CHENNAI MATHEMATICAL INSTITUTE

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

Entrance Examination, 4 October 2020

Part A has 10 questions of 3 marks each. Part B has 7 questions of 10 marks each. The total marks are 100.
Answers to Part A must be given in the special answer sheet provided for it. Answers to Part B must be written only in the designated space after the corresponding question.

Part A

1. Which of the following languages over the alphabet $\{0, 1\}$ are *not* recognized by deterministic finite state automata (DFA) with *three* states?

   (a) Words which do not have 11 as a contiguous subword
   (b) Binary representations of multiples of three
   (c) Words that have 11 as a suffix
   (d) Words that do not contain 101 as a contiguous subword

   Answer: $\boxed{\text{(d) Words that do not contain 101 as a contiguous subword}}$

   For (a), the automaton has states $\{q_0, q_1, q_2\}$ with q_0 as start state, q_0 and q_1 as accept states, and transition function given below:

	0	1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

   For (b), the automaton has states $\{q_0, q_1, q_2\}$ with q_0 as start and accept state, and transition function given below:

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

   For (c), the automaton has states $\{q_0, q_1, q_2\}$ with q_0 as start state, q_2 as accept state, and transition function given below:

	0	1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_2

   For (d), an automaton with three states is not possible, since we need at least four states to remember the four prefixes of 101 that have been seen as a contiguous subword in the input read so far. $\dashv$

2. Consider the following regular expressions over alphabet $\{a, b\}$, where the notation $(a+b)^+$ means $(a+b)(a+b)^*$:

$$r_1 = (a+b)^+ \; a \; (a+b)^*$$
$$r_2 = (a+b)^* \; b \; (a+b)^+$$

Let L_1 and L_2 be the languages defined by r_1 and r_2, respectively. Which of the following regular expressions define $L_1 \cap L_2$?

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

Answer: $\boxed{\text{(c) } (a+b)^* \; b \; (a+b)^* \; a \; (a+b)^*}$

L_1 is the set of all words of length $n \geq 2$ containing an a at one of the positions $2, \cdots, n$. L_2 is the set of all words of length $n \geq 2$ containing a b in one of the positions $1, \cdots, n-1$. The intersection $L_1 \cap L_2$ is the set of all words containing a b at one of the positions $1, \ldots, n-1$ and an a at one of the positions $2, \ldots, n$. This is given by the expression in (c). Option (a) is incorrect since it accepts only words of length at least 4. Option (b) is incorrect since it accepts ab, which is not in $L_1 \cap L_2$. Option (d) is incorrect once again as it accepts ab. ⊣

3. Some children are given boxes containing sweets. Harish is happy if he gets either gems or toffees. Rekha is happy if she gets both bubble gums and peppermints. Some of the boxes are special, which means that if the box contains either gems or toffees, then it also contains bubble gums and peppermints. If Harish and Rekha are given boxes that are not special, which of the following can we infer?

(a) Harish is happy

(b) No bubble gums in Rekha's box

(c) No toffees in Harish's box

(d) There are peppermints in Rekha's box

Answer: $\boxed{\text{(a). Harish is happy.}}$

Let t denote that a box has toffees, g denote that it has gems, b denote the presence of bubble gums and p the presence of peppermints. If a box is special, it satisfies $(g \vee t) \Rightarrow (b \wedge p)$. A non-special box satisfies $(g \vee t) \wedge (\neg b \vee \neg p)$. Since $g \vee t$ holds, Harish is happy and (a) can be inferred. Since it is possible that the box has only toffees and no gems, we cannot infer (c). Since it is possible that there are bubble gums but no peppermints, neither (b) nor (d) can be inferred. ⊣

4. In a class, every student likes exactly one novelist and one musician. If two students like the same novelist, they also like the same musician. The class can be divided into novelist groups, each group consisting of all students who like one novelist. Similarly, musician groups can be formed. So each student belongs to one musician group and one novelist group. Which of the following is a valid conclusion?

(a) There are more musician groups than novelist groups

(b) There are at least as many novelist groups as musician groups

(c) For every musician group, there is a bigger novelist group

(d) For every novelist group, there is a musician group of the same size

Answer: $\boxed{\text{(b) There are at least as many novelist groups as musician groups}}$

Each novelist group is a subset of some musician group. So the number of novelist groups are at least as large as the number of music groups, which is option (b). Options (a), (c) and (d) are all violated in a scenario where there are two students s_1 and s_2, two novelists n_1 and n_2, and one musician m, with s_1 liking n_1 and m, and s_2 liking n_2 and m. ⊣

5. A boolean function on n variables is a function f that takes an n-tuple of boolean values $x \in \{0,1\}^n$ as input and produces a boolean value $f(x) \in \{0,1\}$ as output.

   We say that a boolean function f is symmetric if, for all inputs $x, y \in \{0,1\}^n$ with the same number of zeros (and hence the same number of ones), $f(x) = f(y)$. What is the number of symmetric boolean functions on n variables?

   (a) $n+1$ (b) $n!$ (c) $\sum_{i=0}^{n} \binom{n}{i}$ (d) 2^{n+1}

   Answer: $\boxed{\text{(d) } 2^{n+1}}$

   Let $C_i = \{x \in \{0,1\}^n \mid x \text{ has } i \text{ occurrences of } 0\}$. For any $i \leq n$, a symmetric function has to choose the same output for each $x \in C_i$. Thus a symmetric function can be described by an $(n+1)$-tuple of binary values, and hence the number of symmetric functions is 2^{n+1}. ⊣

6. There are n songs segregated into 3 playlists. Assume that each playlist has at least one song. For all n, the number of ways of choosing three songs consisting of one song from each playlist is:

   (a) $> \frac{n^3}{27}$ (b) $\leq \frac{n^3}{27}$ (c) $\binom{n}{3}$ (d) n^3

   Answer: $\boxed{\text{(b) } \leq \frac{n^3}{27}}$

   Suppose the number of songs in each playlist is n_1, n_2, n_3 respectively. We have $n_1, n_2, n_3 \geq 1$. The number of ways of choosing three songs with one from each playlist is $n_1 \cdot n_2 \cdot n_3$. Since the geometric mean of the numbers is less than or equal to the arithmetic mean, it follows that $n_1 \cdot n_2 \cdot n_3 \leq \frac{n^3}{27}$. So, the answer is Option (b).

   Options (a) and (d) cannot be correct because of the above inequality. Option (c) cannot be true for all n. For instance, choose $n_1 = 2, n_2 = n_3 = 1$. Then Option (c) gives 4 as the answer which is wrong. ⊣

The next two questions are based on the following facts.

Basketball shots are classified into *close-range*, *mid-range* and *long-range* shots. Long range shots are worth 3 points, while close-range and mid-range shots are worth 2 points. Of the shots that LeBron James attempts, 45% are close-range, 25% are mid-range, and 30% are long-range. He successfully makes 80% of the close-range shots, 48% of the mid-range shots, and 40% of the long-range shots.

7. What is the probability that a LeBron shot attempt is successful?

   (a) $\frac{1}{2}$ (b) $\frac{4}{5}$ (c) $\frac{3}{5}$ (d) $\frac{4}{7}$

   Answer: $\boxed{\text{(c) } \frac{3}{5}}$

   For every 100 shots that LeBron James attempts, he makes $80/100 * 45 + 48/100 * 25 + 40/100 * 30 = 36 + 12 + 12 = 60$ of them. So the probablity that LeBron shot is successful is $60/100 = 3/5$. $\dashv$

8. What is the probability that a successful 2-point shot attempt by LeBron is a close-range shot?

   (a) $\frac{2}{5}$ (b) $\frac{3}{5}$ (c) $\frac{3}{7}$ (d) $\frac{3}{4}$

   Answer: $\boxed{\text{(d) } \frac{3}{4}}$

   From the previous solution, we see that out of every 100 shots, 36 are successful close-range shots, and 12 are successful mid-range shots. So out of every 100 shots attempted by LeBron James, 48 are successful 2-point shots, and 36 of them are close-range. Thus the probability that a successful 2-point attempt by LeBron is a close-range shot is $36/48 = 3/4$. $\dashv$

9. A fair coin is repeatedly tossed. Each time a head appears, 1 rupee is added to the first bag. Each time a tail appears, 2 rupees are put in the second bag.

   What is the probability that both the bags have the same amount of money after 6 coin tosses?

   (a) $\frac{1}{2^6}$ (b) $\frac{6!}{2! \cdot 4! \cdot 2^6}$ (c) $\frac{2^2}{2^6}$ (d) $\frac{6!}{2^6}$

   Answer: $\boxed{\text{(b) } \frac{6!}{2! \cdot 4! \cdot 2^6}}$

   For the money to be equal in the bags, the number of heads should be twice the number of tails. This means in 6 tosses, there should be 2 tails and 4 heads. The number of ways of getting 2 tails and 4 heads in 6 coin tosses is ${}^6C_2$. The total number of different occurrences of the coin tosses is 2^6. Hence probability is $\frac{{}^6C_2}{2^6}$. $\dashv$

10. We have a procedure $P(n)$ that makes multiple calls to a procedure $Q(m)$, and runs in polynomial time in n. Unfortunately, a significant flaw was discovered in $Q(m)$, and it had to be replaced by $R(m)$, which runs in exponential time in m. Thankfully, P is still correct when we replace each call to $Q(m)$ with a call to $R(m)$ instead. Which of the following can we *definitely* say about the modified version of P?

    (a) $P(n)$ still runs in polynomial time in n.

    (b) $P(n)$ requires exponential time in n.

    (c) $P(n)$ runs in polynomial time in n if the number of calls made to Q is proportional to $\log n$.

    (d) $P(n)$ runs in polynomial time in n if, for each call $Q(m)$, $m \leq \log n$.

    Answer: $\boxed{\text{(d)}}$

    If $m \leq \log n$ for each call to Q with argument m, then the running time of $R(m)$ is exponential in m but polynomial in n. Since the number of calls to Q originally (and R after the modification) is a polynomial in n, the overall running time of P is polynomial.

    None of the other options definitively hold. $\dashv$

Part B

Answers to Part B must be written only in the designated space after the corresponding question.

1. There are two cities, City X and City Y. Each city has a metro system consisting of three different lines — red line, blue line, and green line. Each station (in both cities) is classified as either *interesting* or *uninteresting*, depending on the places of interest near the station. Both cities have a designated station called *City Centre*. From every station, there is a red line, blue line and a green line, each going to different destinations.

   A sequence of colours represents a journey through the metro system. For example, with the sequence RGGB, one would first take the red line, get down and then take the green line, get down and again take the green line, and so on. Starting from the City Centre, following a sequence of colours will lead to a destination, which may or may not be interesting. Design an algorithm to check if there is a sequence of colours following which one can reach an interesting destination from the City Centre in X, but not from the City Centre in Y.

   Answer:

   The metro system in each city can be modeled as a finite state automaton, with the alphabet being $\{R, G, B\}$. Every metro station is a state. There is a transition from station 1 to station 2 on R (respectively G, B) if the red line (respectively green line, blue line) from station 1 goes to station 2. Final states are the interesting destinations. Now the question is whether there exists a word in the language of the automaton for the first city but not in the language of the automaton for the second city. Letting A_1 and A_2 be the automaton corresponding to the first and second city respectively, we have to determine if $L(A_1) \setminus L(A_2) \neq \emptyset$. We can use standard constructions to first define an automaton A_3 such that $L(A_3) = \{R, G, B\}^* \setminus L(A_2)$, and then an automaton A_4 such that $L(A_4) = L(A_1) \cap L(A_3)$. Finally we use the standard emptiness checking algorithm to check if $L(A_4) \neq \emptyset$. $\dashv$

2. A graph is *finite* if it has a finite number of vertices, and *simple* if it has no self-loops or multiple edges. Assume we are dealing with finite, undirected, simple graphs with at least two vertices. A graph is *connected* if there is a path between any two vertices in the graph, and is *disconnected* otherwise. The *complement* $\overline{G}$ of a graph G has the same vertex set as G, and contains an edge $\{u, v\}$ if and only if G *does not* contain the edge $\{u, v\}$. For the first and third questions below, you will get the credit only if you *also* provide the justification. If your answer is **yes**, drawing an example of such a graph G and its complement $\overline{G}$ suffices as the justification. If your answer is **no**, then you should argue why this is the case.

   (a) Does there exist a graph G with at least two vertices such that *both* G and $\overline{G}$ are *disconnected*? Justify your answer.

   (b) Give an example of a graph G on **four** vertices such that G is isomorphic to its complement $\overline{G}$.

   (c) Does there exist a graph G with at least two vertices such that *both* G and $\overline{G}$ are *connected*? Justify your answer.

Answer:

(a) No, there are no such graphs. Let G be a disconnected graph, and let u, v be two vertices in G such that there is no path in G from u to v. Let C_u be the set of vertices of the connected component of G to which u belongs, and let $R = V(G) \setminus C_u$ be the rest of the vertices. Then $v \in R$ and no vertex in C_u has a neighbour in R. So in $\overline{G}$ *every vertex* in C_u is a neighbour of *every* vertex in R. If $|C_u| = 1$ then this means that $\overline{G}$ is connected. So let $|C_u| \geq 2$.

Let x, y be any two vertices in $\overline{G}$. If one of these is in C_u and the other is in R then (as shown above) $\{x, y\}$ is an edge in $\overline{G}$. If both x and y are in C_u then $\{x, v\}$ and $\{y, v\}$ are edges in $\overline{G}$. If both x and y are in R then $\{x, u\}$ and $\{y, u\}$ are edges in $\overline{G}$. Thus there is a (short!) path between every pair of vertices in $\overline{G}$, and so $\overline{G}$ is connected.

(b) We take $G = (V, E)$ where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}\}$. $\overline{G} = (V, F)$ where $F = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_4\}\}$. G is isomorphic to $\overline{G}$, as given by the map $\{v_1 \mapsto v_2, v_2 \mapsto v_4, v_3 \mapsto v_1, v_4 \mapsto v_3\}$.

(c) Yes. G from part (b) is an example.

$\dashv$

3. A graph is *finite* if it has a finite number of vertices, and *simple* if it has no self-loops or multiple edges.

   Prove or disprove: There exists a finite, undirected, simple graph with at least two vertices in which each vertex has a different degree. To give a proof it suffices to draw an example of such a graph. To disprove the result, you should provide an argument as to why such a graph cannot exist.

   Answer:

   There are no such graphs. If G is such a graph on n vertices then the possible degrees for its vertices are $0, 1, \ldots, (n-1)$, and since there are exactly n of these values, **all** of them must appear as a degree of some vertex exactly once. If a vertex of G has degree $(n-1)$ then it must be adjacent to *every other* vertex in G, and in this case no vertex can have the degree 0. So G cannot exist. $\dashv$

4. Consider the procedure Mystery described in pseudocode below. The procedure takes two non-negative integers as arguments. For a real number x the notation $\lfloor x \rfloor$ denotes the largest integer which is not larger than x.

```
Mystery($p, q$)
1  if $p == 0$
2      then return 0
3  $r \leftarrow \lfloor p/2 \rfloor$
4  $s \leftarrow q + q$
5  $t \leftarrow$ Mystery($r, s$)
6  if $p$ is even
7      then return $t$
8      else  return $t + q$
```

(a) What does Mystery$(100, 150)$ return?

(b) How many times is the statement on line 8 executed in a call to Mystery$(100, 150)$?

(c) In general, what does Mystery(m, n) return for $m, n \geq 0$? Justify your answer with a proof.

Answer:

(a) 15000

(b) This is the number of 1s in the binary representation of 100. So the answer is 3.

(c) Mystery computes the product of the two input numbers. It is an implementation of the following recurrence:

$$x \times y = \begin{cases} 0 & \text{if } x = 0 \\ \lfloor x/2 \rfloor \times 2y & \text{if } x \text{ is even} \\ (\lfloor x/2 \rfloor \times 2y) + y & \text{if } x \text{ is odd} \end{cases}$$

⊣

5. Let $\Sigma = \{a, b\}$. For two non-empty languages L_1 and L_2 over Σ, we define $\mathit{Mix}(L_1, L_2)$ to be $\{w_1\ u\ w_2\ v\ w_3 \mid u \in L_1, v \in L_2, w_1, w_2, w_3 \in \Sigma^*\}$.

   (a) Give two languages L_1 and L_2 such that $\mathit{Mix}(L_1, L_2) \neq \mathit{Mix}(L_2, L_1)$.

   (b) Show that if L_1 and L_2 are regular, the language $\mathit{Mix}(L_1, L_2)$ is also regular.

   (c) Provide languages L_1 and L_2 that are not regular, for which $\mathit{Mix}(L_1, L_2)$ is regular.

Answer:

(a) Take $L_1 = \{a\}$ and $L_2 = \{b\}$. Word ab belongs to $\mathit{Mix}(\{a\}, \{b\})$, but ab is not in $\mathit{Mix}(\{b\}, \{a\})$.

(b) Let A_1 and A_2 be NFA for L_1 and L_2 respectively. We construct an ε-NFA for $\mathit{Mix}(L_1, L_2)$. Consider three new states $\{q_1, q_2, q_3\}$ and add self loops with $\{a, b\}$ on these three states. Add the following ε transitions: q_1 to the initial states of A_1; the final states of A_1 to q_2; q_2 to the initial state of A_2; the final states of A_2 to q_3. Make q_1 as the initial state and q_3 as the (single) final state of this constructed ε-NFA.

(c) Take $L_1 = L_2 = \{a^n b^n \mid n \geq 1\}$. $\mathit{Mix}(L_1, L_2)$ is the following set of all words, which is regular:

$$\{w_1\ ab\ w_2\ ab\ w_3 \mid w_1, w_2, w_3 \in \Sigma^*\}.$$

$\dashv$

6. A password contains exactly 6 characters. Each character is either a lowercase letter $\{a, b, \ldots, z\}$ or a digit $\{0, 1, \ldots, 9\}$. A *valid* password should contain at least one digit. What is the total number of valid passwords?

   (a) Here is an incorrect answer to the above question. Find the flaw in the argument. Let P_i denote the number of passwords where the i-th character is a digit, for $i \in 1, \ldots, 6$.

$$\begin{aligned}
P_1 &= 10 \cdot (36)^5 \\
P_2 &= 36 \cdot 10 \cdot (36)^4 \\
&\cdots \\
P_6 &= (36)^5 \cdot 10
\end{aligned}$$

   Therefore, total number of valid passwords is $6 \cdot 10 \cdot (36)^5$ (the sum of the right hand sides above).

   (b) What is the correct answer? Provide a justification for your answer. You do not need to simplify your expressions (for example, you can write 26^5, $5!$, etc.).

Answer:

(a) There is double counting. For example, a password $0a0aaa$ is counted in P_1 as well as P_3.

(b) The total number of passwords equals 36^5. The number of passwords with no digit equals 26^5. Hence, the number of valid passwords is $36^5 - 26^5$.

$\dashv$

7. We are given an array of N words $W[1 \cdots N]$, and a length array $L[1 \cdots N]$, where each $L[i]$ denotes the length (number of characters) of $W[i]$. We are also given a *line width* M, which is assumed to be greater than every $L[i]$. We would like to print all these words, in the same order, *without breaking any word across multiple lines.*

To illustrate, suppose $M = 15$, $N = 7$, and the words and lengths are as given below:

Word	`Can`	`you`	`solve`	`this`	`using`	`dynamic`	`programming?`
Length	3	3	5	4	5	7	12

Here are four example ways of laying out this text. Note that there should be a space between adjacent words on the same line.

Layout 1	**Layout 2**	**Layout 3**	**Layout 4**

`Can`
`you`
`solve`
`this`
`using`
`dynamic`
`programming?`
	`Can you`
	`solve this`
	`using dynamic`
	`programming?`
		`Can you solve`
		`this using`
		`dynamic`
		`programming?`
			`Can you solve`
			`this using dynamic`
			`programming?`

Of the four layouts above, Layouts 1, 2 and 3 are valid since each line has at most 15 characters, while Layout 4 is invalid since line 2 requires 18 characters, which will spill over beyond the line width of 15.

Each valid layout has a cost, computed as follows. Let the text be spread out over K lines. For each line i, let s_i be the number of spaces that are appended at the end of the line to make the line exactly M characters long. The cost of the layout is $\sum_{i=1}^{K} s_i^3$. For example, Layout 1 has cost

$$(15-3)^3+(15-3)^3+(15-5)^3+(15-4)^3+(15-5)^3+(15-7)^3+(15-12)^3 = 7326.$$

Layout 2 has cost

$$(15-7)^3+(15-10)^3+(15-13)^3+(15-12)^3 = 672.$$

Layout 3 has cost

$$(15-13)^3+(15-10)^3+(15-7)^3+(15-12)^3 = 672.$$

We can use dynamic programming to compute the smallest cost of any layout for N words $W[1 \cdots N]$ with lengths given by $L[1 \cdots N]$, and line width M. For $1 \leq i \leq N$, let $C[i]$ denote the minimum cost of any layout for words $W[i \cdots N]$.

(a) Write a recurrence relation for $C[i]$ in terms of $C[i+1], C[i+2], \ldots, C[N]$.

(b) Implement your recurrence as a dynamic programming algorithm.

(c) How much time and space does your algorithm need, in terms of N?

Answer:

(a) It is convenient to define $C[N+1] = 0$. For $i \leq N$, if we print words $W[i]$ to $W[j]$ on the first line, the number of spaces at the end of the first line is $s_{i,j} = M - (L[i] + \cdots + L[j] + j - i)$. We then have to distribute $W[j+1], \ldots, W[N]$ on the remaining lines, and the minimum cost for that is $C[j+1]$. Thus

$$C[i] = \min\{s_{i,j}^3 + C[j+1] | i \leq j \leq N, s_{i,j} \geq 0\}.$$

(b) We solve this by filling in the C array from entry $C[N+1] = 0$ onwards. To compute $C[i]$, we require $s_{i,j}$ values for $j \geq i$. Rather than naïvely recomputing each s_{ij} as a separate summation, we use the recurrence $s_{i,j} = s_{i,j-1} - L[j] - 1$ to compute the values on the fly. The full program is given below.

```
array C[1..N+1] of int;    int i, j, C, s;
C[N+1] = 0; i = N;
while (i >= 1) {
    j = i; s = M-L[i]; C = s^3 + C[i+1];
    while (j <= N and s >= 0) {
        C = min (C, s^3 + C[j+1]);
        j = j+1; s = s-L[j]-1;
    }
    C[i] = C; i = i-1;
}
```

(c) From the above program, we see that each $C[i]$ can be computed in $O(N)$ time, given the $C[j]$ values for $j \geq i$. Thus the overall time needed is $O(N^2)$.

Since we only store the $C[i]$ values, and a constant number of variables to keep track of the current values of i, j and $s_{i,j}$, the space needed is $O(N)$.

$\dashv$