Εισαγωγή στη Βιοπληροφορική

Μερικές διαφάνειες από διαφάνειες βιβλίου Neil C. Jones, Pavel A. Pevzner, Εισαγωγή στους Αλγορίθμους Βιοπληροφορικής

Και από το βιβλίο: Dan Gusfield Algorithms on Strings, Trees and Sequences, Cambridge University Press,

(

Βασική Προεπεξεργασία (1) (D. Gusfield)

- $Z_i(S)$= το μήκος της μεγαλύτερης υποσυμβολοσειράς του S, που αρχίζει στο i και ταιριάζει πρόθεμα του S.
- Z-box at i= το σύνολο χαρακτήρων που αρχίζουν από i και τελειώνουν στη θέση $i+Z_i(S)-1$.
- Για κάθε i, r_i συμβολίζει το δεξιότερο άκρο των Z-boxes που ξεκινά από ή πριν τη θέση i. Διαφορετικά, r_i είναι η μεγαλύτερη τιμή του $j+Z_j-1$ για κάθε $2<j<=i$. Το αριστερό άκρο (j) το συμβολίζουμε ως l_i.

από το βιβλίο: Dan Gusfield Algorithms on Strings, Trees and Sequences, Cambridge University Press,

Βασική Προεπεξεργασία (2)

Δοθέντων Z_i για i<=k-1 και r, l για Z_{k-1}:

1. If k>r, find Z_k explicitly. If Z_k>0, r=k+Z_k-1, l=k.
2. If k<=r, S[k..r] matches S[k'..Z_l] and the substring at k matches a prefix of S of length >=min($Z_{k'}$,r-k+1) (k'=k-l+1)

   2.a. If $Z_{k'}$ < |S[k..r]| then Z_k=$Z_{k'}$, r,l remain unchanged

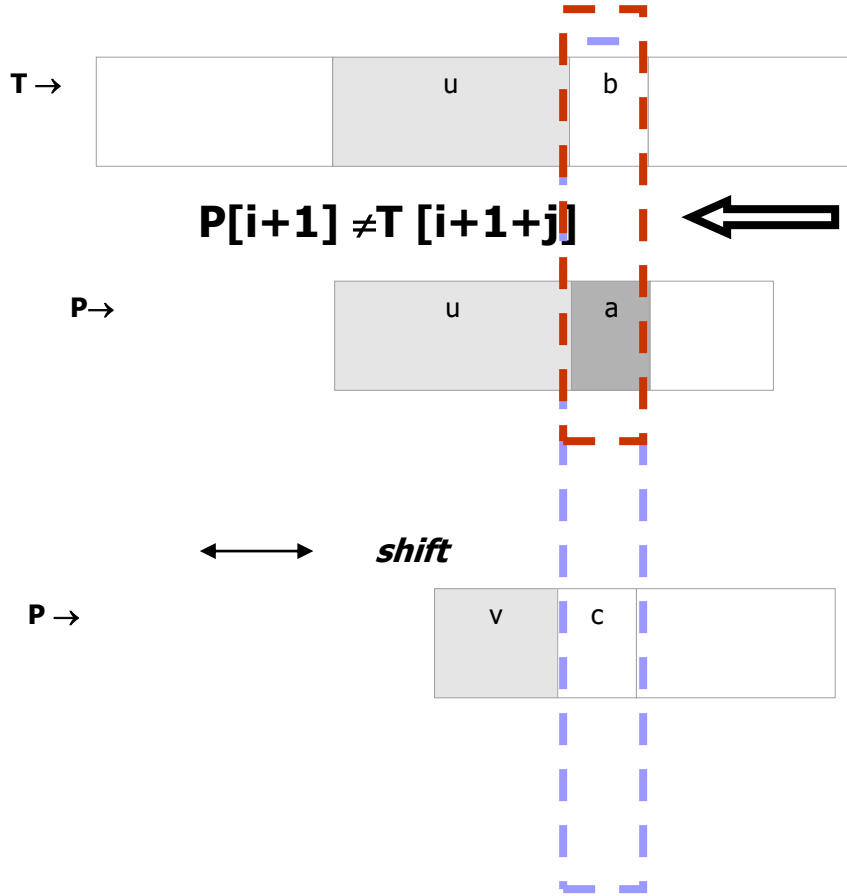
   2.b. Compare the characters starting at r+1 of S with the characters starting at |S[k..r]| until a mismatch. Say the mismatch occurs at q>=r+1. Then Z_k is q-k, r is set to q-1 and l is set to k.

από το βιβλίο: Dan Gusfield Algorithms on Strings, Trees and Sequences, Cambridge University Press,

Βασική Προεπεξεργασία (3)

- Εφάρμοσε τον αλγόριθμο στην ακολουθία P$T
- Κάθε τιμή Z_i=m, i>m σηματοδοτεί match στη θέση i-m-1.
- Η μέθοδος μπορεί να υλοποιηθεί έτσι ώστε να απαιτεί επιπλέον (του χώρου αποθήκευσης P, T), O(m) χώρο.
- Είναι μέθοδος που οι πολυπλοκότητές της είναι ανεξάρτητες από το μέγεθος του αλφαβήτου (ίδια ιδιότητα έχει ο αλγόριθμος Boyer Moore, Knuth Morris Pratt, όχι όμως ο Shift Or και ο αλγόριθμος Aho-Corasick)

Ο αλγόριθμος Knuth-Morris-Pratt



P[1… i] = T [j…i+j-1] = u

⇩

ιδέα: ας στοιχίσουμε το μέγιστο πρόθεμα **prefix(P)=v** με το αντίστοιχο τμήμα από το επίθεμα **u** της ακολουθίας

Ας δούμε τον αλγόριθμο στην πράξη

1ο βήμα: Στοιχίζουμε την ακολουθία και το πρότυπο & συγκρίνουμε τους χαρακτήρες από αριστερά προς τα δεξιά

x	y	a	b	c	x	a	b	c	x	a	d	c	d	q	f	e	g	a	g	t	a	c	g
		1	2	3	4	5	6	7	8	9													
		a	b	c	x	a	b	c	d	e													

2ο βήμα: Μετατοπίζω το πρότυπο κατά 4 θέσεις

x	y	a	b	c	x	a	b	c	x	a	d	c	d	q	f	e	g	a	g	t	a	c	g
						1	2	3	4	5	6	7	8	9									
						a	b	c	x	a	b	c	d	e									

Υλοποίηση ΚΜΡ (1)

- Ορίζω ως $sh_i(P)$ το μήκος του μεγαλύτερου επιθέματος του P[1..i] που ταιριάζει ένα πρόθεμα του P, με την επιπλέον συνθήκη ότι οι χαρακτήρες P[i+1] και $P[sh_i(P)+1]$ είναι διαφορετικοί.
- Η θέση j>1 αντιστοιχίζεται στο i εάν $i=j+Z_j(P) -1$.
- Για κάθε i>1 $sh_i(P)=i-j+1$;οπου j είναι η μικρότερη θέση που αντιστοιχίζεται στο i.

Υλοποίηση KMP (2)

For i=1 to m to $sh_i(P)=0$;

For j=m downto 2 do

begin

    $i=j+Z_j(P)-1$;

    $sh_i(P)=Z_j(P)$;

end

Real Time KMP

- Ορίζω ως $sh_i(P,x)$ το μήκος του μεγαλύτερου επιθέματος του P[1..i] που ταιριάζει ένα πρόθεμα του P, με την επιπλέον συνθήκη ότι ο χαρακτήρας $P[sh_i(P,x)+1]$ είναι ο x.
- Όπως πριν:

```
For i=1 to m to sh_i(P)=0;
For j=m downto 2 do
begin
      i=j+Z_j(P)-1;
      x=P[Z_j(P)+1]
      sh_i(P,x)=Z_j(P);
  end
```

Ανάλυση του αλγορίθμου Knuth-Morris-Pratt σε χρόνο

- Πολυπλοκότητα μεθόδου: O(n+m), όπου |T|=n & |P|=m

1. Σε χρόνο O(m) υπολογίζω, την πληροφορίας που χρειάζεται για το πρότυπο.
2. Σε μία συλλογή από shift και compare φάσεις, έχω O(n) shifts και σε κάθε compare εμπλέκω παλιό χαρακτήρα μία φορά και καινούριους.
3. Σύνολο O(n+m)

Ο αλγόριθμος Boyer-Moore

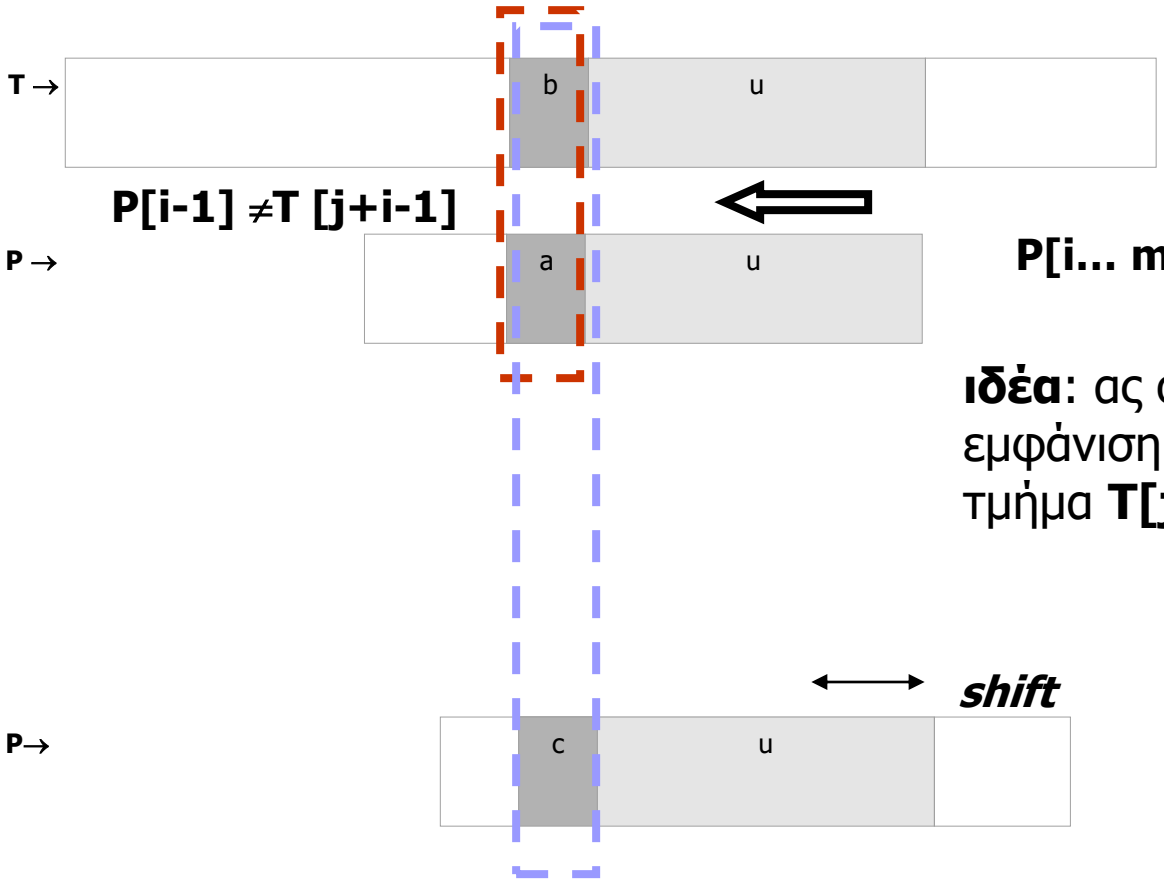
1η ιδέα: Στοιχίζουμε την ακολουθία και το πρότυπο & συγκρίνουμε τους χαρακτήρες από δεξιά προς τα αριστερά

g	c	a	t	c	g	c	a	g	a	g	a	g	t	a	t	a	c	a	g	t	a	c	g
1	2	3	4	5	6	7	8																
g	c	a	g	a	g	a	g																

2η ιδέα: Σε κάθε mismatch μετατοπίζω το πρότυπο περισσότερες από 1 θέσεις βάσει 2 κανόνων

g	c	a	t	c	g	c	a	g	a	g	a	g	t	a	t	a	c	a	g	t	a	c	g
	1	2	3	4	5	6	7	8															
	g	c	a	g	a	g	a	g															

α' κανόνας: "*good suffix shift"*

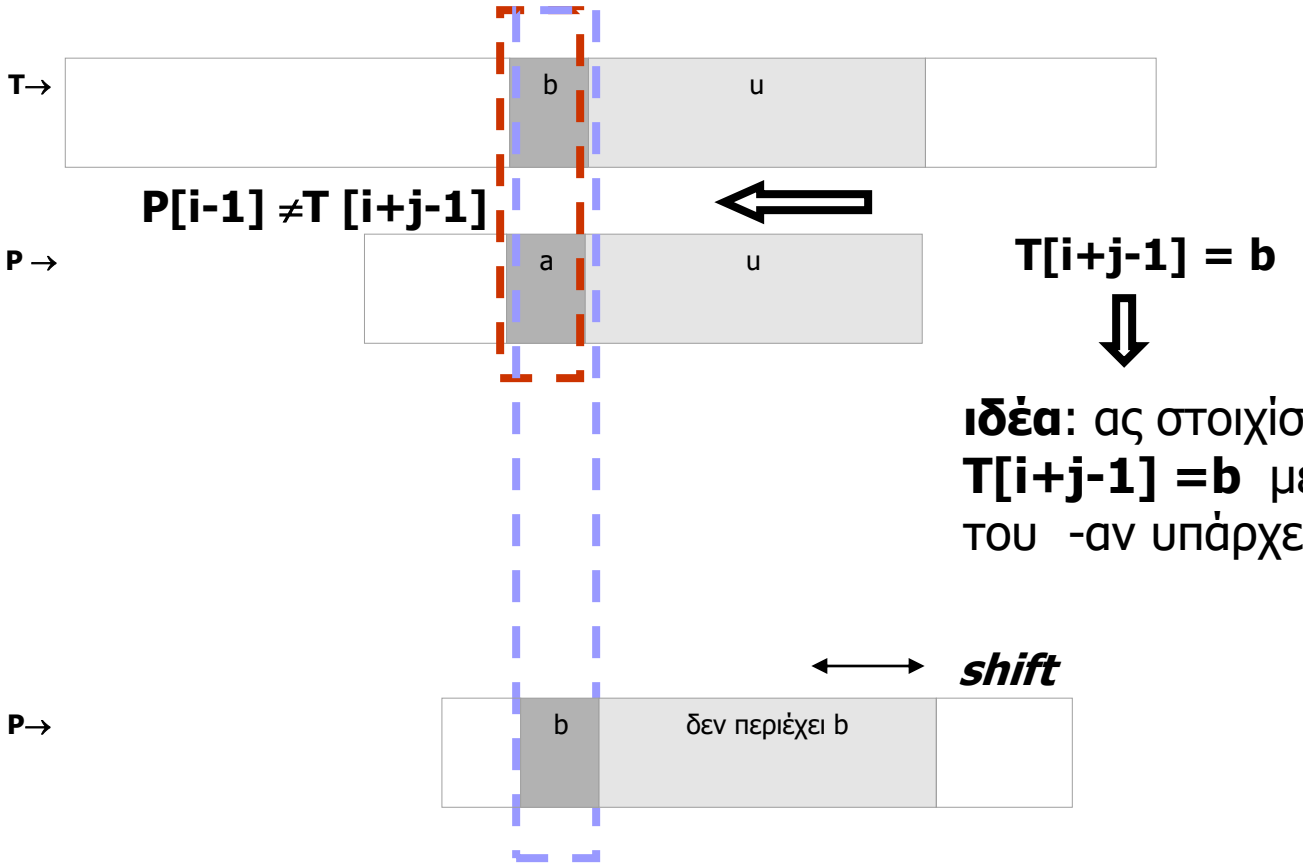


P[i... m] = T[j+i...j+m]

⇩

ιδέα: ας στοιχίσουμε την επόμενη εμφάνιση του **u** με το αντίστοιχο τμήμα **T[j+i, ..., j+m]**

β' κανόνας: *"bad character shift"*



T[i+j-1] = b

⇩

ιδέα: ας στοιχίσουμε το χαρακτήρα **T[i+j-1] =b** με τη δεξιότερη εμφάνισή του -αν υπάρχει- στο πρότυπο **P**

Υλοποίηση “bad character shift”

- (Simple Shift) Χρησιμοποίησε ένα πίνακα μεγέθους m x |Σ| και σάρωσε το πρότυπο από αριστερά προς τα δεξιά.

- (Extended Shift) Σάρωσε το πρότυπο από δεξιά προς τα αριστερά και για κάθε χαρακτήρα δημιούργησε μία λίστα από εμφανίσεις. Σάρωσε την κατάλληλη λίστα όταν εντοπιστεί.

Υλοποίηση “goof suffix shift” (1)

- Έστω L(i) η μεγαλύτερη θέση στο P, έτσι ώστε η P[i..m] να ταιριάζει ένα επίθεμα του P[1..L[i]] με τον επιπλέον περιορισμό ότι ο χαρακτήρας που προηγείται του επιθέματος θα πρέπει να είναι διαφορετικός του P[i-1].
- Ορίζουμε ως l’(i) το μήκος του μεγαλύτερου επιθέματος του P[i..m] που είναι επίσης πρόθεμα του P.

Υλοποίηση “goof suffix shift” (2)

- Έστω $N_j(P)$ το μήκος του μεγαλύτερου επιθέματος της συμβολοσειράς P[1..j] που είναι επίθεμα του P .

  $N_j(P)= Z_{m-j+1}(P^r)$

  Από τον ορισμό αυτό προκύπτεί ότι L(i) είναι η μεγαλύτερη τιμή j<m έτσι ώστε

  $$N_j(P)=|P[i..m]|$$

  Τέλος l’[i] είναι το μεγαλύτερο j<=|P[i..m]|, έτσι ώστε $N_j(P)=j$

Υλοποίηση “goof suffix shift” (3)

```
for j=1 to m-1 do
   begin
   i=m-Nj(P)+1;
   L(i):= j;
   end
```

Ανάλυση του αλγορίθμου Boyer-Moore σε χρόνο

- Πολυπλοκότητα μεθόδου: O(n*m), όπου |T|=n & |P|=m

1. Οι απαιτούμενοι πίνακες υπολογίζονται σε O(m+σ) χρόνο
2. Σε πραγματικές εφαρμογές απαιτούνται "3n" συγκρίσεις.
3. Για μεγάλο αλφάβητο |Σ|= ($\approx|\mathrm{pattern}|$), απαιτούνται O(n/m) συγκρίσεις.
4. Χωρίς match και μόνο με τον good suffix rule ο χρόνος είναι O(n).
5. Μόνο με τον bad character στη μέση περίπτωση υπογραμμικό.
6. Υπάρχουν παραλλαγές με worst-case O(n+m) χρόνο

Ο αλγόριθμος Shift-Or

- Ο αλγόριθμος χρησιμοποιεί αριθμητικές τεχνικές:

1. Έστω για κάθε char c $\in$ Σ, το διάνυσμα S_c μεγέθους m=|P|, που αποθηκεύει τις εμφανίσεις του c μέσα στο πρότυπο (με 0 p προσδιορίζεται η εμφάνιση),

2. Ο πίνακας R[mxn]: 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 στο j-οστό χαρακτήρα του T.

3. $R_{j+1} = \text{Shift}\ (R_j\ )\ \text{OR}\ S_{T[j+1]}$

Ας δούμε τον αλγόριθμο στην πράξη

1ο βήμα: Υπολογίζω τα διανύσματα S_C για το πρότυπο x=gcagagag

	Sa	S_c	S_g	S_t
g	1	1	0	1
c	1	0	1	1
a	0	1	1	1
g	1	1	0	1
a	0	1	1	1
g	1	1	0	1
a	0	1	1	1
g	1	1	0	1

...Ας δούμε τον αλγόριθμο στην πράξη

2ο βήμα: Υπολογίζω τις τιμές του πίνακα R, σύμφωνα με τον τύπο:
R_{j+1} = Shift (R_j) Or $S_{T[j+1]}$

		1	2	3	4	5	6	7	8	9	10	11	12	13
		g	c	a	t	c	g	c	a	g	a	g	a	g
1	g	0	1	1	1	1	0	1	1	0	1	0	1	0
2	c	1	0	1	1	1	1	0	1	1	1	1	1	1
3	a	1	1	0	1	1	1	1	0	1	1	1	1	1
4	g	1	1	1	1	1	1	1	1	0	1	1	1	1
5	a	1	1	1	1	1	1	1	1	1	0	1	1	1
6	g	1	1	1	1	1	1	1	1	1	1	0	1	1
7	a	1	1	1	1	1	1	1	1	1	1	1	0	1
8	g	1	1	1	1	1	1	1	1	1	1	1	1	0

Ανάλυση του αλγορίθμου Shift-Or σε χρόνο

- Πολυπλοκότητα μεθόδου: O(n+m), όπου |T|=n & |P|=m

1. Σε χρόνο O(m*σ) υπολογίζω, τα διανύσματα S_C