CHENNAI MATHEMATICAL INSTITUTE

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

Entrance Examination, 15 May 2019

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. Let $L_1 := \{ a^n b^m \mid m, n \geq 0 \text{ and } m \geq n \}$ and $L_2 := \{ a^n b^m \mid m, n \geq 0 \text{ and } m < n \}$. The language $L_1 \cup L_2$ is:

   (a) regular, but not context-free
   (b) context-free, but not regular
   (c) both regular and context-free
   (d) neither regular nor context-free

   Answer:

   (c) Both regular and context-free.

   $L_1 \cup L_2 = \{a^n b^m \mid m, n \geq 0 \text{ and } (m \geq n \text{ or } m < n)\} = \{a^n b^m \mid m, n \geq 0\}$. This language is regular (accepted by the automaton in Figure 1), and hence context-free.

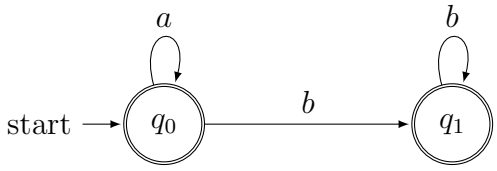


Figure 1: Automaton for $L_1 \cup L_2$.

⊣

2. Let A be an NFA with n states. Which of the following is necessarily true?

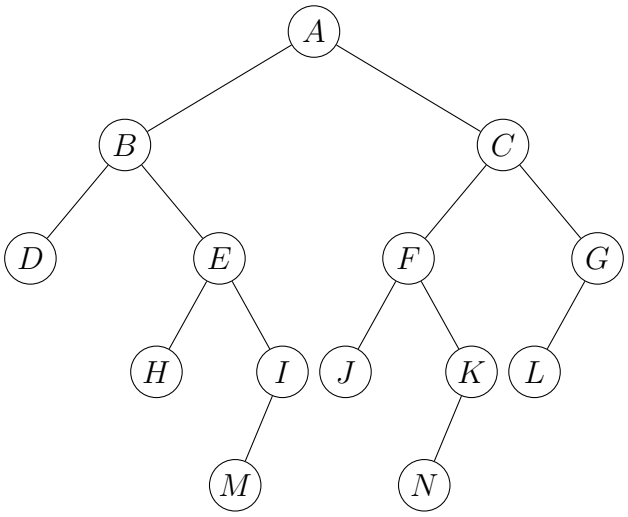
   (a) The shortest word in $L(A)$ has length at most $n-1$.
   (b) The shortest word in $L(A)$ has length at least n.
   (c) The shortest word *not* in $L(A)$ has length at most $n-1$.
   (d) The shortest word *not* in $L(A)$ has length at least n.

   Answer:

   (a) The shortest word in $L(A)$ has length at most $n-1$.

   Assume that $L(A)$ is non-empty. If A accepts a word of length $\geq n$, there is an accepting run of A that has at least one loop. If we cut out all the loops in this run, we get an accepting run (on some other word) that visits each of the n states at most once, and hence there are at most $n-1$ transitions, meaning there is a word in $L(A)$ of length at most $n-1$. Thus the shortest word in $L(A)$ has length at most $n-1$. ⊣

3. Suppose that the figure to the right is a binary search tree. The letters indicate the names of the nodes, not the values that are stored. What is the predecessor node, in terms of value, of the root node A?



(a) D

(b) H

(c) I

(d) M

Answer:

(c) I

The predecessor of A is the rightmost node in the left subtree of A. ⊣

4. There are five buckets, each of which can be open or closed. An arrangement is a specification of which buckets are open and which are closed. Every person likes some of the arrangements and dislikes the rest. You host a party, and amazingly, no two people on the guest list have the same likes and dislikes. What is the maximum number of guests possible?

(a) 5^2 (b) $\binom{5}{2}$ (c) 2^5 (d) 2^{2^5}

Answer:

(d) 2^{2^5}

Let us say a person's *profile* is a function from the set of all arrangements to $\{0, 1\}$ (indicating whether a particular arrangement is liked by that person or not). There are 2^5 possible arrangements. Thus there are 2^{2^5} possible profiles. Two people with the same profile have the same likes and dislikes, and thus, no two guests have the same profile. Thus the maximum number of guests possible is 2^{2^5}. ⊣

5. Let $\pi = [x_1, x_2, \ldots, x_n]$ be a permutation of $\{1, 2, \ldots, n\}$. For $k < n$, we say that π has its first ascent at k if $x_1 > x_2 \ldots > x_k$ and $x_k < x_{k+1}$. How many permutations have their first ascent at k?

(a) $\binom{n}{k}$ - $\binom{n}{(k+1)}$ (b) $\frac{n!}{k!} - \frac{n!}{(k+1)!}$ (c) $\frac{n!}{(k+1)!} - \frac{n!}{(k+2)!}$ (d) $\binom{n}{(k+1)}$ - $\binom{n}{(k+2)}$

Answer:

(b) $\frac{n!}{k!} - \frac{n!}{(k+1)!}$

For $i \in \{1, \ldots, n\}$, let P_i be the set of all permutations $x_1, x_2, \ldots, x_n$ for which $x_1 > x_2 > \cdots > x_i$.

A permutation in P_i is generated by choosing i elements for the first i positions (which are sorted in descending order) and then choosing a permutation of the remaining $n-i$ elements. So $|P_i| = \binom{n}{i} \cdot (n-i)! = \frac{n!}{i!}$.

The desired answer is $|P_k| - |P_{k+1}| = \frac{n!}{k!} - \frac{n!}{(k+1)!}$. ⊣

6. Suppose you alternate between throwing a normal six-sided fair die and tossing a fair coin. You start by throwing the die. What is the probability that you will see a 5 on the die before you see tails on the coin?

(a) $\frac{1}{12}$ (b) $\frac{1}{6}$ (c) $\frac{2}{9}$ (d) $\frac{2}{7}$

Answer:

(d) $\frac{2}{7}$

You stop when you see a 5 (probability 1/6) or tails (probability 1/2). You continue if you see neither (probability $1 - (1/6 + 1/2) = 5/12$). So the overall probability of stopping with 5 before tails is

$$1/6 + 5/12 \cdot 1/6 + (5/12)^2 \cdot 1/6 + \cdots .$$

This is an infinite geometric series with $a = 1/6$, $r = 5/12$, so the sum is

$$\frac{a}{1-r} = \frac{1/6}{1-5/12} = \frac{1/6}{7/12} = \frac{12}{6 \cdot 7} = \frac{2}{7}.$$

⊣

7. An interschool basketball tournament is being held at the Olympic sports complex. There are multiple basketball courts. Matches are scheduled in parallel, with staggered timings, to ensure that spectators always have some match or other available to watch. Each match requires a team of referees and linesmen. Two matches that overlap require disjoint teams of referees and linesmen. The tournament organizers would like to determine how many teams of referees and linesmen they need to mobilize to effectively conduct the tournament. To determine this, which graph theoretic problem do the organizers have to solve?

(a) Find a minimal colouring.

(b) Find a minimal spanning tree.

(c) Find a minimal cut.

(d) Find a minimal vertex cover.

Answer:

(a) Find a minimal colouring.

Matches are nodes. There is an edge between two matches if they overlap. Each team of referees and linesmen is a colour. A valid colouring assigns disjoint teams to overlapping matches. A minimal colouring determines the smallest number of such teams that are needed. ⊣

8. We have constructed a polynomial time reduction from problem A to problem B. Which of the following is *not* a possible scenario?

   (a) We know of polynomial time algorithms for both A and B.

   (b) We only know of exponential time algorithms for both A and B.

   (c) We only know an exponential time algorithm for A, but we have a polynomial time algorithm for B.

   (d) We only know an exponential time algorithm for B, but we have a polynomial time algorithm for A.

   Answer: (c)

   If we have a polynomial time solution for B, the reduction gives us a polynomial time solution for A. So it cannot be the case that we only know an exponential time algorithm for problem A. ⊣

The next two questions refer to the following program.

In the code below `reverse(A,i,j)` takes an array `A`, indices `i` and `j` with `i` $\leq$ `j`, and reverses the segment `A[i]`, `A[i+1]`,. . . ,`A[j]`. For instance if `A = [0,1,2,3,4,5,6,7]` then, after we apply `reverse(A,3,6)`, the contents of the array will be `A = [0,1,2,6,5,4,3,7]`.

```
function mystery (A[0..99]) {
  int i,j,m;
  for (i = 0; i < 100; i++) {
    m = i;
    for (j = i; j < 100, j++) {
      if (A[j] > A[m]) {
        m = j;
      }
    }
    reverse(A,i,m);
  }
  return;
}
```

9. When the procedure terminates, the array `A` has been:

   (a) Sorted in descending order (b) Sorted in ascending order

   (c) Reversed (d) Left unaltered

   Answer:

   (a) Sorted in descending order

   This procedure is call *pancake sort*. Each iteration (of the outer `for` loop) moves the maximum value in `A[i..99]` to `A[i]`.

   ⊣

10. The number of times the test `A[j] > A[m]` is executed is:

    (a) 4950 (b) 5050 (c) 10000 (d) Depends on the contents of `A`

Answer:

(b) 5050

In the first iteration, it is tested 100 times. As we proceed, it is tested $100+99+\cdots+1$ times, so the answer is 5050. ⊣

Part B

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

1. Consider an alphabet $\Sigma = \{a, b\}$.

   Let L_1 be the language given by the regular expression $(a + b)^*bb(a + b)^*$ and let L_2 be the language baa^*b.

   Define $L := \{u \in \Sigma^* \mid \exists w \in L_2 \text{ s.t. } uw \in L_1\}$.

   (a) Give an example of a word in L.

   (b) Give an example of a word *not in* L.

   (c) Construct an NFA for L.

   Answer:

   It can be seen that $L = L_1 \cup \{u \in \Sigma^* \mid u \text{ ends in a } b\}$.

   (a) $abba$ and b are words in L.

   (b) aba is a word not in L.

   (c) The NFA for L is given in Figure 2.

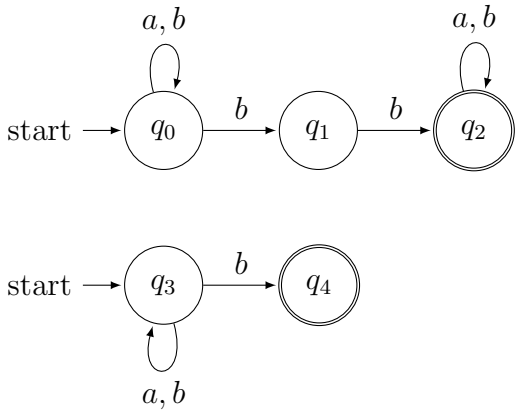


Figure 2: Automaton for L.

⊣

2. Let us assume a binary alphabet $\Sigma = \{a, b\}$. Two words $u, v \in \Sigma^*$ are said to be conjugates if there exist $w_1, w_2 \in \Sigma^*$ such that $u = w_1 w_2$ and $v = w_2 w_1$. Prove that u and v are conjugates if and only if there exists $w \in \Sigma^*$ such that $uw = wv$.

Answer:

Suppose u and v are conjugates, i.e. there exist w_1, w_2 such that $u = w_1 w_2$ and $v = w_2 w_1$. Then, taking $w = w_1$, we have $uw = w_1 w_2 w_1 = wv$.

Conversely, suppose there is w such that $uw = wv$. There are two cases to consider:

$|w| \leq |u|$**:** This means w is a prefix of u, say $u = wx$. Similarly, w is a suffix of v, say $v = yw$. Since $uw = wv$, we have $wxw = wyw$, thus $x = y$. Thus $u = wx$ and $v = xw$, and hence u and v are conjugates. (Note that in the case $|w| = |u|$, we have $w = u = v$ and $x = y = \varepsilon$.)

$|w| > |u|$**:** This means that u is a prefix of w, say $w = ux$. Similarly $w = yv$. Thus, $uyv = uw = wv = uxv$, and hence $x = y$. We thus get $ux = xv$, with $|x| < |w|$. We iterate this procedure until we get a small enough x that falls under the previous case.

$\dashv$

3. There is a party of n people. Each attendee has at most r friends in the party. The friend circle of a person includes the person and all her friends. You are required to pick some people for a party game, with the restriction that at most one person is picked from each friend circle. Show that you can pick $\frac{n}{r^2+1}$ people for the game.

Answer:

Consider an undirected graph where each vertex represents a person and there is an edge between two vertices if the corresponding people are friends. Suppose we pick two people who are at distance ≤ 2 from each other. Then they are either friends or share a friend, so they have overlapping friend circles (which could be the same circle). Thus we should ensure that no two people we pick for the party game are at distance ≤ 2 from each other.

We can now follow a simple strategy: pick an arbitrary vertex, then exclude everything else in the sphere of radius 2 around the picked vertex. The sphere will have at most $r^2 + 1$ vertices, of which we have picked one. We continue similarly for the remaining vertices to pick at least $\frac{n}{r^2+1}$ vertices. From any friend circle, if we have picked one person, all others in that circle are excluded (since they are within distance 2), and thus our restriction for the party game is satisfied.

$\dashv$

4. Consider the assertion: *Any* connected undirected graph G with at least two vertices contains a vertex v such that deleting v from G results in a connected graph.

   Either give a proof of the assertion, or give a counterexample (thereby disproving the assertion).

   Answer:

   The assertion is true. Consider a longest path $v_0 v_1 \dots v_t$ in G. The vertex v_0 can play the role of the vertex v in the question. Suppose deleting v_0 results in a graph with two or more components. All of $v_1, v_2, \dots, v_t$ will be together in one component. There is a vertex u in a different component of $G - v_0$ such that $uv_0v_1 \dots v_t$ is a longer path in G, a contradiction. $\dashv$

5. In the land of Twitter, there are two kinds of people: knights (also called *outragers*), who always tell the truth, and knaves (also called *trolls*), who always lie. It so happened that a person with handle `@anand` tweeted something offensive. It was not known whether `@anand` was a knight or a knave. A crack team, headed by Inspector Chitra, rounded up three suspects and interrogated them.

   The first interrogation went as follows.

Chitra	:	What do you know about `@anand`?
Suspect 1	:	`@anand` once claimed that I was a knave.
Chitra	:	Are you by any chance `@anand`?
Suspect 1	:	Yes.

   The second interrogation:

Chitra	:	Have you ever claimed you were `@anand`?
Suspect 2	:	No.
Chitra	:	Did you ever claim you are *not* `@anand`?
Suspect 2	:	Yes.

   The third suspect arrived with a defense lawyer (also on Twitter):

Lawyer	:	My client is indeed a knave, but he is not `@anand`.
Suspect 3	:	My lawyer always tells the truth.

   Which of the above suspects are innocent, and which are guilty? Explain your reasoning.

   Answer:

   If Suspect 1 were Anand, then since he answers truthfully to the second question, he is a knight, but then his first reply means that he once claimed he was a knave, which knights will not do. Hence Suspect 1 is not Anand.

   If Suspect 2 were Anand, then he is either a knight or a knave. If he were a knight, then the answer to question 2 means he claimed he was not Anand, which is a lie. Hence he is a knave. But then the answer to question 1 is a lie, which means he has once claimed he is Anand, which a liar will not do. This contradiction means that he is not Anand.

   Now Suspect 3 cannot be a knight, since he says his lawyer is truthful, and the lawyer says that he is a knave. So Suspect 3 is a knave. And so what he said is false, and his lawyer is a knave. But then the lawyer has uttered a falsehood. Of the conjuction he uttered, at least one conjunct is false. But the first conjunct is true, so the second is false. And Suspect 3 is indeed Anand!

   ⊣

6. Let A be an $n \times n$ matrix of integers such that each row *and* each column is arranged in ascending order. We want to check whether a number k appears in A. If k is present, we should report its position—that is, the row i and column j such that $A(i, j) = k$. Otherwise, we should declare that k is not present in A.

   (a) Describe an algorithm that solves this problem in time $O(n \log n)$. Justify the complexity of your algorithm.

   (b) Describe an algorithm that solves this problem by examining at most $2n$ values in A. Justify the complexity of your algorithm.

   (c) For both algorithms, describe a worst-case input where k is present in A.

   Answer:

   (a) Since each row is sorted, we can do binary search on each of the n rows. Since binary search works in $O(\log n)$ time, the algorithm works in $O(n \log n)$ time.

   (b) Compare k against the top right corner element $A(1, n)$. If $k = A(1, n)$, we are done. Else there are two cases to consider.

   Case 1: $k < A(1, n)$**.** Then k cannot appear in the last column (since $A(1, n)$ is the smallest entry in that column) so we have to search for A in rows 1 to n, columns 1 to $n - 1$.

   Case 2: $k > A(1, n)$**.** Then k cannot appear in the first row (since $A(1, n)$ is the largest entry in that row) so we have to search for A in rows 2 to n, columns 1 to n.

   In the next step, we compare k against the top right corner of the remaining matrix of interest (the top right corner is $A(1, n - 1)$ if Case 1 holds and $A(2, n)$ if Case 2 holds) and repeat the earlier analysis.

   With each step we reduce the matrix of interest by pruning eliminating one row or one column. Since we started with n rows and n columns, within $2n$ steps (actually $2(n - 1)$ steps) we would have pruned the matrix down to a single row and column, i.e. a single element.

   (c) For the $O(n \log n)$ algorithm, the worst case input is when k appears in the last row (assuming we search each row in order from top to bottom).

   For the algorithm with $2n$ steps, since the search proceeds from the top right to the bottom left, the search goes on longest if the element is at the bottom left corner, $A(n, 1)$.

   $\dashv$

7. A college professor gives several quizzes during the semester, with negative marking. He has become bored of the usual *"Best M out of N quizzes"* formula to award marks for internal assessment. Instead, each student will be evaluated based on the sum of the best contiguous segment (i.e., no gaps) of marks in the overall sequence of quizzes. However, the student is allowed to drop upto K quizzes before calculating this sum.

   Suppose a student has scored the following marks in 10 quizzes during the semester.

Quiz	1	2	3	4	5	6	7	8	9	10
Marks	6	−5	3	−7	6	−1	10	−8	−8	8

   Without dropping any quizzes, the best segment is quiz 5—7, which yields a total of 15 marks. If the student is allowed to drop upto 2 quizzes in a segment, the best segment is quiz 1—7, which yields a total of 24 marks after dropping quizzes 2 and 4. If the student is allowed to drop upto 6 quizzes in a segment, the best total is obtained by taking the entire list and dropping all 5 negative entries, yielding 33 marks.

   For $1 \leq i < N$, $0 \leq j \leq K$, let $B[i][j]$ denote the maximum sum segment ending at position i with at most j marks dropped.

   (a) Write a recursive formula for $B[i][j]$.

   (b) Explain how to calculate, using dynamic programming, the score the professor needs to award each student.

   (c) Describe the space and time complexity of your dynamic programming algorithm.

Answer:

(a) The original array is $Marks[1 \cdots N]$. We build a two-dimensional array $B[0 \cdots N][0 \cdots K]$, where $B[i][j]$ denotes the maximum segment sum ending at (*and including*) index i, while dropping up to j quizzes. The desired solution is

$$\max\{B[i][K] \mid 1 \leq i \leq N\}.$$

The base case is when we are not allowed to drop any quizzes, and the recurrence is as follows:

$$\begin{aligned} B[0][0] &= 0 \\ B[i+1][0] &= \max\{B[i][0] + Marks[i+1], Marks[i+1]\} \end{aligned}$$

For the inductive case, the recurrence is as follows:

$$\begin{aligned} B[0][j+1] &= 0 \\ B[i+1][j+1] &= \max\{B[i-l][j+1-l] + Marks[i+1] \mid 0 \leq l \leq \min\{i, j+1\}\} \end{aligned}$$

Index $i+1$ has to be definitely included. The previous index to be included could be i, in which case we can drop up to $j+1$ indices before it, or we drop i and include $i-1$, in which case we are only allowed to drop j indices before it, and so on.

(b) For $i, j > 0$, computing $B[i][j]$ requires knowing entries $B[i'][j']$ for $i' < i$ and $j' \leq j$. We can either fill the table row by row, left to right, or column by column, top to bottom. For the top row, we have $B[0][j] = 0$ for all j. For the leftmost column, we can start with $B[0][0] = 0$ and fill the rest of the column using the base case recurrence for $B[i+1][0]$.

(c) The table has size $(N + 1) \times (K + 1)$ so the space complexity is $O(KN)$. To compute entry $B[i][j]$, we need to scan $\min(i, j+1)$ entries in the table. Since i is bounded by N, j is bounded by K, and $K \leq N$, $\min(i, j+1)$ is bounded by $K+1$ and asymptotically it takes time $O(KN \cdot K) = O(K^2N)$ to fill the entire table.

⊣