

Biomedica informatics: Lecture 9

Alberto Paoluzzi

Fri, Apr 17, 2015

Lecture 9: Longest Common Subsequence

Longest common subsequence (LCS) problem

Dynamic Programming Approach

LCS : Longest Common Subsequence

Let $X, Y \in Seq$ be the **sequences to compare**, and X_i, Y_j be the **subsequences of their first i, j characters**, respectively.

The integer function

$$LCS : Seq \times Seq \rightarrow Nat$$

gives the **integer length of longest common subsequence** of **any two (sub)sequences**, as follows:

Dynamic Programming Approach

LCS : Longest Common Subsequence

Let $X, Y \in Seq$ be the **sequences to compare**, and X_i, Y_j be the **subsequences of their first i, j characters**, respectively.

The integer function

$$LCS : Seq \times Seq \rightarrow Nat$$

gives the **integer length of longest common subsequence** of **any two (sub)sequences**, as follows:

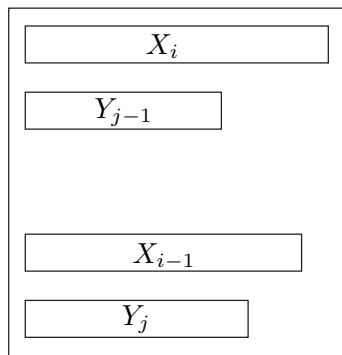
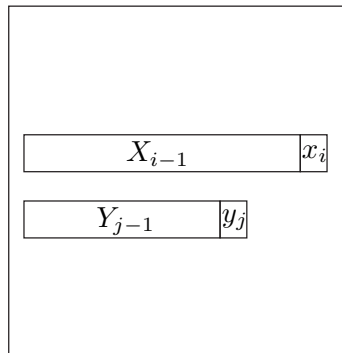
$$LCS(X_i, Y_j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1 & \text{if } x_i = y_j \\ \max(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j \end{cases}$$

Longest common subsequence

LCS function defined

$$x_i = y_j$$

$$LCS(X_i, Y_j) = LCS(X_{i-1}, Y_{j-1}) + 1$$



$$x_i \neq y_j$$

$$LCS(X_i, Y_j) = \max(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j))$$

Recursive implementation

just write down in Python the recursive equations above

```
"""python def cls(X,Y): i,j = len(X),len(Y) if i == 0 or j == 0: return 0 elif  
X[i-1] == Y[j-1]: return cls(X[:i-1],Y[:j-1])+1 else: return  
max(cls(X[:i],Y[:j-1]),cls(X[:i-1],Y[:j])) """
```

Recursive implementation

just write down in Python the recursive equations above

```
"""python def cls(X,Y): i,j = len(X),len(Y) if i == 0 or j == 0: return 0 elif  
X[i-1] == Y[j-1]: return cls(X[:i-1],Y[:j-1])+1 else: return  
max(cls(X[:i],Y[:j-1]),cls(X[:i-1],Y[:j])) """
```

```
"""python print cls("BASKETBALL","BASEBALL") == 8 """  
OK !
```


Recursive implementation

just write down in Python the recursive equations above

```
"""python def cls(X,Y): i,j = len(X),len(Y) if i == 0 or j == 0: return 0 elif
X[i-1] == Y[j-1]: return cls(X[:i-1],Y[:j-1])+1 else: return
max(cls(X[:i],Y[:j-1]),cls(X[:i-1],Y[:j])) """
```

```
"""python print cls("BASKETBALL","BASEBALL") == 8 """
```

OK !

```
"""python print
cls("ABRACADABRA","SUPERCALIFRAGILISTICESPIRALIDOSO") """
```

Recursive implementation

just write down in Python the recursive equations above

```
"""python def cls(X,Y): i,j = len(X),len(Y) if i == 0 or j == 0: return 0 elif
X[i-1] == Y[j-1]: return cls(X[:i-1],Y[:j-1])+1 else: return
max(cls(X[:i],Y[:j-1]),cls(X[:i-1],Y[:j])) """
```

```
"""python print cls("BASKETBALL","BASEBALL") == 8 """
```

OK !

```
"""python print
cls("ABRACADABRA","SUPERCALIFRAGILISTICESPIRALIDOSO") """
```

VERY long execution time ... WHY ?

Memoization technique

In computing, “memoization” is an **optimization technique** used primarily to speed up computer programs by having **function calls avoid repeating the calculation** of results for previously-processed input

- This technique of **saving values that have already been calculated** is frequently used

Memoization technique

In computing, “memoization” is an **optimization technique** used primarily to speed up computer programs by having **function calls avoid repeating the calculation** of results for previously-processed input

- This technique of **saving values that have already been calculated** is frequently used
- Memoization is a means of **lowering a function's time cost in exchange for space cost**; that is, memoized functions become optimized for speed in exchange for a higher use of computer memory space.

Memoization technique

In computing, “memoization” is an **optimization technique** used primarily to speed up computer programs by having **function calls avoid repeating the calculation** of results for previously-processed input

- This technique of **saving values that have already been calculated** is frequently used
- Memoization is a means of **lowering a function's time cost in exchange for space cost**; that is, memoized functions become optimized for speed in exchange for a higher use of computer memory space.
- An efficient LCS procedure requires: **saving the solutions** to one level of subproblem **in a table** so that the solutions are available to the next level of subproblems.

Length of the Longest Common Subsequence

computing the function $LCS : Seq \times Seq \rightarrow Nat$ with memoization

```
"""python def LCS(X, Y): m,n = len(X),len(Y) C = [[0] * (n+1) for i in
range(m+1)] for i in range(1, m+1): for j in range(1, n+1): if X[i-1] == Y[j-1]:
C[i][j] = C[i-1][j-1] + 1 else: C[i][j] = max(C[i][j-1], C[i-1][j]) return C """
```

Usage example — LCSfunction

```
““.python lll X = "AATCC" lll Y = "ACACG" lll m = len(X) lll n =  
len(Y) lll C = LCS(X, Y) ““
```

Usage example — LCSfunction

```
""python lll X = "AATCC" lll Y = "ACACG" lll m = len(X) lll n =  
len(Y) lll C = LCS(X, Y) ""
```

```
""python lll print C [[0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1], [0, 1, 1, 2, 2, 2],  
[0, 1, 1, 2, 2, 2], [0, 1, 2, 2, 3, 3], [0, 1, 2, 2, 3, 3]] ""
```


Usage example — LCSfunction

```
““.python \i\i\ X = "ATGGCCTGGAC" \i\i\ Y = "ATCCGGACC" \i\i\ m =  
len(X) \i\i\ n = len(Y) \i\i\ C = LCS(X, Y) ““
```

Usage example — LCSfunction

```
““.python lll X = "ATGGCCTGGAC" lll Y = "ATCCGGACC" lll m =
len(X) lll n = len(Y) lll C = LCS(X, Y) ““
```

```
““.python lll print C [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[0, 1, 2, 2, 2, 2, 2, 2, 2, 2], [0, 1, 2, 2, 2, 3, 3, 3, 3, 3], [0, 1, 2, 2, 2, 3,
4, 4, 4, 4], [0, 1, 2, 3, 3, 3, 4, 4, 5, 5], [0, 1, 2, 3, 4, 4, 4, 4, 5, 6], [0, 1, 2,
3, 4, 4, 4, 4, 5, 6], [0, 1, 2, 3, 4, 5, 5, 5, 5, 6], [0, 1, 2, 3, 4, 5, 6, 6, 6, 6],
[0, 1, 2, 3, 4, 5, 6, 7, 7, 7], [0, 1, 2, 3, 4, 5, 6, 7, 8, 8]] ““
```

Reading out an LCS

Backtracking on the table from the lower-right corner

```

"""python def backTrack(C, X, Y, i, j): if i == 0 or j == 0: return "" elif X[i-1]
== Y[j-1]: return backTrack(C, X, Y, i-1, j-1) + X[i-1] else: if C[i][j-1] > C[i-1][j]:
return backTrack(C, X, Y, i, j-1) else: return backTrack(C, X, Y, i-1, j) """

```

Usage example — backTrack function

```
"""python lll X = "AATCC" lll Y = "ACACG" lll m = len(X) lll n =  
len(Y) lll C = LCS(X, Y) """
```

Usage example — backTrack function

```
"""python lll X = "AATCC" lll Y = "ACACG" lll m = len(X) lll n =  
len(Y) lll C = LCS(X, Y) """
```

```
"""python lll print "Some LCS: 'Some LCS: 'AAC' """
```

Usage example — backTrack function

```
““.python lll X = "ATGGCCTGGAC" lll Y = "ATCCGGACC" lll m =  
len(X) lll n = len(Y) lll C = LCS(X, Y) ““
```

Usage example — backTrack function

```
““.python lll X = "ATGGCCTGGAC" lll Y = "ATCCGGACC" lll m =  
len(X) lll n = len(Y) lll C = LCS(X, Y) ““
```

```
““.python lll print "Some LCS: 'Some LCS: 'ATCCGGAC' ““
```

Reading out all LCSs

```

"""python def backTrackAll(C, X, Y, i, j): if i == 0 or j == 0: return
set([""]) elif X[i-1] == Y[j-1]: return set([Z + X[i-1] for Z in
backTrackAll(C, X, Y, i-1, j-1)]) else: R = set() if C[i][j-1] != C[i-1][j]:
R.update(backTrackAll(C, X, Y, i, j-1)) if C[i-1][j] != C[i][j-1]:
R.update(backTrackAll(C, X, Y, i-1, j)) return R """

```


Usage example — backTrackAll function

```
““.python lll X = "AATCC" lll Y = "ACACG" lll m = len(X) lll n =  
len(Y) lll C = LCS(X, Y) ““
```

Usage example — backTrackAll function

```
““.python lll X = "AATCC" lll Y = "ACACG" lll m = len(X) lll n =  
len(Y) lll C = LCS(X, Y) ““
```

```
““.python lll print "All LCSs: All LCSs: set(['ACC', 'AAC']) ““
```

Usage example — backTrackAll function

```
““.python lll X = "ATGGCCTGGAC" lll Y = "ATCCGGACC" lll m =  
len(X) lll n = len(Y) lll C = LCS(X, Y) ““
```

Usage example — backTrackAll function

```
““.python lll X = "ATGGCCTGGAC" lll Y = "ATCCGGACC" lll m =  
len(X) lll n = len(Y) lll C = LCS(X, Y) ““
```

```
““.python lll print "All LCSs: All LCSs: set(['ATCCGGAC']) ““
```

BLAST (Basic Local Alignment Search Tool)

BLAST program

Comparison of nucleotide or protein sequences

- The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences

BLAST program

Comparison of nucleotide or protein sequences

- The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences
- The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches

BLAST program

Comparison of nucleotide or protein sequences

- The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences
- The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches
- BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families

BLAST program

Comparison of nucleotide or protein sequences

- The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences
- The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches
- BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families
- BLAST makes it easy to examine a large group of potential gene candidates

BLAST

How to do Batch BLAST jobs

- BLAST makes it easy to examine a large group of potential gene candidates

TUTORIAL

BLAST

How to do Batch BLAST jobs

- BLAST makes it easy to examine a large group of potential gene candidates
- Most likely these are isolated as amplified products from a library of some sort

TUTORIAL

BLAST

How to do Batch BLAST jobs

- BLAST makes it easy to examine a large group of potential gene candidates
- Most likely these are isolated as amplified products from a library of some sort
- There is no need to manually cut and paste a 100 sequences in to the BLAST web pages

TUTORIAL

BLAST

How to do Batch BLAST jobs

- BLAST makes it easy to examine a large group of potential gene candidates
- Most likely these are isolated as amplified products from a library of some sort
- There is no need to manually cut and paste a 100 sequences in to the BLAST web pages
- Using the BLAST web pages it is possible to input "batches" of sequences into one form and retrieve the results

TUTORIAL

BLAST

How to do Batch BLAST jobs

- BLAST makes it easy to examine a large group of potential gene candidates
- Most likely these are isolated as amplified products from a library of some sort
- There is no need to manually cut and paste a 100 sequences in to the BLAST web pages
- Using the BLAST web pages it is possible to input "batches" of sequences into one form and retrieve the results
- There are two methods to do batch BLAST jobs

TUTORIAL

BLAST

How to do Batch BLAST jobs

- BLAST makes it easy to examine a large group of potential gene candidates
- Most likely these are isolated as amplified products from a library of some sort
- There is no need to manually cut and paste a 100 sequences in to the BLAST web pages
- Using the BLAST web pages it is possible to input "batches" of sequences into one form and retrieve the results
- There are two methods to do batch BLAST jobs
- The first is through the web interface and the second is using the standalone BLAST binaries and downloaded NCBI databases

TUTORIAL

BLAST

Example

- BLAST paper
- QuickStart: Example-Driven Web-Based BLAST Tutorial

FASTA (FAST Alignment)

FASTA

Example

FASTA stands for FAST-ALL, reflecting the fact that it can be used for a fast protein comparison or a fast nucleotide comparison

- This program achieves a high level of sensitivity for similarity searching at high speed

FASTA

Example

FASTA stands for FAST-ALL, reflecting the fact that it can be used for a fast protein comparison or a fast nucleotide comparison

- This program achieves a high level of sensitivity for similarity searching at high speed
- This is achieved by performing optimised searches for local alignments using a substitution matrix

FASTA

Example

FASTA stands for FAST-ALL, reflecting the fact that it can be used for a fast protein comparison or a fast nucleotide comparison

- This program achieves a high level of sensitivity for similarity searching at high speed
- This is achieved by performing optimised searches for local alignments using a substitution matrix
- The high speed of this program is achieved by using the observed pattern of word hits to identify potential matches before attempting the more time consuming optimised search

FASTA

Example

FASTA stands for FAST-ALL, reflecting the fact that it can be used for a fast protein comparison or a fast nucleotide comparison

- This program achieves a high level of sensitivity for similarity searching at high speed
- This is achieved by performing optimised searches for local alignments using a substitution matrix
- The high speed of this program is achieved by using the observed pattern of word hits to identify potential matches before attempting the more time consuming optimised search
- The trade-off between speed and sensitivity is controlled by the ktup parameter, which specifies the size of the word

FASTA

Example

FASTA stands for FAST-ALL, reflecting the fact that it can be used for a fast protein comparison or a fast nucleotide comparison

- This program achieves a high level of sensitivity for similarity searching at high speed
- This is achieved by performing optimised searches for local alignments using a substitution matrix
- The high speed of this program is achieved by using the observed pattern of word hits to identify potential matches before attempting the more time consuming optimised search
- The trade-off between speed and sensitivity is controlled by the ktup parameter, which specifies the size of the word
- Increasing the ktup decreases the number of background hits

FASTA

Example

FASTA stands for FAST-ALL, reflecting the fact that it can be used for a fast protein comparison or a fast nucleotide comparison

- This program achieves a high level of sensitivity for similarity searching at high speed
- This is achieved by performing optimised searches for local alignments using a substitution matrix
- The high speed of this program is achieved by using the observed pattern of word hits to identify potential matches before attempting the more time consuming optimised search
- The trade-off between speed and sensitivity is controlled by the ktup parameter, which specifies the size of the word
- Increasing the ktup decreases the number of background hits
- Not every word hit is investigated but instead initially looks for segment's containing several nearby hits

FASTA Web services

Both REST and SOAP web service interfaces are exposed

REST Sample clients are provided for a number of programming languages.

SOAP RPC/encoded SOAP service