COL351: Analysis and Design of Algorithms

Ragesh Jaiswal, CSE, IITD

Dynamic Programming

Problem

Let S and T be strings of characters. S is of length n and T is of length m. Find the *longest common subsequence* in S and T. This is the longest sequence of characters (not necessarily contiguous) that appear in both S and T.

- Example $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$

Problem

Let S and T be strings of characters. S is of length n and T is of length m. Find the *longest common subsequence* in S and T. This is the longest sequence of characters (not necessarily contiguous) that appear in both S and T.

- Example $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
  - The longest common subsequence is XYXP
  - $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
- How do we define the subproblems?

Problem

Let S and T be strings of characters. S is of length n and T is of length m. Find the *longest common subsequence* in S and T. This is the longest sequence of characters (not necessarily contiguous) that appear in both S and T.

- Example $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
  - The longest common subsequence is XYXP
  - $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
- Let $L(i, j)$ denote the length of the longest common subsequence in strings $S[1], ..., S[i]$ and $T[1], ..., T[j]$.
- What is $L(1, j)$ for $1 < j \leq m$?

Problem

Let S and T be strings of characters. S is of length n and T is of length m. Find the *longest common subsequence* in S and T. This is the longest sequence of characters (not necessarily contiguous) that appear in both S and T.

- Example $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
  - The longest common subsequence is XYXP
  - $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
- Let $L(i, j)$ denote the length of the longest common subsequence in strings $S[1], ..., S[i]$ and $T[1], ..., T[j]$.
- What is $L(1, j)$ for $1 < j \leq m$?
  - 1 if $S[1]$ is present in the string $T[1], ..., T[j]$, 0 otherwise.

Problem

Let S and T be strings of characters. S is of length n and T is of length m. Find the *longest common subsequence* in S and T. This is the longest sequence of characters (not necessarily contiguous) that appear in both S and T.

- Example $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
  - The longest common subsequence is XYXP
  - $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
- Let $L(i, j)$ denote the length of the longest common subsequence in strings $S[1], ..., S[i]$ and $T[1], ..., T[j]$.
- What is $L(1, j)$ for $1 < j \leq m$?
  - 1 if $S[1]$ is present in the string $T[1], ..., T[j]$, 0 otherwise.
  - 1 if $S[1] = T[j]$ else $L(1, j) = L(1, j-1)$ (with $L(1, 0) = 0$)

- Example $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
  - The longest common subsequence is XYXP
  - $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
- Let $L(i, j)$ denote the length of the longest common subsequence in strings $S[1], ..., S[i]$ and $T[1], ..., T[j]$.
- What is $L(1, j)$ for $1 < j \leq m$?
  - 1 if $S[1]$ is present in the string $T[1], ..., T[j]$, 0 otherwise.
  - 1 if $S[1] = T[j]$ else $L(1, j) = L(1, j - 1)$ (with $L(1, 0) = 0$)
- Similarly, we can define $L(i, 1)$ for $1 < i \leq n$.
- Can we say something similar for $L(i, j)$ for $i, j \neq 1$?

- Example $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
  - The longest common subsequence is XYXP
  - $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$
- Let $L(i, j)$ denote the length of the longest common subsequence in strings $S[1], ..., S[i]$ and $T[1], ..., T[j]$.
- What is $L(1, j)$ for $1 < j \leq m$?
  - 1 if $S[1]$ is present in the string $T[1], ..., T[j]$, 0 otherwise.
  - 1 if $S[1] = T[j]$ else $L(1, j) = L(1, j-1)$ (with $L(1, 0) = 0$)
- Similarly, we can define $L(i, 1)$ for $1 < i \leq n$.
- Can we say something similar for $L(i, j)$ for $i, j \neq 1$?
  - Claim 1: If $S[i] = T[j]$, then $L(i, j) = 1 + L(i-1, j-1)$.
  - Claim 2: If $S[i] \neq T[j]$, then $L(i, j) = \max\{L(i-1, j), L(i, j-1)\}$.

- What is $L(1, j)$ for $1 < j \leq m$?
  - 1 if $S[1]$ is present in the string $T[1], ..., T[j]$, 0 otherwise.
  - 1 if $S[1] = T[j]$ else $L(1, j) = L(1, j - 1)$ (with $L(1, 0) = 0$)
- Similarly, we can define $L(i, 1)$ for $1 < i \leq n$.
- Can we say something similar for $L(i, j)$ for $i, j \neq 1$?
  - Claim 1: If $S[i] = T[j]$, then $L(i, j) = 1 + L(i - 1, j - 1)$.
  - Claim 2: If $S[i] \neq T[j]$, then $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.

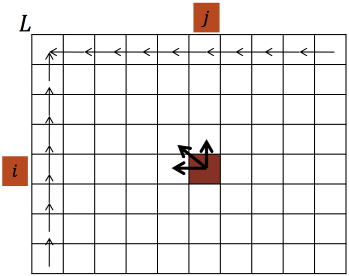


Figure: The arrows show the dependencies between subproblems.

Algorithm

`Length-LCS(S, T)`

- If $(S[1] = T[1])$, then $L[1,1] \leftarrow 1$ else $L[1,1] \leftarrow 0$
- For $j = 2$ to m
  - If $(S[1] = T[j])$, then $L[1,j] \leftarrow 1$ else $L[1,j] \leftarrow L[1,j-1]$
- For $i = 2$ to n
  - If $(S[i] = T[1])$, then $L[i,1] \leftarrow 1$ else $L[i,1] \leftarrow L[i-1,1]$
- For $i = 2$ to n
  - For $j = 2$ to m
    - If $(S[i] = T[j])$ then $L[i,j] \leftarrow 1 + L[i-1,j-1]$
      else $L[i,j] \leftarrow \max\{L[i-1,j], L[i,j-1]\}$
- Return$(L[n,m])$

- How do we find a longest common subsequence?

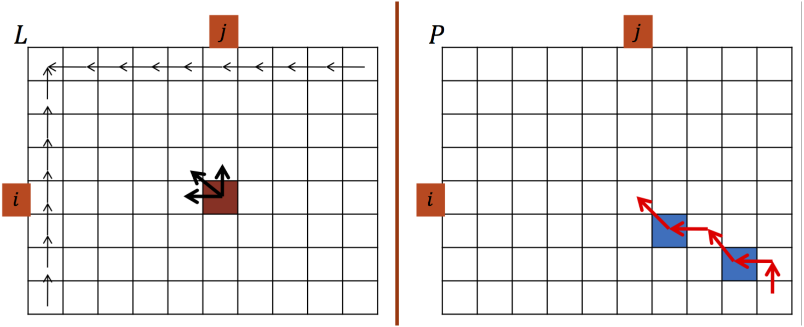


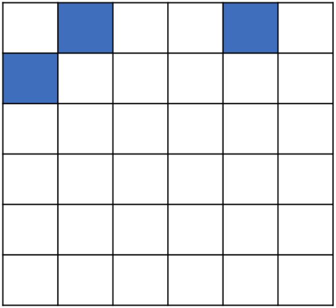
Figure: Array P is used to maintain the pointers to the appropriate subproblem. The blue squares give the position of the characters in a longest common subsequence.

- Example: $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$

L

0	1	1	1	1	1
1					
1					
1					
1					
1					

P

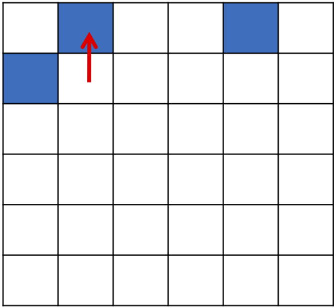


- Example: $S = $ XYXZPQ, $T = $ YXQYXP

L

0	1	1	1	1	1
1	1				
1					
1					
1					
1					

P



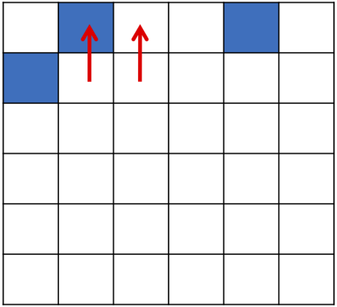
Dynamic Programming
Longest common subsequence

- Example: S = XYXZPQ, T = YXQYXP

L

0	1	1	1	1	1
1	1	1			
1					
1					
1					
1					

P

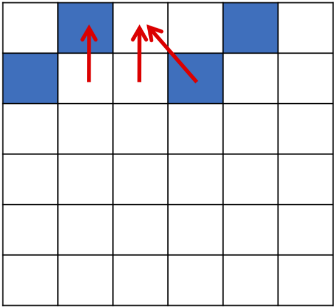


- Example: S = XYXZPQ, T = YXQYXP

L

0	1	1	1	1	1
1	1	1	2		
1					
1					
1					
1					

P



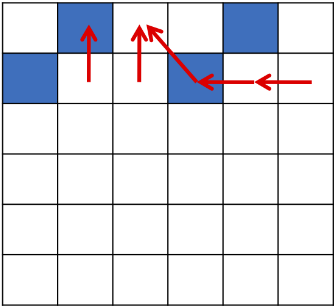
Dynamic Programming
Longest common subsequence

- Example: $S = $ XYXZPQ, $T = $ YXQYXP

L

0	1	1	1	1	1
1	1	1	2	2	2
1					
1					
1					
1					

P



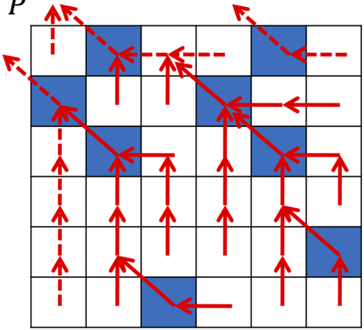
Dynamic Programming
Longest common subsequence

- Example: $S = \text{XYXZPQ}$, $T = \text{YXQYXP}$

L

0	1	1	1	1	1
1	1	1	2	2	2
1	2	2	2	3	3
1	2	2	2	3	3
1	2	2	2	3	4
1	2	3	3	3	4

P



Problem

Let S and T be strings of characters. S is of length n and T is of length m. Find a *longest common subsequence* in S and T. This is a longest sequence of characters (not necessarily contiguous) that appear in both S and T.

- Claim 1: If $i = 0$ or $j = 0$, then $L(i, j) = 0$.
- Claim 2: If $S[i] = T[j]$, then $L(i, j) = 1 + L(i - 1, j - 1)$.
- Claim 3: If $S[i] \neq T[j]$, then $L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$.

- Claim 1: If $i = 0$ or $j = 0$, then $L(i, j) = 0$.
- Claim 2: If $S[i] = T[j]$, then $L(i, j) = 1 + L(i-1, j-1)$.
- Claim 3: If $S[i] \neq T[j]$, then $L(i, j) = \max\{L(i-1, j), L(i, j-1)\}$.
- Here is a simple recursive program to find the length of the longest common subsequence.

Algorithm

LCS-rec(S, n, T, m)
- If ($n = 0$ OR $m = 0$) then return(0)
- If ($S[n] = S[m]$) return(1 + LCS-rec($S, n-1, T, m-1$))
- If ($S[n] \neq T[m]$)
  return(max{LCS-rec($S, n, T, m-1$), LCS-rec($S, n-1, T, m$)})

Algorithm

`LCS-rec`(S, n, T, m)

- If $(n = 0$ OR $m = 0)$ then return(0)
- If $(S[n] = S[m])$ return($1 +$ `LCS-rec`$(S, n-1, T, m-1)$)
- If $(S[n] \neq T[m])$
  return(max{`LCS-rec`$(S, n, T, m-1)$, `LCS-rec`$(S, n-1, T, m)$})

- What is the running time of this algorithm?

Algorithm

$\texttt{LCS-rec}(S, n, T, m)$

- If $(n = 0 \text{ OR } m = 0)$ then return(0)
- If $(S[n] = S[m])$ return$(1 + \texttt{LCS-rec}(S, n-1, T, m-1))$
- If $(S[n] \neq T[m])$
  return$(\max\{\texttt{LCS-rec}(S, n, T, m-1), \texttt{LCS-rec}(S, n-1, T, m)\})$

- What is the running time of this algorithm?
  - This is exponentially large!

Algorithm

LCS-rec(S, n, T, m)
- If $(n = 0$ OR $m = 0)$ then return(0)
- If $(S[n] = S[m])$ return($1 +$ LCS-rec$(S, n-1, T, m-1)$)
- If $(S[n] \neq T[m])$
  return(max{LCS-rec$(S, n, T, m-1)$, LCS-rec$(S, n-1, T, m)$})

- Here is a *memoized* version of the above algorithm.

Algorithm

LCS-mem(S, n, T, m)
- If $(n = 0$ OR $m = 0)$ then return(0)
- If ($L[n, m]$ is known) then return($L[n, m]$)
- If $(S[n] = S[m])$
  - *length* $\leftarrow 1 +$ LCS-mem$(S, n-1, T, m-1)$
- If $(S[n] \neq T[m])$
  - *length* $\leftarrow$ max{LCS-mem$(S, n, T, m-1)$,
    LCS-mem$(S, n-1, T, m)$}
- $L[n, m] \leftarrow$ *length*
- return(*length*)

- Here is a *memoized* version of the recursive algorithm.

Algorithm

LCS-mem(S, n, T, m)
- If $(n = 0 \text{ OR } m = 0)$ then return(0)
- If ($L[n, m]$ is known) then return($L[n, m]$)
- If $(S[n] = S[m])$
  - $length \leftarrow 1 + \texttt{LCS-mem}(S, n-1, T, m-1)$
- If $(S[n] \neq T[m])$
  - $length \leftarrow \max\{\texttt{LCS-mem}(S, n, T, m-1), \texttt{LCS-mem}(S, n-1, T, m)\}$
- $L[n, m] \leftarrow length$
- return($length$)

- What is the running time of the above algorithm?

- Here is a *memoized* version of the recursive algorithm.

Algorithm

$\texttt{LCS-mem}(S, n, T, m)$
- If $(n = 0 \text{ OR } m = 0)$ then return(0)
- If ($L[n, m]$ is known) then return($L[n, m]$)
- If $(S[n] = S[m])$
  - $length \leftarrow 1 + \texttt{LCS-mem}(S, n-1, T, m-1)$
- If $(S[n] \neq T[m])$
  - $length \leftarrow \max\{\texttt{LCS-mem}(S, n, T, m-1), \texttt{LCS-mem}(S, n-1, T, m)\}$
- $L[n, m] \leftarrow length$
- return($length$)

- What is the running time of the above algorithm? $O(nm)$

Problem

You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- Example: Let $(\{1, 2, 3, 5, 6, 7\}, 10)$ be the input instance.
- What is the optimal solution?

Problem

You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- Example: Let $([1, 2, 3, 5, 6, 7], 10)$ be the input instance.
- What is the optimal solution?{2, 3, 4}
  - Since $w(2) = 2, w(3) = 3, w(4) = 5$ and $w(2) + w(3) + w(4) = 10$.

Problem

You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- How do we define the subproblems for the Dynamic Program?

Problem

You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- How do we define the subproblems for the Dynamic Program?
- Let us try the following:
  - $M(i)$: The maximum weight that can be filled using items $\{1, ..., i\}$ subject to the sum being $\leq W$.
  - How do we define $M(i)$ in terms of $M(1), ..., M(i-1)$?

Problem

You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- How do we define the subproblems for the Dynamic Program?
- Let us try the following:
  - $M(i)$: The maximum weight that can be filled using items $\{1, ..., i\}$ subject to the sum being $\leq W$.
  - How do we define $M(i)$ in terms of $M(1), ..., M(i-1)$?
    - Case 1: i^{th} item is not in the optimal solution. Then $M(i) = M(i-1)$.
    - Case 2: i^{th} item is in the optimal solution. There is a problem here.

Problem

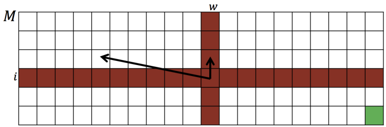
You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- How do we define the subproblems for the Dynamic Program?
- Let us try the following:
  - $M(i, w)$: The maximum weight that can be filled using items $\{1, ..., i\}$ subject to the sum being $\leq w$.
  - Recursive formulation:
    - Case 1: i^{th} item is not in the optimal solution. Then $M(i, w) = M(i-1, w)$.
    - Case 2: i^{th} item is in the optimal solution. Then $M(i, w) = M(i-1, w - w(i)) + w(i)$

Problem

You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- Dynamic Programming solution:
  - $M(i, w)$: The maximum weight that can be filled using items $\{1, ..., i\}$ subject to the sum being $\leq w$.
  - If $w(i) > w$, then $M(i, w) = M(i-1, w)$
  - If $w(i) \leq w$, then $M(i, w) = \max\{M(i-1, w), M(i-1, w-w(i)) + w(i)\}$
  - $\forall w \leq W$, $M(1, w) = w(1)$ if $w(1) \leq w$ and 0 otherwise.

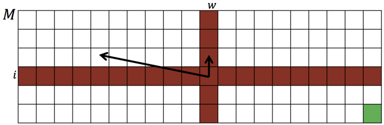


- What is the running time for filling the above table?

Problem

You are given n items with non-negative integer weights $w(i)$ and an integer W. You have to determine a subset S of $\{1, ..., n\}$ such that $\sum_{i \in S} w(i)$ is maximized subject to $\sum_{i \in S} w(i) \leq W$.

- Dynamic Programming solution:
  - $M(i, w)$: The maximum weight that can be filled using items $\{1, ..., i\}$ subject to the sum being $\leq w$.
  - If $w(i) > w$, then $M(i, w) = M(i-1, w)$
  - If $w(i) \leq w$, then $M(i, w) = \max\{M(i-1, w), M(i-1, w-w(i)) + w(i)\}$
  - $\forall w \leq W$, $M(1, w) = w(1)$ if $w(1) \leq w$ and 0 otherwise.



- What is the running time for filling the above table? $O(n \cdot W)$

End