DYNAMIC PROGRAMMING

Those can remember their past are condemned to repeat it

What is Dynamic Programming ?

- Dynamic programming is basically an optimization algorithm. It means that we can solve any problem without using dynamic programming but we can solve it in a better way or optimize it using dynamic programming.

- The basic idea of dynamic programming is to store the result of a problem after solving it. So when we get the need to use the solution of the problem, then we don't have to solve the problem again and just use the stored solution.

Where to use Dynamic Programming ?

1. **Optimal Substructure**

   A problem is said to have optimal substructure if an optimal solution can be constructed efficiently from optimal solution of its subproblems. Simply stated as expressing a bigger problem into smaller sub-problems.

2. **Overlapping Subproblems**

   Same sub-problems occur again and again. So instead of computing for those values again and again we can store them and use it when needed.

   Two different ways to store the value :-

   - Tabulation (Bottom-Up)
   - Memoization (Top-Down)

Note: Every problem may not have Optimal substructure property

- One should be careful not to assume that optimal substructure applies when it does not.
- **Consider the following two problems in which we are given a directed graph $G = (V, E)$ and vertices . $u, v \in V$.**
  - Unweighted shortest path:
    - Find a path from u to v consisting of the fewest edges. **Good for Dynamic programming**.
  - Unweighted longest simple path:
    - Find a simple path from u to v consisting of the most edges. **Not good for Dynamic programming.**

Memoization

Calculate the solution for a problem and all its subproblem as and when it is required. The next time they are required just use the value. Here all the subproblems are not computed, only those that are required are computed. This is the top-down approach which is used in recursive problems.

Tabulation

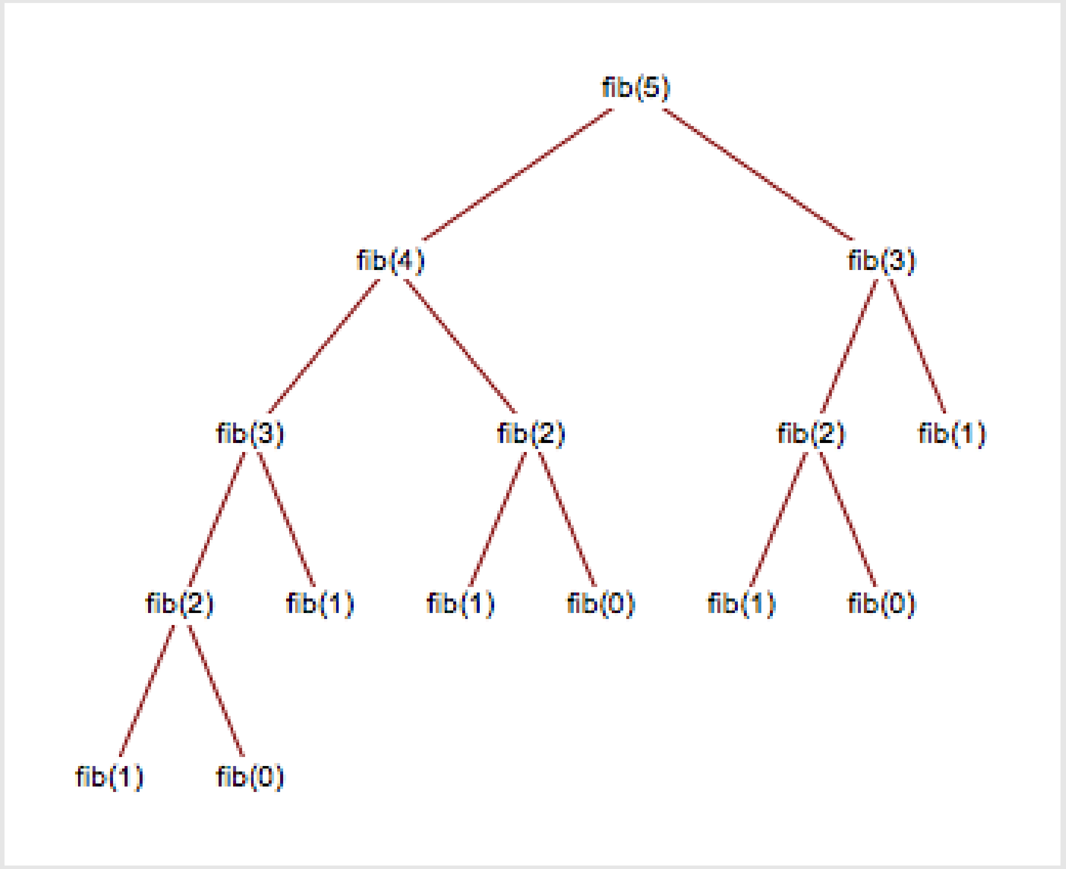
The solution of all smaller sub-problems is computed before hand and then the solution of actual problem is solved.

Lets make a clear understanding on memoization and tabulation approach by taking an example of fibonnaci problem

Fibonnaci Using Pure Recursion

```
#include<bits/stdc++.h>
using namespace std;
int fibonacci(int n)
{
   if(n==0 | | n==1)
   {
      return 1;
   }
   return fibonacci(n-1)+fibonacci(n-2);
}
int main()
{
   int n;
   cin>>n;
   cout<<fibonacci(n);
}
```

An Exponential Approach



Dynamic Programming Approach

```
//Top Down Approach
//Memoization of optimal sub problems in
dp[] array.
int fib(int n,int dp[])
{
   if(n==0 || n==1) //base case
   {
      dp[n]=n;
      return dp[n];
   }
   if(dp[n]!=0)   //check for value of dp if it
isn't zero .
   {
      return dp[n]; //return the value
   }
   dp[n]=fib(n-1,dp)+fib(n-2,dp);
   return dp[n];
}
```

```
//Bottom Up Approach
int fib(int n)
{
   int *dp = new int[n+1];
   dp[n+1] = {0};
   dp[0]=0;
   dp[1]=1;
   for(int i=2;i<=n;i++)
   {
      dp[i]=dp[i-1]+dp[i-2];
//Since,fib(n)=fib(n-1)+fib(n-2);
   }
   return dp[n];
}
```

Q. Are we using a different recurrence relation in the two codes? **No**

Q. Are we doing anything different in the two codes? **Yes**

Top Down Approach

The way we solved the Fibonacci series was the top-down approach. We just start by solving the problem in a natural manner and stored the solutions of the subproblems along the way. We also use the term **memoization**, a word derived from memo for this.

In other terms, it can also be said that we just hit the problem in a natural manner and hope that the solutions for the subproblem are already calculated and if they are not calculated, then we calculate them on the way.

Bottom Up Approach

The other way we have solved the Fibonacci problem was by starting from the bottom i.e., start by calculating the 2nd term and then 3rd and so on and finally calculating the higher terms on the top of these i.e., by using these values. We use a term **tabulation** for this process because it is like filling up a table from the start.

Matrix Chain Multiplication

The problem is not here to multiply but how to multiply so that we get the minimum cost for multiplying the elements.

Suppose there are two matrix . R1 & R2 represents the row of first & second matrix. C1 & C2 represents the row of first & second matrix.

Cost of matrix multiplication = R1*C1*C2

Note:- For Multiplication of matrices . C1==R2

So order in which we multiply the matrices effect the change in cost of matrix multiplication.

No of ways the matrix can be multiplied is given by Catalan number.

We can solve this problem using dynamic programming approach.

For Matrix Multiplication we will use Bottom-Up Approach i.e. Tabulation Step 1 : -

A1	A2	A3	A4
5 * 4	4 * 6	6 * 2	2 * 7

	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

- Diagonal is zero because the same matrix isn't multiplied with anything .

Two matrix are multiplied :-

- Cost[1][2] = 5*4*6=120
- Cost[2][3] = 4*6*2=48
- Cost[3][4] = 6*2*7=84

For Matrix Multiplication we will use Bottom-Up Approach i.e. Tabulation Step 1 : -

A1	A2	A3	A4
5 * 4	4 * 6	6 * 2	2 * 7

	1	2	3	4
1	0	120	88	
2		0	48	
3			0	84
4				0

Three matrix are multiplied :-

- Cost[1][3]

The matrix can be multiplied in two ways :-

a) A1*(A2*A3)

b) (A1*A2)*A3

So,

Cost[1][3] = Cost[1][1]+Cost[2][3]+5*4*2

= 0 + 48 + 40 = 88

Cost[1][3] = Cost[1][2]+Cost[3][3]+5*6*2

= 120 + 0 + 60 =180

For Matrix Multiplication we will use Bottom-Up Approach i.e. Tabulation Step 1 : -

	A1	A2	A3	A4
	5 * 4	4 * 6	6 * 2	2 * 7

	1	2	3	4
1	0	120	88	
2		0	48	104
3			0	84
4				0

Three matrix are multiplied :-

- Cost[2][4]

The matrix can be multiplied in two ways :-

a) A2*(A3*A4)

b) (A2*A3)*A4

So,

Cost[2][4] = Cost[2][2]+Cost[3][4]+4*6*7

= 0 + 84 + 40 = 252

Cost[2][4] = Cost[2][3]+Cost[4][4]+4*2*7

= 48 + 0 + 56 =104

For Matrix Multiplication we will use Bottom-Up Approach i.e. Tabulation

Step 1 : -

A1	A2	A3	A4
5 * 4	4 * 6	6 * 2	2 * 7

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

Now Let us see the Formula and calculate

Cost[1][4]=min{Cost[1][1]+Cost[2][4]+5*4*7,
Cost[1][2]+Cost[3][4]+5*6*7,
Cost[1][3]+Cost[4][4]+5*2*7,}
= 158

In General

Cost[i][j]={ Cost[i][k]+Cost[k+1][j]+d1*d2*d3

Where d1= D(i-1) , d2 = dk , d3=dj

Time Complexity O(n^3)