Longest Common Subsequence

Input: two strings, $u$ and $v$

Output: a common substring (not necessarily contiguous)

Objective: maximize length of the substring

Example: $	ext{carrot}$

\[ S[j,k] = \text{max length of a common substring of } u[1...j] \text{ and } v[1...k] \]

(1st part of the heart)

(3rd part):

\[ \text{return } S[|u|, |v|] \]

length of $u$
Longest Common Subsequence

Input: two strings, $u$ and $v$

Output: a common substring

Objective: maximize length of the substring

The heart of the solution:

$$S[j,k] = \begin{cases} 0 & \text{if } j=0 \text{ or } k=0 \\ \max\{S[j-1,k], S[j,k-1]\} & \text{if } j>0 \text{ and } k>0 \text{ and } u_j \neq v_k \\ 1 + S[j-1,k-1] & \text{if } j>0 \text{ and } k>0 \text{ and } u_j = v_k \end{cases}$$
Longest Common Subsequence

Input: two strings, u and v
Output: a common substring
Objective: maximize length of the substring

The heart of the solution:

\[ S[j][k] = \]
Longest Common Subsequence

Input: two strings, u and v

Output: a common substring

Objective: maximize length of the substring

The heart of the solution:

\[ S[j][k] = \text{the length of the longest common substring of strings } u[1...j] \text{ and } v[1...k] \]
Longest Common Subsequence

\[ S[j][k] = \text{the length of the longest common substring of strings } u[1...j] \text{ and } v[1...k] \]

**LONGEST-COMMON-SUBSEQUENCE** \((u,v)\)

1. init \(S[j][k]\) to 0 for every \(j=0,...,|u|\) and every \(k=0,...,|v|\)
2. for \(j=1\) to \(|u|\) do
3. for \(k=1\) to \(|v|\) do
4. \(S[j][k] = \max\{ S[j-1][k], S[j][k-1] \} \)
5. if \((u[j] = v[k])\) then
6. \(S[j][k] = S[j-1][k-1]+1\)
7. RETURN \(S[|u|][|v|]\)
Matrix Chain Multiplication

Input: a chain of matrices to be multiplied
Output: a parenthesizing of the chain
Objective: minimize number of steps needed for the multiplication

\[
\begin{align*}
A_1 & \times A_2 & \times A_3 & \times A_4 & \times A_5 & \times A_6 & \times A_7 \\
5 \times 3 & 3 \times 7 & 7 \times 2 & 2 \times 1 & 1 \times 5 & 5 \times 3 & 3 \times 4 \\
( & ( & ( & ( & ( & ) & ) \\
( & ( & ( & ( & ( & ) & ) \\
( & ( & ( & ( & ( & ) & ) \\
\end{align*}
\]

Overall cost of this parenthesizing is:

\[
5 \times 3 \times 7 + 7 \times 2 \times 1 + 1 \times 5 \times 3 + 5 \times 3 \times 1 + 1 \times 3 \times 4 + 5 \times 1 \times 4
\]
Matrix Chain Multiplication

Input: a chain of matrices to be multiplied

Output: a parenthesizing of the chain

Objective: minimize number of steps needed for the multiplication

Matrix multiplication:

A of size \( m \times n \), B of size \( n \times p \)

How many steps to compute \( A \cdot B \) ?

Cost of multiplying \( A \cdot B \):

\[
c_{ij} = \sum_{k=1}^{r} a_{ik} \cdot b_{kj}
\]

\( m \cdot r \cdot p \)
Matrix Chain Multiplication

Input: a chain of matrices to be multiplied
Output: a parenthesizing of the chain
Objective: minimize number of steps needed for the multiplication

Example:
\[ a_0, a_1, a_2, a_3, a_4, a_5 \]
\[
4 \quad 2 \quad 5 \quad 1 \quad 2 \quad 3
\]
\[ A_i \text{ is of dim. } a_{i-1} \times a_i \]
Matrix Chain Multiplication

Input: a chain of matrices to be multiplied
Output: a parenthesizing of the chain
Objective: minimize number of steps needed for the multiplication

Heart of the solution:

\[
S[L, R] = \begin{cases} 
\min_{k \in \{L, L+1, \ldots, R-1\}} a_{L-1} \cdot a_k \cdot a_R + S[L, k] + S[k+1, R] & \text{for } 1 \leq L < R \leq n \\
0 & \text{for } L = R 
\end{cases}
\]

return: \( S[1, n] \)
**Matrix Chain Multiplication**

**Input:** a chain of matrices to be multiplied  
**Output:** a parenthesizing of the chain  
**Objective:** minimize number of steps needed for the multiplication

Heart of the solution:

\[
S[L, R] = \text{the minimum number of steps required to multiply matrices from the } L\text{-th to the } R\text{-th}
\]
Matrix Chain Multiplication

**MATRİX-CHAIN-MULTIPLICATION** \((a_0, \ldots, a_n)\)

1. for \(L=1\) to \(n\) do \(S[L,L] = 0\)
2. for \(d=1\) to \(n\) do
3. for \(L=1\) to \(n-d\) do
4. \(R = L+d\)
5. \(S[L,R] = \infty\)
6. for \(k=L\) to \(R-1\) do
7. \(tmp = S[L,k]+S[k+1,R]+a_{L-1}a_ka_R\)
8. if \(S[L,R] > tmp\) then \(S[L,R] = tmp\)
9. return \(S[1,n]\)