

Degree Examination

MX4028 Algorithms

Saturday 30 May 1998

(3pm–5pm)

Attempt *THREE* questions

Calculators may be used *ONLY* for the arithmetic of real numbers or the numerical evaluation of trigonometric, logarithmic and exponential functions. Calculator memories must be clear at the start of the examination; in particular, the use of pre-stored programs is prohibited. Marks may be deducted for answers that do not show clearly how the solution is reached.

1. Write an efficient algorithm, using a pseudocode of your choice, to find the minimum element of a given list of integers.

The algorithm is to be run on an abstract computer on which each basic operation; assignment, comparison between two numbers or incrementing a counter, takes one unit of time. Show that the time $T(n)$ for the algorithm to run on a list of length n satisfies $T(n) = O(n)$.

The timing of one such algorithm is known to be

$$T(n) = 3n - 1 + \mu(n)$$

where μ satisfies $\mu(n) = \mu(n-1) + \frac{1}{n}$ and $\mu(1) = 0$. Obtain an expression for $\mu(n)$ as a series. By comparing with the area under a suitable curve, show that $\mu(n)$ satisfies the inequality

$$\ln n + \frac{1}{n} - 1 < \mu(n) < \ln n.$$

and thus obtain estimates for $T(100)$.

2. Describe a *Huffman* code and give examples of circumstances when it (or a variant) might be used. Illustrate your answer by constructing a binary Huffman code for the string

A TEST EXAMINATION ANSWER

As well as deriving the coding tree, you should give your code for the word ANSWER. Note that the string contains 12 distinct symbols, include the “space” symbol, and 25 symbols in all.

In what sense is a Huffman code optimal?

3. Consider the formal language \mathcal{L} given by the following grammar.

- The alphabet is $A = \{a, b, c\}$.
- The abstract alphabet is $\mathcal{A} = \{\sigma, \alpha, \beta, \gamma\}$.
- The initial symbol is σ .
- The productions are
 1. $\sigma \rightarrow a\sigma\beta\gamma$;
 2. $\sigma \rightarrow a\beta\gamma$;
 3. $\gamma\beta \rightarrow \beta\gamma$;
 4. $a\beta \rightarrow ab$;
 5. $b\beta \rightarrow bb$;
 6. $b\gamma \rightarrow bc$; and
 7. $c\gamma \rightarrow cc$.

(a) Prove that the string abc is in \mathcal{L} .

(b) Prove that the string a^2bcbc is *not* in \mathcal{L} . If productions (4) to (7) are replaced by

$$(4'.) \quad \beta \rightarrow b; \quad \text{and} \quad (5'.) \quad \gamma \rightarrow c.$$

does this remain true?

(c) Prove that the string $a^n b^n c^n$ is in \mathcal{L} for any $n \geq 1$.

(d) Give a simple description of the strings in \mathcal{L} and give arguments to support your claim. [A formal proof is not expected]

4. Let $z^n = 1$, and assume that $z \neq 1$. Verify that

$$1 + z + z^2 + \cdots + z^{n-1} = 0.$$

Let $\omega = \exp(-2\pi i/n)$. Verify that for any integer $n \geq 1$, the Fourier matrix \mathcal{F}_n defined by

$$\mathcal{F}_n = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}$$

has inverse $\bar{\mathcal{F}}_n$, the complex conjugate of \mathcal{F}_n .

Explain how this fact can be used to derive an algorithm which takes time $O(n \log n)$ to multiply two polynomials of degree n . You are not required to prove anything but you should ensure that your notation is defined and that results you use, such as the convolution theorem, are clearly stated.

Degree Examination

SOLUTIONS

MX4028 Algorithms

Saturday 30 May 1998

(3pm–5pm)

1. The algorithm is as follows:

```

algorithm min(x,n) // to find the minimum of x =(x(1),...,x(n))
begin
  min = x(1)
  for i = 2 to n begin
    if ( x(i)< min )
      min = x(i)
  end
  return (min)
end

```

In order to analyse the timing, we present the algorithm as a flowchart:

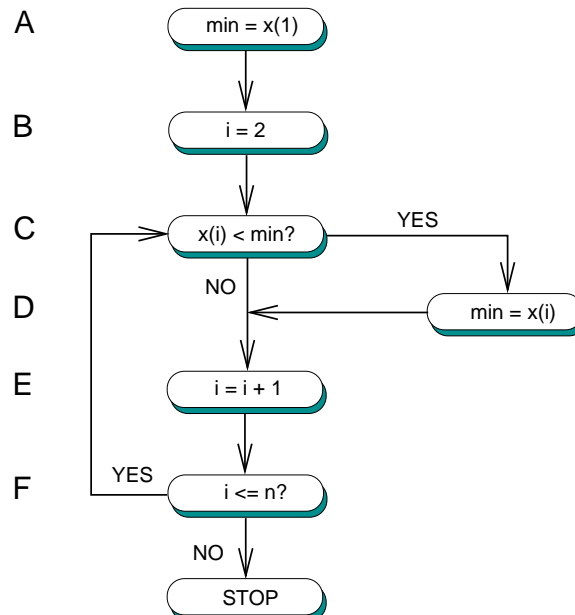


Figure 1: Flow chart to find the minimum element in a list.

We just want to count how many times each box is executed. Boxes A and B are only executed once. Boxes C, E and F are on a loop that the program goes round $n - 1$ times, so each of these is executed $n - 1$ times. That leaves box D; suppose it is visited d times. Clearly the minimum value of d is zero, which occurs whenever the list has its smallest element in first place. And the maximum value of d for a list of n elements is $n - 1$. This will occur when the list is in strictly decreasing order, so that the current value of the minimum has to be updated on each step.

Thus the time $T(n)$ taken by the algorithm to find the minimum of n numbers lies in the range

$$3(n-1) + 2 \leq T(n) \leq 3(n-1) + 2 + (n-1)$$

or

$$3n - 1 \leq T(n) \leq 4n - 2.$$

In order to derive “average” values we assume that the lists that we are dealing with are all the permutations of the list $(1, 2, 3, 4, \dots, n)$. There is nothing limiting about this assumption. We further assume that *all* these permutations are *equally likely* to be presented to the algorithm. Clearly $T(2) = 2.5$ since step D will be visited on half of all “average” inputs. Thus $\mu(2) = \frac{1}{2}$, and so

$$\mu(n) = \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}.$$

Consider now the diagram in Fig 2.

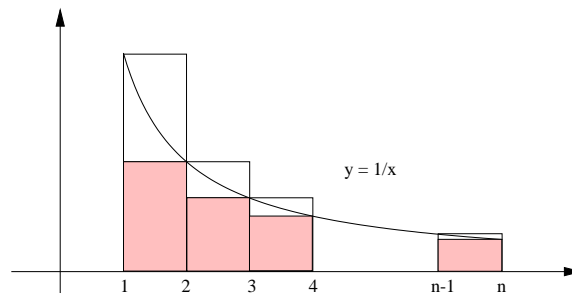


Figure 2: Graph of $y = 1/x$.

By adding up the areas of the rectangles that lie *above* the graph we get

$$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n-1} > \int_1^n \frac{dx}{x} = \ln n.$$

and so $\mu(n) > \ln n - 1 + \frac{1}{n}$. Similarly, by adding up the areas of the rectangles that lie *below* the graph we get

$$\frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} < \int_1^n \frac{dx}{x} = \ln n$$

and $\mu(n) < \ln n$. Combining these we have

$$\ln n - 1 + \frac{1}{n} < \mu(n) < \ln n.$$

Thus

$$3n - 2 + \ln n + \frac{1}{n} < T(n) < 3n - 1 + \ln(n).$$

In particular, if $n = 100$ we have $302.6053 < T(n) < 303.6052$. [And the fact that this estimate can be much improved was noted in the text.]

The Huffman code has the prefix property; no character code is the prefix, or start of the the code for another character. In the same way that the Huffman code was constructed, we can associate a binary code with the prefix property to any binary tree. Given any binary tree with the same set of leaf nodes, and thus able to code the same symbols, the Huffman code is *optimal* in the sense that the corresponding coded message length is at least as short as that from the given tree.

3. (a) We have

$$\sigma \xrightarrow{(2)} a\beta\gamma \xrightarrow{(4)} ab\gamma \xrightarrow{(6)} abc$$

and so abc is in \mathcal{L} .

(b) We first show that a^2bcbc is *not* in \mathcal{L} . Any derivation must start as

$$\sigma \xrightarrow{(1)} a\sigma\beta\gamma \xrightarrow{(2)} a^2\beta\gamma\beta\gamma$$

since this is the only way to derive a string with exactly two copies of a . If we apply (3) at this stage, we are essentially forced to derive the string $a^2b^2c^2$, as in part (c). However making β and γ concrete involves using productions (4) to (7), and the opportunity to use each of these is limited. We must start by using (4), since the others involve b or c . The sequence

$$a^2\beta\gamma\beta\gamma \xrightarrow{(4)} a^2b\gamma\beta\gamma \xrightarrow{(6)} a^2bc\beta\gamma$$

is then forced, and no further productions apply. Thus we cannot derive a^2bcbc .

If productions (4) to (7) are replaced by (4') and (5'), we can derive a^2bcbc as follows:

$$\sigma \xrightarrow{(1)} a\sigma\beta\gamma \xrightarrow{(2)} a^2\beta\gamma\beta\gamma \xrightarrow{(4')} a^2b\gamma b\gamma \xrightarrow{(5')} a^2bcbc.$$

(c) We have the derivation

$$\begin{aligned} \sigma &\xrightarrow{(1)} a^{n-1}\sigma(\beta\gamma)^{n-1} \xrightarrow{(2)} a^{n-1}a(\beta\gamma)^n \xrightarrow{(3)} a^{n-1}a\beta^n\gamma^n \xrightarrow{(4)} \\ &a^{n-1}ab\beta^{n-1}\gamma^n \xrightarrow{(5)} a^n b^{n-1} b\gamma^n \xrightarrow{(6)} a^n b^{n-1} bc\gamma^{n-1} \xrightarrow{(7)} a^n b^n c^n. \end{aligned}$$

Here, each odd labelled production is applied $(n-1)$ times, while each even labelled production is applied only once, and provides the “glue” to move to the next stage.

(d) We have $\mathcal{L} = \{a^n b^n c^n\}$; in other words, \mathcal{L} consists of an arbitrary string of a 's followed by the same number of b 's and concluding with the same number of c 's

To see why this is so; why $a^n b^n c^n$ are the *only* strings which can be derived, we essentially follow the argument in part (b) It is not possible to make β or γ concrete unless we use production (4); since until that is used there is neither b nor c available. But production (4) cannot be used until after (2), which itself stops further use of (1). So at the end of this stage we have $a^n\beta\dots\gamma$ and the intermediate symbols are equal numbers of β 's and γ 's. Note that we can never make concrete a string containing $c\beta$; and one will occur as it did in part (b) unless production (3) is fully utilised. In this latter case we derive $a^n b^n c^n$.

4. It is trivial to verify the factorisation

$$z^n - 1 = (z - 1)(1 + z + z^2 + \dots + z^{n-1}).$$

The result follows, since we are given a zero of the left hand side and $z \neq 1$.

It is enough to show that $\bar{\mathcal{F}}_n \mathcal{F}_n = I_n$, the identity matrix; we have then explicitly produced an inverse. Since the j^{th} row of $\bar{\mathcal{F}}_n$ is

$$\frac{1}{\sqrt{n}}(1, \bar{\omega}^j, \bar{\omega}^{2j}, \dots, \bar{\omega}^{(n-1)j})$$

while the k^{th} column of \mathcal{F}_n is

$$\frac{1}{\sqrt{n}}(1, \omega^k, \omega^{2k}, \dots, \omega^{(n-1)k}),$$

and since $\bar{\omega}^k = \omega^{-k}$, the entry in the $(j, k)^{\text{th}}$ place of the product is just

$$\frac{1}{n}(1 + \omega^{(j-k)} + \omega^{2(j-k)} + \dots + \omega^{(n-1)(j-k)}).$$

We now distinguish two cases. If $j = k$, this is just $(1 + 1 + \dots + 1)/n = 1$, while if $j \neq k$, it is zero by the lemma, since $\omega^{(j-k)}$ is an n^{th} root of unity, but is not equal to 1. It follows that the product is the identity matrix as claimed.

Let

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

be a polynomial of degree $n - 1$, which we identify with a point in \mathbb{C}^n using the map $p \rightarrow (a_0, a_1, \dots, a_n)$. For \mathbf{a} and \mathbf{b} in \mathbb{C}^n , define $\mathbf{a} \star \mathbf{b} \in \mathbb{C}^n$ by

$$(\mathbf{a} \star \mathbf{b})_j = \sum_{k=0}^{n-1} a_{j-k} b_k,$$

where each index in the sum is interpreted mod n , so that for example $a_{-1} = a_{n-1}$ etc. Let $\mathbf{a} = (a_0, a_1, \dots, a_{n-1}, 0, \dots, 0)$ and $\mathbf{b} = (b_0, b_1, \dots, b_{n-1}, 0, \dots, 0)$ be points in \mathbb{C}^{2n} . Then

$$\mathbf{a} \star \mathbf{b} = (a_0b_0, (a_1b_0 + a_0b_1), (a_2b_0 + a_1b_1 + a_0b_2), \dots, (a_{n-1}b_0 + a_{n-2}b_1 + \dots + a_0b_{n-1}), \dots, a_{n-1}b_{n-1}, 0).$$

A calculation shows that if \mathbf{a} and \mathbf{b} are the coefficients of polynomials $p(x)$ and $q(x)$ respectively, then $p(x)q(x)$ has coefficients $\mathbf{a} \star \mathbf{b}$, at least when embedded in \mathbb{C}^{2n} to avoid circular wrap-around.

Let $\mathbf{a} = (a_0, a_1, \dots, a_{n-1})$ be a vector in \mathbb{C}^n , and define \mathbf{F} in \mathbb{C}^n by $\hat{\mathbf{a}} = \mathcal{F}_n(\mathbf{a})$. Since the Fourier matrix is unitary, we have $\mathbf{a} = \bar{\mathcal{F}}_n(\hat{\mathbf{a}})$, and so this transformation, the Discrete Fourier Transform, is invertible. In this context, we need

Convolution Theorem

Let \mathbf{a} and \mathbf{b} be in \mathbb{C}^n . Then

$$\mathcal{F}_n(\mathbf{a})\mathcal{F}_n(\mathbf{b}) = \mathcal{F}_n(\mathbf{a} \star \mathbf{b}).$$

In other words, the discrete Fourier transform maps convolution to pointwise multiplication.

The convolution theorem suggest a different way of computing $\mathbf{a} \star \mathbf{b}$ at least when we embed $\mathbf{a}, \mathbf{b} \in \mathbb{C}^{2n}$; as the inverse transform of $\mathcal{F}_{2n}\mathbf{a} \cdot \mathcal{F}_{2n}\mathbf{b}$. In other words, we first compute the Discrete Fourier Transforms of \mathbf{a} and \mathbf{b} , then compute the pointwise product $\mathcal{F}_{2n}\mathbf{a} \cdot \mathcal{F}_{2n}\mathbf{b}$, and finally compute the inverse Discrete Fourier Transform to obtain $\mathbf{a} \star \mathbf{b}$. It turns out by a careful arrangement of its operations, the Fourier transform can be computed using only $O(n \log n)$ operations. This is known as the Fast Fourier Transform.