Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

Institute: IISc

Week: 01

Lecture 1 Introduction and Principles of Algorithms

Namaskara, warm welcome to the course on graph algorithms. The computers are tools for solving problems. Therefore the main goal of a computer professional is learning and mastering the art of solving real world problems by using computers. The main goal for any computer professional is real world problem solving by using a computer. The real world problems involve physical objects. For example you want to go from home to airport.

What is the best way to reach airport from home? Now this involves you, the car, the roads and so on. Therefore you have a typical real world problem involves physical objects. But what are computers are capable of? They are capable of handling only binary digits - zeros and ones or mathematical object. So if you look into the computers they can handle only mathematical objects.

Real world problems involve dealing with, interacting with and working with physical objects whereas computers are capable of only handling abstract mathematical object. We want to solve the real world problem by using a computer, so how do we bridge the gap? The first thing that we do is formulate the real world problem into a mathematical problem. We choose an appropriate mathematical model and recast the real world problem into a mathematical problem in that mathematical model. What is a mathematical model? Mathematicians call them as algebraic systems, mathematical problem in mathematical model. By mathematical model we mean an algebraic system.

An algebraic system consists of a set of mathematical object or abstract objects or elements, together with a set of operations. You not only define the mathematical objects, we define certain operations on them. For example, you can consider integers and on integers we do addition, subtraction, multiplications and so on. For example if you take integers you do plus, minus, star, mod, div - several operations we do. If you consider sets you do union, intersection and so on.

We may also do on sets - insert an element, delete an element, search for an element in a set and so on. So set is a mathematical object and we are defining a collection of operations on them and it defines an algebraic system. In computer science we call this as abstract data type, in CS we refer this as abstract data type. So abstract data type is a mathematical model together with some set of operations. It is a mathematical model together with a set of operation.

So algebraic system in mathematics is treated as abstract data type by computer scientists, however in computer science, we will deal with concrete mathematical objects their representations and everything is explicitly represented and manipulated. Operations are defined so that they can be carried out by the machine and so on okay. What do we do next? You have a real world problem. You have converted that into a mathematical problem. Now you want to use computers to solve the problem.

You want to find a solution for the problem by using a computer. So what next? Let us begin at the beginning. We are going to take a look at principles of algorithms okay, in fact we are going to title this as principles of algorithms. By principle we mean a basic set of concepts and ideas that form the basic building blocks for a body of knowledge. You might have heard about the terms like principles of mathematics, principles of physics and so on.

By principles, we mean a basic set of ideas, definitions and so on. We start with those concepts and then the entire body of knowledge is built based on them, okay. So what are those basic things that would form the principles for algorithms? The first notion is the notion of problem itself, okay. This is basically a mathematical problem post in a mathematical model. You can call it as algebraic system or abstract data type.

So, there is a framework in which a mathematical problem is posed, it is formulated and then, in mathematics the problems are formulated and stated in variety of ways. In math, we may even ask questions like does there exist a solution, is the solution unique, is it bounded, what are the limits for the values upper bound, lower bound and so on. In mathematics, by the term problem we mean a broad range of questions. But for computer science we are interested in computational problems. The first principle is about problems and computational problems.

What is a computational problem? Every computational problem is required to work on certain values from the mathematical model and they are called inputs. So input is set of values that the problem is required to work with and every computational problem is a characterized by a process. We are interested in finding the solution of the values that we have taken up for working and it involves a process, a sequence of operations. A

sequence of operation performed on the values that you have taken as input. This sequence must be finite, it should be unambiguous, in principle we should be able to perform the sequence of operations on pen and paper and so on.

And then every computational problem is required to produce certain values at the end of processing and that is called the output. It is a set of values produced at the end of the process, okay. So every computational problem has got a specification and a solution. By specification, we mean the relation between input and output values. You have to specify what kind of values you should obtain for the given set of values.

Therefore, specification is nothing but a relation between input and output values. The solution is nothing but explicit description of the process. What are all the operations you have to do on the values that you start with and how you would obtain the output, that has to be specified and that is solution, okay, and in computer science we call this as algorithm. So algorithms consists of an explicit description of a sequence of operations to be performed on the input values. The execution of the algorithm produces output values.

An algorithm is said to be correct if it meets the specification, in the specification, we have stated for what input, what output should be obtained. The execution of the algorithm should precisely produce those outputs. The algorithm when you execute on a specific input, it should produce the corresponding output. Algorithm and its execution on an input produces the corresponding output. The relation is already specified.

So an algorithm for a problem, should take as input that is specified for the problem and it should produce the output that is specified in the relation and this is the first concept that we would start with. Every problem, real world problem that you want to solve by using a computer must be formulated as a computational problem. Only after doing this you can use a computer to solve that, okay. So if there are ambiguous questions and then things that cannot be resolved through a set of operations, all those kind of things cannot be handled. Therefore we do not consider such mathematical problems.

Computers are capable solving only computational problems. So the real world problem must be first converted as a computational problem. So we view as whatever is the real world problem is, we view that as a process to be done on certain input values and it should produce the output. We have to map the real world problem to such a setup, okay. This is the first thing to be done and then since we are using computers it is important that we carefully look at the use of physical resources.

Okay, that is because execution of a program on a computer is going to take lot of memory locations. It may take lot of time to produce the answer and so on. Therefore

there are certain physical resources used when you try to solve the problem by using a computer, okay. The notion of complexity is the next important principle. In computer science, complexity refers to the utilization of resources.

How much of the resource is used? This is what we mean by complexity, how much time we use this is referred as time complexity. How many memory locations are used by the computer for solving the problem that is space complexity? It is a resource utilization, okay. And we may also have a set of computers or trying to solve the problem each doing some work and passing on the intermediate results to the other computers and so on. We have distributed systems which might work by passing on the messages and intermediate values and finally the problem might be solved.

In such situation the key resource there is the communication complexity, okay. Computation can be done very fast. But the communication over the network may take enormous amount of time, therefore you would focus on how much communication is done for solving the problem. So such things like communication complexity, various kinds of problems and problem solving may treat certain resource as a key resource and we have to estimate how much of that resource is used for solving the problem, okay. This is very important because when you are trying to work out a process or a solution or an algorithm, you are going to implement that idea on a computer, as a computer program.

If it takes enormous amount of time, we say that it is an inefficient implementation or an inefficient program. So if there are several ideas and several ways to solve a problem, we are interested in picking up that solution which is using minimum amount of resource. The time complexity for example, if time is a resource you are focusing on, time complexity it should be as small as possible. So your notion of efficiency of an algorithm is related to the complexity of the process, how much resource it consumes. So you focus on a resource and you define your efficiency in terms of the resource consumption, how much time it takes.

For most part of our discussions, we will focus on time complexity. Okay, occasionally we may discuss about the space complexity. There are other situations and algorithms, problems and solutions where other complexity measures are to be used but we are not going to use them. We focus primarily on time complexity, okay. How much time it will take to produce an output? If you give this input, how much time it takes to produce an output? We would like to have an idea on that.

In fact the quality of an algorithm or quality of a solution is measured in terms of how efficient it is when implemented, okay. So the next question is how do we measure the

complexity? If you want to compare two algorithms for their efficiency and if you have two or three options available and if you would like to choose that method which is using the least amount of resource here least amount of time, how do we determine that? How do you compare that? So we need a measure, okay. That means we require a complexity measure. And that leads us to take a look at other important principles. Though, so, the first principle is the notion of problem and computational problem, then we have seen the notion of complexity and now we are going to look at complexity measure.

How do we measure the complexity of a process, a computational process, okay? So we need the following important notion or the concept, this is our next principle, this is called input size. Input size is an important concept and this is addressing what is known as problem of scale. Typically computers are used to solve problems that would work on a large amount of data, okay. Take for example our railway reservation system, when the number of passengers were much less, we were using physical ledgers and notebooks and so on to record reservation details. However, when the number of passengers and stations have become thousands and passengers have become millions.

We just cannot handle that anymore with ledgers and notebooks and so on. So you typically go for a computer based solution when the amount of data to be processed in the problem solving is very large, right. So input size is mathematically modeling the concept of problem of scale. A large amount of data. So how do you measure the amount of data that is available? The concept of input size does it.

So an input size is typically expressed through a small number of parameters and these parameters are variables, these variables will assume values that would indicate the amount of data that you process. Let us take a simple example of sorting. You want to sort an array of numbers, you want to sort in a list, and it does not matter, suppose you want to sort 1000 numbers. Sorting as a problem can be specified as arranging the numbers in increasing order. How many numbers you take it up? If you set n as a parameter which stands for number of items to be sorted.

n is the number of items to be sorted. So as n becomes larger and larger, you can see that you are sorting larger amount of data. You may wish to sort 100 numbers, you may wish to sort 3 numbers, you may wish to sort million numbers. So n varies from 3 to 100 to million to any value. Sorting a set of numbers, how big is that set of numbers? How big is that set? That is parameterized through a single variable n and here n acts as a parameter for input size or as a measure of the input you are processing. Sometimes you may require more parameters to describe the input size, the amount of data you are processing, okay, consider for example matrix multiplication.

So you want to multiply two matrices. A is a matrix and it is dimensions are p cross q ie, $A_{p\times q}$ and B is another matrix its dimensions are $q$and r, ie $B_{q\times r}$. Only when row number of one matching the column number of the other you can define the matrix multiplication. So you want to compute $C = AB$, C is a $p\ by\ r$ matrix. It is a computational problem. Now depending on the values of p, q and r; the amount of data you are processing, the amount of numbers you work with to produce the answer varies.

A very large matrix will have bigger dimensions, but there are three parameters here. Multiply two big matrices, how big is those matrices that is specified by specifying the values of p, q and r therefore the parameters p, q and r are used to model the concept of input size. Variations in p, q and r would model the scale at which we are working on, that is problem of the scale is addressed by the values of these three variables for matrix multiplication problem. If it is a square matrix multiplication, A is a $n \times n$, matrix B is also a $n \times n$ matrix, for square matrix multiplication it is all fixed and there is one parameter is enough and n can act as input size. This is just a parameter, a variable, whose variation would imply the problem of scale, okay.

There are certain subtle part of this definition of an input, we may not need them for dealing with algorithms related to graphs, okay. By the way, we are going to focus on a mathematical model called graphs. Graphs are very interesting and very useful discrete mathematical structure like sets, integers, points, lines and so on. Graph as a mathematical object is an extremely important and interesting discrete structure and the focus of this course is on graph algorithms, computations done on graphs. okay, we will see the formal definition of graph later, assuming that you have some basic familiarity in discrete math and graph you can see that a typical graph is represented by a vertex set and edge set, $G = (V, E)$ therefore if cardinality of the vertex set is , $|V| = n$; cardinality of the edge $|E| = m$ - by cardinality means number of items - m then n and m they act as an input size.

If you are working with a graph with 100 nodes and 758 edges, n equal to 100 and m is 758. There are two parameters we use to describe the input size when we are working with graph and graph algorithms, okay. This addresses the problem of scale, now look at the process, input and next concept is the process. We are interested in the notion of time complexity. How much time a computer will take if you execute this algorithm, algorithm is basically an idea, a sequence of instructions, it works and it takes an input, if you follow that algorithm you would produce an output.

This is the theoretical setup, but when you implement that algorithm how much time it is going to take, physical time in seconds, minutes, hours and so on, how much time it would take; you would like to get an idea about that. That is because if I want to compare

two algorithms for their efficiency, I would like to choose that one which has got, which is taking less amount of time. In the early days, just 60 to 70 years back, the efficiency of an algorithm is determined by physically implementing that algorithm and run that code on few inputs and then check how much physical time it takes, okay. And this way the comparisons of the algorithms were done. But you can see that this is not at all a correct approach.

The reason is when you compare an algorithm A with algorithm B you might have taken an input that favors A than B, right. That algorithm may be a bad input for B but it could be a good input for A. Just running both A and B on that input you cannot conclude that A is better than B. But when you execute, you have to take some sample or random input and then see the performance of A and B that is the only option. So you can have a table of running times and then you may come to some conclusion but that may not be a correct conclusion.

That is the reason why, with respect to the notion of efficiency of an algorithm and comparing algorithm, there was a huge confusion among the computer science community, they did not have quite clear idea as to how can we categorically say algorithm A is better than algorithm B, okay. So computer scientists like Hartmanis, Donald Knuth and several others have pioneered the notion of time complexity. Now this notion is independent of the compiler, operating system, and the machine on which you work, the hardware, the instruction, the basic processor; it is independent of that, it is a characteristic of an algorithm. Basically you have to compare two algorithm means you have to examine the instruction sequence and then arrive at the conclusion about the efficiency of this method. So, remove all those construct and then look at only the algorithm and then conclude about its efficiency, this is what was proposed.

Also, if you look into the running time of an execution of an algorithm, it is made up of executing the sequence of operations. Each operation will take certain amount of time and the total time taken is basically the total time in executing the sequence of operations. That is the reason why the first principle here for time complexity is instruction count or operation count. Number of operations you perform on the data, how many operations you perform to produce the output, okay. Use this to get an idea on the running time, this instruction count is an integer, just a number, okay, and this instruction count is called the time complexity.

So let us understand the difference between time complexity and running time. Running time is measured in seconds, minutes, hours and so on whereas the time complexity is an integer, it stands for an instruction count, how many operations are done. So you can examine the text of the algorithm and then you can arrive at this count. You do not have

to go to the machine, you do not have to write a program, implement, nothing, right, just look at the specification of the algorithm and from that you can figure out the instruction count. This is simply an integer but this integer will give you a detailed idea on the running time, running time is physical in seconds, minutes and so on.

Instruction count is on the other hand an attribute of an algorithm which is figured out by looking into the specification or the text of the algorithm. Look into that and then you can get this. That way even before implementing an algorithm, I will get an idea about how good this algorithm will be if implemented. So when you have four or five algorithms, four or five ideas to solve a problem, all I have to do is that for each algorithm figure out the instruction count and then determine which one is the most efficient one, choose only that for implementation. You do not have to write code for all of them, run all of them on all kinds of inputs to find out which one is better.

That is not the right way. The right way to comment on the efficiency of an algorithm is identifying the instruction count. So instruction count is used as a figure of merit of an algorithm. Instruction count is used to comment on the running time that the algorithm might take when implemented as a program, okay. It is helpful in commenting on that. This instruction count is used to compare two different algorithms to determine which one is better.

So I stop at this point and continue the discussions in the next session, thank you.