Scribed Note

Date : 14/11/22

Roll No : 202212031

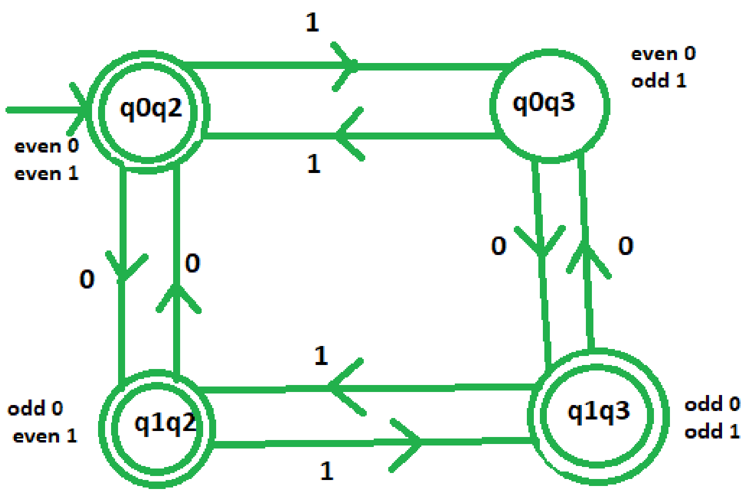
202212032

202212033

202212034

202212035

Q). DFA machines accepting odd number of 0's or/and even number of 1's



DFA

Steps :

- Let's take the input 01110011 now first q0 will start and on 0 input, it will go to state q1.
- Now we need to give input 1, then it will go to state q3.
- Now on q3 again, we give input 1. It will go to state q1 then again 1 and it will reach at q3 after reading 0111.
- Now give the input 0, it will go to state q2 then again on input 0 it will come on state q3.
- Now again when we give 1 as an input, it will go to state q2. Then finally, on getting 1 input on state q2, it will reach state q3 which is the final state.

Q). What is overlapping subproblem?

Ans: *A problem is said to have overlapping subproblems if the problem can be broken down into subproblems which are reused several times **OR** a recursive algorithm for the problem solves the same subproblem over and over rather than always generating new subproblems.*

Example :

For overlapping subproblems to occur, our function will be called with the same input value multiple times. This most often happens when we have more than one recursive call in a recursive function.

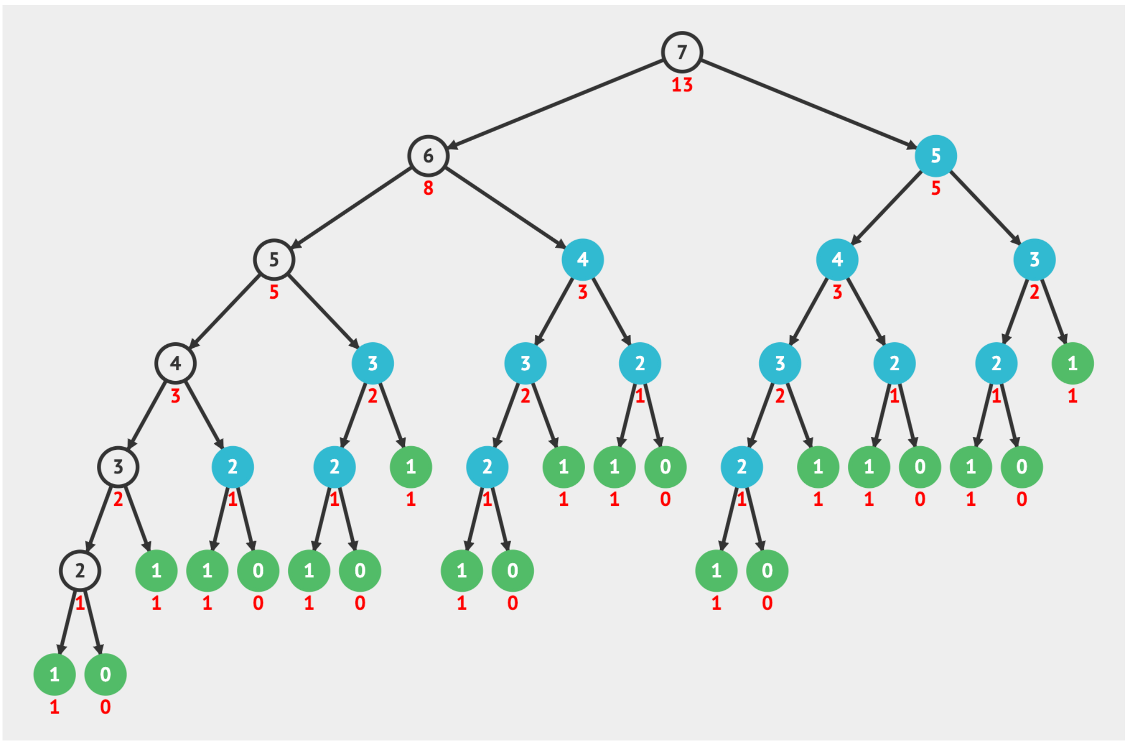
For example, the logic for finding a Fibonacci number is the following:

fib(n) = fib(n - 1) + fib(n - 2)

We have subproblems where we can calculate the Fibonacci number for any value n. So then to calculate any given Fibonacci number, we just need the results of the Fibonacci numbers before it.

We require two recursive function calls to calculate a number: fib(n - 1) and fib(n - 2). So if we wanted fib(100) = fib(99) + fib(98), both fib(99) and fib(98) would separately generate all the numbers fib(97) to fib(1) because fib(99) = fib(98) + fib(97) and fib(98) = fib(97) + fib(96). Each step generates many more function calls, and we call fib with repeated input values many times. This branches outward with each nested step continuing to do the same thing.

If we look at fib(7), we can see the repeated calculations :



The items highlighted in blue are repetitive functions calls that use the same input multiple times. These are the overlapping subproblems. When visualizing the recursive calls as a tree, the branching is a sign that we'll have subproblems that overlap.

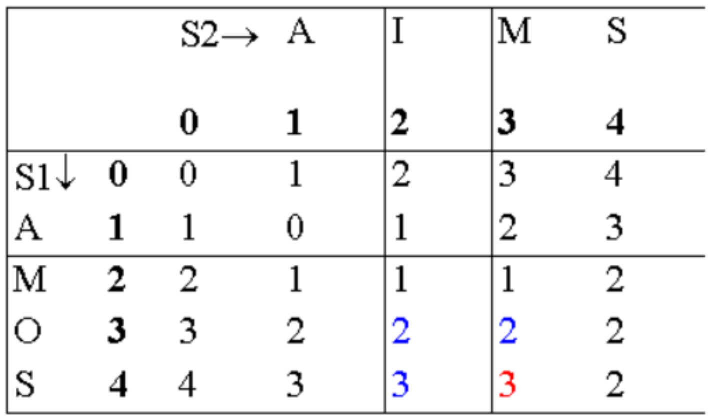
On the other hand, when calculating the factorial, we have subproblems, but we don't have overlapping subproblems. Every time we call factorial, we pass it a new value that we haven't calculated the subproblem answer for. So instead of creating a tree structure with branching, it simply forms a straight line.

Q). What is DPT(Dynamic Programming Table)?

Ans : This is one of the most helpful visualization techniques for designing bottom-up DP algorithms when the problem is a **multi-prefix/multi-suffix** or **subsequence** problem type. It is basically a way of drawing the DAG of computation when the computational structure of your problem is best explained in two dimensions. Let's motivate this with a popular example problem.

For Example :

Put value in DPT by analysing graph

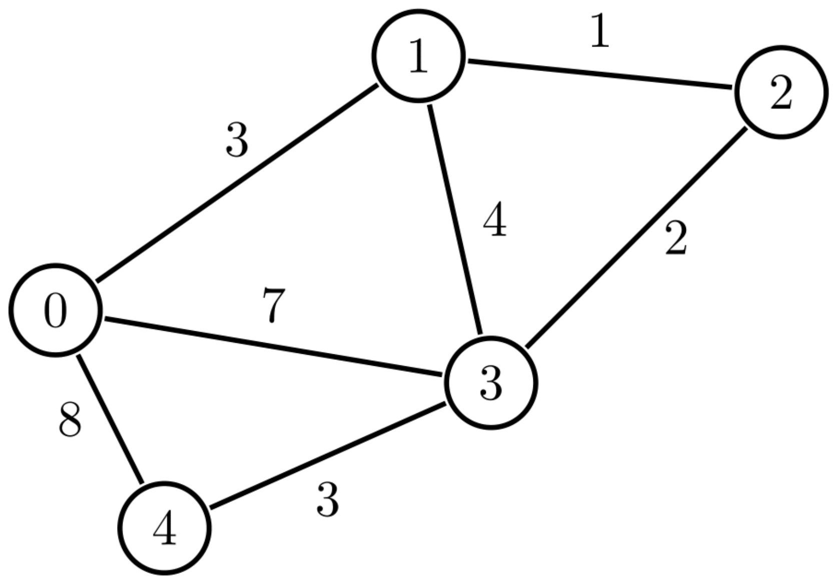


		S2→	A	I	M	S
		0	**1**	**2**	**3**	**4**
S1↓	**0**	0	1	2	3	4
A	**1**	1	0	1	2	3
M	**2**	2	1	1	1	2
O	**3**	3	2	2	2	2
S	4	4	3	3	3	2

Q). what is weighted graph?

Ans : A *graph* having a weight, or number, associated with each *edge*. Some algorithms require all weights to be nonnegative, integral, positive, etc.

Example :



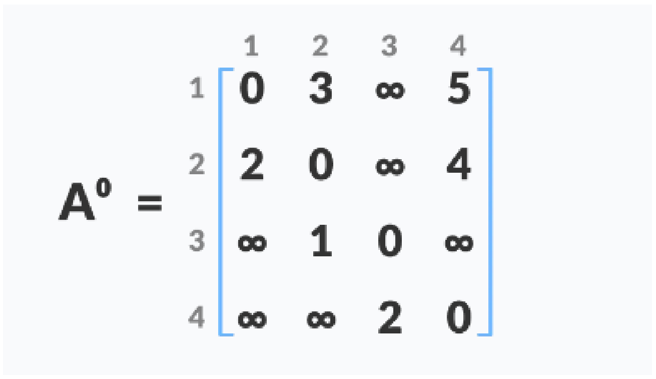
Q). Explain Floyd Warshall Algorithm

Ans : Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

How Floyd-Warshall Algorithm Works?

1. Create a matrix A0 of dimension n*n where n is the number of vertices. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph.

   Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity.



$$A^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & \infty & 4 \\ 3 & \infty & 1 & 0 & \infty \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

2. Now, create a matrix A1 using matrix A0. The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way.

   Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]).

   That is, if the direct distance from the source to the destination is greater than the path through the vertex k, then the cell is filled with A[i][k] + A[k][j].

   In this step, k is vertex 1. We calculate the distance from source vertex to destination vertex through this vertex k.

$$A^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & & \\ 3 & \infty & & 0 & \\ 4 & \infty & & & 0 \end{array} \longrightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & 9 & 4 \\ 3 & \infty & 1 & 0 & 8 \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

3. For example: For A1[2, 4], the direct distance from vertex 2 to 4 is 4 and the sum of the distance from vertex 2 to 4 through vertex (ie. from vertex 2 to 1 and from vertex 1 to 4) is 7. Since 4 < 7, A0[2, 4] is filled with 4.

   Similarly, A2 is created using A1. The elements in the second column and the second row are left as they are.

   In this step, k is the second vertex (i.e. vertex 2). The remaining steps are the same as in step 2.

$$A^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & & \\ 2 & 2 & 0 & 9 & 4 \\ 3 & & 1 & 0 & \\ 4 & & \infty & & 0 \end{array} \longrightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 9 & 5 \\ 2 & 2 & 0 & 9 & 4 \\ 3 & 3 & 1 & 0 & 5 \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

4. Similarly, A3 and A4 is also created.

$$A^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & & \infty & \\ 2 & & 0 & 9 & \\ 3 & \infty & 1 & 0 & 8 \\ 4 & & & 2 & 0 \end{array} \longrightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 9 & 5 \\ 2 & 2 & 0 & 9 & 4 \\ 3 & 3 & 1 & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array}$$

$$A^4 = \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & 0 & & & 5 \\ 2 & & 0 & & 4 \\ 3 & & & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array} \longrightarrow \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 7 & 5 \\ 2 & 2 & 0 & 6 & 4 \\ 3 & 3 & 1 & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array}$$

5. A4 gives the shortest path between each pair of vertices.

➢ Floyd Warshall Algorithm :

```
n = no of vertices
A = matrix of dimension n*n
for k = 1 to n
   for i = 1 to n
      for j = 1 to n
         Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j])
return A
```

➢ Find Shortest path :

Let d(k)ijdij(k) be the weight of a shortest path from vertex ii to jj for which all intermediate vertices are in the set {1,2,⋯,k}{1,2,⋯,k} then

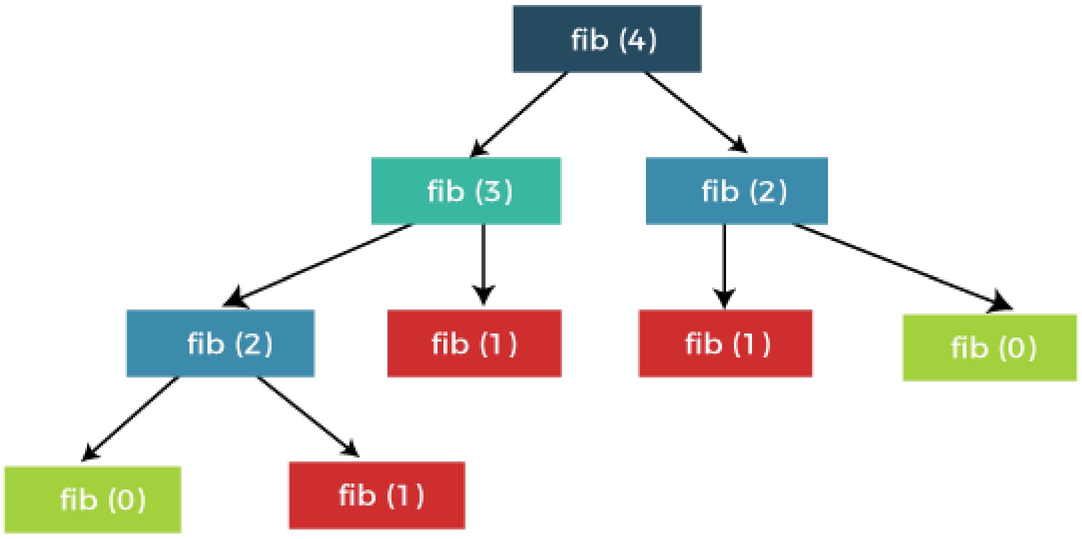
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{if } k > 0 \end{cases}$$

In fact it considers whether kk is an intermediate vertex in the shortest path from ii to jj or not. If kk is an intermediate it selects d(k−1)ik+d(k−1)kjdik(k−1)+dkj(k−1) becuase it decomposes the shortest path to i→p1k→p2ji→p1k→p2j otherwise d(k−1)ijdij(k−1) since kk is not an intermediate vertex so it has no effect on the shortest path.

Q). What is optimal structure?

Ans : A given problem is said to have Optimal Substructure Property if the optimal solution of the given problem can be obtained by using the optimal solution to its subproblems instead of trying every possible way to solve the subproblems.

Exmple :


fib (4)
fib (3)
fib (2)
fib (2)
fib (1)
fib (1)
fib (0)
fib (0)
fib (1)