Hello !!!

→ Trie
→ Word ladder
→ DP
  → Theory
→ Ex 1
→ Ex 2

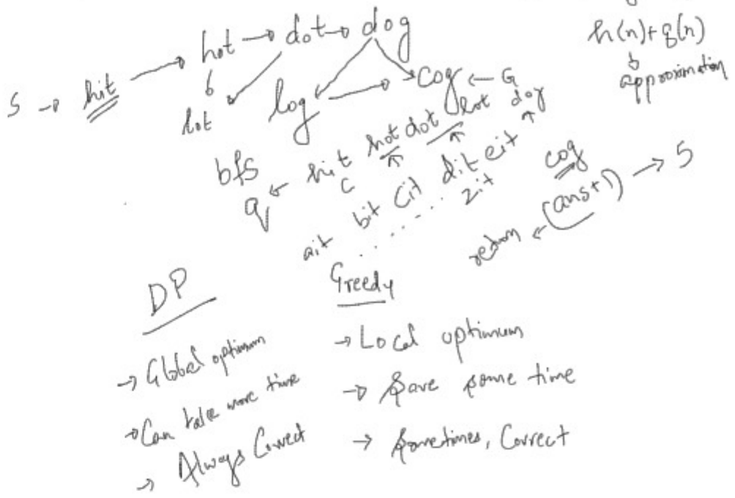
Trie

(insert, search, startsWith) → O(K)

Space: insert → O(K)
search, startsWith → O(1)
K → length of word

Word Ladder

A* → Greedy Algo
h(n) + g(n)
↓ approximation



DP

→ Global optimum
→ Can take more time
→ Always Correct

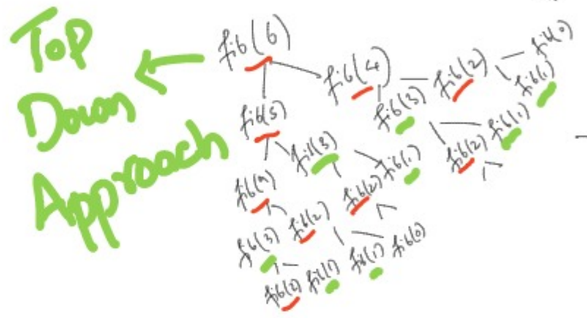
Greedy

→ Local optimum
→ Save some time
→ Sometimes, Correct

Fibonacci sequence → 0 1 1 2 3 5 8 13 ....

$$fib(n) = \begin{cases} n & n<2 \\ fib(n-1)+fib(n-2) & \end{cases}$$ → Recurrence Relation

Top Down Approach



→ TC

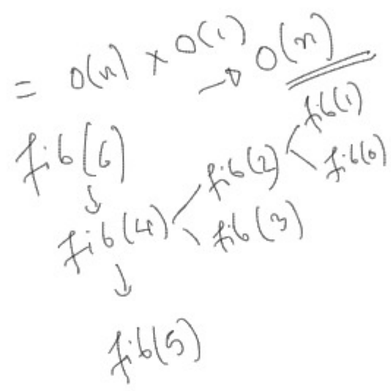
2^n

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

$T(n) = T(n-1) + T(n-2) + C$

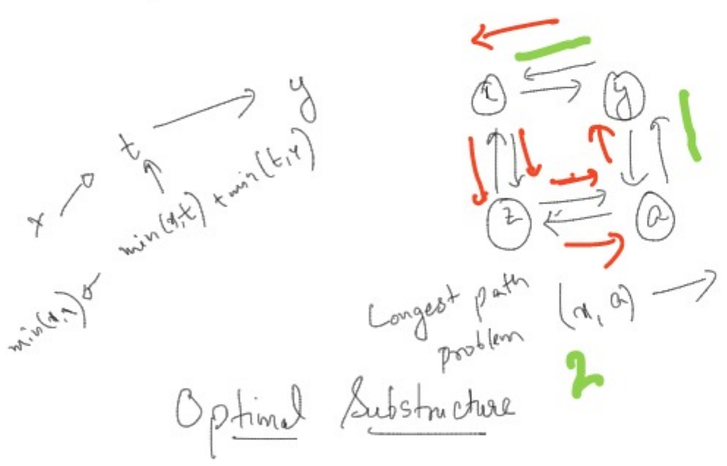
```
int arr[n] = {-1};
fib(n)
{
  if(n<2)
    arr[n] = n;
  else
    if(arr[n-2] == -1)
      fib(n-2)
    if(arr[n-1] == -1)
      fib(n-1)
    arr[n] = arr[n-1] + arr[n-2]
}
```

→ TC = O(n) × O(1) → O(n)



↳ Memoisation

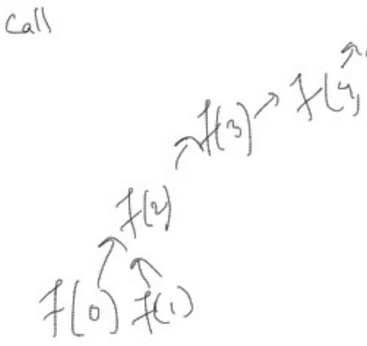
→ $fib(n) = fib(n-1) + fib(n-2)$



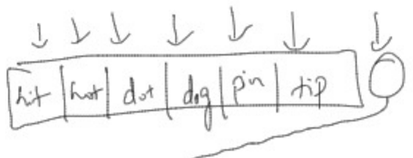
Optimal Substructure

TC (DP) = Distinct function calls * Time to execute a function call

→ Bottom UP
→ Tabulation




```
fib(n)
{
  arr[0] = 0, arr[1] = 1;
  for(i=2; i<=n; i++)
    arr[i] = arr[i-1] + arr[i-2];
  return arr[n];
}
```




```
if(i=0; i<n; i++)
  if(d[i] == "tip")
    return i;
```

```
find(vec.begin(), vec.end(), tip)
  == vec.end()
vec.erase(i);
```

vector<string>::iterator