CIS 263 Introduction to Data Structures and Algorithms

Dynamic Programming

Dynamic Programming

Recursion:

```
int fib(int n)
{
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}
```

Exponential time

Dynamic Programming

Recursion:

```
int fib(int n)
{
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}
```

Exponential time

Dynamic Programming :

```
f[0] = 0;
f[1] = 1;

for (i = 2; i <= n; i++)
{
    f[i] = f[i-1] + f[i-2];
}

return f[n];
```

Linear time

Recursion

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

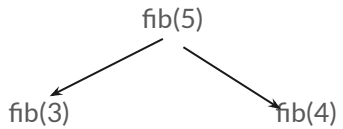
fib(5)

$$fib(n) = \begin{cases} 0 \text{ if } n = 0 \\ 1 \text{ if } n = 1 \\ fib(n-2) + fib(n-1) \text{ if } n > 2 \end{cases}$$

```
def fib(n):
    if n <=1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

Recursion

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

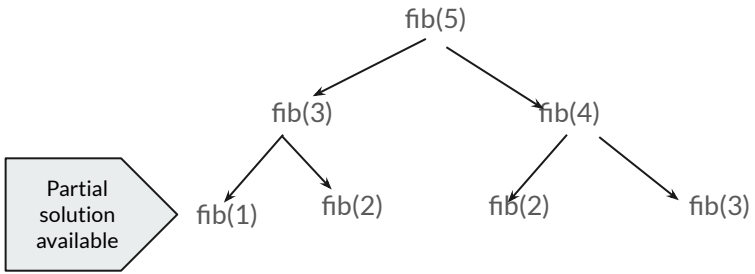


$$fib(n) = \begin{cases} 0 \text{ if } n = 0 \\ 1 \text{ if } n = 1 \\ fib\ (n-2) + fib(n-1) \text{ if } n > 2 \end{cases}$$

```
def fib(n):
    if n <=1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

Recursion

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

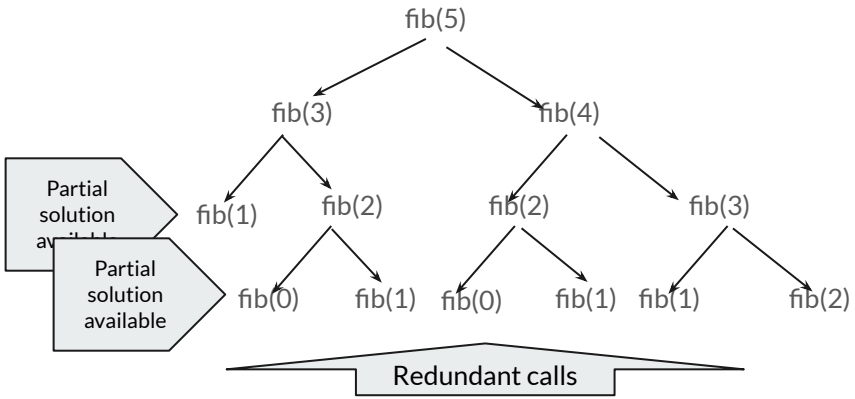


$$fib(n) = \begin{cases} 0 \text{ if } n = 0 \\ 1 \text{ if } n = 1 \\ fib(n-2) + fib(n-1) \text{ if } n > 2 \end{cases}$$

```
def fib(n):
    if n <=1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

Recursion

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

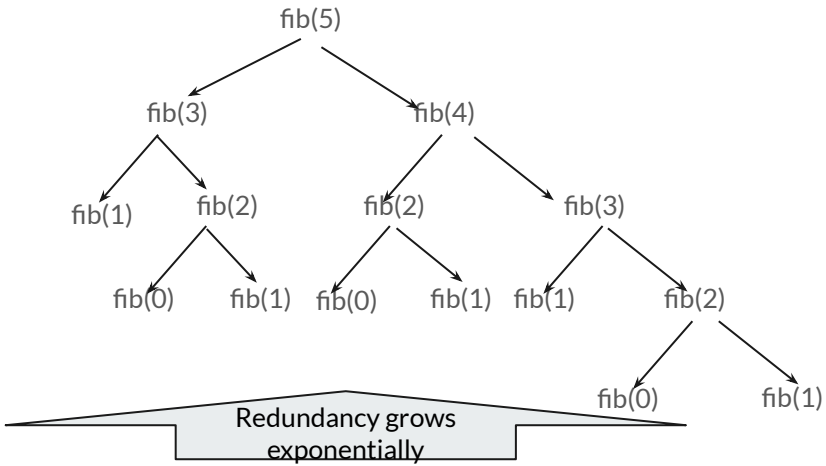


$$fib(n) = \begin{cases} 0 \text{ if } n = 0 \\ 1 \text{ if } n = 1 \\ fib(n-2) + fib(n-1) \text{ if } n > 2 \end{cases}$$

```
def fib(n):
    if n <=1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

Recursion

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

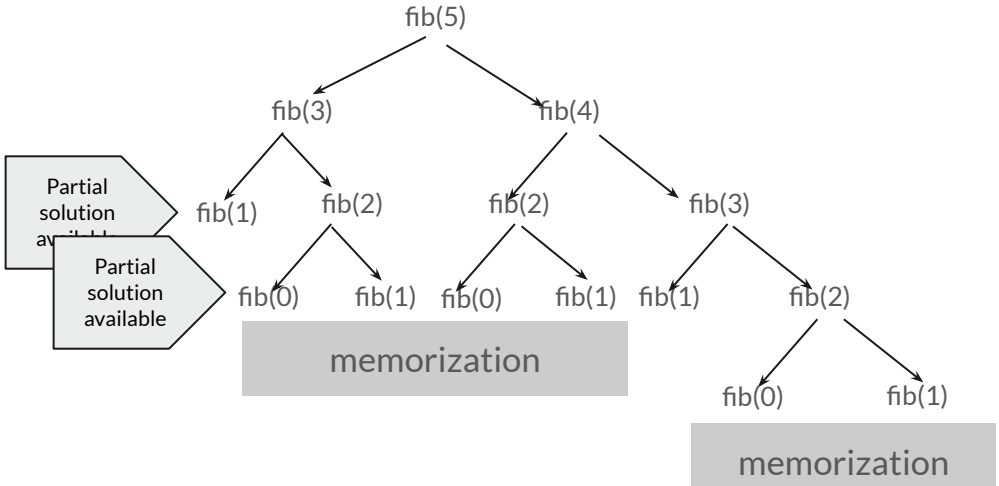


$$fib(n) = \begin{cases} 0 \text{ if } n = 0 \\ 1 \text{ if } n = 1 \\ fib(n-2) + fib(n-1) \text{ if } n > 2 \end{cases}$$

```
def fib(n):
    if n <=1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

Recursion

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5



$$fib(n) = \begin{cases} 0 \text{ if } n = 0 \\ 1 \text{ if } n = 1 \\ fib(n-2) + fib(n-1) \text{ if } n > 2 \end{cases}$$

```
def fib(n):
    if n <=1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

Dynamic programming

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

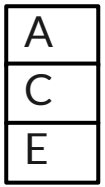
```
def fib(n):
    f = [0]*n
    f[0] = 0
    f[1]  = 1

    for i in range(2, n):
        f[i] = f[i-1] + f[i -2]
    return f
```

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

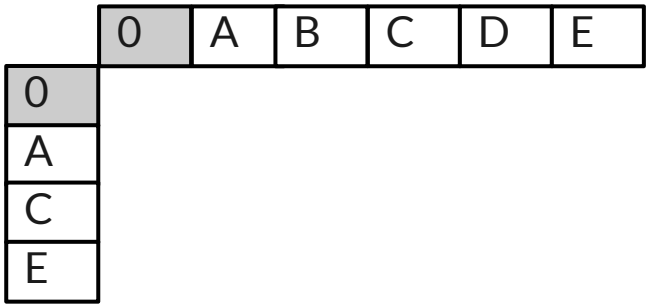
Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”



Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”



Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0						
A						
C						
E						

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A						
C						
E						

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0					
C	0					
E	0					

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0					
C	0					
E	0					

match

```
If str1[i-1] == str2[j-1]:
        lcd[i][j] = 1 +lcd[i-1][j-1]
```

no-match

```
else:
        lcd[i][j] =  max (
                lcd[i-1][j],
                lcd[i][j-1]
                )
```

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1				
C	0					
E	0					

match

If str1[i-1] == str2[j-1]:
    lcd[i][j] = 1 +lcd[i-1][j-1]

no-match

else:
    lcd[i][j] = max (
        lcd[i-1][j],
        lcd[i][j-1]
)

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0					
E	0					

match

If str1[i-1] == str2[j-1]:
    lcd[i][j] = 1 +lcd[i-1][j-1]

no-match

else:
    lcd[i][j] = max (
        lcd[i-1][j],
        lcd[i][j-1]
)

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
E	0	1	1	2	2	3

match

```
If str1[i-1] == str2[j-1]:
        lcd[i][j] = 1 +lcd[i-1][j-1]
```

no-match

```
else:
        lcd[i][j] =  max (
                lcd[i-1][j],
                lcd[i][j-1]
                )
```

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
E	0	1	1	2	2	3

Recurrence relations

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
E	0	1	1	2	2	3

Recurrence relations

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
E	0	1	1	2	2	3

Backtracking

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
E	0	1	1	2	2	3

E

Backtracking

Longest Common Subsequence

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
E	0	1	1	2	2	3

A	C	E

Backtracking

Minimum Edit Distance

str1 = “ABCDE”
str2 = “ACE”

Minimum Edit Distance

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0						
A						
C						
E						

What should be the initial values?

Minimum Edit Distance

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	1	2	3	4	5
A	1					
C	2					
E	3					

Minimum Edit Distance

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	1	2	3	4	5
A	1	0	1	2	3	4
C	2	1	1	1	2	3
E	3	2	2	2	2	2

```
If str1[i-1] == str2[j-1]:
        med[i][j] = med[i-1][j-1]
```

```
else:
        med[i][j] = 1 + min (
                med[i-1][j],
                med[i][j-1]
                med[i-1][j-1]
                )
```

Minimum Edit Distance

str1 = “ABCDE”
str2 = “ACE”

	0	A	B	C	D	E
0	0	1	2	3	4	5
A	1	0	1	2	3	4
C	2	1	1	1	2	3
E	3	2	2	2	2	2

A == A

add “C”: ACB

replace “B” with E: ACBE

replace “B” with “C”: ABC

add “E”: ABCE

AB ⇨ ACE

Other applications

- Matrix chain multiplication
- Sequence alignment
- Tower of Hanoi puzzle
- Markov decision process, Viterbi Algorithm
- Dynamic Time warping
- Floyd’s shorted path algorithm