Announcements

- Exam 2 Solutions Online
- Homework 4 Online, Due Friday
- Change to time and location of Stanlislaw's Tuesday office hours

Today

- Dynamic Programming
- Longest Common Subsequence

Dynamic Programming (Ch 6)

- Background and past examples
- Longest Common Subsequence
- Knapsack
- Chain Matrix Multiplication
- All-Pairs Shortest Paths
- Independent Sets of Trees
- Travelling Salesman

Computing Fibonacci Numbers

Recall:

$$F_n = 1 \text{ if } n = 0 \text{ or } 1$$

$$F_n = F_{n-1} + F_{n-2} \text{ otherwise}$$

Naïve Algorithm

```
Fib(n)
  If n ≤ 1
    Return 1
  Else
    Return Fib(n-1)+Fib(n-2)
```

Naïve Algorithm

```
Fib(n)
  If n ≤ 1
     Return 1
  Else
     Return Fib(n-1)+Fib(n-2)
```

Far too slow!

Improved Algorithm

```
Fib2(n)
  Initialize A[0..n]
  A[0] = A[1] = 1
  For k = 2 to n
     A[k] = A[k-1] + A[k-2]
  Return A[n]
```

Improved Algorithm

```
Fib2(n)
  Initialize A[0..n]
  A[0] = A[1] = 1
  For k = 2 to n
     A[k] = A[k-1] + A[k-2]
  Return A[n]
```

Tabulation of answers avoids runaway recursive calls.

Another Example

Something similar happens with our algorithm for shortest paths in DAGs.

Another Example

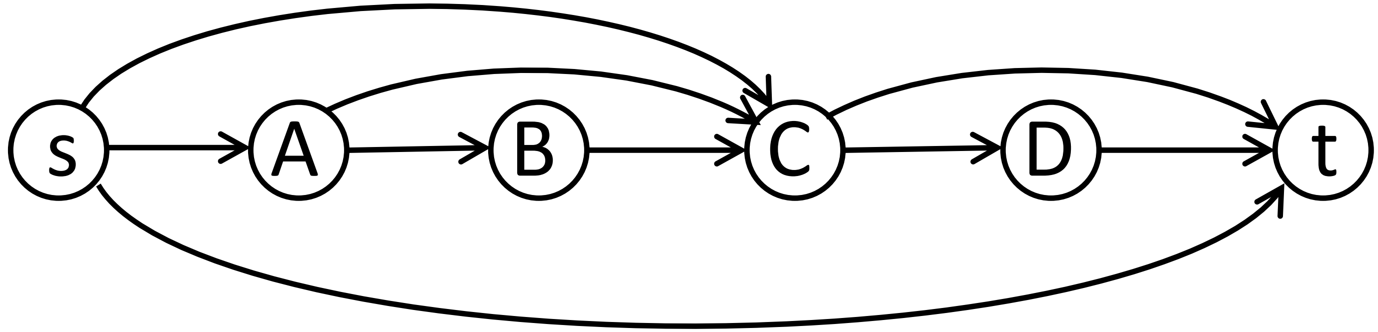
Something similar happens with our algorithm for shortest paths in DAGs.

This was based on the basic recursive formula

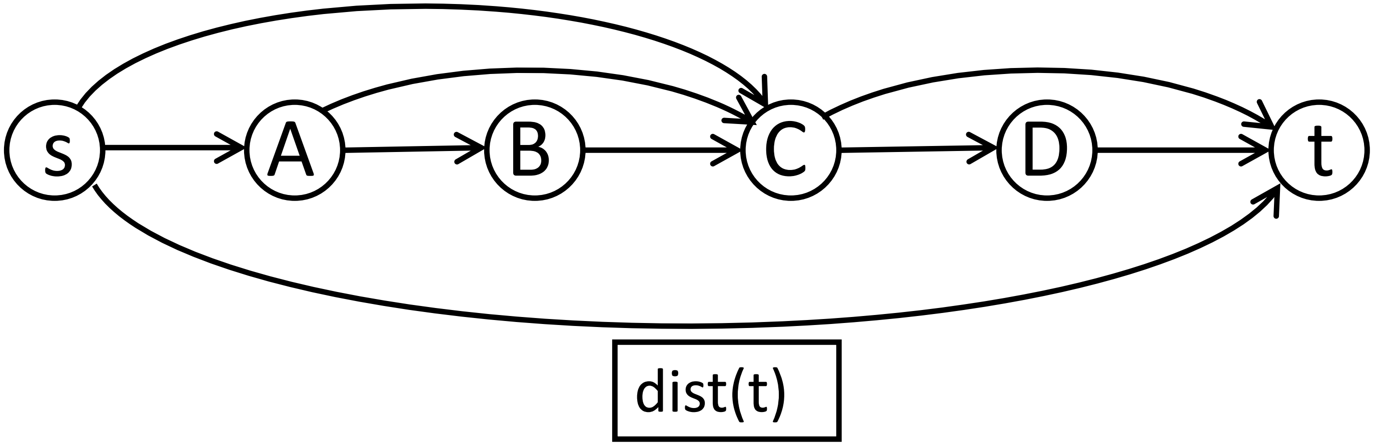
$$\text{dist}(w) = \min_{(v,w)\in E} \text{dist}(v) + \ell(v,w).$$

applied to vertices in topological order.

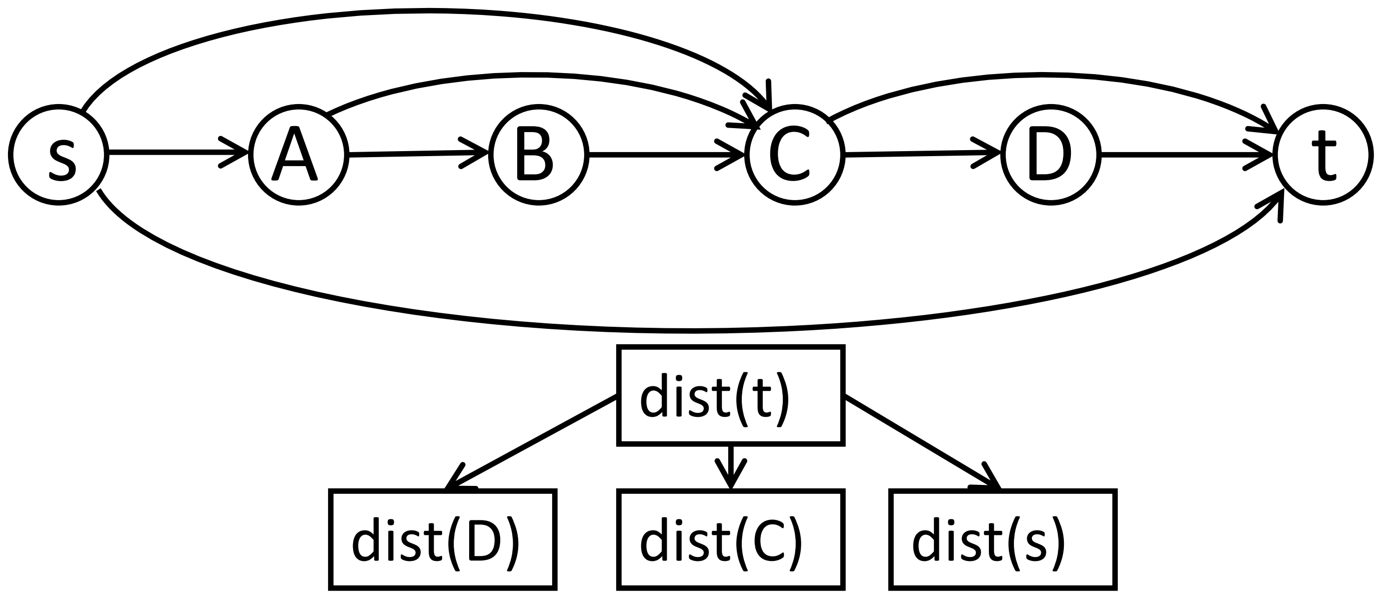
Example



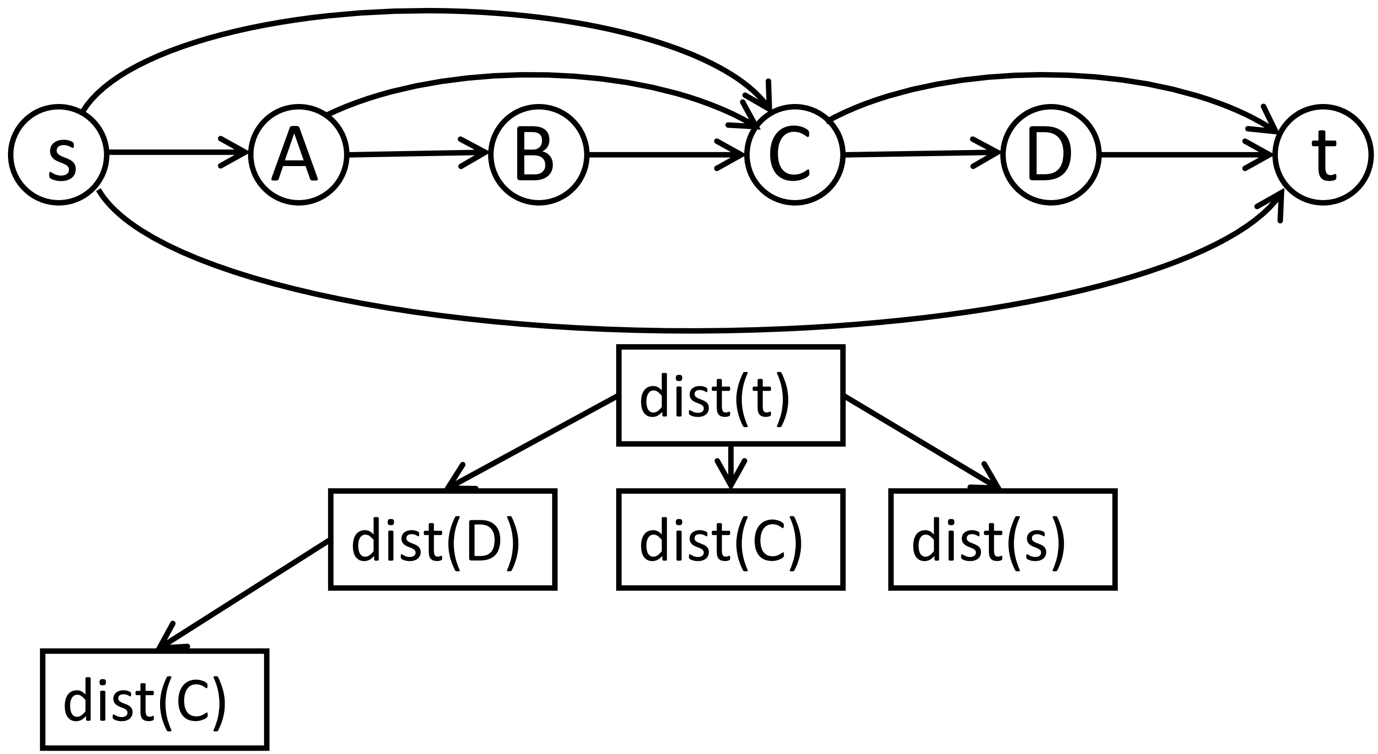
Example



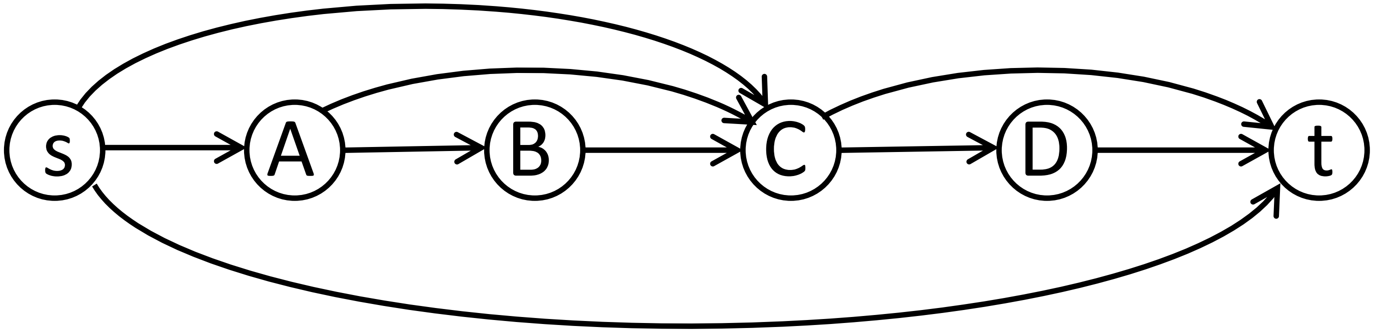
Example

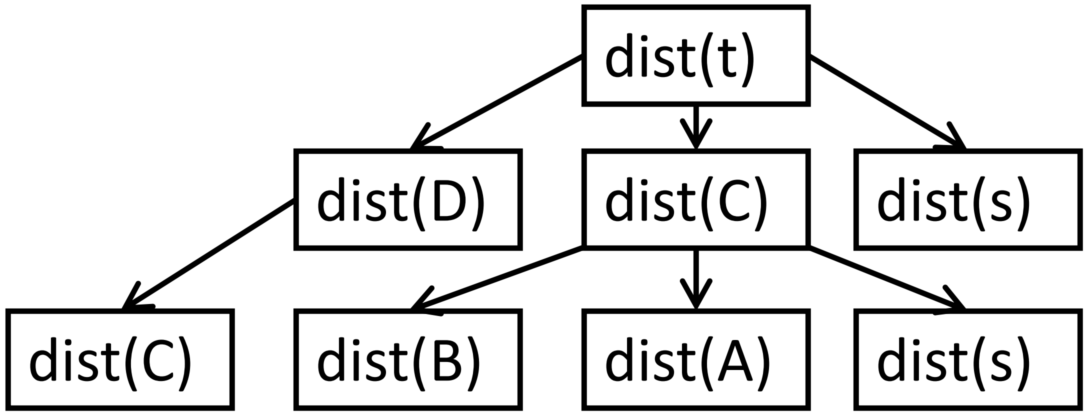


Example

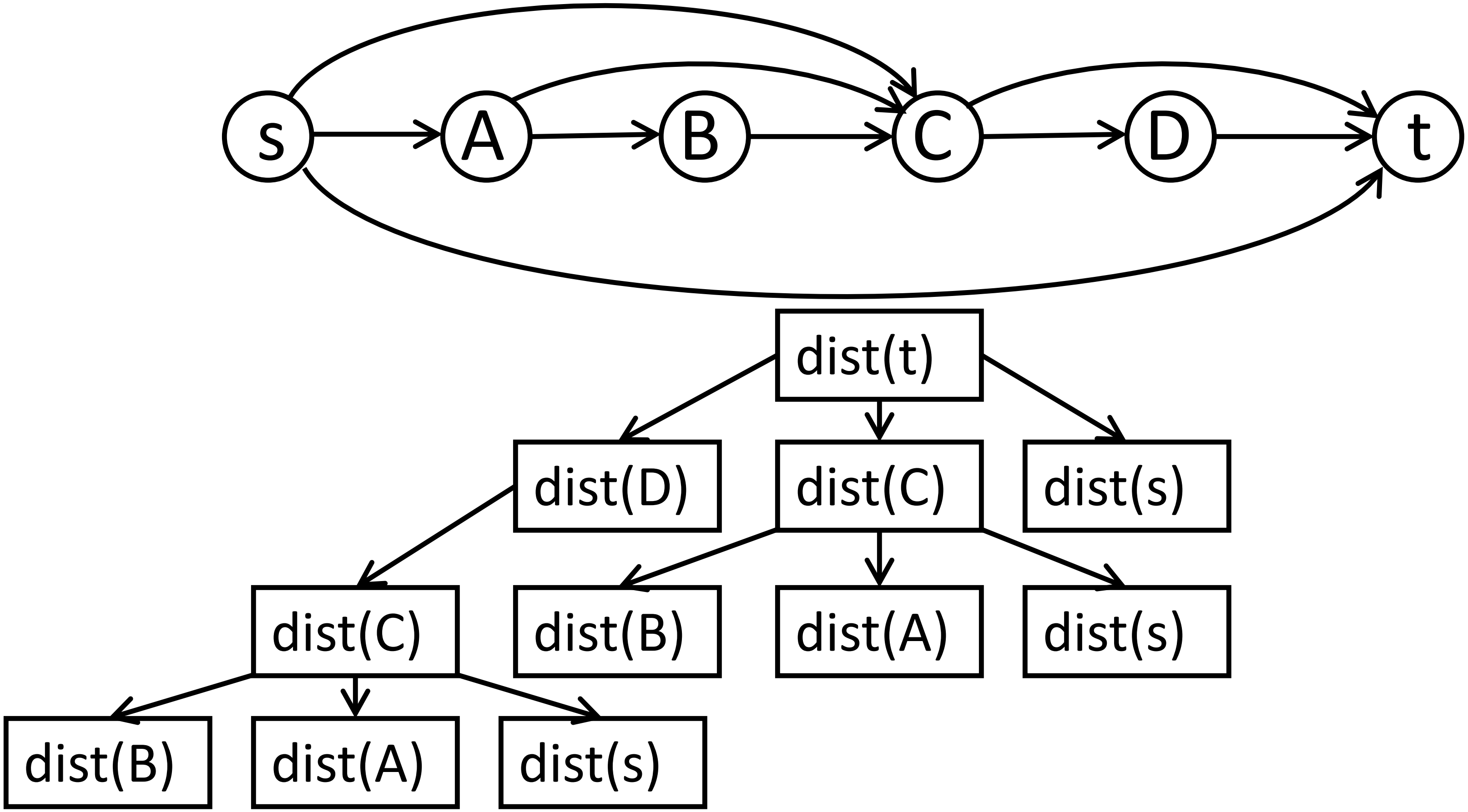


Example

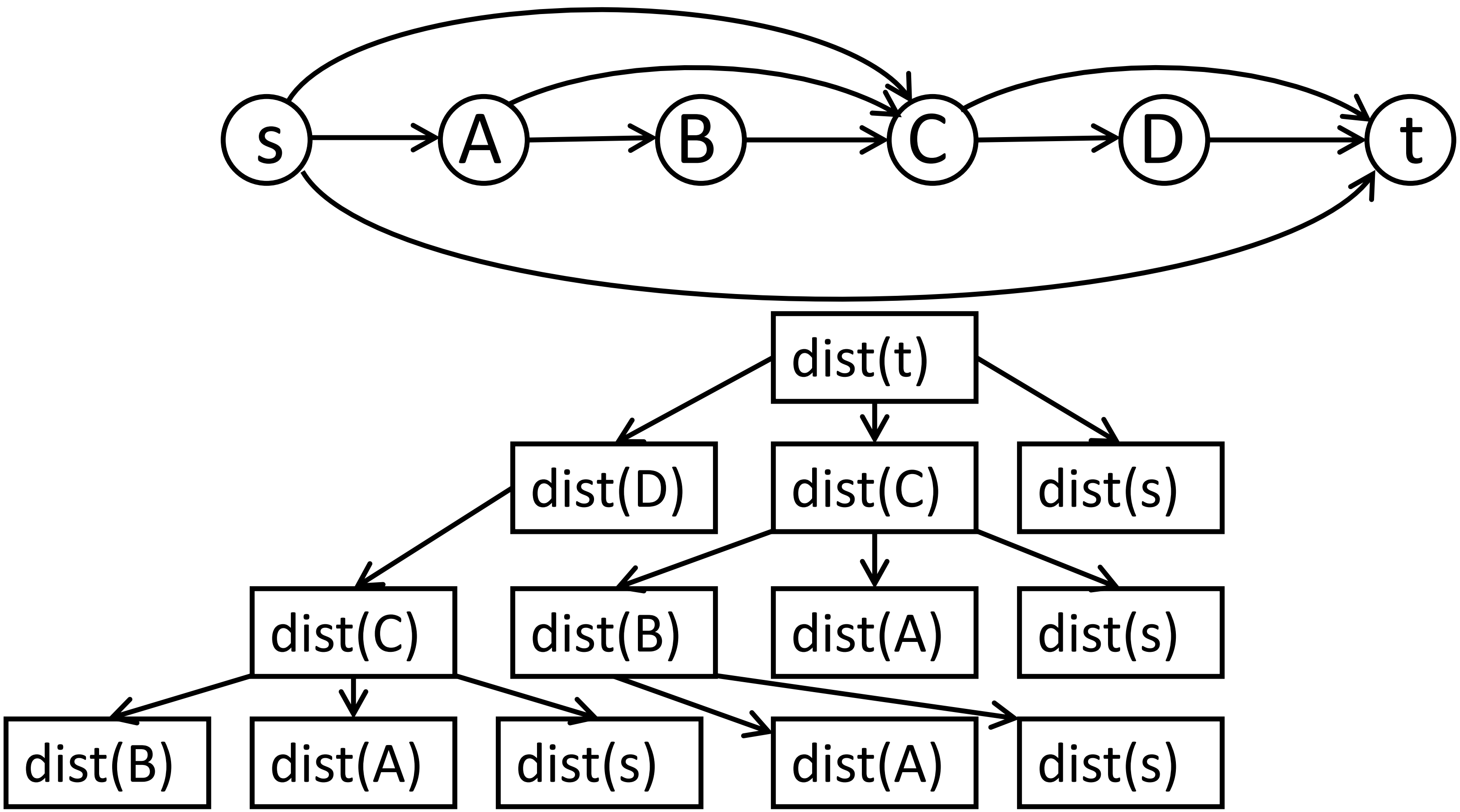




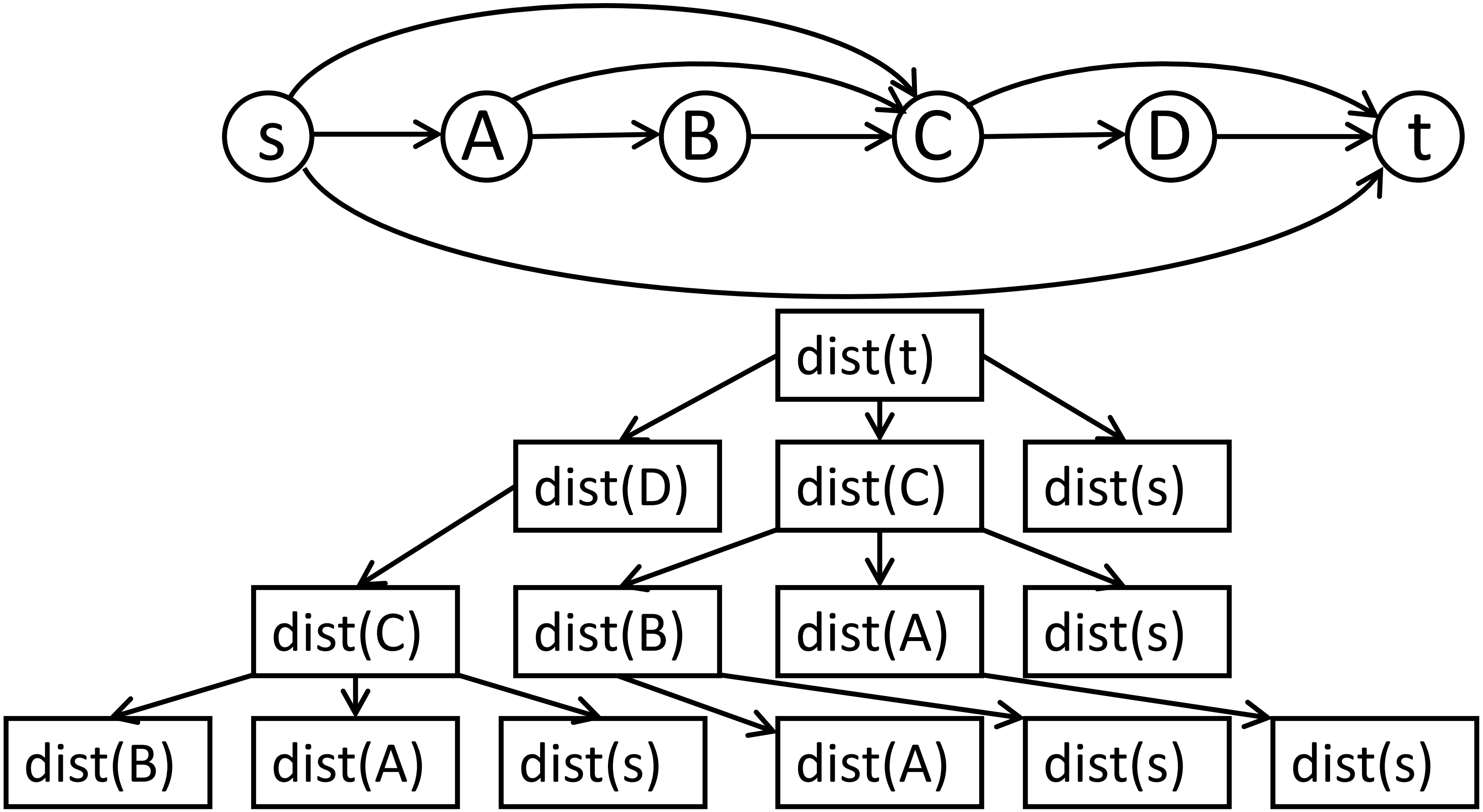
Example


s
A
B
C
D
t
dist(t)
dist(D)
dist(C)
dist(s)
dist(C)
dist(B)
dist(A)
dist(s)
dist(B)
dist(A)
dist(s)

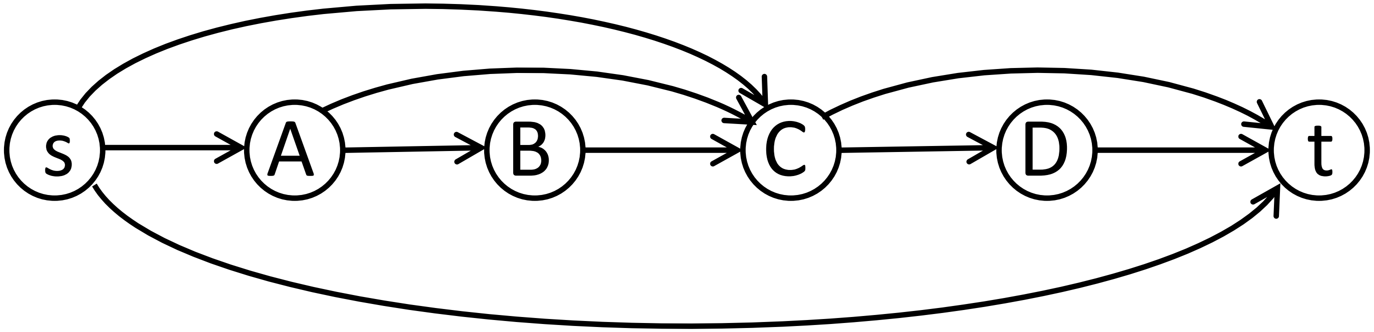
Example


s
A
B
C
D
t
dist(t)
dist(D)
dist(C)
dist(s)
dist(C)
dist(B)
dist(A)
dist(s)
dist(B)
dist(A)
dist(s)
dist(A)
dist(s)

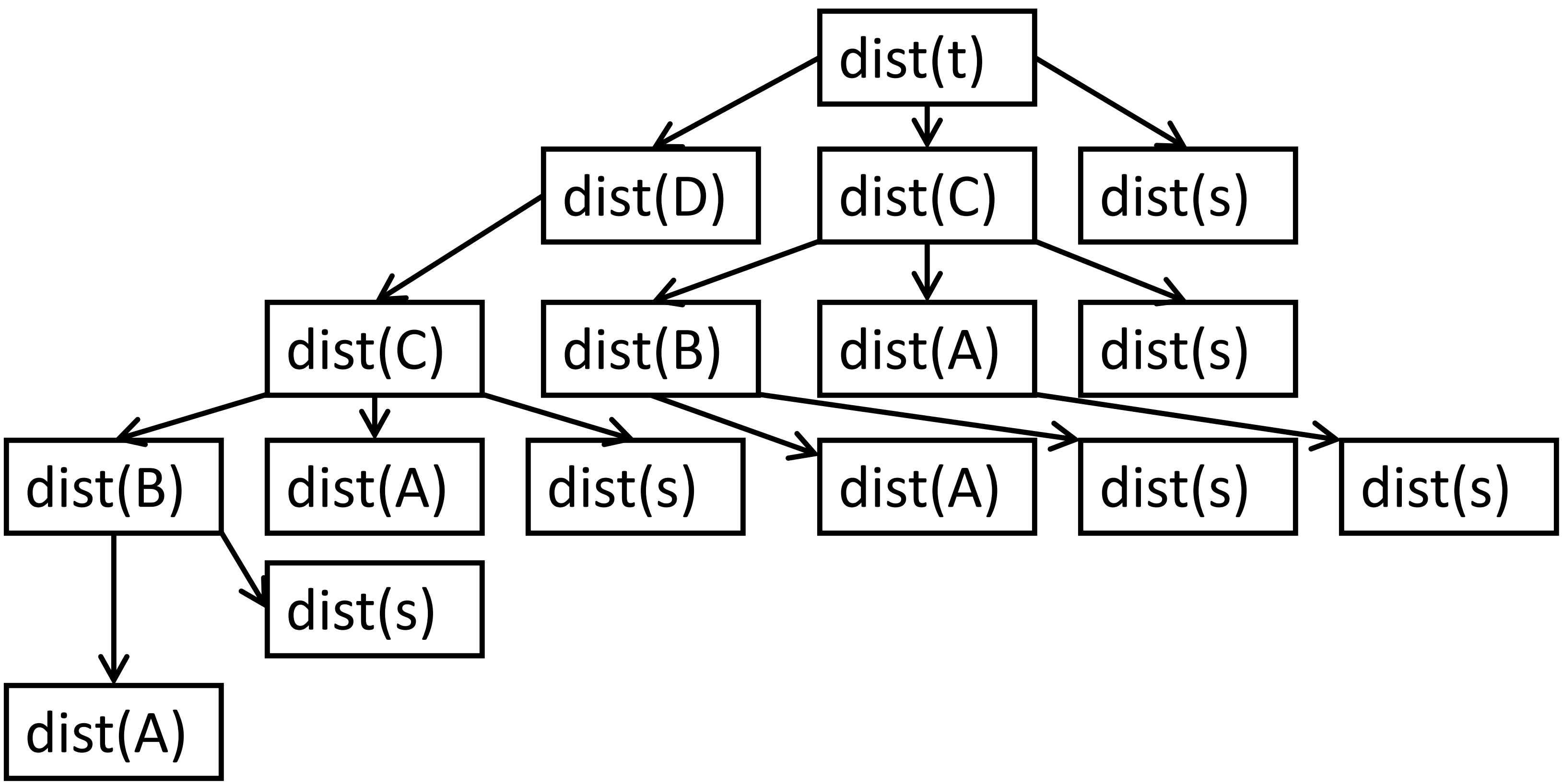
Example


s
A
B
C
D
t
dist(t)
dist(D)
dist(C)
dist(s)
dist(C)
dist(B)
dist(A)
dist(s)
dist(B)
dist(A)
dist(s)
dist(A)
dist(s)
dist(s)

Example

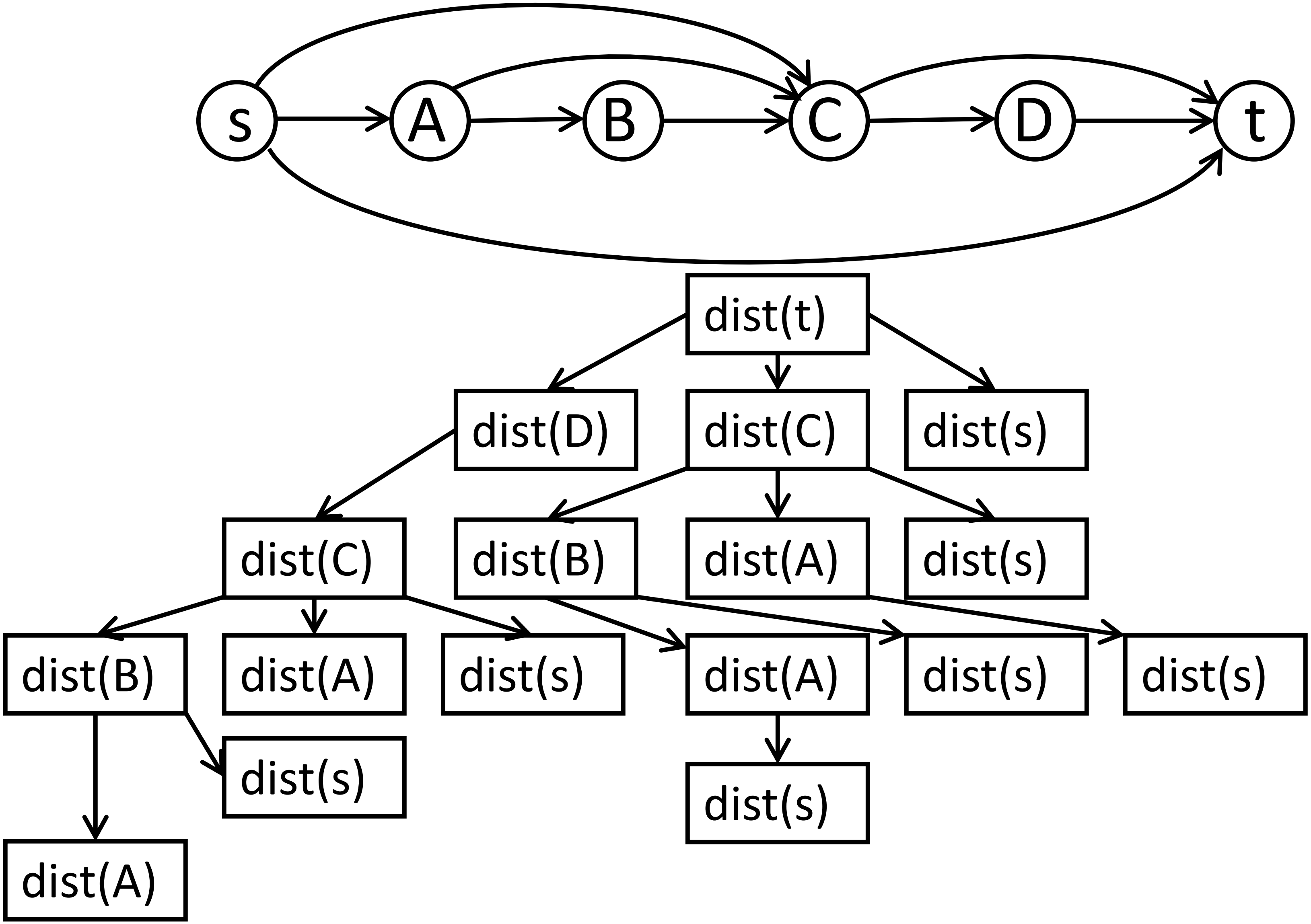


s
A
B
C
D
t

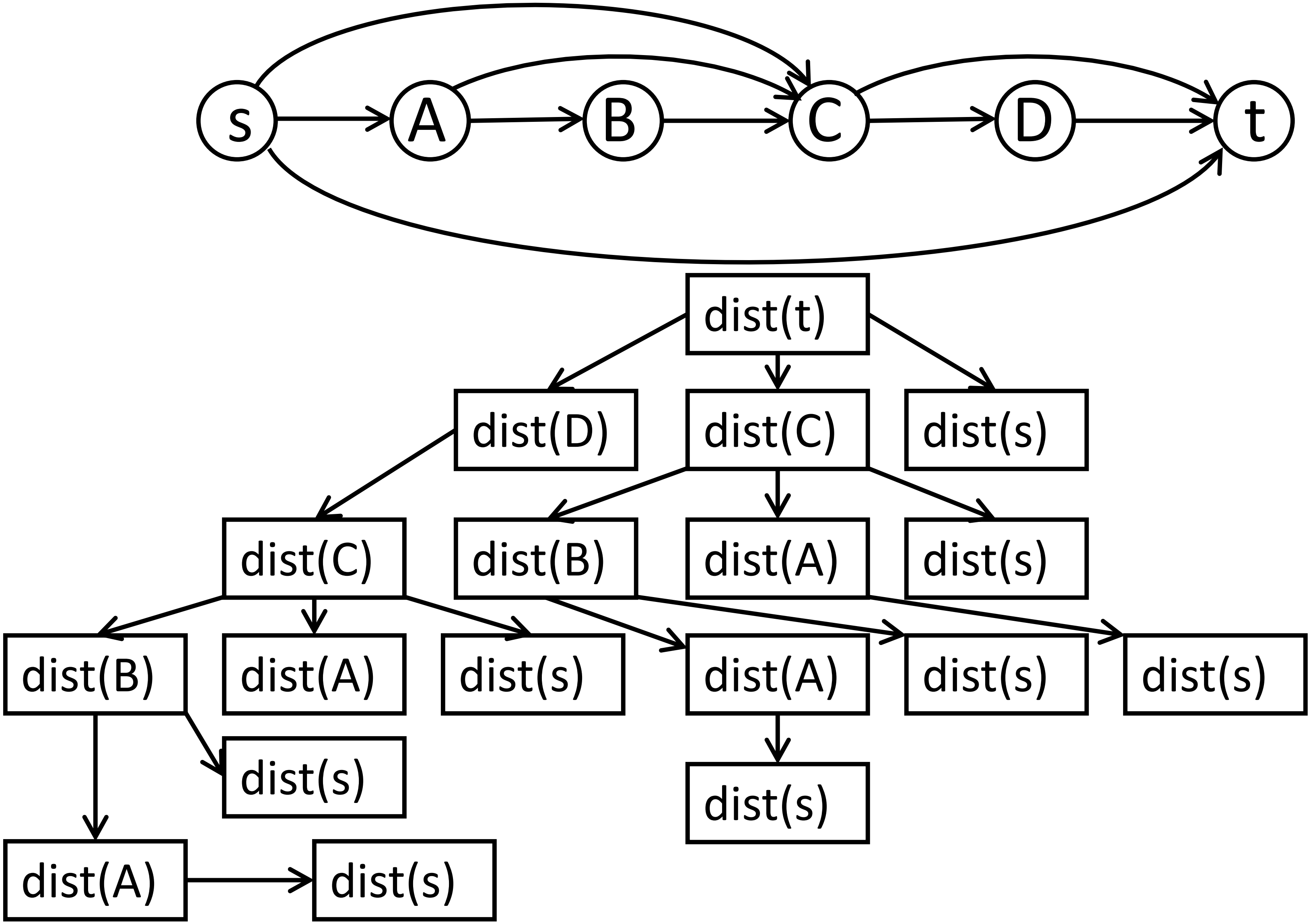


dist(t)
dist(D)
dist(C)
dist(s)
dist(C)
dist(B)
dist(A)
dist(s)
dist(B)
dist(A)
dist(s)
dist(A)
dist(s)
dist(s)
dist(s)
dist(A)

Example


s
A
B
C
D
t
dist(t)
dist(D)
dist(C)
dist(s)
dist(C)
dist(B)
dist(A)
dist(s)
dist(B)
dist(A)
dist(s)
dist(A)
dist(s)
dist(s)
dist(s)
dist(A)
dist(s)

Example


s
A
B
C
D
t
dist(t)
dist(D)
dist(C)
dist(s)
dist(C)
dist(B)
dist(A)
dist(s)
dist(B)
dist(A)
dist(s)
dist(A)
dist(s)
dist(s)
dist(s)
dist(s)
dist(A)
dist(s)

Simplify by Tabulating

Instead of computing these values recursively, compute them one at a time, recording them. Then in the future, you only need to do table lookups.

Dynamic Programming

Our final general algorithmic technique:

1. Break problem into smaller subproblems.
2. Find recursive formula solving one subproblem in terms of simpler ones.
3. Tabulate answers and solve all subproblems.

Question: Dynamic Program

Which of the following algorithms that we have covered so far involves a dynamic program?

A) Bellman-Ford
B) Optimal Caching
C) Computing SCCs
D) Closest Pair of Points
E) Karatsuba Multiplication

Question: Dynamic Program

Which of the following algorithms that we have covered so far involves a dynamic program?

A) Bellman-Ford
B) Optimal Caching
C) Computing SCCs
D) Closest Pair of Points
E) Karatsuba Multiplication

$$\mathrm{dist}_k(w) = \min_{(v,w)\in E} \mathrm{dist}_{k-1}(v) + \ell(v,w).$$

Subsequences

Given a sequence, say `ABCBA`, a <u>subsequence</u> is the sequence obtained by deleting some letters and leaving the rest in the same order.

Subsequences

Given a sequence, say ABCBA, a subsequence is the sequence obtained by deleting some letters and leaving the rest in the same order.

For example, ABCBA would have a subsequence ABCBA = ACB.

Longest Common Subsequence

We say that a sequence is a common subsequence of two others, if it is a subsequence of both.

Longest Common Subsequence

We say that a sequence is a <u>common subsequence</u> of two others, if it is a subsequence of both.

For example `ABC` is a common subsequence of `ADBCA` and `AABBC`.

Longest Common Subsequence

We say that a sequence is a common subsequence of two others, if it is a subsequence of both.

For example `ABC` is a common subsequence of `ADBCA` and `AABBC`.

Problem: Given two sequences compute the longest common subsequence. That is the subsequence with as many letters as possible.

Question: LCSS

What is the length of the longest common subsequence of `ABCBA` and `ABACA`?

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

Question: LCSS

What is the length of the longest common subsequence of ABCBA and ABACA?

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

ABCA = ABCBA = ABACA

Case Analysis

How do we compute LCSS($A_1A_2 \ldots A_n$, $B_1B_2 \ldots B_m$)?

Case Analysis

How do we compute LCSS($A_1A_2 \ldots A_n$, $B_1B_2 \ldots B_m$)?

Consider cases for the common subsequence:

1. It does not use A_n.
2. It does not use B_m.
3. It uses both A_n and B_m and these characters are the same.

Case 1

If the common subsequence does not use A_n, it is actually a common subsequence of

$A_1A_2 \ldots A_{n-1}$, and $B_1B_2 \ldots B_m$

Case 1

If the common subsequence does not use A_n, it is actually a common subsequence of

$A_1A_2 \ldots A_{n-1}$, and $B_1B_2 \ldots B_m$

Therefore, in this case, the longest common subsequence would be LCSS($A_1A_2 \ldots A_{n-1}$, $B_1B_2 \ldots B_m$).

Case 2

If the common subsequence does not use B_m, it is actually a common subsequence of

$A_1 A_2 \ldots A_n$, and $B_1 B_2 \ldots B_{m-1}$

Case 2

If the common subsequence does not use B_m, it is actually a common subsequence of

$A_1A_2 \ldots A_n$, and $B_1B_2 \ldots B_{m-1}$

Therefore, in this case, the longest common subsequence would be LCSS($A_1A_2 \ldots A_n$, $B_1B_2 \ldots B_{m-1}$).

Case 3

If a common subsequence uses both A_n and B_m…

Case 3

If a common subsequence uses both A_n and B_m…

- These characters must be the same.

Case 3

If a common subsequence uses both $\mathtt{A}_\mathtt{n}$ and $\mathtt{B}_\mathtt{m}$…

- These characters must be the same.
- Such a subsequence is obtained by taking a common subsequence of:
  $\mathtt{A}_1\mathtt{A}_2\ldots\mathtt{A}_{\mathtt{n-1}}$, and $\mathtt{B}_1\mathtt{B}_2\ldots\mathtt{B}_{\mathtt{m-1}}$
  and adding a copy of $\mathtt{A}_\mathtt{n}$ = $\mathtt{B}_\mathtt{m}$ to the end.

Case 3

If a common subsequence uses both A_n and B_m…

- These characters must be the same.
- Such a subsequence is obtained by taking a common subsequence of:
  $A_1A_2…A_{n-1}$, and $B_1B_2…B_{m-1}$
  and adding a copy of $A_n = B_m$ to the end.
- The longest length of such a subsequence is
  LCSS($A_1A_2…A_{n-1}$, $B_1B_2…B_{m-1}$)+1.

Recursion

On the other hand, the longest common subsequence must come from one of these cases. In particular, it will always be the one that gives the biggest result.

Recursion

On the other hand, the longest common subsequence must come from one of these cases. In particular, it will always be the one that gives the biggest result.

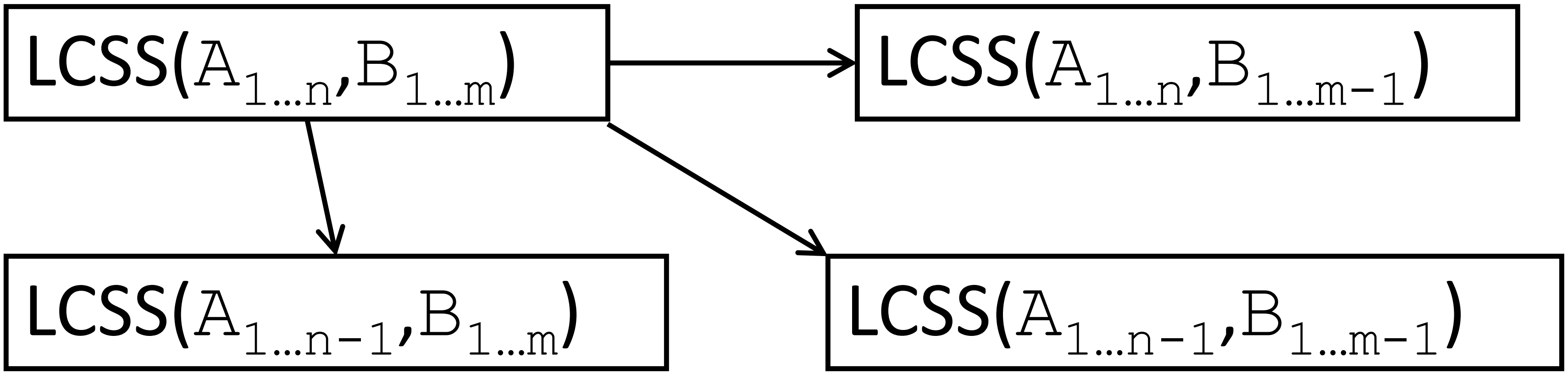
$\text{LCSS}(A_1A_2\ldots A_n, B_1B_2\ldots B_m) =$
$\quad \text{Max}(\text{LCSS}(A_1A_2\ldots A_{n-1}, B_1B_2\ldots B_m),$
$\quad\quad \text{LCSS}(A_1A_2\ldots A_n, B_1B_2\ldots B_{m-1}),$
$\quad\quad [\text{LCSS}(A_1A_2\ldots A_{n-1}, B_1B_2\ldots B_{m-1})+1])$

[where the last option is only allowed if $A_n = B_m$]

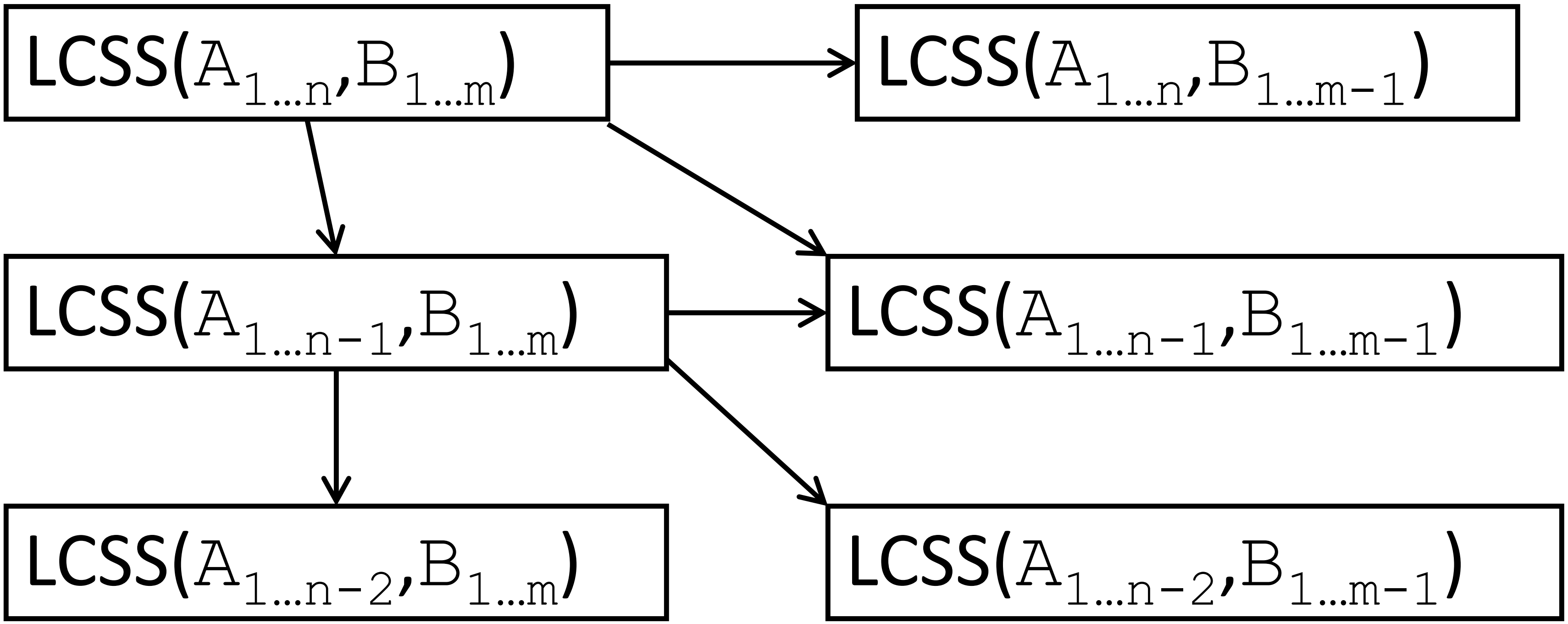
Recursion

LCSS($A_{1\ldots n}$,$B_{1\ldots m}$)

Recursion

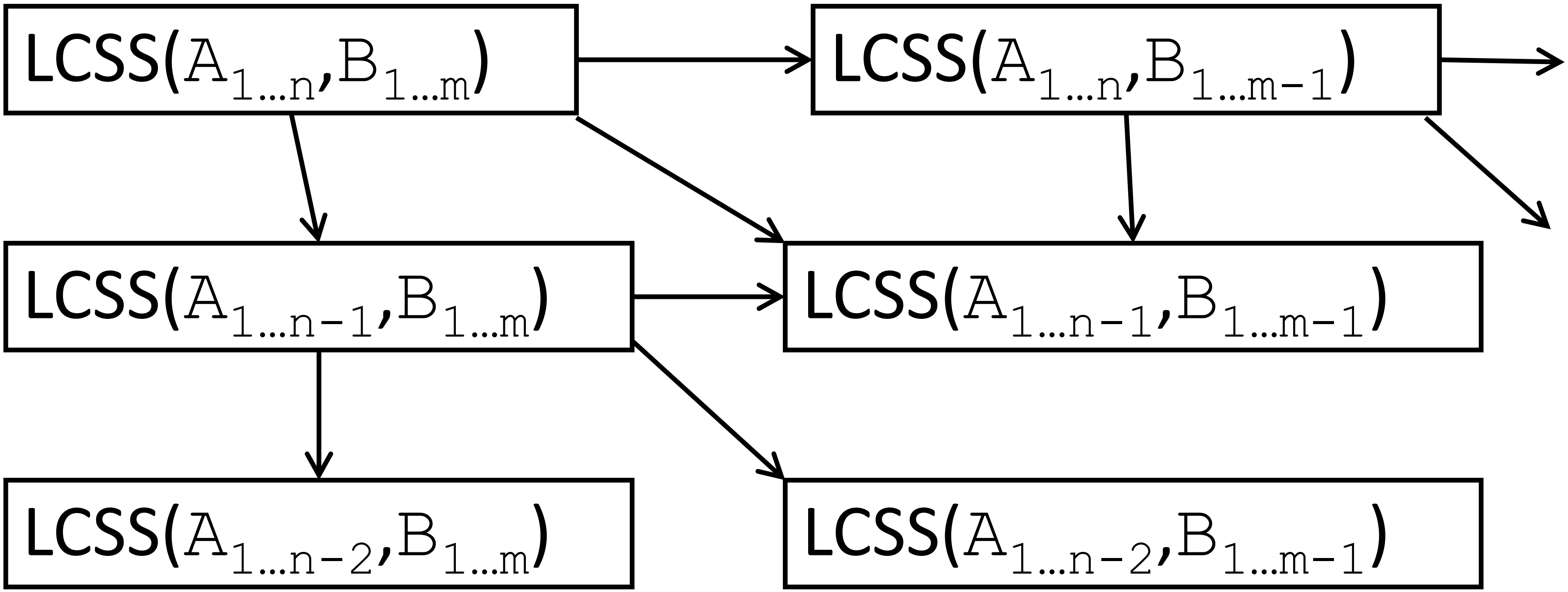

LCSS($A_{1\ldots n}$,$B_{1\ldots m}$)
LCSS($A_{1\ldots n}$,$B_{1\ldots m-1}$)
LCSS($A_{1\ldots n-1}$,$B_{1\ldots m}$)
LCSS($A_{1\ldots n-1}$,$B_{1\ldots m-1}$)

Recursion



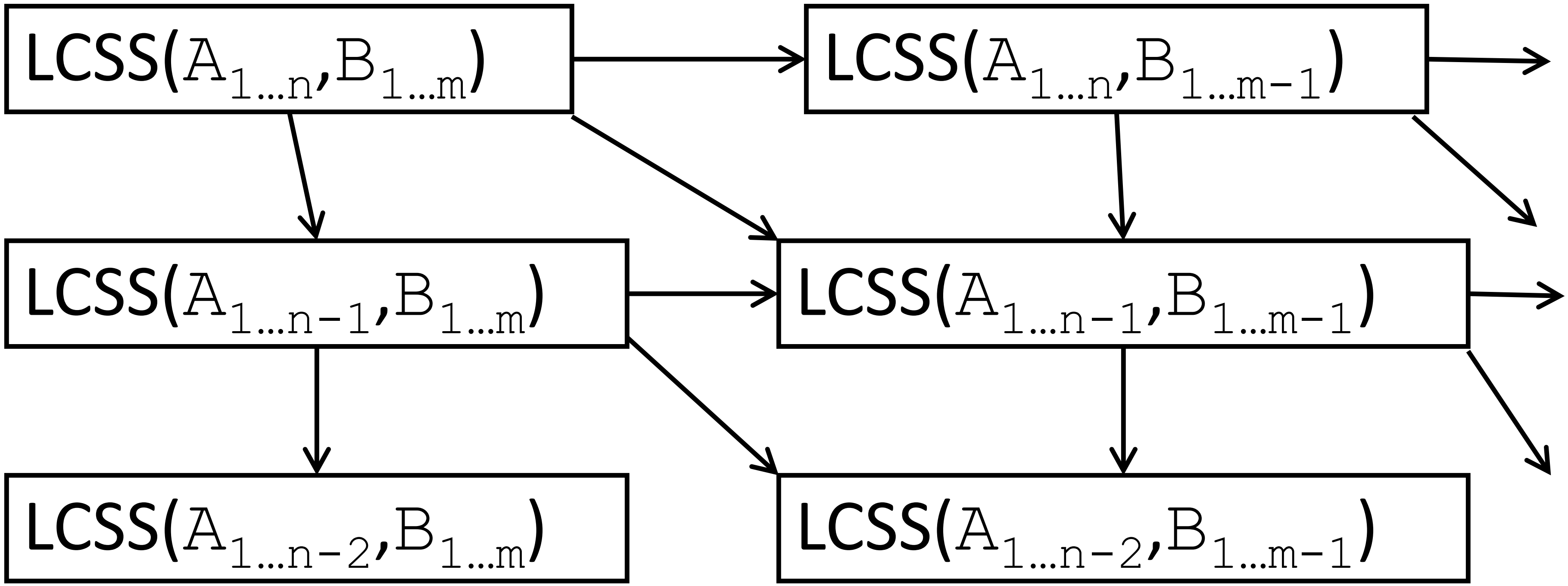
LCSS($A_{1…n}$,$B_{1…m}$)
LCSS($A_{1…n}$,$B_{1…m-1}$)
LCSS($A_{1…n-1}$,$B_{1…m}$)
LCSS($A_{1…n-1}$,$B_{1…m-1}$)
LCSS($A_{1…n-2}$,$B_{1…m}$)
LCSS($A_{1…n-2}$,$B_{1…m-1}$)

Recursion

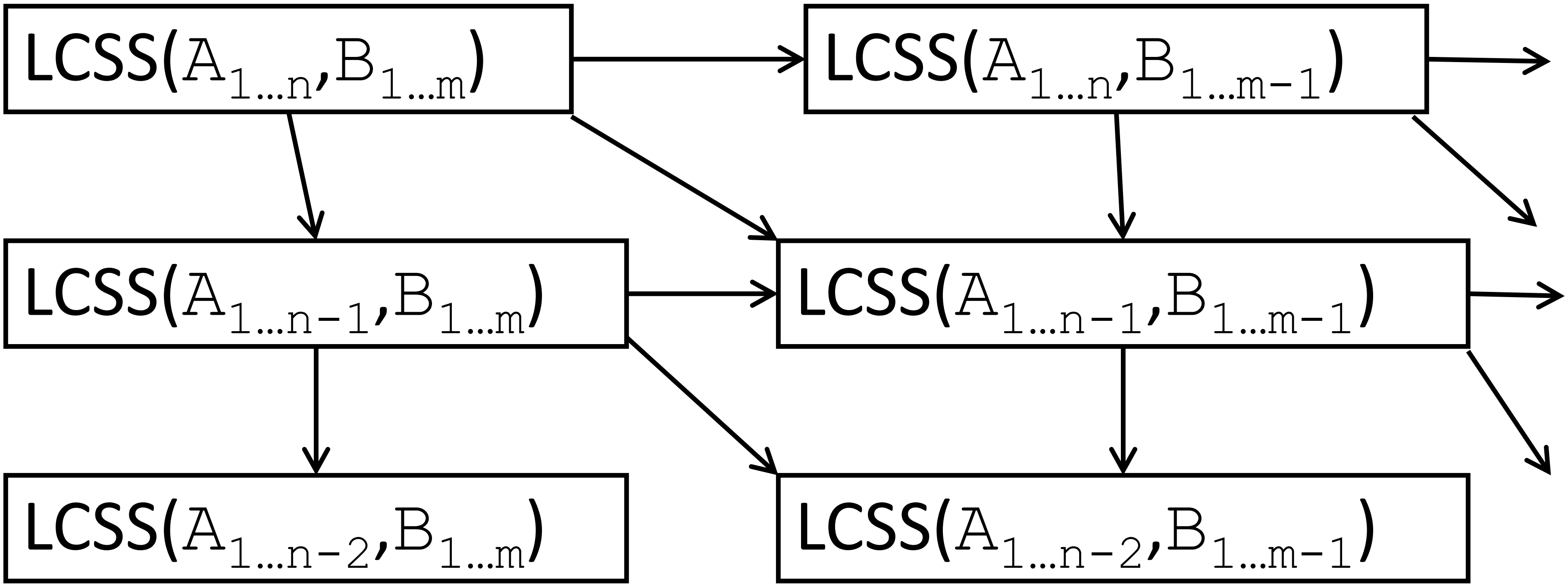


LCSS(A1…n,B1…m)
LCSS(A1…n,B1…m-1)
LCSS(A1…n-1,B1…m)
LCSS(A1…n-1,B1…m-1)
LCSS(A1…n-2,B1…m)
LCSS(A1…n-2,B1…m-1)

Recursion


LCSS(A1…n,B1…m)
LCSS(A1…n,B1…m-1)
LCSS(A1…n-1,B1…m)
LCSS(A1…n-1,B1…m-1)
LCSS(A1…n-2,B1…m)
LCSS(A1…n-2,B1…m-1)

Recursion



Key Point: Subproblem reuse

Only ever see LCSS($A_1A_2...A_k$, $B_1B_2...B_\ell$)

Base Case

Our recursion also needs a base case.

Base Case

Our recursion also needs a base case.

In this case we have:

LCSS(Ø,$B_1B_2...B_m$) = LCSS($A_1A_2...A_n$, Ø) = 0.

Algorithm

```
LCSS(A1A2…An,B1B2…Bm)
  Initialize Array T[0…n,0…m]
    \\ T[i,j] will store LCSS(A1A2…Ai,B1B2…Bj)
  For i = 0 to n
    For j = 0 to m
      If (i = 0) OR (j = 0)
        T[i,j] ← 0
      Else If Ai = Bj
        T[i,j] ← max(T[i-1,j],T[i,j-1],T[i-1,j-1]+1)
      Else
        T[i,j] ← max(T[i-1,j],T[i,j-1])
  Return T[n,m]
```

Algorithm

```
LCSS(A_1A_2…A_n,B_1B_2…B_m)
  Initialize Array T[0…n,0…m]
    \\ T[i,j] will store LCSS(A_1A_2…A_i,B_1B_2…B_j)
  For i = 0 to n
    For j = 0 to m
      If (i = 0) OR (j = 0)
        T[i,j] ← 0
      Else If A_i = B_j
        T[i,j] ← max(T[i-1,j],T[i,j-1],T[i-1,j-1]+1)
      Else
        T[i,j] ← max(T[i-1,j],T[i,j-1])
  Return T[n,m]
```

O(nm) iterations

Algorithm

$\texttt{LCSS}(A_1A_2\ldots A_n, B_1B_2\ldots B_m)$

  Initialize Array T[0…n,0…m]

    \\ T[i,j] will store $\texttt{LCSS}(A_1A_2\ldots A_i, B_1B_2\ldots B_j)$

  For i = 0 to n

    For j = 0 to m

O(nm) iterations

      If (i = 0) OR (j = 0)

        T[i,j] ← 0

      Else If $A_i = B_j$

        T[i,j] ← max(T[i-1,j],T[i,j-1],T[i-1,j-1]+1)

      Else

        T[i,j] ← max(T[i-1,j],T[i,j-1])

O(1)

  Return T[n,m]

Example

	∅	A	A B	A B A	A B A C	A B A C A
∅						
A						
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0					
A						
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0				
A						
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0			
A						
AB						
ABC						
ABCB						
ABCBA						

Example

	Ø	A	AB	ABA	ABAC	ABACA
Ø	0	0	0	0		
A						
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	
A						
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A						
AB						
ABC						
ABCB						
ABCBA						

Example

	Ø	A	AB	ABA	ABAC	ABACA
Ø	0	0	0	0	0	0
A	0					
AB						
ABC						
ABCB						
ABCBA						

Example

	Ø	A	AB	ABA	ABAC	ABACA
Ø	0	0	0	0	0	0
A	0	1				
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1			
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1		
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB						
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0					
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1				
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2			
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2		
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC						
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0					
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1				
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2			
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2		
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB						
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0					
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1				
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2			
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2		
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	
ABCBA						

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA						

Example

	Ø	A	AB	ABA	ABAC	ABACA
Ø	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0					

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1				

Example

	Ø	A	AB	ABA	ABAC	ABACA
Ø	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2			

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3		

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

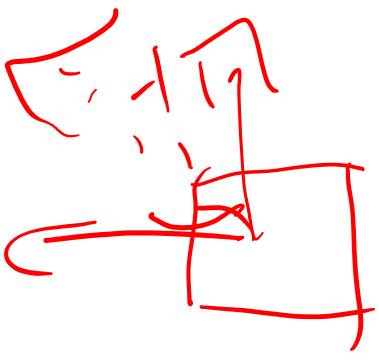
	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4



Example

	Ø	A	AB	ABA	ABAC	ABACA
Ø	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

Example

	∅	A	AB	ABA	ABAC	ABACA
∅	0	0	0	0	0	0
A	0	1	1	1	1	1
AB	0	1	2	2	2	2
ABC	0	1	2	2	3	3
ABCB	0	1	2	2	3	3
ABCBA	0	1	2	3	3	4

String:

ABCA

Proof of Correctness

Prove by induction that each value assigned to T[i,j] is the correct value for LCSS($A_1 A_2 \ldots A_i, B_1 B_2 \ldots B_j$).

Proof of Correctness

Prove by induction that each value assigned to T[i,j] is the correct value for LCSS($A_1A_2\ldots A_i, B_1B_2\ldots B_j$).

Base Case: When i or j is 0 we assign 0.

Proof of Correctness

Prove by induction that each value assigned to T[i,j] is the correct value for LCSS($A_1 A_2 \ldots A_i, B_1 B_2 \ldots B_j$).

Base Case: When i or j is 0 we assign 0.

Inductive Step: Assuming that previous values are assigned correctly, T[i,j] gets correct value because of recursion for LCSS and inductive hypothesis (and that we have previously filled in T[i-1,j], T[i,j-1] and T[i-1,j-1]).

Notes about DP

- General Correct Proof Outline:
  - Prove by induction that each table entry is filled out correctly
  - Use base-case and recursion

Notes about DP

- General Correct Proof Outline:
  - Prove by induction that each table entry is filled out correctly
  - Use base-case and recursion
- Runtime of DP:
  - Usually [Number of subproblems]x[Time per subproblem]