Dynamic dictionary matching problem

Έχουμε ένα σύνολο πρότυπων $D = \{P_1, P_2, \ldots, P_k\}$

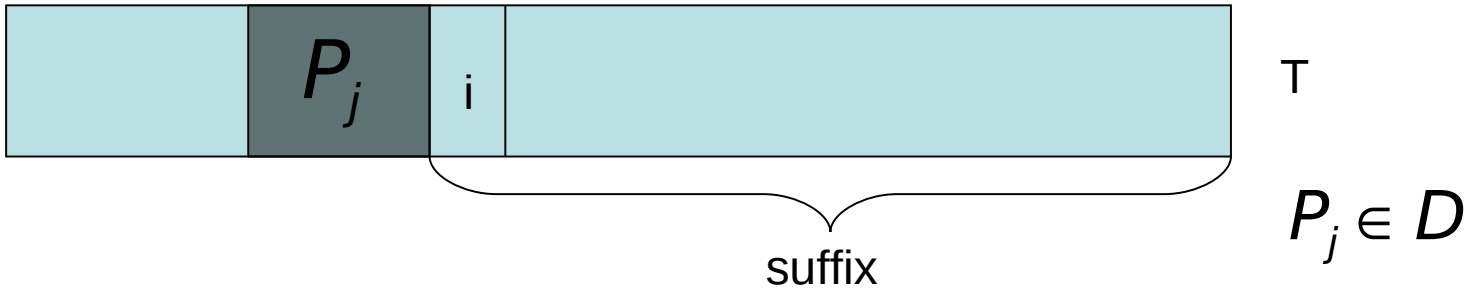
οπου D το λεξικό και ένα αυθαίρετο κειμενο T [1,n]

Το σύνολο των πρότυπων αλλάζει με το χρόνο (ρεαλιστική συνθήκη). Οι επιτρεπτές λειτουργίες είναι :

- **Insert (P)** : Προσθέτει ένα νέο πρότυπο P στο λεξικό D

- **Delete (P)** : Διαγράφει το πρότυπο P από το λεξικό D

- **Query (T)** : Για κάθε θέση i , επιστρέφει μια λίστα με όλα τα πρότυπα που είναι προθέματα του επιθέματος T [i,n] .



* (occurrence του πρότυπου P στη θέση i του T)

Επίλυση προβλήματος

- πολυπλοκότητες αλγόριθμων επίλυσης του προβλήματος

TABLE 1

A Summary of the Known Dynamic Dictionary Matching Solutions, where $p = |P|$, $n = |T|$, and d Is the Total Dictionary Size.

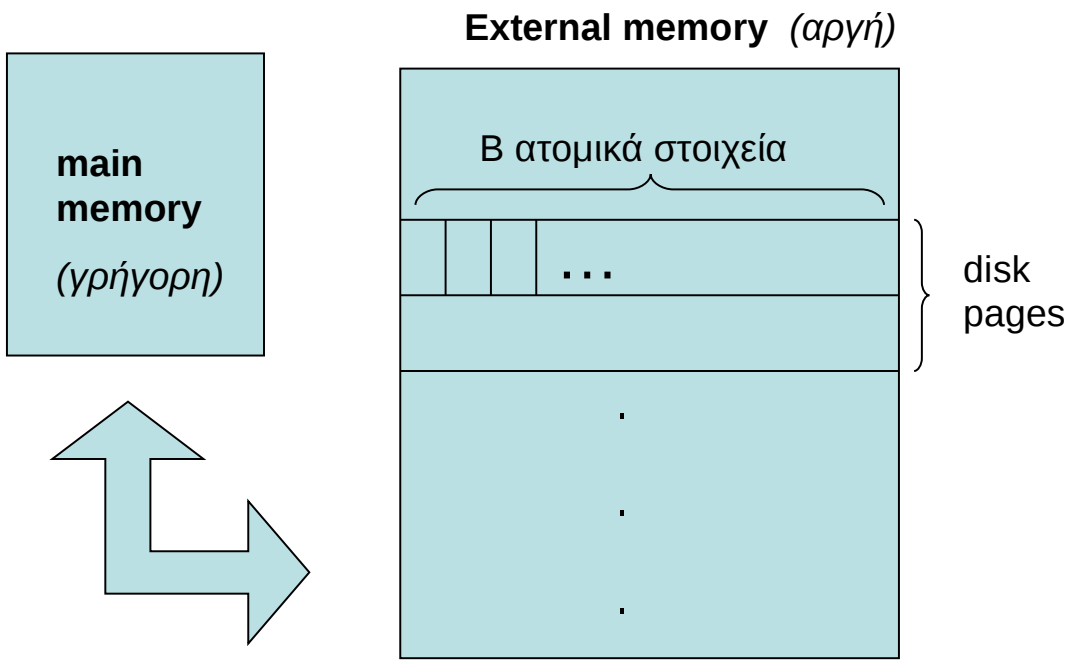
Dynamic Dictionary Matching			
Algorithm	Ins/Del(P)	Query(T)	Preprocessing
Amir–Farach [2]	$O(p \log d)$	$O((n + tocc) \log d)$	$O(d \log d)$
Amir *et al.* [3]	$O(p \log d)$	$O((n + tocc) \log d)$	$O(d \log d)$
Idury–Schäffer [15]	$O(p \log d)$	$O((n + tocc) \log d)$	$O(d)$
Amir *et al.* [4]	$O\left(p \frac{\log d}{\log \log d}\right)$	$O\left((n + tocc) \frac{\log d}{\log \log d}\right)$	$O(d)$
Sahinalp–Vishkin [23]	$O(p)$	$O(n + tocc)$	$O(d)$

The time complexity of the deletion operation in [2] is *amortized*. The alphabet is assumed to be bounded.

- τα παραπάνω έχουν επιτευχθεί κυρίως με την εκμετάλλευση των ιδιοτήτων της δομής δεδομένων δέντρων επιθήματος που χάνει στη συνέχεια την αποδοτικότητα όταν προσαρμόζεται να εργαστεί στην εξωτερική μνήμη . Άρα χρειάζεται ένας νέος αλγόριθμος .

Βασικά στοιχεία για τη συνεχεία …

- Μοντέλο ιεραρχικής μνήμης 2 επιπέδων :



Ασυμπτωτικα όρια για :

α) συνολικός αριθμός των προσβάσεων δίσκου

β) συνολικός αριθμός κατειλημμένων σελίδων δίσκου

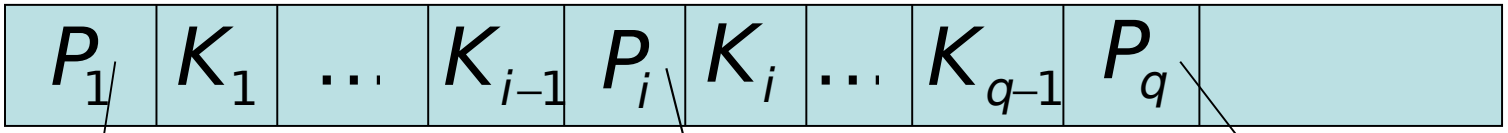
- *lcp (X,Y)* : το μέγιστο μήκος κοινού προθέματος των X , Y
- *max_lcp (X,Φ)* = max{lcp(X,Z):Z ε Φ}
- $\leq_L$: αυξανόμενη λεξικογραφική σειρά

B+ Δένδρο

α) Εσωτερικός κόμβος με q-1 τιμές αναζήτησης

β) κόμβος - φύλλο

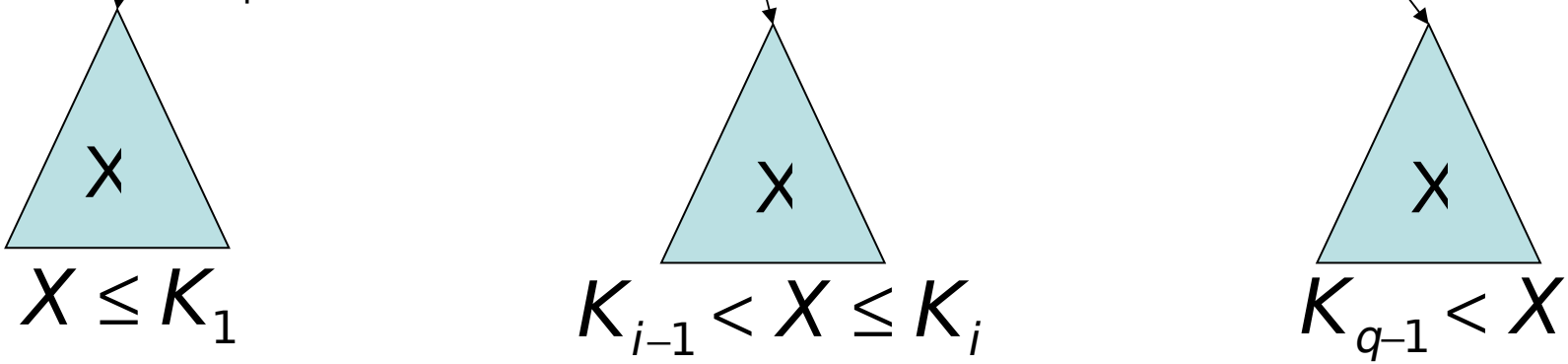
α)



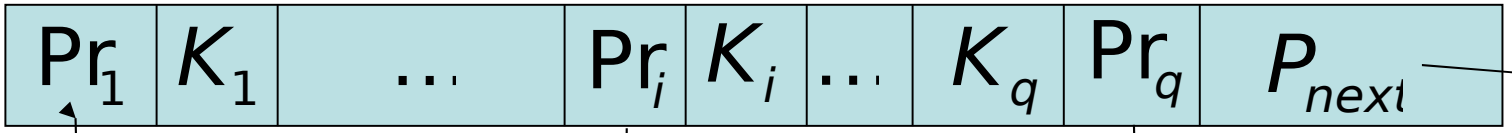
δείκτης δέντρου

δείκτης δέντρου

δείκτης δέντρου



β)



Δείκτης προς τον επόμενο κόμβο – φύλλο του δέντρου

δείκτης δεδομένων

δείκτης δεδομένων

δείκτης δεδομένων

Γενίκευση του προβλήματος

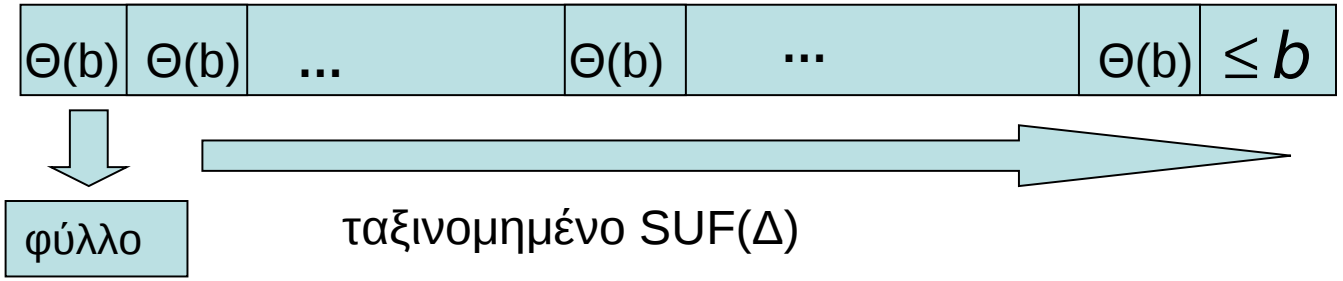
Πρόβλημα 2.1. Έστω το Δ = $\{T_1, \ldots, T_k\}$ να είναι ένα σύνολο σειρών κειμένων των οποίων το συνολικό μήκος είναι $N = \sum_{i=1}^{k} |T_i|$. Αποθήκευσε το Δ στην εξωτερική μνήμη και το διατήρησε το με την εισαγωγή και τη διαγραφή μεμονωμένων strings κειμένων. Υποστηρίζει την ερώτηση Substring Search (P) που αποτελείται από την εύρεση όλων των occurrences ενός πρότυπου P στα strings του Δ . Έστω ότι το occ δείχνει το συνολικό αριθμό τέτοιων occurrences .

Θεώρημα 2.2. Το Substring Search(P) κάνει $O((p+occ)/B+\log_B N)$ προσβάσεις στο δίσκο στη χειρότερη περίπτωση (όπου p = |P|).
Η διαγραφή ή η εισαγωγή ενός string μήκους m στο Δ κάνει $O(m\log_B(N+m))$ προσβάσεις στο δίσκο στη χειρότερη περίπτωση . Ο απαιτούμενος χώρος είναι Θ(N/B) σελίδες δίσκου .

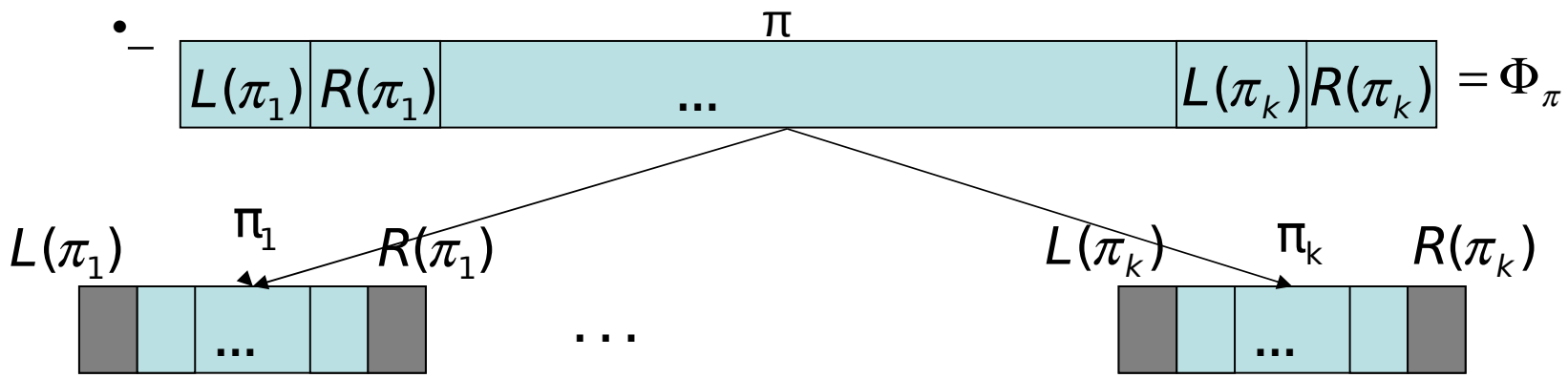
SB δέντρο (παρόμοιο με το B+ Δένδρο)

Ιδιότητες :

- τα κλειδιά βρίσκονται αποθηκευμένα στα φύλλα και μερικά αντίτυπα στους εσωτερικούς κόμβους
- τα κλειδιά είναι ταξινομημένα κατά $\leq_L$
- κάθε σελίδα δίσκου έχει το πολύ 2b κλειδιά , όπου b = Θ(B)
-



-



Λήμμα 2.4. Έστω ένας κόμβος π ενός Sb - δέντρου και ένας ακέραιος αριθμός $\ell \geq 0$ έτσι ώστε ισχύει : Υπάρχει ένα string του Φ_π του οποίου οι πρώτοι χαρακτήρες είναι ίσοι με του P .

Η εκτέλεση SB-Search-Up-Down (P , π , ℓ) επιστρέφει την τριπλετα (τ , j , lap), οπού τ είναι το φύλλο του Sb - δέντρου που περιέχει τη θέση του P στο SUF (Δ), το j είναι η θέση του P στο σύνολο string Φ_π , και lap = max_lcp (P , SUF (Δ)) . Το συνολικό κόστος είναι $O((lpc-\ell)/B+\log_B N)$ προσβάσεις δίσκων .

Ιδιότητα 2.5. Έστω $X_1, X_2, \ldots, X_h$ είναι μια lexicographically διαταγμένη ακολουθία από strings . Για οποιοδήποτε ζευγάρι των δεικτών i < j , έχουμε

$$LCP(X_i , X_j) = \min_{u=i,\ldots j-1}\{LCP(X_u, X_{u+1}\} \quad .$$

Επαυξημένο SB - δέντρο

Λόγοι επαύξησης :

• Τα φύλλα SBT_D αποθηκεύουν και τα σχέδια του D και τα επιθέματά τους σε λεξικογραφική σειρά . Ως εκ τούτου, τα strings για τα οποία ενδιαφερόμαστε (δηλ., τα πρότυπα του D) μεταθέτονται με έναν μεγαλύτερο αριθμό άλλων strings (δηλ., τα επιθέματα των πρότυπων) σύμφωνα με τη λεξικογραφική σειρά . Αυτό σημαίνει ότι τα σχέδια στο D που είναι προθέματα του T[i, n], δεν είναι εγγυημένο ότι θα αποθηκευτούν σε μια παρακείμενη ακολουθία φύλλων του SBT_D

• Επιπλέον, αυτά τα σχέδια δεν αποθηκεύονται απαραιτήτως στα φύλλα SBT_D πλησίον στη λεξικογραφική θέση του T[i, n] στο SUF (D) . Αυτό υπονοεί ότι δεν μπορούμε οικονομικά να εφαρμόσουμε την query (T) με τη χρησιμοποίηση της ίδιας προσέγγισης της Substring Search . Στην πραγματικότητα, η λεξικογραφική θέση του επιθέματος T [i, n] στο SBT_D δεν βοηθοί πολύ εδώ επειδή αρχίζοντας από εκείνη την θέση μπορεί να πρέπει να ανιχνεύσουμε έναν μεγάλο αριθμό φύλλων πριν ανακτήσουμε όλα τα σχέδια που είναι προθέματα του T[i, n].

Τρόπος δημιουργίας επαυξημένου SB – δέντρου

Φ_{π}^{tot} : το σύνολο των strings (με λεξικογραφική σειρά) που είναι αποθηκευμένο στα φύλλα που κατεβαίνουν από το π .

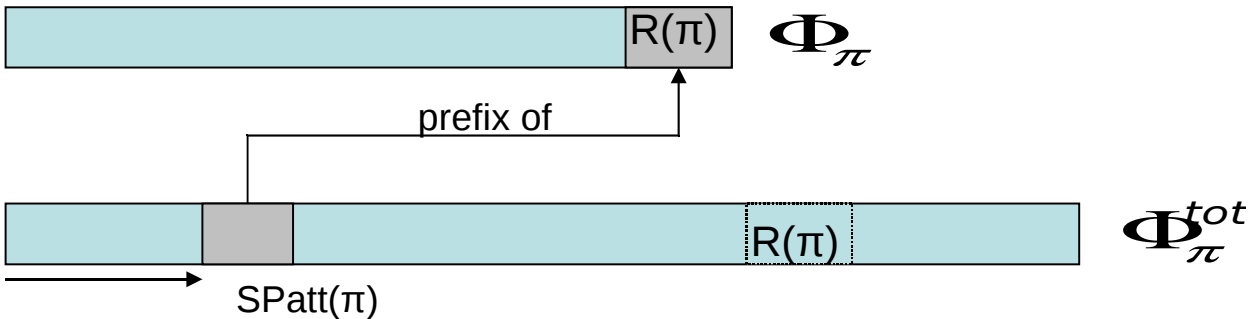
Ιδιότητες :

Αποθηκεύουμε σε κάθε κόμβο π τα παρακάτω:

LCP(Z_j,Z_j+1): lpc(Z_j,Z_j+1) για κάθε ζευγάρι από παρακείμενα strings Z_j και Z_j+1 στο Φ_π.

Spatt (π) : το αριστερότερο pattern στο Φ_{π}^{tot} που είναι επίθεμα του R(π) (μπορεί να είναι και ίδιο το R(π)). Αν ένα τέτοιο pattern δεν υπάρχει , θέτουμε SPatt(π)=Λ όπου Λ είναι ένα ειδικό string το οποίο δεν είναι επίθεμα κανενός αλλού string. Επίσης αποθηκευουμε στο π το μήκος |SPatt(π)|.

Spattchild (j) :το pattern Spat(π_j) όπου κάθε παιδί π_j του π και έχει μήκος |Spat(π_j)| .



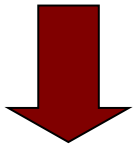
Γεγονός 3,1. Λαμβάνοντας υπόψη μια σειρά Υ και τη θέση της ι σε SUF (D) (δηλ. $X_i \leq_L Y <_L X_{i+1}$), έχουμε:

1. Τα σχέδια λεξικών που είναι προθέματα του Υ βρίσκονται αριστερά του Xi (συμπεριλαμβανομένων και αυτού) και διατάζονται από δεξιά με αυξανόμενο μήκος.
2. Αν Xj είναι ένα πρόθεμα του Υ, τότε Xj είναι ένα πρόθεμα όλων των strings Xh που βρίσκονται μεταξύ Xj και Υ (αυτό σημαίνει , Xj είναι ένα πρόθεμα από Xh , για όλα τα h=j, j+1, ..., i).
3. Ένα string Z είναι ένα πρόθεμα του Υ αν και μόνον αν, δοθέντος ενός Xj ε SUF(D) τέτοιο ώστε $Z \leq_L X_j \leq_L Y$, έχουμε ότι Z είναι ένα πρόθεμα του Xj και $|Z| \leq$ LCP(Xj , Y).

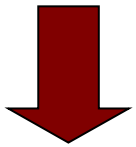
Το γεγονός 3,1 και ο καθορισμός του SPatt υπονοούν ότι SPatt (π) είναι το πιο σύντομο pattern σε Φ_π^{tot} που είναι πρόθεμα του R (π).

Κατασκευή αλγόριθμου για το Query(T)

1. αλγόριθμος που ανακτά το μέγιστο πρότυπο λεξικού το οποίο είναι πρόθεμα του Y



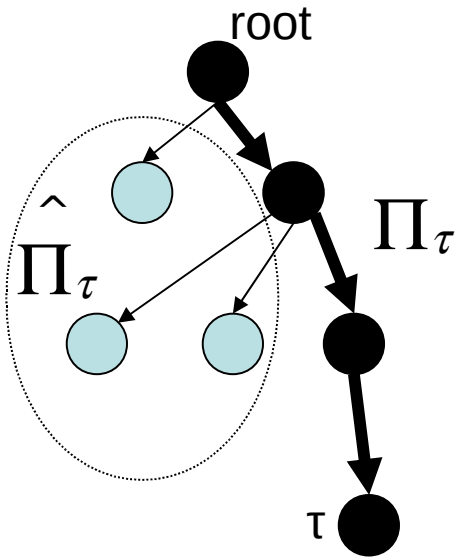
2. αλγόριθμος που δίνει τη λίστα όλων των πρότυπων λεξικού (από το μέγιστο στο ελάχιστο) που είναι προθέματα του Y



3. βάζουμε κάθε δυνατό επίθεμα του T στη θέση του Y και εκτελούμε τον παραπάνω αλγόριθμο όποτε έχουμε έναν αλγόριθμο για το Query (T)

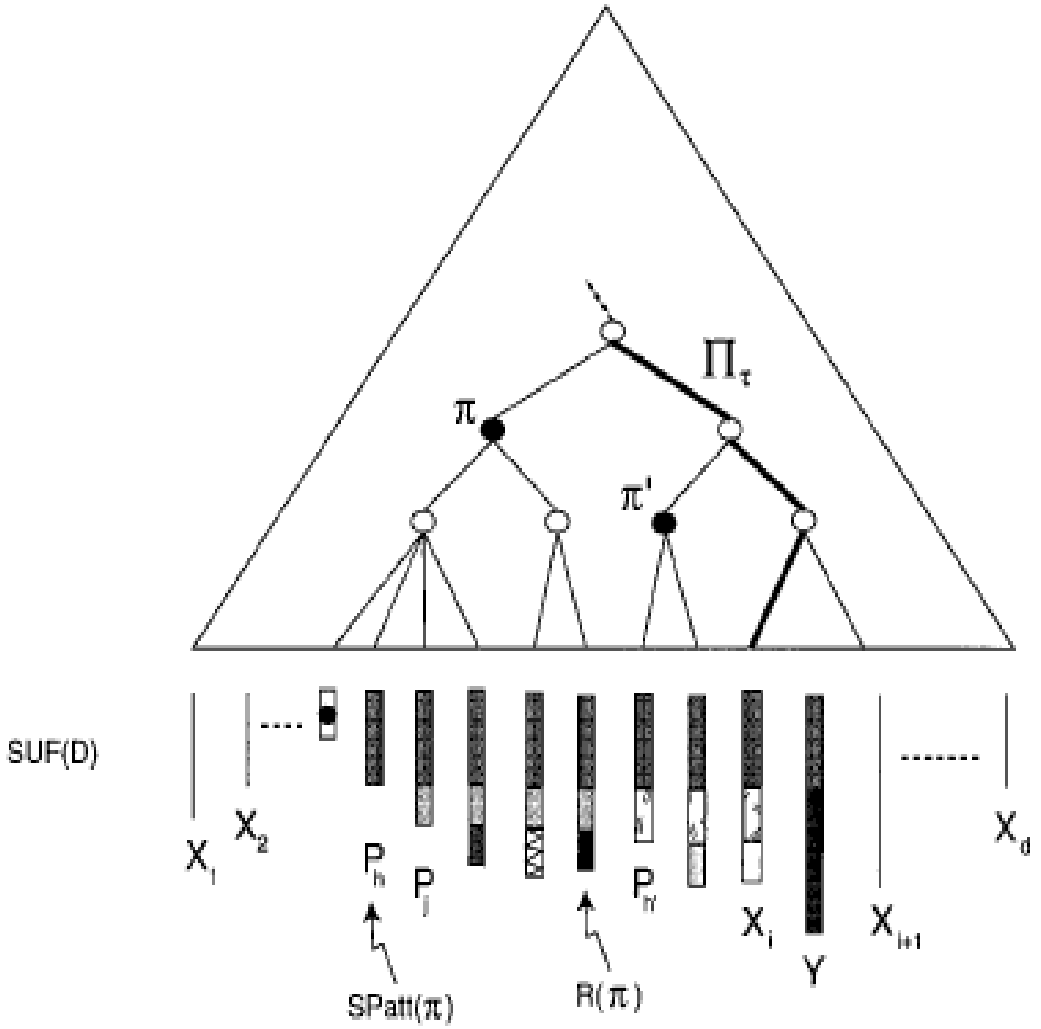
Κατασκευαστικός ορισμός Π_τ , $\hat{\Pi}_\tau$

Π_τ είναι το ανοδικό μονοπάτι στο SBT_D που συνδέει τον κόμβο τα στη ρίζα του Sb-δεντρου.



Το διαταγμένο σύνολο $\hat{\Pi}_\tau$ μπορεί να ληφθεί με το να διασχίσουμε το μονοπάτι Π_τ προς τα πάνω και τη λήψη σε κάθε κόμβο όλων των αριστερών αδερφών του, που διαβάζονται από το δεξί στο αριστερό.

Θεώρημα 4,2. Λαμβάνοντας υπόψη ένα string Y και τη λεξικογραφική θέση του i στο SUF (D), το longest pattern στο D που είναι ένα πρόθεμα του Y (ενδεχομένως) είτε ανήκει στο φύλλο Sb - δέντρου που περιέχει το string Xi , για παράδειγμα τ , ή κατεβαίνει από τον πρώτο κόμβο π στο $\hat{\Pi}_\tau$ έτσι ώστε SPatt (π) είναι ένα πρόθεμα του Y.



Παράδειγμα για θεώρημα 4.2

το longest pattern

είτε θα είναι το $X_i \equiv \tau$, ΟΧΙ

είτε θα είναι το Spatt (π_j) όπου π είναι το 1ο π που θα βρεθεί ανεβαίνοντας από το τ και θα ικανοποιεί την συνθηκη

Spat (π') = $P_{h'}$, ΟΧΙ

Spat (π') = P_h

είναι το πρόθεμα που θέλουμε

1. Αλγοριθμος που ανακτά το μέγιστο πρότυπο λεξικού το οποίο είναι πρόθεμα του Y

P_{long} = το μέγιστο πρότυπο στο D που είναι prefix του T

1η Φάση

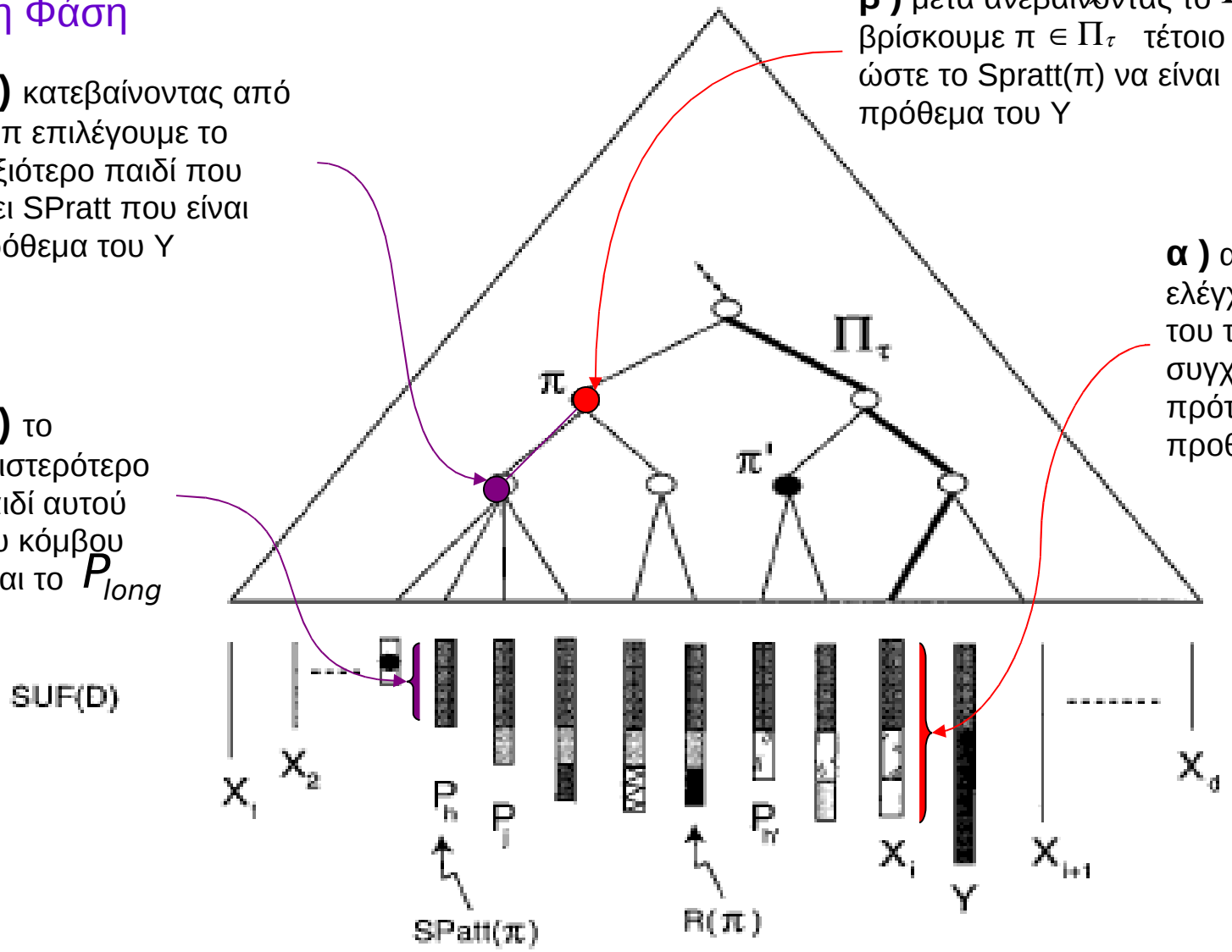
2η Φάση

α) κατεβαίνοντας από το π επιλέγουμε το δεξιότερο παιδί που έχει SPratt που είναι πρόθεμα του Y

β) μετά ανεβαίνοντας το Π_τ βρίσκουμε $\pi \in \Pi_\tau$ τέτοιο ώστε το Spratt(π) να είναι πρόθεμα του Y

α) αρχικά ελέγχουμε τα strings του τ αν είναι συγχρόνως πρότυπα του D και προθέματα του Y

β) το αριστερότερο παιδί αυτού του κόμβου είναι το P_{long}



procedure Find-Long(Y, i, lcp);

(1) Load the SB-tree leaf, τ, containing X_i; /* Let $\mathcal{S}_\tau = \{X_s, \ldots, X_i, \ldots\}$ */
(2) **For** $j = i$ **downto** s **do**
(3) **if** $X_j \in D$ and $|X_j| \leq lcp$ **then** return(X_j);
(4) **if** $j > s$ **then** $lcp = \min\{LCP(X_{j-1}, X_j), lcp\}$;
endfor

(5) Load the parent of τ, say π; /* Upward Search */
(6) **while true do** /* Let $\pi_1, \ldots, \pi_g$ be π's children, where $\pi_{h+1} \in \Pi_\tau$ */
/* and let $\mathcal{S}_\pi = \{S_1, \ldots S_{2h+1}, S_{2h+2} \ldots\}$ */
(7) **For** $j = h$ **downto** 1 **do**
(8) $lcp = \min\{LCP(S_{2j}, S_{2j+1}), lcp\}$;
(9) **if** $|\texttt{SPattchild}(j)| \leq lcp$ **then** $\pi = \pi_j$; Load node π; **exit-while**;
(10) $lcp = \min\{LCP(S_{2j-1}, S_{2j}), lcp\}$;
endfor
(11) **if** $\pi =$ root **then return**(Λ) **else** $\pi = parent(\pi)$; Load node π;
endwhile

(12) **while** π is not a leaf **do** /* Downward Search */
(13) **For** $j = g$ **downto** 1 **do** /* Let $\mathcal{S}_\pi = \{S_1, S_2, \ldots, S_{2g}\}$ */
(14) **if** $|\texttt{SPattchild}(j)| \leq lcp$ **then exit-for**;
(15) $lcp = \min\{LCP(S_{2j-2}, S_{2j-1}), LCP(S_{2j-1}, S_{2j}), lcp\}$;
endfor
(16) $\pi = j$-th child of the current node π; Load new node π;
endwhile

(17) **For** $j = f'$ **downto** s' **do** /* Let $\mathcal{S}_\pi = \{X_{s'}, X_{s'+1}, \ldots, X_{f'}\}$ */
(18) **if** $X_j \in D$ and $|X_j| \leq lcp$ **then** return(X_j);
else $lcp = \min\{LCP(X_{j-1}, X_j), lcp\}$;
endfor

1 – 4

Φάση 1 α

6 – 11

Φάση 1 β

12 – 18

Φάση 2 α , β

Πολυπλοκότητα Find - long και ανάγωγη σε Find - all

Λήμμα 4,4. Έχοντας ένα string Y, τη λεξικογραφική της θέση i στο SUF (D) και την τιμή LCP(Xi, Y), ο αλγόριθμος **Find – Long (Y, i, LCP(Xi, Y))** ανακτά το μέγιστο πρότυπο στο D που είναι ένα πρόθεμα του Y, σε $O(\log_B d)$ προσβάσεις δίσκου στη χειρότερη περίπτωση , όπου d το συνολικό μέγεθος του λεξικού .

Θεώρημα 4,5. Έχοντας ένα string Y, τη λεξικογραφική της θέση i στο SUF (D) και την τιμή LCP(Xi, Y), ο αλγόριθμος **find - All(Y, i , LCP(Xi, Y))** ανακτά όλα τα πρότυπα στο D που είναι προθέματα του Y, στη χειρότερη περίπτωση οι προσβάσεις δίσκου είναι $O(q\log_B d)$, όπου το q είναι ο αριθμός ανακτημένων πρότυπων .

Αλγόριθμος για εξωτερική μνήμη

Πρόβλημα :

Το Spat πρέπει να ανανεωθεί *μόνο όταν* το Φ_{π}^{tot} αλλάζει μετά από μια εισαγωγή ή διαγραφή των επιθεμάτων του P .

δηλαδή : (ι) ολόκληρο το πρότυπο P παρεμβάλλεται ή διαγράφεται σε ένα από τα φύλλα που κατεβαίνουν από π

(ΙΙ) μια διάσπαση ή μια λειτουργία συγχώνευσης εκτελείται στο π .

Έστω π είναι ένας κόμβος στο SBT_D και $\delta_1, \delta_2, \ldots, \delta_g$ είναι τα παιδιά του. Το SPatt(π) είναι ίσο με το αριστερότερο πρότυπο SPattchild(j) που είναι ένα πρόθεμα του R(π), για j=1..., g . Εάν αυτό το σχέδιο δεν υπάρχει, SPatt(π) = Λ . Ένας τέτοιος υπολογισμός SPatt(π) δεν απαιτεί προσβάσεις δίσκων.

```
procedure Query(T);

    /* Phase (1) */
(1) (τ1, pos1, lcp1) = SB-Search-Up-Down(T[1,n], root, 0);
(2) For i = 1 to n − 1 do                    /* Let X_p(i) be the pos_i-th string in S_τi */
(3)     If lcp_i = LCP(X_p(i), T[i,n]) then γ = succ(X_p(i)) else γ = succ(X_p(i)+1);
(4)     (τ_i+1, pos_i+1, lcp_i+1) = SB-Search-Up-Down(T[i+1,n], γ, max{lcp_i − 1, 0});
    endfor

    /* Phase (2) */
(5) For i = 1 to n do
(6)     ℓ_i = LCP(X_p(i), T[i,n]);           /* Directly available in SBT_D */
(7)     Find-All(T[i,n], p(i), ℓ_i);
    endfor
```

Η φάση 1 αναζητεί την λεξικογραφική θέση όλων των επιθεμάτων κειμένων στο διαταγμένο σύνολο SUF (D) με τη χρήση της διαδικασίας SB-Search-Up-Down (λήμμα 2.4) .

Η φάση 2 εκτελεί τον αλγόριθμο Find-All(T[i,n],p(i),ℓ_i),

όπου p(i) είναι η θέση του T[i, n] στο SUF(D), και $\ell_i = \mathrm{LCP}(X_{p(i)}, T[i,n])$.

Αποτελέσματα - Πολυπλοκότητες

Θεώρημα 5,2. Ένα σχέδιο P[1, p] που μπορεί να παρεμβληθεί ή να διαγραφεί από το D απαιτεί $O(p \log_B (d+p))$ στη χειρότερη περίπτωση προσβάσεις δίσκου.

Θεώρημα 5,3. Μια ερώτηση σε ένα αυθαίρετο κείμενο T[1, n] μπορεί να απαντηθεί σε $O((n+tocc)\log_B d)$ στη χειρότερη περίπτωση προσβάσεις δίσκου . (tocc : ο αριθμός των εμφανίσεων (occurences))

Θεώρημα 5,4. Ένα λεξικό σχεδίων, συνολικού μήκους d, μπορεί να διατηρηθεί κάτω από την εισαγωγή και τη διαγραφή ενός σχεδίου P[1, p], σε O(p log d) χρόνο στη χειρότερη περίπτωση . Ψάχνοντας για όλα τα occurences tocc των πρότυπων λεξικών σε ένα αυθαίρετο κείμενο T[1, n] απαιτεί O(n+tocc) log d) χρόνο στη χειρότερη περίπτωση. Ο συνολικός απαραίτητος χώρος είναι Θ (d).

Αναφορές

Dynamic Dictionary Matching in External Memory, *Paolo Ferragina and Fabrizio Luccio, Information and Computation*