School of Computer Science and Engineering, UNSW
COMP 3121/3821/9101/9801
2017
Solutions to midterm problems

1. You have a processor that can operate 24 hours a day, every day. People have submitted n requests to run daily jobs on the processor. Each such job comes with a start time s_i and an finishing time f_i; if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and finishing times (Note that certain jobs can begin before midnight and end after midnight!). Given a list of n such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n. You may assume for simplicity that no two jobs have the same start or end times. Prove that your algorithms is correct and estimate its asymptotic running time.

   Solution This problem is similar to the activity selection problem, except that time is now a 24hr circle, rather than an interval, because there might be jobs whose start time is before the midnight and finishing time after midnight. However, we can reduce it to several instances of the interval case.

   Let $S = \{J_1, J_2, \dots, J_k\}$ be the set of all jobs whose start time is before the midnight and finishing time after midnight. We now first exclude all of these jobs and sort the remaining jobs by their finishing time. We now solve in the usual manner the activity selection problem with the remaining jobs only, by always picking a non-conflicting job with the earliest finishing time. We record thus obtained number of accepted jobs a_0. We then pick job J_1 and then pick among all jobs which are non conflicting with J_1 one which finishes earliest, and continue in such a manner, always picking a non conflicting job which ends earliest and ends before the start of J_1. We record the number of accepted jobs a_1. We proceed with all jobs $J_2, \dots, J_k$ in the same manner recording the number of accepted jobs $a_2, \dots, a_k$. Finally we pick the selection of jobs for which the corresponding a_m, $0 \leq m \leq k$ is the largest.

   To prove optimality, we note that the optimal solution either does not contain any of the jobs from the set S or it contains one of them. If it does not contain any of the jobs from S, then it would have been constructed when the problem was solved for all jobs outside S, by the argument we did in class for the "linear case"; if it does contain say $J_m \in S$, then it would have been constructed during the round when we started with J_m.

   Time complexity: To sort all jobs which are not in the set $J_1, \dots, J_k$ takes at most $n \log n$ steps. Each of $k + 1$ procedures run in linear time (because we go through the list of all jobs by their finishing time only once, checking if their start time is before or after the finishing time of the last chosen activity). The number of rounds is at most n. so the complexity is $O(n \log n + n^2) = O(n^2)$.

2. Assume that you got a fabulous job and you wish to repay your student loan as quickly as possible. Unfortunately, the bank *Western Road Robbery* which gave you the loan has the condition that you must start your monthly repayments by paying off \$1 and

then each subsequent month you must pay either double the amount you paid the previous month, the same amount as the previous month or a half of the amount you paid the previous month. On top of these conditions, your schedule must be such that the last payment is \$1. Design an algorithm which, given the size of your loan, produces a payment schedule which minimises the number of months it will take you to repay your loan while satisfying all of the banks requirements. If optimality of your solution is obvious from the algorithm description, you do not have to provide any further correctness proof.

Solution: Assume the value of your loan is S; we first find the largest n such that $2(1+2+2^2+\ldots+2^n) \leq S$. This amounts to finding the largest n such that $2(2^{n+1}-1) \leq S$. Let $R = S - 2(2^{n+1}-1)$. Since $2(2^{n+2}-1) > S$, we get

$$R = S - 2(2^{n+1}-1) < 2(2^{n+2}-1) - 2(2^{n+1}-1) = 2^{n+3} - 2^{n+2} = 2^{n+2}$$

i.e.,

$$R \leq 2^{n+2} - 1 = 1 + 2 + 2^2 + \ldots + 2^{n+1}$$

You can now return your loan as follows: start with \$1 and keep doubling until you reach 2^n. If $R \geq 2^{n+1}$ double once again; if this is the case let $P = R - 2^{n+1}$; otherwise let $P = R$. Represent P in binary; you now start halving the amount you pay, but you repeat once the amount 2^k whenever the k^{th} bit of P is one.

It should be clear that this produces an optimal solution; for those who like rigorous proofs, one can prove optimality in a very similar way how we did for *giving change* problem with coins $1, c, c^2, c^3, \ldots, c^n$,.

3. (a) Compute the DFT of the sequence $(1, -1, -1, 1)$ by **any method** you wish.
   (b) Compute the linear convolution of sequences $(1, -1, -1, 1)$ and $(-1, 1, 1, -1)$ by **any method** you wish.

   Solution:

   (a) DFT of sequence $(1, -1, -1, 1)$ is just the sequence of values of the associated polynomial $P(x) = 1 - x - x^2 + x^3$ at the roots of unity of order 4, which are $1, i, -1, -i$. Thus $P(1) = 1 - 1 - 1 + 1 = 0$; $P(i) = 1 - i - i^2 + i^3 = 1 - i - (-1) - i = 2 - 2i$; $P(-1) = 1 - (-1) - (-1)^2 + (-1)^3 = 0$ and finally $P(-i) = 1 - (-i) - (-i)^2 + (-i)^3 = 1 + i + 1 + i = 2 + 2i$. Thus, $DFT(1, -1, -1, 1) = (0, 2 - 2i, 0, 2 + 2i)$. Note: you could have chosen to evaluate at the roots of unity ω_4^{-k}, in which case you would have gotten the complex conjugate of the above sequence, i.e., sequence $(0, 2 + 2i, 0, 2 - 2i)$.

   (b) As defined in lecture slides #2, linear convolution of two sequences is just the sequence of the coefficients of the polynomial which is the product of the two polynomials associated with the two sequences. Thus we form $P(x) = 1 - x - x^2 + x^3$ and $Q(x) = -1 + x + x^2 - x^3$ and we simply multiply them by brute force because the polynomials are of such small degree: $P(x)Q(x) = -1 + 2x + x^2 - 4x^3 + x^4 + 2x^5 - x^6$. So linear convolution of these two sequences is $(1, -1, -1, 1) * (-1, 1, 1, -1) = (-1, 2, 1, -4, 1, 2, -1)$.

4. Find the square of a complex number $a + i\,b$ where a and b are real numbers, using only two multiplications and at most 3 additions or subtractions. In other words, you

have to find two real numbers c and d such that $c + i\,d = (a + i\,b)^2$ using only two multiplications and at most 3 additions or subtractions of real numbers.

Solution: $(a + i\,b)^2 = a^2 - b^2 + 2ab\,i = (a + b)(a - b) + (ab + ab)i$. Let $p = a + b$, $q = a - b$ and $r = ab$; so we perform an addition and a subtraction to compute p and q; we then perform two multiplications to compute pq and $r = ab$; finally we perform another addition to get $2r = r + r$; the desired complex number is $pq + 2r\,i$.

Extended classes (comp3821/9801) only:

5. You are given a fair coin, i.e., a coin with equally likely head and tail outcomes. Design an algorithm which uses such a coin to simulate a 6 sided fair dice, i.e., a dice with 6 equally likely outcomes. Compute the expected number of coin tosses needed to obtain an outcome of throwing such a dice once?

   Solution Toss the coin tree times; consider 6 possible outcomes as legitimate and 2 illegitimate, say HHH and TTT are illegitimate; among the legitimate outcomes consider H to stand for 1 and T for 0 to get a 3 bit number between 1 and 6. If you get an illegitimate outcome repeat the experiment of throwing the coin 3 times all over again until you get a legitimate outcome.

   The probability to get a legitimate outcome in the first trial is 6/8, after two trials is $2/8 \cdot 6/8$, after three trials is $(2/8)^2 \cdot 6/8$, etc. Thus the expected total number of coin tosses is

$$\begin{aligned} E &= 3 \cdot \frac{6}{8} + 2 \cdot 3 \cdot \frac{2}{8} \cdot \frac{6}{8} + 3 \cdot 3 \cdot \left(\frac{2}{8}\right)^2 \cdot \frac{6}{8} + 4 \cdot 3 \cdot \left(\frac{2}{8}\right)^3 \cdot \frac{6}{8} + \dots \\ &= 3 \cdot \frac{6}{8}\left(1 + 2 \cdot \left(\frac{2}{8}\right) + 3 \cdot \left(\frac{2}{8}\right)^2 + 4 \cdot \left(\frac{2}{8}\right)^3 + \dots\right) \\ &= 3 \cdot \frac{3}{4}\left(1 + 2 \cdot \left(\frac{1}{4}\right) + 3 \cdot \left(\frac{1}{4}\right)^2 + 4 \cdot \left(\frac{1}{4}\right)^3 + \dots\right) \end{aligned}$$

   The sum in the brackets can be evaluated by the methods we showed in class, see the slides, and is equal 16/9. Thus,

$$E = 3 \cdot \frac{3}{4} \cdot \frac{16}{9} = 4$$

   If you find this surprisingly small, the following gives the probabilities that you will have at least k trials of tossing a coin 3 times, i.e., $P(k) = \frac{3}{4}\sum_{j=k}^{\infty} \frac{1}{4^k}$:

$$P(2) = 0.25 \quad P(3) = 0.0625 \quad P(4) = 0.015625 \quad P(5) = 0.00390625 \quad \dots$$