEERING TRIPOS PART IIA

Monday 30 April 2007 2.30 to 4

Module 3I1

DATA STRUCTURES AND ALGORITHMS CRIB

*Answer **all** of Section A (which consists of short questions), and **two** questions from Section B.*

All questions carry the same number of marks.

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

STATIONERY REQUIREMENTS	SPECIAL REQUIREMENTS
Single-sided script paper	none

You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

Version: draft 1

SECTION A

Answer all parts of this question. The question in this section will be marked out of the same total as each question in section B, and each part carries the same weight.

1 (a) A graph has all its edges of length 1. Explain how you would find a minimum-cost spanning tree for it, commenting on any ways you exploit your knowledge about its edge-lengths. [10%]

0.1 Notes

I could use a varient on either Kruskal or Prim's algorithm, but the key nice thing here is that I avoid all need to sort. Each of those have to select the shortest edge that would not form a cycle with others already selected. In this case my method is just to select any edge that does not form a cycle with others.

That leaves me with the challenge of telling when I might introduce a cycle. If I use a mathod based on Prim, I start with an arbitrary vertex in a set of "vertices not yet completely dealt with". At each stage I pick a vertex out from that. I then consider each edge emerging from it. The edge can either reach a new vertex or return to one already processed, In the former case I add the edge to the tree I am growing, and mark the vertex at its end as reached, and put that vertex in my queue. An edge that leads to a vertex already marked would give a cycle so is ignored.

The cost of this is clearly equal to the sum of the number of vertices and edges. Not having to look at edges in order of their length avoids needs for sorting or priority queues.

(b) Why do we normally say that sorting is an $n\log(n)$ process? [10%]

0.2 Notes

Two reasons! Firstly a look at the fact that N items can be in N! different orders and so any decision tree that can discriminate between all of them must have hight at least $N\log(N)$ establishes a lower bound for any sorting alborith that is both general and based on pairwise comparisons. Secondly there are algorithms (eg heapsort) that achieve

a cost proportionl to that target, so it is both an upper and lower assymptotic bound for the complexity of sorting.

(c) A Huffman-style coding will be used to process N symbols each of which consists of 8-bit. In other words it will work over an alphabet of size 256. Rather than counting character frequencies in the material to be handled a fixed coding tree will be used. What is the greatest compression that could possibly be obtained, and what is the worst expansion that could be experienced? In each case explain briefly when and how the case arises. [10%]

0.3 Notes

In the best case each symbol in the message could map onto just one bit! That would be a fator or 8 compression. In the worst case the statistics used would have led to allocation of codes 0, 10, 110, 1110, 11110 (statistical modelling would give each symbol i a probability 2^{-i}) and so each byte in the input could expect to be 256 bits, an expansion by 256/8=32 times.

(d) What does it mean for a sorting method to be *stable*? Which, if any, of Quicksort, Shell's Sort and Heapsort is liable to be stable if implemented simply? [10%]

0.4 Notes

A sort method is stable if it preserves the initial ordering of items whose key is equal. None of the three methods mentioned is stable in obvious implementation.

(e) For a buddy system of free-store administration explain the steps that are taken when a block of store is to be released for re-cycling. [10%]

0.5 Notes

Evaluate size of block being returned. Check if its buddy is free, and if so remove buddy from its freechain and merge them. If you do that you then need to check the buddy of the coalesced block, and so on up to the size of the full heap. When you end up with a block, insert it in a freechain dedicated to blocks whose size is that power of 2. Note total cost can be hoped to be around log(heap size).

(f) Explain the terms *directed graph*, *undirected graph* and *bipartite graph*. [10%]

0.6 Notes

A directed graph is a set of vertices and edges. An undirected one is such that if there is an edge (i,j) then (j,i) also exists. A bipartite graph is a graph where the vertices can be partitioned into two sets A and B such that each edge has one of its vertices in A and the other in B.

(g) Explain how to add a single new item to a *heap*, preserving the heap property. What is the cost of this operation? [10%]

0.7 Notes

The only place in memort you can add an item in the standard representation of a heap is at the end. The value so added may then not honour the heap property, but by doing a comparison above it and in a chain from there to the top of the heap the situation is easily resolved in $\log(n)$ *cost.*

(h) You are going to compress a document using Lempel Zif (LZ) compression. The document is a slightly updated version of one you compressed and sent to the recipient last week. Explain why it could make sense to run the previous version through the compressor, discarding the compressed output, before processing the new version. How would the recipient have to act to recover the text? [10%]

0.8 Notes

Processing the previous version seeds LZ with statistical information about your document, and so lets it start saving space straight away while normally LZ compresses the initial segment of a document less well as it "learns" what strings are present. The recipient would need to seed the decoder by feeding the prior document through so that its internal tables could be in step.

(i) Explain one benefit and one limitation of a garbage collection scheme that works by copying all active data. [10%]

0.9 Notes

Benefits: resolves fragmentation, can permit simple sequential allocation, can restore data locality. Disadvantages: Moves data so that must be legal. Must know where ALL pointers and references are. Probably is STOP and copy, and the delay in the STOP may be bad.

(j) What is a b-tree and how would you look up information in one? You do not need to explain how to create or modify a b-tree. [10%]

0.10 Notes

A b-tree is like a simple search-tree with a large fan-out. In fact the fan-out at any one level will be in some range $n/2$ to n. A common use will have n large and the object of the data structure is to minimise the number of nodes touched. This is a good aim when nodes are stored as blocks on disc. In such case the exact details of the in-memory processing within a block are of only secondary importance.

To look up a key, search for it in top block. If present then OK. Otherwise identify two keys in that block one just larger one just smaller. The block will refer to another for keys between them. Delegate the search down to there.

SECTION B

*Attempt **two** questions from this section. Each question has the same weight in marks.*

2 (a) Suppose that a file-sharing service on the internet has several million subscribers. Some internet service providers attempt to block the network, and so at any one moment only certain pairs of users can exchange information directly. The file sharing software will try to pass on material by fetching it indirectly via other users when that is possible. Explain an algorithm that (supposing you know exactly which pairs of nodes can communicate) can find the shortest number of hops to get data from user A to user B. Explain how its costs will scale as the network grows. [30%]

0.11 Notes

This is Dijkstra's algorithm in the case that all edge-lengths are 1. It can be done by keeping (at a general state in the middle) three sets which contain vertices at distances $n = 1$, n and $n+1$ from the source vertex A. Scan the middle set and each neighbour of a vertex in it must be in one of the other two! Add it to the $n+1$ set if it is not already known. At end of this scan discard the $n-1$ set and rename the other two. When node B is first encountered a distance to it can be deduced. The cost could be as bad as the number of edges in the graph.

(b) An upgrade to the file-sharing software now wants to route information to maximise speed. If assigns each point-to-point link a "cost" based on how long it takes a transfer a file between them. If an intermediate node does not start forwarding data until it has finished receiving it the cost associated with a sequence of links is just the sum of the individual link costs. Explain how to find the cheapest route from A to B in this model. Again give cost predictions for the calculation and discuss how it might scale with network size. [40%]

0.12 Notes

Regular Dijkstra, where this time you need to visit vertices in order of their distance from A. This involves setting up a priority queue. But between this part and the previous one

students can show if they have read the books!

(c) For a rather smaller network you have been challenged to produce a complete table showing the expected costs of communication between all pairs of nodes. But this time your cost model will suppose that intermediate nodes can start retransmitting material as soon as they receive anything, so for instance the cost associated with a path A-B-C-D will now be the greatest of the individual costs A-B, B-C and C-D. Invent an algorithm that will find all the costs in this case, and analyse its complexity. [30%]

0.13 Notes

Walshall, but with $\min(a_{i,j}, \max(a_{i,k}.a_{k,j})$ *as the inner loop.*

3 (a) "treesort" works by adding the items that are to be sorted to an ordered binary tree and then flattening that tree. Explain why the numbers of comparisons performed will be the same as the number done by a variant of quicksort. [25%]

0.14 Notes

One can view the first item added as a first pivot.

(b) What is a splay tree? What are their properties and why might they be preferred to some other related data structures? You are not required to give full details of the exact transformations that maintain splay trees. [25%]

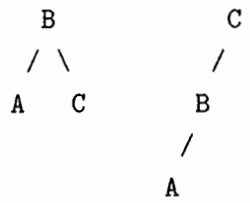
0.15 Notes

Binary ordered tree that is subject to certain rotations each time it is accessed to bring the accessed item to the root. Amortised time is always $\log(n)$ *plus eg n is count of elements actually accessed, not full count of those stored. Preferred to simple binary search tree because they avoid the degenerate worst cases that can arise with ill-balanced trees.*

(c) A splay tree is made by starting with an empty tree and adding items in ascending order. No look-up operations are performed, just the ones that add n items in order. Explain carefully what happens as each item is added. Show what shape the tree ends up and discuss the total cost of building it. [25%]

0.16 Notes

Need to explain the 1-level rotation that always brings the new item to the top again, but that is easy



End result is a fully linear tree, but total cost of building it has been just linear.

(d) "splaysort" works by taking a sequence of items that are to be sorted and adding them one at a time into an initially empty splay tree. A simple traversal of that tree (not changing it at all while traversing it) can then read off the items left-to-right in sorted order on O(n) time. Make predictions about the cost and practicality of splaysort as compared with the use of quicksort. [25%]

0.17 Notes

If input data was in order or reverse order the case from the previous section applies and total cost is linear. The general property of splay trees ensures that cost in worst case is ($n\log(n)$). *In view of part 1 here this might be expected to be a bit like quicksort, but it has better best cases. But it will have worse overhead in average case.*

4 (a) A computer has the text of this examination question and it is going to try to find the word "examination" in it. It will so the search by first checking the final letter of the word searched for. So its initial step will be to compare the underlined characters as here:

- `A computer_has ...`
- `examination`

Working by hand show how the search can proceed from there and comment on how well looking at the final character first helps speed the search up. You do not need to explain in detail how to set up any tables that your method would use. [25%]

0.18 Notes

```
A computer has the text of this examination question..
examination          |        |          ||
           examination        |          ||
                    examination          ||
                               examination|
                                examination  whoopee!
```

The steps onwards are based on considering, given the character observed, where the next possible match for the target string could be.

(b) Instead of comparing individual characters it will now try a method based on a hash value for the target word. Explain how this can work and discuss how the cost will depend on the lengths of both the word sought and the amount of text that it is to be looked for within. [25%]

0.19 Notes

use a hash function of the form `'n'+k*('o+K*('i'+ ...` *where you can incrementally compute hash values at each position in the text you are ssearching through. Cost proportional to length of material searched plus length of target.*

(c) A sequence of k independent hash-values are computed, each in the range from 0 to $N-1$. What is the probability that they are all different? Given that $(1-j/N)$ has about the same value as $(1-1/N)^j$ and that $(1-1/N)^N$ is close to $1/e = 1/2.718 = 0.367$ give a rough estimate, in terms of N, of how large k should be to give a reasonable chance that there has been a hash-collision. [25%]

0.20 Notes

{P = (1-1/N)(1-2/N)(1-3/N)... (1-k/N)}

This is about $(1-1/N)^q$ *where* $q = 1+2+3+\ldots+k = k(k+1)/2$.

We deduce that k proportional to $\sqrt{(N)}$ *is about the critical level. This is of course the traditional Birthday Paradox.*

(d) A programmer generates a sequence that is expected to be "random" by starting with a seed value r_0. At each stage they obtain the next number in their sequence by computing a hash value on the current one. Their way of finding a hash function that works on integers is to start by converting the integer to a string, as if for printing, and then hashing that string. All their integers are kept as 32-bit values. Comment on this scheme. [25%]

0.21 Notes

In light of the previous you might expect to re-visit a value after around 2^16 *samples, and from then you will loop. So however good the hash function is this is liable to be a poor random generator.*

END OF PAPER