

**Αλγόριθμοι και Στοιχεία Πολυπλοκότητας.**

**Ενότητα 10:** Γράφοι.

Διδάσκων: Ηλίας Κ Σάββας, Αναπληρωτής Καθηγητής.

Τμήμα Μηχανικών Πληροφορικής, Τεχνολογικής Εκπαίδευσης.

**Άδειες χρήσης.**

* Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons (C C). **Αναφορά δημιουργού (B Y), Μη εμπορική χρήση (N C), Μη τροποποίηση (N D), 3.0, Μη εισαγόμενο.**
* Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



**Χρηματοδότηση.**

* Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



# Σκοποί ενότητας.

Ο αναγνώστης να μπορεί να:

1) αντιληφθεί την έννοια της δομής γράφου,

2) επιλύει προβλήματα γράφων.

# ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ.

[Σκοποί ενότητας. 2](#_Toc368160998)

[ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ. 3](#_Toc368160999)

[ΑΛΓΌΡΙΘΜΟΙ. 3](#_Toc368161000)

[ΠΡΟΓΡΆΜΜΑΤΑ. 3](#_Toc368161001)

[ΣΧΉΜΑΤΑ. 3](#_Toc368161002)

[ΠΊΝΑΚΕΣ. 3](#_Toc368161003)

[10. ΓΡΆΦΟΙ (Graphs). 4](#_Toc368161004)

[10.1 Εισαγωγικές έννοιες – Ορισμοί. 4](#_Toc368161005)

[10.2 Αλγόριθμος ψαξίματος γράφου κατά βάθος (depth-first-search ή dfs). 6](#_Toc368161006)

[10.3 Ο Αλγόριθμος του DIJKSTRA, (Οικονομικότεροι δρόμοι). 7](#_Toc368161007)

# ΑΛΓΌΡΙΘΜΟΙ.

|  |  |
| --- | --- |
| ***Αλγόριθμος 1****: Αλγόριθμος Depth First Search.* | 6 |
| ***Αλγόριθμος 2****: Αλγόριθμος Dijkstra – Οικονομικότεροι δρόμοι.* | 7 |

# ΠΡΟΓΡΆΜΜΑΤΑ.

|  |  |
| --- | --- |
| ***Πρόγραμμα 1****: Ο αλγόριθμος Dijkstra (Pascal).* | 10 |

# ΣΧΉΜΑΤΑ.

|  |  |
| --- | --- |
| ***Σχήμα 10.1****: Γράφος 5 κορυφών.* | 5 |
| ***Σχήμα 10.2****: Αλγόριθμος Dijkstra.* | 8 |
| ***Σχήμα 10.3****: Ορθότητα αλγόριθμου Dijkstra, 1η περίπτωση.* | 9 |
| ***Σχήμα 10.4****: Ορθότητα αλγόριθμου Dijkstra, 2η περίπτωση.* | 9 |

# ΠΊΝΑΚΕΣ.

|  |  |
| --- | --- |
| ***Πίνακας 1.*** *Πίνακας D.* | 8 |

# 10. ΓΡΆΦΟΙ (Graphs).

## 10.1 Εισαγωγικές έννοιες – Ορισμοί.

Πρώτος που χρησιμοποίησε σε θεωρητικό επίπεδο γράφους, ήταν ο Euler το 1736, στην δημοσίευση με τίτλο “Solutio Problematis ad Geometrian Situs Pertinentis”. Προσπάθησε στην αρχή από απλή μαθηματική περιέργεια, να λύσει το πρόβλημα: «Πως μπορεί κάποιος να διασχίσει τις επτά γέφυρες του ποταμού Konigsberg σε μία βόλτα, αλλά να περάσει μόνο και μία μόνο φορά από την καθεμία».

Ένας ***γράφος (graph)*** $G=(V, E)$*,*αποτελείται από δύο σύνολα: ένα πεπερασμένο σύνολο *V*, από στοιχεία που ονομάζονται κορυφές ή κόμβοι, (nodes ή vertices), και ένα πεπερασμένο επίσης σύνολο *E* ακμών. Εάν οι ακμές ορίζονται σαν διατεταγμένα ζευγάρια κορυφών, τότε ο γράφος ονομάζεται *προσανατολισμένος (directed ή oriented)*, ενώ σε αντίθετη περίπτωση ονομάζεται *μη προσανατολισμένος (undirected ή no oriented, Σχήμα 10.1)*. Στις προκείμενες σημειώσεις, γίνεται λόγος για μη προσανατολισμένους γράφους.

Συνήθως, οι κορυφές συμβολίζονται σαν, $V=\{v\_{1}, v\_{2}, … ,v\_{n}\} $, και οι ακμές σαν $E=\{e\_{1}, e\_{2}, …, e\_{m}\}$. Εάν οι κορυφές, $v\_{i}$ και $v\_{j}$, συσχετίζονται με την ακμή $e\_{i}$, τότε ονομάζονται καταληκτικές κορυφές αυτής της ακμής, και συμβολίζεται σαν $e\_{i}=(v\_{i}, v\_{j})$. Όλες οι ακμές που έχουν τις ίδιες καταληκτικές κορυφές, ονομάζονται ***παράλληλες ακμές***. Επιπλέον, εάν $e\_{i}=(v\_{i}, v\_{i})$, τότε η ακμή $e\_{i}$, αποτελεί ένα ***αυτό-βρόγχο*** της κορυφής $v\_{i}$. Ένας γράφος που δεν έχει παράλληλες ακμές, αλλά ούτε και αυτό-βρόγχους, ονομάζεται ***απλός γράφος***. Ένας γράφος χωρίς ακμές ονομάζεται άδειος γράφος, ενώ χωρίς κορυφές (οπότε και χωρίς ακμές) ονομάζεται ***κενός γράφος***.

* Ο αριθμός των ακμών που έχει μία κορυφή ονομάζεται ***βαθμός της κορυφής*** και συμβολίζεται σαν . Εάν ισχύει  τότε αυτή η κορυφή ονομάζεται ***απομονωμένη κορυφή***.

**ΠΑΡΆΔΕΙΓΜΑ.**

Στο γράφο του παραδείγματος ισχύουν τα ακόλουθα:

1. Είναι ένας απλός γράφος,
2. $V=(v\_{1}, v\_{2}, v\_{3}, v\_{4}, v\_{5}\}$.
3. $E=(e\_{1}, e\_{2}, e\_{3}, e\_{4}, e\_{5}, e\_{6}\}$.
4. $e\_{1}=\left(v\_{1}, v\_{2}\right), e\_{2}=(v\_{2}, v\_{5}), e\_{3}=\left(v\_{5}, v\_{4}\right), e\_{4}=\left(v\_{1}, v\_{4}\right), e\_{5}=\left(v\_{3}, v\_{4}\right), e\_{6}=(v\_{1}, v\_{3})$.
5. $d\left(v\_{1}\right)=3, d\left(v\_{2}\right)=2, d\left(v\_{3}\right)=2, d\left(v\_{4}\right)=3, d\left(v\_{5}\right)=2$.



***Σχήμα 10.1****: Γράφος 5 κορυφών.*

**Περίπατος** (walk) σε ένα γράφο *G*, θεωρείται κάθε ακολουθία κορυφών, $v\_{0}, v\_{1}, …, v\_{k}$, όπου το κάθε ζευγάρι κορυφών,  αποτελεί ακμή του γράφου *G*. Συνήθως ένας περίπατος συμβολίζεται με τις καταληκτικές κορυφές του, πχ $v\_{0}- v\_{k}$. Κάθε κορυφή, σε ένα περίπατο μπορεί να εμφανίζεται περισσότερες από μία φορές. Ένας περίπατος ονομάζεται ***ανοικτός****,* εάν οι καταληκτικές κορυφές του είναι διαφορετικές, αλλιώς ονομάζεται ***κλειστός***. Στο γράφο του *Σχήματος 10.1*, ένας ανοικτός περίπατος είναι ο $v\_{2}, v\_{5}, v\_{4}, v\_{1}, v\_{2}, v\_{5}$, ενώ ένας κλειστός θα μπορούσε να είναι ο $v\_{1}, v\_{4}, v\_{3},v\_{1}$.

Ένας περίπατος ονομάζεται ***πέρασμα*** (trail), εάν όλες οι κορυφές του είναι διαφορετικές μεταξύ τους πχ. $v\_{1}, v\_{4}, v\_{3}$. Ένα ανοικτό πέρασμα ονομάζεται ***μονοπάτι*** (path), ενώ ένα κλειστό, ***κύκλωμα*** (circuit).

Ο αριθμός των ακμών σε ένα μονοπάτι ορίζουν το ***μήκος*** του μονοπατιού. ***Απόσταση*** μεταξύ δύο κορυφών v και *u*, συμβολίζεται σαν , και ορίζεται σαν το μήκος του συντομότερου μονοπατιού μεταξύ των δύο αυτών κορυφών. Εάν δεν υπάρχει τέτοιο μονοπάτι, αυτή η απόσταση ορίζεται σαν άπειρη. ***Διάμετρος*** ενός γράφου *G*, ονομάζεται η μέγιστη απόσταση μεταξύ δύο οποιονδήποτε κορυφών αυτού του γράφου, και συμβολίζεται σαν .

Ισχύουν τα ακόλουθα:

* + Σε ένα μονοπάτι, ο βαθμός κάθε εσωτερικής κορυφής είναι 2, ενώ των καταληκτικών είναι 1.
	+ Σε ένα μονοπάτι, ο αριθμός των κορυφών είναι κατά ένα μεγαλύτερος, από το μήκος του μονοπατιού.

Η ***αναπαράσταση*** ενός γράφου μπορεί να επιτευχθεί με τον ονομαζόμενο ***πίνακα γειτνίασης*** (adjacency matrix) *A*. Ο πίνακας αποτελείται από στοιχεία, εάν υποθέσουμε ότι έχουμε ένα γράφο *n* κορυφών. Το στοιχείο , εάν υπάρχει η ακμή $(v\_{1}, v\_{j})$, ενώ αλλιώς είναι 0. Προφανώς αυτός ο πίνακας είναι συμμετρικός. Το μόνο μειονέκτημα αυτής της αναπαράστασης, είναι ότι απαιτεί  αποθηκευτικό χώρο, ενώ συνήθως ο αριθμός των ακμών είναι μικρότερος του , και φυσικά η προσπέλαση του πίνακα απαιτεί  χρόνο. Για το γράφο του *Σχήματος 10.1*, ο πίνακας γειτνίασης είναι ο ακόλουθος:

Επίσης, ένας γράφος μπορεί να αναπαρασταθεί και με την λίστα γειτνίασης, όπου για τον γράφο του *Σχήματος 10.1*, η λίστα γειτνίασης είναι:

**Α**: 2, 3, 4.

**Β**: 1, 5.

**Γ**: 1, 4.

**Δ**: 1, 3, 5.

**Ε**: 2, 4.

**Εφαρμογές Γράφων:** Αν και οι εφαρμογές της θεωρίας των γράφων είναι πάρα πολλές, αναφέρονται μερικές από τις πιο ενδιαφέρουσες: Δίκτυα (υπολογιστών, οδικά, ηλεκτρικών κυκλωμάτων, τηλεπικοινωνιών, και άλλα), Μηχανική, Ηλεκτρισμός, Χημεία, Βιολογία, και άλλες.

## 10.2 Αλγόριθμος ψαξίματος γράφου κατά βάθος (depth-first-search ή dfs).

 Αυτός ο αλγόριθμος φροντίζει να επισκεφτεί όλους τους κόμβους ενός γράφου, μετακινούμενος πάντα προς κάποιο γειτονικό κόμβο, εάν τέτοιος υπάρχει και δεν τον έχει ήδη ξανά-επισκεφθεί. Εάν δεν υπάρχει τέτοιος κόμβος, ο αλγόριθμος επιστρέφει προς τα πίσω και εκτελείται αναδρομικά. Χρησιμοποιεί δε ένα μονοδιάστατο πίνακα S, με πλήθος στοιχείων όσο και το πλήθος των κορυφών του γράφου. Ο πίνακας αυτός, αρχικοποιείται θέτοντας μηδέν, όλα τα στοιχεία του μηδέν και μετατρέπονται σε ένα, αν κατά την διάρκεια εκτέλεσης του αλγόριθμου, πραγματοποιηθεί επίσκεψη της αντίστοιχης κορυφής του γράφου. Ο αλγόριθμος χρησιμοποιεί τον πίνακα γειτνίασης του γράφου.

Δεδομένα: Ο πίνακας γειτνίασης ******, v το πλήθος των κορυφών, και ****** ο πίνακας που θα κρατάει την πληροφορία, αναφορικά με ποιές κορυφές έχει ήδη επισκεφτεί ο αλγόριθμος. Αρχικά ***.***

***Αλγόριθμος 1****: Αλγόριθμος Depth First Search.*

1: Διαδικασία DFS(k).

2: Αρχή.

3: Για i = k έως ν

4: Εάν S[i] = 0,

5: Τότε,

6: S[k] = 1,

7: Εμφάνισε k,

8: Τέλος Εάν.

9: Για j = i+1 έως ν.

10: Εάν Α[i,j] = 1,

11: Τότε DFS(j),

12: ΤέλοςΕάν.

13: ΤέλοςΓια (j=…).

14: Τέλος Για (i=…).

15: Τέλος Διαδικασίας (DFS).

## 10.3 Ο Αλγόριθμος του DIJKSTRA, (Οικονομικότεροι δρόμοι).

Σε δεδομένο γράφο ******,ένα από τα πιο σημαντικά προβλήματα, αποτελεί τον υπολογισμό του κόστους των διαδρομών από κάποιο κόμβο, (πηγαίο ⁄ source), προς όλους τους άλλους κόμβους του γράφου. Το κόστος του κάθε μονοπατιού, είναι το άθροισμα των αντιστοίχων κοστών των συμμετεχόντων ακμών. Για την επίλυση αυτού του προβλήματος, ο προτεινόμενος αλγόριθμος ανήκει στην κατηγορία των άπληστων (greedy) αλγορίθμων, και συχνά αναφέρεται σαν ο αλγόριθμος του Dijkstra (1959).

Η γενική ιδέα του αλγόριθμου είναι η εξής: Ο αλγόριθμος δημιουργεί ένα σύνολο *S* κόμβων, των οποίων τα μονοπάτια από τον πηγαίο κόμβο είναι ήδη γνωστά. Αρχικά το σύνολο *S*, περιέχει μόνο τον πηγαίο κόμβο. Σε κάθε επανάληψη του αλγόριθμου, προστίθεται ένας κόμβος, του οποίου το μονοπάτι από τον πηγαίο, έχει όσο πιο μικρό κόστος γίνεται. Επίσης, σε κάθε βήμα ο αλγόριθμος ενημερώνει ένα *Πίνακα D*, με το κόστος της οικονομικότερης διαδρομής για κάθε κόμβο. Υποτίθεται ότι υπάρχει ένας πίνακας δύο διαστάσεων, *******,* εάν *n* είναι το πλήθος των κόμβων του γράφου, όπου το στοιχείο, *******,* δίνει το κόστος της ακμής $e\_{i, j}$. Σε περίπτωση που η ακμή δεν υπάρχει, τότε******,ή ένας πολύ μεγάλος αριθμός, ανάλογα με την υλοποίηση του αλγόριθμου.

***Αλγόριθμος 2****: Αλγόριθμος Dijkstra – Οικονομικότεροι δρόμοι.*

1: Έστω Γράφος n κόμβων

2: Έστω S = {1}, / Ο εναρκτήριος κόμβος.

3: Για i = 2 μέχρι n,

4: Di = C1i ,  / Αρχικοποίηση του D,

5: Τέλος Για (i).

6: Για i = 1 μέχρι n-1,

7: Επιλογή κόμβου από το σύνολο V -S, τέτοιον ώστε, το αντίστοιχο Dw να είναι ελάχιστο,

8: Πρόσθεσε τον κόμβο w στο σύνολο S, δηλαδή S = S + {w},

9: Για κάθε κόμβο v του συνόλου V -S,

10: Dv = ελάχιστο(Dv, Dw + Cwv),

11: Τέλος Για.

12: Τέλος του αλγόριθμου «Αλγόριθμος Dijkstra – Οικονομικότεροι δρόμοι».

***Παρατήρηση:*** Εάν ζητείται ο οικονομικότερος δρόμος προς κάποιο συγκεκριμένο κόμβο, ο αλγόριθμος θα μπορούσε να διακόπτεται μόλις φθάσει σε αυτόν τον κόμβο.

* **Παράδειγμα:**

Έστω ο γράφος του *Σχήματος 10.2*, και έστω ότι:

C[1,2] = 10, C[1,4] = 30, C[1,5] = 100.

C[2,3] = 50.

C[3,4] = 20.

C[4,3] = 20, C[4,5] = 60.



***Σχήμα 10.2****: Αλγόριθμος Dijkstra.*

Κατά την εφαρμογή του αλγόριθμου, αρχικά είναι S = {1}, D[2] = 10, D[3] = $\infty $, D[4] = 30, και D[5] = 100. Μετά την πρώτη επανάληψη επιλέγεται ο κόμβος 2, επειδή παρουσιάζει την ελάχιστη τιμή στον *Πίνακα D*. Επίσης το D[3] γίνεται 60, επειδή D[3] = ελάχιστο,($\infty $, 10 + 50). Τα D[4] και D[5] δεν αλλάζουν, διότι δεν επικοινωνούν άμεσα με τον δεύτερο κόμβο, οπότε διατηρείται η αρχική τους τιμή. Η εξέλιξη των τιμών του *Πίνακα D*, είναι ως εξής:

***Πίνακας 1.*** *Πίνακας D.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Επαναλήψεις** | **S** | **W** | **D[2]** | **D[3]** | **D[4]** | **D[5]** |
| Αρχική | 1 |  | 10 | $$\infty $$ | 30 | 100 |
| 1 | 1, 2 | 2 |  10 | 60 | 30 | 100 |
| 2 | 1, 2, 4 | 4 | 10 | 50 | 30 | 90 |
| 3 | 1, 2, 4, 3 | 3 | 10 | 50 | 30 | 60 |
| 4 | 1, 2, 4, 3, 5 | 5 | 10 | 50 | 30 | 60 |

Στην περίπτωση που θέλουμε να κρατάμε και τις διαδρομές, αυτό μπορεί να συμβεί με την χρήση ενός ακόμη πίνακα P, ο οποίος αρχικά θα περιέχει σε όλα τα στοιχεία του τον πηγαίο κόμβο, και μετά από την γραμμή 9 του *Αλγόριθμου 2* θα τοποθετεί, P[v] = w. Αυτό σημαίνει, ότι από κάθε στοιχείο του πίνακα θα φαίνεται ο αμέσως προηγούμενος κόμβος, που θα οδηγήσει προς τον πηγαίο. Οπότε, εάν θέλουμε να υπολογίσουμε την διαδρομή 1, 5, αρκεί να διαβαστεί ο πίνακας από το στοιχείο 5, μέχρι να μας οδηγήσει στον κόμβο 1. Για το γράφο του παραδείγματος, είναι P[1] = 1, P[2] = 1, P[3] = 4, P[4] = 1, και P[5] = 3. Οπότε η διαδρομή 1, 5, δίνεται εάν διαβασθεί ο πίνακας από το πέμπτο στοιχείο, και μέχρι να οδηγήσει στο πρώτο, δηλαδή 5, 3, 4, 1, οπότε και η διαδρομή είναι 1, 4, 3, 5.

* **Πολυπλοκότητα:** Ο αλγόριθμος επαναλαμβάνει *n-1* φορές, τους απαιτούμενους υπολογισμούς, (γραμμή 3), και για κάθε επανάληψη, η εσωτερική επανάληψη εκτελείται το πολύ *n-2* φορές. Οπότε έχουμε ένα μέγιστο *(n-1)\*(n-2)* επαναλήψεων, επομένως η πολυπλοκότητα του αλγόριθμου είναι της τάξης του ******.
* **Απόδειξη ορθότητας του αλγόριθμου:**

***Περίπτωση 1:*** Εάν υποθέσουμε ότι σε κάποια φάση έχει επιλεγεί ο κόμβος w, και έχει ήδη προστεθεί στο σύνολο S, ενώ ταυτόχρονα υπάρχει ένας άλλος κόμβος *,* όπου η διαδρομή S – x – w, είναι οικονομικότερη, όπως φαίνεται στο *Σχήμα 10.3*.



***Σχήμα 10.3****: Ορθότητα αλγόριθμου Dijkstra, 1η περίπτωση.*

Αλλά εάν αυτό ίσχυε, τότε η διαδρομή S – x θα ήταν οικονομικότερη από την διαδρομή S – w, οπότε και θα έπρεπε να είχε επιλεγεί ο κόμβος x, αντί του w, οπότε και καταλήγουμε σε άτοπη υπόθεση.

***Περίπτωση 2:*** Εάν υποθέσουμε τώρα ότι πάλι έχει επιλεγεί ο κόμβος w, και έχει ήδη προστεθεί στο σύνολο S.



***Σχήμα 10.4****: Ορθότητα αλγόριθμου Dijkstra, 2η περίπτωση.*

Υπάρχει το ενδεχόμενο να υπάρχει οικονομικότερος δρόμος προς κάποιο κόμβο v, ο οποίος διέρχεται μεν από τον w, αλλά διέρχεται και από κάποιον κόμβο x, που ήδη ανήκει στο S. Αυτό όμως δεν γίνεται, γιατί μιας και ο κόμβος x έχει ήδη επιλεγεί νωρίτερα από τον w, οπότε η διαδρομή S – x – v, έχει ήδη ελεγχθεί και ενημερωθεί ο *Πίνακας D.*

Οπότε από τα προηγούμενα συνάγεται ότι ο προτεινόμενος αλγόριθμος είναι σωστός.

***Πρόγραμμα 1****: Ο αλγόριθμος Dijkstra (Pascal),*

program dijkstra\_algorithm;

uses crt;

const nn=20; {μέγιστο πλήθος κόμβων}

var

c:array[1..nn,1..nn] of integer; {Πίνακας Κόστους},

d:array[1..nn] of integer; {Πίνακας ελάχιστου κόστους},

s:array[1..nn] of integer; {Πίνακας όπου προστίθενται οι κόμβοι},

pr:array[1..nn] of integer; {Πίνακας μονοπατιού},

n : integer; {Πραγματικός αριθμός κόμβων},

i : integer;

procedure init;

var i, j:integer;

gr:text; {αρχείο που περιέχει τον αριθμό κόμβων και τον πίνακα κόστους}

begin

assign(gr, 'graph.dat');

reset(gr);

readln(gr, n);

for i:=1 to n do

begin

for j:=1 to n

do begin read(gr,c[i,j]); write(c[i,j]:7);

readln(gr); writeln;

end;

close(gr);

end; {init}

procedure da;

{Ο αλγόριθμος},

var i, ii, j, k, min, th:integer;

l:integer;

t:array[2..nn] of integer; {Πίνακας που κρατάει την πληροφορία σε σχέση ποιοι κόμβοι έχουν απομείνει. Σε περίπτωση που t[i]}= -1 ο κόμβος έχει ήδη προστεθεί στο S},

begin

{Αρχικοποιήσεις},

s[1]:=1; d[1]:=0; t[1]:=-1;

for i:=2 to n

do d[i]:= c[1,i];

for i:=1 to n

do pr[i]:= 1;

for i:=2 to n

do t[i]:= i;

k:= 1;

for l:=2 to n do

begin

k:= k+1;

{Επιλογή εναπομεινάντων κόμβων},

ii:= 2;

repeat

i:= t[ii];

ii:= ii+1;

until i<>-1;

min:= d[i];th:= i;

for j:=2 to n

do if t[j]<>-1 then

if min > d[j] then

begin

min:= d[j];

th:= j;

end;

s[k]:= th; d[th]:= min;

t[th]:= -1; writeln(th);

for j:=2 to n do

begin

if t[j]<>-1 then

if d[j] > d[th] + c[th,j] then

begin

d[j]:= d[th] + c[th,j];

pr[j]:= th;

end;

end;

end;

end; {da}

begin {main}

clrscr;

init;

da;

for i:=1 to n

do writeln(s[i]:2, d[s[i]]:10);

writeln('---------------------');

for i:=1 to n

do writeln(pr[i]);

end.

Το αρχείο **Graph.dat** με δεδομένα που αντιπροσωπεύουν τον κόμβο του *Σχήματος ??* (το ∞ εδώ υλοποιείται σαν 32000).

5

32000 10 32000 30 100

32000 32000 50 32000 32000

32000 32000 32000 32000 10

32000 32000 20 32000 60

32000 32000 32000 32000 32000

Τέλος δέκατης ενότητας.

 