Algorithms in Bio.

Dynamic Programming

Recap

- HW due
- Next HW: posted shortly
  covering greed
- I'm gone next Thursday
  no class

Recall: Recursion:

High level idea:

- Find a small choice that reduces the problem size

- For each answer to the choice, choose answer + recurse

  (while considering only subsolutions consistent with that choice)

Simple example:

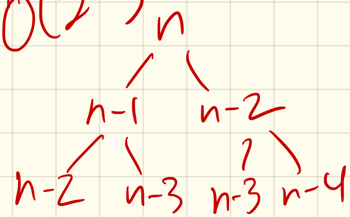
Fibonacci Numbers:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \quad \forall n \geq 2$$

Directly get an algorithm:

```
FIB(n):
if n < 2:
    return n
else
    return FIB(n-1) + FIB(n-2)
```

Runtime: exponential $\approx O(2^n)$



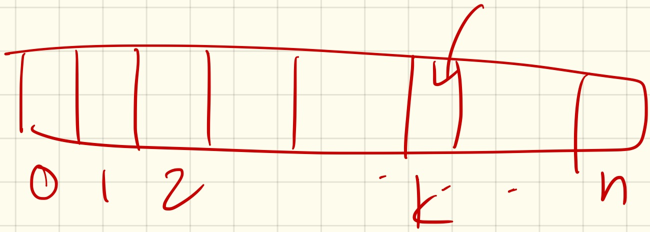
Correctness: follows from recursive dfn

How to improve?

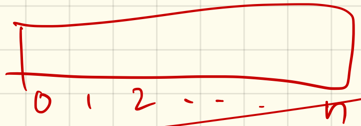
Avoid repeating work!

Make array to store answers, so next "call" becomes a $O(1)$ table lookup

```
MEMFIBO(n):
  if (n < 2)
      return n
  else
      if F[n] is undefined
          F[n] ← MEMFIBO(n − 1) + MEMFIBO(n − 2)
      return F[n]
```



Better yet:

F: 

create empty array F

$$\begin{array}{l}
\underline{\textsc{IterFibo}(n):} \\
\quad \left.\begin{array}{l} F[0] \leftarrow 0 \\ F[1] \leftarrow 1 \end{array}\right\} O(1) \\
O(n) \left[\begin{array}{l} \text{for } i \leftarrow 2 \text{ to } n \\ \qquad F[i] \leftarrow F[i-1] + F[i-2] \end{array}\right. \\
\quad \text{return } F[n] \;\;) \; O(1)
\end{array}$$

one store

1 addition

Correctness:

Obvious from rec. dfn

Run time & space:

time: $O(n)$

space: $O(n)$

Even better!

```
ITERFIBO2(n):
    prev ← 1
    curr ← 0
    for i ← 1 to n
        next ← curr + prev
        prev ← curr
        curr ← next
    return curr
```

i: ~~1~~ ~~2~~ ~~3~~ 4

prev: ~~1~~ ~~0~~ ~~1~~ ~~1~~ 2

curr: ~~0~~ ~~1~~ ~~1~~ ~~2~~ 3

next: ~~1~~ ~~1~~ ~~2~~ 3

Run time/space:

O(n)

4 variables

O(1)

Making change (again) (6.2)

How to make change using the smallest # of coins.

Suppose coins were 1¢, 3¢, & 7¢.

Make change for 77¢.

Options?

Think recursively!

Could choose: (first)

1¢: 1 + (# coins for 76¢)

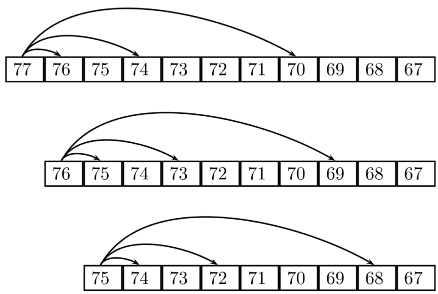
3¢: 1 + (# for 74¢)

7¢: 1 + (# for 70¢)

Compute all 3 options recursively, & choose best.

Formally:

$$bestNumCoins_M = \min \begin{cases} bestNumCoins_{M-1} + 1 \\ bestNumCoins_{M-3} + 1 \\ bestNumCoins_{M-7} + 1 \end{cases}$$



If you have coins $C_1 \ldots C_d$, get:

$$bestNumCoins_M = \min \begin{cases} bestNumCoins_{M-c_1} + 1 \\ bestNumCoins_{M-c_2} + 1 \\ \vdots \\ bestNumCoins_{M-c_d} + 1 \end{cases}$$

"Obvious" algorithm:

```
RecursiveChange(M, c, d)
if M = 0
    return 0
bestNumCoins ← ∞
for i ← 1 to d
    if M ≥ c_i
        numCoins ← RecursiveChange(M − c_i, c, d)
        if numCoins + 1 < bestNumCoins
            bestNumCoins ← numCoins + 1
return bestNumCoins
```

Correctness:

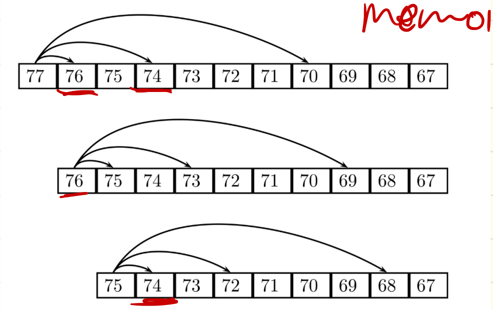
Try all options!

Runtime:

$\approx O(d^n)$ (actually worse...)

Problem: Just like Fibonacci algorithm!

memoization

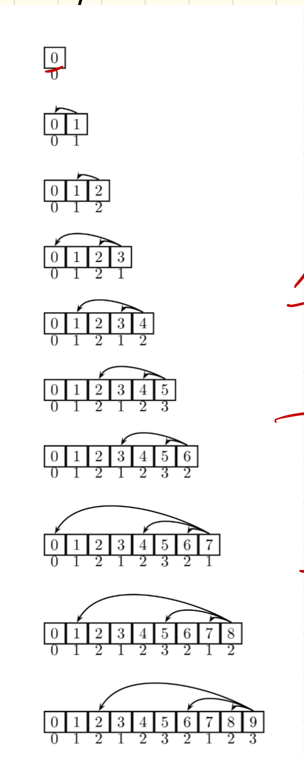


Solution: Don't recompute things!

DPChange$(M, \mathbf{c}, d)$

$bestNumCoins_0 \leftarrow 0$
for $m \leftarrow 1$ **to** M ← value to give change for
  $bestNumCoins_m \leftarrow \infty$
  for $i \leftarrow 1$ **to** d
    if $m \geq c_i$
      if $bestNumCoins_{m-c_i} + 1 < bestNumCoins_m$
        $bestNumCoins_m \leftarrow bestNumCoins_{m-c_i} + 1$
return $bestNumCoins_M$

Why is this better?



Runtime:

$O(Md)$

1¢, 3¢, 7¢

Table has M spots

each takes $O(d)$ time to fill in

Edit Distance (6.4 in book)

The minimum number of deletions, insertions, or substitutions of letters to transform between two strings.

Ex: F O O D

sub sub sub insertion

M O N E Y

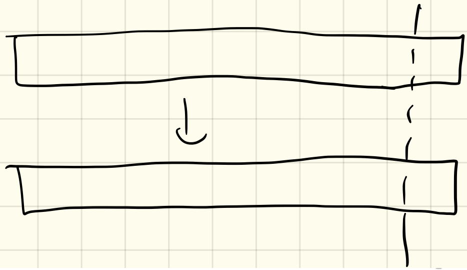
Note: Not Hamming distance.

Recursive formulation:

If I align like this, can observe:

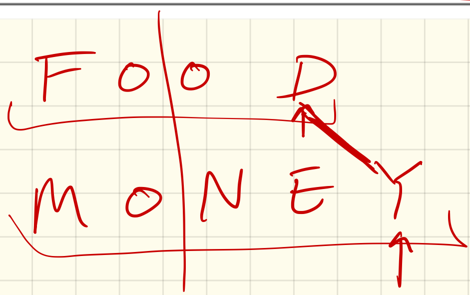
If you delete last (aligned) column, the rest will still be optimal for shorter substrings edit distance.

Why?



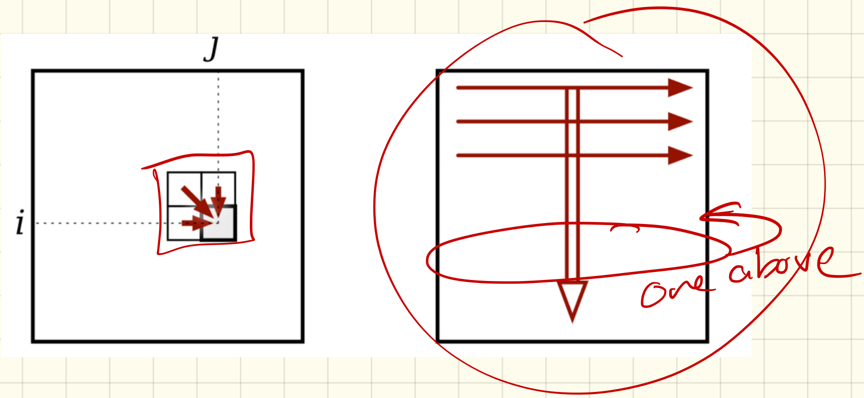
$$Edit(A[1..m],B[1..n]) = \min \begin{cases} Edit(A[1..m-1],B[1..n])+1 \\ Edit(A[1..m],B[1..n-1])+1 \\ \underline{Edit(A[1..m-1],B[1..n-1])}+[A[m]\neq B[n]] \end{cases}$$

add 1 if letters are different, or 0 if letters are same



Turn into "nice" recursion:

$$Edit(i,j) = \begin{cases} i & \text{if } j=0 \\ j & \text{if } i=0 \\ \min\left\{\begin{array}{l} Edit(i-1,j)+1, \\ Edit(i,j-1)+1, \\ Edit(i-1,j-1)+\left[A[i]\neq B[j]\right] \end{array}\right\} & \text{otherwise} \end{cases}$$



empty

		A	L	G	O	R	I	T	H	M
	0	1	2	3	4	5	6	7	8	9
A	1	0	1	2	3	4	5	6	7	8
L	2	1	0							
T	3									
R	4									
U	5									
I	6									
S	7									
T	8									
I	9									
C	10									

$$
Edit(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \left\{ \begin{array}{l} Edit(i-1,j)+1, \\ Edit(i,j-1)+1, \\ Edit(i-1,j-1)+\big[A[i] \neq B[j]\big] \end{array} \right\} & \text{otherwise} \end{cases}
$$

i

j

		A	L	G	O	R	I	T	H	M
	0	1	2	3	4	5	6	7	8	9
A	1	**0**	1	2	3	4	5	6	7	8
L	2	1	**0**	1	2	3	4	5	6	7
T	3	2	1	1	2	3	4	**4**	5	6
R	4	3	2	2	2	**2**	3	4	5	6
U	5	4	3	3	3	3	3	4	5	6
I	6	5	4	4	4	4	**3**	4	5	6
S	7	6	5	5	5	5	4	4	5	6
T	8	7	6	6	6	6	5	**4**	5	6
I	9	8	7	7	7	7	**6**	5	5	6
C	10	9	8	8	8	8	7	6	6	6

The memoization table for *Edit*(ALGORITHM, ALTRUISTIC)

A	L	G	O	R		I		T	H	M
A	L	T		R	U	I	S	T	I	C

6

Correctness:

Trying all possibilities

Edit[i,j] will always be best alignment of A[1..i] & B[1..j]

Runtime

$\textsc{EditDistance}(A[1..m], B[1..n])$:
  for $j \leftarrow 1$ to n
    $Edit[0, j] \leftarrow j$
  for $i \leftarrow 1$ to m
    $Edit[i, 0] \leftarrow i$
    for $j \leftarrow 1$ to n
      if $A[i] = B[j]$
        $O(1) \rightarrow Edit[i, j] \leftarrow \min\{Edit[i-1, j] + 1,\ Edit[i, j-1] + 1,\ Edit[i-1, j-1]\}$
      else
        $O(1) \rightarrow Edit[i, j] \leftarrow \min\{Edit[i-1, j] + 1,\ Edit[i, j-1] + 1,\ Edit[i-1, j-1] + 1\}$
  return $Edit[m, n]$

do A[i] & B[j] match

sub

O(mn)

O(n)

O(1)

$O(mn)$

Space?

input: $O(n+m)$

matrix: $O(nm)$

Improve:

If score is all you need, 2 arrays of size $\underline{O(n)}$