DYNAMIC PROGRAMMING

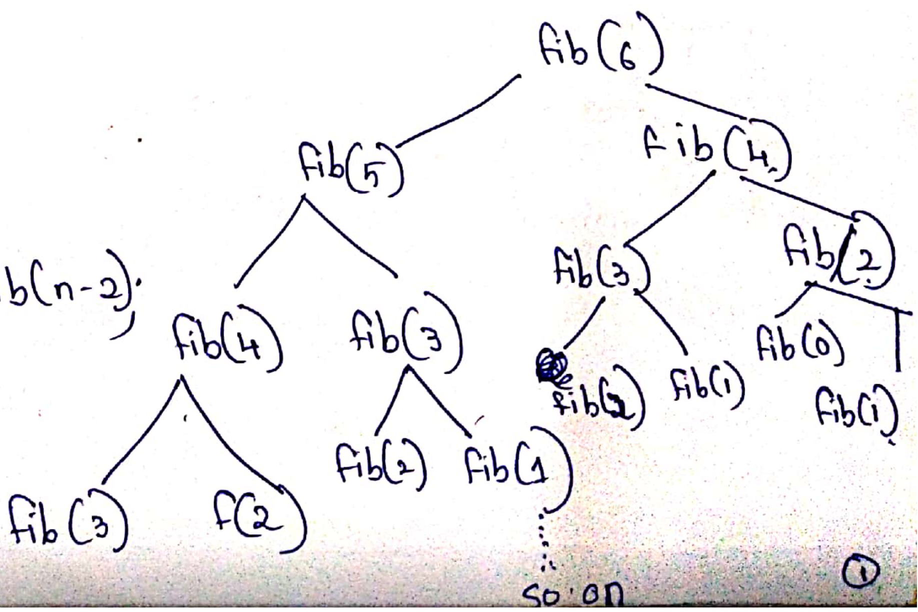
- Dynamic programming is a technique for solving problems with overlapping subproblems.
- Typically, these subproblems arise from a recurrence relating a solution to a given problem with solution to its smaller subproblems of the same type.
- Dynamic programming suggests solving each smaller subproblem once and recording the results in a table from which a solution to the original problem can be obtained.
- Dynamic programming = Recursion + Common Sense.
  = memoization [remembering results of a few states].

Eg: the Fibonacci Series.

pure recursion.

```
if fib(int n)
{
  if(n<2)
    return 1;
  return fib(n-1)+fib(n-2);
}
```

trace : fib(6).



fib(i) is calculated multiple times in recursion method.

Time complexity : $O(2^n)$.

dynamic programming (adding memory to recursion method).

```
void fib().
{
    fibresult[0] = 1;
    fibresult[1] = 1;
    for (int i=2; i<n; i++).
        fibresult[i] = fibresult[i-1] + fibresult[i-2];
}.
```

Time complexity = $O(n)$, space complexity = $O(n)$.

- Two major properties of Dynamic programming :

→ to decide whether a problem can be solved by applying dynamic programming we have to check two. properties.

(a) overlapping Sub-problems

(b) Optimal Substructure.

(a) Overlapping sub-problems.

→ suggests that the subproblem needs to be solved again and again.

→ In recursion we solve those problems every time. Eg: fibonacci with pure recursion method.

→ In dynamic programming we solve these subproblems only once and store it for future use.

Eg: Fibonacci with dynamic programming.

(b) Optimal Substructure.

→ If a problem can be solved using the solutions of the sub problems then we say that problem has a Optimal Substructure Property.

- Dynamic Programming Approaches:

(a) Bottom-up approach. [Tabular].

→ suppose we need to solve the problem for N, we start solving the problem with the smallest possible inputs and store it for future.

→ Now as you calculate for the bigger values use the stored solutions (solution for smaller problems).

(b) Top-down approach. [memoization].

→ Break the problem into sub-problem and solve them as needed and store the solution for future.

Problems on DP (Dynamic programming) Approach.

1) 0/1 Knapsack.
2) Warshall's Algorithm. [all possible paths]
3) Floyd Warshall Algorithm [all pairs shortest distances]
4) Longest common subsequence.

NOTE: Examples for all the above have been done in class. Refer class notes.

① 0/1 KNAPSACK.

I Algorithm for 0/1 knapsack.

$$v[i, wt] = \max\{V[i-1, wt], V[i-1, w-wt[i]] + P[i]\}$$

II. Time complexity = $O(2^n)$.

② WARSHALL'S ALGORITHM.

I Algorithm for Warshall's

Input : Adjacency matrix A of a directed graph with 'n' vertices.

Output : The transitive closure of the directed graph

$R^{(0)} \leftarrow A$.

for $k \leftarrow 1$ to n do.

  for $i \leftarrow 1$ to n do.

    for $j \leftarrow 1$ to n do.

      $R^{(k)}[i,j] \leftarrow R^{(k-1)}[i,j]$ or

        $R^{(k-1)}[i,k]$ and $R^{(k-1)}[k,j])$.

Return $R^{(n)}$.

II Time complexity.

Time complexity = $\Theta(n^3)$.

③ FLOYD WARSHALL ALGORITHM.

I Algorithm.

Input : The weight matrix W of a graph with no negative length cycle. cycle with weights that sum to negative number.

Output : The distance matrix of the shortest path's length.

$D \leftarrow W$

for $k \leftarrow 1$ to n do.

for $i \leftarrow 1$ to n do.

for $j \leftarrow 1$ to n do.

$D[i,j] \leftarrow \min\{D[i,j], D[i,k] + D[k,j]\}$

return D.

II Time complexity.

Time complexity $= \Theta(n^3)$.

④ LONGEST COMMON SUBSEQUENCE.

I Algorithm.

if $(A[i] = B[j])$.

$LCS[i,j] = 1 + LCS[i-1, j-1]$

else.

$LCS[i,j] = \max\{LCS[i-1,j], LCS[i,j-1]\}$

II Time complexity $= O(mn)$

Space complexity $= O(mn)$.

0/1 knapsack problems.

1)

wt	pt.
2	1
3	2
3	5
4	9
6	4

capacity = 10.

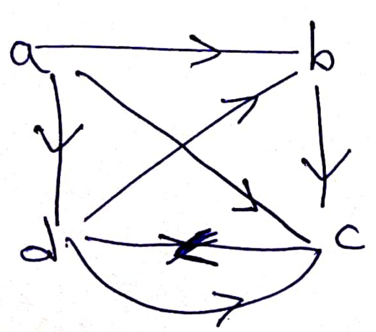
2).

wt	pt.
3	100
2	20
4	60
1	40.

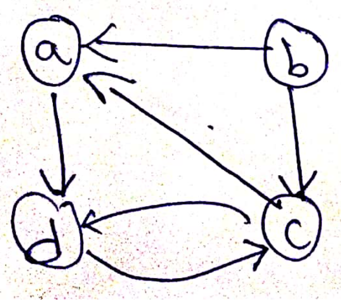
capacity = 5.

Warshall's Algorithm problems

1)

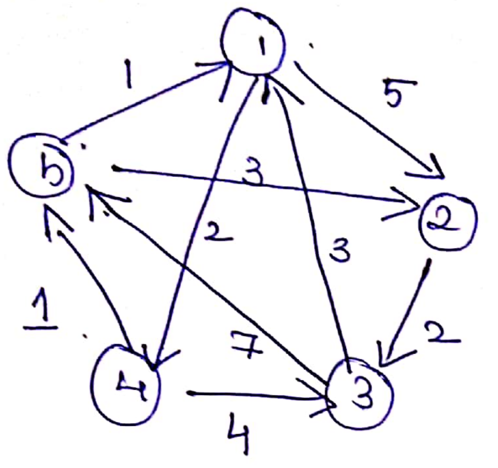


2)

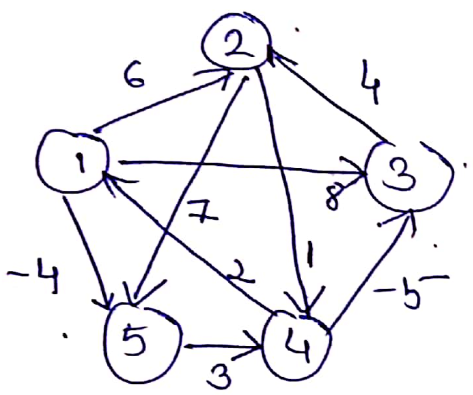


Floyd Warshall algorithm Problems

1)



2)



Longest common subsequence Problems

1) A - stone
   B - longest

2) A - lesson
   B - learnt.

3) A - Engineer.
   B - Edition.