CHENNAI MATHEMATICAL INSTITUTE

M.Sc. / Ph.D. Programme in Computer Science

Entrance Examination, 2024

Solutions

Draft: 26 May 2024

- Please send comments/clarifications to `phdcs2024@cmi.ac.in`
- For Part B, each answer is graded on its merit. Any correct solution with an approach different from the one given in the draft will receive credit.

Part A has 10 questions of 3 marks each. Each question in Part A has four choices, of which exactly one is correct. Part B has 7 questions of 10 marks each. The total marks are 100. For questions in Part A, you have to provide the answers on the computer. For questions in Part B, you have to write your answers in the appropriate place in the answer booklet. If you need more space, you may continue on the pages provided for rough work. Any such overflows must be clearly labeled.
In all questions related to graphs, unless otherwise specified, we use the word "graph" to mean an undirected graph with no self-loops, and at most one edge between any pair of vertices.

Part A

1. All inhabitants of the Old Forest are either Ents or Bents. Ents always tell the truth and Bents always lie. During a visit to the Old Forest, you encounter four inhabitants – A, B, C and D. They make the following assertions.

   A: Exactly one of us is a Bent.
   B: Exactly two of us are Bents.
   C: Exactly three of us are Bents.
   D: Exactly four of us are Bents.

   How many of them are Bents?

   (a) 1 (b) 2 (c) 3 (d) 4

   Solution. (c). Since all the statements are (pairwise) mutually contradictory, at most one of them is an Ent. If there are no Ents, then D would be telling the truth, making him an Ent. But that is a contradiction. So there is exactly one Ent, and consequently three Bents. So C, as the only truth-teller, is an Ent, and the others are Bents.

2. Friends Akshay and Bipasha go to a fun fair, and decide to explore the stalls separately. Each of them visits a stall every 10 minutes, starting at 8am. If a stall that Akshay visits is one that he has not seen before, he texts *new* to Bipasha; otherwise he texts *old*. If a stall that Bipasha visits is new to her she texts *new* to Akshay; otherwise she texts *old*. A *round* refers to such a pair of messages, one sent by Akshay and the other sent by Bipasha. If there are 50 stalls, what is the maximum number of rounds they can go before both of them text *old* to each other in the same round?

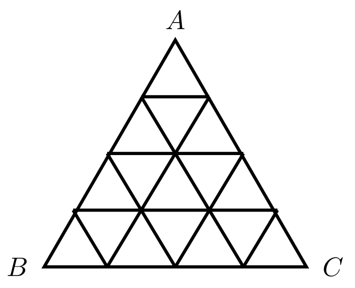
(a) 99 (b) 100 (c) 2500 (d) 50

Solution. (a) At each round, the texts can be either (old, old), (old, new), (new, old) or (new, new). There can be atmost 50 "new" visits for each friend. When the message is (new, old), one "new" visit for Akshay is consumed. Similarly, (old, new) consumes one "new" visit for Bipasha, and (new, new) consumes a "new" visit for both. The first round is (new, new). After the first round each of them is left with 49 new visits. Therefore, there can be at most a sequence of 98 rounds without (old, old). In total, there can be at most 99 rounds.

Here is a sequence that hits 99 rounds. Akshay visits the stall in this order: 1 1 2 2 3 3 ... 50 50. Bipasha visits in this order: 1 2 1 3 1 4.. 1 50. The texts would be (new, new) (old, new) (new, old) (old, new)....(old, new).

3. You live in Δ-City, which is a large equilateral triangle with each side of length 2km. The corners are named A, B and C. The city is partitioned into triangular blocks with sides of 500m each, and there are roads at the boundaries of the blocks.

   You live in corner A and your office is at corner B.



   You have decided to walk from home to the office everyday, and you are willing to change your route, as long as the total distance is at most 2.5kms. How many such routes are possible from A to B?

   (a) 5 (b) 11 (c) 21 (d) 45

   Solution (b). The smallest route is to go directly from A to B in 4 steps, moving *South-West* (each step is 500m long). There is no other route with 4 steps. To go from A to B in 5 steps, you can take a diagonal path going *South-East*, and compensate for it later with a horizontal *West* step. So, in all, you can take three *South-West* steps, with a *South-East* step and a later *West* step inserted. The number of permutations of 3 *South-West*, 1 *South-East*, 1 *West*, where *South-East* occurs before *West* is $\frac{5!}{3! \cdot 2} = 10$. Overall, there are 11 routes.

4. Rohit and Ben participate in a new toss system introduced by the ICC. The umpire repeatedly tosses a fair coin (which comes up heads with probability $\frac{1}{2}$ and tails with probability $\frac{1}{2}$), until one of them wins. Rohit wins if the result of two consecutive tosses is heads (i.e., the pattern HH is seen), while Ben wins if the pattern TH is seen. What are the winning probabilities for Rohit and Ben?

   (a) $\frac{1}{4}$ and $\frac{3}{4}$ (b) $\frac{1}{4}$ and $\frac{1}{4}$ (c) $\frac{1}{2}$ and $\frac{1}{2}$ (d) $\frac{1}{3}$ and $\frac{2}{3}$

   Solution: (a). Suppose the first toss results in a T. Then Rohit can never win from here on: either the rest of the tosses yield T, or the first time a H appears, Ben wins.

The probability that all tosses are T is 0. Therefore, if the first toss results in T, Ben wins with probability 1.

Suppose the first toss results in a H. Then, if the second toss is also H, Rohit wins. Otherwise, the second toss is T. Applying the same argument as above, from this point, Ben wins with probability 1.

Therefore, Ben wins if either the first toss is T or if the first two tosses give HT. Therefore Ben wins with probability $\frac{1}{2} + \frac{1}{4} = \frac{3}{4}$. For Rohit to win, the first two tosses need to be HH. Hence, he wins with probability $\frac{1}{4}$.

5. Let $\Sigma = \{a, b, c\}$. What is the language generated by the following grammar?

$$S := \epsilon \mid aS \mid Sb \mid cS$$

(a) $(a + b + c)^* b^*$ (b) $(a + b + c)^* c^*$
(c) $(a + c)^* b^*$ (d) $(a + b)^* c^*$

Solution: (c). Consider a word in $(a + c)^* b^*$. It is of the form $w_1 w_2 \ldots w_m b^n$ where each w_i is either a or c and $n \geq 0$. This can be generated as follows: $S \to w_1 S \to w_1 w_2 S \to \cdots \to w_1 w_2 \ldots w_m S$ followed by a sequence of application of the rule $S \to Sb$ ending in $S \to \epsilon$. So each word in $(a + c)^* b^*$ can be generated by the grammar.

Conversely, we need to show that any word generated by the grammar satisfies the regular expression $(a + c)^* b^*$, that is, it is of the form $w_1 w_2 \ldots w_m b^n$ as above. Notice that in any word generated by S, after a b, there can be no a or c; and before an a or c there can be no b. We can show by induction, that in any derivation of k steps the string generated is of the form $w_1 \ldots w_m S b^n$ where each $w_i \in \{a, c\}$.

6. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ for some $n \geq 1$. Consider the two languages:

$$L_1 = \{w \in \Sigma^* \mid \exists a_i \in \Sigma \text{ such that } a_i \text{ occurs at least two times in } w\}$$
$$L_2 = \Sigma^* \ (a_1 a_1 + a_2 a_2 + \cdots a_n a_n) \ \Sigma^*$$

Which of the following statements are true?

(I) There is an NFA with $O(n)$ states accepting L_1.

(II) There is a DFA with $O(n)$ states accepting L_1.

(III) There is an NFA with $O(n)$ states accepting L_2.

(IV) There is a DFA with $O(n)$ states accepting L_2.

(a) All of the above (b) I, III, IV
(c) I and III (d) I, II and III

Solution. (b).

Here is an NFA with $O(n)$ states for L_1. There are n initial states q_i, each of them guessing the letter a_i that will occur at least twice. Transitions are as follows: $q_i \xrightarrow{\Sigma \backslash a_i} q_i$, $q_i \xrightarrow{a_i} p_i$, $p_i \xrightarrow{\Sigma \backslash a_i} p_i$, $p_i \xrightarrow{a_i} r_i$, $r_i \xrightarrow{\Sigma} r_i$. Each r_i is a final state.

Here is a DFA with $O(n)$ states for L_2. States are q_0, $q_1, \ldots, q_n$ and f, with q_0 the initial state, and f the final state. Transitions are as follows: $q_0 \xrightarrow{a_i} q_i$, $q_i \xrightarrow{a_i} f$, $q_i \xrightarrow{a_j \neq a_i} q_j$, $f \xrightarrow{\Sigma} f$. Since a DFA is also an NFA, this is also an NFA with $O(n)$ states.

Finally, we will show that there is no DFA with $O(n)$ states for L_1. In fact, any DFA for L_1 will require 2^n states. Let w and w' be two words such that no letter appears twice in each of them, and there exists a_i such that is in w but not w'. Then $wa_i \in L_1$, but $w'a_i \notin L_1$. This shows that w and w' need to go to different states. Therefore, for every two distinct subsets of letters, we can find words that go to different states. This shows there are at least 2^n states.

7. Consider the following two recurrence relations:

- $T_1(n) = T_1(\frac{n}{2}) + T_1(\frac{n}{3}) + \Theta(n)$, $T_1(1) = 2$
- $T_2(n) = T_2(\frac{2n}{3}) + T_2(\frac{n}{3}) + \Theta(n)$, $T_2(1) = 2$

Which of the following statements is true?

(a) $T_1(n) = \Theta(n)$ and $T_2(n) = \Theta(n)$

(b) $T_1(n) = \Theta(n \log n)$ and $T_2(n) = \Theta(n \log n)$

(c) $T_1(n) = \Theta(n)$ and $T_2(n) = \Theta(n \log n)$

(d) $T_1(n) = \Theta(n \log n)$ and $T_2(n) = \Theta(n)$

Solution. (c).

(a) We consider $T_1(n)$ first. For any n, there is a k such that $n \leq 6^k \leq 6n$. Now, suppose $T_1(n) = T_1(\frac{n}{2}) + T_1(\frac{n}{3}) + 5n$. Define $S(i,j) = T_1(2^i 3^j)$. We thus get a recurrence for S as: $S(0,0) = 2$ and $S(i,j) = S(i-1,j) + S(i,j-1) + 5 \cdot 2^i \cdot 3^j$. We can show by induction on $i+j$ that $S(i,j) \leq 30 \cdot 2^i 3^j$. For the base case, $S(0,0) = 2 \leq 30 \cdot 2^0 3^0$. For the induction step,

$$\begin{aligned} S(i,j) &= S(i-1,j) + S(i,j-1) + 5 \cdot 2^i 3^j \\ &\leq 30 \cdot 2^{i-1} 3^j + 30 \cdot 2^i 3^{j-1} + 5 \cdot 2^i \cdot 3^j \\ &\leq 2^i 3^j (15 + 10 + 5) \\ &= 30 \cdot 2^i 3^j. \end{aligned}$$

Thus $T_1(n) \leq S(k,k) = 30 \cdot 6^k \leq 30 \cdot 6n = 180n$. Thus $T_1(n) = \Theta(n)$.

(b) We consider $T_2(n)$ next. For any n, there is a k such that $n \leq 3^k \leq 3n$. Now, suppose $T_2(n) = T_2(\frac{2n}{3}) + T_2(\frac{n}{3}) + 8n$. Define $S(i,j) = T_2(2^i 3^j)$. We get the following recurrence for S: $S(0,0) = 2$ and $S(i,j) = S(i+1,j-1) + S(i,j-1) + 8 \cdot 2^i 3^j$. By induction on $i+2j$, we can show that $S(i,j) \leq 8(i+2j+1)2^i 3^j$. For the base case, we have $S(0,0) = 2 \leq 8(0+1)2^0 3^0$. For the induction step,

$$\begin{aligned} S(i,j) &= S(i+1,j-1) + S(i,j-1) + 8.2^i 3^j \\ &\leq 8(i+1+2j-2+1)2^{i+1}3^{j-1} + 8(i+2j-2+1)2^i 3^{j-1} + 8 \cdot 2^i 3^j \\ &\leq 8(2(i+2j)/3 + (i+2j)/3 + 1)2^i 3^j \\ &= 8(i+2j+1)2^i 3^j. \end{aligned}$$

Now

$$\begin{aligned} T_2(n) &\le T_2(3^k) \\ &= S(0,k) \\ &\le 8(0+2k+1)2^0 3^k \\ &\le 8 \cdot 3n \cdot (2\log_3 n + 3) \\ &\le 96 n \log_3 n. \end{aligned}$$

Thus $T_2(n) = \Theta(n \log n)$.

8. Let G be an undirected connected graph with distinct edge weights. Let e_{max} be the edge with maximum weight and e_{min} the edge with minimum weight. What can you say about the following statements?

   (I) Every minimum spanning tree of G must contain e_{min}

   (II) Every minimum spanning tree must exclude e_{max}

   (a) I is True, but II is False (b) I is False, but II is True
   (c) Both I and II are False (d) Both I and II are True

   Solution. (a). Recall Kruskal's algorithm for finding minimum spanning tree. It starts by picking the edge with the smallest weight. As for the second statement, consider a graph which is simply a path with two edges $e_{\text{min}}e_{\text{max}}$. Here, we need to pick both the edges to span all the vertices.

9. Let G be a directed graph with distinct and nonnegative edge weights. Let s be a starting vertex and t a destination vertex. Assume that G has at least one s-t path. What can you say about the following statements?

   (I) Every shortest s-t path (minimum weight) must include the minimum-weight edge of G.

   (II) Every shortest s-t path must exclude the maximum-weight edge of G.

   (a) I is True, but II is False (b) I is False, but II is True
   (c) Both I and II are False (d) Both I and II are True

   Solution (c). Consider a graph: $s \xrightarrow{0} s_1 \xrightarrow{100} t$ and $s \xrightarrow{10} t$. Clearly, the shortest path is $s \xrightarrow{10} t$ which excludes the minimum-weight edge.

   Consider a graph with just two edges $s \xrightarrow{0} s_1 \xrightarrow{1} t$. The shortest $s - t$ path needs to include both the edges, in particular, the one with maximum-weight.

10. In the following code, A is an array indexed from 0, and for two integers a, b the expression $a//b$ returns $\lfloor \frac{a}{b} \rfloor$, the largest integer which is not larger than a/b.

```
FOO($A$, $first$, $last$)
1  if $first \geq last$
2     then return $A[first]$
3     else
4         $mid \leftarrow (first + last)//2$
5         $l \leftarrow$ FOO($A$, $first$, $mid$)
6         $r \leftarrow$ FOO($A$, $mid+1$, $last$)
7         return $l + r$
```

If $A = [1, 2, 3, 4, 5, 6]$, what will FOO$(A, 0, 5)$ return?

(a) 3 (b) 7 (c) 15 (d) 21

Solution: (d), 21.

For any $i_1 \leq j$, we can show by induction on $j - i$ that whenever i and j are valid indices in A, the call FOO(A, j, i) computes the sum of the elements in $A[i \ldots j]$. In the base case, we have $i = j$ and the function returns $A[i]$, which is the sum of all elements in $A[i \ldots i]$. For the induction step, we see that recursive calls return the sums of $A[i \ldots m]$ and $A[m + 1 \ldots j]$, so their sum is $A[i \ldots j]$, as desired.

Part B

1. A *binary search tree* is a binary tree whose proper subtrees are binary search trees, and whose root is strictly greater than all elements in the left subtree, and strictly less than all elements in the right subtree. A *preorder listing* of a tree is obtained by listing the root first, then recursively listing all elements of the left subtree in preorder, followed by all elements of the right subtree in preorder.

   Provide algorithms for the following two problems, and calculate their worst-case running times.

   (a) Input is an array of integers $A[1..n]$. Output is "Yes" if A is the preorder listing of a binary search tree, and "No" otherwise.

   (b) Input is an array of integers $A[1..n]$ which is guaranteed to be the preorder listing of a binary search tree. Output is a binary tree t such that A is the preorder listing of t.

   Solution.

   (a) Given $A[1..n]$, check if there is a $j \in [2..n]$ such that every element in $A[2..j]$ is strictly less than $A[1]$, and every element in $A[j+1..n]$ is strictly greater than $A[1]$. If no such j exists, return "No". Else, make a recursive call of the algorithm on $A[2..j]$ and $A[j+1..n]$.

   (b) We can build the tree using the above algorithm. Each call now returns a tree. Since $A[1..n]$ is guaranteed to be a preorder listing of a binary search tree, the call to $A[1..n]$ will have a j, as above. In fact, it will be a unique j. Make $A[1]$ as root, and the tree obtained by the call to $A[2..j]$ as the left sub-tree, and the one obtained by $A[j+1..n]$ as the right sub-tree.

   Since the tree could be skewed, both the above algorithms take $O(n^2)$ time in the worst case.

2. Let $M_1 = (Q_1, \{q_1\}, \Delta_1, F_1)$, where $\Delta_1 \subseteq Q_1 \times (\Sigma \cup \{\epsilon\}) \times Q_1$, be a non deterministic finite automaton (NFA) accepting a language $L_1 \subseteq \{0,1\}^*$. Let ϵ denote the null string. We construct a new NFA $M_2 = (Q_2, \{q_2\}, \Delta_2, F_2)$, where $\Delta_2 \subseteq Q_2 \times (\Sigma \cup \{\epsilon\}) \times Q_2$, as follows.

   - $Q_2 = Q_1$.
   - $q_2 = q_1$.
   - $F_2 = F_1 \cup \{q_1\}$.
   - $(p, a, p') \in \Delta_2$ iff either $(p, a, p') \in \Delta_1$ or ($p \in F_1$ and $a = \epsilon$ and $p' = q_1$)

   Prove or disprove: The language L_2 accepted by M_2 is L_1^*.

   Solution. No. In M_2 we make q_1 a final state, whereas q_1 may not be final in M_1. Let M_1 be the automaton with two states q_1 and p, with p the single final state. Transitions are:

$$q_1 \xrightarrow{a} q_1$$
$$q_1 \xrightarrow{b} p$$
$$p \xrightarrow{a,b} p$$

   Automaton M_1 accepts all words containing a b. The automaton M_2 that we construct from M_1 makes q_1 a final state, and therefore also accepts the word a.

3. You are using a fair coin to walk along the number line, starting at position 0. You toss the coin. If you get heads you move two steps to your right, to position 2. On the other hand if you get tails you move one step to your right, to position 1. You continue tossing the coin. Every time you get heads you move two steps to the right of the position you are in currently, and if you get tails you move one step to the right of the position you are in currently.

   Show that the probability of landing in position n is

$$\frac{1}{3}[2 + (-\frac{1}{2})^n]$$

   .

   Solution.

   Denote by P_n the probability of landing in position n. We prove by induction on $n \geq 0$ that $P_n = \frac{1}{3}[2 + (-\frac{1}{2})^n]$.

   For $n = 0$, we start at position 0, so $P_0 = 1 = \frac{1}{3}[2 + 1] = \frac{1}{3}3[2 + (-\frac{1}{2})^0]$. For $n = 1$, we lanf at position 1 if the first toss lands tails up. So $P_1 = \frac{1}{2} = \frac{1}{3}[2 - \frac{1}{2}] = \frac{1}{3}[2 + (-\frac{1}{2})^1]$.

   For $n \geq 2$, we can get to n by first getting to $n - 2$ and tossing heads, or by getting to $n - 1$ and tossing a tail. These two are disjoint events, so the probabilities can be added up, and we get

$$\begin{aligned}
P_n &= \frac{1}{2}P_{n-1} + \frac{1}{2}P_{n-1} \\
&= \frac{1}{2} \cdot \frac{1}{3}[2 + (-\frac{1}{2})^{n-2}] + \frac{1}{2} \cdot \frac{1}{3}[2 + (-\frac{1}{2})^{n-1}] \\
&= \frac{1}{6}[4 + (-\frac{1}{2})^{n-2}(1 + (-\frac{1}{2}))] \\
&= \frac{1}{3}[\frac{4}{2} + \frac{1}{2}(-\frac{1}{2})^{n-2}\frac{1}{2}] \\
&= \frac{1}{3}[2 + (\frac{1}{2})^2(-\frac{1}{2})^{n-2}] \\
&= \frac{1}{3}[2 + (-\frac{1}{2})^2(-\frac{1}{2})^{n-2}] \\
&= \frac{1}{3}[2 + (-\frac{1}{2})^n]
\end{aligned}$$

4. Suppose all points on the 2D plane with integer coordinates are coloured either blue or green. Show that there is an isosceles right angled triangle all of whose vertices are the same colour.

   Solution. If we have two consecutive points $(a, b), (a, b+1)$ on a line parallel to y axis which are coloured blue and if any of $(a-1, b), (a+1, b), (a-1, b+1), (a+1, b+1)$ is coloured the same, we are done. Otherwise there are two points with coordinates (x, y), $(x+2, y)$ with green colour. (Likewise if we have consecutive points on a line parallel to x axis coloured identically we are done or (x, y), $(x, y+2)$ will be coloured the same). Assume the first case - if both (x, y) and $(x+2, y)$ are coloured green, consider the points $(x+1, y+1)$, $(x, y+2)$ and $(x+2, y+2)$. If any one is green we are done. Otherwise all are blue and we are done.

5. Let Σ be a finite alphabet. The *reverse* of a word is defined inductively as follows: $\mathsf{rev}(\epsilon) = \epsilon$ and $\mathsf{rev}(wa) = a \cdot \mathsf{rev}(w)$ for $w \in \Sigma^*$ and $a \in \Sigma$. For example, $\mathsf{rev}(aab) = baa$.

   For a language L, we define $\mathsf{rev}(L) := \{\mathsf{rev}(w) \mid w \in L\}$.

   (a) Is $\mathsf{rev}(L_1 \cap L_2)$ equal to $\mathsf{rev}(L_1) \cap \mathsf{rev}(L_2)$?

   (b) Prove or disprove the following statement: $w \in L \cap \mathsf{rev}(L)$ iff $w = \mathsf{rev}(w)$.

   (c) Show that if L is regular, $\mathsf{rev}(L)$ is also regular.

Solution.

(a) Yes.

To show $\mathsf{rev}(L_1 \cap L_2) \subseteq \mathsf{rev}(L_1) \cap \mathsf{rev}(L_2)$: Suppose $w \in \mathsf{rev}(L_1 \cap L_2)$. Then, $\mathsf{rev}(w) \in L_1 \cap L_2$. This implies, $\mathsf{rev}(w) \in L_1$ and $\mathsf{rev}(w) \in L_2$. Hence, $w \in \mathsf{rev}(L_1)$ and $w \in \mathsf{rev}(L_2)$, proving that $w \in \mathsf{rev}(L_1) \cap \mathsf{rev}(L_2)$.

To show $\mathsf{rev}(L_1) \cap \mathsf{rev}(L_2) \subseteq \mathsf{rev}(L_1 \cap L_2)$: Pick $w \in \mathsf{rev}(L_1) \cap \mathsf{rev}(L_2)$. Hence $\mathsf{rev}(w) \in L_1 \cap L_2$. This implies $w \in \mathsf{rev}(L_1 \cap L_2)$.

(b) No. Take $L = \{ab, ba\}$ and $w = ab$. Observe that $\mathsf{rev}(L) = L$. Therefore, $w \in L \cap \mathsf{rev}(L)$. But $w \neq \mathsf{rev}(w)$.

(c) Take a DFA A for L. Reverse all transitions: change $q \xrightarrow{a} q'$ to $q' \xrightarrow{a} q$. Make all final states of A as initial states, and the initial state of A as a final state. The resulting automaton accepts $\mathsf{rev}(L)$.

6. A *tournament* is a directed graph that has exactly one directed edge between each pair of vertices. A king in a tournament is a vertex v such that every other vertex is reachable from v via a directed path of length at most 2.

   (a) Prove that in any tournament there is at least one king.

   (b) Can there be more than one king in a tournament? Justify your answer.

Solution.

(a) Proof by induction. Suppose the statement is true for $n-1$ vertices. Consider a graph G with n vertices, and pick an arbitrary vertex v. Let G' be the tournament obtained by deleting v. By induction hypothesis, there is a king v' in G'. We claim that either v or v' is a king in G.

If G contains edge $v' \to v$, then v' is also a king in G. Otherwise, there is a edge $v \to v'$. In G', let V_1 be the set of vertices reachable from king v' in one step. Similarly, let V_2 be the vertices reachable from v' in exactly 2 steps. Since v' is king, $V_1 \cup V_2$ contains all vertices of G', except v'. If there is an edge from v to all vertices of V_1, then v is a king in G. Else, there is an edge from some $u \in V_1$ to v. In this case, v' is also a king in G: there is a path $v' \to u \to v$.

(b) Yes. Consider a directed triangle. All vertices are kings.

7. A *subsequence* of an array A is any sub-array of A, obtained by deleting zero or more elements of A *without* changing the order of the remaining elements. The input to the SUBSEQUENCE SUM problem consists of (i) an array $A[1\dots n]$ of n positive integers, and (ii) a target integer $T \geq 0$. The problem is to decide if there exists a subsequence B of A such that the sum of all the elements of B is exactly T. We define the sum of the *empty* subsequence (one with no elements) to be zero, and the sum of a subsequence with one element, to be that element.

   Describe an algorithm that solves this problem in $O(n^cT^d)$ time for some constants c, d. The algorithm should take an array $A[1\dots n]$ and an integer T as described above. It should output True if there is a subsequence B of A such that the sum of all the elements of B is exactly T, and False otherwise. It is *not* required that the algorithm find a subsequence whose sum is T.

   Clearly explain why your algorithm correctly solves the problem, and why it runs in time $O(n^cT^d)$. What are the constants c, d that you get?

   Solution.

   First, do some preprocessing: If T is zero, return True. Otherwise, if no element of A is smaller than T, return False.

   Construct an $n \times (T+1)$ table $DP[1\dots n][0\dots T]$. For $1 \leq i \leq n$ and $0 \leq j \leq T$, we will compute $DP[i][j]$ to be True if *some subsequence of* $A[1\dots i]$ adds up to exactly $j \leq T$, and False otherwise. We prove correctness by induction on the row index of the DP table. Initialize all entries of DP to False.

   For $1 \leq i \leq n$, do:

   (a) Set $DP[i][0] = True$
   (b) if $A[i] \leq T$ then set $DP[i][A[i]] = True.$

   The first assignment is correct because the sum of the empty subsequence is zero by definition. The second assignment is correct because the sum of the subsequence $\{A[i]\}$ is $A[i]$ by definition. These assignments ensure that the base case of the induction is correct; that is, that all the entries in the row $DP[1]$ are set correctly.

   For $2 \leq i \leq n$ and for $1 \leq j \leq T$, do:

   (a) If $DP[i-1][j] = True$ then set $DP[i][j] = True$
   (b) Else, if $j \geq A[i]$ and $DP[i-1][j-A[i]] = True$ then set $DP[i][j] = True$

   The first assignment is correct because $DP[i-1][j]$ is True—inductively—iff there is some subsequence of $A[1\dots(i-1)]$ that already adds up to j. The second assignment is correct because $DP[i-1][j-A[i]]$ is True—inductively—iff there is some subsequence of $A[1\dots(i-1)]$ that adds up to $j - A[i]$. And adding $A[i]$ to this sum results in the sum being j.

   If there is a subsequence of $A[1\dots i]$ that adds up to j, then it either contains $A[i]$ or not. The two assignments cover both these cases. So these are exhaustive.

   After the loops are done, return $DP[n][T]$.

   Filling in the table takes $O(nT)$ time, and the other operations take $O(n)$ or $O(T)$ time each.