

PROGRAMMING ASSIGNMENT 5

Subject : Numpy Features, Recursive Functions and Dynamic Programming

Advisor : Res. Assist. (Selim YILMAZ, Levent KARACAN, Burçak ASAL, Feyza Nur ÇUBUKÇUOĞLU)

Due Date : 24.12.2015

Introduction: The aim of this experiment is to introduce numpy features, recursion and dynamic programming. By the help of this experiment students will learn the basics of recursion and dynamic programming by utilizing different numpy features. Dynamic programming is an efficient algorithmic paradigm that breaks up a complex problem into smaller sub-problems, stores the answers of these sub-problems and uses these answers to solve the whole problem. This method has a big advantage over the classic recursive method because it avoids recomputing the same sub-problems over and over again.

Genetic Sequence Alignment



Biology review: A genetic sequence is a string of biological macromolecules referred to together as the DNA bases. A gene is a genetic sequence that contains the information needed to construct a protein. All of your genes taken together are referred to as the human genome, a blueprint for the parts needed to construct the proteins that form your cells. Each new cell produced by your body receives a copy of the genome. This copying process, as well as natural wear and tear, introduces a small number of changes into the sequences of many genes. Among the most common changes are the substitution of one base for another and the deletion of a substring of bases; such changes are generally referred to as point mutations. As a result of these point mutations, the same gene sequenced from closely related organisms will have slight differences.

The Problem: There are a huge number of genetic sequences that biologists and computer scientists have laboriously determined (and published) from many organisms (including humans). We can leverage this information. In this experiment, you are expected to compare and align a two genetic sequences (strings) by using a distance measure, which is the edit distance. Each string are composed of a four-letter alphabet (Adenine (A), Thymine (T), Guanine (G), Cytosine (C)).

Edit Distance: This distance measure is a popular method widely used in many applications such as spell checking, speech recognition, computational linguistics and so on. This method compares two genetic strings and aligns them by selecting optimal solution in multiple possible solutions. It uses three-rules penalty system, in which a solution gaining minimum penalty score is the best solution(alignment). These three rules are defined as follows;

Operation	Cost
Insert a gap	2
Align two characters that mismatch	1
Align two characters that match	0

The example below shows two possible solutions(alignments) of the strings;

X = "AACAGTTACC" and Y = "TAAGGTCA"

X	Y	Cost	X	Y	Cost
A	T	1	A	T	1
A	A	0	A	A	0
C	A	1	C	-	2
A	G	1	A	A	0
G	G	0	G	G	0
T	T	0	T	G	1
T	C	1	T	T	0
A	A	0	A	-	2
C	-	2	C	C	0
C	-	2	C	A	1
8			7		

The method can insert gaps in either string (and any index position for each string) to make them have same length. In the example, the first alignment has a score of 8, while the second alignment has a score of 7. The edit-distance is the score of the best possible alignment(solution) between the two genetic sequences(strings) over all possible alignments. In the example, the second alignment is the optimal, so the edit-distance between the two strings (X and Y) is 7.

Possible Recursive Solution

The edit distance between two strings X and Y can be determined by solving that edit distance sub-problems on smaller suffixes of the two strings. Let we first define the notations that are used in the paper;

1. $X[i]$: Defines character i of the X string.
2. $X[i..M]$: Defines the suffix of X consisting of the characters $X[i]$, $X[i+1]$, ..., $X[M-1]$.
3. $Opt[i][j]$: Defines the edit distance of $X[i..M]$ and $Y[j..N]$.

For example, let's assume we are given two strings X = "AACAGTTACC" and Y = "TAAGGTCA" of length M = 10 and N = 8, respectively. Then, $X[2]$ is 'C', $X[2..M]$ is "CAGTTACC", and $Y[8..N]$ is the empty string. The edit distance of x and y is $Opt[0][0]$.

Now consider the first pair of characters in an optimal alignment of $X[i..M]$ with $Y[j..N]$. There are three possibilities:

1. **Case of matching $X[i]$ with $Y[j]$:** The method pays a penalty of either 0 or 1, depending on whether $X[i]$ equals $Y[j]$ or not. $X[i + 1..M]$ and $Y[j + 1..N]$ still should be aligned, which is equal to $Opt[i + 1][j + 1]$
2. **Case of matching $X[i]$ with a gap:** The method pays a penalty of 2 for a gap. $X[i + 1..M]$ with $Y[j..N]$ still should be aligned, which is equal to $Opt[i + 1][j]$
3. **Case of matching $Y[j]$ with a gap:** The method pays a penalty of 2 for a gap. $X[i..M]$ with $Y[j + 1..N]$ still should be aligned, which is equal to $Opt[i][j + 1]$

So, the formula below can be calculated recursively as follows;

$$Opt[i][j] = \min(Opt[i + 1][j + 1] + 0/1, Opt[i + 1][j] + 2, Opt[i][j + 1] + 2) \quad (1)$$

assuming $i < M$ and $j < N$.

The equations below should also be taken into account, as one of two strings can be empty string;

1. $Opt[M][j] = 2(N - j)$
2. $Opt[i][N] = 2(M - i)$

Bottom-up Dynamic Programming Approach

A direct usage of the recursive method will work but it is extremely inefficient because of recalculating so many recursive calls for sub-problems again and again. Instead of this, a bottom-up dynamic approach is much more efficient. This new approach uses a 2D matrix to store sub-solutions $Opt[i + 1][j + 1]$, $Opt[i + 1][j]$, and $Opt[i][j + 1]$ **in the right order** to calculate $Opt[i][j]$ by avoiding recalculation. The example matrix has been shown below;

		0	1	2	3	4	5	6	7	8
	X/Y	T	A	A	G	G	T	C	A	-
0	A	7	8	10	12	13	15	16	18	20
1	A	6	6	8	10	11	13	14	16	18
2	C	6	5	6	8	9	11	12	14	16
3	A	7	5	4	6	7	9	11	12	14
4	G	9	7	5	4	5	7	9	10	12
5	T	8	8	6	4	4	5	7	8	10
6	T	9	8	7	5	3	3	5	6	8
7	A	11	9	7	6	4	2	3	4	6
8	C	13	11	9	7	5	3	1	3	4
9	C	14	12	10	8	6	4	2	1	2
10	-	16	14	12	10	8	6	4	2	0

While calculating the example matrix above, you should use **numpy** array to utilize it's features. It will make easier to use different operations on 2D array and increase performance of your implementation.

Recovering the solution

The final step of the implementation is printing the optimal alignment. To do this, you should re-discover the path of choices (highlighted in red in the example matrix above) from $Opt[0][0]$ to $Opt[M][N]$ by considering three possibilities;

1. If optimal alignment matches $X[i]$ up $Y[j]$, then $Opt[i][j] = Opt[i+1][j+1] + 0/1$,
(Whether $X[i]$ equals $Y[j]$ or not) (Move diagonally)
2. If optimal alignment matches $X[i]$ up with a gap, then $Opt[i][j] = Opt[i+1][j] + 2$
(Move down)
3. If optimal alignment matches $Y[j]$ up with a gap, then $Opt[i][j] = Opt[i][j+1] + 2$
(Move right)

Implementation Details

You should implement both the recursive and dynamic methods. Additionally, you should also implement a program that draws the curve of $((M+N)/\text{Time(in seconds)})$ for the given example input files by using **matplotlib** demonstrating the complexity of the algorithm. A sample plot is given below;

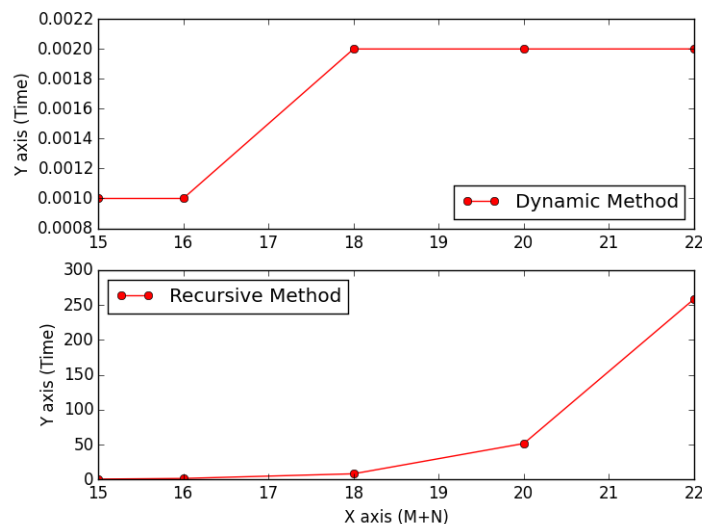


Figure 1: An example for the graph

Input-Output Format

You should read two strings from the input file and write the optimal alignment to the output file. An example of console, input and output file format has been stated below;

Your-program Input-path Output-path

Table 1: An example input file

AACAGTTACC
TAAGGTCA

Table 2: An example output file

Edit Distance = 7		
A	T	1
A	A	0
C	-	2
A	A	0
G	G	0
T	G	1
T	T	0
A	-	2
C	C	0
C	A	1

Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (<https://piazza.com/hacettepe.edu.tr/fall2015/bbm103>) and you are supposed to be aware of everything discussed in Piazza.
- The submissions whose upload score is 0 will not be considered for evaluation.
- You will submit your work from <https://submit.cs.hacettepe.edu.tr/index.php> with the file hierarchy as below:

This file hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system)

```
→ <student id>  
  → hw5-recursion.py  
  → hw5-dynamic.py  
  → hw5-plot.py
```

This file hierarchy must be zipped before submitted (Not .rar , only .zip files are supported by the system).

Additional Note

If you find multiple optimal solutions (or multiple minimum values in equation (1)), you should choose one optimal solution. To do this, first implement the three-rules in the same order with equation (1), and then choose first-corresponding minimum value.

References

This assignment was adapted from the assignment by T.Clarke, R.Sedgewick, S.Vafai, K.Wayne and B.Brown.