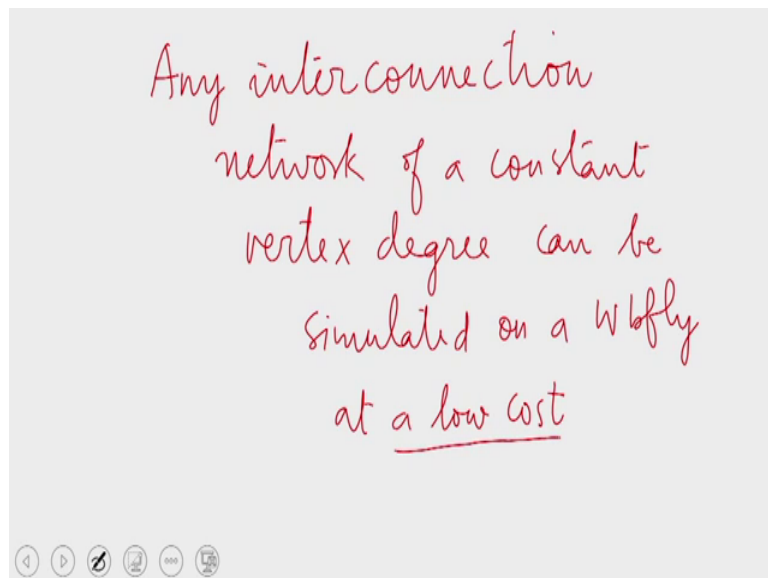


Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 32
Butterfly, CCC and Benes Networks

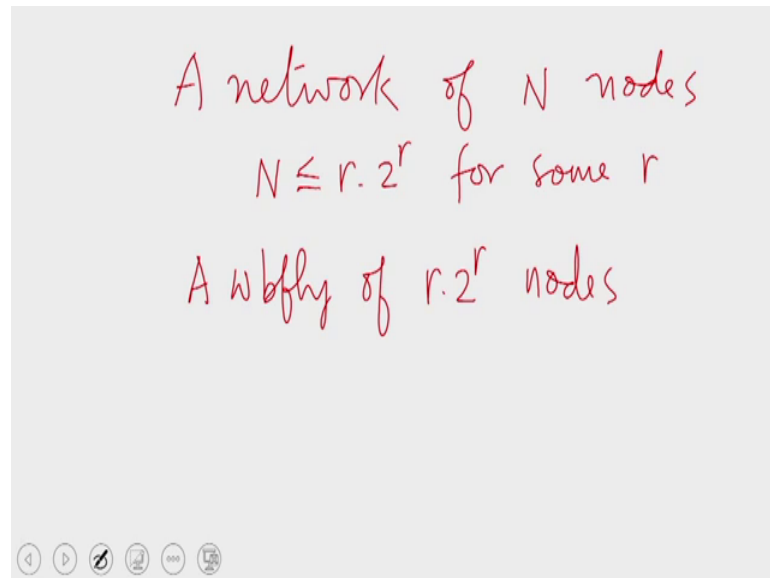
Welcome to the thirty second lecture of the MOOC on Parallel Algorithms. Today, we shall see some more results on a wrap butterfly and Cube Connected Cycles. A wrap butterfly and in consequence a butterfly and will as well as a cube connected cycle are quite versatile networks.

(Refer Slide Time: 00:52)



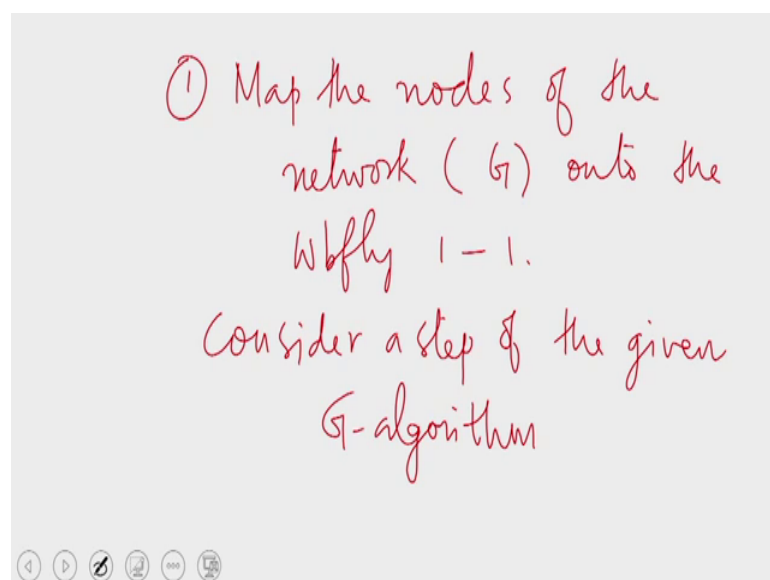
The result that we are going to show is that any general network, interconnection network of a constant vertex degree can be simulated on a wrap butterfly at a low cost. We shall see what the low cost is. So, we assume that we are given a general interconnection network.

(Refer Slide Time: 01:10)



Let us say we have a network of N nodes where N is r times 2 power r for some r or N is less than r equal to that. Let us take a wrap butterfly of r times 2 power r nodes and we want to simulate the algorithm that is given for the network on the wrap butterfly. So, we have a generic network and we have some algorithm given on this generic network. We want to simulate this algorithm on a wrap butterfly; that is the problem at hand.

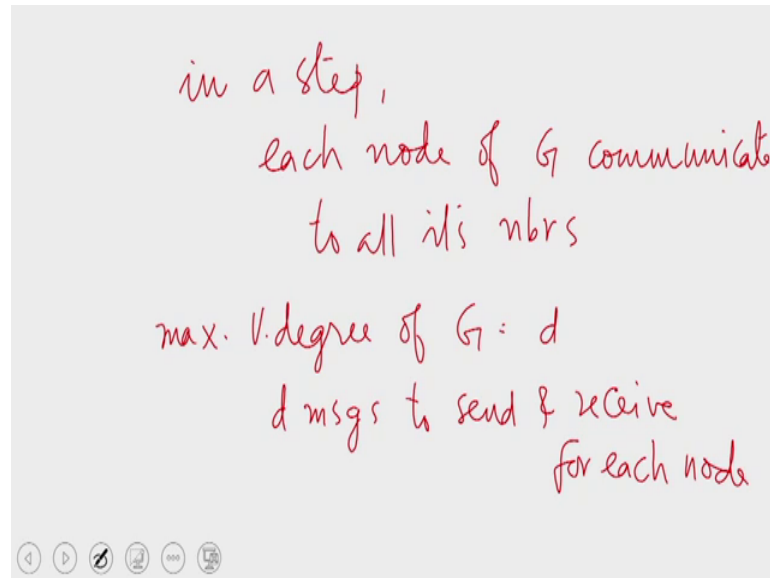
(Refer Slide Time: 01:51)



So, the first thing that we do is to map the nodes of the network. Let me call the network G onto the wrap butterfly 1-1. This 1-1 mapping could be anything. So, what we do is to

map the vertices of the given network on to the vertices of the wrap butterfly anyhow; the only requirement is that it has to be a 1-1 mapping. And then consider a step of the given algorithm. This given algorithm runs on G . So, let me call it the G -algorithm.

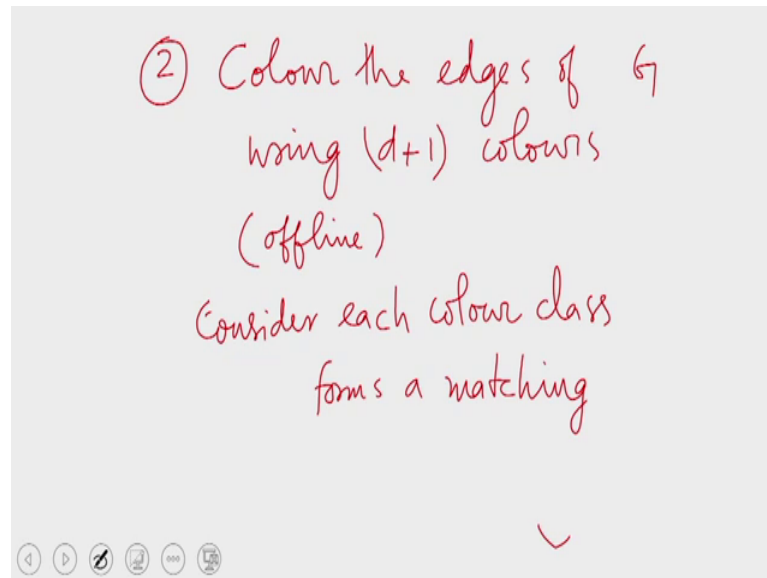
(Refer Slide Time: 02:25)



So, we consider a simple step of the G -algorithm as is customary in all interconnection networks what we assume is that in a step each node of G communicates to all its neighbors. A node has a constant number of neighbors; let us say the vertex degree of G is d , the maximum vertex degree in G is d which means a node can have at most d neighbors. So, in every step of the algorithm, a node may have to communicate with all its d neighbors. So, a node has d messages to send and d messages to receive for each node. So, that is the constraint we have. These many messages have to be transferred.

Once we have a way of transferring all these messages, then we can say that the step is similar to neighbors because we have mapped the nodes 1-1. We have one processor for every node of G . So, the internal computations can be done by the assigned processor, but once the messages are handled, then we would have achieved the simulation.

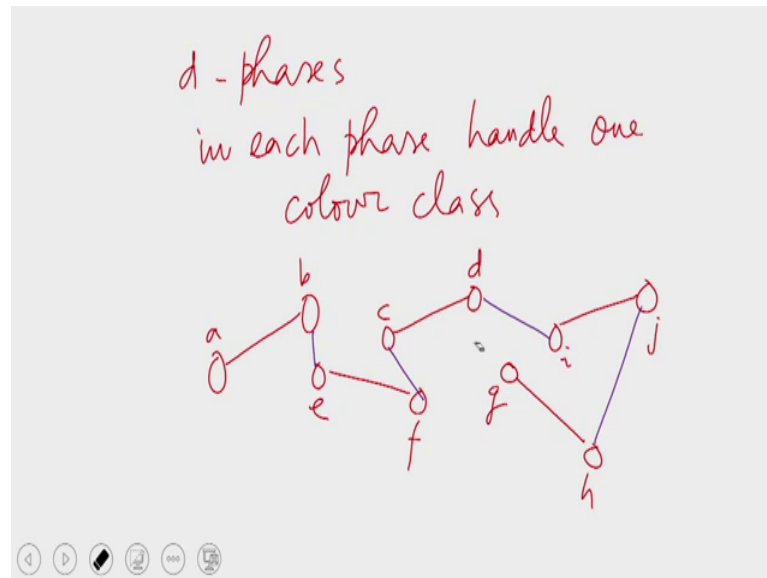
(Refer Slide Time: 03:22)



So, the second step was to colour the edges of G using d plus 1 colours. The given graph G , it is a generic graph. So, we do not know the minimum number of colours required to edge colour the graph. But in any case for a d degree graph, it is possible to edge colour the graph with d plus 1 colours. So, that is what we are going to do, we will colour the edges of G using d plus 1 colours.

Now, this colouring is done one once and only once. So, this is an offline operation. Therefore, we could do this sequentially. So, we are not going to count this in the cost of our parallel algorithm we somehow edge colour this network g using d plus one colours and then consider edge colour class separately. A colour class is the set of all edges belonging to the same colour. So, in particular if we consider all edges of colour one, this form; they form the colour class one and we know that they form a matching; no two of them can be adjacent. So, each colour class forms a matching.

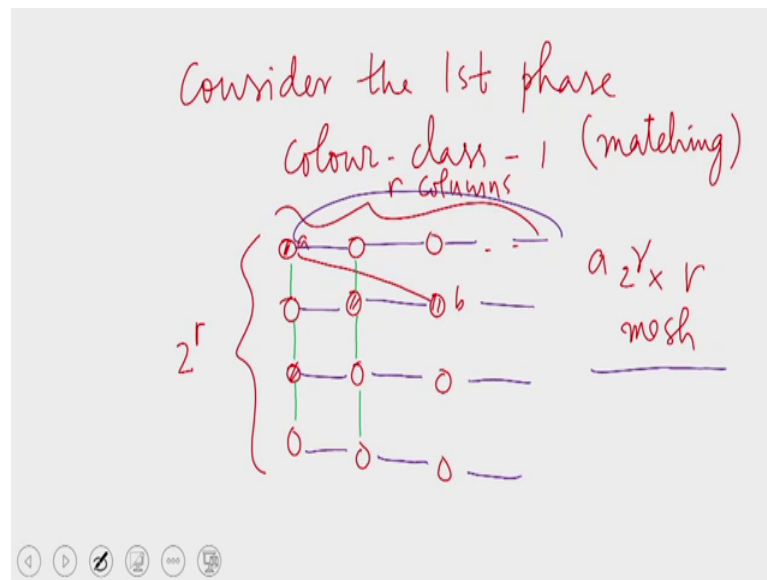
(Refer Slide Time: 04:33)



So, our idea is to simulate the message passing step using d-phases. In each phase, we handle one colour class; in each phase, we handle one colour class. So, there are d-phases. So, when I consider one particular colour class, we find a matching. So, the edges of the matching could be like this. So, if the vertices are labeled in this fashion, what it means is that; a has a message to send to b and possibly b has a message to send to a. So, the messages between a and b will be exchanged in this phase. At the same time the messages between c and d will be exchanged in this phase and so on. So, all these edges are of the same colour. Therefore, the messages that should pass through these edges will be handled in the first phase. This let us assume as colour class one.

When we consider another colour class of course, no two of them could be adjacent. So, blue is another colour class here. This possibly could be the second colour class. So, these edges could be considered in the second phase. So, in the second phase we will exchange the messages that are between b and e, c and f, d and i and h and j. So, in this manner, we will consider all the colour classes. Once all the colour classes are considered, we would have consider all the edges. So, all the messages that are just trying to be delivered by every single edge would be delivered at the end of the simulation. So, we have d-phases to the simulation.

(Refer Slide Time: 05:59)



So, now let us consider any single phase; in particular consider the 1st phase in which we consider colour-class-1. What matters is that we have a matching; no two edges of the same colour are adjacent. So, this forms a matching. So, let us consider the edges of colour-class1, the messages on these edges have to be delivered, but how would we deliver them. In our example, we find that a and b belong to colour class 1. So, do e f, c d, g h and i j. These edges all belong to the same colour class; let us say colour class 1. So, these edges are active at the moment, but a and b are adjacent in graph g, but that does not necessarily mean that the vertices that are mapped to a and b in the butterfly network or the wrap butterfly network are adjacent to each other. Because what we did initially was to map the vertices of G anyhow to the vertices of the wrap butterfly; a wrap butterfly with at least as many vertices.

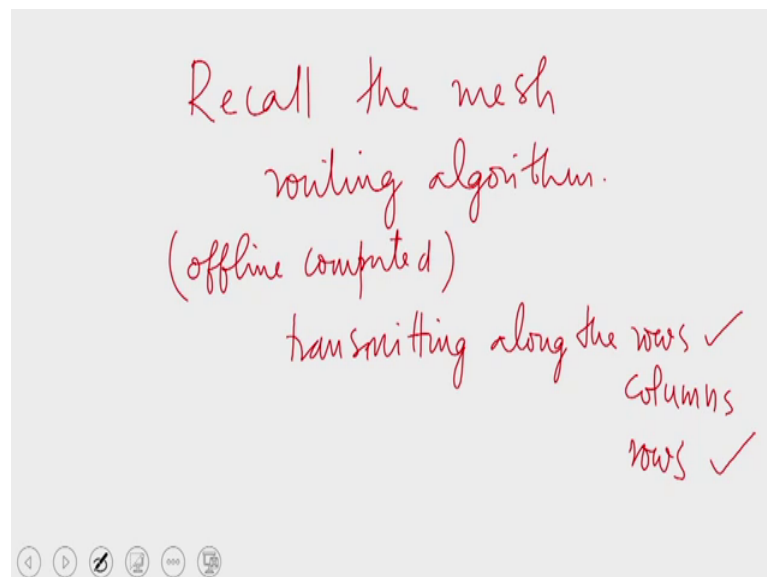
So, it is not necessary that a and b are adjacent in wrap butterfly, but we still have to send a message from a to b. How do we do this? So, now, if you consider a wrap butterfly; if I do not draw the edges, what we find is that there are 2^r rows and r columns. Now the situation that we have is this, we have a message sitting at each vertex. If this is a and this happens to be b, their excess an edge between these two in graph g, but this edge is not present in the butterfly network.

Now, we have a message from a that should be going to message b. So, we have essentially a routing problem on the butterfly network on the wrap butterfly off dammed

of size 2^r by r , we have a routing problem. Every node in the wrap butterfly network holds a message in this every message has a unique destination. So, the question is how do you deliver the messages. Looking at the nodes of the wrap butterfly gives us an idea. When we look at the nodes the wrap butterfly we find that they are arranged much the same way as the nodes of a 2^r by r mesh would have the nodes arranged exactly the same way only that the edge sets would be different.

But then there is some similarity in the edge sets too because the horizontal edges in a wrap butterfly are exactly the way, they are in a mesh. Except in that there is a feedback, the last node is connected to the first node except in that a wrap butterfly is very similar to a mesh. But then the cross edges here are quite different from the vertical edges of the mesh. In a mesh, we have edges which are vertical. For example, we have edges of this sort in the mesh, these edges are missing a wrap butterfly. So, we have an arrangement of the processes that is very similar to that of a mesh, but the vertical connections are missing instead of the vertical connections what we have are the cross connections. Now the question is using these cross connections can we actually managed to transmit the messages using the messages vertically.

(Refer Slide Time: 09: 28)

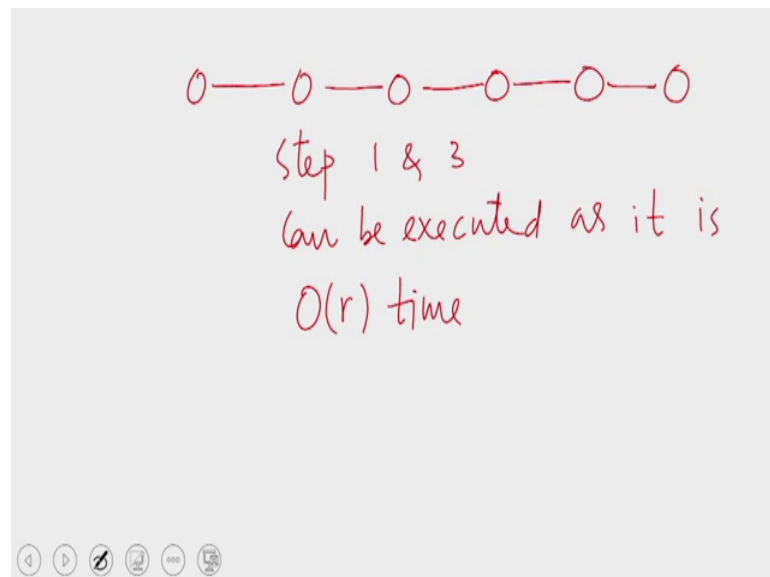


Let us consider recall the algorithm for transmitting messages on the mesh. This mesh routing algorithm had some offline computation which was not to be executed in parallel necessarily. So, the offline computation would be executed sequentially too. So, we are

not going to delve on that. The online computation involved essentially transmitting messages along the rows followed by transmitting messages along the columns and then again along the rows. So, the mesh routing algorithm had three phases. In the first phase we transmitted a message along the rows, in the second phase we transmitted the messages along the columns and in the third phase we again with transmitted the message along the rows.

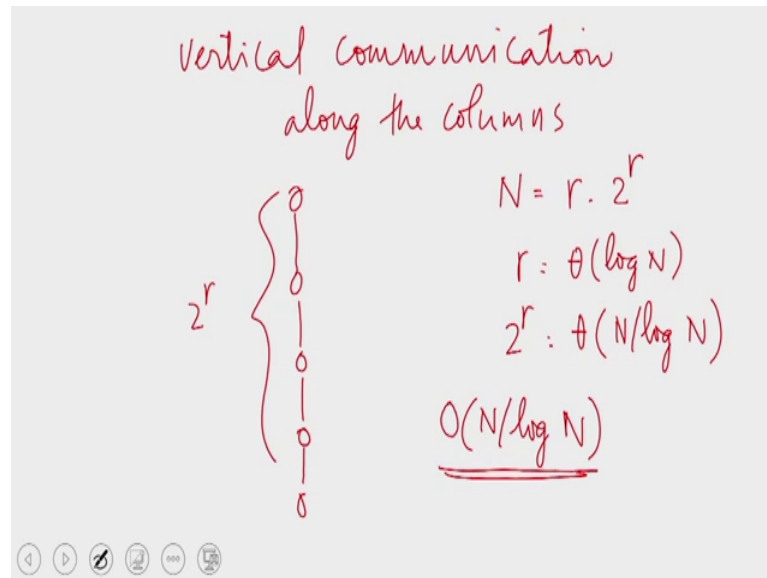
Now, we find that here transmitting messages along the rows is easy enough because every row has connection using straight edges.

(Refer Slide Time: 10:21)



We have a connection of the sort exactly as in the case of a mesh. So, steps 1 and 3 can be executed exactly as it is these executions will take order of r time.

(Refer Slide Time: 10:37)

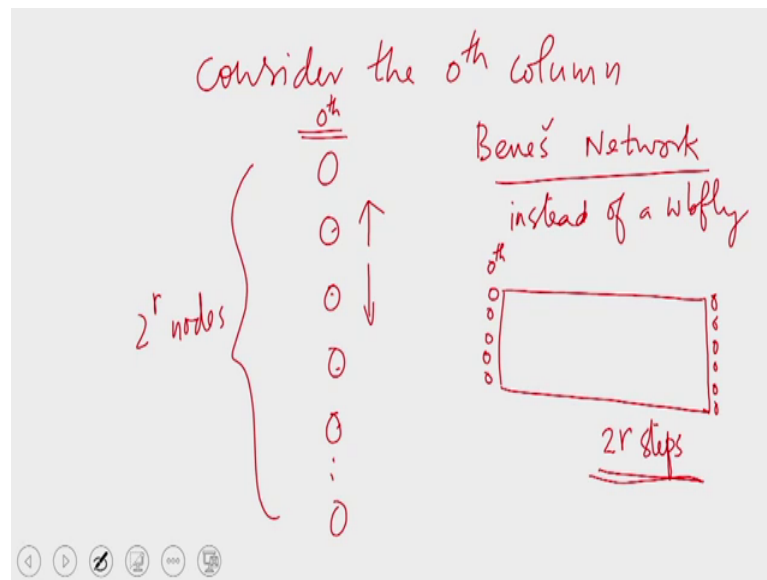


What remains is the vertical communication along the columns. If we had vertical connections of the sort that is had our network been, but 2^r by r mesh instead of a wrap butterfly as it is now, then it would have taken 2^r steps to do the vertical communications. But then if N equals r times 2^r ; then r is $\theta(\log N)$ and 2^r is $\theta(N/\log N)$.

Therefore the time taken would have been order of $N/\log N$ we would have had only an order of $N/\log N$ time simulation of every single step which is not good enough. That is why we are not attempting the simulation using a 2^r by r mesh. Instead we are considering a 2^r by r wrap butterfly which is an r dimensional wrap butterfly.

So, now the question is this, how do we achieve the vertical communication along the columns. In particular let us consider the 0th column.

(Refer Slide Time: 11:40)

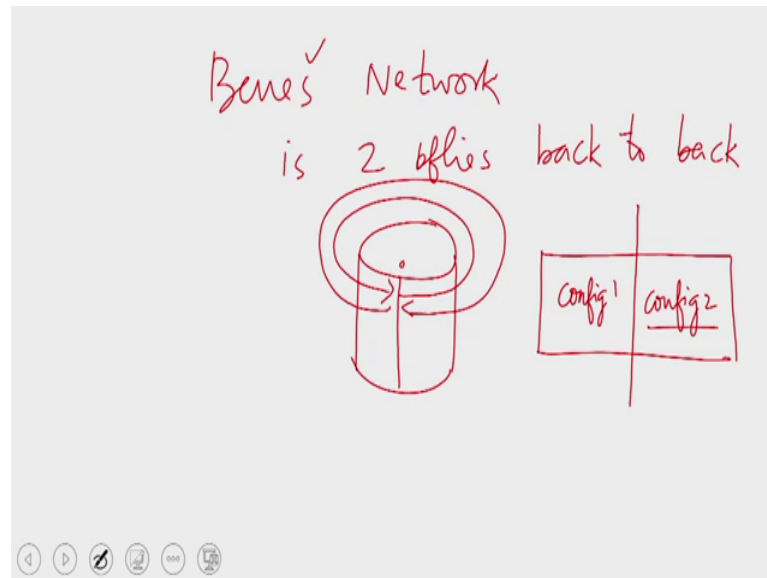


So, in the 0^{th} column we have 2^r nodes and each of these nodes hold some messages and these messages have to be communicated vertically. How would we do this? If we had our a Benes network instead of a wrap butterfly network. As we saw in the last class given any permutation π of the input values Benes network will allow us to communicate; to realize this permutation using edge disjoint paths. So, that is the technique we would use here. So, if we had a Benes network instead of a wrap butterfly, we would be able to place exactly these packets here. The packets belonging to the 0^{th} row as inputs to the Benes network and then with an appropriate configuration, we would be able to deliver them here after $2r$ steps.

So, given these permutations, we would be able to compute the necessary configurations of all the switches here. So, if you assume that every Benes network switch is a reconfigurable switch, then we would be able to compute the reconfigurable the reconfiguration necessary for the switches so that the transmission could be achieved in $2r$ steps. This is what we saw in the last class. So, for any fixed by there is one reconfiguration and this reconfiguration could be computed and implemented. Once it is implemented, the delivery of the inputs would be achieved in $2r$ steps.

So, now we have 2^r packages back packets that are to be delivered to the same nodes, but then we do not have a Benes network all that we have a wrap butterfly.

(Refer Slide Time: 13:31)

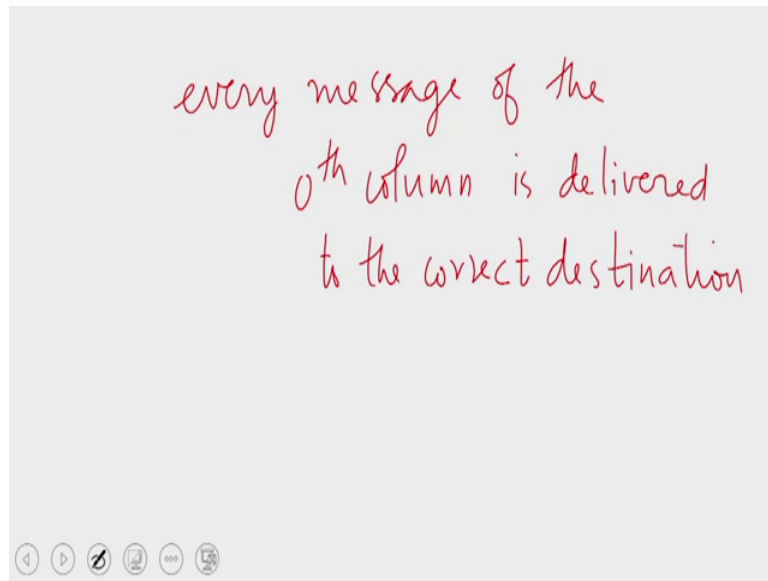


But then a Benes network is nothing, but 2 butterflies pasted back to back. So, in a wrap butterfly if I start with the 0th column; suppose this is the 0th column. If I send the messages forward until they come to the 0th column; again which also happens to be the last column and then allow the messages to be sent back in the reverse order. Until they come back to the 0th column, we would have achieved the effect of a Benes network. All that we have to do is to reconfigure the switches accordingly.

So, we take the Benes network the imagined Benes network that is necessary for delivering the messages with the permutation realized. Divide the Benes network into two see that there are different configurations. So, this is config 1 and this is config 2. Configure the switches according to config 1 and then send the messages forward until they complete one full revolution and then change all the configurations to the second what second set and then send the messages back. We would have achieved the same result as we would have sent the messages through a Benes network.

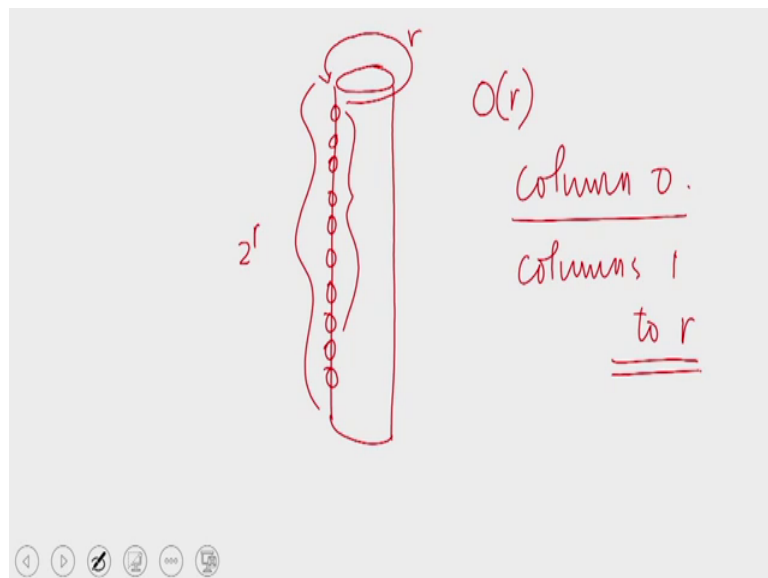
So, we are essentially simulating a Benes network using a wrap butterfly network. In a wrap butterfly, every message going clockwise once and then counterclockwise once will have the effect of passing them through a Benes network.

(Refer Slide Time: 15:03)



So, using the result of the previous class, we will be able to make sure that every message of the 0th column is delivered to the correct destination.

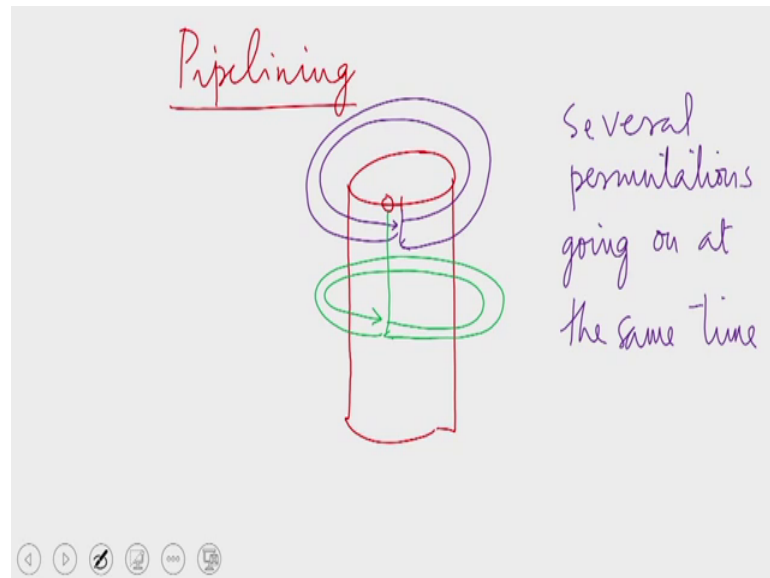
(Refer Slide Time: 15:18)



Now, you should remember that a wrap butterfly is in fact, a rather tall cylinder. It has a height of 2^r whereas, this is only r . Therefore, the time taken for two revolutions is order of r . In order of r steps we managed to send all messages which are in column 1 to their correct destination. So, we managed to permute this column in order of r steps.

So, there are 2^r nodes here. They get permuted, the way we want in order of r steps. So, in a mesh what would have taken 2^r steps is achieved in order of r steps here, but then this will ensure the permutation only of column 0. But then what about the remaining columns? Columns 1 to r also have to be permuted the same way. How would we handle them?

(Refer Slide Time: 16:29)



To handle these columns, we use pipelining exactly analogous to column 0 column 1 also can be permuted this way. You will start from column 1 and do two revolutions; let us say this is column 1 and this is column 0. I will use two different colours to explain what is happening to column 1. Column 1 makes one full revolution and then goes back. Once this is done, column 1 would have been permuted. Column 0 can be handled as we said before from column 0 we do one full revolution and then go back to make sure that column 1 0 is permuted.

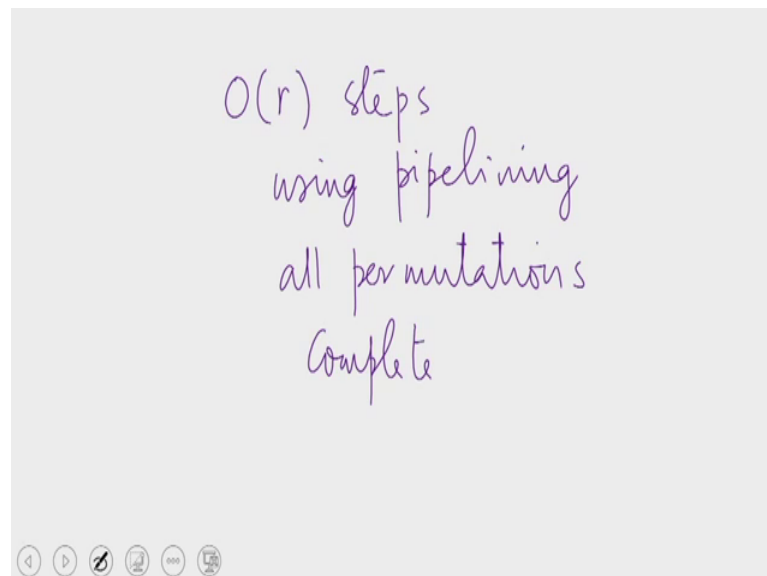
So, we can compute the configurations that are necessary for the column 0 permutation as well as the column one permutation and by extension for every single column. But then what we find is that the rotation for column 1 is one step ahead of the rotation for column 0. Therefore, when column 0 is configured using the configuration requirements necessary for revolving the column 0 elements. Column 1 could be configured the way it is necessary for permuting the column 1 elements.

So, we have several permutations going on at the same time. We have precomputed all the configurations that are necessary to achieve each of these permutations and then we interleave them appropriately. That is the first step of all these configurations are picked up; picked out and these configurations are adopted throughout the cylinder.

So, when column 0 is adopting, the configuration that is necessary for column 0 permutation column 1 is adopting the configurations that are necessary for the first step of the column 1 permutation and so on. After that, these configurations will slide by one position column 0 permutation will move on to column 1 while column 1 permutation will move on to column 2 and accordingly we will pick the permutations. So, all these permutations are rotating simultaneously. So, when column 0 completes one full revolution, column 1 also would have completed one full revolution. Only that the column 1 revolution will always be one column ahead of column 0, but then the completion of the revolution happens exactly at the same time

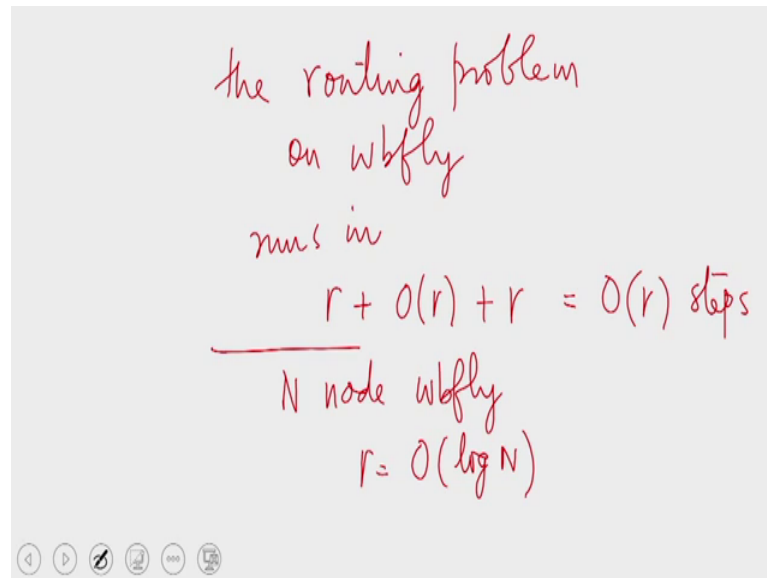
So, what happens is that all these permutations are rotating simultaneously, complete the revolution simultaneously and then rotate back simultaneously. So, when two revolutions are complete, we would have achieved the permutation of every single column.

(Refer Slide Time: 19:34)



So, in order of r steps using pipelining as I explained just now, all permutations can be executed.

(Refer Slide Time: 20:04)

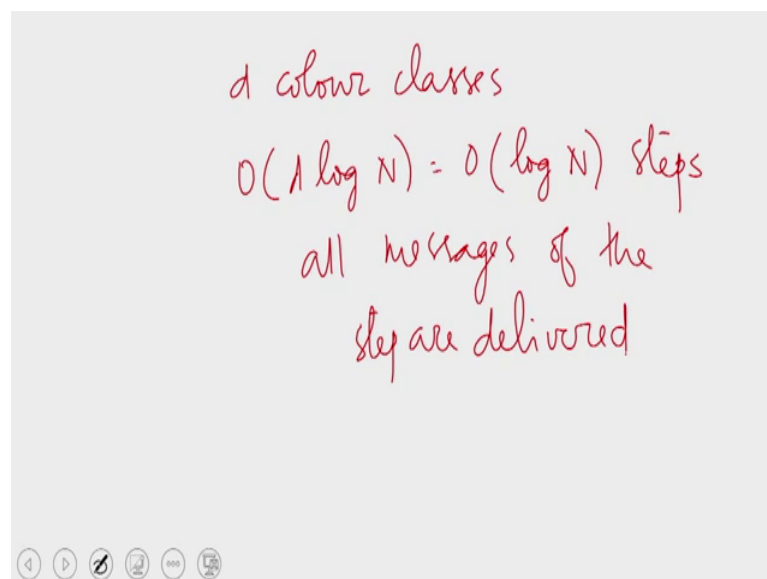


the routing problem
on wrap butterfly
runs in
 $r + O(r) + r = O(r)$ steps

N node wrap butterfly
 $r = O(\log N)$

So, now putting it all together what we find is that the routing problem on the wrap butterfly runs in r plus order of r plus r steps which is a which is order of r steps, but then remember we have an N node wrap butterfly. Therefore the dimension of this wrap butterfly is order of $\log N$. Therefore, this routing problem executes in order of $\log N$ time. Now this routing problem was the key to sending the messages. We are now considering one particular colour class of the original interconnection network. So, all the messages of this one single colour class can be delivered in order of $\log N$ time.

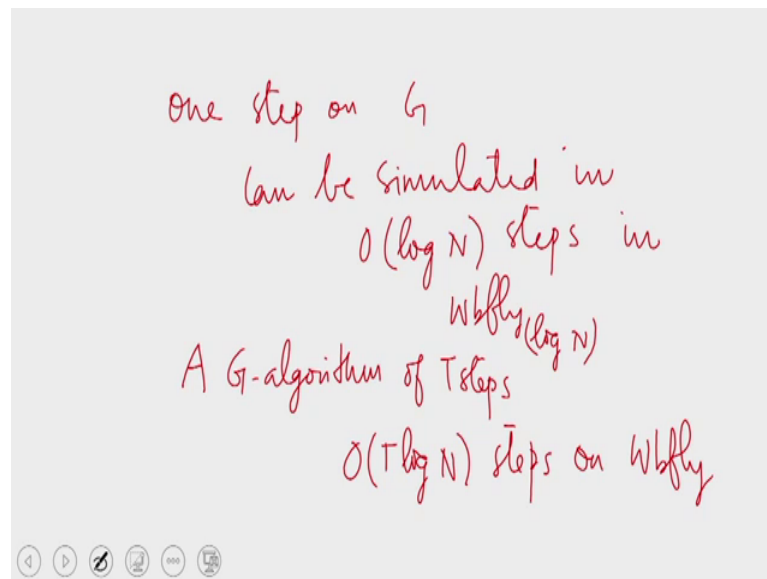
(Refer Slide Time: 21:17)



d colour classes
 $O(1 \log N) = O(\log N)$ steps
all messages of the
step are delivered

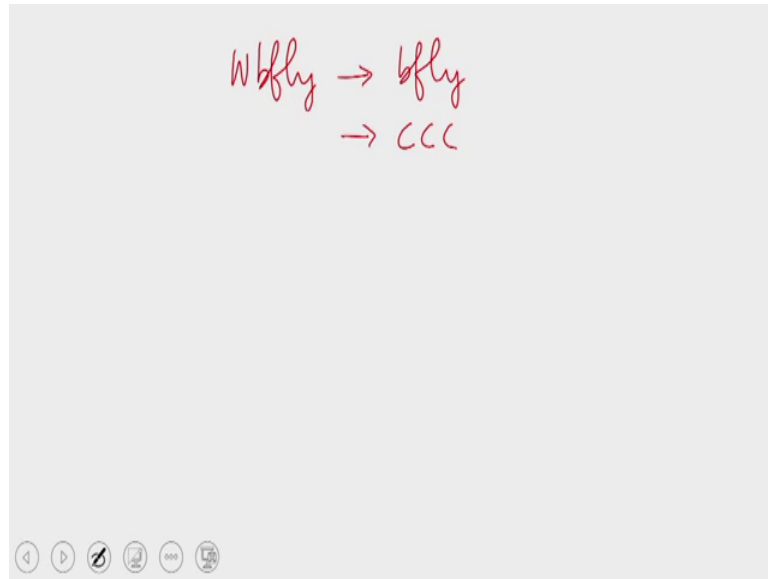
Then consider all the colour classes; we have d colour classes. Therefore, in order of $d \log N$ steps which is order of $\log N$ if these order of 1. All messages of the step are delivered. So, once the messages are delivered to complete the step all that is necessary is to do their internal computations. Since we have exactly as many processes in the wrap butterfly as there are in the given network; the internal computations call or can all be done in order of one time.

(Refer Slide Time: 22:10)



Therefore one step of G can be simulated in order of $\log N$ steps in the wrap butterfly of dimensions $\log N$. In other words a G -algorithm that runs in T steps; runs in order of $T \log N$ steps on our wrap butterfly of the same number of nodes. So, this establishes that the wrap butterfly is a pretty generic network.

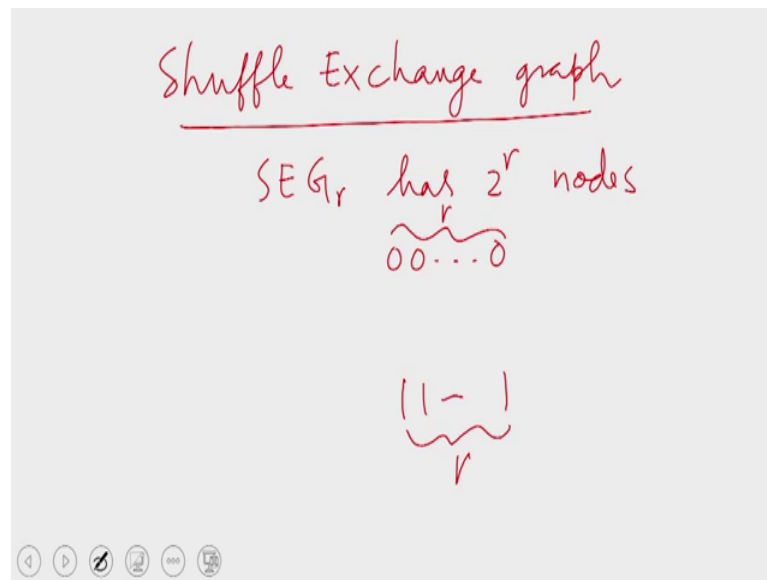
(Refer Slide Time: 23:17)



And by extension if a wrap butterfly can do this, a butterfly can do this and also a CCC can do this. All these networks we have seen are quite similar to each other therefore, what we do in one can be simulated on the other in order one extra time. Therefore, in all these models G-algorithm that runs in T steps can be simulator order of $T \log N$ steps. So, all these are pretty versatile models and they also have the additional advantage that the maximum vertex degree of an orders are constant it is either 3 or 4. In comparison a hypercube has vertex degrees that are quite large on a hypercube of N nodes, the vertex degree this $\log V$. So, that is the advantage of these networks over a hypercube.

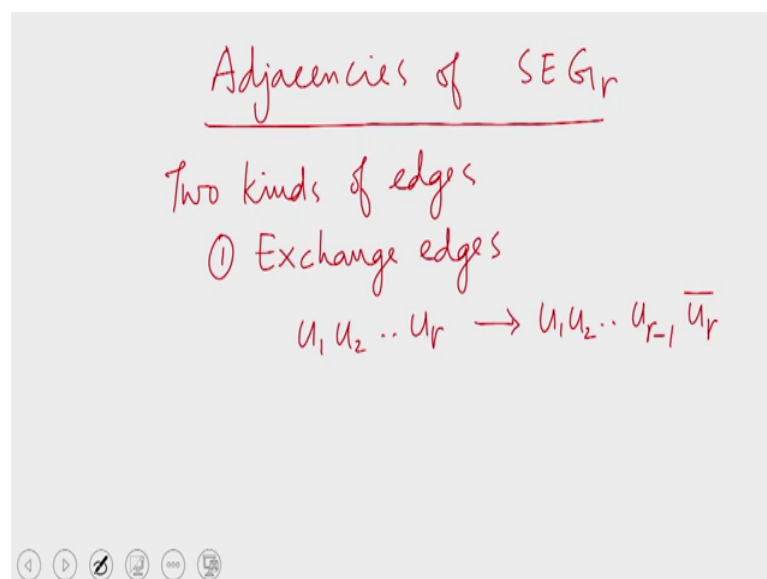
Now, we shall see another network which is also closely related to a hypercube and has several nice properties.

(Refer Slide Time: 24:28)



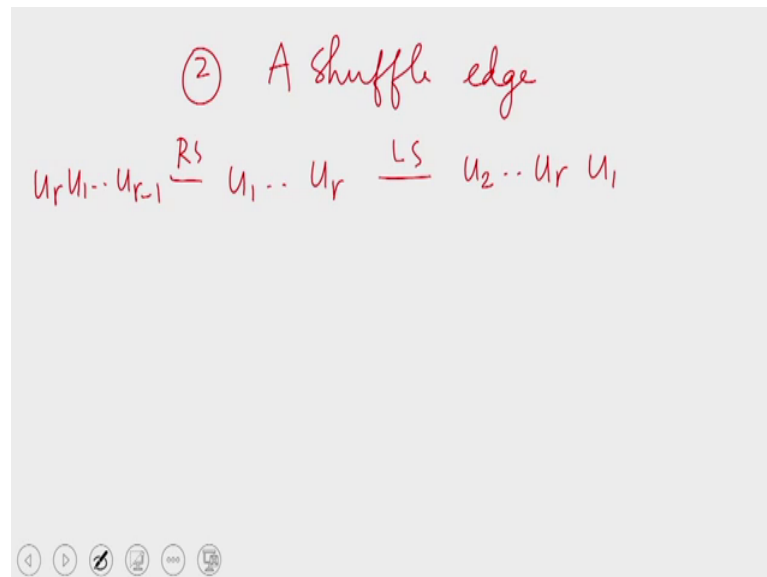
This is a shuffle exchange graph. In a shuffle exchange graph of dimensions r has 2^r nodes. We assume that the nodes are numbered consecutively using binary strings starting from all zeroes to all ones. So, there are 2^r binary strings of this sort. So, the nodes are labeled in this fashion and using these labels we can define the adjacency of a shuffle exchange graph. There are two kinds of edges in a shuffle exchange graph.

(Refer Slide Time: 25:07)



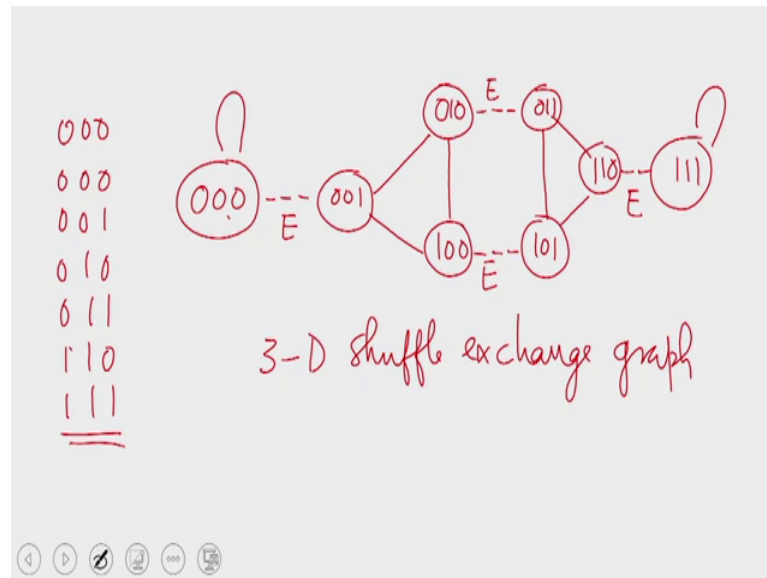
There are exchange edges between a vertex of this form and the vertex of this form. There is a vertex adjacent to a vertex whose label is identical to every bit except the least significant one. So, if two nodes differ exactly in the least significant bit, then they would be adjacent using an edge called an exchange edge.

(Refer Slide Time: 25:49)



The second kind of edge is called a shuffle edge. A shuffle edge is where we cyclically rotate the representation u_1 through u_r is adjacent to u_2 through u_r and u_1 and by extension, it should also be adjacent to u_1 through u_r minus 1 and then u_r coming in the beginning. We are now right shifting the bits. So, this is the right shift and this is the left shift. So, from a node, there could be two such shuffle edges.

(Refer Slide Time: 26:35)

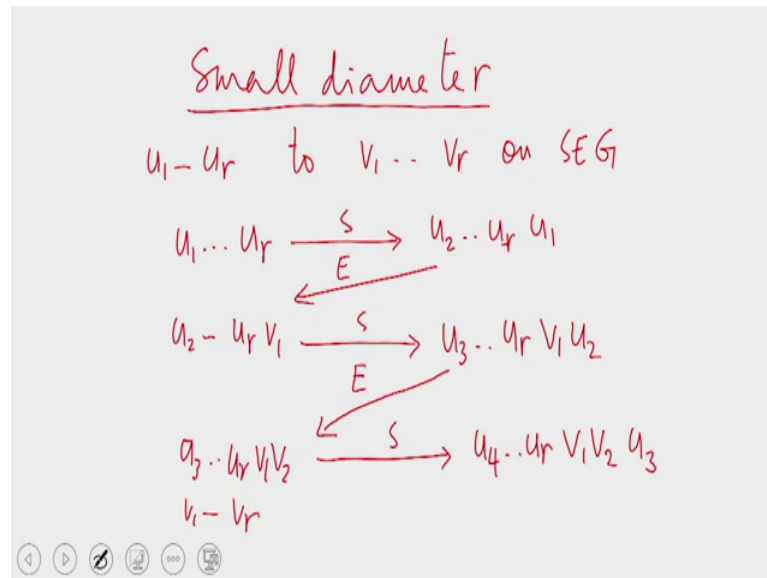


Let us consider a three dimensional shuffle exchange graph. In a three dimension shuffle exchange graph, we have adjacencies of the sort. The dotted line represents an exchange edge. A three dimensional shuffle exchange graph looks like this. So, this is an exchange edge between triple 0 and double 0 1 and this is also an exchange edge between 110 and 111. This is an exchange edge between 0 1 0 and 0 1 1 1 double 0 and 1 0 1 are also connected using an exchange edge.

So, you can see that in all the exchange edges, the two N nodes differ by exactly the least significant bit. In shuffle exchange in shuffle edges, we perform cyclic rotations. When we perform, left rotations 0 0 1 becomes 0 1 0 and when we rotate again we get 1 0 0 which we rotate again to get double 0 one. So, when we cyclically rotate to the left, we will be going through this cycle double 0 1 0 1 0 and 1 double 0. Similarly or in a mirror image situation when we start from 0 double 1 and do a left shift, we will be going to double 1 0, another left will left shift will take us to 1 0 1, another left the left shift will take us back to 0 double 1; right shifts will take us through the cycle in the opposite direction.

Now, 1 double 1 when is subjected to a cyclic rotation will give us 1 double 1 again. So, there is a shuffle edge from triple 1 to itself and also a shuffle edge from triple 0 to itself. So, this is a three dimensional shuffle exchange graph. This graph also has several nice properties; in particular it has a small diameter.

(Refer Slide Time: 28:58)



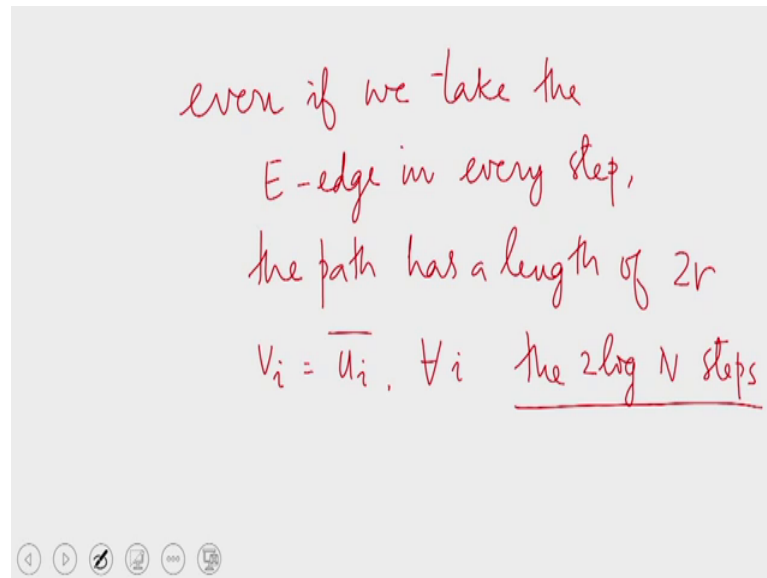
Let us say we want to go from u_1 through u_r to v_1 through v_r on the shuffle exchange graph. There could be several such paths, but we would demonstrate one such path which is of length order of $\log N$.

So, we start from u_1 through u_r ; let us say we do a shuffle which will take us to u_2 through u_r and u_1 . So, a shuffle edge will take us from u_1 through u_r to u_2 through u_r and u_1 . Then we take an exchange edge to go to u_2 through u_r v_1 . This exchange edge is optional. If u_1 is identical to v_1 , then we do not have to take this exchange edge. If u_1 is identical to v_1 , then these two vertices u_2 through u_r u_1 and u_2 through u_r v_1 are identical. So, we do not have to take the exchange edge.

So, this exchange edge is optional in our path. So, using this exchange edge, if necessary we have now reached u_2 through u_r v_1 . If we perform another shuffle, we will come to u_3 through u_r v_1 u_2 . Then again let us take an exchange edge to come to u_3 through u_r v_1 v_2 . This exchange edge is also optional, then again we will take a shuffle edge to go to u_4 through u_r v_1 v_2 u_3 and so on.

Finally with an optional exchange edge, we will end up converting every bit to the v s that is starting from u_1 through u_r , we will end up reaching v_1 through v_r . So, if you take shuffle on the exchange edges in this manner, we will be able to go from any vertex u_1 through u_r to v_1 through v_r . In the worst case we will have to take an exchange edge for every single step.

(Refer Slide Time: 31:18)



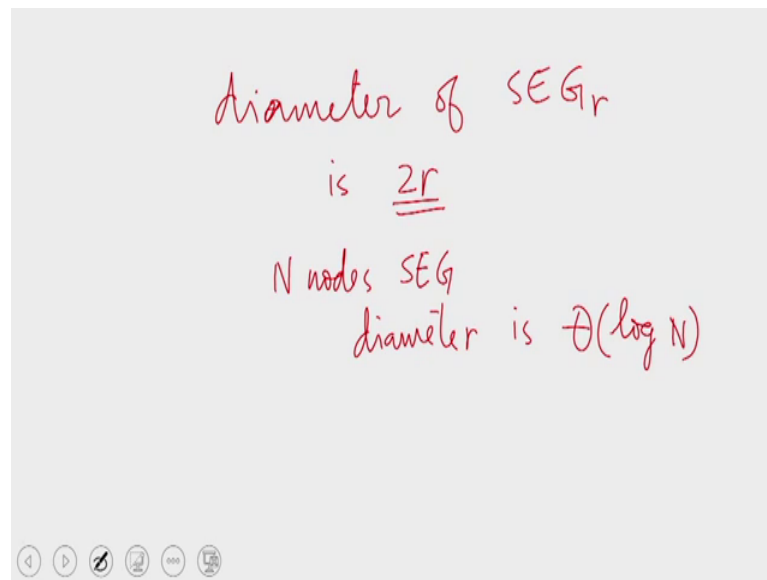
So, even if we take the exchange edge in every single step, the path has a length of $2 \log N$; $2r$ in this case. Since r equal $2 \log N$ the length of the path is $2 \log N$. So, with a shuffle edge and an exchange edge we are fixing every single; fixing a single bit. So, with $2 \log N$ steps, we will be able to fix every single bit. We will be able to transform the label from u_1 through u_r to v_1 through v_r .

So, in particular if v_i is it is the complement of u_i for all i , then $2 \log N$ steps are necessary to achieve this transformation. But that is the worse that can be therefore, from every vertex in the graph to; from you, you can go from any vertex to any vertex using at most $2 \log N$ steps. And the maximum will be taken when the source and the destination are bit wise complements of each other. For example, here if you want to go from double 0 to triple 1, you can change the bits one at a time, you perform a shuffle edge, you take a shuffle edge, come back to double 0 triple 0 and then take an exchange edge.

So, you will start from triple 0 the shuffle edge will take us to triple 0 again, then exchanges edge will take us to double 0 1, then 1 shuffle will take us to 0 1 0, exchange will take us to 0 1 1 and then another shuffle will take us to 1 1 0 and then an exchange will take us to 1 1 1. So, this is a path of length 6 from triple 0 to triple 1.

You can see that this path is not necessarily the shortest possible path in the graph, but such a path certainly exists. So, what we demonstrate is that for any vertex u_1 through u_r and v_1 through v_r , there is a path of length at most $2 \log N$ between these two vertices.

(Refer Slide Time: 33:35)



In other words the diameter of a shuffle exchange graph or when we consider a shuffle exchange graph of N nodes; the diameter is $\Theta(\log N)$; that establishes that the diameter of a shuffle exchange graph is low indeed. We shall see more properties of a shuffle exchange graph in the next lecture; that is it from this lecture. Hope to see you in the next.

Thank you.