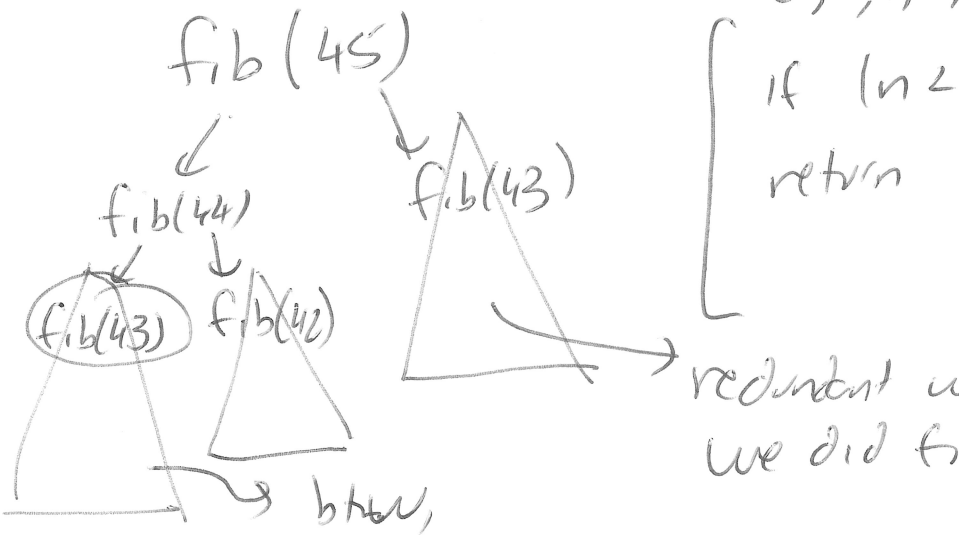
COP 3503 - 4/5/22 DP Lec 1

Dynamic Programming - rewriting a recursive solution that makes redundant recursive calls in such a way that we avoid making those calls by storing answers to previous calls in memory. (Problem must have optimal substructure)

0, 1, 1, 2, 3, 5, 8, ...

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



Idea:

(1) Create array to store all answers

	0	1	2	3	4	5	
fibArr	0	1	1	2	3		

etc.

(2) fill array -1

(3) before recursion see if I have ans stored (-1 = not stored >=0 stored)
if yes, return ans

(4) if not, do recursion, but store answer before we return.

This idea is called memoization!
(Some people call this DP.)

pitch up orig recursive solution to add memory & so we don't make redundant rec. calls!

Dynamic Programming

(1) Create mem to store all ans to possible recursive calls

(2) Seed table with small ans (base cases)

(3) fill table in order so that whenever you need an ans to rec call, it's already done.

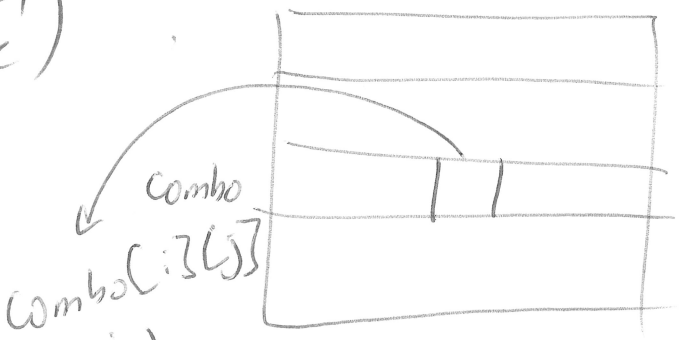
Example #2

Combinations

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

combo(int n, int k)



$= \binom{i}{j}$

2 Problems Left

Longest Common Subsequence

ways making change

LCS

Subsequence - subset of items from a list in the order they appear.

LCS(GCAATA, ACGACTA)

GAA is a common subsequence.

Q: What is the length of the longest common subsequence

greedy match!

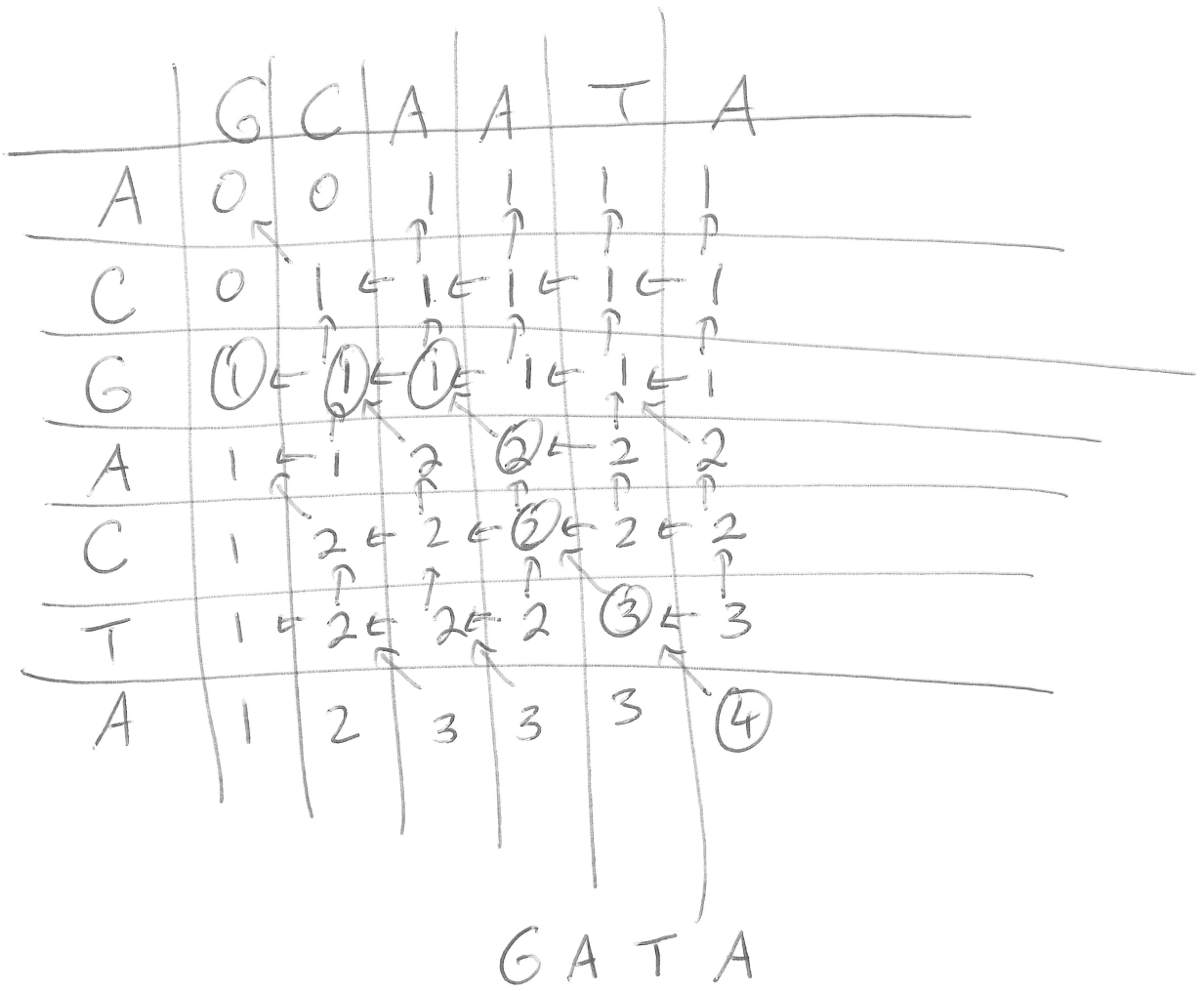
LCS(GCAATA, ACGACTA) =
1 + LCS(GCAAT, ACGACT)

if last let match

LCS(GCAA, ACGAC) = → at least 1 will not be included

max(LCS(GCAA, ACGA),
LCS(GCA, ACGAC))

lcs(x, y) = length of the lcs of
the 1st x-1 letters of 1st string
y-1 letters of 2nd string



Change Problem

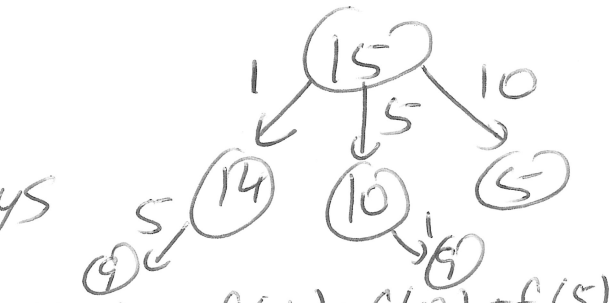
Given coin denominations 1, 5, 10, 25

cents to make change = 15

how many ways can I make change for 15 cents

10, 5
10, 1, 1, 1, 1, 1
5, 5, 5
5, 5, 1, 1, 1, 1, 1
5, 1×10
1×15

6 ways



$f(15) = f(14) + f(10) + f(5)$

# of different orders of providing change

Characterizing the problem like this doesn't work!

$f(n,d)$ = # ways to make change for n cents using denomination d or less.

$f(n,0) = 1$, all pennies!

$f(n,1)$ = # ways using pennies or nickels

etc.

	0	1	2	3
denom =	1	3	4	13

$f(n,d)$ $f(n - coin[0], 0) +$
$f(n - coin[1], 1) +$
$f(n - coin[2], 2) + \cdots$
$f(n - coin[d], d)$

$$\sum_{i=0}^{d} f(n - c_i, i)$$

OR

$$f(n,d) = f(n - coin[d], d) + f(n, d-1)$$

↓ All w/o using denom d

↑ all ways to use denom d.