MIT Dynamic Programming (DP)

* DP ≈ "careful brute force." DP ≈ subproblems + "reuse"

Fibonacci numbers

$$\begin{cases} F_1 = F_2 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

Goal : compute F_n

Naive Recursive algorithm.

```
fib():
    if n ≤ 2 : f = 1
    else f = fib(n-1) + fib(n-2)
    return f
```

Exponential Time

$T(n) = T(n-1) + T(n-2) + \Theta(1)$

Memoized Dynamic Programming algorithm

```
memo = { }
fib(n):
    if n in memo : return memo[n]
    if n ≤ 2 : f = 1
    else: f = fib(n-1) + fib(n-2)
    memo[n] = f
    return f
```

fib(k) only recurses the first time it's called, $\forall k$

→ memoized calls cost $\Theta(1)$

→ # non memoized calls is n :

fib(1), fib(2) ..... fib(n)

– non recursive work per call = $\Theta(1)$

⇒ time = $\Theta(n)$

In General DP ≈ recursion + memoization.

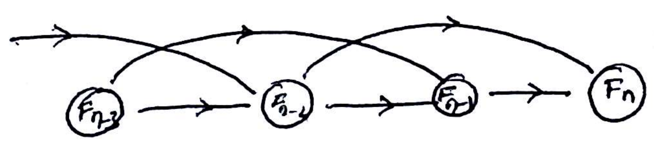
→ memoize (remember) & re-use solutions to subproblems that help solve the problem

⇒ time # subproblem · $\boxed{\text{time /subproblems} \ \ O(1)}$
n

Bottom-up DP algorithm.

```
fib = { }
for k in range(n)
    [ if k≤2  f=1
    [ else f= fib[k-1] + fib[k-2]
    fib[k] = f
return fib[n]
```

- exact same computation as memoization
- topological sort of subproblem dependency DAG.



- can often save space.

Shortest path algorithm

Bellman-Ford algorithm.

```
Bellman-Ford (G, w, s)
    Initialize-single-source (G, s)
    for i ← 1 to |V[G]| - 1
        do for each edge (u,v) ∈ E[G]
            do Relax (u, v, w)
    for each edge (u,v) ∈ E[G]
        do if d[v] > d[u] + w(u,v)
            then return False
    return True
```

```
Relax (u, v, w)
    if d[v] > d[u] + w(u,v)
        then d[v] ← d[u] + w(u,v)
            π[v] ← u
```

6.23. If $i > 0$ then

$$OPT(i, v) = \min\left(OPT(i-1, v), \min_{w \in V}\left(OPT(i-1, w) + C_{vw}\right)\right).$$

```
Shortest-Path (G, s, t)
    n = number of nodes in G
    Array M[0 .... n-1, V]
    Define M[0,t] = 0 and M[0,v] = ∞ for all v ∈ V
    For i = 1 ..... n-1.
        For v ∈ V in any order
            compute M[i,v] using 6.23
        Endfor
    End for
    Return M[n-1, s]
```

Scanned by CamScanner

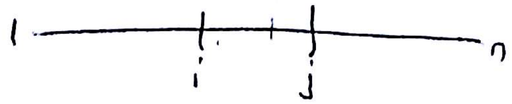
RNA Secondary structure: DP over intervals

value of

$OPT(i,j) \Rightarrow$ Secondary structure that maximizes number of pairs when we restrict attention to the subsequence between i and j

want $OPT(1,n)$ $\qquad$ $i \geq 1$ $\; j \leq n$

$i \leq j$



Recurrence

Base case $\quad OPT(i,j) = 0 \quad$ if $i \geq j-4$.

$$OPT(i,j) = \max\Big[OPT(i, j-1),\ \max\big(1 + OPT(i, t-1) + OPT(t+1, j-1)\big)\Big]$$

if char in pos t and j can pair

Weighted Interval Scheduling

$$OPT(j) = \max\big(v_j + OPT(p(j)),\ OPT(j-1)\big)$$

Segmented Least squares

$$OPT(j) = \min_{1 \leq i \leq j}\big(e_{i,j} + C + OPT(i-1)\big)$$

Knapsack

If $w < w_i$ then $OPT(i,w) = OPT(i-1, w)$ otherwise

$$OPT(i,w) = \max\big(OPT(i-1, w),\ v_i + OPT(i-1, w-w_i)\big)$$

Scanned by CamScanner