Cipherkey – Λύση με hashing

Ο σκοπός αυτής της λύσης είναι η βελτίωση της πολυπλοκότητας της αρχικής λύσης $O(N*M)$, όπου N, M τα μεγέθη των S,T αντίστοιχα. Αρχικά ξεκινάμε με μια παρατήρηση, εφ' όσον θέλουμε να μετρήσουμε τον αριθμό των substrings του S που ταυτίζονται με το T, με εξαίρεση ίσως έναν χαρακτήρα, μπορούμε απλά να παράξουμε όλες τις δυνατές περιπτώσεις που προκύπτουν, αν αλλάξουμε απο το T το πολύ έναν χαρακτήρα και έπειτα να ψάξουμε γι' αυτά μέσα στο S. Δηλαδή να φτιάξουμε τα strings που προκύπτουν αν βάλουμε στην πρώτη θέση του T τον χαρακτήρα a,b,c, … και ούτω καθεξής.

Ο αριθμός των strings που προκύπτουν είναι πολυνωνυμικός ως προς M και συγκεκριμένα είναι |Σ| * M, όπου Σ η αλφάβητος που χρησιμοποιούμε για την παραγωγή των strings. Τώρα, εφ' όσον ξέρουμε ότι οποιοδήποτε substring που θέλουμε να προσμετρήσουμε στην λύση μας είναι ένα απ' αυτά, μπορούμε απλά να μετρήσουμε πόσες φορές εμφανίζεται το κάθε ένα μέσα στο S. Η πολυπλοκότητα που προκύπτει από αυτή την εργασία είναι $O(\Sigma*N*M^2)$ κάτι το οποίο φαίνεται άχρηστο αρχικά εφ' όσον η πολυπλοκότητα ανέβηκε από αυτήν της brute-force, ωστόσο έχουμε πετύχει κάτι αρκετά χρήσιμο, έχουμε πλέον ανάγει το αρχικό μας πρόβλημα σε ένα πολύ γνωστό πρόβλημα γνωστό ως Pattern Matching.

Για την επίλυση του παραπάνω προβλήματος θα χρησιμοποιήσουμε μια μέθοδο που είναι γνωστή και ως αλγόριθμος Rabin-Karp ή Rolling Hashing. Συνοπτικά, με την μέθοδο αυτή χειριζόμαστε κάθε string ως έναν |Σ|-αδικό αριθμό (στην προκειμένη περίπτωση 26-αδικό) και τον μετατρέπουμε σε δεκαδικό. Με αυτόν τον τρόπο πετυχαίνουμε σύγκριση 2 strings σε O(1), δηλαδή όσο χρόνο παίρνει και η σύγκριση ακεραίων. Είναι δηλαδή:

$H(T[1...M])=(26^{M-1}*T[1]+26^{M-2}*T[2]+...+26*T[M-1]+T[M])\,mod\,P$, όπου M το μέγεθος του T και P ένας αρκετά μεγάλος πρώτος αριθμός, ώστε να αποφύγουμε τόσο το overfloating όσο και τα collisions.

Περισσότερα σχετικά με αυτή την τεχική υπάρχουν σε αυτό το άρθρο. Πλέον, με αυτόν τον τρόπο, μπορούμε αντί να κρατάμε τα αυτούσια strings που προκύπτουν αν αλλάξουμε έναν χαρακτήρα του T, να κρατάμε τα hashes αυτών σε μια δομή δεδομένων, απλά αλλάζοντας κάθε φορά κάθε χαρακτήρα σε όλα τα δυνατά γράμματα της αλφαβήτου. Αφού το κάνουμε αυτό, μπορούμε (με την τεχνική του rolling hash), να παράξουμε το hash κάθε substring μεγέθους M του S, σε γραμμικό χρόνο συνολικά και να βλέπουμε κάθε φορα αν το αντίστοιχο substring υπάρχει στην δομή μας. Η δομή αυτή, μπορεί να είναι στην προκειμένη περίπτωση ένα set, το οποίο μας επιτρέπει εισαγωγή στοιχείου καθώς και αναζήτηση σε λογαριθμικό χρόνο. Συνεπώς η πολυπλοκότητα της λύσης μας είναι $O(\Sigma*M^2*logM+NlogM)$ το οποίο πλησιάζει σε μια επαρκή λύση για τα δεδομένα του προβλήματος.

Για την τελική μας λύση, κάνουμε χρήση ορισμένων ιδιοτήτων της αριθμητικής υπολοίπων. Στην πολυπλοκότητα της λύσης μας υπάρχει ένας τετραγωνικός όρος, ο οποίος προκύπτει από το γεγονός ότι για κάθε αλλαγή γράμματος που κάνουμε στο T, υπολογίζουμε το hash από την αρχή, πράγμα που είναι γραμμικό ως προς το M. Λαμβάνοντας υπ' όψην τις παρακάτων ιδιότητες των υπολοίπων:

$(\alpha\,mod\,P+\beta\,mod\,P)\,mod\,P=(\alpha+\beta)\,mod\,P$
$(\alpha-\beta)\,mod\,P=(\alpha\,mod\,P-\beta\,mod\,P+P)\,mod\,P \quad , \quad \alpha,\beta,P\in Z$

Μπορούμε να ρίξουμε την πολυπλοκότητα της λύσης μας.

Για να εκμεταλλευτούμε αυτές τις ιδιότητες, αρχικά υπολογίζουμε το αρχικό hash του T:

$$H(T[1...M])=(26^{M-1}*T[1]+26^{M-2}*T[2]+...+26*T[M-1]+T[M])\,mod\,P$$

τώρα έστω ότι θέλουμε να αλλάξουμε τον i – οστό χαρακτήρα του T στο k – οστό γράμμα της αλφαβήτου (το 'a' είναι το πρώτο, το 'b' το δεύτερο κτλ.). Στην προηγούμενη λύση μας, αυτό που κάναμε ήταν να αλλάξουμε τον αντίστοιχο χαρακτήρα του T και να πάμε να υπολογίσουμε το hash από την αρχή, όμως κάνοντας χρήση των ιδιοτήτων των υπολοίπων έχουμε:

$$(H'(T[1...M])-H(T[1...M]))\,mod\,P=(26^{(M-i)}*k-26^{(M-i)}*T[i])\,mod\,P\Rightarrow$$

$$H'(T[1...M])=(((H(T[1...M])-26^{(M-i)}*T[i]+P)\,mod\,P)+26^{(M-i)}*k)\,mod\,P$$

Έτσι, έχουμε πετύχει τον υπολογισμό του καινούριου hash από το αρχικό μόλις σε O(1), δεδομένου βέβαια ότι έχουμε υπολογίσει ήδη τις απαραίτητες πληροφορίες όπως δυνάμεις κτλ.

Συνεπώς, πλέον μπορούμε να παράξουμε γρήγορα τα hashes και η πολυπλοκότητα της λύσης μας γίνεται $O(\Sigma*M*logM+N\log M)$ που είναι επαρκής για το 100% των testcases του προβλήματος.