HW3 - S16

Chris Troiano
CMPS 102
April 19, 2016

1. *Hadamard matrices* $H_0, H_1, H_2, \ldots$ are defined as follows:

   (a) H_0 is the 1×1 matrix $[1]$.

   (b) For $k > 0, H_k$ is the $2^k \times 2^k$ matrix.

$$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}$$

   (a) Show by induction that $(H_k)^2 = 2^k I_k$, where I_k is the identity matrix of dimension 2^k.

   (b) Note that Hadamard matrices are symmetric, i.e. $H_k = H_k^\top$. Thus by the above, $H_k H_k^\top = 2^k I_k$ as well.
   Use this fact for deriving a formula for the dot product between the i-th and j-th row of H_k, for $1 \leq i, j \leq 2^k$.

   Solution:

   (a) Prove $(H_k)^2 = 2^k I_k$
   Base: $H_0 = [1]^2 = [1] \Rightarrow 2^0[1] = 1[1] = 1$ ✓
   Induction: Assume for some integer $k \geq 0, (H_k)^2 = 2^k I_k$. We must show that $(H_{k+1})^2 = 2^{k+1} I_{k+1}$ is also true.

$$(H_{k+1})^2 = \begin{bmatrix} H_k & H_k \\ H_k & -H_k \end{bmatrix}^2 = \begin{bmatrix} 2(H_k)^2 & 0 \\ 0 & 2(H_k)^2 \end{bmatrix} = 2 \begin{bmatrix} (H_k)^2 & 0 \\ 0 & (H_k)^2 \end{bmatrix}$$

   By Inductive Hypothesis

$$2 \begin{bmatrix} 2^k I_k & 0 \\ 0 & 2^k I_k \end{bmatrix} = 2 \cdot 2^k \begin{bmatrix} I_k & 0 \\ 0 & I_k \end{bmatrix} = 2^{k+1} I_{k+1}$$

   (b) $\sum_0^i \sum_0^j i \cdot j \Rightarrow (H_k)^2 = \begin{bmatrix} i_0 \cdot j_0 & i_0 \cdot j_1 \\ i_1 \cdot j_0 & i_1 \cdot j_1 \end{bmatrix}$

2. Consider the Coin Changing problem with the European coin set:

$$\{1, 2, 5, 10, 20, 50, 100, 200\}.$$

   Prove that the Cashier's Algorithm is optimal given the above set of coins. Use the same proof method that was used for the American coin set in class.

   Solution:
   Proof by keeping ahead & contradiction

Consider two sets of coins that have the same value; our set (Greedy algorithm) S, and the optimal set S'.
S : {50, 20, 20...}
r^
S' : {50, 20, 10, 10...}
r^
Both algorithms produce the same coins up until the r^{th} position. If the r^{th} and $r^{th}+1$ position of the S' set of coins are replaced with the r^{th} coin in the S set, the S' will agree with the S set. If we continue this for the entire set, the optimal set will agree with the greedy set. This shows that the greedy set uses fewer coins than the optimal set. This is a contradiction.

3. Given a sorted array of distinct integers $A[1,...,n]$, you want to find out whether there is an index i for which $A[i]=i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$.
Solution:

Algorithm 1 $O(\log n)$ Algorithm for finding whether there is an index i such that $A[i]=i$

```
procedure SEARCH($A, min, max$)
    if $min > max$ then – {return $-1$}
    end if$m = \lceil \frac{(min+max)}{2} \rceil$
    if $A[m] > m$ then – SEARCH($A, min, m-1$)
    end if
    if $A[m] < m$ then – SEARCH($A, m+1, max$)
    end if
    if A[m] == m then – {return $m$}
    end if
end procedure
```

This is essentially the binary search algorithm. Every recursive call, we start off by searching the array from the middle, this effectively decreases the sub-problems by $\frac{1}{2}$. It takes constant time to divide these sub-problems. This gives us the recurrence relation $T(n) = T(\frac{n}{2} = O(1)$
By Master Theorem:
$n^{\log_2^1} = n^0 = 1 = \Theta(n^{\log_2^1} \log n) = O(\log n)$

4. Suppose that you want to multiply the two polynomials $x+1$ and x^2+1 using the FFT.
   (a) Choose an appropriate power of two (the FFT dimension), find the FFT of the two sequences, multiply the resulting sequence componentwise, and then compute the inverse FFT to get the coefficients of the product polynomial. Do the transforms by using matrix vector products as in problem 5 of the previous homework.
   (b) Explicitly compute the product of your polynomial and check your result.

   Solution:

(a) The 4th root of unity is $\omega_4 = e^{\frac{\pi i}{2}} = i$. This gives us the four point FFT.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -1-i \\ 0 \\ -1+i \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -2 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ -1-i \\ 0 \\ -1+i \end{bmatrix} \times \begin{bmatrix} 2 \\ 0 \\ -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}^{-1} \Rightarrow \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(b) $(x+1)(x^2+1) = x^3 + x^2 + x + 1$ $\surd$

5. Given an array of positive integers $A[1, \ldots, n]$, and an integer $M > 0$, you want to partition the array into segments $A[1, \ldots, i_1]$, $A[i_1+1, \ldots, i_2]$, $\ldots$, $A[i_{k-1}+1, \ldots, i_k]$, so that the sum of integers in every segment does not exceed M, while minimizing the number of segments k. You can assume that $M \geq A[i]$ for all i (which guarantees that such a partition exists). Design an $O(n)$ greedy algorithm for solving this problem and prove that it is optimal (i.e. that the obtained partition has the smallest possible number of segments).

   Solution:

Algorithm 2 $O(n)$ Algorithm for partitioning an array into segments, such that the sum of integers in each segment does not exceed M

```
procedure PARTITION($A, M$)
    $j = 0$
    $sum = 0$
    while $j < A.length$ AND $sum \leq M$ do
        $sum += A[j]$
        if $sum > M$ then – { The partition has been found }
            $i_k = j - 1$
            $sum = 0$
        end if
        j++
    end while
end procedure
```

This algorithm runs in $O(n)$ time because the array is already sorted, we simply iterate through the entire array from $A[1...n]$ once. Thus producing an $O(n)$ time complexity.

Proof by keeping ahead and contradiction:

Consider two arrays, one partitioned with a greedy algorithm A, and the other with an optimal algorithm A'.
Both arrays agree on partitions up to the i_k partition. The sum of values from $A'[i_k...i_{k+1}]$ is less than the sum of $A[i_k...i_{k+1}]$. To improve the optimal algorithm, we simply move the A''s i_{k+1} index to match A's i_{k+1} index. This will continue until A''s partitions match A's. This shows that the optimal algorithm uses more partitions than greedy, and thus gives a contradiction.

6. EC: Consider the Coin Changing problem with the following coin set:

$$\left\{1, a_1, a_1a_2, \ldots, \prod_{i=1}^{k} a_i\right\},$$

where $a_1, \ldots, a_k$ are integers greater than 1.

   (a) Prove that the Cashier's Algorithm is optimal given the above set of coins.

   (b) Discuss how the above is related to the fact that every natural number has a unique representation base k, for any positive integer $k \geq 2$.

Solution:
base: The a_1 coin value is equal to $1 \cdot a_1$. That means, if we have a_1 "pennies", then we have the same amount as a single a_1 coin. Instead of using a_1 "pennies", the cashier's algorithm tells us to simply use one a_1 coin. By induction, this will be true for the a_k and a_{k+1} case.