Train V

Oct 12. Ye Cao

To wrap up what we covered so far:

1. C++
   - pointer
   - Iterator
   - Sort & Binary search
2. Tree
   - Binary search tree, Heap
   - Binary Indexed tree, Range Query
3. Cumulative List
   - Sqrt decomposition
   - Fenwick Tree
   - Segment Tree

Dynamic Programming (DP)

ways to think of dp:

1. Optimization Problem
2. Exhaustive Search done in a "CLEVER" Way

1: Shortest path problem e.g.
2: When there's a polynomial time algorithm, DP is the general approach.

Specifically Session V will bring a few easy & intuitive examples about DP.

Dynamic Programming I.

- Memorization & Subproblems
- Fibonacci
- Shortest path
- Guessing & DAG (Directed Acyclic Graph) View

A little peek of what DP is:

DP = subproblem + "reuse"

Fibonacci numbers:

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

Compute F_n $(1 \le n \in \mathbb{N})$

Naive recursive algorithm

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

Exponential Time

proof: $T(n) = T(n-1) + T(n-2) + O(1)$

$T(n) \ge 2T(n-2)$

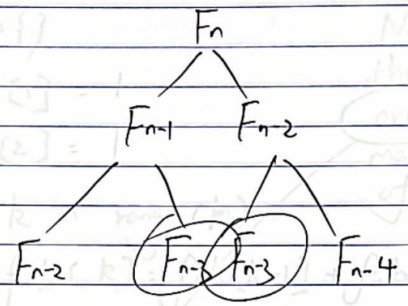
$= \Theta(2^{n/2})$

To make this algorithm work

Memorize DP algorithm

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

Why is the second algorithm more efficient?



fib(k) only recureses the first time it's called

- memorized calls cost O(1).
- non-memorized calls is O(n)
  - fib(1), ... fn)
  - O(N)

Not the best/efficient algorithm, best one O(log(N))

DP ≒ recursion + memorization.

- Memerize (remember)
  - reuse solutions (subproblem)
- Time = # subproblems · time/subproblem

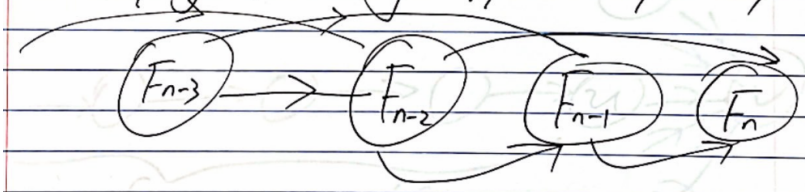
Don't count memorized recursions O(1) (this case).

Bottom-up DP Algorithm:

```
fib = {}
fib[1] = 1
fib[2] = 1
for k in range(n):
    fib[k] = fib[k-1] + fib[k-2]
return fib[n]
```

Note that I'm computing these values in a certain order (ascending) to make sure the "prereqs" of a value is already calulat

- Exactly same computation
- Topological sort of subproblem dependency DAG



Benefits of bottom-up: Save space

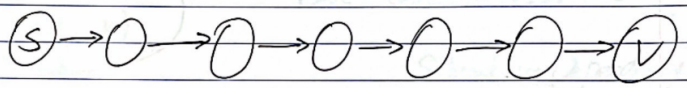
Less function calls.

Shortest paths : $\delta(s, v)$ $\forall v$. Single Source Shortest Path

Guessing : don't know the answers ? guess

Try (ALL) Guess

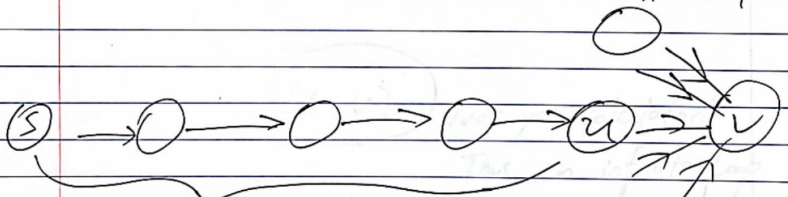
DP = recursion + memorization + (Guessing)



Guess Last edge approach

$\delta(s, v)$ $\forall v$

Single Source Shortest Path.

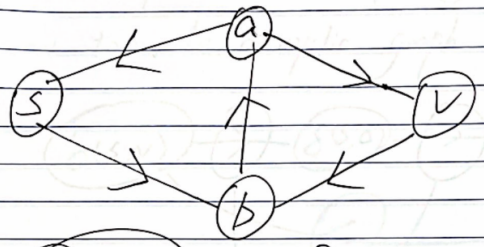


$\delta(s,u) = \min(\delta(s,u) + w(u,v))$

$\delta(s,v) = \min(\delta(s,u) + w(u,v) \quad \forall (u,v) \in E)$.

Memorization δ table

Is there any problem with this algorithm?



Base case $\delta(s,s) = 0$

$\delta(s, v)$

$\delta(s, a) + w(a, v)$

$\delta(s, b)$

$\delta(s,s)$ $\quad$ $\delta(s,v)$ Not yet calculated

Thus an infinite loop on graph with cycles

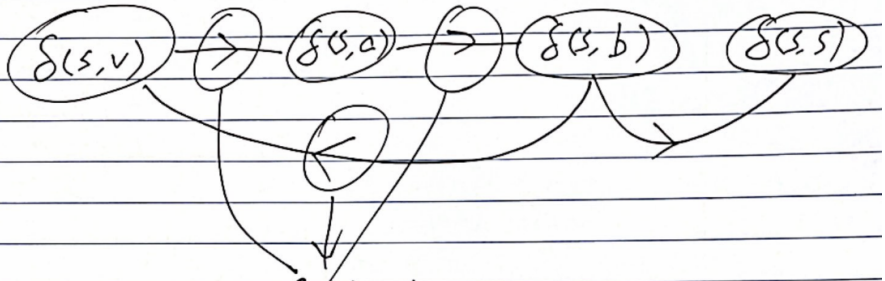
Subproblem Dependency should be acyclic otherwise it's an infinite algorithm.

DAG : $O(V+E)$ on directed acyclic graph

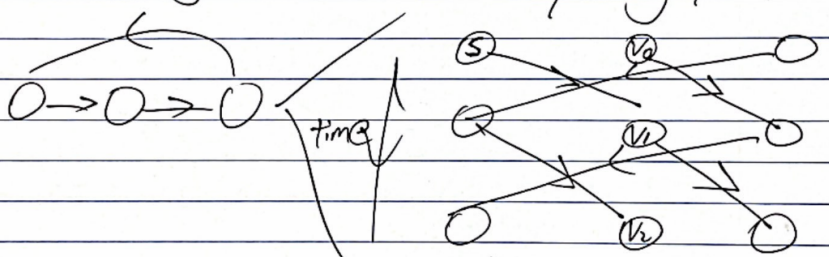
However, on cyclic graph, it's infinite Loop!

How to fix to run cyclic graphs ?

how to make a cyclic graph into an acyclic one?



Cyclic graph → Cycle here !!! → acyclic graph.



explode graph to make it acyclic.

Change of our memorization table :
add another time dimension

$\delta_k(s, v)$ vs. $\delta(s, v)$

↓

weight of shortest path s to v using $\leq k$ edges.

$$\delta_k(s, v) = \min_{(u,v)\in E} \left(\delta_{k-1}(s, u) + w(u, v) \right)$$

This time total running time O(VE)

- This is Bell-Man Ford Algorithm !!!