Dynamic Programing

DP = Brute Force + Memory

Important conditions:

1. Overlapping Subproblem
2. Optimal Substructure

If the above two properties hold then can be solved using:

1. DP
2. Memoization

Fibonacci:

1. Recursion: O(`2^n`)
2. DP- `O(n)`

The above problem can be solved by

Memoization

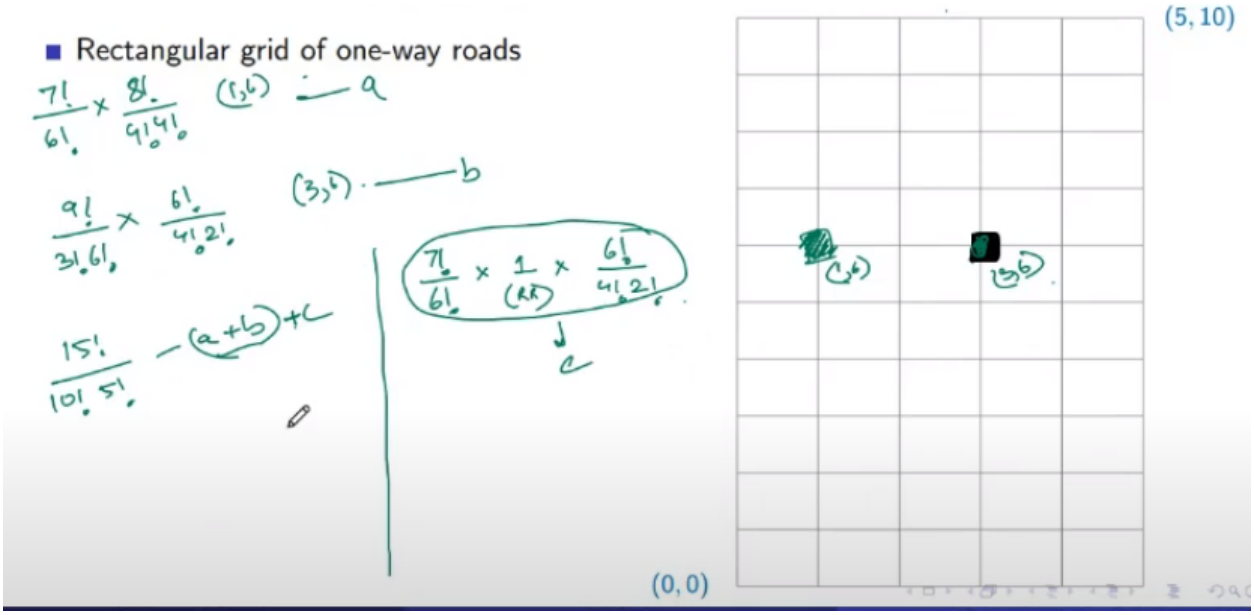
- Recursion
- Top Down

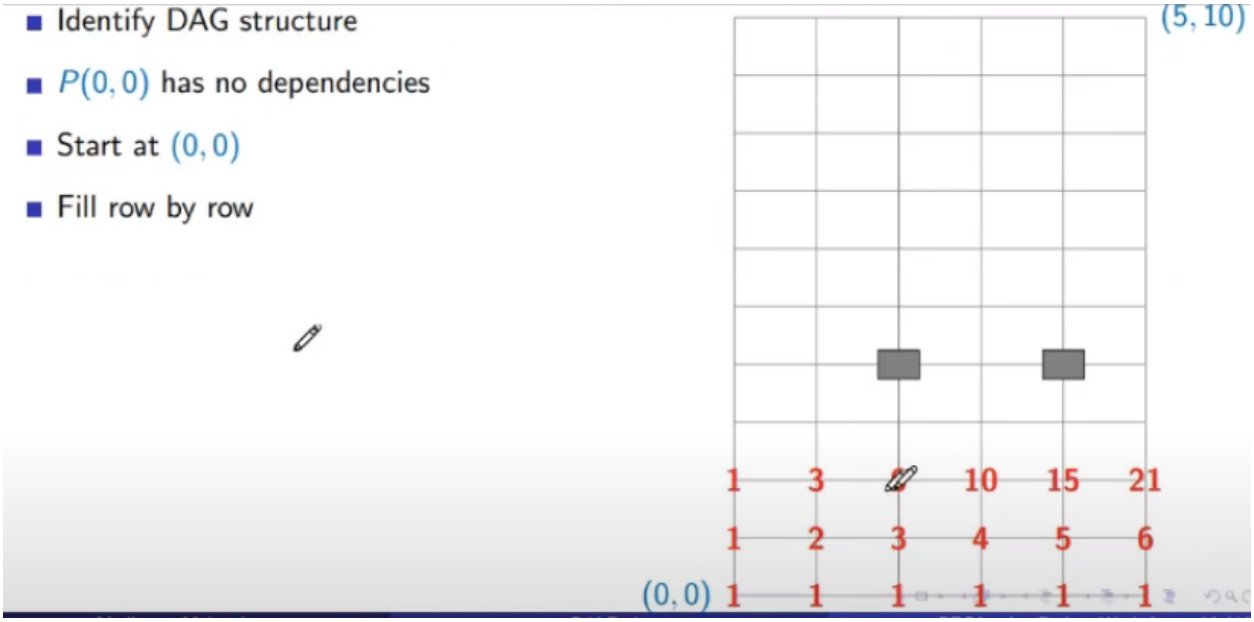
DP

- Iterations
- Bottom Up approach

GRID PATHS

Question: Number of ways to go from (0,0) to (5,10)





STEPS:

1. Find the recursive relation
2. Find the base case relation
3. Use the base case relation to understand the loop initialization
4. use the recursive relation in the program body to evaluate the values

Longest common substring:

```
def LCW(u,v):
  import numpy as np
  (m,n) = (len(u),len(v))
  lcw = np.zeros((m+1,n+1))

  maxlcw = 0

  for c in range(n-1,-1,-1):
    for r in range(m-1,-1,-1):
      if u[r] == v[c]:
        lcw[r,c] = 1 + lcw[r+1,c+1]
      else:
        lcw[r,c] = 0
      if lcw[r,c] > maxlcw:
        maxlcw = lcw[r,c]

  return(maxlcw)
```

Complexity

- Recall that brute force was $O(mn^2)$
- Inductive solution is $O(mn)$, using dynamic programming or memoization

Longest common subsequence:

```
def LCS(u,v):
  import numpy as np
  (m,n) = (len(u),len(v))
  lcs = np.zeros((m+1,n+1))

  for c in range(n-1,-1,-1):
    for r in range(m-1,-1,-1):
      if u[r] == v[c]:
        lcs[r,c] = 1 + lcs[r+1,c+1]
      else:
        lcs[r,c] = max(lcs[r+1,c],
                       lcs[r,c+1])
  return(lcs[0,0])
```

Complexity

- Again $O(mn)$, using dynamic programming or memoization

EDIT OPERATION

- Edit operation is essentially to transform document
  - Insert a character
  - Delete a character
  - Substitute one character by another
- convert `aba` into `acaba` using minimum number of operations:
  - Three operations
    - a
    - a `c`
    - a `c` a `b a`
  - Two Operations:
    - `ac` aba