UNIT – V

Unit-V/Lecture-01

Computationability

Computational complexity theory is a branch of the theory of computation in theoretical computer science and mathematics that focuses on classifying computational problems according to their inherent difficulty, and relating those classes to each other.
A problem is regarded as inherently difficult if its solution requires significant resources, whatever the algorithm used.

A computational problem can be viewed as an infinite collection of instances together with a solution for every instance. The input string for a computational problem is referred to as a problem instance, and should not be confused with the problem itself. In computational complexity theory, a problem refers to the abstract question to be solved. In contrast, an instance of this problem is a rather concrete utterance, which can serve as the input for a decision problem.

When considering computational problems, a problem instance is a string over an alphabet. Usually, the alphabet is taken to be the binary alphabet (i.e., the set {0,1}), and thus the strings are bitstrings. As in a real-world computer, mathematical objects other than bitstrings must be suitably encoded. For example, integers can be represented in binary notation, and graphs can be encoded directly via their adjacency matrices, or by encoding their adjacency lists in binary.

Decision problems are one of the central objects of study in computational complexity theory. A decision problem is a special type of computational problem whose answer is either yes or no, or alternately either 1 or 0. A decision problem can be viewed as a formal language, where the members of the language are instances whose output is yes, and the non-members are those instances whose output is no. The objective is to decide, with the aid of an algorithm, whether a given input string is a member of the formal language under consideration. If the algorithm deciding this problem returns the answer yes, the algorithm is said to accept the input string, otherwise it is said to reject the input.

To measure the difficulty of solving a computational problem, one may wish to see how much time the best algorithm requires to solve the problem. However, the running time may, in general, depend on the instance. In particular, larger instances will require more time to solve. Thus the time required to solve a problem (or the space required, or any measure of complexity) is calculated as a function of the size of the instance. This is usually taken to be the size of the input in bits. Complexity theory is interested in how algorithms scale with an increase in the input size.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Unit-V/Lecture-02

NP-Hard and NP- Complete Problems

P: Problems in class P can be solved with algorithms that run in polynomial time.
Let you have an algorithm that finds the smallest integer in an array. One way to do this is by iterating over all the integers of the array and keeping track of the smallest number you've seen up to that point. Every time you look at an element, you compare it to the current minimum, and if it's smaller, you update the minimum.
How long does this take? Let's say there are n elements in the array. For every element the algorithm has to perform a constant number of operations. Therefore we can say that the algorithm runs in O (n) time, or that the runtime is a linear function of how many elements are in the array. So this algorithm runs in linear time.
You can also have algorithms that run in quadratic time (O (n^2)), exponential time (O (2^n)), or even logarithmic time (O(log n)). Binary search (on a balanced tree) runs in logarithmic time because the height of the binary search tree is a logarithmic function of the number of elements in the tree.
If the running time is some polynomial function of the size of the input**, for instance if the algorithm runs in linear time or quadratic time or cubic time, then we say the algorithm runs in polynomial time and the problem it solves is in class P.

NP: Now there are a lot of programs that don't (necessarily) run in polynomial time on a regular computer, but do run in polynomial time on a nondeterministic Turing machine. These programs solve problems in NP, which stands for nondeterministic polynomial time. A nondeterministic Turing machine can do everything a regular computer can and more.*** This means all problems in P are also in NP.

An equivalent way to define NP is by pointing to the problems that can be verified in polynomial time. This means there is not necessarily a polynomial-time way to find a solution, but once you have a solution it only takes polynomial time to verify that it is correct.

Say P = NP, which means any problem that can be verified in polynomial time can also be solved in polynomial time and vice versa.

NP-hard: What does NP-hard mean? A lot of times you can solve a problem by reducing it to a different problem. I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.")

If a problem is NP-hard, this means I can reduce any problem in NP to that problem. This means if I can solve that problem, I can easily solve any problem in NP. If we could solve an NP-hard problem in polynomial time, this would prove P = NP.

NP-complete: A problem is NP-complete if the problem is both

NP-hard, and in NP.

* A technical point: O(n) actually means the algorithm runs in asymptotically linear time, which means the time complexity approaches a line as n gets very large. Also, O(n) is technically an upper bound, so if the algorithm ran in sublinear time you could still say it's O(n), even if that's not the best description of it.

** Note that if the input has many different parameters, like n and k, it might be polynomial in n and exponential in k

Unit-V/Lecture-03
AVL Tree
Unit-V/Lecture-04
B Tree
Unit-V/Lecture-05
Graph Traversal-DFS
Unit-V/Lecture-06
Graph Traversal-BFS
Unit-V/Lecture-07
Binary Search Tree
Unit-V/Lecture-08
2-3 trees

we dont take any liability for the notes correctness.

http://www.rgpvonline.com