ΠΙΘΑΝΕΣ ΕΡΩΤΗΣΕΙΣ ΒΙΟΠΛΗΡΟΦΟΡΙΚΗΣ

Φυλλογεννητικά δέντρα

Στο παρελθόν στηρίζονταν σε μορφολογικά χαρακτηριστικά, σήμερα στηρίζονται στις αλληλουχίες DNA

Στα φύλλα γονιδιώματα(οργανισμοί) ή χαρακτηριστικά(συμβολοσειρές) που υπάρχουν τώρα

Στους κόμβους υποθετικά προγονικά είδη

Η ρίζα(αν υπάρχει) αναπαριστά τον αρχαιότερο εξελικτικό πρόγονο

Οι ακμές μπορεί να έχουν συντελεστές στάθμισης που αναπαριστούν:

1. αριθμό μεταλλάξεων από γονίδιο σε ένα είδος στο ίδιο γονίδιο άλλου είδους
2. χρονική εκτίμηση για την εξέλιξη του ενός είδους στο άλλο

di,j(T): απόσταση δέντρου(μήκος διαδρομής) μεταξύ i, j(το άθροισμα όλων των συντελεστών στις ακμές)

Di,j: απόσταση μετασχηματισμού μεταξύ i, j στην μήτρα αποστάσεων n*n(n: αριθμός ειδών)
Χρειάζεται να κατασκευαστεί δέντρο με την καλύτερη προσαρμογή στη μήτρα αποστάσεων
Αν υπάρχει δέντρο στο οποίο για κάθε i, j ισχύει di,j(T) = Di,j, τότε είναι μοναδικό και η μήτρα προσθετική.

Για την κατασκευή του δέντρου συμπιέζουμε «γειτονικά» φύλλα και αντικαθιστούμε τις αντίστοιχες γραμμές, στήλες τους στον πίνακα με 1 κοινή γραμμή, στήλη(κοινή κορυφή στο δέντρο): neighborjoin. Γειτονικά φύλλα i, j συμπιέζονται σε «γονική» κορυφή k και υπολογίζουμε την απόσταση από την k προς οποιοδήποτε άλλο φύλλο m: Dk,m = (Di,m + Dj,m -Di,j) / 2
ΠΡΟΣΟΧΗ: δεν είναι γειτονικά πάντα αυτά που στο αρχικό δέντρο είναι πιο κοντά μεταξύ τους.

Το δέντρο μπορεί να κατασκευαστεί και με UPGMA αλλά μειονέκτημα: η απόσταση της ρίζας από όλα τα φύλλα ίδια γιατί υποθέτει ότι η εξέλιξη(μεταλλάξεις) σε όλα τα υποδέντρα συμβαίνουν ταυτόχρονα(μοριακό ρολόι).

Εκφυλισμένη τριάδα κόμβων είναι τριάδα σημείων 1<= i, j , k <= n με Di,j + Djk = Dik(στο δέντρο ακμές με βάρος 0 -> το j μπορεί να αφαιρεθεί).

Η μήτρα είναι προσθετική αν και μόνο αν ισχύει η συνθήκη των 4 σημείων για κάθε 4 διαφορετικά φύλλα 1<= i, j, k, l <= n: δύο από τα αθροίσματα (Di,j + Dk,l) (Di,k + Dj,l) (Di,l +Dj,k) είναι ίδια και το τρίτο έχει μικρότερη τιμή

Αν η μήτρα δεν είναι προσθετική ψάχνουμε το δέντρο που την προσεγγίζει καλύτερα, που έχει δηλαδή μικρότερο τετράγωνο σφάλματος Σ(di,j – Di,j)^2 (NP-hard πρόβλημα)

Μια μήτρα στοίχισης n*m για ακολουθίες μπορεί να μετασχηματιστεί σε μήτρα αποστάσεων n*n αλλά το αντίστροφο ΔΕΝ γίνεται πάντα

Άλλος τρόπος κατασκευής του δέντρου: με βάση τους χαρακτήρες και χρήση της μήτρας στοίχισης n*m. Κάθε χαρακτήρας έχει κατάσταση(πχ: A,C, G, ή T αν είναι νουκλεοτίδιο). Για n είδη και m χαρακτήρες για κάθε είδος, ψάχνουμε δέντρο με τη μικρότερη βαθμολογία οικονομίας(τον μικρότερο άθροισμα μεταλλάξεων-συντελεστών των ακμών).

Μικρό πρόβλημα οικονομίας:
Για αλφάβητο k γραμμάτων

είσοδος: δέντρο T όπου κάθε φύλλο έχει ετικέτα μια ακολουθία m χαρακτήρων ή έναν χαρακτήρα, αν όλοι οι χαρακτήρες ανεξάρτητοι
έξοδος: η αντιστοίχιση ετικετών στις εσωτερικές κορυφές του T που ελαχιστοποιεί τη βαθμολογία οικονομίας

Παραλλαγή: σταθμισμένο μικρό πρόβλημα οικονομίας
υπάρχει επιπλέον στην είσοδο μήτρα βαθμολόγησης με το κόστος μετασχηματισμού από κάθε κατάσταση σε άλλη

Αλγόριθμος Sankoff:
Ξεκινά από τα φύλλα. Αν κάποιο φύλλο έχει χαρακτήρα που θέλουμε, βαθμολογία 0 αλλιώς άπειρο. Προχωρά προς τα πάνω στο δέντρο μέχρι τη ρίζα. Κάθε κόμβος-πρόγονος παίρνει βαθμολογία την ελάχιστη βαθμολογία του υποδέντρου που έχει αυτόν ρίζα. Αφού υπολογιστεί η βαθμολογία στη ρίζα του δέντρου, μετακινείται προς τα κάτω αναθέτοντας το βέλτιστο χαρακτήρα σε κάθε κορυφή. O(n*k)

Αλγόριθμος Fitch:
Παρόμοιος αλλά σε κάθε κορυφή αντιστοιχίζει ένα σύνολο γραμμάτων. Το σύνολο κάθε κόμβου-προγόνου είναι ο συνδυασμός των συνόλων των απογόνων του. O(n*k)

Πανομοιότυποι(δίνουν το ίδιο βέλτιστο σύνολο ετικετών).

Μεγάλο πρόβλημα οικονομίας:

είσοδος: μήτρα M n*m για n είδη, καθένα m χαρακτήρες
έξοδος: δέντρο T με n φύλλα που έχουν ετικέτες τις γραμμές της M, αντιστοίχιση ετικετών σε κόμβους που ελαχιστοποιεί τη βαθμολογία οικονομίας για όλα τα δυνατά δέντρα και όλες τις δυνατές σημάνσεις των εσωτερικών κόμβων(NP-complete)

Hidden Markov Model(HMM)

Αφηρημένη μηχανή, μοντελοποίηση γεννήτριας συμβολοσειρών. Έχει σύνολο Q από κρυφές καταστάσεις. Από κάθε κατάσταση μεταβαίνει σε κάποια άλλη και εκπέμπει ένα σύμβολο από αλφάβητο Σ με κάποια πιθανότητα.

akl: μήτρα |Q|x|Q| που περιγράφει την πιθανότητα να πάμε από κατάσταση k σε κατάσταση l

ek(b): μήτρα |Q|x|Σ| που περιγράφει την πιθανότητα να εκπέμψουμε το σύμβολο b όταν βρισκόμαστε στην κατάσταση k

Δεν έχουμε μνήμη αλλά πλέγμα Manhattan για τις μεταβάσεις. Επιτρέπεται κίνηση μόνο ανατολικά

Πρόβλημα αποκωδικοποίησης: να βρεθεί η βέλτιστη κρυφή διαδρομή καταστάσεων με δεδομένες παρατηρήσεις

> είσοδος: ακολουθία παρατηρήσεων x = x1,…,xn
> έξοδος: η διαδρομή π = π1,…,πn που μεγιστοποιεί την πιθανότητα για όλες τις δυνατές διαδρομές στο πλέγμα

Εκμάθηση Viterbi: μήτρες a, ek άγνωστες
αναζήτηση στο χώρο όλων των πιθανών διαδρομών
πολυπλοκότητα: O(n*|Q|)

Τα HMMs μπορούν να χρησιμοποιηθούν για στοίχιση μιας αλληλουχίας με ένα προφίλ που αναπαριστά μια οικογένεια πρωτεϊνών. Ουσιαστικά πολλαπλή στοίχιση με πιθανότητες.

Ορισμός βιοπληροφορικής

Η εφαρμογή υπολογιστικών τεχνικών και μεθόδων στην προσπάθεια κατανόησης και οργάνωσης δεδομένων και πληροφοριών που σχετίζονται με βιολογικά μακρομόρια.

Τομείς έρευνας της βιοπληροφορικής

σχεδιασμός και υλοποίηση υπολογιστικών εργαλείων για ανάκτηση γνώσης από βάσεις δεδομένων

ανάλυση βιολογικών δεδομένων

κατηγοριοποίηση βιολογικών δεδομένων

μοριακή μοντελοποίηση

σχεδιασμός φαρμάκων με χρήση Η/Υ

Τι είναι γονιδίωμα

το γενετικό υλικό ενός οργανισμού(ολόκληρη η ακολουθία DNA του)

κάθε κύτταρο περιλαμβάνει όλο το γονιδίωμα ενός οργανισμού

Τι είναι γονίδιο

βασική φυσική και λειτουργική μονάδα κληρονομικότητας

βρίσκεται στα χρωμοσώματα

αποτελείται από DNA

κωδικοποιεί τις οδηγίες για την παρασκευή πρωτεϊνών

Τι είναι πρωτεϊνη

συστατικό στοιχείο της κυτταρικής δομής

μεγάλο σύνθετο μόριο που αποτελείται από μικρότερες υπομονάδες, τα αμινοξέα(20 διαφορετικά)

σχηματίζει ένζυμα που στέλνουν σήματα σε άλλα κύτταρα και ρυθμίζουν τις δραστηριότητες των γονιδίων

Τι είναι DNA

βιολογικό μακρομόριο

αποθηκεύει πληροφορίες για τον τρόπο λειτουργίας του κυττάρου και τη γενετική πληροφορία του οργανισμού

διπλή ελικοειδής δομή που σχηματίζεται από τις βάσεις αδενίνη(A), κυτοσίνη(C), γουανίνη(G), θυμίνη(T) με βάση την αρχή της συμπληρωματικότητας: A-T, G-C

Τι είναι RNA

βιολογικό μακρομόριο

μεταφέρει τμήματα πληροφοριών σε διαφορετικά σημεία του κυττάρου

παρέχει πρότυπα για τη σύνθεση πρωτεϊνών

προκύπτει από το DNA με βάση την αρχή της συμπληρωματικότητας αλλά η θυμίνη(T) αντικαθίσταται με ουρακίλη(U)

Κεντρικό Δόγμα μοριακής βιολογίας

DNA αντιγράφεται σε DNA, RNA αντιγράφεται σε RNA

DNA μεταγράφεται σε RNA

RNA μεταφράζεται σε πρωτεϊνη

Τι είδη βιολογικών ακολουθιών συναντάμε

νουκλεοτιδικές: DNA, RNA

αμινοξικές: πρωτεϊνες

Ποια είναι τα είδη βιολογικών βάσεων δεδομένων

- Γενικευμένες:
  περιέχουν νουκλεοτιδικές και αμινοξικές ακολουθίες από γονιδιώματα οργανισμών που έχουν αποκρυπτογραφηθεί πλήρως(πρωτογενείς) ή περιέχουν τρισδιάστατες δομές νουκλεϊνικών οξέων και πρωτεϊνών
  Παραδείγματα: NCBI, EMBL, DDBJ

- Δευτερεύουσες:
  προκύπτουν από ανάλυση των δεδομένων που είναι αποθηκευμένα στις αρχειακές βάσεις δεδομένων

- Εξειδικευμένες: για μικροσυστοιχίες ή μεταβολικά μονοπάτια

- Βιβλιογραφικές

- Βιολογικές βάσεις δεδομένων ιστοσελίδων με εγγραφές άλλες βιολογικές βάσεις ή συνδέσμους μεταξύ βιολογικών βάσεων

Ποια είναι η πιο γνωστή βάση δεδομένων για πρωτεϊνες

Protein Data Bank(PDB) για τρισδιάστατα μοντέλα πρωτεϊνών

Πως σχεδιάζονται φάρμακα με χρήση υπολογιστή

Σχεδίαση της ένωσης-οδηγού

Βελτιστοποίηση της ένωσης οδηγού

Δοκιμές in vitro(στο εργαστήριο) και in vivo(σε έμβιους οργανισμούς)

Τοξικολογικές δοκιμές

Δοκιμές στον άνθρωπο

Έλεγχος απόδοσης

Πιο είναι το κάτω όριο συγκρίσεων για την εύρεση προτύπου P (μέγεθος m) σε κείμενο T (μέγεθος n)

$\Omega(n+m)$ γιατί πρέπει να διαβαστεί ολόκληρο το T τουλάχιστον 1 φορά (χρόνος $\Omega(n)$) και να διαβαστεί το P (χρόνος $\Omega(m)$).

Τι είναι exact pattern matching

Έστω μια ακολουθία χαρακτήρων T. Αναζητούμε τις θέσεις εμφάνισης του προτύπου P μέσα στην ακολουθία T.

/**
Όταν λέμε matching γενικά εννοούμε εμφάνιση ολόκληρου του P μέσα στο T αλλά στους αλγορίθμους σε κάθε βήμα όταν λέμε match ή mismatch εννοούμε ταίριασμα ή όχι για χαρακτήρα. Αν όλοι συνεχόμενοι χαρακτήρες είναι matched τότε θα λέμε ότι έχουμε match του P με το T

Όταν λέμε πόσες φορές υπάρχει το P μέσα στο T εννοούμε σε ποιες θέσεις i του T ξεκινάει το P
**/

Πως λειτουργεί το exact pattern matching με k-mers

Σπάμε το T σε k-mers (μικρότερα τμήματα μήκους ίδιου με το P). Φτιάχνουμε hash table με ένα κελί για κάθε k-mer. Σε κάθε κελί αντιστοιχεί λίστα με τις εμφανίσεις του k-mer στο T και τις θέσεις τους στο T.

Αν κωδικοποιήσουμε κάθε k-mer ως μια συμβολοσειρά από 0 και 1, μπορούν επιπλέον να χρησιμοποιηθούν Bloom filters για βελτίωση της απόδοσης: εισαγωγή των k-mers στο φίλτρο και σύγκριση κάθενός με το P. Αν το φίλτρο δείχνει πιθανό ταίριασμα, εφαρμογή exact pattern matching για το συγκεκριμένο k-mer και το P, αλλιώς όχι σύγκριση.

Χρησιμοποιείται για την εύρεση επαναλήψεων

Μπορεί να χρησιμοποιηθεί και για approximate matching, με χρήση του Bloom filter για να φιλτραριστούν περιοχές του T όπου το P είναι απίθανο να ταιριάζει. Χρειάζεται όμως και πάλι exact pattern matching

Ποια η διαφορά μεταξύ substring και subsequence

substring: ψάχνουμε εμφανίσεις του P στο T σε συνεχόμενες θέσεις (το P πρέπει να εμφανίζεται ολόκληρο όπως είναι, δεν επιτρέπονται κενά ή άλλα σύμβολα μεταξύ συμβόλων)

subsequence: ψάχνουμε εμφανίσεις του P στο T αλλά όχι απαραίτητα σε συνεχόμενες θέσεις

Γενικά λέξη != συμβολοσειρά != ακολουθία

Τι σημαίνει proper prefix (αντίστοιχα για το proper suffix)

είναι substring συμβολοσειράς S με μήκος αυστηρά μικρότερο από το μήκος της S (δλδ proper prefix δεν μπορεί να είναι ολόκληρη η S ούτε το κενό)

Πότε μια συμβολοσειρά έχει όριο το w;

Όταν το w είναι ταυτόχρονα prefix και suffix

Τι κοινό έχουν οι αλγόριθμοι Z-preprocessing, Boyer-Moore, KMP

exact pattern matching

η πολυπλοκότητά τους είναι ανεξάρτητη του μεγέθους του αλφαβήτου

επίσης έχουν κοινό ότι όλοι κάνουν preprocessing

Πως λειτουργεί ο αλγόριθμος βασικής προεπεξεργασίας Z

Για συμβολοσειρά S κατασκευάζει Z-array μεγέθους |S|, όπου
Zi(S): το μήκος του μεγαλύτερου substring του S που ξεκινά από τη θέση i και είναι πρόθεμα του S

Z-box-at-i: το σύνολο των χαρακτήρων που αρχίζουν από το i και τελειώνουν στη θέση i + Zi(S) – 1. Έχει μεταβλητό μέγεθος για να χωράει κάθε δεδομένη στιγμή το μεγαλύτερο substring που ταιριάζει με πρόθεμα

Για κάθε i, ri συμβολίζει το δεξιότερο άκρο του Z-box που ξεκινά πριν από ή στη θέση i, li συμβολίζει το αριστερότερο άκρο

Κάθε τιμή Zi=m, i>m σηματοδοτεί ταίριασμα στη θέση i-m-1
(εξήγηση: έχουμε βρεί ταίριασμα μήκους m-όσο δηλαδή είναι το P- που ξεκινά από τη θέση i)

Με δεδομένα Zi τουλάχιστον μέχρι τη θέση k-1:
Αν k < r υπολογίζουμε το Zk. Αν Zk>0 τότε r = k + Zk -1 και l = k
(εξήγηση: αν βρισκόμαστε μέσα στο Z-box και στη θέση k ξεκινά substring που ταιριάζει με πρόθεμα του S, μετακίνηση των δεικτών ώστε να χωρέσει το substring μέσα στο νέο Z-box)
Αν k <= r τότε το S[k…r] ταιριάζει με το S[k-l+1…Zl]
(εξήγηση: αν βρισκόμαστε μέσα στο Z-box ή έχουμε φτάσει στο δεξί όριό του, τότε το substring από τη θέση k έως το δεξί όριο ταιριάζει με το substring S[k-l+1…Zl]. Ελέγχουμε αν Zk' < |S[k…r]|. Αν ναι, Zk=Zk' και δεν αλλάζουν οι δείκτες. Συνεχίζουμε τη σύγκριση από τη θέση r+1 μέχρι να βρεθεί mismatch και τότε αλλάζουν οι δείκτες

Για exact pattern matching κειμένου T με πρότυπο P:
συνένωση του P με το T σε ενιαία συμβολοσειρά S = P$T ($ σύμβολο που δεν ανήκει στο αλφάβητο) και εφαρμογή του αλγορίθμου
Πολυπλοκότητα χρόνου/χώρου: O(n+m)

Πως λειτουργεί ο αλγόριθμος Knuth-Morris-Pratt(KMP)

Στόχος: να βρεί το μέγιστο πρόθεμα του T που είναι επίθεμα του P

Στοιχίζει το P με το T ώστε ο αριστερότερος χαρακτήρας του P να είναι κάτω από τον αριστερότερο του T. Συγκρίνει από αριστερά προς τα δεξιά το P με το T. Σε περίπτωση mismatch μεταφέρει το P δεξιά κατά X θέσεις, σύμφωνα με το prefix table.

Για κάθε θέση i στο P, sp(i) είναι το μέγεθος του μεγαλύτερου proper suffix της υποσυμβολοσειράς P[1...i] που ταιριάζει με κάποιο πρόθεμα του T. Οι χαρακτήρες P[i+1] και P[sp(i) +1] πρέπει να είναι διαφορετικοί

1. Κατασκευή prefix table LPS με βάση το P (preprocessing).

2. Αναζήτηση:
   Αρχικοποίηση δείκτη i(τρέχουσα θέση στο T) και δείκτη j(τρέχουσα θέση στο P).
Αυξάνοντας σε κάθε βήμα το i κατά 1:
   αν T[i] == P[j] άυξησε κατά 1 τα i,j
   αν το j φτάσει στο τέλος του P, έχει βρεθεί match
   αν υπάρχει mismatch και j>0 (δεν είμαστε στην αρχή), ενημέρωσε το j με βάση το prefix table

Χρονική πολυπλοκότητα: O(n+m)
είναι βέλτιστος αλγόριθμος και λαμβάνει υπόψη του τις επανεμφανίσεις
Χωρική πολυπλοκότητα: O(m)

Είναι ο μόνος real-time αλγόριθμος (ακουμπάει κάθε σύμβολο μόνο 1 φορά)

Πως λειτουργεί ο αλγόριθμος Boyer-Moore

Στοίχιση των P και T όπως στον KMP. Η μετατόπιση του P γίνεται σύμφωνα με τους κανόνες:

right to left scan: σύγκριση χαρακτήρων από δεξιά προς τα αριστερά

bad character rule:
Έστω ότι ο τελευταίος χαρακτήρας του P είναι y και ο χαρακτήρας στην αντίστοιχη θέση του T είναι x. Αν γνωρίζουμε τη δεξιότερη θέση του x στο P, μπορούμε να μετατοπίσουμε το P όσες θέσεις δεξιά χρειάζεται ώστε ο δεξιότερος χαρακτήρας x του P να στοιχηθεί με τον αντίστοιχο χαρακτήρα x του T

(Με απλά λόγια αν έχουμε mismatch ψάχνουμε τη δεξιότερη θέση του x στο T και ολισθαίνουμε το P δεξιά μέχρι να γίνει match. Αν δεν υπάρχει το x στο T, ολισθαίνουμε το P δεξιά μέχρι να προσπεράσουμε το x)

good suffix rule:
Έστω μια δεδομένη στοίχιση του P με το T και μιας υποσυμβολοσειράς t του T που ταιριάζει με ένα επίθεμα του P, όπου στην αμέσως επόμενη θέση υπάρχει mismatch.
Βρες αν υπάρχει αντίγραφο t' του t στο δεξιότερο τμήμα του P, το οποίο να μην είναι επίθεμα του P και ο χαρακτήρας αριστερότερα του t' να είναι διαφορετικός από τον αντίστοιχο χαρακτήρα του t. Αν υπάρχει, μετατόπισε δεξιά το P ώστε το t' να στοιχηθεί κάτω από το t.
Αν δεν υπάρχει, μετατόπισε το αριστερό άκρο του P μετά το αριστερό άκρο του t στο T κατά τον ελάχιστο αριθμό θέσεων, ώστε ένα πρόθεμα του P να ταιριάζει με κάποιο επίθεμα του T

(Με απλά λόγια ολίσθηση του P δεξιά μέχρι να βρεθεί νέο match του t' με το T ή μέχρι να προσπεραστεί το t'. Αν το t' δεν υπάρχει, μετατόπισε το αριστερότερο άκρο του P μετά το αριστερό άκρο του t στο T ώστε ένα πρόθεμα του P να ταιριάζει με ένα επίθεμα του t στο T)

Σε περίπτωση που μπορούν να εφαρμοστούν και οι δυο κανόνες, εφαρμόζουμε αυτόν που θα κάνει τις περισσότερες ολισθήσεις

preprocessing: bad character table, good suffix table (δείχνουν με βάση το P πόσους χαρακτήρες μπορούμε να κάνουμε κάθε φορά skip)

Χρονική πολυπλοκότητα:
γραμμικός χρόνος αν P μικρό σε σχέση με το T
υπογραμμικός O(n/m) στη μέση περίπτωση
στο worst case O(n*m) αλλά αυτό συμβαίνει σπάνια

Πολύ αποδοτικός για μεγάλα T γιατί μειώνει τον αριθμό των συγκρίσεων

Πως λειτουργεί ο αλγόριθμος Shift-Or

Έστω για κάθε χαρακτήρα c στο αλφάβητο Σ το διάνυσμα Sc μεγέθους m, που αποθηκεύει τις εμφανίσεις του c μέσα στο πρότυπο P. Ο πίνακας R mxn είναι bit array, όπου R[i,j] είναι 0 αν και μόνο αν οι πρώτοι i χαρακτήρες του P ταιριάζουν με τους i χαρακτήρες που τελειώνουν στον j-οστό χαρακτήρα του T
R(j+1) = Shift(R(j)) OR ST[j+1] :για κάθε χαρακτήρα κάνουμε shift αριστερά μια θέση και OR με το αντίστοιχο bitmask

Χρονική πολυπλοκότητα: O(n+m)

Αποδοτικός για σχετικά μικρό αλφάβητο και μικρό P (λίγες και φθηνές πράξεις)
μειονέκτημα: αποδοτικός μόνο για n < logn

Πως λειτουργεί ο αλγόριθμος Rabin-Karp

Αναζήτηση του προτύπου με βάση ένα hash function

preprocessing: υπολογίζουμε το hash value του P και το hash value του πρώτου substring του T με μήκος όσο το P

αναζήτηση: για κάθε θέση συγκρίνουμε το hash value του τρέχοντος substring με το hash value του P. Αν οι τιμές ταιριάζουν, κάνουμε exact matching στη θέση αυτή για να επιβεβαιώσουμε ότι υπάρχει ταίριασμα. Συνεχίζουμε με τον υπολογισμό του hash value για τον επόμενο χαρακτήρα. Αν οι τιμές δεν ταιριάζουν, μετακινούμαστε 1 χαρακτήρα δεξιά και επαναυπολογίζουμε

Χρονική πολυπλοκότητα: O(n+m)
στο worst case O(n*m) λόγω συγκρούσεων

Αν η μνήμη είναι πολύ μικρή ποιόν από τους παραπάνω αλγορίθμους πρέπει να χρησιμοποιήσουμε;

KMP γιατί έχει καλύτερη πολυπλοκότητα χώρου

Οι παραπάνω αλγόριθμοι δουλεύουν για παραπάνω από 1 κείμενο;

ΝΑΙ

KMP: τρέχουμε ξεχωριστά για κάθε κείμενο με preprocessing 1 φορά

Shift-Or: το ίδιο με επαναχρησιμοποίηση των bitmasks

Rabin-Karp: το ίδιο με υπολογισμό των hash values ξεχωριστά για κάθε κείμενο

Boyer-Moore: το ίδιο με preprocessing 1 φορά

Η επιλογή ποιόν θα χρησιμοποιήσουμε εξαρτάται από το μέγεθος του pattern, το αλφάβητο και τη διαθέσιμη μνήμη

Τι είναι το Αυτόματο Aho-Corasick και πως ορίζεται

Επέκταση του KMP για πολλαπλά πρότυπα με χρήση δέντρου και συνάρτησης failure function

Χειρίζεται πολλαπλά πρότυπα παράλληλα μέσω FSM

Κατασκευάζει prefix TRIE από τα πρότυπα. Κάθε κόμβος αναπαριστά μια κατάσταση, κάθε ακμή μια μετάβαση.

Κατασκευάζει failure function κάνοντας BFS αναζήτηση στο δέντρο και εντοπίζοντας τα failure links. Για κάθε κόμβο το failure link δείχνει το longest proper suffix που είναι και prefix στο δέντρο και επιτρέπει τη μετάβαση του αυτομάτου σε άλλη κατάσταση όταν υπάρχει mismatch

Κατασκευάζει output function αντιστοιχίζοντας πρότυπα με τις καταστάσεις στο TRIE όπου καταλήγουν. Αν το αυτόματο φτάσει σε κατάσταση που ανταποκρίνεται σε πρότυπο, το output function ανακτά το pattern

goto(s,a) = s' : το αυτόματο μεταπηδά στην κατάσταση s' και ο επόμενος χαρακτήρας της ακολουθίας διαβάζεται στην είσοδο

g(s,a) = fail : το αυτόματο μεταβαίνει στην κατάσταση s' = f(s) σύμφωνα με το failure function. Η αναζήτηση συνεχίζεται με τρέχουσα κατάσταση την s' και σύμβολο εισόδου τον χαρακτήρα a που ήδη έχει διαβαστεί στην είσοδο

χρόνος O(n) για αναζήτηση

Πως γίνεται το exact pattern matching με don't care ('*') bits;

Είναι εφαρμογή του Aho-Corasick

Χωρίζουμε το P σε substrings που δεν περιέχουν τον don't care χαρακτήρα και κάθε substring αντιμετωπίζεται σαν ξεχωριστό pattern

Φτιάχνουμε το αυτόματο Aho-Corasick χρησιμοποιώντας τα substrings

Ψάχνουμε για εμφανίσεις κάθε substring μέσα στο T

Ελέγχουμε αν τα matches που βρέθηκαν μπορούν να συνδυαστούν ώστε να καλύπτουν ολόκληρο το pattern

Αλγόριθμος:
Έστω C ένας πίνακας ακεραίων μήκους n αρχικοποιημένος με μηδενικά, P = {P1, P2,…, Pn} το σύνολο των υποσυμβολοσειρών του P που δεν περιέχουν χαρακτήρες μπαλαντερ και I1, I2,…, In οι αρχικές θέσεις στο P καθεμιάς από τις από τις υποσυμβολοσειρές
Χρησιμοποιώντας τον αλγόριθμο Aho-Corasick βρείτε για κάθε συμβολοσειρά Pi στο P όλες τις θέσεις του Pi στο T. Για κάθε θέση j του Pi στο T, αυξήστε τον αριθμό στο κελί j- Ii + 1 του C κατά 1
Σαρώστε τον πίνακα C για την εύρεση κελιών με τιμή k. Υπάρχει εμφάνιση του P στο T ξεκινώντας από τη θέση k αν και μόνο αν C(P) = k

Πως γίνεται το two-dimensional pattern matching;

Είναι εφαρμογή του Aho-Corasick

Το P είναι ουσιαστικά υποπίνακας του πίνακα T

'Εστω T κείμενο δύο διαστάσεων με n = n1 x n2 κελιά και P πρότυπο δύο διαστάσεων με m = m1 x m2 κελιά
1η φάση: αναζητούμε όλες τις εμφανίσεις καθεμιάς από τις σειρές του P μεταξύ σειρών του T. Για να το κάνουμε αυτό προσθέτουμε έναν δείκτη τέλους γραμμής (κάποιον χαρακτήρα που δεν υπάρχει στο αλφάβητο) σε κάθε γραμμή του T και συνενώνουμε αυτές τις γραμμές σε μια συμβολοσειρά T' μήκους O(n)
Αντιμετωπίζοντας κάθε γραμμή του P ως ξεχωριστό μοτίβο, χρησιμοποιούμε τον αλγόριθμο Aho-Corasick για να αναζητήσουμε όλες τις εμφανίσεις στο T' οποιασδήποτε σειράς του P
Κάθε φορά που εντοπίζεται εμφάνιση της γραμμής i του P ξεκινώντας από τη θέση (p,q) του T, γράφουμε τον αριθμό i στη θέση (p,q) ενός άλλου πίνακα M ίδιων διαστάσεων με το T

Η πρώτη φάση προσδιορίζει όλες τις εμφανίσεις πλήρων σειρών P και παίρνει χρόνο O(n+m)
2η φάση: Σαρώνουμε κάθε στήλη του M αναζητώντας μια εμφάνιση της συμβολοσειράς 1, 2,…, m1 σε διαδοχικά κελιά μιας στήλης. Αυτό δίνει λύση O(n*k + m) αν χρησιμοποιηθεί αλγόριθμος pattern matching γραμμικού χρόνου (πχ: Z-preprocessing, KMP) σε κάθε στήλη, ανεξάρτητα από το μέγεθος του αλφαβήτου

Σε περίπτωση που οι σειρές του P δεν είναι όλες διακριτές, αρκεί να εντοπίσουμε όλες τις πανομοιότυπες και να τους δώσουμε κοινή ετικέτα

Τι είναι Suffix Tree

Δέντρο επιθεμάτων για μια συμβολοσειρά n χαρακτήρων είναι ένα κατευθυνόμενο δέντρο με ρίζα που έχει ακριβώς n φύλλα αριθμημένα από 1 έως n. Κάθε εσωτερικός κόμβος εκτός από τη ρίζα έχει τουλάχιστον 2 παιδιά και κάθε ακμή σημειώνεται με μια μη-κενή υποσυμβολοσειρά. Δεν επιτρέπεται από έναν κόμβο να υπάρχουν δύο ακμές που ξεκινούν με τον ίδιο χαρακτήρα. Για οποιοδήποτε φύλλο i η συνένωση των ετικετών των ακμών στη διαδρομή από τη ρίζα σε αυτό είναι ίση με το επίθεμα της συμβολοσειράς που ξεκινά από τη θέση i.

Αποθηκεύει όλα τα δυνατά επιθέματα συμβολοσειράς S

Στα φύλλα συνηθίζεται να αποθηκεύεται αριθμός που δείχνε σε ποιο index της συμβολοσειράς ξεκινά το substring

suffix links: δείχνουν που μπορούμε να πάμε από κάθε κόμβο ---> επιταχύνουν το ψάξιμο

Είναι συμπαγές TRIE, δηλαδή σε κάθε ακμή αντί για έναν χαρακτήρα όπως στο TRIE μπορεί να έχει και υποσυμβολοσειρά

Χρόνος για κατασκευή: O(n^2), εκτός αν χρησιμοποιηθεί κάποιος αλγόριθμος
Χώρος για κατασκευή: O(n), υποθέτοντας ότι έχουμε άπειρη μνήμη

Χρόνος για αναζήτηση του P: O(m)

Πως κάνουμε pattern matching με Suffix Tree

Για κείμενο T, πρότυπο P:
Κατασκευάζουμε suffix tree για το T
Ξεκινώντας από τη ρίζα, διαπερνάμε το δέντρο χαρακτήρα-χαρακτήρα μέσω των ακμών
Για κάθε χαρακτήρα στο P, ακολουθούμε την ακμή που αντιστοιχεί σε αυτόν. Αν η ακμή έχει ετικέτα ένα substring, συνεχίζουμε με τους χαρακτήρες αυτού του substring
Αν καταφέρουμε να διαπεράσουμε το δέντρο και να καταλήξουμε σε κόμβο ή φύλλο, τότε το P υπάρχει στο T και όλα τα φύλλα του υποδέντρου που ξεκινούν από τον κόμβο αυτόν αντιπροσωπεύουν τις θέσεις εκκίνησης του P στο T.

Χρόνος κατασκευής: O(n)
Χρόνος αναζήτησης: O(m)
Χρόνος εύρεσης όλων των εμφανίσεων k του P στο T (πχ με DFS): O(k)

Συνολικά O(n+m+k)

ΠΡΟΣΟΧΗ: αν ψάχνουμε για suffixes μόνο, τότε match έχουμε μόνο αν φτάσαμε σε φύλλο ή στο σύμβολο τερματισμού ($)

Τι κάνουν οι αλγόριθμοι Weiner, McCreight, Ukonnen, Farach;

Κατασκευάζουν suffix tree για συμβολοσειρά S

Weiner: ξεκινά με το μικρότερο επίθεμα και προσθέτει επιθέματα στο δέντρο ένα-ένα
O(n^2)

Ukonnen: ξεκινά με άδειο δέντρο(μόνο ρίζα). Για κάθε χαρακτήρα στο S επεκτείνει το δέντρο ώστε να περιέχει τα επιθέματα που καταλήγουν στον χαρακτήρα αυτόν
O(n)

McCreight: ξεκινά με τη ρίζα και τον πρώτο χαρακτήρα του S. Για κάθε επόμενο χαρακτήρα επεκτείνει το δέντρο με τα επιθέματα που δημιουργούνται(ξεκινούν από) τον χαρακτήρα αυτόν
O(n)

Farach: χωρίζει το S σε segments και φτιάχνει suffix tree για κάθε segment. Χρησιμοποιεί suffix links για να τα ενώσει σε ένα δέντρο
O(n)

Τι είναι generalized suffix tree (GST);

Δέντρο επιθεμάτων που αποθηκεύει όλα τα δυνατά επιθέματα ενός συνόλου συμβολοσειρών {S1, S2,…, Sn}

Χρήσιμα για τον εντοπισμό κοινών υποσυμβολοσειρών ανάμεσα στις διαφορετικές συμβολοσειρές και για τον εντοπισμό επαναλήψεων

Χρόνος κατασκευής: O(n1 + n2 +…+ nk) εκτός αν χρησιμοποιηθεί κάποιος από τους αλγορίθμους παραπάνω

Πως γίνεται exact pattern matching με πολλαπλά πρότυπα (multiple pattern matching);

Κατασκευή generalized suffix tree

Ανάγουμε τα διαφορετικά πρότυπα σε 1 με αυτόματο Aho-Corasick

Ξεκινώντας από τη ρίζα, αναζητούμε διαδοχικά το σύνολο των προτύπων {P1, P2,…, Pρ}

Κόστος O(n + m + kρ), όπου kρ το πλήθος των εμφανίσεων των προτύπων στο T

Μπορεί να έχουμε πρόβλημα όταν κάποιο από τα πρότυπα είναι substring κάποιου άλλου

Πως βρίσκουμε το Longest Common Subsequence (LCS) δύο συμβολοσειρών με suffix tree;

Κατασκευάζουμε generalized suffix tree για τις ακολουθίες εισόδου S1, S2 αφού πρώτα συνενωθούν σε μια ενιαία συμβολοσειρά όπου ανάμεσα στις συμβολοσειρές εισόδου θα παρεμβάλλεται κάποιος χαρακτήρας (πχ: $) που δεν ανήκει σε καμία από τις 2

Σημειώνουμε κάθε εσωτερικό κόμβο κ του δέντρου με “1” ή “2” αν εμπεριέχει στο υποδέντρο του, u, κάποιο φύλλο που αναπαριστά κάποιο επίθεμα της S1 ή της S2 αντίστοιχα

Η ετικέτα μονοπατιού (path label) κάθε εσωτερικού κόμβου που σημειώνεται ταυτόχρονα με “1” και “2”, αποτελεί μια κοινή υποσυμβολοσειρά των S1, S2

Σάρωμα του GST μέχρι να βρεθεί ο βαθύτερος εσωτερικός κόμβος που έχει απογόνους φύλλα και από το S1 και από το S2. Το μονοπάτι από τη ρίζα μέχρι αυτόν τον κόμβο αναπαριστά το LCS

Πολυπλοκότητα χρόνου: O(n)

Τι ορίζουμε maximal pairs;

maximal pair (i, j) για δύο συμβολοσειρές S1, S2 είναι ένα ζεύγος ετικετών τέτοιων ώστε τα S[i…k] και S[j…k] είναι τα longest common substrings που ξεκινούν στις θέσεις i και j αντίστοιχα Είναι maximal γιατί δεν μπορούν να επεκταθούν προς τα αριστερά ή προς τα δεξιά χωρίς να πάψουν να είναι ίδια. Είναι δηλαδή οι μεγαλύτερες επαναλαμβανόμενες υποσυμβολοσειρές της συμβολοσειράς S1$S2 που δεν επικαλύπτονται

Τι είναι παλίνδρομο

συμβολοσειρά που διαβάζεται το ίδιο από αριστερά και από δεξιά

Τι είναι συμπληρωματικό παλίνδρομο

προκύπτει από την αντικατάσταση όλων των χαρακτήρων από την αρχή μέχρι τη μέση με τις συμπληρωματικές βάσεις

Τι είναι approximate pattern matching

Αναζητούμε τις θέσεις εμφάνισης του προτύπου P μέσα στην ακολουθία T, επιτρέποντας errors και mismatches

edit (Levenshtein) distance: για 2 συμβολοσειρές ορίζουμε το ελάχιστο πλήθος πράξεων μετασχηματισμού που απαιτούνται για να μετασχηματιστεί η πρώτη συμβολοσειρά στη δεύτερη

edit transcript: η ακολουθία των πράξεων μετασχηματισμού για να μετασχηματιστεί η πρώτη συμβολοσειρά στη δεύτερη. Οι πράξεις είναι:
insert(I)
delete(D)
replace(R)
match(M) (δεν είναι ακριβώς πράξη δλδ δεν αλλάζει τη συμβολοσειρά, απλά δείχνει ταίριασμα, ΔΕΝ λαμβάνεται υπόψιν στο edit distance)

sequence alignment: τοποθετεί τη μια συμβολοσειρά κάτω από την άλλη με χρήση κενών έτσι ώστε οι κοινοί χαρακτήρες να τοποθετούνται στις ίδιες θέσεις

Γιατί είναι ίδια τα sequence alignment και edit distance;

Γιατί κάθε πράξη μετασχηματισμού αντιστοιχεί απευθείας σε μια αλλαγή στη στοίχιση

Τι είναι δυναμικός προγραμματισμός για τη στοίχιση δύο ακολουθιών

Έστω 2 ακολουθίες S1, S2. Θα συμβολίζουμε ως D(i,j) την απόσταση μετασχηματισμού μεταξύ των προθεμάτων S1[1…i] και S2[1…j], δηλαδή τον ελάχιστο αριθμό πράξεων μετασχηματισμού που απαιτούνται για να μετασχηματίσουμε τους i πρώτους χαρακτήρες της S1 στους j πρώτους χαρακτήρες της S2

Σχέση αναδρομής:
D(i, j) = min[D(i-1, j) + 1, D(i, j-1) + 1, D(i-1, j-1) + t(i, j)]

D(i-1, j) + 1 : πρέπει να διαγράψουμε το χαρακτήρα S1[i]
D(i, j-1) + 1 : πρέπει να ενθέσουμε το χαρακτήρα S2[j]
D(i-1, j-1) + 1 : για να μετασχηματίσουμε το χαρακτήρα S1[i] στον S2[j] πρέπει να αντικαταστήσουμε το χαρακτήρα S1[i] με το χαρακτήρα S2[j] (t(i, j) = 1 αν S1[i] != S2[j])
D(i-1, j-1) : έχουμε ταίριασμα (t(i, j) = 0 αν S1[i] = S2[j])

Κατασκευάζουμε πίνακα δυναμικού προγραμματισμού (n+1) x (m+1) και ψάχνουμε το ελάχιστο στον πίνακα

- με μήκη
- με βάρη

Κάθε κελί του πίνακα δυναμικού προγραμματισμού περιέχει το edit distance μεταξύ των χαρακτήρων που αντιστοιχούν στη θέση αυτή της στοίχισης. Βάζουμε στις 2 πλευρές του πίνακα τις 2 συμβολοσειρές που θέλουμε να συγκρίνουμε και στην αρχή του πίνακα μια επιπλέον γραμμή και μια επιπλέον στήλη (για να συμπεριλάβουμε και την περίπτωση μια από τις συμβολοσειρές να είναι η κενή.
Η τιμή για κάθε νέο κελί του πίνακα (κάτω-δεξιά) είναι το min των τιμών στα κελιά πάνω-αριστερά, κάτω-αριστερά και πάνω-δεξιά(κοιτάμε 4-αδα κελιών σε κάθε βήμα) + 1, εκτός αν έχουμε match. Τότε η τιμή του νέου κελιού είναι ίση με του πάνω-αριστερά.
Το πάνω-αριστερά κελί αναπαριστά το replace
Το πάνω δεξιά αναπαριστά το insert
Το κάτω-αριστερά κελί αναπαριστά το delete
Μόλις συμπληρωθεί όλος ο πίνακας ξεκινάμε από το τέρμα κάτω δεξιά κελί D(n,m) και προχωράμε προς τα πίσω στον πίνακα ακολουθώντας τα βέλη μέχρι κάποιο κελί-στόχο(traceback). Το άθροισμα των τιμών της διαδρομής είναι το minimum edit distance και η ακολουθία πράξεων (μονοπάτι) το edit transcript

Αν υπάρχουν πολλαπλά μονοπάτια τότε υπάρχουν πολλαπλές (βέλτιστες) στοιχίσεις με ίδιο κόστος

Πολυπλοκότητα:
- αρχικοποίηση: O(n) + O(m)
- σχέση αναδρομής: O(n*m)
- δείκτες οπισθοχώρησης: O(n+m)

Συνολικά; O(n^2)

weighted edit distance: κάθε πράξη μετασχηματισμού έχει ένα κόστος/βάρος(βέλτιστο = ελάχιστο)
insert ή delete: d
replace: r
match: m
Με βάση τα βάρη αυτά αλλάζει ο τύπος για τη σχέση αναδρομής:

D(i, j) = min[D(i-1, j) + d, D(i, j-1) + d, D(i-1, j-1) + t(i, j)],
t(i, j) = r αν S1[i] != S2[j]
t(i, j) = e αν S1[i] = S2[j]

alphabet weighted: βέλτιστο = μέγιστο και το βάρος εξαρτάται από τον χαρακτήρα του αλφαβήτου

Ποια είναι η διαφορά μεταξύ global και local alignment

ολική στοίχιση(global alignment): ψάχνουμε το καλύτερο ταίριασμα μεταξύ δύο συμβολοσειρών σε όλο το μήκος τους. Χρήσιμο όταν οι συμβολοσειρές έχουν παρόμοιο μήκος
-> Needleman-Wunsch:

στο traceback προχωράμε προς τα πίσω στον πίνακα μέχρι να φτάσουμε στο κελί D(0,0)
gap penalty: η ποινή για την εισαγωγή ενός κενού με εισαγωγή ή διαγραφή(συνυπολογίζεται στον τύπο για το σκορ)
Κάθε κενό πρέπει να βρίσκεται απέναντι από χαρακτήρα

τοπική στοίχιση(local alignment): ψάχνουμε το καλύτερο ταίριασμα τοπικά. Χρήσιμο όταν οι συμβολοσειρές έχουν διαφορετικά μήκη ή όταν υπάρχει ομοιότητα μόνο σε συγκεκριμένα τμήματά τους
-> Smith-Waterman:
στο traceback προχωράμε προς τα πίσω **από την μεγαλύτερη τιμή του πίνακα** (ΟΧΙ από την κάτω δεξιά γωνία) μέχρι να φτάσουμε στο πρώτο κελί με σκορ 0 ή μέχρι να φτάσουμε σε όριο του πίνακα. Αν φτάσουμε σε παραπάνω από 1 μηδενικά τότε κάθε μια από τις διαδρομές είναι μια τοπική στοίχιση

Είναι ίδιο το edit distance με την απόσταση Hamming;

ΟΧΙ

Το hamming distance είναι το ίδιο αλλά μόνο για συμβολοσειρές ίδιου μήκους

Πως προκύπτει το LCS από ολική στοίχιση

αν έχουμε βάρος για ταίριασμα 1 και για όλες τις άλλες πράξεις 0

Πως βρίσκουμε το LCS με δυναμικό προγραμματισμό

κάνουμε traceback από το D(0,0) μέχρι το D(n,m) αντίστροφη πορεία???

Πως μπορούν να υλοποιηθούν τα συνεχόμενα κενά στο global alignment;

Βάζοντας διαφορετική ποινή όταν δημιουργείται νέο κενό και όταν επεκτείνεται ένα κενό που ήδη υπάρχει
δυναμικός προγραμματισμός αλλά με 1 επιπλέον πίνακα για τα κενά
Είναι καλύτερη προσέγγιση από το να θεωρούμε ότι κάθε κενό έχει το ίδιο βάρος ανεξάρτητα από το μήκος

τύπος για τις ποινές κενού: $-(\rho + \sigma * x)$
αρχικά το ρ έχει μεγάλη τιμή και κάθε φορά που συναντάμε κενό μειώνεται

συνάρτηση που έχει μέσα τα κενά και κάνουμε ολική στοίχιση

Τι είναι PAM και BLOSUM

πίνακες αντικατάστασης

PAM: χρησιμοποιείται για στοίχιση ακολουθιών από αμινοξέα. Τα στοιχεία του προκύπτουν συνήθως από εξελικτικά μοντέλα Το σκορ δείχνει ομοιότητα μεταξύ πρωτεϊνών και την πιθανότητα αντικαταστάσεων (substitutions)
PAM250: οι γραμμές και οι στήλες αναπαριστούν τα 20 διαφορετικά αμινοξέα

BLOSUM: χρησιμοποιείται για τον εντοπισμό conserved regions στις πρωτεΐνες, στοίχιση πρωτεϊνών αναζητήσεις σε βάσεις δεδομένων και ανάλυση. Τα στοιχεία του προκύπτουν από περιοχές(blocks) οικογενειών πρωτεϊνών
BLOSUM62: για συμβολοσειρές με 62% ομοιότητα

Όσο μεγαλώνει ο δείκτης PAM τόσο μικραίνει ο δείκτης BLOSUM και τόσο πιο μακριά βρίσκονται τα στοιχεία που συγκρίνονται

Στη διαγώνιο πάντα θετικές τιμές και στις ασυμφωνίες αρνητικές η 0

Τι είναι maximal segment pairs

Ένα ζεύγος τμημάτων είναι μέγιστο αν έχει την καλύτερη βαθμολογία ανάμεσα σε όλα τα ζεύγη τμημάτων
Ένα ζεύγος τμημάτων είναι τοπικά μέγιστο αν η βαθμολογία του δεν μπορεί να βελτιωθεί με επέκταση ή περικοπή
Στο suffix tree τα maximal pairs α βρίσκονται σε διαφορετικά υποδέντρα

Ποιες είναι οι βασικές προσεγγίσεις για indexing

κατάλληλες για το RAM μοντέλο, αν έχουμε δευτερεύουσα μνήμη είναι καλύτερα τα δέντρα

- k-grams: όπως k-mers
- direct indexing
- suffix arrays: διατεταγμένη λίστα όλων των επιθεμάτων, μπορεί να κατασκευαστεί τοποθετώντας τα φύλλα του suffix tree από αριστερά προς τα δεξιά μετά από DFS προσπέλαση του suffix tree
- vector-space indexing: κάθε κείμενο και κάθε ερώτημα αναπαρίσταται ως διάνυσμα m όρων, όπου m ο αριθμός των μοναδικών όρων της συλλογής. Κάθε όρος έχει ένα βάρος. Πιο γνωστή μέθοδος απόδοσης βάρους: TF-IDF, έχει μεγάλη τιμή για έναν όρο μέσα σε κείμενο όταν ο όρος εμφανίζεται συχνά στο κείμενο. TF: συχνότητα εμφάνισης του όρου, IDF: αντίστροφη συχνότητα

Πως μπορούμε να κάνουμε approximate matching με μεθόδους που είναι για exact pattern matching;

- Shift-Or με κατώφλι για τα λάθη
- δυναμικός προγραμματισμός
- τροποποιημένο αυτόματο Aho-Corasick με μεταβάσεις για mismatches, insertions και deletions
- k-mer indexing σε συνδυασμό με το edit distance για τον εντοπισμό προτύπων που είναι όμοια στον αριθμό των replace αλλά όχι insertions, deletions
- suffix tree, suffix array

Τι είναι πολλαπλή στοίχιση (multiple alignment)

Στοίχιση για περισσότερες από 2 συμβολοσειρές

Για κάθε συμβολοσειρά μια διάσταση οπότε αντί για πίνακα έχουμε k-διάστατο πλέγμα δυναμικού προγραμματισμού (πχ: κύβος Manhattan)

Χρόνος εκθετικός: O(2^k * n^k), χρειάζεται βελτίωση ---------->

ClustalW: εργαλείο
κάνει στοίχιση των συμβολοσειρών ανά δύο, φτιάχνει ένα distance matrix με σκορ ομοιότητας μεταξύ των συμβολοσειρών και κατασκευάζει με neighborjoin φυλλογεννητικό δέντρο-οδηγό που έχει τις συμβολοσειρές στους κόμβους. Συγχωνεύει συμβολοσειρές με κοινό προφιλ
χρόνος: O(k^2 *n^2)

Center-Star: αλγόριθμος
Αρχικά συγκρίνουμε τις συμβολοσειρές ανα 2 και βάζουμε αυτήν που έχει τις περισσότερες ομοιότητες με όλες τις άλλες σαν «κέντρο» και τις υπόλοιπες γύρω από αυτήν με σχηματισμό «άστρου». Στη συνέχεια προσαρμόζει τις αποστάσεις και τις ποινές για τα κενά ώστε η συνολική στοίχιση να είναι βέλτιστη
χρόνος: O((k-1) * n^2)

Τι είναι FASTA και BLAST

αλγόριθμοι πολλαπλής στοίχισης

BLAST: συντηρείται από το NCBI

χρήση για σύγκριση νουκλεοτιδικών ή πρωτεϊνικών αλληλουχιών σε μεγάλες βάσεις δεδομένων με βάση λέξεις-κλειδιά. Όταν συναντά mismatch επεκτείνει τις διαγωνίους με κενά μέχρι η τιμή να γίνει μικρότερη από προκαθορισμένο κατώφλι
ο πίνακας είναι μήτρα βαθμολόγησης
εντοπίζει κοινές υποακολουθίες ίδιου μήκους (segment pairs)
μπορεί να επεκταθεί για να εντοπίζει maximal segment pairs
γρήγορος, αποδοτικός

FASTA: βρίσκει μικρές λέξεις(words ή k-tuples) που εμφανίζονται και στις δύο ακολουθίες αναζητά λέξεις συγκεκριμένου μήκους(hot spots) στον πίνακα δυναμικού προγραμματισμού και εντοπίζει τις καλύτερες τροχιές(runs). Συνδυάζει τις καλύτερες υπο-στοιχίσεις και παράγει το βέλτιστο μονοπάτι
ο πίνακας είναι μήτρα κουκκίδων
Ταυτόχρονα είναι και format για την αναπαράσταση συμβολοσειρών σε προγράμματα σύγκρισής τους
FASTQ: δείχνει και quality scores

Μπορούμε από μια πολλαπλή στοίχιση να πάρουμε πολλές απλές στοιχίσεις;

ΝΑΙ

Μπορούμε από πολλές απλές στοιχίσεις να πάρουμε μια πολλαπλή;

ΟΧΙ πάντα (δεν θα είναι βέλτιστη)

Σε τι διαφέρει η συσταδοποίηση από την κατηγοριοποίηση;

στην συσταδοποίηση οι κατηγορίες δεν είναι προκαθορισμένες