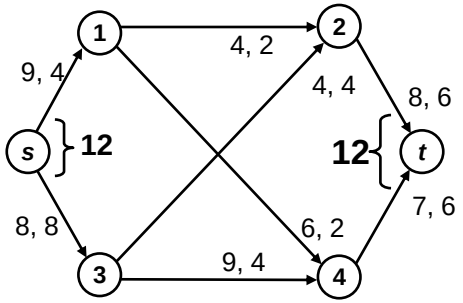
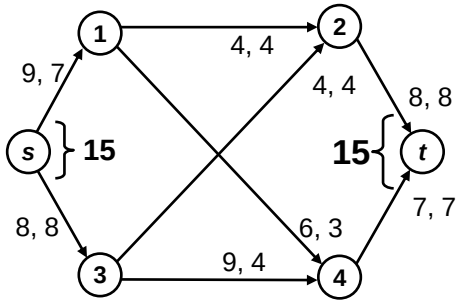
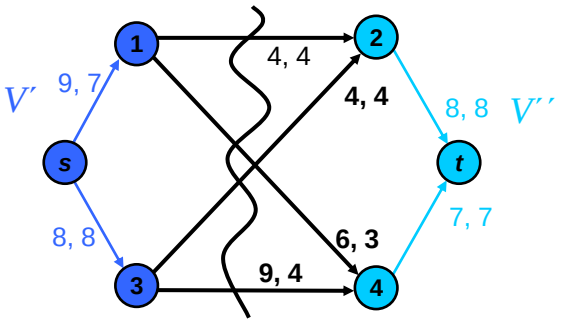
Ροές

- Ορισμοί
  - Δίκτυο Μεταφοράς ή Ροής
    - Βεβαρημένο γράφημα με χωρητικότητες
    - Πηγή, Καταβόθρα
  - Ροή
    - Απεικόνιση τιμών στις ακμές, φραγμένων από τις αντίστοιχες χωρητικότητες
    - Κορεσμένες, Ακόρεστες ακμές
    - Μέγιστη ροή
  - Τομή
    - Χωρητικότητα Τομής
    - Χωρητικότητα Δικτύου
- Ισοδυναμία Μέγιστης Ροής-Ελάχιστης Τομής

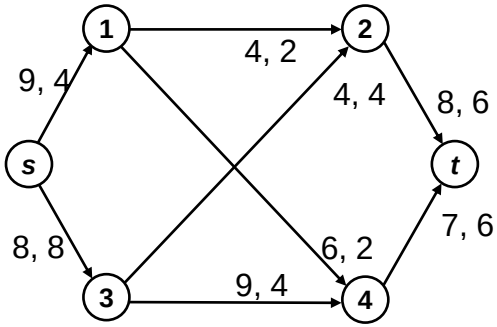


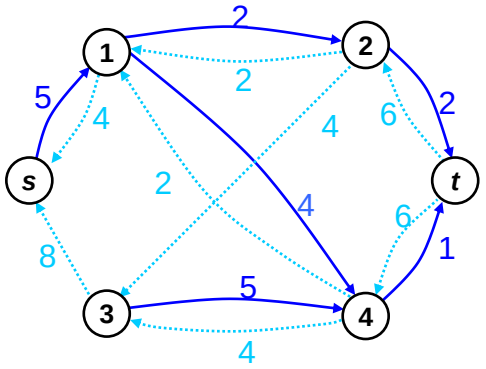


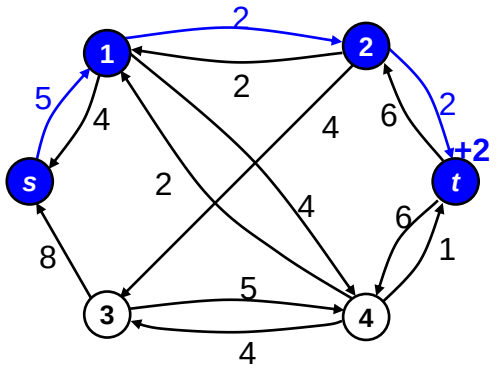


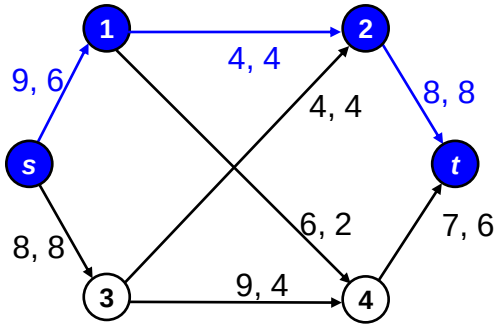
Βοηθητικές Συνδυαστικές Δομές

- Υπολειμματικό Γράφημα
  - Για κάθε ροή ακμής, περιέχει δύο ακμές: ***προωθήσεως*** και ***ακυρώσεως ροής***
- Επαυξητικό μονοπάτι
  - Κάθε μονοπάτι στο υπολειμματικό γράφημα από την πηγή στην καταβόθρα, με τιμή ροής ίση με την ελάχιστη τιμή ακμής
  - Βοηθά στην εύρεση, νέας, βελτιωμένης ροής









Θεώρημα Επαυξητικού Μονοπατιού

- Ροή Μέγιστη ανν το αντίστοιχο υπολειμματικό γράφημα δεν διαθέτει επαυξητικό μονοπάτι
  - Έστω μέγιστη ροή. Εάν υπήρχε επαυξητικό μονοπάτι, τότε θα ήταν δυνατή η επαύξηση (άτοπο)
  - Εάν δεν υπάρχει επαυξητικό μονοπάτι, οι κορυφές χωρίζονται σε δύο σύνολα *V*, *V'*: το πρώτο περιλαμβάνει τις προσπελάσιμες από την πηγή κορυφές και το δεύτερο τις υπόλοιπες. Τότε,
    - Ακμές *V* προς *V'* κορεσμένες
    - Ακμές *V'* προς *V* δίχως ροή
    - *V*, *V'* τομή, όπου η χωρητικότητα φράσσει την ροή

Θεώρημα Ακέραιας Ροής

- Εάν οι χωρητικότητες είναι ακέραιες τότε και η μέγιστη ροή έχει ακέραια τιμή
  - Παρατηρούμε ότι τα επαυξητικά μονοπάτια έχουν ακέραιες τιμές ροών

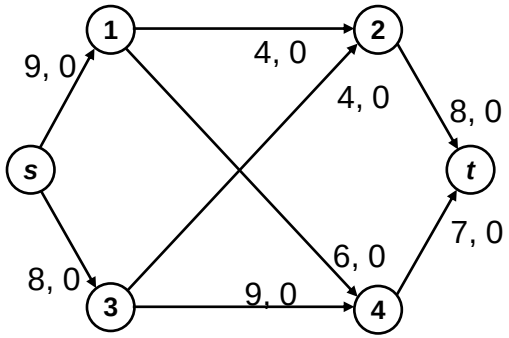
Μέθοδος Ford-Fulkerson

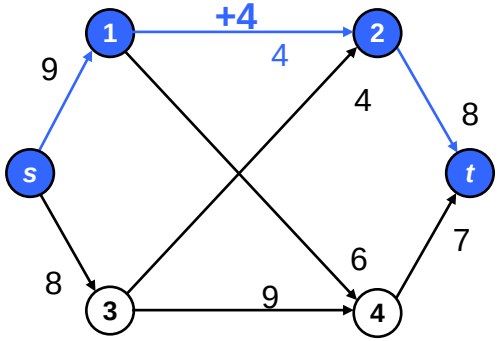
- Βασίζεται στο θεώρημα επαυξητικού μονοπατιού:

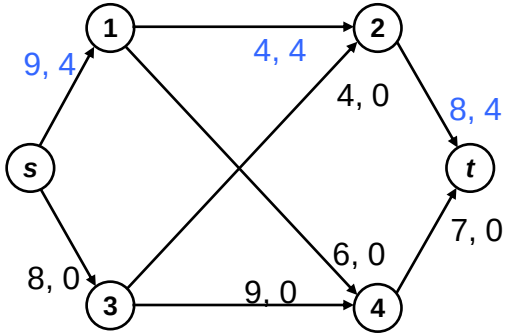
  «*όσο υπάρχουν επαυξητικά μονοπάτια, διαλέγουμε ένα και επαυξάνουμε την ροή κατά μήκος του*»

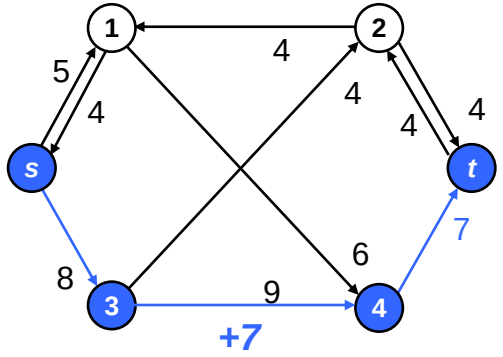
- Αποκαλείται μέθοδος, γιατί ***δεν*** προσδιορίζεται πώς θα βρεθεί το επαυξητικό μονοπάτι
- Στην σχετική δημοσίευση, οι Ford-Fulkerson είχαν προτείνει γενικευμένο ψάξιμο στο υπολειμματικό γράφημα, με ***τυχαία*** ουρά προτεραιότητας

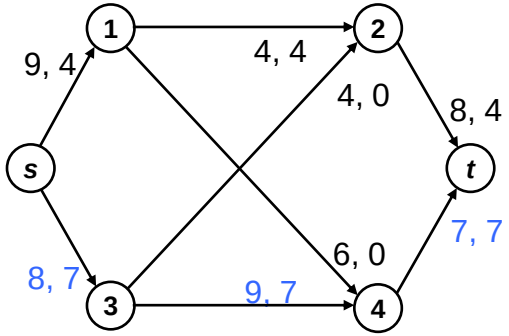
Παράδειγμα

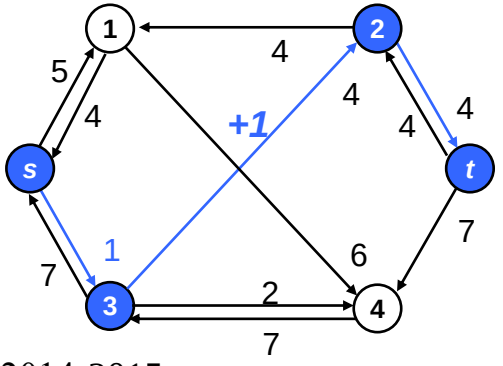

1
2
s
t
3
4
9, 0
4, 0
4, 0
8, 0
8, 0
6, 0
7, 0
9, 0


1
2
s
t
3
4
+4
4
9
4
8
8
6
7
9

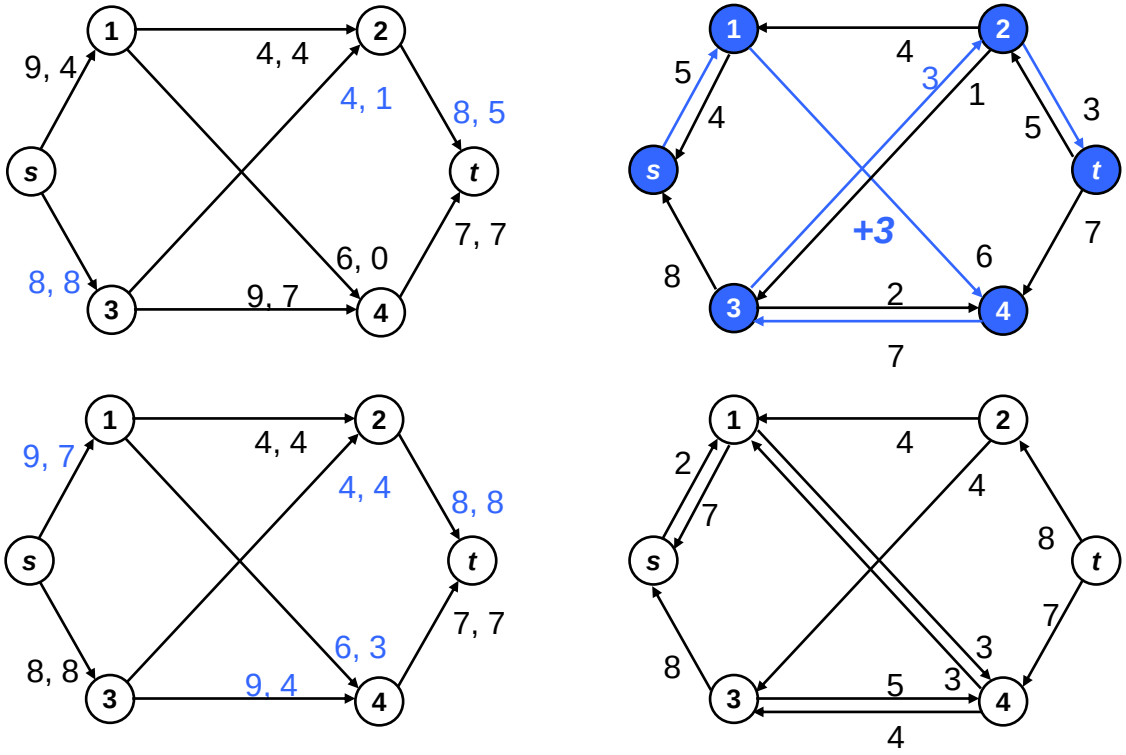

1
2
s
t
3
4
9, 4
4, 4
4, 0
8, 4
8, 0
6, 0
7, 0
9, 0


1
2
s
t
3
4
5
4
4
4
4
4
7
8
6
9
+7


1
2
s
t
3
4
9, 4
4, 4
4, 0
8, 4
8, 7
6, 0
7, 7
9, 7

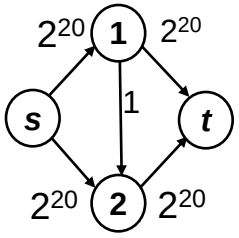

1
2
s
t
3
4
5
4
4
+1
4
4
7
1
7
6
2
7

Παράδειγμα (συν.)



Ανάλυση Μεθόδου

- Πολυπλοκότητα $O(VEM)$=O($|f|E$)
  - Στην χειρότερη περίπτωση, κάθε βήμα θα αυξάνει την χωρητικότητα κατά μία μονάδα.
  - Άρα $|f|$ ψαξίματα μονοπατιού, κόστους $O(V+E)$=O(E).
  - Επιπλέον, $|f|=O(VM)$ καθώς η τιμή τομής είναι $O(VM)$
  - Το όριο είναι ***σφιχτό*** (***tight***), όπως δείχνει και το διπλανό στιγμιότυπο, και ***ψευδοπολυωνυμικό***, γιατί εξαρτάται από την $|f|$
- Άλλο μειονέκτημα:
  - Αστάθεια συγκλίσεως στην τελική ροή, εάν οι χωρητικότητες είναι πραγματικοί αριθμοί

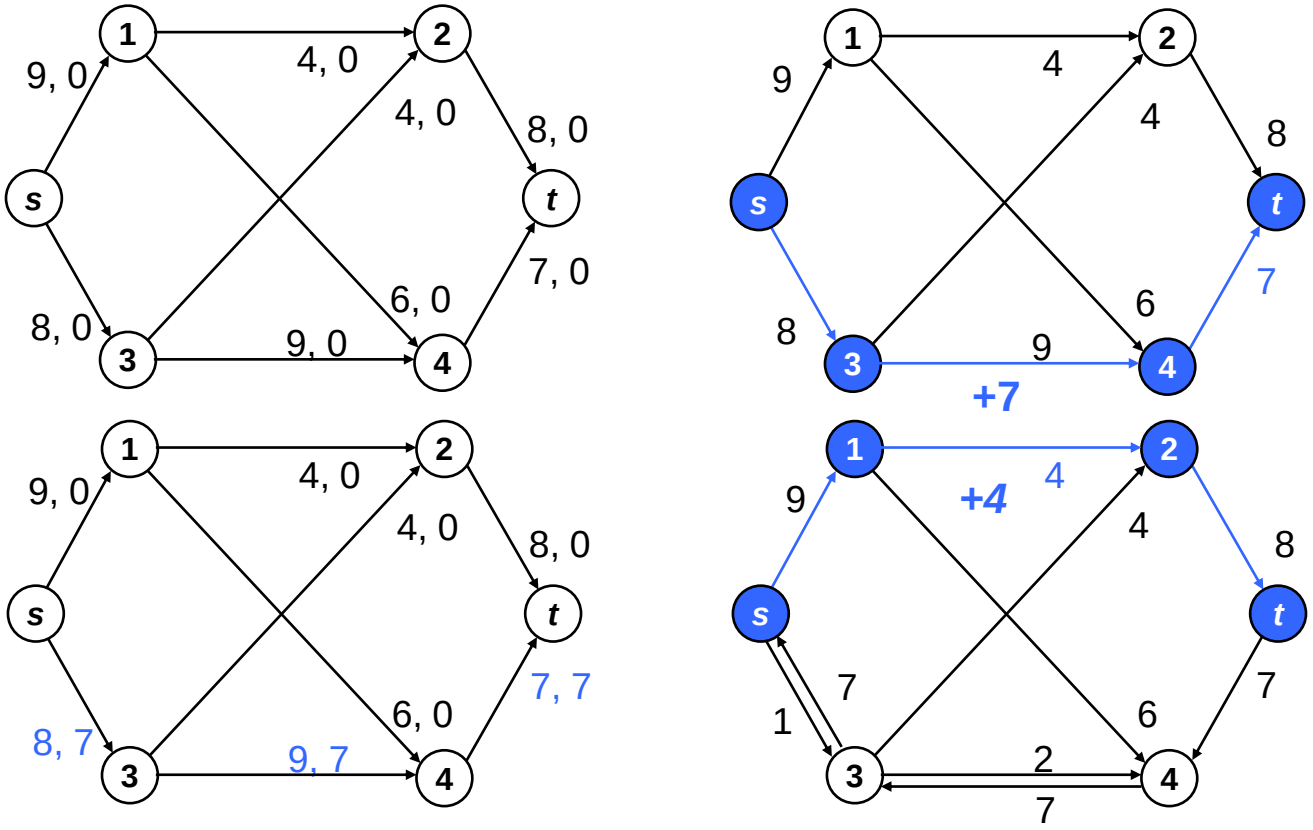


Αλγόριθμοι Edmonds-Karp

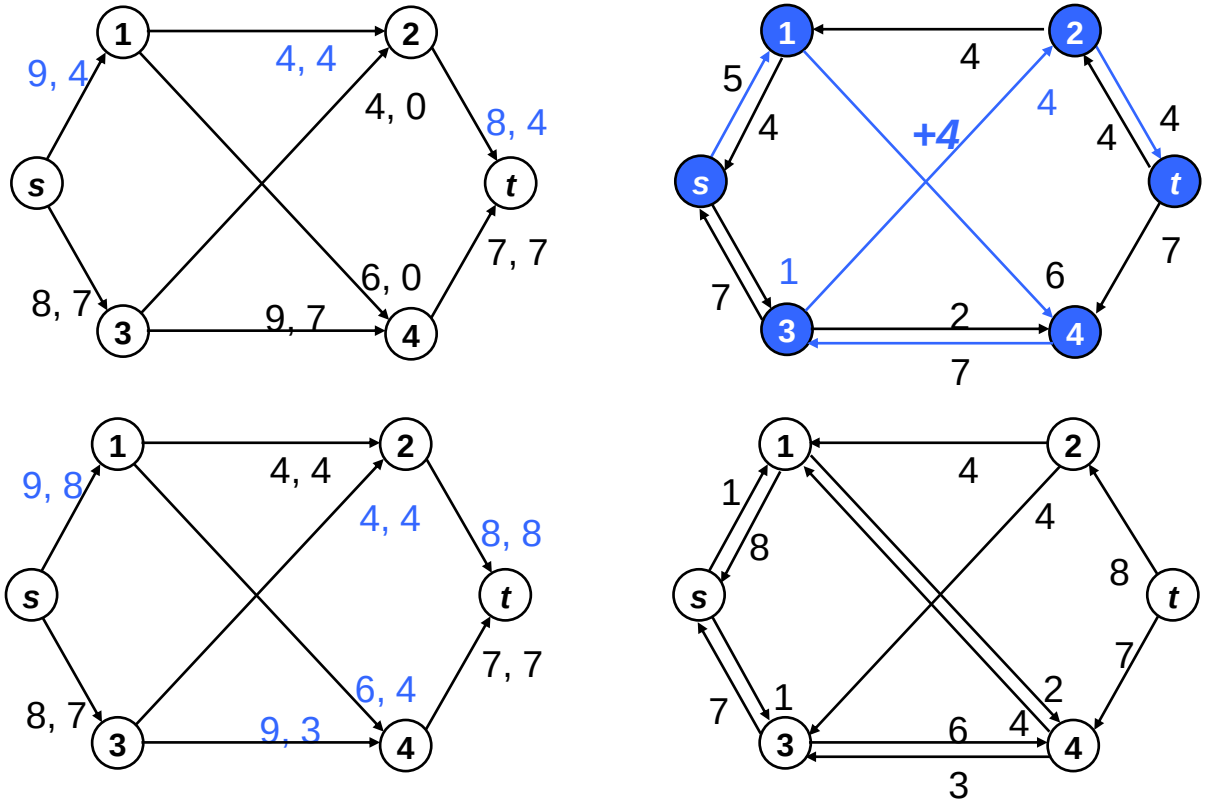
- Οι Edmonds-Karp παρατήρησαν πως, εάν τα επαυξητικά μονοπάτια αναζητηθούν
  - είτε *βάσει της μέγιστης τιμής ροής*
  - είτε *βάσει του μικρότερου πλήθους ακμών*,

  τότε η Μέθοδος Ford-Fulkerson γίνεται ***πολυωνυμική***

Παράδειγμα Μέγιστης Τιμής Επαυξητικού Μονοπατιού



Παράδειγμα Μέγιστης Τιμής Επαυξητικού Μονοπατιού (συν.)

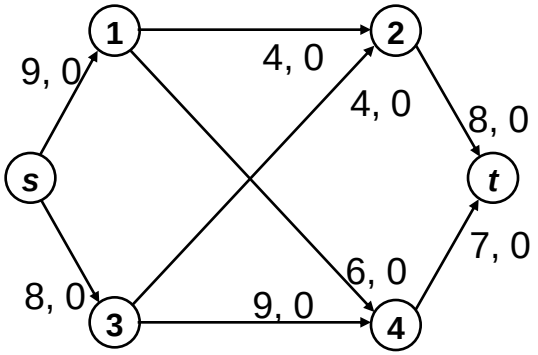


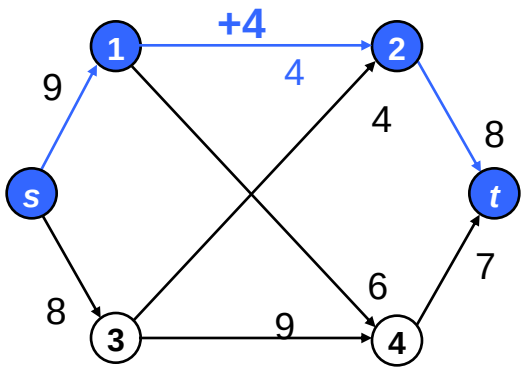
Ανάλυση

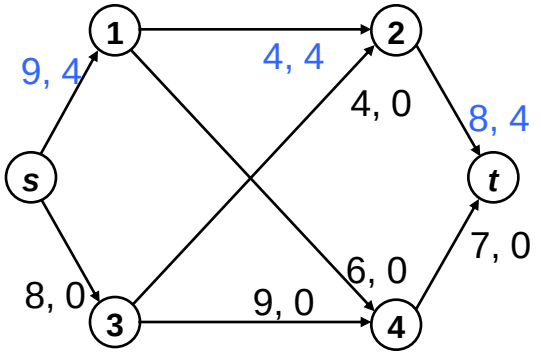
- Το πλήθος των επαυξητικών μονοπατιών είναι $O(E\log M)$, M μέγιστη τιμή χωρητικότητας
  - Έστω ότι η ροή f' έχει τιμή $|f'|$ και $|f|$ είναι η τελική τιμή
  - Υπάρχουν το πολύ E διακριτά επαυξητικά μονοπάτια, συνολικής ροής $|f| - |f'|$
  - Βάσει της αρχής του περιστερώνα, το μέγιστο μονοπάτι θα έχει τιμή ***τουλάχιστον*** $(|f| - |f'|)/E$
  - Θεωρούμε την ακολουθία των επαυξητικών βημάτων σε υποομάδες των 2Ε. Έστω μία υποομάδα $2E$ επαυξητικών βημάτων που δημιουργούν ροές $f'_1, f'_2, f'_3, \ldots, f'_i, \ldots$ και f' η ροή πριν την έναρξή της. Τότε,
    - Εάν κάθε ένα από αυτά τα βήματα προωθεί ροή τιμής *τουλάχιστον* $|f| - |f'|/2E$, τότε η διαδικασία τερματίζει μετά από $2E$ βήματα, το πολύ, και αυτή είναι η τελευταία υποομάδα που μελετούμε. Διαφορετικά,
    - Εάν υπάρχει έστω και ένα επαυξητικό βήμα, πχ, το i-στο, με τιμή ροής λιγότερη από $(|f| - |f'|)/(2E)$, τότε

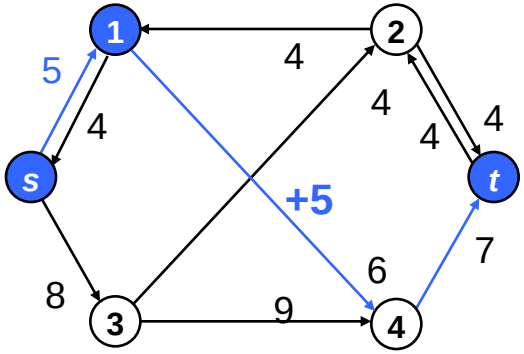
      $(|f| - |f'_{i-1}|)/E \le |f'_i| - |f'_{i-1}| < (|f| - |f'|)/(2E) \Rightarrow [(|f| - |f'_{i-1}|)/E]/[(|f| - |f'|)/E] \le 1/2$
    - Άρα, στην χειρότερη περίπτωση, μετά από $2E$ βήματα, υποδιπλασιάζεται η τιμή του «βαρύτερου» μονοπατιού
- Χρόνος $O(\log_{E/V} V \log M\ E(V+E))$, M μέγιστη χωρητικότητα ακμής
  - $V+E$ πράξεις στην ουρά προτεραιότητας, πχ, E/V-σωρός, σε κάθε έναν από τους $E\log M$ υπολογισμούς επαυξητικού μονοπατιού

Παράδειγμα Συντομότερου Επαυξητικού Μονοπατιού

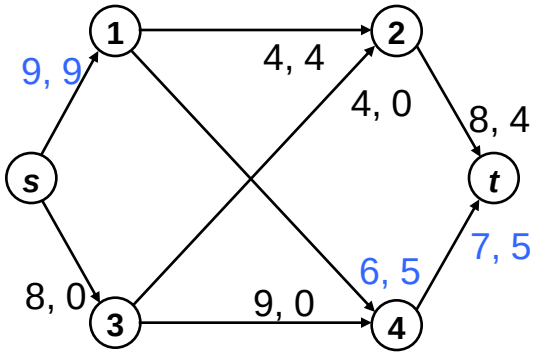


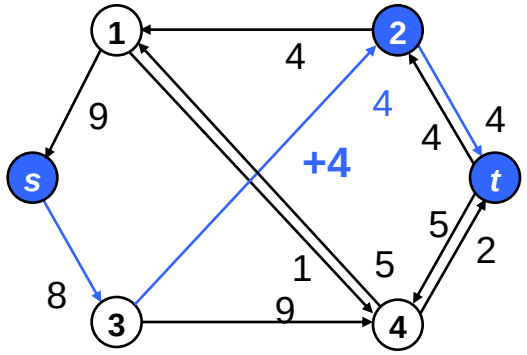


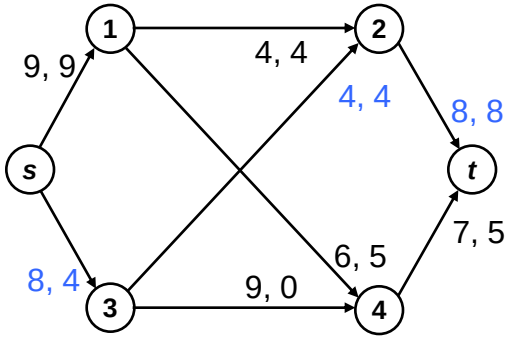


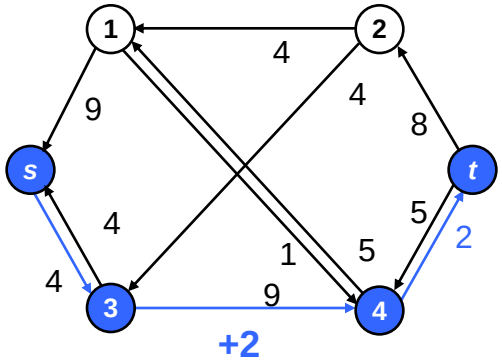


Παράδειγμα Συντομότερου Επαυξητικού Μονοπατιού (συν.)

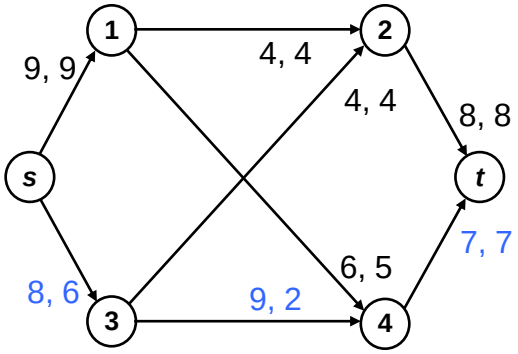


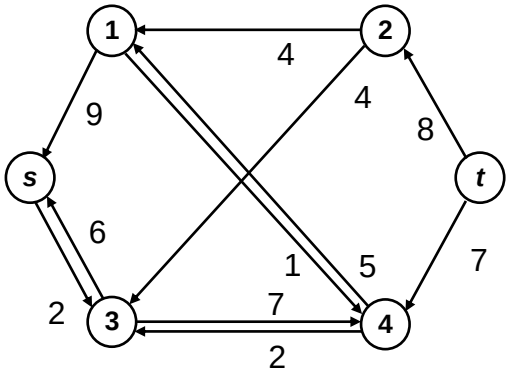






Παράδειγμα Συντομότερου Επαυξητικού Μονοπατιού (συν.)

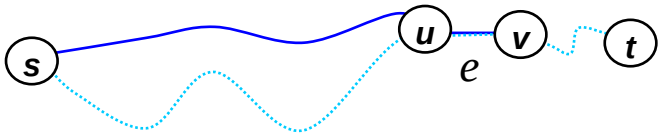




Ανάλυση

- **Ορθότητα (Σχεδιάγραμμα)** Στηρίζεται στις κάτωθι παρατηρήσεις:
  - **Πρώτον,** τα συντομότερα μονοπάτια συνεχώς αυξάνουν σε μήκος, και
  - **Δεύτερον,** κάθε ακμή του υπολειμματικού γραφήματος ξαναεμφανίζεται σε συντομότερο μονοπάτι που είναι κατά 2 ακμές μεγαλύτερο από την προηγούμενη φορά:

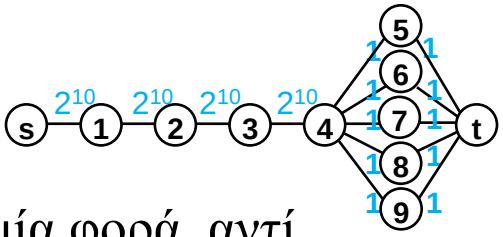
$$d(u) = d(v) + 1 \geq d'(v) + 1 = d'(u) + 1 + 1 = d'(u) + 2$$



  - Άρα, θα χρησιμοποιηθούν συνολικά $VE/2$ επαυξητικά μονοπάτια
- Χρόνος $O(E(VE))=O(VE^2)$

Τεχνική Προροής-Προωθήσεως

- Διαίσθηση
  - Γιατί να μην σταλεί με μιας ροή 5 στην κορυφή 4;
  - Το $s \rightarrow 4$ θα συμμετείχε μόνο μία φορά, αντί σε 5 διαφορετικά επαυξητικά μονοπάτια, μεγέθους 1



- Τεχνική Προροής-Προωθήσεως
  - Προσπαθούμε να οδηγήσουμε σε κορεσμό τις ακμές, διοχετεύοντας όση ροή επιτρέπεται
  - Οι ενδιάμεσες κορυφές μπορούν να αποθηκεύουν πλεονάζουσα ροή – χαρακτηρίζονται, τότε, ως ***ενεργές (active)***
  - Έτσι δημιουργείται μία προσέγγιση ροής, η ***προροή***, η οποία θα συγκλίνει στην τελική «κανονική» ροή

Τεχνική Προροής-Προωθήσεως (συν.)

- Προκειμένου η προροή να μετασχηματιστεί σε νόμιμη ροή, πρέπει πρώτα να φθάσει ροή στην καταβόθρα και, κατόπιν, το πλεόνασμα να επιστραφεί στην πηγή
- Δηλαδή, ***πρέπει να εξαλειφθούν οι ενεργές κορυφές***. Πώς;
  - Υπολειμματικό γράφημα
  - Συνάρτηση ύψους στο υπολειμματικό γράφημα
- Συνάρτηση ύψους
  - $h(t) = 0$
  - $h(v) \leq h(u) + 1$, $(v,u) \in E_r$

  Π.χ., $h \equiv$ απόσταση από καταβόθρα
- Όταν $h(v) = h(u) + 1$, τότε η (v,u) καλείται ***νόμιμη***

Ιδιότητες

- Το ύψος μίας κορυφής v δεν ξεπερνά το μήκος της συντομότερης αποστάσεως από την v προς την καταβόθρα t
  - Έστω $v \to v_1 \to v_2 \to \cdots \to t$ το συντομότερο μονοπάτι. Τότε:

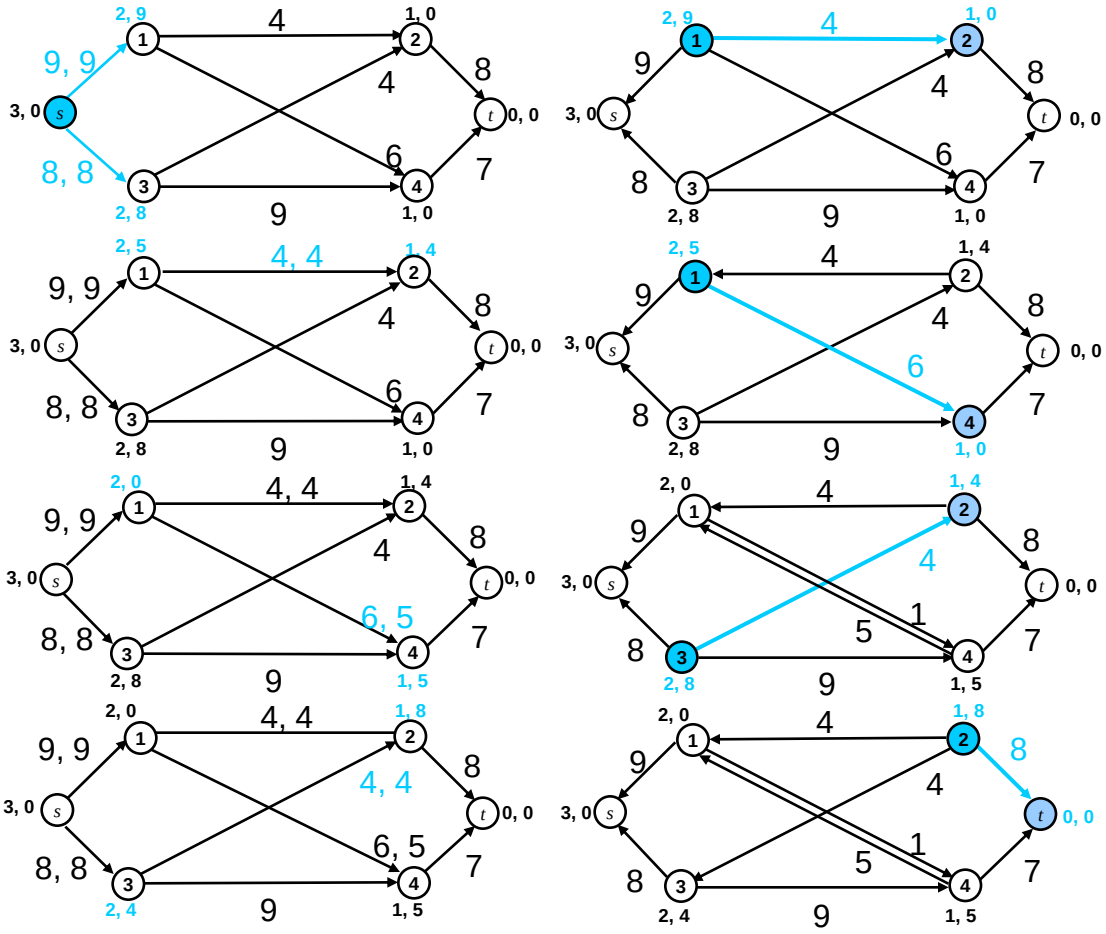
$$h(v) \leq h(v_1) + 1 \leq h(v_2) + 2 \leq \cdots \leq h(t) + d = 0 + d$$

- Όταν $h(v) > V$, τότε δεν υπάρχει μονοπάτι από την v προς την καταβόθρα t (γιατί;)

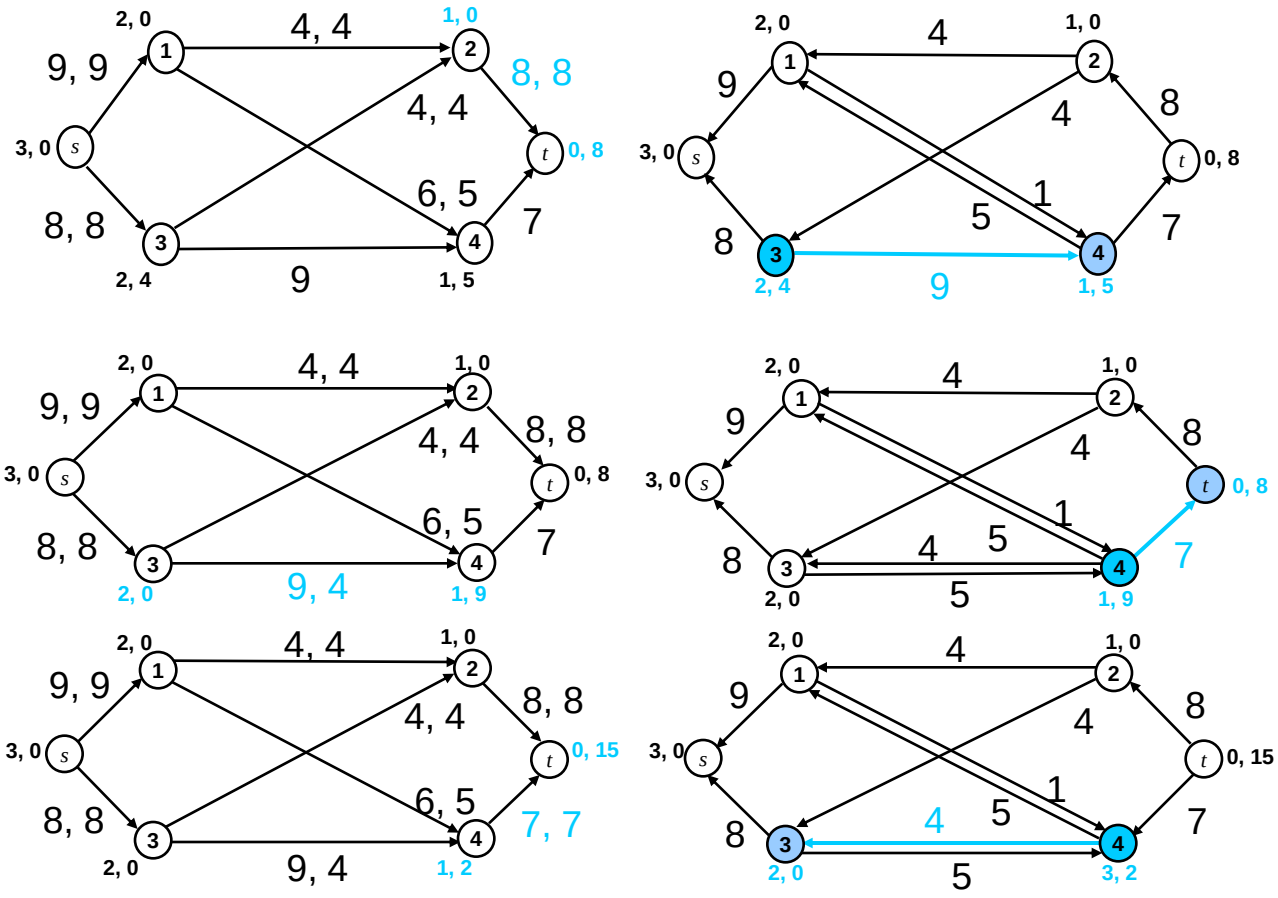
Γενικό Σχεδιάγραμμα Αλγορίθμων Προροής-Προωθήσεως

- Αναθέτουμε ύψη βάσει των συντομότερων μονοπατιών προς καταβόθρα
- Φέρουμε σε κορεσμό όλες τις ακμές της πηγής
- Όσο υπάρχουν ενεργές κορυφές, επιλέγουμε μία, έστω *ν*, και
  - είτε σπρώχνουμε ροή μέσω μίας ***νόμιμης ακμής*** της
  - είτε, εάν δεν υπάρχουν νόμιμες ακμές, υψώνουμε την *ν*, ώστε να είναι κατά ***μία μονάδα υψηλότερη*** από τον ***χαμηλότερό της γείτονα***

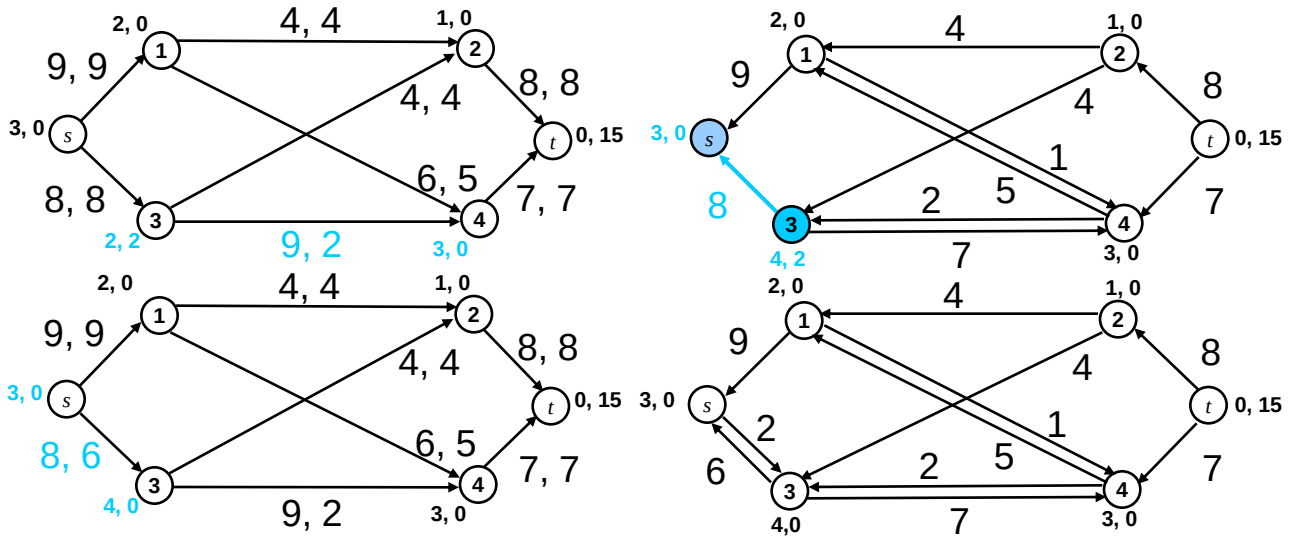
Παράδειγμα



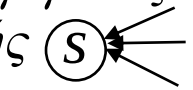
Παράδειγμα (συν.)



Παράδειγμα (συν.)



Γιατί δουλεύει;

- Μηχανικό ανάλογο
- Στο υπολειμματικό γράφημα υπάρχει μονοπάτι από κάθε ενεργή κορυφή προς την πηγή, ενώ η πηγή είναι απομονωμένη από την καταβόθρα
  - *Πολύ εύκολα, με επαγωγή, ξεκινώντας από το γεγονός πως, αρχικά, οδηγούνται σε κορεσμό αυτές της πηγής* 
  - *Οι μόνες που «βλέπει» η πηγή, έχουν ύψος* $> h(s)$ *(αφού της επιστρέφουν ροή)* $\Rightarrow$ *δεν επικοινωνούν με την καταβόθρα (ιδιότητα ύψους* $h(v) \leq h(u) + 1$, $(v,u) \in E_r$, *σταθερό 0)*
- Καμμία κορυφή δεν αποκτά ύψος μεγαλύτερο του $2V$
  - *Η πηγή έχει ύψος* $\leq V$, *ενώ κάθε ενεργή κορυφή* v *απέχει το πολύ* V-2 *ακμές από την* s: $v \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow s$

    $h(v) \leq h(v_1) + 1 \leq h(v_2) + 2 \leq \cdots \leq h(s) + d \leq V + d \leq 2V\text{-}2$
  - *Όταν η* v *καταστεί ανενεργή, το ύψος της είτε είναι το ίδιο είτε κατά ένα μεγαλύτερο από τον χαμηλότερο γείτονά της.*

Γιατί δουλεύει; (συν.)

- Όταν τερματίζει:
  - δεν υπάρχουν ενεργές κορυφές
  - η πηγή είναι αποκομμένη από την καταβόθρα
- Άρα
  - δημιουργείται κανονική ροή
  - δεν υπάρχουν επαυξητικά μονοπάτια

Ανάλυση

- Μέσω Συναρτήσεων Δυναμικού (σχεδιάγραμμα)
  - $\mathbf{\Phi} = \sum h(v)$, v ενεργή
    - αρχική τιμή $\leq V \times 2V$
    - τελική τιμή 0
  - Πώς μεταβάλλεται το δυναμικό
    - Α΄ περίπτωση: ύψωση κορυφής κατά $c \Rightarrow \Delta\mathbf{\Phi} = c$. Συνολικά, $2V^2$ ($V \times 2V$)
    - Β΄ περίπτωση: διοχέτευση ροής.
      - Κορεσμός ακμής:
        $\Delta\mathbf{\Phi} = \mathbf{2}V \Rightarrow 2V^2\ E$ (τυχόν ενεργοποίηση κορυφής απολήξεως)
      - Ακόρεστη ακμή:
        $\Delta\mathbf{\Phi} = -h(v)+h(v)-1 = -1$ (γίνεται ενεργή γειτονική w, με $h(w)= h(v)-1$) ή
        $\Delta\mathbf{\Phi} = -h(v)$ (ήδη ενεργή η γειτονική w)

Ανάλυση (συν.)

- Σύνοψη
  - Αρχικά:

    $\mathbf{\Phi} = 2V^2$
  - Μεταβολές:

    $+(2V^2+2\,V^2E)$ από αλλαγές ύψους/κορεσμούς ακμών
  - Τελικά:

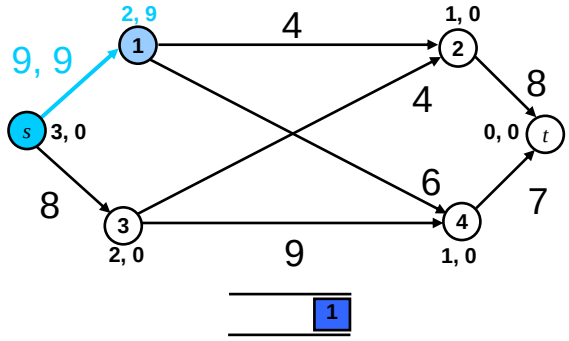
    $\mathbf{\Phi} = 0 \Rightarrow 2V^2+2\,V^2E = O(V^2E)$ ακόρεστες προωθήσεις

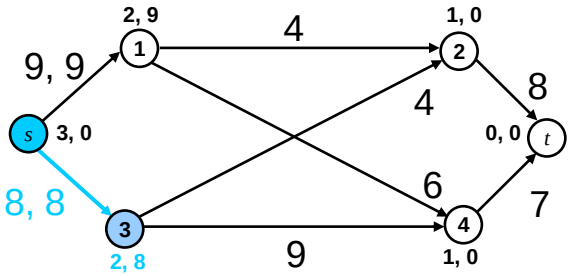
    (έκαστη προκαλεί $\Delta\mathbf{\Phi} \leq -1$)
- Χρόνος $O(V^2E)$

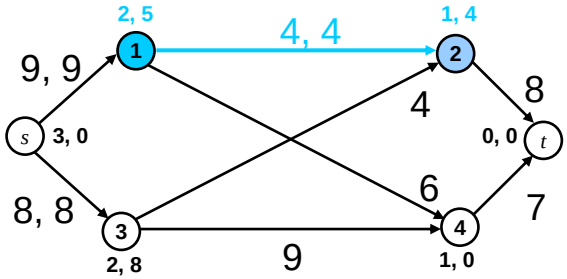
Τροποποίηση Goldberg-Tarjan

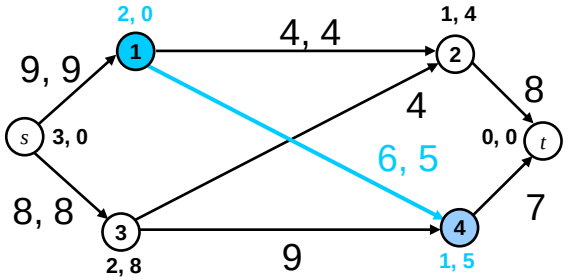
- Επιλογή ενεργής κορυφής για προώθηση ροής μέσω ουράς FiFo
- Όταν επιλεγεί ενεργή κορυφή, προωθείται ***συνεχώς*** ροή μέσω νομίμων ακμών μέχρι
  - είτε να καταστεί ανενεργή
  - είτε να τελειώσουν οι νόμιμες ακμές, οπότε προσαρμόζεται το ύψος της

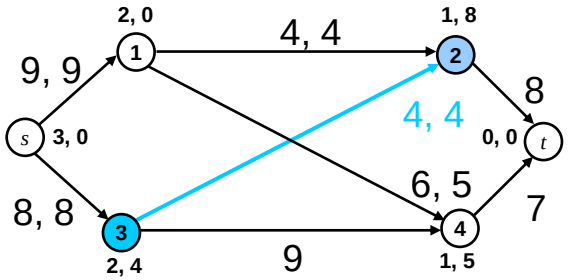
Παράδειγμα

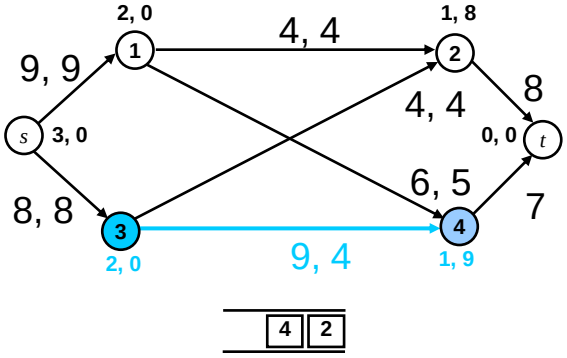

2, 9
1, 0
9, 9
4
4
8
3, 0
0, 0
6
8
7
2, 0
9
1, 0


2, 9
1, 0
9, 9
4
4
8
3, 0
0, 0
8, 8
6
7
2, 8
9
1, 0

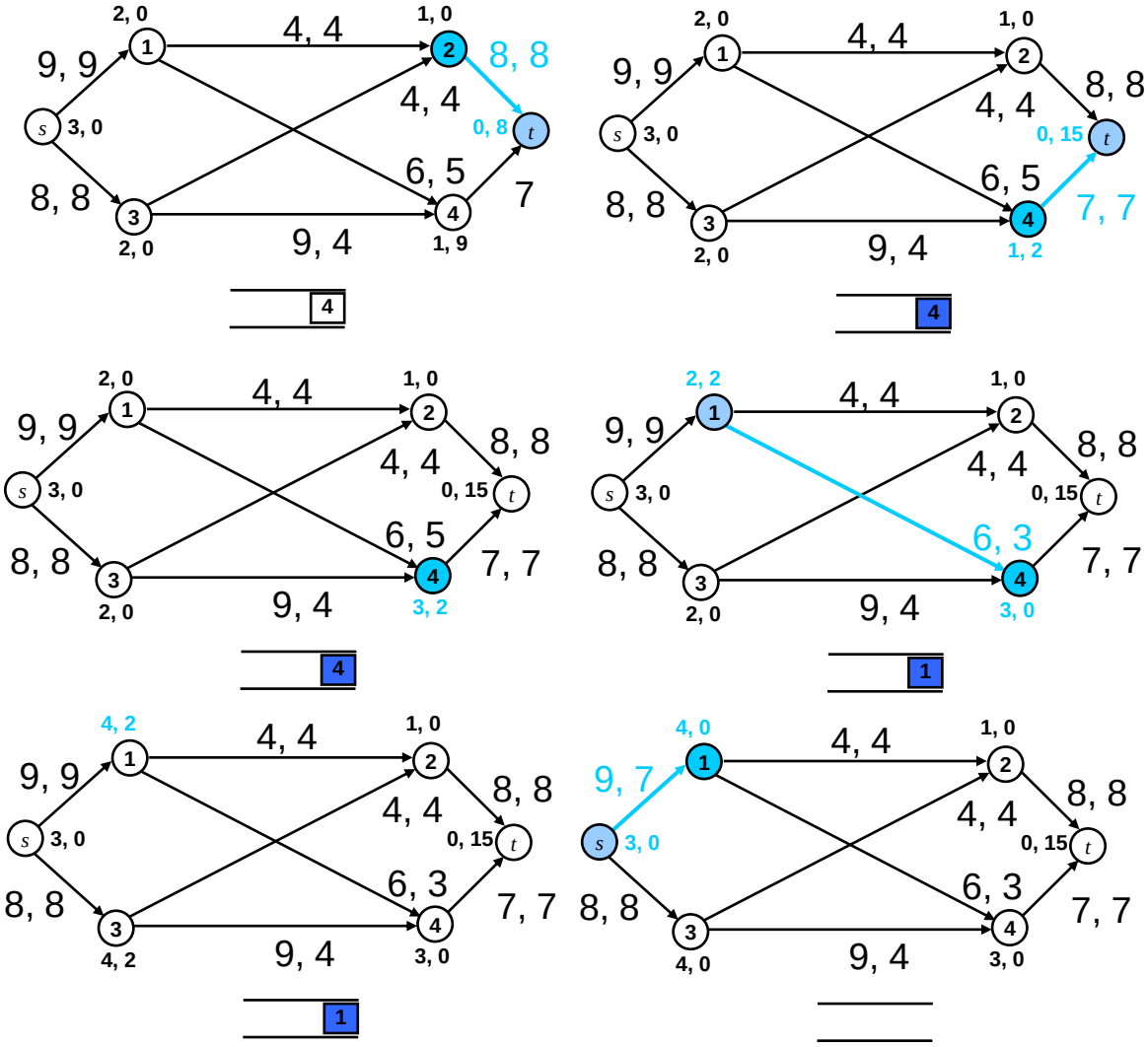

2, 5
4, 4
1, 4
9, 9
4
8
3, 0
0, 0
6
8, 8
7
2, 8
9
1, 0


2, 0
4, 4
1, 4
9, 9
4
8
3, 0
0, 0
6, 5
8, 8
7
2, 8
9
1, 5


2, 0
4, 4
1, 8
9, 9
4, 4
8
3, 0
0, 0
6, 5
8, 8
7
2, 4
9
1, 5

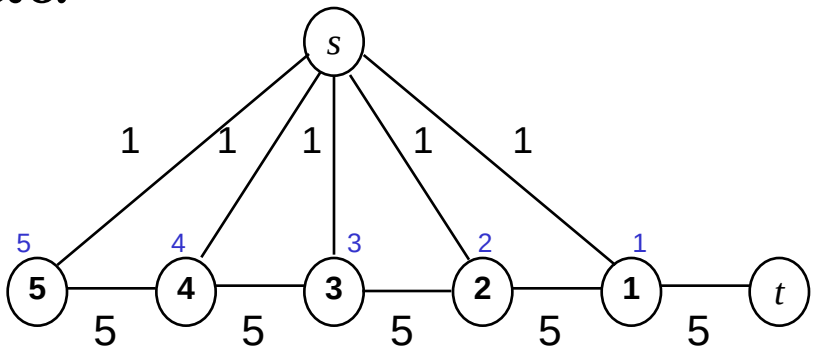

2, 0
4, 4
1, 8
9, 9
4, 4
8
3, 0
0, 0
6, 5
8, 8
7
2, 0
9, 4
1, 9

Παράδειγμα (συν.)



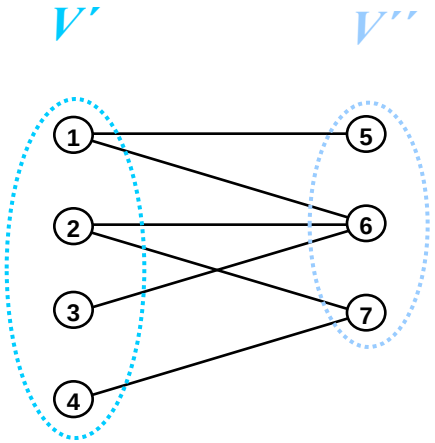
Ανάλυση

- Πολυπλοκότητα $O(V^2E)$
  - Αποδεικνύεται με συνάρτηση δυναμικού
- Το όριο είναι σφικτό, όπως δείχνει και το κάτωθι σχήμα



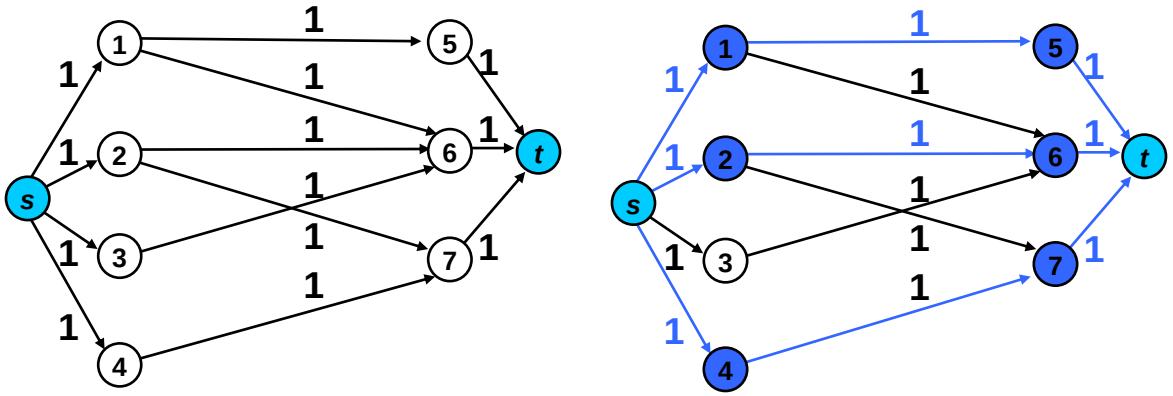
Μέγιστα Ταιριάσματα σε Διμελή Γραφήματα

- Ορισμοί
  - Κάθε σύνολο ακμών με διαζευγμένα άκρα
  - Μέγιστο, εάν δεν υπάρχει άλλο πολυπληθέστερο που να το περιέχει



Επίλυση

- Αναγωγή σε πρόβλημα ροής
  - Προσθήκη πηγής *s*
  - Προσθήκη καταβόθρας *t*
  - Μοναδιαία βάρη
- Χρόνος $O((V'+V'')E)$, καθώς
  - το μέγιστο ταίριασμα έχει $(V'+V'')/2$ ακμές, ενώ
  - όλες οι χωρητικότητες ισούνται με 1 $\Rightarrow$ μέγιστη ροή $(V'+V'')/2$



Ροές Ελαχίστου Κόστους

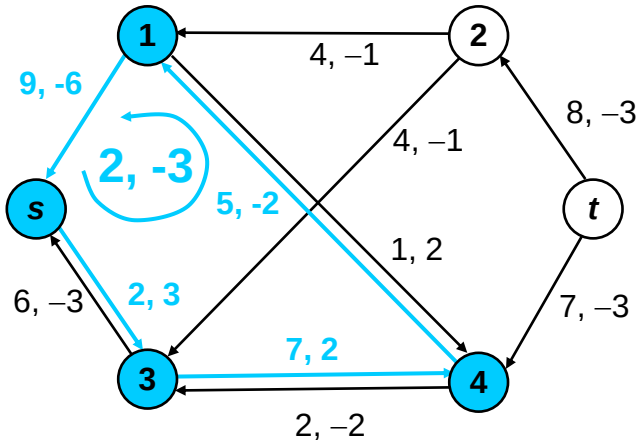
- Ορισμοί
  - Με κάθε ακμή e συσχετίζεται και μία τιμή κόστους $b(e)$ ανά μονάδα ροής
  - Κόστος μίας ροής f :

$$\Sigma f(e)b(e)$$

  - Μία ροή χαρακτηρίζεται ως ***ελαχίστου κόστους*** εάν επιδεικνύει το ***μικρότερο κόστος μεταξύ όλων των ροών της ίδιας τιμής***
  - Μέγιστη ροή ελαχίστου κόστους το ζητούμενο

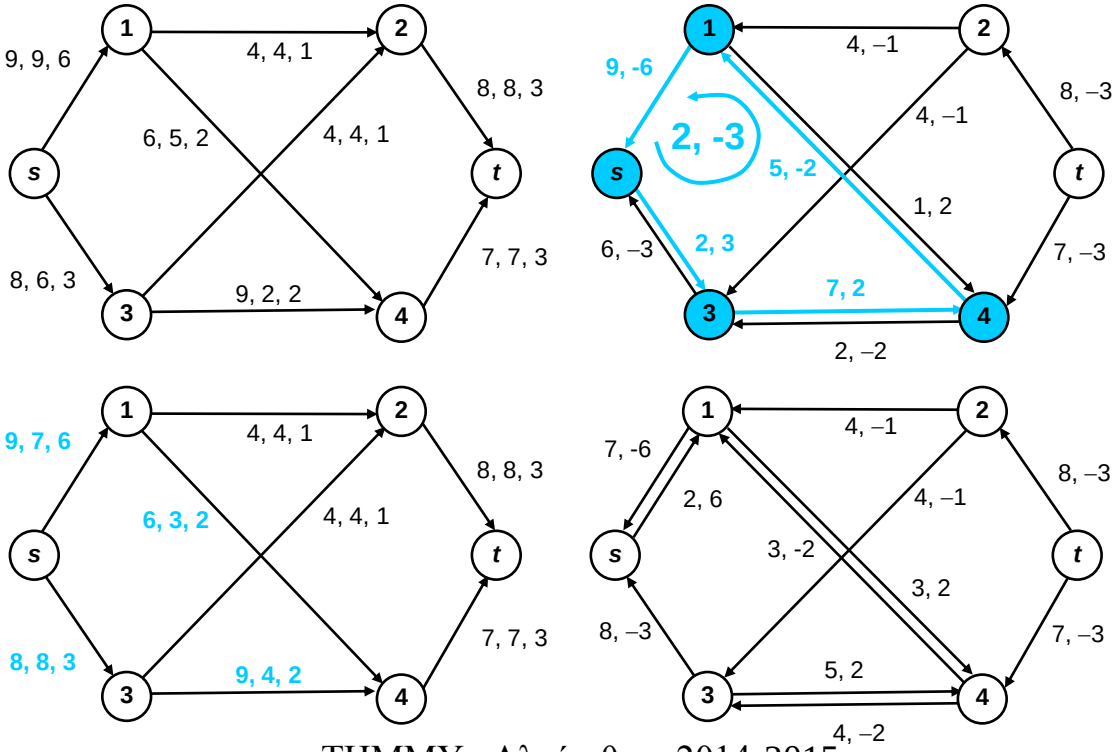
Επέκταση Υπολειμματικού Γραφήματος

- Αναγκαία η επέκταση του υπολειμματικού γραφήματος, ώστε
  - με κάθε πραγματική ακμή e να σχετίζεται κόστος $b(e)$
  - με κάθε οπισθοακμή, κόστος $-b(e)$ (καθώς τυχόν επιλογή της ελαττώνει το κόστος)



Πρώτη Λύση

- Χρήση επαυξητικών κύκλων αρνητικού κόστους
  - *Η ύπαρξή τους σημαίνει δυνατότητα μείωσης του κόστους,* ***χωρίς μεταβολή της τιμής της ροής***

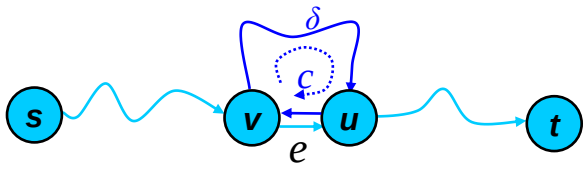


Γιατί δουλεύει;

- Μία ροή είναι ελαχίστου κόστους ανν δεν υπάρχει αρνητικός επαυξητικός κύκλος
  - ($\Rightarrow$) Εάν υπάρχει αρνητικός προφανώς μπορεί κανείς να μειώσει το κόστος
  - ($\Leftarrow$) Εάν η f είναι μέγιστη, όχι ελαχίστου κόστους, αλλά με θετικούς κύκλους, τότε επαύξηση κατά μήκος του κύκλου θα μείωνε το κόστος, δίχως να πειραχθεί η τιμή (άτοπο)
- Πολυπλοκότητα $O(BVE)$, B αρχικό κόστος μέγιστης ροής
  - Edmonds-Karp: για αρχική ροή, κόστους B
  - Bellman-Ford: για εντοπισμό B, το πολύ, αρνητικών κύκλων

Δεύτερη Λύση

- Έστω ροή f ελαχίστου κόστους και επαυξητικό μονοπάτι π ελαχίστου κόστους. Τότε η f': $f \rightarrow_{\pi} f'$ είναι ροή ελαχίστου κόστους, με μεγαλύτερη, όμως, τιμή ροής:
  - Εάν η f' δεν είναι ελαχίστου κόστους, τότε πρέπει να υπάρχει αρνητικός κύκλος c, με τουλάχιστον μία κοινή ακμή με την π (γιατί; οι δύο ροές διαφέρουν μόνο στις ακμές $\pi = f' - f$ και, άρα, σε αντίθετη περίπτωση ο κύκλος θα υπήρχε και στην f):



  - $b(s \rightarrow v)+b(\delta)+b(u \rightarrow t) < b(s \rightarrow v)+b(v \rightarrow u)+b(u \rightarrow t)$, καθώς
    $b(\delta)+ b(u \rightarrow v) < 0 \Rightarrow b(\delta)-b(v \rightarrow u) < 0 \Rightarrow b(\delta) < b(v \rightarrow u)$

Ανάλυση

- Πολυπλοκότητα
  - $O(|f|VE)$: Bellman-ford για τα ΣΜΜΠ, βάσει της b
  - $O(|f|E\log V)$: Εναλλακτικά, για αραιά γραφήματα, Johnson

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

- Σ: αλφάβητο
- Σ*: όλες οι συμβολοσειρές πεπερασμένου μήκους, συμπεριλαμβανομένης της κενής $\in$
- $|x|$ μήκος συμβολοσειράς x
- $x=\alpha\beta$, $y=\gamma\delta\varepsilon$ συμβολοσειρές
  $xy=\alpha\beta\gamma\delta\varepsilon$ συμβολοσειρά συνενώσεώς τους
- $x=\alpha\beta\gamma\delta\varepsilon\zeta$
  - *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)

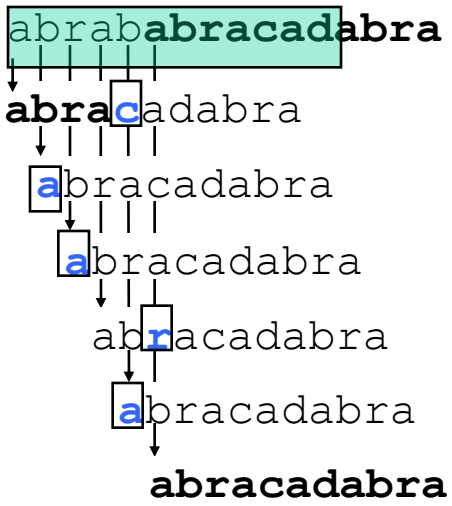
- *«Δοθεισών ενός κειμένου Τ, μήκους n, και ενός προτύπου Ρ, μήκους m, απαιτείται ο εντοπισμός όλων των στιγμιοτύπων του Ρ στο Τ.»*
  - Π.χ. για Τ=ααβαββαββββα και Ρ=βα, έχουμε

    Aα**βα**β**βα**βββ**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 = `aaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

    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

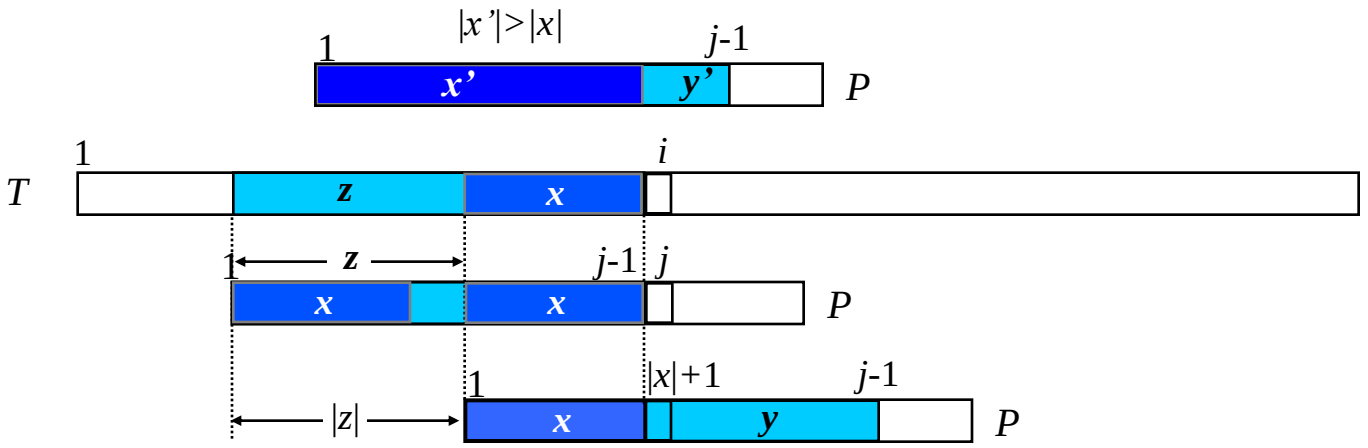
Πίνακας Αποτυχίας f

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

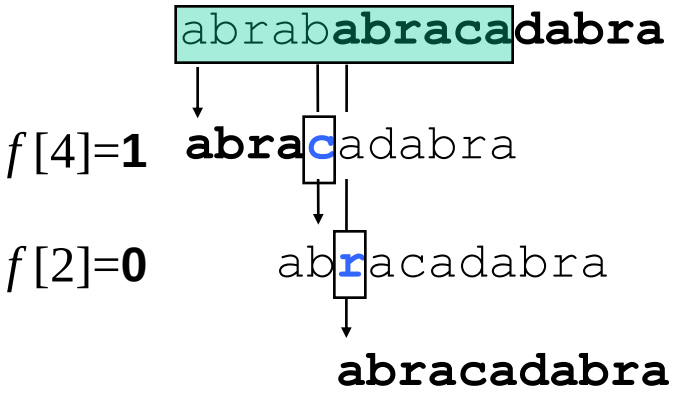
1	2	3	4	5	6	7	8	9	10	11
a										
0										

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!

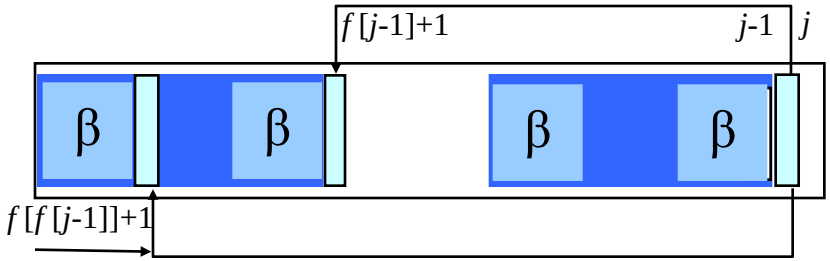


Παράδειγμα


abrababracadabra
f [4]=1
abracadabra
f [2]=0
abracadabra
abracadabra

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

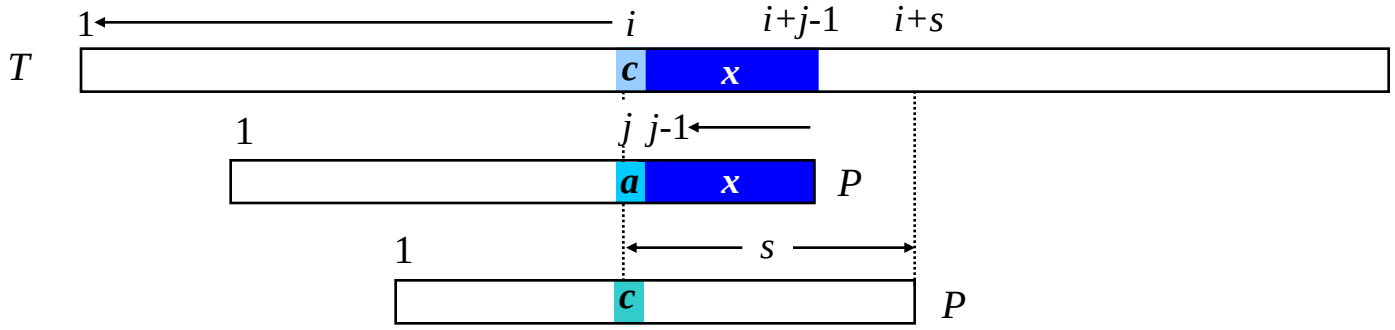
- Ο υπολογισμός της f γίνεται σε χρόνο $O(m)$
  - Η γραμμή 24 του kmpPM καθορίζει την πολυπλοκότητα
- Οι συγκρίσεις είναι μεταξύ n και $2n$ διότι κάθε φορά
  - εάν η σύγκριση είναι επιτυχής, μεταβαίνουμε στον επόμενο χαρακτήρα του κειμένου και ο αντίστοιχος δείκτης i ποτέ δεν μειούται
  - εάν είναι ανεπιτυχής, 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(Σ)



Παράδειγμα

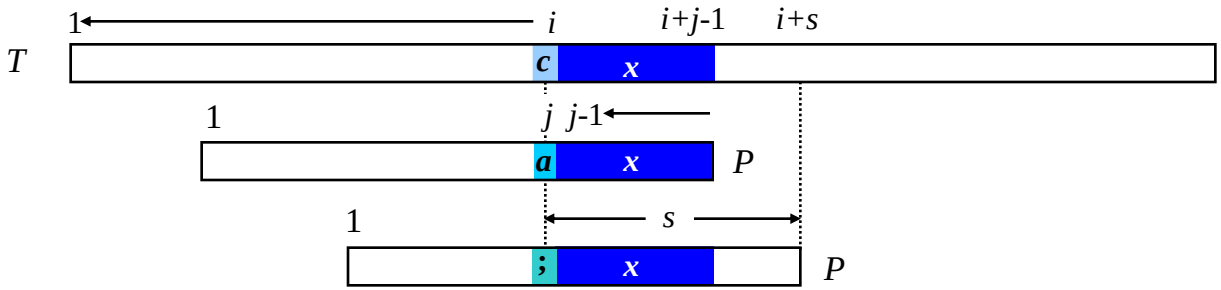
d’	a	b	c	d	r
	11	**9**	**5**	**7**	**10**

d= m-d’	a	b	c	d	r
	0	**2**	**6**	**4**	**1**

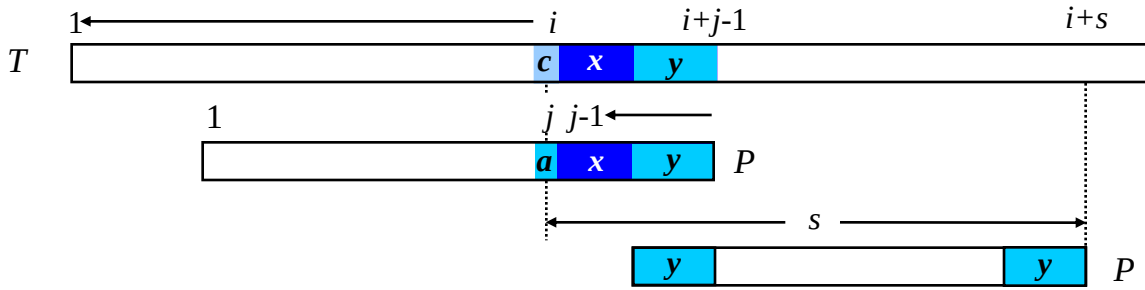
1	2	3	4	5	6	7	8	9	10	11
a	b	r	a	**c**	a	d	a	b	r	a

Ευρετικό Ταιριάσματος (Match Heuristic)

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



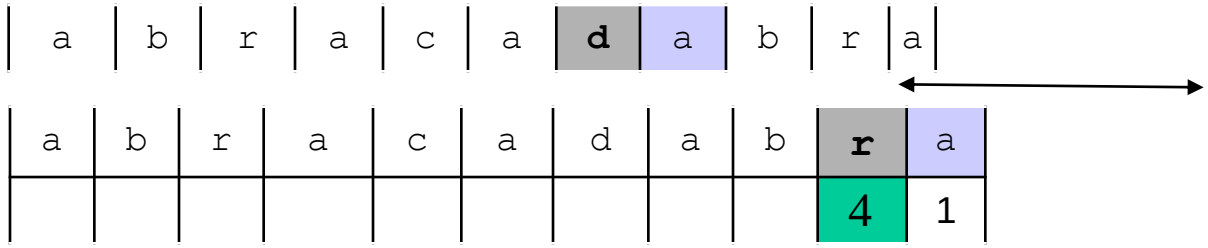
(α)

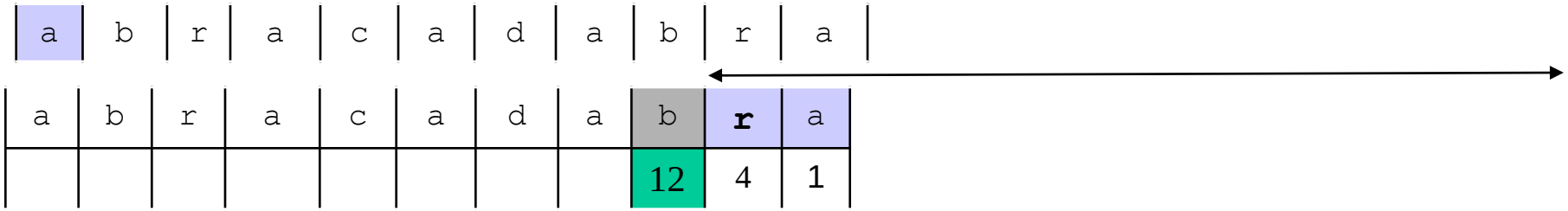


(β)

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!



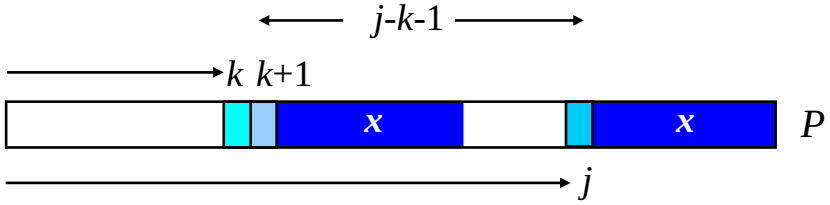


Ευρετικό Ταιριάσματος (συν.)

Τελικώς…

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

Μία Εξήγηση των Υπολογισμών

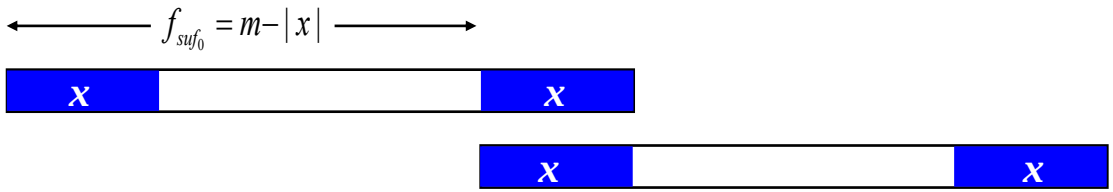


(α)

Έστω ότι υπολογίζουμε «ανάποδα» τον πίνακα ΚΜΡ, ώστε

- Κάθε θέση $k+1$ μας δίδει 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	**3**	**1**
match		**7**	**7**	**7**	**7**	**7**	**7**	**7**	**10**	**10**	3	1
+		10	9	8	7	6	5	4	3	2	1	0
match		**17**	**16**	**15**	**14**	**13**	**12**	**11**	**13**	**12**	**4**	**1**

- Η πολυπλοκότητα είναι ανάλογη του KMP…

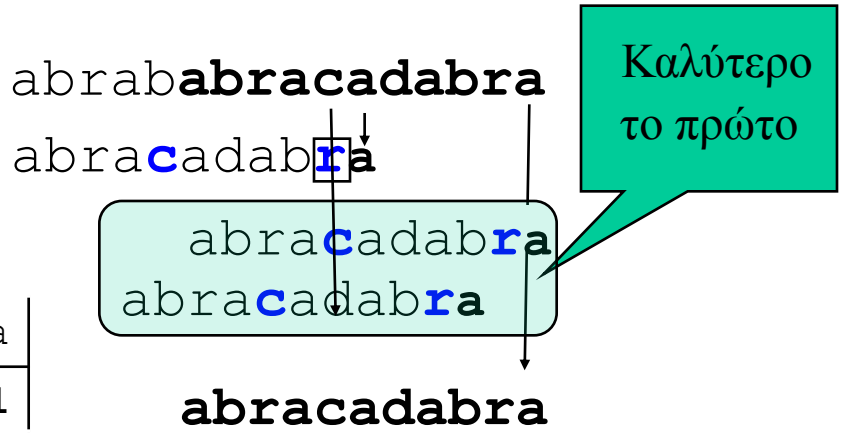
Παράδειγμα Τρεξίματος

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



- 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=αβγδ**2**τδ**λ**

  x'=**β2λ**, με i_1=2 < i_2=5 < i_3=8

  – Παρατηρήστε πως σεβόμαστε την από αριστερά προς τα δεξιά διάταξη των χαρακτήρων
  – Έτσι, η βαδ ***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

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

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:

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

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:

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

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

		0	1 (Α)	2 (Β)	3 (Ε)	4 (Λ)
	0	0	←0	←0	←0	←0
Β	1	↑0	↑0	↖1	←1	←1
Α	2	↑0	↖1	←1	←1	←1
Β	3	↑0	↑1	↖2	←2	←2
Ε	4	↑0	↑1	↑2	↖3	←3
Λ	5	↑0	↑1	↑2	↑3	↖4

Ελάχιστη Απόσταση Διαμορφώσεως

- Ορισμοί
  - Πράξεις διαμορφώσεως:
    - delete(*c*): διαγραφή του χαρακτήρα *c*
    - insert(*c*): εισαγωγή του χαρακτήρα *c*
    - 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

  λεί**ε**πει ⇒ λίπ**ε**ι ⇒ λίπ**ι** ⇒ λίπη

  Διαγραφή Διαγραφή Αντικατάσταση

- 4 πράξεις

  λεί**ε**πει ⇒ λί**ί**πει ⇒ λίπ**ε**ι ⇒ λίπη**ι** ⇒λίπη

  Αντικατάσταση Διαγραφή Αντικατάσταση Διαγραφή

Παρατήρηση

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

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

Παράδειγμα

	–	Λ	Ι	Π	Η
–	0	1	2	3	4
Λ	1	0=	←1Ε	←2Ε	←3Ε
Ε	2	↑1Δ	↖1Α	↖2Α	↖3Α
Ι	3	↑2Δ	↖1=	↖2Α	↖3Α
Π	4	↑3Δ	↑ 2 Δ	↖1=	← 2Ε
Ω	5	↑4Δ	↑ 3 Δ	↑ 2Δ	↖2Α

Ψευδοκώδικας

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

```
1.  n = x.length; m = y.length;
2.  for (i = 0; i <= n; i++)        // # O(n)
3.     c[i][0] = i;
4.  for (i = 0; i <= m; i++)        // # O(m)
5.      c[0][i] = i;
6.  for (i = 1; i <= n; i++)        // # O(nm)
7.     for (j = 1; j <= 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));
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 $O(n^{\varepsilon})$ στο πλήθος n των λέξεων (νόμος Heaps)

Παράδειγμα

1

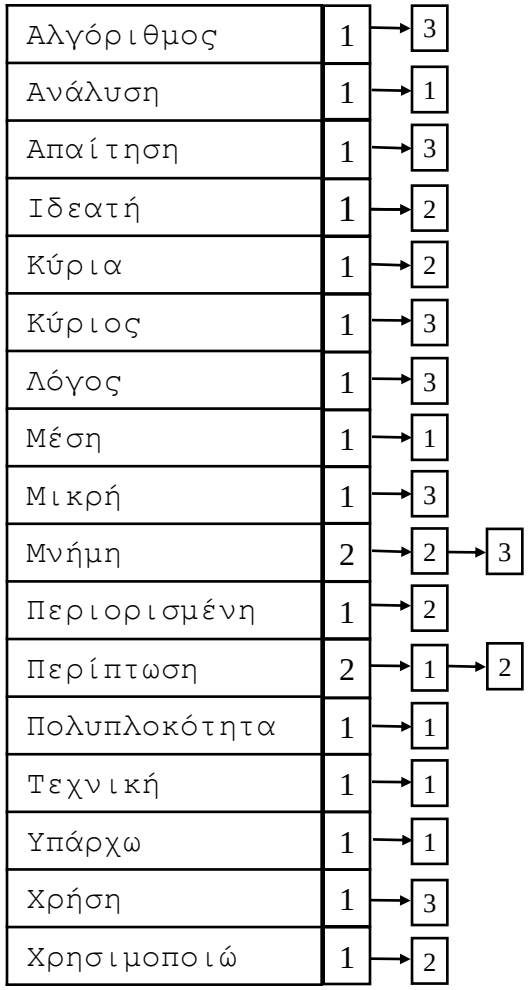
Υπάρχουν δύο τεχνικές ανάλυσης πολυπλοκότητας: η ανάλυση χειρότερης περιπτώσεως και η ανάλυση μέσης περιπτώσεως.

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.

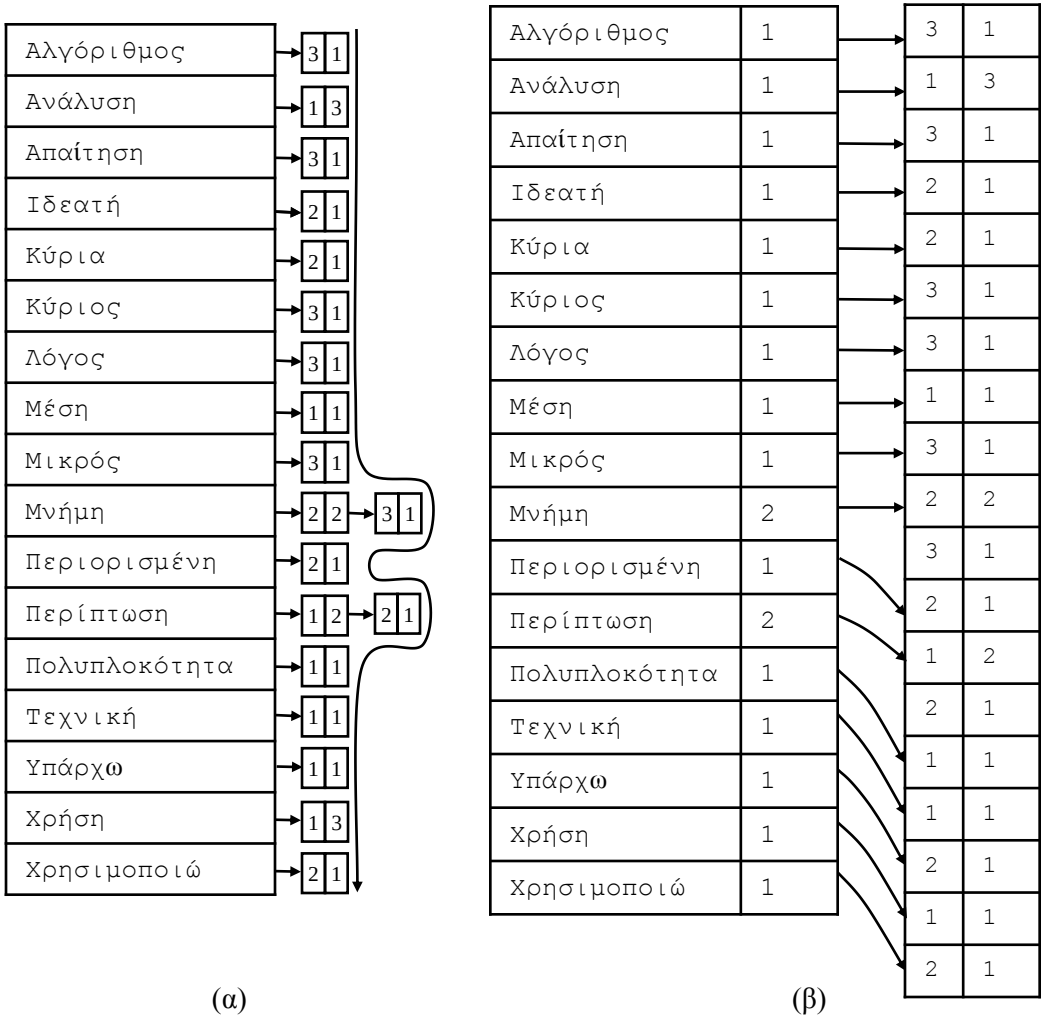


Πώς χρησιμεύει

- 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

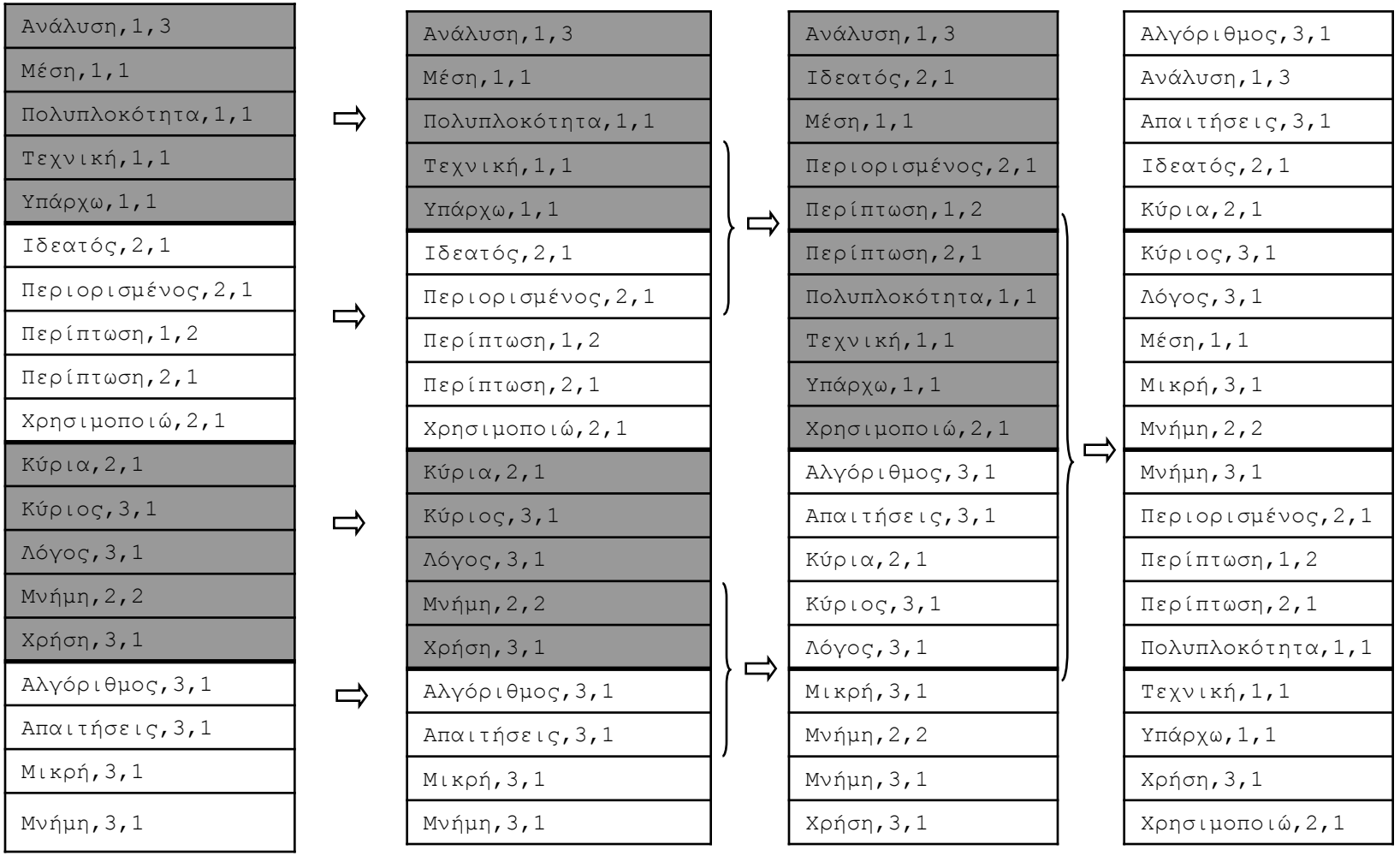
Δημιουργία

- 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

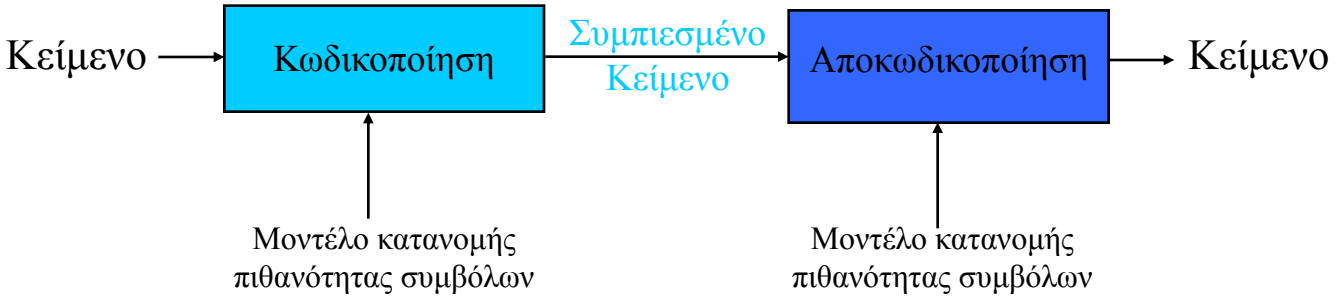
  (α) να καταλαμβάνει μικρότερο χώρο ή να χρειάζεται λιγότερο χρόνο μεταδόσεως και

  (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 Κωδικοποιήσεως - Αποκωδικοποιήσεως



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

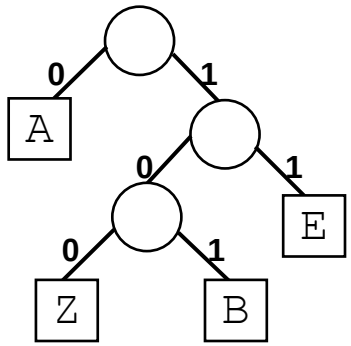
Μέθοδοι Κωδικοποίησης

- **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;

$$E(\textit{κειμένου } K) = \sum_{x\in K}\mathbf{Prob}[x]\text{len}(x)$$

$$= |K| \sum_{c\in C}\mathbf{Prob}[c]\text{len}(c),$$

όπου C *το σύνολο των διακεκριμένων χαρακτήρων*

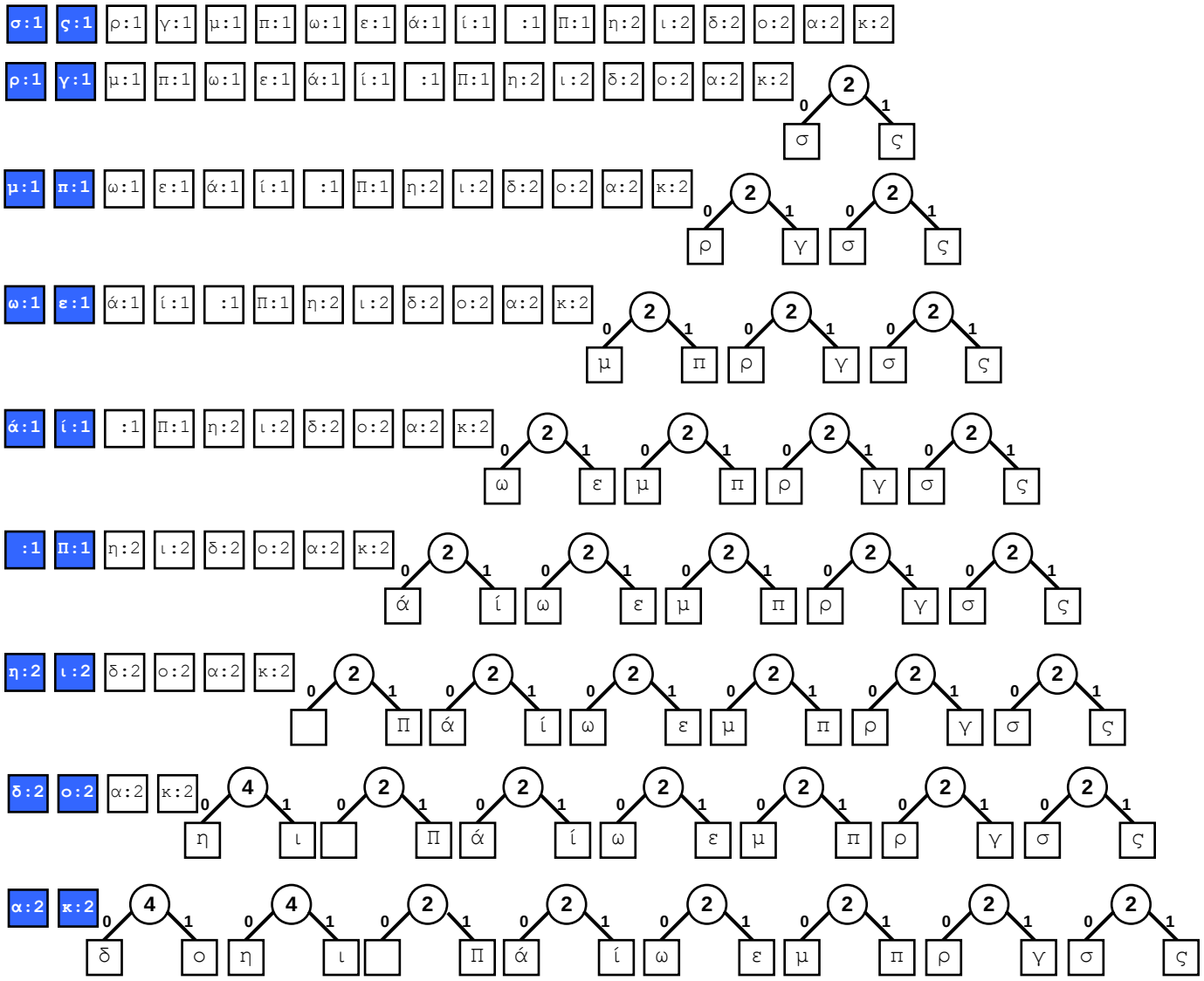
Κεντρική Ιδέα

- Στους χαρακτήρες που έχουν μεγάλη συχνότητα εμφανίσεως (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

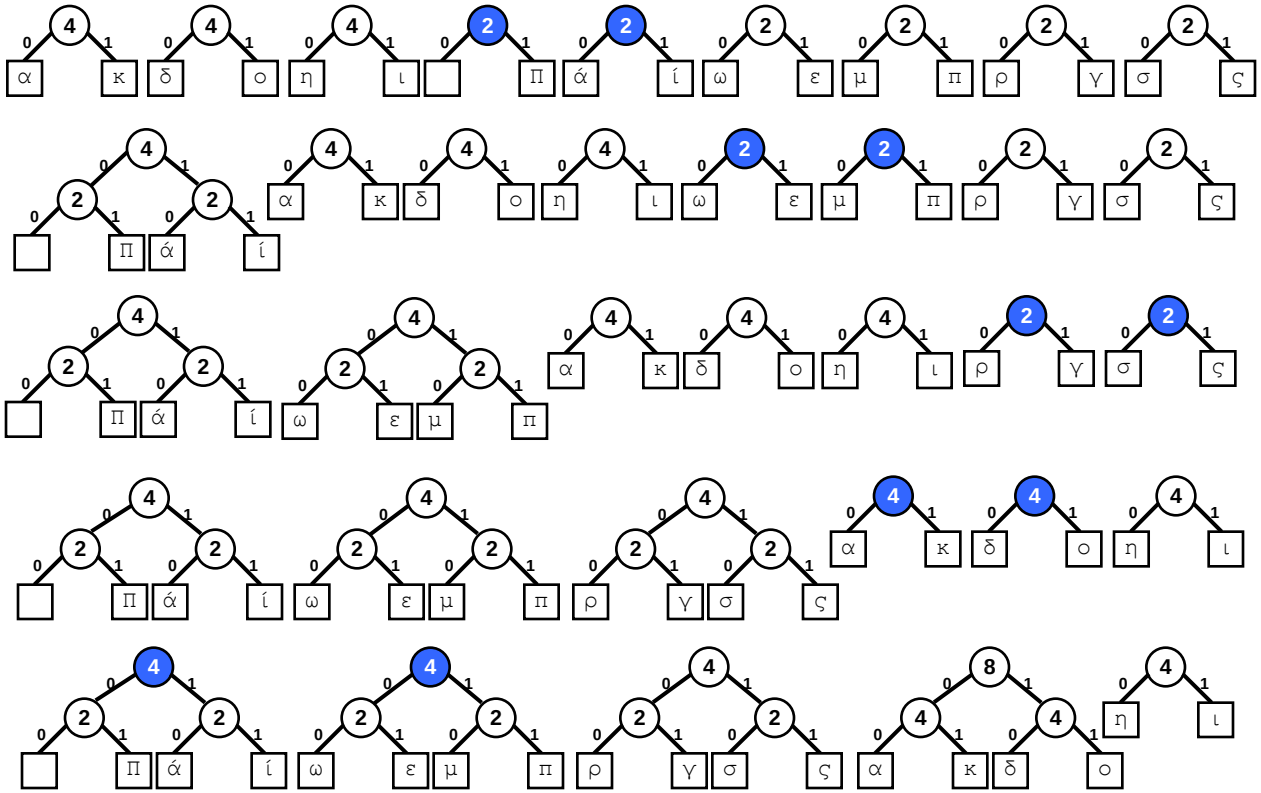
Περιγραφή

- 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

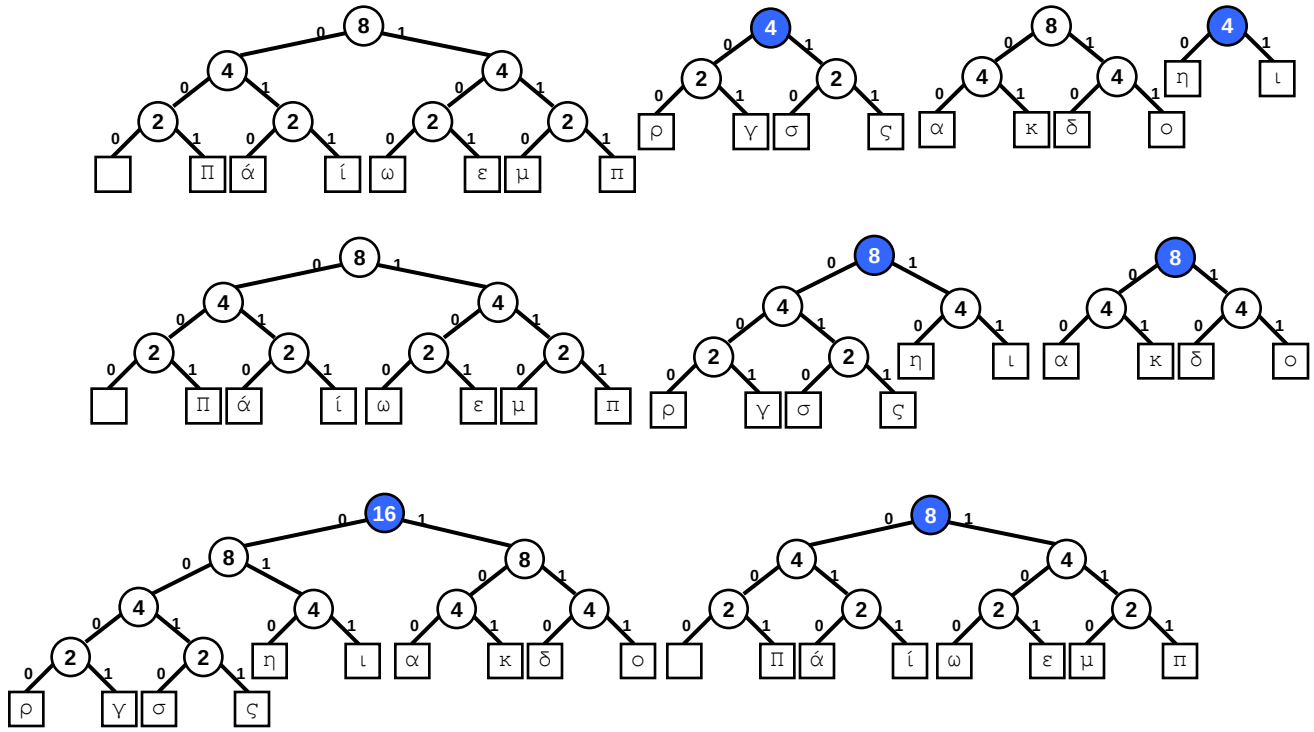
Παράδειγμα


σ:1 ς:1 ρ:1 γ:1 μ:1 π:1 ω:1 ε:1 ά:1 ί:1 :1 Π:1 η:2 ι:2 δ:2 ο:2 α:2 κ:2
ρ:1 γ:1 μ:1 π:1 ω:1 ε:1 ά:1 ί:1 :1 Π:1 η:2 ι:2 δ:2 ο:2 α:2 κ:2
μ:1 π:1 ω:1 ε:1 ά:1 ί:1 :1 Π:1 η:2 ι:2 δ:2 ο:2 α:2 κ:2
ω:1 ε:1 ά:1 ί:1 :1 Π:1 η:2 ι:2 δ:2 ο:2 α:2 κ:2
ά:1 ί:1 :1 Π:1 η:2 ι:2 δ:2 ο:2 α:2 κ:2
:1 Π:1 η:2 ι:2 δ:2 ο:2 α:2 κ:2
η:2 ι:2 δ:2 ο:2 α:2 κ:2
δ:2 ο:2 α:2 κ:2
α:2 κ:2

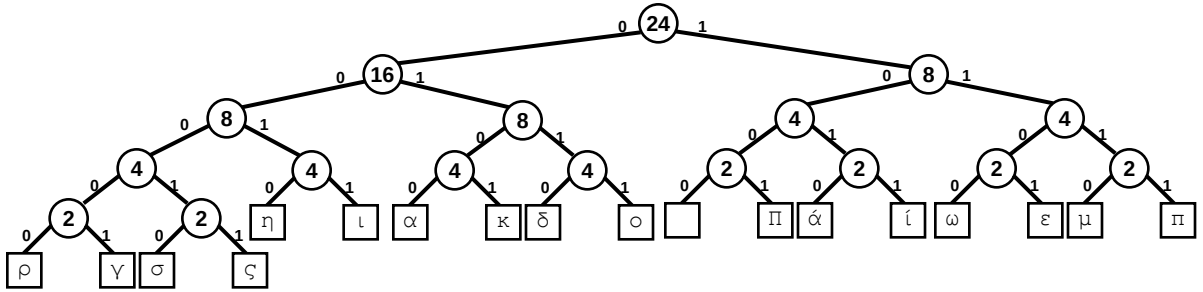
Παράδειγμα (συν.)



Παράδειγμα (συν.)



Παράδειγμα (συν.)



Παράδειγμα κωδικοποιήσεως

ASCII

1101000**11100001**1111000**11011100**11100100
111001011110100**11100011**1110110**11100001**
1010000**11101010**1111001**11100101**11101001
111010101110111**11110000**1110111**11011111**
1110011**11110011**1110011**11110010**

Huffman

1001**0100**00000**1010**0110**1101**0011**00001**111**01**
001000**0101**1100**0110**0011**0101**0111**1111**011**10**
110010**00010**0010**00011**

Υλοποίηση

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

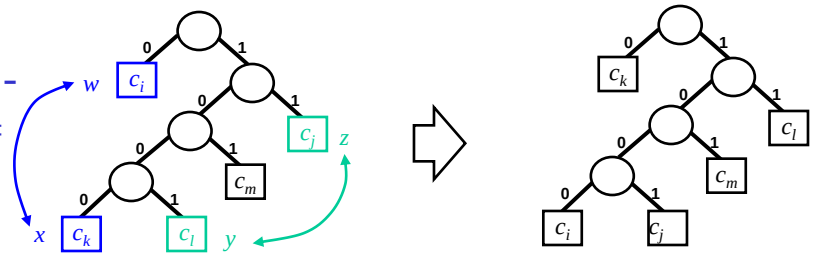
```
1.  pqueue pq = new pqueue();      //ουρά προτεραιότητας
2.  for (i = 0; i < C.length; i++)    // αρχικοποίηση με τα φύλλα – χρόνος O(nlogn)
3.     pq.insKey(new bNode(new Item(C[i],F[i]), F[i]); // χρόνος O(logn)
4.  for (i = 1; i< C.length; i++){   // χρόνος O(nlogn)
5.     bNode t = new bNode();       // νέος εσωτερικός κόμβος
6.      x = pq.delMin();            // χρόνος O(logn)
7.      y = pq.delMin();            // χρόνος O(logn)
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.    qp.insKey(t, t.key);          // και συχνότητα το άθροισμά τους – χρόνος O(logn)
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 αδέλφια

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



$-\{\mathbf{Prob}[c_l](d_y-d_z)- \mathbf{Prob}[c_j](d_y-d_z)\} \leq 0$

  - 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)