ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Εισαγωγή στη Βιοπληροφορική

Ενότητα 8: Δομές Δεικτοδότησης

Μακρής Χρήστος, Τσακαλίδης Αθανάσιος,
Περδικούρη Αικατερίνη

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Σκοποί ενότητας

- Ο σκοπός της ενότητας είναι η παρουσίαση των δομών δεικτοδότησης (εκτός από suffix trees που έχουν ήδη παρουσιαστεί)

Βασικές Βιβλιογραφικές Πηγές στις οποίες βασίζονται οι διαφάνειες

- Ferragina, Roberto Grossi: The String B-tree: A New Data Structure for String Search in External Memory and Its Applications. J. ACM 46(2): 236-280 (1999)
- Manber, Eugene W. Myers: Suffix Arrays: A New Method for On-Line String Searches. SIAM J. Comput. 22(5): 935-948 (1993)
- I. Witten, A. Moffat, T. Bell, Managing gigabytes: compressing and indexing documents and images, Morgan Kaufmann Publishers, 1999.
- Dan Gusfield , Algorithms on Strings, Trees and Sequences,, Cambridge University Press, 10th edition 2007

Περιεχόμενα ενότητας

- Πλαίσιο μελέτης
- Μέθοδοι και συγκρίσεις μεθόδων
- Μοντέλο μνήμης
- Πολυπλοκότητα
- Αλγόριθμοι

Δομές Δεικτοδότησης

Πλαίσιο Μελέτης

1. Τύπος συμβολοσειρών που μπορούν να δεικτοδοτηθούν:
   - δομές που δεικτοδοτούν συμβολοσειρές/κείμενα φυσικού λόγου (linguistic texts)
   - δομές πλήρους δεικτοδότησης

   Στην πρώτη κατηγορία: ανεστραμμένα αρχεία (inverted files), αρχεία υπογραφών (signature files), bitmaps

   Στην δεύτερη κατηγορία: δέντρα επιθεμάτων (suffix trees), πίνακες επιθεμάτων (suffix arrays), άκυκλοι κατευθυνόμενοι γράφοι λέξεων (DAWG και cDAWG).

2. Μοντέλο υπολογισμού: οι ανωτέρω δομές είναι κατάλληλες για RAM, για τη δευτερεύουσα μνήμη πιο κατάλληλη η δομή: string b-tree.

Linguistic Text Indexing

Χρήση: Σε μεγάλες συλλογές κειμένων. Οργάνωση πληροφορίας σε μορφή περιεχομένων (σε ποια κείμενα βρίσκεται κάθε όρος). Απάντηση σε ερωτήματα.

Μέθοδοι:

- Ανεστραμμένα Αρχεία (Inverted Files)
- Αρχεία Υπογραφών (Signature Files)
- Bitmaps

Ερωτήματα

Boolean Queries :
- Διαζευκτικά (Disjunctive $t_1 \vee t_2 \vee \ldots \vee t_q$)
- Συζευκτικά (Conjunctive $t_1 \wedge t_2 \wedge \ldots \wedge t_q$)
- Συνδυασμός τους.

Ranked Queries :
Υπολογίζεται ένα score ομοιότητας.

Proximity Queries :
Μεσολαβεί κάποια απόσταση μεταξύ των όρων.

Ανεστραμμένα Αρχεία

Αποτελούνται από ένα λεξικό και μια ανεστραμμένη λίστα για κάθε όρο με δείκτες προς κείμενα.

Ανάγκη για συμπίεση: Χρήση d-gaps και συμπίεσή τους με global ή local μεθόδους. Οι global είναι παραμετροποιημένες ή μη παραμετροποιημένες.

Μέθοδοι Συμπίεσης Ανεστραμμένων Αρχείων

- Binary : $\lceil \log N \rceil$ bits για κάθε δείκτη.
- Unary : Για έναν αριθμό x θέλουμε x – 1 ασους και 1 μηδέν στο τέλος. Γενικά ασύμφορη μέθοδος.
- γ :Unary του $1 + \lfloor \log x \rfloor$ ακολουθούμενη από binary του $x – 2 \wedge (\lfloor \log x \rfloor)$. Απαιτούνται περίπου $1 + 2\log x$ bits.
- δ : γ κωδικοποίηση του $1 + \lfloor \log x \rfloor$ ακολουθούμενη από binary του $x – 2 \wedge (\lfloor \log x \rfloor)$. Για μεγάλους αριθμούς υπερέχει της γ.

Οι ανωτέρω τεχνικές είναι μη παραμετροποιημένες, στις παραμετροποιημένες ακολουθείται μία μοντελοποίηση με χρήση bernouli trials όπου $prob(x)=(1-p)^{x-1}p$, Και μετα ακολουθείται κωδικοποίηση huffmann ή αριθμητική.

Μέθοδοι Συμπίεσης Ανεστραμμένων Αρχείων

- *Global Bernoulli*: χρήση αριθμητικής κωδικοποίησης ή μέθοδος Golomb.

Για μία παράμετρο b, κάθε αριθμός x>0 κωδικοποιείται σε δύο μέρη: q+1 σε unary και το υπόλοιπο r=x-qb-1 σε δυαδικό (περίπου logb bits).
Έχει δειχθεί ότι για:

$$b = \left\lceil \frac{\log(2-p)}{-\log(1-p)} \right\rceil$$

η τεχνική ισοδυναμεί με τον optimal huffman κώδικα για ένα άπειρο σύνολο πιθανοτήτων.
Χρήση παραμέτρου b για όλες τις λίστες με b = (0.69*N*n) / f.

- *Local Bernoulli*: Διαφορετικό b = 0.69N / f_t για κάθε λίστα. Καλύτερη συμπίεση από την global.

- *Interpolative* : Εκμεταλλεύεται το clustering. Συμπιέζει δείκτες και όχι d-gaps. Η πιο αποδοτική μέθοδος αλλά η πιο πολύπλοκη. Ακολουθεί λογική συμπίεσης σε επίπεδα.

Μέθοδοι Συμπίεσης Ανεστραμμένων Αρχείων

Γενική Σύγκριση: Local καλύτερες από Global. Καλή απόδοση η local Bernoulli με Golomb, και γ, δ απλές με σχετικά καλή απόδοση, χωρίς απαίτηση παραμέτρων, εύκολα υλοποιούνται.

Αρχεία Υπογραφών

- Bitstring Signature Files: Hash string για κάθε όρο. Σε κάθε κείμενο μια υπογραφή που είναι το OR των hash strings των λέξεών του. Για τα queries φτιάχνονται query hash strings. Έλεγχος για false matches. Αποδοτικά για πιο συγκεκριμένα queries.

- Bitsliced Signature Files: Αναστροφή πίνακα υπογραφών σε σύνολο bitslices. Ανάγνωση λιγότερων bits. Καλύτερα για αραιά bitslices, τότε λιγότερα false matches.

Αρχεία Υπογραφών

- *Blocked Signature Files*: Οργάνωση Εγγραφών σε blocks. Κάθε bit σε κάθε slice αντιστοιχεί σε B εγγραφές (B = blocking factor). Διαφορετική αντιστοίχηση αριθμών εγγραφών σε αριθμούς blocks σε κάθε slice. Μεγάλο B οδηγεί σε περισσότερους ελέγχους για false matches, προσπελάσεις στο δίσκο.

- *Bitmaps:* Κάθε όρος θέτει 1 bit σε κάθε υπογραφή μήκους ίσου με τον αριθμό των όρων. "1-1" hash function. Εύχρηστα, γρήγορα αλλά καταναλώνουν χώρο. Καλύτερα για common terms. Υβριδικές τεχνικές.

Συμπίεση Αρχείων Υπογραφών

Η συμπίεση δεν αποδίδει όπως στα ανεστραμμένα αρχεία. Έχει απώλειες και εισάγεται θόρυβος. Περισσότερα false matches. Απαιτείται χρόνος για αποσυμπίεση. Πίνακας διευθύνσεων slices στην κύρια μνήμη. Άλλα θέματα :

- Εγγραφές με μήκη με μεγάλες διακυμάνσεις (χαμηλή απόδοση).
- Μεταχείριση συνηθισμένων όρων (πρόβλημα αν υπάρχει διαμοίραση όρων με σπάνιους όρους).

Σύγκριση Μεθόδων Indexing

- Μια ανεστραμμένη λίστα δεν είναι ποτέ πιο πυκνή από ένα αντίστοιχο bitslice. Άρα τα ανεστραμμένα αρχεία δεν κάνουν ποτέ περισσότερες προσβάσεις στο δίσκο. Εξαίρεση όταν το λεξιλόγιο δε χωράει στην κύρια μνήμη. Τα bitslices μπορεί να υπερέχουν όταν οι όροι του query είναι πολλοί σε ειδικές περιπτώσεις.

- Χώρος στο Δίσκο: τα IF απαιτούν 6-10% και τα bitsliced SF το 25-40% της συλλογής.

- Απαιτήσεις Μνήμης: Τα SF δεν απαιτούν χώρο για το λεξιλόγιο, αλλά για όλα τα υπόλοιπα κόστη βρίσκονται σε συγκρίσιμα μεγέθη με αυτά των IF.

- Κατασκευή Index: πιο χρονοβόρα τα SF (hash functions κλπ).

- Δυνατότητες Παραλληλίας: κάθε ερώτημα το χειρίζεται διαφορετικός επεξεργαστής.

Σύγκριση Μεθόδων Indexing

- Ενημέρωση: αναγκαία σε δυναμικές ΒΔ. Πιο εύκολη για τα bitstring SF. Το κόστος μειώνεται με batching, και τότε τα IF είναι καλύτερα.
- Scalability: Ο υπολογισμός query για SF είναι γραμμικός ως προς το μέγεθος της ΒΔ, ενώ στα IF υπογραμμικός.
- Ranking: Τα IF χειρίζονται καλύτερα ranking queries.
- Επεκτασιμότητα: Τα IF τροποποιούνται εύκολα για την απάντηση proximity, NOT, ranked queries.

Συμπεράσματα

Για τυπικές εφαρμογές δεικτοδότησης τα IF υπερέχουν. Τα SF απαιτούν μεγάλο χώρο και πολύ χρόνο να κατασκευαστούν. Τα IF είναι πιο αποδοτικά σε ranked queries και proximity queries.

Full Text Indexing

-Χωρίς δομή δεικτοδότησης (απλοϊκός αλγόριθμος, αλγόριθμος βασικής προεπεξεργασίας, Knuth Morris Pratt, Boyer Moore, Aho Corasick Automaton)

-Με δομή δεικτοδότησης (suffix tree, suffix array, dwag, string b-tree)

Βασικοί Ορισμοί

- Συμβολοσειρά-string: x=x[1]x[2]…..x[n], x[i]$\in\Sigma$ & |x|=n
  x= acgttaaaca, |x|=10 & Σ={a,c,g,t}

- Κενή συμβολοσειρά: ε
- Υπο-συμβολοσειρά-substring w: x=uwv
- Πρόθεμα –Prefix w: x=wu
- Επίθεμα-Suffix w: x=uw
- Κάθε συμβολοσειρά S, μήκους |S|=m, έχει m δυνατά μη κενά επιθέματα που είναι τα ακόλουθα: S[1…m], S[2…m], …. S[m-1…m] και S[m].
- Παράδειγμα "sequence" : *sequence*, *equence*, *quence*, *uence*, *ence*, *nce*, *ce*, *e*.

Suffix Arrays

Έστω:

- αλφάβητο Σ

- κείμενο $A \equiv \alpha_0 \alpha_1 \ldots \alpha_{N-1}$ μεγέθους N, $\alpha_i \in \Sigma$,

  $0 \leq i < N$.

Συμβολοσειρά $W \equiv w_0 w_1 \ldots w_{P-1}$

Ζητείται να βρεθούν όλες οι εμφανίσεις του W στο A.

1η προσέγγιση - Suffix trees

- Suffix tree για το *A*:
  - Ακριβώς *N* φύλλα, αριθμημένα 0 έως *N-1*
  - Κάθε εσωτερικός κόμβος εκτός της ρίζας έχει τουλάχιστον δύο παιδιά
  - Σε κάθε ακμή ∃ ετικέτα ≡ μη κενό *substring* του *A*
    - Σε κάθε κόμβο, οι ετικέτες των εξερχόμενων ακμών ξεκινάνε από διαφορετικό χαρακτήρα
  - Συνένωση των ετικετών των ακμών σε μονοπάτι από τη ρίζα προς φύλλο *k* ≡ suffix του *A* στη θέση *k*.
- *Κατασκευή :*χρόνος O(*N log*|*Σ*|), χώρος O(*N*)

Υπάρχει γραμμική λύση (αλγόριθμος Farach)

- *Απάντηση ερωτημάτων: χρόνος* O(*P log*|*Σ*|)

2η προσέγγιση - Suffix Arrays

Διατεταγμένη λίστα όλων των επιθεμάτων του *A*.

- *Κατασκευή :*χρόνος O(*N log N*) (στη μέση περίπτωση όμως ο χρόνος είναι O(N))
- *Χώρος 2N* ακέραιοι
- *Απάντηση ερωτημάτων: χρόνος P+*$\lceil log2(N-1) \rceil$ συγκρίσεις συμβόλων

Suffix Trees vs. Suffix Arrays

- Query: “Is W a substring of A?”
- Suffix Tree:
  - O(Plog|Σ|) with O(N) space, or:
  - O(P) with O(N|Σ|) space (impractical)
- Suffix Array:
  - Competitive/better O(P+logN) search
  - Main advantage: ***Space***: 2N integers

    (In practice, problem is space overhead of query data structure)
  - Another advantage: Independent of |Σ|

Ορισμοί

- $A_i \equiv$ suffix του A που ξεκινά στη θέση i, δηλ. $A_i = \alpha_i\alpha_{i+1}\ldots\alpha_{N-1}$
- Αν u *string*, τότε $u_p \equiv$ πρόθεμα των πρώτων p συμβόλων του u.
- u, υ strings, τότε ορίζουμε τη σχέση $<_p$ ως εξής:
  - $u <_p \upsilon \Leftrightarrow u_p < \upsilon_p$
- Ομοίως και οι σχέσεις $=_p$, $>_p$, $\leq_p$, $\geq_p$, $\neq_p$

Αναζήτηση

Pos : πίνακας *N* θέσεων, λεξικογραφικά διατεταγμένα τα επιθέματα του *A*.

$$A_{Pos[0]} < A_{Pos[1]} <\ldots< A_{Pos[N-1]}$$

Έστω :

$L_W = \min \{k : W \leq_P A_{Pos[k]}$ ή $k=N\}$

$R_W = \max \{k : W \geq_P A_{Pos[k]}$ ή $k=-1\}$

Αν $L_W < R_w$, τότε για κάθε $k \in [L_W, R_w]$, και $i = Pos[k]$, θα είναι

$W \equiv \alpha_i\alpha_{i+1}\ldots \alpha_{i+P-1}$, και αντίστροφα.

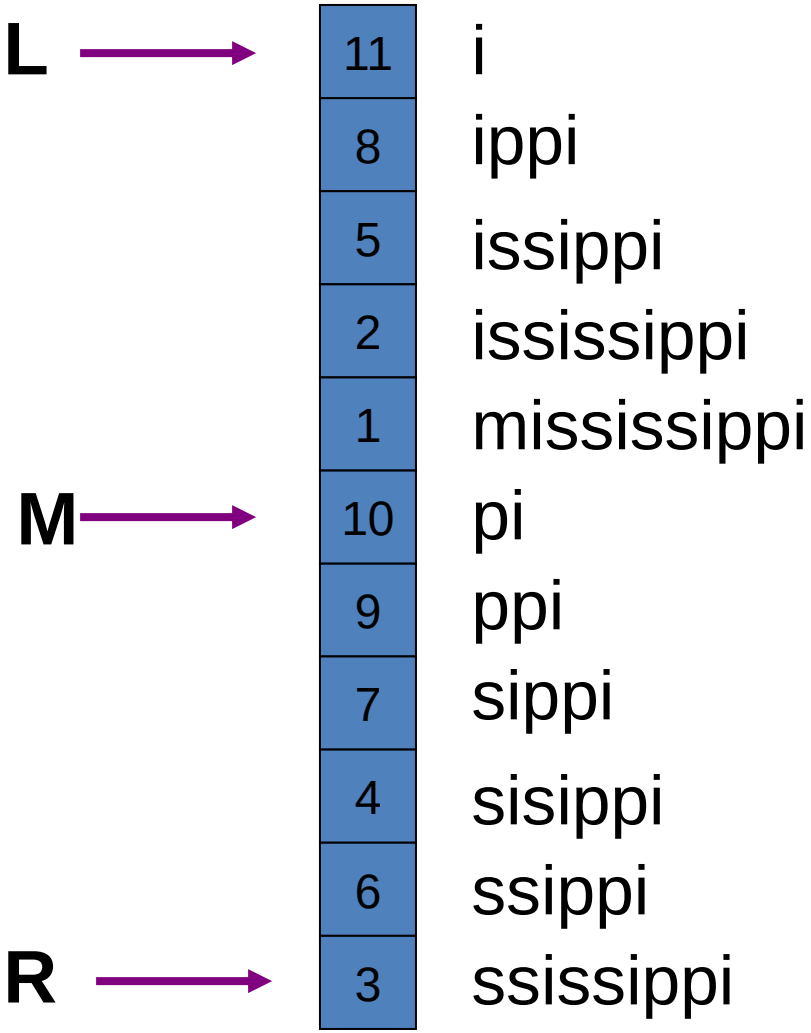
Δηλ, αν $L_W < R_w$, τότε υπάρχουν $R_W - L_W + 1$ εμφανίσεις του *W* στο *A*.

Pos διατεταγμένος $\Rightarrow$ binary search για L_W, R_w

$\Rightarrow$ χρόνος $O(P\ logN)$.

Let A = mississippi

Let W = issa



Πώς θα μειωθεί ο χρόνος;

lcp(*u*,*w*) ≡ μέγεθος του *longest common prefix* *u, w*

$l=lcp(A_{Pos[L]},W)$, $r=lcp(W, A_{Pos[R]})$.
Αρχικά, $l=lcp(A_{Pos[0]},W)$, $r=lcp(W, A_{Pos[N-1]})$

Κάθε σύγκριση του *W* με το $A_{Pos[M]}$ ενημερώνει το *l* ή το *r.*

$$A_{Pos[L]} =_l W,\ W =_r A_{Pos[R]},\ \Rightarrow\ A_{Pos[k]}=_h W,\ \forall k\in[L,R]\ ,\ h = \min(l, r)$$

$\Rightarrow$ το πλήθος των συγκρίσεων συμβόλων που απαιτούνται μειώνεται κατά *h.*

Πώς θα μειωθεί επιπλέον ο χρόνος;

- προϋπολογισμένη πληροφορία σχετικά με το *lcp* του $A_{Pos[M]}$ με τα $A_{Pos[L]}$, $A_{Pos[R]}$.
- *N-2* τριάδες (*L*, *M*, *R*) με μεσαίο σημείο $M \in [1, N-2]$, $0 \leq L < M < R \leq N-1$.
- (L_M, *M*, R_M) η μοναδική τριάδα με το *M* μεσαίο σημείο.
- L_{lcp} , R_{lcp} πίνακες μεγέθους *N-2:*
  - $L_{lcp}[M]=lcp(A_{Pos[LM]}, A_{Pos[M]})$
  - $R_{lcp}[M] = lcp(A_{Pos[M]}, A_{Pos[RM]})$.

Μείωση συγκρίσεων Ι

- Έστω μία επανάληψη του βρόγχου αναζήτησης, τριάδα (L, M, R), H= max(l, r), ΔH η διαφορά μεταξύ της τιμής του H στην αρχή και το τέλος της επανάληψης.

- Έστω $r \leq l = H$. (l=$lcp(A_{Pos[L]},W)$)

- <u>Περίπτωση 1:</u> $Llcp[M] > l$, άρα $lcp(A_{pos[LM]}, A_{Pos[M]}) > l$.
  - Τότε, $A_{Pos[M]} =_{l+1} A_{Pos[L]} \neq_{l+1} W$.
    - W είναι στο δεξί μισό, $L = M$, l μένει όπως είναι.

- <u>Περίπτωση 2:</u> $Llcp[M] < l$, άρα $lcp(A_{pos[LM]}, A_{Pos[M])} < l$.
  - Τότε, $W =_l A_{Pos[L]} <_l A_{Pos[M]}$,
    - W είναι στο αριστερό μέρος, νέα τιμή του r είναι $Llcp[M]$.

Μείωση συγκρίσεων ΙΙ

- Περίπτωση 3: $Llcp[M] = l$, άρα $lcp(A_{Pos[LM]}, A_{Pos[M]}) = l$.
  - Τότε, $A_{Pos[M]} =_l W$.
    - Συγκρίνουμε το $l+1$ σύμβολο, το $l+2$ κτλ, μέχρι $l+j$, το πρώτο για το οποίο είναι $W \neq_{l+j} A_{Pos[M]}$.
    - Το $l+j$ καθορίζει αν το W είναι στην αριστερή ή δεξιά πλευρά, νέα τιμή του r ή του l είναι $l+j-1$.
    - Στην αρχή της επανάληψης είναι $l = h$, άρα $\Delta H+1$ συγκρίσεις συμβόλων.
- $\Delta H + 1$ συγκρίσεις για κάθε επανάληψη, $\Sigma \Delta H \leq P$
- Συνολικό πλήθος συγκρίσεων συμβόλων
  - το πολύ $P + \lceil log2(N-1) \rceil$
  - O($P+ log\ N$) χρόνος στη χειρότερη περίπτωση.

Δευτερεύουσα Μνήμη

- Μεγάλες συλλογές ηλεκτρονικών κειμένων τα οποία είναι ετερογενή μεταξύ τους και προέρχονται από διάφορες πηγές
- Μεγέθη ηλεκτρονικών ΒΔ της τάξης των Gigabyte / Terabyte
- Χρήση συσκευών δευτερεύουσας μνήμης (σκληρός δίσκος, μαγνητική ταινία, DVD)
- Οι δομές δεικτοδότησης των δεδομένων και οι μηχανές αναζήτησης είναι βασικά εργαλεία για την αποθήκευση, ενημέρωση και εξαγωγή χρήσιμης πληροφορίας

Δομές Δεδομένων

Supra-suffix array (υπερ-πίνακας επιθεμάτων – υβριδική δομή)

Compact pat-trees (συμπαγή pat-δέντρα – δυαδική αναπαράσταση, μεταφορά της δομής σε δευτερεύουσα μνήμη)

B-δέντρο προθεμάτων (prefix B-tree)

String B-tree

String B-Tree

- Συνδυασμός των B-Trees και Patricia Tries για δείκτες εσωτερικών κόμβων
- Έχουν προστεθεί επιπλέον δείκτες για να αυξηθεί η ταχύτητα αναζήτησης και οι λειτουργίες ενημέρωσης
- Τα String B-Trees έχουν την ίδια απόδοση χειρότερης περίπτωσης με τα κανονικά B-Trees αλλά επιτυγχάνουν αποθήκευση απεριόριστου μήκους αλφαριθμητικών και εκτελούν τις λειτουργίες αναζήτησης αντίστοιχα με τα suffix trees.

String B-Tree : Συμβολισμοί

- Συμβολίζουμε ένα αλφαριθμητικό s χαρακτήρων με X[1 , s] και με |X| θα ορίζουμε το μήκος του (s)
- Θα συμβολίζουμε με X[1,i] ένα πρόθεμα, με X[j ,s] ένα επίθεμα και με X[i, j] ένα υπό-αλφαριθμητικό του X, όπου $1 \leq i \leq j \leq s$
- Ένα αλφαριθμητικό μοτίβο P υπάρχει μέσα στο X όταν μπορούμε να βρούμε υπό-αλφαριθμητικό του X[i, i + |P| - 1] που να ισούται με το P

Πρόβλημα 1: Αναζήτηση Προθεμάτων και Ερώτημα Εύρους

Ορίζουμε $\Delta = \{\delta_1, \ldots\ldots, \delta_\kappa\}$ ένα σύνολο αλφαριθμητικών ενός κειμένου που το συνολικό τους μήκος είναι N

Αποθηκεύουμε το Δ και το κρατάμε ταξινομημένο στην εξωτερική μνήμη

1) **Prefix Search (P)** :Ανακτά όλα τα αλφαριθμητικά του Δ που το πρόθεμά τους είναι το μοτίβο P

2) **Range Query (K', K")** που ανακτά όλα τα αλφαριθμητικά του Δ ανάμεσα στο K' και το K" σε λεξικογραφική σειρά

Πρόβλημα 2: Αναζήτηση Υπό-αλφαριθμητικού

- Το ερώτημα Substring Search (P) βρίσκει όλα τα P που υπάρχουν στα αλφαριθμητικά του Δ
- Το occ δηλώνει το πλήθος των υπάρξεων
- Αποτελεί επέκταση του προβλήματος 1, με το σύνολο να απαρτίζεται από όλα τα επιθέματα αλφαριθμητικών του Δ.

Μοντέλο Μνήμης

- Θα αναλύσουμε τα προβλήματα 1 και 2 με βάση το κλασσικό μοντέλο μνήμης 2 επιπέδων [Cormen et al. 1990]
- Θεωρούμε ότι υπάρχει μια γρήγορη και μικρή κύρια μνήμη (RAM) και μια πιο αργή αλλά πολύ μεγάλη εξωτερική μνήμη (σκληρός δίσκος, DVDs). Η εξωτερική μνήμη είναι χωρισμένη σε blocks μεταφοράς, που καλούνται *σελίδες δίσκου (disk pages)*

Μοντέλο Μνήμης

- Κάθε σελίδα δίσκου περιέχει Β ατομικά αντικείμενα που μπορεί να είναι ακέραιοι, χαρακτήρες ή δείκτες
- Το Β ονομάζεται μέγεθος σελίδας δίσκου (disk page size) και η εγγραφή ή η ανάγνωση προσπέλαση στον δίσκο (disk access)

Πολυπλοκότητα : Πρόβλημα 1

- Η Prefix Search (P) απαιτεί $O((p + occ) / B + \log_B k)$ προσπελάσεις στον δίσκο στην χειρότερη περίπτωση, $p = |P|$
- Η Range Query (K',K") απαιτεί $O((k' + k" + occ) / B + \log_B k)$ προσπελάσεις στην χειρότερη περίπτωση, $k' = |K'|$ και $k" = |K"|$
- Η εισαγωγή ή διαγραφή ενός αλφαριθμητικού μήκους m του συνόλου Δ απαιτεί $O(m/B + \log_B k)$ προσπελάσεις στην χειρότερη περίπτωση
- Η χρήση του χώρου είναι $\Theta(k/B)$ σελίδες δίσκου, όπου ο χώρος που καταλαμβάνεται από το σύνολο Δ είναι $\Theta(N/B)$ σελίδες

Πολυπλοκότητα : Πρόβλημα 2

- Η Substring Search (P) απαιτεί $O((p + occ) / B + \log_B N)$ προσπελάσεις στον δίσκο στην χειρότερη περίπτωση, όπου $p = |P|$
- Η εισαγωγή ή διαγραφή ενός αλφαριθμητικού μήκους m του συνόλου Δ απαιτεί $O(m \log_B (N+m))$ προσπελάσεις στην χειρότερη περίπτωση
- Ο χώρος που χρησιμοποιείται από το String B-Tree και από το σύνολο Δ είναι $\Theta(N / B)$ σελίδες δίσκου

Αποθήκευση Αλφαριθμητικών

- Με αυτή τη δομή μπορούμε να εντοπίσουμε τη σελίδα δίσκου που περιέχει τον i-οστό χαρακτήρα ενός αλφαριθμητικού εκτελώντας έναν σταθερό αριθμό απλών αριθμητικών πράξεων στον δείκτη του
- Μπορούμε να ομαδοποιήσουμε Θ(B) λογικούς δείκτες σε μια μόνο σελίδα δίσκου, αλλά αν διαβάσουμε μόνο αυτή τη σελίδα δεν θα είμαστε σε θέση να ανακτήσουμε όλους τους χαρακτήρες των αλφαριθμητικών

Αποθήκευση Αλφαριθμητικών

- Μπορούμε να συγκρίνουμε οποιαδήποτε δύο αλφαριθμητικά χαρακτήρα προς χαρακτήρα, αλλά αυτή η πράξη είναι πολύ αναποτελεσματική αν επαναλαμβάνεται διότι το κόστος χειρότερης περίπτωσης είναι ανάλογο του μήκους των δύο αλφαριθμητικών
- Καλούμε το πρόβλημα αυτό *επανασάρωση (rescanning)* διότι οι ίδιοι χαρακτήρες επανεξετάζονται πολλές φορές

B-tree Like Δομή

- Αναπαριστούμε τα αλφαριθμητικά μέσω λογικών δεικτών
- Σαν είσοδο έχουμε το σύνολο αλφαριθμητικών Δ με συνολικό αριθμό χαρακτήρων N
- Ορίζουμε $K = \{K_1,\ldots, K_\kappa\}$ τα στοιχεία του συνόλου Δ σε αύξουσα λεξικογραφική σειρά ($\leq_L$)
- Τα αλφαριθμητικά του K διανέμονται στα φύλλα του B-Tree, και μόνο ορισμένα αλφαριθμητικά βρίσκονται στους εσωτερικούς κόμβους

B-tree Like Δομή

- Ορίζουμε το διατεταγμένο σύνολο αλφαριθμητικών που συνδέονται με τον κόμβο π ως $\Phi_\pi \subseteq K$, και συμβολίζουμε το αριστερότερο και δεξιότερο αλφαριθμητικά του Φ_π με $L(\pi)$ και $R(\pi)$ αντίστοιχα
- Κάθε κόμβο π τον αποθηκεύουμε σε μια σελίδα δίσκου και θέτουμε περιορισμό στο πλήθος των αλφαριθμητικών του : $b \leq |\Phi_\pi| \leq 2b$, όπου $b=\Theta(B)$ είναι ένας ζυγός ακέραιος επιλεγμένος έτσι ώστε να μπορεί ένας κόμβος να χωρέσει σε μια σελίδα δίσκου
- Μόνο η ρίζα επιτρέπεται να έχει λιγότερα από b αλφαριθμητικά

B-tree Like Δομή

- Χωρίζουμε το σύνολο K σε ομάδες των b συνεχόμενων αλφαριθμητικών
- Χαρτογραφούμε κάθε ομάδα σε ένα φύλλο, έστω π, και αναπτύσσουμε το Φ_π
- Κάθε εσωτερικός κόμβος π έχει n(π) παιδιά $\sigma_1,\ldots,\sigma_{n(\pi)}$ και το διατεταγμένο σύνολό του που είναι το $\Phi_\pi = \{L(\sigma_1), R(\sigma_1), \ldots, L(\sigma_{n(\pi)}), R(\sigma_{n(\pi)})\}$
- Αφού $n(\pi)= |\Phi_\pi| / 2$, κάθε κόμβος έχει από b/2 μέχρι b παιδιά εκτός από τη ρίζα και τα φύλλα. Άρα το τελικό ύψος του B-Tree που σχηματίστηκε είναι $\mathbf{H=O(\log_{b/2} k) = O(\log_B k)}$

Prefix Search (P)

Για την ανάλυση της πράξης Prefix Search(P) θα βασιστούμε σε δυο παρατηρήσεις των Manber και Mayers :

- *Τα αλφαριθμητικά που έχουν πρόθεμα P καταλαμβάνουν γειτονικές θέσεις του K*
- *Το αριστερότερο αλφαριθμητικό που έχει πρόθεμα P είναι γειτονικό του P στο σύνολο K με βάση την αύξουσα λεξικογραφική σειρά*

Prefix Search(P)

- Αναπαριστούμε τη θέση του P στο σύνολο K με το ζεύγος (τ , j) έτσι ώστε το τ να είναι το φύλλο που έχει αυτή τη θέση και j-1 να είναι το πλήθος των αλφαριθμητικών του Φ_τ που είναι λεξικογραφικά μικρότερα από το P, όπου $1 \leq j \leq |\Phi_\tau| + 1$
- Λέμε ότι το j είναι η θέση του P στο Φ_τ
- Δηλώνουμε την διαδικασία που καθορίζει την θέση του P στο σύνολο Φ_π με την πράξη PT-Search(P, Φ_π)

Prefix Search(P): Αλγόριθμος

- Ελέγχουμε τις δύο περιπτώσεις στις οποίες το P είναι μικρότερο / μεγαλύτερο από όλα τα αλφαριθμητικά του K
- Αν και οι δύο έλεγχοι είναι αρνητικοί , ξεκινάμε με π=root και εκτελούμε σάρωση προς τα κάτω του B-tree
- Όταν επισκεπτόμαστε έναν κόμβο π, φορτώνουμε την αντίστοιχη σελίδα δίσκου και εφαρμόζουμε την διαδικασία PT-Search ώστε να βρούμε τη θέση j του P στο σύνολο Φ_π
- Τα δύο γειτονικά αλφαριθμητικά καθορίζονται από τη σχέση $K'_{j-1} <_L P \leq_L K'_j$

Prefix Search(P): Αλγόριθμος

- Αν ο κόμβος π είναι φύλλο σταματάμε τη σάρωση, αλλιώς έχουμε τις ακόλουθες δύο περιπτώσεις

1. Αν τα αλφαριθμητικά K'_{j-1} και K'_j ανήκουν σε δύο ξεχωριστά παιδιά του π τότε τα δύο αλφαριθμητικά είναι γειτονικά στο Κ. Έστω σ το παιδί του π που περιέχει το K'_j Διαλέγουμε το τ ως το αριστερότερο φύλλο του δέντρου που κατάγεται από το σ και ξέρουμε ότι το P βρίσκεται στην πρώτη θέση του Φ_τ διότι $L(\tau) = L(\sigma) = K'_j$
2. Αν τα K'_{j-1} και K'_j ανήκουν στο ίδιο παιδί του π, συνεχίζουμε τη σάρωση περιοδικά στο επόμενο επίπεδο

Prefix Search(P): Αλγόριθμος

- Σε κάθε περίπτωση , στο τέλος της σάρωσης θα βρούμε ένα ζεύγος (τ_L , j_L) που θα αντιπροσωπεύει τη θέση του αριστερότερου αλφαριθμητικού του K που έχει πρόθεμα P
- Αντίστοιχα μπορούμε να βρούμε το ζεύγος (τ_R , j_R)
- Για να απαντήσουμε στο ερώτημα Prefix Search (P) θα πρέπει να σαρώσουμε τη σειρά των φύλλων που ορίζονται από τα τ_L και τ_R και να φτιάξουμε λίστα από το j_L-οστό αλφαριθμητικό του $\Phi_{\tau L}$ έως το ($j_R - 1$)-οστό του $\Phi_{\tau R}$

Απλοποιημένο String B-Tree

- Είναι ο συνδυασμός της B-Tree-like δομής που έχουμε με το Patricia Trie
- Σκοπός μας είναι να οργανώσουμε σωστά τα αλφαριθμητικά και να κάνουμε αναζητήσεις που να απαιτούν σύγκριση μόνο ενός αλφαριθμητικού του Φ_π στην χειρότερη περίπτωση, σε σχέση με τις $\log_2|\Phi_\pi|$ που χρειάζεται η απλή δυαδική αναζήτηση

Απλοποιημένο String B-Tree

Μπορούμε να ορίσουμε το PT σε δύο βήματα :

(1) Κατασκευάζουμε ένα συμπαγές Trie με τα αλφαριθμητικά του Φ_π

(2) Βάζουμε ετικέτα σε κάθε κόμβο του Trie με βάση το μήκος του υπό-αλφαριθμητικού που αποθηκεύεται σε αυτόν και αντικαθιστούμε κάθε αλφαριθμητικό πάνω στα κλαδιά με τον πρώτο του χαρακτήρα

Απλοποιημένο String B-Tree

- Το Patricia Trie χάνει κάποια πληροφορία σε σχέση με το συμπαγές Trie λόγω της διαγραφής των χαρακτήρων, εκτός του πρώτου, στα κλαδιά του δέντρου
- Βάζει Θ(B) αλφαριθμητικά σε έναν κόμβο του B-Tree, ανεξαρτήτως του μεγέθους τους
- Επιτρέπει λεξικογραφικές αναζητήσεις σε έναν κόμβο, χωρίς επιπλέον προσπελάσεις του δίσκου

Παράδειγμα PT-Search

Θα δούμε ένα παράδειγμα PT-Search(P, Φ_π) , όπου

P = “bcbabcba”

- Αριστερά φαίνεται η πρώτη φάση στην οποία το l αναπαριστά το δεξιότερο φύλλο
- Ξεκινάμε προσδιορίζοντας το κοινό πρόθεμα του αλφαριθμητικού του l και του P (π.χ. ‘bcb’) και στη συνέχεια βρίσκουμε τον χαμηλότερο πρόγονο του l που έχει ετικέτα μεγαλύτερη από |’bcb’|=3
- Στην συνέχεια χρησιμοποιούμε τον αταίριαστο χαρακτήρα P[4] = ‘a’ για να βρούμε την ακριβή θέση του P (j = 4) διασχίζοντας τα μαρκαρισμένα τόξα

Παράδειγμα PT-Search

- Συνδυάζοντας τα Patricia Tries με το B-Tree αποφεύγουμε τη δυαδική αναζήτηση στους κόμβους που περνάμε και έτσι μειώνουμε τη συνολική πολυπλοκότητα
- Το όριο αυτό δεν είναι ακόμα ικανοποιητικό. Ο λόγος είναι ότι σε κάθε κόμβο που έχουμε επισκεφτεί ξανασαρώνουμε (rescanning) το P από την αρχή

Παράδειγμα PT-Search

- Πρέπει να σχεδιάσουμε έτσι τη διαδικασία PT-Search ώστε να εκμεταλλεύεται καλύτερα τις ιδιότητες του String B-Tree και του Patricia Trie
- Θα χρησιμοποιήσουμε τρεις παραμέτρους εισόδου (P, Φ_π, l) όπου η παράμετρος l ικανοποιεί την ιδιότητα ότι *υπάρχει ένα αλφαριθμητικό στο Φ_π που οι l πρώτοι χαρακτήρες του είναι ίσοι με του P*
- Η διαδικασία PT-Search (P, Φ_π, l) επιστρέφει ζεύγος (j, lcp) όπου το j είναι η θέση του P στο Φ_π και η παράμετρος lcp είναι το μήκος του κοινού προθέματος

Πολυπλοκότητα Prefix Search

- Ανακτάμε τα αλφαριθμητικά του K που έχουν πρόθεμα P εξετάζοντας τα φύλλα του String B-Tree που ορίζονται από τα τ_L και τ_R σε O(occ/B) προσπελάσεις στον δίσκο
- Το συνολικό κόστος του Prefix Search (P) είναι **$O((p + occ) / B + \log_B k)$** προσπελάσεις στον δίσκο

Πρόβλημα 2 : Substring Search

- Το πρόβλημα 2 αφορά σε μια πιο αποδοτική πράξη Substring Search(P) που αναζητά εμφανίσεις του P στα αλφαριθμητικά του Δ
- Η αναζήτηση βασίζεται στην εύρεση υπό-αλφαριθμητικών μήκους ρ που ισούνται με το P
- Ορίζουμε το σύνολο των επιθεμάτων ως SUF(Δ) = {δ[i, |δ|] : $1 \leq i \leq |\delta|$, όπου $\delta \in \Delta$}, το οποίο περιέχει N λεξικογραφικά διατεταγμένα επιθέματα

Πρόβλημα 2 : Substring Search

- Το πρόβλημά μας είναι να ανακτήσουμε όλα τα SUF(Δ) αλφαριθμητικά που έχουν πρόθεμα P
- η πράξη Substring Search(P) στο σύνολο Δ μετασχηματίζεται σε Prefix Search(P) στο σύνολο SUF(Δ)

Παράδειγμα Substring Search

- P=‘at’, Δ= {‘aid’, ‘atlas’,‘atom’, ‘attenuate’, ‘car’, ‘patent’, ‘zoo’}, b = 4 και K={‘aid’, ‘ar’, ‘as’, . . . , ‘uate’, ‘zoo’}
- Στο συγκεκριμένο παράδειγμα έχουμε occ=4 εμφανίσεις ‘**at**las’,‘**at**om’, ‘**at**tenu**ate**’ και ‘p**at**ent’
- Τα επιθέματα με πρόθεμα P που ανταποκρίνονται σε αυτές τις υπάρξεις έχουν τους λογικούς τους δείκτες αποθηκευμένους σε γειτονικά φύλλα του δέντρου
- Μπορούμε να ορίσουμε το σύνολο K=SUF(Δ) και το μέγεθός του k=N και να εκτελέσουμε την πράξη Prefix Search(P)

String B-Tree

- Η εισαγωγή ενός μόνο αλφαριθμητικού Υ στο σύνολο Δ, όπου m = |Υ|, απαιτεί την εισαγωγή όλων των m επιθεμάτων στο σύνολο SUF(Δ) σε λεξικογραφική σειρά
- Το πρόβλημα είναι ότι χειριζόμαστε τα m εισαγόμενα επιθέματα σαν τελείως ξεχωριστά αλφαριθμητικά και παρουσιάζεται το φαινόμενο του rescanning
- Επεκτείνουμε το απλοποιημένο String B-Tree εισάγοντας δύο τύπους βοηθητικών δεικτών με τους οποίους θα αποφύγουμε το rescanning κατά τη διαδικασία της ενημέρωσης

String B-Tree

- Ο ένας τύπος δείκτη είναι ο γνωστός *δείκτης γονέα* που ορίζεται για κάθε κόμβο
- Ο άλλος είναι ο *succ δείκτης* που ορίζεται για κάθε αλφαριθμητικό στο SUF(Δ) ως εξής : Ο succ δείκτης για το δ[i, |δ|] ∈SUF(Δ) οδηγεί στο φύλλο του String B-Tree που περιέχει το δ[i+1, |δ|]. Αν ισχύει i = |δ| τότε ο succ δείχνει στο ίδιο του το φύλλο (self-loop pointer)

Αλγόριθμος Εισαγωγής

Μετά την εισαγωγή του Y[i, m] πρέπει να ικανοποιούνται οι παρακάτω δύο συνθήκες

- Τα επιθέματα Y[j, m] είναι αποθηκευμένα στο String B-Tree, για $1 \leq j \leq i$ και το Y[i, m] μοιράζεται τους hi πρώτους χαρακτήρες του με ένα από τα γειτονικά του αλφαριθμητικά
- Όλοι οι succ δείκτες είναι σωστά τοποθετημένοι για όλα τα αλφαριθμητικά στο String B-Tree εκτός από το Y[i, m]. Αυτό σημαίνει ότι ο succ(Y[i, m]) είναι ο μόνος εξαρτώμενος δείκτης εκτός και αν i = m οπότε και δείχνει στο ίδιο του το φύλλο

Αλγόριθμος Εισαγωγής

- Εισάγουμε τα επιθέματα του Y στο String B-Tree αποθηκεύοντας το SUF(Δ) στην αρχή από το μεγαλύτερο στο μικρότερο
- Συνεχίζουμε την εισαγωγή για i = 1, 2, …, m
- Για i=1, βρίσκουμε τη θέση του Y[1, m] σαρώνοντας το δέντρο όπως και στην περίπτωση του Prefix Search(Y[i, m])
- Για τα υπόλοιπα επιθέματα του Y (i>1) χρησιμοποιούμε διαφορετική προσέγγιση για να αποφύγουμε το rescanning

Αλγόριθμος Εισαγωγής

- Όταν βρούμε τη θέση του Y[i, m], αντί να ξεκινήσουμε από τη ρίζα, σαρώνουμε το δέντρο από το τελευταίο φύλλο που επισκεφτήκαμε
- Ξέρουμε ότι το Y[i-1, m] μοιράζεται τους h_{i-1} πρώτους χαρακτήρες του με κάποιο από τα γειτονικά του αλφαριθμητικά
- Μπορούμε να πάρουμε τον δείκτη succ του γειτονικού αλφαριθμητικού, και να καταλήξουμε σε ένα φύλλο με την παρακάτω ιδιότητα : *να περιέχει ένα αλφαριθμητικό που να μοιράζεται τους πρώτους* $max\{0, h_{i-1} - 1)$ *χαρακτήρες με το Y[i,m]*

Αλγόριθμος Εισαγωγής

- Συνεχίζουμε την εισαγωγή εκτελώντας μια προς τα πάνω και προς τα κάτω σάρωση του String B-Tree μέχρι να φτάσουμε στο φύλλο π_i που θα περιέχει τη θέση του Y[i, m]
- Από τη στιγμή που μπορούμε να αποδείξουμε ότι $h_\iota \geq \max\{0, h_{i-1} - 1)$ ο αλγόριθμός μας αποφεύγει το rescanning εξετάζοντας μόνο τους χαρακτήρες του Y στις θέσεις $i + \max\{0, h_{i-1} - 1), \ldots, i + h_i$

Αλγόριθμος Εισαγωγής

- Ένας απ' ευθείας χειρισμός των δεικτών parent και succ θα χρειαστεί $O(B \log_B (N+m))$ προσπελάσεις στον δίσκο ανά εισαγόμενο επίθεμα στην χειρότερη περίπτωση
- Συνολικά $\Sigma^{m}_{i=1} di = O(m \log_B(N + m))$ προσπελάσεις στον δίσκο απαιτούνται για την εισαγωγή του Υ στο Δ
- Επιτυγχάνουμε ίδια απόδοση χειρότερης περίπτωσης με την εισαγωγή m ακέραιων σε ένα κανονικό B-δέντρο

Τέλος Ενότητας

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σημειώματα

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.