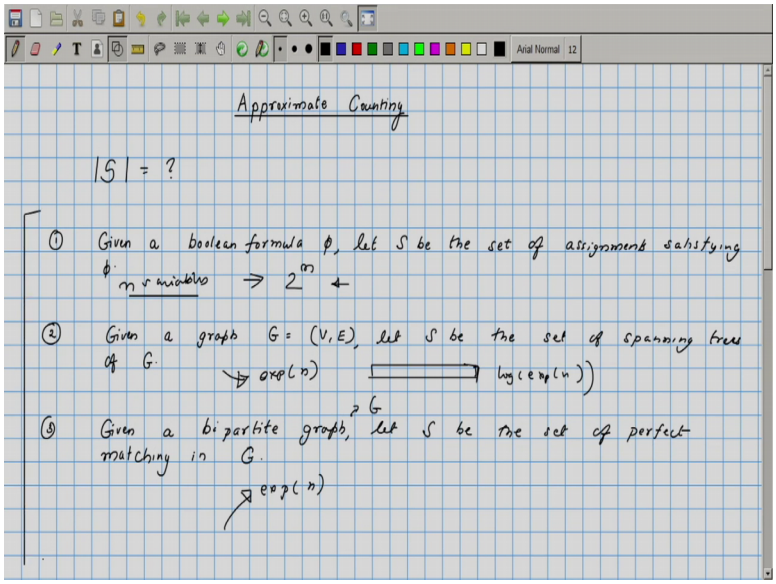
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 29
Introduction to Approximate Counting

In the coming few lectures we will learn about Approximate Counting.

(Refer Slide Time: 00:37)



So, we can look at the problem in the following way there is a set S and its description is given in some particular way and we need to know how many elements are there the set S? Now the presentation of S is extremely important how exactly is S given and that is going to have bearings on the computational cost associated with this counting. Let us look at couple of examples of such counting problems.

1 given a Boolean formula phi let S be the set of assignments satisfying phi ok. So, we need to count the number of satisfying assignments for a formula another kind of counting problem could be given a graph G equals V comma E, let S be the set of spanning trees of G. A 3rd example could be given a bipartite graph let S be the set of perfect matchings in.

So, let us say if you call this bipartite graph G you would not know the number of perfect matchings that are there in G. So, if you think of each of these as computational

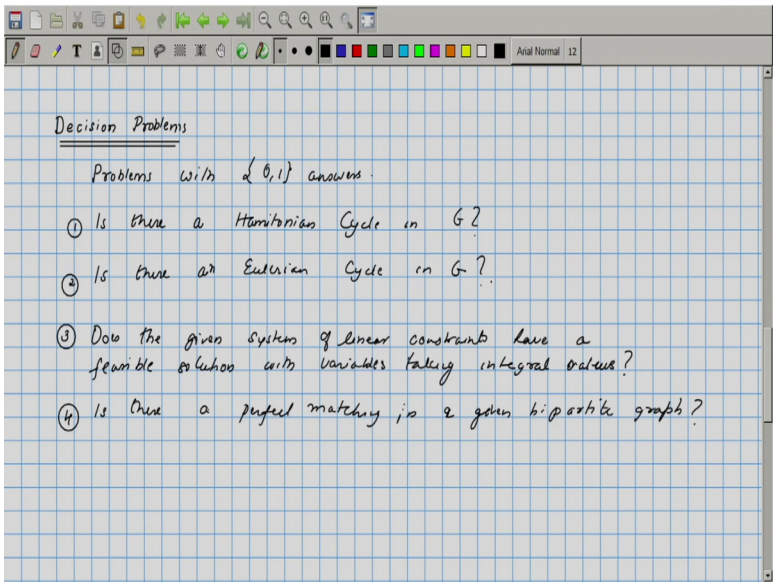
problems in the first case the input is a Boolean formula of a certain length and in the second case it is a graph of a certain size, in the third case it is a bipartite graph of a certain size and our answers depend on the formula and the graphs that are given is input.

But how large can these answers be? For a Boolean formula let us say n variables the answer the count could be as large as let us say 2 to the power n. And given a graph the set of spanning trees also could be exponential in the number of vertices and same for perfect matching ok, but these numbers and therefore, they can be output by some string of length log exp n ok, in the sense the output is going to be polynomially long in all these 3 cases.

When the count is extremely large that is it is prohibitively large to even display it I mean that is not a problem within our computational abilities. But see all these problems are the characteristic the answer the final answer is not too large it is polynomially long in the input size. Contrast this with the case where we had to list the set of all satisfying assignments and that would have been a very difficult task because there could be as large as 2 to the n satisfying assignments and there could be exponentially many spanning trees and there could be exponentially many perfect matchings. A related problem would be to sample uniformly from these exponentially large sets.

So, the set S as such could be exponentially large if we were able to count it that is interesting thing in many cases and sometimes we are more ambitious we want to list the set of elements in S that could be troublesome because the set could be very large. In that case we might reduce our ambition and say that we want to uniformly sample elements of S and there are some interesting relationship between sampling and counting. So, now, let us look at these problems a little more firmly what are the questions that we will be addressing and what are the requirements on the quality of our answers.

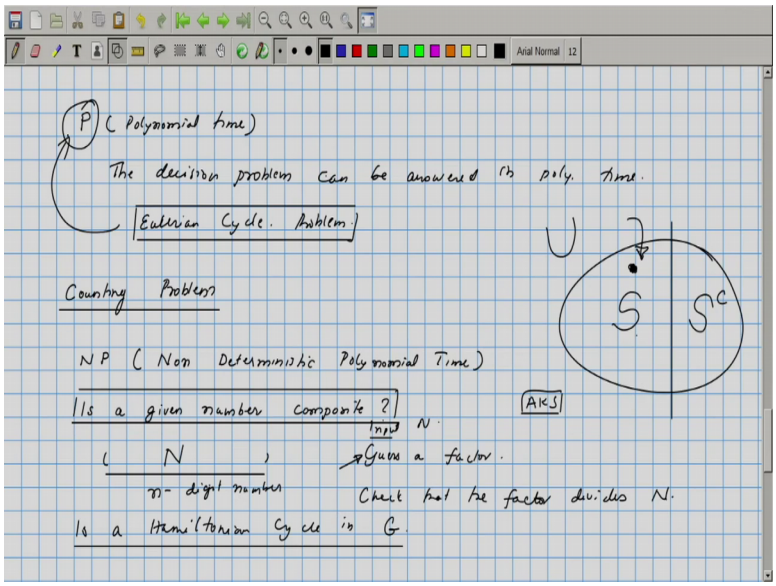
(Refer Slide Time: 05:36)



So, let us look at one class of computational problems called as decision problems. So, these are computational problems with; so, we will just uniformally write it as with 0 1 answers. So, examples would be is there a Hamiltonian cycle in G, another example would be is there an Eulerian cycle in G so, G is the input. The 3rd example would have been does the given system of linear constraints have a feasible solution with variables taking integral values

Another example of a decision problem would be is there a perfect matching in a given bipartite graph these are all problems known as decision problems. And, some of these decision problems is nice algorithms and some of them have are known to be a little more difficult to solve.

(Refer Slide Time: 07:53)



So, we could say that a certain decision problem belongs to the class P or polynomial time if we could answer the decision problem ok, so, in polynomial time. The decision problem can be answered in poly time. So, in particular if you look at say Eulerian cycle problem the input is an arbitrary graph and we want to check if it contains an Eulerian cycle there is an algorithm which will answer this question for every graph and the algorithms running time is bounded by a polynomial in the input size.

Therefore, we will say that this problem Eulerian cycle problem belongs to the class called as P or polynomial time. Now we could ask this with associated with each decision problem we could also construct a related counting problem how many Eulerian cycles are present in this particular graph that is a more complicated question. Now, note that if you could solve the counting problem we will always solve the decision problem, because the counting problem, if the answer to the counting problem is greater than or equal to 1 then we know that the decision problems answer is yes.

So, counting problems one should expect that the computational cost associated to the counting problem is more than that of the decision problem. Let us look at another complexity class called as NP its called as non deterministic polynomial time. So, look at all decision problems for which you can have a non deterministic polynomial time algorithm that will essentially be called as the class NP.

Let us take an example is a given number composite. So, the input is a let us say n digit number denoted by capital N, we want to know whether this is a composite number. Now the non deterministic polynomial time algorithm to do that would be guess a factor check that the factor divides N.

So, the input is N ok, if the factor divides N you will say its composite otherwise you will say not composite. Now the algorithm has the feature that for any composite number there is a correct guess that will cause the algorithm to say yes the number is composite and for every prime number there is no guess which would force the algorithm to say that the number is composite.

Of course for composite numbers there are wrong guesses that is true, but for every composite number there is at least one right guess and therefore, we will call that this problem or this decision problem composite versus non composite that is an NP that is of course, the polynomial time AKS algorithm which checks whether the number is composite or not and it does not involve any guessing.

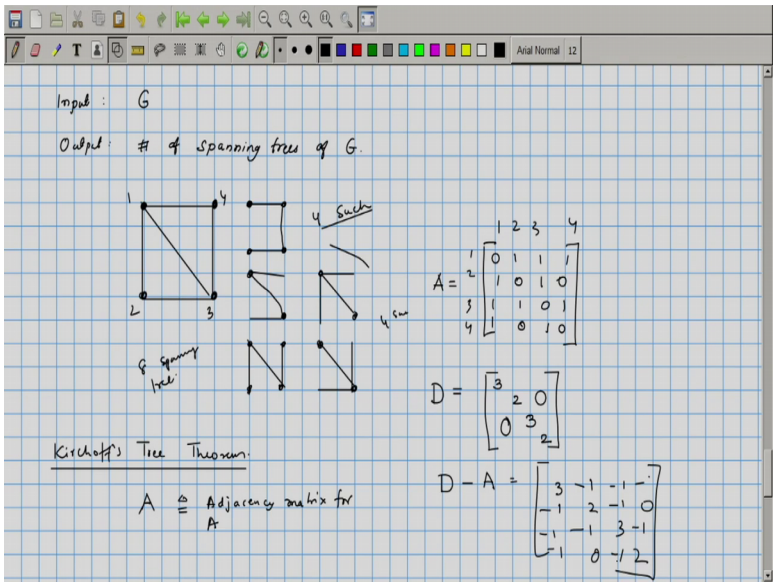
Similar question would be is there a Hamiltonian cycle in G. Again you can have a non deterministic polynomial time algorithm that is an algorithm which will make a guess and verify using that guess that the graph contains a Hamiltonian cycle and the entire computation gets done in polynomial time.

So, now why are we looking at P and NP when we look at the various counting problems that were faced with we could imagine that there is an underlying sample space a large sample space and say which we call U and this is split into 2 parts the set which has the required property and the complement set. So, for each element we could check whether the particular element I mean let us say take one element and check whether it belongs to S or it belongs to S complement you do this for every element in the universe you will get a count well that is not an efficient algorithm if U is very large.

But this question itself there is the sampled element or the arbitrarily picked element have the particular property or not that we can view as a decision problem that decision problem may have a polynomial time algorithm or in some cases it might have non deterministic polynomial time algorithm.

The class of problems that we are interested in for counting is those problems whose decision problems have a non deterministic polynomial time algorithm, they should have at least a non deterministic polynomial time algorithm naturally this class P is completely contained inside NP ok.

(Refer Slide Time: 14:06)



So, now let us look at some more concrete example let us look at this problem the input is a graph and the output we want to be the number of spanning trees of G. Clearly thus G have a spanning tree this question can be answered in polynomial time its connected we can say that it contains a spanning tree or if we just assume that the input graph is connected we know that (Refer Time: 14:36) it is going to have a spanning tree.

So, the decision problem associated with it is a very simple problem it can be resolved Trevelyan polynomial time, but what about the counting problem, can be estimate or can we find out the number of spanning trees in a given graph. For example, if we have taken this particular graph it is a simple graph. So, this is 4 vertices and 5 edges, how many spanning trees does this have? Well, if you keep the edge 1 3 in the spanning tree.

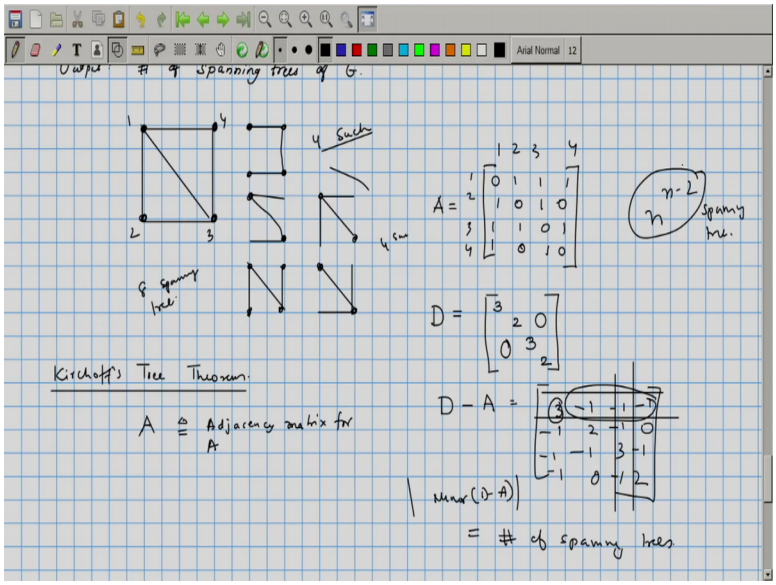
Then so, let us look at the number of spanning trees for this particular graph this could be one and there are 4 such and these are the ones without the diagonal with the diagonal is there that this is one, this is another ok.

So, there are 4 such as well. So, in total there are 8 spanning trees. So, here we had enumerated them and one can verify that there are no more. So, it looks like some kind of I mean you have to try many possibilities and check if it is a spanning tree and so on; there is a theorem called Kirchoff's Tree theorem which basically helps us solve this problem quickly. So, the theorem says suppose you look at the matrix A which is the adjacency matrix for the matrix for the graph ok.

So, A let us call it as the. So, in this case it would be the matrix with some 1 2 3 4 2 1 as an edge 2 0 is not an edge 2 3 is an edge this is not an edge 4 1 is an edge 4 3 is an edge others are not an edge this is the matrix A. And we can have a diagonal matrix which lists the degree of each vertex. So, one is the vertex degree 3 and the others are of 2 is of degree 2 3 is of degree 3 and 4 is of degree 2 and everything else 0. So, if you look at the matrix D minus A that is going to be the a following matrix with 3 2 3 2 on the diagonal they were 0 and the other entries are all negative ok.

So, this matrix D minus A in general is going to be and for any undirected graph, this is going to be a matrix whose determinant is 0, because if you add up any row for that matter any column the symmetric you will get 0, because for each edge there is a minus 1 and you have the degree on the diagonal.

(Refer Slide Time: 18:57)

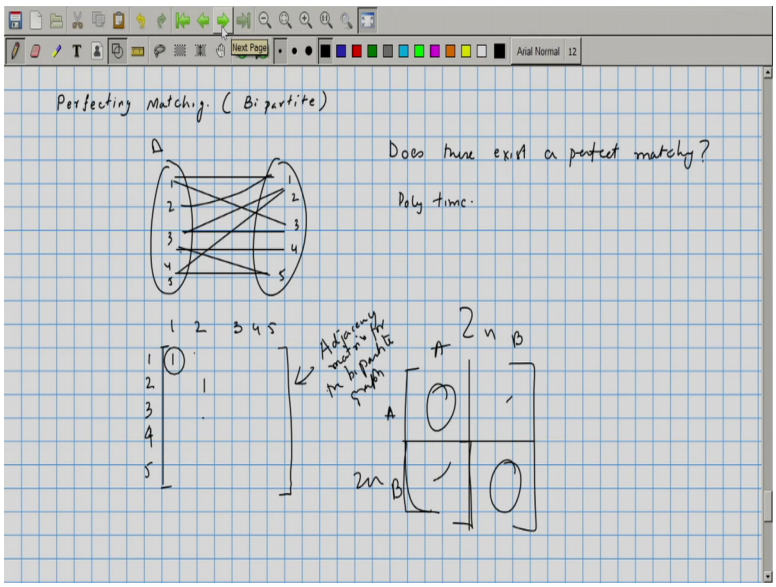


So, if you add up in your row you will get 0. So, the determinant of this is going to be 0, but you can just remove a row and a column and then try and evaluate the determinant

and that determinant would be called as a minor. So, look at a minor of D minus A and look at its determinant that will be equal to number of spanning trees ok. So, this is Kirchoff's tree theorem and because of the tree theorem we can compute all these quantities determinant computations is can also be done in polynomial time. And therefore, this problem of computing the number of spanning tree can be done in polynomial time.

So, this is an example where the decision problem is also called polynomial time and the associated counting problem is also a polynomial time even though the total number could have been exponentially large. If you take the complete graph it will have n raise to n minus 2 spanning trees. So, it is a large number, but still that computation could have been done in linear time. If we had let us say taken each spanning tree and check whether it is a spanning tree then, we will have to do large number of computations we do not have to do it because of Kirchoff's tree theorem.

(Refer Slide Time: 20:33)



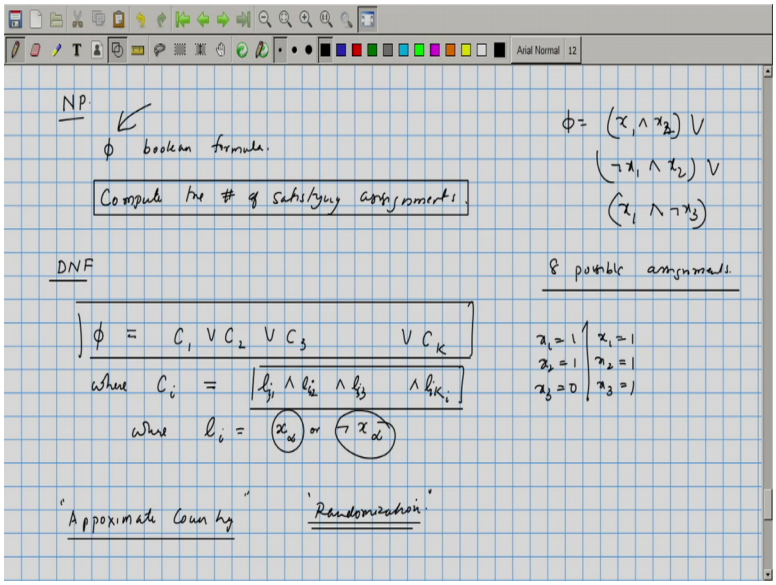
Now let us look at another problem called as perfect counting the number of perfect matchings ok. So, we will look at a bipartite graph and the number of vertices on either sides will be equal because otherwise there will not be a perfect matching, we want every vertex to be matched to some other vertex. So, let us say this is 1 2 3 4 5 1 2 3 4 5 we can call them as a 1 a 2 a 3 a 4 a 5 and b 1 b 2 b 3 b b 5. So, again the question of existence does there exist a perfect matching this can be solved in polynomial time because it can

be converted into a network flow problem it can be solved in many other ways. So, this question can be solved in polynomial time, but we want to know how many are there and that becomes a more complicated problem what is known as the following. So, let us say we can again construct a matrix corresponding to it ok.

So, we will have vertices 1 2 3 4 5 on the columns on the rows this is corresponding to the side A and for the other side we will have it on the columns 1 1 is an edge. So, you put a 1 2 1 is an edge, so, you put a 1 and so on ok. So, this is some kind of adjacency matrix for the bipartite graph you can think of it as a reduced adjacency matrix because ideally there would have been 2 n vertices ok, but because it is bipartite let us say this where A this is B; this is B this is A this is B 8 ways are no edges. So, this will be 0 B to B there will be no edges and by since we are looking at an undirected bipartite graph we can say that these parts will be symmetric.

So, we just looking at this part looked at 2 problems; one is the spanning tree and the other is bipartite matching. The count the number of spanning trees and the number of perfect matchings although both the decision problems a polynomial time we saw that one can be easily computed by the other we do not readily have an algorithm to do those computations in polynomial time.

(Refer Slide Time: 23:33)



We could also look at problems which are in NP for example, we could look at a general Boolean formula and we want to compute the number of satisfying assignments. If you

could do this for arbitrary Boolean formulas depending on whether that number is 0 or 1 we would have solved the decision problem for sat and sat being an NP complete problem you could have concluded that P equal if there was an algorithm to count the number of satisfying assignments for an arbitrary formula we could have easily solved P versus NP problem.

So, this one should expect that this is going to be difficult, but what about suitable restrictions if we restrict these Boolean formulas to certain kind of formulas. So, the kind we are interested in is what is called as disjunctive normal form ok. So, let us describe what is disjunctive normal form? So, our formula phi is of the form C 1 or C 2 or C 3 or C k, where each C i is of the form l 1 and l 2 and l 3 and l k where each a l i let us call this C l i 1 one l i 2 2 l i 3 or l i 1 l i 2 and l i k i ok.

Were each l i known as the literal is either let us say x alpha or not x alpha its an either variable or the negation of the variable. And we may assume that each clause contains at most one occurrence of a variable, because if two occurrence of same variable appears if they are in the same format we can just replace it by a single one or if they are of opposite kinds we can say that that particular clause is unsatisfiable and therefore, the formula can be simplified further.

So, we are given a formula of this kind and we are asked the question how many satisfying assignments does this particular formula have? For example, if were taking phi is equal to x 1 or x 2 sorry x 1 and x 2 or not x 1 and x 2 or x 1 and not x 3 ok. So, there are 8 possible assignments it is clear that by taking just the first clause itself we can find an assignment which satisfies them satisfies all of them ok.

For example x 1 equals 1 x 2 equals 1 and x 3 equals 0 would satisfy the first clause and therefore, it would try to say the entire formula, but there are other things as well x 1 equals 1 x 2 equals 1 x 3 equals 1 is another one. So, we can just look at all possible 8 assignments and describe the total number, but that is an exponential time algorithm.

So, these are the typical kind of problems that we are going to look at when we study counting; when we talk about approximate counting that arises because the exact counting may be difficult in many cases. In cases spanning tree we could get the exact count by a polynomial time algorithm because of Kirchoffs Tree theorem. In case of perfect matching we can think of the problem as computing the permanent of a matrix,

but computing permanent is known to be difficult. So, we cannot hope to find the exact values. In case of satisfying assignments for a Boolean formula again if we could compute it exactly then we would have solved the P versus NP problem.

So, therefore, we will be less ambitious and say that instead of computing the exact value can we get the approximate answer that is the general theme of approximate counting. And we will in addition allow randomization, in the sense our approximate counting needs to be correct only with a certain probability ok; we will mix two themes the formal definitions for that we will see in the next lecture.