

# Συμβολοσειρές

- Σημαντικές λόγω των εφαρμογών τους
  - Επεξεργασία κειμένου
  - Διαδίκτυο
  - Επεξεργασία βιολογικών δεδομένων
    - DNA ως συμβολοσειρές 4 συμβόλων: C (κυτοσίνη), G (γουανίνη), T (θυμίνη), A (αδενίνη)

# Ορισμοί

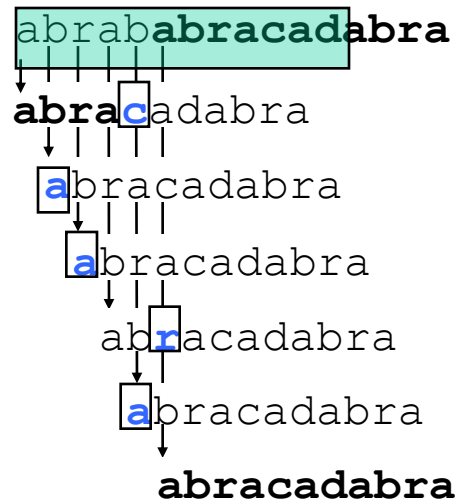
- $\Sigma$ : αλφάβητο
- $\Sigma^*$ : όλες οι συμβολοσειρές πεπερασμένου μήκους, συμπεριλαμβανομένης της κενής  $\epsilon$
- $|x|$  μήκος συμβολοσειράς  $x$
- $x=αβ$ ,  $y=γδε$  συμβολοσειρές  
 $xy=αβγδε$  συμβολοσειρά συνενώσεώς τους
- $x=αβγδεζ$ 
  - Πρόθεμα μήκους  $i$ : τα  $i$  πρώτα σύμβολα, πχ,  $i=3$ ,  $αβγ$
  - Πρόσφυμα μήκους  $j$ : τα  $j$  τελευταία σύμβολα, πχ,  $j=4$ ,  $γδεζ$

# Ακριβές Ταίριασμα Προτύπου (Exact Pattern Matching)

- «Δοθεισών ενός κειμένου  $T$ , μήκους  $n$ , και ενός προτύπου  $P$ , μήκους  $m$ , απαιτείται ο εντοπισμός όλων των στιγμιοτύπων του  $P$  στο  $T$ .»
  - Π.χ. για  $T=ααβαββαβββα$  και  $P=βα$ , έχουμε  
 $Aα**βα**β**βα**ββ**βα**$
- Εφαρμογές:
  - Κειμενογράφοι
  - αναζήτηση υποδομής σε ακολουθία DNA
  - αναζήτηση λέξεως στα περιεχόμενα μιας σελίδας διαδικτύου

# Απλοϊκή Λύση

- Αντιπαραβολή του προτύπου, από όλες τις  $n-m+1$  δυνατές θέσεις εκκινήσεως



# Κώδικας

**Algorithm** bruteForcePM(char [] T, char [] P)

**Input:** Κείμενο T και πρότυπο P

**Output:** Εντοπισμός του P στο T

```
1. n = T.length; m = P.length;
2. for (i = 1; i = n - m + 1; i++) { // i η τρέχουσα θέση του παραθύρου
3.   j = 1; // δείκτης θέσεως εντός παραθύρου-προτύπου
4.   while ((j <= m) && (T[i+j-1] == P[j]))
5.     j++;
6.   if (j == m + 1)
7.     return i; // βρέθηκε η αρχική θέση του προτύπου
8. }
9. return -1; // δεν υπάρχει
```

# Ανάλυση

- Πολυπλοκότητα  $O(nm)$

- Χειρότερη Περίπτωση

Σε κάθε αντιπαραβολή να γίνονται  $m-1$  επιτυχημένες συγκρίσεις και να σημειώνεται αποτυχία στην τελευταία. Πχ,

T = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

P = aaaaab

$n-m+1$  αντιπαραβολές των  $m$  συγκρίσεων  
έκαστη

# Knuth-Morris-Pratt

- Ανακαλύφθηκε ανεξάρτητα από τους Knuth-Morris και Pratt και, κατόπιν, παρουσιάστηκε ως κοινή εργασία το 1977
- Στηρίζεται στην γραμμική  $-O(m)$ , δηλ.-επεξεργασία του *προτύπου και όχι του κειμένου*, με σκοπό
  - την εξαγωγή βοηθητικού πίνακα  $f$ , και
  - την χρήση της πληροφορίας του  $f$  για την μείωση του αριθμού των αντιπαραβολών

# Πίνακας Αποτυχίας $f$

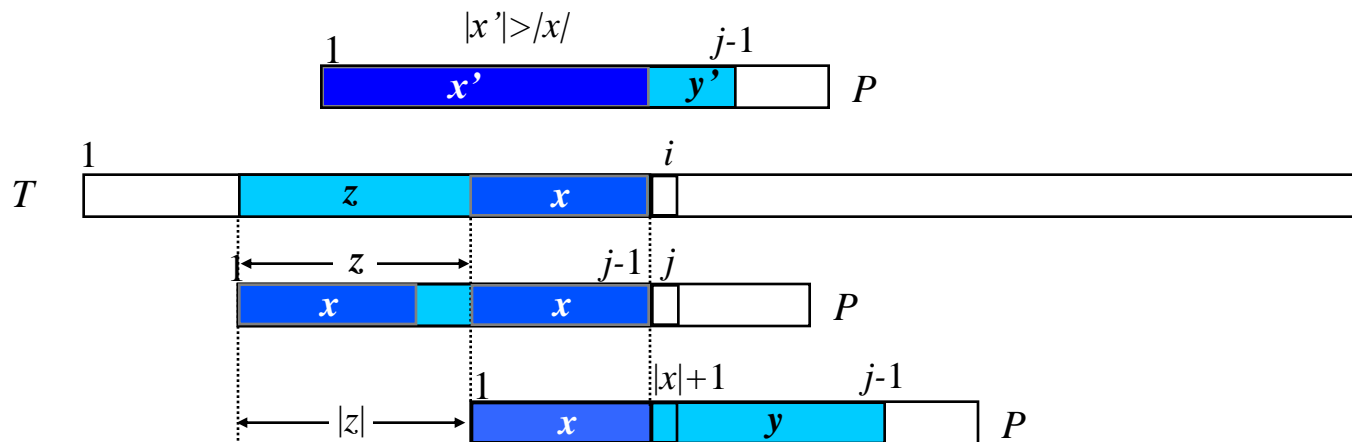
- $f[j] =$  μέγιστο πρόθεμα τού  $P[1..j]$  που αποτελεί και πρόσφυμά του

1	2	3	4	5	6	7	8	9	10	11
a										
<b>0</b>										

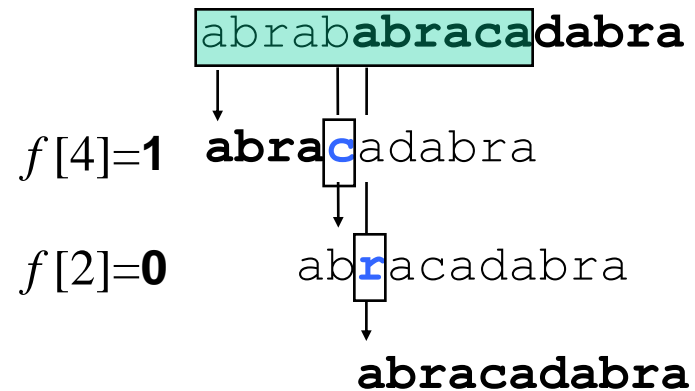


# Ποια η χρήση του $f$

- Εάν έχουν ταιριαστεί οι  $j-1$  πρώτοι χαρακτήρες, ενώ υπάρχει διαφορά στον  $j$ -στό, τότε
  - μπορούμε να ολισθήσουμε την αντιπαραβολή κατά  $j-1-f[j-1]$  θέσεις
  - δεν χάνουμε κανένα στιγμιότυπο!

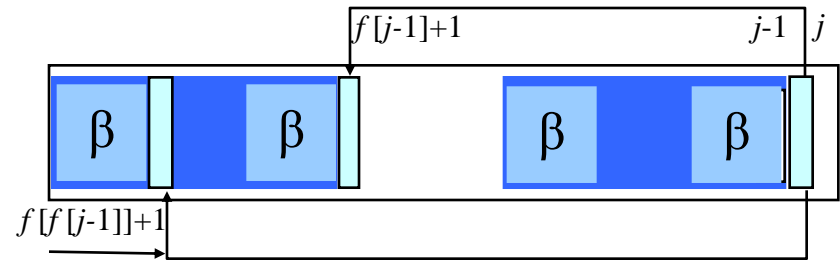


# Παράδειγμα



# Τρόπος Υπολογισμού

- Εάν  $P[j]=P[f[j-1]+1]$  τότε  $f[j] = f[j-1] + 1$
- Διαφορετικά,
  - είτε είναι το μέγιστο κοινό πρόθεμα του  $P[1..f[j-1]]$  ( $P[1..f[f[j-1]]]$ ) συν τον επόμενο χαρακτήρα
  - είτε συνεχίζουμε την αναζήτηση αναδρομικά...



**Algorithm** kmpPM(char[] T, char[] P)

```
1. n = T.length; m = P.length;
2. int[] f = failureKMP(P, new int[m]);
3. int i = 1, j = 1;
4. while (i < n){
5.   if (P[j] == T[i]){ // ταίριασμα
6.     if (j = m)      // εύρεση!
7.       return i - m;
8.   else{           // μετακίνηση εντός παραθύρου 1 θέση δεξιότερα
9.     i++;
10.    j++;
11.   }
12. else if (j > 1)  // βρέθηκε ταίριασμα εντός παραθύρου
13.   j = f[j-1]      // οπότε γίνεται χρήση της failure function
14. else
15.   i++;
16. }
17. return -1;
18.
19. int[] failureKMP(int[] P, int[] f){
20. int m = P.length, k = 0;
21. f[1] = 0;
22. for (j = 2; j <= m; j++){
23.   while ((k > 0) && (P[k+1] != P[j])) // καθορίζει και την πολυπλοκότητα υπολογισμού
24.     k = f[k]; // P[f[k]+1] ≠ P[j]
25.   if (P[k+1] == P[j])
26.     k++; // κρίσιμο σημείο!!!
27.   f[j] = k;
28. }
29. return f;
```

# Πολυπλοκότητες

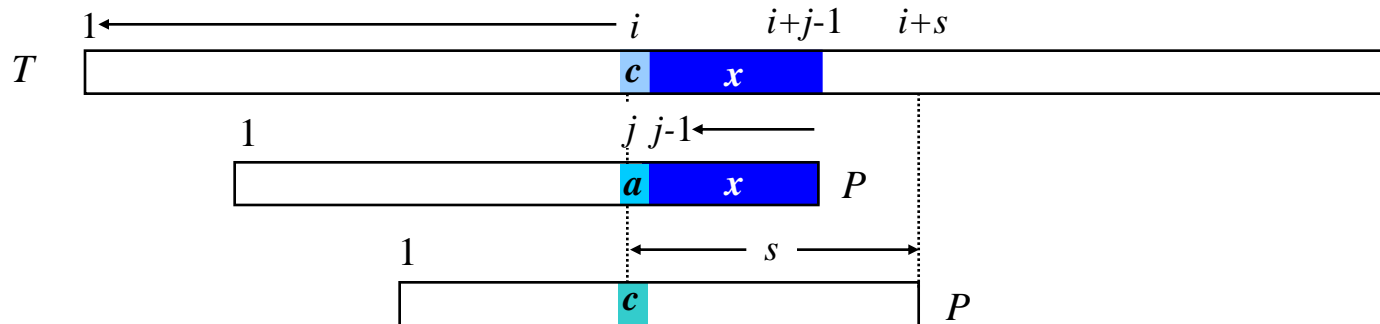
- Ο υπολογισμός της  $f$  γίνεται σε χρόνο  $O(m)$ 
  - Η γραμμή 24 του kmpPM καθορίζει την πολυπλοκότητα
- Οι συγκρίσεις είναι μεταξύ  $n$  και  $2n$  διότι κάθε φορά
  - εάν η σύγκριση είναι επιτυχής, μεταβαίνουμε στον επόμενο χαρακτήρα του κειμένου και ο αντίστοιχος δείκτης  $i$  ποτέ δεν μειούται
  - εάν είναι ανεπιτυχής, ολισθαίνουμε τουλάχιστον κατά μία θέση το παράθυρο στο κείμενο, οπότε μία θέση του κειμένου εγκαταλείπεται οριστικά - ο δείκτης  $j$  ποτέ δεν μειούται

# Boyer-Moore

- Χρησιμοποιείται στο εργαλείο agrep
- Κύρια ιδέα
  - Οι συγκρίσεις να γίνονται *από δεξιά προς τα αριστερά*
  - Όταν σημειώνεται αποτυχία, εφαρμόζονται δύο ευρετικοί κανόνες (heuristics), ώστε βρεθεί η μέγιστη ολίσθηση:
    1. Ευρετικό εμφανίσεως
    2. Ευρετικό ταιριάσματος

# Ευρετικό Εμφανίσεως (Occurrence Heuristic)

- Ταίριασμα του χαρακτήρα  $c$  του κειμένου που προκάλεσε την αποτυχία με την δεξιότερη εμφάνισή του στο πρότυπο.
- Εάν αυτή είναι δεξιότερα της  $j$ , τότε, απλώς, ολισθαίνουμε κατά μία θέση.
- Πολυπλοκότητα  $O(\Sigma)$



# Παράδειγμα

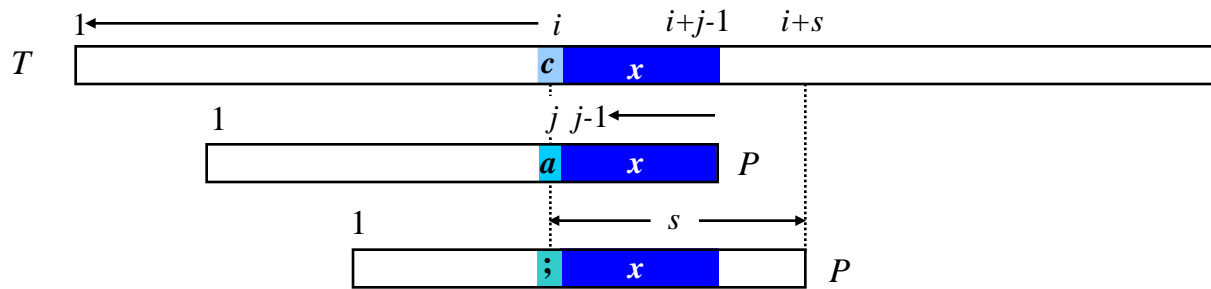
$d'$	a	b	c	d	r	$d = m - d'$	a	b	c	d	r
	11	9	5	7	10		0	2	6	4	1

1	2	3	4	5	6	7	8	9	10	11
a	b	r	a	<b>c</b>	a	d	a	b	r	a

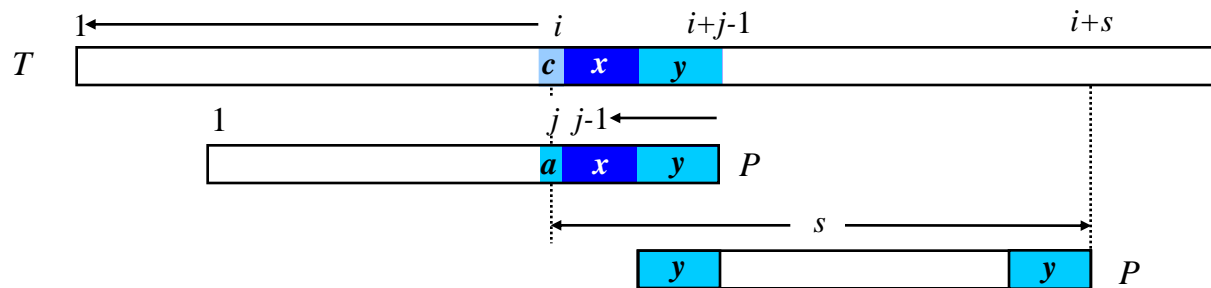


# Ευρετικό Ταίριασματος (Match Heuristic)

- Δεξιότερο μέγιστο ταίριασμα των χαρακτήρων του κοινού προσφύματος με διαφορετικό χαρακτήρα-«κεφαλή» από αυτόν που προκάλεσε την διαφορά.



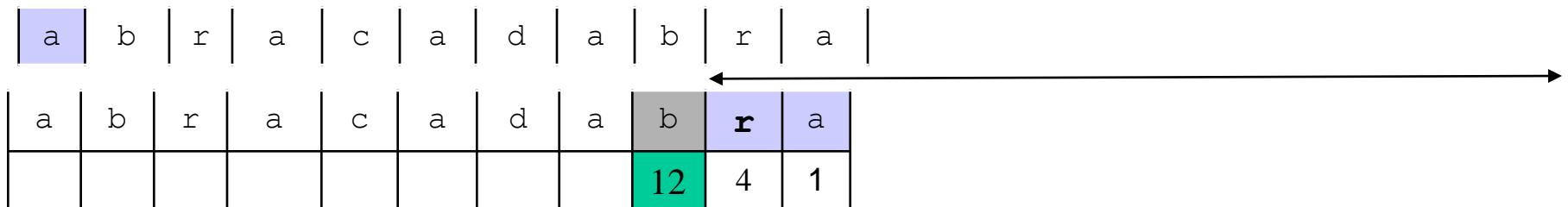
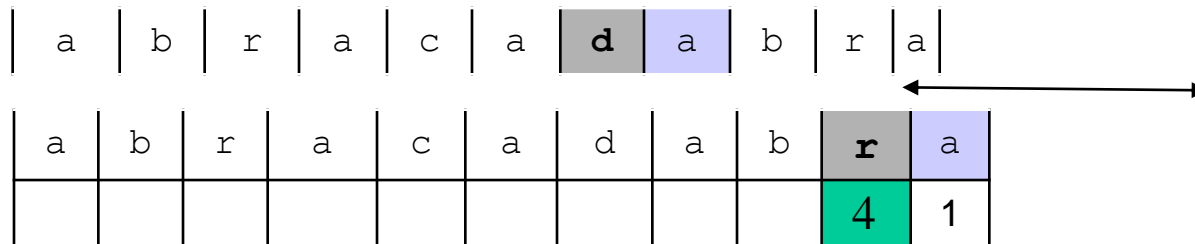
(α)



(β)

# Ευρετικό Ταιριάσματος (συν.)

- Η υλοποίηση είναι πολύ δύσκολη! Σημειωτέον πως και οι ίδιοι οι ερευνητές είχαν δώσει λάθος κώδικα, τον οποίο διόρθωσε το 1980 ο Rytter!

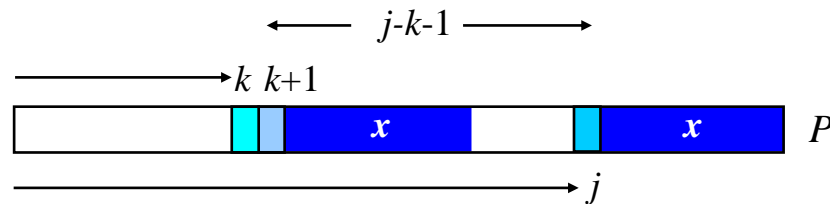


# Ευρετικό Ταιριάσματος (συν.)

- Τελικώς...

a	b	r	a	c	a	d	a	b	r	a
<b>17</b>	<b>16</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>13</b>	<b>12</b>	<b>4</b>	<b>1</b>

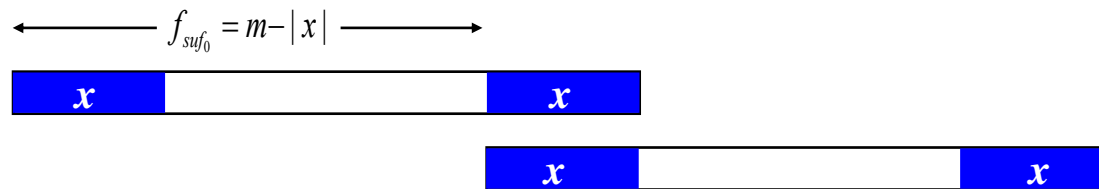
# Μία Εξήγηση των Υπολογισμών



(α)

- Έστω ότι υπολογίζουμε «ανάποδα» τον πίνακα KMP, ώστε
  - Κάθε θέση  $k+1$  μας δίνει την θέση  $j$ , όπου προηγείται του προσφύματος  $x$
  - $P_j \neq P_{k+1}$ , τότε βρήκαμε μία δεξιότερη εμφάνιση του  $x$  σε απόσταση  $j-k-1$

# Μία Εξήγηση των Υπολογισμών (συν.)



(β)

- Για την δεύτερη περίπτωση:
  - εάν  $x$  είναι το μεγαλύτερο πρόσφυμα που είναι και πρόθεμα, τότε για τις  $m - |x|$  πρώτες θέσεις η εικονιζόμενη τοποθέτηση είναι ασφαλής
  - Για τις υπόλοιπες του προσφύματος  $x$ , ο υπολογισμός είναι αναδρομικός...

# Παράδειγμα

	0	1	2	3	4	5	6	7	8	9	10	11
		a	b	r	a	c	a	d	a	b	r	a
fsuf	7	8	9	10	11	10	11	10	11	11	11	12
match		12	12	12	12	12	12	12	12	12	<b>3</b>	<b>1</b>
match		<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>10</b>	<b>10</b>	3	1
+		10	9	8	7	6	5	4	3	2	1	0
match		<b>17</b>	<b>16</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>13</b>	<b>12</b>	<b>4</b>	<b>1</b>

- Η πολυπλοκότητα είναι ανάλογη του KMP...

# Παράδειγμα Τρεξίματος

d

a	b	c	d	r
0	2	6	4	1

m

a	b	r	a	c	a	d	a	b	r	a
17	16	15	14	13	12	11	13	12	4	1

abr**ab**racadabra

abra**c**adab**r**a

abra**c**adab**r**a  
abra**c**adab**r**a

**abracadabra**

Καλύτερο  
το πρώτο

- Την πρώτη φορά έχουμε ολίσθηση βάσει ευρετικού εμφανίσεως  $\max\{d[c]=6, m[r]=4\}=6$
- Την δεύτερη, πλήρες ταίριασμα

# Πολυπλοκότητα

- Υπολογισμός ευρετικού εμφανίσεως  $O(m+|\Sigma|)$  – μειονέκτημα η εξάρτηση από το αλφάβητο
- Υπολογισμός ευρετικού ταιριάσματος
- Συνολικά
  - Χειρότερη Περίπτωση  
 $O(nm)$
  - Μέση Περίπτωση υπογραμμικός!  
 $O(n \log m/m)$  -δηλαδή, αποφεύγει να διαβάσει όλο το κείμενο



# Γενική Συζήτηση

- Στην πράξη, ο Knuth-Morris-Pratt εμφανίζει συμπεριφορά ελάχιστα καλύτερη του απλοϊκού αλγορίθμου
- Ο Boyer-Moore και οι παραλλαγές του είναι η μέθοδος που εφαρμόζεται στην πράξη καίτοι έχει (θεωρητικά) πολύ κακή χειρότερη συμπεριφορά

# Ομοιότητα Συμβολοσειρών

- Υπάρχουν εφαρμογές που αιτούν πόσο όμοιες ή ανόμοιες είναι δύο συμβολοσειρές  $x, y$ . Π.χ.,
  - Όταν βρίσκουμε τα ορθογραφικά λάθη και προσπαθούμε να βρούμε την σωστή ορθογραφία
  - Προγράμματα-«αράχνες» που βλέπουν πόσο όμοιες είναι οι σελίδες με ήδη καταχωρημένες, ώστε να αποφασίσουν εάν θα τις καταχωρήσουν ή όχι
  - Ακολουθίες DNA, για τις οποίες ομοιότητα σημαίνει και δομική ή λειτουργική ομοιότητα

# Μέγιστη Κοινή Υπακολουθία

- Ορισμός υπακολουθίας συμβολοσειράς

$$x = \alpha \mathbf{\beta} \gamma \delta \mathbf{2} \tau \delta \mathbf{\lambda}$$

$$x' = \mathbf{\beta} \mathbf{2} \mathbf{\lambda}, \text{ με } i_1 = 2 < i_2 = 5 < i_3 = 8$$

- Παρατηρήστε πως σεβόμαστε την από αριστερά προς τα δεξιά διάταξη των χαρακτήρων
- Έτσι, η  $\beta\alpha\delta$  *δεν είναι υπακολουθία*

# Μέγιστη Κοινή Υπακολουθία

- Ορισμός

*Εάν  $x, y$  είναι δύο συμβολοσειρές, τότε η μέγιστη κοινή υπακολουθία τους ορίζεται ως η μεγαλύτερου μήκους υπακολουθία που εμφανίζεται και στις δύο.*

– Δεν είναι απαραίτητο να υπάρχει μόνο μία. Π.χ., οι  
αβελ και βαλε

έχουν τις εξής μέγιστες, μήκους 2, κοινές  
υπακολουθίες:

αλ, αε, βε, βλ

# Απλοϊκή Λύση

- Δημιουργία όλων των υπακολουθιών των δύο συμβολοσειρών και σύγκριση μεταξύ τους.
- Κόστος

$$\sum_{i=1}^{\max\{n,m\}} i2^i = O(\max\{n,m\}2^{\max\{n,m\}}), \quad n = |x|, m = |y|$$

# Λύση Δυναμικού Προγραμματισμού

- Παρατήρηση

Έστω  $x = x[1..n]$ ,  $y = y[1..m]$  οι συμβολοσειρές και  $z = z[1..k]$  μία μέγιστη κοινή υπακολουθία τους. Τότε, ισχύει:

$$x[n] \begin{cases} = y[m] \Rightarrow z[k] = x[n], z[1..k-1] = \text{μκυ}(x[1..n-1], y[1..m-1]) \\ \neq y[m] \text{ και } z[k] \neq x[n], z[1..k] = \text{μκυ}(x[1..n-1], y[1..m]) \\ \neq y[m] \text{ και } z[k] \neq y[m], z[1..k] = \text{μκυ}(x[1..n], y[1..m-1]) \end{cases}$$

# Λύση Δυναμικού Προγραμματισμού (συν.)

- Επομένως, εάν  $c[i,j]$  είναι το μήκος της μέγιστης κοινής υπακολουθίας των  $x[1..i]$ ,  $y[1..j]$ , τότε ισχύει:

$$c[i,j] = \begin{cases} 0 & i = 0 \text{ ή } j = 0 \\ c[i-1, j-1] + 1 & x[i] = y[j] \\ \max\{c[i, j-1], c[i-1, j]\} & \text{διαφορετικά} \end{cases}$$

# Ψευδοκώδικας

**Algorithm** largestCommonSubSeq(char[] x, char[] y)

```
1. n = x.length; m = y.length;
2. int[][] c = new int[n+1][m+1];
3. int[][] aux = new int[n+1][m+1];
4. for (i = 0; i <= n; i++){ // # O(n)
5.   c[i][0] = 0;
6.   aux[i][0] = NOT X;
7. }
8. for (i = 0; i <= m; i++){ // # O(m)
9.   c[0][i] = 0;
10.  aux[0][i] = NOT Y;
11. }
12. for (i = 1; i <= n; i++) // # O(nm)
13.  for (j = 1; j <= m; j++){
14.    if (x[i] == y[j]) { // ταίριασμα
15.      c[i][j] = c[i-1][j-1]+1;
16.      aux[i][j] = XY;
17.    }
18.    else if (c[i-1][j] >= c[i][j-1]) // το x[i] δεν ανήκει
18.      c[i][j] = c[i-1][j];
20.    aux[i][j] = NOT_X;
21.  }
22.  else {
19.    c[i][j] = c[i][j-1]; // το y[j] δεν ανήκει
20.    aux[i][j] = NOT_Y;
21.  }
22. }
23. return c[n][m];
```



# Παράδειγμα

↑ : απόρριψη από ΒΑΒΕΛ

← : απόρριψη από ΑΒΕΛ

↖ : ταίριασμα

		<b>A</b>	<b>B</b>	<b>E</b>	<b>Λ</b>
	0	1	2	3	4
0	0	←0	←0	←0	←0
<b>B</b> 1	↑0	↑0	↖1	←1	←1
<b>A</b> 2	↑0	↖1	←1	←1	←1
<b>B</b> 3	↑0	↑1	↖2	←2	←2
<b>E</b> 4	↑0	↑1	↑2	↖3	←3
<b>Λ</b> 5	↑0	↑1	↑2	↑3	↖4

# Ελάχιστη Απόσταση Διαμορφώσεως

- Ορισμοί
  - Πράξεις διαμορφώσεως:
    - $\text{delete}(c)$ : διαγραφή του χαρακτήρα  $c$
    - $\text{insert}(c)$ : εισαγωγή του χαρακτήρα  $c$
    - $\text{replace}(c,k)$ : αντικατάσταση του χαρακτήρα  $c$  από τον  $k$

# Ελάχιστη Απόσταση Διαμορφώσεως (συν.)

- Πρόβλημα

*Δοθεισών δύο συμβολοσειρών  $x, y$ , ποια είναι η ελάχιστη ακολουθία πράξεων διαμορφώσεως που μετασχηματίζει την  $x$  σε  $y$ ;*

- Εφαρμογές

- Αποθήκευση εκδοχών αρχείων αποθηκεύοντας μόνο το πλήθος των αλλαγών τους
- Σύγκριση αρχείων
- Σύγκριση ακολουθιών DNA

# Παράδειγμα

- 3 πράξεις

$\lambda\epsilon\acute{\iota}\pi\epsilon\iota \Rightarrow \lambda\acute{\iota}\pi\epsilon\iota \Rightarrow \lambda\acute{\iota}\pi\iota \Rightarrow \lambda\acute{\iota}\pi\eta$   
Διαγραφή                      Διαγραφή                      Αντικατάσταση

- 4 πράξεις

$\lambda\epsilon\acute{\iota}\pi\epsilon\iota \Rightarrow \lambda\acute{\iota}\acute{\iota}\pi\epsilon\iota \Rightarrow \lambda\acute{\iota}\pi\epsilon\iota \Rightarrow \lambda\acute{\iota}\pi\eta\iota \Rightarrow \lambda\acute{\iota}\pi\eta$   
Αντικατάσταση                      Διαγραφή                      Αντικατάσταση                      Διαγραφή

# Παρατήρηση

- Έστω  $c[i,j]$  είναι ο ελάχιστος αριθμός διαμορφώσεως των  $x[1..i]$ ,  $y[1..j]$ . Τότε ισχύει:

$$c[i,j]=\min \begin{cases} c[i-1,j] + 1 & \text{del}(x[i]) & // x[1..i-1] \Rightarrow y[1..j] \\ c[i,j-1] + 1 & \text{ins}(y[j]) & // x[1..i] \Rightarrow y[1..j-1] \\ c[i-1,j-1]+1 & \text{repl}(x[i], y[j]) & // x[1..i-1] \Rightarrow y[1..j-1] \\ c[i-1,j-1] & x[i]=y[j] & // x[1..i-1] \Rightarrow y[1..j-1] \end{cases}$$

# Παράδειγμα

	-	Λ	Ι	Π	Η
-	0	1	2	3	4
Λ	1	0=	←1E	←2E	←3E
E	2	↑1Δ	↖1A	↖2A	↖3A
Ι	3	↑2Δ	↖1=	↖2A	↖3A
Π	4	↑3Δ	↑ 2 Δ	↖1=	← 2E
Ω	5	↑4Δ	↑ 3 Δ	↑ 2Δ	↖2A

# Ψευδοκώδικας

**Algorithm** minEdtDis(char[] x, char[] y)

1.  $n = x.length; m = y.length;$
2. **for** ( $i = 0; i \leq n; i++$ )     // #  $O(n)$
3.      $c[i][0] = i;$
4. **for** ( $i = 0; i \leq m; i++$ )     // #  $O(m)$
5.      $c[0][i] = i;$
6. **for** ( $i = 1; i \leq n; i++$ )     // #  $O(nm)$
7.     **for** ( $j = 1; j \leq m; j++$ )
8.          $c[i][j] = \min(c[i-1][j]+1, c[i][j-1]+1, (x[i] == y[j] ? c[i-1][j-1] : c[i-1][j-1]+1));$
9. **return**  $c[n][m];$

- Πολυπλοκότητα  $O(nm)$ : ανάλογη, δηλαδή, του μεγέθους του πίνακα  $c$

# Αντεστραμμένα Αρχεία

- Αποτελούν την βασική δομή των search engines
- Βασίζονται στην ανάλυση του κειμένου σε λέξεις με:
  - μη διακριτική ικανότητα (άρθρα, σύνδεσμοι κ.ο.κ.), και
  - διακριτική ικανότητα
- Το σύνολο των κοινών, δίχως διακριτική ικανότητα, λέξεων αποτελεί την **τερματική λίστα** (*stop list*)
- Οι λέξεις με σημασιολογική σπουδαιότητα καλούνται **όροι** (*terms*) και αποτελούν το **λεξικό**



# Περιγραφή

- Με κάθε όρο του λεξικού σχετίζεται μία αντεστραμμένη λίστα με
  - τις εμφανίσεις του όρου –οι δείκτες συνήθως είναι το id του κειμένου–
  - συσχετιζόμενη βοηθητική πληροφορία, π.χ., συχνότητα εμφανίσεως
- Οι όροι είναι ταξινομημένοι κατά αλφαβητική σειρά
- Το μέγεθος του λεξικού είναι ΥΠΟΓΡΑΜΜΙΚΟ  $O(n^c)$  στο πλήθος  $n$  των λέξεων (νόμος Heaps)

# Παράδειγμα

1 Υπάρχουν δύο τεχνικές ανάλυσης πολυπλοκότητας: η ανάλυση χειρότερης περιπτώσεως και η ανάλυση μέσης περιπτώσεως.

2 Σε περίπτωση που η κύρια μνήμη είναι περιορισμένη, χρησιμοποιείται η ιδεατή μνήμη.

3 Κύριος λόγος χρήσεως του αλγορίθμου αυτού είναι οι μικρές απαιτήσεις σε μνήμη.

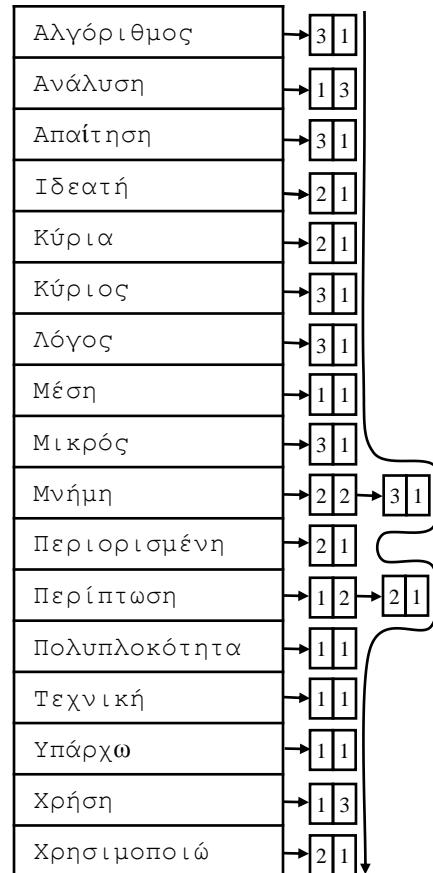
Αλγόριθμος	1	→	3
Ανάλυση	1	→	1
Απαίτηση	1	→	3
Ιδεατή	1	→	2
Κύρια	1	→	2
Κύριος	1	→	3
Λόγος	1	→	3
Μέση	1	→	1
Μικρή	1	→	3
Μνήμη	2	→	2 → 3
Περιορισμένη	1	→	2
Περίπτωση	2	→	1 → 2
Πολυπλοκότητα	1	→	1
Τεχνική	1	→	1
Υπάρχω	1	→	1
Χρήση	1	→	3
Χρησιμοποιώ	1	→	2

# Πώς χρησιμεύει

- Μία ερώτηση με ‘και’ (βρες μου όλα τα έγγραφα που περιέχουν ‘πληροφορική’ και ‘πολυτεχνείο’) αντιστοιχεί σε τομή των αντίστοιχων λιστών
- Μία ερώτηση με ‘ή’ (βρες μου όλα τα έγγραφα που περιέχουν ‘πληροφορική’ ή ‘πολυτεχνείο’) αντιστοιχεί σε ένωση των αντίστοιχων λιστών
- Μία ερώτηση με άρνηση (βρες μου όλα τα έγγραφα που δεν περιέχουν την λέξη ‘πληροφορική’) αντιστοιχεί σε αποκλεισμό της αντίστοιχης λίστας
- Μεικτές ερωτήσεις με ‘και’ , ‘ή’ και ‘όχι’ αναλύονται σε αντίστοιχες πράξεις ενώσεως, τομής και συμπλήρωσης (άρνησης) συνόλων με την μορφή λιστών

# Δημιουργία

- 1ος τρόπος:  
στην κύρια  
μνήμη

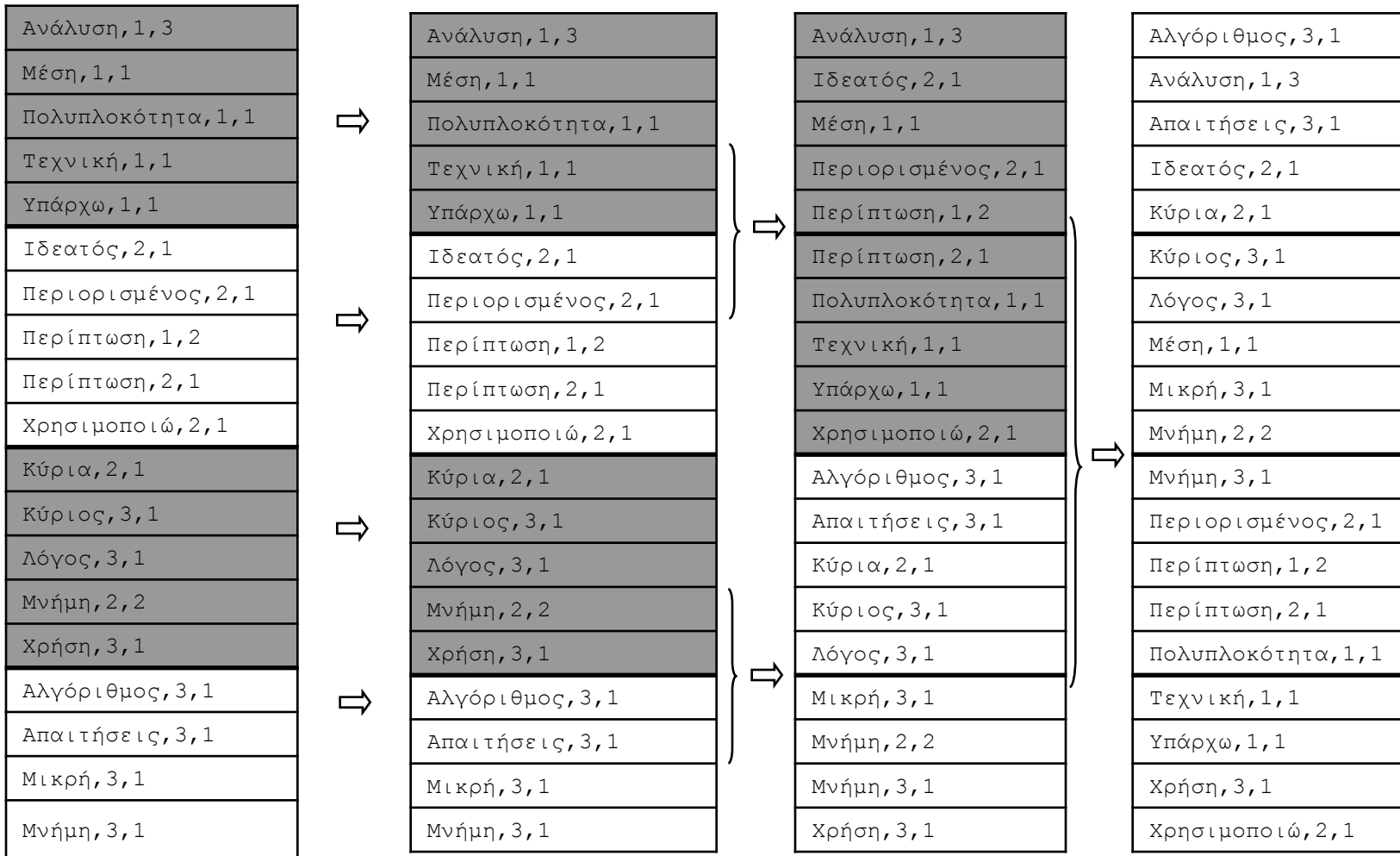


(α)



(β)

- 2ος τρόπος: Δημιουργία στον σκληρό δίσκο με merge sort!



# Κωδικοί Huffman

- Συμπίεση Κειμένου

Η διαδικασία αλλαγής της αναπαραστάσεως ώστε

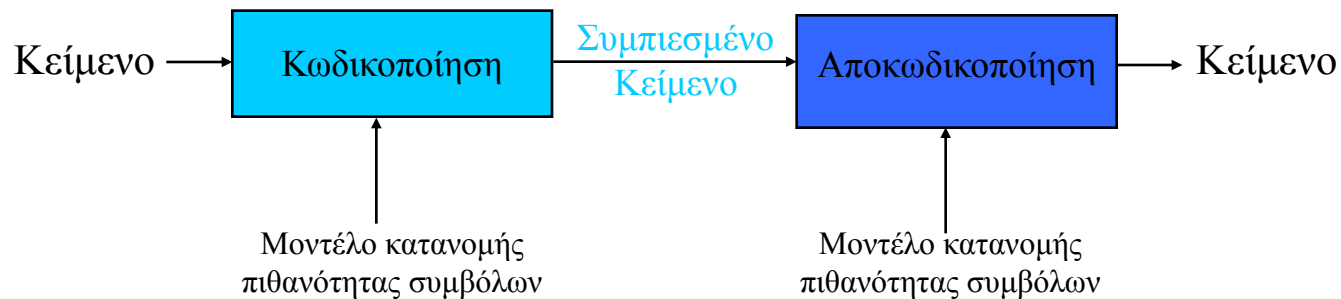
(α) να καταλαμβάνει μικρότερο χώρο ή να χρειάζεται λιγότερο χρόνο μεταδόσεως και

(β) να είναι δυνατή η **πλήρης** αποκατάσταση της αρχικής μορφής του κειμένου

- Κρίσιμη Παρατήρηση:

Εν αντιθέσει με την εικόνα ή τον ήχο, όπου τυχόν απώλειες (θόρυβος) είναι αποδεκτές, στα κείμενα **δεν είναι ανεκτές οι απώλειες**

# Γενική Διαδικασία Κωδικοποίησης - Αποκωδικοποίησης



- Η ανάθεση κωδικών στους χαρακτήρες γίνεται βάσει του πιθανοτικού μοντέλου που διέπει την εμφάνιση των χαρακτήρων του κειμένου

# Μέθοδοι Κωδικοποίησης

- **Huffman**

- Η κύρια μέθοδος μέχρι τα τέλη του '70. Την ανακάλυψε ο Huffman στα πλαίσια μίας εργασίας σε μάθημα κατά την διάρκεια των σπουδών του (!)
- Πετυχαίνει συμπίεση 5 μπιτ ανά χαρακτήρα.
- Την χρησιμοποιεί η εντολή pack στο unix.
- Μειονέκτημα ότι πρέπει πρώτα να σαρωθεί όλο το κείμενο.

- **Ziv-Lempel**

- Συμπιέζει 'on-the-fly' βρίσκοντας επαναλαμβανόμενα patterns.
- Πετυχαίνει 4 μπιτ ανά χαρακτήρα.
- Σε αυτήν στηρίζονται τα gzip και compress.

- **Αριθμητικοί Μέθοδοι**

- Συνήθως προτιμούνται για μετάδοση δεδομένων, καθώς πετυχαίνουν κάτι ελάχιστα παραπάνω από 2 μπιτ ανά χαρακτήρα.

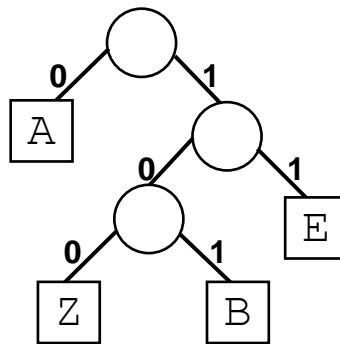


# Κωδικοί Προθέματος (Prefix Codes)

- **Ορισμός**

Δυαδικές συμβολοσειρές (bit strings) με την *κρίσιμη* ιδιότητα *καμμία να μην αποτελεί πρόθεμα της άλλης*.

- Αναπαρίστανται μέσω Tries, οπότε είναι εύκολη η αποκωδικοποίηση/κωδικοποίηση με διέλευση του δένδρου. Π.χ.,



# Κωδικοί Προθέματος (συν.)

- Κόστος δένδρου

$$E(T) = \sum_{c \in T} \mathbf{Prob}[c] \text{len}(c)$$

Άρα ζητούμενο η εύρεση δένδρου  $T$  που ελαχιστοποιεί το κόστος. Γιατί;

$$\begin{aligned} E(\text{κειμένου } K) &= \sum_{x \in K} \mathbf{Prob}[x] \text{len}(x) \\ &= |K| \sum_{c \in C} \mathbf{Prob}[c] \text{len}(c), \end{aligned}$$

όπου  $C$  το σύνολο των διακεκριμένων χαρακτήρων

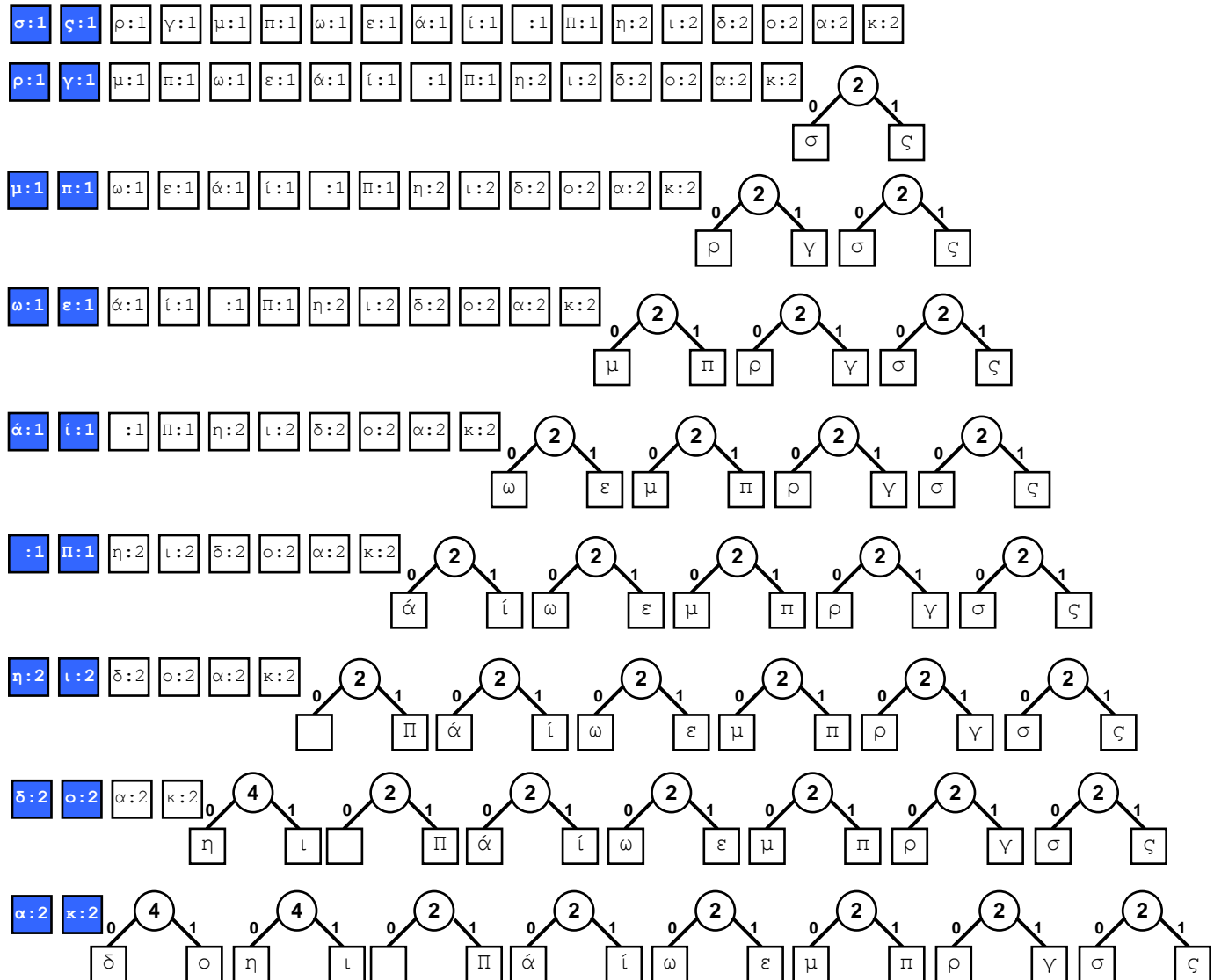
# Κεντρική Ιδέα

- Στους χαρακτήρες που έχουν μεγάλη συχνότητα εμφάνισης (άρα, έχουν μικρή διακριτική αξία) να δοθούν μικρού μήκους κωδικοί, ενώ στους σπάνιους (με μεγάλη σημασιολογική αξία) να δοθούν μεγαλύτερου μήκους κωδικοί.
- Ο Huffman ανακάλυψε έναν *απλό, βέλτιστο και άπληστο* αλγόριθμο κατασκευής κωδικών

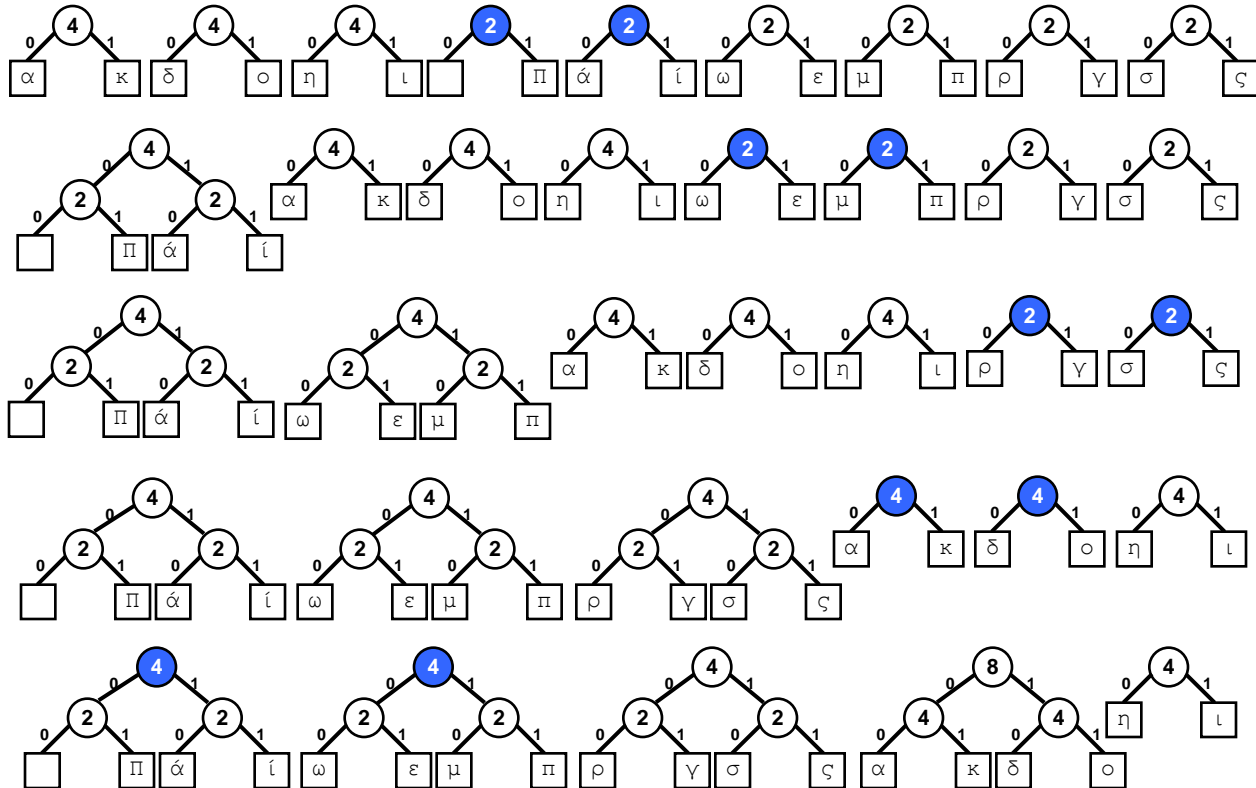
# Περιγραφή

- Το αρχικό πρόβλημα κωδικοποιήσεως  $n$  χαρακτήρων ανάγεται στην αναδρομική επίλυση ενός προβλήματος  $n-1$  χαρακτήρων. Πώς;
  - Με την επιλογή των δύο χαρακτήρων  $c_i, c_j$  μικρότερης συχνότητας  $f_i, f_j$ , αντίστοιχα, και την αντικατάστασή τους με έναν τεχνητό χαρακτήρα, συχνότητας  $f_i+f_j$

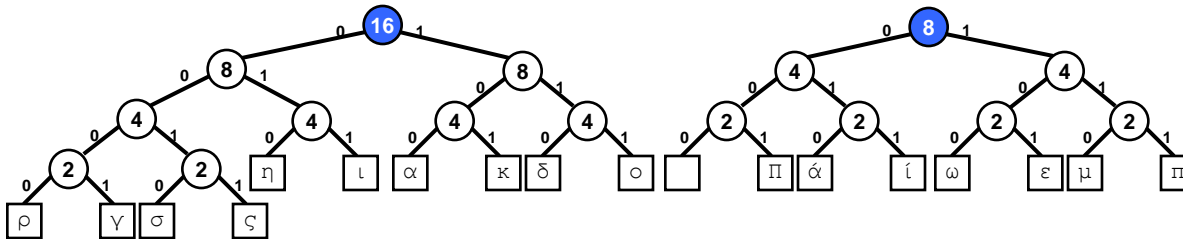
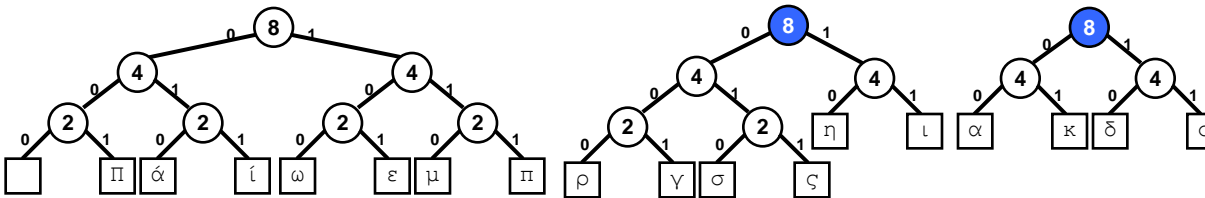
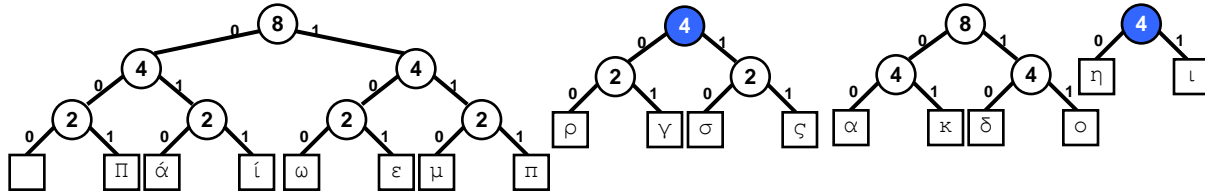
# Παράδειγμα



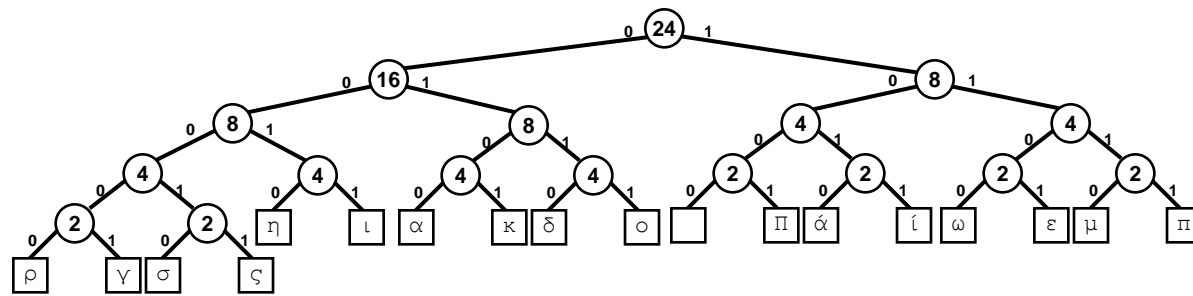
# Παράδειγμα (συν.)



# Παράδειγμα (συν.)



# Παράδειγμα (συν.)



Παράδειγμα κωδικοποίησης

ASCII

```

11010000111000011111100011101110011100100
1110010111101001111000111110110011100001
1010000011101010111110011110010111101001
1110101011101111111100001110111111011111
11100111111100111111001111110010
    
```

Huffman

```

1001010000000101001101101001100001111001
00100001011100011000110101011111111011110
11001000010001000011
    
```



# Υλοποίηση

**Algorithm** huffmanCoding(char[] C, int[] F)

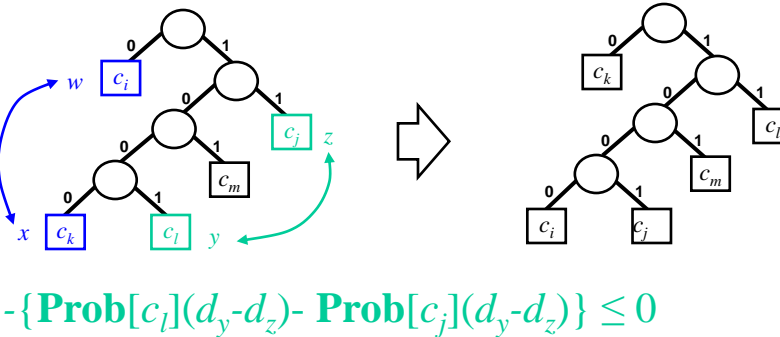
1. PriorityQueue pq = **new** PriorityQueue(); // ουρά προτεραιότητας
2. **for** (i = 0; i < C.length; i++) // αρχικοποίηση με τα φύλλα – χρόνος  $O(n \log n)$
3. pq.insKey(**new** bNode(**new** Item(C[i], F[i]), F[i])); // χρόνος  $O(\log n)$
4. **for** (i = 1; i < C.length; i++){ // χρόνος  $O(n \log n)$
5. bNode t = **new** bNode(); // νέος εσωτερικός κόμβος
6. x = pq.delMin(); // χρόνος  $O(\log n)$
7. y = pq.delMin(); // χρόνος  $O(\log n)$
8. t.setLeftSon(x);
9. t.setRightSon(y);
10. t.setItem(**new** Item(O, x.Key+y.Key)) // με στοιχείο τον τεχνητό χαρακτήρα O
11. pq.insKey(t, t.key); // και συχνότητα το άθροισμά τους – χρόνος  $O(\log n)$
12. }
13. **return** pq.delMin(); // ρίζα του δένδρου ο εναπομείναν κόμβος

- **Πολυπλοκότητα:**  $O(n \log n)$  με χρήση δυωνυμικής ουράς

# Απόδειξη ορθότητας

- Η άπληστη επιλογή οδηγεί σε βέλτιστη λύση· ήτοι, υπάρχει δένδρο κωδικών με τους σπανιότερους χαρακτήρες να είναι αδέρφια

$$\begin{aligned}
 & -\mathbf{Prob}[c_k]d_x + \mathbf{Prob}[c_k]d_w - \\
 & \mathbf{Prob}[c_i]d_w + d_x \mathbf{Prob}[c_i] = \\
 & -\{\mathbf{Prob}[c_k](d_x - d_w) - \\
 & \mathbf{Prob}[c_i](d_x - d_w)\} \leq 0
 \end{aligned}$$



- Η αλλαγή δίδει δένδρο με μικρότερο ή ίσο κόστος με το παλιό καθώς  $\mathbf{Prob}[c_i] \leq \mathbf{Prob}[c_k]$  και  $\mathbf{Prob}[c_j] \leq \mathbf{Prob}[c_l]$

# Απόδειξη ορθότητας (συν.)

- Το δένδρο που προκύπτει συνδυάζοντας την λύση για τους  $n-1$  χαρακτήρες είναι βέλτιστο.

Με επαγωγή στο πλήθος  $n$  των χαρακτήρων:

- Με την αντικατάσταση, προκύπτει ένα στιγμιότυπο  $n-1$  χαρακτήρων, όπου ο τεχνητός χαρακτήρας  $x$  έχει βάθος  $d_x$ , βέλτιστου -βάσει επαγωγής- κόστους  $E(T_{n-1})$
- Με την τοποθέτηση των δύο χαρακτήρων σε βάθος  $d_x+1$ , προκύπτει δένδρο κόστους:

$$E(T_n) = E(T_{n-1}) + \mathbf{Prob}[c_i] + \mathbf{Prob}[c_j]$$

- Αυτό είναι βέλτιστο, γιατί:
  - Εάν δεν ήταν, τότε υπάρχει κάποιο άλλο με καλύτερο κόστος, όπου τα  $c_i, c_j$  έχουν μέγιστο βάθος.
  - Αντικαθιστώντας τα με ένα τεχνητό χαρακτήρα, προκύπτει στιγμιότυπο για  $n-1$  χαρακτήρες με καλύτερη λύση (άτοπο)

# Πρόβλημα Κρυπτογραφίας

- Ανταλλαγή μηνυμάτων μεταξύ δύο μερών ώστε το περιεχόμενό τους, σε περίπτωση υποκλοπής από τρίτο, να παραμένει ακατάληπτο

*Για παράδειγμα, οι ηλεκτρονικές αγορές στηρίζονται στην αποστολή αριθμού πιστωτικής κάρτας από τον Η/Υ μας στον Η/Υ της εταιρίας μέσω πολλαπλών ενδιάμεσων Η/Υ*

# Ορολογία

- Τα επικοινωνούντα μέρη επικράτησε να αποκαλούνται *Αλίκη* και *Μποπ* (*Alice, Bob*), ενώ ο υποκλοπέας *Εύα* (*Eve*)
- **Κρυπτογράφηση/Αποκρυπτογράφηση:**
  - Το κείμενο  $M$  (plaintext) κρυπτογραφείται (μετατρέπεται) στο κρυπτογράφημα (ciphertext)  $C$  από την Αλίκη και αποστέλλεται στον Μποπ
  - Αυτός πρέπει να είναι σε θέση να το αποκρυπτογραφήσει, δηλαδή να μετατρέψει το  $C$  σε  $M$
  - Ακόμη και αν η Εύα υποκλέψει το  $C$  δεν πρέπει να είναι σε θέση να το αποκαταστήσει στην αρχική του μορφή  $M$
- **Ψηφιακές Υπογραφές:**
  - Όταν γίνεται η αποστολή μηνυμάτων, η Αλίκη τα «υπογράφει» ώστε ο Μποπ να είναι σίγουρος ότι αποστολέας τους είναι η Αλίκη και όχι η Εύα υποκρινόμενη την Αλίκη

# Προσεγγίσεις

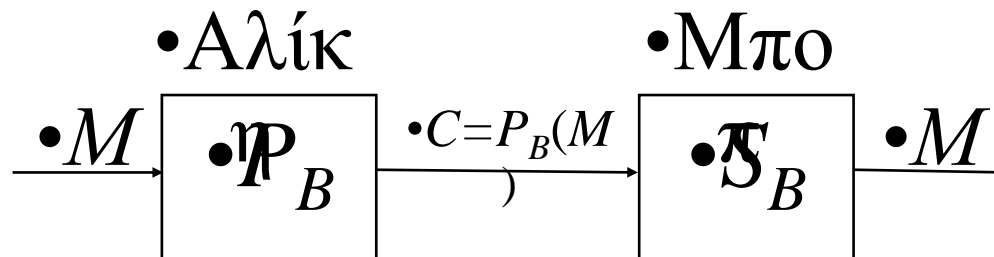
- **Μυστικό Κλειδί**

- Τα δύο μέρη ανταλλάσσουν ένα μυστικό κλειδί, το οποίο χρησιμοποιείται *τόσο για την κρυπτογράφηση όσο και την αποκρυπτογράφηση*
- Πρέπει να το ανταλλάξουν με ασφάλεια πριν αρχίσει η επικοινωνία
- **Παράδειγμα: Κρυπτογράφηση Ιουλίου Καίσαρα**
  - το  $i$ -στό γράμμα αντικαθίσταται με το  $(i+3)$ -στό mod 26
  - Το 3 αποτελεί το μυστικό κλειδί

# Προσεγγίσεις (συν.)

- **Δημόσιο Κλειδί**

- Η επικοινωνία γίνεται μέσω δύο κλειδιών, ενός δημόσιου  $P$  και ενός μυστικού  $S$
- Πχ, για την επικοινωνία του ο Μποπ διαλέγει δύο κλειδιά  $P_B, S_B$ :
  - Το πρώτο, το ανακοινώνει, και το δεύτερο, το κρατά για τον εαυτό του
  - Εάν η Αλίκη θέλει να επικοινωνήσει μαζί του, χρησιμοποιεί το  $P_B$  για την παραγωγή τού  $C$
  - Ο Μποπ αποκρυπτογραφεί το  $C$  μέσω του  $S_B$



# Προσεγγίσεις (συν.)

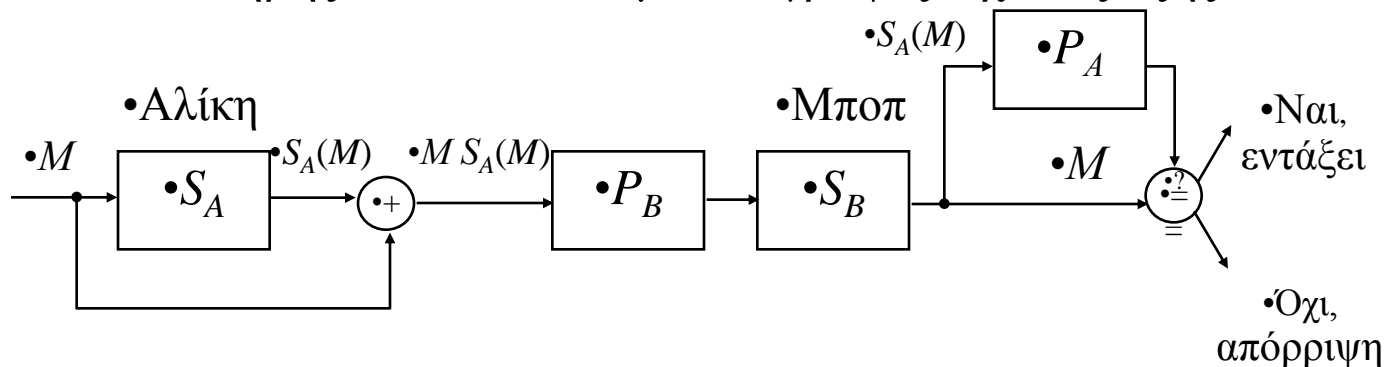
## – Προδιαγραφές $P, S$ :

1.  $P(S(M)) = S(P(M))$
2. Να υπολογίζονται εύκολα
3. Η εξαγωγή του  $S$  από το  $P$  να είναι πρακτικώς αδύνατη

– Η ιδιότητα 1 εξασφαλίζει τον σχηματισμό ψηφιακής υπογραφής  $S(M)$ , καθώς:

- $M = P(S(M))$ ,
- συνεπάγεται πως ο Μποπ είναι ο συντάκτης του κειμένου  $M$

– Η πλήρης επικοινωνία, με υπογραφές, έχει ως εξής:





# Βασικές Μαθηματικές Έννοιες

- Πρώτος Ακέραιος:
  - Κάθε ακέραιος με διαιρέτες τον εαυτό του και την μονάδα
  - Υπάρχουν άπειροι πρώτοι αριθμοί
  - Κάθε ακέραιος γράφεται με μοναδικό τρόπο ως γινόμενο πρώτων
  - Η πυκνότητα των πρώτων στο διάστημα  $[1..n]$  είναι ανάλογη του  $n / \log n$  (**Θεώρημα πρώτων αριθμών**)

# Βασικές Μαθηματικές Έννοιες (συν.)

- Μέγιστος κοινός διαιρέτης (gcd)  
 $\gcd(x,y) = \min \{xa+yb \mid x,y \in \mathbb{Z}\} = \gcd(y, x \bmod y)$
- Σχετικώς πρώτοι αριθμοί  $x,y$   
 $\gcd(x,y) = 1$
- $\mathbb{Z}_p = \{0, 1, 2, \dots, p-1\}$
- $\mathbb{Z}_p^* = \{a \in \mathbb{Z}_p \mid \gcd(a,p) = 1\}$  (με την πράξη επί modulo, συνιστά αβελιανή πολλαπλασιαστική ομάδα-κλειστότητα, προσεταιριστικότητα, αντιμεταθετικότητα, ουδέτερο/αντίστροφο στοιχείο-ενώ  $a^{\mathbb{Z}_p^*} \equiv 1 \pmod p, a \in \mathbb{Z}_p^*$ )
- Για  $p$  πρώτο ισχύει  $\mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$
- Μικρό Θεώρημα Φερμά  
Εάν  $p$  πρώτος, τότε για κάθε ακέραιο  $a \in \mathbb{Z}_p^*$ , ισχύει:  
 $a^p \equiv a \pmod p$  ή  $a^{p-1} \equiv 1 \pmod p$   
**Το αντίστροφο δεν ισχύει, πχ,  $234 = 11 \times 31$**

# Βασικές Μαθηματικές Έννοιες (συν.)

- Αντίστροφος  $a^{-1} \pmod n$ 
  - $aa^{-1} \equiv 1 \pmod n$
  - Υπάρχει, εάν  $\gcd(a,n) = 1$
- Θεώρημα κινέζικου υπολοίπου:

Το σύστημα

$$x \equiv a_1 \pmod{n_1}$$

$$x \equiv a_2 \pmod{n_2}$$

$\vdots$

$$x \equiv a_k \pmod{n_k}$$

όπου οι  $n_1, n_2, \dots, n_k$  ανά δύο σχετικώς πρώτοι, έχει μοναδική λύση  $\pmod{n_1 n_2 \cdots n_k}$ . Δηλαδή,  $x \equiv y \equiv a_i \pmod{n_i}$  αν και μόνον αν  $x \equiv y \pmod{n_1 n_2 \cdots n_k}$

# Βασικές Μαθηματικές Έννοιες (συν.)

- Συνέπεια θεωρήματος κινέζικου υπολοίπου
  - Έστω οι  $n_1, n_2, \dots, n_k$  ανά δύο σχετικώς πρώτοι. Για κάθε ζεύγος ακεραίων  $x, a$  ισχύει

$$x \equiv a \pmod{n_1}$$

$$x \equiv a \pmod{n_2}$$

⋮

$$x \equiv a \pmod{n_k}$$

αν και μόνον αν

$$x \equiv a \pmod{n_1 n_2 \dots n_k}$$

# Κρυπτοσύστημα Δημοσίου Κλειδιού RSA (Riverest, Shamir, Adleman)

- Ο αλγόριθμος εξαγωγής δημόσιου και κρυφού κλειδιού είναι πολύ απλός:
  1. Επιλογή δύο *τυχαίων, μεγάλων, πρώτων ακεραίων*  $p, q$
  2. Υπολογισμός του γινομένου  $n = pq$
  3. Επιλογή ενός μικρού, περιττού ακεραίου  $e$ :  $\gcd(e, \varphi(n))=1$  ( $e, \varphi(n)$  σχετικώς πρώτοι), με  $\varphi(n)=(p-1)(q-1)$
  4. Υπολογισμός του  $d=e^{-1} \bmod \varphi(n)$  ( $de=1 \bmod \varphi(n)$ )
  5. Ανακήρυξη του  $P = (e, n)$  ως *δημόσιο κλειδί RSA*
  6. Ανακήρυξη του  $S = (d, n)$  ως *μυστικό κλειδί RSA*

Οπότε, για  $M < n$ , έχουμε

$$P(M)=M^e \bmod n \text{ και } S(C)=C^d \bmod n$$

# Απόδειξη Ιδιότητας Αντιστροφής

- Πρέπει να δείξουμε ότι:

$$P(S(M)) = S(P(M)) = M$$

- Έχουμε ότι:

$$P(S(M)) = S(P(M)) = M^{ed} \pmod n$$

$$ed = ee^{-1} \pmod{\varphi(n)} = 1 + k\varphi(n) = 1 + k(p-1)(q-1)$$

$$M^{1+k(p-1)(q-1)} \pmod p = M(M^{(p-1)})^{k(q-1)} \pmod p$$

- Όμως  $p$  πρώτος, άρα

$$M^{(p-1)} \pmod p = 1 \text{ (μικρό θεώρημα Φερμά)}$$

και συνεπώς

$$M(M^{(p-1)})^{k(q-1)} \pmod p = M \cdot 1 \pmod p = M \pmod p (*)$$

# Απόδειξη Ιδιότητας Αντιστροφής (συν.)

- Παρομοίως,  $q$  πρώτος, άρα:

$$M^{(q-1)} \bmod q = 1 \text{ (μικρό θεώρημα Φερμά)}$$

και συνεπώς:

$$M(M^{(q-1)})^{k(p-1)} \bmod q = M \cdot 1 \bmod q = M \bmod q \quad (\dagger)$$

$$(*) \text{ και } (\dagger) \Rightarrow M(M^{(q-1)})^{k(p-1)} \bmod pq = M \bmod pq$$

(συνέπεια θεωρήματος κινέζικου υπολοίπου)

# Παράδειγμα

- Έστω ότι θέλουμε να κρυπτογραφήσουμε το μήνυμα:  
Σύστημα RSA
- Σε ASCII η μορφή του είναι  
211**253**243**244**231**236**255**032**082**083**041
- Εάν επιλέξουμε  
 $p=19, q=31, e=53,$   
τότε  
 $n=19 \cdot 31=586, \varphi(n)=18 \cdot 30=540$  και  $d = 377.$
- Το κείμενο πρέπει να χωριστεί σε τμήματα των *δύο* ψηφίων, τα οποία κρυπτογραφούνται και μετά συνενώνονται:  
211253243244231236255032082083041  
↓  
**048084375517497205396084180377350497202267296157001**
- Η αποκωδικοποίηση γίνεται τμήμα-τμήμα. Π.χ.,  
 $048^{377} \bmod 586 = 21$



# Πράξεις που απαιτούνται

- Υπολογισμός γινομένου δύο ακεραίων-εύκολη
- Ύψωση σε δύναμη modulo  $n$  -εύκολη αναδρομική διαδικασία, αφού:
  - $a^b = a^{b/2} a^{b/2} = (a^2)^{b/2}$ ,  $b$  άρτιος, ή
  - $a^b = a a^{(b-1)/2} a^{(b-1)/2} = a(a^2)^{(b-1)/2}$ ,  $b$  περιττός

## Υπολογισμός αντίστροφου $e$ modulo $n$ -εύκολος

- Ισχύει  $\gcd(a,b)=xa+yb$ .
- Εμείς θέλουμε να βρούμε τον αριθμό  $x$ , τέτοιον ώστε  $ex \equiv 1 \pmod n$  ή  $ex = 1+cn, c \in \mathbb{N}$ .
- Όμως  $e$  αντιστρέψιμος σημαίνει  $\gcd(e,n) = 1$  ή  $xe+yn = 1$ .
- Άρα, αρκεί να βρούμε τον συντελεστή  $x$  του μέγιστου κοινού διαιρέτη. Αυτό γίνεται μέσω του αλγορίθμου του Ευκλείδη

**Algorithm** gcdExtended (int a, int b, coeff c) // απόδειξη με επαγωγή

```
1. if (b == 0){
2.   c.x = 1;
3.   c.y = 0;
4. }
5. else{
6.   gcdExtended(b, a % b, c); // Επαγ. Υπόθεση:  $\gcd(b,a\%b)=x1*b+y1*(a\%b)$ 
7.   x1 = c.x; y1 = c.y; //  $\gcd(a,b) = y1*a+(x1-(a/b)*y1)*b = y1*(a-(a/b)b)+x1*b$ 
8.   c.x = y1;           //           =  $y1*(a\%b)+ x1*b = \gcd(b,a\%b)$  ΟΕΔ
9.   c.y = x1-(a/b)*y1;
10. }
```

## Εύρεση δύο πρώτων ακεραίων $p, q$ (λίγο δύσκολο)

- Υπάρχουν πιθανοτικά τεστ που αποφαινόνται εάν ένας ακέραιος είναι πρώτος ή όχι.
- Το πιο γνωστό είναι το τεστ Miller-Rabin, το οποίο βασίζεται στο μικρό θεώρημα του Φερμά:

$$p \text{ πρώτος και } a \in \mathbb{Z}_p^* \Rightarrow a^{p-1} = 1 \pmod{p}$$
$$a^{p-1} \neq 1 \pmod{p} \Rightarrow p \text{ σύνθετος}$$

**Algorithm** millerRabinTest(int n, int s)

1. **for** (i = 0; i < s; i++){
2.   a = random(1, n-1);
3.   **if** ((power(a, n-1, n) != 1) //  $a^{n-1} \neq 1 \pmod{n}$
4.     **return** composite;       // σίγουρα σύνθετος
5. }
6. **return** prime;           // με επιφύλαξη

Για  $s = 51$  η πιθανότητα λάθους είναι μόλις  $4.44 \cdot 10^{-16}$

# Μειονεκτήματα RSA

- Αργοί οι υπολογισμοί εν σχέσει με τους αλγορίθμους μυστικών κλειδιών
- Δεν υπάρχει απόδειξη ότι δεν «σπάει»: Στηρίζεται στην *εικασία* ότι *μόνον εάν βρεθεί αποτελεσματική διαδικασία παραγοντοποιήσεως ακεραίων, ο αλγόριθμος σπάει*