CSCE 222-200 Discrete Structures for Computing

Fall 2024

Instructor: Dr. Jianer Chen
Office: PETR 428
Phone: (979) 845-4259
Email: `chen@cse.tamu.edu`
Office Hours: T+R 2:00pm–3:30pm

Teaching Assistant: Evan Kostov
Office: EABC Cubicle 6
Phone: (469) 996-5494
Email: `evankostov@tamu.edu`
Office Hours: MW 4:00pm-5:00pm

Assignment #4 Solutions

1. Give a recursive algorithm for finding the maximum value in an array $A[1..n]$, making use of the fact that the maximum in $A[1..n]$ is the larger of $A[n]$ and the maximum in $A[1..n-1]$. What is the time complexity of your algorithm in terms of big-O notation?

Solution. The recursive algorithm Largest(k) is given as follows.

```
Largest($k$) //return the largest value in $A[1..k]$
1. if ($k < 1$) return ('the array is empty');
2. if ($k == 1$) return($A[1]$);
3. $m$ = Largest($k-1$);
4. if ($m > A[k]$) return ($m$) else return ($A[k]$).
```

The main algorithm calls Largest(n), which returns the largest value in $A[1..n]$.

To get the time complexity of the algorithm, let $T(k)$ be the running time of the algorithm Largest(k) on the array $A[1..k]$ of k elements. Steps 1, 2, and 4 take time $O(1)$, and step 3 takes time $T(k-1)$. Thus, the recurrence relation for $T(k)$ is

$$T(1) = O(1), \qquad \text{and} \qquad T(k) = T(k-1) + O(1) \ \text{ for } k > 1.$$

To solve this recurrence relation, we first convert it to

$$T(1) \le C, \qquad \text{and} \qquad T(k) \le T(k-1) + C \ \text{ for } k > 1.$$

Replacing k in the inequality $T(k) \le T(k-1) + C$ with $k-1$ and $k-2$, we get

$$\begin{aligned} T(k) &\le T(k-1) + C \le (T(k-2) + C) + C = T(k-2) + 2C \\ &\le (T(k-3) + C) + 2C = T(k-3) + 3C. \end{aligned}$$

From this, it is easy to verify (and formally prove) that for a general $h < k$, we have $T(k) \le T(k-h) + h\cdot C$. Letting $h = k-1$, we get $T(k) \le T(1) + (k-1)C \le k\cdot C$. Since C is a constant, we get $T(k) = O(k)$. In conclusion, the running time of the main algorithm on the array $A[1..n]$ of n values is $O(n)$.

2. Consider the following two recursive algorithms for computing the function $H(n)$ for Hanoi Tower, where n is a positive integer and the function $H(n)$ is defined as $H(1) = 1$, and $H(n) = 2H(n-1) + 1$ for $n > 1$. What is the time complexity of each of these algorithms in terms of big-O notation?

```
Function $H_1(n)$
1. **if** ($n == 1$) **then** return(1);
2. $h = H_1(n-1) + H_1(n-1) + 1$;
3. return($h$).
```

```
Function $H_2(n)$
1. **if** ($n == 1$) **then** return(1);
2. $h = 2 \cdot H_2(n-1) + 1$;
3. return($h$).
```

Solution. First consider the algorithm $H_1(n)$. Let $T_1(n)$ be the running time of the algorithm $H_1(n)$ on n disks. Since step 2 of the algorithm $H_1(n)$ (recursively) calls the algorithm $H_1(n-1)$ twice, we have the following recurrence relation:

$$T_1(1) = O(1), \qquad \text{and} \qquad T_1(n) = 2T_1(n-1) + O(1) \text{ for } k > 1.$$

Converting it into

$$T_1(1) \leq C, \qquad \text{and} \qquad T_1(n) \leq 2T_1(n-1) + C \text{ for } k > 1,$$

and replacing n in the inequality $T_1(n) \leq 2T_1(n-1) + C$ with $n-1$ and $n-2$ give

$$\begin{aligned} T_1(n) &\leq 2T_1(n-1) + C \leq 2\left(2T_1(n-2) + C\right) + C = 2^2 T_1(n-2) + 2C + C \\ &\leq 2^2\left(2T_1(n-3) + C\right) + 2C + C = 2^3 T_1(n-3) + 2^2 C + 2C + C. \end{aligned}$$

From this, it is easy to formally prove that for a general $k < n$, we have

$$T_1(n) \leq 2^k T_1(n-k) + (2^{k-1} + 2^{k-2} + \cdots + 2 + 1)C = 2^k T_1(n-k) + (2^k - 1)C.$$

Letting $k = n-1$, we get

$$T_1(n) \leq 2^{n-1} T_1(1) + (2^{n-1} - 1)C \leq 2^{n-1} C + (2^{n-1} - 1)C = (2^n - 1)C = O(2^n),$$

where the last equality is because C is a constant. In conclusion, the running time of the algorithm $H_1(n)$ on n disks is $O(2^n)$.

Similarly, we can find the time complexity $T_2(n)$ of the algorithm $H_2(n)$. The major difference here is that instead $H_2(n)$ recursively calls $H_2(n-1)$ only once. Thus, from the recurrence relation

$$T_2(1) \leq C, \qquad \text{and} \qquad T_2(n) \leq T_2(n-1) + C \text{ for } k > 1,$$

replacing n with $n-1$ and $n-2$ gives $T_2(n) \leq T_2(n-3) + 3C$, from which we get, for a general $k < n$, $T_2(n) \leq T_2(n-k) + k \cdot C$. Finally, letting $k = n-1$ gives

$$T_2(n) \leq T_2(1) + (n-1)C \leq C + (n-1)C = n \cdot C = O(n).$$

In conclusion, the running time of the algorithm $H_2(n)$ on n disks is $O(n)$.

We remark that this question shows the significance of developing "good" algorithms for a problem. Under the current computer power, algorithm $H_2(n)$ of running time $O(n)$ runs in less than a second for very large n (e.g., n is a million), while algorithm $H_1(n)$ of running time $O(2^n)$ would take several weeks even for small values of n such as $n = 50$ (see Question 3 in Homework #3).

3. In the following, $n \geq 2$ is an integer.

(a) How many different binary strings are there of length n?

(b) How many different binary strings are there of length not larger than n?

(c) How many different binary strings are there of length n that start with 1?

Solution.

(a) There are 2^n different binary strings of length n. This can be proved by induction on n: for $n = 1$, there are exactly $2 = 2^1$ binary strings of length 1, i.e., 0 and 1. Assume the statement is true for $n - 1$. Now consider binary strings of length n. Consider all binary strings of length n whose first bit is 0. Each of these binary strings is given by the leading bit 0 followed by a binary string of length $n - 1$. By the inductive hypothesis, there are 2^{n-1} different binary strings of length $n - 1$, of which each plus the leading bit 0 gives a binary string of length n with the leading bit 0. Thus, there are 2^{n-1} binary strings of length n whose leading bit is 0. Similarly, there are 2^{n-1} binary strings of length n whose leading bit is 1. Since every binary string of length n has a leading bit either 0 or 1, and no two binary strings can be the same if they have different leading bits, we conclude that there are exactly $2^{n-1} + 2^{n-1} = 2^n$ binary strings of length n. This completes the inductive proof for the conclusion of our answer..

(b) The set of binary strings of length not larger than n consists of binary strings whose length can be any of the values k for $1 \leq k \leq n$. As proved in (a), for each $1 \leq k \leq n$, there are exactly 2^k binary strings of length k. Therefore, the number of binary strings whose length is not larger than n is equal to

$$\sum_{k=1}^{n} 2^k = 2^{n+1} - 2.$$

(c) As we have discussed in (a), each binary string of length n with a leading bit 1 corresponds to a different binary string of length $n - 1$. By (a), there are 2^{n-1} different binary strings of length $n - 1$. We conclude that there are 2^{n-1} different binary string of length n that starts with 1.

4. Give a formal proof for the following statement: for 145 people of different heights standing in a line, it is always possible to find 13 people (not necessarily consecutively) in the order they are standing in the line with heights that are either increasing or decreasing.

Proof. The proof follows the same one we gave in class for general n: a sequence of $n^2 + 1$ distinct numbers contains either an increasing subsequence of length $n + 1$ or a decreasing subsequence of length $n + 1$.

Let $\langle a_1, a_2, a_3, \ldots, a_{144}, a_{145} \rangle$ be the heights of the people, in the order they are standing in the line. Assume the contrary that there are no 13 people in the line whose heights are either increasing or decreasing in the order they are standing in the line. Thus, for each person a_k in the line, we assign a_k a pair (i_k, d_k), where i_k and d_k, respectively, are the length of the longest increasing subsequence and longest decreasing subsequence starting from a_k in $\langle a_k, a_{k+1}, \ldots, a_{145}\rangle$.

By the assumption, $1 \leq i_k, d_k \leq 12$ (note that both i_k and d_k are at least 1 because a_k by itself is an incresing and decreasing subsequence of length 1). Thus, there are at most 12 different values for each of i_k and d_k, so there are at most $12 \cdot 12 = 144$ different pairs of the form (i_k, d_k). Since there are totally 145 people in the line, by the Pigeonhole principle, there are two different people a_h and a_k in the line, $h < k$, such that the pairs (i_h, d_h) and (i_k, d_k) are the same (i.e., $i_h = i_k$ and $d_h = d_k$). However, this leads to a contradiction (note all a_k are distinct):

(1) if $a_h < a_k$, then a_h plus the increasing subsequence of length i_k starting from a_k gives an increasing subsequence of length $i_k + 1 = i_h + 1$ starting from a_h, contradicting the definition that i_h is the length of the longest increasing subsequence from a_h.

(2) if $a_h > a_k$, then a_h plus the decreasing subsequence of length d_k starting from a_k gives a decreasing subsequence of length $d_k + 1 = d_h + 1$ starting from a_h, contradicting the definition that d_h is the length of the longest decreasing subsequence from a_h.

By the principle of proof by contradiction, this proves that there are 13 people in the line whose heights are either increasing or decreasing in the order they are standing in the line.

5. A *circular permutation* of n people is a seating of the n people around a circular table, where seatings are considered to be the same if they can be obtained from each other by rotating the table. How many different circular permutations are there for n people? Give an explanation on your answer.

Solution. Let the n people be p_1, p_2, …, p_n, and let the table seats be labeled $1, 2, \ldots, n$. Since two seatings are regarded the same if one can be obtained from the other by rotating the table, we can assume that in all seatings, person p_1 is always sitting at seat 1 (otherwise, we can rotate the table to move p_1 to seat 1).

Therefore, a seating of the n people is to assign the $n-1$ people p_2, p_3, …, p_n to the $n-1$ seats $2, 3, \ldots, n$. There is an obvious one-to-one correspondence between the set of permutations of the $n-1$ people p_2, p_3, …, p_n and the set of seatings of the n people: a permutation $\langle p_{i_2}, p_{i_3}, \ldots, p_{i_n}\rangle$ of the $n-1$ people p_2, p_3, …, p_n corresponds to the seating that assigns person p_1 to seat 1, and assigns person p_{i_j} to seat j for all $2 \leq j \leq n$. As a result, the number of different circular permutations of the n people p_1, p_2, …, p_n, i.e., the number of different seatings of the n people p_1, p_2, …, p_n around the circular table, is equal to the number of permutations of the $n-1$ people p_2, p_3, …, p_n, which is $(n-1)!$.

An interesting extension of the problem. If we assume the round table has m seats, where $m \geq n$, then how many different seatings are there for n people?