Κεφάλαιο 6

Αλγόριθμοι Γράφων

6.1 Διάσχιση γράφων

6.1.1 Γενικά

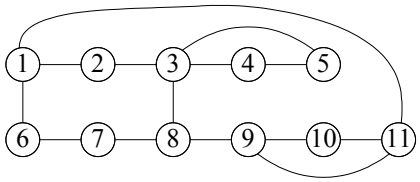
Οι τεχνικές διάσχισης γράφων μας βοηθούν στο να επισκεπτόμαστε συστηματικά τους κόμβους ενός γράφου G(V,E) έτσι ώστε να δίνουμε γρήγορα απαντήσεις σε προβλήματα όπως τα παρακάτω (*G.A.P., Graph Accessibility Problem*):

- Στο γράφο G υπάρχει μονοπάτι από τον κόμβο v στον κόμβο u;
- Ο γράφος G είναι ακυκλικός;
- Ποιες είναι οι συνεκτικές συνιστώσες (*strong components*) του γράφου G; Δηλαδή ζητάμε να βρεθούν όλα τα υποσύνολα του συνόλου V που είναι τέτοια ώστε όλοι οι κόμβοι που ανήκουν σε αυτά να είναι μεταξύ τους συνδεδεμένοι.
- Ποια είναι τα σημεία σύνδεσης (*articulation points*) του γράφου G; Δηλαδή ζητάμε όλους εκείνους τους κόμβους του γράφου, που αν αφαιρεθούν μαζί με τις προσπίπτουσες ακμές τους, χωρίζουν το γράφο σε δύο ή περισσότερες συνεκτικές συνιστώσες.

6.1.2 Αναζήτηση κατά πλάτος (Breadth First Search)

Στην αναζήτηση κατά πλάτος από κάθε κόμβο v που επισκεπτόμαστε, αναζητούμε κόμβους όσο το δυνατόν κατά πλάτος, δηλαδή αμέσως μετά την επίσκεψη του κόμβου v επισκεπτόμαστε όλους τους γειτονικούς του κόμβους. Έτσι λοιπόν, κατά τη διάρκεια της διαδικασίας αυτής μπορούμε να ξεχωρίσουμε δύο κατηγορίες κόμβων:

- κόμβους που έχουμε επισκεφθεί (*visited nodes*)
- κόμβους που έχουμε επισκεφθεί αυτούς και όλους τους γειτονικούς τους (*explored nodes*)



Σχήμα 6.1: Παράδειγμα γράφου στον οποίο θα γίνει αναζήτηση

Μετά το τέλος της διαδικασίας της αναζήτησης κατά πλάτος στο γράφο G, προκύπτει το συνδετικό δέντρο με προτεραιότητα πλάτους (*breadth first spanning tree*) του γράφου G. Η υλοποίηση της διαδικασίας φαίνεται στον αλγόριθμο 6.1.

Αλγόριθμος 6.1 Αναζήτηση κατά πλάτος

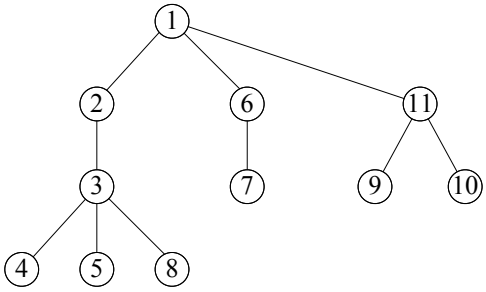
```
procedure bfs(v:vertex);
begin
  initialize queue with v; visited[v]:=true;
  repeat dequeue(u);
    for all vertices w adjacent to u do
      if not visited[w] then
        begin visited[w] := true; enqueue(w) end
  until queue is empty
end
```

Παράδειγμα 6.1. Έστω ο γράφος που φαίνεται στο σχήμα 6.1. Αν καλέσουμε τη διαδικασία **bfs** με **bfs**(1) έχουμε τα βήματα που φαίνονται στο πίνακα 6.1. Το συνδετικό δέντρο με προτεραιότητα πλάτους, είναι αυτό που φαίνεται στο σχήμα 6.2. Οποιαδήποτε υλοποίηση και να χρησιμοποιηθεί, ο αλγόριθμος 6.1 χρειάζεται τον επιπλέον χώρο μιας ουράς μήκους n και ένα πίνακα επίσης μήκους n. Συνεπώς ο χώρος που χρειάζεται είναι της τάξης $O(n)$. Αν ο γράφος υλοποιηθεί με πίνακα γειτονίας, ο χρόνος που χρειάζεται για να διατρέξουμε τον πίνακα είναι $O(n^2)$, ενώ αν έχουμε λίστες γειτονικών κορυφών ο χρόνος είναι $O(n + e)$.

Για να βρούμε όλες τις συνεκτικές συνιστώσες του γράφου G διασχίζουμε το γράφο με προτεραιότητα πλάτους (*breadth first traversing*). Η υλοποίηση αυτής της διάσχισης φαίνεται στον αλγόριθμο 6.2. Στον αλγόριθμο αυτό είναι φανερό ότι αν ο γράφος είναι συνεκτικός, τότε με την πρώτη κλήση της bfs θα επισκεφθούμε όλους τους κόμβους του.

6.1.3 Αναζήτηση κατά βάθος (Depth First Search)

Στην αναζήτηση κατά βάθος από κάθε κόμβο v που επισκεπτόμαστε αναζητάμε άλλους κόμβους όσο το δυνατό βαθύτερα. Μετά το τέλος της διαδικασίας προκύπτει το συνδετικό δέντρο με προτεραιότητα βάθους (*depth first spanning tree*). Η υλοποίηση της διαδικασίας φαίνεται στον αλγόριθμο 6.3, ενώ το συνδετικό δέντρο με προτεραιότητα βάθους είναι αυτό που φαίνεται στο σχήμα 6.3.



Σχήμα 6.2: Αναζήτηση κατά πλάτος στον γράφο

visited nodes	Queue
1	1
2	2
6	2-6
11	2-6-11
3	6-11-3
7	11-3-7
9	3-7-9
10	3-7-9-10
4	7-9-10-4
5	7-9-10-4-5
8	7-9-10-4-5-8
-	9-10-4-5-8
-	10-4-5-8
-	4-5-8
-	5-8
-	8
-	Ø

Πίνακας 6.1: Εκτέλεση αναζήτησης κατά πλάτος

Ο Αλγόριθμος **dft**, που βρίσκει όλες τις συνεκτικές συνιστώσες ενός γράφου, είναι ίδιος με τον **bft**, με την διαφορά ότι αντικαθιστούμε την κλήση της **bfs** με την κλήση της **dfs**.

D-Search Ένας ακόμη αλγόριθμος διάσχισης είναι ο **D-search** που περιγράφεται ακριβώς όπως ο **bfs** εκτός από το ότι χρησιμοποιεί μια στοίβα (*stack*) αντί ουράς. Έτσι συμπεριφέρεται σαν ένα κράμα **bfs** και **dfs** (θυμηθείτε σε αυτό το σημείο ότι ο **dfs** μπορεί να υλοποιηθεί και με επαναληπτικό αλγόριθμο, που χρησιμοποιεί στοίβα).

6.2 Το πρόβλημα των συντομότερων μονοπατιών, single source shortest paths problem

Μας δίνεται ένας κατευθυνόμενος γράφος με (θετικά) βάρη και ένας κόμβος του σαν πηγή (*source*). Ζητάμε να βρούμε τα συντομότερα μονοπάτια από την πηγή προς όλες τις άλλες κορυφές του γράφου. Το πρόβλημα της εύρεσης του συντομότερου μονοπατιού από την πηγή προς ένα συγκεκριμένο κόμβο του γράφου είναι εξίσου δύσκολο.

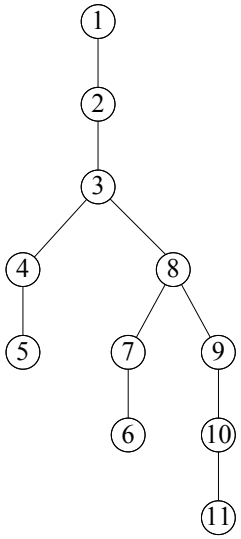
Ο αλγόριθμος που θα περιγράψουμε στηρίζεται στη μέθοδο που αναπτύχθηκε από τον *Dijkstra* (1959). Αριθμούμε τις κορυφές του γράφου και σαν πηγή λαμβάνουμε την κορυφή 1. Έστω V το σύνολο των κορυφών του γράφου και S το σύνολο των κορυφών για τις οποίες το συντομότερο μονοπάτι από την πηγή είναι ήδη γνωστό. Χρησιμοποιούμε τους πίνακες C, D, P. Με $C[i, j]$ συμβολίζουμε το κόστος μετάβασης από την κορυφή i στην κορυφή j, δηλαδή το κόστος μετάβασης της ακμής $i \rightarrow j$. Αν η ακμή $i \rightarrow j$ δεν υπάρχει, τότε $C[i, j] = \infty$, δηλαδή μια τιμή μεγαλύτερη από οποιοδήποτε πραγματικό κόστος. Το $D[v]$ περιέχει σε κάθε βήμα το κό-

Αλγόριθμος 6.2 Εύρεση συνεκτικών συνιστωσών

```
procedure bft(G);
begin
  initialize visited with false;
  for i:=1 to n do
    if not visited[i] then bfs(i)
end
```

Αλγόριθμος 6.3 Αναζήτηση κατά βάθος

```
procedure dfs(v:vertex);
begin
  visited[v]:=true;
  for all vertices u adjacent to v do
    if not visited[u] then dfs(u)
end
```



Σχήμα 6.3: Αναζήτηση κατά βάθος στον γράφο

στος του τρέχοντος συντομότερου από την πηγή στον κόμβο v, ενώ το $P[v]$ περιέχει την αμέσως προηγούμενη κορυφή από την v στο συντομότερο μονοπάτι. Ο αλγόριθμος ξεκινάει με $S = 1$ και προχωράει σε βήματα κατά τα οποία επιλέγει μια κορυφή w έτσι ώστε το $D[w]$ να είναι το ελάχιστο. Στη συνέχεια, αφαιρεί την κορυφή w από το V και την προσθέτει στο S. Έπειτα επαναϋπολογίζει τον πίνακα D για κάθε κορυφή v που ανήκει στο $V - S$ ως εξής:

$$D[v] := min(D[v], D[w] + C[w, v])$$

Αν ισχύει ότι $D[v] > D[w]+C[w, v]$ τότε θέτει $P[v] = w$. Ο αλγόριθμος σταματά όταν $V-S = \emptyset$. Η υλοποίηση της μεθόδου του *Dijkstra* φαίνεται στον αλγόριθμο 6.4.

Από τον αλγόριθμο 6.4 φαίνεται ότι αρχικά το σύνολο $V - S$ έχει $n - 1$ στοιχεία άρα για να βρεθεί το ελάχιστο θα πρέπει να γίνουν $n - 2$ συγκρίσεις. Ακολούθως το $V - S$ θα έχει $n - 2$ στοιχεία άρα θα χρειάζονται $n - 3$ συγκρίσεις κ.ο.κ. Συνολικά λοιπόν οι συγκρίσεις είναι:

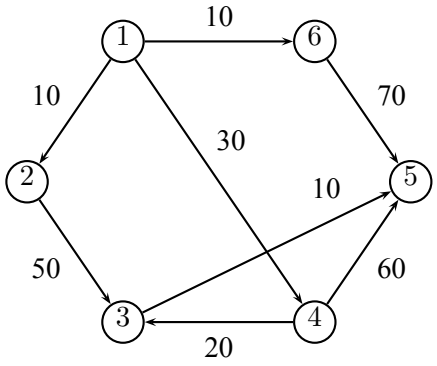
$$\sum_{i=1}^{n-2} i = \frac{(n-1)(n-2)}{2} = O(n^2)$$

Παράδειγμα 6.2. Έστω ότι μας δίνεται ο γράφος που φαίνεται στο σχήμα 6.4. Η εκτέλεση του αλγορίθμου φαίνεται στον πίνακα 6.2. Για να τυπώσουμε το συντομότερο μονοπάτι από ένα κόμβο σε ένα άλλο ανιχνεύουμε τους προκατόχους του τελευταίου κόμβου χρησιμοποιώντας τον πίνακα P. Αν για παράδειγμα θέλουμε το συντομότερο μονοπάτι από τον κόμβο 1 στον κόμβο 5 βλέπουμε ότι $P[5] = 3$, δηλαδή ο 3 είναι ο προκάτοχος του 5, $P[3] = 4$ δηλαδή ο 4 είναι προκάτοχος του 3. 'Αρα το συντομότερο μονοπάτι από τον κόμβο 1 στον 5 είναι το $1 \to 4 \to 3 \to 5$.

Αλγόριθμος 6.4 Εύρεση συντομότερων μονοπατιών (single source shortest paths) με τη μεθόδο του Dijkstra

```
procedure Dijkstra;
begin (* Αρχικοποίηση*)
  for i:=2 to n do begin D[i]:=cost[1,i]; P[i]:=1 end;
  S := {1};

  for i:=2 to n − 1 do
  begin
    Select w from V − S such that D[w] is minimum;
    S := S + {w};
    for all v in V − S do if D[v] > D[w] + C[w,v] then
        begin P[v] := w; D[v] := D[w] + C[w,v] end
  end
end
```



Σχήμα 6.4: Κατευθυνόμενος γράφος με βάρη

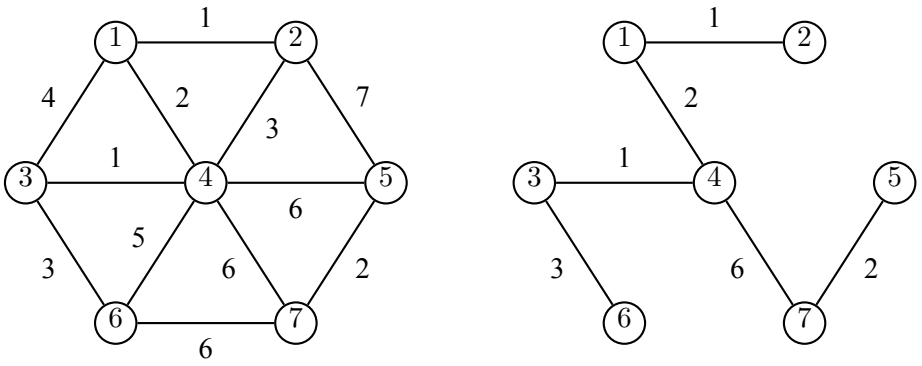
6.3 Ελάχιστο συνδετικό δέντρο (MST)

Στο πρόβλημα αυτό έχουμε ένα συνεκτικό γράφο $G(V, E)$ και ένα πίνακα κόστους $c_{|V|\times|V|}$ έτσι ώστε για κάθε ακμή $(u, v) \in E$ να υπάρχει το αντίστοιχο (βάρος) κόστος $c[v, u] > 0$. Ζητάμε ένα δέντρο $T(V, E')$ για το οποίο ισχύει:

$$\begin{cases} E' & \subseteq E \\ \& & \sum_{(u,v)\in E'} c[u,v] = MINIMUM \end{cases}$$

δηλαδή ένα δέντρο-υπογράφο που συνδέει όλες τις κορυφές του V και το συνολικό κόστος των ακμών του είναι το ελάχιστο δυνατό. Το δέντρο αυτό λέγεται ελαχίστου κόστους συνδετικό δέντρο του γράφου G (σχήμα 6.5).

Τα ελάχιστα συνδετικά δέντρα βρίσκουν εφαρμογή στο σχεδιασμό τηλεπικοινωνιακών δικτύων. Οι κόμβοι του γράφου αναπαριστούν πόλεις και οι ακμές αναπαριστούν πιθανούς τηλεπικοινωνιακούς συνδέσμους μεταξύ των πόλεων. Το κόστος κάθε ακμής είναι το κόστος κατασκευής του



Σχήμα 6.5: Ο γράφος G και ένα ελαχίστου κόστους συνδετικό δένδρο αυτού

συγκεκριμένου συνδέσμου για το τελικό δίκτυο. Το ελάχιστο συνδετικό δέντρο αναπαριστά ένα τηλεπικοινωνιακό δίκτυο που ενώνει όλες τις πόλεις και έχει το ελάχιστο κόστος κατασκευής. Η άπληστη μέθοδος αρχίζει με την κενή λύση και κατασκευάζει το δέντρο ακμή-ακμή (*edge-by-edge*) ακολουθώντας είτε το κριτήριο του *Prim* είτε το κριτήριο του *Kruskal*, είτε άλλα κριτήρια.

Κριτήριο του Prim: Προσθέτουμε κάθε φορά την ακμή που έχει το ελάχιστο κόστος έτσι ώστε ο νέος υπογράφος να παραμένει δέντρο. Αρχικοποιούμε με ένα δέντρο που αποτελείται από έναν μόνο κόμβο (π.χ. τον κόμβο 1). Η υλοποίηση που χρησιμοποιεί αυτό το κριτήριο φαίνεται στον αλγόριθμο 6.5 ενώ στο σχήμα 6.6 φαίνεται η ακολουθία των ακμών που προστίθενται στο ελάχιστο συνδετικό δέντρο μέχρι αυτό να πάρει την τελική μορφή του.

Κριτήριο του Kruskal: Προσθέτουμε κάθε φορά την ακμή ελαχίστου κόστους έτσι ώστε ο νέος υπογράφος να μην έχει κύκλους. Σχηματίζουμε έτσι ένα δάσος το οποίο τελικά γίνεται δέντρο. Η υλοποίηση που χρησιμοποιεί αυτό το κριτήριο φαίνεται στον αλγόριθμο 6.6 ενώ στο σχήμα 6.7 φαίνεται η ακολουθία των ακμών που προστίθενται στο ελάχιστο συνδετικό δέντρο μέχρι αυτό να πάρει την τελική μορφή του.

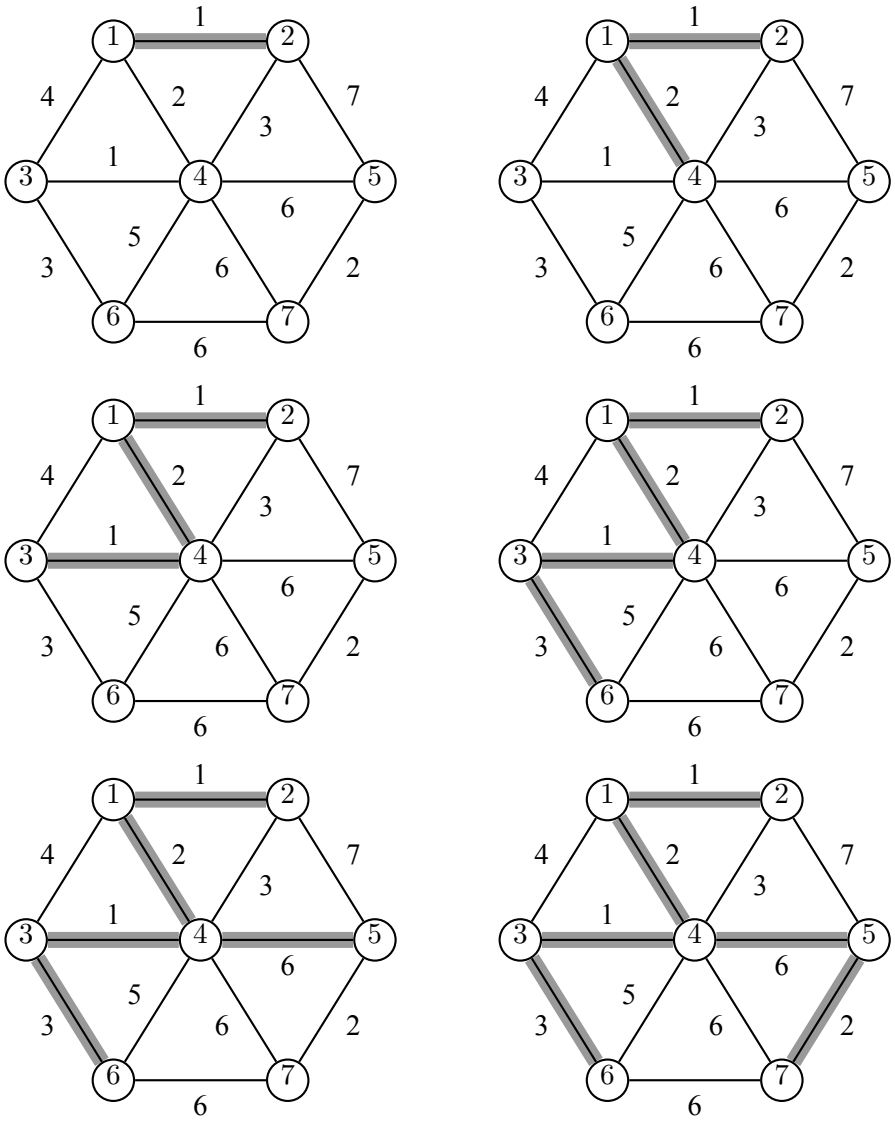
Αν $n = |V|$ και $e = |E|$ τότε ο αλγόριθμος 6.5 (με το κριτήριο του Prim) έχει πολυπλοκότητα $O(n^2)$ ενώ ο αλγόριθμος 6.6 (με το κριτήριο του Kruskal) έχει πολυπλοκότητα $O(e \log e)$. Για

Βήμα	S	w	D					P				
			2	3	4	5	6	2	3	4	5	6
-	{1}	-	10	∞	30	∞	10	1	1	1	1	1
2	{1,2}	2		60	30	∞	10		2			
3	{1,2,6}	6		60	30	80					6	
4	{1,2,6,4}	4		50		80			4			
5	{1,2,6,4,3}	3				60					3	
6	{1,2,6,4,3,5}	5										

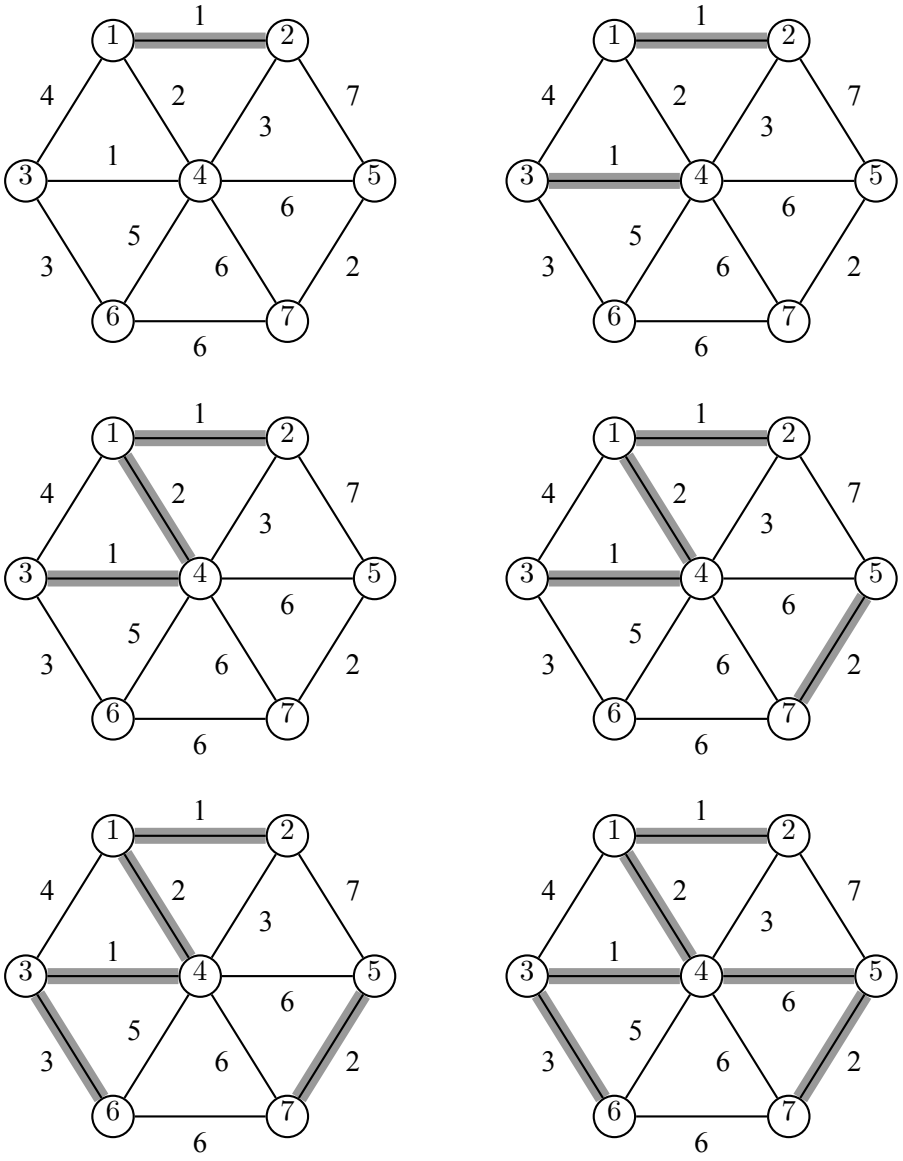
Πίνακας 6.2: Εφαρμογή της μεθόδου του Dijkstra για το γράφο G

Αλγόριθμος 6.5 Άπληστη μέθοδος εύρεσης ελαχίστου κόστους συνδετικού δέντρου (minimum cost spanning tree) με το κριτήριο του Prim

```
procedure Prim (Input: graph G(V, E) with costs on edges;
                Output: MST T of minimum total cost);
  var DistFromTree: array[1..n] of real;
      Edge: array[1..n] of edges; i, j, k, l: integer;
  (* DistFromTree[v] : το ελάχιστο από τα κόστη των ακμών
   που συνδέουν τον κόμβο v με το μέχρι στιγμής κατασκευασμένο
   συνδετικό δέντρο *)
  (* Edge[v] : η ακμή με αυτό το κόστος *)
begin
  (* αρχικοποίηση με βάση τον κόμβο 1 *)
  V_T := {1}
  for v:=1 to n do
  begin
    DistFromTree[v]:=cost[(1, v)]; Edge[v] := (1, v)   (* ενημέρωση απόστασης από δέντρο *)
  end;
  for i:=1 to n − 1 do
  begin
    select v ∈ V \ V_T such that DistFromTree[v] is minimum;
    V_T := V_T ∪ {v}; T := T ∪ Edge[v];
    for u := 1 to n do
    begin
      if cost[(v, u)] < DistFromTree[u] then
      begin
        DistFromTree[u]:=cost[(v, u)]; Edge[u]:=(v, u)
      end
    end
  end
end
```



Σχήμα 6.6: Εφαρμογή του κριτηρίου του Prim για την εύρεση ελαχίστου κόστους συνδετικού δένδρου



Σχήμα 6.7: Εφαρμογή του κριτηρίου του Kruskal για την εύρεση ελαχίστου κόστους συνδετικού δένδρου

Αλγόριθμος 6.6 Απληστη μέθοδος εύρεσης ελαχίστου κόστους συνδετικού δέντρου (minimum cost spanning tree) με το κριτήριο του Kruskal

```
procedure Kruskal (Input: graph G(V, E) with costs on edges;
                   Output: MST T of minimum total cost);
                   (* θεωρεί ακμές ταξινομημένες κατά βάρος *)
begin
  forest := ∅;
  while |forest| < n − 1 do
  begin
    select minimum cost edge (u, w) and delete it from E;
    if (u, w) does not create a cycle in forest then
      add it to forest
    else discard it
  end
end
```

συνεκτικούς γράφους ισχύει:

$$n - 1 \leq e \leq \frac{n(n-1)}{2}$$

Βλέπουμε λοιπόν ότι ο αλγόριθμος του *Kruskal* έχει καλύτερη απόδοση στους αραιούς (*sparse*) γράφους όπου η πολυπλοκότητά του είναι $O(n \log n)$, σε αντίθεση με τον αλγόριθμο του *Prim* που έχει πολυπλοκότητα $O(n^2)$.

6.4 Μέγιστη ροή – Ελάχιστη τομή (Max Flow – Min Cut)

Ένα πρόβλημα που, όπως θα δούμε, ανάγεται στο πρόβλημα της διάσχισης είναι το *Πρόβλημα Μέγιστης Ροής* (Max Flow Problem): Δοθέντος ενός *δικτύου*, δηλαδή ενός κατευθυνόμενου γράφου με βάρη που αντιπροσωπεύουν *χωρητικότητες* και δύο κόμβων s (source, πηγή), t (target, στόχος), ζητείται να δρομολογηθεί όσο το δυνατόν μεγαλύτερη ροή από τον s στον t έτσι ώστε να ισχύει η αρχή διατήρησης της ροής και να μην παραβιάζονται οι χωρητικότητες των ακμών.

Ας σημειωθεί ότι η *ροή* (όπως και η χωρητικότητα) ορίζεται τυπικά ως μια συνάρτηση πάνω στις ακμές που παίρνει μη αρνητικές πραγματικές τιμές $f : E \to \mathbf{R}_+$ (για την χωρητικότητα συμβολίζουμε με $c : E \to \mathbf{R}_+$).

Η αρχή διατήρησης της ροής απαιτεί σε κάθε κόμβο εκτός των s, t το άθροισμα της f στις εισερχόμενες ακμές να είναι ίδιο με το άθροισμα της f στις εξερχόμενες ακμές (πρβλ. νόμο Kirchhoff). Η *τιμή* της ροής ορίζεται ως η καθαρή ροή που εξέρχεται από τον κόμβο s (ή ισοδύναμα, που εισέρχεται στον κόμβο t), δηλαδή το άθροισμα της f στις εξερχόμενες ακμές του κόμβου s μείον το άθροισμα της f στις εισερχόμενες ακμές του κόμβου s.[1]

[1] Στη βιβλιογραφία χρησιμοποιούνται και ορισμοί του προβλήματος που επιτρέπουν αρνητική ροή, ή/και απαιτούν διατήρηση της ροής και στους κόμβους s, t (με προσθήκη μιας ακμής (t, s) εάν δεν υπάρχει). Μπορεί να δειχθεί ότι

Ισχύει το παρακάτω σημαντικό θεώρημα:

Θεώρημα 6.3. *(Max Flow – Min Cut) Η μέγιστη ροή ισούται με την ελάχιστη (ως προς χωρητικότητα) τομή (σύνολο ακμών) που διαχωρίζει τον s από τον t.*

Η απόδειξη του θεωρήματος αυτού, μαζί με τον αλγόριθμο που υπολογίζει τη μέγιστη ροή, δόθηκε από τους Ford και Fulkerson(1962).

Αλγόριθμος 6.7 Αλγόριθμος Ford-Fulkerson

Επαναλαμβάνονται τα παρακάτω βήματα στο δίκτυο για όσο υπάρχει μονοπάτι από τον s στον t (μετά την 1η επανάληψη χρησιμοποιείται το παραμένον δίκτυο):

1. Επιλογή μονοπατιού από τον s στον t. Δρομολόγηση ροής ίσης με την ελάχιστη χωρητικότητα ακμής στο μονοπάτι.
2. Υπολογισμός της *παραμένουσας χωρητικότητας* σε κάθε ακμή του μονοπατιού p (καθώς και στις αντίθετές τους): κατασκευή του *παραμένοντος δικτύου (residual network)*

Ας εξηγήσουμε λίγο περισσότερο το βήμα 2. Έστω f_i η ροή που προστίθεται στην i-οστή επανάληψη (δηλαδή $f_i = \min_{e \in p} c_{i-1}(e)$, όπου c_{i-1} είναι η συνάρτηση χωρητικότητας στο τέλος της $(i-1)$-οστής επανάληψης). Σε κάθε μία από τις ακμές του μονοπατιού p αφαιρούμε χωρητικότητα ίση με f_i και στις αντίθετες ακμές προσθέτουμε χωρητικότητα ίση με f_i (αν δεν υπάρχουν κάποιες από τις αντίθετες ακμές, τις προσθέτουμε). Το δίκτυο που προκύπτει είναι το παραμένον δίκτυο.

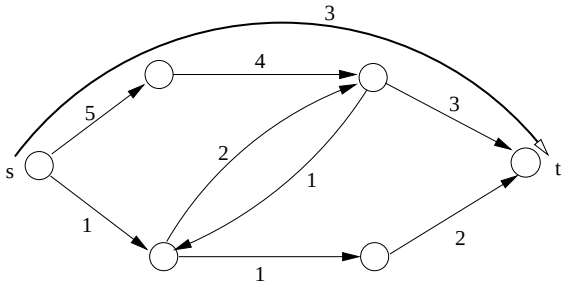
Η τελική ροή είναι το άθροισμα όλων των f_i. Αν σε κάποιο ζεύγος αντίθετων ακμών υπάρχει ροή και στις δύο ακμές, τότε η διαφορά τους αποδίδεται στην ακμή που είχε τη μεγαλύτερη ροή, ενώ η ροή της αντίθετης ακμής μηδενίζεται. Ένα παράδειγμα της εκτέλεσης του αλγορίθμου δίνεται στα Σχήματα 6.8, 6.9, 6.10.

Τα μονοπάτια που χρησιμοποιεί ο αλγόριθμος λέγονται *μονοπάτια επαύξησης (augmenting paths)*.

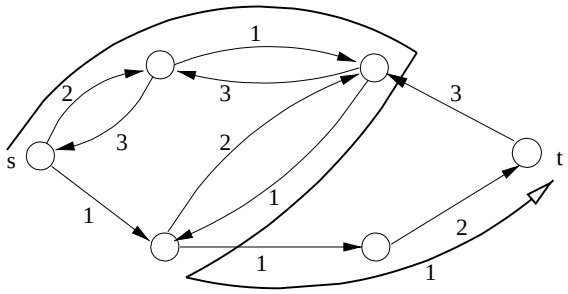
Πολυπλοκότητα του αλγορίθμου: για ακέραιες χωρητικότητες είναι $O(|f^*||E|)$, όπου f^* η τιμή της μέγιστης ροής, καθώς σε κάθε επανάληψη η ροή αυξάνεται τουλάχιστον κατά 1 μονάδα, και η εύρεση μονοπατιού από τον s στον t γίνεται σε χρόνο $(|E|)$. Επομένως, ο αλγόριθμος αυτός δεν είναι πολυωνυμικού χρόνου στη χειρότερη περίπτωση (γιατί;).

Εάν όμως επιλέγεται κάθε φορά το συντομότερο μονοπάτι, τότε ο αριθμός των επαναλήψεων γίνεται πολυωνυμικός, όπως απέδειξαν οι Edmonds-Karp (1972) - ο αντίστοιχος αλγόριθμος έχει πολυπλοκότητα $O(|V||E|^2)$. Ακόμη ταχύτερος είναι ο αλγόριθμος του Goldberg (1986-87) με πολυπλοκότητα $O(|V|^2|E|)$ και $O(|V|^3)$ (μέθοδος preflow-push).

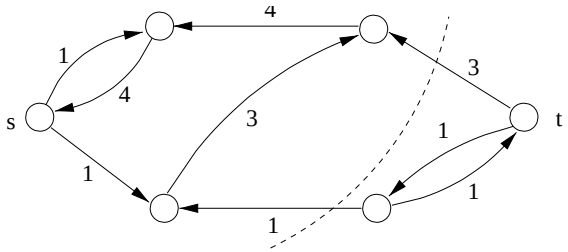
όλοι αυτοί οι ορισμοί είναι ισοδύναμοι με αυτόν που δίνουμε εδώ.



Σχήμα 6.8: Το αρχικό δίκτυο και η πρώτη ροή, μεγέθους 3, από τον κόμβο s στον t.



Σχήμα 6.9: Το παραμένον δίκτυο και η δεύτερη ροή, μεγέθους 1, από τον κόμβο s στον t.



Σχήμα 6.10: Το παραμένον δίκτυο. Δεν υπάρχει μονοπάτι από τον κόμβο s στον t. Η συνολική ροή (δεν απεικονίζεται) προκύπτει από τη διαφορά της χωρητικότητας σε σχέση με το αρχικό δίκτυο. Η ελάχιστη τομή (με διακεκομμένη γραμμή) έχει συνολική χωρητικότητα 4, ίση με τη μέγιστη ροή.

Πρόβλημα ταιριάσματος (Matching) Αξίζει να σημειωθεί ότι ένα άλλο γνωστό πρόβλημα, το πρόβλημα Maximum Matching (Μέγιστο Ταίριασμα), και επομένως και η ειδική του περίπτωση Perfect Matching (Τέλειο Ταίριασμα), λύνονται, όταν η είσοδος είναι διμερής γράφος, με αναγωγή στο πρόβλημα Maximum Flow.

6.5 Πολυπλοκότητα γραφοθεωρητικών προβλημάτων

Υπάρχουν πολυωνυμικοί (ντετερμινιστικοί) αλγόριθμοι για τα κάτωθι προβλήματα: κύκλος Euler, reachability και διάσχιση γράφων, συνεκτικές συνιστώσες, συντομότερα μονοπάτια, ελάχιστο συνδετικό δένδρο (minimum spanning tree), μέγιστη ροή, ταίριασμα (matching), χρωματισμός ακμών σε διμερείς γράφους (bipartite graph edge coloring).

Αντιθέτως, για τα κάτωθι προβλήματα (που είναι όλα ισοδύναμα από άποψη πολυπλοκότητας, NP-πλήρη) είναι γνωστοί μόνο μη ντετερμινιστικοί αλγόριθμοι πολυωνυμικού χρόνου και η ακριβής πολυπλοκότητά τους είναι ανοιχτό πρόβλημα: κάλυψη με κορυφές (vertex cover), κλίκα (clique), κύκλος Hamilton (Hamilton circuit), πρόβλημα πλανόδιου πωλητή (traveling salesperson problem), 3-χρωματισμός (3-colorability), ισομορφισμός υπογράφων (subgraph isomorphism), 3-διάστατο ταίριασμα (3-dimensional matching, 3DM).

6.6 Διαδραστικό Υλικό – Σύνδεσμοι

- Διαδραστικό site οπτικοποιημένων αλγορίθμων και δομών δεδομένων: http://visualgo.net/
- Διαδραστικό site με animated παρουσίαση βασικών αλγορίθμων: http://www.algomation.com/
- Ηλεκτρονικό σύγγραμμα για Foundations of Computer Science, με περιγραφή και ανάλυση πλήθους βασικών αλγορίθμων. Al Aho and Jeff Ullman, "Foundations of Computer Science": http://infolab.stanford.edu/~ullman/focs.html
- Ηλεκτρονικό σύγγραμμα για Αλγόριθμους, βασικούς και προχωρημένους. S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, "Algorithms": http://code.google.com/p/eclipselu/downloads/detail?name=algorithms.pdf
- Λογισμικό LEDA για ανάπτυξη αλγορίθμων (η βασική έκδοση διατίθεται ελεύθερα): http://algorithmic-solutions.com/leda/ledak/index.htm

6.7 Ασκήσεις

1. Σε έναν κατευθυνόμενο γράφο, ένας κόμβος με indegree (πλήθος εισερχομένων ακμών) μηδέν λέγεται πηγή. Θεωρήστε ότι ο γράφος δίνεται σε μορφή πίνακα γειτνίασης.

   (α□) Αποδείξτε ότι σε κάθε ακυκλικό κατευθυνόμενο γράφο υπάρχει τουλάχιστον μία πηγή.

   (β□) Δείξτε ότι ένας κατευθυνόμενος γράφος με n κόμβους είναι ακυκλικός αν και μόνο αν μπορούμε να τοποθετήσουμε ετικέτες $1, 2, \ldots, n$ στους κόμβους ώστε όλες οι ακμές να κατευθύνονται από κόμβο με μικρότερη ετικέτα σε κόμβο με μεγαλύτερη ετικέτα.

(γ□) Περιγράψτε αποδοτικό αλγόριθμο που να αποφαίνεται αν ένας κατευθυνόμενος γράφος είναι ακυκλικός.

2. (α□) Στην όχθη ενός ποταμού βρίσκονται ένας λύκος, ένα πρόβατο κι ένα καφάσι με μαρούλια. Υπάρχει μόνο μια βάρκα, η οποία εκτός από το βαρκάρη μπορεί να μεταφέρει μόνο ένα από τα προηγούμενα κάθε φορά. Όταν ο βαρκάρης είναι παρών τότε επικρατεί ασφάλεια στο σύστημα. Παρόλα αυτά όταν απουσιάζει, κάποια από τα παραπάνω μπορούν το ένα να φάει το άλλο. Συγκεκριμένα ισχύουν τα εξής:

   - Αν ο λύκος και το πρόβατο μείνουν αφύλακτα στην όχθη όσο ο βαρκάρης μεταφέρει το καφάσι με τα μαρούλια, ο λύκος μπορεί να φάει το πρόβατο.
   - Αν το πρόβατο μείνει αφύλακτο μαζί με τα μαρούλια στην όχθη όσο ο βαρκάρης μεταφέρει το λύκο, το πρόβατο θα φάει τα μαρούλια.

   Βρείτε έναν τρόπο ώστε να καταφέρει ο βαρκάρης να τα μεταφέρει και τα τρία άθικτα στην απέναντι όχθη.

   (β□) Γενικεύοντας, έστω ότι υπάρχουν n αντικείμενα $\{x_1, x_2, \ldots x_n\}$, τα οποία ο βαρκάρης επιθυμεί να περάσει στην απέναντι όχθη. Δίνεται γι' αυτά ένας γράφος ασυμβατοτήτων, του οποίου οι κορυφές είναι τα n αντικείμενα και το x_i συνδέεται με το x_j όταν τα x_i και x_j δεν επιτρέπεται να μείνουν αφύλακτα μαζί στην ίδια όχθη (δηλαδή όταν κάποιο από τα δύο μπορεί να φάει το άλλο).

   Βρείτε ποιά πρέπει να είναι τουλάχιστον η χωρητικότητα της βάρκας ώστε το πρόβλημα να λύνεται όταν ο γράφος ασυμβατοτήτων είναι:

   - αλυσίδα (chain) P_n,
   - δακτύλιος (ring) C_n,
   - αστέρι (star) S_n,
   - ο πλήρης γράφος K_n,
   - πλέγμα (grid) διαστάσεων $\sqrt{n} \times \sqrt{n}$,
     Πλέγμα (grid) διαστάσεων $m \times n$ είναι ο γράφος $G(V, E)$, με $V = \{(i, j) : i = 1 \ldots m, j = 1 \ldots n\}$ και $E = \{((i, j), (i', j')) : |i - i'| + |j - j'| = 1\}$

   Πόσες φορές πρέπει ο βαρκάρης να διασχίσει τον ποταμό σε κάθε μια από τις παραπάνω περιπτώσεις; Δώστε μέθοδο επίλυσης.

   (γ□) Περιγράψτε έναν αλγόριθμο για τη γενική περίπτωση του προβλήματος: είσοδος η χωρητικότητα της βάρκας και ο γράφος ασυμβατοτήτων.

3. Ένας οδηγός αποφασίζει να κάνει ένα ταξίδι από την πόλη X μέχρι την πόλη Y με το αυτοκίνητό του, το οποίο έχει αυτονομία κίνησης ως προς τη βενζίνη που μπορεί να αποθηκεύσει, k χιλιόμετρα. Αν ο οδηγός διαθέτει χάρτη στον οποίο αναφέρονται όλα τα βενζινάδικα της διαδρομής με τις αποστάσεις τους και επιπλέον επιθυμεί να πραγματοποιήσει όσο το δυνατό λιγότερες στάσεις, πώς πρέπει να προγραμματίσει τον ανεφοδιασμό καυσίμων; Αποδείξτε την ορθότητα του αλγορίθμου που επινοήσατε.

   Σημείωση: δεν υπάρχει ζεύγος διαδοχικών βενζινάδικων που να απέχουν μεταξύ τους πάνω από k χιλιόμετρα.

4. Κάποιος ισχυρίζεται οτι παρόλο που ο αλγόριθμος Dijkstra δε δουλεύει για κατευθυνόμενους γράφους με αρνητικά βάρη, μπορεί να τροποποιηθεί ώστε να δίνει σωστά αποτελέσματα. Πιο συγκεκριμένα αν v είναι η ακμή με το ελάχιστο βάρος (αρνητικό), προτείνει να προσθέσουμε σε όλες τις ακμές βάρος $|w(v)|$ ώστε να αποκτήσουν όλες οι ακμές θετικό βάρος. Στη συνέχεια προτείνει να εφαρμόσουμε τον αλγόριθμο του Dijkstra ως έχει. Εξηγήστε αν η παραπάνω σκέψη λύνει το single-source πρόβλημα σωστά, δίνοντας απόδειξη ή αντιπαράδειγμα.

Βιβλιογραφία

[1] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. “Algorithms”, MacGraw-Hill, 2006. “Αλγόριθμοι”, ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2008.

[2] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein: “Introduction to Algorithms”, 3rd edition, MIT Press, 2009. Ελληνική έκδοση: “Εισαγωγή στους Αλγόριθμους”, Πανεπιστημιακές Εκδόσεις Κρήτης, 2012.

[3] D. Harel: “Algorithmics: The Spirit of Computing”, Addison-Wesley, Reading, MA, 1st edition, 1987; 2nd edition, 1992. 3rd edition (with Y. Feldman), 2004.

[4] A.V. Aho, J.E. Hopcroft, and J.D. Ullman: “The Design and Analysis of Computer Algorithms”, Addison-Wesley Series in Computer Science andInformation Processing, 1974.

[5] A. Levitin: “Ανάλυση και Σχεδίαση Αλγορίθμων”, Εκδόσεις Τζιόλα, 2007.

[6] Reinhard Diestel, “Graph Theory” (3rd edition), Springer 2005.

[7] H.R. Lewis and C.H. Papadimitriou: “Elements of the Theory of Computation”, 2nd edition, Prentice Hall, 1997. Μεταφρασμένη έκδοση: ”Στοιχεία Θεωρίας Υπολογισμού”, Εκδόσεις Κριτική, 2005.

[8] J.A. Bondy, U.S.R. Murty. “Graph Theory with Applications”. North Holland, 1976. Διατίθεται ελεύθερα στο διαδίκτυο.

[9] Ronald Graham, Donald Knuth, Oren Patashnik. “Συνκριτά Μαθηματικά” (δεύτερη έκδοση, μτφ. Χ. Καπούτσης, επιμ. Ε. Ζάχος) Εκδόσεις Κλειδάριθμος, 2011.

[10] M. Sipser: “Introduction to the Theory of Computation”, 2nd edition, Course Technology, 2005. Μεταφρασμένη έκδοση: “Εισαγωγή στη Θεωρία Υπολογισμού”, Παν. Εκδόσεις Κρήτης, 2007.