ΜΙΑ ΝΕΑ ΠΡΟΣΕΓΓΙΣΗ ΣΤΗΝ ΕΥΡΕΣΗ ΤΗΣ ΜΕΓΑΛΥΤΕΡΗΣ ΚΟΙΝΗΣ ΑΚΟΛΟΥΘΙΑΣ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΜΙΑ ΠΡΩΤΑ ΣΕ ΒΑΘΟΣ ΑΝΑΖΗΤΗΣΗ

ΦΡΑΓΚΙΑΔΑΚΗ ΕΛΕΝΗ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων Καθηγητής: Σαμαράς Νικόλαος
Εξεταστές:Χατζηγεωργίου Αλέξανδρος

Τμήμα Εφαρμοσμένης Πληροφορικής

Πανεπιστήμιο Μακεδονίας
Θεσσαλονίκη

Σεπτέμβριος 2006

ΠΕΡΙΛΗΨΗ

Το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών παρουσιάζει μεγάλο ενδιαφέρον, λόγω της ευρείας χρήσης του σε πολλούς επιστημονικούς τομείς, ένας από τους οποίους είναι και ο κλάδος της πληροφορικής. Το παρόν paper περιλαμβάνει μια παρουσίαση των συναφών προβλημάτων εύρεσης κάποιας κοινής ακολουθίας ανάμεσα σε δύο αλφαριθμητικά που βασίζονται σε κάποιο ή κάποια κριτήρια. Επίσης περιγράφεται και αναλύεται ο κλασικός αλγόριθμος που χρησιμοποιείται για την εύρεση της μεγαλύτερης κοινής ακολουθίας καθώς και μια νέα προσέγγιση στο πρόβλημα αυτό που στηρίζει την επίλυση του προβλήματος σε μια σε βάθος πρώτα αναζήτηση. Γίνεται αναφορά σε επιστημονικά περιοδικά και δημοσιεύσεις που έχουν γίνει σχετικές με το θέμα, ενώ περιλαμβάνονται και πηγές που παρουσιάζουν ενδιαφέρον.

ABSTRACT

The problem of finding the longest common subsequence of two strings is one of the problems that the scientific community is interested in due to its implementation in a variety of sciences, one of which is the science of informatics. In the present paper, the algorithm that solves the problem is examined as part of the coherent algorithms of finding some common subsequence between two strings based on one or more criteria. Additionally the classic algorithm for finding the longest common subsequence is presented, as well as a new approach for finding the longest common subsequence which is based on a depth first search. Sscientific journals, publications and other interesting sources are being referenced by the paper.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΟΙΚΟΓΕΝΕΙΑΣ ΑΛΓΟΡΙΘΜΩΝ ΕΥΡΕΣΗΣ ΚΑΠΟΙΑΣ ΚΟΙΝΗΣ ΑΚΟΛΟΥΘΙΑΣ ΔΥΟ ΑΛΦΑΡΙΘΜΗΤΙΚΩΝ

1.1 Εισαγωγή

Το πρόβλημα της εύρεσης ομοιοτήτων ανάμεσα σε δύο διαφορετικά κομμάτια κειμένου έχει απασχολήσει ευρέως την επιστημονική κοινότητα τις τελευταίες δεκαετίες. Το πρόβλημα αυτό βρίσκει εφαρμογή σε πολλούς διαφορετικούς τομείς της επιστήμης, όπως στην βιοϊατρική και στην πληροφορική.

Τα τελευταία χρόνια έχουν διεξαχθεί και συνεχίζουν να διεξάγονται αρκετές έρευνες σε ότι αφορά τις ακολουθίες νουκλεοτιδίων που αποτελούν το DNA και RNA τόσο του ανθρώπινου οργανισμού όσο και άλλων οργανισμών. Οι μελέτες αυτές έχουν ως σκοπό την χαρτογράφηση του DNA και του RNA των οργανισμών, προσπαθώντας να εντοπίσουν ομοιότητες και διαφορές τόσο ανάμεσα σε διαφορετικά δείγματα του ίδιου οργανισμού, όσο και σε δείγματα από διαφορετικούς οργανισμούς. Με αυτόν τον τρόπο θα διευρύνουμε τις γνώσεις μας σε ότι αφορά τις διαφορές ανάμεσα στα είδη οργανισμών, στα χρωμοσώματα, στις γενετικές μεταλλάξεις και στις γενετικές ασθένειες.

Σε αυτές τις επιστήμες και τις έρευνες υπάρχει λοιπόν ανάγκη για τον εντοπισμό των κοινών ακολουθιών ανάμεσα σε δύο διαφορετικές ακολουθίες DNA ή πρωτεϊνών. Οι αλγόριθμοι αυτής της οικογένειας αποσκοπούν στην επίλυση τέτοιου είδους προβλημάτων και χρησιμοποιούνται στις έρευνες των επιστημόνων αφού τους παρέχουν τα εργαλεία, έτσι ώστε με την βοήθεια των υπολογιστών να διευκολύνουν και να επισπεύδουν την παραγωγή αποτελεσμάτων.

Ένας άλλος τομέας στον οποίο βρίσκουν εφαρμογή οι αλγόριθμοι αυτής της οικογένειας είναι στον εντοπισμό ομοιοτήτων ανάμεσα σε δύο διαφορετικά κομμάτια κώδικα ή κειμένου γενικότερα. Οι αλγόριθμοι αυτοί επιτρέπουν την σύγκριση δύο κομματιών κειμένου και τον εντοπισμό ομοιοτήτων και διαφορών. Με αυτόν τον τρόπο μπορεί κανείς να γνωρίζει, με κάποια επιπλέον εργασία πάνω στα αποτελέσματα, αν τα δύο κομμάτια κειμένου προέρχονται από το ίδιο πηγαίο κομμάτι ή ακόμα και τις αλλαγές που έχει υποστεί το πηγαίο κομμάτι κειμένου μέχρι να φτάσει στην μορφή που συγκρίνουμε.

Τέτοιοι αλγόριθμοι χρησιμοποιούνται ακόμα και στο Διαδίκτιο καθώς επίσης και στην εύρεση κοινών στοιχείων ανάμεσα σε δύο διαφορετικά κομμάτια μουσικού κειμένου, για την διαπίστωση της ύπαρξης κάποιας μελωδίας σε δύο διαφορετικά μουσικά κομμάτια.

1.1.1 Γενική Ορολογία

Όταν αναφερόμαστε σε ένα αλφαριθμητικό, αναφερόμαστε σε μια ακολουθία από χαρακτήρες ή αριθμούς. Ένα από τα σημαντικότερα χαρακτηριστικά ενός

αλφαριθμητικού είναι το μήκος του, δηλαδή ο αριθμός των χαρακτήρων που το αποτελούν.

Με τον όρο κοινή ακολουθία(common subsequence) 2 αλφαριθμητικών αναφερόμαστε στους κοινούς χαρακτήρες που εμφανίζονται στα 2 αλφαριθμητικά που εξετάζουμε σε μια αυξανόμενη σειρά αλλά όχι απαραίτητα σε γειτονικές θέσεις.

1.2 Οι αλγόριθμοι εύρεσης κάποιας κοινής ακολουθίας

Η εύρεση της κοινής αυτής ακολουθίας ανάμεσα σε 2 αλφαριθμητικά μπορεί να υπόκειται σε κάποιους κανόνες που πρέπει να ακολουθούν τα στοιχεία που θα συμπεριλάβουμε στην ακολουθία αυτή. Ανάλογα λοιπόν με τα κριτήρια που πρέπει να πληρούν τα στοιχεία της κοινής αυτής ακολουθίας έχουν αναπτυχθεί και χρησιμοποιούνται διαφορετικοί αλγόριθμοι που όμως ανήκουν στην ίδια οικογένεια, αυτή της εύρεσης κάποιας κοινής ακολουθίας ανάμεσα σε 2 αλφαριθμητικά.

4 είναι τα βασικά προβλήματα που καλούνται να επιλύσουν οι αλγόριθμοι αυτής της οικογένειας.

- Το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας – Longest Common Subsequence (LCS) [1]
    - Το πρόβλημα της εύρεσης όλων των μεγαλύτερων κοινών ακολουθιών – All substrings Longest Common Subsequence (ALCS) [2] [3] [4]
    - Το πρόβλημα της εύρεσης της περιορισμένης μεγαλύτερης κοινής ακολουθίας – Constrained Longest Common Subsequence (CLCS) [5] [6]
    - Το Time – Warped LCS [7]
- Το πρόβλημα της εύρεσης της μεγαλύτερης αυξανόμενης ακολουθίας – Longest Increasing Subsequence (LIS) [8] [9] [10]
- Το πρόβλημα της εύρεσης της βαρύτερης κοινής ακολουθίας – Heaviest Common Subsequence (HCS) [8]
- Το πρόβλημα της εύρεσης της βαρύτερης αυξανόμενης ακολουθίας – Heaviest Increasing Subsequence (HIS)

Κάθε ένα από αυτά τα προβλήματα αντιμετωπίζει την εύρεση κάποιας κοινής ακολουθίας δύο αλφαριθμητικών με διαφορετικό τρόπο γιατί σε κάθε ένα διαφοροποιείται το κριτήριο που θέτουμε για τα στοιχεία που θα αποτελούν την ακολουθία αυτή.

1.2.1 Το πρόβλημα της εύρεσης του Longest Common Subsequence

Στην οικογένεια της εύρεσης κάποιας κοινής ακολουθίας ανάμεσα σε 2 αλφαριθμητικά εντάσσεται και το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας(Longest Common Subsequence – LCS) 2 αλφαριθμητικών. Ανάμεσα σε 2 αλφαριθμητικά μπορούν να υπάρχουν πολλές κοινές ακολουθίες, με διαφορετικό μήκος η καθεμία. Οι αλγόριθμοι που επιλύουν το πρόβλημα αυτό, εξετάζουν τις ακολουθίες αυτές και επιστρέφουν εκείνη που έχει το μεγαλύτερο μήκος, δηλαδή

εκείνη που αποτελείται από τους περισσότερους δυνατούς χαρακτήρες που υπάρχουν και στα δύο αλφαριθμητικά σε αυξανόμενη σειρά.

Για παράδειγμα ας υποθέσουμε ότι το πρώτο αλφαριθμητικό, στο οποίο θα αναφερόμαστε ως X, αποτελείται από τους χαρακτήρες AACTGGCTAT, και το δεύτερο αλφαριθμητικό, στο οποίο θα αναφερόμαστε ως Y, αποτελείται από τους χαρακτήρες GACGTA. Το μήκος του X είναι 10, αφού αποτελείται από 10 χαρακτήρες και το μήκος του Y είναι 6. Για να μπορούμε να αναφερόμαστε στους χαρακτήρες του κάθε αλφαριθμητικού, τα παρουσιάζουμε παρακάτω με μορφή πίνακα με αριθμημένες θέσεις.

Πίνακας 1.1 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
X	A	A	C	T	G	G	C	T	A	T

Πίνακας 1.2 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**	**4**	**5**
Y	G	A	C	G	T	A

Μια κοινή ακολουθία των δύο αυτών αλφαριθμητικών είναι η GAT.

Για να την δημιουργήσουμε χρησιμοποιήσαμε τα ακόλουθα γράμματα:

- το G που βρίσκεται στην θέση 4 του X και στην θέση 0 του Y
- το A που βρίσκεται στην θέση 8 του X και στην θέση 1 του Y
- το T που βρίσκεται στην θέση 9 του X και στην θέση 4 του Y

Μια άλλη κοινή ακολουθία είναι η GCTA.

Για να την δημιουργήσουμε χρησιμοποιήσαμε τα ακόλουθα γράμματα:

- το G που βρίσκεται στην θέση 4 του X και στην θέση 0 του Y
- το C που βρίσκεται στην θέση 6 του X και στην θέση 2 του Y
- το T που βρίσκεται στην θέση 7 του X και στην θέση 4 του Y
- το A που βρίσκεται στην θέση 8 του X και στην θέση 5 του Y

Το μήκος της πρώτης κοινής ακολουθίας είναι 3 και το μήκος της δεύτερης 4. Το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας όμως θέλουμε να μας επιστρέψει την μεγαλύτερη από τις κοινές ακολουθίες των 2 αλφαριθμητικών που είναι η ACGTA που αποτελείται από 5 χαρακτήρες.

Και για να την δημιουργήσουμε χρησιμοποιήσαμε τα ακόλουθα γράμματα:

- το A που βρίσκεται στην θέση 0 του X και στην θέση 1 του Y
- το C που βρίσκεται στην θέση 2 του X και στην θέση 2 του Y
- το G που βρίσκεται στην θέση 4 του X και στην θέση 3 του Y
- το T που βρίσκεται στην θέση 7 του X και στην θέση 4 του Y

- το A που βρίσκεται στην θέση 8 του X και στην θέση 5 του Y

1.2.1.1 Το πρόβλημα της εύρεσης των All Substring Longest Common Subsequence

Με βάση αυτά στα οποία αναφερθήκαμε στην προηγούμενη παράγραφο, το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών μας επιστρέφει την μεγαλύτερη κοινή ακολουθία των δύο αλφαριθμητικών.

Όμως είναι πιθανό, δύο αλφαριθμητικά να έχουν περισσότερες από μία ακολουθίες κοινές και όλες να έχουν το ίδιο μήκος, ίσο με αυτό της μεγαλύτερης κοινής ακολουθίας. Το προηγούμενο πρόβλημα αρκείται στο να μας επιστρέψει μία από αυτές τις μεγαλύτερες κοινές ακολουθίες.

Η συγκεκριμένη κατηγορία προβλημάτων, που εξετάζεται σε αυτήν την παράγραφο, μας επιστρέφει όχι μόνο μια από τις μεγαλύτερες κοινές ακολουθίες, αλλά όλες, εάν και εφόσον βέβαια υπάρχουν πάνω από μία.

Για παράδειγμα ας υποθέσουμε ότι το αλφαριθμητικό X έχει μήκος 10 και αποτελείται από τα στοιχεία AACTGGCTAT και το αλφαριθμητικό Y έχει μήκος 7 και αποτελείται από τα στοιχεία AGCTTGA. Τα αλφαριθμητικά αυτά παρουσιάζονται με μορφή πινάκων με αριθμημένες τις θέσεις των στοιχείων τους παρακάτω.

Πίνακας 1.3 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
X	A	A	C	T	G	G	C	T	A	T

Πίνακας 1.4 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**	**4**	**5**	**6**
Y	A	G	C	T	T	G	A

Ο αλγόριθμος της εύρεσης της μεγαλύτερης κοινής ακολουθίας θα μας επέστρεφε, ότι το μέγεθος της μεγαλύτερης κοινής ακολουθίας των X και Y είναι 5 και μια από τις πιθανές ακολουθίες που θα εντοπίζαμε είναι η AGCTT, για την δημιουργία της οποίας χρησιμοποιήσαμε:

- το A που βρίσκεται στην θέση 0 του X και στη θέση 0 του Y
- το G που βρίσκεται στην θέση 4 του X και στη θέση 1 του Y
- το C που βρίσκεται στην θέση 6 του X και στη θέση 2 του Y
- το T που βρίσκεται στην θέση 7 του X και στην θέση 3 του Y
- το T που βρίσκεται στην θέση 9 του X και στην θέση 4 του Y

Όμως αυτή δεν είναι η μόνη από τις κοινές ακολουθίες των X, Y που έχει μήκος 5 (ίσο με το μεγαλύτερο δυνατό πλήθος κοινών στοιχείων των δύο αλφαριθμητικών).

Υπάρχουν άλλες 3 κοινές ακολουθίες που έχουν ακριβώς το ίδιο μήκος και αυτές είναι :

- η ακολουθία AGCTA για την οποία χρησιμοποιήσαμε :
    - το A που βρίσκεται στην θέση 0 του X και στην θέση 0 του Y
    - το G που βρίσκεται στην θέση 4 του X και στην θέση 1 του Y
    - το C που βρίσκεται στην θέση 6 του X και στην θέση 2 του Y
    - το T που βρίσκεται στην θέση 7 του X και στην θέση 3 του Y
    - το A που βρίσκεται στην θέση 8 του X και στην θέση 6 του Y
- η ακολουθία ACTGA
    - το A που βρίσκεται στην θέση 0 του X και στην θέση 0 του Y
    - το C που βρίσκεται στην θέση 2 του X και στην θέση 2 του Y
    - το T που βρίσκεται στην θέση 3 του X και στην θέση 3 του Y
    - το G που βρίσκεται στην θέση 4 του X και στην θέση 5 του Y
    - το A που βρίσκεται στην θέση 8 του X και στην θέση 6 του Y
- η ακολουθία ACTTA
    - το A που βρίσκεται στην θέση 0 του X και στην θέση 0 του Y
    - το C που βρίσκεται στην θέση 2 του X και στην θέση 2 του Y
    - το T που βρίσκεται στην θέση 3 του X και στην θέση 3 του Y
    - το T που βρίσκεται στην θέση 7 του X και στην θέση 4 του Y
    - το A που βρίσκεται στην θέση 8 του X και στην θέση 6 του Y

Όλες οι ακολουθίες που παρουσιάστηκαν σε αυτήν την ενότητα, είναι πιθανά αποτελέσματα στο πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας, αλλά και οι λύσεις στο πρόβλημα της εύρεσης όλων των μεγαλύτερων κοινών ακολουθιών των X και Y.

Θα μπορούσαμε να πούμε ότι μια ειδική κατηγορία αυτού του τύπου προβλημάτων είναι η εύρεση μόνο του αριθμού των μεγαλύτερων κοινών ακολουθιών. Αυτοί οι αλγόριθμοι ασχολούνται μόνο με την εύρεση του πλήθους των ακολουθιών αυτών και όχι με την εύρεση των πραγματικών στοιχείων που τις αποτελούν [11]. Για παράδειγμα, στο πρόβλημα που εξετάσαμε προηγούμενα σε αυτήν την παράγραφο, ένας αλγόριθμος που υπάγεται σε αυτήν την κατηγορία θα μας επέστρεφε μονάχα των αριθμό 4, εφόσον καταλήξαμε ότι το μήκος της μεγαλύτερης κοινής ακολουθίας των X και Y είναι 5 και υπάρχουν 4 κοινές ακολουθίες των δύο αλφαριθμητικών με τόσα ακριβώς στοιχεία.

1.2.1.2 Το πρόβλημα της εύρεσης του Constrained Longest Common Subsequence

Το πρόβλημα της εύρεσης της περιορισμένης μεγαλύτερης κοινής ακολουθίας, συνεπάγεται την εύρεση της μεγαλύτερης κοινής ακολουθίας ανάμεσα σε δύο

αλφαριθμητικά με την προϋπόθεση ότι αυτή η κοινή ακολουθία περιλαμβάνει μια άλλη ακολουθία, γνωστή εκ των προτέρων.

Για παράδειγμα, ας υποθέσουμε ότι θα χρησιμοποιήσουμε τα X, Y που παρουσιάστηκαν στην προηγούμενη ενότητα και επιθυμούμε να βρούμε την μεγαλύτερη κοινή ακολουθία τους η οποία θέλουμε να έχει ως subsequence την ακολουθία K που είναι η GTA.

Πίνακας 1.5 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
X	A	A	C	T	G	G	C	T	A	T

Πίνακας 1.6 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**	**4**	**5**	**6**
Y	A	G	C	T	T	G	A

Πίνακας 1.7 Η ακολουθία K

	0	**1**	**2**
K	G	T	A

Χρησιμοποιώντας την γνώση από το παράδειγμα της προηγούμενης ενότητας, στον παρακάτω πίνακα παρουσιάζουμε τις μεγαλύτερες κοινές ακολουθίες των X και Y.

Πίνακας 1.7 Οι μεγαλύτερες κοινές ακολουθίες των X, Y

	0	**1**	**2**	**3**	**4**
1	A	G	C	T	T
2	A	G	C	T	A
3	A	C	T	G	A
4	A	C	T	T	A

Παρατηρώντας αυτές τις ακολουθίες μπορούμε εύκολα να εντοπίσουμε ότι μόνο η ακολουθία που βρίσκεται στην δεύτερη σειρά του πίνακα (η AGCTA) περιλαμβάνει την ακολουθία GTA που θέλουμε. Οπότε η λύση στο συγκεκριμένο πρόβλημα είναι μία, γιατί μόνο μια από τις μεγαλύτερες κοινές ακολουθίες έχει ως υποακολουθία την ακολουθία GTA.

Αυτό που θα θέλαμε να σημειώσουμε εδώ είναι η διαφορετική αντιμετώπιση των επιστημόνων στο συγκεκριμένο πρόβλημα. Κάποιοι υποστηρίζουν ότι το πρόβλημα του Constrained Longest Common Subsequence είναι μια παραλλαγή του κλασικού προβλήματος του LCS. Θεωρούν ότι εφόσον το συγκεκριμένο πρόβλημα απλά προσθέτει μια καινούργια παράμετρο στο πρόβλημα του LCS δεν διαφοροποιείται από το αρχικό πρόβλημα ιδιαίτερα, αφού το μόνο που θέλουμε είναι απλά στην

μεγαλύτερη κοινή ακολουθία που θα μας επιστρέψει να περιλαμβάνεται μια άλλη ακολουθία (02).

Υπάρχει όμως και μια αντίθετη άποψη που θεωρεί ότι το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών είναι ουσιαστικά μια υποκατηγορία του γενικότερου προβλήματος του Constrained Longest Common Subsequence. Πραγματικά, αν θεωρήσουμε ότι η ακολουθία που περιορίζει τα αποτελέσματα του αλγορίθμου, δεν περιλαμβάνει κανένα απολύτως στοιχείο, άρα πρόκειται για το κενό αλφαριθμητικό, τότε έχουμε υποπέσει στην επίλυση της εύρεσης του προβλήματος LCS που ακολουθώντας αυτήν την φιλοσοφία όντως μπορεί να θεωρεί ειδική περίπτωση του CLCS(04).

1.2.1.3 To Time – Warped Longest Common Subsequence

Ο συγκεκριμένος αλγόριθμος έχει πιο ειδική χρήση από ότι οι υπόλοιποι αλγόριθμοι και προβλήματα που παρουσιάζουμε. Εξειδικεύεται σε μουσικά κομμάτια. Η έρευνα για τον συγκεκριμένο αλγόριθμο ξεκίνησε λόγω της μεγάλης διάδοσης που έχουν τα μουσικά κομμάτια στον παγκόσμιο ιστό. Έχουν δημιουργηθεί βάσης δεδομένων που φιλοξενούν ολόκληρα μουσικά κομμάτια ή μέρος της μελωδίας τους, από τις οποίες ο χρήστης του διαδικτύου μπορεί να αναζητήσει ένα κομμάτι που τον ενδιαφέρει.

Η αναζήτηση αυτή γίνεται είτε με κάποια σύμβολα που εισάγει ο χρήστης από το πληκτρολόγιό του είτε ακόμα και μελωδία που τραγουδά ο ίδιος ο χρήστης από κάποιο μικρόφωνο. Στην δεύτερη περίπτωση είναι κατανοητό ότι ο χρήστης είναι πιθανό να μην μπορέσει να αποδώσει με ακριβή τρόπο την μελωδία που αναζητά, είτε λόγο λάθους στον ρυθμό είτε λόγο λάθους σε κάποιες νότες. Τα λάθη αυτά οφείλονται κυρίως στο γεγονός ότι όταν κάποιος επιχειρήσει να αναπαραγάγει μια μελωδία συνήθως μερικές από τις νότες τις προφέρει είτε σε μεγαλύτερο είτε σε μικρότερο χρονικό διάστημα από αυτό που υποστηρίζει η συγκεκριμένη μελωδία. Αυτά τα προβλήματα απασχολούν κυρίως εκείνους που ασχολούνται με συστήματα ανάκτησης μουσικών κομματιών.

Ο συγκεκριμένος αλγόριθμος χρησιμοποιεί τεχνικές που του επιτρέπουν να χειρίζεται αλλαγές στην ταχύτητα της μελωδίας και ανακολουθίες στο ρυθμό, ενσωματώνοντας τον αλγόριθμο για την εύρεση της μεγαλύτερης κοινής ακολουθίας (που επιτρέπει την ύπαρξη απόστασης ανάμεσα σε δύο από τα στοιχεία της ακολουθίας αυτής στο αρχικό αλφαριθμητικό) και του δυναμικού Time wraping αλγορίθμου (DTW).

1.2.2 Το πρόβλημα της εύρεσης του Longest Increasing Subsequence

Στην ίδια οικογένεια αλγορίθμων ανήκουν και οι αλγόριθμοι που επιλύουν το πρόβλημα της εύρεσης της μεγαλύτερης αυξανόμενης ακολουθίας δύο αλφαριθμητικών. Για παράδειγμα αν τα δύο αλφαριθμητικά τα οποία εξετάζουμε αποτελούνται από τα ψηφία 0 έως 9, τότε η εύρεση της μεγαλύτερης αυξανόμενης ακολουθίας προϋποθέτει την εύρεση της κοινής ακολουθίας της οποίας τα στοιχεία διατάσσονται κατά αύξουσα σειρά.

Ας υποθέσουμε ότι το πρώτο αλφαριθμητικό ονομάζεται X και αποτελείται είναι τα εξής ψηφία, 203405487 και το δεύτερο αλφαριθμητικό είναι το Y, που έχει την εξής

μορφή, 320385457. Τα X, Y έχουν μήκος 9 και παρουσιάζονται στους πίνακες που ακολουθούν.

Πίνακας 1.7 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
X	2	0	3	4	0	5	4	8	7

Πίνακας 1.8 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Y	3	2	0	3	8	5	4	5	7

Η μεγαλύτερη κοινή ακολουθία που εντοπίζουμε στα X, Y είναι η 203547 η οποία έχει μήκος 6. Για να την δημιουργήσουμε χρησιμοποιήσαμε τα ακόλουθα στοιχεία:

- το 2 που βρίσκεται στη θέση 0 του X και στη θέση 1 του Y
- το 0 που βρίσκεται στη θέση 1 του X και στη θέση 2 του Y
- το 3 που βρίσκεται στη θέση 2 του X και στη θέση 3 του Y
- το 5 που βρίσκεται στη θέση 5 του X και στη θέση 5 του Y
- το 4 που βρίσκεται στη θέση 6 του X και στη θέση 6 του Y
- το 7 που βρίσκεται στη θέση 8 του X και στη θέση 8 του Y

Αν και αυτή είναι η μεγαλύτερη κοινή ακολουθία των X, Y δεν είναι λύση στο πρόβλημα της εύρεσης της μεγαλύτερης αυξανόμενης ακολουθίας διότι οι αριθμοί που την αποτελούν δεν είναι διατεταγμένοι σε αύξουσα σειρά. Η λύση στο συγκεκριμένο πρόβλημα είναι η ακολουθία 23457, η οποία έχει μικρότερο μήκος από την ακολουθία που εξετάσαμε προηγουμένως αλλά οι αριθμοί που την αποτελούν βρίσκονται σε αύξουσα σειρά.

Για να την δημιουργήσουμε χρησιμοποιήσαμε τα ακόλουθα στοιχεία

- το 2 που βρίσκεται στη θέση 0 του X και στη θέση 1 του Y
- το 3 που βρίσκεται στη θέση 2 του X και στη θέση 3 του Y
- το 4 που βρίσκεται στη θέση 3 του X και στη θέση 6 του Y
- το 5 που βρίσκεται στη θέση 5 του X και στη θέση 5 του Y
- το 7 που βρίσκεται στη θέση 8 του X και στη θέση 8 του Y

1.2.3 Το πρόβλημα της εύρεσης του Heaviest Common Subsequence

Ένα άλλο πρόβλημα που επιλύουν οι αλγόριθμοι αυτής της οικογένειας είναι η εύρεση της βαρύτερης κοινής ακολουθίας δύο αλφαριθμητικών. Ας υποθέσουμε ότι έχουμε δύο αλφαριθμητικά το $X = x_1 \ldots x_n$ που έχει μήκος n (αποτελείται από n χαρακτήρες) και το $Y = y_1 \ldots y_m$ που έχει μήκος m. Αν μια κοινή ακολουθία τους είναι η $s = s_1 \ldots s_t$ (μήκους t) τότε αυτή η ακολουθία έχει ένα βάρος το οποίο προκύπτει από τους χαρακτήρες οι οποίοι την αποτελούν και από τις θέσεις στις οποίες αυτοί οι

χαρακτήρες εντοπίζονται στα δύο αλφαριθμητικά. Το βάρος αυτό υπολογίζεται από μια συνάρτηση που είναι γνωστή εκ των προτέρων με βάση τον τύπο που ακολουθεί.

$$W(s) = \sum_{p=1}^{t} f(i_p, j_p, s_p)$$

Στον παραπάνω τύπο το W(s) είναι το βάρος της κοινής ακολουθίας s των δύο αλφαριθμητικών, που προκύπτει από το άθροισμα των βαρών των χαρακτήρων που αποτελούν την ακολουθία αυτή. Το βάρος του κάθε χαρακτήρα προκύπτει από την συνάρτηση f και εξαρτάται τόσο από τον χαρακτήρα όσο και από την θέση στην οποία αυτός εντοπίζεται σε κάθε ένα αλφαριθμητικό.

1.2.4 Το πρόβλημα της εύρεσης του Heaviest Increasing Subsequence

Το συγκεκριμένο πρόβλημα, θα μπορούσαμε να πούμε ότι είναι συνδυασμός δύο προβλημάτων στα οποία αναφερθήκαμε προηγούμενα, του προβλήματος της εύρεσης της Longest Increasing Subsequence και του προβλήματος της Heaviest Common Subsequence. Ουσιαστικά η λύση αυτού του προβλήματος προϋποθέτει την εύρεση της μεγαλύτερης κοινής ακολουθίας των δύο αλφαριθμητικών η οποία πρέπει να αποτελείται από αυξανόμενους με βάση κάποιο κριτήριο χαρακτήρες, αλλά θα πρέπει ταυτόχρονα αυτή η ακολουθία να έχει και το μεγαλύτερο βάρος.

ΚΕΦΑΛΑΙΟ 2

ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ CLASSIC LONGEST COMMON SUBSEQUENCE – CLCS

2.1 Εισαγωγή

Όπως έχουμε ήδη αναφέρει το πιο κλασικό πρόβλημα που συναντάμε όταν εξετάζουμε την εύρεση κάποιας κοινής ακολουθίας δύο αλφαριθμητικών είναι η εύρεση της μεγαλύτερης κοινής ακολουθίας [12].

Για την εύρεση της μεγαλύτερης κοινής ακολουθίας έχουν δημιουργηθεί και χρησιμοποιούνται πολλοί αλγόριθμοι, ενώ από τους πρώτους που χρησιμοποιήθηκαν σε μεγάλο βαθμό είναι ο αλγόριθμος των Hunt και Szymanski. Σε αυτόν τον αλγόριθμο έχουν στηριχθεί και πολλοί από τους αλγόριθμους που αναπτύχθηκαν στην συνέχεια για την επίλυση του συγκεκριμένου προβλήματος [1] και είναι ο αλγόριθμος που θα εξετάσουμε διεξοδικά στο κεφάλαιο αυτό.

Οι αλγόριθμοι που έχουν αναπτυχθεί για την επίλυση του συγκεκριμένου προβλήματος θα μπορούσαν να χωριστούν σε κάποιες κατηγορίες ανάλογα με τον τρόπο που επιλύουν το πρόβλημα. Εκτός από αυτούς που ακολουθούν τον κλασικό αλγόριθμο για την εύρεση του LCS και επιλύουν το πρόβλημα σειριακά, υπάρχουν και άλλοι που ακολουθούν διαφορετικούς τρόπους επίλυσης. Στους τρόπους αυτούς εντάσσονται και αλγόριθμοι που επιλύουν το πρόβλημα χρησιμοποιώντας τον παράλληλο προγραμματισμό [13] καθώς επίσης και τεχνικές δανεισμένες από την επιστήμη της τεχνητής νοημοσύνης και των νευρωνικών δικτύων [14].

Στην δεύτερη περίπτωση αναφερόμαστε σε αλγορίθμους οι οποίοι χρησιμοποιούν κάποιο νευρωνικό δίκτυο, το οποίο εκπαιδεύεται στην εύρεση της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών και καταλήγει σε αποτέλεσμα με μικρό ποσοστό λάθους.

2.2 Λειτουργία του αλγορίθμου

Ο αλγόριθμος των Hunt και Szymanski, στηρίζει την λειτουργία του στην λειτουργία μιας συγκεκριμένης δομής που επιτρέπει τον υπολογισμό του μήκους της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών και όλων των prefixes τους.

Με τον όρο prefix αναφερόμαστε σε ένα αλφαριθμητικό, για παράδειγμα Κ, που προκύπτει από κάποιο άλλο, έστω το Χ, αν χρησιμοποιήσουμε τα n πρώτα στοιχεία του Χ. Για παράδειγμα έστω ότι έχουμε το αλφαριθμητικό Χ που αποτελείται από τα στοιχεία AC4G3T45TA35C4G. Το αλφαριθμητικό αυτό αποτελείται από 15 στοιχεία και άρα έχει μήκος 15. Αν επιθυμούσαμε να παράγουμε όλα τα prefixes του συγκεκριμένου αλφαριθμητικού θα αποκτούσαμε ένα σύνολο από 15 αλφαριθμητικά, κάθε ένα από τα οποία ξεκινάει με το πρώτο στοιχείο του αλφαριθμητικού Χ και συνεχίζει με 1 μέχρι 14 από τα επόμενα στοιχεία του χωρίς να παραλείπεται κάποιο. Ειδική περίπτωση prefix είναι το κενό αλφαριθμητικό που θεωρείται prefix ενός οποιουδήποτε αλφαριθμητικού.

Πιο αναλυτικά στον παρακάτω πίνακα παρατίθεται το αλφαριθμητικό X με τα στοιχεία που το αποτελούν, αριθμώντας το πρώτο στοιχείο από την θέση 0 και φτάνοντας μέχρι την θέση 14.

Πίνακας 2.1 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
X	A	C	4	G	3	T	4	5	T	A	3	5	C	4	G

Στον πίνακα που ακολουθεί παρουσιάζονται στον οριζόντιο άξονα τα στοιχεία που αποτελούν το κάθε ένα prefix του αλφαριθμητικού X και στον κατακόρυφο απλά μια αρίθμηση του πλήθους των prefixes. Παρατηρούμε ότι το πρώτο prefix είναι το κενό αλφαριθμητικό, όπως έχουμε ήδη αναφέρει και στην τελευταία θέση υπάρχει το ίδιο το αλφαριθμητικό X, το οποίο και μπορεί να θεωρηθεί ως prefix του εαυτού του.

Πίνακας 2.2 Τα prefixes του X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1															
2	A														
3	A	C													
4	A	C	4												
5	A	C	4	G											
6	A	C	4	G	3										
7	A	C	4	G	3	T									
8	A	C	4	G	3	T	4								
9	A	C	4	G	3	T	4	5							
10	A	C	4	G	3	T	4	5	T						
11	A	C	4	G	3	T	4	5	T	A					
12	A	C	4	G	3	T	4	5	T	A	3				
13	A	C	4	G	3	T	4	5	T	A	3	5			
14	A	C	4	G	3	T	4	5	T	A	3	5	C		
15	A	C	4	G	3	T	4	5	T	A	3	5	C	4	
16	A	C	4	G	3	T	4	5	T	A	3	5	C	4	G

Ο αλγόριθμος λοιπόν υπολογίζει την μεγαλύτερη κοινή ακολουθία ανάμεσα σε δύο αλφαριθμητικά, υπολογίζοντας στην πορεία της εκτέλεσής του και την μεγαλύτερη κοινή ακολουθία κάθε ζεύγους prefixes των δύο αλφαριθμητικών, στα οποία στο εξής θα αναφερόμαστε ως X και Y. Επίσης θεωρούμε ότι το μήκος του X είναι n και το μήκος του Y είναι m (τα X, Y αποτελούνται από n και m στοιχεία αντίστοιχα).

Κατά την εκτέλεση του αλγορίθμου δημιουργείται ένας πίνακας με τόσες γραμμές όσα είναι τα στοιχεία από τα οποία αποτελείται το X και τόσες στήλες όσα είναι τα

στοιχεία του Y. Ο πίνακας λοιπόν αυτός, που είναι και η βασική δομή του συγκεκριμένου αλγορίθμου, είναι διαστάσεων nxm. Για να δημιουργηθεί αυτός ο πίνακας σαρώνεται το X στοιχείο – στοιχείο και για κάθε ένα από αυτά τα στοιχεία εξετάζεται αν στην αντίστοιχη θέση του Y μπορούμε να εντοπίσουμε το στοιχείο αυτό. Ανάλογα με το αποτέλεσμα της σύγκρισης αυτής καταχωρούνται στις κατάλληλες θέσεις του πίνακα οι αντίστοιχες τιμές.

Σκοπός της καταχώρησης αυτών των τιμών είναι να ορίζουν το μέγεθος της μεγαλύτερης κοινής ακολουθίας που έχει βρεθεί μέχρι το σημείο που εξετάζουμε, δηλαδή μέχρι την συγκεκριμένη θέση του X και την αντίστοιχή της στο Y. Με την λήξη της δημιουργίας του πίνακα και της καταχώρησης των τιμών γνωρίζουμε το μήκος της μεγαλύτερης κοινής ακολουθίας των δύο αλφαριθμητικών και είναι η τιμή που εντοπίζεται στην θέση [n, m] του πίνακα.

Βέβαια, παρόλο που γνωρίζουμε το μήκος της μεγαλύτερης κοινής ακολουθίας των δύο αλφαριθμητικών, δεν γνωρίζουμε την ίδια την ακολουθία. Δηλαδή δεν γνωρίζουμε ακριβώς ποια είναι τα στοιχεία εκείνα που εντοπίσαμε και στα δύο αλφαριθμητικά και αποτελούν τα στοιχεία της μεγαλύτερης κοινής ακολουθίας. Για τον υπολογισμό και της μεγαλύτερης κοινής ακολουθίας με τα στοιχεία που την αποτελούν απαιτούνται κάποιες επιπλέον πράξεις.

2.3 Ψευδοκώδικας

Παρακάτω παρατίθεται ο ψευδοκώδικας του αλγορίθμου ο οποίος έχει ως είσοδο τα δύο αλφαριθμητικά X, Y και παράγει ως έξοδο το μήκος της μεγαλύτερης κοινής ακολουθίας των δύο αυτών αλφαριθμητικών. Θεωρούνται επίσης γνωστά και τα μήκη των X, Y που είναι n και m αντίστοιχα. Για να είναι πιο κατανοητές οι πράξεις στις οποίες προβαίνει ο αλγόριθμος, θεωρούμε ότι ο πίνακας ο οποίος κατασκευάζεται έχει μια επιπλέον γραμμή που αριθμείται με τον αριθμό -1 καθώς και μια επιπλέον στήλη η οποία αριθμείται και εκείνη με τον αριθμό -1.

Τα πρώτα βήματα του αλγορίθμου αποτελούν οι δύο αρχικοποιήσεις της γραμμής -1 και της στήλης -1 στις οποίες θέτουμε όλα τα στοιχεία ίσα με το μηδέν. Ο πίνακας ο οποίος κατασκευάζεται από τον αλγόριθμο είναι ο πίνακας L. Ακολουθεί ένας βρόγχος επανάληψης ο οποίος προσπελαύνει εξωτερικά ένα προς ένα όλα τα στοιχεία του αλφαριθμητικού X και εσωτερικά ένα προς ένα όλα τα στοιχεία του αλφαριθμητικού Y.

Σε κάθε μια από τις εσωτερικές επαναλήψεις γίνεται ένας έλεγχος αν το τρέχων στοιχείο του αλφαριθμητικού X, που είναι το στοιχείο i, είναι το ίδιο με το τρέχων στοιχείο του αλφαριθμητικού Y, που είναι το j. Από αυτόν τον έλεγχο μπορούμε να έχουμε 2 πιθανά αποτελέσματα:

- Τα δύο στοιχεία να είναι ίδια
- Τα δύο στοιχεία να μην είναι ίδια

Στην πρώτη περίπτωση, που τα δύο στοιχεία είναι ίδια, στην τρέχουσα θέση του πίνακα L, καταχωρείται η τιμή που βρίσκεται στην θέση [i-1, j-1] του πίνακα αυξημένη κατά μια μονάδα. Το στοιχείο στο οποίο αναφερθήκαμε είναι ουσιαστικά το στοιχείο που βρίσκεται ακριβώς διαγώνια και προς τα πίσω από την τρέχουσα θέση του πίνακα. Στην δεύτερη περίπτωση, που τα δύο στοιχεία δεν είναι ίδια, στην τρέχουσα θέση του πίνακα καταχωρείται η μεγαλύτερη από τις τιμές που βρίσκονται

στις θέσεις [i-1, j] και [i, j-1] του πίνακα. Με άλλα λόγια, αν τα στοιχεία δεν είναι ίδια στην τρέχουσα θέση του πίνακα καταχωρείται η μεγαλύτερη από τις τιμές που βρίσκονται στην θέση πάνω από την τρέχουσα και στην θέση αριστερά από την τρέχουσα.

Τελειώνοντας αυτός ο επαναληπτικός βρόχος, έχει κατασκευαστεί ο πίνακας L και το στοιχείο του L[n-1, m-1] έχει καταχωρημένο τον αριθμό που μας δείχνει το μήκος της μεγαλύτερης κοινής ακολουθίας των X, Y. Ο αλγόριθμος επιστρέφει τον πίνακα L.

Επίσης έχοντας τον πίνακα L, μπορούμε να εξετάσουμε ποιο είναι το μέγεθος της μεγαλύτερης κοινής ακολουθίας κάποιων prefixes των δύο αρχικών αλφαριθμητικών μας. Για παράδειγμα σε κάποια θέση του πίνακα L, [i, j] είναι καταχωρημένος ο αριθμός που μας δίνει το μήκος της μεγαλύτερης κοινής ακολουθίας του prefix του X [0..i] και του prefix του Y [0…j].

Είσοδος: τα αλφαριθμητικά X, Y

Έξοδος: το μήκος L[n-1, m-1] της μεγαλύτερης κοινής ακολουθίας των X[0..n-1] και Y[0..m-1]

```
for i ←1 to n-1 do
        L[i,-1] ←0
for j ←1 to m-1 do
        L[-1,j] ←0
for i  ← 0 to n-1 do
        for j ← 0 to m-1 do
                if X[i] = Y[j] then
                        L[i,j] ← L[i-1,j-1] + 1
                else
                        L[i,j] ← max{L[i-1,j], L[i,j-1]}
return array L
```

2.4 Παράδειγμα χρήσης του αλγορίθμου

Ας υποθέσουμε ότι έχουμε δύο αλφαριθμητικά το X και το Y και επιθυμούμε να βρούμε την μεγαλύτερη κοινή ακολουθία των δύο αυτών αλφαριθμητικών. Το X έχει μήκος 10 και αποτελείται από τα στοιχεία AACTGGCTAT. Το Y έχει μήκος 6 και αποτελείται από τα στοιχεία GACGTA. Παρακάτω παρουσιάζονται με την μορφή πίνακα τα δύο αλφαριθμητικά, όπου στην πρώτη γραμμή αναφέρονται οι θέσεις των στοιχείων του αλφαριθμητικού και στην δεύτερη το όνομα του και τα στοιχεία που το αποτελούν.

Πίνακας 2.3 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
X	A	A	C	T	G	G	C	T	A	T

Πίνακας 2.4 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**	**4**	**5**
Y	G	A	C	G	T	A

Όπως έχουμε ήδη αναφέρει ο πίνακας L θα έχει τόσες στήλες όσο είναι το μήκος του αλφαριθμητικού Y και τόσες γραμμές όσο είναι το μήκος του αλφαριθμητικού X. Σε αυτές προστίθεται άλλη μια γραμμή και μια στήλη για να διευκολυνθούμε στις πράξεις. Στην πρώτη σειρά του πίνακα εμφανίζονται οι αριθμοί των στηλών, ξεκινώντας την αρίθμηση από το -1, και στην πρώτη στήλη εμφανίζονται οι αριθμοί των γραμμών, πάλι ξεκινώντας την αρίθμηση από το -1.

Τα πρώτα βήματα του αλγορίθμου είναι να αρχικοποιήσει τα στοιχεία του πίνακα L που βρίσκονται στην γραμμή -1 και στην στήλη -1, δίνοντας σε όλα την τιμή 0. Μετά την εκτέλεση τους ο πίνακας L έχει την παρακάτω μορφή.

Πίνακας 2.3 Ο πίνακας L μετά την αρχικοποίηση της γραμμής -1 και της στήλης -1

	-1	**0**	**1**	**2**	**3**	**4**	**5**
-1	0	0	0	0	0	0	0
0	0						
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						
8	0						
9	0						

Στην συνέχεια ακολουθεί η εκτέλεση του επαναληπτικού βρόχου για τον οποίο έχουμε:

- Η πρώτη επανάληψη ξεκινάει με το i να έχει πάρει την τιμή 0
    - Στην συνέχεια το j παίρνει την τιμή 0
        - Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το A, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το G.
        - Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
        - Δηλαδή L[0, 0] = max {L[-1,0], L[0, -1]} = max{0, 0} = 0
    - Στην συνέχεια το j παίρνει την τιμή 1

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το A, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το A.
- Τα δύο στοιχεία είναι ίσα, οπότε στην θέση [i, j] του πίνακα L τοποθετούμε την τιμή που βρίσκεται στην θέση [i-1, j-1] του πίνακα L, αυξημένη κατά μια μονάδα.
- Δηλαδή L[0,1] = L[-1, 0] + 1 = 0 + 1 = 1

o Στην συνέχεια το j παίρνει την τιμή 2

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το A, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το C.
- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[0, 2] = max {L[-1, 2], L[0, 1]} = max{0, 1} = 1

o Στην συνέχεια το j παίρνει την τιμή 3

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το A, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το G.
- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[0, 3] = max {L[-1, 3], L[0, 2]} = max{0, 1} = 1

o Στην συνέχεια το j παίρνει την τιμή 4

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το A, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το T.
- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[0, 4] = max {L[-1, 4], L[0, 3]} = max{0, 1} = 1

o Στην συνέχεια το j παίρνει την τιμή 5

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το A, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το A.
- Τα δύο στοιχεία είναι ίσα, οπότε στην θέση [i, j] του πίνακα L τοποθετούμε την τιμή που βρίσκεται στην θέση [i-1, j-1] του πίνακα L, αυξημένη κατά μια μονάδα.
- Δηλαδή L[0,5] = L[-1, 4] + 1 = 0 + 1 = 1

Σε αυτό το σημείο της εκτέλεσης του αλγορίθμου έχουμε ολοκληρώσει με την προσθήκη των τιμών στα στοιχεία της γραμμής 0 του πίνακα L, και ο πίνακας έχει την μορφή που φαίνεται παρακάτω.

Πίνακας 2.4 Ο πίνακας L μετά την ολοκλήρωση της εσωτερικής επανάληψης του βρόχου. (συμπλήρωση τιμών των στοιχείων της γραμμής 0)

	-1	0	1	2	3	4	5
-1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						
8	0						
9	0						

Με τον ίδιο τρόπο γίνονται και οι υπόλοιπες επαναλήψεις, οπότε φτάνοντας στο σημείο όπου το i παίρνει την τιμή 9, ο πίνακας L έχει την μορφή που φαίνεται παρακάτω.

Πίνακας 2.5 Ο πίνακας L μετά την ολοκλήρωση της 8^{ης} επανάληψης του εξωτερικού βρόχου. (συμπλήρωση τιμών των γραμμών 0 έως και 8)

	-1	0	1	2	3	4	5
-1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
1	0	0	1	1	1	1	2
2	0	0	1	2	2	2	2
3	0	0	1	2	2	3	3
4	0	1	1	2	3	3	3
5	0	1	1	2	3	3	3
6	0	1	1	2	3	3	3
7	0	1	1	2	3	4	4
8	0	1	2	2	3	4	5
9	0						

Κατά την τελευταία επανάληψη του εξωτερικού βρόχου έχουμε :

- Το i να έχει πάρει την τιμή 9
    - Στην συνέχεια το j παίρνει την τιμή 0
        - Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το T, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το G.

- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[9, 0] = max {L[8,0], L[9, -1]} = max{1, 0} = 1

o Στην συνέχεια το j παίρνει την τιμή 1

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το T, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το A.
- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[9, 1] = max {L[8,1], L[9, 0]} = max{2, 1} = 2

o Στην συνέχεια το j παίρνει την τιμή 2

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το T, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το C.
- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[9, 2] = max {L[8, 2], L[9, 1]} = max{2, 2} = 2

o Στην συνέχεια το j παίρνει την τιμή 3

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το T, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το G.
- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[9, 3] = max {L[8, 3], L[9, 2]} = max{3, 2} = 3

o Στην συνέχεια το j παίρνει την τιμή 4

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το T, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το T.
- Τα δύο στοιχεία είναι ίσα, οπότε στην θέση [i, j] του πίνακα L τοποθετούμε την τιμή που βρίσκεται στην θέση [i-1, j-1] του πίνακα L, αυξημένη κατά μια μονάδα.
- Δηλαδή L[9,4] = L[8, 3] + 1 = 3 + 1 = 4

o Στην συνέχεια το j παίρνει την τιμή 5

- Γίνεται ο έλεγχος αν το στοιχείο του X, X[i] που είναι το T, είναι ίδιο με το στοιχείο του Y, Y[j] που είναι το A.
- Τα δύο αυτά στοιχεία δεν είναι ίδια οπότε στην θέση [i, j] του πίνακα L τοποθετούμε το μεγαλύτερο από τα στοιχεία που βρίσκονται στις θέσεις [i-1,j] και [i, j-1].
- Δηλαδή L[9, 5] = max {L[8, 5], L[9, 4]} = max{5, 4} = 5

Στο σημείο αυτό έχει ολοκληρωθεί η δημιουργία του πίνακα L καθώς επίσης και η εκτέλεση του αλγορίθμου. Ο πίνακας L, που είναι η δομή που χρησιμοποιεί ο συγκεκριμένος αλγόριθμος, έχει την τελική μορφή που φαίνεται παρακάτω.

Πίνακας 2.6 Ο πίνακας L μετά την ολοκλήρωση του αλγορίθμου

	-1	**0**	**1**	**2**	**3**	**4**	**5**
-1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
1	0	0	1	1	1	1	2
2	0	0	1	2	2	2	2
3	0	0	1	2	2	3	3
4	0	1	1	2	3	3	3
5	0	1	1	2	3	3	3
6	0	1	1	2	3	3	3
7	0	1	1	2	3	4	4
8	0	1	2	2	3	4	5
9	0	1	2	2	3	4	5

Έχοντας ολοκληρώσει την εκτέλεση του αλγορίθμου και την δημιουργία της δομής, δηλαδή του πίνακα L, έχουμε και τα αποτελέσματα για το μήκος της μεγαλύτερης κοινής ακολουθίας των δύο αλφαριθμητικών. Το μήκος αυτό αντιπροσωπεύει ο αριθμός που βρίσκεται στην θέση [9,5] του πίνακα. Ουσιαστικά ο αλγόριθμος μας επιστρέφει ότι το μήκος της μεγαλύτερης κοινής ακολουθίας των X, Y που είναι 5.

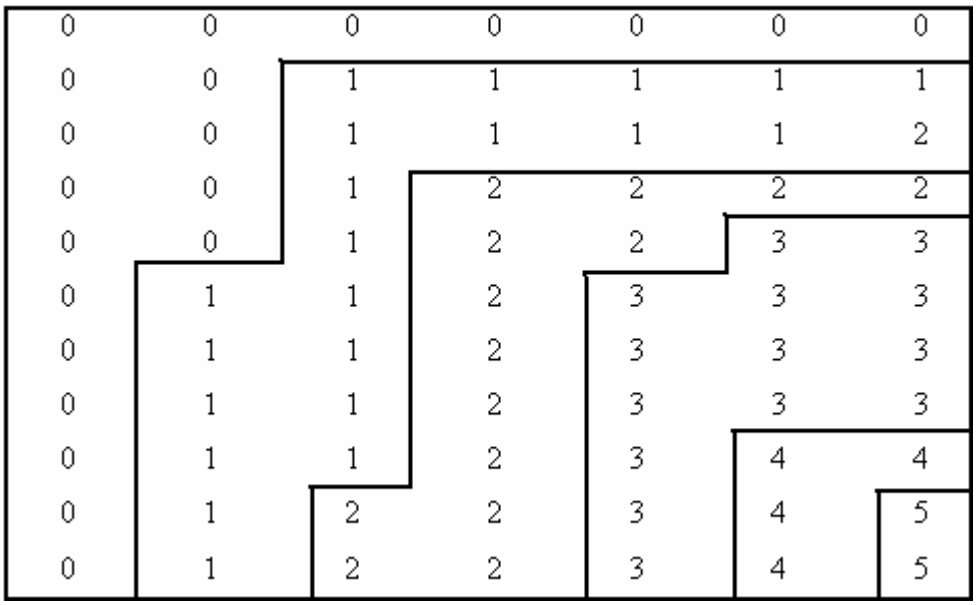
2.5 Αλγόριθμοι που στηρίζονται στον Classic LCS

Αρχικά ο αλγόριθμος των Hunt και Szymanski περιελάμβανε έναν απλό τρόπο για να μας επιστρέφει την μεγαλύτερη κοινή ακολουθία των δύο αλφαριθμητικών. Ένας τρόπος υπολογισμού της μεγαλύτερης κοινής ακολουθίας που βασίζεται στον αλγόριθμο των Hunt και Szymanski και χρησιμοποιείται μέχρι και σήμερα παρουσιάζεται λίγα χρόνια αργότερα από τον Hirschberg και είναι ο αλγόριθμος ALGD [15].

Ο αλγόριθμος αυτός χρησιμοποιεί τον πίνακα που κατασκευάζει ο Classic LCS αλγόριθμος και βρίσκει τα στοιχεία που αποτελούν την μεγαλύτερη κοινή ακολουθία των αλφαριθμητικών που δόθηκαν ως είσοδος στον αλγόριθμο. Προκειμένου να κάνει τους υπολογισμούς του, χωρίζει τον πίνακα L στα λεγόμενα contours. Αυτά τα contours είναι ουσιαστικά κομμάτια του πίνακα που περιλαμβάνουν τους κοινούς αριθμούς. Από αυτά στην συνέχεια χρησιμοποιεί τα στοιχεία εκείνα του κάθε contour που ονομάζονται candidates. Ως candidates θεωρούνται τα στοιχεία εκείνα που οριοθετούν τις γωνίες κάθε contour και αντιπροσωπεύουν ουσιαστικά υποψήφια στοιχεία για να συμπεριληφθούν στην μεγαλύτερη κοινή ακολουθία.

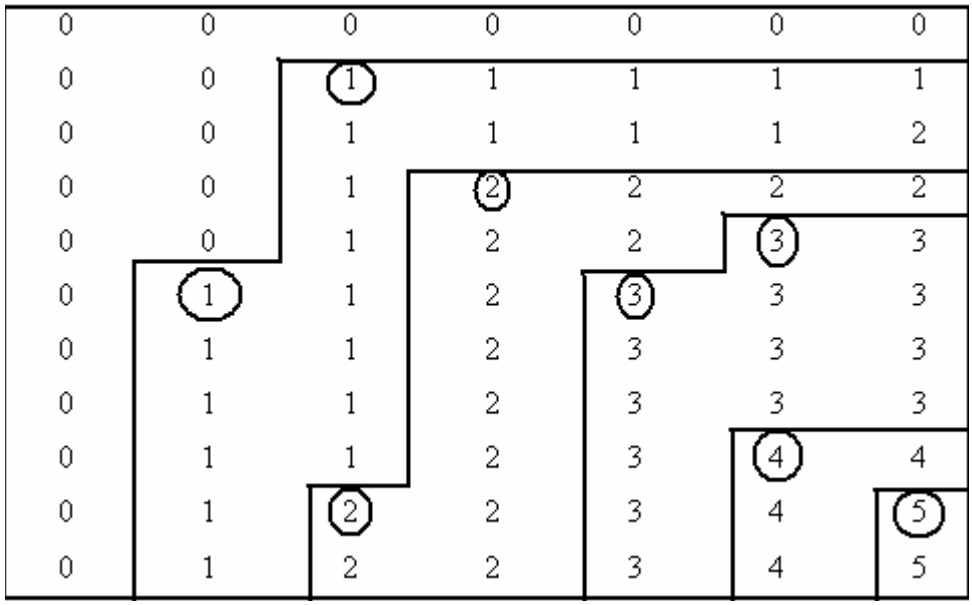
Για παράδειγμα στον πίνακα L που κατασκευάσαμε στην προηγούμενη ενότητα, θα είχαμε τα contours που φαίνονται στο παρακάτω σχήμα.

Σχήμα 2.1 Ο πίνακας L και τα contours που τον αποτελούν



Με αυτόν τον τρόπο λοιπόν χωρίζεται ο πίνακας L που κατασκευάσαμε στην προηγούμενη ενότητα σε contours κάθε ένα από τα οποία περιλαμβάνει θα μπορούσαμε να πούμε μόνο εκείνα τα στοιχεία του πίνακα που έχουν την ίδια τιμή. Για να επιλέξουμε τα candidate στοιχεία από κάθε contour αρκεί να επιλέξουμε τα στοιχεία εκείνα που οριοθετούν τις γωνίες του κάθε contour. Μετά την επιλογή των candidates ο πίνακας L θα μπορούσαμε να πούμε ότι έχει την μορφή που φαίνεται στο ακόλουθο σχήμα, όπου τα στοιχεία που είναι κυκλωμένα είναι τα candidates.

Σχήμα 2.2 Ο πίνακας L, τα contours που τον αποτελούν και τα candidate στοιχεία



Απλοποιώντας την λειτουργία του αλγορίθμου για να κατανοήσουμε το πώς τελικά επιστρέφει τη μεγαλύτερη κοινή ακολουθία θα μπορούσαμε να πούμε τα εξής. Για να

εντοπίσουμε τα 5 στοιχεία, που στην συγκεκριμένη περίπτωση αποτελούν την μεγαλύτερη κοινή ακολουθία θα μπορούσαμε να ξεκινήσουμε από το τελευταίο υποψήφιο στοιχείο που βρίσκεται στην θέση [8, 5] του πίνακα με σκοπό να βρούμε εκείνο το μονοπάτι από το τελευταίο αυτό στοιχείο στην αρχή του πίνακα έχοντας έτσι υπολογίσει στην πορεία τα στοιχεία που αποτελούν την μεγαλύτερη κοινή ακολουθία των X και Y. Από κάθε ένα contour θα χρησιμοποιήσουμε μόνο ένα candidate στοιχείο αφού στο κάθε contour ένα είναι το κοινό στοιχείο ανάμεσα στα δύο αλφαριθμητικά.

Στη συνέχεια μετακινούμαστε στο contour που βρίσκεται στα αριστερά και εκεί εντοπίζουμε μόνο ένα candidate στοιχείο, αυτό που βρίσκεται στην θέση [7, 4] του πίνακα και επομένως είναι και αυτό που χρησιμοποιήσουμε από αυτό το contour στην σύνθεση της μεγαλύτερης κοινής ακολουθίας.

Στο contour που βρίσκεται αριστερά , αποτελούμενο από τα στοιχεία που είναι ίσα με 3, παρατηρούμε ότι υπάρχουν δύο candidate στοιχεία από τα οποία θα πρέπει να επιλέξουμε ποιο θα είναι τελικά αυτό που θα χρησιμοποιηθεί. Στην διαδικασία της επιλογής πρέπει να τηρούνται φυσικά οι κανόνες για την μεγαλύτερη κοινή ακολουθία. Για παράδειγμα δεν μπορούμε να χρησιμοποιήσουμε το candidate στοιχείο της θέσης [3, 4] του πίνακα διότι θα έπρεπε στην μεγαλύτερη κοινή ακολουθία να συμπεριλάβουμε το στοιχείο που βρίσκεται στην τέταρτη θέση του Y δύο φορές κάτι που φυσικά παραβαίνει τον ορισμό και την σημασία της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών. Επομένως από το συγκεκριμένο contour θα επιλεγεί το στοιχείο της θέσης [4, 3].

Στο επόμενο προς τα αριστερά contour και πάλι εμφανίζονται δύο candidate στοιχεία, ένα στην θέση [2, 2] και ένα στην θέση [8, 1]. Και εδώ θα πρέπει να επιλεγεί αυτό που συμφωνεί με τους κανόνες. Ας ανακεφαλαιώσουμε από ποια στοιχεία έχουμε καταλήξει ότι θα αποτελείται η μεγαλύτερη κοινή ακολουθία ξεκινώντας από το τέλος της και οδεύοντας προς την αρχή.

- Το στοιχείο που βρίσκεται στην θέση 8 του X και 5 του Y
- Το στοιχείο που βρίσκεται στην θέση 4 του X και 3 του Y

Μετά από την ανακεφαλαίωση αυτή είναι φανερό ότι δεν μπορεί να χρησιμοποιηθεί το candidate στοιχείο της θέσης [8, 1] από αυτό το contour που εξετάζουμε αυτήν την στιγμή. Είμαστε υποχρεωμένοι να χρησιμοποιήσουμε κάποιο στοιχείο του οποίου η θέση στο X θα είναι μικρότερη από τον αριθμό 4 (θέση του τελευταίου στοιχείου που χρησιμοποιήσαμε μέχρι στιγμής στην μεγαλύτερη κοινή ακολουθία) και στο Y σε θέση μικρότερη του 3 αντίστοιχα. Το candidate στοιχείο που συμφωνεί με αυτό το κριτήριο είναι αυτό που βρίσκεται στην θέση [2, 2].

Συνεχίζοντας και στο επόμενο contour και χρησιμοποιώντας την ίδια λογική το candidate στοιχείο που θα χρησιμοποιηθεί είναι αυτό στην θέση [0, 1] του πίνακα. Σε αυτό το σημείο έχουμε ολοκληρώσει την διαδικασία εύρεσης της μεγαλύτερης κοινής ακολουθίας των δύο αλφαριθμητικών και αυτή αποτελείται από τα στοιχεία

- A που βρίσκεται στην θέση 0 του X και 1 του Y
- C που βρίσκεται στην θέση 2 του X και 2 του Y
- G που βρίσκεται στην θέση 4 του X και 3 του Y
- T που βρίσκεται στην θέση 7 του X και 4 του Y

- Α που βρίσκεται στην θέση 8 του X και 5 του Y

Επομένως χρησιμοποιώντας τον πίνακα L καταλήξαμε ότι η μεγαλύτερη κοινή ακολουθία των X και Y έχει μήκος 5 και η ακολουθία αυτή είναι η ACGTA.

Από εκείνο το σημείο και έπειτα έχουν υλοποιηθεί αρκετοί αλγόριθμοι που εντοπίζουν την μεγαλύτερη κοινή ακολουθία δύο αλφαριθμητικών χρησιμοποιώντας την δομή (πίνακα) που κατασκευάζει ο αλγόριθμος των Hunt και Szymanski, αλλά και γενικότερα την φιλοσοφία που παρουσιάστηκε προηγούμενα με τον αλγόριθμο του ALGD του Hirschberg. Οι αλγόριθμοι αυτοί διαφέρουν στον τρόπο που χρησιμοποιούν τον πίνακα για να εντοπίσουν τα contours, τα candidate στοιχεία του κάθε contour αλλά και στον τρόπο που αποφασίζουν τελικά ποιο από τα candidate στοιχεία του κάθε contour θα χρησιμοποιηθεί τελικά στην μεγαλύτερη κοινή ακολουθία που θα επιστρέψει ο κάθε αλγόριθμος [16] [17]. Γενικά μπορούμε να πούμε ότι έχουν αναπτυχθεί τρεις μεγάλες κατηγορίες στις οποίες εντάσσονται οι αλγόριθμοι της εύρεσης της μεγαλύτερης κοινής ακολουθίας δύο αριθμών, ανάλογα με τον τρόπο που χειρίζονται τον πίνακα. Οι τρεις αυτές κατηγορίες έχουν να κάνουν με το πώς από την στιγμή που έχει δημιουργηθεί ο πίνακας χειρίζονται τα δεδομένα του, για την εύρεση των contours και των candidate στοιχείων που αναφέραμε προηγούμενα. Οι κατηγορίες αυτές ονομαστικά είναι οι εξής

- Η πρώτη κατηγορία επεξεργάζεται τα δεδομένα του πίνακα σειρά προς σειρά
- Η δεύτερη κατηγορία επεξεργάζεται τα δεδομένα του πίνακα προχωρώντας από contour σε contour
- Η τρίτη κατηγορία επεξεργάζεται τα επεξεργάζεται διαγωνίως

Ανάλογα λοιπόν με τον τρόπο προσέγγισης υπάρχουν και χρησιμοποιούνται διαφορετικές βοηθητικές σε κάθε περίπτωση.

Όλοι οι αλγόριθμοι στους οποίους αναφερθήκαμε προηγούμενα σε αυτήν την παράγραφο λειτουργούν με βάση τις αρχές του κλασικού προγραμματισμού, δηλαδή σε έναν υπολογιστή σειριακά. Εκτός από αυτούς τους αλγορίθμους έχουν αναπτυχθεί και χρησιμοποιούνται και άλλοι αλγόριθμοι οι οποίοι εκμεταλλεύονται γνώσεις που έχουμε και από άλλους τομείς της πληροφορικής.

Έχουν παρουσιαστεί και αλγόριθμοι οι οποίοι στηρίζουν την εκτέλεσή τους σε κάποιο νευρωνικό δίκτυο [14]. Τα νευρωνικά δίκτυα και η γενικότερη φιλοσοφία τους είναι ιδιαίτερα γνωστή σε εκείνους που ασχολούνται με την επιστήμη της τεχνητής νοημοσύνης, των νευρωνικών δικτύων και της θεωρίας παιγνίων.

Η επίλυση του προβλήματος της εύρεσης της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών με την βοήθεια νευρωνικού δικτύου συνίσταται στην εύρεση και στην δημιουργία ενός κατάλληλου για το πρόβλημα νευρωνικού δικτύου. Στην συνέχεια ακολουθεί η λεγόμενη εκπαίδευση του νευρωνικού δικτύου. Στα πλαίσια της εκπαίδευσης αυτής εισάγονται στο νευρωνικό δίκτυο ως δεδομένα κάποια αλφαριθμητικά των οποίων η μεγαλύτερη κοινή ακολουθία είναι γνωστή. Με αυτόν τον τρόπο εκπαιδεύεται το δίκτυο πάνω σε συγκεκριμένα παραδείγματα με το επιθυμητό αποτέλεσμα, δηλαδή την μεγαλύτερη κοινή ακολουθία να είναι εκ των προτέρων γνωστό. Στην συνέχεια αυτό το δίκτυο δοκιμάζεται σε άλλα παραδείγματα και εξάγει αποτελέσματα για τα παραδείγματα αυτά.

Το μειονέκτημα θα μπορούσαμε να πούμε τις μεθόδου αυτής είναι ότι εξάγει τα αποτελέσματα με κάποιο ποσοστό ακρίβειας που καθορίζεται εκ των προτέρων. Αυτό

αυτόματα σημαίνει ότι δεν μπορούμε να είμαστε απολύτως σίγουροι για την ορθότητα του αποτελέσματος ακόμα και αν το αποδεκτό σφάλμα είναι της τάξης του 5%.

Μια άλλη προσέγγιση στο πρόβλημα της μεγαλύτερης κοινής ακολουθίας έχει γίνει και με την χρήση παράλληλου προγραμματισμού, όπου ένα πλέγμα από υπολογιστές χρησιμοποιείται για επίλυση του προβλήματος. Με την χρήση του παράλληλου προγραμματισμού, μεταφερόμαστε σε άλλη φιλοσοφία προγραμματισμού, αφού στην συγκεκριμένη περίπτωση η επίλυση του προβλήματος γίνεται από πολλούς υπολογιστές ταυτόχρονα που για παράδειγμα συντονίζονται από κάποιον κεντρικό υπολογιστή, ως προς τις ενέργειες που είναι να επιτελέσει ο κάθε ένας. Σε κάθε περίπτωση όμως οι υπολογιστές που συμμετέχουν με αυτόν τον τρόπο στην επίλυση του προβλήματος μοιράζονται μια κοινή μνήμη από την οποία παίρνουν τα δεδομένα για τους υπολογισμούς τους και στην οποία επιστρέφουν τα αποτελέσματα στα οποία έχει καταλήξει ο κάθε ένας από αυτούς.

Σε αυτούς τους αλγορίθμους αυτό που διαφοροποιείται και διαφοροποιεί και τις επιδόσεις τους είναι το μοντέλο που χρησιμοποιείται, διότι ανάλογα με το μοντέλο που χρησιμοποιείται υπάρχει και διαφορετική φιλοσοφία προγραμματισμού, διαφορετικός τρόπος επικοινωνίας ανάμεσα στους υπολογιστές που συμμετέχουν στην επίλυση και διαφορετική διαχείριση και χρήση της μνήμης. Για παράδειγμα κάποιοι από αυτούς χρησιμοποιούν το μοντέλο CREW – PRAM [13], άλλοι το μοντέλο LARPBS [18] καθώς και το cluster Beowulf [19].

2.6 Πλεονεκτήματα – Μειονεκτήματα του Classic LCS

Συνοψίζοντας τα συμπεράσματα στα οποία καταλήξαμε για τον αλγόριθμο των Hunt και Szymanski παρουσιάζονται σε αυτήν την ενότητα αυτά που θεωρούμε ότι αποτελούν τα βασικά πλεονεκτήματα και μειονεκτήματα του αλγορίθμου.

Το βασικότερο πλεονέκτημα του αλγορίθμου είναι η απλότητα με την οποία αντιμετωπίζει το πρόβλημα και ο πολύ γρήγορος, από άποψη χρονικής πολυπλοκότητας, τρόπος με τον οποίο καταλήγει στην απάντηση του προβλήματος της εύρεσης της μεγαλύτερης κοινής ακολουθίας. Με τον πίνακα τον οποίο κατασκευάζει μπορεί σε nxm χρόνο να μας παρέχει την απάντηση για το μήκος της μεγαλύτερης κοινής ακολουθίας.

Επιπρόσθετα μας παρέχει, όπως έχουμε ήδη αναφέρει την πληροφορία για όλα τα prefixes των δύο αλφαριθμητικών που δίνουμε ως είσοδο. Δηλαδή, μας επιστρέφει και το μήκος της μεγαλύτερης κοινής ακολουθίας οποιονδήποτε prefixes των αρχικών αλφαριθμητικών.

Το βασικό πλεονέκτημα του συγκεκριμένου αλγορίθμου είναι ο τρόπος με τον οποίο χειρίζεται τα δεδομένα που χρειάζεται για να καταλήξει στην ζητούμενη απάντηση. Ο πίνακας που χρησιμοποιείται και που ουσιαστικά είναι και αυτός που θα μας επιστρέψει το μήκος της μεγαλύτερης κοινής ακολουθίας των δύο αλφαριθμητικών, έχει, όπως έχουμε ήδη παρουσιάσει σε προηγούμενη ενότητα, n γραμμές όπου n είναι το μήκος του πρώτου αλφαριθμητικού και m στήλες, όπου m είναι το μήκος του δεύτερου αλφαριθμητικού. Αυτό αυτόματα σημαίνει ότι ο πίνακας αυτός μέχρι το τέλος της εκτέλεσης του αλγορίθμου θα φιλοξενεί nxm στοιχεία καταλαμβάνοντας τον αντίστοιχο χώρο στην μνήμη του υπολογιστή.

Όταν αναφερόμαστε βέβαια σε δύο αλφαριθμητικά των οποίων το μήκος δεν είναι ιδιαίτερα μεγάλο, αυτό δεν αποτελεί πρόβλημα, αφού μπορούμε να διατηρούμε ολόκληρο το μέγεθος του πίνακα στην μνήμη RAM του υπολογιστή. Η μνήμη αυτή είναι ιδιαίτερα γρήγορη και είναι πολύ εύκολο να ανακτήσουμε δεδομένα από αυτήν καθώς και να αποθηκεύσουμε δεδομένα σε αυτήν σε πολύ μικρούς χρόνους. Όμως, όταν αναφερόμαστε σε μεγάλα μεγέθη για τα δύο αλφαριθμητικά, μεγέθη που μπορεί να φτάνουν τα αρκετά εκατομμύρια στοιχεία, τότε ο πίνακας που κατασκευάζεται από τον αλγόριθμο αυτό έχει τόσα στοιχεία που αφενός αυξάνεται κατά πολύ το μέγεθος της μνήμης που καταναλώνεται από τον αλγόριθμο και αφετέρου το πλήθος των στοιχείων αυτών δεν μας επιτρέπει να διατηρούμε ολόκληρο τον πίνακα στην μνήμη RAM του υπολογιστή.

Σε αυτές τις περιπτώσεις πραγματοποιείται το λεγόμενο swapping του δίσκου, όπου κάποιο κομμάτι του σκληρού δίσκου του υπολογιστή χρησιμοποιείται σαν μνήμη. Αυτό που πρέπει να επισημάνουμε σε αυτό το σημείο είναι ότι ο σκληρός δίσκος είναι πολύ αργός σε σχέση με την μνήμη RAM του υπολογιστή τόσο κατά την εγγραφή στοιχείων σε αυτόν όσο και κατά την ανάγνωσή τους. Αυτό έχει ως αποτέλεσμα να αυξάνεται πάρα πολύ ο χρόνος τον οποίο χρειάζεται για να ολοκληρώσει τους υπολογισμούς του αυτός ο αλγόριθμος, αφού σε πολλές περιπτώσεις υποχρεώνει την μεταφορά τμημάτων από τον σκληρό δίσκο στην μνήμη RAM και αντίστροφα.

ΚΕΦΑΛΑΙΟ 3

ΟΙ ΙΔΙΑΙΤΕΡΟΤΗΤΕΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΕΥΡΕΣΗΣ ΤΟΥ LCS

3.1. Εισαγωγή

Το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών είναι όπως είδαμε και στις προηγούμενες παραγράφους ένα πρόβλημα που έχει αντιμετωπιστεί με ποικίλους τρόπους. Το πρόβλημα έχει κάποιες ιδιαιτερότητες και αυτές είναι που κάνουν την επίλυση του προβλήματος μια διαδικασία που εξαρτάται και επηρεάζεται από πολλούς παράγοντες. Οι ιδιαιτερότητες αυτές οφείλονται κυρίως σε δύο παράγοντες. Την ίδια την φύση του προβλήματος αλλά και το αλφάβητο από το οποίο είναι κατασκευασμένα τα αλφαριθμητικά.

3.2. Η ιδιαιτερότητα του προβλήματος

Σε σχέση με άλλα προβλήματα που έχουν να κάνουν με τον εντοπισμό κάποιου αλφαριθμητικού μέσα σε κάποιο άλλο παρουσιάζει μια μεγάλη διαφορά. Στα προβλήματα εύρεσης ενός αλφαριθμητικού μέσα σε κάποιο άλλο, το αλφαριθμητικό που αναζητούμε το αναζητούμε ακριβώς στην μορφή που είναι χωρίς να μπορούμε να καταλήξουμε σε κάποιο αποτέλεσμα παρεμβάλλοντας κάποιους χαρακτήρες ή μη χρησιμοποιώντας κάποιους από τους χαρακτήρες που αναζητούμε.

Για παράδειγμα αν έχουμε το αλφαριθμητικό Χ, που είναι η λέξη «ΑΛΓΟΡΙΘΜΟΣ» και ζητούμε να εντοπίσουμε μέσα σε αυτήν το αλφαριθμητικό Υ που είναι η λέξη «ΡΥΘΜΟΣ», ουσιαστικά δεν θα απαντήσουμε θετικά στο ερώτημα του αν υπάρχει το Υ μέσα στο Χ. Αν όμως επιζητούσαμε την μεγαλύτερη κοινή ακολουθία αυτών των αλφαριθμητικών τότε θα καταλήγαμε στο συμπέρασμα ότι αυτή είναι η «ΡΘΜΟΣ» έχοντας ουσιαστικά διαγράψει από το Υ το δεύτερο γράμμα του. Καταλαβαίνουμε λοιπόν ότι το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας παρουσιάζει δυσκολίες στην αντιμετώπισή του.

3.2.1. Η φύση του προβλήματος

Όπως έχουμε ήδη περιγράψει σε προηγούμενη ενότητα η εύρεση της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών συνεπάγεται την εύρεση του μεγαλύτερου κοινού αριθμού στοιχείων ανάμεσα σε δύο αλφαριθμητικά. Τα στοιχεία που την αποτελούν πρέπει να βρίσκονται σε τέτοιες θέσεις μέσα και στα δύο αλφαριθμητικά έτσι ώστε οι αριθμοί των θέσεων αυτών να βρίσκονται σε αύξουσα σειρά. Επίσης τα στοιχεία αυτά δεν είναι απαραίτητο να βρίσκονται συνεχόμενα, σε γειτονικές δηλαδή θέσεις, σε κάποιο από τα δύο αλφαριθμητικά.

Με βάση αυτήν την περιγραφή του προβλήματος, θα μπορούσαμε να αναγάγουμε το πρόβλημα στην σταδιακή διαγραφή κάποιων από τα στοιχεία που αποτελούν τα δύο αλφαριθμητικά μέχρις ότου να καταλήξουμε σε μια κοινή ακολουθία που να είμαστε όμως σίγουροι ότι αποτελείται από το μέγιστο δυνατό πλήθος στοιχείων.

Κάτι τέτοιο δεν είναι ιδιαίτερα εύκολο, μιας και απαιτείται πολύ καλός χειρισμός των δεδομένων του προβλήματος αλλά και των δεδομένων που προκύπτουν κατά την επίλυσή του για να μπορέσει κάποιος να απαντήσει μετά βεβαιότητας ότι κατέληξε στην μεγαλύτερη κοινή ακολουθία, ειδικά όταν αναφερόμαστε σε αλφαριθμητικά μεγάλου μήκους.

3.2.2. Η σημασία του αλφάβητου

Μέχρι τώρα δεν έχουμε αναφερθεί καθόλου στην έννοια του αλφάβητου, σε σχέση πάντα με την δημιουργία και χρήση αλφαριθμητικών. Με τον όρο αλφάβητο εννοούμε τους διακριτούς εκείνους χαρακτήρες, ψηφία, σύμβολα στίξης κτλ. τα οποία χρησιμοποιήσαμε για την δημιουργία ενός αλφαριθμητικού.

Σε ότι αφορά το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας, όταν αναφερόμαστε στο αλφάβητο εννοούμε το αλφάβητο που χρησιμοποιήθηκε για την κατασκευή και των δύο αλφαριθμητικών που εξετάζουμε. Βέβαια αν θα μπορούσε ίσως κάποιος να διαχωρίσει το αλφάβητο από το οποίο προήλθε το ένα αλφαριθμητικό από το αλφάβητο από το οποίο προήλθε το άλλο.

Ας υποθέσουμε ότι έχουμε δύο αλφαριθμητικά το Χ και το Υ. Αν το Χ περιλαμβάνει κάποιον ή κάποιους χαρακτήρες οι οποίοι δεν συμπεριλαμβάνονται στο αλφάβητο που χρησιμοποιήσαμε για την δημιουργία του Υ τότε σίγουρα αυτοί οι χαρακτήρες δεν θα μπορούν να βρεθούν μέσα στο Υ και σίγουρα δεν θα περιλαμβάνονται και στην μεγαλύτερη κοινή ακολουθία. Για παράδειγμα έστω ότι το Χ είναι το ACGFEEEGAAA και το Υ είναι το AGGGEEE. Με βάση όσα έχουμε αναφέρει μέχρι στιγμής το Χ έχει μήκος 11 και το Υ έχει μήκος 7.

Πίνακας 3.1 Το αλφαριθμητικό Χ

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
X	A	C	G	F	E	E	E	G	A	A	A

Πίνακας 3.2 Το αλφαριθμητικό Υ

	0	**1**	**2**	**3**	**4**	**5**	**6**
Y	A	G	G	D	E	E	E

Κάνοντας μια απλή σάρωση του αλφαριθμητικού Χ και σημειώνοντας τα διακριτά στοιχεία από τα οποία αποτελείται καταλήγουμε στο συμπέρασμα ότι το αλφάβητο το οποίο χρησιμοποιήθηκε για την δημιουργία του είναι το S_1 = {A, C, E, F, G}. Πραγματοποιώντας ακριβώς την ίδια σάρωση και στα στοιχεία που αποτελούν το Υ καταλήγουμε ότι το αλφάβητο από το οποίο προήλθε το Υ είναι το S_2 = {A, D, E, G}. Συγκρίνοντας τα δύο αυτά σύνολα χαρακτήρων παρατηρούμε ότι και στα δύο εμφανίζονται τα στοιχεία A, E και G ενώ το πρώτο περιλαμβάνει επίσης τα C και F ενώ το δεύτερο περιλαμβάνει επίσης το D. Αν θελήσει όμως κανείς να ορίσει το αλφάβητο από το οποίο προήλθαν και τα δύο αλφαριθμητικά τότε αυτό θα είναι η ένωση των S_1 και S_2 , θα είναι δηλαδή το σύνολο S = {A, C, D, E, F, G}.

Προσπαθώντας κανείς να εξετάσει λοιπόν ποια είναι η μεγαλύτερη κοινή ακολουθία των X και Y δεν έχει νόημα να εξετάζει τους χαρακτήρες του X που είναι οι C και F, διότι αυτοί δεν υπάρχουν στο Y και άρα δεν θα μπορούσαν να αποτελούν μέρος της μεγαλύτερης κοινής ακολουθίας τους. Αντίστοιχα οι χαρακτήρες του Y που είναι ίσοι με D θα έπρεπε να αγνοηθούν.

Επίσης θα μπορούσε κανείς να εξετάσει την ακραία περίπτωση που τα δύο αλφαριθμητικά έχουν κατασκευαστεί από δύο αλφάβητα που έχουν ένα μόνο κοινό στοιχείο ή ακόμα και κανένα. Σε αυτές τις περιπτώσεις η εύρεση της μεγαλύτερης κοινής ακολουθίας απλοποιείται αρκετά.

3.3. Προσέγγιση βάση της εξαντλητικής μεθόδου

Η πιο απλή προσέγγιση που θα μπορούσε κανείς να έχει πάνω στο συγκεκριμένο πρόβλημα είναι να εξετάσει όλες τις πιθανές ακολουθίες του μικρότερου σε μήκος αλφαριθμητικού. Στην συνέχεια θα έπρεπε να εντοπίσει ποιες από αυτές τις ακολουθίες υπάρχουν και στο άλλο αλφαριθμητικό και τέλος ποια από αυτές είναι η μεγαλύτερη.

Ο λόγος που εξετάζουμε τις υποακολουθίες του μικρότερου από τα δύο αλφαριθμητικά, είναι γιατί, αν επιλέγαμε να εξετάσουμε την ύπαρξη των υποακολουθιών του μεγάλου αλφαριθμητικού στο μικρό, κάποιες από αυτές τις υποακολουθίες θα είχαν μεγαλύτερο μήκος από αυτό του αλφαριθμητικού και σίγουρα δεν θα μπορούσαμε να τις εντοπίσουμε. Με αυτόν τον τρόπο θα κάναμε άσκοπους υπολογισμούς που σίγουρα δεν θα μας επέστρεφαν κάποιο αποτέλεσμα. Επομένως αυτό που εξετάζουμε είναι την ύπαρξη των υποακολουθιών του μικρότερου σε μήκους αλφαριθμητικού στο αλφαριθμητικό με το μεγαλύτερο μήκος.

Για παράδειγμα έστω ότι έχουμε το αλφαριθμητικό X ,που έχει μήκος 6 και αποτελείται από τα στοιχεία GTTCAA, και το αλφαριθμητικό Y που έχει μήκος 4 και αποτελείται από τα στοιχεία GACA. Τα αλφαριθμητικά αυτά παρουσιάζονται στους πίνακες που ακολουθούν.

Πίνακας 3.3 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**
X	G	T	T	C	A	A

Πίνακας 3.4 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**
Y	G	A	C	A

Σύμφωνα με την μέθοδο που περιγράψαμε θα έπρεπε κανείς να εξετάσει όλες τις πιθανές υποακολουθίες που μπορούμε να εντοπίσουμε στο Y. Οι υποακολουθίες αυτές είναι $2^4 = 16$.

Παρακάτω παρουσιάζεται ένας πίνακας με τις υποακολουθίες αυτές. Στην πρώτη στήλη αναγράφεται ο αριθμός της υποακολουθίας. Στις στήλες που ακολουθούν αναγράφονται τα στοιχεία του Y που χρησιμοποιήθηκαν στην δημιουργία της κάθε

υποακολουθίας. Τα στοιχεία του Υ είναι 4, οπότε σε κάθε μια στήλη αναγράφεται είτε το στοιχείο που βρίσκεται στην αντίστοιχη θέση του Υ, εάν αυτό έχει χρησιμοποιηθεί στην ακολουθία που εξετάζουμε ή το σύμβολο =, για να δηλώσουμε ότι το στοιχείο που βρίσκεται στην αντίστοιχη θέση του Υ δεν έχει χρησιμοποιηθεί στην συγκεκριμένη ακολουθία.

Πίνακας 3.5 Οι υποακολουθίες του Υ

1	G	A	C	A
2	G	A	C	=
3	G	A	=	A
4	G	A	=	=
5	G	=	C	A
6	G	=	C	=
7	G	=	=	A
8	G	=	=	=
9	=	A	C	A
10	=	A	C	=
11	=	A	=	A
12	=	A	=	=
13	=	=	C	A
14	=	=	C	=
15	=	=	=	A
16	=	=	=	=

Στην συνέχεια θα έπρεπε κανείς να προσπελάσει το αλφαριθμητικό Χ, προσπαθώντας να εξακριβώσει αν η συγκεκριμένη υποακολουθία του Υ υπάρχει και στο Χ. Όμως αυτήν την φορά θα έπρεπε να εντοπιστούν όλα τα στοιχεία της συγκεκριμένης υποακολουθίας στο Χ για να έχουμε μια θετική απάντηση στο ερώτημά μας.

Για παράδειγμα εξετάζοντας την πρώτη από τις υποακολουθίες του Υ, θα επιχειρούσαμε να εντοπίσουμε στο Χ

- το γράμμα G. Αυτό το γράμμα όντως υπάρχει και είναι στην θέση 0 του Χ.
- το γράμμα A. Αυτό το γράμμα όντως υπάρχει και είναι στην θέση 4 του Χ.
- το γράμμα C. Αυτό το γράμμα όμως παρόλο που υπάρχει στο Χ δεν μπορεί να χρησιμοποιηθεί αφού βρίσκεται στην θέση 3 του Χ, ενώ εμείς εξετάζουμε την ύπαρξή του σε κάποια θέση του Χ που να βρίσκεται μετά την 4.

Με βάση λοιπόν τα παραπάνω οι υπολογισμοί μας για την συγκεκριμένη υποακολουθία του Υ, θα σταματούσαν και θα καταλήγαμε στο συμπέρασμα ότι αυτή η υποακολουθία δεν υπάρχει στο Χ. Οι υπολογισμοί μας θα συνέχιζαν με την προσπάθειά μας να εντοπίσουμε την επόμενη υποακολουθία του Υ στο Χ. Όταν εντοπίσουμε μια υποακολουθία του Υ που να υπάρχει στο Χ, πρέπει να την

αποθηκεύσουμε έτσι ώστε να γνωρίζουμε ποια είναι και ποιο είναι το μήκος της. Στην συνέχεια αν εντοπίσουμε και κάποια άλλη υποακολουθία του Y στο X, πρέπει να γίνει ένας έλεγχος αν το μήκος της καινούργια υποακολουθίας που εντοπίσαμε είναι μεγαλύτερο από το μήκος αυτής που είχαμε εντοπίσει προηγούμενα. Αν κάτι τέτοιο ισχύει, αποθηκεύουμε την καινούργια, διαφορετικά δεν κάνουμε κάποια αλλαγή.

Αυτό που έχουμε να παρατηρήσουμε είναι ότι ανάλογα με το πλήθος των στοιχείων του αλφαριθμητικού προκύπτουν και οι υποακολουθίες που μπορούμε να κατασκευάσουμε από αυτό. Το πλήθος των υποακολουθιών αυτών αυξάνεται εκθετικά, και μάλιστα είναι δύναμη του 2, με εκθέτη το πλήθος των στοιχείων του αλφαριθμητικού. Είναι βέβαια κατανοητό ότι όταν το μήκος του αλφαριθμητικού αυξηθεί σημαντικά το πλήθος των υποακολουθιών που έχουμε να εξετάσουμε αυξάνεται με εκθετικό ρυθμό. Για παράδειγμα αν το μήκος του Y ήταν 10, το πλήθος των υποακολουθιών του είναι $2^{10} = 1024$. Κάτι τέτοιο φυσικά είναι μεγάλο εμπόδιο στους υπολογισμούς μας, αφού μια μικρή αύξηση του μήκους του Y, μόλις κατά 4 χαρακτήρες, αποφέρει μια μεγάλη αύξηση στις ακολουθίες που έχουμε να εξετάσουμε. Από 16 που ήταν στην πρώτη περίπτωση τώρα φτάνουν τις 1024, μια αύξηση κατά 1008 ακολουθίες.

ΚΕΦΑΛΑΙΟ 4

Ο ΑΛΓΟΡΙΘΜΟΣ ΤΗΣ ΣΕ ΒΑΘΟΣ ΠΡΩΤΑ ΑΝΑΖΗΤΗΣΗΣ ΤΟΥ LCS – DEPTH FIRST SEARCH LCS (DFS – LCS)

4.1. Εισαγωγή

Η προσέγγιση η οποία έχει υιοθετηθεί για τον DFS – LCS είχε ως κύριο μέλημα την μείωση του απαιτούμενου χώρου που καταλαμβάνουν τα δεδομένα του προβλήματος στην μνήμη του υπολογιστή. Όπως έχουμε αναφέρει προηγούμενα ο κλασικός αλγόριθμος για την εύρεση της μεγαλύτερης κοινής ακολουθίας, ο αλγόριθμος των Hunt και Szymanski, πάνω στον οποίο έχουν στηριχτεί και πολλοί άλλοι αλγόριθμοι παρουσιάζει ένα σημαντικό μειονέκτημα και αυτό είναι οι μεγάλες απαιτήσεις που έχει σε μνήμη. Για να μπορέσουμε λοιπόν να μειώσουμε τις απαιτήσεις αυτές, ο DFS – LCS αλγόριθμος χρησιμοποιεί διαφορετικές δομές με διαφορετική λειτουργία για να φτάσει στην λύση του προβλήματος.

Ο DFS – LCS στηρίζει την λειτουργία του σε μια αναδρομική συνάρτηση η οποία διεκπεραιώνει όλους τους απαραίτητους υπολογισμούς. Η συνάρτηση αυτή εξετάζει όλους τους διαφορετικούς συνδυασμούς που μπορούμε να εντοπίσουμε στο μικρότερο από τα δύο αλφαριθμητικά που εξετάζουμε προκειμένου να καταλήξει στην μεγαλύτερη κοινή ακολουθία. Σε αυτό το κεφάλαιο περιγράφεται αναλυτικά ο DFS – LCS, καθώς και οι τρόποι με τους οποίους μειώνει το πλήθος των συνδυασμών που ελέγχει ο αλγόριθμος για να καταλήξει στο επιθυμητό αποτέλεσμα. Περιλαμβάνεται ο ψευδοκώδικας του αλγορίθμου καθώς και ένα αντιπροσωπευτικό παράδειγμα της λειτουργίας του.

4.2. Περιγραφή του αλγορίθμου

Ουσιαστικά σε κάθε βήμα εκτέλεσης του αλγορίθμου, η αναδρομική συνάρτηση εξετάζει εάν η χρήση του τρέχοντος γράμματος του Y στην δημιουργία της μεγαλύτερης κοινής ακολουθίας επιφέρει καλύτερο αποτέλεσμα από τη μη χρησιμοποίηση του τρέχοντος γράμματος.

Για να γίνει πιο αποδοτική η εκτέλεση του αλγορίθμου έχουν χρησιμοποιηθεί κάποιες μέθοδοι που εξασφαλίζουν την καλύτερη λειτουργία του. Αυτές οι μέθοδοι ονομαστικά είναι

- Ο έλεγχος μόνο των διαφορετικών υποακολουθιών του αλφαριθμητικού
- Η συνάρτηση που αγνοεί κάποια από τα στοιχεία του αλφαριθμητικού, χωρίς να πραγματοποιεί κάποιον έλεγχο σε αυτά
- Η χρήσης της δομής cache

Οι μέθοδοι αυτοί περιγράφονται αναλυτικά στις παραγράφους που ακολουθούν.

4.2.1. Έλεγχος μόνο των διαφορετικών υποακολουθιών του αλφαριθμητικού

Βέβαια όπως έχουμε ήδη αναφέρει η διαδικασία της εξέτασης όλων των πιθανών συνδυασμών του αλφαριθμητικού Υ έχει ως αποτέλεσμα την εξέταση ενός πολύ μεγάλου αριθμού συνδυασμών. Για να μειωθεί αυτός ο αριθμός των πιθανών συνδυασμών που έχουμε να ελέγξουμε ο αλγόριθμος χρησιμοποιεί μια συνάρτηση βάση της οποίας κάποια από τα στοιχεία του Υ αγνοούνται.

Για παράδειγμα, έστω ότι έχουμε το αλφαριθμητικό Υ που αποτελείται από τα στοιχεία GAAC.

Πίνακας 4.1 Το αλφαριθμητικό Υ

	0	**1**	**2**	**3**
Υ	G	A	A	C

Το Υ έχει μήκος 4, αποτελείται από 4 στοιχεία, επομένως από το Υ προκύπτουν $2^4 =$ 16 υποακολουθίες. Για να καταλήξουμε σε αυτόν τον αριθμό θεωρούμε ότι η θέση στην οποία θα εντοπίσουμε ένα στοιχείο, είναι σημαντική και επομένως δεν μπορεί να αγνοηθεί. Άρα για κάθε μια από τις θέσεις του Υ έχουμε ως δεδομένο ότι είτε θα συμπεριλαμβάνεται το στοιχείο της συγκεκριμένης θέσης στην υποακολουθία που συμπεριλαμβάνουμε, είτε όχι. Επομένως κάθε μια από τις θέσεις μπορεί να βρεθεί σε 2 καταστάσεις, να φιλοξενεί το γράμμα ή να μην το φιλοξενεί. Στις περιπτώσεις που το στοιχείο συμπεριλαμβάνεται στην δημιουργία της ακολουθίας τότε αναγράφεται το στοιχείο στην συγκεκριμένη θέση, ενώ αν δεν συμπεριλαμβάνεται τότε αναγράφεται το σύμβολο =. Οι υποακολουθίες του αλφαριθμητικού Υ παρουσιάζονται στον πίνακα που ακολουθεί.

Πίνακας 4.2 Οι υποακολουθίες του αλφαριθμητικού Υ

1	G	A	A	C
2	G	A	A	=
3	G	A	=	C
4	G	A	=	=
5	G	=	A	C
6	G	=	A	=
7	G	=	=	C
8	G	=	=	=
9	=	A	A	C
10	=	A	A	=
11	=	A	=	C
12	=	A	=	=
13	=	=	A	C

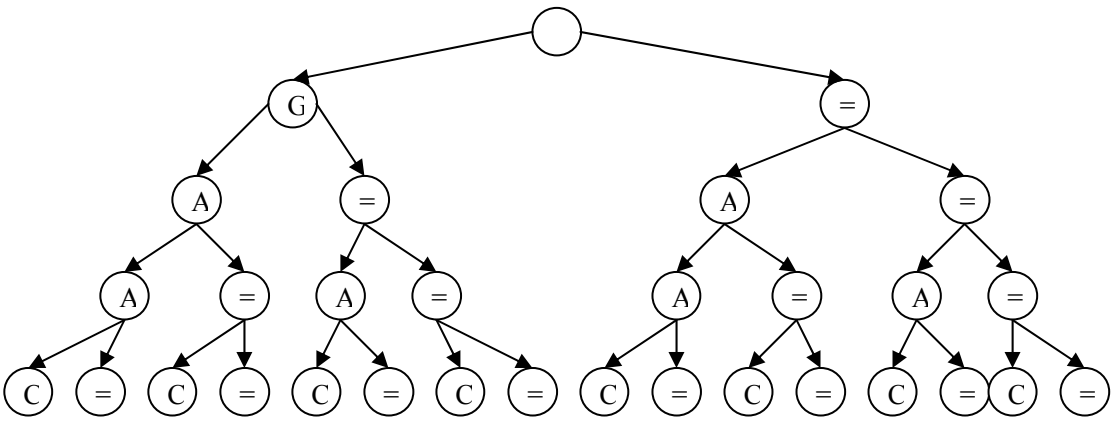
14	=	=	A	=
15	=	=	=	C
16	=	=	=	=

Για παράδειγμα για την δημιουργία της υποακολουθίας με τον αριθμό 3 έχουν χρησιμοποιηθεί τα στοιχεία που βρίσκονται στις θέσεις 0, 1, 3 του Y, ενώ το στοιχείο στην θέση 2 έχει παραληφθεί. Πρέπει να παρατηρήσουμε εδώ ότι η ακολουθία με τον αριθμό 3 και η ακολουθία με τον αριθμό 5 αποτελούνται από ακριβώς τα ίδια στοιχεία, τα οποία είναι διατεταγμένα και με την ίδια σειρά, τα GAC. Αυτό που διαφέρει ανάμεσα στις δύο περιπτώσεις είναι ότι για την δημιουργία της υποακολουθίας με τον αριθμό 3 έχει χρησιμοποιηθεί το A που βρίσκεται στην θέση 1 του Y, ενώ για την δημιουργία της υποακολουθίας με τον αριθμό 5 έχει χρησιμοποιηθεί το A που βρίσκεται στην θέση 2 του Y.

Επομένως μπορούμε να πούμε ότι οι υποακολουθίες ενός αλφαριθμητικού είναι δύναμη του 2 με εκθέτη το μήκος του αλφαριθμητικού, όταν λαμβάνουμε υπόψιν μας και τις θέσεις στις οποίες εντοπίζουμε ένα στοιχείο. Αν δεν λαμβάνουμε υπόψιν μας την θέση στην οποία εντοπίζουμε ένα στοιχείο, οι διαφορετικές υποακολουθίες ενός αλφαριθμητικού μπορεί να είναι πολύ λιγότερες. Οι διαφορετικές υποακολουθίες του αλφαριθμητικού Y είναι ουσιαστικά 12 και ο αριθμός αυτός εξαρτάται από το αλφαριθμητικό που ελέγχουμε κάθε φορά και φυσικά από την μορφή που αυτό έχει.

Τις υποακολουθίες του Y θα μπορούσαμε να τις παρουσιάσουμε και με το δένδρο που απεικονίζεται στο παρακάτω σχήμα.

Σχήμα 4.1 Οι υποακολουθίες του αλφαριθμητικού Y σε μορφή δένδρου



Το δένδρο αυτό έχει βάθος ίσο με το μήκος του αλφαριθμητικού και σε κάθε επίπεδο εξετάζουμε την ύπαρξη ή όχι του στοιχείου του αλφαριθμητικού στην υποακολουθία. Για παράδειγμα στο πρώτο επίπεδο και ακολουθώντας το αριστερό υποδένδρο, εμφανίζονται όλες οι υποακολουθίες του αλφαριθμητικού Y που συμπεριλαμβάνουν το πρώτο στοιχείο του, που είναι το γράμμα G. Οι ακολουθίες αυτές είναι 8, όσα είναι και τα φύλλα αυτού του υποδένδρου. Αντίστοιχα στο δεξί υποδένδρο, παρουσιάζονται οι 8 υποακολουθίες που δεν συμπεριλαμβάνουν το G.

Ο αλγόριθμος εξετάζει κάθε μια διαφορετική ακολουθία μονάχα μια φορά, οπότε δεν καταναλώνεται στον έλεγχο ακολουθιών που είναι ίδιες αλλά τα στοιχεία τους βρίσκονται σε διαφορετικές θέσεις στο Y. Για παράδειγμα, χρησιμοποιώντας και πάλι

το αλφαριθμητικό Y που περιγράψαμε προηγούμενα η ακολουθία GAC θα ελεγχόταν μόνο μια φορά, παρόλο που μπορούμε να την δημιουργήσουμε με 2 διαφορετικούς τρόπους.

4.2.2. Η συνάρτηση που αγνοεί κάποιους χαρακτήρες

Επανερχόμενοι τώρα στην συνάρτηση του αλγορίθμου που αγνοεί κάποιους από τους χαρακτήρες του αλφαριθμητικού κατά την εκτέλεσή του έχουμε να παρατηρήσουμε τα εξής. Έστω ότι εξετάζουμε το αλφαριθμητικό Y, που παρουσιάστηκε προηγούμενα και ελέγχουμε αν η χρησιμοποίηση του στοιχείου της θέσης 1 του Y, που είναι το γράμμα A, είναι προτιμότερο ή όχι να συμπεριληφθεί στην μεγαλύτερη κοινή ακολουθία. Ας υποθέσουμε ότι καταλήγουμε στο συμπέρασμα ότι είναι προτιμότερο το γράμμα A να μην συμπεριληφθεί, παρόλο που υπάρχει και στο X και στο Y, διότι αν συμπεριλάβουμε ένα A την συγκεκριμένη στιγμή στην μεγαλύτερη κοινή ακολουθία καταλήγουμε σε χειρότερο αποτέλεσμα από την περίπτωση που δεν το συμπεριλάβουμε αυτή τη στιγμή.

Στην συνέχεια θα επιχειρήσουμε να κάνουμε τον ίδιο έλεγχο και με το επόμενο στοιχείο του Y, που όμως στην προκειμένη περίπτωση είναι και πάλι το γράμμα A, που όμως αυτή τη φορά βρίσκεται στην θέση 2. Σε αυτές τις περιπτώσεις που περιγράψαμε, μόλις ο αλγόριθμος συναντήσει το γράμμα A θα το αγνοήσει και δεν θα προβεί σε κανέναν απολύτως έλεγχο, αφού ουσιαστικά αυτό που θα έκανε, θα ήταν να επαναλάβει τους ελέγχους που κάναμε για το προηγούμενο A και πάλι θα καταλήγαμε στο ίδιο συμπέρασμα.

Ουσιαστικά ο αλγόριθμος γνωρίζει ανά πάσα στιγμή ποια είναι τα στοιχεία του Y που χρειάζεται να αγνοεί σε κάθε αναδρομική του κλίση. Βέβαια αυτά τα στοιχεία δεν αγνοούνται μέχρι την ολοκλήρωση του αλγορίθμου, γιατί κάτι τέτοιο θα μας οδηγούσε σε εσφαλμένα αποτελέσματα. Τα στοιχεία αγνοούνται μέχρι να συμπεριληφθεί κάποιο άλλο στοιχείο στην μεγαλύτερη κοινή ακολουθία, οπότε από το σημείο αυτό και έπειτα αν ξανασυναντήσουμε κάποιο στοιχείο που πρωτύτερα αγνοούσαμε το ελέγχουμε κανονικά.

4.2.3. Η χρήση της δομής cache

Όπως έχουμε ήδη αναφέρει ο αλγόριθμος ελέγχει κάθε διαφορετική υποακολουθία του Y με σκοπό να διαπιστώσει αν αυτή η υποακολουθία υπάρχει με την ίδια ακριβώς μορφή και στο αλφαριθμητικό X. Από αυτές που υπάρχουν και στα δύο αλφαριθμητικά ο αλγόριθμος εντοπίζει και επιστρέφει την μεγαλύτερη, δίνοντάς μας έτσι την μεγαλύτερη κοινή ακολουθία των X και Y.

Όμως παρόλο που ο αλγόριθμος εξετάζει κάθε μια υποακολουθία μόνο μια φορά, μερικές υποακολουθίες αποτελούνται από κάποια κοινά κομμάτια. Τα κοινά αυτά κομμάτια εξετάζονται τόσες φορές όσες και οι υποακολουθίες που τα εμπεριέχουν. Προκειμένου να εξετάζεται όχι μόνο κάθε διαφορετική υποακολουθία αλλά και κάθε κομμάτι αυτής μόνο μια φορά, ο αλγόριθμος χρησιμοποιεί μια δομή που ονομάζεται cache. Στην δομή αυτή, αποθηκεύονται πληροφορίες για κάθε ένα στοιχείο που εντοπίζουμε ότι υπάρχει και στα δύο αλφαριθμητικά.

Για παράδειγμα έστω ότι εξετάζουμε το στοιχείο που βρίσκεται στην θέση j του αλφαριθμητικού Y και έχουμε καταλήξει ότι το στοιχείο αυτό υπάρχει και στο X

στην θέση i. Είναι πιθανό το στοιχείο αυτό που βρίσκεται συγκεκριμένα στην θέση j του Y και στην θέση i του X, να χρησιμοποιείται σε πολλές από τις υποακολουθίες του Y. Οπότε προκειμένου να αποφύγουμε την επανάληψη των υπολογισμών μας για την εύρεση της μεγαλύτερης κοινής ακολουθίας από το σημείο αυτό και μέχρι το τέλος του Y, αποθηκεύουμε την πληροφορία που έχουμε αποκτήσει μέχρι τώρα για αυτό το στοιχείο στην δομή cache.

Η πληροφορία αυτή περιλαμβάνει την θέση του στοιχείου στο αλφαριθμητικό Y, την θέση του στοιχείου στο αλφαριθμητικό X, το μήκος του LCS που έχουμε βρει από το σημείο αυτό και μέχρι το τέλος του Y, καθώς και το ποια είναι αυτή η μεγαλύτερη κοινή ακολουθία. Με αυτόν τον τρόπο την επόμενη φορά που κάποια υποακολουθία θα περιλαμβάνει το στοιχείο, γίνεται πρώτα ένας έλεγχος στην δομή cache για το αν έχουμε αποθηκευμένη την ζητούμενη πληροφορία. Αν ισχύει κάτι τέτοιο δεν χρειάζεται να κάνουμε κανέναν περαιτέρω υπολογισμό για το LCS από το στοιχείο αυτό και μέχρι το τέλος και απλά το αντικαθιστούμε με την πληροφορία που μας παρέχει η cache. Με αυτόν τον τρόπο κάθε διαφορετικός συνδυασμός, δηλαδή στοιχείο του X και στοιχείο του Y, υπολογίζεται μονάχα μια φορά.

4.3. Λειτουργία του αλγορίθμου

Στην παράγραφο αυτή θα περιγράψουμε αναλυτικά την λειτουργία του αλγορίθμου. Επίσης θα περιγράψουμε τον τρόπο με τον οποίο χρησιμοποιεί τις μεθόδους που αναλύσαμε στην προηγούμενη παράγραφο για την καλύτερη απόδοσή του.

Ο αλγόριθμος αποτελείται από δύο στάδια

- Το στάδιο της αρχικοποίησης
- Το κυρίως σώμα του αλγορίθμου

Εκτός από την περιγραφή των σταδίων του αλγορίθμου θα αναλύσουμε και τις δομές που αυτός χρησιμοποιεί για την αποθήκευση της απαραίτητης πληροφορίας.

4.3.1. Η αρχικοποίηση του αλγορίθμου

Η αρχικοποίηση του αλγορίθμου έχει να κάνει με την μετατροπή των δεδομένων του αλγορίθμου σε μορφή τέτοια ώστε να μπορεί να χρησιμοποιηθεί από τον αλγόριθμο. Η αρχικοποίηση αυτή αποτελείται από δύο βήματα.

Το πρώτο βήμα περιλαμβάνει τον προσδιορισμό του αλφάβητου από το οποίο έχουν δημιουργηθεί τα δύο αλφαριθμητικά, των οποίων επιθυμούμε να βρούμε την μεγαλύτερη κοινή ακολουθία. Το αλφάβητο αυτό είναι εύκολο να προσδιοριστεί με μια απλή ανάγνωση των αλφαριθμητικών. Με την ολοκλήρωση αυτού του βήματος, γνωρίζουμε ακριβώς πόσα είναι τα διαφορετικά εκείνα στοιχεία από τα οποία έχουν δημιουργηθεί τα αλφαριθμητικά αλλά και ποιο ακριβώς είναι το αλφάβητο.

Το δεύτερο βήμα της αρχικοποίησης περιλαμβάνει την δημιουργία του Index Table του X, θεωρώντας πάντα ότι το X είναι το μεγαλύτερο σε μήκος από τα δύο αλφαριθμητικά που δίνονται ως είσοδο στον αλγόριθμο. Ο Index Table του X είναι ένας πίνακας του οποίου κάθε στοιχείο είναι και μια λίστα. Το πλήθος των στοιχείων του πίνακα, και κατ' επέκταση το πλήθος των λιστών που θα δημιουργηθούν σε αυτό το βήμα, είναι ίσο με τα διαφορετικά στοιχεία του αλφαβήτου το οποίο υπολογίσαμε

στο προηγούμενο βήμα της αρχικοποίησης του αλγορίθμου. Στοιχεία της πρώτης λίστας είναι οι θέσεις στο X στις οποίες εντοπίσαμε το πρώτο στοιχείο της αλφαβήτου. Για να δημιουργήσουμε το Index Table του X προσπελάζουμε ένα προς ένα τα στοιχεία που το αποτελούν και προσθέτουμε την τρέχουσα θέση στην αντίστοιχη λίστα, ανάλογα με το ποιο στοιχείο του αλφαβήτου φιλοξενείται στην θέση αυτή.

Για παράδειγμα έστω ότι το αλφαριθμητικό X έχει μήκος 20 και αποτελείται από τα στοιχεία AGCCTGAGTCCAATGCGTAA και το αλφαριθμητικό Y έχει μήκος 9 και αποτελείται από τα στοιχεία AGATCGATG. Στους πίνακες που ακολουθούν παρουσιάζονται τα δύο αλφαριθμητικά.

Πίνακας 4.3 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
X	A	G	C	C	T	G	A	G	T	C	C	A	A	T	G	C	G	T	A	A

Πίνακας 4.4 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Y	A	G	A	T	C	G	A	T	G

Στο πρώτο βήμα της αρχικοποίησης του αλγορίθμου, όπως έχουμε αναφέρει θα εξακριβώσουμε ποιο είναι το αλφάβητο από το οποίο έχουν δημιουργηθεί τα δύο αλφαριθμητικά. Το αλφάβητο αυτό αποτελείται από 4 στοιχεία και είναι το S = { A, C, G, T }.

Στο δεύτερο βήμα της αρχικοποίησης θα κατασκευάσουμε τον Index Table του X. Αυτός ο πίνακας θα αποτελείται από τέσσερα συνολικά στοιχεία, αφού τέσσερα είναι τα στοιχεία του αλφαβήτου, και κάθε ένα από αυτά τα στοιχεία θα είναι μια λίστα. Αρχίζοντας την προσπέλαση του X, διαπιστώνουμε ότι το πρώτο στοιχείο του X είναι το γράμμα A του αλφαβήτου, οπότε στην πρώτη λίστα θα προσθέσουμε τον αριθμό 0 που είναι η θέση στην οποία εντοπίσαμε το γράμμα. Το δεύτερο στοιχείο του X είναι το γράμμα G, που είναι το τρίτο γράμμα της αλφαβήτου. Άρα στην τρίτη λίστα θα προσθέσουμε τον αριθμό 1. Συνεχίζοντας με τον ίδιο τρόπο, φτάνοντας στο στοιχείο 19 του X, προσθέτουμε τον αριθμό 19 στην πρώτη λίστα και ο Index Table του X έχει την μορφή που φαίνεται παρακάτω, όπου στην πρώτη στήλη του πίνακα για λόγους ευκολίας αναγράφεται το γράμμα της αλφαβήτου.

Πίνακας 4.5 Ο Index Table του X

	0	**1**	**2**	**3**	**4**	**5**
A	0	6	11	12	18	19
C	2	3	9	10	15	
G	1	5	7	14	16	
T	4	8	13	17		

4.3.2. Το κυρίως σώμα του αλγορίθμου

Το κυρίως σώμα είναι η αναδρομική συνάρτηση που αναφέραμε προηγουμένως και είναι ουσιαστικά και το σημείο στο οποίο πραγματοποιούνται όλοι οι απαραίτητοι υπολογισμοί για την εύρεση της μεγαλύτερης κοινής ακολουθίας των δύο αλφαριθμητικών. Η αναδρομική αυτή συνάρτηση ονομάζεται find_lcs και παίρνει 7 παραμέτρους

- Το αλφαριθμητικό X
- Το αλφαριθμητικό Y
- Το μήκος του αλφαριθμητικού X που είναι n
- Το μήκος του αλφαριθμητικού Y που είναι m, θεωρούμε ότι n > m
- Την θέση εκκίνησης στο αλφαριθμητικό X, που ονομάζεται x_pos
- Την θέση εκκίνησης στο αλφαριθμητικό Y, που ονομάζεται y_pos
- Τον ignoremask πίνακα

Ο πίνακας ignoremask είναι 1xs διαστάσεων όπου s είναι το πλήθος των στοιχείων της αλφαβήτου από την οποία έχουμε δημιουργήσει τα δύο αλφαριθμητικά. Κάθε μια θέση του πίνακα ignoremask, αντιστοιχεί σε ένα από τα στοιχεία της αλφαβήτου. Τα στοιχεία του πίνακα ignoremask παίρνουν ως τιμές το 0 και το 1. Έστω ότι στη θέση k του πίνακα εντοπίσουμε την τιμή 0 σημαίνει ότι δεν πρέπει να αγνοήσουμε το k στοιχείο του αλφαβήτου στους υπολογισμούς που θα κάνουμε, ενώ αν εντοπίζαμε στην συγκεκριμένη θέση τον αριθμό 1 θα έπρεπε να αγνοήσουμε το στοιχείο αυτό, στην τρέχουσα εκτέλεση της κλήσης της αναδρομικής συνάρτησης.

Η αναδρομική συνάρτηση παράγει ως έξοδο, το μήκος της μεγαλύτερης κοινής ακολουθίας, το οποίο αποθηκεύεται στην μεταβλητή lcssz καθώς και τα στοιχεία που αποτελούν την ακολουθία αυτή, που βρίσκονται αποθηκευμένα στην μεταβλητή lcs. Η πρώτη κλήση της αναδρομικής συνάρτησης πραγματοποιείται ξεκινώντας από το πρώτο στοιχείο του Y και το πρώτο στοιχείο του X.

Αρχικά μέσα στην συνάρτηση εξετάζουμε εάν έχουμε φτάσει στο τέλος κάποιου από τα δύο αλφαριθμητικά. Εάν κάτι τέτοιο ισχύει η συνάρτηση επιστρέφει ότι το μήκος του LCS ξεκινώντας από την θέση x_pos του αλφαριθμητικού X και την θέση y_pos του αλφαριθμητικού Y είναι 0 και αυτή η κλήση της αναδρομικής συνάρτησης τερματίζεται.

Εάν δεν έχουμε φτάσει στο τέλος κάποιου από τα δύο αλφαριθμητικά τότε θέτουμε την τιμή -1 στις παρακάτω μεταβλητές

- στην μεταβλητή sz2, η οποία αντιπροσωπεύει το μήκος του LCS ξεκινώντας από την θέση x_pos στο X και την θέση y_pos στο Y , με δεδομένο ότι το στοιχείο που βρίσκεται στην θέση y_pos του Y θα χρησιμοποιηθεί στον υπολογισμό για το LCS.
- στην μεταβλητή sz1, η η οποία αντιπροσωπεύει το μήκος του LCS ξεκινώντας από την θέση x_pos στο X και την θέση y_pos στο Y , με δεδομένο ότι το στοιχείο που βρίσκεται στην θέση y_pos του Y δεν θα χρησιμοποιηθεί στον υπολογισμό για το LCS.

Στο σημείο αυτό η συνάρτηση ελέγχει εάν το τρέχων στοιχείο, δηλαδή το στοιχείο που βρίσκεται στην θέση y_pos του Y, θα πρέπει να αγνοηθεί ή αν θα πρέπει να πραγματοποιήσουμε κανονικά τους υπολογισμούς μας για το στοιχείο αυτό. Την πληροφορία αυτή την ανακτούμε από τον πίνακα ignoremask. Πραγματοποιείται δηλαδή ένας έλεγχος για το αν το στοιχείο που βρίσκεται στην αντίστοιχη θέση του πίνακα ignoremask έχει τιμή 0 ή 1.

Αν η τιμή που ανακτούμε από τον πίνακα ignoremask είναι 0, που όπως έχουμε ήδη αναφέρει σημαίνει ότι δεν πρέπει να αγνοήσουμε το τρέχων στοιχείο του Y, τότε συνεχίζουμε τους υπολογισμούς μας για την εύρεση της μεγαλύτερης κοινής ακολουθίας από το σημείο αυτό και μέχρι το τέλος του Y. Ο τρέχων χαρακτήρας του Y αποθηκεύεται στην μεταβλητή y_element.

Το επόμενο βήμα που θα ακολουθήσουμε στους υπολογισμούς μας περιλαμβάνει έναν έλεγχο για το αν το τρέχων στοιχείο του Y (y_element) υπάρχει στο αλφαριθμητικό X και είναι διαθέσιμο για χρήση. Με τον όρο διαθέσιμο για χρήση εννοούμε ότι δεν είναι αρκετό να υπάρχει το τρέχων στοιχείο του Y στο X, αλλά ότι θα πρέπει να βρίσκεται και σε τέτοια θέση στο X ώστε να μπορούμε να το χρησιμοποιήσουμε στην κατασκευή του LCS. Η θέση αυτή θα πρέπει να είναι η πρώτη μεγαλύτερη θέση μετά από την θέση του τελευταίου στοιχείου του X που χρησιμοποιήσαμε πρωτύτερα στους υπολογισμούς μας σε αυτόν τον κλάδο της αναδρομικής συνάρτησης.

Τον έλεγχο αυτόν τον πραγματοποιεί η συνάρτηση indexTable_find_first. Για να καλέσουμε αυτήν την συνάρτηση απαιτούνται δύο παράμετροι.

- Η πρώτη παράμετρος είναι ο τρέχων χαρακτήρας του Y τον οποίο επιθυμούμε να εντοπίσουμε μέσα στο X
- Η δεύτερη παράμετρος είναι η θέση στο X από την οποία και ύστερα καλούμαστε να εντοπίσουμε τον τρέχων χαρακτήρα του Y.

Η συνάρτηση αυτή επιστρέφει -1 αν δεν κατάφερε να εντοπίσει τον τρέχων χαρακτήρα του Y στο X ή την θέση στην οποία κατάφερε να τον εντοπίσει. Σε αυτό το σημείο πρέπει να σημειώσουμε ότι η αναζήτηση αυτή δεν πραγματοποιείται στην αρχική μορφή του αλφαριθμητικού X αλλά στον Index Table του X που κατασκευάσαμε στο δεύτερο στάδιο της αρχικοποίησης. Ο πίνακας αυτός με τον τρόπο που είναι κατασκευασμένος επιτρέπει στην γρήγορη αναζήτηση κάποιου στοιχείου. Για παράδειγμα όλα τα στοιχεία που είναι ίσα με το πρώτο στοιχείο της αλφαβήτου είναι στοιχεία της λίστας που βρίσκεται στην πρώτη θέση του πίνακα αυτού. Οπότε αν το τρέχων στοιχείο είναι ίσο με το πρώτο γράμμα της αλφαβήτου η αναζήτηση θα γίνει στην λίστα αυτή. Επίσης ως στοιχεία της λίστας αυτής έχουμε τις θέσεις στις οποίες εντοπίσαμε το στοιχείο αυτό στο X και μάλιστα αυτές οι θέσεις είναι διατεταγμένες σε αύξουσα σειρά λόγω του τρόπου που έγινε η κατασκευή της λίστας. Συνεπώς για την αναζήτηση μέσα στην λίστα χρησιμοποιείται η δυαδική αναζήτηση που είναι η πιο γρήγορη μορφή αναζήτησης.

Το αποτέλεσμα που επιστρέφει η συνάρτηση indexTable_find_first αποθηκεύεται στην μεταβλητή pos_found για να μπορέσει να χρησιμοποιηθεί στην συνέχεια. Αν η τιμή της μεταβλητής pos_found είναι -1 τότε η τιμή της μεταβλητής sz2 τίθεται ίση με το 0. Αυτό σημαίνει ότι ο τρέχων χαρακτήρας του Y δεν βρέθηκε μέσα στο X και συνεπώς το μήκος του LCS από τον τρέχων χαρακτήρα του Y μέχρι το τέλος είναι 0. Αν το αποτέλεσμα που επιστρέφει η indexTable_find_first είναι διαφορετικό του -1 τότε θέτουμε τιμές στις παρακάτω μεταβλητές

- Η τιμή της μεταβλητής is_cached τίθεται ίση με το 0. Ο ρόλος της μεταβλητής αυτής είναι να μας γνωστοποιεί εάν ένα ενδεχόμενο του συγκεκριμένου συνδυασμού βρίσκεται στην δομή cache ή όχι.
- Η τιμή της μεταβλητής tocheck τίθεται ίση με 1. Ο ρόλος της μεταβλητής αυτής είναι να μας υπενθυμίζει εάν πρέπει να πραγματοποιήσουμε όλους τους απαραίτητους συνδυασμούς για τον συνδυασμό αυτό ή εάν βρήκαμε ένα ενδεχόμενο του στην δομή cache οπότε και δεν χρειάζεται να προβούμε σε υπολογισμούς. Η τιμή 1 αντιπροσωπεύει την πρώτη κατάσταση ενώ η τιμή 0 την δεύτερη.

Στην συνέχεια καλείται η συνάρτηση indexTable_get. Πρέπει να διευκρινίσουμε σε αυτό το σημείο ότι η συνάρτηση indexTable_find_first που εξετάσαμε προηγούμενα μας επιστρέφει -1 αν δεν βρήκαμε το στοιχείο ή διαφορετικά την θέση του στοιχείου μέσα στον Index Table του X. Για να μας επιστραφεί η πραγματική θέση του στοιχείου στο αλφαριθμητικό X, πρέπει να ανακτήσουμε το στοιχείο που βρίσκεται στην pos_found θέση του Index Table του X στην αντίστοιχη λίστα. Αυτήν την λειτουργία την αναλαμβάνει η συνάρτηση indexTable_get η οποία καλείται με δύο παραμέτρους.

- Η πρώτη παράμετρος είναι ο τρέχων χαρακτήρας του Y που όπως είπαμε είναι αποθηκευμένος στην μεταβλητή y_element
- Και η δεύτερη παράμετρος είναι η θέση στον Index Table στην οποία εντοπίσαμε το στοιχείο αυτό.

Το αποτέλεσμα αυτής της συνάρτησης αποθηκεύεται στην μεταβλητή k. Σε αυτό το σημείο της εκτέλεσης αυτής της κλήσης της αναδρομικής συνάρτησης έχουμε εξακριβώσει ότι ο τρέχων χαρακτήρας του Y που βρίσκεται στην θέση y_pos και είναι αποθηκευμένος στην μεταβλητή y_element υπάρχει στο X στην θέση k και είναι διαθέσιμος για χρήση.

Μέχρι αυτό το σημείο της εκτέλεσης του αλγορίθμου δεν έχουμε εξακριβώσει εάν ο συγκεκριμένος συνδυασμός στοιχείων βρίσκεται αποθηκευμένος στην δομή cache ή όχι. Τον έλεγχο αυτόν τον πραγματοποιεί η συνάρτηση cache_find η οποία και καλείται σε αυτό το σημείο. Η κλήση αυτής της συνάρτηση απαιτεί δύο παραμέτρους

- Η πρώτη παράμετρος είναι η θέση του στοιχείου στο αλφαριθμητικό X, που είναι αποθηκευμένη όπως έχουμε πει στην μεταβλητή k
- Και η δεύτερη παράμετρος είναι η θέση του στοιχείου στο αλφαριθμητικό Y που όπως γνωρίζουμε είναι αποθηκευμένη στην μεταβλητή y_pos.

Η συνάρτηση αυτή επιστρέφει το μήκος του LCS που είναι αποθηκευμένο για αυτόν τον συνδυασμό στην cache καθώς και την πραγματική ακολουθία. Τα αποτελέσματα αυτά αποθηκεύονται στις μεταβλητές sz2 (για το μήκος) και lcs2 (για την ακολουθία). Αν η τιμή της μεταβλητής sz2 είναι μεγαλύτερη ή ίση με το 0, αυτό σημαίνει ότι όντως έχουμε ξανασυναντήσει τον συγκεκριμένο συνδυασμό σε κάποια προηγούμενη κλήση της αναδρομικής μας συνάρτησης και άρα υπάρχει αποθηκευμένος στην cache. Οπότε η τιμή της μεταβλητής tocheck τίθεται ίση με το 0 αφού δεν χρειάζεται να πραγματοποιήσουμε κανέναν υπολογισμό για τον συνδυασμό αυτό. Επίσης η τιμή της μεταβλητής is_cached τίθεται ίση με τη μονάδα, αφού όντως ο συνδυασμός βρίσκεται ήδη αποθηκευμένος στην cache.

Στην περίπτωση που δεν βρήκαμε τον συνδυασμό που αναζητούσαμε στην cache, πράγμα που σημαίνει ότι η μεταβλητή sz2 είχε τιμή μικτότερη του μηδενός, πρέπει να συνεχίσουμε με τους υπολογισμούς μας για να υπολογίσουμε το LCS από το τρέχων στοιχείο του Y μέχρι το τέλος του. Σε αυτό το σημείο ένας νέος προσωρινός πίνακας για το ignoremask δημιουργείται και ονομάζεται im.

Στην συνέχεια καλείται η συνάρτηση indexTable_get_cur_begin η οποία απαιτεί μια παράμετρο που είναι το τρέχων στοιχείο του Y. Η συνάρτηση αυτή υπολογίζει την θέση στην αντίστοιχη λίστα του Index Table του X από την οποία θα πρέπει να συνεχίσουν οι υπολογισμοί μας, και η τιμή αυτή αποθηκεύεται στην μεταβλητή pos. Στην συνέχεια καλείται η συνάρτηση indexTable_set_cur_begin η οποία απαιτεί δύο παραμέτρους.

- Η πρώτη παράμετρος είναι και πάλι το τρέχων στοιχείο του Y (y_element) για να μπορέσουμε να εντοπίσουμε την λίστα στην οποία αντιστοιχεί το στοιχείο αυτό
- Και η δεύτερη είναι η θέση στην αντίστοιχη λίστα του Index Table του X στην οποία εντοπίσαμε το τρέχων στοιχείο του Y αυξημένη κατά μια μονάδα.

Σε αυτό το σημείο έχουμε την αναδρομική κλήση στην συνάρτηση find_lcs για να μπορέσουμε να εντοπίσουμε το LCS από αυτό το σημείο και έπειτα. Η θέση εκκίνησης στο X είναι η θέση στην οποία εντοπίσαμε προηγούμενα το τότε τρέχων στοιχείο του Y αυξημένη κατά μια μονάδα (k + 1). Επίσης αυξημένη κατά μια μονάδα είναι και η θέση εκκίνησης στο Y που γίνεται πλέον y_pos + 1. Την θέση του πίνακα ignoremask έχει πάρει ο πίνακας im που ορίσαμε προηγούμενα. Τα αποτελέσματα από αυτήν την κλίση της συνάρτησης αποθηκεύονται στις μεταβλητές sz2 (που αποθηκεύει το μήκος) και lcsx (που αποθηκεύει την ίδια την ακολουθία). Με άλλα λόγια το LCS όταν συμπεριλαμβάνουμε το στοιχείο που βρίσκεται στην θέση y_pos του Y, είναι το ίδιο το στοιχείο (το οποίο εντοπίσαμε στην θέση k του X) ακολουθούμενο από το καλύτερο LCS που μπορούμε να βρούμε από εκείνο το σημείο μέχρι το τέλος.

Ουσιαστικά ο αλγόριθμος έχει φτάσει στο στάδιο της εκτέλεσης της αναδρομικής του συνάρτησης στο οποίο θα σταματήσει η εκτέλεση της πρώτης κλίσης και θα γίνει η εκκίνηση της δεύτερης, η οποία φυσικά θα λειτουργήσει με τον ίδιο ακριβώς τρόπο. Όταν ολοκληρωθούν όλες αυτές οι αναδρομικές κλίσεις θα συνεχίσει κανονικά η εκτέλεση της πρώτης αναδρομικής κλίσης. Για να γίνει κάτι τέτοιο, πρέπει να επαναφέρουμε την θέση από την οποία ξεκινάμε την ερευνά μας στον Index Table του X στην προηγούμενη τιμή της. Αυτό γίνεται καλώντας την συνάρτηση indexTable_set_cur_begin η οποία παίρνει ,όπως έχουμε αναφέρει, δύο παραμέτρους.

- Η μια παράμετρος είναι και πάλι το στοιχείο του Y το οποίο είναι το τρέχων και είναι αποθηκευμένο στην μεταβλητή y_element
- Η δεύτερη παράμετρος είναι η θέση pos που είναι η θέση στην αντίστοιχη λίστα για το συγκεκριμένο στοιχείο από την οποία και έπειτα μπορούσαμε πριν να αναζητήσουμε το στοιχείο y_element.

Επίσης σε αυτό το σημείο το μήκος του LCS όταν χρησιμοποιούμε τον τρέχων χαρακτήρα του Y αυξάνεται κατά ένα.

Με την ολοκλήρωση των εντολών που περιγράψαμε είμαστε πλέον σε θέση να γνωρίζουμε εάν ο συνδυασμός που εξετάζαμε βρισκόταν αποθηκευμένος στην cache ή όχι, απλά εξετάζοντας την τιμή της μεταβλητής is_cached. Αν η τιμή της

μεταβλητής αυτής είναι 0, σημαίνει ότι δεν βρήκαμε καταχώρηση του συνδυασμού αυτού στην cache και άρα πρέπει να τον προσθέσουμε. Την λειτουργία αυτήν την αναλαμβάνει η συνάρτηση cache_update η οποία καλείται με τέσσερις παραμέτρους.

- Η πρώτη παράμετρος είναι η θέση στην οποία εντοπίσαμε στο X τον τρέχων χαρακτήρα του Y. Η θέση αυτή είναι αποθηκευμένη στην μεταβλητή k.
- Η δεύτερη παράμετρος είναι η θέση του τρέχοντος στοιχείο του Y στο Y, που είναι αποθηκευμένη στην μεταβλητή y_pos.
- Η τρίτη παράμετρος είναι το μήκος του LCS που βρέθηκε από αυτόν τον χαρακτήρα και μέχρι το τέλος του Y.
- Η τέταρτη παράμετρος είναι το ίδιο το LCS δηλαδή η ακολουθία των στοιχείων.

Μέχρις αυτό το σημείο της εκτέλεσης του αλγορίθμου έχουμε εξετάσει την περίπτωση που ο τρέχων χαρακτήρας έχει χρησιμοποιηθεί στην δημιουργία του LCS και μας έχει επιστραφεί το αποτέλεσμα αυτής της διαδικασίας. Δηλαδή μας είναι γνωστό το LCS που προκύπτει αν επιχειρήσουμε να προσθέσουμε το τρέχων στοιχείο του Y στην ακολουθία αυτή. Κατά την διάρκεια των υπολογισμών μας εξετάσαμε την περίπτωση που το τρέχων στοιχείο του Y, βρέθηκε στο X και ήταν διαθέσιμο για χρήση, επίσης γνωρίζαμε ότι το στοιχείο αυτό δεν πρέπει να αγνοηθεί. Εξετάσαμε και τις δύο περιπτώσεις όπου για τον συγκεκριμένο συνδυασμό υπήρχε καταχώρηση στην cache καθώς και την περίπτωση στην οποία δεν υπήρχε καταχώρηση στην cache. Στην συνέχεια θα εξετάσουμε την περίπτωση στην οποία ο τρέχων χαρακτήρας του Y θα έπρεπε να αγνοηθεί. Εδώ μπορούμε να καταλήξουμε αν στον έλεγχο που κάναμε στα πρώτα βήματα για την τιμή που υπάρχει στην αντίστοιχη θέση του πίνακα ignoremask αντί της τιμής 0 εντοπίζαμε την τιμή 1. Σε αυτήν την περίπτωση το μήκος του LCS τίθεται -1 για να δείξουμε ότι ο τρέχων χαρακτήρας δεν χρησιμοποιήθηκε στην δημιουργία του LCS.

Έχοντας ολοκληρώσει λοιπόν με τους υπολογισμούς μας κατά τους οποίους επιθυμούμε να συμπεριλάβουμε τον τρέχων χαρακτήρας του Y στην δημιουργία του LCS, πρέπει να ελέγξουμε ποιο είναι το LCS στο οποίο θα καταλήξουμε εάν δεν συμπεριλάβουμε τον τρέχων χαρακτήρα του Y στην δημιουργία του LCS. Ουσιαστικά ο τρέχων χαρακτήρας του Y θα αγνοηθεί. Αυτό σημαίνει ότι πρέπει να πάμε και να ανανεώσουμε την τιμή του πίνακα ignoremask για το συγκεκριμένο στοιχείο, ή πιο συγκεκριμένα για το στοιχείο εκείνο της αλφαβήτου που είναι ίσο με το τρέχων στοιχείο του Y. Η μεταβλητή oldignore αποθηκεύει την παλιά τιμή της αντίστοιχης θέσης στον πίνακα ignoremask. Αν η τιμή αυτή είναι 0, που σημαίνει ότι το στοιχείο αυτό δεν το αγνοούσαμε κατά την εκτέλεση, τώρα γίνεται 0 και αντίστροφα.

Στην συνέχεια καλείται και πάλι αναδρομικά η συνάρτηση find_lcs για να διαπιστώσουμε ποιο είναι το LCS αν δεν χρησιμοποιήσουμε τον τρέχων χαρακτήρα του Y. Οι παράμετροι οι οποίες συνοδεύουν αυτήν την κλίση είναι οι εξής

- Η θέση εκκίνησης στο αλφαριθμητικό X, η οποία δεν έχει αλλάξει αφού δεν έχουμε χρησιμοποιήσει κάποιο στοιχείο από το X και είναι αποθηκευμένη στην μεταβλητή x_pos
- Η θέση εκκίνησης στο Y η οποία εδώ είναι αυξημένη κατά 1, εφόσον αγνοήσαμε το στοιχείο του Y που βρίσκεται στην θέση y_pos για να εντοπίσουμε το LCS χωρίς αυτό το στοιχείο, είναι ουσιαστικά ισοδύναμο με

το να εντοπίσουμε το LCS από αυτό το στοιχείο μέχρι το τέλος χωρίς να συμπεριλάβουμε το στοιχείο αυτό. Επομένως η θέση εκκίνησης στο Y είναι η θέση y_pos + 1.

- Επίσης χρησιμοποιείται ο πίνακας ignoremask που αναφέραμε προηγούμενα.

Τα αποτελέσματα της κλίσης αυτής αποθηκεύονται στις μεταβλητές sz1 (που αποθηκεύει το μήκος του LCS στην περίπτωση που το στοιχείο που βρίσκεται στην θέση y_pos του Y δεν συμπεριληφθεί στο LCS) και lcs1 (για τα στοιχεία που αποτελούν την ακολουθία αυτή). Για να διατηρήσουμε τα στοιχεία του πίνακα ignoremask συνεπή με την προηγούμενη κατάστασή τους, εξετάζουμε την τιμή που έχει το αντίστοιχο στοιχείο του πίνακα για το στοιχείο που βρίσκεται στην θέση y_pos του Y. Ουσιαστικά ελέγχουμε την τιμή της μεταβλητής oldignore με την αντίστοιχη τιμή που βρίσκεται στον πίνακα ignoremask και ανάλογα πραγματοποιούνται οι αλλαγές που απαιτούνται.

Τέλος ο αλγόριθμος αποφασίζει για το ποιο είναι το καλύτερο αποτέλεσμα από τα δύο. Δηλαδή αποφασίζει για το αν η χρησιμοποίηση του στοιχείου του Y, που βρίσκεται στην θέση y_pos στην δημιουργία του LCS επιστρέφει καλύτερο αποτέλεσμα από την μη χρησιμοποίησή του. Για τον έλεγχο αυτό χρησιμοποιούνται οι μεταβλητές sz2 και sz1 που έχουν αποθηκευμένα τα αντίστοιχα μήκη των ακολουθιών. Χρησιμοποιούνται οι μεταβλητές lcs και sz, εκ των οποίων η μεταβλητή lcs αποθηκεύει ποια είναι η μεγαλύτερη κοινή ακολουθία που εντοπίστηκε και η μεταβλητή sz αποθηκεύει το μήκος της. Τέλος χρησιμοποιείται και η μεταβλητή lcssz που έχει τιμή ίση με την τιμή της sz.

4.4. Ψευδοκώδικας

```
find_lcs()
        input: strings X, Y, n, m, x_pos, y_pos, ignoremask
        output: lcs, lcssz
lcs ← []
lcssz ←-1
if (x_pos = n or y_pos = m)
        lcssz ← 0
        return
sz1 ← -1
sz2 ← -1
if (ignoremask(Y(y_pos))=0)
        y_element ←Y(y_pos)
        pos_found ←indexTable_find_first(y_element, x_pos)
        if (pos_fount = -1)
                sz2 ← 0
        else
```

```
                is_cached ← 0
                tocheck ← 1
                k ← indextable_get(y_element, pos_found)
                (sz2, lcs2) ← cache_find(k, y_pos)
                if (sz2 >=0)
                        tocheck ← 0
                        is_cached ←1
                if (tocheck = 1)
                        im ← []
                        pos ← indexTable_get_cur_begin(y_element)
                        indexTable_set_cur_begin(y_element, pos_found+1)
                        (lcsx, sz2) ← find_lcs(k+1, y_pos+1, im)
                        lcs2 ← [k lcsx]
                        indexTable_set_cur_begin(y_element, pos)
                        sz2 ← sz2 + 1
                if (is_cached = 0)
                        cache_update(k, y_pos, sz2, lcs2)
else
                sz2 ← -1
k ← Y(y_pos)
oldignore ← ignoremask(k)
if (oldignore=0)
        ignoremask(k) ← 1
(lcs1, sz1) ← find_lcs(x_pos, y_pos+1, ignoremask)
if (oldignore=0)
        ignoremask(k) ← 0
sz ← 0
if (sz1>0 or sz2>0)
        if (sz1>=sz2)
                lcs ← lcs1
                sz ← sz1
        else
                lcs ← lcs2
                sz ←sz2
lcssz ← sz
```

4.5. Παράδειγμα χρήσης του αλγορίθμου

Έστω ότι τα δύο αλφαριθμητικά που δίνονται είναι το X που έχει μήκος 9 και αποτελείται από τα στοιχεία GGCTACACC και το Y που έχει μήκος 4 και αποτελείται από τα στοιχεία CAAG. Παρακάτω παρουσιάζονται τα αλφαριθμητικά αυτά σε μορφή πίνακα.

Πίνακας 4.6 Το αλφαριθμητικό X

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
X	G	G	C	T	A	C	A	C	C

Πίνακας 4.7 Το αλφαριθμητικό Y

	0	**1**	**2**	**3**
Y	C	A	A	G

Όπως έχουμε ήδη αναφέρει αρχικά διαπιστώνουμε ποιο είναι το αλφάβητο το οποίο χρησιμοποιήθηκε για την δημιουργία των αλφαριθμητικών που δίνονται ως είσοδος στον αλγόριθμο. Το αλφάβητο αυτό αποτελείται από 4 στοιχεία και είναι το S = {A, C, G, T}. Στο επόμενο στάδιο δημιουργείται ο Index Table του X. Με βάση τα στοιχεία που έχουμε ο πίνακας αυτός θα αποτελείται από 4 γραμμές, μια για κάθε στοιχείο του αλφαβήτου, και σε κάθε μια γραμμή θα έχει ως στοιχείο μια λίστα που αντιστοιχεί στις θέσεις που εντοπίσαμε το αντίστοιχο στοιχείο της αλφαβήτου στο X. Για την κατασκευή του πίνακα αυτού εργαζόμαστε ως εξής, διαβάζουμε το πρώτο στοιχείο του αλφαριθμητικού που είναι το G, διαπιστώνουμε με την χρήση του αλφάβητου ότι το στοιχείο αυτό είναι το τρίτο της αλφαβήτου, οπότε στην τρίτη γραμμή του πίνακα και στην πρώτη θέση της λίστας που βρίσκεται εκεί προσθέτουμε ως στοιχείο τον αριθμό 0 που είναι η θέση στην οποία βρίσκεται το πρώτο G στο X. Ακολουθώντας αυτήν την διαδικασία για όλα τα στοιχεία του X, εξετάζοντάς τα σειριακά προκύπτει ο Index Table του συγκεκριμένου X που παρουσιάζεται παρακάτω σε μορφή πίνακα.

Πίνακας 4.8 Ο Index Table του αλφαριθμητικού X

	0	**1**	**2**	**3**
0	4	6		
1	2	5	7	8
2	0	1		
3	3			

Σε αυτό το σημείο γίνεται επίσης αρχικοποίηση της cache καθώς και του πίνακα ignoremask. Ο πίνακα αυτός όπως έχουμε ήδη αναφέρει θα είναι μονοδιάστατος με τόσα στοιχεία όσα και στοιχεία του αλφαβήτου που χρησιμοποιήσαμε. Επομένως στο

συγκεκριμένο παράδειγμα θα αποτελείται από 4 στοιχεία, κάθε ένα από τα οποία αντιστοιχεί σε ένα από γράμματα του αλφάβητου S.

Εφόσον λοιπόν το στάδιο της αρχικοποίησης του αλγορίθμου έχει ολοκληρωθεί, ξεκινάει η εκτέλεση του κυρίως σώματος του αλγορίθμου. Όπως έχουμε αναφέρει, το κυρίως σώμα του αλγορίθμου αποτελείται από την αναδρομική συνάρτηση find_lcs. Επομένως γίνεται η κλίση στην συνάρτηση αυτή με τις εξής παραμέτρους

- Η θέση εκκίνησης για το αλφαριθμητικό X είναι η 0
- Η θέση εκκίνησης για το αλφαριθμητικό Y είναι η 0. Η θέση εκκίνησης είναι και στα δύο αλφαριθμητικά η 0 αφού τώρα ξεκινάει η εκτέλεση του αλγορίθμου και θέλουμε να βρούμε το LCS για ολόκληρα τα X και Y.
- Ο πίνακας ignoremask που αρχικοποιήσαμε προηγούμενα και έχει όλα τα στοιχεία του ίσα με 0. Τα στοιχεία είναι ίσα με 0 διότι στην εκκίνηση της εκτέλεσης του αλγορίθμου δεν επιθυμούμε να αγνοήσουμε κανένα στοιχείο του Y.

Σύμφωνα με την ανάλυση που έγινε σε προηγούμενη ενότητα, κάθε μια από τις κλίσεις της αναδρομικής συνάρτησης έχει δύο κλάδους. Ο ένας κλάδος εξετάζει τα δύο αλφαριθμητικά για την εύρεση του LCS με δεδομένο ότι θα χρησιμοποιηθεί ο τρέχων χαρακτήρας του Y, ενώ ο δεύτερος με δεδομένο ότι ο τρέχων χαρακτήρας του Y δεν θα χρησιμοποιηθεί και άρα αγνοείται. Κάθε ένας κλάδος είναι και μια νέα κλίση στην αναδρομική συνάρτηση.

Στο παράδειγμα που ακολουθεί όπου αναφερόμαστε σε κλίση είναι μια νέα κλίση της αναδρομικής συνάρτησης ενώ σαν βήματα παρουσιάζουμε κάποια από τα σημαντικά βήματα που εκτελεί ο αλγόριθμος σε κάθε κλίση. Η δομή είναι τέτοια ώστε βήματα που εκτελούνται για μια συγκεκριμένη κλίση να έχουν την ίδια στοίχιση.

Κλίση 1 – Η θέση εκκίνησης για το X είναι η 0 και η θέση εκκίνησης για το Y είναι επίσης η 0 (0, 0):

Βήμα 1.1: Όπως αναφέραμε ο πρώτος κλάδος θα εξετάσει ποιο είναι το LCS χρησιμοποιώντας τον τρέχων χαρακτήρα του Y. Ο τρέχων χαρακτήρας του Y, δηλαδή ο χαρακτήρας που βρίσκεται στην θέση 0 είναι ο C. Στο βήμα αυτό ο αλγόριθμος εξετάζει αν ο χαρακτήρας αυτός υπάρχει στο X και αν είναι διαθέσιμος για χρήση. Όντως εξετάζοντας τον Index Table του X βρίσκουμε ότι ο χαρακτήρας αυτός υπάρχει και είναι διαθέσιμος για χρήση. Η θέση στην οποία τον εντοπίσαμε είναι η 2.

Βήμα 1.2: Ο αλγόριθμος εξετάζει αν ο χαρακτήρας που βρίσκεται στην θέση 2 του X και στην θέση 0 του Y, υπάρχει καταχωρημένος στην cache. Η αναζήτησή μας στην cache για αυτόν τον συνδυασμό (2, 0) δεν μας επιστρέφει κάποιο αποτέλεσμα μιας και δεν έχουμε προσθέσει κανένα συνδυασμό μέχρι τώρα στην cache.

Βήμα 1.3: Κλίση 2 – Η θέση εκκίνησης στο X είναι η 2 + 1 και η θέση εκκίνησης στο Y είναι η 0 + 1 (2+1, 0+1):

> **Βήμα 2.1:** Και πάλι όπως στο βήμα 1 της κλίσης 1 της αναδρομικής συνάρτησης εξετάζουμε εάν ο τρέχων χαρακτήρας του Y υπάρχει στο X και είναι διαθέσιμος για χρήση. Ο χαρακτήρας αυτός είναι ο A και εντοπίστηκε στην θέση 4 του X.
>
> **Βήμα 2.2:** Αναζητούμε τον συνδυασμό (4, 1) στην cache για να διαπιστώσουμε αν έχει γίνει κάποια καταχώρηση και να μην χρειαστεί να

συνεχίσουμε τους υπολογισμούς μας. Όμως και πάλι δεν υπάρχει κάποια τέτοια καταχώρηση στην cache.

Βήμα 2.3: Κλίση 3 – Η θέση εκκίνησης στο X είναι η 4+1 και η θέση εκκίνησης στο Y είναι η 1+1 (4+1, 1+1):

Βήμα 3.1: Αναζητούμε τον χαρακτήρα του Y που βρίσκεται στην θέση 2 και είναι ο A στο X. Ο χαρακτήρας αυτός βρίσκεται στο X και είναι διαθέσιμος για χρήση στην θέση 6.

Βήμα 3.2: Ελέγχουμε πριν συνεχίσουμε με τους υπολογισμούς μας στην cache για να δούμε αν υπάρχει καταχώρηση για τον συνδυασμό (6, 2) αλλά δεν υπάρχει στην cache αυτή η καταχώρηση.

Βήμα 3.3: Κλίση 4 – Η θέση εκκίνησης στο X είναι η 6+1 και η θέση εκκίνησης στο Y είναι η 2+1 (6+1, 2+1):

Βήμα 4.1: Θα εξετάσουμε την περίπτωση όπου ο χαρακτήρας που βρίσκεται στην τρέχουσα θέση του Y που είναι το γράμμα G στην θέση 3, συμπεριλαμβάνεται στην εύρεση του LCS. Ο αλγόριθμος εξετάζει αν ο χαρακτήρας αυτός βρίσκεται στο X είναι διαθέσιμος για χρήση. Με δεδομένο ότι αναζητούμε τον χαρακτήρα αυτό από την $7^{η}$ θέση του X και μετά, βλέπουμε ότι ο χαρακτήρας αυτός δεν είναι διαθέσιμος για χρήση και άρα δεν εντοπίζουμε κάποια θέση, οπότε και οι υπολογισμοί μας σταματούν.

Βήμα 4.2: Έχοντας φτάσει σε αυτό το σημείο θα εξετάσουμε την περίπτωση όπου το τρέχων στοιχείο του Y, δεν συμπεριλαμβάνεται στην εύρεση του LCS και άρα αγνοείται. Το τρέχων στοιχείο του Y είναι αυτό που βρίσκεται στην θέση 3 και είναι το G. Σε αυτό το σημείο γίνεται και η ενημέρωση του πίνακα ignoremask. Εφόσον το τρέχων στοιχείο το αγνοούμε προχωράμε μια θέση στο Y.

Βήμα 4.3: Κλίση 5 – Η θέση εκκίνησης στο X είναι η 6+1 και η θέση εκκίνησης στο Y είναι η 3+1:

Βήμα 5.1: Πριν να αρχίσουν οι υπολογισμοί ο αλγόριθμος πάντα εξετάζει μήπως έχουμε φτάσει στο τέλος κάποιου από τα δύο αλφαριθμητικά, όπως στην περίπτωση που εξετάζουμε αυτή τη στιγμή. Εφόσον λοιπόν το Y έχει τερματίσει δεν έχουμε κάποιο υπολογισμό να κάνουμε και άρα επιστρέφουμε στην προηγούμενη κλίση της συνάρτησης.

Βήμα 4.4: Η κλίση 4 της συνάρτησης εξέτασε την περίπτωση που θα χρησιμοποιούσε τον χαρακτήρα της θέσης 3 του Y και κατέληξε ότι δεν υπάρχει χαρακτήρας τέτοιος στο X που θα μπορούσε να χρησιμοποιήσει. Επίσης εξέτασε την περίπτωση να μην χρησιμοποιήσει αυτόν τον χαρακτήρα και απλά να προχωρήσει στον επόμενο, όμως το Y τερμάτισε. Οπότε έχοντας εξετάσει και τις δύο περιπτώσεις έχει καταλήξει στο αποτέλεσμα ότι δεν μπορεί να χρησιμοποιηθεί κάποιος

χαρακτήρας από το σημείο (7, 3) και έπειτα, και επιστρέφει στην προηγούμενη κλίση (κλίση 3) με το αποτέλεσμα 0.

Βήμα 3.4: Η κλίση 3 της συνάρτησης έχει τερματίσει τον κλάδο όπου ελέγχει για το LCS χρησιμοποιώντας τον τρέχων χαρακτήρα του Y που είναι ο A στην θέση 2. Οπότε θα προχωρήσει στην εκτέλεση του άλλου κλάδου, όπου ο χαρακτήρας στη θέση 2 θα αγνοηθεί και θα προχωρήσουμε μια θέση στο Y.

Βήμα 3.5: Κλίση 6 – Η θέση εκκίνησης στο X είναι η 5 και η θέση εκκίνησης στο Y είναι η 2+1 (5, 2+1):

Βήμα 6.1: Εξετάζοντας τον χαρακτήρα που βρίσκεται στην $3^{η}$ θέση του Y, καταλήγουμε ότι δεν υπάρχει στο X σε θέση που μπορεί να χρησιμοποιηθεί (μεγαλύτερη του 5), οπότε οι υπολογισμοί μας για την εύρεση του LCS με αυτόν τον χαρακτήρα σταματούν.

Βήμα 6.2: Επομένως θα εξετάσουμε ποιο είναι το LCS χωρίς την χρήση αυτού του χαρακτήρα του Y, δηλαδή θα τον αγνοήσουμε και θα προχωρήσουμε μια θέση στο Y ενημερώνοντας παράλληλα και τον πίνακα ignoremask.

Βήμα 6.3: Κλίση 7 – Η θέση εκκίνησης στο X είναι η 5 και η θέση εκκίνησης στο Y είναι η 3+1 (5, 3+1):

Βήμα 7.1: Έχουμε όμως ξεπεράσει τον αριθμό των στοιχείων του Y, οπότε επιστρέφουμε στην προηγούμενη κλίση της συνάρτησης.

Βήμα 6.4: Έχουμε ολοκληρώσει με τους υπολογισμούς μας για αυτήν την κλίση της αναδρομικής συνάρτησης, οπότε επιστρέφουμε στην προηγούμενη κλίση με το αποτέλεσμα 0.

Βήμα 3.6: Σε αυτό το σημείο η κλίση 3 της συνάρτησης έχει ολοκληρώσει με τους υπολογισμούς της και για τους δύο κλάδους, με και χωρίς το τρέχων στοιχείο του Y που είναι το στοιχείο της θέσης 2. Ολοκληρώσαμε άρα τους υπολογισμούς μας για τον συνδυασμό στοιχείων που βρίσκονται στις θέσεις 6 για το X και 2 για το Y. Το αποτέλεσμα ήταν ότι από το στοιχείο αυτό και μέχρι το τέλος του LCS έχει μήκος 1 και αποτελείται από τον χαρακτήρα A. Το αποτέλεσμα αυτό το προσθέτουμε στην cache για μετέπειτα χρήση.

Βήμα 2.4: Όπως και στο βήμα 3.6 έχουμε τελειώσει με τους υπολογισμούς για το συνδυασμό (4, 1), ο οποίος μας επέστρεψε ως αποτέλεσμα ένα LCS μήκους 2 που αποτελείται από τα στοιχεία AA. Το αποτέλεσμα επίσης προστίθεται στην cache.

Βήμα 2.5: Όπως και στο βήμα 4.2 έχουμε τελειώσει με τον κλάδο αυτής της κλίσης ο οποίος εξέταζε ποιο είναι το καλύτερο LCS που θα μπορούσαμε να έχουμε χρησιμοποιώντας τον τρέχων χαρακτήρα του Y που είναι ο A στην θέση 1 του Y. Οπότε θα εξετάσουμε στην συνέχεια τον κλάδο όπου αυτός ο χαρακτήρας δεν χρησιμοποιείται, προχωρώντας μια θέση στο Y και αγνοώντας τον τρέχων χαρακτήρα ενημερώνοντας τον πίνακα ignoremask.

Βήμα 2.6: Κλίση 8 – Η θέση εκκίνησης στο X είναι η 3 και η θέση εκκίνησης στο Y είναι η 1+1 (3, 1+1):

Βήμα 8.1: Σε αυτό το βήμα θα έπρεπε να εξετάσουμε κανονικά τον κλάδο εκείνο στον οποίο θα χρησιμοποιήσουμε τον τρέχων χαρακτήρα του Y στην δημιουργία του LCS. Ο τρέχων χαρακτήρας του Y είναι αυτός που βρίσκεται στην θέση 2 και είναι ο A. Εξετάζοντας τον πίνακα ignoremask βλέπουμε όμως ότι στο στοιχείο αυτό αντιστοιχεί η τιμή 1 που σημαίνει ότι θα πρέπει να αγνοηθεί. Αυτό προκύπτει από το προηγούμενο βήμα στο οποίο εξετάζαμε την περίπτωση όπου το A (στη θέση 1) δεν συμπεριλαμβάνεται στην εύρεση του LCS. Οπότε οι υπολογισμοί μας για αυτόν τον κλάδο σταματούν.

Βήμα 8.2: Όπως έχουμε δει και σε προηγούμενα βήματα, θα εξετάσουμε τον κλάδο όπου ο τρέχων χαρακτήρας αγνοείται και προχωράμε μια θέση στο Y.

Βήμα 8.3: Κλίση 9 – Η θέση εκκίνησης στο X είναι η 3 και η θέση εκκίνησης στο Y είναι η 2+1 (3, 2+1):

Βήμα 9.1: Ο χαρακτήρας στην θέση 3 του Y είναι ο G, ο οποίος δεν μπορεί να βρεθεί σε τέτοια θέση στο X ώστε να μπορούμε να τον χρησιμοποιήσουμε, οπότε οι υπολογισμοί μας σταματούν για αυτόν τον κλάδο.

Βήμα 9.2: Στην συνέχεια θα εξεταστεί ο κλάδος χωρίς τον τρέχων χαρακτήρα του Y, τον οποίο αγνοούμε ενημερώνοντας τον πίνακα ignoremask και προχωρούμε μια θέση στο Y.

Βήμα 9.3: Κλίση 10 – Η θέση εκκίνησης στο X είναι η 3 και η θέση εκκίνησης στο Y είναι η 3+1 (3, 3+1):

Βήμα 10.1: Το Y έχει φτάσει στο τέλος του οπότε οι υπολογισμοί μας σταματούν και επιστρέφουμε στην προηγούμενη κλίση της συνάρτησης.

Βήμα 9.4: Οι υπολογισμοί μας για την κλίση 9 της αναδρομικής συνάρτησης έχουν ολοκληρωθεί και επιστρέφουμε στην προηγούμενη κλίση με το αποτέλεσμα 0.

Βήμα 8.4: Επίσης έχουν τελειώσει και οι υπολογισμοί μας για τους δύο κλάδους αυτής της κλίσης και πάλι επιστρέφουμε στην προηγούμενη κλίση με το αποτέλεσμα 0.

Βήμα 2.7: Έχουν τελειώσει και οι υπολογισμοί μας για αυτήν την κλίση της συνάρτησης και επιστρέφουμε στην προηγούμενη κλίση με το αποτέλεσμα ότι το καλύτερο LCS που έχουμε βρει μέχρι αυτό το σημείο είναι το AA και έχει μήκος 2.

Βήμα 1.4: Σε αυτό το σημείο έχουμε ολοκληρώσει με τον κλάδο εκείνο που περιλαμβάνει το τρέχων στοιχείο του Y που είναι το C στην θέση 1, και με την ολοκλήρωση της κλίσης 2, γνωρίζουμε ότι το LCS με αυτό το χαρακτήρα έχει μήκος 3 και είναι το CAA. Οπότε προσθέτουμε στην cache αυτόν τον συνδυασμό.

Βήμα 1.5: Στην συνέχεια εξετάζουμε τον κλάδο όπου το τρέχων στοιχείο του Y δεν χρησιμοποιείται οπότε ενημερώνουμε τον πίνακα ignoremask και προχωράμε μια θέση στο Y.

Βήμα 1.6: Κλίση 11 – Η θέση εκκίνησης στο X είναι η 0 και η θέση εκκίνησης στο Y είναι η 0+1 (0, 0+1):

Βήμα 11.1: Το τρέχων στοιχείο του Y είναι το A που είναι διαθέσιμο για χρήση στο X στην θέση 4.

Βήμα 11.2: Ο αλγόριθμος εξετάζει αν αυτός ο συνδυασμός υπάρχει στην cache. Όντως υπάρχει τέτοια καταχώρηση στην cache οπότε παίρνουμε το αποθηκευμένο αποτέλεσμα που είναι LCS μήκος 2 και στοιχεία τα AA και τερματίζουμε τους υπολογισμούς μας για αυτόν τον κλάδο.

Βήμα 11.3: Θα εξετάσουμε τον κλάδο που δεν περιλαμβάνει το τρέχων στοιχείο του Y που είναι το A, οπότε ενημερώνεται ο πίνακας ignoremask και προχωρούμε μια θέση στο Y.

Βήμα 11.4: Κλίση 12 – Η θέση εκκίνησης στο X είναι η 0 και η θέση εκκίνησης στο Y είναι η 1+1 (0, 1+1):

Βήμα 12.1: Σε αυτό το βήμα βρισκόμαστε πάλι στην περίπτωση όπου το τρέχων στοιχείο του Y θα αγνοηθεί οπότε δεν συνεχίζουμε με τους υπολογισμούς μας.

Βήμα 12.2: Θα εξετάσουμε τον κλάδο όπου δεν θα συμπεριλάβουμε το τρέχων στοιχείο και άρα προχωράμε μια θέση στο Y.

Βήμα 12.3: Κλίση 13 – Η θέση εκκίνησης στο X είναι η 0 και η θέση εκκίνησης στο Y είναι η 2+1 (0, 2+1):

Βήμα 13.1: Το τρέχων στοιχείο του Y είναι το G το οποίο το εντοπίσαμε στο X στην θέση 0 και είναι διαθέσιμο για χρήση.

Βήμα 13.2: Εξετάζοντας την cache για το αν υπάρχει καταχώρηση αυτού του συνδυασμού δεν παίρνουμε κάποιο αποτέλεσμα, οπότε συνεχίζουμε με τους υπολογισμούς μας.

Βήμα 13.3: Κλίση 14 – Η θέση εκκίνησης στο X είναι η 0+1 και η θέση εκκίνησης στο Y είναι η 3+1 (0+1, 3+1):

Βήμα 14.1: Έχουμε φτάσει στο τέλος του Y οπότε δεν συνεχίζουμε με τους υπολογισμούς μας και επιστρέφουμε στην προηγούμενη κλίση.

Βήμα 13.4: Θα εξετάσουμε τον κλάδο όπου δεν θα συμπεριλάβουμε το τρέχων στοιχείο του Y, οπότε ενημερώνουμε τον πίνακα ignoremask και προχωρούμε μια θέση στο Y.

Βήμα 13.5: Κλίση 15 – Η θέση εκκίνησης στο X είναι η 0 και η θέση εκκίνησης στο Y είναι η 3+1 (0, 3+1):

Βήμα 15.1: Και πάλι έχουμε ήδη φτάσει στο τέλος στο Y οπότε επιστρέφουμε στην προηγούμενη κλίση.

Βήμα 13.6: Έχοντας τελειώσει τους υπολογισμούς μας για αυτήν την κλίση γνωρίζουμε ότι ο συνδυασμός (0, 3) μας επιστρέφει LCS μήκους 1 που είναι το G. Την πληροφορία αυτήν την καταχωρούμε στην cache.

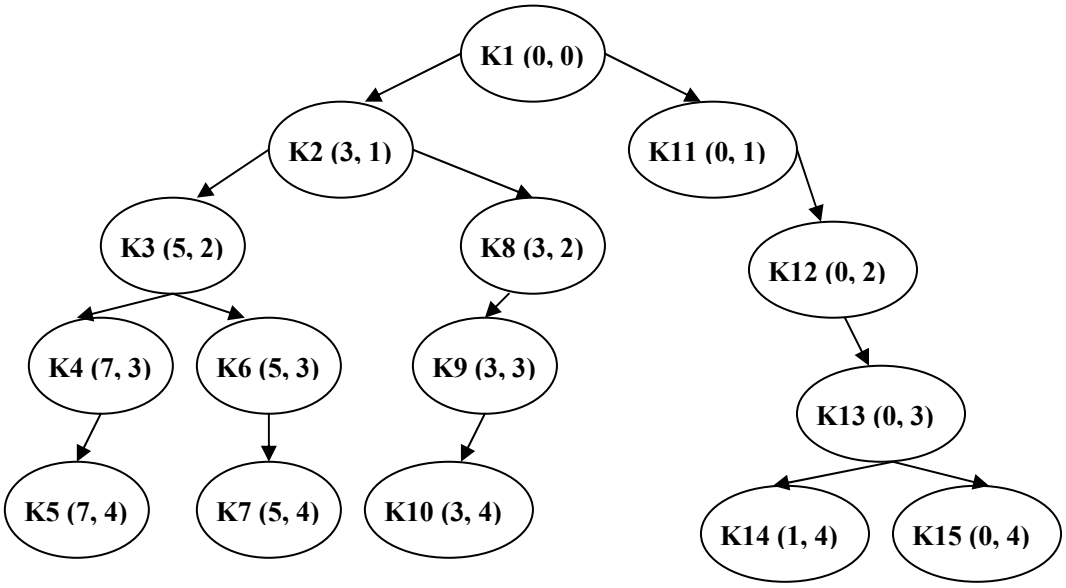
Βήμα 12.4: Έχουμε τελειώσει τους υπολογισμούς μας για αυτήν την κλίση και επιστρέφεται στην προηγούμενη κλίση ότι το αποτέλεσμα είναι ένα LCS μήκους 1 που είναι το G.

Βήμα 11.5: Έχουμε ολοκληρώσει τους υπολογισμούς μας για αυτήν την κλίση της συνάρτησης και επιστρέφουμε στην προηγούμενη κλίση ότι το LCS έχει μήκος 2 και είναι το AA.

Βήμα 1.4: Έχουμε ολοκληρώσει με την κλίση αυτής της συνάρτησης και το αποτέλεσμα που θα επιστρέψει τελικά ο αλγόριθμος είναι ότι το LCS των X, Y έχει μήκος 3 και είναι το CAA.

Στο διάγραμμα που ακολουθεί παρουσιάζονται οι κλίσεις που έγιναν στην αναδρομική συνάρτηση. Μέσα στην παρένθεση παρουσιάζονται οι θέσεις εκκίνησης στο X και στο Y αντίστοιχα.

Διάγραμμα 4.3 Οι κλίσεις της αναδρομικής συνάρτησης



ΚΕΦΑΛΑΙΟ 5

ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ

5.1 Εισαγωγή

Προκειμένου να επαληθεύσουμε την απόδοση του αλγορίθμου σε πραγματικές συνθήκες πραγματοποιήσαμε μια σειρά από πειράματα για να εξετάσουμε την απόδοσή του σε ότι αφορά τον χρόνο εκτέλεσης και την κατανάλωση μνήμης. Τα αποτελέσματα των πειραμάτων αυτών παρουσιάζονται στο κεφάλαιο αυτό. Επίσης πραγματοποιείται και μια σύγκριση του DFS – LCS με τον κλασικό αλγόριθμο για την επίλυση του προβλήματος που είναι ο αλγόριθμος των Hunt και Szymanski.

5.2 Ο υπολογιστής και η γλώσσα προγραμματισμού

Ο υπολογιστής ο οποίος χρησιμοποιήθηκε για την διεξαγωγή των πειραμάτων έχει τα εξής χαρακτηριστικά

- Είναι Intel Celeron με επεξεργαστή στα 2.4 GHz
- Έχει 768 MB μνήμης RAM και δυνατότητα για 2.048 GB μνήμης swap (δηλαδή σε περίπτωση που χρησιμοποιηθεί ολόκληρη η μνήμη RAM, υπάρχει δυνατότητα να χρησιμοποιηθούν 2.048 GB του σκληρού δίσκου σαν μνήμη)
- Το λειτουργικό σύστημα που είναι εγκατεστημένο στον υπολογιστή είναι το DEBIAN GNU / LINUX με πυρήνα 2.6.8 – 2 – 686

Σε ότι αφορά τον προγραμματισμό του αλγορίθμου αυτός προγραμματίστηκε σε C/C++ ενώ ο κλασικός LCS προγραμματίστηκε σε C. Ο compiler που χρησιμοποιήθηκε ήταν ο 3.3.5 (DEBIAN 1:3.3.3 – 13).

Προκειμένου να εξετάσουμε τις επιδόσεις τους σε κάποια παραδείγματα, επιλέξαμε τα παραδείγματα αυτά να δημιουργηθούν τυχαία. Για τον σκοπό αυτό χρησιμοποιήθηκε η συνάρτηση Random του λειτουργικού συστήματος. Η συνάρτηση χρησιμοποιεί μια μη γραμμική γεννήτρια τυχαίων αριθμών, που χρησιμοποιεί έναν πίνακα για να επιστρέψει ψευδο – τυχαίους αριθμούς στο διάστημα από 0 μέχρι RAND_MAX. Δημιουργήσαμε ένα σύνολο από 33 πειράματα, κάθε ένα από τα οποία έχει ένα αλφαριθμητικό που το ονομάζουμε X(με μήκος n) και ένα αλφαριθμητικό που το ονομάζουμε Y(με μήκος m, ενώ για τα μήκη των δύο αλφαριθμητικών ισχύει ότι $n > m$).

Για την δημιουργία τους χρησιμοποιήθηκε ένα σύνολο 64 στοιχείων, από το οποίο το πρόγραμμα με την βοήθεια της συνάρτησης Random επιλέγει τυχαία τα στοιχεία που θα αποτελούν το κάθε αλφαριθμητικό. Σαν παράμετροι στην δημιουργία του κάθε αλφαριθμητικού δίνονται το μήκος που επιθυμούμε να έχει το κάθε αλφαριθμητικό καθώς και το πλήθος των στοιχείων της αλφαβήτου που θα χρησιμοποιηθεί για την δημιουργία του.

Για παράδειγμα αν θέλουμε το αλφαριθμητικό X να αποτελείται από 1000 στοιχεία και το μέγεθος της αλφαβήτου από το οποίο θα κατασκευαστεί είναι 8, τότε το πρόγραμμα ανατρέχει στον πίνακα με τα 64 στοιχεία και με την συνάρτηση που

αναφέραμε επιλέγει μια ακολουθία που να αποτελείται από τα πρώτα 8 στοιχεία του πίνακα και να έχει μήκος 1000. Αντίστοιχα αν για το ίδιο πείραμα επιθυμούμε να δημιουργήσουμε ένα αλφαριθμητικό με 300 στοιχεία και το αλφάβητο από το οποίο το δημιουργήσαμε να έχει μήκος 4, τότε το πρόγραμμα θα χρησιμοποιήσει τα πρώτα 4 στοιχεία του πίνακα και θα παράγει μια ακολουθία από 300 στοιχεία. Με βάση λοιπόν τα παραπάνω, ουσιαστικά το αλφάβητο που χρησιμοποιήσαμε για την κατασκευή των δύο αλφαριθμητικών έχει μήκος που δίνεται από τον τύπο max{μήκος του αλφάβητου του X, μήκος του αλφάβητου του Y}.

Ο λόγος που κάθε φορά επιλέγουμε τα πρώτα στοιχεία του πίνακα με τα 64 στοιχεία που μπορούμε να χρησιμοποιήσουμε ως αλφάβητο, είναι για να διασφαλίσουμε ότι τα δύο αλφαριθμητικά που θα δημιουργήσουμε για το συγκεκριμένο πείραμα θα έχουν κάποια κοινά στοιχεία για να μπορέσουμε να εξετάσουμε την μεγαλύτερη κοινή ακολουθία τους.

Επίσης αυτό που πρέπει να αναφέρουμε είναι ότι και οι δύο αλγόριθμοι είναι προγραμματισμένοι με τέτοιο τρόπο έτσι ώστε την αρχική είσοδο των αλφαριθμητικών προς εξέταση να την δέχονται από αρχεία. Δηλαδή τα αλφαριθμητικά X και Y τα δημιουργούμε και αποθηκεύονται σε αρχεία κειμένου, τα οποία εισάγονται στους αλγορίθμους. Η τεχνική αυτή επιλέχθηκε λόγω του μεγάλου μεγέθους που έχουν σε αρκετά παραδείγματα τα δύο αλφαριθμητικά.

5.3 Πειράματα με κριτήριο τον χρόνο εκτέλεσης

Στην ενότητα αυτή θα εξετάσουμε την απόδοση των δύο αλγορίθμων με κριτήριο τον χρόνο εκτέλεσης τους. Εξετάζουμε τον χρόνο δηλαδή που χρειάστηκε ο κάθε ένας για να επιστρέψει το επιθυμητό αποτέλεσμα, δηλαδή το μέγεθος της μεγαλύτερης κοινής ακολουθίας αλλά και την ίδια την ακολουθία.

Στον πίνακα που ακολουθεί παρουσιάζουμε αναλυτικά τα δεδομένα που χρησιμοποιήθηκαν για την διεξαγωγή κάθε πειράματος καθώς και τα αποτελέσματα τους από άποψη χρόνου. Οι στήλες του πίνακα αντιπροσωπεύουν (με αναφορά από τα αριστερά προς τα δεξιά)

- Η πρώτη στήλη αναφέρει το διάγραμμα στο οποίο παρουσιάζονται τα αποτελέσματα των αντίστοιχων γραμμών του πίνακα
- Η δεύτερη στήλη παρουσιάζει το μήκος του αλφαριθμητικού X, δηλαδή το πλήθος των στοιχείων που το αποτελούν
- Η τρίτη στήλη παρουσιάζει το μήκος του αλφαριθμητικού Y, δηλαδή το πλήθος των στοιχείων που το αποτελούν
- Η τέταρτη στήλη παρουσιάζει το πλήθος των στοιχείων της αλφαβήτου που χρησιμοποιήθηκε για την κατασκευή του αλφαριθμητικού X
- Η πέμπτη στήλη παρουσιάζει το πλήθος των στοιχείων της αλφαβήτου που χρησιμοποιήθηκε για την κατασκευή του αλφαριθμητικού Y.
- Οι δύο επόμενες στήλες, η έκτη και η έβδομη παρουσιάζουν τους χρόνους εκτέλεσης DFS – LCS. Οι χρόνοι που παρουσιάζονται είναι δύο
    - Ο ένας είναι ο USER που μετριέται σε seconds. Ο χρόνος αυτός είναι ο πραγματικός χρόνος που χρειάστηκε η συγκεκριμένη εφαρμογή,

δηλαδή το πρόγραμμα που υλοποιεί τον αλγόριθμο, για να ολοκληρώσει την λειτουργία του.

- Ο άλλος είναι ο SYSTEM που μετριέται επίσης σε seconds. Ο χρόνος αυτός είναι ο χρόνος που καταναλώθηκε από τον αλγόριθμο και έχει να κάνει με την χρησιμοποίηση του λειτουργικού συστήματος. Αυτό που πρέπει να αναφέρουμε είναι ότι ο CPU χρόνος που χρειάστηκε συνολικά ο αλγόριθμος για να εκτελεστεί στον συγκεκριμένο υπολογιστή και στο συγκεκριμένο λειτουργικό σύστημα είναι το άθροισμα των δύο χρόνων που παρουσιάσαμε παραπάνω.

- Οι δύο επόμενες στήλες, η όγδοη και η ένατη, αναφέρονται στον κλασικό αλγόριθμο LCS και παρουσιάζουν αντίστοιχα τον USER και SYSTEM χρόνο που χρειάστηκε ο συγκεκριμένος αλγόριθμος για εκτελεστεί.

Όταν κάποιος από τους δύο αλγορίθμους χρειάστηκε κατά την διάρκεια της εκτέλεσής του περισσότερους πόρους από αυτούς που μπορούσε να διαθέσει το λειτουργικό σύστημα, τότε έχουμε βίαιη παύση της εκτέλεσης του συγκεκριμένου προγράμματος από το λειτουργικό σύστημα. Αυτό αναφέρεται στον πίνακα με την λέξη killed που σημαίνει ότι ο αλγόριθμος δεν ολοκλήρωσε τους υπολογισμούς του, αλλά τερματίστηκε από το λειτουργικό σύστημα.

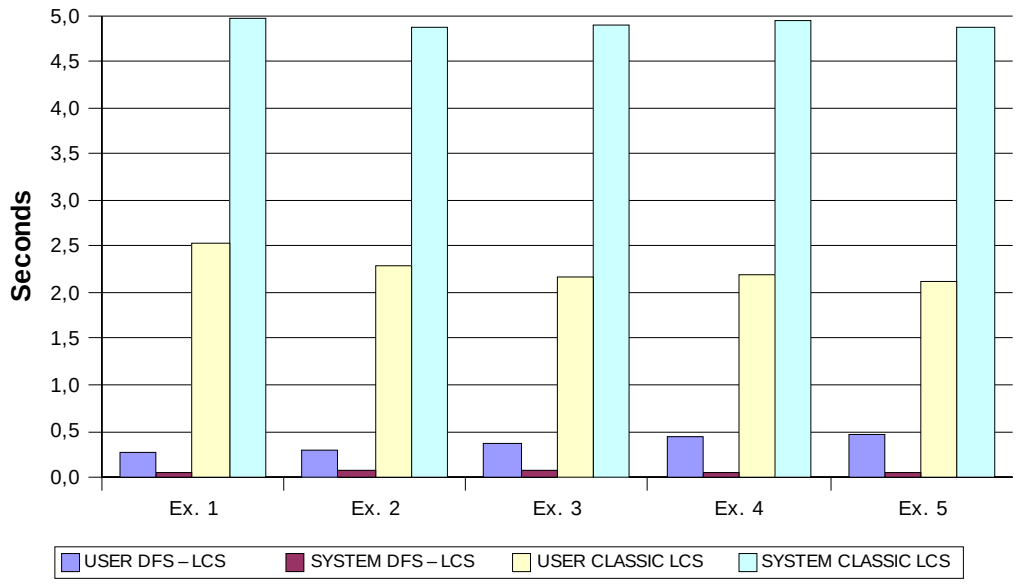
Πίνακας 5.1 Τα αποτελέσματα των πειραμάτων που πραγματοποιήθηκαν για την εξέταση των χρόνων ολοκλήρωσης της εκτέλεσης των δύο αλγορίθμων.

			Μέγεθος της αλφαβήτου		**DFS - LCS**		**Classic LCS**	
	Μήκος του X	**Μήκος του Y**	**X**	**Y**	**User (seconds)**	**System (seconds)**	**User (seconds)**	**System (seconds)**
Διάγραμμα 1	1.000.000	100	4	4	0,268	0,055	2,530	4,962
	1.000.000	100	8	8	0,298	0,072	2,287	4,883
	1.000.000	100	16	16	0,367	0,070	2,176	4,904
	1.000.000	100	24	24	0,430	0,060	2,205	4,950
	1.000.000	100	36	36	0,457	0,058	2,128	4,868
	1.000.000	200	4	8	0,307	0,053	5,357	12,858
	1.000.000	200	4	16	0,227	0,058	4,782	12,255
	1.000.000	200	8	24	0,319	0,065	5,033	12,909
	1.000.000	200	8	36	0,247	0,072	4,873	12,505
Διάγραμμα 2	10.000.000	100	4	4	2,210	0,656	Killed	
	10.000.000	200	4	4	2,423	0,723	Killed	
	10.000.000	300	4	4	2,949	0,683	Killed	
	10.000.000	400	4	4	4,229	0,845	Killed	
	10.000.000	500	4	4	6,031	0,967	Killed	
	10.000.000	600	4	4	9,364	1,147	Killed	

	10.000.000	700	4	4	12,342	1,406	Killed	
	10.000.000	800	4	4	17,917	1,705	Killed	
	10.000.000	900	4	4	27,581	2,446	Killed	
	10.000.000	1000	4	4	35,741	4,324	Killed	
	10.000.000	1100	4	4	45,476	4,616	Killed	
	10.000.000	1200	4	4	63,460	5,414	Killed	
	10.000.000	1300	4	4	78,130	7,106	Killed	
	10.000.000	1400	4	4	106,083	8,756	Killed	
Διαγράμματα 3 – 4	3.000	100	4	4	0,055	0,006	0,008	0,004
	3.000	200	4	4	0,282	0,017	0,013	0,007
	3.000	400	4	4	2,051	0,138	0,024	0,016
	3.000	800	4	4	16,633	1,074	0,042	0,033
	3.000	1200	4	4	62,488	3,696	0,068	0,047
	3.000	1600	4	4	Killed		0,092	0,062

Στην συνέχεια παρουσιάζονται με μορφή διαγραμμάτων τα αποτελέσματα από τα πειράματα που πραγματοποιήσαμε.

Διάγραμμα 5.1 Χρόνοι εκτέλεσης για μεγάλο Χ και μικρό Υ. Τα Χ, Υ έχουν δημιουργηθεί χρησιμοποιώντας το ίδιο ακριβώς αλφάβητο που αυξάνει από παράδειγμα σε παράδειγμα.



Στο διάγραμμα αυτό στον οριζόντιο άξονα παρουσιάζονται τα 5 πειράματα που πραγματοποιήσαμε, θέλοντας να εξετάσουμε την απόδοση των δύο αλγορίθμων όταν χρησιμοποιείται ένα μεγάλο Χ και ένα μικρό Υ. Για κάθε παράδειγμα έχουμε 4 στήλες, οι 2 πρώτες αντιστοιχούν στον DFS – LCS και οι 2 επόμενες στον κλασικό

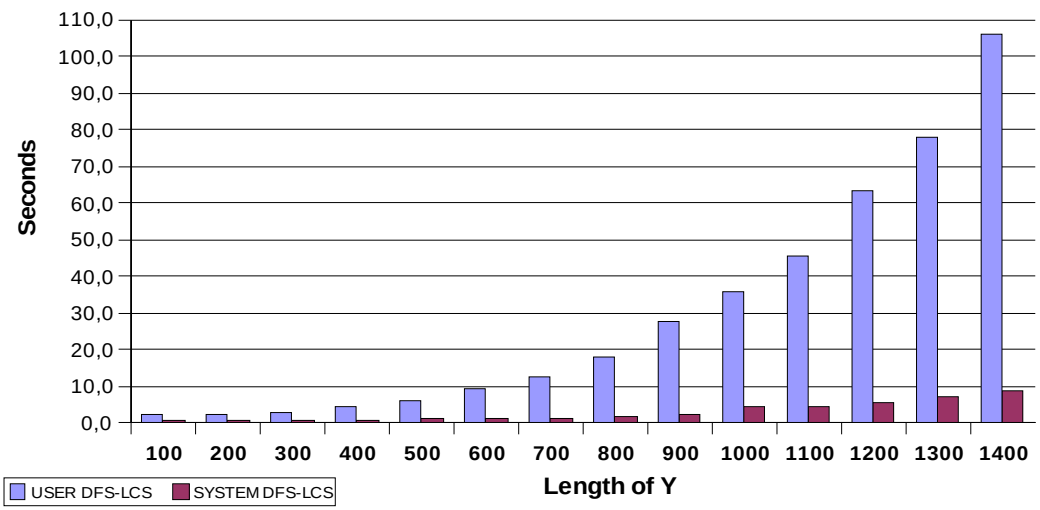
LCS. Στον κατακόρυφο άξονα παρουσιάζονται οι χρόνοι για κάθε μια στήλη με μονάδα μέτρησης το second.

Όπως φαίνεται και στο διάγραμμα ο DFS – LCS αλγόριθμος έχει πολύ μικρότερους χρόνους εκτέλεσης. Αυτό οφείλεται στο γεγονός ότι έχει πολύ μικρότερες απαιτήσεις σε μνήμη. Αντίθετα οι απαιτήσεις σε χρόνο του κλασικού LCS αλγορίθμου είναι πολύ μεγαλύτερες λόγω του μεγάλου όγκου μνήμης που απαιτεί για τους υπολογισμούς του. Λόγω του τρόπου λειτουργίας του ο κλασικός LCS προκειμένου να καταλήξει στο επιθυμητό αποτέλεσμα θα κατασκευάσει έναν πίνακα με διαστάσεις όσα τα μήκη των δύο αλφαριθμητικών. Στην περίπτωσή μας δηλαδή 1000000 x 100, όσο τα μήκη των X και Y αντίστοιχα. Κάτι τέτοιο φυσικά, έχει ως αποτέλεσμα η μνήμη RAM του υπολογιστή να μην επαρκεί και να προκαλείται swapping στον σκληρό δίσκο του υπολογιστή. Το swapping του σκληρού δίσκου επιτρέπει φυσικά την παροχή επιπρόσθετης μνήμης για την εκτέλεση του αλγορίθμου, αλλά με κόστος στον χρόνο εκτέλεσης διότι η ανάγνωση και εγγραφή δεδομένων από και προς τον σκληρό δίσκο του υπολογιστή είναι μια εξαιρετικά αργή διαδικασία σε σχέση με τις αντίστοιχες διαδικασίες της μνήμης RAM.

Για παράδειγμα στο δεύτερο πείραμα που πραγματοποιήσαμε, όπου το X έχει μήκος 1000000, το Y έχει μήκος 100 και το αλφάβητο που χρησιμοποιήθηκε έχει μήκος 8, ο CPU χρόνος για τον DFS – LCS είναι 0.323 seconds ενώ ο χρόνος του κλασικού LCS αλγορίθμου είναι 7.17 seconds. Με άλλα λόγια ο χρόνος που χρειάστηκε ο κλασικός αλγόριθμος LCS είναι 22.2 φορές μεγαλύτερος από τον χρόνο που χρειάστηκε ο DFS – LCS αλγόριθμος για την επίλυση του συγκεκριμένου προβλήματος.

Στο πείραμα που ακολουθεί εξετάζουμε τις απαιτήσεις των δύο αλγορίθμων σε χρόνο, όταν το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας περιλαμβάνει ένα αλφαριθμητικό X που αποτελείται από 10000000 στοιχεία. Το μήκος του Y στα πειράματα αυτά ξεκινά από 100 (Y αποτελούμενο από 100 στοιχεία) και αυξάνει κατά 100 σε κάθε παράδειγμα. Για την δημιουργία και των δύο αλφαριθμητικών έχει χρησιμοποιηθεί ένα αλφάβητο με 4 στοιχεία, το οποίο είναι σταθερό σε όλα τα παραδείγματα.

Διάγραμμα 5.2 Χρόνοι εκτέλεσης για μεγάλο X και μικρό Y. Τα X και Y έχουν δημιουργηθεί με το ίδιο αλφάβητο και το μήκος του Y αυξάνει κατά 100 σε κάθε πείραμα.



Στο συγκεκριμένο διάγραμμα δεν εμφανίζονται τα αποτελέσματα για τον κλασικό αλγόριθμο του LCS και αυτό γιατί λόγω των μεγάλων απαιτήσεων του σε μνήμη χρησιμοποίησε όλη τη μνήμη RAM καθώς και όλο το κομμάτι του σκληρού δίσκου που μπορούσε να κάνει swap, χωρίς όμως να ολοκληρώσει τους υπολογισμούς του. Αυτό όπως έχουμε ήδη αναφέρει είχε ως αποτέλεσμα τον τερματισμό του συγκεκριμένου προγράμματος από το λειτουργικό σύστημα. Για αυτόν τον λόγο το διάγραμμα παρουσιάζει μόνο τους χρόνους του DFS – LCS αλγορίθμου.

Στον οριζόντιο άξονα παρουσιάζεται το μήκος του Y που αυξάνεται σε κάθε παράδειγμα για να δείξουμε πώς αυξάνεται και ο χρόνος εκτέλεσης του συγκεκριμένου αλγορίθμου. Αυτή η αύξηση στον χρόνο εκτέλεσης οφείλεται στην αναδρομική συνάρτηση που χρησιμοποιεί ο αλγόριθμος για την διεκπεραίωση των υπολογισμών του. Λόγω της αύξησης του μήκους του Y, η αναδρομική συνάρτηση φτάνει σε μεγάλο βάθος, οπότε και παρουσιάζεται αύξηση του χρόνου εκτέλεσης.

Πρέπει να σημειωθεί ότι κατά την εκτέλεση του DFS – LCS, στον συγκεκριμένο υπολογιστή, αν το μέγεθος του Y υπερβεί τα 1500 στοιχεία ο αλγόριθμος δεν μπορεί να ολοκληρώσει τους υπολογισμούς του, γιατί εξαρτάται όπως είπαμε από το μήκος του Y. Βέβαια κάτι τέτοιο ισχύει κυρίως σε περιπτώσεις όπου το αλφάβητο από το οποίο έχουμε κατασκευάσει τα δύο αλφαριθμητικά είναι το ίδιο, όπως στα παραδείγματά μας είναι 4.

Ο συγκεκριμένος αλγόριθμος λόγω του τρόπου που λειτουργεί λαμβάνει υπόψιν του το αλφάβητο που χρησιμοποιούμε. Για παράδειγμα αν κάποια από τα στοιχεία του Y δεν υπάρχουν καθόλου στο X (λόγω του ότι το Y έχει δημιουργηθεί από αλφάβητο που περιλαμβάνει περισσότερα στοιχεία από ότι το αλφάβητο από το οποίο έχει δημιουργηθεί το X), τότε ο αλγόριθμος απαιτεί λιγότερο χρόνο για να εκτελεστεί. Αυτό οφείλεται στο γεγονός ότι γίνονται πολύ λιγότερες κλίσεις της αναδρομικής συνάρτησης κατά την εκτέλεση του αλγορίθμου. Κάτι τέτοιο βέβαια δεν ισχύει για τους χρόνους εκτέλεσης του κλασικού αλγορίθμου του LCS ο οποίος δεν λαμβάνει υπόψιν του τις διαφορές στα αλφάβητα των δύο αλφαριθμητικών και οι απαιτήσεις του εξαρτώνται μονάχα από τα μήκη των αλφαριθμητικών και όχι από τα στοιχεία τους.

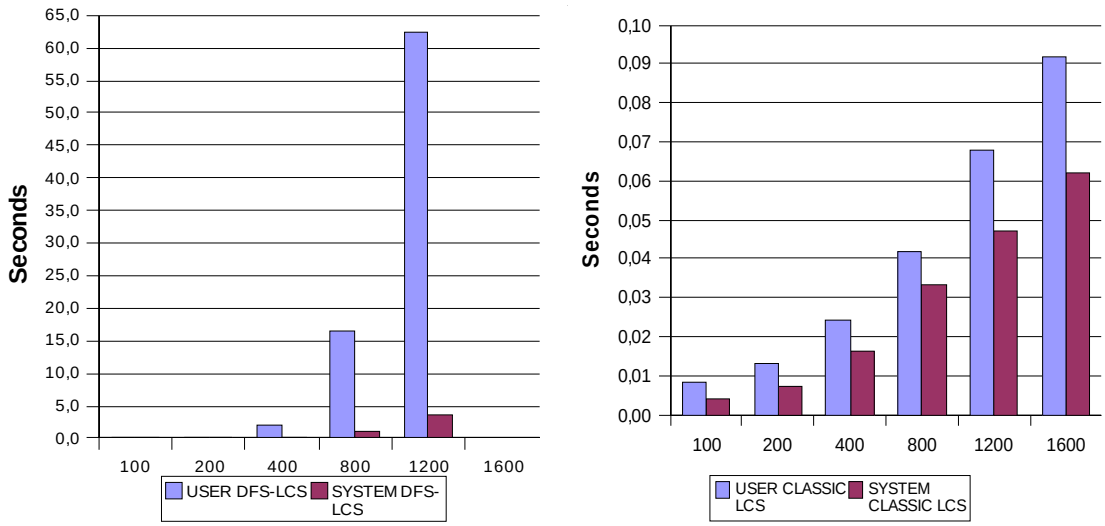
Με βάση λοιπόν τα παραπάνω έχουμε να παρατηρήσουμε ότι σε περιπτώσεις όπου το αλφάβητο που χρησιμοποιούμε για το Y είναι μεγαλύτερο από αυτό που χρησιμοποιούμε για το X, τότε το μέγεθος του Y μπορεί να αυξηθεί σημαντικά.

Στο επόμενο πείραμά μας εξετάζουμε τις απαιτήσεις που έχουν οι δύο αλγόριθμοι σε χρόνο όταν το αλφαριθμητικό X που χρησιμοποιούμε έχει μήκος 3000, ενώ το μήκος του Y αυξάνεται και πάλι σε κάθε παράδειγμα κατά 100. Και εδώ το αλφάβητο από το οποίο δημιουργήσαμε τα δύο αλφαριθμητικά είναι το ίδιο και αποτελείται από 4 στοιχεία.

Αυτό που έχουμε να παρατηρήσουμε είναι ότι στην προκειμένη περίπτωση ο πίνακας που δημιουργείται από τον κλασικό αλγόριθμο LCS είναι μικρός σε διαστάσεις (3000 x 1600 το πολύ), οπότε και ολοκληρώνει τους υπολογισμούς του σε όλα τα παραδείγματα. Σε αντίθεση και πάλι παρουσιάζεται το φαινόμενο που περιγράψαμε στο προηγούμενο πείραμά μας, ότι δηλαδή η εκτέλεση του DFS – LCS αλγορίθμου τερματίζεται από το λειτουργικό σύστημα, όταν το μήκος του Y φτάσει στο 1600, λόγω του μεγάλου βάθους στο οποίο φτάνει η αναδρομική συνάρτηση που χρησιμοποιεί ο αλγόριθμος.

Τα αποτελέσματα του πειράματος αυτού παρουσιάζονται σε δύο διαγράμματα. Αυτό οφείλεται στο γεγονός ότι οι χρόνοι εκτέλεσης του κλασικού LCS αλγορίθμου είναι αρκετά μικρότεροι σε σχέση με τον DFS – LCS. Οπότε για ευκρινέστερη παρουσίαση των αποτελεσμάτων έχει χρησιμοποιηθεί ένα διάγραμμα με τους χρόνους του DFS – LCS και ένα διάγραμμα με τους χρόνους του κλασικού LCS αλγορίθμου.

Διαγράμματα 3 & 4: Χρόνοι εκτέλεσης για μικρά X και Y. Τα X και Y έχουν δημιουργηθεί με το ίδιο αλφάβητο και το μήκος του Y αυξάνει κατά 100 σε κάθε πείραμα.



5.4 Πειράματα με κριτήριο την κατανάλωση μνήμης

Στην ενότητα αυτή θα εξετάσουμε τις απαιτήσεις που έχουν σε μνήμη οι δύο αλγόριθμοι. Θα δείξουμε δηλαδή την ποσότητα μνήμης που απαιτεί ο καθένας για την ολοκλήρωση των υπολογισμών του και την εξαγωγή του αποτελέσματος, που είναι η μεγαλύτερη κοινή ακολουθία των αλφαριθμητικών που θα δώσουμε ως είσοδο.

Στον πίνακα που ακολουθεί παρουσιάζουμε τα αποτελέσματα των πειραμάτων που πραγματοποιήσαμε για τον σκοπό αυτό. Οι στήλες του πίνακα που ακολουθεί παρουσιάζουν τα εξής στοιχεία

- Η πρώτη στήλη παρουσιάζει ποιες γραμμές του πίνακα έχουν χρησιμοποιηθεί για την κατασκευή των διαγραμμάτων που παρουσιάζονται στην πορεία. Κάθε μια γραμμή του πίνακα αποτελεί και ένα πείραμα που τρέξαμε.
- Η δεύτερη στήλη παρουσιάζει το μήκος του αλφαριθμητικού X
- Η τρίτη στήλη παρουσιάζει το μήκος του αλφαριθμητικού Y
- Η τέταρτη στήλη παρουσιάζει το πλήθος των στοιχείων της αλφαβήτου που χρησιμοποιήσαμε για την κατασκευή του αλφαριθμητικού X
- Η πέμπτη στήλη παρουσιάζει το πλήθος των στοιχείων της αλφαβήτου που χρησιμοποιήσαμε για την κατασκευή του αλφαριθμητικού Y

- Οι επόμενες δύο στήλες, η έκτη και η έβδομη, αναφέρονται στον DFS – LCS και παρουσιάζουν την κατανάλωση μνήμης από τον συγκεκριμένο αλγόριθμο.
    - Η έκτη στήλη παρουσιάζει συνολικά το μέγεθος της μνήμης που χρησιμοποίησε ο συγκεκριμένος αλγόριθμος και η μονάδα μέτρησης είναι τα MB. Αυτό που πρέπει να παρατηρήσουμε σε αυτό το σημείο είναι ότι τα MB που αναφέρονται στον πίνακα αυτόν είναι συνολικά η μνήμη που χρησιμοποιήθηκε από τον αλγόριθμο και περιλαμβάνει και την μνήμη RAM του υπολογιστή αλλά και το κομμάτι του σκληρού δίσκου που χρησιμοποιήθηκε ως μνήμη.
    - Στην έβδομη στήλη παρουσιάζεται το μέγεθος της μνήμης που χρειάστηκε για την αποθήκευση της δομής cache που εξετάσαμε προηγούμενα. Το κομμάτι αυτό για τον συγκεκριμένο αλγόριθμο περιλαμβάνεται στις μετρήσεις που παρουσιάζονται στην προηγούμενη στήλη, απλά παρουσιάζεται ξεχωριστά για να μας δώσει μια πιο ακριβή εικόνα για την κατανάλωση μνήμης του αλγορίθμου. Και εδώ η μονάδα μέτρησης είναι το MB.
- Η τελευταία στήλη αφορά τον κλασικό LCS αλγόριθμο και παρουσιάζει το μέγεθος της μνήμης που καταναλώνει αυτός ο αλγόριθμος για τους υπολογισμούς του.

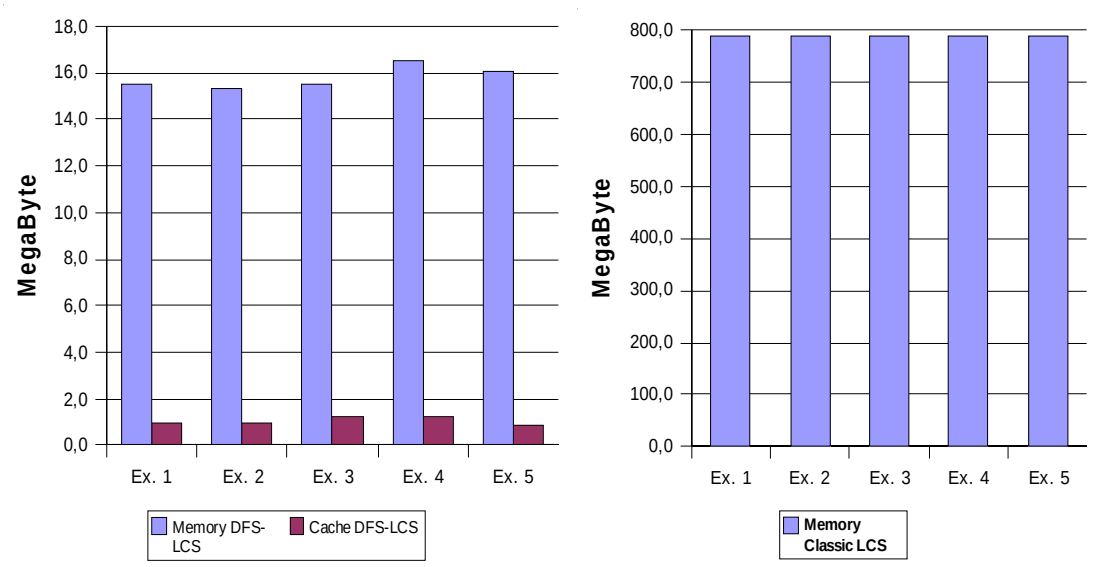
Πίνακας 5.2: Τα αποτελέσματα των πειραμάτων που πραγματοποιήθηκαν για την εξέταση της κατανάλωσης μνήμης για την ολοκλήρωση της εκτέλεσης των δύο αλγορίθμων.

			Μέγεθος του αλφάβητου		**DFS – LCS**		**Classic LCS**
	Μήκος του X	**Μήκος του Y**	**X**	**Y**	**Memory (MB)**	**Cache (MB)**	**Memory (MB)**
Διαγράμματα 5 – 6	1.000.000	100	4	4	15,523	0,976	788,000
	1.000.000	100	8	8	15,277	1,004	788,000
	1.000.000	100	16	16	15,500	1,196	788,000
	1.000.000	100	24	24	16,500	1,281	788,000
	1.000.000	100	36	36	16,063	0,916	788,000
Διαγράμματα 7 – 8	1.000.000	200	4	8	15,785	0,746	1551,000
	1.000.000	200	4	16	15,281	0,183	1551,000
	1.000.000	200	8	32	11,430	0,037	1551,000
	1.000.000	200	8	64	10,434	0,015	1551,000

Διάγραμμα 9	10.000.000	100	32	32	185,000	1,090	killed
	10.000.000	200	32	32	193,000	7,835	killed
	10.000.000	300	32	32	213,000	26,904	killed
	10.000.000	400	32	32	245,000	57,503	killed
	10.000.000	500	32	32	311,000	121,678	killed
	10.000.000	600	32	32	400,000	208,335	killed
	10.000.000	700	32	32	493,766	299,190	killed
	10.000.000	800	32	32	681,629	483,730	killed
	10.000.000	900	32	32	909,594	708,135	killed
	10.000.000	1000	32	32	1125,781	921,697	killed
	10.000.000	1100	32	32	1491,523	1281,904	killed
	10.000.000	1200	32	32	1887,000	1672,493	killed
	10.000.000	1300	32	32	2245,000	2026,931	killed
	10.000.000	1400	32	32	2705,000	2483,199	killed

Τα ακόλουθα διαγράμματα παρουσιάζουν την κατανάλωση μνήμης από τον κάθε αλγόριθμο για την εύρεση της μεγαλύτερης κοινής ακολουθίας των αλφαριθμητικών που δίνονται ως είσοδος στους δύο αλγορίθμους.

Διαγράμματα 5 & 6: Κατανάλωση μνήμης χρησιμοποιώντας ένα μεγάλο X και ένα μικρό Y. Τα X και Y δημιουργήθηκαν χρησιμοποιώντας το ίδιο αλφάβητο το οποίο αυξάνει από παράδειγμα σε παράδειγμα



Και στα δύο διαγράμματα στον οριζόντιο άξονα αναφέρεται το παράδειγμα, δηλαδή τα αποτελέσματα που βρίσκονται στην αντίστοιχη γραμμή του πίνακα αποτελεσμάτων που παρουσιάσαμε προηγούμενα. Στην κατακόρυφη στήλη παρουσιάζεται η κατανάλωση μνήμης του κάθε αλγορίθμου και η μονάδα μέτρησης

που χρησιμοποιείται είναι όπως έχουμε ήδη αναφέρει τα MB. Στο αριστερό διάγραμμα (διάγραμμα 5), έχουμε τα αποτελέσματα για τον DFS – LCS ενώ στο διάγραμμα στα δεξιά έχουμε τα αποτελέσματα του κλασικού αλγόριθμου για το LCS. Στο διάγραμμα 5 οι δύο στήλες αντιπροσωπεύουν την συνολική μνήμη που καταναλώθηκε από τον αλγόριθμο και την μνήμη που χρησιμοποιήθηκε για την δομή cache. Χρησιμοποιήθηκαν δύο διαγράμματα, λόγω της μεγάλης διαφοράς σε κατανάλωση μνήμης που παρουσιάζουν οι δύο αλγόριθμοι. Στο διάγραμμα του DFS – LCS η μεγαλύτερη τιμή που παρουσιάζεται είναι τα 16.5 MB ενώ στο διάγραμμα του κλασικού LCS αλγορίθμου η μεγαλύτερη τιμή που παρουσιάζεται είναι η 788MB.

Για όλα τα παραδείγματα του πειράματος αυτού χρησιμοποιήθηκε ένα αλφαριθμητικό X με μήκος 1000000 και ένα αλφαριθμητικό Y με μήκος 100. Το αλφάβητο ήταν το ίδιο και για την δημιουργία του X και για την δημιουργία του Y και ξεκινά από 4 στοιχεία και σε κάθε παράδειγμα αυτό αυξάνει.

Σύμφωνα λοιπόν με τα πειράματα αυτά, η κατανάλωση μνήμης από τον DFS – LCS είναι κατά πολύ μικρότερη από την κατανάλωση που παρουσιάζει ο κλασικός LCS αλγόριθμος για τα συγκεκριμένα παραδείγματα. Για παράδειγμα στο πρώτο πείραμα που κάναμε, οι μετρήσεις μας δείχνουν ότι ο κλασικός LCS αλγόριθμος απαιτεί 50.8 φορές την μνήμη που απαιτεί ο DFS – LCS αλγόριθμος για την ολοκλήρωση των υπολογισμών του και για την εξαγωγή του αποτελέσματος που είναι φυσικά η μεγαλύτερη κοινή ακολουθία των X και Y. Αντίστοιχα είναι και τα μεγέθη και για τα υπόλοιπα παραδείγματα που παρουσιάζονται στα διαγράμματα αυτά.

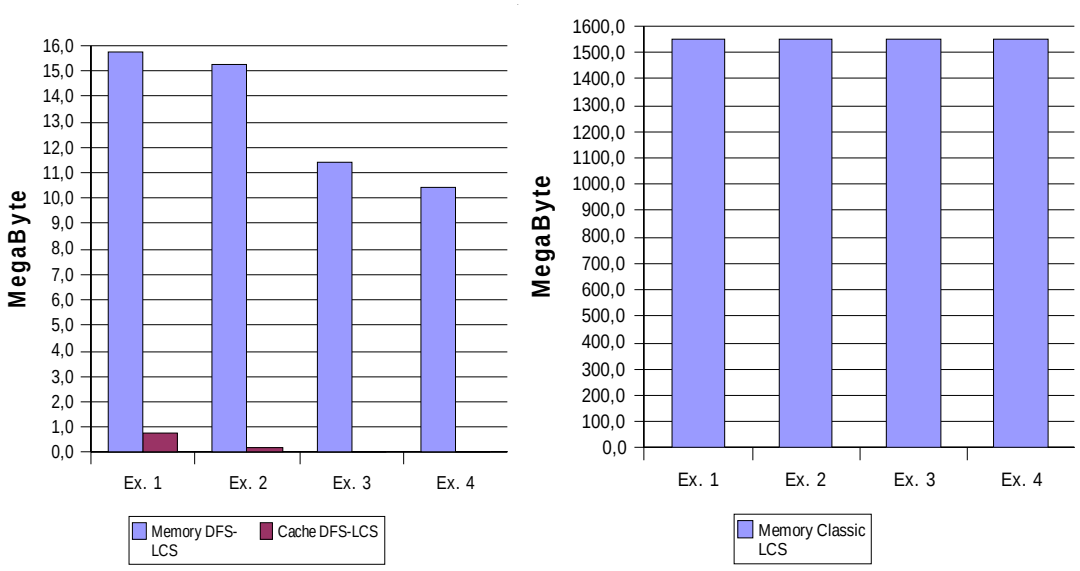
Συνεχίζοντας παρουσιάζουμε τα αποτελέσματα από τα επόμενα πειράματα που πραγματοποιήσαμε. Όπως έχουμε ήδη αναφέρει στην προηγούμενη ενότητα, το μέγεθος του αλφαβήτου από το οποίο έχει δημιουργηθεί το αλφαριθμητικό Y και η σχέση μεγέθους που έχει αυτό το αλφάβητο με το αλφάβητο από το οποίο έχει δημιουργηθεί το αλφαριθμητικό X, επηρεάζουν την απόδοση του DFS – LCS αλγορίθμου. Επίσης πρέπει να θυμόμαστε ότι κατά την εκτέλεση του DFS – LCS αλγορίθμου οι συνδυασμοί χαρακτήρων που καταφέραμε να εντοπίσουμε στο X και στο Y αποθηκεύονται στην δομή cache για να μπορέσουμε σε κάποιο επόμενο έλεγχο να χρησιμοποιήσουμε έτοιμα τα αποτελέσματα στα οποία καταλήξαμε προηγούμενα χωρίς να χρειάζεται να τα υπολογίσουμε ξανά. Αυτό αυτόματα σημαίνει ότι όσο περισσότεροι είναι αυτοί οι συνδυασμοί τόσο θα αυξάνει και η μνήμη που απαιτείται από την δομή cache για την αποθήκευση των αποτελεσμάτων που υπολογίζουμε, κάτι το οποίο εξαρτάται φυσικά και από το μέγεθος των δύο αλφαριθμητικών. Στις περιπτώσεις όμως όπου το αλφαριθμητικό Y έχει δημιουργηθεί από κάποιο αλφάβητο που περιλαμβάνει περισσότερα στοιχεία από τα στοιχεία του αλφαβήτου από το οποίο έχει δημιουργηθεί το αλφαριθμητικό X, τότε οι συνδυασμοί στοιχείων από το X και το Y είναι λιγότεροι και άρα οι εγγραφές στην cache είναι και αυτές αισθητά λιγότερες.

Τα αποτελέσματα από τα πειράματα αυτά παρουσιάζονται στα διαγράμματα που ακολουθούν όπου όπως θα αναλύσουμε στην συνέχεια φαίνεται καθαρά ότι οι απαιτήσεις σε μνήμη του DFS – LCS αλγορίθμου σε αυτές τις περιπτώσεις είναι ακόμα λιγότερες ενώ οι απαιτήσεις του κλασικού αλγόριθμου LCS δεν αλλάζουν. Αυτό οφείλεται στο γεγονός ότι ο κλασικός αλγόριθμος LCS δεν διακρίνει διαφορά ανάμεσα στα αλφάβητα από τα οποία δημιουργήσαμε τα δύο αλφαριθμητικά.

Και πάλι χρησιμοποιούνται δύο διαγράμματα, ένα για κάθε αλγόριθμο, διότι οι απαιτήσεις σε μνήμη κυμαίνονται σε τελείως διαφορετικά επίπεδα. Στους οριζόντιους

άξονες αναφέρεται ο αριθμός του παραδείγματος ενώ στους κατακόρυφους παρουσιάζονται οι απαιτήσεις σε μνήμη μετρημένες σε MB.

Διαγράμματα 7 & 8: Η κατανάλωση μνήμης από τους αλγορίθμους όταν χρησιμοποιούμε ένα μεγάλο X και ένα μικρό Y. Το X είναι κατασκευασμένο από ένα αλφάβητο 4 και 8 στοιχείων ενώ το αλφάβητο του Y αυξάνει από παράδειγμα σε παράδειγμα



Αναφέρουμε ότι η μεγαλύτερη τιμή για την κατανάλωση μνήμης από τον DFS – LCS είναι 15,785MB ενώ για τον κλασικό LCS αλγόριθμο είναι 1551MB. Επίσης αξίζει να παρατηρήσουμε ότι το μέγεθος της μνήμης που απαιτείται από τον DFS – LCS αλγόριθμο, όσο μεγαλώνει το αλφάβητο από το οποίο έχει δημιουργηθεί το αλφαριθμητικό X, μειώνεται συνεχώς. Επίσης μειώνεται κατά πολύ και το μέγεθος της cache και στο τελευταίο παράδειγμα όπου το αλφάβητο του X αριθμεί 4 στοιχεία και το αλφάβητο του Y αριθμεί 64 στοιχεία, το μέγεθος της cache είναι μόλις 0,015MB.

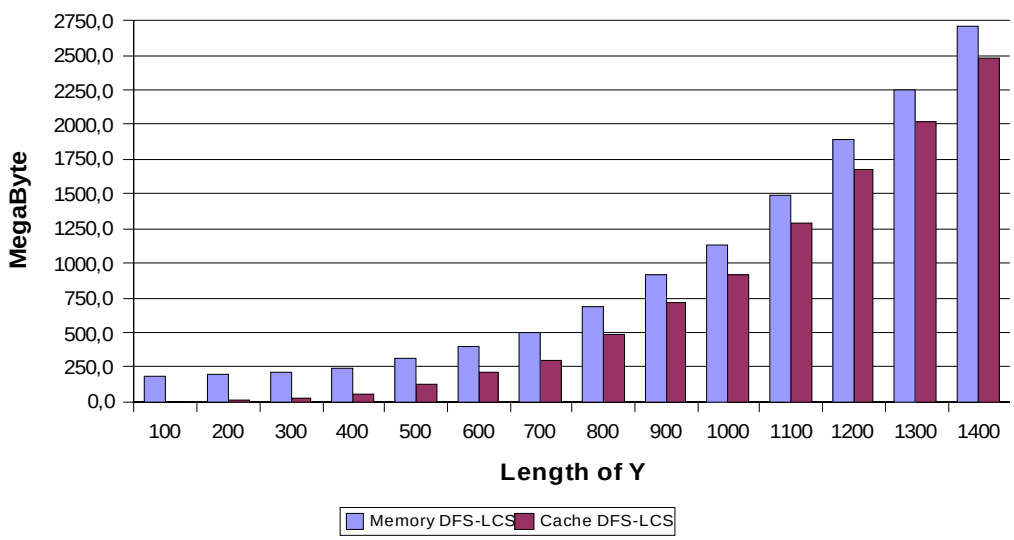
Συγκεκριμένα για το τέταρτο πείραμα αυτής της ενότητας αυτό που έχουμε να παρατηρήσουμε είναι ο κλασικός LCS αλγόριθμος απαιτεί για την ολοκλήρωση των υπολογισμών του 98.2 φορές την μνήμη που απαιτεί ο DFS – LCS αλγόριθμος για την εξαγωγή του αποτελέσματος.

Στο τελευταίο πείραμά μας παρουσιάζουμε την περίπτωση όπου χρησιμοποιείται ένα μεγάλο X μήκους 10000000 και ένα Y που αποτελείται αρχικά από 100 στοιχεία και το μήκος του αυξάνει κατά 100 σε κάθε παράδειγμα. Τα X και Y έχουν δημιουργηθεί χρησιμοποιώντας ακριβώς το ίδιο αλφάβητο που αποτελείται από 32 στοιχεία. Στο διάγραμμα που ακολουθεί παρουσιάζονται μόνο τα αποτελέσματα για τον DFS – LCS αλγόριθμο. Αυτό οφείλεται στο γεγονός ότι οι απαιτήσεις του κλασικού LCS αλγορίθμου σε μνήμη για την επίλυση των συγκεκριμένων προβλημάτων, ήταν τέτοιες που υπερέβησαν την μνήμη που διέθετε το συγκεκριμένο μηχάνημα και το πρόγραμμα τερματίστηκε από το λειτουργικό σύστημα.

Στον οριζόντιο άξονα παρουσιάζεται το μήκος του αλφαριθμητικού Y, που όπως είπαμε αυξάνεται κατά 100 σε κάθε πείραμα. Στον κατακόρυφο άξονα είναι και πάλι τα MB της μνήμης που χρησιμοποιήθηκαν από τον αλγόριθμο για τους υπολογισμούς

του. Η πρώτη στήλη μας δείχνει γενικά ολόκληρο το μέγεθος της μνήμης ενώ η δεύτερη απεικονίζει πόση μνήμη καταναλώθηκε αποκλειστικά από την δομή cache του αλγορίθμου.

Διάγραμμα 9: Η κατανάλωση μνήμης από τον DFS – LCS όταν χρησιμοποιούμε μεγάλο X και Y του οποίου το μέγεθος αυξάνει κατά 100 σε κάθε παράδειγμα. Τα X και Y δημιουργήθηκαν από το ίδιο αλφάβητο 32 στοιχείων.



Αυτό που έχουμε να παρατηρήσουμε εδώ είναι ότι όσο αυξάνεται το μήκος του Y, αυξάνονται και οι απαιτήσεις του αλγορίθμου σε μνήμη, οι οποίες ξεκινούν από αρκετά χαμηλά επίπεδα για να φτάσουν σε αρκετά υψηλότερα για Y που φτάνει τα 1400 στοιχεία. Επίσης το μεγαλύτερο μέρος της μνήμης αυτής καταναλώνεται από την δομή cache που χρησιμοποιείται.

ΚΕΦΑΛΑΙΟ 6

ΣΥΜΠΕΡΑΣΜΑΤΑ

Παρουσιάσαμε το πρόβλημα της εύρεσης της μεγαλύτερης κοινής ακολουθίας δύο αλφαριθμητικών. Όπως αναφέρθηκε το πρόβλημα αυτό εντάσσεται σε μια ευρύτερη κατηγορία αλγορίθμων που μελετούν διαφορετικές πτυχές του προβλήματος της εύρεσης κάποιας κοινής ακολουθίας ανάμεσα σε δύο αλφαριθμητικά.

Ο αλγόριθμος που παρουσιάστηκε στην μελέτη αυτή αναλυτικά ήταν ο κλασικός αλγόριθμος για την εύρεση της μεγαλύτερης κοινής ακολουθίας, που είναι ο αλγόριθμος των Hunt και Szymanski, και ο οποίος έχει αποτελέσει αντικείμενο έρευνας και εξέλιξης πολλών επιστημόνων.

Επίσης παρουσιάστηκε μια διαφορετική προσέγγιση στο πρόβλημα αυτό χρησιμοποιώντας τον DFS –LCS αλγόριθμο. Ο αλγόριθμος αυτός σε σύγκριση με τον κλασικό LCS αλγόριθμο παρουσιάζει ένα μεγάλο πλεονέκτημα. Σε όλες τις περιπτώσεις όπου το μέγεθος του Y είναι σχετικά μικρό, καταναλώνει πολύ λιγότερη μνήμη από ότι ο κλασικός αλγόριθμος, ενώ παρουσιάζει πολύ μικρή εξάρτηση από το μέγεθος του αλφαριθμητικού X. Επίσης μπορεί να διακρίνει τις διαφορές στα αλφάβητα που χρησιμοποιήθηκαν για την κατασκευή των δύο αλφαριθμητικών και αυτή του η ιδιότητα του παρέχει την δυνατότητα να εξετάζει με μεγαλύτερη ευκολία τα δύο αλφαριθμητικά.

Από άποψη χρόνου εκτέλεσης, παρατηρούμε ότι στις περιπτώσεις που αναφέραμε προηγούμενα, οι χρόνοι εκτέλεσης του συγκεκριμένου αλγορίθμου είτε δεν διαφέρουν ιδιαίτερα από τους χρόνους εκτέλεσης του κλασικού αλγόριθμου για το LCS είτε είναι καλύτεροι, ιδιαίτερα σε περιπτώσεις μεγάλου X.

Σε αντίθεση ο κλασικός αλγόριθμος για το LCS, δοκιμαζόμενος στα ίδια παραδείγματα, κατανάλωσε πολύ μεγαλύτερες ποσότητες μνήμης, ενώ ταυτόχρονα προκάλεσε και μεγάλες καθυστερήσεις λόγω του swap που προκάλεσε στον σκληρό δίσκο του υπολογιστή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] James W. Hunt, Thomas G. Szymanski (1977), "A Fast Algorithm for Computing Longest Common Subsequences", *Association for Computing Machinery*, 20, pp. 350 – 353

[2] C. E. R. Alves, E. N. Cáceres, S. W. Song (2003), "A BSP/CGM Algorithm for the All-Substrings Longest Common Subsequence Problem", In: *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*

[3] C. E. R. Alves, E. N. Cáceres, S. W. Song (2003), "*Sequential and Parallel Algorithms for the All-Substrings Longest Common Subsequence Problem*", Technical Report RT-MAC-2003-03, Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo

[4] Ronald I. Greenberg (2002) "Fast and Simple Computation of All Longest Common Subsequences", arXiv:cs.DS/0211001v1

[5] Francis Y. L. Chin, Alfredo De Santis, Anna Lisa Ferrara, N. L. Ho, S. K. Kim (2004), "A simple algorithm for the constrained sequence problems", *Elsevier B.V.* doi:10.1016/j.ipl.2004.02.008

[6] Chao-Li Peng (2003), "*An Approach for Solving the Constrained Longest Common Subsequence Problem*", Thesis, National Sun Yat-sen University

[7] AnYuan Guo, Hava Siegelmann (2004), "Time – Warped Longest Common Subsequence Algorithm for music retrieval", In: *Proceedings of the 5th International Conference on Music Information Retrieval*, Barcelona, Spain

[8] D. Huson (2005), "Comp. Sequence Analysis", WS'04 ZBIT

[9] David Liben – Nowell, Erik Vee, An Zhu (2003), "Finding Longest Increasing and Common Subsequence in Streaming Data", In: *Proceedings of the 11th International Computing and Combinatorics Conference (COCOON)*

[10] Sergei Bespamyatnikh, Michael Segal (1999), "Enumerating Longest Increasing Subsequences and Patience Sorting", Pacific Inst. for the Math. Sci. Preprints, PIMS-99-3

[11] Ronald I. Greenberg (2002) "Fast and Simple Computation of All Longest Common Subsequences", arXiv:cs.DS/0211001v1

[12] Michael T. Goodrich, Roberto Tamassia (2001), "*Algorithm Design*", pp. 443-446, Wiley Publications

[13] Mi Lu, Hua Lin (1994), "Parallel Algorithms for the Longest Common Subsequence Problem", *IEEE Transactions on Parallel and Distributed Systems*, 5(8)

[14] Eric A. Breimer, Mark K. Goldberk (1998), "*Case Study of a Learning Algorithm for the Longest Common Subsequence Problem*", In Technical Report TR 98-3, Computer Science Department, Rensselaer Polytechnic Institute

[15] Daniel S. Hirschberg (1977), "Algorithms for the Longest Common Subsequence Problem", *Journal of the Association for the Computing Machinery*, 24(4), 664 – 675

[16] H. Goeman, M. Clausen (1999), “A New Practical Linear Space Algorithm for the Longest Common Subsequence Problem”, *The Prague Stringology Club Workshop*

[17] L. Bergroth, H. Hakonen, T. Raita (2000) “A Survey of Longest Common Subsequence Algorithms”, In: *Proceedings of the Seventh International Symposium on String Processing and Information Retrieval (IEEE)*

[18] Xiaohua Xu, Ling Chen, Yi Pan, Ping He (2005), “Fast Parallel Algorithms for the Longest Common Subsequence Problem Using an Optimal Bus”, ICCSA 2005, LNCS 3482, 338-348

[19] C. E. R. Alves, E. N. Cáceres, S. W. Song (2003), “Comparison of Genomes using High – Performance Parallel Computing”, In: *Proceedings of the 15th Symposium of Computer Architecture and High – Performance Computing (SBAC –PAD ’03)*