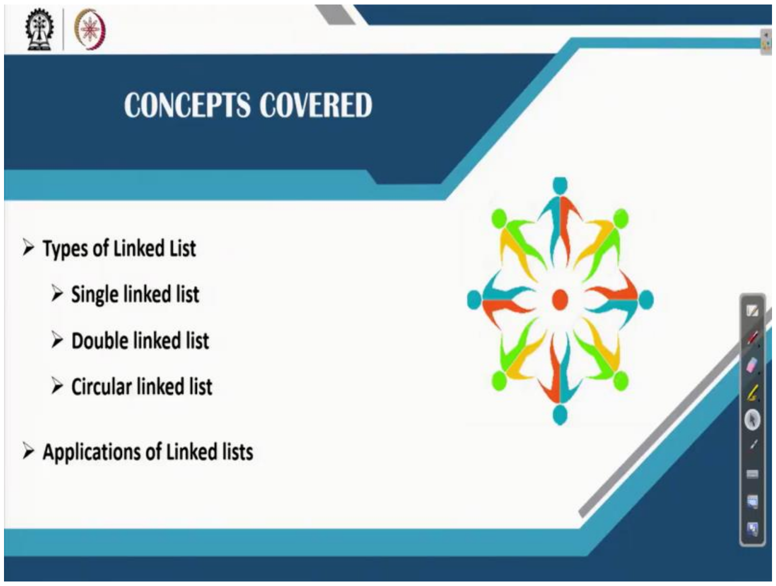
Data Structures and Algorithms Using Java
Professor. Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 17
Linked List Data Structure (Part-II)

In this fifth module we are discussing about linked list structure. In the last video you have planned about basic concept of linked list structure. Mainly, we have discussed about single linked list and circular linked list. There is another type of linked list; it is double linked list. So, today in this lecture we will study about the double linked list.

(Refer Slide Time: 0:50)



CONCEPTS COVERED

- Types of Linked List
  - Single linked list
  - Double linked list
  - Circular linked list
- Applications of Linked lists

And then few application of linked list. Now, let us first start about doubly linked list.

(Refer Slide Time: 0:57)

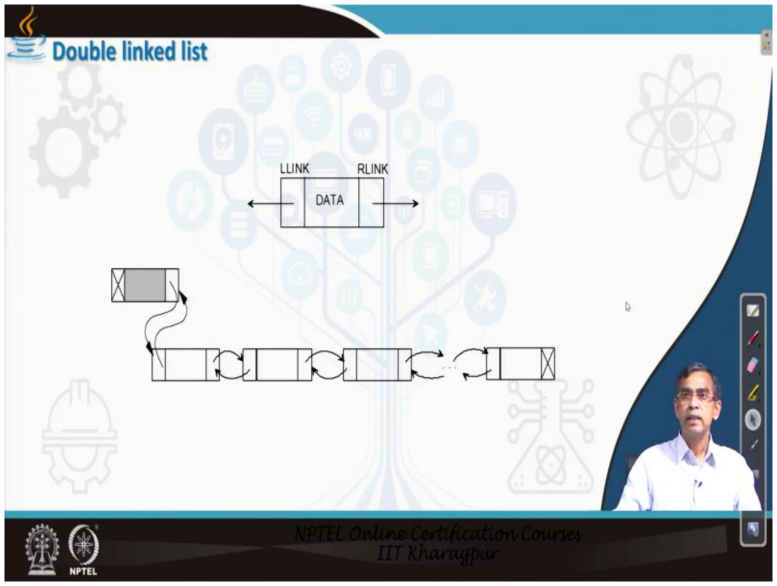
Different linked list structures

- Singly Linked List
- Circular linked list
- Doubly linked list

As the name implies in contrast to the singly or circular single linked list, doubly linked list basically points from a given node; we can point two nodes. Now, these basically pointing two nodes or linked for the two nodes for the movement of two directions. Now, if you can consider singly linked list, you can move starting from the header node only in one directions. So, from header you just go to left actually.

Now, if you maintain two links; one link from towards left and another from the right, then it will basically enables you to move in both directions to left as well as to right. So, this is basically the fundamental reason behind, why two links are to be maintained for each node. Now, so this is a concept that is basically double linked list.

(Refer Slide Time: 2:05)


Double linked list

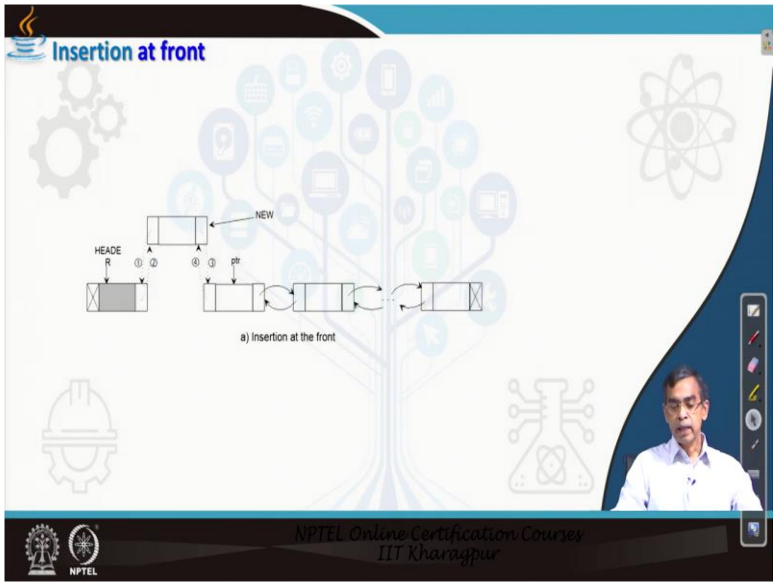
Now, unlike the single linked list, the node of a down linked list has 3 information to be stored. In this nodes structure as you can here, it has basically data like the single linked list; it contains the actual information. And then it has the two links; one is called the left link and another is called the right link. So, it basically the right link is to move towards the right only starting from header; and here left link is basically to move to the, towards the header actually.

So, it is away from header and to the header actually; from any node therefore you can move in two directions. So, a pictorially a double linked list look like this; so this is the header and you can see linked list is a homogeneous structure all nodes are of same side. So, header is also same; it has two fields. Here, the right link will gives the link to the first node, and you see for the header node, the left link is null; this indicates that this is the end actually. Now, again here you see the right link points to the first node. In point of first node whose left link is point to the header node.

And you see the two links are there who is basically use; so here from the header you can come to this one; from the first node you can come to this one. So, this say both way the movement is possible. Likewise, from any node likewise from any node here either you can move to this direction or move to this direction; that is all, just only traversing the pointer. So, if you want to move only; you can just concern only right link. And if you want to move to the left, then you only concern the left link. So, this basically the concept that you have to follow, in order to traverse a double linked list.

Now, insertion and other operations like deletion merging is similar to the single linked list operation. But, it is more complex; it is complex because we have to manage many links for both insertion and deletion.

(Refer Slide Time: 4:45)



Insertion at front

I will give a quick view of it, how the different links are to be managed for the operation. Now, let us first look at these figures, so insertion at the front; so, this is the new element that needs to be insert. Now, which are the different order the links are to be managed, I have mentioned here.

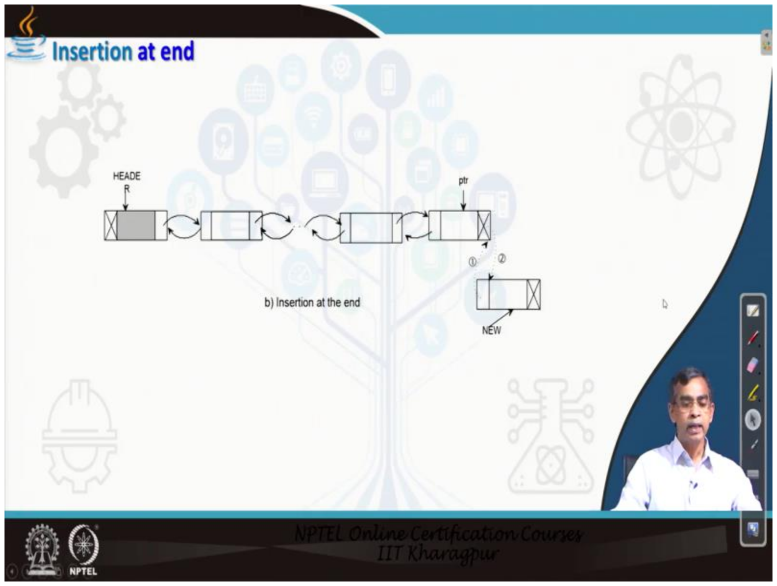
So, the first link that you should mention is basically now at this point, suppose this is the initial link list given to you, double linked list given to you; and you want to add new nodes here. So, you know that this is the address link of the first node. So, this is basically the ptr known to you; and this is the header node always known to you. And these are the fields from header you can go this, from here you can go this.

So, this field is known to you, this field is known to you, this field is also known to you; all these information is known to you. Now, what you do this? In this case is that in order to insert at front is, first the left link of the new node needs to be adjusted. So, it basically adjusted to this one, indicating that this basically points to the header. Then second is that the header links to be adjusted to the new links; so it is basically R right link of the header node needs to be set to this one.

So, this way the pointers from header to this node, and from this node to header is fixed. On fixing this way, it basically just like a singly linked list you set the pointer from header to the first node. Next is basically we have to setting the address of this field to this and to this one also. So, here first you have to do is at set the right link of the new node to the next node; that mean ptr. So, is basically you can initialize this value by ptr actually; so, this will point this one.

And left link of this node will be set by this one; by this process initially it is connected to this one; that means this is basically connected this one. Whenever, you said this, it basically lost; so this that is all. So, this way if you see the connection from this node to this node established; and this basically the insertion at front. Now, next come to the next case insertion at end.

(Refer Slide Time: 7:31)

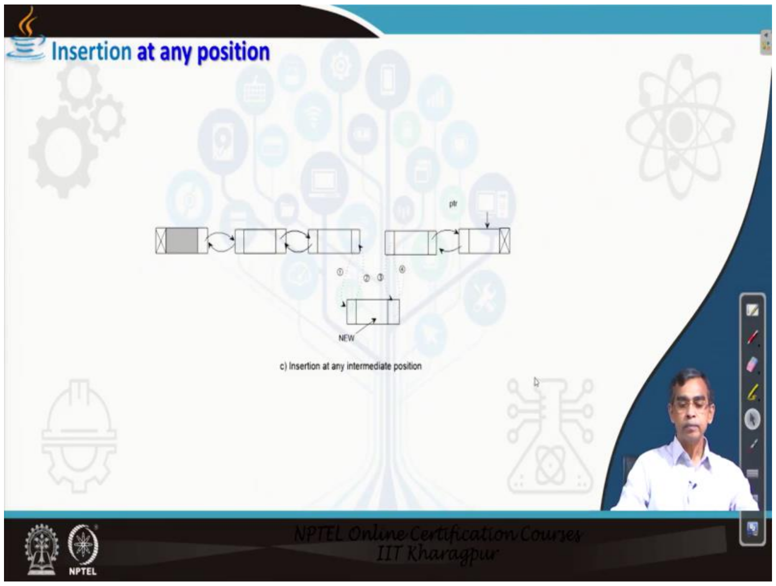


Now, here relatively easier compared to the insertion at front or insertion at mean position; because, only few links are to be mentioned here. Now, let us consider a case, so initially this is your node given to you and this is the last node whose address is ptr; that you have to keep a track of it. Now, what are the links ought to be managed here, I have mentioned here. The first the link that needs to be maintained is this is the new node that you want to add it.

So, its left link ought to be set by the new this is this is the pointer. So, ptr so here it basically we can write the ptr; so left link should be set as ptr. This means that it basically gives the link to the last node in the given list; and then we can set the right link of this node which is basically initially end empty. But, it will be set with this new node; so if the address of this node is NEW, we can set as NEW; so, this will point this one.

So, this way in given this node and after inserting, it will basically there. So, from there you can go to here, or from there you can go to there. So, this is the case of the insertion of node at the end. Now, let us come to the case of insertion at any position.

(Refer Slide Time: 9:03)



Insertion at any position

c) Insertion at any intermediate position

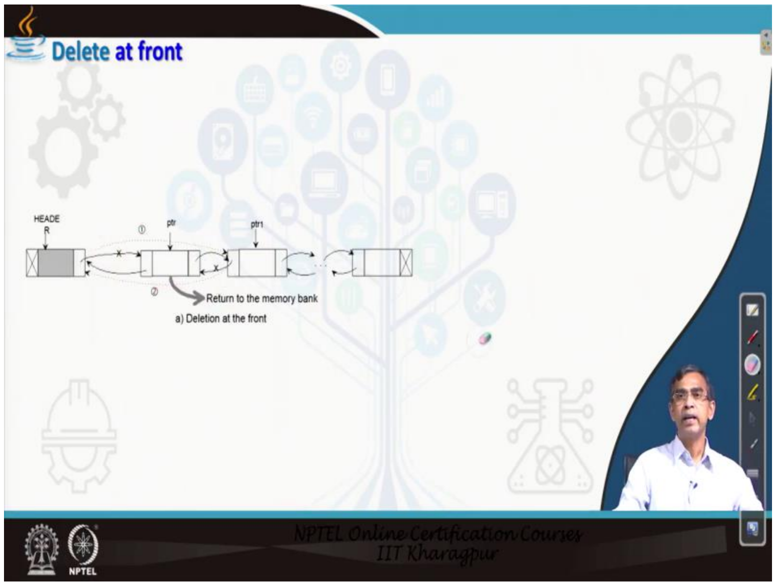
First, you have to go to the place where you want to actually insert the new node. Now, this is the case suppose; this is the initially given list; and this is the new node that you want to add into the list. And here you have to keep a track that this is the node, after which I want to insert it. So, this address is there; then you have to address the pointer of this actually, this is recovered.

Now, management of the link that needs to be carried out after keeping track of these two addresses is pretty simple. What you have to first is that right link of this node to be set by new node. So, this basically address of this will be set as a NEW; so this will link. Then the left link of this NEW node is to be set by this address. So, here we can set as a ptr; so if it is a ptr, so I can set it as a ptr. So, this link is equals to as a ptr; this that it is this one. Next we have the ptr 1 suppose, this is a ptr 1; we can do first is that ptr wants left link should be set by the NEW.

So, this basically address will be set as NEW, that means it points to this one; and right link of this field will be set by ptr 1. So, we can in the field of right link, we can right it ptr 1, so ptr 1 if you write it, then it will write this one. So, this way the new node will be added here; so that from this node, you can go this direction as well as from go this direction.

So, this is the case of insertion of node at any position; and how the link needs to be set or managed that is shown clearly. So, as you see in this case 4 links are to be adjusted; so this is about the insertion operation. Now, let us come to the deletion like insertion deletion operations also matter of link management.

(Refer Slide Time: 11:31)



Delete at front

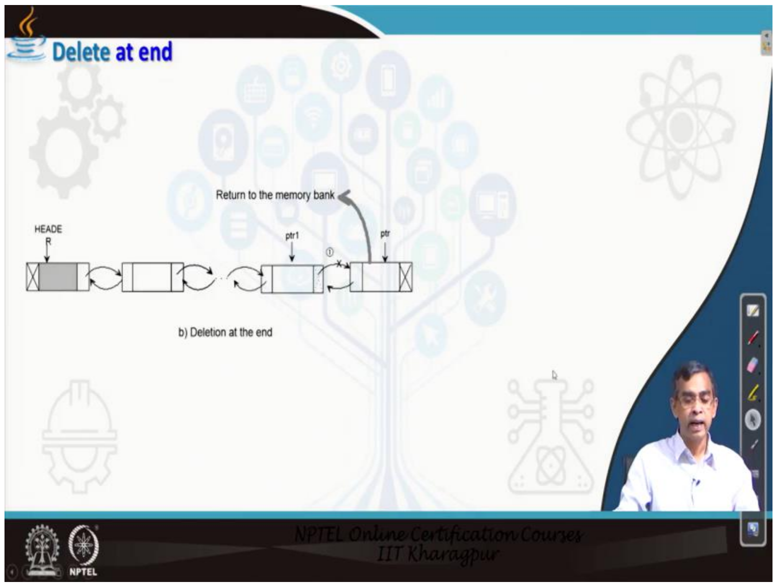
Deletion operation again at the front at end and at any position. Now, let us first discuss about deletion of node at front and how many links have to be mentioned here. From this picture you can see only 2 links have to be managed or set or manipulated. So, which two links it is shown here; so initially given this is the list; and we want to delete the first node this one. So, as a matter of this deletion, first link that needs to be managed is basically, let us come to the header.

After going to the header you you have to keep a track of the address of the first node, and then address of the ptr 1. Now, in case of suppose the double linked list is empty; so this ptr and ptr 1, both are null actually, so no problem. So, this ptr and ptr 1 is basically points to the first node and points to the second node. And if link list is empty, then ptr is null in this case; ptr 1 is null but the same thing will takes place here also. Now, let us see what we are doing here; so whenever this header is there. First we have to set the right link of this header so, right link of this header will be initialized by the ptr 1.

So, it basically gives the link to this one. So, this way the link from header to the first node is lost. So, this is the first link that you have to manage. Then second link that you have to manage is basically it is the second node, so ptr 1. So, from this ptr 1, its left link needs to be adjusted to the header, so this way it is there. And in earlier this link was to point to the first node; it is now lost. So, this way now in the modified here you see from the header you have to go there, show list and from there you have to go there is this; and this list is now make fit.

So, in order to make this long empty, we have to just make this field as empty, and this field as empty; because this last link need not to be maintained here. So, just make them null and this node free return to the bank of memory, bank memory bank. So, this is the idea about, how the deletion of operation can be done and what are the links that needs to be manipulated in this case. Now, let us come to the next deletion at end.

(Refer Slide Time: 14:19)



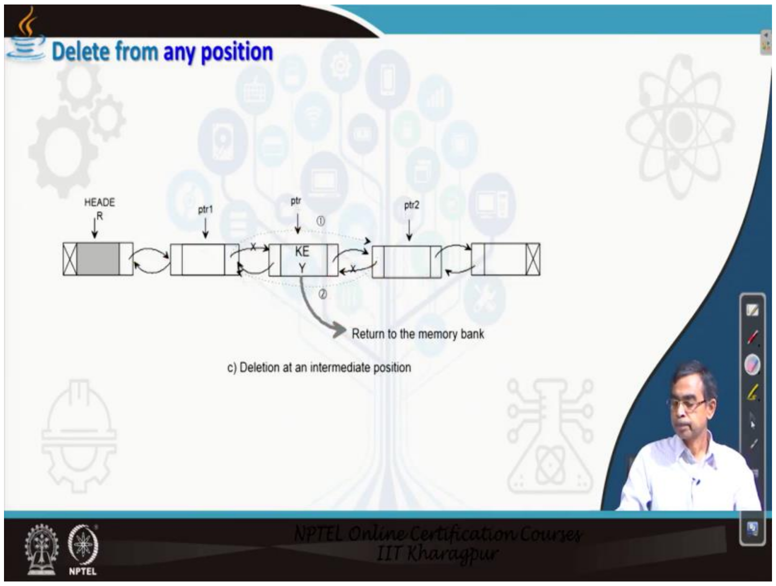
Delete at end

Deletion at end pretty simple. So, first you have to go to the last node, and also last but one node. Now, again same algorithm you can see it can work also; if we want to delete this case as end. How many we come to the programming all those things will taken care that I kept. I keep it for time being. Now, let us come to the deletion at end. So, here is the list that is given to you. And we want to remove this node from the list; and we have to keep the track of the two.

So, this is the node to be deleted and then one node just disappear this one. Now, so these are track fast; now what are the pointers needs to be managed here, first it is like this, ptr 1 which is the first but one, last but one node whose right link, this is the right link should be set by ptr. So, here initially it was pointing to this, now it will basically make it basically null; because it is numeric word.

And by setting this one, it basically null and as this needs to be return, so we can make it null. So, only this is basically setting null, and this is null and this is always null. So, only this link needs to be manipulated and it is done, then this link needs to be return to the memory bank. So, this is the deletion at end, which is comparatively simpler than the previous two cases, the last cases of insertion-deletion at front. Now, so this is deletion of the two cases, deletion at front and deletion at end.

(Refer Slide Time: 16:18)



Delete from any position

c) Deletion at an intermediate position

NPTEL Online Certification Courses
IIT Kharagpur

Now, let us come to the case of deletion at any position, which is the most complex cases out of the 3 different cases of deletion. And here is a situation this is the list that is given to you and we want to remove this element from the list. So, this is the element to, this is the node to be removed.

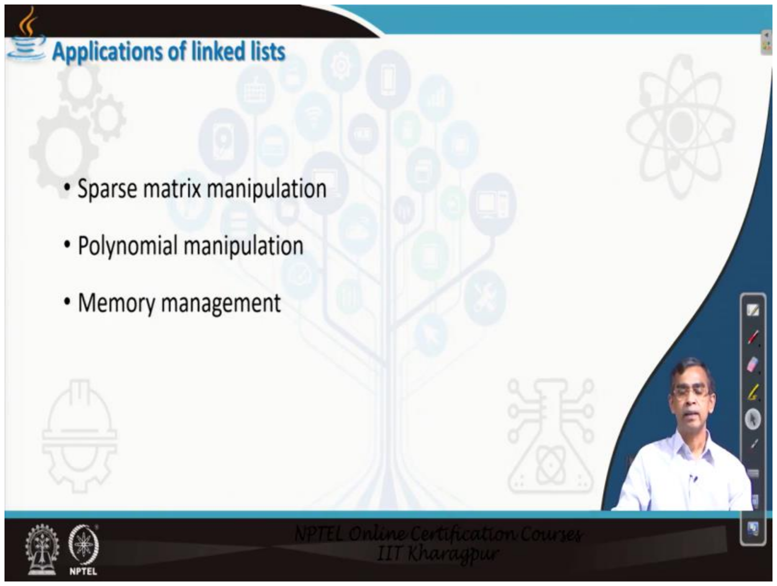
Now, for the deletion you have to first reach here; and while you are reaching here keep track in addition to this pointer. The two pointers 1 just one node before this to the left and one node just after just after this to the right so, let this the ptr 2 is the right most node, right most node of this node, and the left most node of this node. So, these are two neighbouring nodes, left neighbours and right neighbours.

Now, here the link management is like this. So, link management will start from this current node, the node that needs to be deleted here. So, what you have to do is that first we have to set this one. Initially, this link was set to this one. But, we will make this ptr 2 so this link field will be initiate by ptr 2. So, this link field is equal to ptr 2, so this it is there. So, this way you can lost this link that means earlier it was pointing to this one, now it is pointing to this one.

Next, we have to come to the right neighbour whose left link pointing to the node to be deleted. Instead of that we will just set to the ptr 1 so this link field will be initialize by ptr 1. So, this link is equals to ptr 1; if we make this then it basically set it 1. So, now this way if you see from this node to go this one, from this node to go there and this node is made free. In order to make free, make the link null and this also null, and then return to the memory bank. So, this is the case of deletion of linked list node in a double linked list from at any position.

Now, we will discuss about application of linked list. It has many applications that is why it is so popular among the programmers. But, it is not possible to discuss all applications.

(Refer Slide Time: 18:59)



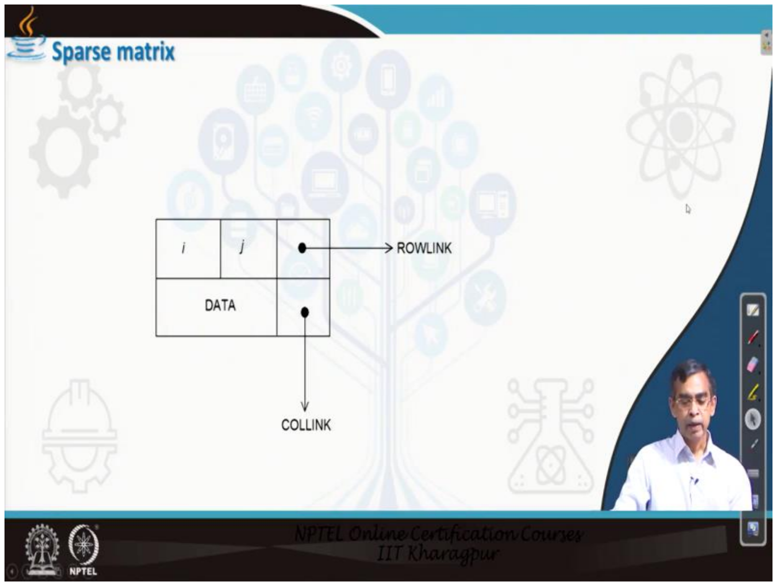
Applications of linked lists

- Sparse matrix manipulation
- Polynomial manipulation
- Memory management

I will discuss about only one application. Let us see, there are many applications of course. Sparse matrix is widely used in many mathematical operations, in many research oriented simulation; so we can use Sparse matrix. You can use the link list to represent the Sparse matrix. Then Polynomial is also very important in mathematical operation; so, a Polynomial can be represented using linked list. And the last application is Memory management, which is most frequently used on operation application particularly in operating system implementation, system software, like here where memory needs to be managed.

In fact, your Java run time memory management manager follows this linked list concept to manage their memory. These are internal to that program. I mean Java say environment design. Anyway these are the different 3 operations; we will quickly try to explain these 3 applications in this lesson. So, let us first discuss about sparse matrix representation.

(Refer Slide Time: 20:24)



Now, what is exactly a sparse matrix sees? It is just like a matrix, 2-D matrix. But, in case of 2-D matrix, all elements are not there. There are many entries are there in the matrix, which is basically null. Now, if it is a large matrix and many entries are null, then there is a huge wastage of memory actually. So, this is a memory saving one policy that using linked list; we can represent a sparse matrix taking few our memory than the actual memory that is required to represent maintaining the null entries in the matrix; so, the concept is there.

Now, in order to realize this concept, we have to decide a node structure little bit in a different way; it has again double linked list structure actually. So, like the double linked list structure two links we named this link, one is called the rowlink. It basically link any elements to a particular row and we decide another link it is just like a column link, it is. So, if it is it is a matrix; so there are different rows are there and then different columns are there, the concept is it is like this.

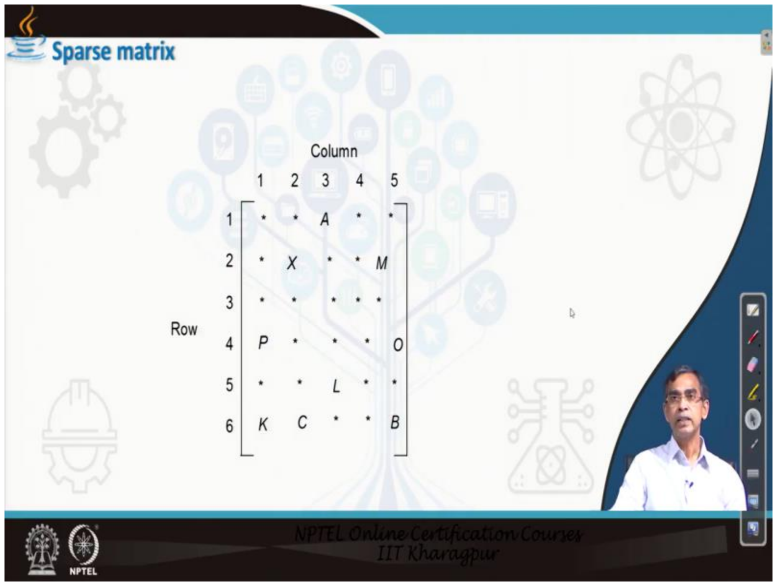
So, we will use linked list structure to represent this matrix. So, so there is basically this is one node, this is another node and so on. So, all this node is basically point by rowlink to the next neighbour in the row. And any elements, the next element in the same column will be point by this one. So, it is the concept, it is like this. So, this is basically one node to be represent as an entry, and rowlink will basically leave a point is points to the next neighbour in the same row towards right.

And the column link will basically gives the link to the immediate next to the next element in the same column; so, this concept it is there. Now, another so this is basically the links field that you have to mention maintained for a Sparse representation of a matrix, and then the data. So, this is the data field, here data field is little bit composite, each has 3parts. This is a data itself; for example in a matrix you want to store numbers, so number itself. And i and j indicating this node in which row and in which column; so, i indicates the number in the row where it is present.

So, here this is row suppose, so i value of this is row 0 and column value of this is say 1. So, this basically i equals to 0 and j equals to 1. Now, for all nodes in this row, i equal to 0. For all nodes in this row, the value of j equals to 1 and so on. So, i and j are the two information regarding the number of rows and the regarding the row and column in which a particular node belongs.

So, this is the idea is basically behind the Sparse matrix representation. Now, let us come to an example programming and everything. If we follow this concept, it is basically it is we have to write few codes that is all programming will come later on.

(Refer Slide Time: 24:42)



Sparse matrix

Column

$$
\begin{array}{c|c|ccccc|}
 & & 1 & 2 & 3 & 4 & 5 \\
 & 1 & * & * & A & * & * \\
 & 2 & * & X & * & * & M \\
 & 3 & * & * & * & * & * \\
\text{Row} & 4 & P & * & * & * & O \\
 & 5 & * & * & L & * & * \\
 & 6 & K & C & * & * & B
\end{array}
$$

NPTEL Online Certification Courses
IIT Kharagpur

Now, let us see one example, so this is one matrix. And this is a basically 6 cross 5 matrix; where 6 rows are there and 5 columns are there. The entries in this matrix are basically characters so this is basically this. Now, here only few elements are there and those are the stars these are basically the null entries. They are no elements here.

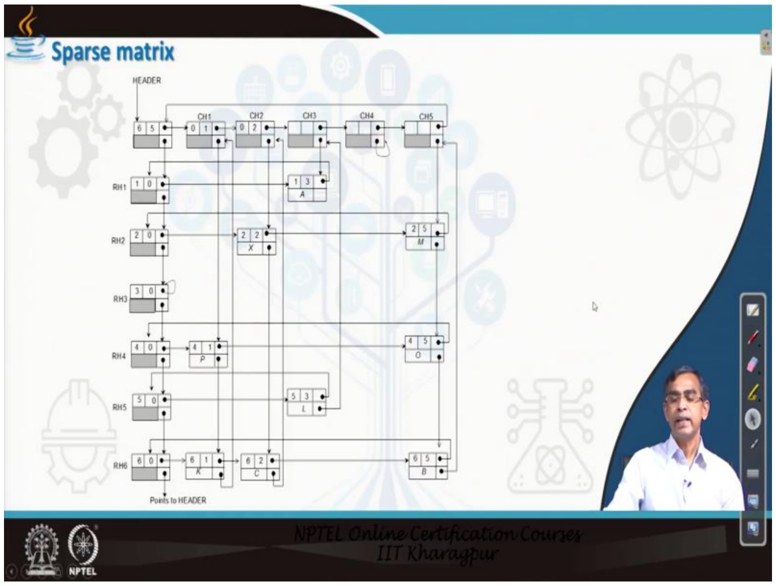
So, this is basically one Sparse matrix. Usually, usual definition of 2-D arrays and then you can maintain it easily. But, think about it is not 6-5; it is basically 600 and then maybe 500, and maybe only 10 percent of elements are there. So, 600, 500, so 30000 element out to 5 percent; means only maybe 150 elements are there. But, you have to store all the elements so unnecessary wastage of space. But, using Sparse representation, we can save this space and this is basically idea about how it can be pictorially let us see.

Anyway, so idea about is that we have to just for each row so one node have to be mention only, how many nodes are there. So, here so 1node in the first row, 3 nodes in the right and then 5 so, these are the 3, 2, 5, 6, 7, 8 and 9. So, 9 nodes are required, 9 nodes are required to store it. In addition to that we want to maintain few auxiliary nodes for each row representing; so, these are the things. So, 6 nodes are to be there representing that row. Similarly, the 5 auxiliary nodes for each column are to be there.

So, all together here total 9, 6 and 5, 20 nodes are to be store there. So, for this simple example it little bit looks heavy because out of 30, we are storing 20 for node structure. That is fine, but actual benefits can be realized whenever the size is very large, I mean matrix size is very large. In that case it will take only fewer but it will save much about it.

Now, let us see how actually physically not exactly physically, rather I should say logically, how logically the representation the Sparse representation of such a sparse matrix look like. So, you can see a logical view of the sparse representation of the matrix look like here.

(Refer Slide Time: 27:55)



As I repeat again, these are the auxiliary nodes representing each row and this is the auxiliary nodes here. And you see here rows, so zeros and these are columns so on. So, so this basically the columns are there, and then here the data field will be there. Now, actually so far our linked list concept is concern, they are basically header nodes. They are auxiliary node called so is called the row headers, and these are called the column headers.

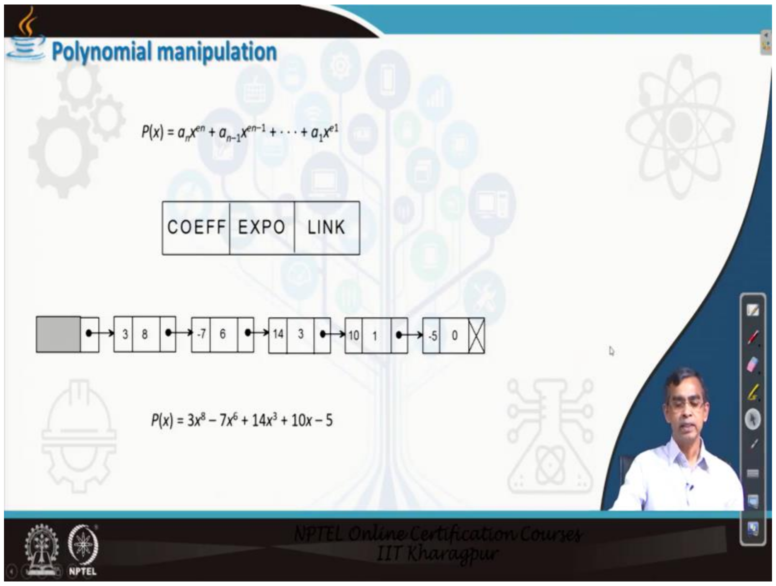
So, once the row header and column header's is setting, then the actual elements that can be stored here, this is there. So, so, here these are the elements. Now, here this basically show at showing at any nodes from here, you can show that this element is which row and in which column. So, this basically the fifth row and third column, like this one. So, effectively we are using this one as our actual data representation.

And then any other operation about the matrix, like any matching or matrix multiplication and everything little bit difficult I mean, complex algorithm is there. But, they are they cab be done.

See if you have to save the memory, maybe that you have to give some efforts so far programming is concerned. So, this is the idea about it, so it is basically memory advantages if you want to achieve, then you can think for sparse representation using linked list.

And as you can see that this basically we have followed here double linked list structure to store a Sparse matrix actually. So, this is the one application.

(Refer Slide Time: 29:58)



Polynomial manipulation

$P(x) = a_n x^{en} + a_{n-1} x^{en-1} + \cdots + a_1 x^{e1}$

COEFF	EXPO	LINK

$P(x) = 3x^8 - 7x^6 + 14x^3 + 10x - 5$

Now, next let us come to another application polynomial manipulation. Now, here I show one polynomial of single variable of degree n and here is the coefficient of polynomial as you know. Now we want to store this polynomial, how it can be? You can use array of course. So, here if you see that n number of terms, n arrays, and for each location you can store the coefficients.

So, this is the coefficients if nth degree, coefficients of nth minus 1 degree and so on; this is a one way. But, again the polynomial all terms may not be present. If it is not there, then many many array elements will remain empty. So, linked list implementation is an, is a good approach to save memory.

I am telling again, linked list concept is basically memory saving concept actually. So, if you want to save memory, then you should go for linked list structure. Other than the dynamic nature that (())(31:14). So, dynamic concept is there also memory savings. For example, if you want to static memory allocation, you have to define 100 array size. But, actually you are you want to use 10, or you want to use 200 but you defined 2, 100, then it is no more use or not applicable.

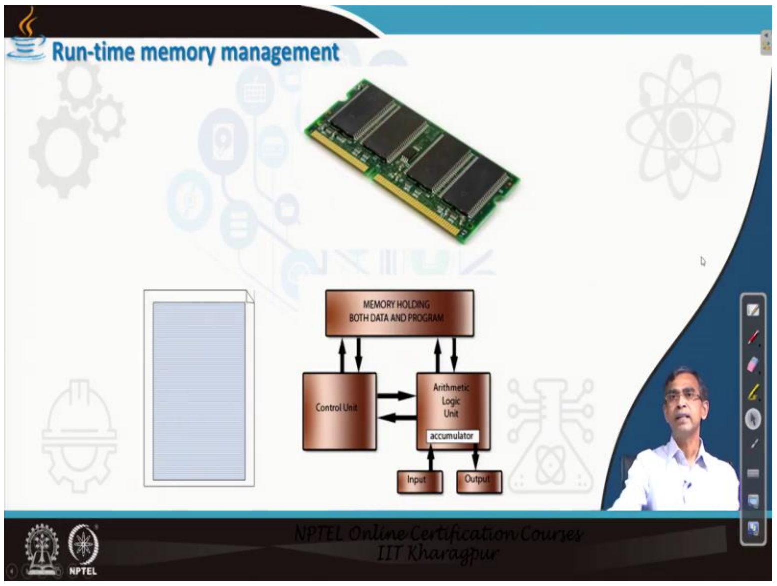
So, you your your system cannot run but in case of linked list structure as per your need you allocate, as per your requirement you free. So, that memory can be efficiently used, and then memory constant application if you want to develop program, then for in memory constant application then linked list is the only solution for you. Anyway let us come to the discussion again, so this is the polynomial as we have discussed about it. Now, here a little bit linked structure that needs to be decided, you little bit different because we have to use the many information other than the linked information. So, single linked list will follow.

As per the concept of single linked list, we have two fields, one is the data field. So, this is the data field and this is the linked list; so, this is the linked list theory will follow. But, here you see data field has the two components: one is the coefficient, and the exponent value is there, so that means degree. Now, as an example I can tell you, so this is an example. This is an example, I used linked list structure to represent a polynomial and my polynomial is this is my polynomial. You can understand what is this, so is basically coefficient of the highest degree, degree is n.

So, 3 x to the power 8, minus 7x to the power 6, 14; plus 14x to the power 3 plus 10x and minus 5; so this is the polynomial actually. Now, so so this way we can represent a polynomial of any degree n, not only any degree n. Here is a polynomial of single variable one variable, you can again modify this structure to represent a polynomial of two variables, three variables. So, in that case the data field needs to be modified, that is all. So, this is the polynomial representation polynomial representation of a simple polynomial look like this one.

Now, after having representation many more operation; for example, you want to value calculate P 5. That means for x equals to 5 what is the value or minimum value or maximum value. And in many manipulations related to the polynomial, you can accomplish using this representation. So, this is the polynomial manipulation.

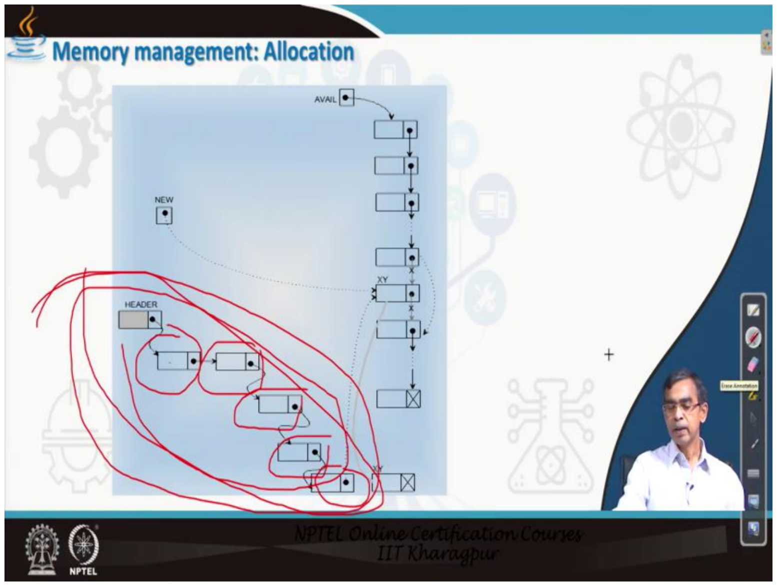
(Refer Slide Time: 34:31)


Run-time memory management
MEMORY HOLDING BOTH DATA AND PROGRAM
Control Unit
Arithmetic Logic Unit
accumulator
Input
Output
NPTEL Online Certification Courses
IIT Kharagpur

Next is memory management. Now, here whenever you need memory, you can place a request to your manager. For example, operating system. Operating system search the memory bank, whether your request can be servable or not. If the memory is available as per your demand, it returned to you to your program, your program can use. When your program no more required your memory, what you can do is that you can return the unused memory to the manager; manager will return to the memory bank.

So, this is the concept of memory management. More precisely it is called run-time memory management which is most frequent for running any application, any windows or any operating system, whatever it is there. In Java programming, it is no exception you have to manage it. Now, the memory management it is just like allocation and de-allocation concept.

(Refer Slide Time: 35:33)

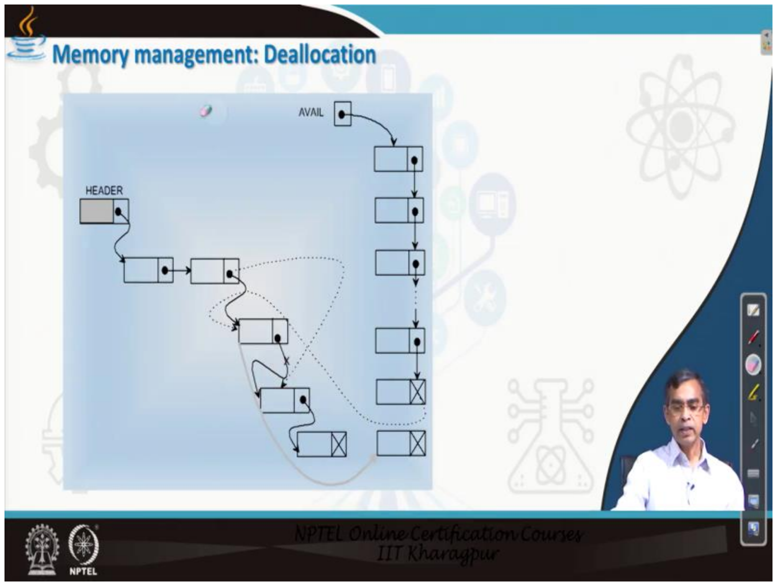

Memory management: Allocation
AVAIL
NEW
HEADER
XY
NPTEL Online Certification Courses
IIT Kharagpur

Same as the concept that adding element into your program. So, here if this is your program, program occupying your memory; managing from the different for different purpose, for different data structure. I mean, you want to maintain array, you want to maintain some variable, you want to maintain some another variable and so on. So, for everything you can maintain a node.

So, this node for storing array, this node for storing what is called the string, this node for storing another array, this node for storing integer variable, another integer variable and so on. So, if you define any other variable and for which you want to allocate memory; then you need memory again. So, you can place a request, your program can place a request; go to the memory bank, give it and then you add this memory into your.

So, total memory that is used in use in your program can be maintained by a list; as a single linked list or a double linked list, depending on what direction or both way or one way. Sometimes circular linked list also you can follow. What are the ways that you want to implement your concept that is there. So, this is the memory management, so for allocation concerned. Like allocation, there is a de-allocation also you can do.

(Refer Slide Time: 36:57)

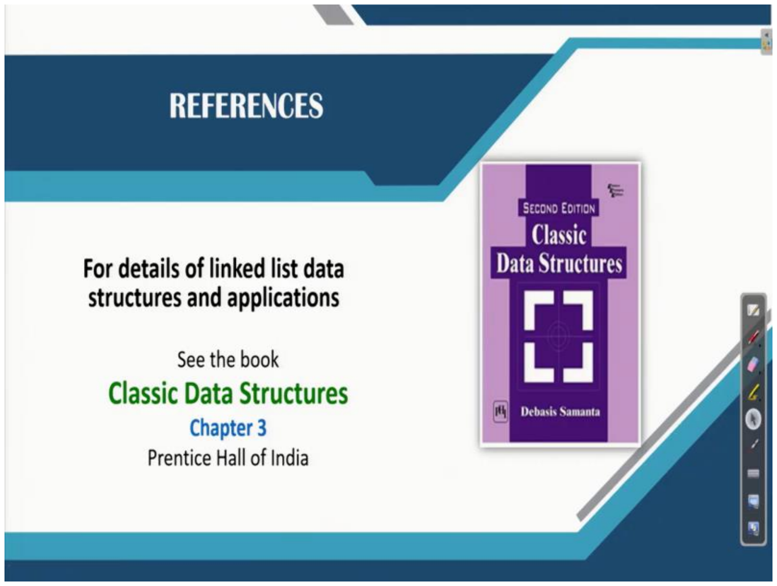

Memory management: Deallocation

De-allocation is like this. So, this is your memory. At present your application going on, and sometimes you see that this memory is no more required. You want to send back to this memory, unused memory and you can go. So, this is a memory management concept it is there. What I want to say is that although memory management concept is basically from the application development point of view, obviously these are coming there.

But, many programming language take care about all those memory management. For example, C they can allow you both static memory as well as dynamic memory. C++ same, Java is also same; Java supports also both memory static and dynamic. For dynamic memory, internally the Java run-time manager basically do all those memory allocation, policy or memory run-time memory is carried out by the Java runtime manager there.

In addition to the operating system which basically runs your Java program. So, these are the concept is basically application, really very useful application are there in many areas actually and it is not possible to discuss all applications in this limited session.

(Refer Slide Time: 38:18)



REFERENCES

For details of linked list data structures and applications

See the book

Classic Data Structures

Chapter 3

Prentice Hall of India

Second Edition
Classic
Data Structures
Debasis Samanta

For many applications if you want to have you can cancel this book again, and you can find many applications, their implementation, all programming details. Particularly, linked list related to many situations like, operations, different types of linked list and then application exact discussion is there.

And you can find the discussion there in link also that is given to you that you can follow. So, today with this I just want to remain it here, will carry this discussion towards the programming. Now, will go to Java programming issue, whatever the topic that we have learned. How they can be implemented in Java programming there; so our next session will be programming for linked list. Thank you.