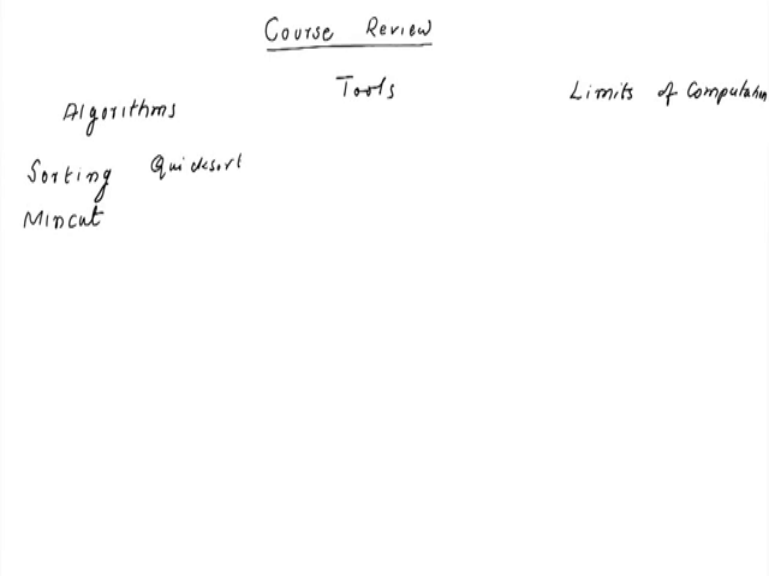
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 40
Summary

So, welcome to the last week of this course on randomized algorithms so, this week we will see, what all are the various things we have learnt during the course. It is a recap of whatever we have seen so far.

(Refer Slide Time: 00:43)



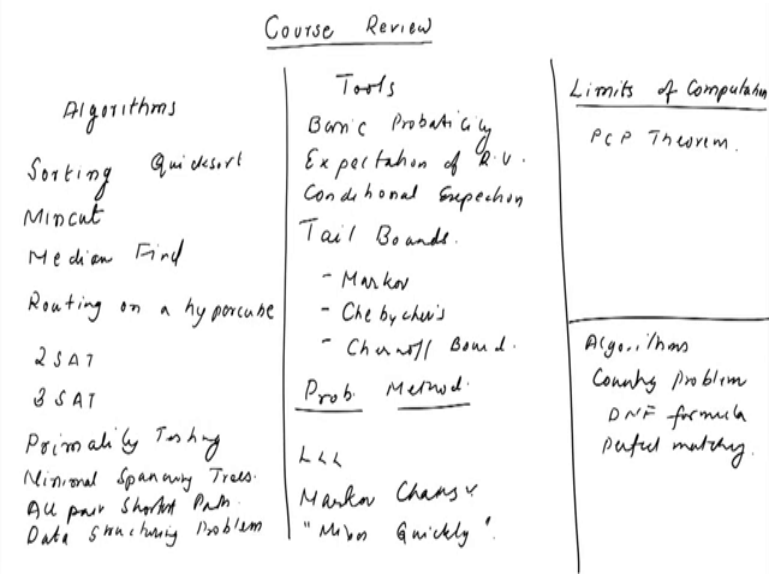
So, this is a course on randomized algorithms. So, randomized algorithms is an interplay between randomness and algorithm or computation. So, there were three main components in the course; one you can think of it is algorithms and the second is tools from probability or just say name as tools, which is helping us design these algorithms and the third aspect was the limits of computation.

So, we used the tools from randomness or probability theory, which we use to create algorithms, to somehow argue about the inherent limits of computations as well. So, the in understanding what is computation, we used these tools. So, that is the third part that we had covered while looking at algorithms, we had studied many problems. So, we looked at the problem of sorting, where in you are given a collection of numbers and you want to get this in the sorted order and the quick sort algorithm said that pick a random

element instead of spending time on analyzing, which is the best element to compare with the other elements. We just decided that by picking an element randomly and partitioning the entire collection into numbers, which is smaller than and larger than this number. We could split it into two parts and then recursively analyze or recursively sort the components.

So, that was our quick sort algorithm and our tools from randomness or probability theory, essentially told us that quick sort will work in n log n time and then the other problems that we looked at were the Mincut problem. When we are looking at the Mincut problem, we look at a graph and we want to split the graph into two parts, split the vertices into two parts.

(Refer Slide Time: 02:58)



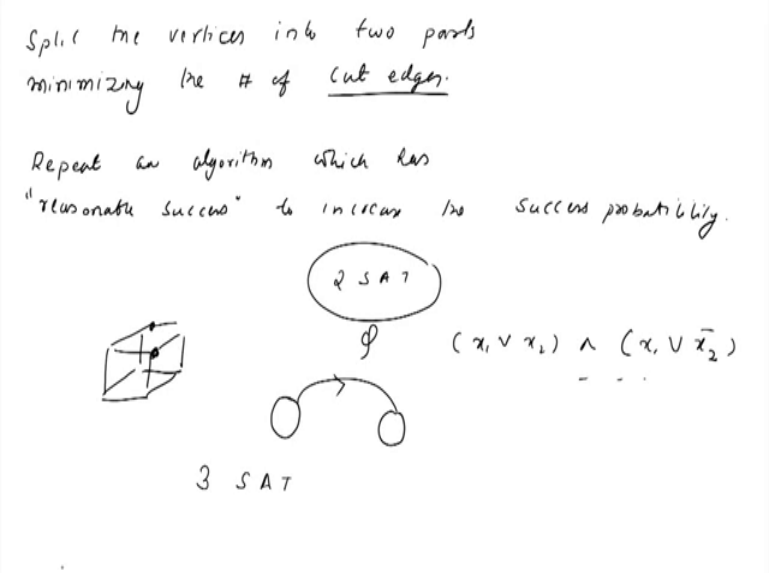
So, that there is minimum number of edges from one part to the other and then we had an algorithm to do that, we took a particular edge and contracted it and repeatedly did this till the entire graph was contracted into two vertices and between these two vertices, these contracted super vertices. The number of edges will be, there is a cut edge and that certainly is an upper bound on the size of the minimum cut. Any other cut can only be smaller than that particular cut and we argued that with sufficiently high probability, if we do this algorithm, there is a sufficiently high probability that we will actually find out the minimum cut and therefore, by repeating.

So, that was an important theme in this repeat an algorithm, which has reasonable success to increase the success probability. So, we looked at the Mincut problem and then we looked at the median find algorithm, where in you are given a collection of numbers and you wanted to find out the middle most number and in finding the median, what we argued is by choosing a sample from the entire collection of numbers and finding the median of the sample, helps us figure out the actual median we used some non trivial tools to do this.

So, amongst tools we had learnt about basic probability and we learned about expectation of random variables and conditional expectations and then we learnt about what are known as tail bounds and that what helped us answer questions regarding median. So, tail bounds essentially means; we looking at certain event ok. So, if you are looking at an event, it has a certain probability. The random variable associated with it, has a certain expectation. What is the probability that the random variable deviates by a large amount from its expected value.

So, we looked at Markov inequality, looked at Cheby chev's and we also looked at Chernoff bound, we used Cheby chev's inequality to show that this technique of extracting a subset from the elements of set, whose median we want to find out and using the median of the sample set in directing the median search helps us improve the performance of the algorithm and gives reasonable guarantees for the median find. So, we got a linear time algorithm for median find using Cheby chev's inequality. We looked at the routing problem on hyper cube.

(Refer Slide Time: 07:08)



So, here we were looking at an n dimensional hyper cube and each vertex, we are looking at permutation routing where in from each vertex, you are given some other vertex to which you have to send an information or sent a packet and we can use only the edges of the hyper cube in doing that and the input set of vertices is permuted to give, get an output set of vertices.

In other words, for every input, for every vertex of the graph, there is another unique vertex of the same graph to which you have to route a particular message. How many hops do we have to do on these graphs? In order to solve this problem we use randomness.

We said that we will send the packet from one particular vertex to an arbitrary vertex and from that arbitrary vertex we will send back to the destination vertex and by setting up the probabilistic setup, to do this we argued via Chernoff bounds that this method will bring down the time taken for communication, for permutation routing significantly. And we were able to provide bounds for how quickly does the permutation routing will work on a hyper cube if you are doing randomized routing.

And then we looked about probabilistic method; that is a tool where in by use of randomness we could show the existence of certain communitorial objects and we used tools like Lovazs local lemma in order to get stronger bounds.

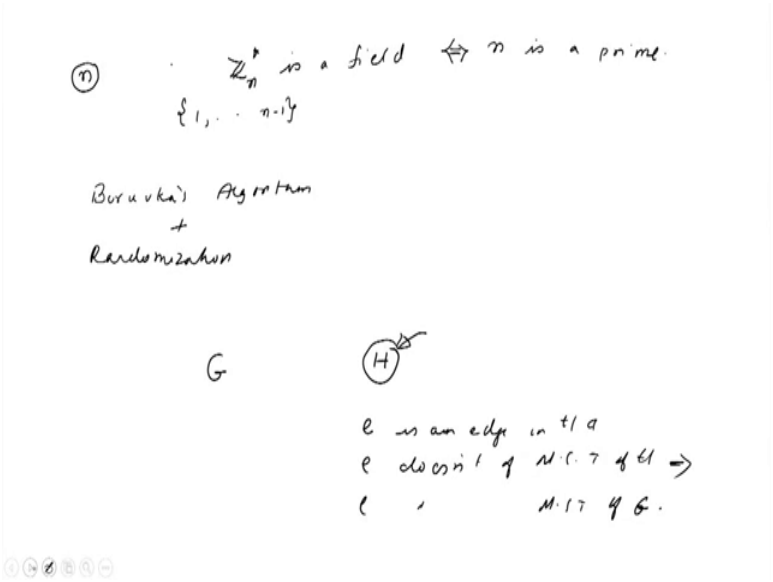
So, here you can think of these as algorithmic problems, if you wanted to find the existence of find certain objects. The first step in finding an object would be to prove its existence. So, probabilistic method gave us tools to prove the existence of certain objects and later on in some cases, we were able to de randomize, take away the probability and deterministically find those elements, find those objects which had certain properties and then we looked at the 2 SAT problem and the 3 SAT problem.

So, when you look at 2 SAT you are given a formula phi, we were given a Boolean formula in conjunctive normal form and we wanted to know whether this has a satisfying assignment or not. For this, we had an algorithm, which randomly chooses an assignment for the variables and keeps on changing the assignment in a random way. We argued that this changing the assignment in a particular way, which is dependent on the random choices, that we make. By doing so, we could argue that for a 2 SAT expressions this quickly finds a satisfying assignment.

The probability of finding a satisfying assignment, if there was a satisfying assignment was reasonably high and the repeating paradigm that is repeat algorithm, which is reasonable success helped us by increasing the probability of success. So, if we could show that the random choices of assignments and small tweaking of these assignments could lead to finding a satisfying assignment with some probability by repeating the same method multiple times we could enhance the success probability. And in order to argue that there is a reasonable probability of success, we used what are known as Markov chains. The assignments themselves were thought of some kind of states and then on these states you have some probability distribution, which means from one particular state, you could move to another state with certain probability and so on.

And we argued that you will reach a satisfying state; that means, all the clauses being satisfied with reasonably high probability. We use this method to give a significantly improved performance for 3 SAT as well, in case of 2 SAT we were able to show that the algorithm works in expected polynomial amount of time, whereas in case of 3 SAT, we improved the running time. Although, we got exponential time algorithm that was significantly better than the brute force exponential algorithm and while doing so, we learnt about the tool called Markov chains and then we looked at primality testing.

(Refer Slide Time: 12:18)



So, here we are given a number n and we wanted to quickly tell whether a number is prime or composite and we spent some time understanding, what are the properties of prime numbers. We learned a bit of number theory and we argued that a number is prime, then if you look at Z n star, which is all the numbers from 1 to n minus 1 module and arithmetic being modulo and multiplication, if n is prime, this is a field.

So, here when we say Z n star, we mean the number 1 to n minus 1 is a field if and only if n is a prime number and based on this we device some algorithms, some randomized algorithms, which could show, which could test for primality and then we looked at minimal spanning trees. It is a, given arbitrary graph in order to find a minimal spanning tree. It takes say n log n amount of time or e log e amount of time, where e is number of edges, but we were able to come up with an algorithm, which works in linear time.

So, that involved use of the Boruvka's algorithm coupled with randomization. So, we will do Boruvka phases for some amount of time. So, we will find certain edges that will, that is guaranteed to be there in the minimal spanning tree and then for a randomly sampled subset of the remaining graph we will find a minimal spanning tree. Anything which is not there in the minimal spanning tree of the randomized subset cannot be there in the minimal spanning tree of the original graph as well.

So, in other words suppose, you had a graph G and if you take a random subset of G, let us call it as H and if a particular edge, e is an edge in H and e does not belong to M S T

of H. This would imply that e does not belong to M S T of G as well. And this H by choosing this randomly, we were able to show that we could get a, an algorithm, which works in expected linear amount of time. We also looked at other problems on graphs, the other graph algorithm that we considered were the all pair shortest path, if you given two, if you given a graph and if you were given to find out the distance between any pair of vertices, we were able to compute this with a randomized algorithm.

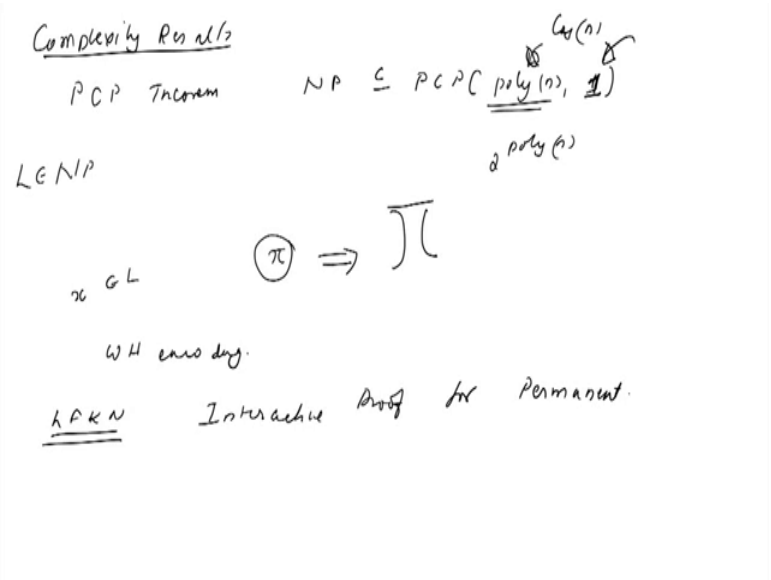
We related it to the problem of matrix multiplication or binary matrix multiplication and witnesses for binary matrix multiplication. The other problems that we looked at data structuring problems, we had some large collection of data. How do we do inserts, delete and other bookkeeping tasks associated with this data and we wanted to do queries on them we wanted to search, we want to know if is a particular element present or absent. And in order to do this, we looked at hashing we looked at random tree heaps and so on.

We also looked at counting problems. We wanted to look at the number of satisfiable assignments for a D N F formula, we wanted to look at the number of perfect matchings in a graph and we argued we had first converted the counting problem into a sampling problem. We said that, if we could approximately sample, if you could sample almost uniformly then we could approximately count and for sampling we again used Markov chains.

We argued that we could setup a Markov chain and if this Markov chain mixes quickly, then we can use it to sample elements and that gives an approximate or an almost uniform sampling and we argued that almost uniform sampling can be used to approximately count.

So, counting problems were solved using randomnization in this particular manner and while looking at limits of computation, we looked at P C P theorem. It was the main result we, main computational complexity result that we had discussed. So, P C P theorem, in some sense says that you look at any proof for membership from an N P language. So, take so were looking at P C P theorem.

(Refer Slide Time: 18:03)



So, we related it to the complexity class called N P. So, N P tells, I mean if a language L belongs to N P; that means, there is a proof for membership of strings. So, you look at a string x belonging to L, you can present its proof in a certain way so that a verifier can just read constantly many; can read polynominally bits, can read the proof, the proof is not very long and do some polynomial amount of computations.

By use of randomness, we showed how this proof can be converted into let us say an enlarged proof, let me just write it as large pi and this has a property that by examining just few bits in the proof; so, this proof is of course, going to be large. So, the variant of P C P theorem that we proved said that N P is contained inside P C P poly n comma 1, which means once a proof has been enlarged you need to query just constantly many bits. This 1 refers to constant. So, constantly many bits of the proof is all that you have to read in order to be reasonably convinced that the string x indeed belongs to the language and while doing so the proof will not become, let us say much larger than exponential.

So, here poly n means amount of randomness used, the amount of randomness is bounded by poly n and therefore, the total amount of locations that can be addressed is bounded by 2 raised to poly n. Of course, now there are better P C Ps, where this is bounded by log n, but we proved the weak version, wherein there is a blow up in the size of the proof, but although there is a blow up in the size of the proof. The verifier needs to

just check constantly many bits and the amount of randomness that he uses is bounded by poly n and to prove this we use the Walsh Hadamard encoding.

And we use the properties of Walsh Hadamard encoding which helped us decode by with high probability by just randomly choosing points from where you could read off the values of the proof or the bit positions in the proof. We also looked at the L F K N protocol, which essentially is an interactive proof for permanent. So, we were looking at the problem of computing the permanent of a matrix.

We argued that if you given a matrix and if somebody is extremely powerful in terms of computation, they can convince you by some number of rounds, polynomial amount of rounds of interactions and the protocol had the property that if the prover tries to convince you that the permanent of a matrix is different from what it actually is, even though the proof prover is infinitely powerful, you can detect the lie that the prover is making by adhering to the protocol. So, this is a short recap of whatever we had done in the course; all the best for the exams.