ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΡΟΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ

ΑΛΓΟΡΙΘΜΟΙ

ΔΙΑΛΕΞΗ 8

ΔΙΔΑΣΚΩΝ
ΚΩΣΤΑΣ ΚΟΛΟΜΒΑΤΣΟΣ

Δυναμικός Προγραμματισμός

Εισαγωγή (1/3)

Ο **δυναμικός προγραμματισμός (dynamic programming)** επιλύει τα προβλήματα μέσω του συνδυασμού των λύσεων ενός αριθμοού υποπροβλημάτων

Διαίρει και βασίλευε

- Διασπά το πρόβληματα σε διακριτά υποπροβλήματα, επιλύει το καθένα από αυτά αναδρομικά και στη συνέχεια συνδυάζει τις λύσεις

Δυναμικός προγραμματισμός

- Εφαρμόζεται όταν τα υποπροβλήματα επικαλύπτονται
- Επιλύει κάθε υπο-υποπρόβλημα μόνο μια φορά και αποθηκεύει την λύση / απάντηση σε ένα πίνακα

Εισαγωγή (2/3)

- Αποφεύγει την προσπάθεια εύρεσης των λύσεων κάθε φορά που επιλύει ένα υπο-υποπρόβλημα

Εφαρμόζουμε το δυναμικό προγραμματισμό σε προβλήματα βελτιστοποίησης

Τα προβλήματα βελτιστοποίησης μπορεί να έχουν πολλαπλές λύσεις

Κάθε λύση έχει μια τιμή / αξία και προσπαθούμε να βρούμε τη λύση με τη μεγαλύτερη αξία

Η λύση αυτή καλείται η **βέλτιστη (optimal)** λύση

Εισαγωγή (3/3)

Κατά την ανάπτυξη αλγορίθμων δυναμικού προγραμματισμού εφαρμόζουμε τα ακόλουθα βήματα:

- Χαρακτηρίζουμε τη δομή μιας βέλτιστης λύσης
- Ορίζουμε αναδρομικά την τιμή / αξία μιας βέλτιστης λύσης
- Υπολογίζουμε την τιμή των βέλτιστων λύσεων τυπικά μέσω της προσέγγισης bottom-up
- Δημιουργούμε μια βέλτιστη λύση από τις παραπάνω πληροφορίες

Rod Cutting (1/14)

Παράδειγμα: διαίρεση μιας ράβδου (rod cutting)

Ας υποθέσουμε ότι μια επιχείρηση αγοράζει ατσάλινες ράβδους και στη συνέχεια της διαιρεί και τις πουλάει ανεξάρτητα

Γνωρίζουμε την τιμή p_i, i=1, 2, 3, … στην οποία η επιχείρηση πουλάει το κάθε κομμάτι

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod Cutting (2/14)

Πρόβλημα

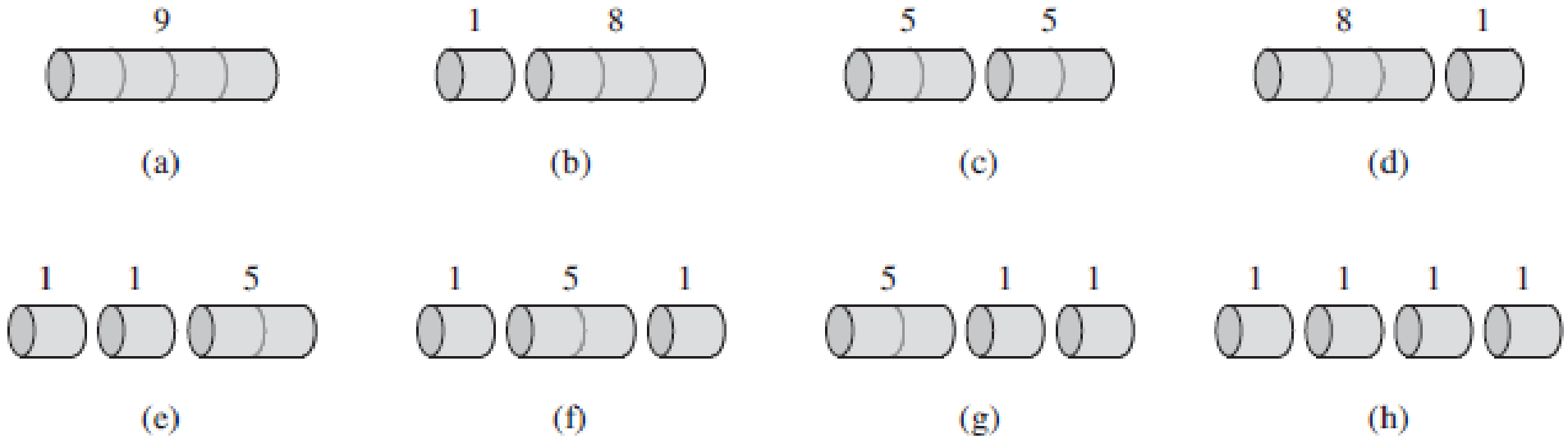
Δοσμένης μιας ράβδου μήκους n και ενός πίνακα τιμών για διάφορα μεγέθη κάποιων τμημάτων της ράβδου, να καθορίσουμε το μέγιστο κέρδος r_n το οποίο θα αποκομίσουμε αν διαιρέσουμε τη ράβδο σε κομμάτια και τα πουλήσουμε

Σημείωση: αν η τιμή p_n είναι αρκετά μεγάλη, τότε ίσως να μην χρειάζεται να διαιρέσουμε τη ράβδο

Rod Cutting (3/14)

Ας υποθέσουμε ότι n=4. Τότε έχουμε το ακόλουθο σχήμα:

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30



Rod Cutting (4/14)

Μπορούμε να διαιρέσουμε τη ράβδο με 2^{n-1} διαφορετικούς τρόπους

Η επιλογή μας θα είναι κάθε φορά να προχωρήσουμε στη διαίρεση ή όχι σε απόσταση i από αριστερά για i=1,2,…,n-1

Παράδειγμα:

- Η διαίρεση 6=1+2+3 σημαίνει ότι διαιρούμε μια ράβδο μήκους 6 σε 1, 2 και 3 μέτρα αντίστοιχα

Rod Cutting (5/14)

Αν μια βέλτιστη λύση είναι να διαιρέσουμε τη ράβδο σε k τμήματα για κάποιο 1<=k<=n, τότε μια βέλτιστη υποδιαίρεση

$$n = i_1 + i_2 + \cdots + i_k$$

Σε τμήματα μήκους

$$i_1, i_2, \ldots, i_k$$

Μας δίνει κέρδος

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

Rod Cutting (6/14)

Γενικά, θα μπορούσαμε να προσδιορίσουμε το κέρδος σε σχέση με το βέλτιστο κέρδος κάθε προηγούμενου διαχωρισμού

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$$

Το p_n αντιστοιχεί στο κέρδος της μη διαίρεσης της ράβδου

Οι υπόλοιπες υποδιαιρέσεις αφορούν στο κέρδος που προκύπτει από την υποδιαίρεση της ράβδου σε 2 κομμάτια

Για να λύσουμε το πρόβλημα που έχει μέγεθος n, πρέπει να λύσουμε ιδίου τύπου υποπροβλήματα αλλά με μικρότερο μέγεθος

Αν κάνουμε την πρώτη υποδιαίρεση, τότε θεωρούμε τα δύο κομμάτια σαν ανεξάρτητες ‘εκδόσεις’ του αρχικού προβλήματος

Rod Cutting (7/14)

Το πρόβλημα επιδεικνύει **optimal substructure**

- Οι βέλτιστες λύσεις σε ένα πρόβλημα ενσωματώνουν τις βέλτιστες λύσεις στα σχετικά υποπροβλήματα τα οποία μπορούν να λυθούν ανεξάρτητα

Προσέγγιση:

- Θεωρούμε μια διαίρεση της ράβδου που αποτελείται από το αριστερό κομμάτι που δεν μπορεί να διαιρεθεί περαιτέρω και το δεξιό το οποίο θα διαιρεθεί στη συνέχεια
- Παίρνουμε την ακόλουθη εξίσωση για το κέρδος $r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$
- Η εξίσωση περιλαμβάνει την επίλυση ενός υποπροβλήματος και όχι δύο

Rod Cutting (8/14)

Top-down προσέγγιση

Είσοδοι: πίνακας p[1..n] με τιμές και η τιμή n

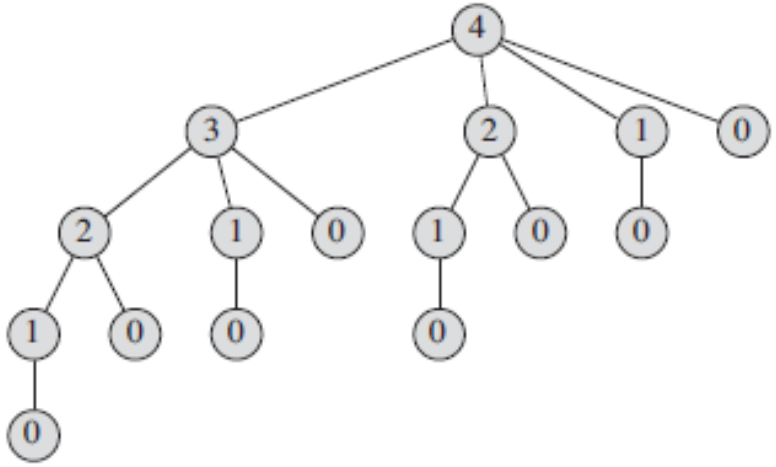
Μη αποδοτική λύση – λύνει τα ίδια υποπροβλήματα συνεχώς μέσω της αναδρομικής κλήσης

Παράδειγμα για n=4

Πολυπλοκότητα $T(n) = 1 + \sum_{j=0}^{n-1} T(j)$

και άρα $T(n) = 2^n$

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```



Rod Cutting (9/14)

Χρήση δυναμικού προγραμματισμού

- Καθορίζουμε **την επίλυση κάθε υποπροβλήματος μόνο μια φορά** και όχι πολλαπλές όπως με την αναδρομική κλήση
- Αποθηκεύουμε τη λύση έτσι ώστε αν χρειαστεί **να την προσπελάσουμε μελλοντικά**, τότε να τη δούμε απλά και να μην την ξανα-υπολογίσουμε
- Προφανώς, η λύση μέσω δυναμικού προγραμματισμού **θα απαιτήσει επιπλέον μνήμη αλλά γλιτώνει υπολογιστικό χρόνο**
- Μπορεί μια λύση εκθετικού χρόνου να μετατραπεί σε λύση πολυωνυμικού χρόνου

Rod Cutting (10/14)

Χρήση δυναμικού προγραμματισμού

- Top-down
  - Τροποποιούμε την αναδρομική λύση ώστε να αποθηκεύεται κάθε φορά η λύση των υποπροβλημάτων
- Bottom-up
  - Ταξινομούμε τα υποπροβλήματα κατά μέγεθος και λύνουμε πρώτα αυτά που έχουν μικρότερο μέγεθος

Rod Cutting (11/14)

Top-down with memorization

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = -∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8  r[n] = q
9  return q
```

Rod Cutting (12/14)

Bottom-up approach

- Πολυπλοκότητα $\Theta(n^2)$
  - **Γιατί?**
  - **Πόσο έχει η top-down?**

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

Rod Cutting (13/14)

Οι προηγούμενες λύσεις επιστρέφουν το μέγιστο κέρδος άλλα όχι τη βέλτιστη διαίρεση της ράβδου

Οι διπλανοί αλγόριθμοι υπολογίζουν και τυπώνουν το βέλτιστο διαχωρισμό

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

Rod Cutting (14/14)

Παράδειγμα εκτέλεσης

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

Τι θα τυπωθεί με τις ακόλουθες κλήσεις?

PRINT-CUT-ROD-SOLUTION$(p, 10)$

PRINT-CUT-ROD-SOLUTION$(p, 7)$

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

Longest Common Subsequence (1/12)

Χρησιμοποιείται κυρίως σε βιολογικές εφαρμογές

Παράδειγμα

- Σύγκριση αλυσίδων DNA

$S_1 = \texttt{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$

$S_2 = \texttt{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$

Προσπαθούμε να βρούμε πόσο μοιάζουν δύο συμβολοσειρές

Υιοθετούμε συγκεκριμένες μετρικές

Longest Common Subsequence (2/12)

Φορμαλισμός

- Δοσμένης μιας συμβολοσειράς $X=(x_1, x_2, \ldots, x_m)$ μια άλλη συμβολοσειρά $Z=(z_1, z_2, \ldots, z_k)$ είναι μια **υποακολουθία** (subsequence) της X, αν υπάρχει μια αυστηρά 'αυξανόμενη' ακολουθία $<i_1, i_2, \ldots, i_k>$ στοιχείων/δεικτών της X ώστε για όλα τα $j=1,2,\ldots,k$ ισχύει ότι $x_{ij}=z_j$
- Η Z=(B,C,D,B) είναι μια υποακολουθία της X=(A,B,C,B,D,A,B) για τους δείκτες (2,3,5,7)
- Δοσμένων δύο συμβολοσειρών X, Y, λέμε ότι η Z είναι μια **κοινή υποακολουθία** (common subsequence) των X και Y όταν η Z είναι υποακολουθία της X και της Y

Longest Common Subsequence (3/12)

Παράδειγμα

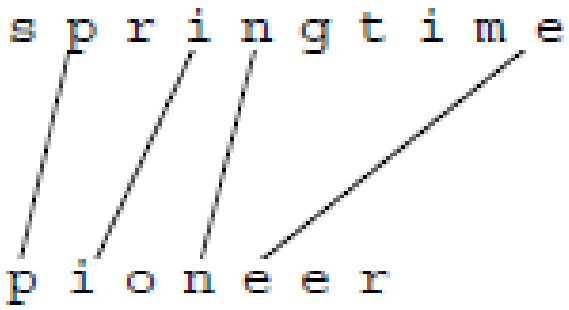
- X=(A,B,C,B,D,A,B), Y=(B,D,C,A,B,A)
  - Η (B,C,A) είναι μια κοινή υποακολουθία αλλά δεν είναι **η μέγιστη κοινή υποακολουθία (longest common subsequence -LCS)**
  - Η (B,C,A) έχει μήκος 3 ενώ η (B,C,B,A) και (B,D,A,B) έχουν μήκος 4 (και οι δύο είναι LCS)

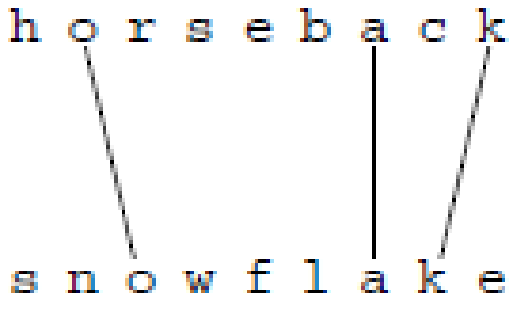
Πρόβλημα

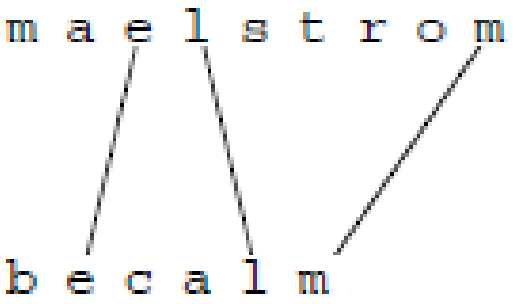
- Μας δίνονται δύο ακολουθίες $X=(x_1, x_2, \ldots, x_m)$ και $Y=(y_1, y_2, \ldots, y_n)$ και μας ζητείται να βρούμε την LCS

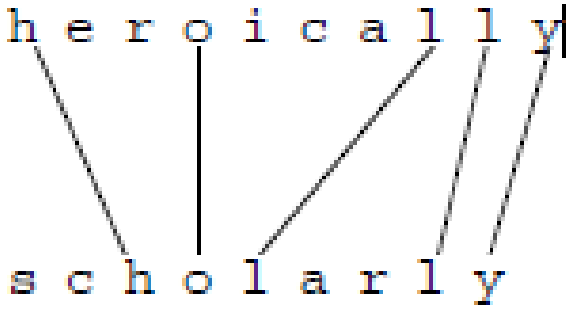
Longest Common Subsequence (4/12)

Παραδείγματα









Longest Common Subsequence (5/12)

Μια brute force λύση θα απαριθμούσε όλες τις κοινές υποακολουθίες και στη συνέχεια θα έβρισκε τη μέγιστη

Λόγω του ότι η X έχει 2^m υποακολουθίες, ο χρόνος που απαιτείται είναι εκθετικός

Μέσω δυναμικού προγραμματισμού προσπαθούμε να βρούμε τα prefixes των συμβολοσειρών

Longest Common Subsequence (6/12)

Ισχύει το ακόλουθο θεώρημα

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}

Longest Common Subsequence (7/12)

Το θεώρημα ουσιαστικά μας λέει:

- Η LCS δύο ακολουθιών περιέχει μια LCS κάποιων prefixes των δύο ακολουθιών
- Με αυτό το σκεπτικό η εύρεση της LCS έχει της ιδιότητα της βέλτιστης υποδομής (optimal substructure property)

Βασιζόμενοι στο προηγούμενο θεώρημα, μπορούμε να ορίσουμε μια επαναληπτική λύση

Longest Common Subsequence (8/12)

Το θεώρημα υποδεικνύει ότι:

- Αν $x_m=y_n$, πρέπει να βρούμε την LCS των X_{m-1}, Y_{n-1}. Στη συνέχεια προσθέτοντας τα x_m, y_n, έχουμε την LCS των αρχικών ακολουθιών
- Αν $x_m<>y_n$, πρέπει να λύσουμε δύο υπο-προβλήματα:
  - Εύρεση της LCS των X_{m-1}, Y
  - Εύρεση της LCS των X, Y_{n-1}
  - Όποια από αυτές τις δύο LCSs είναι μεγαλύτερη θα είναι και η τελική LCS
  - Κάθε ένα από αυτά τα υπο-προβλήματα εμπλέκει την εύρεση της LCS για τα X_{m-1}, Y_{n-1}

Longest Common Subsequence (9/12)

Έστω c[i,j] το μήκος μιας LCS των Xi, Yj

Αν i=0 ή j=0, τότε μια από τις δύο ακολουθίες έχει μήκος 0 οπότε η LCS έχει μήκος 0

Η αναδρομική σχέση έχει ως εξής

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Longest Common Subsequence (10/12)

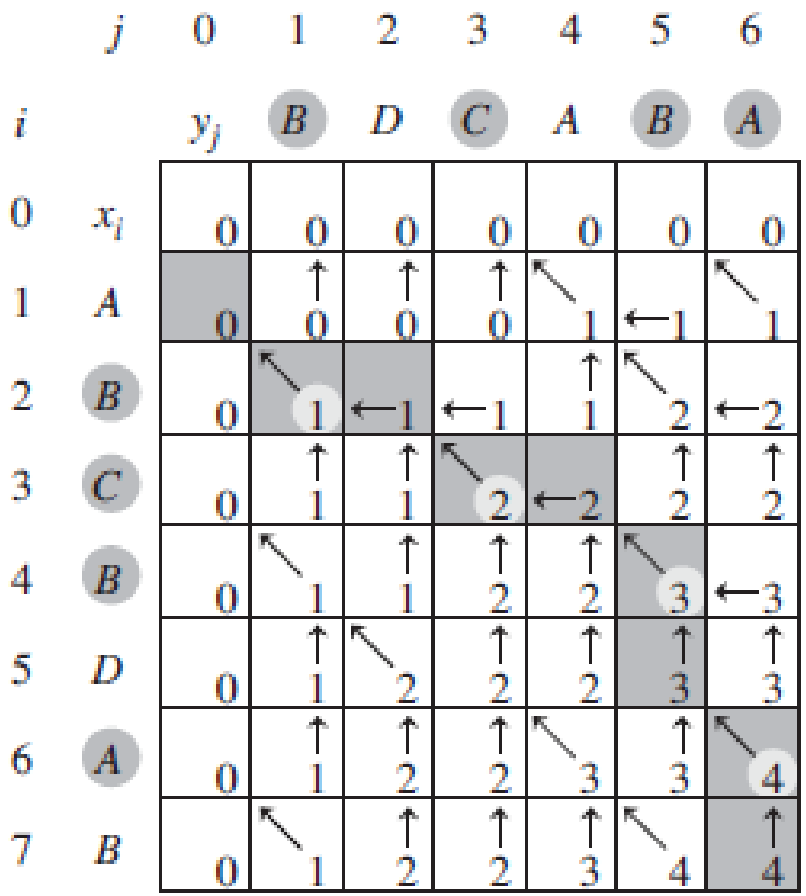
Αλγόριθμος

- Γεμίζει τον πίνακα c γραμμές
- Ο πίνακας b δείχνει στο κελί που αντιστοιχεί στη βέλτιστη λύση του υπο-προβλήματος όταν υπολογίζουμε το c[I,j]
- Το μήκος της LCS βρίσκεται στο c[m,n]

```
LCS-LENGTH(X, Y)
 1  m = X.length
 2  n = Y.length
 3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
 4  for i = 1 to m
 5      c[i, 0] = 0
 6  for j = 0 to n
 7      c[0, j] = 0
 8  for i = 1 to m
 9      for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j − 1]
17              b[i, j] = "←"
18  return c and b
```

Longest Common Subsequence (11/12)

Παράδειγμα εκτέλεσης



i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Longest Common Subsequence (12/12)

Εκτύπωση της LCS

- Κλήση

  PRINT-LCS$(b, X, X.length, Y.length)$

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2      return
3  if b[i, j] == "↖"
4      PRINT-LCS(b, X, i − 1, j − 1)
5      print x_i
6  elseif b[i, j] == "↑"
7      PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

Knapsack Problem (1/7)

Υιοθετούμε λύση με χρήση δυναμικού προγραμματισμού

Αντικείμενα: n, βάρη: w_i, αξίες: v_j

Χωρητικότητα σάκου: W

Για τη λύση του δυναμικού προγραμματισμού πρέπει να βρούμε την αναδρομική σχέση

Σε κάθε βήμα, παρουσιάζεται ένα στιγμιότυπο του σάκου σε σχέση με τις λύσεις στα μικρότερα προβλήματα

Knapsack Problem (2/7)

Εστιάζουμε σε ένα στιγμιότυπο των πρώτων i αντικειμένων

Αντικείμενα: $1<=i<=n$, Βάρη: $w_1, w_2, \ldots, w_i$, αξίες: $v_1, v_2, \ldots, v_i$

Χωρητικότητα σάκου: $1<=j<=W$

Έστω F(i,j) είναι η αξία μιας βέλτιστης λύσης

Μπορούμε να διαιρέσουμε όλα τα υποσύνολα των πρώτων i αντικειμένων που ταιριάζουν στη χωρητικότητα j του σάκου σε δύο κατηγορίες:

- Αυτά που δεν περιέχουν το i αντικείμενο
- Αυτά που περιέχουν το i αντικείμενο

Knapsack Problem (3/7)

Στα σύνολα που δεν περιέχουν το i αντικείμενο, η αξία ενός βέλτιστου υποσυνόλου είναι F(i-1,j)

Στα σύνολα που δεν περιέχουν το i αντικείμενο, ένα βέλτιστο υποσύνολο αποτελείται από ένα βέλτιστο υποσύνολο των πρώτων i-1 αντικειμένων που χωράνε στο σάκο με χωρητικότητα $j-w_i$

- Η αξία τους είναι $v_i+F(i-1,j-w_i)$

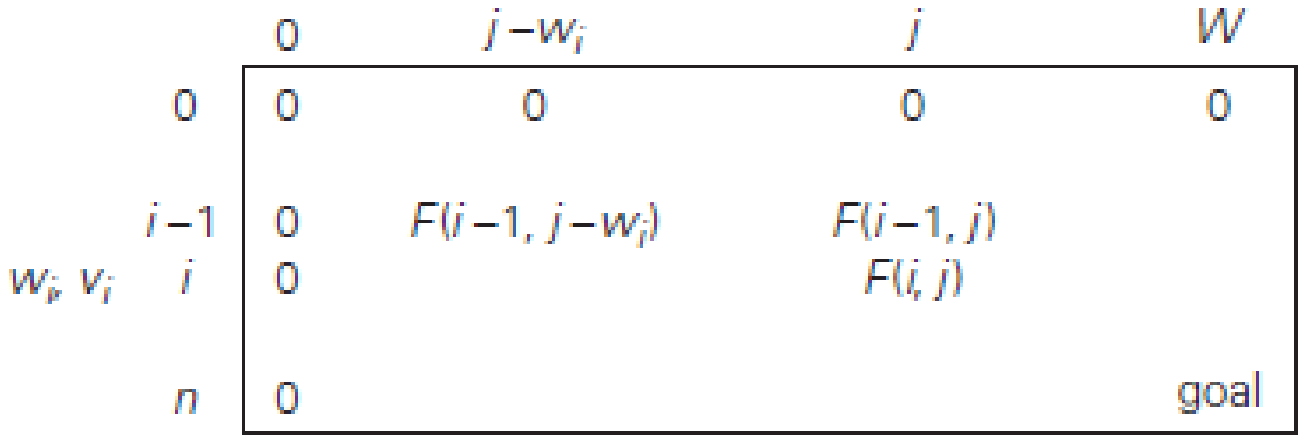
Συνεπώς

$$F(i, j) = \begin{cases} \max\{F(i-1, j), v_i + F(i-1, j-w_i)\} & \text{if } j - w_i \geq 0 \\ F(i-1, j) & \text{if } j - w_i < 0 \end{cases}$$

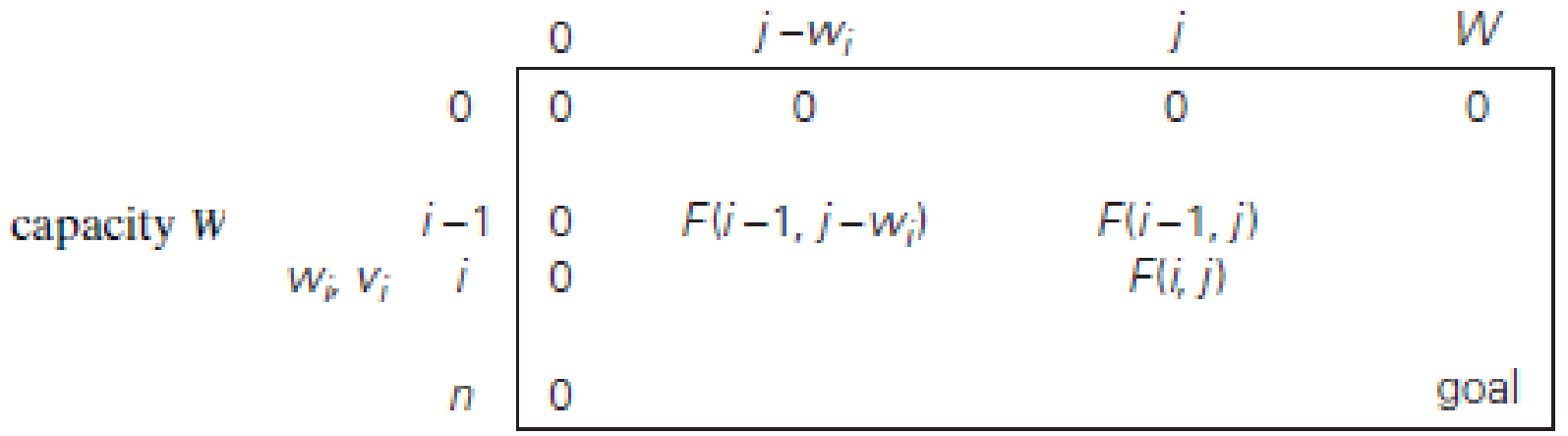
$$F(0, j) = 0 \text{ for } j \geq 0 \quad \text{and} \quad F(i, 0) = 0 \text{ for } i \geq 0$$

Knapsack Problem (4/7)

Για να υπολογίσουμε την τιμή ενός κελιού σε μια γραμμή i και στήλη j, F(i,j), υπολογίζουμε το μέγιστο ανάμεσα στην ***προηγούμενη γραμμή και την ίδια στήλη***, και στην εγγραφή ***στην προηγούμενη γραμμή κα w_i στήλες προς τα αριστερά***



item	weight	value
1	2	\$12
2	1	\$10
3	3	\$20
4	2	\$15



$$F(i, j) = \begin{cases} \max\{F(i-1, j),\ v_i + F(i-1, j-w_i)\} & \text{if } j - w_i \geq 0 \\ F(i-1, j) & \text{if } j - w_i < 0 \end{cases}$$

$w_1 = 2,\ v_1 = 12$
$w_2 = 1,\ v_2 = 10$
$w_3 = 3,\ v_3 = 20$
$w_4 = 2,\ v_4 = 15$

	capacity j					
i	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	**37**

Knapsack Problem (6/7)

Απαλείφουμε τα πλεονάζοντα υποπροβλήματα με χρήση των memory functions – επιλύουμε μόνο τα απαραίτητα υπο-προβλήματα

ALGORITHM *MFKnapsack*(i, j)

```
//Implements the memory function method for the knapsack problem
//Input: A nonnegative integer i indicating the number of the first
//       items being considered and a nonnegative integer j indicating
//       the knapsack capacity
//Output: The value of an optimal feasible subset of the first i items
//Note: Uses as global variables input arrays Weights[1..n], Values[1..n],
//and table F[0..n, 0..W] whose entries are initialized with −1's except for
//row 0 and column 0 initialized with 0's
if F[i, j] < 0
    if j < Weights[i]
        value ← MFKnapsack(i − 1, j)
    else
        value ← max(MFKnapsack(i − 1, j),
                     Values[i] + MFKnapsack(i − 1, j − Weights[i]))
    F[i, j] ← value
return F[i, j]
```

Knapsack Problem (7/7)

Παράδειγμα εκτέλεσης

		capacity j					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2,\ v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1,\ v_2 = 10$	2	0	—	12	22	—	22
$w_3 = 3,\ v_3 = 20$	3	0	—	—	22	—	32
$w_4 = 2,\ v_4 = 15$	4	0	—	—	—	—	**37**

Άπληστη Μέθοδος

Εισαγωγή (1/2)

Η άπληστη μέθοδος στοχεύει στη λύση που φαίνεται η βέλτιστη μέχρι στιγμής

Επιλέγει την τοπικά βέλτιστη λύση με την ελπίδα ότι αυτή η λύση θα είναι τελικά η βέλτιστη

Μπορούμε να συνθέσουμε μια ολικά βέλτιστη λύση μέσα από την επιλογή τοπικών βέλτιστων λύσεων

Οι άπληστοι αλγόριθμοι δεν δίνουν πάντα τις βέλτιστες

Πρόκειται για μια ισχυρή μέθοδο που χρησιμοποιείται σε μεγάλο εύρος προβλημάτων

Εισαγωγή (2/2)

Διαφορές με το δυναμικό προγραμματισμό

- Στην άπληστη μέθοδο επιλέγουμε την τοπικά βέλτιστη λύση χωρίς να εστιάζουμε στις λύσεις των υπο-προβλημάτων
- Οι επιλογές που κάνουμε στο δυναμικό προγραμματισμό βασίζονται στις λύσεις των υπο-προβλημάτων
- Τυπικά στο δυναμικό προγραμματισμό ακολουθούμε μια bottom-up προσέγγιση όπου από πιο απλά προβλήματα φτάνουμε στη λύση πιο σύνθετων προβλημάτων
- Οι επιλογές στην άπληστη μέθοδο μπορεί να βασίζονται σε παλιές επιλογές αλλά ποτέ δεν βασίζονται σε μελλοντικές
- Ο δυναμικός προγραμματισμός επιλύει τα υπο-προβλήματα πριν κάνει μια επιλογή, ενώ η άπληστη μέθοδος κάνει την επιλογή πριν λύσει τα υπο-προβλήματα

Κώδικες Huffman (1/19)

Οι κώδικες Huffman στοχεύουν στη συμπίεση δεδομένων

Θεωρούμε τα δεδομένα ως ακολουθίες χαρακτήρων

Η προσέγγιση με την άπληστη μέθοδο υιοθετεί ένα πίνακα που αποθηκεύει πόσο συχνά εμφανίζεται ο κάθε ένας χαρακτήρας

Απεικονίζει κάθε χαρακτήρα σαν ένα binary string

Κώδικες Huffman (2/19)

Παράδειγμα

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Με την fixed length μέθοδο χρειαζόμαστε 300.000 bits για ένα αρχείο των 100.000 χαρακτήρων

Με την variable length μέθοδο χρειαζόμαστε

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1{,}000 = 224{,}000 \text{ bits}$$

Κώδικες Huffman (3/19)

Εστιάζουμε στα prefix codes, κώδικες που δεν είναι προθέματα άλλων κωδίκων

Η τελική κωδικοποίηση είναι η συγχώνευση των κωδίκων για κάθε χαρακτήρα

Αν το prefix code είναι μοναδικό για κάθε χαρακτήρα τότε η αποκωδικοποίηση είναι εύκολη

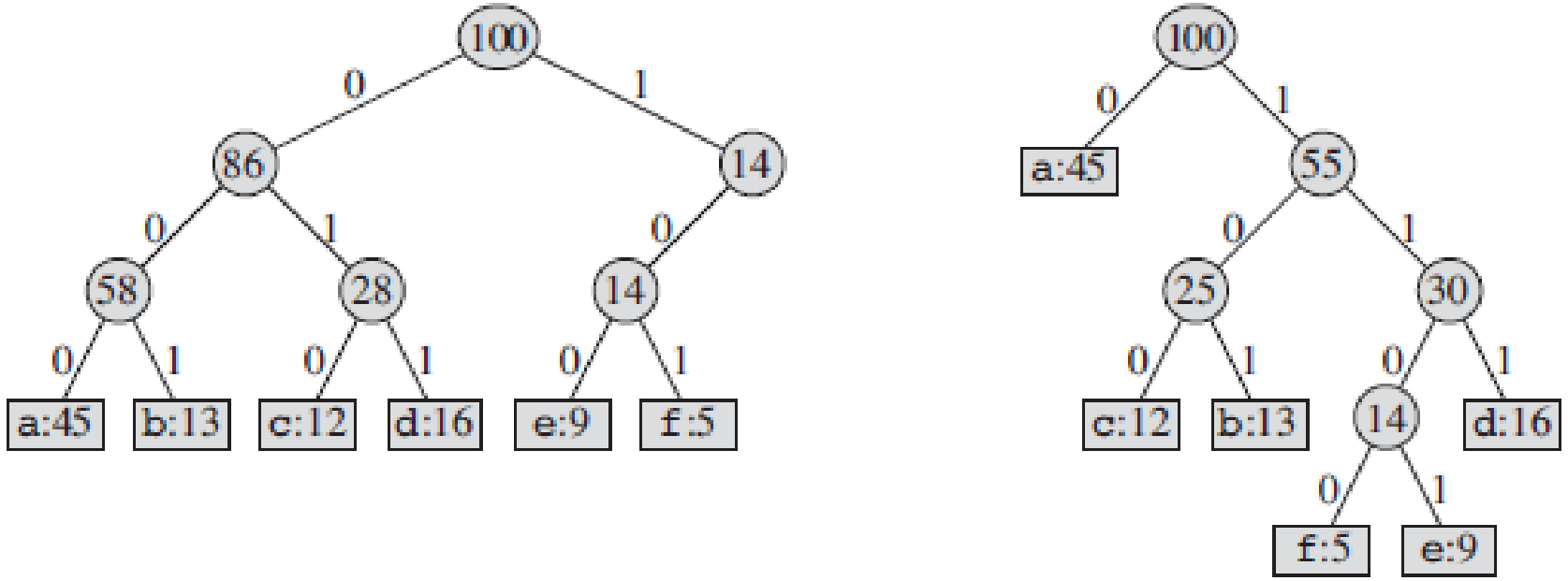
Υιοθετούμε ένα δυαδικό δένδρο στο οποίο τα φύλλα είναι οι χαρακτήρες

Ο δυαδικός κώδικας είναι η διαδρομή από τη ρίζα του δένδρου προς τα φύλλα

Κώδικες Huffman (4/19)

Το 0 αντιστοιχεί στην κίνηση προς το αριστερό παιδί, ενώ το 1 την κίνηση προς το δεξιό παιδί

Παράδειγμα



Κώδικες Huffman (5/19)

Ο βέλτιστος κώδικας εμπλέκει ένα πλήρες δυαδικό δένδρο στο οποίο ο κάθε κόμβος έχει δύο παιδιά

Η fixed length προσέγγιση δεν οδηγεί σε βέλτιστο κώδικα

Έστω C το αλφάβητο για το οποίο πρέπει να κατασκευάσουμε τον κώδικα

Το δένδρο έχει |C| φύλλα

Το δένδρο έχει ακριβώς |C|-1 εσωτερικούς κόμβους

Κώδικες Huffman (6/19)

Δοσμένου ενός δένδρου Τ που αντιστοιχεί σε ένα κώδικα, μπορούμε να υπολογίσουμε εύκολα το πλήθος των bits που απαιτούνται για να κωδικοποιήσουμε το αρχείο

Έστω c ένας χαρακτήρας του C, c.freq είναι η συχνότητα του c και $d_T(c)$ είναι το βάθος του c στο δένδρο

Το $d_T(c)$ είναι ταυτόχρονα και το μήκος του κώδικα του c

Το πλήθος των bits θα είναι

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$$

Κώδικες Huffman (7/19)

Υποθέτουμε ότι το C είναι ένα σύνολο n χαρακτήρων

Ο άπληστος αλγόριθμος ξεκινά από τα φύλλα και εκτελεί |C|-1 συγχωνεύσεις για να δημιουργήσει το τελικό δένδρο

Υιοθετεί μια ουρά ελάχιστης προτεραιότητας (min-priority queue)

Μέσω της ουράς αναγνωρίζει τα δύο ελάχιστα συχνά στοιχεία για να συγχωνεύσει

Όταν γίνεται η συγχώνευση το αποτέλεσμα είναι ένα νέο στοιχείο του οποίου η συχνότητα είναι το άθροισμα των συχνοτήτων των δύο στοιχείων

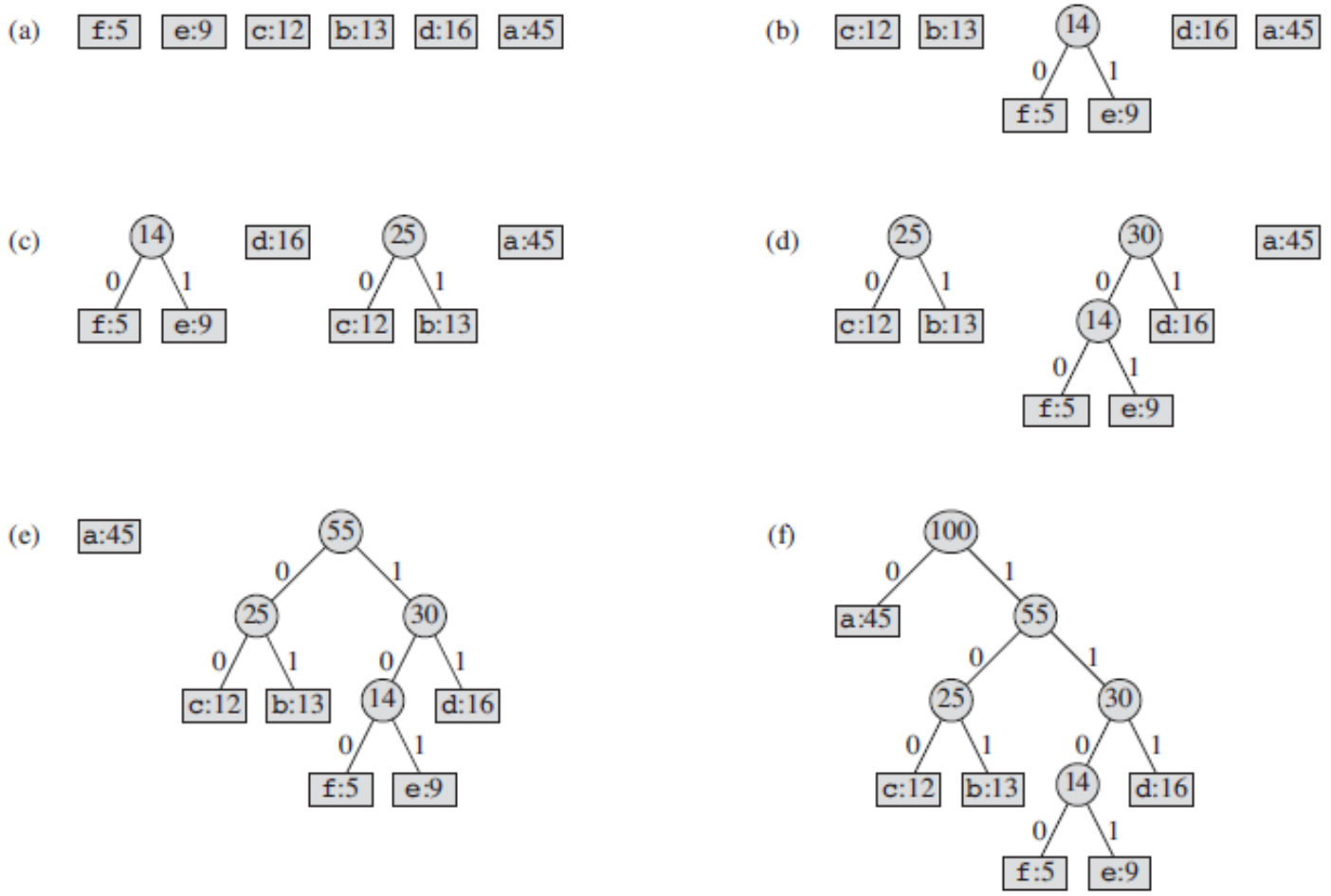
Κώδικες Huffman (8/19)

Αλγόριθμος

- Η αρχική ουρά έχει μέγεθος ίσο με το n
- Οι γραμμές 3-8 εξάγουν επαναληπτικά τους δύο κόμβους x, y με την ελάχιστη συχνότητα αντικαθιστώντας τους στην ουρά με ένα νέο κόμβο z
- Η συχνότητα του z είναι το άθροισμα των συχνοτήτων
- Ο z έχει αριστερό παιδί το x και δεξιό παιδί το y

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Κώδικες Huffman (9/19)



Κώδικες Huffman (10/19)

Lemma 1

Για ένα αλφάβητο C όπου ο κάθε χαρακτήρας c έχει συχνότητα c.freq, αν x, y είναι οι δύο χαρακτήρες με τη χαμηλότερη συχνότητα, τότε υπάρχει ένα βέλτιστο πρόθεμα (prefix) στο οποίο οι κώδικες για τα x, y έχουν το ίδιο μήκος αλλά διαφέρουν στο τελευταίο bit

Απόδειξη

Έστω a, b δύο χαρακτήρες που είναι αδέρφια σε μέγιστο βάθος στο T.

Υποθέτουμε ότι $a.freq \leq b.freq$ και $x.freq \leq y.freq$.

Κώδικες Huffman (11/19)

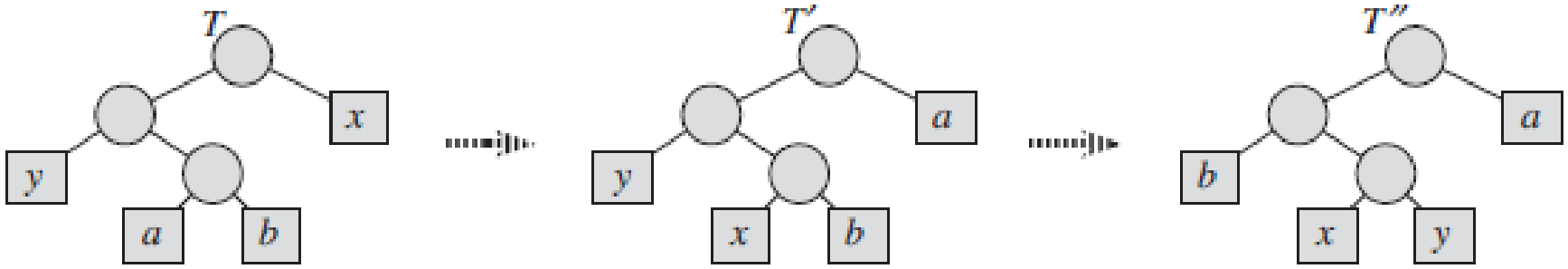
Αφού τα x.freq & y.freq είναι οι ελάχιστες συχνότητες, και a.freq & b.freq είναι δύο συχνότητες χαρακτήρων θα έχουμε x.freq<=a.freq & y.freq<=b.freq

Αν x.freq=a.freq & y.freq=b.freq τότε θα έχουμε x.freq=a.freq=y.freq=b.freq οπότε το λήμμα ισχύει

Αν υποθέσουμε ότι x.freq<>b.freq, τότε θα είναι x<>b

Προχωρούμε σε αλλαγή στο δένδρο με αντιμεταθέσεις θέσεων στα a,x & b,y ως ακολούθως:

Κώδικες Huffman (12/19)

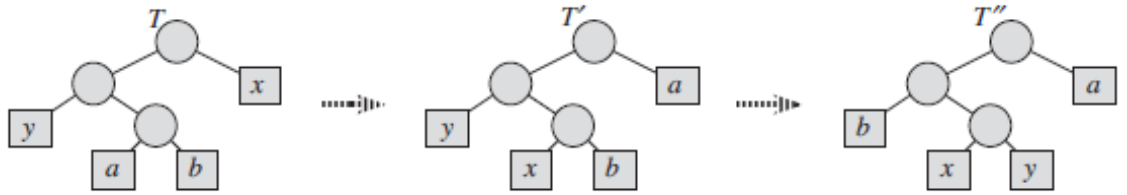


Αν x=b & y<>a το T’’ δεν έχει τα x,y σαν ‘αδέρφια’

Η διαφορά στο κόστος των δένδρων T, T’ είναι:

Κώδικες Huffman (13/19)

$$
\begin{aligned}
& B(T) - B(T') \\
&= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_{T'}(x) - a.freq \cdot d_{T'}(a) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_T(a) - a.freq \cdot d_T(x) \\
&= (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\geq 0
\end{aligned}
$$



διότι τα a.freq-x.freq & $d_T(a)$-$d_T(x)$ είναι μη αρνητικά

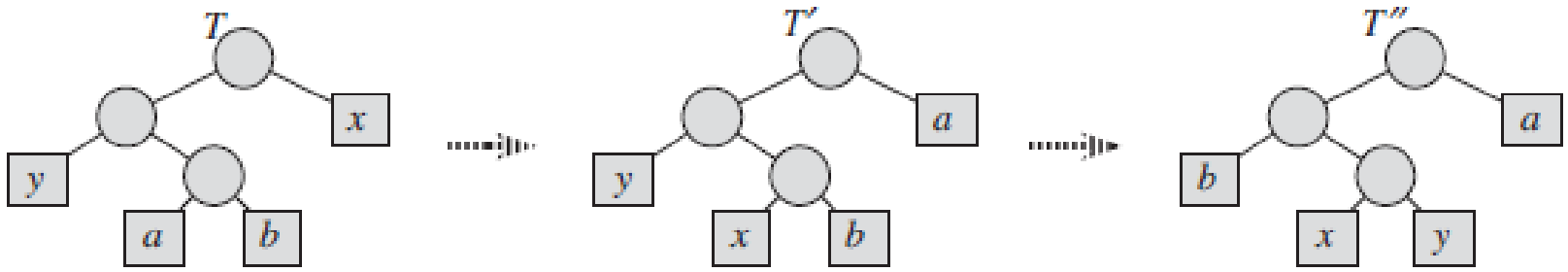
Το a.freq-x.freq είναι μη αρνητικό διότι το x έχει ελάχιστη συχνότητα και το $d_T(a)$-$d_T(x)$ είναι μη αρνητικό αφού το a είναι ένα φύλλο μέγιστου βάθους

Κώδικες Huffman (14/19)

Αν αντιμεταθέσουμε τα y, b, το κόστος δεν αυξάνει, οπότε το B(T')-B(T'') είναι μη αρνητικό

Συνεπώς, B(T'')<=B(T) και αφού το T είναι το βέλτιστο έχουμε B(T)<=B(T'') το οποίο μας δίνει ότι B(T'')=B(T)

Οπότε το T'' είναι βέλτιστο και τα x, y είναι αδέρφια



Κώδικες Huffman (15/19)

Lemma 2

Έστω C είναι ένα αλφάβητο με συχνότητα c.freq για κάθε χαρακτήρα c του αλφαβήτου. Έστω x,y δύο χαρακτήρες με την ελάχιστη συχνότητα. Έστω C’ ένα αλφάβητο χωρίς τα x,y και με ένα νέο χαρακτήρα z τέτοιο ώστε C’=C-{x,y}U{z} και z.freq=x.freq+y.freq. Έστω T’ οποιοδήποτε δένδρο που αναπαριστά τον κώδικα του βέλτιστου προθέματος για το αλφάβητο C’. Τότε το δένδρο T το οποίο εξάγεται από το T’ αντικαθιστώντας το φύλο z με ένα εσωτερικό κόμβο που έχει παιδιά τα x, y, αναπαριστά ένα βέλτιστο κώδικα για το αλφάβητο C.

Κώδικες Huffman (16/19)

Απόδειξη

Για κάθε χαρακτήρα $c \in C - \{x, y\}$.

Έχουμε $d_T(c) = d_{T'}(c)$

Και έτσι $c.freq \cdot d_T(c) = c.freq \cdot d_{T'}(c)$

Αφού $d_T(x) = d_T(y) = d_{T'}(z) + 1$

Έχουμε

$$\begin{aligned} x.freq \cdot d_T(x) + y.freq \cdot d_T(y) &= (x.freq + y.freq)(d_{T'}(z) + 1) \\ &= z.freq \cdot d_{T'}(z) + (x.freq + y.freq) \end{aligned}$$

Κώδικες Huffman (17/19)

Συμπεραίνουμε πως $B(T) = B(T') + x.freq + y.freq$

Ή $B(T') = B(T) - x.freq - y.freq$

Έστω τώρα ότι το Τ δεν αναπαριστά ένα βέλτιστο κώδικα

Υπάρχει ένα Τ'' τέτοιο ώστε $B(T'') < B(T)$

Το Τ'' έχει τα x, y ως αδέρφια

Έστω το Τ''' το δένδρο Τ'' με το κοινό πατέρα των x, y που έχουν αντικατασταθεί από το z με συχνότητα z.freq=x.freq+y.freq

Κώδικες Huffman (18/19)

Τότε

$$\begin{aligned} B(T''') &= B(T'') - x.freq - y.freq \\ &< B(T) - x.freq - y.freq \\ &= B(T') \end{aligned}$$

Το οποίο δεν ισχύει αφού το Τ’ αναπαριστά ένα βέλτιστο κώδικα του C’

Έτσι το Τ είναι ο βέλτιστος κώδικας του C

Κώδικες Huffman (19/19)

Θεώρημα

Η διαδικασία HUFFMAN παράγει ένα βέλτιστο κώδικα

Απόδειξη

Εξάγεται εύκολα από τα δύο προηγούμενα λήματα