ASSIGNMENT 3 – COMP 252

Alexandre St-Aubin and Dylan Telio

February 9, 2024

1. DECISION TREE LOWER BOUND.

   (i) Give a simple algorithm for this problem and an upper bound on the complexity. For n = 5, your answer should be 7.

   Solution:

Algorithm 1: An algorithm to classify the n monkeys.

```
Input: An array Monkeys of monkeys.
Output: Three arrays separating the monkey species.
Species_1 ← new Array;
Species_2 ← new Array;
Species_3 ← new Array;
$i \leftarrow 1$;
Species_1.append(Monkeys[0]);
while Monkeys[0] = Monkeys[$i$] do
    Species_1.append(Monkeys[$i$]);
    $i++$;
// loop above exits only if a second monkey species was found
Species_2.append(Monkeys[$i$]);
// for each remaining monkey, compare to the 2 known species
for $j$ in range $[i,n]$ do
    if Monkeys[$j$] = Species_1[0] then
        Species_1.append (Monkeys [j]);
    else if Monkeys[$j$] = Species_2[0] then
        Species_2.append (Monkeys [j]);
    else
        Species_3.append (Monkeys [j]);
```

Remark. The worst case for the above algorithm is when the two first monkeys are different, and each remaining monkey is of the third or second species, hence must be compared twice (once to each of the first two species), in order to be classified. This yields an upper bound complexity of $1+2(n-2)$. When $n=5$, this is indeed equal to 7.

(ii) Show that all leaves in the decision tree must be different (i.e., correspond to different outputs).

Proof. Fix a sequence of monkeys. The decision tree representing the algorithm described in part (i) is binary, where each decision takes as input two monkeys, and outputs whether they are of the same species or not. Every monkey must be compared at most twice to be classified. We can represent each comparison by a bit, 1 being a match, and 0 otherwise. The output for a given monkey will therefore be:

$$\begin{cases} 10 & \text{if match with Species\_1} \\ 01 & \text{if match with Species\_2} \\ 00 & \text{if match with Species\_3} \end{cases}$$

We note that in reality, if there is a match with Species_1, the monkey will not be compared with the other species, as it would be useless. So, the notation 10 is not entirely representative of what truly happens in the algorithm, it should rather be 1, as only one comparison is made. However, for sake of notation, we keep it as 10.

Hence, one easily sees that each leaf in the decision tree can be represented as a binary sequence of comparisons, where each 2 consecutive bits represents the classification of a monkey. In order to relax notation, let $10 = 1$, $01 = 2$, $00 = 3$, then each leaf is given as a sequence of the form

$$1233322123123121$$

We show by induction that each leaf represents a unique monkey sequence. The base case is clear when $n = 2$. Now, assume that for $n-1$ monkeys, each leaf is represented by a unique sequence of the numbers $\{1,2,3\}$ of length $n-1$. Then, if we add a monkey to the end of the array of monkeys, for each existing sequence $\{a_i\}_{i=1}^{n-1}$, 3 sequences of length n will be created:

$$a_{i,1} := \{a_i\}_{i=1}^{n-1} \cup \{a_n = 1\}$$
$$a_{i,2} := \{a_i\}_{i=1}^{n-1} \cup \{a_n = 2\}$$
$$a_{i,3} := \{a_i\}_{i=1}^{n-1} \cup \{a_n = 3\}$$

Clearly the 3 new sequences are unique for each i, and since the $n-1$ first elements of every sequence are unique by the induction hypothesis, it follows that all of the sequences representing the decision tree for n monkeys are unique. Hence, the leaves represent unique outputs.

□

(iii) Using a decision tree-based method, show that the lower bound for this problem is 6 when $n = 5$.

Proof. The algorithm outputs three sets of monkeys of identical species. For each monkey, there are three possibilities of species, hence 3^5 total permutations. Yet, this accounts for the order of the species, which we don't care about. That is, it shouldn't matter whether the *popa langurs* are grouped in Species_1, Species_2, or Species_3. To remove this unwanted notion of order, we divide by 3!. It follows that the total number of possible outputs for this problem is given by

$$\frac{3^n}{3!}$$

By part (ii), each leaf corresponds to a different output, so we have a tree with $\frac{3^n}{3!}$ leaves. Thus, by the method of decision trees, the lower bound is

$$\left\lceil \log_2\left(\frac{3^5}{3!}\right)\right\rceil = 6$$

In reality, we might be removing too many leaves by dividing by 3!, as not every grouping of monkeys is seen 3! times, i.e. $(3,1,1)$ occurs only 3 times, not $3! = 6$. However, this doesn't matter, as by assuming less leaves, we achieve a lower lower bound. □

(iv) For general n, show that the decision tree lower bound is at least $an-b$ and at most $an+b$ for $n \geq n_0$, for some positive numbers a, b, n_0. Determine suitable values for these numbers. Compare this with your answer in *(i)* and conclude that the decision tree bound for this problem is rather weak.

Solution:
For a general n, the number of leaves in the decision tree is upper bounded by 3^n, because there are 3 possible groups that each monkey can be placed in. However, two leaves may correspond to the same permutation of monkeys, which will result in less than 3^n leaves, but it remains an upper bound. A lower bound for the number of leaves is $\frac{3^n}{3!}$, which assumes that for each grouping of monkeys, all permutations of this outcome appear as a unique leaf in the decision tree. With at most 3^n leaves, the decision tree lower bound is at most

$$\lceil \log_2(3^n)\rceil = \lceil n\log_2 3\rceil \leq n\log_2 3 + \log_2 6$$

With at least $\frac{3^n}{3!} = \frac{3^n}{6}$ leaves, the decision tree lower bound is at least

$$\left\lceil \log_2 \frac{3^n}{6}\right\rceil = \lceil n\log_2 3 - \log_2 6\rceil \geq n\log_2 3 - \log_2 6,$$

thus $a = \log_2 3$ and $b = \log_2 6$, and for the decision tree lower bound (DCB), we have

$$an - b = n\log_2 3 - \log_2 6 \leq \text{DCB} \leq n\log_2 3 + \log_2 6 = an + b$$

This decision tree bound is rather week, as for $n = 5$, it is at most $5\log_2 3 + \log_2 6$, which is approximately 10.5, i.e., way more than the algorithm found in part (i) which had a complexity of 7.

2. ADVERSARIAL LOWER BOUND. Using the method of adversaries, show that the lower bound for this problem is 7 when $n = 5$.

Solution:
ADVERSARY STRATEGY: The adversary always answers NO, unless it has to answer YES.

Claim: This strategy always yields at least 7 comparisons.

Proof. Denote the set of monkeys by $\Omega := \{a, b, c, d, e\}$. There are two cases where the adversary must answer "yes" to a comparison between two monkeys. Without loss of generality, let $a \in$ Species_1 and $x \in \Omega$, then the adversary will be forced to answer yes if,

(i) x was already compared to Species_2 and Species_3, and the adversary answered "no" to both.

(ii) x was already compared to Species_1 and the adversary answered "yes".

Notice that in either case, the algorithm doesn't gain any new information. We may therefore assume that the adversary will never be forced to answer "yes", as any good algorithm would not waste comparisons to obtain information it already knows.

Therefore, our claim is that no matter the comparisons asked (except those that are useless, as above), answering "no" to each will always require at least 7 comparisons before the 5 monkeys are classified. We prove this using *brute force*, i.e. by trying to find a way for the 5 monkeys to be correctly classified in 6 comparisons, failing to do so, and concluding that at least 7 will be required. Since classifying the first 2 monkeys always takes just one comparison, the remaining 3 monkeys must therefore be classified with 5 comparisons. Let,

(a) $X := \{a, b\}$ be the two monkeys compared at first;

(b) $Y := \{c, d, e\}$ the 3 remaining monkeys that need to be classified.

We call *internal* the comparisons that are made within Y, and *external* those that are made between X and Y. Making only external comparisons is equivalent to the algorithm in (i), and clearly yields 6 additional comparisons (2 for each of $\{c, d, e\}$). So, we try and make internal comparisons. Without loss of generality[1], there are 3 possible strategies:

(i) 3 internal comparisons: $c\&d$, $d\&e, e\&c$ to obtain that they are all different species. But then at least 3 external comparisons will be required to classify them: $c\&a$, $c\&b$, $d\&a$ (recall that each comparison returns "not equal" by our chosen adversary).

(ii) 2 internal comparisons: $c\&d$, $d\&e$ to get that $c \neq d$, $d \neq e$ and $e?d$. Then, 4 external comparisons must be made: $c\&a$, $c\&b$, $d\&a$, $e\&a$.

(iii) 1 internal comparison: $c\&d$, similarly requires 5 external comparisons with a, b.

Hence, the claim holds and by the adversary method, we conclude that the lower bound for the problem is 7 when $n = 5$. □

[1]The reason why it's *wlog* is that the order that the comparisons are made does not matter, and nor does which monkeys they are made between (by relabelling).

3. LOWER BOUNDS BY THE METHOD OF ADVERSARIES, AND AN ALGORITHM. Let the data consist of an $n \times n \times n$ matrix of different integers that are sorted along all three axes from small to large. We have a ternary comparison oracle at our disposal. For a given integer x, we are asked to verify whether x is an element of the matrix. The only comparisons that are allowed are between x and matrix elements.

   (a) Using the method of adversaries, derive a lower bound (called $L(n)$) for the worst-case number of times the oracle has to be consulted to perform a successful search for x.

   Solution:
   We first define an adversary strategy for the $2D$ case. Suppose you have a $n \times n$ ordered matrix of the form:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} \end{pmatrix}$$

   Where $a_{1,1}$ is the smallest entry and $a_{n,n}$ the greatest.

   2D ADVERSARY STRATEGY: when asked to compare x with the entry $a_{i,j}$, the adversary will answer :

   i. $(x > a_{i,j})$, if $i + j \leq n + 1$;
   ii. $(x < a_{i,j})$, if $i + j > n + 1$;
   iii. $(x = a_{i,j})$, if $a_{i,j}$ is the last element left in the matrix.

   In other words, if the algorithm asks about a matrix entry below the *bottom-left to top-right diagonal,* answer **smaller**, and if it is on, or above it, answer **greater**.

   Remark (1). We say an entry $a_{i,j} \in A$ is *removed* from A when the adversary confirms that it is not equal to x.

   Remark (2). Notice that if the algorithm asks about $a_{i,j} \in A$, and that $i + j \leq n + 1$, then for every $k \leq i$, $l \leq j$, $a_{k,l}$ will be removed.

   Now, by the way our adversary is defined, from the point of view of the algorithm, the only way to remove elements of the *bottom-left to top-right diagonal* is to compare each of its entries to x. This process is also sufficient to remove the upper left half of the matrix. Thus, removing every entry $a_{i,j}$ such that $j + i \leq n + 1$ takes **at least** n steps. By an analogous argument, removing the remaining entries will take at least $n - 1$ steps (by comparing x with each entry directly below the diagonal). Therefore, this adversary gives a lower bound of $2n - 1$.

   Remark (3). An alternate way to view the above adversary is that it picks the answer that minimizes the area of the square of entries removed.

   3D ADVERSARY STRATEGY: We use a similar strategy as in the 2D case, but instead of minimizing an area, as stated in *Remark 3*, we minimize a volume. Let $\Omega = \{<, >, =\}$, and let

$$F : [1,n] \times [1,n] \times [1,n] \subseteq \mathbb{N}^3 \to \Omega$$

be given by

$$F(i,j,k) = \begin{cases} <, & \text{if } i \times j \times k \leq (n+1-i) \times (n+1-j) \times (n+1-k) \\ >, & \text{if } i \times j \times k > (n+1-i) \times (n+1-j) \times (n+1-k) \\ =, & \text{if } a_{i,j,k} \text{ is the last entry not removed.} \end{cases}$$

where F is exactly our adversary for the 3D case.

Claim : The adversary defined by F gives a lower bound of $O(n^2)$.

(b) Give an algorithm that takes worst case time $O(L(n))$.

Solution:

Algorithm 2: An algorithm to find if x is in a 3D ordered matrix, in $O(L(n))$.

Input: A sorted $n \times n \times n$ matrix M of unique integers and an integer x. That is, $M[0][0][0]$ is the smallest element and $M[n-1][n-1][n-1]$ the biggest.

Output: True or False, whether $x \in M$

```
1  Function Is_x_in_this_2D_Matrix(Matrix_2D):
       // start the iteration at top right corner of matrix
2      row ← 0;
3      col ← $n-1$ ;
4      while col $\geq 0$ and row $\leq n-1$ do
5          if x = Matrix_2D[row][col] then
6              return True;
7          else if $x <$ Matrix_2D[row][col] then
8              col ← col − 1;
9          else
10             row ← row + 1;
11     return False;
   // go through each 2D matrix in the 3D matrix, starting with the top one.
12 for all $k \in [0, n-1]$ do
       // if x > M[n-1][n-1][n-1], no need to check this 2D matrix
13     if x $\leq M[n-1][n-1][n-1]$ then
14         if Is_x_in_this_2D_Matrix(M[k]) then
15             return True;
16 return False;
```

Remark. In the above algorithm, the time complexity of the function at line 1 is easily seen to be $O(n)$, as it takes at most $2n$ steps. Now, since this function will be called at most n times (once for each level of the 3D matrix), we conclude that the overall time complexity of the algorithm is $O(n^2)$, as desired.

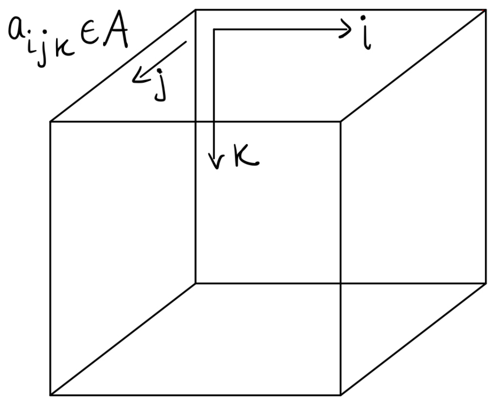

Figure 1: The 3D matrix in problem 3.