SUBSEQUENCES AND SOME REAL WORLD APPLICATIONS

HARSHITA GUPTA 2020101078
AMEYA SHARMA 2020101059
YEDURU RUPASREE 2020101097

ACKNOWLEDGEMENT

Framing of our project would not have been possible without the kind support and help of our esteemed Professor and teaching assistants and IIIT Hyderabad which gave us a marvellous opportunity to do this project. We would like to extend our sincere thanks to all of them. We would like to profusely thank our professor, Prof. Laxmi Burra for her guidance and constant supervision. We are also very grateful to our teaching assistants whose valuable guidance and honest criticism has helped us to complete our project successfully. The preparation of this project was a wonderful experience. We are really pleased that all the above-mentioned people lifted our spirits and supported us throughout the project.

ABSTRACT

The aim of this project is to explain subsequence and its related theorems, and elaborate on some real-world applications of subsequences. The concepts of subsequences are used in different fields including computer science, computational linguistics and bioinformatics, be it for DNA sequencing or File Compression. First, the definitions and related theorems are discussed, followed by how they are applied in the field of bioinformatics and software development. The applications include - DNA alignment using Longest Common Subsequence, file comparison and version control systems, and file compression.

TABLE OF CONTENTS

SUBSEQUENCES

- At the first place ,a sequence of real numbers is a function defined on the set N={1,2,3…..} of natural numbers whose range is contained in the set R of real numbers.
- Let P = (p_n) be a sequence of real numbers and let $n_1<n_2<n_3<\ldots<n_k<..$ be a strictly increasing sequence of positive integers. Then the sequence P'=$\{p_{nj}\}$ is called a Subsequence of P.
- For example,$(a_2 , a_4 , a_6 , \ldots)$ is a subsequence of $(a_1 , a_2 , a_3 , a_4 , \ldots)$.

So is $(a_1 , a_{10} , a_{100} , a_{1000} , \ldots)$.

- In other words, a subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

  For example, the sequence { A,B,D } is a subsequence of { A,B,C,D,E,F } obtained after removal of elements C,E and F.

- The relation of one sequence being the subsequence of another is a Preorder (A preorder is a binary relation that is reflexive and transitive).

- A subsequence which consists of a consecutive run of elements from the original sequence, such as <A,B,C> from <A,B,C,D,E,F> is a Substring(a continuous part or subpart of a string or a sequence).

- Is a sequence subsequence of itself ?
  Yes, a sequence is one of its own subsequence, which is similar to an idea that any set is a subset of itself.

- If we want to specify that a subsequence is *not* the entire sequence, we refer to it as a *proper* subsequence. That is, every subsequence except for the sequence itself is a proper subsequence.

- Does a subsequence have to be infinite? Or can it be finite too?
  Yes, since sequences are defined to be infinite and subsequences are sequences at the first place they need to be infinite too. A subsequence is an infinite ordered subset of a sequence.

- Given a sequence $X = \{x_1,x_2,....x_m\}$, we have 2^m subsequences of X.
  Explanation : For any sequence $X = \{x_1,x_2,....x_m\}$, you can "choose" subsequences of length 0,1,2,...,m, mathematically it is $C(m,0) + C(m,1) + ... C(m,m)$ which is equal to 2^m.

- If you have a sequence S, what happens when we add a new element x to the end of S ?

All the subsequences of S are still subsequences of the new sequence. So are all those subsequences with x added at the end, that implies every time you add an element, you double the number of subsequences.

Bolzano – Weierstrass Theorem

Every bounded sequence has a convergent subsequence.

The Bolzano–Weierstrass theorem is named after mathematicians Bernard Bolzano and Karl Weierstrass. It was actually first proved by Bolzano in 1817 as a lemma in the proof of the intermediate value theorem. Some fifty years later the result was identified as significant in its own right, and proved again by Weierstrass. It has since become an essential theorem of analysis.

Proof for Bolzano – Weierstrass Theorem

Lemma : Every infinite sequence $(x_{\{n\}})$ in R^1 has a monotone subsequence.

Let us call a positive integer-valued index n of a sequence the "peak of the sequence" when both of the following are true:

1) n is less than some other index m and

2) $x_n > x_m$.

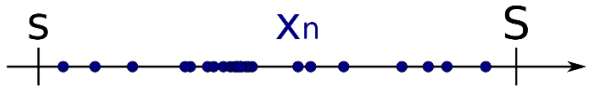
Suppose first that the sequence has infinitely many peaks, which means there is a subsequence with the following indices $n_1 < n_2 < n_3 < n_4 \ldots < n_j < \ldots\ldots$ and the following terms $x_{n1} > x_{n2} > x_{n3} > \ldots > x_{nj} > \ldots$ So, the infinite sequence (x_n) in R^1 has a monotone subsequence, which is (x_{nj}).

But suppose now that there are only finitely many peaks, let N be the final peak and let the first index of a new subsequence (x_{nj}) be set to $n_1 = N + 1$. Then n_1 is not a peak, since n_1 comes after the final peak, which implies the existence of n_2 with $n_1 < n_2$ and $x_{n1} \leq x_{n2}$. Again, n_2 comes after the final peak, hence there is an n_3 where $n_2 < n_3$ with $x_{n2} \leq x_{n3}$. Repeating this process leads to an infinite non-decreasing subsequence $x_{n1} \leq x_{n2} \leq x_{n3} \leq \ldots\ldots$, thereby proving that every infinite sequence (x_n) in R^1 has a monotone subsequence.

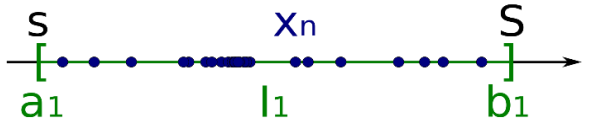
Now suppose one has a bounded sequence in R^1; by the lemma proven above there exists a monotone subsequence, necessarily bounded. It follows from the monotone convergence theorem that this subsequence must converge.

Finally, the general case R^n, can be reduced to the case of R^1as follows: given a bounded sequence in R^n, the sequence of first coordinates is a bounded real sequence, hence it has a convergent subsequence. One can then extract a sub-subsequence on which the second coordinates converge, and so on, until in the end we have passed from the original sequence to a subsequence n times—which is still a subsequence of the original sequence—on which each coordinate sequence converges, hence the subsequence itself is convergent.

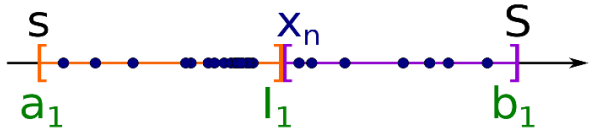
Alternate Proof for Bolzano – Weierstrass Theorem



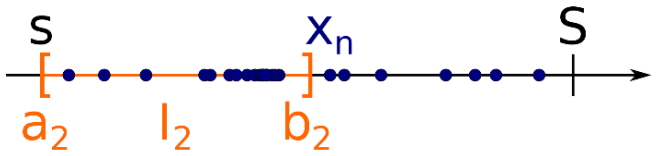
1.This is a bounded sequence x_n. It has S as it's upper bound and s as it's lower bound.



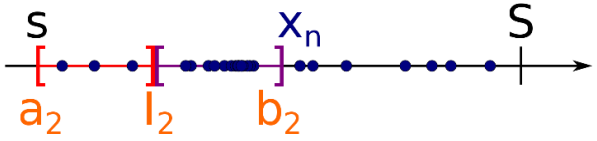
2. We take $I_1 = [s,S]$ as the first interval for the sequence of nested intervals.



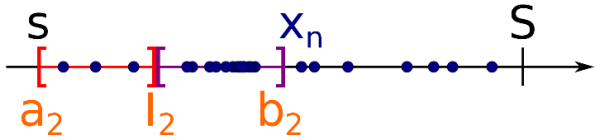
3. Then we split I_1 at the mid into two equally sized subintervals.



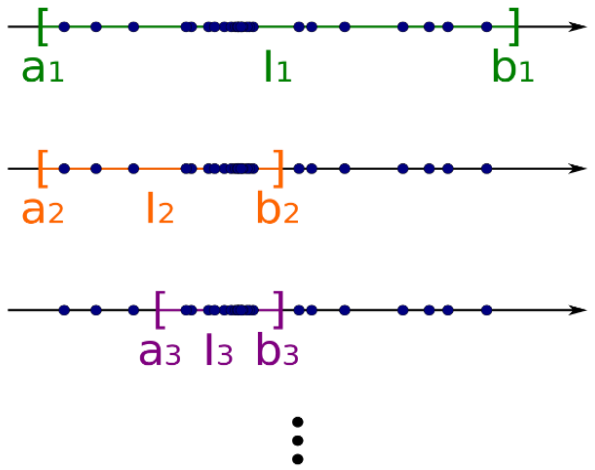
4. We take this subinterval as the second interval I_2 of the sequence of nested intervals which contains infinitely many members of $(x_n)n \in N$. Because each sequence has infinitely many members, there must be at least one subinterval which contains infinitely many members.



5. Then we split I_3 again at the mid into two equally sized subintervals.



6. Again we take this subinterval as the third subinterval I_3 of the sequence of nested intervals, which contains infinitely many members of $(x_n)\, n \in N$.



7. We continue this process infinitely many times. Thus, we get a sequence of nested intervals.

Because we half the length of an interval at each step, the limit of the interval's length is zero. Thus, there is a number x which is an interval I_n. Now we show that x is an accumulation point of (x_n)

Take a neighbourhood U of x. Because the length of the intervals converges to 0, which is a subset of U. Because I_n contains by construction infinitely many members of (x_n) and $I_n \in U$, also U contains infinitely many members of (x_n). This proves that x is an accumulation point of (x_n). Thus, there is a subsequence of (x_n) which converges to x.

Hence, proved.

LIMIT SUPERIOR AND INFERIOR

- Limit inferior and limit superior of a sequence can be thought of as limiting (i.e., eventual and extreme) bounds on the sequence.
- The upper limit is called limit superior, supremum limit, limit supremum, outer limit, etc.
- The limit superior of a sequence x_n is denoted by

  $\lim_{n\to\infty} sup\, x_n$ or $\overline{\lim_{n\to\infty}}\, x_n$.
- The lower limit is called limit inferior, infimum limit, limit infimum, inner limit etc.
- The limit inferior of a sequence x_n is denoted by

  $\lim_{n\to\infty} \inf x_n$ or $\underline{\lim_{n\to\infty}}\, x_n$.

SUBSEQUENTIAL LIMIT

Given a sequence (a_n), we say that L is a subsequential limit of (a_n) if there is a subsequence (a_{nk}) converging to L. Equivalently, L is a subsequential limit of (a_n) if for every $\varepsilon > 0$, infinitely many n satisfy $|a_n - a| < \varepsilon$.

One thing to remember is that every cluster point is a subsequential limit, but the converse is not true.

Erdos-Szekeres theorem

Any sequence of real numbers which is of length (m-1)(n-1)+1 contains a monotonically increasing sequence of length m or a monotonically decreasing sequence of length n ,where $m,n \in N$.

Proof of Erdos- Szekeres theorem:

Let $a_1, a_2, a_3 \ldots\ldots a_N$ be a sequence of N real numbers where N=(m-1)(n-1)+1. Let us start assigning coordinates(Xi,Yi) for every a_i, where

Xi=length of longest increasing subsequence ending at a_i

Yi=length of longest decreasing subsequence starting at a_i

For any two numbers a_i , a_j (where i<j),

if $a_i \leq a_j$ then Xi<Xj as sequence ending at ai can be extended by aj.

if $a_i \geq a_j$ then Yi>Yj as sequence starting at aj can be extended by ai.

From this we can conclude that if($i \neq j$) then $(Xi,Yi) \neq (Xj,Yj)$ and there must be N distinct co-ordinates. But there are (m-1)(n-1) possible coordinates (Xi,Yi) if Xi<m and Yi<n for all i. So ,by pigeon hole principle there must be a value of i for which Xi or Yi is out of range. If Xi is out of range then a_i is a part of increasing subsequence of length m ,if Yi is out of range then a_i is a part of decreasing subsequence of length n.

LONGEST COMMON SUBSEQUENCE

Given two sequences *X* and *Y*, a sequence *Z* is said to be a *common subsequence* of *X* and *Y*, if *Z* is a subsequence of both *X* and *Y*. For example, if

X={A,C,B,D,E,G,C,E,D,B,G}

Y={B,E,G,C,F,E,U,B,K} and

Z={B,E,E}

Then Z is said to be a common subsequence of X and Y. This would not be the longest common subsequence, since Z only has 3 elements and the common subsequence {B,E,E,B} has length 4. The LCS of X and Y is {B,E,G,C,E,B}.

The problem of LCS is to find longest common subsequence of given subsequences often just two sequences.

Example: ABCBDAB,BDCABA , BCBA and BDAB are two LCSs of length 4

Methods to find LCS of given two subsequences:

1.Brute force method

2.Recursion

3.Dynamic programming

1.Brute force method:

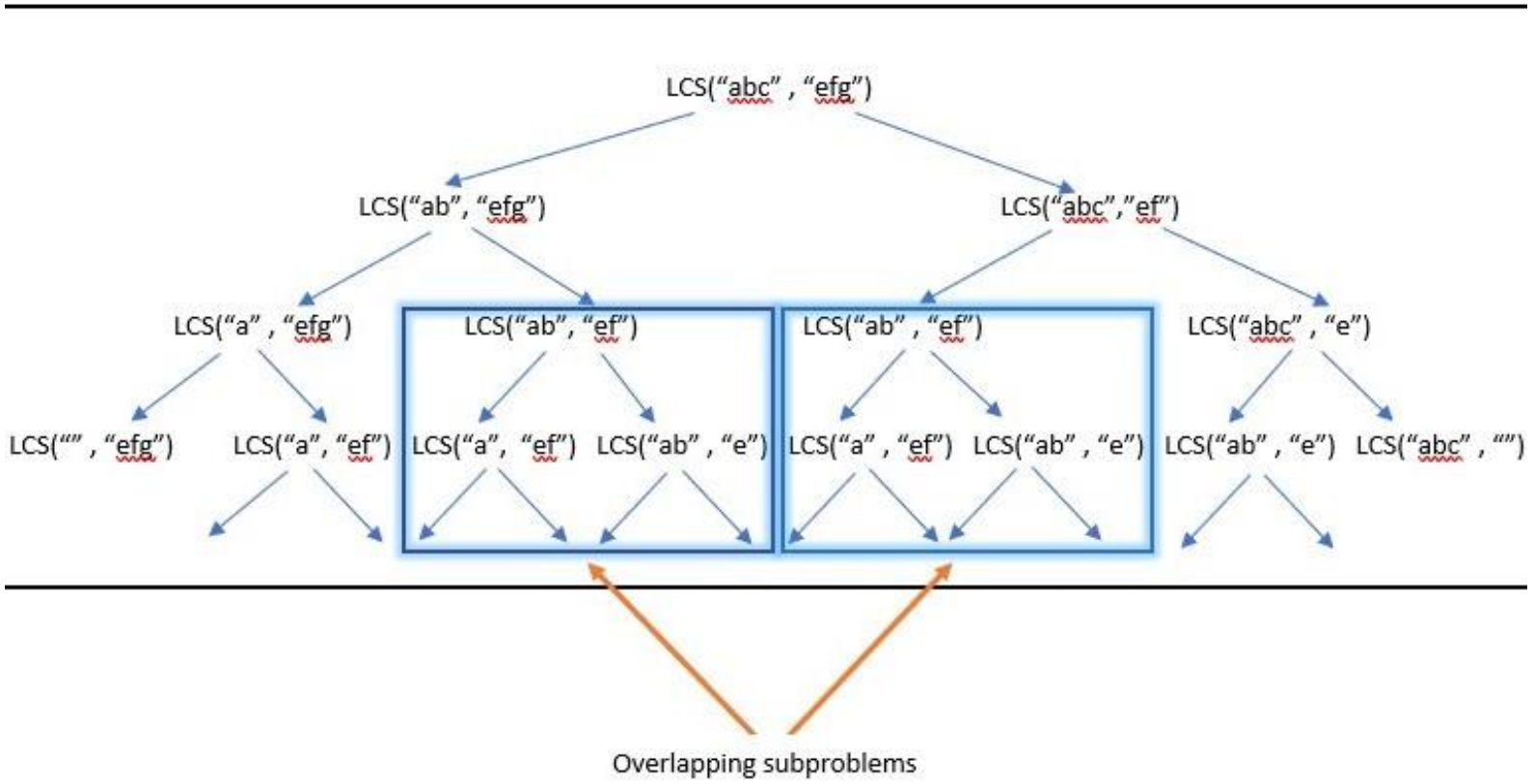
For every subsequence of first sequence,check if it is a subsequence of second sequence.
Suppose the two sequences has length m,n respectively first sequence will have 2^m subsequences. Each check takes O(n) time as we scan second subsequence for first element, then scan for second element and so on.
In worst case time complexity would be $O(n2^m)$.

2.Recursion method:

Let $a_1, a_2, a_3 \ldots\ldots a_m$, $b_1, b_2\ b_3 \ldots\ldots b_n$ be two sequences.

LCS ALGORITHM:

$$LCS(i,j)= \begin{cases} 0 \text{ if } i=0, \text{ or } j=0 \\ LCS(i-1,j-1)+1 \text{ if } i,j>0 \text{ and } a_i = b_i \\ \max(LCS(i-1,j),LCS(i,j-1)) \text{ if } a_i \neq b_i \end{cases}$$



In the above recursion tree, we can observe that there are many sub problems which are solved again and again.It will have time complexity $O(2^n)$.So this problem of overlapping cab be avoided by using memoization.

3.DYNAMIC PROGRAMMING:

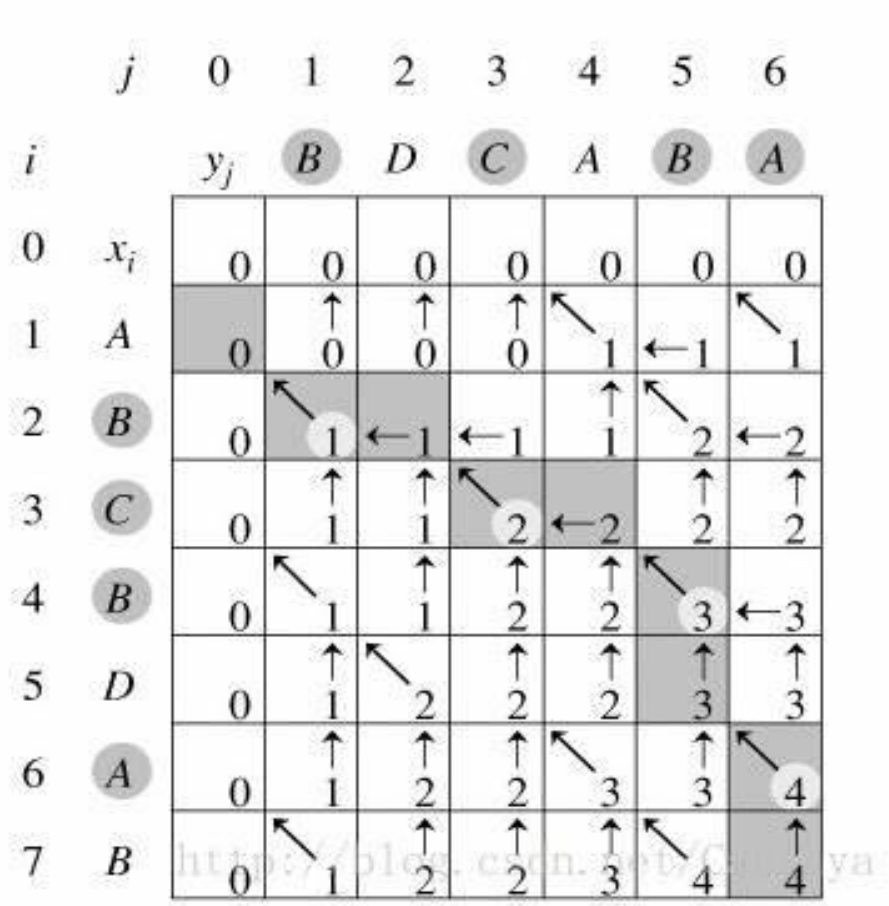
In dynamic programming we construct a 2D table L[m+1][n+1].The value of L[m][n] contains the length of LCS.

Algorithm:

$$LCS[i][j]=\begin{cases} 0 \text{ if } i=0 \text{ or } j=0 \\ LCS[i-1][j-1]+1 \text{ if } A[i-1]==B[j-1] \\ \max\{LCS[i][j-1],LCS[i-1][j]\} \text{ if } A[i-1] \neq B[j-1] \end{cases}$$

To find an LCS follow the arrows,for each diagonal arrow there is a member of LCS

Time complexity : O(m*n)



LONGEST INCREASING SUBSEQUENCE PROBLEM

- The Longest increasing subsequence problem is to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible.
- For example, in the sequence, 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15, the longest increasing subsequence is 0, 2, 6, 9, 11, 15. This subsequence has length six ; the input sequence has no seven-member increasing subsequence. This longest increasing subsequence in this example is not the only solution, for instance, 0, 4, 6, 9, 11, 15 ; 0, 2, 6, 9, 13, 15 ; 0, 4, 6, 9, 13, 15 are other increasing subsequences of equal length in the same input sequence.
- The longest increasing subsequence problem is closely related to the longest common subsequence problem, the longest increasing subsequence of a sequence S is the longest common subsequence of S and T, where T is the result of sorting S.
- The longest increasing subsequence problem is solvable in time $O(n \log n)$, where n denotes the length of the input sequence.
- Length bounds :According to the Erdős–Szekeres theorem, any sequence of n^2+1 distinct integers has an increasing or a decreasing subsequence of length $n + 1$.

ALGORITHM

Recursive Approach

Let Arr[0..n-1] be the input array and L[i] be the length of the LIS ending at index i such that Arr[i] is the last element of the LIS.

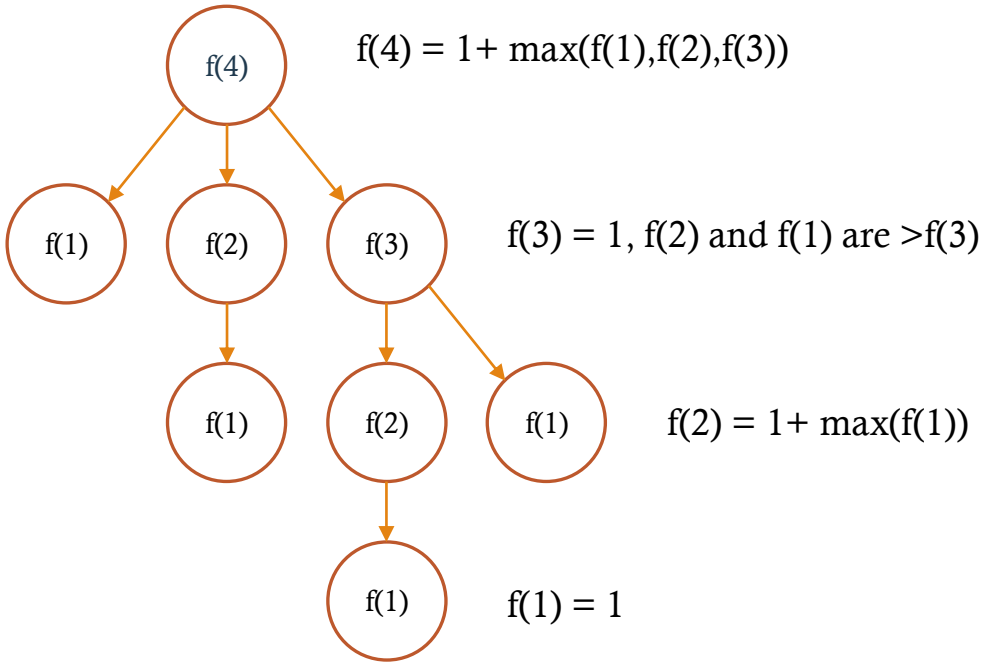
Then, L[i] can recursively written as:

L[i] = 1+max(l[j]) where 0< j < I and Arr[j] < Arr[i] ; or

L[i] = 1, if no such j exists.

To find the LIS for a given array, we need to return max(L[i]) where 0 < i < n.

Input : Arr[ ]={3,10,2,11}
f(i) : denotes the length of LIS of subarray ending at index 'i'.



i=1, j=0

iterator	j	i		
Arr[]	3	10	2	11
LIS	1	2	1	1

i=2, j=0

iterator	j		i	
Arr[]	3	10	2	11
LIS	1	2	1	1

i=2, j=1

iterator		j	i	
Arr[]	3	10	2	11
LIS	1	2	1	1

i=3, j=0

iterator	j			i
Arr[]	3	10	2	11
LIS	1	2	1	2

i=3, j=1

iterator		j		i
Arr[]	3	10	2	11
LIS	1	2	1	3

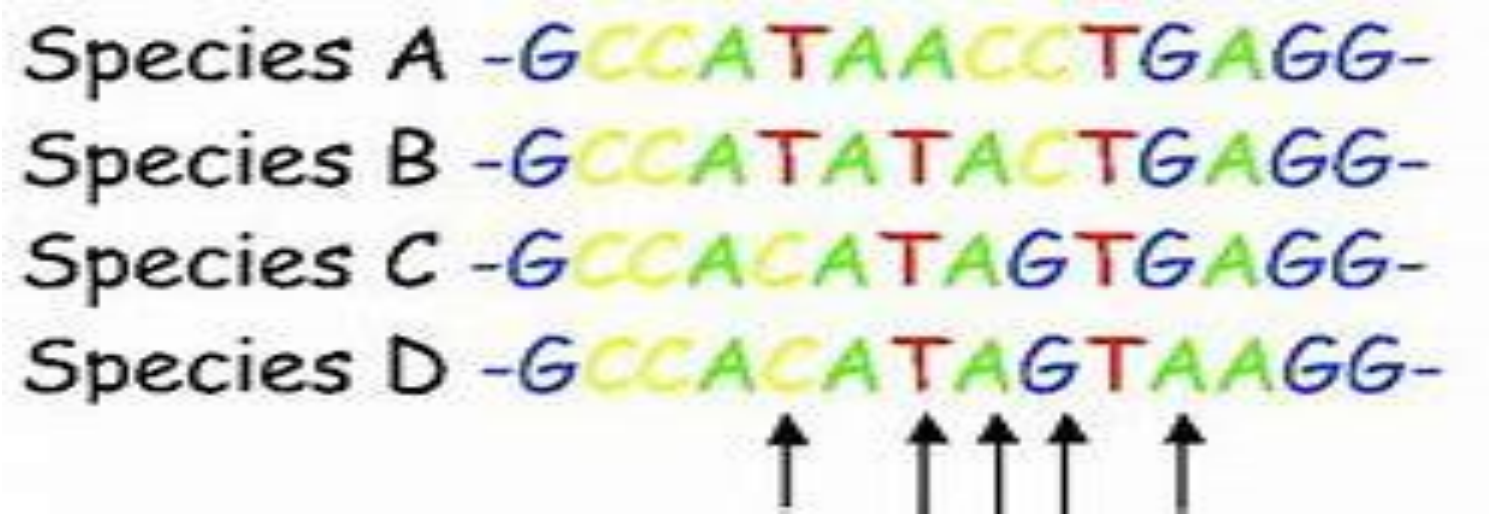
i=3, j=2

iterator			j	i
Arr[]	3	10	2	11
LIS	1	2	1	3

APPLICATIONS OF SUBSEQUENCES IN MOLECULAR BIOLOGY

Subsequences have applications in molecular biology to compare, analyse and store DNA,RNA and protein sequences.

DNA sequences(genes) can be represented as sequences of four letters ACGT corresponding to the four molecules forming DNA. When biologists find new sequences, they typically want to know what other sequences it is most similar to. One way of computing how similar two sequences are is to find their longest common subsequence. Longer the LCS, higher the similarity.



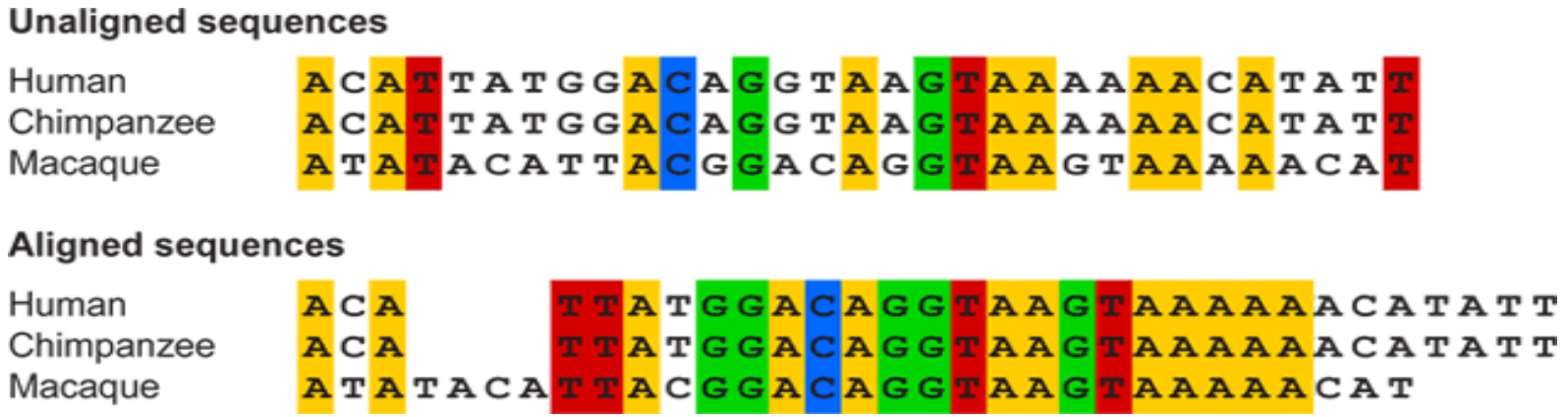
Comparison of two sequences ,known as sequence comparison , either from same organism or different organism is an important task in molecular biology, it is helpful in

- Inferring evolutionary history and relatedness of species.
- Predicting structure and function of proteins.
- Locating common subsequences in genes/proteins to identify common motifs.
- As a subproblem in genome assembly for DNA sequencing.

In order to perform sequence comparison ,we first perform sequence alignment.

SEQUENCE ALIGNMENT USING LCS

A Sequence alignment is a process of aligning two sequences to achieve maximum levels of identity between them.



Let S1=ATCAGTGC ,S2=TCTGACA

LCS(S1,S2)=TCTGC

Since sequences from different organisms may be of different sizes ,an alignment might require addition of extra spaces or a special character.

After alignment,S1=ATCAGTG_ C_

S2=_TC_ _TGACA

FILE COMPARISON

In computing, file comparison is the calculation and display of the differences and similarities between data objects, typically text files such as source code. The methods, implementations, and results are typically called a diff, after the Unix diff utility.

The central algorithm of *diff* solves the 'longest common subsequence problem' to find the lines that do not change between files. Any data not in the longest common subsequence is presented as an insertion or deletion. It works by finding a longest common subsequence of the lines of the two files; any line in the subsequence has not been changed, so what it displays is the remaining set of lines that have changed. In this instance of the problem we should think of each line of a file as being a single complicated character in a string.

Solving the longest common subsequence problem : No uniformly good way of solving the longest common subsequence problem is known. The simplest idea—go through both files line by line until they disagree, then search forward somehow in both until a matching pair of lines is encountered, and continue similarly.

VERSION CONTROL SYSTEMS

LCS is widely used by revision control systems such as Git for reconciling multiple changes made to a revision controlled collection of files.

In software engineering, **version control** is a class of systems responsible for managing changes to computer programs, documents, large web sites, or other collections of information. Version control is a component of software configuration management.

Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision".

For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

Revision Control System (RCS) is an early version control system (VCS). It is a set of UNIX commands that allow multiple users to develop and maintain program code or documents. With RCS, users can make their own revisions of a document, commit changes, and merge them. RCS was originally developed for programs but is also useful for text documents or configuration files that are frequently revised.
RCS stores a set of edit instructions to go back to an earlier version of the file.

RCS revolves around the usage of "revision groups" or sets of files that have been checked-in via the co (checkout) and ci (check-in) commands Once checked in, RCS stores revisions in a tree structure that can be followed so that a user can revert a file to a previous form if necessary.

Git is a distributed version-control system for tracking changes in any set of files, originally designed for coordinating work among programmers cooperating on source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

In version control, **merging** (also called integration) is a fundamental operation that reconciles multiple changes made to a version-controlled collection of files. Most often, it is necessary when a file is modified on two independent branches and subsequently merged. The result is a single collection of files that contains both sets of changes.

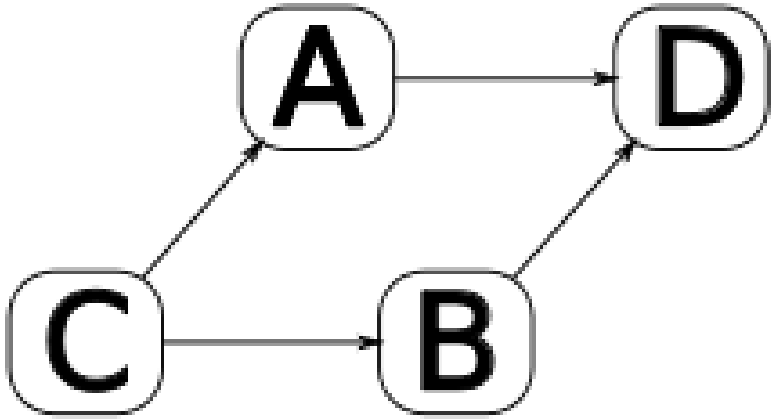
Three-way merge

C is the origin, A and B are derivatives of C, and D is the new output version

A three-way merge is performed after an automated difference analysis between a file "A" and a file "B" while also considering the origin, or common ancestor, of both files "C". It is a rough merging method, but widely applicable since it only requires one common ancestor to reconstruct the changes that are to be merged.

The three-way merge looks for sections which are the same in only two of the three files. In this case, there are two versions of the section, and the version which is in the common ancestor "C" is discarded, while the version that differs is preserved in the output. If "A" and "B" agree, that is what appears in the output. A section that is the same in "A" and "C" outputs the changed version in "B", and likewise a section that is the same in "B" and "C" outputs the version in "A".

Sections that are different in all three files are marked as a conflict situation and left for the user to resolve.



An interesting real-world application of LCS is Patience Diff, (In computing, the utility **diff** is a data comparison tool that computes and displays the differences between the contents of files) which is used in the **Bazaar version control system** .

The regular diff algorithm involves computing the LCS (Longest Common Subsequence) between two documents.

In a nutshell, the algorithm is:

1) Find unique lines which are common to both documents.
2) Take all such lines from the first document and order them according to their appearance in the second document.
3) Compute the LCS of the resulting sequence, getting the longest matching sequence of lines, a correspondence between the lines of two documents.
4) Recurse the algorithm on each range of lines between already matched ones.

FILE COMPRESSION

File compression is a process in which the size of a file is reduced by re-encoding the file data to use fewer bits of storage than the original file. A fundamental component of file compression is that the original file can be transferred or stored, recreated, and then used later (with a process called decompression).

Today, there are many different types of algorithms and implementations that allow the everyday user to compress files, two of them which is suited to many applications are : Lossless and Lossy Compression

Lossy compression algorithms reduce the size of files by discarding the less important information in a file, which can significantly reduce file size but also affect file quality.

Lossless compression reduces file size without removing any bits of information. This format works by removing redundancies within data to reduce the overall file size. With lossless, it is possible to perfectly reconstruct the original file. Lossless compression reduces bits by identifying and eliminating statistical redundancy.

Lossy compression methods include DCT (Discreet Cosine Transform), Vector Quantisation and Transform Coding while Lossless compression methods include RLE (Run Length Encoding), string-table compression, LZW (Lempel Ziff Welch) and zlib.

LZW ALGORITHM

Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. The algorithm is simple to implement and has the potential for very high throughput in hardware implementations. It is the algorithm of the widely used Unix file compression utility compress and is used in the GIF image format. It is lossless, meaning no data is lost when compressing.

The idea relies on reoccurring patterns to save data space. LZW is the foremost technique for general purpose data compression due to its simplicity and versatility. It is the basis of many PC utilities that claim to "double the capacity of your hard drive".

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression.
Characteristic features of LZW includes :

- LZW compression uses a code table, with 4096 as a common choice for the number of table entries. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code table.
- Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

Example: ASCII code. Typically, every character is stored with 8 binary bits, allowing up to 256 unique symbols for the data. This algorithm tries to extend the library to 9 to 12 bits per character. The new unique symbols are made up of combinations of symbols that occurred previously in the string. It does not always compress well, especially with short, diverse strings. But is good for compressing redundant data, and does not have to save the new dictionary with the data.

Implementation

The idea of the compression algorithm is the following: as the input data is being processed, a dictionary keeps a correspondence between the longest encountered words and a list of code values. The words are replaced by their corresponding codes and so the input file is compressed. Therefore, the efficiency of the algorithm increases as the number of long, repetitive words in the input data increases.

PSEUDOCODE

1. Initialize table with single character strings
2. P = first input character
3. WHILE not end of input stream
    1. C = next input character
    2. IF P + C is in the string table
       P = P + C
    3. ELSE
       Output the code for P
       Add P + C to the string table
       P = C
4. END WHILE
5. Output code for P

Example: The string to be encoded is "TOBEORNOTTOBEORTOBEORNOT#".

Current Sequence	Next Char	Output		Extended Dictionary		Comments
		Code	Bits			
NULL	T					
T	O	20	10100	27:	TO	27 = first available code after 0 through 26
O	B	15	01111	28:	OB	
B	E	2	00010	29:	BE	
E	O	5	00101	30:	EO	
O	R	15	01111	31:	OR	
R	N	18	10010	32:	RN	32 requires 6 bits, so for next output use 6 bits
N	O	14	001110	33:	NO	
O	T	15	001111	34:	OT	
T	T	20	010100	35:	TT	
TO	B	27	011011	36:	TOB	
BE	O	29	011101	37:	BEO	
OR	T	31	011111	38:	ORT	
TOB	E	36	100100	39:	TOBE	
EO	R	30	011110	40:	EOR	
RN	O	32	100000	41:	RNO	
OT	#	34	100010			# stops the algorithm; send the cur seq
		0	000000			and the stop code

Unencoded length = 25 symbols × 5 bits/symbol = 125 bits
Encoded length = (6 codes × 5 bits/code) + (11 codes × 6 bits/code) = 96 bits.
Using LZW has saved 29 bits out of 125, reducing the message by more than 23%. If the message were longer, then the dictionary words would begin to represent longer and longer sections of text, sending repeated words very compactly.

Advantages of LZW

- LZW requires no prior information about the input data stream.
- LZW can compress the input stream in one single pass.
- Another advantage of LZW its simplicity, allowing fast execution.

REFERENCES

- "Elementary Analysis: The Theory of Calculus" by Kenneth A. Ross
- "Principles of Mathematical Analysis" by Walter Rudin
- "Elementary Real Analysis" by Brian S. Thomson, Judith B. Bruckner, Andrew M. Bruckne
- https://en.wikipedia.org/wiki/Bolzano%E2%80%93Weierstrass_theorem
- https://math.usu.edu/rheal/math4200/class_material/Sequences/Bolzano_Weierstrass.pdf
- https://www.cs.dartmouth.edu/~doug/diff.pdf
- https://en.wikipedia.org/wiki/Subsequential_limit
- https://medium.com/cantors-paradise/monotonicity-will-prevail-finite-sequences-the-pigeonhole-principle-and-the-erd%C5%91s-szekeres-f5980f88d17
- https://en.m.wikipedia.org/wiki/Erd%C5%91s%E2%80%93Szekeres_theorem
- https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/
- https://www.geeksforgeeks.org/longest-common-subsequence-dp-using-memoization/
- https://www.commonlounge.com/discussion/d731468749074 70ba34c54a22214d067
- https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/
- https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
- https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/

CONCLUSION

The project talks about subsequence and its applications in the real world. Thus, highlighting that subsequence and its related concepts are a key factor in a vast variety of fields, and there remains scope for further research in the future to improve the existing concepts and algorithms in order to improve computational efficiency and provide better methods to solve real world problems.

THANK YOU