Find Common Subsequence

1: $i \leftarrow n, j \leftarrow m, S \leftarrow ()$
2: while $i > 0$ and $j > 0$ do
3:   if $\pi[i, j] = "\downarrow"$ then
4:       add $A[i]$ to beginning of $S$, $i \leftarrow i - 1, j \leftarrow j - 1$
5:   else if $\pi[i, j] = "\uparrow"$ then
6:       $i \leftarrow i - 1$
7:   else
8:       $j \leftarrow j - 1$
9: return $S$
Variants of Problem

**Edit Distance with Insertions and Deletions**

**Input:** a string $A$ and a string $B$
- each time we can delete a letter from $A$ or insert a letter to $A$

**Output:** minimum number of operations (insertions or deletions) we need to change $A$ to $B$?

**Example:**
$A = \text{ocurrance}$, $B = \text{occurrence}$
- 3 operations: insert 'c', remove 'a' and insert 'e'

Obs. $\#\text{OPs} = \text{length}(A) + \text{length}(B) - 2 \cdot \text{length}(\text{LCS}(A, B))$
Variants of Problem

Edit Distance with Insertions and Deletions

**Input:** a string $A$ and a string $B$

each time we can delete a letter from $A$ or insert a letter to $A$

**Output:** minimum number of operations (insertions or deletions) we need to change $A$ to $B$?

Example:

- $A = \text{ocurrance}, \ B = \text{occurrence}$
- 3 operations: insert 'c', remove 'a' and insert 'e'
Variants of Problem

Edit Distance with Insertions and Deletions

**Input:** a string $A$ and a string $B$

each time we can delete a letter from $A$ or insert a letter to $A$

**Output:** minimum number of operations (insertions or deletions) we need to change $A$ to $B$?

**Example:**

- $A = \text{ocurrance}, \ B = \text{occurrence}$
- 3 operations: insert 'c', remove 'a' and insert 'e'

**Obs.** \[ \#\text{OPs} = \text{length}(A) + \text{length}(B) - 2 \cdot \text{length}(\text{LCS}(A, B)) \]
Variants of Problem

Edit Distance with Insertions, Deletions and Replacing

**Input:** a string $A$ and a string $B$

each time we can delete a letter from $A$, insert a letter to $A$ or change a letter

**Output:** how many operations do we need to change $A$ to $B$?
Variants of Problem

**Edit Distance with Insertions, Deletions and Replacing**

**Input:** a string $A$ and a string $B$

each time we can delete a letter from $A$, insert a letter to $A$ or change a letter

**Output:** how many operations do we need to change $A$ to $B$?

**Example:**

- $A = \text{o}c\text{urrance}$, $B = \text{o}c\text{currence}$.
- 2 operations: insert 'c', change 'a' to 'e'
Variants of Problem

Edit Distance with Insertions, Deletions and Replacing

**Input:** a string $A$ and a string $B$

  each time we can delete a letter from $A$, insert a letter to $A$ or change a letter

**Output:** how many operations do we need to change $A$ to $B$?

**Example:**

- $A = \text{occurrance}, \ B = \text{occurrence}$.
- 2 operations: insert 'c', change 'a' to 'e'

- Not related to LCS any more
Need to match letters in $A$ and $B$, every letter is matched at most once and there should be no crosses.

However, we can match two different letters: Matching a same letter gives score 2, matching two different letters gives score 1.

Need to maximize the score.

DP recursion for the case $i > 0$ and $j > 0$:

$$opt[i, j] = \begin{cases} 
opt[i - 1, j - 1] + 2 & \text{if } A[i] = B[j] \\
\max \left\{ \begin{array}{ll}
opt[i - 1, j] & \text{if } A[i] = B[j] \\
opt[i, j - 1] & \text{if } A[i] \neq B[j]
\end{array} \right. \\
\max \left\{ \begin{array}{ll}
opt[i - 1, j - 1] + 1 & \text{if } A[i] = B[j]
\end{array} \right.
\end{cases}$$

Relation: $\#OPs = \text{length}(A) + \text{length}(B) - \text{max\_score}$
Edit Distance (with Replacing): using DP directly

- $opt[i, j], 0 \leq i \leq n, 0 \leq j \leq m$: edit distance between $A[1 .. i]$ and $B[1 .. j]$. 
**Edit Distance (with Replacing): using DP directly**

- $opt[i, j], 0 \leq i \leq n, 0 \leq j \leq m$: edit distance between $A[1 .. i]$ and $B[1 .. j]$.
- if $i = 0$ then $opt[i, j] = j$; if $j = 0$ then $opt[i, j] = i$. 


Edit Distance (with Replacing): using DP directly

- \(\text{opt}[i, j], 0 \leq i \leq n, 0 \leq j \leq m: \) edit distance between \(A[1 .. i]\) and \(B[1 .. j]\).

- if \(i = 0\) then \(\text{opt}[i, j] = j\); if \(j = 0\) then \(\text{opt}[i, j] = i\).

- if \(i > 0, j > 0\), then

\[
\text{opt}[i, j] = \begin{cases} \text{if } A[i] = B[j] \\ \text{if } A[i] \neq B[j] \end{cases}
\]

\[
\text{opt}[i, j] = \begin{cases} j & \text{if } A[i] = B[j] \\ i & \text{if } A[i] \neq B[j] \end{cases}
\]
Edit Distance (with Replacing): using DP directly

- \( opt[i, j], 0 \leq i \leq n, 0 \leq j \leq m \): edit distance between \( A[1 .. i] \) and \( B[1 .. j] \).
- If \( i = 0 \) then \( opt[i, j] = j \); if \( j = 0 \) then \( opt[i, j] = i \).
- If \( i > 0, j > 0 \), then

\[
opt[i, j] = \begin{cases} 
  opt[i - 1, j - 1] & \text{if } A[i] = B[j] \\
  \min \{ opt[i - 1, j], opt[i, j - 1], opt[i - 1, j - 1] + 1 \} & \text{if } A[i] \neq B[j]
\end{cases}
\]
Edit Distance (with Replacing): using DP directly

- $opt[i, j]$, $0 \leq i \leq n$, $0 \leq j \leq m$: edit distance between $A[1..i]$ and $B[1..j]$.

- If $i = 0$ then $opt[i, j] = j$; if $j = 0$ then $opt[i, j] = i$.

- If $i > 0$, $j > 0$, then

$$
opt[i, j] = \begin{cases} 
  opt[i - 1, j - 1] & \text{if } A[i] = B[j] \\
  \min \begin{cases} 
    opt[i - 1, j] + 1 \\
    opt[i, j - 1] + 1 \\
    opt[i - 1, j - 1] + 1 
  \end{cases} & \text{if } A[i] \neq B[j]
\end{cases}
$$
Outline

1. Weighted Interval Scheduling
2. Subset Sum Problem
3. Knapsack Problem
4. Longest Common Subsequence
   - Longest Common Subsequence in Linear Space
5. Shortest Paths in Directed Acyclic Graphs
6. Matrix Chain Multiplication
7. Optimum Binary Search Tree
8. Summary
9. Summary of Studies Until April
Computing the Length of LCS

1: for $j \leftarrow 0$ to $m$ do
2: \hspace{1em} $opt[0, j] \leftarrow 0$
3: for $i \leftarrow 1$ to $n$ do
4: \hspace{1em} $opt[i, 0] \leftarrow 0$
5: for $j \leftarrow 1$ to $m$ do
6: \hspace{2em} if $A[i] = B[j]$ then
7: \hspace{3em} $opt[i, j] \leftarrow opt[i - 1, j - 1] + 1$
8: \hspace{2em} else if $opt[i, j - 1] \geq opt[i - 1, j]$ then
9: \hspace{3em} $opt[i, j] \leftarrow opt[i, j - 1]$
10: \hspace{2em} else
11: \hspace{3em} $opt[i, j] \leftarrow opt[i - 1, j]$

Obs. The $i$-th row of table only depends on $(i - 1)$-th row.
Reducing Space to $O(n + m)$

**Obs.** The $i$-th row of table only depends on $(i - 1)$-th row.

**Q:** How to use this observation to reduce space?
Reducing Space to $O(n + m)$

**Obs.** The $i$-th row of table only depends on $(i - 1)$-th row.

**Q:** How to use this observation to reduce space?

**A:** We only keep two rows: the $(i - 1)$-th row and the $i$-th row.
Linear Space Algorithm to Compute Length of LCS

1: for $j \leftarrow 0$ to $m$ do  
2: \hspace{1em} $opt[0, j] \leftarrow 0$  
3: for $i \leftarrow 1$ to $n$ do  
4: \hspace{1em} $opt[i \mod 2, 0] \leftarrow 0$  
5: for $j \leftarrow 1$ to $m$ do  
6: \hspace{2em} if $A[i] = B[j]$ then  
7: \hspace{3em} $opt[i \mod 2, j] \leftarrow opt[i - 1 \mod 2, j - 1] + 1$  
8: \hspace{2em} else if $opt[i \mod 2, j - 1] \geq opt[i - 1 \mod 2, j]$ then  
9: \hspace{3em} $opt[i \mod 2, j] \leftarrow opt[i \mod 2, j - 1]$  
10: \hspace{2em} else  
11: \hspace{3em} $opt[i \mod 2, j] \leftarrow opt[i - 1 \mod 2, j]$  
12: return $opt[n \mod 2, m]$
Only keep the last two rows: only know how to match $A[n]$
How to Recover LCS Using Linear Space?

- Only keep the last two rows: only know how to match \( A[n] \)
- Can recover the LCS using \( n \) rounds: \( \text{time} = O(n^2 m) \)
How to Recover LCS Using Linear Space?

- Only keep the last two rows: only know how to match $A[n]$
- Can recover the LCS using $n$ rounds: time $= O(n^2m)$
- Using Divide and Conquer + Dynamic Programming:
How to Recover LCS Using Linear Space?

- Only keep the last two rows: only know how to match $A[n]$
- Can recover the LCS using $n$ rounds: time = $O(n^2m)$
- Using **Divide and Conquer** + Dynamic Programming:
  - Space: $O(m + n)$
How to Recover LCS Using Linear Space?

- Only keep the last two rows: only know how to match $A[n]$.
- Can recover the LCS using $n$ rounds: time $= O(n^2m)$.
- Using Divide and Conquer + Dynamic Programming:
  - Space: $O(m + n)$
  - Time: $O(nm)$
Outline

1. Weighted Interval Scheduling
2. Subset Sum Problem
3. Knapsack Problem
4. Longest Common Subsequence
   - Longest Common Subsequence in Linear Space
5. Shortest Paths in Directed Acyclic Graphs
6. Matrix Chain Multiplication
7. Optimum Binary Search Tree
8. Summary
9. Summary of Studies Until April
**Def.** A directed acyclic graph (DAG) is a directed graph without (directed) cycles.

---

**Directed Acyclic Graphs**

Diagram 1: Not a DAG

Diagram 2: A DAG
Directed Acyclic Graphs

**Def.** A directed acyclic graph (DAG) is a directed graph without (directed) cycles.

**Lemma** A directed graph is a DAG if and only its vertices can be topologically sorted.
**Shortest Paths in DAG**

**Input:** directed acyclic graph $G = (V, E)$ and $w : E \to \mathbb{R}$.
Assume $V = \{1, 2, 3 \cdots, n\}$ is topologically sorted: if $(i, j) \in E$, then $i < j$

**Output:** the shortest path from 1 to $i$, for every $i \in V$
Shortest Paths in DAG

Input: directed acyclic graph $G = (V, E)$ and $w : E \to \mathbb{R}$.
Assume $V = \{1, 2, 3 \cdots, n\}$ is topologically sorted: if $(i, j) \in E$, then $i < j$

Output: the shortest path from 1 to $i$, for every $i \in V$
Shortest Paths in DAG

- \( f[i] \): length of the shortest path from 1 to \( i \)

\[
f[i] = \begin{cases} 
  0 & \text{if } i = 1 \\
  \min_{j:(j,i) \in E} \{ f[j] + w[j,i] \} & \text{if } i = 2, 3, \ldots, n 
\end{cases}
\]
$f[i]$: length of the shortest path from 1 to $i$

$$f[i] = \begin{cases} 
0 & i = 1 \\
& i = 2, 3, \ldots, n 
\end{cases}$$
Shortest Paths in DAG

- $f[i]$: length of the shortest path from 1 to $i$

$$f[i] = \begin{cases} 
0 & i = 1 \\
\min_{j: (j,i) \in E} \{ f(j) + w(j, i) \} & i = 2, 3, \ldots, n
\end{cases}$$
Use an adjacency list for incoming edges of each vertex $i$

```plaintext
Shortest Paths in DAG

1: $f[1] \leftarrow 0$
2: for $i \leftarrow 2$ to $n$ do
3:     $f[i] \leftarrow \infty$
4: for each incoming edge $(j, i)$ of $i$ do
5:     if $f[j] + w(j, i) < f[i]$ then
6:         $f[i] \leftarrow f[j] + w(j, i)$
```
Use an adjacency list for incoming edges of each vertex $i$

**Shortest Paths in DAG**

1. $f[1] \leftarrow 0$
2. **for** $i \leftarrow 2$ to $n$ **do**
3. \hspace{1cm} $f[i] \leftarrow \infty$
4. **for** each incoming edge $(j, i)$ of $i$ **do**
5. \hspace{1cm} **if** $f[j] + w(j, i) < f[i]$ **then**
6. \hspace{2cm} $f[i] \leftarrow f[j] + w(j, i)$
7. \hspace{2cm} $\pi(i) \leftarrow j$
Shortest Paths in DAG

- Use an adjacency list for incoming edges of each vertex $i$

---

Shortest Paths in DAG

1. $f[1] \leftarrow 0$
2. **for** $i \leftarrow 2$ to $n$ **do**
3. \hspace{1em} $f[i] \leftarrow \infty$
4. **for** each incoming edge $(j, i)$ of $i$ **do**
5. \hspace{1em} **if** $f[j] + w(j, i) < f[i]$ **then**
6. \hspace{2em} $f[i] \leftarrow f[j] + w(j, i)$
7. \hspace{2em} $\pi(i) \leftarrow j$

---

print-path($t$)

1. **if** $t = 1$ **then**
2. \hspace{1em} print(1)
3. \hspace{1em} return
4. \hspace{1em} print-path($\pi(t)$)
5. \hspace{1em} print(“,”, $t$)
Example
Example
Example
Example
Example
Example
Example
Example
Heaviest Path in a Directed Acyclic Graph

**Input:** directed acyclic graph \( G = (V, E) \) and \( w : E \rightarrow \mathbb{R} \).
Assume \( V = \{1, 2, 3 \cdots, n\} \) is topologically sorted: if \((i, j) \in E\), then \(i < j\)

**Output:** the path with the largest weight (the heaviest path) from 1 to \(n\).

- \(f[i]\): weight of the heaviest path from 1 to \(i\)

\[
f[i] = \begin{cases} i = 1 \\ i = 2, 3, \cdots, n \end{cases}
\]
Heaviest Path in a Directed Acyclic Graph

**Input:** directed acyclic graph $G = (V, E)$ and $w : E \rightarrow \mathbb{R}$.

Assume $V = \{1, 2, 3 \cdots, n\}$ is topologically sorted: if $(i, j) \in E$, then $i < j$

**Output:** the path with the largest weight (the heaviest path) from 1 to $n$.

- $f[i]$: weight of the heaviest path from 1 to $i$

$$f[i] = \begin{cases} 0 & i = 1 \\ i = 2, 3, \cdots, n \end{cases}$$
Heaviest Path in a Directed Acyclic Graph

**Input:** directed acyclic graph $G = (V, E)$ and $w : E \rightarrow \mathbb{R}$.

Assume $V = \{1, 2, 3, \ldots, n\}$ is topologically sorted: if $(i, j) \in E$, then $i < j$

**Output:** the path with the largest weight (the heaviest path) from 1 to $n$.

- $f[i]$: weight of the heaviest path from 1 to $i$

$$f[i] = \begin{cases} 0 & i = 1 \\ \max_{j: (j, i) \in E} \{f(j) + w(j, i)\} & i = 2, 3, \ldots, n \end{cases}$$
Outline

1. Weighted Interval Scheduling
2. Subset Sum Problem
3. Knapsack Problem
4. Longest Common Subsequence
   - Longest Common Subsequence in Linear Space
5. Shortest Paths in Directed Acyclic Graphs
6. **Matrix Chain Multiplication**
7. Optimum Binary Search Tree
8. Summary
9. Summary of Studies Until April
Matrix Chain Multiplication

**Input:** $n$ matrices $A_1, A_2, \cdots, A_n$ of sizes $r_1 \times c_1, r_2 \times c_2, \cdots, r_n \times c_n$, such that $c_i = r_{i+1}$ for every $i = 1, 2, \cdots, n - 1$.

**Output:** the order of computing $A_1 A_2 \cdots A_n$ with the minimum number of multiplications.

**Fact** Multiplying two matrices of size $r \times k$ and $k \times c$ takes $r \times k \times c$ multiplications.
Example:

\( A_1 : 10 \times 100, \quad A_2 : 100 \times 5, \quad A_3 : 5 \times 50 \)

\[
\begin{align*}
10 \times 100 & \quad 100 \times 5 & \quad 5 \times 50 \\
10 \times 5 & \quad 10 \cdot 100 \cdot 5 & = 5000 \\
10 \times 50 & \quad 10 \cdot 5 \cdot 50 & = 2500 \\
\end{align*}
\]

Cost = 5000 + 2500 = 7500

\[
\begin{align*}
10 \times 100 & \quad 100 \times 5 & \quad 5 \times 50 \\
10 \times 5 & \quad 100 \cdot 5 \cdot 50 & = 25000 \\
10 \times 50 & \quad 10 \cdot 100 \cdot 50 & = 50000 \\
\end{align*}
\]

Cost = 25000 + 50000 = 75000

\( (A_1 A_2) A_3 \): \( 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500 \)

\( A_1 (A_2 A_3) \): \( 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000 \)
Example:

- $A_1 : 10 \times 100, \quad A_2 : 100 \times 5, \quad A_3 : 5 \times 50$

  
  \[
  \begin{align*}
  \text{10} \times 100 & \quad \text{100} \times 5 & \quad 5 \times 50 \\
  \text{10} \times 5 & \quad 10 \cdot 100 \cdot 5 & \quad \text{5} \times 50 \\
  \text{10} \times 50 & \quad 10 \cdot 5 \cdot 50 & \quad \text{10} \cdot 100 \cdot 50
  \end{align*}
  \]

  \[
  \begin{align*}
  = 5000 & \quad = 2500 & \quad = 50000
  \end{align*}
  \]

  \[
  \text{cost} = 5000 + 2500 = 7500
  \]

  \[
  \text{cost} = 25000 + 50000 = 75000
  \]

- $(A_1 A_2) A_3 : 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$

- $A_1 (A_2 A_3) : 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$
Matrix Chain Multiplication: Design DP

Let's assume the last step is

\[(A_1 A_2 \cdots A_i)(A_i+1 A_i+2 \cdots A_n)\]

Cost of last step:

\[r_1 \times c_i \times c_n\]

Optimality for sub-instances: we need to compute

\[A_1 A_2 \cdots A_i\]

and

\[A_i+1 A_i+2 \cdots A_n\]

optimally

\[\text{opt}[i, j] = \begin{cases} 
0 & \text{if } i = j \\
\min_k \{i \leq k < j \left( \text{opt}[i, k] + \text{opt}[k+1, j] + r_i c_k c_j \right) \} & \text{otherwise}
\end{cases} \]