COMP 775 - Advanced Design and Analysis of Algorithms Assignment 3

Jordan Sturtz

3-17-2023

Problem 4-5

Question Professor Diogenes has n supposedly identical integrated-circuit chips that in principle are capable of testing each other. The professor's test jig accommodates two chips at a time. When the jig is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the professor cannot trust the answer of a bad chip. Thus, the four possible outcomes of a test are as follows:

Chip A says	Chip B Says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

a. Show that if at least n/2 chips are bad, the professor cannot necessarily determine which chips are good using any strategy based on this kind of pairwise test. Assume that the bad chips can conspire to fool the professor

Answer:

We will solve with mathematical induction.

- **Base Case**

  Suppose there are only two chips, at least half of which are defective, i.e. one or both are defective.

  There are two possibilities.

  (a) Suppose both chips are defective. In that case, either both chips say that the other is good, or at least one says the other is bad. If both say that they are good, then the professor can conclude only that *both are good or both are bad.* If at least one chip says the other is bad, then the professor can conclude only that *at least one is bad.*

  (b) Suppose only one chips is defective. In that case, the professor can conclude only that *at least one is bad.*

  In either case, the professor will conclude from this test either that *both are good or both are bad* OR that *at least one is bad.* In other words, for $n = 2$, the professor cannot determine which among his chips are good. This proves the base case.

- **Recursive Case** Now suppose the professor is already uncertain about n chips, at least half of which are defective where $n \geq 2$. If another chip was added such that $\lceil \frac{n+1}{2} \rceil$ chips are defective, can the professor determine which among the $n + 1$ chips are good?

  The professor already cannot determine from the n chips which are defective. Thus, if the professor cannot determine the defectiveness of the new chip, then they cannot determine the defectiveness of all $n + 1$ chips.

  The newly added chip is either defective or it isn't:

  (a) Suppose the new chip is not defective. If the chip is not defective, the professor can attempt to pair this chip with one of their chips from the n chips of indeterminate defectiveness:

    i. Suppose the second chip chosen from n is also not defective. Then, the professor can only conclude that *both are good or both are bad.*

    ii. Suppose instead the second chip chosen from n is defective. Then, the professor can only conclude that either *both are good or both are bad* or *at least one is bad.*

Therefore, if the new chip is not defective, the professor once again will not be able to determine whether this is so.

(b) Now suppose the new chip is defective. In that case, regardless of what it is paired with, the professor will either conclude that *both are good or both are bad* or *at least one is bad*. In other words, in this case as well, the professor cannot determine whether the new chip is defective.

Therefore, professor will be unable to determine the defectiveness of the new chip. Since the professor cannot determine the defectiveness of the new chip, we can conclude the recursive case: that if the professor is unsure about n chips where $\lceil \frac{n}{2} \rceil$ chips are defective, then the professor is also unsure about $n+1$ chips where $\lceil \frac{n+1}{2} \rceil$ are defective.

Since the professor cannot determine the defectiveness in the base case, i.e. with two chips, and the professor cannot determine the defectiveness in the recursive case, then it is hopeless. There is no n for which the professor will be able to determine the defectiveness of his chips.

b. Consider the problem of finding a single good chip from among n chips, assuming that more than n/2 of the chips are good. Show that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.

Answer:

We need a method to wittle down the size of n while retaining the key property that more than half of n are good chips.

What we do is that we arbitrarily match each chip in n with another chip in n. If there is an odd number of chips, then we set aside one chip from this pairing. Since we stipulate that more than half are good chips, if we do this one-to-one matching and we have odd chips, it must be the case if we leave out the one-off chip that the number of chips in the pairings are at least half good.

The goal is to eliminate chips from consideration. When we pair these chips and pairwise test them, there are two possibilities and two different rules for removing chips from consideration:

- At least one chip reports that the other is bad. This means that *at least one is bad*. If at least one chip is bad, then if we ignore these two chips in subsequent comparisons, i.e. **remove both chips from consideration completely**, then the rest of the chips must also still contain at least half good chips.
- Both chips agree that the other is good. In this case, there are two possibilities. Either both chips are good, or both are bad. Either way, we know that these chips share the same state, so we lose no information by keeping both chips in consideration. So, **we discard one chip from this pairwise test**.

After doing this for our initial pairs, we will have some subset of our chips left, i.e. those that agreed with each other that the other is good. Moreover, this subset is guaranteed to have at least half good chips. We recursively apply this rule until we have one chip or no chips remaining.

In the case of this algorithm for the odd-numbered n, there are two possibilities. The first is that we end up by chance pairing up every remaining good chip with a bad chip, thus causing all the pairs to be discarded, leaving just the odd-one out. In that case, the odd-one out must be good, since we stipulated that more than half the chips are good. The second possibilitiy is that by the end of the recursion, we have one remaining chip in our pairs, in which case that chip must be good, since once again we stipulated that more than half the chips are good. Either way, we can know with certainty at least one good chip.

The reason that $\lfloor n/2 \rfloor$ is sufficient to reduce the problem to one of nearly half the size is that during the recursion we eliminate at least one chip from consideration for each pairwise test, so each pass over the data in the worst case halves the number of possibilities.

c. Show that the good chips can be identified with $\Theta(n)$ pairwise tests, assuming that more than n/2 of the chips are good. Give and solve the recurrence that describes the number of tests.

Answer:

Once we find a single good chip, it can be used with $n-1$ more pairwise tests to identify all the good and bad chips. Thus, we need only apply the solution discovered in part (b), then check that good chip against every other chip.

To see that runtime is $\Theta(n)$ pairwise tests, consider that during the recursion, each pass over the data eliminates at least half of the possibilities and that the number of steps to perform during the recursion is $\lfloor n/2 \rfloor$. Thus, we can express the recurrence relation as such:

$$T(N) \leq T(\lceil N/2 \rceil) + \lfloor n/2 \rfloor$$
$$T(1) = \Theta(1)$$

By the master theorem, case 2, $T(n)$ must be $O(n)$. Once we have identified the good chip, we need only perform n - 1 more tests, so the total time is $O(n) + n - 1$, which is $\Theta(n)$.

Problem 4-6

Question An $m \times n$ array A of real numbers is a **Monge array** if for all i, j, k, and l such that $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$, we have

$$A[i,j] + A[k,l] \leq A[i,l] + A[k,j]$$

In other words, whenever we pick two rows and two columns of a Monge array and consider the four elements at the intersections of the rows and the columns, the sum of the upper-left and lower-right elements is less than or equal to the sum of the lower-left and upper-right elements. For example, the following array is Monge:

10	17	13	28	23
17	22	16	29	23
24	28	22	34	24
11	13	6	17	7
45	44	32	37	23
36	33	19	21	6
75	66	51	53	34

a. Prove that an array is Monge if and only if for all $i = 1, 2, ..., m - 1$ and $j = 1, 2, ..., n - 1$, we have

$$A[i,j] + A[i+1,j+1] \leq A[i,j+1] + A[i+1,j]$$

(*Hint*: For the "if" part, use induction separately on rows and columns.)

Answer:

First, we prove that if for all $i = 1, 2, ..., m - 1$ and $j = 1, 2, ..., n - 1$, then array A (shown above) is Monge.

By definition, i must be positive. Therefore, $i+1$ must be greater than i. Let $k = i+1$. Likewise, since j is positive, $j+1$ must be greater than j. Let $l = j+1$. By substituting into the above expression, we get the definition of a Monge.

Now, we prove converse that if an array A is Monge, then for all $i = 1, 2, ..., m - 1$ and $j = 1, 2, ..., n - 1$, then array shown above is true.

If we substitute $i+1$ for k and $j+1$ for l, then we get the definition above, which shows that if A is Monge, then the above expression holds true. We have therefore shown both sides of the biconditional expression, which proves the theorem.

b. The following array is not Monge. Change one element in order to make it Monge (*Hint:* Use part (a).)

37	23	22	32
21	6	7	10
53	34	30	31
32	13	9	6
43	21	15	8

Answer:

37	23	**24**	32
21	6	7	10
53	34	30	31
32	13	9	6
43	21	15	8

c. Let $f(i)$ be the index of the column containing the leftmost minimum element of row i. Prove that $f(1) \leq f(2) \leq \cdots \leq f(m)$ for any $m \times n$ Monge array.

Answer:

From the theorem proved in (a), we can say that:

$$A[i, f(i)] + A[i+1, f(i)+1] \leq A[i+1, f(i)] + A[i, f(i)+1]$$

In other words, whatever the index column of the smallest element in row i, it will be the case that the Monge property holds for the next index.

Now if it's true that $f(i)$ is the index of the column containing the leftmost minimum element of row i, then the following is also true:

$$A[i, f(i)] \leq A[i, f(i)+1]$$

In other words, the value immediately to the right of $A[i, f(i)]$ is obviously larger or equal if $A[i, f(i)]$ is the smallest minimum value in row i.

We can then subtract $A[i, f(i)]$ from the left-hand side of the above equation and subtract $A[i, f(i)+1]$ from the right-hand side, since this will maintain the truth of the inequality:

$$\begin{gathered} A[i, f(i)] + A[i+1, f(i)+1] \leq A[i+1, f(i)] + A[i, f(i)+1] \\ A[i, f(i)] + A[i+1, f(i)+1] - A[i, f(i)] \leq A[i, f(i)+1] + A[i+1, f(i)] - A[i, f(i)+1] \\ A[i+1, f(i)+1] \leq A[i+1, f(i)] \end{gathered}$$

This statement, $A[i+1, f(i)+1] \leq A[i+1, f(i)]$, proves our theorem, since it says in the next row, the value to the right of $f(i)$ must be less than or equal to the value at $f(i)$, which is to say that the index of the minimum smallest value in the next row must be greater than or equal to the index in the current row. In other words, for any $f(i)$, $f(i) \leq f(i+1)$.

d. Here is a description of a divide-and-conquer algorithm that computes the leftmost minimum element in each row of an $m \times n$ Monge array A:

- Construct a submatrix A' of A consisting of the even-numbered rows of A. Recursively determine the leftmost minimum for each row of A'. Then compute the leftmost minimum in the odd-numbered rows of A.

Explain how to compute the leftmost minimum in the odd-numbered rows of A (given that the leftmost minimum of the even-numbered rows is known) in $O(m+n)$.

Answer:

Once the indices of the minimum values in the even-numbered rows are known, we do not have to check all columns for the odd-numbered rows. We know this because of the theorem we proved in part (c). In fact, the only columns we need to check for row i are the columns between $f(i-1)$ and $f(i+1)$, since the theorem in part (c) proved that $f(i-1) \leq f(i) \leq f(i+1)$ where $i > 1$ and $f(i)$ is the function of the ith row that returns the index of the minimum leftmost value in row i.

Let even_minimums contain the leftmost index columns, [f(0), f(2), f(4)...f(m)], for an $m \times n$ matrix. A simple algorithm to find the minimums for the odd rows, [f(1), f(3), f(5)...] is the following:

Algorithm 1 get_odd_mins(A, even_mins)

```
1: odds ← odd rows from A
2: odd_mins ← []
3: for i, row in enumerate(odds) do
4:    colstart ← even_mins[i]
5:    colend ← even_mins[i+1]
6:    mincol ← colstart
7:    min ← row[mincol]
8:    for i in range(colstart, colend) do
9:       if row[i] < min then
10:         mincol ← i
11:         min ← row[i]
12:      end if
13:   end for
14:   odd_mins ← odd_mins + [mincol]
15: end for
16: return odd_mins
```

In other words, we only check the columns between $f(i-1)$ and $f(i+1)$ since the smallest value must be between those columns.

The for loop iterates over $\lfloor m/2 \rfloor$ elements, doing constant time operations until we hit the inner for loop. This inner for-loop will iterate over a certain subset of columns. Specifically, each for-loop will iterate over a unique subset of columns, since the column position of the minimum value in each row is "sandwhiched" between two columns. In other words, if we sum the number of loops over columns across the entire outer for loop, we would find a total of n steps. Thus, the outer for loop can be thought of as adding $\lfloor m/2 \rfloor$ steps plus n steps, which is to say, $O(m+n)$.

e. Write the recurrence describing the running time of the algorithm described in part (d). Show that its solution is $O(m + n \log m)$

Answer:

The algorithm described in part (d) involves finding the minimums of the evens, then finding the minimums of the odds. The running time to find the minimum of the odds is $O(m+n)$, thus we need only to find the running time to find the minimum of the evens. The algorithm involves a simple divide-and-conquer approach, which halves the number of elements in the matrix. Thus, our recurrence is the following:

$$T(m) = T(m/2) + \Theta(m+n)$$
$$T(m) = T(m/2) + mc_1 + nc_2$$

First, we unroll with different inputs:

$$\begin{aligned} T(m) &= T(m/2) + mc_1 + nc_2 \\ T(m/2) &= T(m/4) + \frac{m}{2}c_1 + nc_2 \\ T(m/4) &= T(m/8) + \frac{m}{4}c_1 + nc_2 \\ &\dots \end{aligned}$$

We substitute recursively to unroll $T(n)$:

$$\begin{aligned} T(m) &= mc_1 + nc_2 + T(m/2) \\ &= \frac{m}{2}c_1 + mc_1 + nc_2 + nc_2 + T(m/2^2) \\ &= \frac{m}{4}c_1 + \frac{m}{2}c_1 + mc_1 + nc_2 + nc_2 + nc_2 + T(m/2^3) \\ &= mc_1 \sum_{i=0}^{k-1} \frac{1}{2^k} + knc_2 + T(m/2^k) \\ &= mc_1(2 - \frac{2}{2^k}) + knc_2 + T(m/2^k) \\ &= mc_1(2 - \frac{2}{m}) + \log_2 mnc_2 + T(1) \\ &\leq mc_1 + nc_2 \log_2 m + \Theta(1) \end{aligned}$$

Thus, the running time of $T(m)$ is $O(m + n \log m)$.

Leetcode 1

Answer: See leetcode1.txt

Leetcode 2

Answer: See leetcode2.txt