1 Introductory Story

Why suddenly a course in mathematics at the outset of a computer science course? As it so happens, any serious computer science course to start with has a discrete mathematics course. So, it must be useful mathematics for computer science students. Discrete mathematics usually means the mathematics of discrete sets. You will appreciate the role of discrete mathematics in computer science as you go along - not only for the duration of this course but for subjects like data structures, design and analysis of algorithms, logic, automata, languages and computation which you will study in coming semesters.

Though the scope of discrete mathematics is broad, we will focus on three aspects of discrete mathematics during the course of this semester. The following is to give you a broad overview of what we will (try to) cover, not necessarily in the following order.

Combinatorics: Asymptotic notations, functions and counting using functions, equivalences, ordered sets, basic counting rules, derangement, recurrences, generating functions, pigeonhole principle, Pólya’s theory of counting (if time permits), Ramsey theory (if time permits)

Graph Theory: basic introduction, graph traversals, graph acyclicity, topological sorting, strongly connected component, shortest path, graph isomorphism, graph planarity, Eulerian path and tour, Hamiltonian path and tour, counting in graphs, introduction to clique, vertex cover, independent set, dominating set, chromatic number, etc. of a graph.

Logic: Proposition, negation, conjunction, disjunction, universal and existential quantifiers, Implications - conditional, biconditional; tautology, logical equivalence, propositional calculus - tautologies, adequate set of connectives, axiom system of propositional calculus, compactness and resolution, formal reducibility, soundness and completeness.

Too much, isn’t it? Combinatorics, Graph Theory and Logic are by themselves subjects that demand separate treatment. So, how to cope? Enjoying the subject might lessen the pain!

Now, to some basics. We will meet twice a week for two lectures each of two period duration. They are Tuesday and Thursday both from 9:30 am to 11:10 am. We will have tutorials and homeworks. The marks distribution are as follows:

1. Assignment and class test: 25

2. Mid semester exam: 25
3. End semester exam: 50

The course web-page will be maintained at:
http://www.isical.ac.in/~arijit/courses/course.html.

There is no alternative to reading books. The following references [1, 2, 3, 4, 5, 6, 7, 8, 9] will be used during the course.

2 A few warm up problems

Let us start with a few warm up problems that would set the ball rolling!

2.1 Counting regions in the plane

Problem 1 A set $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ of $n$ lines in the plane are said to be in general position if (i) no two lines are parallel and (ii) no three lines intersect at a common point. Find out the number of vertices, edges and regions $\mathcal{L}$ will create on the plane.

Counting of the vertices is easy as all $\binom{n}{2}$ lines will intersect to create $\binom{n}{2}$ vertices. For the number of edges, notice that a line $\ell_i \in \mathcal{L}$ will be intersected by all the other $n-1$ lines creating $n$ edges per line. Thus, the total count is $n^2$. Next, we try to find the number of regions $\mathcal{R}^n$ formed by $n$ lines in general position. What better technique to use than induction?

The idea is as follows. Let the number of regions formed after adding the $(n-1)^{th}$ line be $\mathcal{R}^{n-1}$. We try to find out the number of new regions $r_n$ formed after adding the $n^{th}$ line. Then, surely $\mathcal{R}^n = \mathcal{R}^{n-1} + r_n$. We now try to find out $r_n$.

![Figure 1: Region formation for lines in general position.](image)

(a) Regions in the presence of $n-1$ lines.  
(b) Regions after addition of one more line.

Lemma 1 Adding the $n^{th}$ line to the already existing $(n-1)$ lines, the number of new regions added is $n$.

Proof: The $n^{th}$ line can cut each of the previously existing $(n-1)$ lines once. The line after each such intersection breaks an existing region into two and gives rise to
a new region after the intersection; i.e. we have \((n-1)\) regions. Add to that one more region formed before the first intersection. Thus, we have \(n\) new regions. See Fig. 1 for an illustration.

**Theorem 1** The number of regions in the plane formed by \(n\) lines in general position is \(\frac{n(n+1)}{2} + 1\).

**Proof:** We proceed to prove this by induction. Let us first look into the base case.

**Base case:** The first line introduces two new regions.

**Induction Hypothesis:** For every \(m > 1\), the number of new regions added is \(m\).

**Induction Step:** Assuming that \(m\) lines have added \(m\) new regions, the \((m+1)th \) line by Lemma 1 adds \((m+1)\) new regions.

So, the total number of regions \(R^n = 2 + \sum_{i=2}^{n} i = \frac{n(n+1)}{2} + 1\).

### 2.2 A few more problems

We study two more problems here.

**Problem 2** Let \(R\) be a unit square (i.e. a square with side length equal to one). Let \(P\) be a set of 5 points inside or on the boundary of \(R\). There will be \(\binom{5}{2}\) distances. Of these distances, there will be a minimum distance, say \(d_{\text{min}}\). How big can \(d_{\text{min}}\) be?

Problem 2 is a straight application of pigeonhole principle where \(R\) can be divided into four equal regions by a horizontal and vertical line going through the middle of \(R\). With five points, \(\exists\) a square that will have at least \(\lceil \frac{5}{4} \rceil = 2\) points in it. These two points can at most be at a distance of \(\frac{1}{\sqrt{2}}\). Thus, \(d_{\text{min}} \leq \frac{1}{\sqrt{2}}\).

**Problem 3** Let us have 50 chits with the \(i^{th}\) chit having the number \(i\) written on it, \(1 \leq i \leq 50\). The chits are now folded so that the number written on it cannot be seen. The folded chits are put into a basket. Now, we keep doing the following. At each step, we pick up two chits from the basket, open them, see the numbers (say \(a\) and \(b\)), throw away these two chits, write \(|a-b|\) on a chit and put it into the basket. If we keep doing this for 49 times, we will be left with only one chit in the basket. What can we say about the number on this chit: odd, even, indeterminate?

This problem is a classic example of loop invariance that you would need very much for proving correctness of algorithms while you study design and analysis of algorithms. The solution was discussed in the class.

### 2.3 Problems on sorting

We will look into the problem of sorting and study two methods for it.
Problem 4  Let $S = \{s_1, \ldots, s_n\}$ be $n$ numbers. The problem is to sort $S$ in an ascending order.

**Method 1:** At the first step, we find the minimum number $s_{\text{min}} \in S$ taking $n - 1$ comparisons. In the second step, we find the minimum number in $S \setminus s_{\text{min}}$ taking $n - 2$ comparisons. Thus, we can sort $S$ taking $(n - 1) + (n - 2) + \ldots + 1 = \frac{n(n-1)}{2}$. This way of sorting $S$ is a recursive technique. We keep doing the same thing on reduced instances of the problem. So, we can also formulate the number of comparisons as follows, where $T(n)$ denotes the number of comparisons needed for sorting $n$ numbers:

$$T(n) = \begin{cases} T(n-1) + (n-1) & \text{if } n > 2; \\ 1 & \text{if } n = 2. \end{cases}$$

**Method 2:** At the first step, we find the median $s_{\text{med}} \in S$ taking $c' \cdot n$ comparisons, where $c'$ is a constant. (Finding median using only $c' \cdot n$ comparisons is a non-trivial task. You will learn it in your design and analysis of algorithms course.) We put $s_{\text{med}}$ in the middle location of the array and divide $S$ into two parts $S_L$ and $S_R$. All numbers in $S_L$ are less than $s_{\text{med}}$ and all numbers in $S_R$ are greater than $s_{\text{med}}$. We keep doing this method recursively. The number of comparisons needed in the first step equals $c' \cdot n + (n - 1) = c \cdot n$, where $c$ is a suitable constant. Notice that, $S_L$ and $S_R$ are roughly of size $\frac{n}{2}$. So, the number of comparisons equals $c \cdot n + 2c \cdot \frac{n}{2} + \ldots$. Assuming, $n = 2^k$, where $k$ is a positive integer, the above sum equals $c \cdot n \log n$. We can again formulate this like before:

$$T(n) = \begin{cases} 2T(n/2) + c \cdot n & \text{if } n > 2; \\ 1 & \text{if } n = 2. \end{cases}$$

This sort of equation is called a recurrence equation. We will see later how to solve them. But, the point we take from here is that the time or comparisons needed by recursive methods can be formulated as recurrence equations.

**References**


