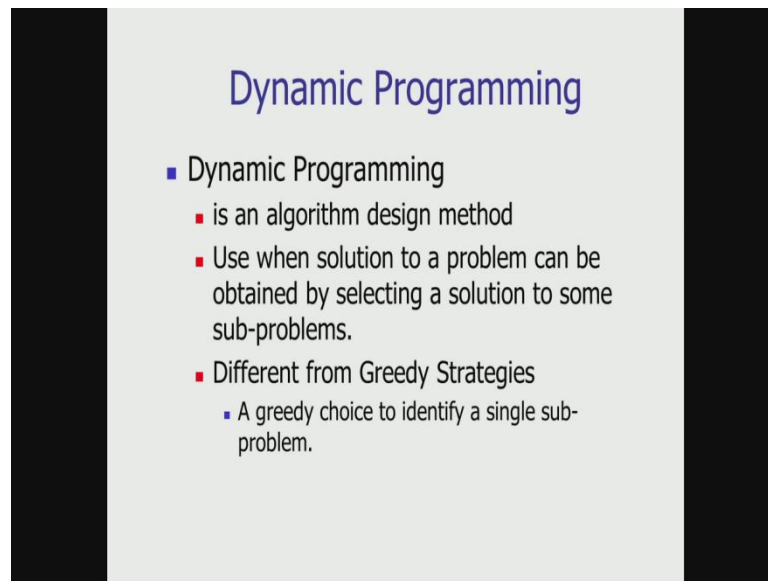


**Algorithms Dynamic Programming**  
**Prof. N.S. Narayanaswamy**  
**Department of Computer Science and Engineering**  
**Indian Institution Technology, Madras**

**Lecture - 56**

After looking at the basic algorithms and techniques like the greedy techniques, today we come to advance techniques called dynamic programming.

(Refer Slide Time: 00:21)



**Dynamic Programming**

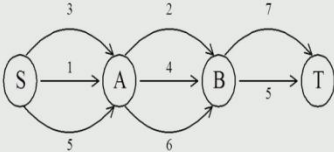
- Dynamic Programming
  - is an algorithm design method
  - Use when solution to a problem can be obtained by selecting a solution to some sub-problems.
  - Different from Greedy Strategies
    - A greedy choice to identify a single sub-problem.

Dynamic programming is an algorithm design method, it is used when the solution to a problem can be obtained by selecting a solution to some sub problems, this is very loose way of putting it. But, we will set of all the formal details in the lecture today. Most importantly the role of this slide is to you tell you that, dynamic programming techniques are very fundamental different from greedy algorithmic techniques. Often in the greedy algorithms there is this power of making a greedy choice, which we have seen in earlier lectures. And the greedy choice identifies a single recurrence sub problem to solve.

(Refer Slide Time: 01:03)

### The shortest path

- To find a shortest path in a layered graph



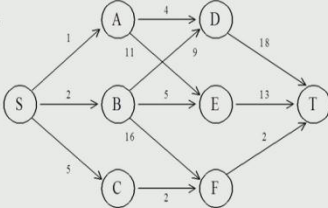
- Apply the greedy method :  
the shortest path from S to T :  
 $1 + 2 + 5 = 8$

Let us look at the shortest path problem as an example, what you see is the layered graph it has 4 vertices S, A, B and T and you also see the edge cos. And if one uses the greedy technique to find the length of the shortest path from S to T by being greedy. In the sense that from S we choose the edge of these to 8, then from A we chosen edge of the least weight and form and B we choose an edge of the least weight. Then, we can conclude that the greedy algorithm will output path of the length 8; however, it is quite clear that the length of the shortest path is indeed 9.

(Refer Slide Time: 01:44)

### The shortest path in layered graphs

- e.g.



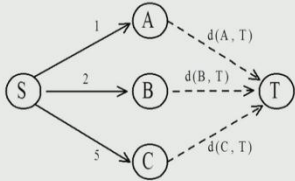
- The greedy method can not be applied to this case: (S, A, D, T)  $1+4+18 = 23$ .
- The shortest path is:  
(S, C, F, T)  $5+2+2 = 9$ .

In this example, the algorithm does output the shortest path, but let us look at the shortest path in a slickly more complicated looking graph. If we are greedy then the greedy algorithm will choose the path, which goes from S to the vertex A. Because, the edge weight 1 is shortest among all the edge are go out of S and from A it is choose us the cheapest edge width, which is 4 and a takes it to the vertex B and D we choose the shortest edge, whose cause is 18 and we conclude that the greedy algorithm outputs a path whose length is of 23 units. As opposed the shortest path which is from S to C, C to F and F to T and the cos of this path is indeed 9. Therefore, it is clear that to solve the shortest problem a greedy strategy does not work.

(Refer Slide Time: 02:45)

### Dynamic programming approach

- Dynamic programming approach:



```

graph LR
    S((S)) -- 1 --> A((A))
    S -- 2 --> B((B))
    S -- 5 --> C((C))
    A -.- d(A,T) --> T((T))
    B -.- d(B,T) --> T
    C -.- d(C,T) --> T
  
```

- $d(S, T) = \min\{1+d(A, T), 2+d(B, T), 5+d(C, T)\}$
- Recursive formulation of shortest path.

Let us just look at the dynamic programming approach and let us also makes some observations about the shortest path. The length of the shortest path from S to T is obtain by calculating the shortest paths viva the vertices A B and C respectively and then picking the shortest path among the 3 in this example, it is quite clear S and S has 3 neighbors only, which are called A, B and C respectively.

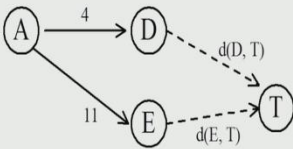
Now, if we know the shortest path from A to T and the shortest path from B to T and this shortest path from C to T respectively, then adding of the appropriate edge cos from S and then choosing the minimum will very clearly give the shortest path from S to T. Therefore, what we have now observed is that, it is solve the shortest path problem, we need to solve recursive versions of the shortest path problem from neighboring vertices.

Let us this look at the formulation of the distance from S to T R. In other words shortest path from S to T, clearly as we just discuss distance from S to T is minimum over 1 plus the distance from A to T, the one comes because the edge weight from S to A is 1, 2 plus distance from B to T. Because, edge weight from S to B is of cost 2 and 5 plus distance from C to T the edge weight from S to C being 5. Indeed if we know the shortest path from A to T, B to T and C to T respectively, this minimum gives us shortest path from S to T.

(Refer Slide Time: 04:33)

**Formulating the optimum**

- $d(A,T) = \min\{4+d(D,T), 11+d(E,T)\}$   
=  $\min\{4+18, 11+13\} = 22.$

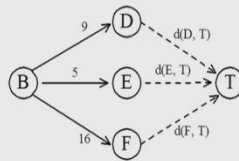


Now, the distance from A to T the relevant part of the graph shown here, the distance from A to T as you can see is the minimum of four 4 distance 4 plus the distance from D to T plus. The distance from A to T is given by the minimum of 4 plus the distance from D to T and 11 plus the distance from E to T, where D and T are only neighbors of A.

(Refer Slide Time: 05:09)

## Dynamic programming

- $d(B, T) = \min\{9+d(D, T), 5+d(E, T), 16+d(F, T)\}$   
 $= \min\{9+18, 5+13, 16+2\} = 18.$
- $d(C, T) = \min\{2+d(F, T)\} = 2+2 = 4$
- $d(S, T) = \min\{1+d(A, T), 2+d(B, T), 5+d(C, T)\}$   
 $= \min\{1+22, 2+18, 5+4\} = 9.$



We can similarly now formulate, the distance from B to T the distance from C to T and distance from S to T can then be used to calculate. So, the distance from B to T is again 9 plus the distance from D to T. The distance from B to T is given by the minimum of the 3 distance is viva D viva E and viva F respectively. Similarly, the distance from C to T that is just single path that goes through F and this is given us 2 plus the distance from F to T.

One can over observe that the distance from S to T will indeed choose the path that goes viva C which is of cost 9, because that is where the minimum is obtain. What we have done in the slide show for is so observe that a recursive formulation of the distance function in terms of distances from other vertices to the designation is helpful in computing the shortest path.

(Refer Slide Time: 06:26)

## Calculating the distances

- $d(S, A) = 1$   
 $d(S, B) = 2$   
 $d(S, C) = 5$
- $d(S, D) = \min\{d(S, A) + d(A, D), d(S, B) + d(B, D)\}$   
 $= \min\{1 + 4, 2 + 9\} = 5$   
 $d(S, E) = \min\{d(S, A) + d(A, E), d(S, B) + d(B, E)\}$   
 $= \min\{1 + 11, 2 + 5\} = 7$   
 $d(S, F) = \min\{d(S, A) + d(A, F), d(S, B) + d(B, F)\}$   
 $= \min\{2 + 16, 5 + 2\} = 7$

It is also possible to observe that one can calculate the distances from the recursive formulations which we have just observed. One can observe with the distance from S to D and intermediate vertex, not essentially T is the minimum of the distance from S to A plus a distance from A to T. And similarly distance from S to B plus the distance from B to D, in other words we are taking the minimum of the distances via intermediate points A and B respectively. Similarly, the distance from S to E and distance S to F are also presented.

(Refer Slide Time: 07:11)

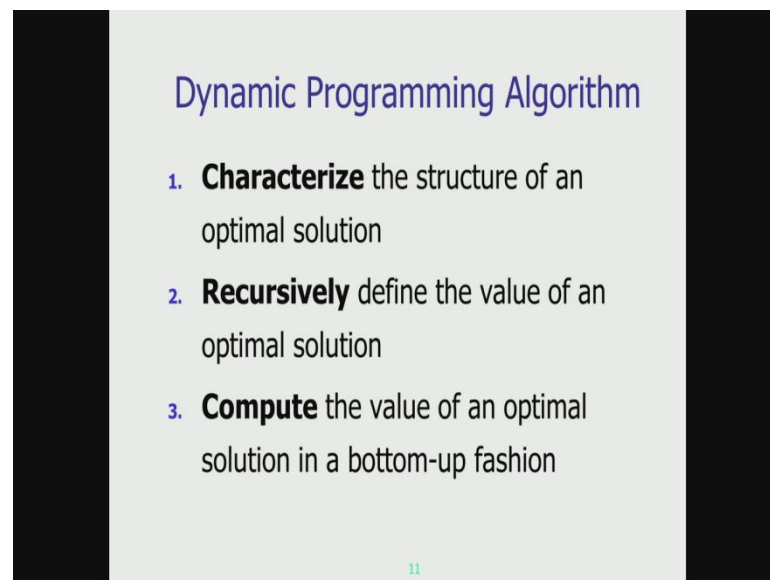
## Dynamic Programming

- Used for **optimization problems**
  - A set of choices must be made to get an optimal solution
  - Find a solution with the optimal value (minimum or maximum)
  - There may be many solutions that lead to an optimal value
  - Our goal: **find an optimal solution**

Now, that we can see in this example, let us just formally state dynamic programming and its purposes. It is used for solving optimization problems which are minimization problems or maximization problems. And in all these problems a set of choices must be made to get an optimum solution, in the examples that we look at the appropriate edges must be chosen and find the length of the shortest path from S to T.

The choices are the edges that must be taken by a shortest path from S to T, there may be many search paths. For example, in an undirected graph I am every path with a least number of edges from S to T is indeed the shortest path and our goal is to find an optimal solution, then other words to find the optimal set of choices.

(Refer Slide Time: 08:05)



**Dynamic Programming Algorithm**

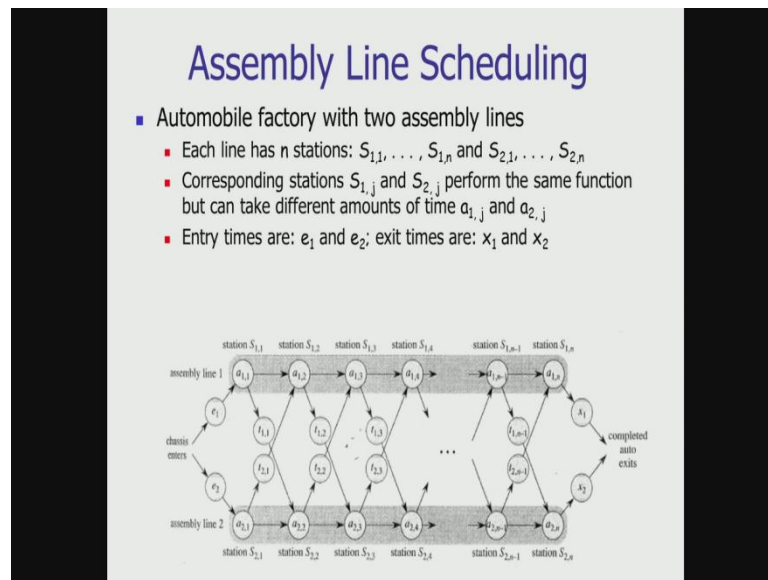
1. **Characterize** the structure of an optimal solution
2. **Recursively** define the value of an optimal solution
3. **Compute** the value of an optimal solution in a bottom-up fashion

11

When we designed a dynamic programming algorithm, the approaches that we take is to first understand the structure of the optimal solution and characterize this optimal solution. By characterization we mean, that we identify certain very important properties of solutions which necessitated then to be optimal solutions, that is what we mean by a characterization. After characterizing the structure of the optimal solution, we then recursively go ahead and write the recursion the recursive expression for the optimal solution.

And then compute the value of the optimum solution in a bottom up fashion, the last computation is indeed done via program. But, we first to are analytical exercises and the computation indeed implements the recursion.

(Refer Slide Time: 09:04)



Has an example, we study the assembly line scheduling exercise, which is an introductory exercise and dynamic programming in the standard textbook by corner license and drivers and the pictures that you see are taken from the textbook. Assembly line scheduling can be very intuitively visualize, it is a very important problem in the area of the manufacturing. The frame work is that is often automobile factoring, the frame work is often a automobile factory, which has two assembly lines, each line has  $n$  processing station.

On the first line the processing stations are called  $S_{1,1}$  to  $S_{1,n}$  and in the second line the processing stations are call  $S_{2,1}$  to  $S_{2,n}$  respectively there are  $n$  stations. Now, corresponding to the stations, each of the stations have a certain processing time, in particular for the station  $S_{1,j}$  that is the  $j$  station on the line 1 the processing time is  $a_{1,j}$  units of time. And similarly at the station  $S_{2,j}$  the processing time is  $a_{2,j}$  units of time.

The entry times into the station are  $e_1$  and  $e_2$  respectively, the entry times into the lines that is the entry into the first station  $S_{1,1}$  and the entry into the first station  $S_{2,1}$  are  $e_1$  and  $e_2$  respectively and the exit times from the station are  $x_1$  and  $x_2$  respectively.



(Refer Slide Time: 10:58)

## Assembly Line Scheduling

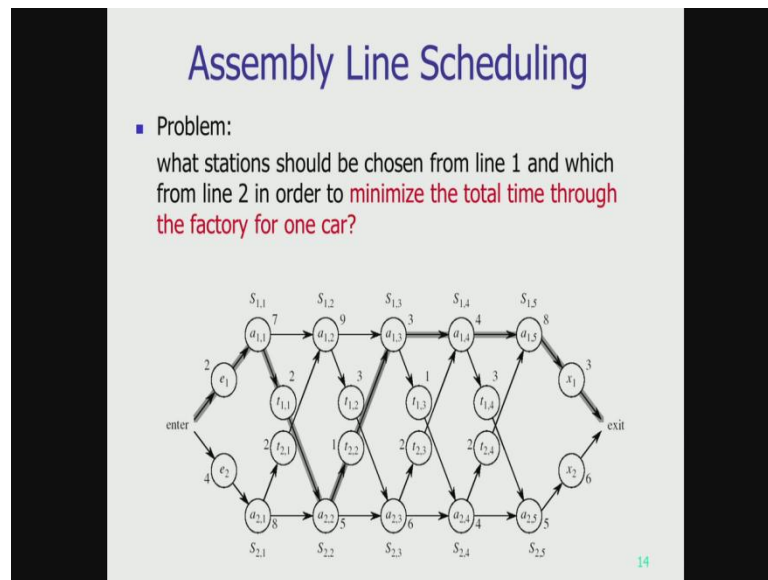
- After going through a station, can either:
  - stay on same line at no cost, or
  - transfer to other line: cost after  $S_{i,j}$  is  $t_{i,j}$ ,  $j = 1, \dots, n - 1$

13

One can assume that in this automobile factory are car chase enters one of these to assembly lines and then goes through from one station to another and exists with a completed vehicle alone the way. The car station, the car alone the way are car can stay on the same line by going to the next station in the line and it does not pay any cost in terms of time duration or it go transfer to the other line with a cost which is  $t_{i,j}$  units of time.

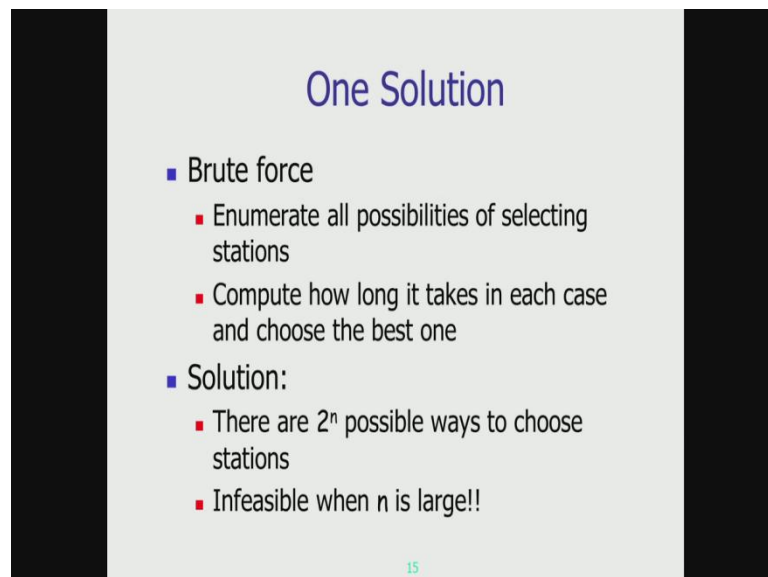
In other words if a car get process at station a  $S_{1,1}$  then it can either go instantaneous to station  $s_{1,2}$  are in  $t_{1,1}$  units of time it can go to the station  $a_{2,2}$ .

(Refer Slide Time: 12:11)



So, what is the problem in this whole exercise? The problem with a whole exercise is that a chase must be rooted through this network, it must visit every station it does not matter in which line it get's processed. But, it must get process at every station and exist in the minimum amount of time, this is for a single chase.

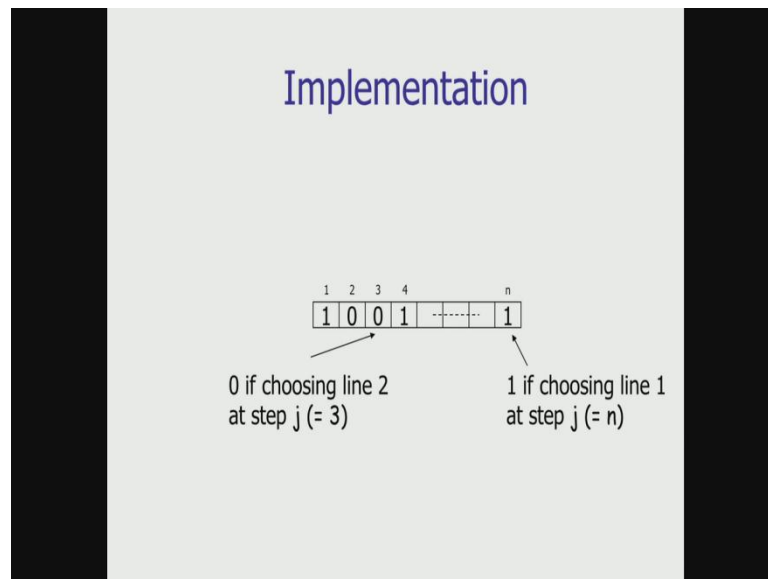
(Refer Slide Time: 12:48)



Now, one of the ways of solving the problem is to actually try all the possible ways by which a chase go through the assembly lines. And it is very clear that there are 2 power n of them and computationally spending 2 power n units of time to actually find out which

is the most optimal sequence is not an efficient approach.

(Refer Slide Time: 13:22)



So, the implementation of this idea is essentially to have a big binary vector, which essentially says if a certain bit. For example, in this case at step 3 if a bit set to 0 then the vehicle goes through station 2 in step 3 and if it is want the vehicle goes through station 1 in step 3. The implementation of this Brute force algorithm is to keep a big vector, where 0 indicates the chase will go through line 2 and 1 indicates chase will go through line 1.

And once these are fixed the time that the chase spends in the factory can be easily calculated, it is just a length of the path that goes to these stations. The some of the time durations on edge will tell us, the total time that the vehicle will tell us the total time chase spend in the factory, before existing. And then we can choose the big vector which gives the least solution and indeed we would have solve the desire problem; however, we would as spend  $2^n$  units of time.

(Refer Slide Time: 14:45)

### Structure of the Optimal Solution

- How do we compute the minimum time of going through a station?

17

To avoid this inefficient see in terms of amount of time that has been spend by the algorithm, in finding the optimal path. We try an understand the structure of an optimal solution, which is the first step in the design of the dynamic programming algorithm. In this graphics we indeed have highlighted, the optimum path in dark color, we will see how the algorithm will indeed find this particular path.

(Refer Slide Time: 15:23)

### Structure of the Optimal Solution

- Number of ways from the start via  $S_{1,j}$ 
  - We have two choices of how to get to  $S_{1,j}$ :
    - Through  $S_{1,j-1}$ , then directly to  $S_{1,j}$
    - Through  $S_{2,j-1}$ , then transfer over to  $S_{1,j}$

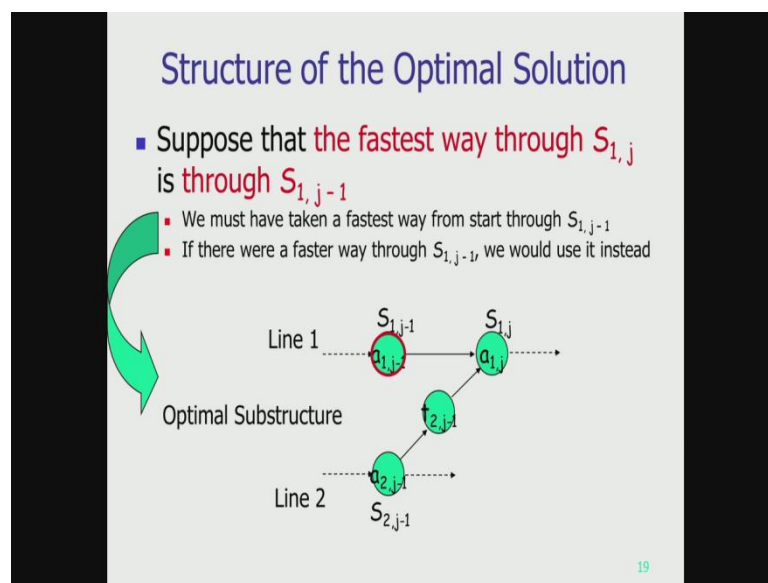
18

To understand the optimal solution, we invent a parameter of interest. Let us look at the station  $j$  on the first line and let us ask, if a shortest path goes through  $S_{1,j}$  then do we

have some information has to shortest paths to other stations that presided it that are computed in that shortest path. To understand these structure of the optimal solution, we ask the following question, if you look at the set of all possible ways from the start to exit viva  $S_{1,j}$  there are two possibilities for are path that goes viva  $S_{1,j}$ .

The previous vertex before the station  $S_{1,j}$  could have been the station  $S_{1,j-1}$  and direct transfer to  $S_{1,j}$  are the previous station could have been  $S_{2,j-1}$  that is the station on that second line and the transfer over to  $S_{1,j}$ .

(Refer Slide Time: 16:52)



We known make the observation that, the fastest way through  $S_{1,j}$  is through  $S_{1,j-1}$ . In other words, if we consider all the paths that go from the start to exist, which go through  $S_{1,j}$  and among these, if we consider the path which is the fastest. And let us say that, such as shortest path or such as fastest path indeed goes through  $S_{1,j-1}$ . Then, it is also clear that such a path must have taken the fastest way from start through  $S_{1,j-1}$ .

The reason is very straight forward, indeed if there is a faster way through  $S_{1,j-1}$  then we could have used it and found a faster path through  $S_{1,j}$ . This is really the optimal structure are it is a structure of an optimal solution.

(Refer Slide Time: 18:08)

### Optimal Substructure

- **Generalization:** an optimal solution to the problem *find the fastest way through  $S_{1,j}$*  contains within it an optimal solution to sub-problems *:find the fastest way through  $S_{1,j-1}$  or  $S_{2,j-1}$*
- This is referred to as the **optimal substructure** property
- We use this property to construct an optimal solution to a problem from optimal solutions to sub-problems

20

The structure of an optimal solution is, the fastest ways through  $S_{1, j}$  contains with it an optimal solution to the fastest way through  $S_{1, j-1}$  or  $S_{2, j-1}$  whichever is there in the fastest way through  $S_{1, j}$ . This is refer to as we optimal sub structure property, it is this property that is used to right down the recurrence for the length of the shortest path from S to T or the fastest path from S to T.

(Refer Slide Time: 18:43)

### A Recursive Solution

- Define the value of an optimal solution in terms of the optimal solution to subproblems

21

So, what we are going to do now, which is a second step in the process of dynamic programming is to right down the optimal solution in terms of the optimal solutions to

sub problems.

(Refer Slide Time: 18:57)

### A Recursive Solution (cont.)

- Definitions:
  - $f^*$ : the fastest time to get through the entire factory
  - $f_{i,j}$ : the fastest time to get from the starting point through station  $S_{i,j}$

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$$

Let us introduce some necessary notation, let us assume that  $f^*$  is the fastest time to go through the whole factory. And let  $f_{i,j}$  denote the fastest time to go from the starting point, through the station  $S_{i,j}$ . Therefore,  $f^*$  is very clear,  $f^*$  is a minimum of the two terms which is  $f_1[n] + x_1$  that is the time taken to exit from line 1 comma  $f_2[n] + x_2$  which is the time taken to exist from line 2.

Therefore this expression tells us that we need to compute  $f_1[n]$  and  $f_2[n]$  recursively.

(Refer Slide Time: 19:53)

### A Recursive Solution (cont.)

- Base case:  $j = 1, i=1,2$  (getting through station 1)  
 $f_1[1] = e_1 + a_{1,1}$   
 $f_2[1] = e_2 + a_{2,1}$

23

Let us start of with a base case, let us consider  $f_1$  of  $n$ , let us consider  $f_1$  of 1 that is through station 1, what is the fastest path that goes through station 1 on line 1, it is a time taken into enter line 1 and get process set of the first station, in this case as you can see it is 9 units of time. But, the generic formula that we can write down is that,  $f_1$  of  $n$  is  $e_1$  plus  $a_{1,1}$  this the time taken to enter into line 1 and the time taken to be process at the first station on line 1.

Similarly,  $f_2$  of 1 is a time taken to enter into line 2 plus the time taken to be process at station 1 on line 2, the formulae are return there  $f_1$  of 1 is  $e_1$  plus  $a_{1,1}$  and  $f_2$  of 1 plus  $e_2$  plus  $a_{2,1}$ .



(Refer Slide Time: 21:01)

### A Recursive Solution (cont.)

- General Case:  $j = 2, 3, \dots, n$ , and  $i = 1, 2$
- Fastest way through  $S_{i,j}$  is either:
  - the way through  $S_{i,j-1}$  then directly through  $S_{i,j}$ , or
 
$$f_i[j-1] + a_{i,j}$$
  - the way through  $S_{2,j-1}$ , transfer from line 2 to line 1, then through  $S_{1,j}$ 

$$f_2[j-1] + t_{2,j-1} + a_{1,j}$$

The diagram illustrates the transition between two lines. Line 1 has stations  $S_{1,j-1}$  and  $S_{1,j}$ . Line 2 has station  $S_{2,j-1}$ . Arrows show the path from  $S_{1,j-1}$  to  $S_{1,j}$  (labeled  $a_{1,j-1}$  and  $a_{1,j}$ ), and from  $S_{2,j-1}$  to  $S_{1,j}$  (labeled  $a_{2,j-1}$  and  $t_{2,j-1}$ ). A central node represents the transfer point with time  $t_{2,j-1}$ .

$$f_i[j] = \min(f_i[j-1] + a_{i,j}, f_2[j-1] + t_{2,j-1} + a_{1,j})$$

24

In general we can write a down recursive specification of  $f_1, j$  or  $f_2, j$  they are symmetric let us just look at the case for  $f_1, j$ . So, the fastest way through  $s_{i,j}$  has only two possibilities. The two possibilities are use the fastest way to come to station  $j$  minus 1 on line 1, use the fastest way to cross station  $j$  minus 1 on line 1 and then the process at station  $j$  on line 1 or use a the fastest way to arrive at station  $j$  minus 1 on line 2, then transfer to line 1 and then get process at station  $j$  on line 1.

Therefore,  $f_1$  of  $j$  is a minimum of  $f_1$  of  $j$  minus 1 plus  $a_{1,j}$  comma  $f_2$  of  $j$  minus 1 plus time to transfer out of the  $j$  minus 1th station on line 2 to the  $j$  station of line 1 that is  $t_{2,j-1}$  plus  $a_{1,j}$ . Observe that this is the recursive specification of  $f_1$  of  $j$ , the recursive specification  $f_2$  of  $j$  is also similar.

(Refer Slide Time: 22:38)

### A Recursive Solution (cont.)

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$
$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

25

And this basically tells us the recursive solution of interest for every station on each of the two lines. Having formulated this recursive specification of the time taken to cross station  $j$  on line 1 on the time taken to cross station  $j$  on line 2 and this we done for every  $j$  between 1 and  $n$ , we know wonder how what it is to calculate these values from these formulae.

(Refer Slide Time: 23:18)

### Computing the Optimal Solution

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$$
$$f_1[j] = \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j})$$
$$f_2[j] = \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j})$$

- Solving top-down would result in exponential running time

26

And this is the exercise of the computing the optimum solution. And one of the ways of solving the optimum solution is this solve the unit top down fashion and clearly if on

tries to compute  $f_1$  of  $n$  and  $f_2$  of  $n$  in a top down fashion at every step, we are taking the minimum of the two quantities. And it is clear that the running time of this algorithm which is to evaluate the recurrence, in the top down fashion will take an exponential time.

In other words, it will take  $2^n$  units of time, which does not seem to be a significant improvement, which is not an improvement at all over the simple algorithm that we started out with the brute force algorithm.

(Refer Slide Time: 24:11)

**Computing the Optimal Solution**

- For  $j \geq 2$ , each value  $f_i[j]$  depends only on the values of  $f_1[j - 1]$  and  $f_2[j - 1]$
- Idea: compute the values of  $f_i[j]$  as follows:
 

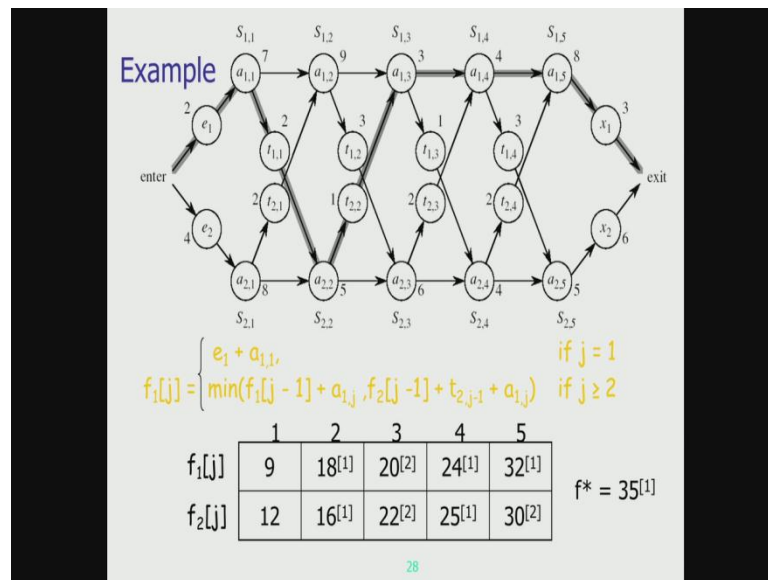
in increasing order of  $j$

	1	2	3	4	5
$f_1[j]$					
$f_2[j]$					
- Bottom-up approach
  - First find optimal solutions to subproblems
  - Find an optimal solution to the problem from the subproblems

27

On the other hand, if one uses the bottom up approach, one can indeed observe that the values of interest  $f_1$  of  $n$  and  $f_2$  of  $n$  can be very easily calculated. The main observation that we make is for  $j$  greater than or equal to 2, the value of  $f_i$  of  $j$  depends only on  $f_1$  of  $j$  minus 1 and  $f_2$  of  $j$  minus 1. In other words, the fastest way of crossing station  $j$  on either line depends only on the fastest way of crossing station  $j$  minus 1 on line 1 and the fastest way of crossing station  $j$  minus 1 on line 2. And we compute the values of  $f_i$   $j$  as describe here, in increasing order of  $j$ .

(Refer Slide Time: 25:07)



Let us look at the time taken to cross station 1 which we know, in this example we can see that the fastest way of crossing station 1 on line 1 is 9 units of time, on station 2 it is 12 units of on line 2 the fastest way of crossing station 1 takes 12 units of time. After this let us look at station 2, the fastest way of crossing station 2 on line 1 is 18 units of time and the number in parenthesis says, which station on which line was a previous station.

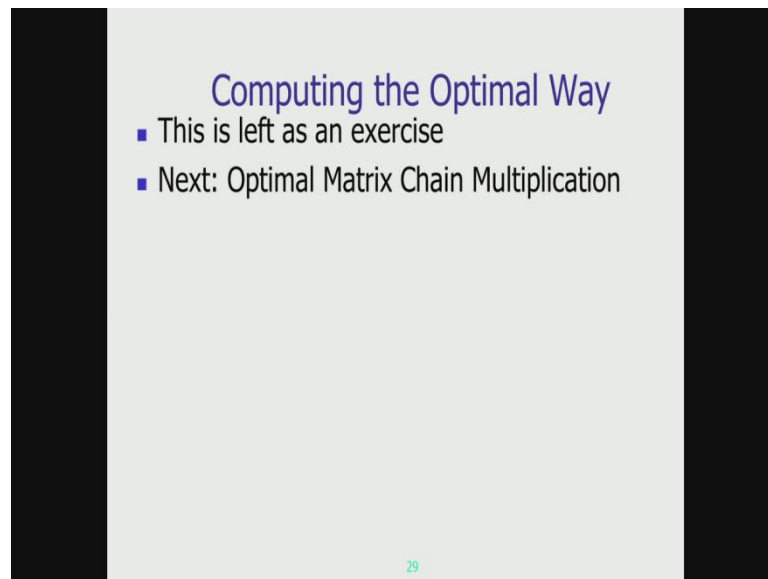
In this case, the fastest way of crossing station 2 on line 1 is via station 1 on line 1 itself, which is an instantaneously transfer to station 2 and then 9 units of processing time. Also the fastest way of crossing station 2 on line 2 is to actually through station 1 in line 1 and then transfer to station 2 on line 2 and the process at station 2 on line 2. Observe that, this is 16 units of time has suppose to 12 units of time plus 2 unit of time to transfer has suppose to 12 units of time at station 1 on line 2.

And then instantaneously be transfer 2 station 2 on line 2 and then use for 5 units of time which makes it is 23 units of time. Similarly, at station 3 on line 1 observe that the fastest way of crossing station 3 on line 1 goes through station 2 on line 2 and uses 20 units of time. As suppose to crossing station 3 on line 2 which takes 22 units of time, similarly crossing station for on line 4 takes 24 units of time viva station 3 on line 1 itself.

And on line 2 it takes 25 units of time viva station 3 on line 1 and crossing station 5 takes 32 units of time viva station 4 on line 1 and 30 units of time viva station 4 on line 2 and then the exit takes 3 and 5 units of time respectively. Therefore, the quickest way of

getting from entry to exit is 35 units of time viva station 5 on line 1.

(Refer Slide Time: 28:31)



Having gone through these hole exercise of recursive of identifying the optimal sub structure and recursively specifying the distance function. And observing there it can be computed in at a bottom of fashion, computing the optimal root that is the sequence of stations through which the shortest path pass through is left as an exercise. Then, next example of dynamic programming which we will study is the problem of optimal matrix chain multiplication which will be done then will be meet next.

Thank you.