

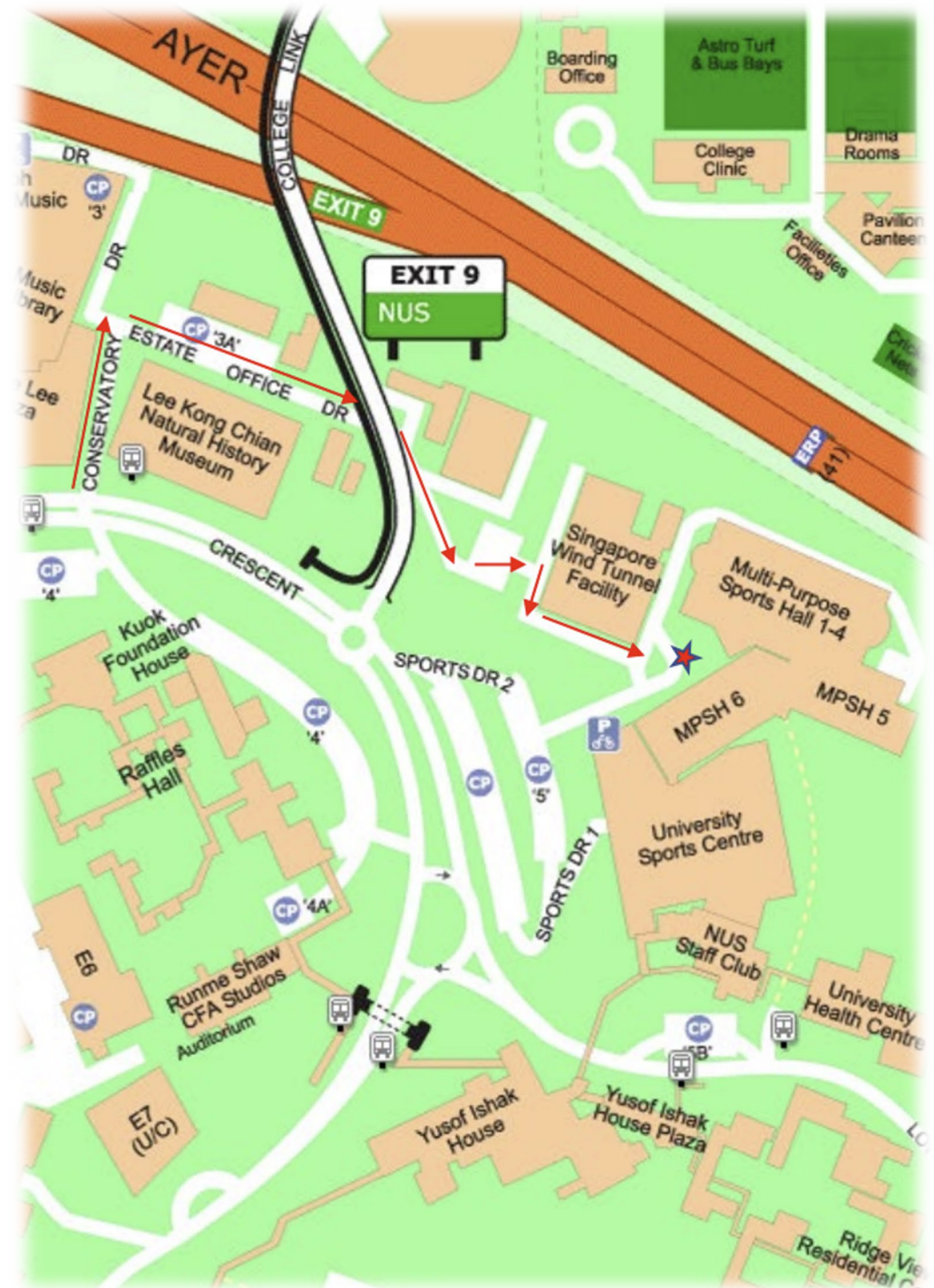
Midterm exam information

- Venue: MPSH5
- Date and Time: Sat, 04 Oct 25 (wk7), 02:00-04:00pm (120 mins)
- Hall entry and exit time: 01:30pm and 04:30pm
- Mode of assessment: Hardcopy (Pen and Paper)
- Scope: **Lectures & tutorials 1–6.**
- Open book, but no electronic devices.
- Allow calculators, but not the programmable ones.
- Bring 2B pencils and erasers.



Write your MCQ answers in the answer box below using (2B) pencil:

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
B	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
C	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
D	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
E	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>



CS3230 – Design and Analysis of Algorithms
(S1 AY2025/26)

Lecture 6: Dynamic Programming

Fibonacci numbers

- **Recall:** Two approaches to compute Fibonacci numbers.
 - Iterative algorithm.
 - Recursive algorithm.

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - $prev2 = 0$
 - $prev1 = 1$
 - for $i = 2$ to n
 - $temp = prev1$
 - $prev1 = prev1 + prev2$
 - $prev2 = temp$
 - return $prev1$

Fibonacci numbers

- **Recall:** Two approaches to compute Fibonacci numbers.
 - Iterative algorithm.
 - Recursive algorithm.

$\Omega(2^{n/2})$ time

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

What is the source of inefficiency?

$O(n)$ time

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - prev2 = 0
 - prev1 = 1
 - for $i = 2$ to n
 - temp = prev1
 - prev1 = prev1+prev2
 - prev2 = temp
 - return prev1

Recursion tree

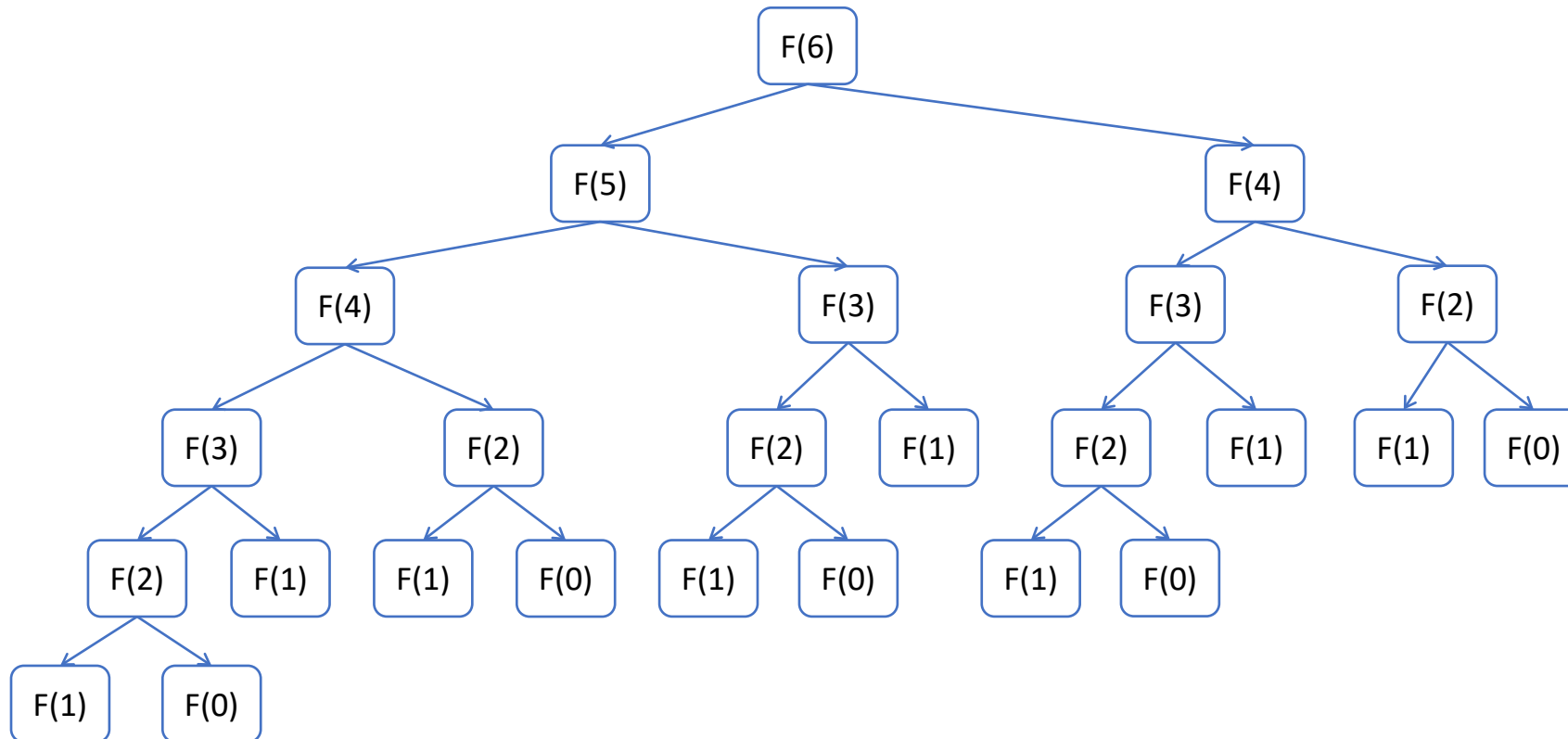
$\Omega(2^{n/2})$ time

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

What is the source of inefficiency?

Overlapping subproblems.



Memoization

$\Omega(2^{n/2})$ time

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

Fib(n)

- If $n \leq 1$, return n .
- Else
 - If $A[n - 1] = \perp$, $A[n - 1] \leftarrow$ **Fib**($n - 1$).
 - If $A[n - 2] = \perp$, $A[n - 2] \leftarrow$ **Fib**($n - 2$).
- return $A[n - 1] + A[n - 2]$.

A recursive call is invoked only if it has not been invoked yet. ←

$A[0..n - 1]$ is initialized as $[\perp, \dots, \perp]$

Memoization

$\Omega(2^{n/2})$ time

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

Fib(n)

- If $n \leq 1$, return n .
- Else
 - If $A[n - 1] = \perp$, $A[n - 1] \leftarrow$ **Fib**($n - 1$).
 - If $A[n - 2] = \perp$, $A[n - 2] \leftarrow$ **Fib**($n - 2$).
- return $A[n - 1] + A[n - 2]$.

$O(n)$ time

$A[0..n - 1]$ is initialized as $[\perp, \dots, \perp]$

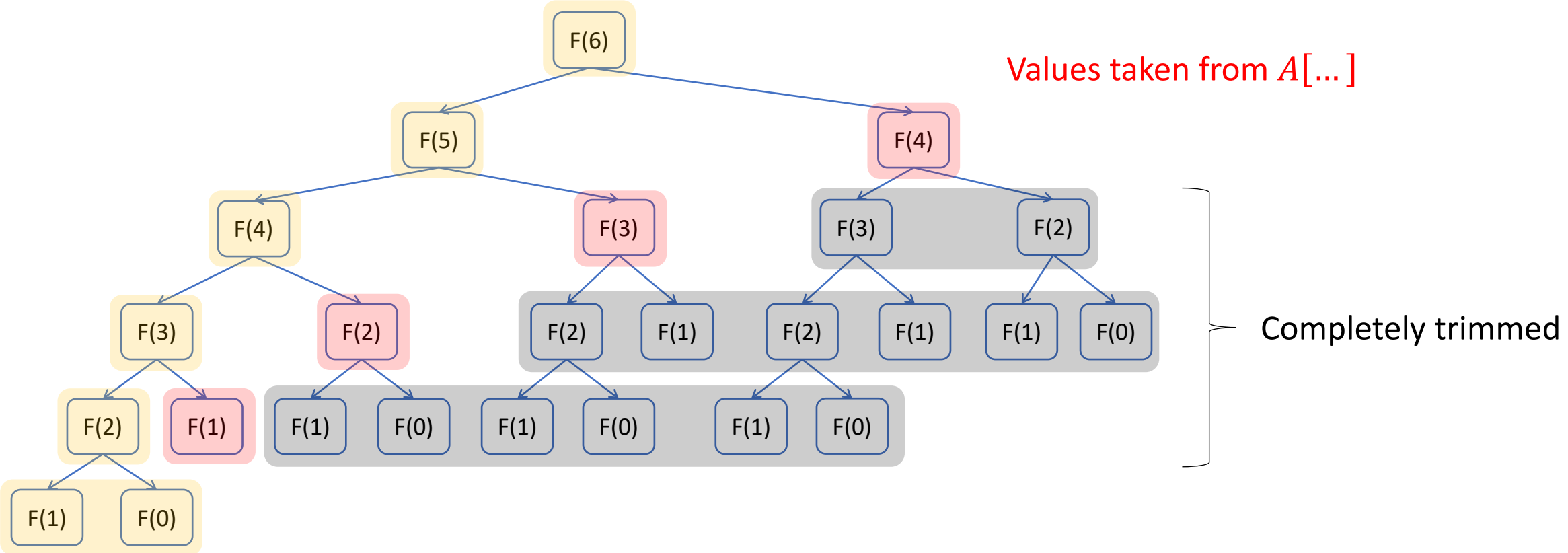
A recursive call is invoked only if it has not been invoked yet.

Each of **Fib**(n), **Fib**($n - 1$), ..., **Fib**(1), **Fib**(0) is invoked exactly once.

Memoization

Fib(n)

- If $n \leq 1$, return n .
- Else
 - If $A[n - 1] = \perp$, $A[n - 1] \leftarrow \mathbf{Fib}(n - 1)$.
 - If $A[n - 2] = \perp$, $A[n - 2] \leftarrow \mathbf{Fib}(n - 2)$.
 - return $A[n - 1] + A[n - 2]$.



Dynamic programming

Divide and conquer:

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively.
3. Combine the subproblem solutions to get the solution of the full problem.



Dynamic programming:

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively with **memoization**.
3. Combine the subproblem solutions to get the solution of the full problem.

Dynamic programming

Divide and conquer:

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively.
3. Combine the subproblem solutions to get the solution of the full problem.



Dynamic programming:

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively with **memoization**.
3. Combine the subproblem solutions to get the solution of the full problem.

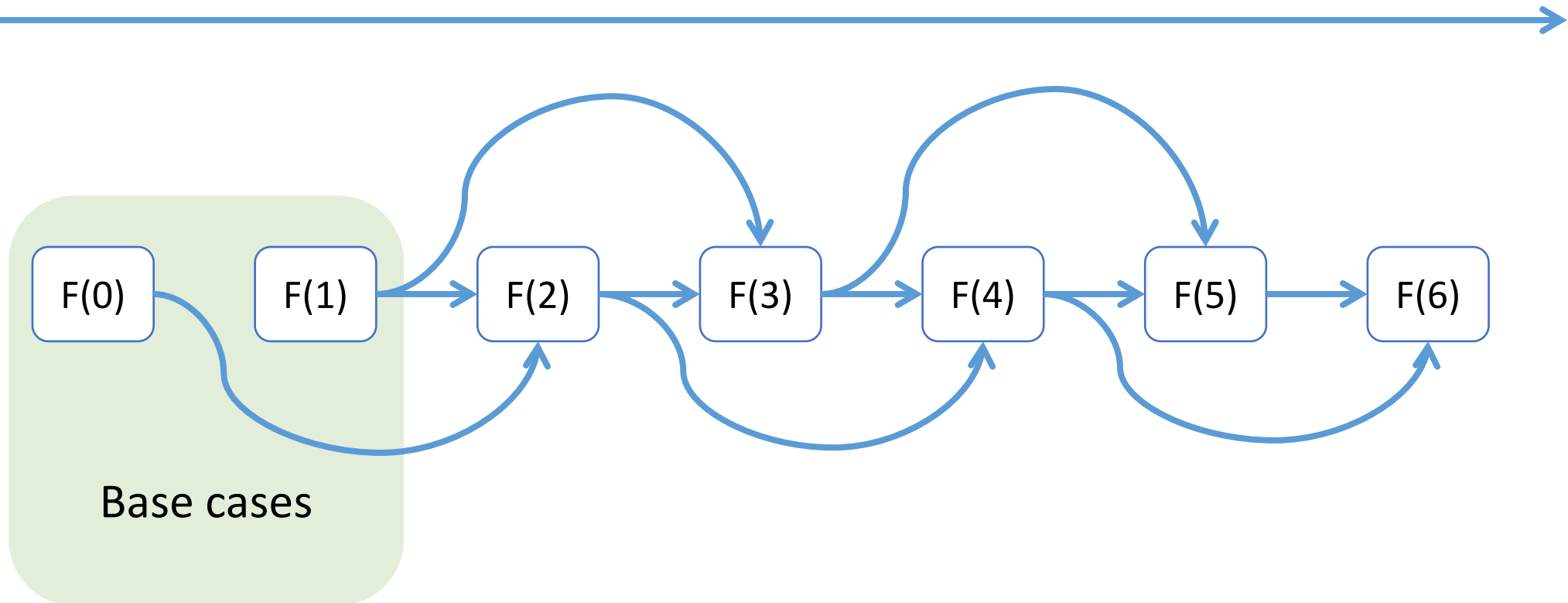
overlapping



Alternatively, a **bottom-up** approach can be used.

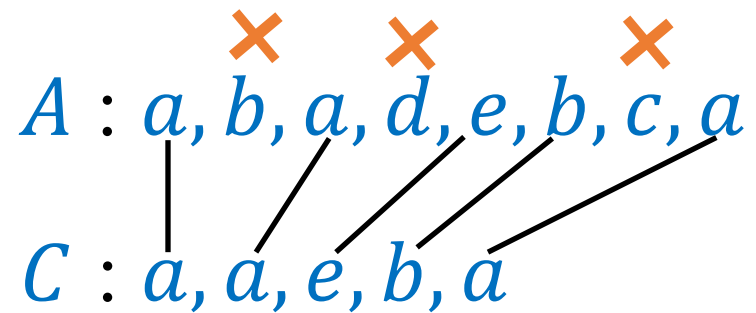
Bottom-up approach

- Just compute the values in the array $A[\dots]$ in this order.



Subsequence

- Let $A = (a_1, a_2, \dots, a_n)$ be a sequence.
- A sequence C is a **subsequence** of A if C can be obtained by removing elements from A .



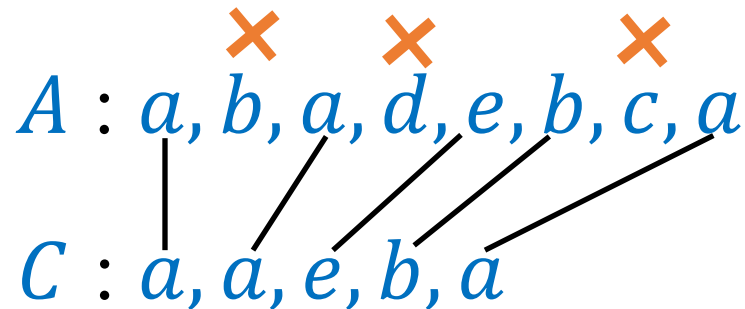
Subsequence

$A = (a_1, a_2, \dots, a_n)$ is seen as an **array** $A[1..n]$.

- $A[i] = a_i$

- Let $A = (a_1, a_2, \dots, a_n)$ be a sequence.
- A sequence C is a **subsequence** of A if C can be obtained by removing elements from A .

A sequence is also a subsequence of itself.



Longest common subsequence

- **Input:** two arrays $A[1..n]$ and $B[1..m]$.
- **Output:** a longest common subsequence C .

C is a subsequence of both A and B .



Longest common subsequence

- **Input:** two arrays $A[1..n]$ and $B[1..m]$.
- **Output:** a longest common subsequence C .

C is a subsequence of both A and B .

$A : a a s b d e s b$
 $B : a c b s d c d e b$

$C : a b d e b$

There can be more than one longest common subsequence.

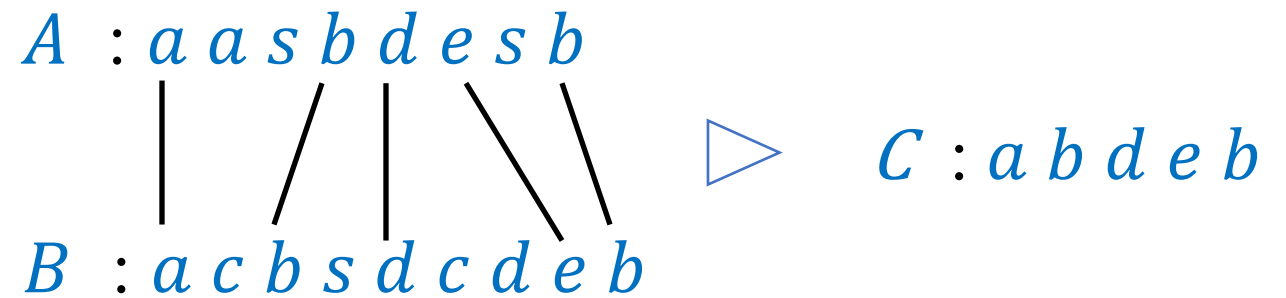
Longest common subsequence

- **Input:** two arrays $A[1..n]$ and $B[1..m]$.
- **Output:** a longest common subsequence C .

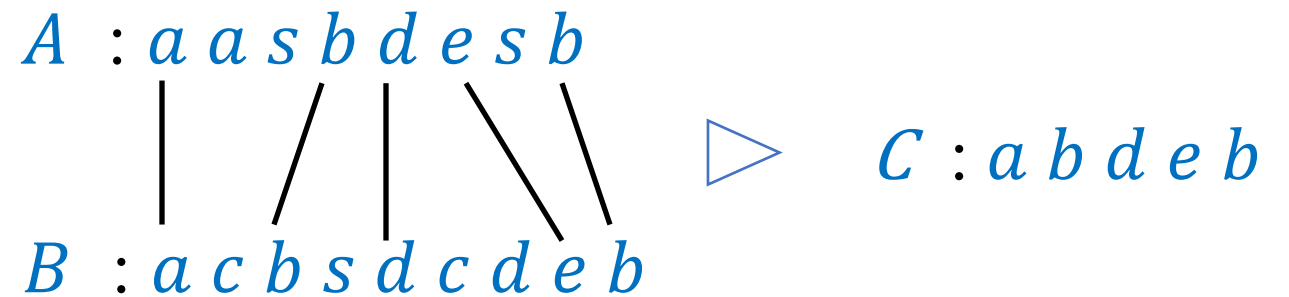
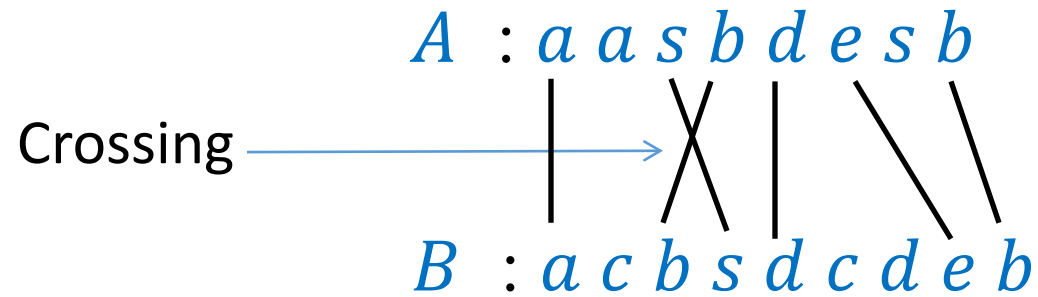
C is a subsequence of both A and B .

- **Applications:**

- Data comparison.
- Computational linguistics.
- Bioinformatics.
- ...



Longest common subsequence



Equivalent definition: Find a largest non-crossing matching between A and B .

Longest common subsequence

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

Longest common subsequence



First attempt

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

Longest common subsequence



- **First attempt:** Check all possible subsequences of A to see if it is also a subsequence of B , and then output the longest one.

First attempt

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

Longest common subsequence



2^n subsequences

- **First attempt:** Check all possible subsequences of A to see if it is also a subsequence of B , and then output the longest one.

$O(m)$ time

Time complexity: $O(m \cdot 2^n)$


Recursive approach

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

$\text{LCS}(n, m)$



- **Recursive solution:**

- Compute $\text{LCS}(i, j)$ for all $i \in [n]$ and $j \in [m]$ recursively.

$\text{LCS}(i, j)$: a longest common subsequence of $A[1..i]$ and $B[1..j]$.

Recursive approach

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

$\text{LCS}(n, m)$

- **Recursive solution:**

- Compute $\text{LCS}(i, j)$ for all $i \in [n]$ and $j \in [m]$ recursively.

$\text{LCS}(i, j)$: a longest common subsequence of $A[1..i]$ and $B[1..j]$.



- There can be more than one solution.
- Computing any one of them is fine.

Base case

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

$\text{LCS}(n, m)$

- **Recursive solution:**

- Compute $\text{LCS}(i, j)$ for all $i \in [n]$ and $j \in [m]$ recursively.

$\text{LCS}(i, j)$: a longest common subsequence of $A[1..i]$ and $B[1..j]$.

- **Base case:**

- $\text{LCS}(i, 0) = \emptyset$ for all i .
- $\text{LCS}(0, j) = \emptyset$ for all j .

$A[1..0] = \emptyset$ and $B[1..0] = \emptyset$

Inductive step

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute $\mathbf{LCS}(n, m)$ recursively?

Suppose we have already computed $\mathbf{LCS}(n - 1, m)$, $\mathbf{LCS}(n, m - 1)$, and $\mathbf{LCS}(n - 1, m - 1)$.

$\mathbf{LCS}(i, j)$: a longest common subsequence of $A[1..i]$ and $B[1..j]$.

Inductive step

Observation:

- If $a_n \neq b_m$, we can take $\mathbf{LCS}(n, m)$ as the longer one of $\mathbf{LCS}(n - 1, m)$ and $\mathbf{LCS}(n, m - 1)$.

Inductive step

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.

Proof:

A common subsequence
of $A[1..n]$ and $B[1..m]$

\longleftrightarrow
if and only if

A common subsequence of $A[1..n - 1]$ and $B[1..m]$

or

A common subsequence of $A[1..n]$ and $B[1..m - 1]$

Inductive step

$A : a_1, a_2, \dots, a_n$
Crossing
 $B : b_1, b_2, \dots, b_m$

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.

In any common subsequence of $A[1..n]$ and $B[1..m]$, a_n cannot be matched with b_m .

In any common subsequence of $A[1..n]$ and $B[1..m]$, at least one of a_n and b_m is not matched.

A common subsequence of
 $A[1..n - 1]$ and $B[1..m]$


A common subsequence of
 $A[1..n]$ and $B[1..m - 1]$

Inductive step

Observation:

- If $a_n \neq b_m$, we can take $\mathbf{LCS}(n, m)$ as the longer one of $\mathbf{LCS}(n - 1, m)$ and $\mathbf{LCS}(n, m - 1)$.
- If $a_n = b_m = x$, we can take $\mathbf{LCS}(n, m)$ as $\mathbf{LCS}(n - 1, m - 1) \circ x$.

Concatenation



Inductive step

$$\begin{array}{l} A : a_1, a_2, \dots, a_n \\ | \\ B : b_1, b_2, \dots, b_m \end{array}$$

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.
- If $a_n = b_m = x$, we can take $\text{LCS}(n, m)$ as $\text{LCS}(n - 1, m - 1) \circ x$.

Claim: There exists an LCS of $A[1..n]$ and $B[1..m]$ such that a_n is matched with b_m .

Inductive step

$$\begin{array}{c} A : a_1, a_2, \dots, a_n \\ | \\ B : b_1, b_2, \dots, b_m \end{array}$$

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.
- If $a_n = b_m = x$, we can take $\text{LCS}(n, m)$ as $\text{LCS}(n - 1, m - 1) \circ x$.

Claim: There exists an LCS of $A[1..n]$ and $B[1..m]$ such that a_n is matched with b_m .

$$\begin{array}{c} A : a_1, a_2, \dots, a_n \\ \text{---} \\ B : b_1, b_2, \dots, b_m \end{array}$$



$$\begin{array}{c} A : a_1, a_2, \dots, a_n \\ | \\ B : b_1, b_2, \dots, b_m \end{array}$$

$$\begin{array}{c} A : a_1, a_2, \dots, a_n \\ \text{---} \\ B : b_1, b_2, \dots, b_m \end{array}$$



$$\begin{array}{c} A : a_1, a_2, \dots, a_n \\ | \\ B : b_1, b_2, \dots, b_m \end{array}$$

$$\begin{array}{c} A : a_1, a_2, \dots, a_n \\ \text{---} \\ B : b_1, b_2, \dots, b_m \end{array}$$



$$\begin{array}{c} A : a_1, a_2, \dots, a_n \\ | \\ B : b_1, b_2, \dots, b_m \end{array}$$

Proof of the claim

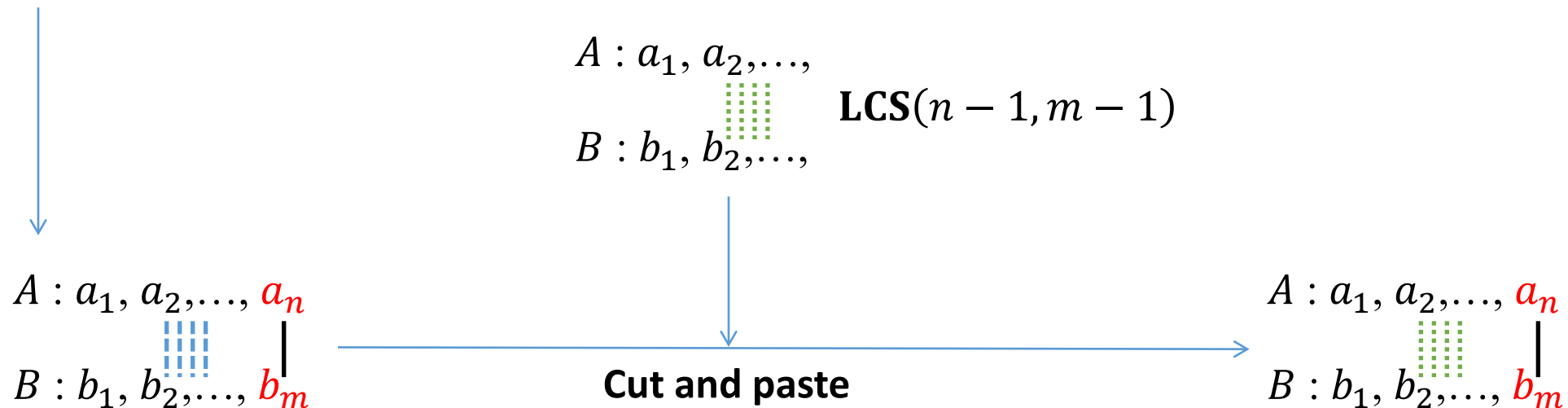
Cut and paste

$$\begin{array}{l} A : a_1, a_2, \dots, a_n \\ | \\ B : b_1, b_2, \dots, b_m \end{array}$$

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.
- If $a_n = b_m = x$, we can take $\text{LCS}(n, m)$ as $\text{LCS}(n - 1, m - 1) \circ x$. ← Now we use the claim to prove our observation using **cut and paste**.

Claim: There exists an LCS of $A[1..n]$ and $B[1..m]$ such that a_n is matched with b_m .



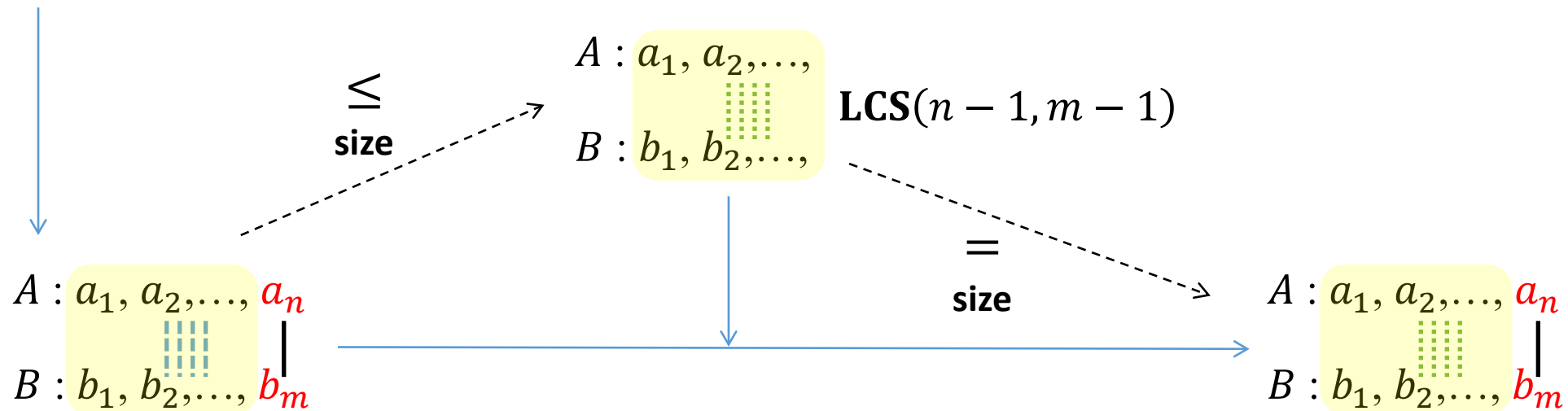
Cut and paste

$$\begin{array}{l} A : a_1, a_2, \dots, a_n \\ | \\ B : b_1, b_2, \dots, b_m \end{array}$$

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.
- If $a_n = b_m = x$, we can take $\text{LCS}(n, m)$ as $\text{LCS}(n - 1, m - 1) \circ x$. ← Now we use the claim to prove our observation using **cut and paste**.

Claim: There exists an LCS of $A[1..n]$ and $B[1..m]$ such that a_n is matched with b_m .



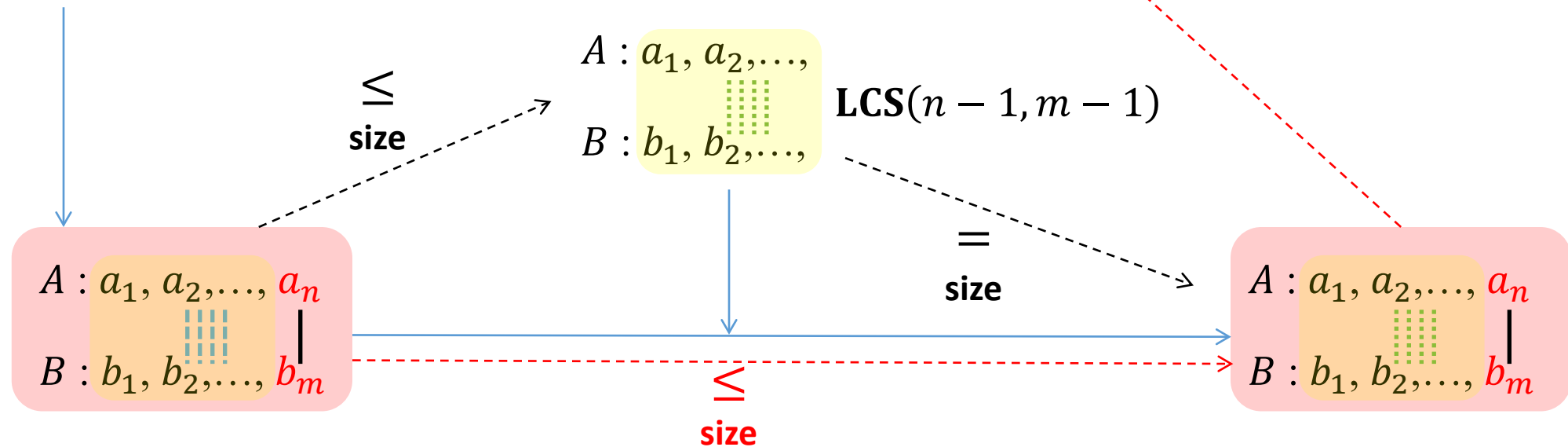
Cut and paste

$$\begin{array}{l}
 A : a_1, a_2, \dots, a_n \\
 | \\
 B : b_1, b_2, \dots, b_m
 \end{array}$$

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.
- If $a_n = b_m = x$, we can take $\text{LCS}(n, m)$ as $\text{LCS}(n - 1, m - 1) \circ x$. ← Now we use the claim to prove our observation using **cut and paste**.

Claim: There exists an LCS of $A[1..n]$ and $B[1..m]$ such that a_n is matched with b_m .



Key idea: Optimal substructures

Observation:

- If $a_n \neq b_m$, we can take $\text{LCS}(n, m)$ as the longer one of $\text{LCS}(n - 1, m)$ and $\text{LCS}(n, m - 1)$.
- If $a_n = b_m = x$, we can take $\text{LCS}(n, m)$ as $\text{LCS}(n - 1, m - 1) \circ x$.

Optimal substructures

An optimal solution to a problem can be constructed from optimal solutions to its subproblems.

Finding LCS recursively

$\text{LCS}(n, m)$

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

Base case:

- $\text{LCS}(i, 0) = \emptyset$ for all i .
- $\text{LCS}(0, j) = \emptyset$ for all j .

$m = 6$	\emptyset						
5	\emptyset						
4	\emptyset						
3	\emptyset						
2	\emptyset						
1	\emptyset						
0	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
	0	1	2	3	4	5	$n = 6$

Finding LCS recursively

LCS(n, m)

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

Base case:

- $\text{LCS}(i, 0) = \emptyset$ for all i .
- $\text{LCS}(0, j) = \emptyset$ for all j .

Inductive step:

- If $a_i \neq b_j$, $\text{LCS}(i, j)$ = the longer one of $\text{LCS}(i - 1, j)$ and $\text{LCS}(i, j - 1)$.
- If $a_i = b_j = x$, $\text{LCS}(i, j)$ = $\text{LCS}(i - 1, j - 1) \circ x$.

$m = 6$	\emptyset				...	$\text{LCS}(5,6)$	$\text{LCS}(6,6)$
5	\emptyset				...	$\text{LCS}(5,5)$	$\text{LCS}(6,5)$
4	\emptyset			
3	\emptyset						
2	\emptyset						
1	\emptyset						
0	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
	0	1	2	3	4	5	$n = 6$

Red arrows indicate dependencies: a horizontal arrow from $\text{LCS}(5,6)$ to $\text{LCS}(6,6)$, a diagonal arrow from $\text{LCS}(5,5)$ to $\text{LCS}(6,6)$, and a vertical arrow from $\text{LCS}(6,5)$ to $\text{LCS}(6,6)$.

Finding LCS recursively

LCS(n, m)

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

Base case:

- $\text{LCS}(i, 0) = \emptyset$ for all i .
- $\text{LCS}(0, j) = \emptyset$ for all j .

Inductive step:

- If $a_i \neq b_j$, $\text{LCS}(i, j) =$ the longer one of $\text{LCS}(i - 1, j)$ and $\text{LCS}(i, j - 1)$.
- If $a_i = b_j = x$, $\text{LCS}(i, j) = \text{LCS}(i - 1, j - 1) \circ x$.

Bottom-up approach:

$m = 6$	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	...	\rightarrow LCS(5,6)	\rightarrow LCS(6,6)
5	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	...	\rightarrow LCS(5,5)	\rightarrow LCS(6,5)
4	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	...	\rightarrow ...	\rightarrow ...
3	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow
2	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow
1	\emptyset	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow
0	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
	0	1	2	3	4	5	$n = 6$	

Finding LCS recursively

LCS(n, m)

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

Base case:

- $\text{LCS}(i, 0) = \emptyset$ for all i .
- $\text{LCS}(0, j) = \emptyset$ for all j .

Inductive step:

- If $a_i \neq b_j$, $\text{LCS}(i, j) =$ the longer one of $\text{LCS}(i - 1, j)$ and $\text{LCS}(i, j - 1)$.
- If $a_i = b_j = x$, $\text{LCS}(i, j) = \text{LCS}(i - 1, j - 1) \circ x$.

Bottom-up approach:

$m = 6$	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	\dots	\rightarrow	$\text{LCS}(5,6)$	\rightarrow	$\text{LCS}(6,6)$
5	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	\dots	\rightarrow	$\text{LCS}(5,5)$	\rightarrow	$\text{LCS}(6,5)$
4	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	\dots	\rightarrow	\dots	\rightarrow	\dots
3	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow
2	\emptyset	$\leftarrow \rightarrow$	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow
1	\emptyset	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow	\rightarrow
0	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
	0	1	2	3	4	5	$n = 6$			

Alternatively, an LCS can be computed recursively with **memorization**.

Finding LCS recursively

$A : a_1, a_2, \dots, a_n$

$B : b_1, b_2, \dots, b_m$

- **Question:** How to compute an LCS of $A[1..n]$ and $B[1..m]$ efficiently?

- **Next:** Concrete algorithm description + time and space analysis.



For simplicity, we focus on computing only the **length of LCS**.

Length of LCS

Base case:

- $\mathbf{LCS}(i, 0) = \emptyset$ for all i .
- $\mathbf{LCS}(0, j) = \emptyset$ for all j .

Inductive step:

- If $a_i \neq b_j$, $\mathbf{LCS}(i, j) =$ the longer one of $\mathbf{LCS}(i - 1, j)$ and $\mathbf{LCS}(i, j - 1)$.
- If $a_i = b_j = x$, $\mathbf{LCS}(i, j) = \mathbf{LCS}(i - 1, j - 1) \circ x$.

- Let $L(i, j)$ be the **length** of an LCS of $A[1..n]$ and $B[1..m]$.

Length of LCS

Base case:

- $\mathbf{LCS}(i, 0) = \emptyset$ for all i .
- $\mathbf{LCS}(0, j) = \emptyset$ for all j .

Inductive step:

- If $a_i \neq b_j$, $\mathbf{LCS}(i, j) =$ the longer one of $\mathbf{LCS}(i - 1, j)$ and $\mathbf{LCS}(i, j - 1)$.
- If $a_i = b_j = x$, $\mathbf{LCS}(i, j) = \mathbf{LCS}(i - 1, j - 1) \circ x$.

- 
- Let $L(i, j)$ be the **length** of an LCS of $A[1..n]$ and $B[1..m]$.

Base case:

- $L(i, 0) = 0$ for all i .
- $L(0, j) = 0$ for all j .

Length of LCS

Base case:

- $\mathbf{LCS}(i, 0) = \emptyset$ for all i .
- $\mathbf{LCS}(0, j) = \emptyset$ for all j .

Inductive step:

- If $a_i \neq b_j$, $\mathbf{LCS}(i, j) =$ the longer one of $\mathbf{LCS}(i - 1, j)$ and $\mathbf{LCS}(i, j - 1)$.
- If $a_i = b_j = x$, $\mathbf{LCS}(i, j) = \mathbf{LCS}(i - 1, j - 1) \circ x$.

- Let $L(i, j)$ be the **length** of an LCS of $A[1..n]$ and $B[1..m]$.

Base case:

- $L(i, 0) = 0$ for all i .
- $L(0, j) = 0$ for all j .

Inductive step:

- If $a_i \neq b_j$, $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.
- If $a_i = b_j$, $L(i, j) = L(i - 1, j - 1) + 1$.

Length of LCS

$m = 6$	a	0	1	2	2	3	3	4
5	d	0	1	2	2	3	3	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Base case:

- $L(i, 0) = 0$ for all i .
- $L(0, j) = 0$ for all j .

Inductive step:

- If $a_i \neq b_j$, $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.
- If $a_i = b_j$, $L(i, j) = L(i - 1, j - 1) + 1$.

Length of LCS

$m = 6$	a	0	1	2	2	3	3	4
5	d	0	1	2	2	3	3	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Bottom-up approach:

For $(i = 0 \text{ to } n) L(i, 0) \leftarrow 0$

For $(j = 0 \text{ to } m) L(0, j) \leftarrow 0$

For $(j = 1 \text{ to } m)$

 For $(i = 1 \text{ to } n)$

 If $(a_i = b_j)$ then $L(i, j) \leftarrow L(i - 1, j - 1) + 1$

 If $(a_i \neq b_j)$ then $L(i, j) \leftarrow \max\{L(i - 1, j), L(i, j - 1)\}$

Base case:

- $L(i, 0) = 0$ for all i .
- $L(0, j) = 0$ for all j .

Inductive step:

- If $a_i \neq b_j$, $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.
- If $a_i = b_j$, $L(i, j) = L(i - 1, j - 1) + 1$.

Length of LCS

$m = 6$	a	0	1	2	2	3	3	4
5	d	0	1	2	2	3	3	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Bottom-up approach:

For $(i = 0 \text{ to } n) L(i, 0) \leftarrow 0$

For $(j = 0 \text{ to } m) L(0, j) \leftarrow 0$

For $(j = 1 \text{ to } m)$

 For $(i = 1 \text{ to } n)$

 If $(a_i = b_j)$ then $L(i, j) \leftarrow L(i - 1, j - 1) + 1$

 If $(a_i \neq b_j)$ then $L(i, j) \leftarrow \max\{L(i - 1, j), L(i, j - 1)\}$

Base case:

- $L(i, 0) = 0$ for all i .
- $L(0, j) = 0$ for all j .

Inductive step:

- If $a_i \neq b_j$, $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.
- If $a_i = b_j$, $L(i, j) = L(i - 1, j - 1) + 1$.

Length of LCS

$m = 6$	a	0	1	2	2	3	3	4
5	d	0	1	2	2	3	3	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Bottom-up approach:

For $(i = 0 \text{ to } n) L(i, 0) \leftarrow 0$

For $(j = 0 \text{ to } m) L(0, j) \leftarrow 0$

For $(j = 1 \text{ to } m)$

 For $(i = 1 \text{ to } n)$

 If $(a_i = b_j)$ then $L(i, j) \leftarrow L(i - 1, j - 1) + 1$

 If $(a_i \neq b_j)$ then $L(i, j) \leftarrow \max\{L(i - 1, j), L(i, j - 1)\}$

Base case:

- $L(i, 0) = 0$ for all i .
- $L(0, j) = 0$ for all j .

Inductive step:

- If $a_i \neq b_j$, $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.
- If $a_i = b_j$, $L(i, j) = L(i - 1, j - 1) + 1$.

Length of LCS

$m = 6$	a	0	1	2	2	3	3	4
5	d	0	1	2	2	3	3	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Bottom-up approach:

For $(i = 0 \text{ to } n) L(i, 0) \leftarrow 0$

For $(j = 0 \text{ to } m) L(0, j) \leftarrow 0$

For $(j = 1 \text{ to } m)$

 For $(i = 1 \text{ to } n)$

 If $(a_i = b_j)$ then $L(i, j) \leftarrow L(i - 1, j - 1) + 1$

 If $(a_i \neq b_j)$ then $L(i, j) \leftarrow \max\{L(i - 1, j), L(i, j - 1)\}$

Base case:

- $L(i, 0) = 0$ for all i .
- $L(0, j) = 0$ for all j .

Inductive step:

- If $a_i \neq b_j$, $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.
- If $a_i = b_j$, $L(i, j) = L(i - 1, j - 1) + 1$.

Length of LCS

Space complexity: $O(mn)$

$m = 6$	a	0	1	2	2	3	3	4
5	d	0	1	2	2	3	3	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Bottom-up approach:

For $(i = 0 \text{ to } n) L(i, 0) \leftarrow 0$

For $(j = 0 \text{ to } m) L(0, j) \leftarrow 0$

For $(j = 1 \text{ to } m)$
 For $(i = 1 \text{ to } n)$ } mn iterations

If $(a_i = b_j)$ then $L(i, j) \leftarrow L(i - 1, j - 1) + 1$

If $(a_i \neq b_j)$ then $L(i, j) \leftarrow \max\{L(i - 1, j), L(i, j - 1)\}$

$O(1)$ time

Time complexity: $O(mn)$

Question 1 @ VisuAlgo online quiz

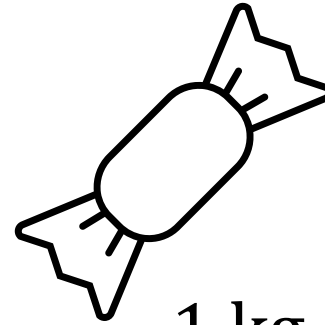
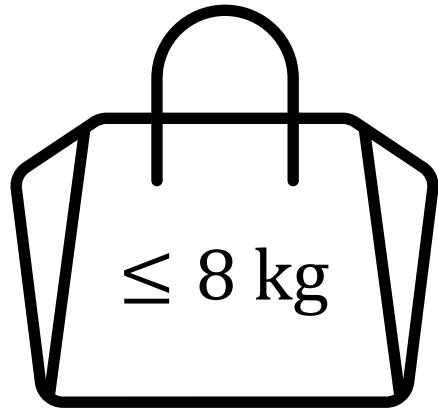
Who is the **Master of Algorithms** pictured below?

- Richard Bellman
- Stephen Cook
- John Hopcroft
- Robert Tarjan

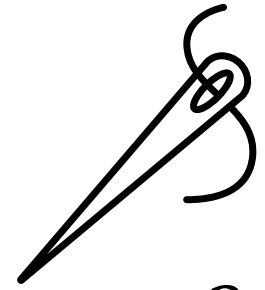


Knapsack problem

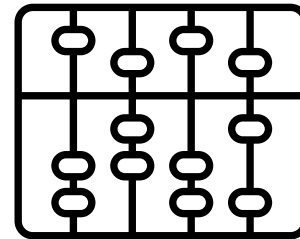
Goal: Pack a set of items into a container with a capacity constraint to maximize the total value.



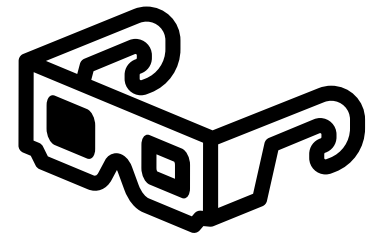
1 kg, \$5



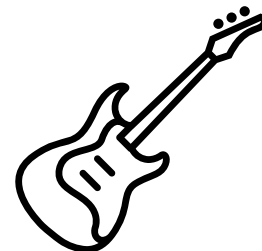
3 kg, \$8



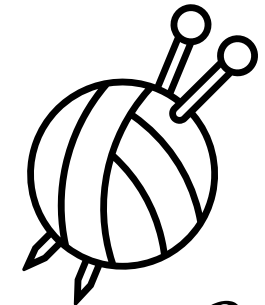
2 kg, \$9



4 kg, \$6



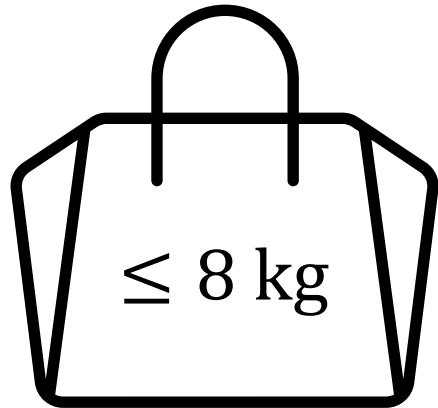
7 kg, \$8



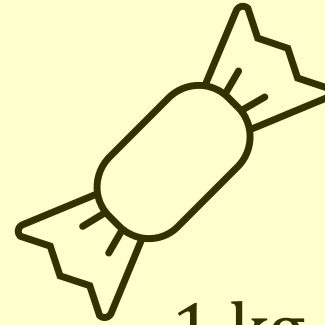
3 kg, \$2

Knapsack problem

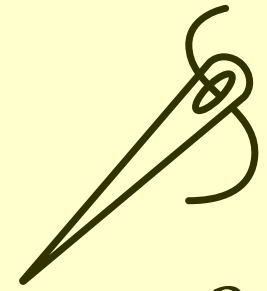
Goal: Pack a set of items into a container with a capacity constraint to maximize the total value.



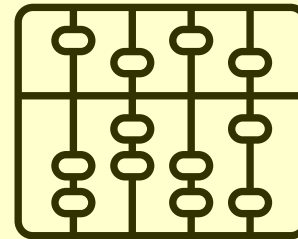
Optimal solution: \$22



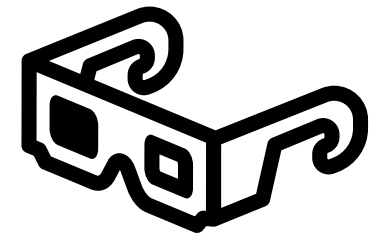
1 kg, \$5



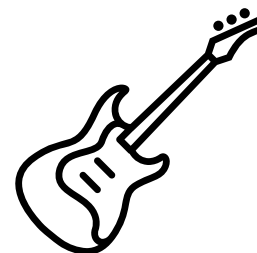
3 kg, \$8



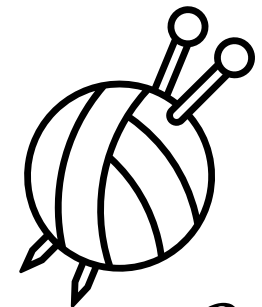
2 kg, \$9



4 kg, \$6



7 kg, \$8



3 kg, \$2

Formal definition

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$. All of them are positive integers.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Formal definition

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

n items

Weight Value Capacity of the bag

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Formal definition

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

n items

Weight Value Capacity of the bag

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

A subset of items Capacity constraint is satisfied Maximizing the total value

Optimal substructures

Are there optimal structures?

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Suppose S is an optimal solution.

Optimal substructures

Are there optimal structures?

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Suppose S is an optimal solution.

Case 1: $n \in S$ \longrightarrow $S \setminus \{n\}$ is an optimal solution to the subproblem:

- $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W - w_n$

Optimal substructures

Are there optimal structures?

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Suppose S is an optimal solution.

Case 1: $n \in S$ \longrightarrow $S \setminus \{n\}$ is an optimal solution to the subproblem:
• $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W - w_n$

Proof: If not, then a solution better than S can be obtained by **cut-and-paste**.

Replacing $S \setminus \{n\}$ with an optimal solution to the **subproblem**.

Optimal substructures

Are there optimal structures?

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Suppose S is an optimal solution.

Case 1: $n \in S$ \longrightarrow $S \setminus \{n\}$ is an optimal solution to the subproblem:
• $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W - w_n$

Case 2: $n \notin S$ \longrightarrow S is an optimal solution to the subproblem:
• $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W$

Optimal substructures

Are there optimal structures?

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Suppose S is an optimal solution.

Case 1: $n \in S$ \longrightarrow $S \setminus \{n\}$ is an optimal solution to the subproblem:
• $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W - w_n$

Case 2: $n \notin S$ \longrightarrow S is an optimal solution to the subproblem:
• $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W$

Proof: If not, then a solution better than S can be obtained by **cut-and-paste**.

Replacing S with an optimal solution to the **subproblem**.

The value an optimal solution

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

$m[n, W]$

$m[i, j]$ = the value of an optimal solution for the instance:

- $(w_1, v_1), (w_2, v_2), \dots, (w_i, v_i), j$.

The first i items

Recursion

$m[i, j]$ = the value of an optimal solution for the instance:

- $(w_1, v_1), (w_2, v_2), \dots, (w_i, v_i), j$.

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Suppose S is an optimal solution.

$$m[n, W] = m[n - 1, W - w_n] + v_n$$

Case 1: $n \in S$



$S \setminus \{n\}$ is an optimal solution to the subproblem:

- $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W - w_n$

Case 2: $n \notin S$



S is an optimal solution to the subproblem:

- $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W$

$$m[n, W] = m[n - 1, W]$$

Recursion

$m[i, j]$ = the value of an optimal solution for the instance:

- $(w_1, v_1), (w_2, v_2), \dots, (w_i, v_i), j$.

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

Suppose S is an optimal solution.

$$m[n, W] = m[n - 1, W - w_n] + v_n$$

Case 1: $n \in S$



$S \setminus \{n\}$ is an optimal solution to the subproblem:

- $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W - w_n$

Case 2: $n \notin S$



S is an optimal solution to the subproblem:

- $(w_1, v_1), (w_2, v_2), \dots, (w_{n-1}, v_{n-1}), W$

$$m[n, W] = m[n - 1, W]$$

$$m[n, W] = \max\{m[n - 1, W - w_n] + v_n, m[n - 1, W]\}$$

Recursion

$m[i, j]$ = the value of an optimal solution for the instance:

- $(w_1, v_1), (w_2, v_2), \dots, (w_i, v_i), j$.

Input: $(w_1, v_1), (w_2, v_2), \dots, (w_n, v_n), W$.

Output: A subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ that maximizes $\sum_{i \in S} v_i$.

$m[n, W]$

For maximization problems, if no feasible solution exists, the solution value is defined as $-\infty$, in line with the convention that the maximum of an empty set is often taken to be $-\infty$.

Negative weight is impossible.

$$m[i, j] = \begin{cases} -\infty & j < 0 \\ 0 & (i = 0) \wedge (j \geq 0) \\ \max\{m[i-1, j-w_i] + v_i, m[i-1, j]\} & \text{otherwise} \end{cases}$$

Can only achieve zero value with zero items.

Dynamic programming

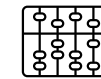
		j									
		< 0	0	1	2	3	4	5	6	7	8
i	0	$-\infty$	0	0	0	0	0	0	0	0	0
	1	$-\infty$									
	2	$-\infty$									
	3	$-\infty$									
	4	$-\infty$									
	5	$-\infty$									
	6	$-\infty$									



$W = 8$



$w_1 = 1, v_1 = 5$



$w_2 = 2, v_2 = 9$



$w_3 = 7, v_3 = 8$



$w_4 = 3, v_4 = 8$



$w_5 = 4, v_5 = 6$



$w_6 = 3, v_6 = 2$

Negative weight is impossible.

$$m[i, j] = \begin{cases} -\infty & j < 0 \\ 0 & (i = 0) \wedge (j \geq 0) \\ \max\{m[i - 1, j - w_i] + v_i, m[i - 1, j]\} & \text{otherwise} \end{cases}$$

Can only achieve zero value with zero items.

Dynamic programming

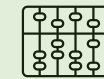
		j									
		< 0	0	1	2	3	4	5	6	7	8
i	0	$-\infty$	0	0	0	0	0	0	0	0	0
	1	$-\infty$	0	5	5	5	5	5	5	5	5
	2	$-\infty$	0	5	9	14	-----	-----	-----	-----	-----
	3	$-\infty$	←-----	-----	-----	-----	-----	-----	-----	-----	-----
	4	$-\infty$									
	5	$-\infty$									
	6	$-\infty$									



$W = 8$



$w_1 = 1, v_1 = 5$



$w_2 = 2, v_2 = 9$



$w_3 = 7, v_3 = 8$



$w_4 = 3, v_4 = 8$



$w_5 = 4, v_5 = 6$



$w_6 = 3, v_6 = 2$

$$m[i, j] = \begin{cases} -\infty & j < 0 \\ 0 & (i = 0) \wedge (j \geq 0) \\ \max\{m[i - 1, j - w_i] + v_i, m[i - 1, j]\} & \text{otherwise} \end{cases}$$

Dynamic programming



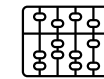
$W = 8$

		j									
		W									
		< 0	0	1	2	3	4	5	6	7	8
i	0	$-\infty$	0	0	0	0	0	0	0	0	0
	1	$-\infty$	0	5	5	5	5	5	5	5	5
	2	$-\infty$	0	5	9	14	14	14	14	14	14
	3	$-\infty$	0	5	9	14	14	14	14	14	14
	4	$-\infty$	0	5	9	14	14	17	22	22	22
	5	$-\infty$	0	5	9	14	14	17	22	22	22
	n 6	$-\infty$	0	5	9	14	14	17	22	22	22

Optimal solution: \$22



$w_1 = 1, v_1 = 5$



$w_2 = 2, v_2 = 9$



$w_3 = 7, v_3 = 8$



$w_4 = 3, v_4 = 8$



$w_5 = 4, v_5 = 6$



$w_6 = 3, v_6 = 2$

$$m[i, j] = \begin{cases} -\infty & j < 0 \\ 0 & (i = 0) \wedge (j \geq 0) \\ \max\{m[i - 1, j - w_i] + v_i, m[i - 1, j]\} & \text{otherwise} \end{cases}$$

Time complexity: $O(nW)$

Change-making problem

Goal: Find the minimum number of coins of denominations d_1, d_2, \dots, d_k that add up to n cents.

Example: If the denominations are $\{1, 10, 25\}$, then the optimal solution for $n = 30$ is $10 + 10 + 10$ (3 coins).

Change-making problem

Goal: Find the minimum number of coins of denominations d_1, d_2, \dots, d_k that add up to n cents.

Example: If the denominations are $\{1, 10, 25\}$, then the optimal solution for $n = 30$ is $10 + 10 + 10$ (3 coins).

Let $M[n]$ be the fewest number of coins needed to change n cents.

Question 2 @ VisuAlgo online quiz

Goal: Find the minimum number of coins of denominations d_1, d_2, \dots, d_k that add up to n cents.

Example: If the denominations are $\{1, 10, 25\}$, then the optimal solution for $n = 30$ is $10 + 10 + 10$ (3 coins).

Let $M[n]$ be the fewest number of coins needed to change n cents.

Question: Which of the following is true?

- $M[n] = 1 + \min_{i \in [k]} M[n - d_i]$
- $M[n] = 1 + \max_{i \in [k]} M[n - d_i]$
- $M[n] = 1 + M[n - d_k]$
- $M[n] = \underline{d_k} + \underline{M[n - d_k]}$

Additional exercises

(We will cover these only if time permits.)

Exercise 1

Space complexity: $O(mn)$

$m = 6$	a	0	1	2	2	3	3	3
5	d	0	1	2	2	3	2	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Bottom-up approach:

For $(i = 0 \text{ to } n) L(i, 0) \leftarrow 0$

For $(j = 0 \text{ to } m) L(0, j) \leftarrow 0$

For $(j = 1 \text{ to } m)$

For $(i = 1 \text{ to } n)$

mn iterations

If $(a_i = b_j)$ then $L(i, j) \leftarrow L(i - 1, j - 1) + 1$

If $(a_i \neq b_j)$ then $L(i, j) \leftarrow \max\{L(i - 1, j), L(i, j - 1)\}$

$O(1)$ time

Time complexity: $O(mn)$

Exercise 1: Improve the space complexity to $O(\min\{m, n\})$.

Exercise 2

Space complexity: $O(mn)$

$m = 6$	a	0	1	2	2	3	3	3
5	d	0	1	2	2	3	2	3
4	c	0	1	1	2	2	2	3
3	a	0	1	1	1	2	2	3
2	d	0	0	1	1	2	2	2
1	d	0	0	1	1	1	1	1
0		0	0	0	0	0	0	0
			a	d	c	d	s	a
		0	1	2	3	4	5	$n = 6$

Bottom-up approach:

For $(i = 0 \text{ to } n) L(i, 0) \leftarrow 0$

For $(j = 0 \text{ to } m) L(0, j) \leftarrow 0$

For $(j = 1 \text{ to } m)$
 For $(i = 1 \text{ to } n)$ } mn iterations

If $(a_i = b_j)$ then $L(i, j) \leftarrow L(i - 1, j - 1) + 1$

If $(a_i \neq b_j)$ then $L(i, j) \leftarrow \max\{L(i - 1, j), L(i, j - 1)\}$

$O(1)$ time

Time complexity: $O(mn)$

Exercise 2: Compute an LCS of $A[1..n]$ and $B[1..m]$ in $O(mn)$ time.

↑
not just the length

Longest palindrome subsequence

- A palindrome is a string that reads the same forwards and backwards.
 - **Examples:** *racecar*, *civic*, *a*.

Exercise 3

- A palindrome is a string that reads the same forwards and backwards.
 - **Examples:** *racecar*, *civic*, *a*.

Exercise 3: Using the LCS algorithm as a blackbox to design an $O(n^2)$ –time algorithm to find a longest palindrome subsequence of an input string.

if input is *character*, output should be *carac*.

Exercise 4

- A palindrome is a string that reads the same forwards and backwards.
 - **Examples:** *racecar*, *civic*, *a*.

Exercise 4: Design an $O(n^2)$ –time algorithm from scratch by dynamic programming.

Acknowledgement

- The slides are modified from previous editions of this course and similar course elsewhere.
- **List of credits:**
 - Surender Baswana
 - Arnab Bhattacharya
 - Diptarka Chakraborty
 - Yi-Jun Chang
 - Erik Demaine
 - Steven Halim
 - Sanjay Jain
 - Wee Sun Lee
 - Charles Leiserson
 - Hon Wai Leong
 - Wing-Kin Sung