CS 473
Sep 4, 2018

Sep 4-1:56 PM

Dynamic Programming

Fibonacci numbers
$F_0 = F_1 = 1$
$F_{i+1} = F_i + F_{i-1}$
F_n

Sep 4-1:58 PM

```
int fib(n){
  if (n=0) return 1
  if (n=1) return 1
  return fib(n-1)+fib(n-2)
```

fib(n) $F_n \approx 2^{\Theta(n)}$

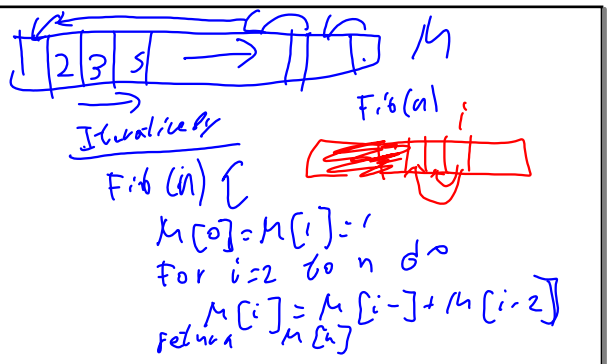
Sep 4-2:04 PM

Remember things you already computed
"CACHE"
MEMOIZATION

Sep 4-2:06 PM

```
M[0...n] = -1
Fib(i):   or (i=1)   Explicit memorization
  if (i=0) return 1   Implicit M
  if M[i]>0 return M[i]
  r ← Fib(i-1)+Fib(i-2)
  M[i] ← r
  return r
```

Sep 4-2:12 PM



Sep 4-2:15 PM

```
Fib(n):
  a = b = 1
  For i = 2 to n do
     c ← a + b
     a ← b
     b ← c
  return c
```

Sep 4-2:19 PM

$2^{O(n)}$ recursive alg

$O(n)$ time

BIR numbers

n

$O(n)$ time $O(1)$ space

$O(n^2)$

Sep 4-2:21 PM

Partition number

n how many ways to write it as a sum of numbers

2 = 2, 1+1

3 = 3, 2+1, 1+1+1

4 = 4, 3+1, 2+2, 2+1+1, 1+1+1+1

Sep 4-2:29 PM

5 = 5, 4+1, 3+2, 3+1+1, 2+2+1, 2+1+1+1, 1+1+1+1+1

$p(5) = 7$

$2^{n/2} \leq p(n) \leq 2^{O(n \log n)}$

Sep 4-2:32 PM

$p(n,d)$ ≡ number of way to write n as a sum of number s.t. d is the first number in the summation

$p(n) = \sum_{i=1}^{n} p(n,i)$

Sep 4-2:34 PM

$p(n,d)$: n^2

code

```
if (n==1) return 1
if (d>n) return 0
n' = n - d; s ← 0
For i = 1 to d do
    s ← p(n-d, i) + s
return s.
```

$O(n^2)$ $\times O(1)$

no sub. others

$p(100, n)$

$p(100, n)$

$O(n)$

Sep 4-2:37 PM

$f(p_1, p_2 \ldots)$ | recursive function

- # of distinct calls (small)

$p(i, d)$ n n n^2

$i \ldots i \Rightarrow 1 \ldots n$

$1 \ldots n$ $p(n)$

Sep 4-2:42 PM

Filling a table/array | Memoization

$f(p_1, p_2, p_3)$; $O(1)$

Boundary checks / Base cases

If cache$(p_1, \ldots, p_n)$ has value return from cache

→ COMPUTATION ⇐ change

store computed value in cache

return it.

Sep 4-2:46 PM

Running time | Amortized analysis

[# distinct recursive calls] V

* [amount of work to compute a single entry] V

Sep 4-2:52 PM

For partition

$p(n) \equiv O(n)$

$p(i, d)$: $\begin{bmatrix} O(n) & i \\ O(n) & d \end{bmatrix} = O(n^2)$ distinct recursive calls

work inside recursive call $O(n)$

RT: $O(n \cdot n^2) = O(n^3)$

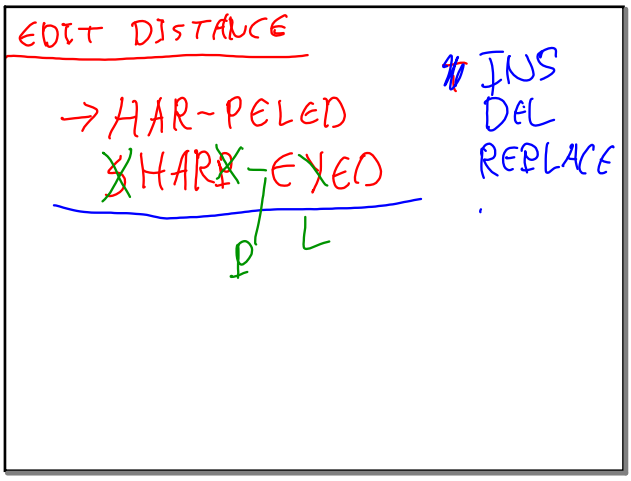
Sep 4-2:56 PM

Subproblems are defined compactly

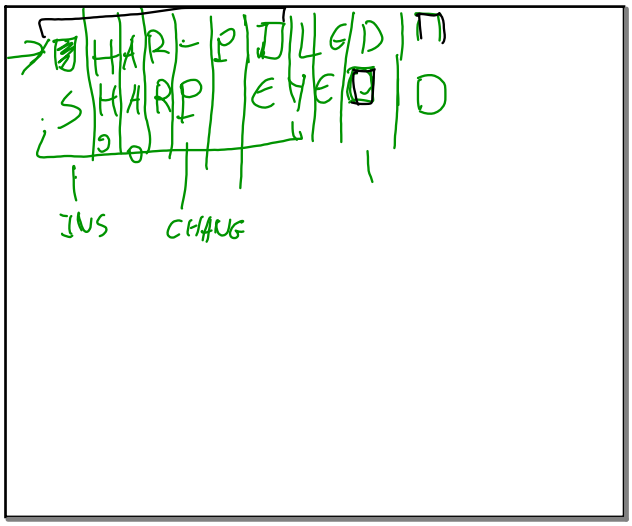
Optimal solution has some structure

Sep 4-2:59 PM

Sep 4-3:04 PM



Sep 4-3:07 PM



Sep 4-3:14 PM