Προβλήματα ελαχίστων μονοπατιών από κοινή αφετηρία

(Single-source shortest path problems)

Αφορμή για την εξέτασή τους

- Έχουν μια άπληστη λύση (Αλγόριθμος του Dijkstra).
- Βρίσκουν εφαρμογή σε πολλά πραγματικά προβλήματα.

Παραδείγματα

- Με δεδομένο έναν οδικό χάρτη της Βόρειας Αμερικής (ή οποιασδήποτε άλλης περιοχής) να βρεθούν οι αποστάσεις από τη Νέα Υόρκη προς όλες τις άλλες πόλεις του χάρτη.

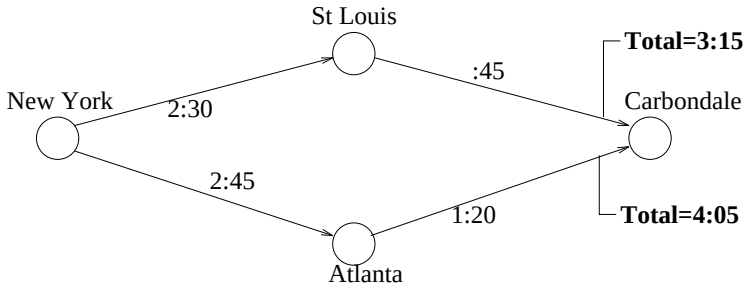
  Αναπαριστούμε τις πόλεις σαν κόμβους,
  τους δρόμους σαν ακμές, και
  τις αποστάσεις μεταξύ πόλεων σαν βάρη/ετικέτες των ακμών.

  Άρα, αρκεί να βρεθούν τα ελάχιστα μονοπάτια προς όλους τους κόμβους από τον κόμβο που αναπαριστά τη Νέα Υόρκη.

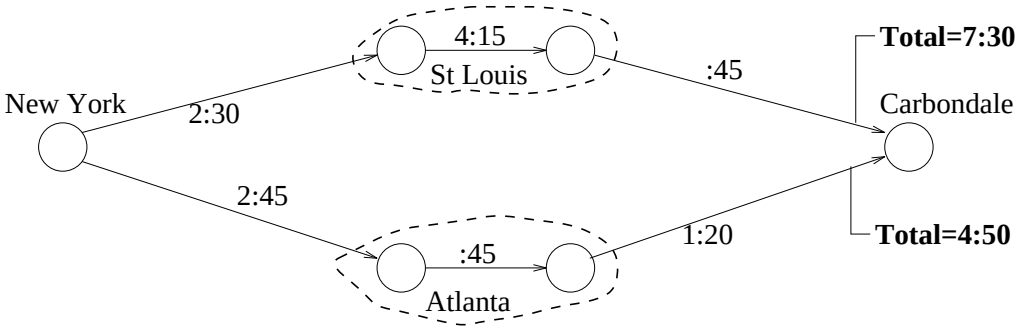
- Με δεδομένο το πρόγραμμα πτήσεων και τη διάρκεια κάθε πτήσης, να βρεθεί ο ταχύτερος τρόπος για να ταξιδεύσει κάποιος από τη Νέα Υόρκη στο Carbondale της πολιτείας Illinois.

Λύση

Να αναπαρασταθούν οι πόλεις σαν κόμβοι, οι πτήσεις σαν ακμές, και η διάρκεια-πτήσεων σαν βάρη ακμών.



- Πως θα αντιμετωπίσουμε το πρόβλημα εάν θέλουμε να ληφθεί υπ΄ όψιν και ο χρόνος αναμονής μεταξύ των πτήσεων·



Προβλήματα ελαχίστων μονοπατιών από κοινή αφετηρία

Είσοδος:
- Ένα κατευθυνόμενο, με βάρη, γράφημα $G = (V, E)$ και μία συνάρτηση βάρους $w : E \to \mathcal{R}$ που αντιστοιχεί σε κάθε ακμή ένα πραγματικό αριθμό (βάρος).
- Ένας κόμβος αφετηρία s.

Έξοδος: Τα ελάχιστα μονοπάτια από το s προς κάθε άλλο κόμβο του G.

Ορισμοί

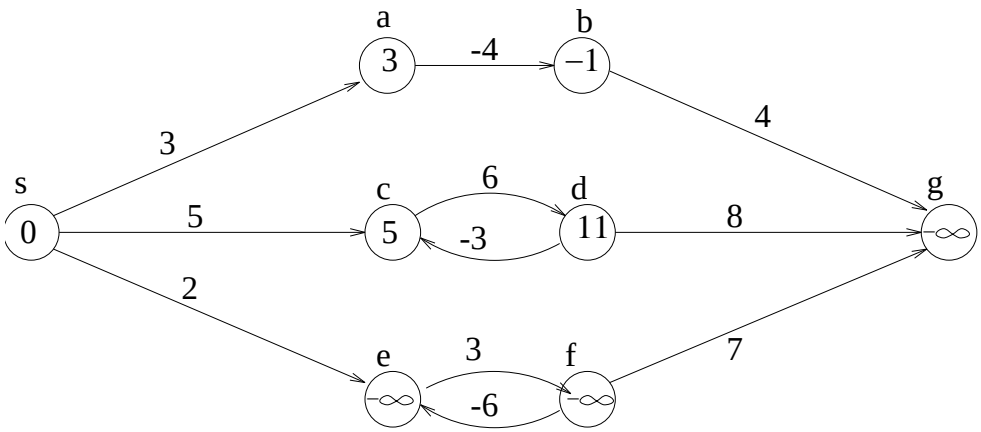
Το *βάρος του μονοπατιού* $p = (v_0, v_1, \cdots, v_k)$ είναι ίσο με το άθροισμα των βαρών των ακμών που αποτελούν το μονοπάτι.

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Το *μονοπάτι ελάχιστου βάρους* από τον u προς τον v ορίζεται σαν:

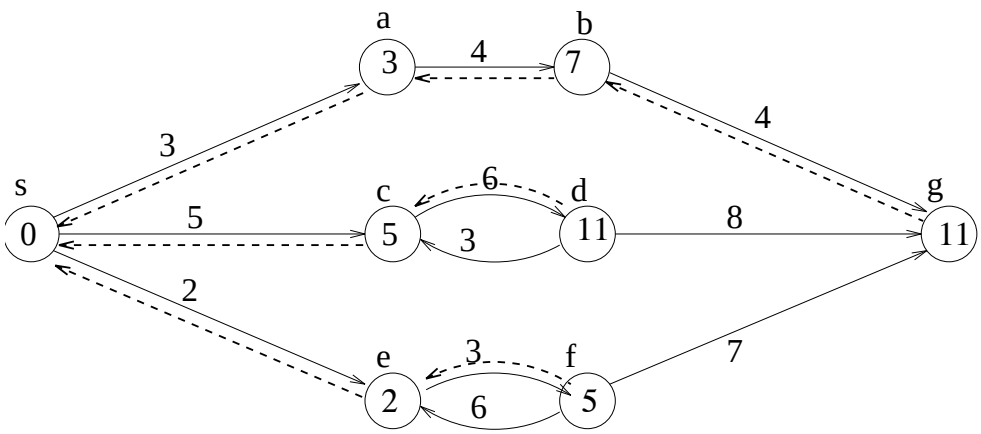
$$\delta(u,v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\} & \text{εάν υπάρχει μονοπάτι από τον } u \text{ προς τον } v \\ +\infty & \text{σε κάθε άλλη περίπτωση} \end{cases}$$

Αρνητικά βάρη ακμών



Κύκλοι που περιέχουν ακμές με αρνητικά βάρη:
- Κόστος κύκλου ≥ 0.
- Κόστος κύκλου < 0.

Πως αναπαριστάμε τα ελάχιστα μονοπάτια

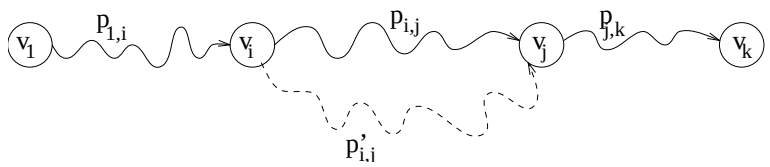


- Στη συνέχεια, θα επικεντρώσουμε τη προσοχή μας μόνο στην εύρεση του κόστους των ελαχίστων μονοπατιών.
- Τα μονοπάτια μπορεί να ανακτηθούν με τον ίδιο ακριβώς αλγόριθμο που χρησιμοποιήθηκε για την ανάκτηση των μονοπατιών από ένα δένδρο κατά-πλάτους αναζήτησης.

Βέλτιστη δομή ενός ελαχίστου μονοπατιού

Λήμμα (*Τα υπό-μονοπάτια ενός ελαχίστου μονοπατιού είναι ελάχιστα μονοπάτια*) Υποθέστε ένα κατευθυνόμενο, με βάρη, γράφημα $G = (V, E)$ και μία συνάρτηση βάρους w, και έστω $p = (v_1, v_2, \cdots, v_k)$ είναι ένα ελάχιστο μονοπάτι από τον v_1 προς τον v_k. Για $1 \leq i \leq j \leq k$, έστω $p_{i,j} = (v_i, \cdots, v_j)$ το υπό-μονοπάτι του p από τον v_i προς τον v_j. Τότε, το $p_{i,j}$ είναι ένα ελάχιστο μονοπάτι.

Απόδειξη (Με εις άτοπο επαγωγή)



- $w(p) = w(p_{1,i}) + w(p_{i,j}) + w(p_{j,k})$ και το $w(p)$ είναι ελάχιστο.
- Υποθέστε ότι το $p_{i,j}$ δεν είναι ελάχιστο. Έστω ότι το $p'_{i,j}$ είναι ένα ελάχιστο μονοπάτι.
- $w(p'_{i,j}) < w(p_{i,j})$.
- Τότε, το p δεν είναι ελάχιστο μονοπάτι. Μια καθαρή αντίφαση.
  (Το μονοπάτι $p' : \ p_{1,i} - p'_{i,j} - p_{j,k}$ έχει μικρότερο βάρος από το p.)

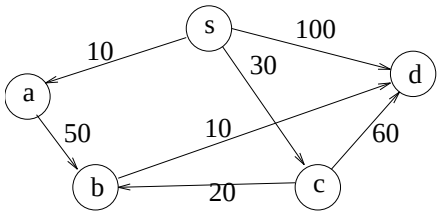
□

Ο αλγόριθμος του Dijkstra

- Λύνει το πρόβλημα των ελαχίστων μονοπατιών από κοινή αφετηρία για ένα κατευθυνόμενο γράφημα $G = (V, E)$ με μη αρνητικά βάρη ακμών.
- S: σύνολο κόμβων προς τους οποίους το τελικό ελαχίστου βάρους μονοπάτι από την αφετηρία s έχει ήδη προσδιοριστεί.
- $d[v]$: ένα άνω-όριο για το βάρος του ελαχίστου μονοπατιού από τον s προς τον v.
- Q: μία ουρά προτεραιότητας που περιέχει τους κόμβους του συνόλου $V - S$ με κλειδί τις d τιμές τους.

$Dijkstra(G, w, s)$
for each vertex $v \in V$ **do**
    $d[v] = +\infty$
$d[s] = 0$
$Q = V$
while $Q \neq \emptyset$ **do**
    $u = Extract_min(Q)$
    $S = S \cup \{u\}$
    for each vertex $v \in adj[u]$ **and** $v \in V - S$ **do**
        if $d[v] > d[u] + w[u, v]$ **then**
            $d[v] = d[u] + w[u, v]$

Παράδειγμα



Επανάληψη	S	u	$d[s]$	$d[a]$	$d[b]$	$d[c]$	$d[d]$
αρχικά	$\emptyset$	-	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	$\{s\}$	s	0	10	$+\infty$	30	100
2	$\{s, a\}$	a	0	10	60	30	100
3	$\{s, a, c\}$	c	0	10	50	30	90
4	$\{s, a, b, c\}$	b	0	10	50	30	60
5	$\{s, a, b, c, d\}$	d	0	10	50	30	60

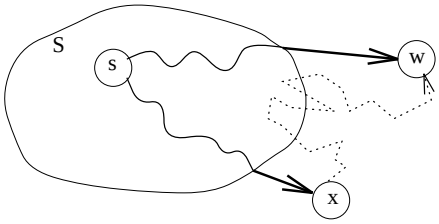
Ανάλυση

- Εάν η Q υλοποιηθεί σαν ένα διάνυσμα: $O(n^2)$.
- Εάν η Q υλοποιηθεί σαν ένας δυαδικός σωρός: $O(E \log V)$.
  (Προτιμάται για αραιά γραφήματα.)

Ορθότητα του αλγόριθμου του Dijkstra

- Το $d[v]$, για κάθε κόμβο v, περιέχει το κόστος του ελάχιστου μονοπατιού που αποτελείται μόνο από κόμβους του συνόλου S.
- Κάθε φορά που εισάγουμε ένα νέο κόμβο w στο S, το $d[w]$ περιέχει το κόστος του ελαχίστου μονοπατιού από την αφετηρία s προς τον w.

Απόδειξη



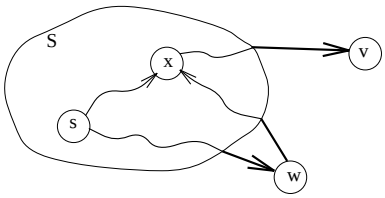
Εξετάστε ένα υποθετικό ελάχιστο μονοπάτι από τον s προς τον w, το οποίο 'βγαίνει' από το S και πηγαίνει πρώτα στον x, για να φθάσει αργότερα στον w.

Εάν το μονοπάτι αυτό έχει μικρότερο βάρος, τότε ο x έπρεπε να είχε επιλεγεί από τον αλγόριθμο αντί του w. Άρα, δεν εκτελέσαμε τον αλγόριθμο σωστά.
(Κατανοήστε πόσο σημαντικό είναι το ότι δεν έχουμε αρνητικά βάρη.)

□

- Το $d[v]$ είναι **πάντα** το κόστος του ελάχιστου μονοπατιού από τον s προς τον v που περνάει μόνο από κόμβους του S.

Απόδειξη



Όταν προσθέτουμε τον w στο S πρέπει να βεβαιωθούμε ότι εάν υπάρχει ένα μικρότερου κόστους μονοπάτι που καταλήγει στον v με προηγούμενο σταθμό τον w, τότε ενημερώνουμε κατάλληλα την τιμή $d[v]$.

- Εάν το μονοπάτι πηγαίνει μέσω του 'παλιού' S στον w και μετά κατευθείαν στον v, τότε η τιμή $d[v]$ έχει ενημερωθεί σωστά.
- Τι συμβαίνει εάν το μονοπάτι πηγαίνει στον w, μετά στον $x \in S$ και μετά στον v; (Ο αλγόριθμος δεν κάνει τίποτε για αυτή την περίπτωση.)

Η περίπτωση αυτή δεν μπορεί να συμβεί!

Μιας και ο x εισήχθη στο S πριν από τον w, το ελάχιστο από όλα το μονοπάτια από την αφετηρία s προς τον x διέρχεται αποκλειστικά μέσω του παλιού S. Άρα, το μονοπάτι προς τον x μέσω του w δεν έχει μικρότερο κόστος συγκριτικά με το μονοπάτι που περνά αποκλειστικά από κόμβους του S. Άρα, είναι αδύνατο να βελτιωθεί η τιμή $d[v]$. □

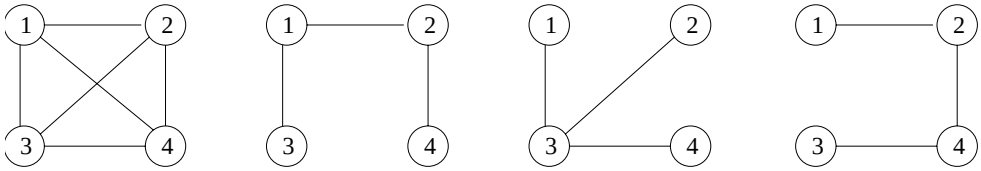
Ελάχιστα διασυνδετικά δένδρα (Minimum spanning trees)

Αφορμή για την εξέτασή τους

- Έχουν άπληστες λύσεις.
- Βρίσκουν εφαρμογή σε πολλά πραγματικά προβλήματα.

Ορισμός Έστω $G = (V, E)$ ένα μη-κατευθυνόμενο συνδεδεμένο γράφημα. Ένα υπογράφημα $T = (V, E')$ του G είναι ένα *διασυνδετικό δένδρο (spanning tree)* του G **εάν και μόνο εάν** το T είναι ένα δένδρο.

Παράδειγμα Ένα πλήρες γράφημα 4 κόμβων και 3 από τα διασυνδετικά δένδρα του.



- Εάν σε κάθε ακμή του γραφήματος προσυνάψουμε ένα βάρος, τότε μας ενδιαφέρει να υπολογίσουμε ένα *διασυνδετικό δένδρο ελαχίστου βάρους (minimum-weight spanning tree)*.

- Το συνολικό βάρος του διασυνδετικού δένδρου T ενός γραφήματος G είναι:

$$w(T) = \sum_{e \in T} w(e)$$

όπου e είναι μία ακμή και $w(e)$ είναι το βάρος της.

Εφαρμογή Στο σχεδιασμό ηλεκτρονικών κυκλωμάτων, συχνά μας ενδιαφέρει να καταστήσουμε n ακίδες (pins) ηλεκτρικά ισοδύναμες μέσω της καλωδίωσης τους. Θέλουμε να υπολογίσουμε μία 'καλή' καλωδίωση, δηλαδή, μια καλωδίωση ελάχιστου κόστους Το κόστος της ένωσης δύο οποιονδήποτε ακίδων θεωρείται δεδομένο.

Λύση

- Κάθε ακίδα αναπαριστάται από ένα κόμβο.
- Κάθε πιθανή ένωση αναπαρίσταται από μία ακμή κατάλληλου βάρους.

Τότε, αρκεί να ευρεθεί ένα ελάχιστο διασυνδετικό δένδρο.

Ανάπτυξη ενός ελάχιστου διασυνδετικού δένδρου

Είσοδος: Ένα συνδεδεμένο μη-κατευθυνόμενο γράφημα $G = (V, E)$ και μία συνάρτηση βάρους $w : E \to \mathcal{R}$.

Έξοδος: Ένα ελάχιστο διασυνδετικό δένδρο για το G.

- **Ένας γενικός αλγόριθμος**

$Generic_MST(G, w)$
$A = \emptyset$
while A does not form a spanning tree **do**
  Find an edge (u, v) that is safe for A.
  $A = A \cup \{(u, v)\}$
return A

Σταθερά (invariant) του αλγορίθμου 'Το A είναι πάντοτε ένα υποσύνολο κάποιου ελάχιστου διασυνδετικού δένδρου.'

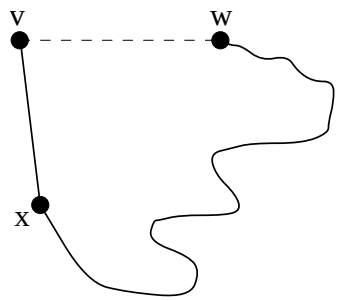
- Μία ακμή (u, v) είναι μία *ασφαλής ακμή (safe edge)* για το A εάν μπορεί να προστεθεί στο A χωρίς να διαταραχθεί η σταθερά του αλγορίθμου.

  Εάν το A δεν έχει ήδη σχηματίσει ένα (ελάχιστο) διασυνδετικό δένδρο, τότε πάντοτε μπορούμε να υπολογίσουμε μία ασφαλή για το A ακμή.

Υπολογισμός μίας ασφαλής ακμής για το A

Λήμμα Θεωρήστε ένα οποιοδήποτε κόμβο v. Έστω (v, w) μία ελαχίστου βάρους, προσκείμενη στον v, ακμή. Τότε, υπάρχει ένα ελάχιστο διασυνδετικό δένδρο το οποίο περιέχει την ακμή (v, w).

Απόδειξη (Με εις άτοπο επαγωγή)

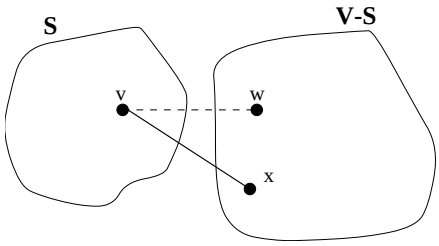


- Υποθέστε ότι ένα τέτοιο δένδρο δεν υπάρχει.
- Θεωρήστε ένα οποιοδήποτε ελάχιστο διασυνδετικό δένδρο T.
  Έστω (v, x) η ακμή του T που είναι προσκείμενη στον κόμβο v.
- Προσθέτοντας την (v, w) στο T δημιουργούμε ένα κύκλο. Ο κύκλος αυτός μπορεί να 'σπάσει' με την αφαίρεση της (v, x). Έτσι δημιουργείται ένα νέο διασυνδετικό δένδρο $T' = T - \{(v, x)\} \cup \{(v, w)\}$.
- $w(T') \leq w(T)$ διότι $w((v, w)) \leq w((v, x))$. Άρα, το T' περιέχει την (v, w) και επιπλέον δεν έχει βάρος μεγαλύτερο από το ελάχιστο διασυνδετικό δένδρο T. Μία καθαρή αντίφαση. □

$\Rightarrow$ Από το παραπάνω λήμμα, **γνωρίζουμε πώς να εντοπίσουμε μία ασφαλή ακμή**.

Λήμμα Έστω A ένα υποσύνολο των ακμών ενός ελάχιστου διασυνδετικού δένδρου και έστω S το σύνολο των προσκείμενων στις ακμές του A κόμβων. Έστω (v, w) μία ελάχιστου βάρους ακμή τέτοια ώστε $v \in S$ και $w \in V - S$. Τότε, η (v, w) είναι μία ασφαλής ακμή για το A.

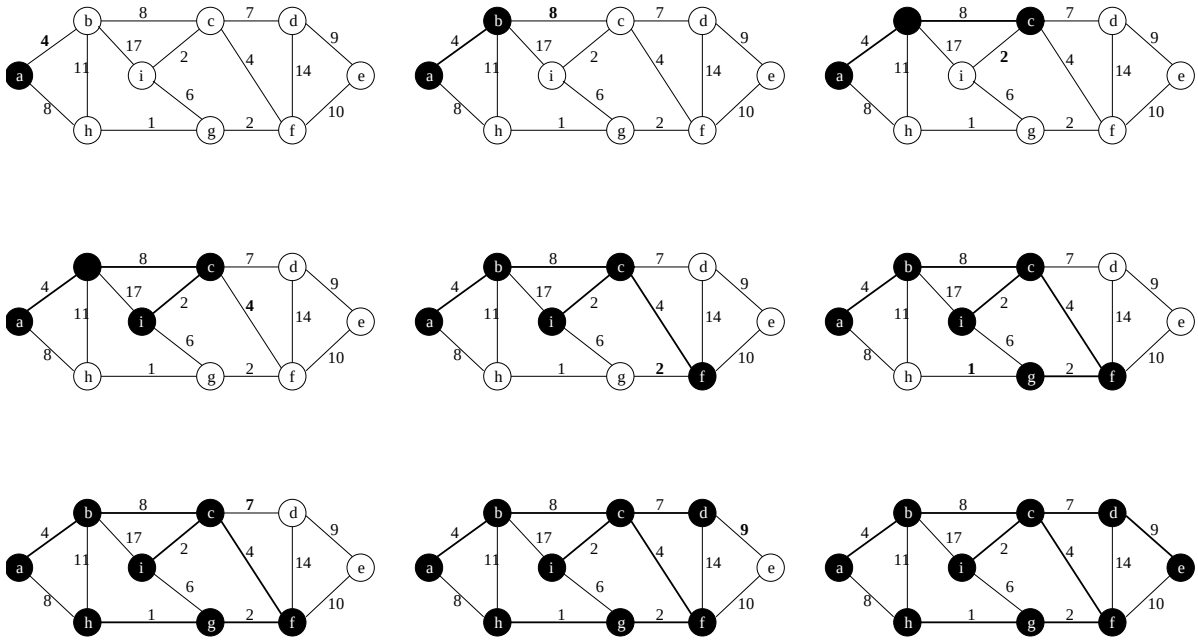
Απόδειξη (Με εις άτοπο επαγωγή)



- Έστω ότι το T είναι ένα ελάχιστο διασυνδετικό δένδρο και η ακμή του (v, x) η οποία ενώνει το S με το $V - S$.
- Το $T' = T - \{(v, x)\} \cup \{(v, w)\}$ είναι ένα διασυνδετικό δένδρο, ίσου ή μικρότερου βάρους με το T, το οποίο επιπλέον περιέχει την ακμή (v, w). ία καθαρή αντίφαση.

□

Παράδειγμα



Ο αλγόριθμος του Prim

- Κατά την εκτέλεση του αλγόριθμου, όλοι οι κόμβοι που δεν έχουν εισαχθεί στο δένδρο βρίσκονται σε μία ουρά προτεραιότητας Q με κλειδί το πεδίο key.
- Για κάθε κόμβο $v \in V, key[v]$ είναι το ελάχιστο βάρος της ακμής που ενώνει τον κόμβο v με έναν κόμβο του δένδρου.
- Το πεδίο $\pi[v]$ δηλώνει τον 'πατέρα' του v στο δένδρο.

$MST_prim(G, w, r)$ /* r is the root of the spanning tree */
for each $u \in V$ **do** $key[u] = +\infty$
$key[r] = 0$
$\pi[r] = nil$
$Q = V$
while $Q \neq \emptyset$ **do**
  $u = extract_min(Q)$
  for each $v \in adjacent_list(u)$ **do**
    if $v \in Q$ **and** $w[u, v] < key[v]$ **then**
      $\pi[v] = u$
      $key[v] = w(u, v)$

Ανάλυση: $O(V \log V + E \log V) = O(E \log V)$ εάν η ουρά προτεραιότητας υλοποιηθεί σαν δυαδικός σωρός.

$O(V \log V + E)$ εάν η ουρά προτεραιότητας υλοποιηθεί σαν σωρός Fibonacci.

Συμπέρασμα

- Πολλά προβλήματα βελτιστοποίησης μπορεί να λυθούν από ένα τετριμμένο αλγόριθμο. Συνήθως, ο αλγόριθμος αυτός **δεν** γρήγορος.

Παραδείγματα

- Ελάχιστα διασυνδετικά δένδρα

  Κατασκεύασε όλα τα διασυνδετικά δένδρα.
  Προσδιόρισε το δένδρο με το ελάχιστο βάρος.

  Ανάλυση: $O\left(C\left(|E|, n-1\right) \cdot n\right)$

- Ελάχιστα μονοπάτια από κοινή αφετηρία (με μη αρνητικά βάρη)

  Για κάθε κόμβο v, κατασκεύασε όλα τα μονοπάτια από την αφετηρία s προς τον κόμβο v.
  Προσδιόρισε το μονοπάτι ελάχιστο βάρους.

  Ανάλυση: $O\left(C\left(|E|, n-1\right) \cdot n \cdot n\right)$

- Η εξεύρεση ιδιοτήτων που διέπουν την βέλτιστη λύση οδηγεί σε γρήγορους αλγόριθμους.

Δυναμικός προγραμματισμός (Dynamic Programming)

Χαρακτηριστικά γνωρίσματα

- Η βέλτιστη λύση του προβλήματος περιέχει βέλτιστες λύσεις των υπο-προβλημάτων του.
- Επικαλυπτόμενα υπό-προβλήματα.

Θα εξετάσουμε:

- Αριθμούς Fibonacci.
- Πολλαπλασιασμός αλυσίδας πινάκων.
- Μεγαλύτερη κοινή υπό-ακολουθία.
- Ελάχιστα μονοπάτια για όλα τα ζεύγη κόμβων.

Αριθμοί Fibonacci

Ο n-οστός *αριθμός Fibonacci* $F(n)$ ορίζεται από την αναδρομική σχέση:

$$\begin{aligned} &F(0) = 0 \\ &F(1) = 1 \\ &F(n) = F(n-1) + F(n-2) \qquad \text{για } n \geq 2 \end{aligned}$$

$F(2) = 1,\ F(3) = 2,\ F(4) = 3,\ F(5) = 5,\ F(6) = 8,\ F(7) = 13,\ F(8) = 21,\ \cdots$

Ένας προφανής αλγόριθμος για τον υπολογισμό του $F(n),\ n \geq 1$

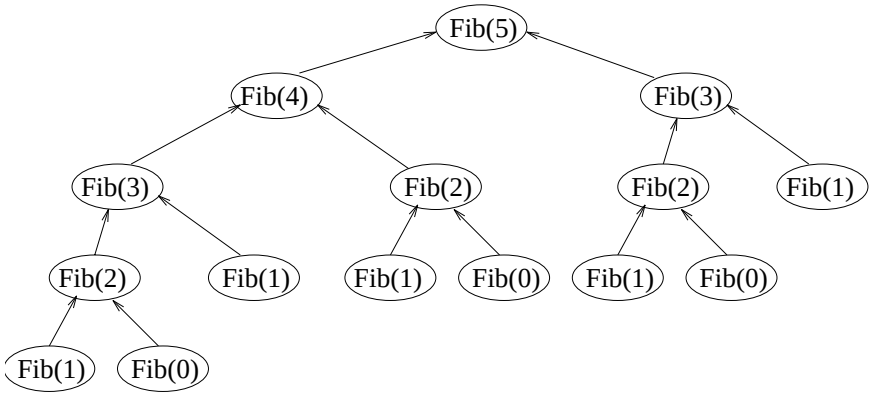
Fibinacci(n)
if $n < 2$ **then return** (n)
else return $(Fibonacci(n-1) + Fibonacci(n-2))$

Ανάλυση $T(n) = \begin{cases} 1 & n < 2 \\ T(n-1) + T(n-2) & n \geq 2 \end{cases}$

$$\Longrightarrow \quad T(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} \quad \Longrightarrow \quad T(n) = \Theta(\phi^n)$$

$\phi = \frac{1+\sqrt{5}}{2} = 1.61803, \quad \hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803$

- Για $n = 5$, το 'δένδρο αναδρομής' (recursion tree) είναι το:



Υπολογίζουμε το : $F(3)$ 2 φορές,
$F(2)$ 3 φορές,
$F(1)$ 5 φορές,
$F(0)$ 3 φορές.

Υπάρχουν πολλοί περιττοί υπολογισμοί!

Μία καλύτερη λύση

- Υπολόγισε το $F(n)$ ξεκινώντας από τη βάση (bottom-up).

$Non_Recursive_Fibonacci(n)$
/* $A[1..n]$ is an array in which we store $F(1) \cdots F(n)$ */
$A[0] = 0$
$A[1] = 1$
for $i = 2$ **to** n **do**
  $A[i] = A[i-1] + A[i-2]$
return $A[n]$

Ανάλυση: $\Theta(n)$

Πολλαπλασιασμός αλυσίδας πινάκων

Δίδεται μία ακολουθία (αλυσίδα) $< A_1,\ A_2,\ \cdots,\ A_n >$ από n πίνακες και θέλουμε να υπολογίσουμε το γινόμενο $A_1 \times A_2 \times \cdots \times A_n$.

- $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$, δηλ., ο πολλαπλασιασμός πινάκων είναι προσεταιριστικός.

  $\Longrightarrow$ Όλες οι παρενθετικοποιήσεις δίνουν το ίδιο αποτέλεσμα.

Παράδειγμα Για $n = 4$, υπάρχουν 5 διακριτές παρενθετικοποιήσεις:

$$(A_1 \times (A_2 \times (A_3 \times A_4)))$$

$$(A_1 \times ((A_2 \times A_3) \times A_4))$$

$$((A_1 \times A_2) \times (A_3 \times A_4))$$

$$((A_1 \times (A_2 \times A_3)) \times A_4)$$

$$(((A_1 \times A_2) \times A_3) \times A_4)$$

Ορισμός Ένα γινόμενο πινάκων είναι *πλήρως παρενθετικοποιημένο* εάν είναι ένας απλός πίνακας ή το γινόμενο δύο πλήρως παρενθετικοποιημένων γινομένων που περικλείεται από παρενθέσεις.

Σημείωση Ο ορισμός αυτός μπορεί να γενικευθεί για κάθε είδους δυαδικού τελεστή.

Άσκηση Να δειχθεί ότι η πλήρη παρενθετικοποίηση μιας μαθηματικής έκφρασης n στοιχείων έχει ακριβώς $n - 1$ ζευγάρια παρενθέσεων. (Υποθέτουμε δυαδικούς τελεστές.)

$Matrix_multiply(A, B)$
if $columns(A) \neq rows(B)$
then error "incompatible dimension"
else for $i = 1$ **to** $rows(A)$ **do**
  for $j = 1$ **to** $columns(B)$ **do**
    $C[i, j] = 0$
    for $k = 1$ **to** $columns(A)$ **do**
      $C[i, j] = C[i, j] + A[i, k] \cdot B[k, j]$

- Εάν ο A είναι ένας $p \times q$ πίνακας και ο B είναι ένας $q \times r$ πίνακας, τότε ο C είναι ένας $q \times r$ πίνακας.
- Ο αλγόριθμος εκτελεί ακριβώς pqr πολλαπλασιασμούς.

Παράδειγμα $A_1 = 10 \times 100, \quad A_2 = 100 \times 5, \quad A_3 = 5 \times 50$

$((A_1 \times A_2) \times A_3)$ χρειάζεται $(10 \cdot 100 \cdot 5) + (10 \cdot 5 \cdot 50) = 7,500$ πολλαπλασιασμούς.
$(A_1 \times (A_2 \times A_3))$ χρειάζεται $(10 \cdot 100 \cdot 50) + (100 \cdot 5 \cdot 50) = 75,000$ πολλαπλασιασμούς.

$\Longrightarrow$ Η σειρά με την οποία εκτελούνται οι πολλαπλασιασμοί των πινάκων είναι πολύ σημαντική!

Ερώτηση Ποιος είναι ο αριθμός των δυνατών παρενθετικοποιήσεων·

Απάντηση Συμβολίζουμε τον αριθμό αυτό με $P(n)$.

$$\underbrace{(A_1 \times A_2 \times \cdots \times A_k)}_{P(k)} \times \underbrace{(A_{k+1} \times \cdots \times A_n)}_{P(n-k)}$$

Μπορούμε να γράψουμε την αναδρομική σχέση:

$$P(n) = \begin{cases} 1 & \text{εάν} \quad n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{εάν} \quad n \geq 2 \end{cases}$$

Η λύση είναι η ακολουθία των αριθμών Catalan. $P(n) = C(n-1)$ όπου,

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega(4^n/n^{3/2})$$

(Εκθετικό ως προς n!)

Η δομή της βέλτιστης παρενθετικοποίησης

- $A_{i..j}$ συμβολίζει τον πίνακα που προκύπτει από τον πολλαπλασιασμό $A_i \times \cdots \times A_j$.

Παρατήρηση Μία βέλτιστη παρενθετικοποίηση του $A_1 \times \cdots \times A_n$ χωρίζει το γινόμενο μεταξύ των A_k και A_{k+1} για κάποιο ακέραιο $k,\ 1 \leq k < n$.

$$((A_1 \times A_2 \times \cdots \times A_k)\ \times\ (A_{k+1} \times \cdots \times A_n))$$

Θεώρημα Η παρενθετικοποίηση της υπό-αλυσίδας $A_1 \times \cdots \times A_k$ σε μία βέλτιστη παρενθετικοποίηση της $A_1 \times \cdots \times A_n$ είναι επίσης βέλτιστη.

Απόδειξη Εάν υπάρχει μία καλύτερη παρενθετικοποίηση για το $A_1 \times \cdots \times A_k$, τότε μπορούμε να κατασκευάσουμε μία καλύτερη παρενθετικοποίηση από τη βέλτιστη για το $A_1 \times \cdots \times A_n$. Μία καθαρή αντίφαση.

□

Σημείωση Όμοιο θεώρημα ισχύει για την υπό-αλυσίδα $A_{k+1} \times \cdots \times A_n$.

Μία αναδρομική λύση

- $m[i,j]$ συμβολίζει τον ελάχιστο αριθμό των αναγκαίων πολλαπλασιασμών για τον υπολογισμό του $A_{i..j}$.
- Θέλουμε να βρούμε την παρενθετικοποίηση του χρειάζεται $m[1,n]$ πολλαπλασιασμούς για τον υπολογισμό του $A_{1..n}$.
- Υποθέτουμε ότι ο πίνακας A_i είναι διαστάσεων $[p_{i-1} \times p_i]$.

Μπορούμε να υπολογίσουμε το $m[1,n]$ μέσω της αναδρομικής σχέσης:

$$m[i,j] = \begin{cases} 0 & \text{εάν} \quad i = j \\ \min_{i \leq k \leq j}\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j & \text{εάν} \quad i < j \end{cases}$$

- Ένας αναδρομικός αλγόριθμος βασιζόμενος στην παραπάνω σχέση χρειάζεται **τουλάχιστον** εκθετικό χρόνο για την εκτέλεσή του!

Μία λύση δυναμικού προγραμματισμού

Παρατήρηση Πρέπει να λύσουμε $\Theta(n^2)$ υπό-προβλήματα, ένα για κάθε τιμή του ζεύγους i και j που ικανοποιεί τη σχέση $1 \leq i \leq j \leq n$.

Ιδέα: Θα υπολογίσουμε το $m[1,n]$ ξεκινώντας από τη βάση (bottom-up). Θα υπολογίσουμε όλα τα $m[i,j]$ για αυξανόμενες τιμές του $(j-i)$.

```
Matrix_Chain_order(p)
/* p =< p_0, p_1, ···, p_n >, the dimensions of the matrices */
n = length(p) − 1
for i = 1 to n do
    m[i,i] = 0
for l = 2 to n do            /* l denotes the "length" of A_{i..j} */
    for i = 1 to n − l + 1 do
        j = i + l − 1
        m[i,j] = 0
        for k = i to j − 1 do
            q = m[i,k] + m[k+1,j] + p_{i−1} · p_k · p_j
            if q < m[i,j] then m[i,j] = q
return m
```

Ανάλυση: $\Theta(n^3)$

Παράδειγμα

Matrix	Dimensions
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

M (i \ j)	1	2	3	4	5	6
1	0	15,750	7,875	9,375	11,875	15,125
2		0	2,625	4,375	7,125	10,500
3			0	750	2,500	5,375
4				0	1,000	3,500
5					0	5,000
6						0

$m[1,2] = 30 \cdot 35 \cdot 15 = 15,750$

$$m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + p_0 \cdot p_1 \cdot p_3 = 0 + 2,625 + 5,250 = \mathbf{7,875} \\ m[1,2] + m[3,3] + p_0 \cdot p_2 \cdot p_3 = 15,750 + 0 + 2,250 = 18,000 \end{cases}$$

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + p_0 \cdot p_1 \cdot p_4 = 0 + 4,375 + 10,500 = 14,875 \\ m[1,2] + m[3,4] + p_0 \cdot p_2 \cdot p_4 = 15,750 + 750 + 4,500 = 21,000 \\ m[1,3] + m[4,4] + p_0 \cdot p_3 \cdot p_4 = 7,875 + 0 + 1,500 = \mathbf{9,375} \end{cases}$$

Ερώτηση Πως θα ανακτήσουμε την παρενθετικοποίηση·

Απάντηση Μέσω ενός επιπρόσθετου πίνακα *split* όπου, $split[i,j]$ δηλώνει τη θέση όπου χωρίζουμε το γινόμενο $A_{i..j}$.

$Matrix_Chain_order_1(p)$
/* $p =< p_0,\ p_1,\ \cdots,\ p_n >$, the dimensions of the matrices */
$n = length(p) - 1$
for $i = 1$ **to** n **do**
    $m[i,i] = 0$
for $l = 2$ **to** n **do** /* l denotes the "length" of $A_{i..j}$ */
    for $i = 1$ **to** $n - l + 1$ **do**
        $j = i + l - 1$
        $m[i,j] = 0$
        for $k = 1$ **to** $j - 1$ **do**
            $q = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$
            if $q < m[i,j]$ **then**
                $m[i,j] = q$
                $split[i,j] = k$
return $m,\ split$

Ο αλγόριθμος πολλαπλασιασμού αλυσίδας πινάκων

$Matrix_Chain_Multiply(A, split, i, j)$
/* Computes $A_{i..j}$. A is the set of matrices */

if $j > i$ **then** $X = Matrix_Chain_Multiply(A, split, i, split[i,j])$
    $Y = Matrix_Chain_Multiply(A, split, split[i,j]+1, j)$
    return $Matrix_Multiply(X, Y)$
else return A_i

- **Στοιχεία δυναμικού προγραμματισμού**
  - Η βέλτιστη λύση του προβλήματος περιέχει βέλτιστες λύσεις των υπό-προβλημάτων του.
  - Επικαλυπτόμενα υπό-προβλήματα.
- **Απομνημόνευση (Memoization)** Χρησιμοποίησε τον αναδρομικό αλγόριθμο, αλλά αποθήκευσε την λύση σε κάθε υπό-πρόβλημα για μελλοντική χρήση. Με τον τρόπο αυτό λύνουμε το κάθε υπό-πρόβλημα μόνο μία φορά.

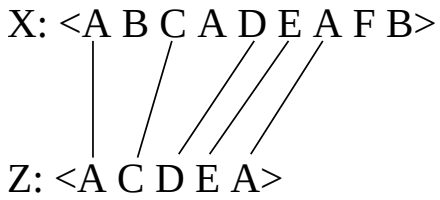
Μέγιστη κοινή ακολουθία

Ορισμός Με δεδομένη μία ακολουθία $X =< x_1, x_2, \cdots, x_m >$, μία άλλη ακολουθία $Z =< z_1, z_2, \cdots, z_k >$ είναι υπό-ακολουθία της X εάν υπάρχει μία γνησίως αύξουσα ακολουθία $< i_1, i_2, \cdots, i_k >$ από δείκτες της X τέτοια ώστε για όλα τα $j = 1 \cdots k$ ισχύει $x_{i_j} = z_j$.

Παράδειγμα $X =<$ A, B, C, A, D, E, A, F, B $>$

Υπό-ακολουθία της X	Ακολουθία δεικτών της X
$Z =<$ A, C, D, E, A $>$	$<$ 1, 3, 5, 6, 7 $>$
$Z =<$ A, B $>$	$<$ 1, 2 $>$, $<$ 1, 9 $>$, $<$ 4, 9 $>$

- Μια καλύτερη αναπαράσταση:



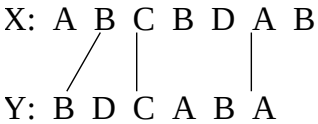
Το πρόβλημα της μέγιστης κοινής υπό-ακολουθίας (ΜΚΥ)

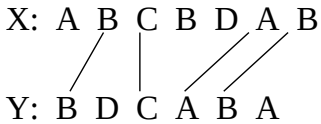
(**The longest common subsequence (LCS) problem**)

Είσοδος: Δυο ακολουθίες X και Y.

Έξοδος: Η μέγιστη κοινή υπό-ακολουθία.

Παραδείγματα





Γιατί μελετάμε το πρόβλημα της ΜΚΥ;

- Λύνεται με χρήση δυναμικού προγραμματισμού.
- Έχει αρκετές εφαρμογές.

Εφαρμογή

Με δεδομένες 2 διαφορετικές εκδόσεις (versions) ενός προγράμματος, να προσδιοριστούν οι αλλαγές που έγιναν από την πρώτη έκδοση έως τη δεύτερη.

Λύση

Βρείτε την ΜΚΥ των δύο προγραμμάτων. Το κείμενο που δεν είναι μέλος της ΜΚΥ προέρχεται από προσθέσεις ή αφαιρέσεις που έγιναν κατά την μετάβαση από την μία έκδοση προς την άλλη.

Μια απλοϊκή προσέγγιση

- Έστω S το σύνολο όλων των δυνατών υπό-ακολουθιών των X και Y.
- Έλεγξε κάθε υπό-ακολουθία του S και προσδιόρισε την ΜΚΥ.

Ερώτηση Ποιο είναι το μέγεθος του συνόλου S;

Απάντηση Έστω $k = \min(legth(X), length(Y))$. Τότε, $|S| = 2^k$.

Αυτό καθιστά την απλοϊκή προσέγγιση ανεδαφική!

Μία αναδρομική λύση

Έστω $X =< x_1, x_2, \cdots, x_m >$ και $Y =< y_1, y_2, \cdots, y_n >$.

Συμβόλισε με X_i την υπό-ακολουθία $X_i =< x_1, x_2, \cdots, x_i >$
και με Y_i ην υπό-ακολουθία $Y_i =< y_1, y_2, \cdots, y_i >$.

Έστω $c[i, j]$ το μέγεθος της ΜΚΥ των X_i και Y_j.

Μας ενδιαφέρει να υπολογίσουμε το $c[m, n]$.

Μπορούμε να γράψουμε την αναδρομική σχέση:

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

Ερώτηση ι) Είναι η παραπάνω σχέση σωστή·
ιι) Πως καταλήξαμε σε αυτή·

Απάντηση ι) ΝΑΙ

ιι) $$c[i,j] = \begin{cases} 0 & \text{εάν } i = 0 \text{ ή } j = 0 \\ c[i-1, j-1] + 1 & \text{εάν } i, j > 0 \text{ και } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{εάν } i, j > 0 \text{ και } x_i \neq y_j \end{cases}$$

- εάν $i = 0$ ή $j = 0$:
  Τουλάχιστον μία από τις ακολουθίες είναι κενή. Άρα, η μόνη υπό-ακολουθία είναι η κενή υπό-ακολουθία (συμβολίζεται με το ϵ).
- εάν $i, j > 0$ και $x_i = y_j$:

$$\begin{array}{c} < x_1, x_2, \cdots, x_{i-1}, x_i > \\ = \\ < y_1, y_2, \cdots, y_{j-1}, y_j > \end{array}$$

$LCS(X_i, Y_j) = LCS(X_{i-1}, Y_{j-1}) \oplus x_i$
$\Longrightarrow c[i, j] = c[i-1, j-1] + 1$

($\oplus$ συμβολίζει τη συνένωση συμβολοσειρών / ακολουθιών)

- εάν $i, j > 0$ και $x_i \neq y_j$:

$$\begin{array}{c} < x_1, x_2, \cdots, x_{i-1}, x_i > \\ \neq \\ < y_1, y_2, \cdots, y_{j-1}, y_j > \end{array}$$

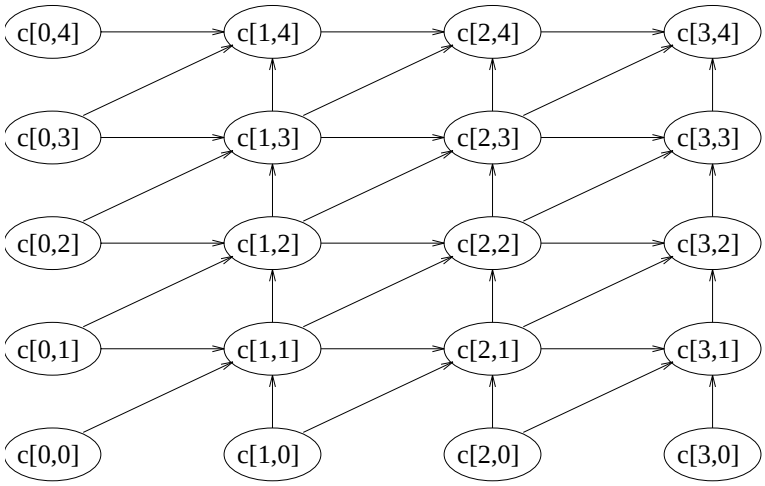
$$LCS[X_i, Y_j] = \begin{cases} LCS[X_i, Y_{j-1}] \\ \text{ή} \\ LCS[X_{i-1}, Y_j] \end{cases}$$

$\Longrightarrow c[i, j] = \max(c[i, j-1], c[i-1, j])$

Μία αναδρομική προσέγγιση

- Στην περίπτωση όπου οι 2 υπό-ακολουθίες δεν έχουν κοινές υπό-ακολουθίες (εκτός της ϵ), ο αλγόριθμος χρειάζεται εκθετικό χρόνο.

Παράδειγμα Σχεδιάστε το 'γράφημα αναδρομής' για το $C[3,4]$.



Άσκηση Να υπολογιστεί ο αριθμός των κλήσεων της διαδικασίας που υπολογίζει το $c[i,j]$ κατά τη διάρκεια του υπολογισμού του $c[m,n]$, όπου $0 \leq i \leq m, \quad 0 \leq j \leq n$.

- Προφανώς, υπάρχουν επικαλυπτόμενα υπό-προβλήματα τα οποία ο αλγόριθμος θα λύσει παραπάνω από μία φορές.

(Ο αναδρομικός αλγόριθμός χρειάζεται τουλάχιστον εκθετικό χρόνο.)

Παρατήρηση Το $c[m,n]$ μπορεί να υπολογιστεί εύκολα ξεκινώντας από τη βάση (bottom-up).

(Να συμπληρωθεί ο πίνακας c γραμμή-γραμμή από τα αριστερά προς τα δεξιά.)

```
LCS_length(X, Y)
m = length(X)
n = length(Y)
for i = 1 to m do   c[i, 0] = 0
for j = 1 to n do   c[0, j] = 0
for i = 1 to m do
        for j = 1 to n do
                if x_i = y_j
                        then c[i, j] = c[i − 1, j − 1] + 1
                             b[i, j] = ↙
                        else if c[i − 1, j] ≥ c[i, j − 1]
                                then c[i, j] = c[i − 1, j]
                                     b[i, j] = ↓
                                else c[i, j] = c[i, j − 1]
                                     b[i, j] = ←
return c, b
```

Σημείωση Ο πίνακας b χρησιμοποιείται για την ανάκτηση της ΜΚΥ.

Παράδειγμα X= A B C B D A B
Y= B D C A B A

i	j	0	1 (B)	2 (D)	3 (C)	4 (A)	5 (B)	6 (A)
7	B	0	↙ 1	↓ 2	↓ 2	↓ 3	↙ 4	↓ 4
6	A	0	↓ 1	↓ 2	↓ 2	↙ 3	↓ 3	↙ 4
5	D	0	↓ 1	↙ 2	↓ 2	↓ 2	↓ 3	↓ 3
4	B	0	↙ 1	↓ 1	↓ 2	↓ 2	↙ 3	← 3
3	C	0	↓ 1	↓ 1	↙ 2	← 2	↓ 2	↓ 2
2	B	0	↙ 1	← 1	← 1	↓ 1	↙ 2	← 2
1	A	0	↓ 0	↓ 0	↓ 0	↙ 1	← 1	↙ 1
0		0	0	0	0	0	0	0

Ερώτηση 1 Ποια είναι η πολυπλοκότητα της LCS()·

Απάντηση: $O(mn)$

Ερώτηση 2 Πόσος χρόνος χρειάζεται για την ανάκτηση της ΜΚΥ από τον πίνακα b·

Απάντηση: $O(m+n)$

Ερώτηση 3 Πόση μνήμη απαιτείται για την εκτέλεση της LCS()·

Απάντηση: $O(mn)$

Ερώτηση 4 Είναι δυνατό να βελτιωθεί η πολυπλοκότητα χώρου στην περίπτωση όπου θέλουμε να υπολογίσουμε μόνο το μήκος της ΜΚΥ·

Απάντηση: Ναι! Μόνο 2 γραμμές του πίνακα c είναι απαραίτητοι.
$\Longrightarrow$ $O(\min(m,n))$ μνήμη.

Σημείωση: Στην περίπτωση αυτή δεν μπορεί να ανακτηθεί η ΜΚΥ.

Ελάχιστα μονοπάτια για κάθε ζεύγος κόμβων

Είσοδος: Ένα κατευθυνόμενο, με βάρη, γράφημα $G = (V, E)$ με συνάρτηση βάρους $w : E \to \mathcal{R}$.

Έξοδος Για κάθε ζεύγος κόμβων $u, v \in V$ το ελάχιστο μονοπάτι από τον u προς τον v.

Εφαρμογή Να δημιουργηθεί ένας πίνακας αποστάσεων μεταξύ των πόλεων ενός οδικού χάρτη.

Ένας απλός αλγόριθμος

$Trivial_All_Pairs(G, w)$

for each $v \in V(G)$ **do**

Run $Dijkstra()$ with v as the source vertex

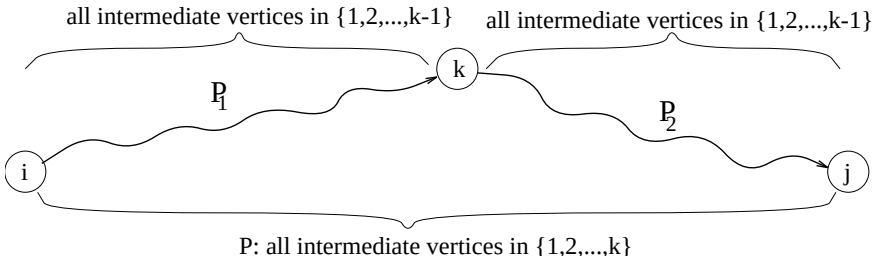
Πολυπλοκότητα Υλοποίηση 'διανύσματος': $O(V^3)$
Υλοποίηση 'δυαδικού σωρού': $O(VE \log V)$

Σημείωση Ο παραπάνω αλγόριθμος δίνει τη σωστή απάντηση μόνο εάν όλα τα βάρη είναι μη-αρνητικά.

Μία προσέγγιση μέσω δυναμικού προγραμματισμού

- **Η δομή ενός ελάχιστου μονοπατιού**

Θεωρήστε το ελάχιστο μονοπάτι από τον κόμβο i προς τον κόμβο j **που διέρχεται μόνο** από κόμβους του συνόλου $\{1, 2, \cdots, k\}$.



- **Μία αναδρομική λύση**

Συμβολισμός $d_{i,j}^{(k)}$ =Το βάρος του ελάχιστου μονοπατιού από τον i προς τον j που διέρχεται μόνο από κόμβους του συνόλου $\{1, 2, \cdots, k\}$.

$$d_{i,j}^{(k)} = \begin{cases} w_{i,j} & \text{εάν } k = 0 \\ \min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}) & \text{εάν } k \geq 1 \end{cases}$$

$Floyd_Warshal(W)$

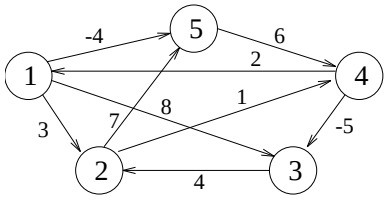
$n = rows(W)$
$d^{(0)} = W$
for $k = 1$ **to** n **do**
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** n **do**
      $d_{i,j}^{(k)} = \min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)})$
return $d^{(n)}$

Πολυπλοκότητα χρόνου & χώρου: $O(V^3)$

Σημείωση 1 Η πολυπλοκότητα χώρου μπορεί να μειωθεί σε $O(V^2)$. Παρατηρήστε ότι ο υπολογισμός του $d^{(k)}$ χρησιμοποιεί μόνο τον $d^{(k-1)}$.

Σημείωση 2 Ο αλγόριθμος $Floyd_Warshal$ δίνει το σωστό αποτέλεσμα μόνο για γραφήματα **χωρίς αρνητικούς κύκλους**. (Γιατί·)

Σημείωση 3 Έως τώρα, υπολογίσαμε μόνο το κόστος των ελάχιστων μονοπατιών. Δεν έχουμε δείξει πως να ανακτήσουμε τα μονοπάτια. (Με ποίο τρόπο ανακτάμε τα μονοπάτια·)



$$d^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$d^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

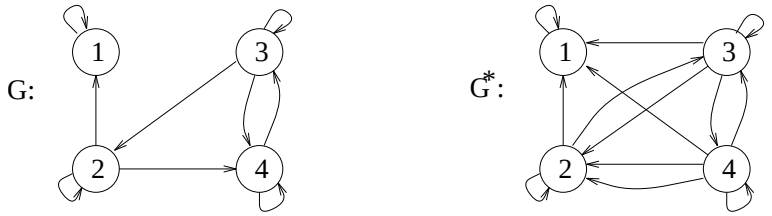
Μεταβατική κλειστότητα κατευθυνόμενων γραφημάτων

Transitive closure of directed graphs

Είσοδος: Ένα κατευθυνόμενο γράφημα $G = (V, E)$.

Έξοδος: Ένα κατευθυνόμενο γράφημα $G^\star = (V, E^\star)$ όπου
$E^\star = \{(i, j) :$ υπάρχει μονοπάτι από τον i προς τον j στο $G\}$.

Παράδειγμα



Ένας απλός αλγόριθμος

- Καταχώρησε βάρος '1' σε όλες τις ακμές του G.
- Εκτέλεσε τον αλγόριθμο $Floyd_Warshal()$.
- Εάν το κόστος του μονοπατιού από τον i προς τον j είναι διάφορο του ∞ τότε $(i, j) \in E^\star$.

Ένας συναφής αλγόριθμος

$$t_{i,j}^{(0)} = \begin{cases} 0 & \text{εάν } i \neq j \text{ και } (i,j) \notin E \\ 1 & \text{εάν } i = j \text{ ή } (i,j) \in E \end{cases} \qquad t_{i,j}^{(k)} = t_{i,j}^{(k-1)} \quad \vee \quad (t_{i,k}^{(k-1)} \ \wedge \ t_{k,j}^{(k-1)})$$

$Transitive_Closure(G)$

$n = |V|$
for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    if $i = j$ **or** $(i, j) \in E$ **then** $t_{i,j}^{(0)} = 1$
    else $t_{i,j}^{(0)} = 0$
for $k = 1$ **to** n **do**
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** n **do**
      $t_{i,j}^{(k)} = t_{i,j}^{(k-1)} \quad \vee \quad (t_{i,k}^{(k-1)} \ \wedge \ t_{k,j}^{(k-1)})$
return $t^{(n)}$

Ανάλυση: Πανομοιότυπη με αυτή του $Floyd_Warshal$.

Πλεονεκτήματα Είναι γρηγορότερος και χρησιμοποιεί λιγότερη μνήμη (κατά ένα σταθερό παράγοντα).

Κάτω φράγματα

Δοθέντος ενός προβλήματος P και ενός αλγορίθμου A που λύνει το P, επιθυμούμε να γνωρίζουμε: *"Εάν είναι ο αλγόριθμος A ο καλύτερος δυνατός αλγόριθμος για το πρόβλημα P; "*

Πιθανές βελτιώσεις του A μπορεί να είναι:

- Μείωση του απαιτούμενου χρόνου
- Μείωση του απαιτούμενου χώρου (μνήμη)

Επιθυμούμε ένα θεώρημα της μορφής:

"Όλοι οι αλγόριθμοι για το πρόβλημα P έχουν πολυπλοκότητα: $T(n) \geq f(n)$. "

- Μία συνάρτηση $f(n)$ που εμφανίζεται σε ένα τέτοιο θεώρημα ονομάζεται ένα ***κάτω φράγμα*** *(lower bound)* της πολυπλοκότητας του προβλήματος P.
- Εάν υπάρχει ένας αλγόριθμος για το P πολυπλοκότητας $f(n)$, τότε ο αλγόριθμος αυτός είναι *βέλτιστος (optimal)*.

Το μοντέλο υπολογισμού πρέπει να είναι καλώς ορισμένο.

Πρέπει να είναι ξεκάθαρο ποιο είναι το μοντέλο υπολογισμού που θεωρούμε όταν καταλήγουμε σε ένα κάτω φράγμα (lower bound).

- Ο τρόπος προσπέλασης των δεδομένων πρέπει να είναι καθορισμένος με ακρίβεια (τυχαία προσπέλαση - σειριακή προσπέλαση).
- Οι επιτρεπτές πράξεις στα δεδομένα πρέπει να είναι καθορισμένες με σαφήνεια (το σύνολο των εντολών που χρησιμοποιεί ο αλγόριθμος).

Παράδειγμα

Ας θεωρήσουμε μία ταξινομημένη λίστα στοιχείων σε μία ταινία, που μπορούμε να διασχίσουμε σειριακά. Η κεφαλή της ταινίας αρχικά βρίσκεται στην αρχή της ταινίας και οι στοιχειώδεις πράξεις είναι:

1. *move_head*(*direction*) (*direction* $\mapsto$ *Left* ή *Right*)
2. *compare*(*key*) (Σύγκριση του *key* με το στοιχείο που βρίσκεται κάτω από την κεφαλή)
3. **if** "condition "**then** "jump "

Υποθέτουμε ότι οι στοιχειώδεις πράξεις απαιτούν $O(1)$ χρόνο.

- Ένα τετριμμένο κάτω φράγμα για την "αναζήτηση "με αυτό το μοντέλο είναι $\Omega(n)$.

  Εάν το στοιχείο *key* είναι το τελευταίο στοιχείο της λίστας, τότε θα πρέπει να εκτελέσουμε τουλάχιστον $n-1$ $move_head(right)$ εντολές.

Ερώτημα Εάν $\Omega(n)$ είναι ένα κάτω φράγμα για το πρόβλημα της αναζήτησης με το μοντέλο που δώθηκε παραπάνω, πώς είναι δυνατόν να έχουμε έναν $O(\log n)$ αλγόριθμο (δυαδική αναζήτηση);

Απάντηση Επειδή ο αλγόριθμος της δυαδικής αναζήτησης είναι σχεδιασμένος για μία τυχαίας προσπέλασης μηχανή (random access machine - RAM). Ένα διαφορετικό μοντέλο υπολογισμού!

> Όταν εκφράζουμε ένα κάτω φράγμα για κάποια προβλήματα, θα πρέπει επίσης να περιγράφουμε το μοντέλο υπολογισμού (model of computation) με βάση το οποίο καταλήξαμε σε αυτό το κάτω φράγμα.

Είσοδος - Έξοδος δεδομένων

- Όταν η είσοδος ενός προβλήματος έχει μέγεθος n, χρειαζόμαστε τουλάχιστον n βήματα για να διαβάσουμε την είσοδο.

  Παρατήρηση: Σε ορισμένες περιπτώσεις θεωρούμε ότι τα δεδομένα βρίσκονται ήδη αποθηκευμένα στην μνήμη.

- Όταν η έξοδος ενός προβλήματος αποτελείται από $f(n)$ στοιχειώδη στοιχεία, τότε χρειαζόμαστε τουλάχιστον $\Omega(f(n))$ χρόνο για να τα αναφέρουμε.

Παραδείγματα

Ταξινόμηση	$\Omega(n)$ κάτω φράγμα.
Πολλαπλασιασμός πινάκων	$\Omega(n^2)$ κάτω φράγμα.
Ελάχιστα μονοπάτια από κοινή αφετηρία	$\Omega(\|V\|)$ κάτω φράγμα.
Ελάχιστα μονοπάτια για κάθε ζεύγος κόμβων	$\Omega(\|V\|^2)$ κάτω φράγμα.
Ελάχιστο διασυνδετικό δένδρο	$\Omega(\|V\|)$ κάτω φράγμα.
κ.ο.κ. ...	

Εύρεση κάτω φράγματος για την μέθοδο του "αντιπάλου"

(Adversary lower bound)

Πρόβλημα Εύρεση του αθροίσματος n αριθμών.

Θεώρημα Η εύρεση του αθροίσματος n αριθμών απαιτεί $\Omega(n)$ χρόνο.

Απόδειξη (Διαίσθηση: ο αλγόριθμος θα πρέπει να εξετάσει όλα τα στοιχεία της εισόδου για να υπολογίσει το άθροισμα τους).

- Ας υποθέσουμε ότι υπάρχει ένας αλγόριθμος που δεν εξετάζει όλα τα στοιχεία.
- Κάποιος αντίπαλος (adversary) παρακολουθεί και όταν ο αλγόριθμος αποφασίσει, παρεμβαίνει και αλλάζει ένα στοιχείο το οποίο δεν εξετάστηκε.
- Εάν εκτελέσουμε ξανά τον αλγόριθμο, τότε θα λάβουμε ξανά το ίδιο αποτέλεσμα, καθώς ο αλγόριθμος δεν θα αντιληφθεί την αλλαγή. Στην πραγματικότητα όμως το αποτέλεσμα άλλαξε και έτσι ο αλγόριθμος είναι λαθεμένος!

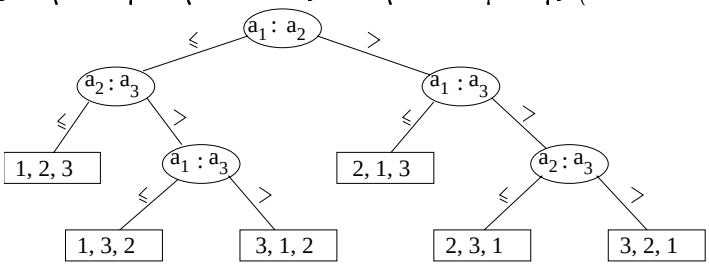
Άρα, κάθε σωστός αλγόριθμος για τον υπολογισμό του αθροίσματος n αριθμών εξετάζει κάθε στοιχείο και έτσι είναι $\Omega(n)$.

Κάτω φράγμα για το πρόβλημα της ταξινόμησης

Μοντέλο υπολογισμού

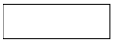
- Μηχανή τυχαίας προσπέλασης (RAM).
- Επιτρέπονται μόνο συγκρίσεις μεταξύ στοιχείων.

- Αλγόριθμοι για το παραπάνω μοντέλο ονομάζονται αλγόριθμοι ταξινόμησης μέσω συγκρίσεων *(comparison sorts)*.
- Μπορούμε να τους παραστήσουμε και ως δένδρα απόφασης *(decision trees)*:



Σημείωση

Εσωτερικός κόμβος. Το a συγκρίνεται με το b.



Φύλλο. Περιέχει την έξοδο του αλγορίθμου.

- Υπάρχουν $n!$ πιθανές μεταθέσεις της εισόδου. Άρα, το δένδρο απόφασης θα πρέπει να έχει τουλάχιστον $n!$ φύλλα.

  Το μήκος του μεγαλύτερου μονοπατιού από την ρίζα του δένδρου απόφασης πρός τα φύλλα παριστάνει το αριθμό των συγκρίσεων που εκτελεί ο αλγόριθμος ταξινόμησης στην χειρότερη περίπτωση.

Προτάσεις

1. Ένα δυαδικό δένδρο ύψους h έχει το πολύ 2^h φύλλα.

2. $n! > \left(\frac{n}{e}\right)^n$

   $e = 2.71828...$

   Ο παραπάνω τύπος απορρέει από την προσέγγιση του Stirling:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

Θεώρημα Κάθε δένδρο απόφασης που ταξινομεί n στοιχεία έχει ύψος $\Omega(n \log n)$.

Απόδειξη

Έστω ότι ένα τέτοιο δένδρο έχει ύψος h. Τότε,

$$n! \leq 2^h \qquad \text{(Πρόταση 1)}$$

$$\Longrightarrow \quad h \geq \log(n!)$$

$$\Longrightarrow \quad h \geq \log\left(\frac{n}{e}\right)^n \qquad \text{(Πρόταση 2)}$$

$$\Longrightarrow \quad h \geq \log\left(\frac{n^n}{e^n}\right) = n \log n - n \log e$$

$$\Longrightarrow \quad h = \Omega(n \log n)$$

□

Θεώρημα Ο αλγόριθμος *MergeSort* είναι ασυμπτωτικά βέλτιστος.

Απόδειξη Ο αλγόριθμος *MergeSort* χρησιμοποιεί το ίδιο μοντέλο υπολογισμού με αυτό που χρησιμοποιήσαμε παραπάνω για να λάβουμε το $\Omega(n \log n)$ κάτω φράγμα. Επιπλέον, η πολυπλοκότητα χρόνου του είναι $O(n \log n)$.

□

Αναγωγές (Reductions)

- Όταν δύο προβλήματα σχετίζονται, είναι δυνατόν να χρησιμοποιήσουμε έναν αλγόριθμο του ενός για να επιλύσουμε το άλλο.
- Έστω P, P' δύο προβλήματα και I ένα αυθαίρετο στιγμιότυπο του P. Επίσης, υποθέτουμε ότι γνωρίζουμε έναν αλγόριθμο A' που επιλύει το P'. Τότε, πιθανόν να μπορούμε να επιλύσουμε το P όπως φαίνεται παρακάτω:

$$
\begin{array}{ccc}
I \longrightarrow & \boxed{\text{μετατροπή στιγμιότυπου}} & \longrightarrow I' \\
\downarrow & & \downarrow \\
\boxed{\text{Αλγόριθμος } A \text{ για το } P} & & \boxed{\text{Αλγόριθμος } A' \text{ για το } P'} \\
\downarrow & & \downarrow \\
S \longleftarrow & \boxed{\text{μετατροπή λύσης}} & \longleftarrow S'
\end{array}
$$

- Εάν υπάρχει ένας αλγόριθμος μετατροπής του I στο I' και του S' στο S, τότε λέμε ότι το P ανάγεται (reduces) στο P' ή, ότι το P μετασχηματίζεται (transforms) στο P'.
- Συμβολισμός: $P \propto P'$.

Θεώρημα

Υποθέτουμε ότι $P \propto P'$ και ότι οι πολυπλοκότητες (στην χείριστη περίπτωση) για την μετατροπή του I στο I' και του S' στο S είναι $f_1(n)$ και $f_2(n)$, αντίστοιχα, όπου n είναι το μέγεθος του στιγμιότυπου I. Τότε,

1. Για κάθε αλγόριθμο για το P' με χείριστη πολυπλοκότητα $W'(n)$, υπάρχει ένας αλγόριθμος για το P με πολυπλοκότητα $f_1(n) + W'(n) + f_2(n)$.

2. Εάν $g(n)$ είναι ένα κάτω φράγμα για την πολυπλοκότητα του P, τότε $g(n) - f_1(n) - f_2(n)$ είναι ένα κάτω φράγμα για την πολυπλοκότητα του P'.

Απόδειξη (Βλέπε προηγούμενα σχήματα)

1. Κατασκευάζοντας τον αλγόριθμο
2. Με εις άτοπο επαγωγή.

□

Ένα κάτω φράγμα για το πρόβλημα κατασκευής δένδρων Huffman

Πρόβλημα Δοθέντος n βάρων $w_1,\ w_2, \ldots, w_n$ να βρεθεί ένα δένδρο Huffman γι αυτά.

- Ένα δένδρο Huffman ελαχιστοποιεί το άθροισμα $\displaystyle\sum_{c \text{ is leaf}} w(c) \cdot d_T(c)$.

Θεώρημα Η κατασκευή των δένδρων Huffman είναι πολυπλοκότητας $\Theta(n \log n)$.

Απόδειξη Θεωρούμε το παρακάτω πρόβλημα $Sort^\star$:

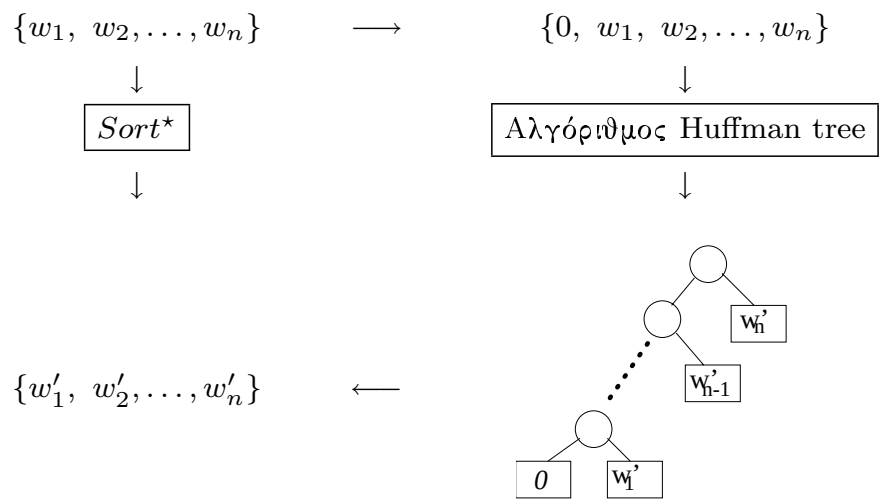
Δίνεται ένα σύνολο n θετικών αριθμών $w_1,\ w_2, \ldots, w_n$ για τους οποίους υπάρχει μία μετάθεση $w'_1,\ w'_2, \ldots, w'_n$ έτσι ώστε:

$$w'_i > \sum_{j=1}^{i-1} w'_j$$

για κάθε i. Να ταξινομηθούν οι αριθμοί αυτοί.

- Είναι εύκολο να δούμε ότι $Sort^\star = \Omega(n \log n)$. Κάθε δένδρο απόφασης θα πρέπει να έχει τουλάχιστον $n!$ φύλλα.

$Sort^\star \propto$ Huffman tree problem



Κατά συνέπεια, κάθε αλγόριθμος για την κατασκευή δένδρων Huffman, ο οποίος βασίζεται σε συγκρίσεις έχει πολυπλοκότητα $\Omega(n \log n)$.

Ταξινόμηση σε γραμμικό χρόνο

Ταξινόμηση μέσω απαρίθμησης (Counting-Sort)

Είσοδος: n ακέραιοι από το σύνολο $[1..k]$.

Έξοδος: Οι n ακέραιοι ταξινομημένοι σε αύξουσα σειρά.

Ιδέα: Αποφάσισε για κάθε στοιχείο x της εισόδου, τον αριθμό των στοιχείων τα οποία είναι $\leq x$.

- Ο αλγόριθμος Counting-Sort χρησιμοποιεί 3 πίνακες:

$A[1..n]$: Περιέχει τα στοιχεία της εισόδου.

$B[1..n]$: Περιέχει την ταξινομημένη έξοδο.

$C[1..k]$: Το $C[i]$ περιέχει τον αριθμό των στοιχείων που είναι $\leq i$.

$COUNTING-SORT(A, B, k)$

for $i \leftarrow 1$ **to** k
  do $C[i] \leftarrow 0$
for $j \leftarrow 1$ **to** $length[A]$
  do $C[A[j]] \leftarrow C[A[j]] + 1$
/* $C[i]$ now contains the number of */
/* elements equal to i (see (a)). */
for $i \leftarrow 2$ **to** k
  do $C[i] \leftarrow C[i] + C[i-1]$
/* $C[i]$ now contains the number of */
/* elements $\leq$ i (see (b)). */
for $j \leftarrow length[A]$ **downto** 1
  do $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

(a)

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6
C	2	0	2	3	0	1

(b)

	1	2	3	4	5	6
C	2	2	4	7	7	8

(c)

	1	2	3	4	5	6	7	8
B							4	

	1	2	3	4	5	6
C	2	2	4	6	7	8

(d)

	1	2	3	4	5	6	7	8
B		1					4	

	1	2	3	4	5	6
C	1	2	4	6	7	8

(e)

	1	2	3	4	5	6	7	8
B		1				4	4	

	1	2	3	4	5	6
C	1	2	4	5	7	8

(f)

	1	2	3	4	5	6	7	8
B	1	1	3	3	4	4	4	6

Πολυπλοκότητα χρόνου: $O(k+n)$ Εάν $k = O(n) \Longrightarrow O(n)$ αλγόριθμος.

Παρατήρηση Το αποτέλεσμα αυτό δεν αντικρούει το $\Omega(n \log n)$ κάτω φράγμα. Δεν χρησιμοποιούμε το μοντέλο συγκρίσεων. Στην πραγματικότητα, ο αλγόριθμος COUNTING-SORT() δεν έκανε καμία σύγκριση μεταξύ στοιχείων της εισόδου.

Ορισμός Ένας αλγόριθμος ταξινόμησης είναι *ευσταθής (stable)* εάν τα στοιχεία με την ίδια τιμή εμφανίζονται στην έξοδο με την ίδια σειρά που εμφανίστηκαν στην είσοδο.

Πρόταση Ο αλγόριθμος COUNTING-SORT() είναι ευσταθής.
(Η απόδειξη γίνεται με εξέταση του αλγορίθμου.)

Ταξινόμηση βάσης (αριθμητικού συστήματος) - Radix-Sort

Είσοδος: n στοιχεία τέτοια ώστε κάθε στοιχείο αποτελείται από d ψηφία όπου το πρώτο (από τα αριστερά) ψηφίο είναι το μικρότερης τάξης ψηφίο και το d-οστό ψηφίο είναι το μεγαλύτερης τάξης ψηφίο.

Έξοδος: Ένας ταξινομημένος πίνακας που περιέχει τα n στοιχεία σε αύξουσα σειρά.

Ιδέα: Ταξινόμησε τα στοιχεία πρώτα με βάση το *ελάχιστα σημαντικό ψηφίο.*

Παράδειγμα:

329		72**0**		7**2**0		**3**29
457		35**5**		3**2**9		**3**55
657		43**6**		4**3**6		**4**36
839	$\Longrightarrow$	45**7**	$\Longrightarrow$	8**3**9	$\Longrightarrow$	**4**57
436		65**7**		3**5**5		**6**57
720		32**9**		4**5**7		**7**20
355		83**9**		6**5**7		**8**39

$RADIX - SORT(A, d)$
for $i = 1$ **to** d
do use a stable sorting algorithm to sort array A on digit i.

Απόδειξη της ορθότητας (με επαγωγή ως προς τον αριθμό των θέσεων.)

- Υποθέτουμε ότι τα ψηφία χαμηλότερης τάξης είναι ταξινομημένα.
- Αποδεικνύουμε ότι η ταξινόμηση ως πρός το επόμενο ψηφίο αφήνει τον πίνακα ορθά ταξινομημένο.

- Έστω δύο τυχόντα στοιχεία:
  - Εάν τα δύο ψηφία σε αυτήν την θέση είναι διαφορετικά, τότε ταξινομώντας τα σε αυτή την θέση θα βρεθούν στην σωστή διάταξη. Τα μικρότερης τάξης ψηφία δεν επηρεάζουν το αποτέλεσμα.
  - Εάν τα δύο ψηφία σε αυτήν την θέση είναι ίσα, τότε τα στοιχεία αυτά βρίσκονται ήδη στην σωστή θέση. *Αφού χρησιμοποιούμε έναν ευσταθή αλγόριθμο ταξινόμησης, τα στοιχεία θα παραμείνουν στην σωστή σειρά.*

□

Ανάλυση

- Κάθε θέση (pass) απαιτεί $\Theta(n + k)$ χρόνο, όπου κάθε ψηφίο μπορεί να πάρει τιμές από το σύνολο $[1..k]$.

  $\Longrightarrow \Theta(d(n + k))$
- Εάν $k = O(n) \Longrightarrow \Theta(dn)$, χρησιμοποιώντας των αλγόριθμο COUNTING-SORT().

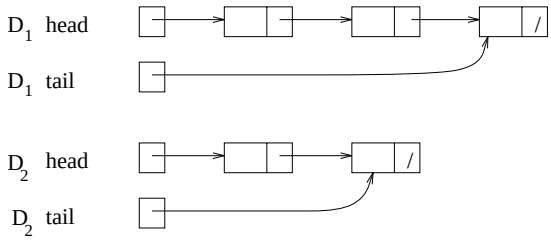
Sort-σε δοχεία(όμοιος με τον Radix-Sort)

Είσοδος: Μία συνδεδεμένη λίστα των n d-ψήφιων στοιχείων όπου κάθε ψηφίο ανήκει στο διάστημα $[1..k]$.

Έξοδος: Μία ταξινομημένη συνδεδεμένη λίστα των n στοιχείων σε αύξουσα σειρά.

Ιδέα:
- Να διαχωρίσουμε τα στοιχεία σε δοχεία *(bins)*.
- Ομοίως με τον αλγόριθμο Radix-Sort.

Απεικόνιση δομής δεδομένων ($k = 2$)



Παράδειγμα Ταξινόμηση της λίστας $< 36,\ 9,\ 0,\ 25,\ 1,\ 49,\ 64,\ 16,\ 81,\ 4 >$.

Bin	Contents	Bin	Contents
0	0	0	0, 1, 4, 9
1	1, 81	1	16
2		2	25
3		3	36
4	64, 4	4	49
5	25	5	
6	36, 16	6	64
7		7	
8		8	81
9	9, 49	9	
	d = 1		**d = 2**

$BIN - SORT(L, d, k)$ /* L είναι η λίστα των στοιχείων στην είσοδο. */
/* Κάθε στοιχείο αποτελείται από d ψηφία. */
/* Κάθε ψηφίο είναι στο σύνολο $[1..k]$. */

for $i = 1$ **to** d **do**

- Process list L sequentially and place each element of L at the top of the list $L_j,\ 1 \leq j \leq k$, where j is the ith digit of the element under consideration.
- Concatenate $L_1,\ L_2, \ldots,\ L_k$ into L.

Πολυπλοκότητα $\Theta(dn + dk)$

- Εάν $k = O(n)$ και d είναι σταθερά $\Longrightarrow \Theta(n)$.

Έλεγχος ακυκλικότητας γραφήματος (Graph Acyclicity Testing)

Είσοδος: Ένα κατευθυνόμενο γράφημα $G = (V, E)$.

Έξοδος: Περιέχει το G κύκλο;

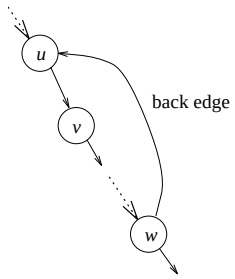
$ACYCLICITY(G)$

1. Εκτέλεσε τον αλγόριθμο $DFS(G)$ και κατηγοριοποίησε τις ακμές σε *ακμές δένδρου, ακμές προς τα εμπρός, ακμές προς τα πίσω και ακμές προς τα πλάγια.*
2. Εάν υπάρχει μία ακμή *προς τα πίσω* τότε το γράφημα G περιέχει κύκλο.

Πολυπλοκότητα: $O(V + E)$

Απόδειξη της ορθότητας

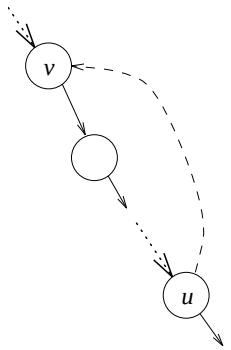
- Εάν το γράφημα G έχει μία ακμή προς τα πίσω τότε προφανώς περιέχει κύκλο.



$\Longrightarrow \langle u,\ v, \ldots,\ w,\ v\rangle$ είναι κύκλος.

Ισχυρισμός Εάν ένα κατευθυνόμενο γράφημα περιέχει κύκλο, τότε μία ακμή προς τα πίσω θα εμφανιστεί σε οποιοδήποτε DFS του γραφήματος.

- Έστω ότι το γράφημα έχει κύκλο.
- Έστω v ο κόμβος με τον μικρότερο χρόνο ανακάλυψης (discovery time) $d[v]$ που ανήκει στο κύκλο.
- Θεωρούμε μία ακμή $u \longrightarrow v$ σε κάποιο κύκλο που περιέχει τον v.



- Αφού ο κόμβος u ανήκει στον κύκλο, τότε ο u είναι απόγονος του v στο DFS δάσος. $\Longrightarrow$ Η ακμή $u \longrightarrow v$ δεν είναι ακμή προς τα πλάγια.
- Αφού $d[v] < d[u] \Longrightarrow$ Η ακμή $u \longrightarrow v$ δεν είναι ακμή δένδρου ή ακμή προς τα εμπρός.
  Άρα, η $u \longrightarrow v$ είναι μία ακμή προς τα πίσω.

□

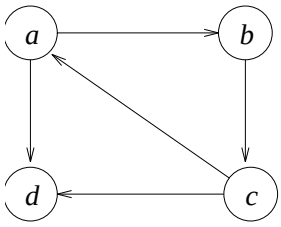
Ισχυρά συνδεδεμένα συστατικά

Ένα *ισχυρά συνδεδεμένο συστατικό (strongly connected component)* ενός κατευθυνόμενου γραφήματος $G = (V, E)$ είναι ένα μη επεκτάσιμο (maximal) σύνολο κόμβων $U \subseteq V$ τέτοιο ώστε για κάθε ζεύγος κόμβων u και v του U, υπάρχει ένα μονοπάτι από τον u προς τον v και ένα μονοπάτι από τον v στον u.

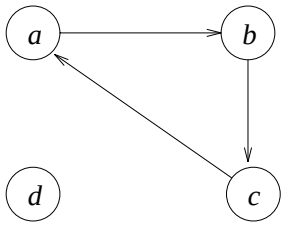
Είσοδος: Ένα κατευθυνόμενο γράφημα $G = (V, E)$.

Έξοδος: Τα ισχυρά συνδεδεμένα συστατικά του G.

Παράδειγμα



A directed graph

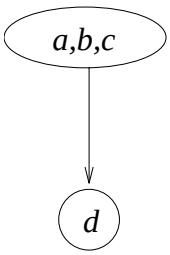


Its strongly connected components

- Κάθε κόμβος ενός κατευθυνόμενου γραφήματος ανήκει σε κάποιο ισχυρά συνδεδεμένο συστατικό, αλλά κάποιες ακμές του γραφήματος μπορεί να μην ανήκουν σε κανένα συστατικό. Τέτοιες ακμές ονομάζονται *μεταξύ συστατικών (cross-component) ακμές*.

- Μπορούμε να παραστήσουμε τις συνδέσεις μεταξύ των συστατικών κατασκευάζοντας ένα νέο γράφημα που ονομάζεται αναγόμενο γράφημα συστατικών *(reduced component graph)*.

Παράδειγμα (συνέχεια)



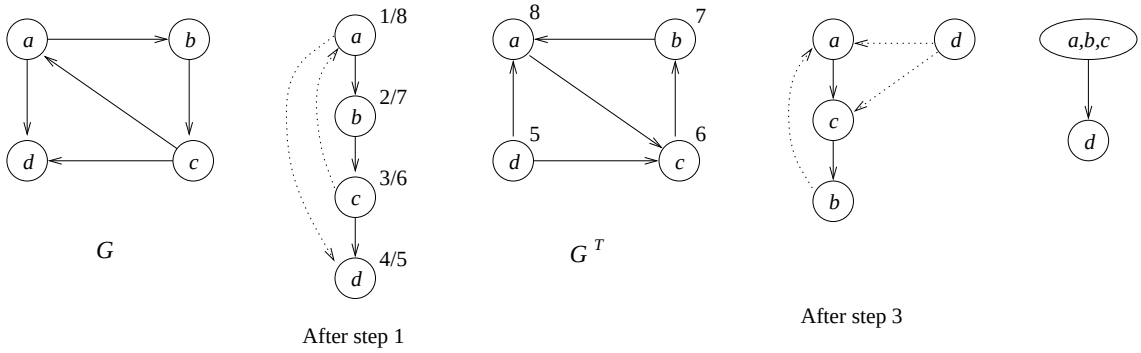
- Το αναγόμενο γράφημα είναι πάντοτε ακυκλικό.

STRONGLY_CONNECTED_COMPONENTS(G)

1. Εκτέλεσε τον αλγόριθμο $DFS(G)$ και υπολόγισε τους χρόνους ολοκλήρωσης της εξερεύνησης (finishing time) $f[u]$ για κάθε κόμβο u του G.
2. Υπολόγισε το ανάστροφο γράφημα G^T του G.
3. Εκτέλεσε τον αλγόριθμο $DFS(G^T)$, αλλά στην κεντρική ανακύκλωση του DFS θεώρησε τις κορυφές σε φθίνουσα τάξη του $f[]$ (όπως υπολογίστηκαν στο βήμα 1).
4. Οι κορυφές του κάθε δένδρου του DFS δάσους που παράχθηκε στο 3° βήμα αποτελούν τα ισχυρά συνδεδεμένα συστατικά του G.

Πολυπλοκότητα: $O(V + E)$

Παράδειγμα



Απόδειξη της ορθότητας

- Εάν v και w είναι δύο κόμβοι του ίδιου συστατικού, τότε υπάρχουν τα μονοπάτια:

- Πρέπει να δείξουμε ότι:

Οι v και w βρίσκονται στο ίδιο ισχυρά συνδεδεμένο συστατικό του G.	$\Longleftrightarrow$	Οι v και w βρίσκονται στο ίδιο δένδρο του DFS-δάσους του G^T.

'$\Longrightarrow$'

- Ας υποθέσουμε ότι στο γράφημα G^T ξεκινάμε την αναζήτηση από κάποια ρίζα x και φτάνουμε στον v. Τότε, ο κόμβος w θα καταλήξει στο ίδιο δένδρο (αφού ο w είναι προσπελάσιμος από τον v).

'$\Longleftarrow$' (Οι κόμβοι v και w βρίσκονται στο ίδιο δένδρο του DFS- δάσους του $G^T \Longrightarrow$ Οι v και w βρίσκονται στο ίδιο ισχυρά συνδεδεμένο συστατικό του G.)

- Έστω x η ρίζα του δένδρου στο DFS-δάσος του G^T στο οποίο ανήκουν οι v και w.
  $\Longrightarrow$ Στο G^T $\exists$ ένα μονοπάτι $x \longrightarrow v$ $\Longrightarrow$ Στο G $\exists$ ένα μονοπάτι $v \longrightarrow x$.

- Στο G^T, όταν ανακαλύφθηκε η x, η κορυφή v δεν είχε ακόμη επισκευθεί.

  $\Longrightarrow$ Στο G, $f[x] > f[v]$.

  $\Longrightarrow$ Στην εξερεύνηση του G, ανακαλύπτουμε τον v κατά την εξερεύνηση από τον x.

  $\Longrightarrow$ Η κορυφή v είναι απόγονος της x στο DFS-δάσος του G.

  $\Longrightarrow$ Στο G $\exists$ ένα μονοπάτι $x \longrightarrow v$. $\Longrightarrow$ Οι x και v βρίσκονται στο ίδιο ισχυρά συνδεδεμένο συστατικό.

- Όμοια αποδεικνύουμε ότι οι x και w βρίσκονται στο ίδιο ισχυρά συνδεδεμένο συστατικό.

$\square$

Ελάχιστα διασυνδετικά δένδρα: Ο αλγόριθμος του Kruskal

Είσοδος: Ένα συνδεδεμένο μη κατευθυνόμενο γράφημα $G = (V, E)$ με μία συνάρτηση βάρους $w : E \to \mathcal{R}$.

Έξοδος: Ένα ελάχιστο διασυνδετικό δένδρο για το G.

- Υπενθυμίζεται ο **γενικός** αλγόριθμος:

Generic_MST(G, w)
$A = \emptyset$
while A does not form a spanning tree **do**
  Find an edge (u, v) that is safe for A.
  $A = A \cup \{(u, v)\}$
return A

Σταθερά (invariant) του αλγορίθμου: "Το A είναι πάντοτε ένα υποσύνολο κάποιου ελάχιστου διασυνδετικού δένδρου".

- Μια ακμή (u, v) είναι μία *ασφαλής ακμή* για το A εάν μπορεί να προστεθεί στο A χωρίς να διαταραχθεί η σταθερά του αλγορίθμου.

MST_Kruskal(G, w)

1. $A = \emptyset$
2. **for** each vertex $v \in V(G)$
3. **do** $MAKE_SET(v)$
4. Sort the the edges of E in non-decreasing weight w.
5. **for** each edge $(u, v) \in E$, in order by non-decreasing weight,
6. **do if** $FIND_SET(u) \neq FIND_SET(v)$
7. **then** $A = A \cup \{(u, v)\}$
8. $UNION(u, v)$
9. **return** A

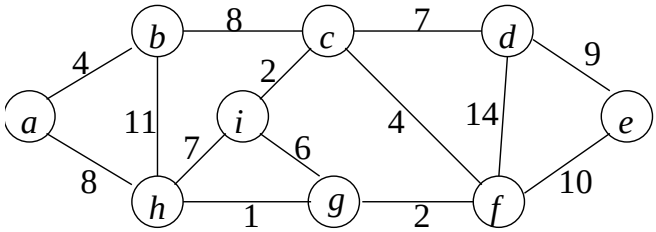
- Χρησιμοποιήσαμε μια ΑΔΔ που υποστηρίζει πράξεις σε ξένα μεταξύ τους (disjoint) σύνολα.
  - $MAKE_SET(v)$: Δημιουργεί ένα νέο σύνολο που περιέχει το στοιχείο v.
  - $FIND_SET(v)$: Επιστρέφει το σύνολο στο οποίο ανήκει το v.
  - $UNION(u, v)$: Δημιουργεί ένα νέο σύνολο που είναι η ένωση των: $FIND_SET(u)$ και $FIND_SET(v)$.

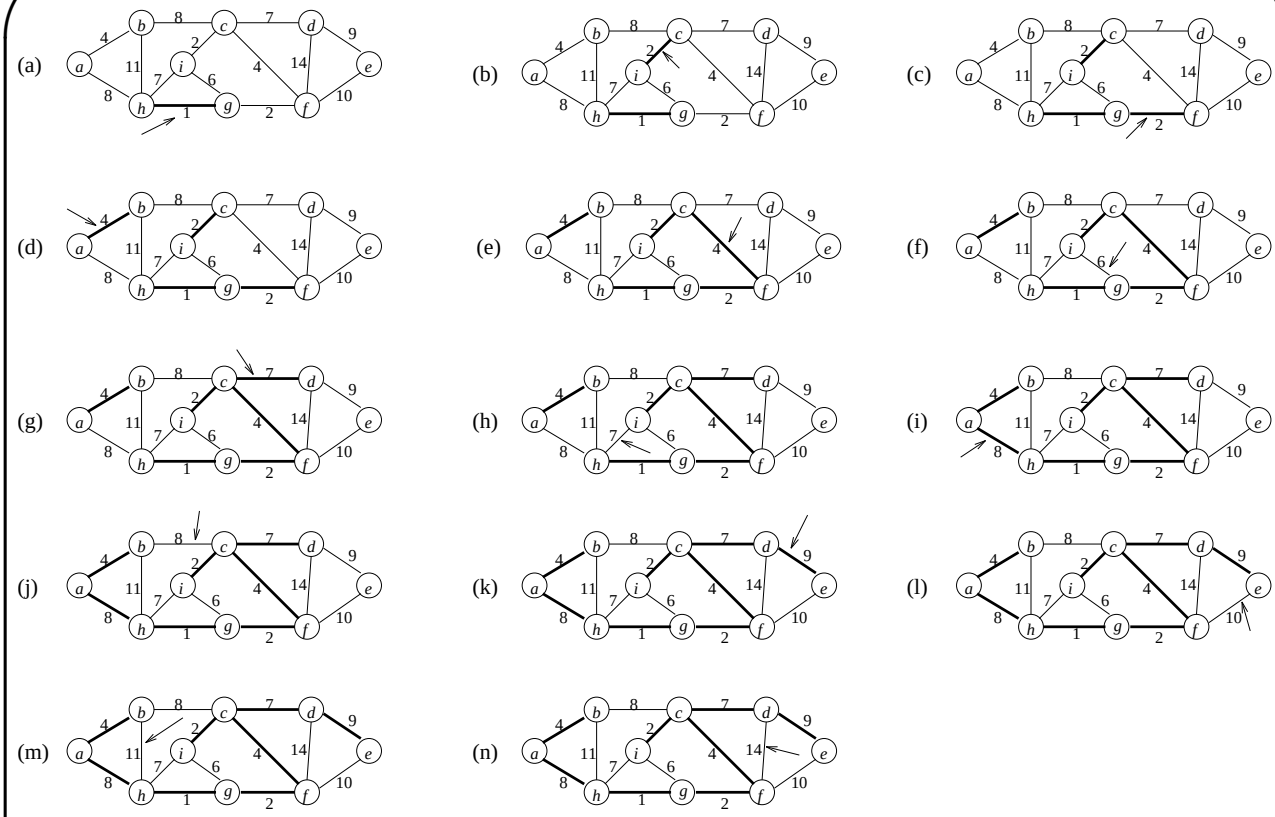
- **Ανάλυση** Βήματα 2-3: $O(V)$
  Βήμα 4: $O(E \log E)$
  Βήματα 5-8: $O(E + V \log V)$
  ΣΥΝΟΛΙΚΑ: $O(E \log E)$

Παράδειγμα

Το E είναι ταξινομημένο σε αύξουσα τάξη:

(h, g)	1
(i, c)	2
(g, f)	2
(a, b)	4
(c, f)	4
(i, g)	6
(c, d)	7
(h, i)	7
(a, h)	8
(b, c)	8
(d, e)	9
(f, e)	10
(b, h)	11
(d, f)	14



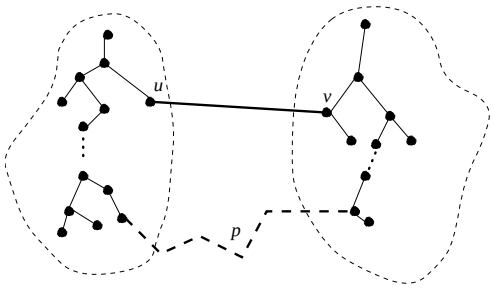


Απόδειξη της ορθότητας του αλγορίθμου του Kruskal

Θεώρημα Εάν αποφασίσουμε να προσθέσουμε την ακμή (u, v) στο A, τότε η (u, v) είναι ασφαλής ακμή.

Απόδειξη Προσθέτουμε την ακμή (u, v) στο A εάν και μόνο εάν οι u και v ανήκουν σε διαφορετικά δένδρα του G (που έχουν ήδη σχηματιστεί από την εκτέλεση του αλγορίθμου του Kruskal).

- Εάν η (u, v) δεν είναι ασφαλής, τότε δεν ανήκει στο ελάχιστο διασυνδετικό δένδρο. Άρα, υπάρχει ένα μονοπάτι p από το δένδρο που περιέχει την u προς το δένδρο που περιέχει την v το οποίο ανήκει στο ελάχιστο διασυνδετικό δένδρο του G.



- Αφού το μονοπάτι p αποτελείται από ακμές με βάρη μεγαλύτερα ή ίσα του $w(u, v)$, μπορούμε να αντικαταστήσουμε οποιαδήποτε ακμή του p με την (u, v) και να λάβουμε ένα ελάχιστο διασυνδετικό δένδρο με το ίδιο βάρος του "υποθετικού "ελάχιστου διασυνδετικού δένδρου του G.

  Άρα, η ακμή (u, v) είναι ασφαλής.

□

Ελάχιστα μονοπάτια από κοινή αφετηρία

Είσοδος: • Ένα κατευθυνόμενο, με βάρη, γράφημα $G = (V, E)$ και μία συνάρτηση βάρους $w : E \to \mathcal{R}$ που αντιστοιχεί σε κάθε ακμή ένα πραγματικό αριθμό (βάρος).
• Ένας κόμβος αφετηρία s.

Έξοδος: Τα ελάχιστα μονοπάτια από το s προς κάθε άλλο κόμβο του G.

Επανεκτίμηση - Relaxation

- Για κάθε κορυφή $v \in V$, το $d[v]$ είναι ένα άνω φράγμα για το βάρος του ελάχιστου μονοπατιού από την αφετηρίαs στην v.

  $d[v] = \text{shortest_path_estimate}$

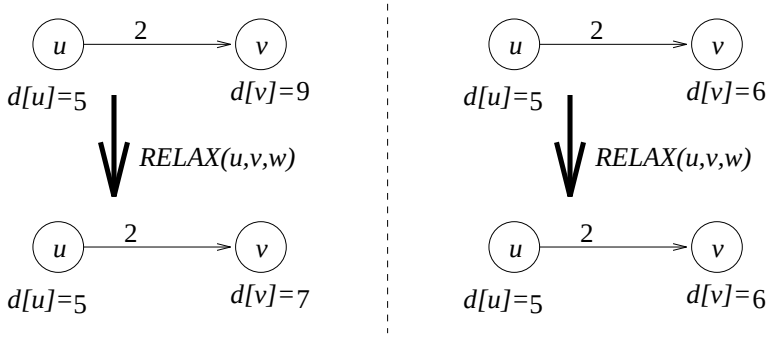
- Η αρχικοποίηση του $d[v]$, $v \in V$, γίνεται ως εξής:

  INITIALIZE_SINGLE_SOURCE(G, s)
  for each vertex $v \in V$
  do $d[v] \leftarrow +\infty$
  $\pi[v] \leftarrow NIL$
  $d[s] \leftarrow 0$

Κατά *την επανεκτίμηση της ακμής* (u, v), ελέγχουμε εάν μπορούμε να βελτιώσουμε το ελάχιστο μονοπάτι από τον s στον v πηγαίνοντας μέσω της ακμής (u, v).

RELAX(u, v, w)
if $d[v] > d[u] + w(u, v)$
  then $d[v] \leftarrow d[u] + w(u, v)$
    $\pi[v] \leftarrow u$

Παράδειγμα



- Αμέσως μετά την επανεκτίμηση της ακμής (u, v) μέσω της $RELAX(u, v, w)$, ισχύει: $d[v] \leq d[u] + w(u, v)$.

- Ο αλγόριθμος του Dijkstra μπορεί να γραφεί και ως:

Dijkstra(G, w, s)
1. ***INITIALIZE_SINGLE_SOURCE***(G, s)
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. **while** $Q \neq \emptyset$
5. **do** $u = extract_min(Q)$
6. $S \leftarrow S \cup \{u\}$
7. **for** each vertex $v \in Adj[u]$
8. **do** ***RELAX***(u, v, w)

Παρατήρηση 1: Στον αλγόριθμος του Dijkstra, κάθε ακμή επανεκτιμάτε *ακριβώς μία φορά*.

Παρατήρηση 2: Ο αλγόριθμος του Dijkstra δουλεύει σωστά μόνο για γραφήματα με μη αρνητικά βάρη στις ακμές.

Ερώτημα: Πώς αντιμετωπίζουμε τα αρνητικά βάρη στις ακμές;

Ο αλγόριθμος των BELLMAN-FORD

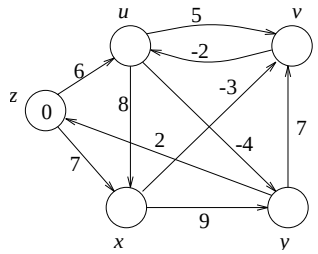
Είσοδος:
- Ένα κατευθυνόμενο, με βάρη, γράφημα $G = (V, E)$ και μία συνάρτηση βάρους $w : E \to \mathcal{R}$ που αντιστοιχεί σε κάθε ακμή ένα πραγματικό αριθμό (βάρος).
- Ένας κόμβος αφετηρία s.

Έξοδος:
- Μία τιμή τύπου boolean που θα δηλώνει την ύπαρξη κύκλου αρνητικού βάρους προσπελάσιμου από την αφετηρία s.
- Εάν δεν υπάρχουν αρνητικοί κύκλοι, τα ελάχιστα μονοπάτια (και τα βάρη τους) από τον s προς κάθε άλλο κόμβο του G.

BELLMAN_FORD(G, w, s)
1. ***INITIALIZE_SINGLE_SOURCE***(G, s)
2. **for** $i = 1$ **to** $|V(G)| - 1$
3. **do for** each edge $(u, v) \in E(G)$
4. **do** ***RELAX***(u, v, w)
5. **for** each edge $(u, v) \in E(G)$
6. **do if** $d[v] > d[u] + w(u, v)$
7. **then return** $FALSE$
8. **return** $TRUE$

Πολυπλοκότητα $O(VE)$ (Κάθε ακμή επανεκτιμάται ακριβώς $|V| - 1$ φορές.)

Παράδειγμα



Επανεκτιμούμε τις κορυφές με την ακόλουθη σειρά:

(z, u)
(z, x)
(u, v)
(u, y)
(u, x)
(v, u)
(x, v)
(x, y)
(y, z)
(y, v)

- Η σειρά της επανεκτίμησης των κόμβων είναι αυθαίρετη. Σε κάθε "πέρασμα"μπορεί να επανεκτιμήσουμε τις ακμές σε διαφορετική σειρά.

Απόδειξη της ορθότητας του αλγορίθμου των BELLMAN-FORD

- Έστω v ένας κόμβος προσπελάσιμος από την αφετηρία s.
- Έστω $p = \langle s, v_1, v_2, \ldots, v_{k-1}, v \rangle$ το ελάχιστο μονοπάτι μήκους k από την s στον v, $k \leq |V| - 1$.
- Θα αποδείξουμε με επαγωγή ότι $d[v_i] = \delta(s, v_i)$ ύστερα από το i-οστό "πέρασμα" των ακμών.

Βάση $d[s] = 0$.

Επαγωγικό βήμα

◇ **Υποθέτουμε** ότι $d[v_{i-1}] = \delta(s, v_{i-1})$ ύστερά από το $(i-1)$-οστό πέρασμα.

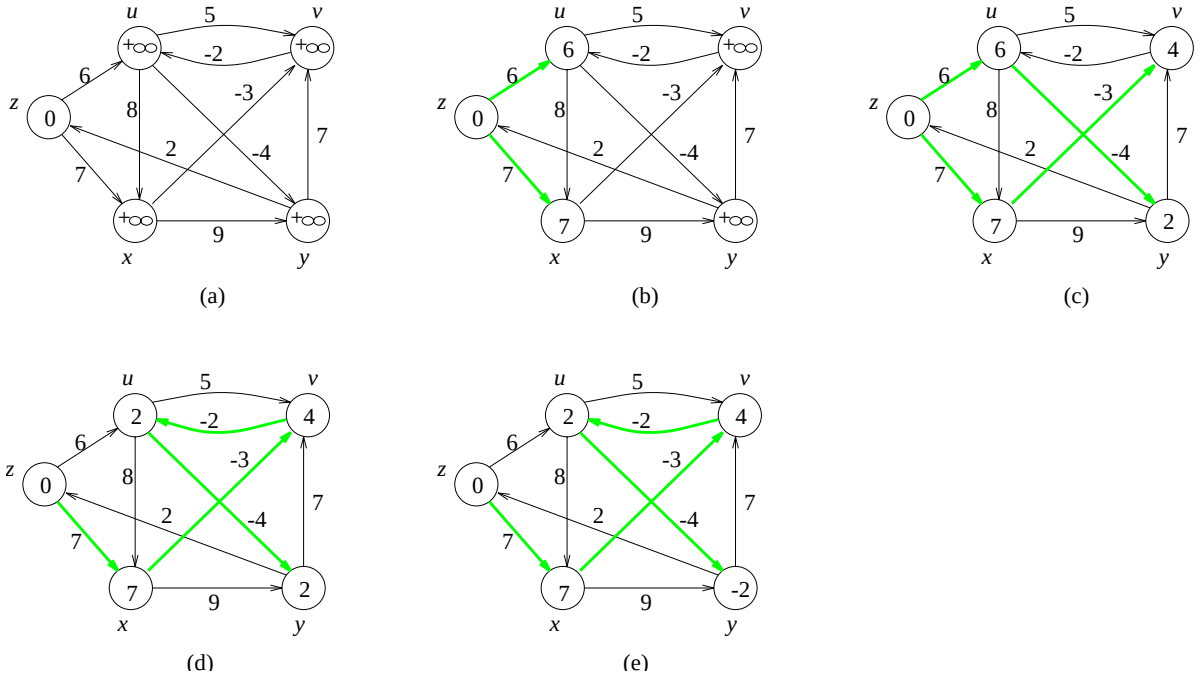
◇ **Θα αποδείξουμε** ότι $d[v_i] = \delta(s, v_i)$ ύστερα από το i-οστό πέρασμα.

Αυτό έπεται από τα παρακάτω:

1. Επανεκτιμούμε την ακμή (v_{i-1}, v_i) κατά το i-οστό πέρασμα.
2. Τα υπό - μονοπάτια των ελάχιστων μονοπατιών είναι ελάχιστα μονοπάτια.

□

Παράδειγμα Οι ακμές επανεκτιμούνται με λεξικογραφική σειρά.



Ελάχιστα μονοπάτια από κοινής αφετηρίας σε DAGs

(DAGs: Κατευθυνόμενα ακυκλικά γραφήματα)

DAGs_Shortest_Paths(G, w, s)

1. Topologically sort the vertices of G
2. ***INITIALIZE_SINGLE_SOURCE***(G, s)
3. **for** each vertex u (in the topologically sorted order)
4. **do for** each vertex $v \in Adj[u]$
5. **do** $RELAX(u, v)$

Πολυπλοκότητα: $O(V + E)$