

DP 2: Longest Common Subsequence and Longest Increasing Subsequence

Mattox Beckman

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF COMPUTER SCIENCE

Fall, 2022

Introduction and Objectives

Two common DP patterns involve picking subsequences from an array that maximize a property. Today we will discuss Longest Common Subsequence and Longest Increasing Subsequence.

Objectives

- ▶ Solve LIS using recursion.
- ▶ Solve LIS using DP techniques.
- ▶ Solve LCS using DP techniques.
- ▶ Solve LIS by converting to LCS.

LCS

The Problem

- ▶ Given two sequences, determine the length of the longest common subsequence.
- ▶ E.g.: a, c, e, h, k and b, c, d, e, k, m have longest common subsequence c, e, k.
- ▶ Note that the sequences do not need to be sorted!

Representation

- ▶ If we have two sequences A and B , we can use a 2-D DP array of $|A| + 1$ columns and $|B| + 1$ rows.
- ▶ Let A and B be 1 index.
- ▶ The DP array position i, j will give the length of the longest common subsequence ending at $A[i]$ and $B[j]$.

Matrix

	\emptyset	a	c	e	h	k
\emptyset	0	0	0	0	0	0
b	0					
c	0					
d	0					
e	0					
k	0					
m	0					

Formula

- ▶ $dp[0, j] = dp[i, 0] = 0$
- ▶ $dp[i, j] = dp[i - 1, j - 1] + 1$ if $A[i] = B[j]$
- ▶ $dp[i, j] = \max(dp[i - 1, j], dp[i, j - 1])$ otherwise

Representation

- ▶ If we have two sequences A and B , we can use a 2-D DP array of $|A| + 1$ columns and $|B| + 1$ rows.
- ▶ Let A and B be 1 index.
- ▶ The DP array position i, j will give the length of the longest common subsequence ending at $A[i]$ and $B[j]$.

Matrix

	\emptyset	a	c	e	h	k
\emptyset	0	0	0	0	0	0
b	0	0	0	0	0	0
c	0					
d	0					
e	0					
k	0					
m	0					

Formula

- ▶ $dp[0, j] = dp[i, 0] = 0$
- ▶ $dp[i, j] = dp[i - 1, j - 1] + 1$ if $A[i] = B[j]$
- ▶ $dp[i, j] = \max(dp[i - 1, j], dp[i, j - 1])$ otherwise

Representation

- ▶ If we have two sequences A and B , we can use a 2-D DP array of $|A| + 1$ columns and $|B| + 1$ rows.
- ▶ Let A and B be 1 index.
- ▶ The DP array position i, j will give the length of the longest common subsequence ending at $A[i]$ and $B[j]$.

Matrix

	\emptyset	a	c	e	h	k
\emptyset	0	0	0	0	0	0
b	0	0	0	0	0	0
c	0	0	1	1	1	1
d	0					
e	0					
k	0					
m	0					

Formula

- ▶ $dp[0, j] = dp[i, 0] = 0$
- ▶ $dp[i, j] = dp[i - 1, j - 1] + 1$ if $A[i] = B[j]$
- ▶ $dp[i, j] = \max(dp[i - 1, j], dp[i, j - 1])$ otherwise

Representation

- ▶ If we have two sequences A and B , we can use a 2-D DP array of $|A| + 1$ columns and $|B| + 1$ rows.
- ▶ Let A and B be 1 index.
- ▶ The DP array position i, j will give the length of the longest common subsequence ending at $A[i]$ and $B[j]$.

Matrix

	\emptyset	a	c	e	h	k
\emptyset	0	0	0	0	0	0
b	0	0	0	0	0	0
c	0	0	1	1	1	1
d	0	0	1	1	1	1
e	0					
k	0					
m	0					

Formula

- ▶ $dp[0, j] = dp[i, 0] = 0$
- ▶ $dp[i, j] = dp[i - 1, j - 1] + 1$ if $A[i] = B[j]$
- ▶ $dp[i, j] = \max(dp[i - 1, j], dp[i, j - 1])$ otherwise

Representation

- ▶ If we have two sequences A and B , we can use a 2-D DP array of $|A| + 1$ columns and $|B| + 1$ rows.
- ▶ Let A and B be 1 index.
- ▶ The DP array position i, j will give the length of the longest common subsequence ending at $A[i]$ and $B[j]$.

Matrix

	\emptyset	a	c	e	h	k
\emptyset	0	0	0	0	0	0
b	0	0	0	0	0	0
c	0	0	1	1	1	1
d	0	0	1	1	1	1
e	0	0	1	2	2	2
k	0					
m	0					

Formula

- ▶ $dp[0, j] = dp[i, 0] = 0$
- ▶ $dp[i, j] = dp[i - 1, j - 1] + 1$ if $A[i] = B[j]$
- ▶ $dp[i, j] = \max(dp[i - 1, j], dp[i, j - 1])$ otherwise

Representation

- ▶ If we have two sequences A and B , we can use a 2-D DP array of $|A| + 1$ columns and $|B| + 1$ rows.
- ▶ Let A and B be 1 index.
- ▶ The DP array position i, j will give the length of the longest common subsequence ending at $A[i]$ and $B[j]$.

Matrix

	\emptyset	a	c	e	h	k
\emptyset	0	0	0	0	0	0
b	0	0	0	0	0	0
c	0	0	1	1	1	1
d	0	0	1	1	1	1
e	0	0	1	2	2	2
k	0	0	1	2	2	3
m	0					

Formula

- ▶ $dp[0, j] = dp[i, 0] = 0$
- ▶ $dp[i, j] = dp[i - 1, j - 1] + 1$ if $A[i] = B[j]$
- ▶ $dp[i, j] = \max(dp[i - 1, j], dp[i, j - 1])$ otherwise

Representation

- ▶ If we have two sequences A and B , we can use a 2-D DP array of $|A| + 1$ columns and $|B| + 1$ rows.
- ▶ Let A and B be 1 index.
- ▶ The DP array position i, j will give the length of the longest common subsequence ending at $A[i]$ and $B[j]$.

Matrix

	\emptyset	a	c	e	h	k
\emptyset	0	0	0	0	0	0
b	0	0	0	0	0	0
c	0	0	1	1	1	1
d	0	0	1	1	1	1
e	0	0	1	2	2	2
k	0	0	1	2	2	3
m	0	0	1	2	2	3

Formula

- ▶ $dp[0, j] = dp[i, 0] = 0$
- ▶ $dp[i, j] = dp[i - 1, j - 1] + 1$ if $A[i] = B[j]$
- ▶ $dp[i, j] = \max(dp[i - 1, j], dp[i, j - 1])$ otherwise

Code

```
1  int LCSLength(vi a, vi b) {
2      int i,j;
3      vvi dp = vvi(a.length()+1,vi(b.length()+1));
4
5      for(i=0; i<=a.length(); ++i)
6          dp[i,0] = 0;
7      for(j=0; j<=b.length(); ++j)
8          dp[0,j] = 0;
9      for(i=1; i<=a.length(); ++i)
10         for(j=1; j<=b.length(); ++j)
11             if (a[i] == b[j])
12                 dp[i,j] = dp[i-1,j-1] + 1;
13             else
14                 dp[i,j] = max(dp[i-1,j], dp[i,j-1]);
15
16     return dp[a.length(),b.length()];
17 }
```

Discussion

Matrix

	\emptyset	<i>a</i>	<i>c</i>	<i>e</i>	<i>h</i>	<i>k</i>
\emptyset	0	0	0	0	0	0
<i>b</i>	0	0	0	0	0	0
<i>c</i>	0	0	1	1	1	1
<i>d</i>	0	0	1	1	1	1
<i>e</i>	0	0	1	2	2	2
<i>k</i>	0	0	1	2	2	3
<i>m</i>	0	0	1	2	2	3

Discussion

- ▶ How can we “read out” the actual subsequence?
- ▶ How can we save memory if the two subsequences are very large?

LIS

The Problem

- ▶ Given a sequences, determine the length of the longest increasing subsequence.
- ▶ E.g.: 2, 1, 5, 8, 3, 5, 10 has LIS of 2, 5, 8, 10.
- ▶ There can be more than one LIS!

Recursive Solution

- ▶ We can use the recursive solution as a start for the DP version.
- ▶ Let $lis(i)$ be the length of the longest increasing subsequence ending at i .
- ▶ Then $lis(0) = 1$.
 - ▶ $lis(i) = 1 + \max_{j=0}^{i-1} lis(j)$ when $a[j] < a[i]$
 - ▶ $lis(i) = 1 \sim$ otherwise.
- ▶ This is exponential time.

Memoizing Version

If we simply memoize *lis* we get $\mathcal{O}(n^2)$ time

```
1  int lis(a) {
2      int i,j,m;
3      vi lis(a.length(),1);
4      for(i=1; i<a.length(); ++i)
5          for(j=0; j<i; ++j)
6              if (a[i] > a[j] && lis[i] <= lis[j])
7                  lis[j] = lis[i] + 1;
8      for(i=0, m=1; i<a.length(); ++i)
9          m = max(m,lis[i]);
10     return m;
11 }
```

Return the elements

- ▶ We can keep track of the “previous” elements to return the actual sequence.

```
1 vi lis(a) {
2     int i,j,m;
3     vi lis(a.length(),1);
4     vi prev(a.length(),-1);
5     for(i=1; i<a.length(); ++i)
6         for(j=0; j<i; ++j)
7             if (a[i] > a[j] && lis[i] <= lis[j]) {
8                 lis[j] = lis[i] + 1;
9                 prev[i] = j;
10            }
11     for(i=0, m=1; i<a.length(); ++i)
12         m = max(m,lis[i]);
13     return prev;
14 }
```


LIS is like LCS!

- ▶ We can use LCS code to solve this.
- ▶ Create a copy of A into B , sorting B .
 - ▶ Remove duplicates if you want a strictly increasing sequence.
 - ▶ Use sets

```
1 vi distinctCopy(vi a) {
2     vi new; set<int> seen;
3
4     for(auto it=a.begin(); it != a.end(); ++it)
5         if (seen.find(*it) == seen.end()) {
6             seen.insert(*it);
7             new.push_back(*it);
8         }
9
10    sort(new.begin(), new.end());
11    return new;
12 }
```