Data Structures

Lecture 4

Stack Representation

Let us start with how stacks are represented. So stacks are represented with a node in the linked list with the node containing data. That is what is to be represented and link pointing to the next node. And the also you have and access to a stack throw a node. Which is designated to a stack and also maintain the count the no. of elements of the stack. Now first think we going doing is create a stack. To creating a stack, you have to first dynamically the allocate a stack. Who size is equal to the size of the stack and if the stack top is equal to null. When that is what you are doing. To the initialize the top and we are also since creating the stack. And any top initialize equal to be null and count to be equal to zero. You are not a actually create a node. This is first step in creating a stack. So once you created a stack we went to do the operation from stack. Important operation has push already seen push operation. This is if you look at linked list representation of stack. In the push operation similar to inserting a node in the front of a list. Because already operations are done with a stack at the top of the you can access the stack accessed to throw only the top. Insertion and deletion when you represent a stack as the linked list is in the front. So here. Same as inserting a new element to the list before the first. So let us assume that this is the element. You want to insert and this is the previous stack. So previous stack has this is the count and is the top.

So top is pointing to the first element in the stack. Are you wanted to insert the new element? First to create an element for the node. And this is the address of the node. And this is containing the data red and pointer next. So that's what is available. When you are going to insert this into the stack. It's come like this. Now this is Pnew and which know this is address of this. You create next. Change the next pointer to point to the address of blue. And change to things top to point to address of red and increment the count be equal to 3. So this is insertion of push operation. We don't call it insert in stacks. We call it push operation. Push into the stack. Which is represented as a linked list. So this is the push operation. The code for the push operation so first thing will you do introduce create a node. In that node you creating a node. You have get it from the allocated area. If there is no allocated area. That is can't get a new node. You can't insert. Once you got that. In the new position you put data. Pointed to that data whatever to insert. And you make the pointer of the new location of the new node. That is to be created.

Pointed to original stack top. And now change the stack top pointed to the new pointer implement top to the point to the new node. And increment the count of the stack by one. And return one. And Return one this is indicated. So this is what we do for insertion into a stack. As we consider to this is very similar to the insertion into the front of the linked list. Now let us look at the next operation of stack. Which is pop. Pop is to delete the element and as you know you can only the delete element from top of the stack. That is the constraint that you have on stack. So this is the same as the removing first element from a linked list. So what you have to do is. This is an original stack.

Now the only element you can delete or remove from the stack from and do the pop operation. In this element. No other element can remove. So what you do is. You make the pop. Now point to blue. Ok. That is it. So now you can access the stack. Only. Throw this. Now you have a node which is short of bankling. So this can be recycled. So you make a pointer to Point to this node. So that it can be recycled. So this is the pop operation. So the code for the pop operation. As follows. So you have a pointer to point to the node. Which you want to the return. And if the stack already discussed you want to a deletion and what do you want we consume must. First ensure that the data structure from which you want do the deletion. Is not empty. So that is what is doing here. Just checking the count what whether equal to zero. You saying that will return zero. Which means that is an underflow. That is no element. To be delete it. Otherwise you make this pointer to point to the original stack. This is for recycling purposes. And then what you do is. Make the stack top. Point to the new stack. Top and if free, dlt pointer. What do you going to doing here is, string is the node. That has been deleted. And decrement the stack count and return the value of stack. This is code for the pop stack. Both these cases the stack is represented using a linked list and these particular data structure is very useful. When we are doing for organizing the nodes in an allocated list. It is called allocation list. And these represented linked list. Who store all the nodes are has been deleted are when you want to delete. You store for recycling take allocated list. Which is represented as stack. When you want to insert to take a node from same allocated list. So the next stack top is already discussed before stack top doesn't change stack at all. Just access the top element of the stack. So again if the count is only greater than zero. That mean stack doesn't contain anything. Then just take out the pointer to the top element. Make that data to be returned in data pointer and return 1. If it is less than 0. That means no stack. So you return. So that is top. Know hearing to change of the list, this is can normal thing. And empty stack a stack count. So what you do here in. if the count is equal to zero then the return value. Otherwise the stack is not empty. This is a code for Boolean function. This is an empty stack. Which

is return value from the stack is empty. Which is value is false. The stack is not empty. This is return a count. So already maintain the count. There are implementation do not maintain count of the stack. Certain implementation you maintain count .so return stack. That is it. So the count can be equal to zero. The stack is empty. You can also check for full stack. This is assuming that u have decided that only so many nodes going to give for the stack. This is a stack. If you have stack full condition then malloc is failed. If malloc is failed, it means, you cannot there is no temporary node to be given to for the operation. In which case you have a return 1.

So this is a full stack. Where you cannot, remember that in full stack condition checking whether there are nodes available for putting into the stack. If there are not saying full otherwise putting full. This is not like an array implementation already allocated the size of the stack. This is the destroying stack Destroying complete stack. So you want to delete all the nodes in the stack. So if the stack top is null it means already stack is empty. You don't have after do any destroying. So if it is not null. Then you make delete pointer to point to the top of the stack top of the point to the next node and free delete pointer. You keep doing this. Till the stacks it will come. Come to the last node in the stack. The Stack is now empty then the destroy the head node. So free that and you return null. So this here having a by loop. Here this is the logic. Where you keep on deleting nodes of push. Popping of the nodes. Till start becomes. Now what you seen upto now is the some of the implementation of stacks. Where we look at linked list implementation of the stack. In this particular implementation of the stack. What you done is we are assume that you have a count. Also it is not necessary. So in case you don't have a count the code has to be appropriately modified.

Stack Applications

Now let us go to next some of the application. Using stack. Very interesting applications. I said used in no.of places in computer science itself. For example, the stack is a particular specific data structure that is used for quick sort algorithm is a difficult data structure. It also used in other applications. Like very important application where you have to maintain the program counter and return address in the case of recursive calls in function calls. So this is very important data structure as well as computer science.is concerned. We are looking at 2 very different application. One is the depth first search. And one is the N_Queens problem. So let us first look at the depth first search. The depth first search actually backtracking type of concept. That it uses. And this backtracking type of concept we do use stack. Stack is a difficult data structure that is used for doing back tracking. So this problem as follows discover a part from start to goal and

the solution you have to go deep. If there is an unvisited neighbour go there then you retreat along the path to find an unvisited neighbour. The outcome is if there is a path from the stack to goal. DFs very finds such a path. So that is going to go deep and cannot enough to do. Go to DFS wise. That is an idea. So this is a difficult application of a stack. So let us soon given a tree and if you want to find the goal. So initialize to start at 1. That is pushed on to the stack. Then you go to two depth number. To get to the child. So push 2. When you push 5. 5 has no children so you go to the sibling with 6. Then Go to child 9. After then you don't have anything. So Pop it up. Pop is 6 out. Then pop 5 out. Pop 2 out. When you go to the next child which is 3. Then you go to 7. Then you go to 10. That is depth first we go to 10. Now you have identifying the value. So that the path followed 1,2,5,6,9. Then back tracking up and then go to 1. Find the next child go down. Ok. That is the path followed by the depth first search. So let us algorithm for the DFS. So the stack is first initialized. And you stack the operation look at push start. Now while the stack is a not empty. This is keep track. You data in the top element. If it is the goal you just put access. If T has a unvisited neighbour. Choose an unvisited neighbour n. Mark N as visited and push n into the stack.
Otherwise that is if doesn't have a unvisited neighbour. When you backtrack take in the next node from the stack. Keep doing this. Till stack is empty. Are you reach the goal? So this is the DFS algorithm. Of course you searching the node in a algorithm you can also happen that here may be a failure. So this is a difficult application of stack. Where the stack is used to maintain the node. When we want to do the DFS algorithm. As specially backtracking algorithm. This is a difficult data structure that used many concept used back tracking. Also used stack is a basic data structure. For implementation purpose.

N-Queens Problem

Now you going to look at what is the N_Queens problem? This is a difficult problem in artificial intelligence and the N_Queens problem is actually problem associated with the chess board. So in this lecture I will explain the problem and then how it is solved using the stack. OK.
So first let us see what the problem is the NQueens problem can be defined as follows. It is a backtracking problem to solve the NQueens. Suppose you have 8 chess queens & and a chess board. Can the queens be placed on the board? So that no two queens are attacking each other? Now what do you mean by attacking. We will see that as we go long. So the problem is this. Two queens are not allowed in the same row, or in the same column. Or along the same diagonal. So that is the idea. So 2 queens are not allowed on the same row, and the same column, or along the same diagonal, the no.of queens and the size of the board can vary.

Normally use all this problem for the NQueens problem. We are going to take it can simple example and show you. But basically solve it for any queen. It after all chess board is the square. So you can 64 * 64. Where N = 64 and so on. So basically NQueens can be represented this way you consist of NQueens. And chess board that you have consist of N rows and N columns. So your aim is placed NQueens on the boards. Such that if you have a queen here. We don't have queen anywhere here and also don't have anywhere queen here. So you gone queen same diagonal on the same row, same columns, along this same row. That's the problem so are idea is to try to find solution to this problem. The computer program to solve this problem. Program tries to find a way to place N queens on the N * N chess board. Following the instruction we already seen.

Now comes the application of stack. We going to uses a stack to keep track of where each queen as already been placed. So first place a queen on the board. The position of the new queen is stored in a record which is placed in the stack. And also you have an integer variable to keep track of how many rows have been filled so for. So these are two things that a going to use keep track of how to place the NQueens. So the is how the program works you have take 4 */ 4 board in very simple example to show the how the program works and this is the no. of queens to be filed. Which is 4 and this is stack to be used. Stack contains record. Which gives the position of the queen. Which is 1,1 is the position 1,2 mean this 1, 4 means first row , 4th column and so on. And this is the variable. Which keeps track of how many queens have already been placed. OK. This is the way. This is the complete show. You have a N * N board. You have NQueens, you have a stack. Which give the position of queen and you have the no. of queens that mean placed directly as the program.

Now let us start the program. So first you place 1 here. Then you come do this what happen so that is 1,1 has been placed here. Then you come here. Now can you placed queen here, you can't the queen placed here. Because along it a same diagonal so the logic uses that you shift it once to the right and see whether breaks it in a rules now that is no queen along the column, that is no queen along the row, that is no queen along the diagonal. So this is allowed. Now you have a queen at 1,1 you have a queen at 2,3 and 2 have been filled. Now we are to place this. N row you can place it here. Can't place it here. Because if it is in the diagonal. So what you do is? Go up here and place it here. can you place it here 3, 4 so 3 rd row, 4 Th column you can't place it here. Because of the fact at that along diagonal.

How the program works? When we run out of remove in or row, pop the stack,

reduce filled by 1 and continue working on the previous row. Let us look at that. What is mean these? When you can't place, pop it off. Go to the previous row and you see whether place it along that. Along the right or left. You will see the logical as we see go long. Again I tell you. When we run out of room in a row, pop the stack, reduced filled by 1 and continue working on the previous row. Whatever you placed in the previous row, here are having so ok. That is what is happened here. This is r th position. We still have to queens 2. And reduced filled by 1. Now we continue working on row 2, shifting the queen to the right. Now instant have it here. We pop it up. And shifting the queen here. Now thus has to been incremented this position has no conflicts, so we can increase filled by 1, and move to row 3. So now we move to row 3. Ion row 3 what happens, we can't place it here. Because diagonal. So placed it here. And then we go to 3 one.

And this is the conflicts. We stack again first column conflicts. Second column that is no conflicts. So that is how we so along to the problem. You can work it on row. You see how 4 queens are filled. So the logical basically to be repeat. That is you take the row placed a queen there. Then you go to the next row place the queen there. When conflict arises. What you do is? You tried to push the queen to right. If it is possible and you know no conflicts arises. If you come across a conflict then you pop the top element from the stack. That's you are a backtracking. You are not using that. Already placed it, removing from the stack. Which remove it. And try again to reworks on the previous row and that's how you do the queen.

Nqueens problem. So for are nqueens problem. Let us see this is slightly different from the problem. That mean normally used. We are going to first talk about the pseudo code.so here what we do is. We first initialize the stack. Which is date structure. We want to use. Keep track of artificians and we also place the first queen. Pushing its position onto the stack and setting filled it equal to be 0. This is pseudo code repeat these steps. If there are no conflicts in the queen then we do else if there is a conflict and there is room to shift the current queen right ward do it. Else if there is a conflict and there is no room to shift the current queen right ward then what we do. You have do keep on doing this. Repeat these steps. There are no conflicts increase filled by 1. If filled is now what then doing here and take this each of the steps and giving the details. If there are no conflicts with the queen what we do increase filled by 1. If filled row is now N. That means filled all the queens then the algorithm is over. Otherwise, move to the next row and place a queen in the first column. So that this steps is over. So what we and done is. We done this. If you take one queen ok. We are placed it the first in the row. We see there is a conflict. Then there is a conflict moving a right. If there is a conflict and

there is a room to shift the current queen rightward. Move the current queen rightward adjusting the record on top of the stack to indicate the new position. So, this is in the same row. Even that you can't do. Position where you have can't fill the there is conflict. Can't move rightward and fill. Because there is a conflict again. If there is a conflict and there is a room to shift the current queen right ward. This is place to have do a back.

So what you do. You keep popping the stack and reducing filled by 1. Until you reach a row where the queen can be shifted rightward. When you shift this queen right. Then what you doing is. What ever done before you backtracking. Can backtracking till you reach the position? Where you can have another alternative. Which is not placing that. But moving it to the right. If it is possible. So the movement to the right once step of the time. But you move one also. So that is the NQueen problem. By the NQueens problem is very challenging is that. We are actually redoing the work. We already done. So that we get the choice of what we doing actually in the algorithm is we first start with the easiest method. Placed the first row that we see. Then we keep on doing it till the and the second we can take is every ok. When I placed in that place there is a conflict. So I will let move right to see. If you move to right and see, fill a conflicts. So I will let more right to se. If you move to right and see, fill a conflicts. Only then I go back. And I check whether. What I will done can be changed so that's can be accompanied. You can properly try thus for 6 * 6 or 4*4 on your own on. Similarly act how it works. That will be a good exercise for you. When you for you do understand. How the stack works and how backtracking algorithm work also. That is explain both the concept of backtracking as well as the application of stack to backtracking. Now let us look at some problems. Associated with stacks. There is know up to implementation of stack.