ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ
ΙΔΡΥΜΑ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ:

ΣΥΓΚΡΙΣΗ ΑΠΟΔΟΣΗΣ ΑΛΓΟΡΙΘΜΩΝ ΛΕΙΤΟΥΡΓΙΑΣ ΑΝΕΛΚΥΣΤΗΡΑ

ΜΙΝΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ

ΕΠΙΒΛΕΠΟΥΣΑ ΚΑΘΗΓΗΤΡΙΑ: ΠΑΠΑΣΤΕΡΓΙΟΥ ΑΝΑΣΤΑΣΙΑ

Α.Τ.Ε.Ι. Θεσσαλονίκης
Τμήμα Ηλεκτρονικής

ΣΕΠΤΕΜΒΡΙΟΣ 2009

ΤΙΤΛΟΣ: Σύγκριση απόδοσης αλγορίθμων λειτουργίας ανελκυστήρα.

ΚΩΔΙΚΟΣ: 09159ΕΣ

ΗΜΕΡΟΜΗΝΙΑ ΕΓΚΡΙΣΗΣ: 17/2/2009
ΗΜΕΡΟΜΗΝΙΑ ΑΝΑΛΗΨΗΣ: 3/3/2009
ΗΜΕΡΟΜΗΝΙΑ ΠΕΡΑΤΩΣΗΣ: 10/9/2009

ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ
Γεώργιος Μινόπουλος
ΚΑΣ: 503307
Email: geomin1986@hotmail.com

ΣΤΟΙΧΕΙΑ ΕΠΙΒΛΕΠΟΝΤΑ
Δρ Αναστασία Παπαστεργίου
Επιστημονική Συνεργάτης, Τμήμα Ηλεκτρονικής, ΣΤΕΦ, ΑΤΕΙΘ
Διδάκτωρ Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πολυτεχνείο, Δ.Π.Θ.
Τηλέφωνο: 2310 791611, 2310 947786
Email: natp@el.teithe.gr

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Το θέμα αυτής της εργασίας είναι η σύγκριση απόδοσης αλγορίθμων για λειτουργία ανελκυστήρα. Στην εισαγωγή αναφέρομαι στους λόγους για τους οποίους ασχολήθηκα με αυτό το θέμα και στη χρηστική αξία των αλγορίθμων στην καθημερινότητα μας. Στο πρώτο κεφάλαιο κάνω μια ιστορική αναδρομή γύρω από τη λειτουργία του ανελκυστήρα, εξηγώ την έννοια των γλωσσών του προγραμματισμού, προβαίνω σε μια κατηγοριοποίηση αυτών και αναλύω την εξέλιξη της γλώσσας C. Στο δεύτερο κεφάλαιο κάνω μια λεπτομερή εισαγωγή στον κόσμο των αλγορίθμων, γράφω για την σπουδαιότητα τους και τα κριτήρια διαλογής τους. Το τρίτο κεφάλαιο παρουσιάζει και εξηγεί τους αλγορίθμους First In First Out, Shortest Path First και Scan. Στο επόμενο κεφάλαιο αναπτύσσω τον αντίστοιχο κώδικα και τα διαγράμματα ροής για τον κάθε αλγόριθμο με τη σειρά που παρουσιάστηκαν στο προηγούμενο κεφάλαιο. Επίσης γίνεται μια σύγκριση των αποδόσεων τους και ανάλυση των αποτελεσμάτων τους. Στο τελευταίο κεφάλαιο αναλύω τα συμπεράσματα που προκύπτουν από την έρευνα που διενεργήθηκε και παρουσιάζω την εφαρμογή που ανέπτυξα.

ABSTRACT

The subject of this work is the comparison of attribution of algorithms for operation of lift. In the introduction I am reported in the reasons for which I dealt with this subject and in the utilitarian value of algorithms in our everyday routine. In the first chapter I make a historical retrospection round the operation of lift, I explain the significance of programming languages, I proceed in a categorisation of these and I analyze the development of language C. In the second chapter I make a detail introduction in the world of algorithms, I write for their importance and their selection criteria. The third chapter presents and explains the algorithms First In First Out, Shortest Path First and Scan. In the next chapter I develop the corresponding code and the flowcharts for each algorithm as they were presented in the previous chapter. Also I do a comparison of their outputs and an analysis of their results. In the last chapter I analyze the conclusions that result from the research that was held and I present the application that I developed.

ΕΙΣΑΓΩΓΗ

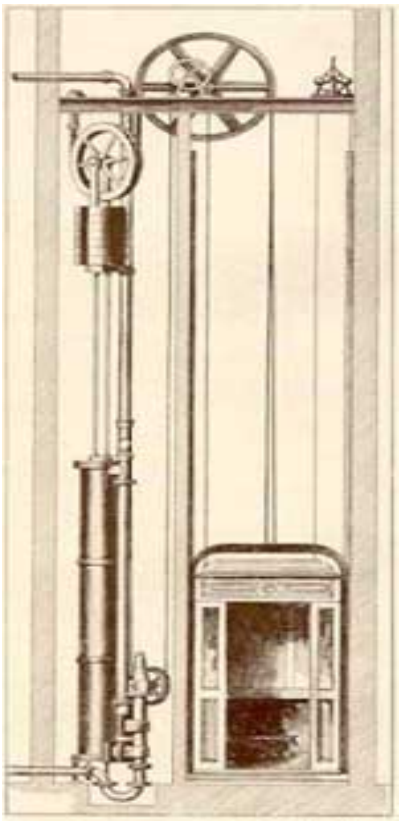
Ο περισσότερος κόσμος σήμερα αγνοεί τη σημασία των αλγορίθμων στη καθημερινότητα μας και ίσως ακόμα να αγνοεί την ίδια την ύπαρξη αυτού του όρου, παρόλο που διάφορες καθημερινές πράξεις αποτελούν μέρος κάποιου αλγόριθμου. Οι αλγόριθμοι μπορεί όντως να απασχολούν κυρίως τους ανθρώπους του κλάδου του προγραμματισμού, βρίσκουν όμως εφαρμογή σε πολλές χρήσιμες λειτουργίες εύκολα προσβάσιμες σε όλους μας, όπως αυτή του ανελκυστήρα.

Ο λόγος για τον οποίο διάλεξα να ασχοληθώ με τους αλγόριθμους στην εργασία αυτή ήταν για να παρουσιάσω τους διαφορετικούς τρόπους με τους οποίους ο κάθε αλγόριθμος μπορεί να επηρεάσει τη λειτουργία του ανελκυστήρα. Ανάλογα με την απαίτηση που έχουμε κάνουμε χρήση και διαφορετικού αλγόριθμου. Στη προσπάθεια μου αυτή ασχολήθηκα με τρείς από τους πλέον δημοφιλής αλγόριθμους. Τον αλγόριθμο FIFO, ο οποίος χρησιμοποιείται κυρίως στα ασανσέρ διώροφων πολυκατοικιών. Τον αλγόριθμο SPF, που βρίσκει χρηστικότητα ιδιαίτερα σε νοσοκομειακά ασανσέρ και τον αλγόριθμο Scan, ίσως ο πιο διαδεδομένος αλγόριθμος στη λειτουργία του ανελκυστήρα, μιας και χρησιμοποιείται σε πολυώροφες πολυκατοικίες, σε ξενοδοχειακές μονάδες ακόμα και σε ουρανοξύστες.

Στόχος της εργασίας είναι να εντοπίσω ποιος από αυτούς τους τρείς αλγόριθμους θέτει τον ανελκυστήρα σε κίνηση με τέτοιο τρόπο, ώστε αυτός να εξυπηρετεί όλους τους χρήστες το συντομότερο δυνατόν. Στο τελευταίο κεφάλαιο με τίτλο «Συμπεράσματα» καταλήγω στο ποιος από τους τρείς αλγόριθμους αποδεικνύεται αποτελεσματικότερος των υπολοίπων.

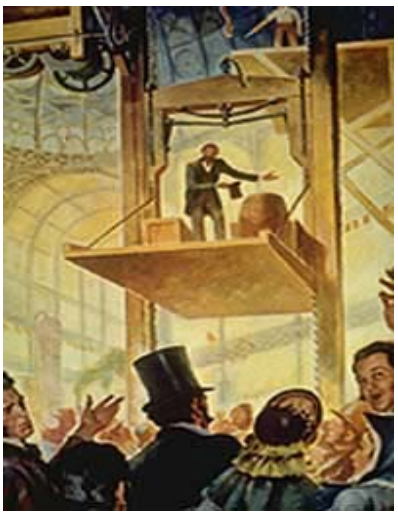
ΚΕΦΑΛΑΙΟ 1^{Ο}

1.1 Ιστορική αναδρομή ανελκυστήρα



Ήταν αρχές του έτους 1853 όταν ο Αμερικανός μηχανικός Ελίσα Γκρέιβις Όπις (Elisha Graves Otis) τελειοποίησε την κατασκευή ενός ανελκυστήρα για να μεταφέρει ανθρώπους. Περιλάμβανε μια ασφαλιστική διάταξη αρπάγης που σφηνωνόταν στους οδηγούς, επάνω στους οποίους κινούνταν το όχημα, μόλις έπαυε να ασκείται δύναμη στο σχοινί ανύψωσης. Έπειτα από τέσσερα χρόνια τέθηκε και επίσημα σε λειτουργία ο πρώτος ανελκυστήρας σε πολυκατάστημα της Νέας Υόρκης. Μέσω μιας ατμομηχανής, η οποία χρησιμοποιούσε ως καύσιμο το κάρβουνο ανέβαινε σε ύψος πέντε ορόφων σε χρονικό διάστημα λιγότερο του ενός λεπτού. Από τότε κι έπειτα άρχισε η ραγδαία εξέλιξη τους και η μαζική χρήση τους.

Αυτό δε σημαίνει ότι δεν έχουμε συναντήσει πρότυπα ανελκυστήρα και στην αρχαιότητα για διάφορες χρήσεις. Οι αρχαίοι Αιγύπτιοι ήδη από το 2700 π.Χ. είχαν την ανάγκη να ανυψώνουν τεράστιες πέτρες για την κατασκευή πυραμίδων - ναών και εκμεταλλευόμενοι την θεωρία του επικλινούς επιπέδου κατορθώνουν με τεράστιες τσουλήθρες να μετακινούν σιγά σιγά προς τα πάνω αυτούς τους τεράστιους όγκους. Η αρχή της χρήσης μηχανών για ανύψωση φορτίων φαίνεται να γίνεται από το 236 π.Χ. όπως αναφέρει ο ρωμαίος αρχιτέκτονας Βιτρούκιος, όπου υπήρχαν τέτοιου είδους συστήματα στα βασιλικά ανάκτορα, που κινούνταν με τη μυϊκή δύναμη ανθρώπων ή ζώων. Στο Μεσαίωνα παρατηρούμε μια



απλή μορφή τους από σχοινιά με γάντζο στην άκρη και ένα καλάθι. Αυτής της μορφής ήταν και οι πρώτοι ανελκυστήρες στην Ελλάδα και συγκεκριμένα στα Μετέωρα με τους οποίους μεταφέρονταν άνθρωποι και εμπορεύματα. Τα πρώτα αυτά μέσα μεταφοράς είχαν ένα σοβαρό πρόβλημα. Αν τυχόν έσπαγε το σχοινί, οι μεταφερόμενοι έπεφταν χωρίς καμιά πιθανότητα σωτηρίας και τα εμπορεύματα δεν είχαν καμιά πιθανότητα να φθάσουν στον προορισμό τους. Έπειτα στη Γαλλία το 17^{ο} αιώνα εφευρέθηκε ένα σύστημα με χρησιμοποίηση αντίβαρου για να ανεβοκατεβαίνει και λίγο αργότερα στην Αγγλία στις αρχές του 18^{ου} αιώνα κατασκευάστηκαν οι πρώτοι υδραυλικοί ανελκυστήρες.

Στη δική μας εποχή μιας και έχουμε την πολυτέλεια του ηλεκτρικού ρεύματος χρησιμοποιούμε μηχανικούς ανελκυστήρες που μας δίνουν την ευελιξία να τροποποιούμε την λειτουργία τους όπως ακριβώς μας βολεύει. Γι αυτό και συναντάμε ασανσέρ σχεδόν σε όλα τα κτίρια των πόλεων, όπως τα σπίτια μας, τα νοσοκομεία, τα εργοστάσια αλλά ακόμα και σε υπερωκεάνια, σε εξέδρες εκτόξευσης πυραύλων και φυσικά στους ουρανοξύστες. Γίνεται εύκολα κατανοητό ότι σε κάθε κτήριο απαιτούνται και διαφορετικές ανάγκες λειτουργίας. Με κύρια χαρακτηριστικά την ταχύτητα μετακινήσεως του ανελκυστήρα, αλλά και τον τρόπο εκτελέσεως των κλήσεων που δέχεται ανά τους ορόφους.

Κάπου εδώ μπαίνει λοιπόν και η αναγκαιότητα του προγραμματισμού. Έχοντας αναπτύξει σε μεγάλο βαθμό τις προγραμματιστικές μας δυνατότητες με την ύπαρξη των πολλών γλωσσών που αυτός διαθέτει μπορούμε να κάνουμε τους ανελκυστήρες απόλυτα λειτουργικούς σε κάθε περίπτωση. Η ανάπτυξη μιας σειράς αλγορίθμων είναι απαραίτητη για να διαπιστώσουμε πότε ο ανελκυστήρας πληροί τις προϋποθέσεις που έχουμε θέση για την αρτιότερη εξυπηρέτηση μας.

1.2 Γλώσσες προγραμματισμού

Για τον περισσότερο κόσμο το ασανσέρ είναι απλά ένας μικρός θάλαμος όπου πατώντας ένα κουμπί μας μεταφέρει κατακόρυφα προς στον όροφο που θέλουμε. Αναμφισβήτητα μια πολλή σημαντική ανακάλυψη που δουλεύει τόσο εύκολα κρύβει από πίσω μια σειρά τεχνογνωσίας για την σωστή λειτουργία του. Πέντε δισεκατομμύρια χιλιόμετρα είναι η απόσταση που διανύουν κάθε χρόνο οι 750 χιλιάδες ανελκυστήρες σε πολυκατοικίες, ξενοδοχεία και ουρανοξύστες των Ηνωμένων Πολιτειών. Από αυτό και μόνο μπορούμε να καταλάβουμε πόσο απαραίτητοι έχουν γίνει στην καθημερινότητα μας.

Το «μυστικό» που δίνει κίνηση στον ανελκυστήρα είναι ο τρόπος με τον οποίο θα τον προγραμματίσουμε. Από τη στιγμή που δέχεται τόσες πολλές εντολές επί καθημερινής βάσης γίνεται κατανοητό ότι πρέπει να γίνει μια πολλή προσεχτική κατασκευή κώδικα για την ορθή εκπλήρωση αυτών των εντολών.

Ως γλώσσα προγραμματισμού λέγεται μια τεχνητή γλώσσα που μπορεί να χρησιμοποιηθεί για τον έλεγχο μιας μηχανής. Οι γλώσσες προγραμματισμού ορίζονται από ένα σύνολο συντακτικών και εννοιολογικών κανόνων, που ορίζουν τη δομή και το νόημα, αντίστοιχα, των προτάσεων της γλώσσας και αυτό είναι που αποκαλείται ως κώδικας του προγραμματισμού. Υπάρχουν χιλιάδες γλώσσες προγραμματισμού και όπως είναι φυσικό όλες διαφέρουν μεταξύ τους στο σύνολο τυπικών προδιαγραφών ή κανόνων που αφορούν το συντακτικό, το λεξιλόγιο και το νόημα της. Κάθε γλώσσα λοιπόν δημιουργείται για να εξυπηρετήσει διαφορετικούς σκοπούς ή για να γίνει ακόμα πιο λειτουργική και εύχρηστη σε σχέση με κάποια άλλη προγενέστερη, συνδυάζοντας τα θετικά στοιχεία αυτής μαζί με την προσθήκη νέων.

Μερικές από τις γνωστότερες γλώσσες προγραμματισμού είναι οι: Pascal, Assembly, Visual Basic, C++, Mat lab, Lisp, Haskell και Java.

1.3 Κατηγοριοποίηση γλωσσών προγραμματισμού

Η κατηγοριοποίηση όλων αυτών των γλωσσών προγραμματισμού δεν είναι ένα εύκολο εγχείρημα, γι' αυτό και υπάρχουν διάφορες περιπτώσεις κατηγοριοποίησης. Ο πλέον διαδεδομένος είναι με βάση τον τρόπο οργάνωσης του προγράμματος. Σ' αυτήν την περίπτωση προκύπτουν τρεις κατηγορίες.

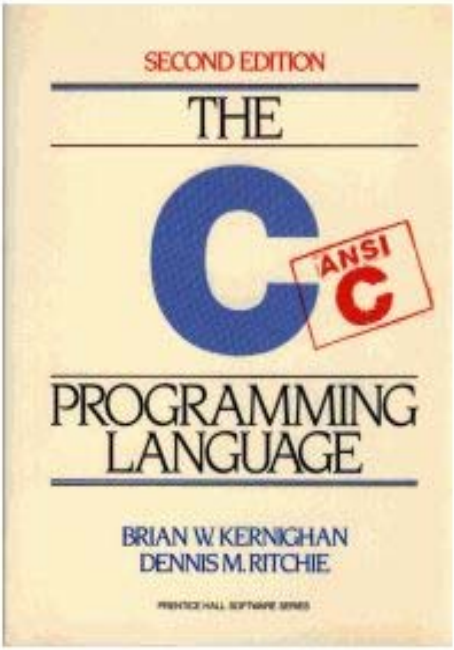
- *Διαδικαστικές γλώσσες* (procedural) όπου το πρόγραμμα είναι οργανωμένο σε διαδικασίες, που αποτελούνται από σειρές εντολών που περιγράφουν αλγορίθμους.
- *Αντικειμενοστρεφείς γλώσσες* (object-oriented) όπου το πρόγραμμα είναι οργανωμένο σε αντικείμενα. Ένα αντικείμενο είναι μια μονάδα που αποτελείται από την περιγραφή κάποιων δεδομένων και την περιγραφή των αλγορίθμων που τα επεξεργάζονται. Ένα αντικειμενοστρεφές πρόγραμμα αποτελείται από διάφορα αντικείμενα που αλληλεπιδρούν μεταξύ τους.
- *Συναρτησιακές γλώσσες* (functional) όπου οι υπολογισμοί εκφράζονται ως εφαρμογές μαθηματικών συναρτήσεων, σε αντίθεση με τα άλλα είδη προγραμματισμού όπου οι υπολογισμοί εκφράζονται ως σειρές εντολών, όπου η κάθε μία αλλάζει με κάποιο τρόπο την κατάσταση του συστήματος.

Στην πρώτη κατηγορία ανήκει και η γλώσσα προγραμματισμού C με την οποία θα ασχοληθούμε ως επί των πλείστων.

1.4 Εξέλιξη γλώσσας C

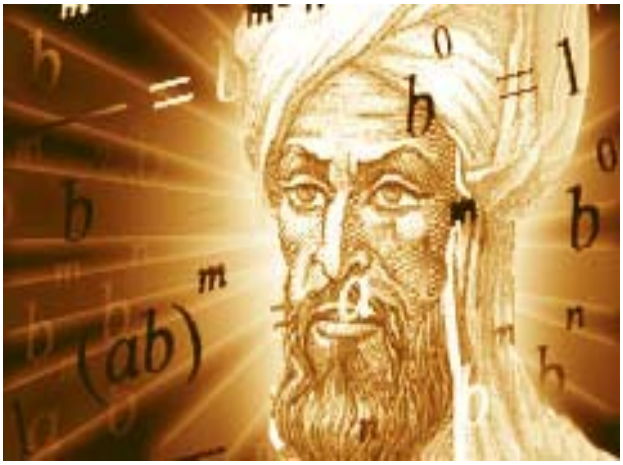
Όπως προείπαμε η C είναι μια γενικής χρήσης διαδικαστική γλώσσα προγραμματισμού η οποία αναπτύχτηκε στις αρχές της δεκαετίας 1970-1980 από τον Dennis Richie. Από τότε χρησιμοποιείται ευρύτατα, και ιδιαίτερα για ανάπτυξη προγραμμάτων συστήματος, αλλά και για απλές εφαρμογές. Ο κυρίως λόγος της ραγδαίας ανάπτυξης της συγκεκριμένης γλώσσας προγραμματισμού είναι η ταχύτητα της.

Αρχικά η C αναπτύχθηκε στα AT&T Bell Labs ανάμεσα στο 1969 και το 1973. Ονομάστηκε "C" λόγω του ότι πολλά από τα χαρακτηριστικά της προήλθαν από μια παλαιότερη γλώσσα, η οποία ονομαζόταν "B". Μέχρι το 1973 η C είχε γίνει αρκετά ισχυρή και αποτελεσματική, ώστε το μεγαλύτερο μέρος του πυρήνα του UNIX, γραμμένο αρχικά σε PDP-11/20 assembly, επανεγγράφηκε σε C. Ήταν ένας από τους πρώτους πυρήνες που υλοποιήθηκε σε μια γλώσσα διαφορετική της assembly. Από εκείνο το σημείο και μετά άρχισε να αυξάνεται όλο και περισσότερο η χρήση της συγκεκριμένης γλώσσας προγραμματισμού για διάφορους σκοπούς κι έτσι το 1978, ο Dennis Ritchie και ο Brian Kernighan δημοσίευσαν την πρώτη έκδοση του "The C Programming Language". Το συγκεκριμένο βιβλίο χρησίμευσε πολλά χρόνια ως ένας ανεπίσημος ορισμός της γλώσσας.



ΚΕΦΑΛΑΙΟ 2^{Ο}

2.1 Αλγόριθμοι



Η λέξη αλγόριθμος προέρχεται από τον Πέρση μαθηματικό του 8^{ου} αιώνα μ.Χ. Αλ Χουαρίζμι (Muhammad ibn Musa Al-Khwarizmi), ο οποίος έγραψε συστηματικές τυποποιημένες λύσεις αλγεβρικών προβλημάτων σε διάφορα συγγράμματά του. Όλες οι τυποποιημένες οδηγίες άρχιζαν με την φράση «ο αλγόριθμος λέει...», έτσι η λέξη αλγόριθμος καθιερώθηκε αργά τα επόμενα χίλια χρόνια με την έννοια «συστηματική διαδικασία αριθμητικών χειρισμών».

Αναφέραμε ότι οι διαδικαστικές γλώσσες προγραμματισμού όπως η C αποτελούνται από σειρές εντολών που περιγράφουν αλγορίθμους. Τι είναι όμως ένας αλγόριθμος; Ως αλγόριθμος ορίζεται μια πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος. Ένας από τους παλαιότερους αλγορίθμους, μεγάλης σπουδαιότητας μάλιστα, μας έγινε γνωστός από το έργο του Ευκλείδη «Στοιχεία» και συγκεκριμένα στο έβδομο από τα δεκατρία βιβλία που περιέχει, είναι ο λεγόμενος αλγόριθμος του Ευκλείδη με τον οποίο βρίσκουμε τον μέγιστο κοινό διαιρέτη δύο ακέραιων αριθμών.

Οι αλγόριθμοι μπορούν να υλοποιηθούν από προγράμματα ηλεκτρονικών υπολογιστών. Ένα λάθος στον σχεδιασμό ενός αλγόριθμου για την λύση ενός προβλήματος μπορεί να οδηγήσει σε αποτυχίες/βλάβες στο εφαρμοσμένο πρόγραμμα. Για οποιαδήποτε υπολογιστική διαδικασία, ο αλγόριθμος πρέπει να είναι αυστηρά ορισμένος για όλες τις πιθανές περιστάσεις που θα μπορούσαν να προκύψουν. Δηλαδή οποιαδήποτε υπό όρους βήματα πρέπει να εξεταστούν συστηματικά, και σε κάθε περίπτωση τα κριτήρια πρέπει να είναι σαφή.

Οι αλγόριθμοι είναι σημαντικοί γιατί σχετίζονται άμεσα με τον τρόπο τον οποίο οι υπολογιστές επεξεργάζονται πληροφορίες. Ένα πρόγραμμα υπολογιστών είναι ουσιαστικά ένας αλγόριθμος που λέει στον υπολογιστή ποια συγκεκριμένα βήματα να εκτελέσει προκειμένου να επιτευχθεί ένας συγκεκριμένος στόχος. Κατά συνέπεια, ένας αλγόριθμος μπορεί να θεωρηθεί οποιαδήποτε ακολουθία εντολών που μπορεί να εκτελεσθεί από ένα πλήρες σύστημα. Χαρακτηριστικά, όταν ένας αλγόριθμος συνδέεται με την επεξεργασία πληροφοριών, τα δεδομένα διαβάζονται από μια συσκευή εισόδου, γράφονται σε μια συσκευή εξόδου, και αποθηκεύονται για την περαιτέρω χρήση. Τα αποθηκευμένα στοιχεία θεωρούνται ως τμήμα της εσωτερικής κατάστασης του συστήματος που εκτελεί τον αλγόριθμο.

2.2 Σπουδαιότητα αλγορίθμων

Έχει γίνει κατανοητό ότι ο αλγόριθμος αποσκοπεί στην επίλυση ενός προβλήματος. Έτσι είναι απαραίτητο να γίνεται μια καλή ανάλυση του κάθε προβλήματος και να προτείνεται συγκεκριμένη μεθοδολογία και ακολουθία βημάτων. Βασικός στόχος μας είναι η πρόταση έξυπνων και αποδοτικών λύσεων. Γι' αυτό προκύπτει ότι για την ανάλυση ενός προβλήματος σε ένα σύγχρονο υπολογιστικό περιβάλλον περιλαμβάνεται η καταγραφή της υπάρχουσας πληροφορίας για το πρόβλημα, η αναγνώριση των ιδιαιτεροτήτων του προβλήματος, η αποτύπωση των συνθηκών και προϋποθέσεων υλοποίησης του, η πρόταση επίλυσης του με χρήση κάποιας μεθόδου και η τελική επίλυση με χρήση υπολογιστικών συστημάτων.

Όποτε κατά την ανάλυση ενός προβλήματος θα πρέπει να δοθεί απάντηση σε κάθε μια από τις πέντε κρίσιμες ερωτήσεις:

1. Ποια είναι τα δεδομένα του προβλήματος.
2. Ποιες είναι οι συνθήκες που πρέπει να πληρούνται για την επίλυση του προβλήματος.
3. Ποια είναι η πλέον αποδοτική μέθοδος επίλυσης τους.
4. Πως θα καταγραφεί η λύση σε ένα πρόβλημα.
5. Ποιος είναι ο τρόπος υλοποίησης στο συγκεκριμένο υπολογιστικό σύστημα.

2.3 Κριτήρια αλγορίθμων

Οι αλγόριθμοι θα πρέπει να πληρούν κάποια πρότυπα και να διατυπώνονται με συγκεκριμένο τρόπο. Έτσι κάθε αλγόριθμος πρέπει απαραίτητα να ικανοποιεί τα επόμενα κριτήρια:

- Είσοδοι. Καμία, μία ή και περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δεν δίνονται τιμές δεδομένων εμφανίζεται όταν ο αλγόριθμος δημιουργεί και επεξεργάζεται κάποιες πρωτογενείς τιμές με τη βοήθεια των συναρτήσεων παραγωγής τυχαίων αριθμών, ή με την βοήθεια άλλων απλών εντολών.
- Έξοδος. Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή δεδομένων ως αποτέλεσμα προς το χρήστη ή προς έναν άλλο αλγόριθμο.
- Καθοριστικότητα. Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσης της. Λόγου χάριν, μία εντολή διαίρεσης πρέπει να θεωρεί και την περίπτωση, όπου ο διαιρέτης λαμβάνει τη μηδενική τιμή.
- Περατότητα. Ο αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα συγκεκριμένο αριθμό βημάτων δεν αποτελεί αλγόριθμο, αλλά λέγεται απλά υπολογιστική διαδικασία.
- Αποτελεσματικότητα. Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι απλή. Αυτό σημαίνει ότι μια εντολή δεν αρκεί να έχει ορισθεί, αλλά πρέπει να είναι και εκτελέσιμη.

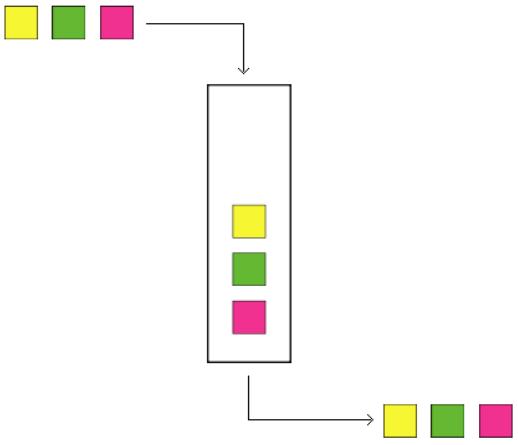
ΚΕΦΑΛΑΙΟ 3^{Ο}

3.1 Αλγόριθμος First In First Out

Ένας από τους γνωστότερους και πλέον διαδεδομένους αλγόριθμους στην επιστήμη των υπολογιστών και όχι μόνο είναι γνωστός με την ονομασία FIFO (First In First Out). Η ονομασία του και μόνο μας δίνει μια πρώτη εικόνα για τον τρόπο με τον οποίο λειτούργει αυτός ο αλγόριθμος. Το κύριο νόημα λειτουργίας του είναι να εξυπηρετεί της αιτήσεις που δέχεται με τη σειρά που αυτές συνέβησαν.

Αναλυτικότερα, όλες οι αιτήσεις μπαίνουν σε μία «ουρά». Η ουρά (queue) είναι ένα συγκεκριμένο είδος συλλογής, στην οποία τα στοιχεία της συλλογής είναι διατεταγμένα και οι πρωτεύουσες πράξεις είναι η εισαγωγή στοιχείων στην πίσω θέση και η διαγραφή στοιχείων από την μπροστά θέση. Στη διαδικασία FIFO λοιπόν, το πρώτο στοιχείο που εισάγεται στην ουρά θα είναι και το πρώτο που θα αφαιρεθεί. Αυτό είναι ισοδύναμο με την απαίτηση ότι, όταν ένα στοιχείο εισάγεται, όλα τα στοιχεία που είχαν εισαχθεί νωρίτερα πρέπει να διαγραφούν πριν το νέο στοιχείο καλεστεί. Ένα χαρακτηριστικό της ουράς είναι ότι δεν έχει συγκεκριμένο μέγεθος. Ασχέτως από το πόσα στοιχεία περιέχονται ήδη, ένα νέο στοιχείο μπορεί πάντα να εισαχθεί. Μπορεί επίσης να είναι άδεια. Σ' αυτό το σημείο, για να διαγραφεί ένα στοιχείο θα είναι αδύνατον μέχρι να εισαχθεί ένα νέο στην ουρά.

Ένα αντιπροσωπευτικό παράδειγμα FIFO είναι οι ουρές των ταμείων μιας τράπεζας. Το πρόσωπο στην αρχή της ουράς είναι το πρώτο που θα φύγει, ενώ στο τέλος της ουράς είναι αυτό που θα φύγει τελευταίο. Κάθε φορά που το πρόσωπο τελειώνει την δουλειά του, εκείνο εγκαταλείπει την ουρά από μπροστά.



3.2 Αλγόριθμος Shortest Path First

Εκ πρώτης όψεως ένας πιο εξελιγμένος αλγόριθμος, για τον καθορισμό της κίνησης του ανελκυστήρα και την εξυπηρέτηση των αιτήσεων, από τον προηγούμενο θεωρείται αυτός που θα ασχοληθούμε τώρα. Ονομάζεται Shortest Path First (SPF) και η βασική του έννοια είναι, ο ανελκυστήρας να εξυπηρετεί πρώτα την αίτηση που βρίσκεται πιο κοντά στον τρέχοντα όροφο. Να κάνει, δηλαδή, κάθε φορά την μικρότερη διαδρομή.

Αυτός ο τρόπος είναι μια άμεση βελτίωση του αλγόριθμου FIFO. Ο αλγόριθμος SPF καθορίζει ποια αίτηση βρίσκεται ποιο κοντά στην τρέχουσα θέση του ανελκυστήρα και μόλις την εξυπηρετήσει, αντίστοιχα, βρίσκει την επόμενη κοντινότερη αίτηση από τη νέα θέση όπου βρίσκεται και συνεχίζει κατ' αυτόν τον τρόπο.

Ο SPF έχει το άμεσο όφελος της απλότητας και θεωρητικά είναι επωφελέστερος σε σύγκριση με τον FIFO, στο ότι η συνολική κίνηση του ανελκυστήρα είναι μειωμένη. Κάτι τέτοιο φέρει ως αποτέλεσμα χαμηλότερο μέσο όρο χρόνου εξυπηρέτησης. Ωστόσο, με δεδομένο ότι θα υπάρχουν σίγουρα νέες αιτήσεις, αυτές μπορούν να ζημιώσουν τον χρόνο εξυπηρέτησης αν βρίσκονται σε κοντινή απόσταση από την τρέχουσα θέση του ανελκυστήρα, γιατί θα εκκρεμούν για μεγάλο χρονικό διάστημα πρότερες αιτήσεις οι οποίες βρίσκονται σε μακρινότερες αποστάσεις. Έτσι λοιπόν με τις μακρινές αιτήσεις να εκκρεμούν αυτός ο αλγόριθμος δε θα είναι σε θέση να σημειώσει πρόοδο.

3.3 Αλγόριθμος Scan

Ένας ακόμα αλγόριθμος με τον οποίο θα ασχοληθούμε είναι αυτός της σάρωσης (Scan). Είναι χαρακτηριστικό ότι τον συγκεκριμένο αλγόριθμο τον συναντάμε συχνά και με την ονομασία Elevator Algorithm, γιατί είναι ο πλέον διαδεδομένος για την λειτουργία του ανελκυστήρα. Πιο συγκεκριμένα ο ανελκυστήρας εξυπηρετεί πρώτα όλες τις αιτήσεις προς τη μία κατεύθυνση (για παράδειγμα προς τα πάνω) και στη συνέχεια όλες τις αιτήσεις που έχουν παρουσιαστεί προς την άλλη κατεύθυνση.

Ο αλγόριθμος Scan είναι ένας απλός αλγόριθμος, πολύ σημαντικός όμως, γιατί ένας ανελκυστήρας μπορεί να αποφασίσει που θα σταματήσει. Με τη μέθοδο της σάρωσης ο ανελκυστήρας κατευθύνεται στην πιο κοντινή αίτηση που βρίσκεται στην κατεύθυνση προς στην οποία ήδη σαρώνει. Μόνο όταν ικανοποιήσει όλες τις αιτήσεις προς την ίδια κατεύθυνση, μπορεί είτε να σταματήσει εάν δεν υπάρχουν άλλες αιτήσεις σε εκκρεμότητα και να παραμείνει σε αδράνεια, είτε αντιστρέφεται η φορά της σάρωσης για να εξυπηρετηθούν και οι αιτήσεις που απομένουν.

Ενώ ο Scan φαίνεται να λύνει το πρόβλημα που αντιμετωπίζει ο SPF, έχει κι αυτός τις αδυναμίες του. Το πρόβλημα αυτό εντοπίζεται στις νέες αιτήσεις, που λαμβάνονται κατά τη διάρκεια της κίνησης του ασανσέρ, όταν αντιστοιχούν στον εκάστοτε όροφο που μόλις ο ανελκυστήρας εξυπηρέτησε. Αν ο ανελκυστήρας έχει ξεκινήσει για παράδειγμα την πορεία του προς τα πάνω και του ζητηθεί να εξυπηρετήσει τον όροφο που μόλις πέρασε θα συνεχίσει την πορεία του κανονικά χωρίς να παρεκκλίνει. Δηλαδή, πρώτα θα πάει προς τους ψηλότερους ορόφους όπου του ζητήθηκε και έπειτα θα κατέβει στον συγκεκριμένο όροφο παρόλο που αρχικά βρισκόταν κοντά.

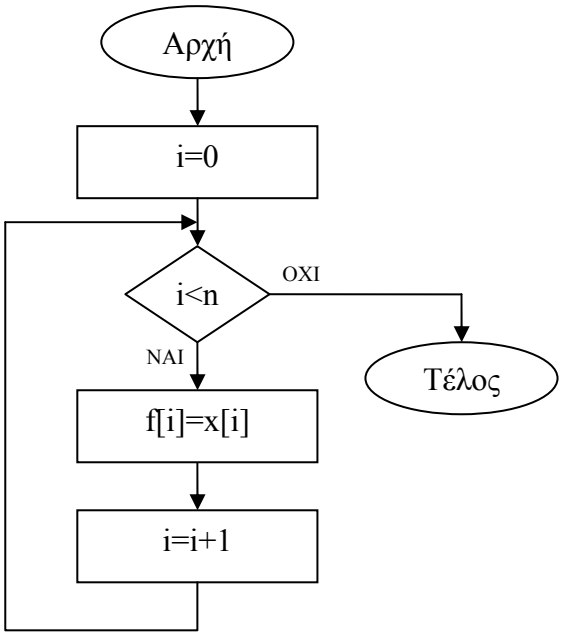
ΚΕΦΑΛΑΙΟ 4^{Ο}

4.1 Υλοποίηση FIFO

Η ανάπτυξη των τριών αλγορίθμων γίνεται μέσω των συναρτήσεων orderFIFO, orderSPF, orderSCAN στις οποίες ορίζουμε τα τρία ορίσματα που θα χρησιμοποιήσουμε. Στην περίπτωση μας τα δυο πρώτα ορίσματα είναι πίνακες και το τρίτο όρισμα καθορίζει το μέγεθος που αυτοί έχουν. Ο δεύτερος πίνακας περιέχει τις αρχικές τιμές. Η κάθε συνάρτηση παίρνει αυτές τις τιμές και τις ανασυντάσσει όπως εμείς επιθυμούμε.

Όπως εξηγήσαμε ο τρόπος που λειτουργεί ο FIFO είναι να εκτελεί τις αιτήσεις με την χρονική σειρά που τις δέχεται. Ο πίνακας x λοιπόν περιέχει τις αιτήσεις που θα δώσουμε στο πρόγραμμα. Ο αλγόριθμος παίρνει αυτές τις τιμές με τη σειρά που δόθηκαν και τις τοποθετεί στον πίνακα f για να τις εκτελέσει. Αυτό που παρατηρούμε τελικά στον αλγόριθμο FIFO είναι ότι ο πίνακας f που δημιουργείτε θα είναι ακριβώς ίδιος με τον πίνακα x που περιέχει τις αρχικές τιμές.

Διάγραμμα ροής:



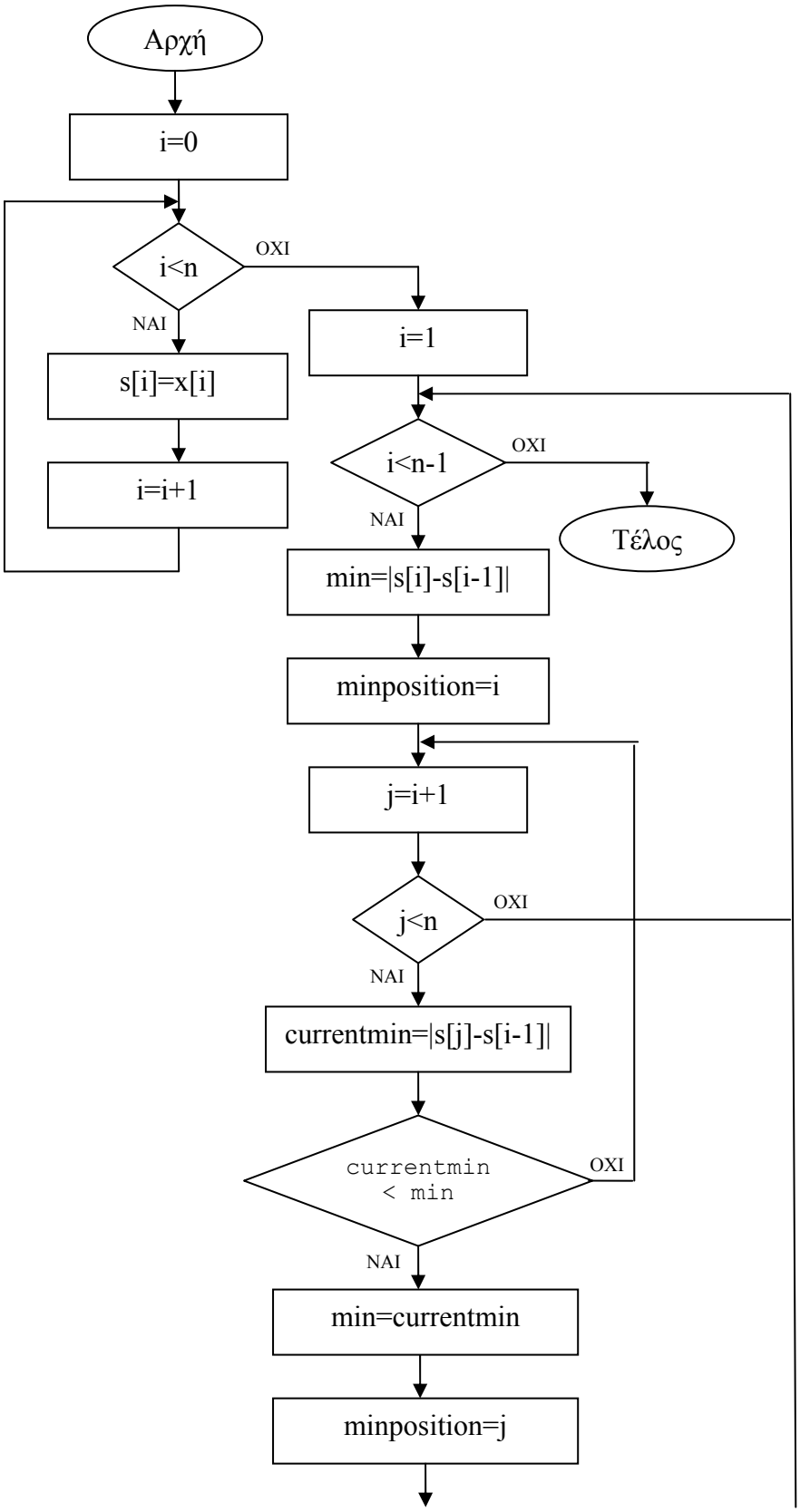
Κώδικας:

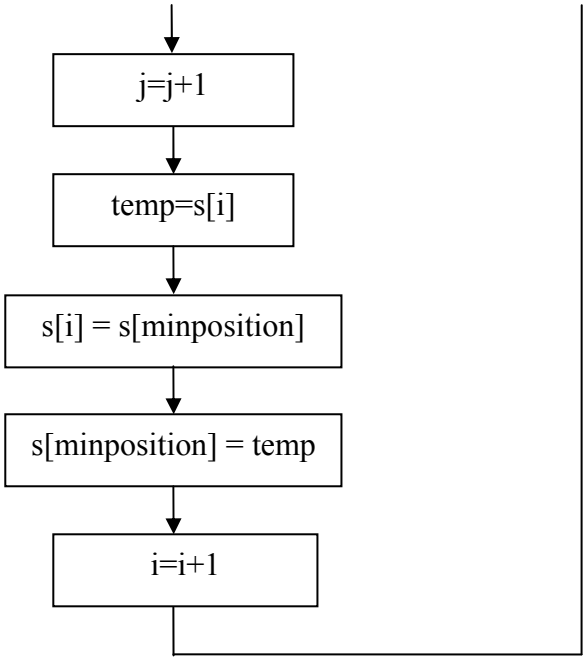
```
void orderFIFO(int *f, int *x, int n)
{
     int i;
     for (i = 0; i < n; i++)
          f[i] = x[i];
}
```

4.2 Υλοποίηση SPF

Σε αυτόν τον αλγόριθμο θέλουμε να εκτελείται η πιο κοντινή σε απόσταση αίτηση σε σχέση με την προηγούμενη. Οπότε αρχικά δημιουργούμε έναν πίνακα s ίδιο με τον πίνακα x, που περιέχει τις αρχικές αιτήσεις. Για να εκτελεστούν οι αιτήσεις με τη σειρά που θέλουμε αφήνουμε την τιμή της πρώτης θέσης ίδια και ύστερα ο αλγόριθμος αντιμεταθέτει στην δεύτερη θέση την αίτηση που βρίσκεται σε απόσταση πιο κοντά στην πρώτη θέση. Αφού το πετύχουμε αυτό, για την τρίτη θέση ο αλγόριθμος μεταφέρει την τιμή που βρίσκεται πιο κοντά στην δεύτερη. Αυτή η διαδικασία με τις αντιμεταθέσεις γίνεται συνεχώς για όλες τις τιμές του πίνακα s. Για την περίπτωση όπου ο ανελκυστήρας βρίσκεται ανάμεσα σε δύο αιτήσεις με την ίδια απόσταση από αυτόν, ο αλγόριθμος επιλέγει να ικανοποιήσει την αίτηση που δόθηκε χρονικά πρώτη εκ των δύο και ύστερα με τον τρόπο που αναφέρουμε πιο πάνω θα έρθει και η στιγμή που θα ικανοποιήσει και την άλλη αίτηση.

Διάγραμμα ροής:




j=j+1
temp=s[i]
s[i] = s[minposition]
s[minposition] = temp
i=i+1

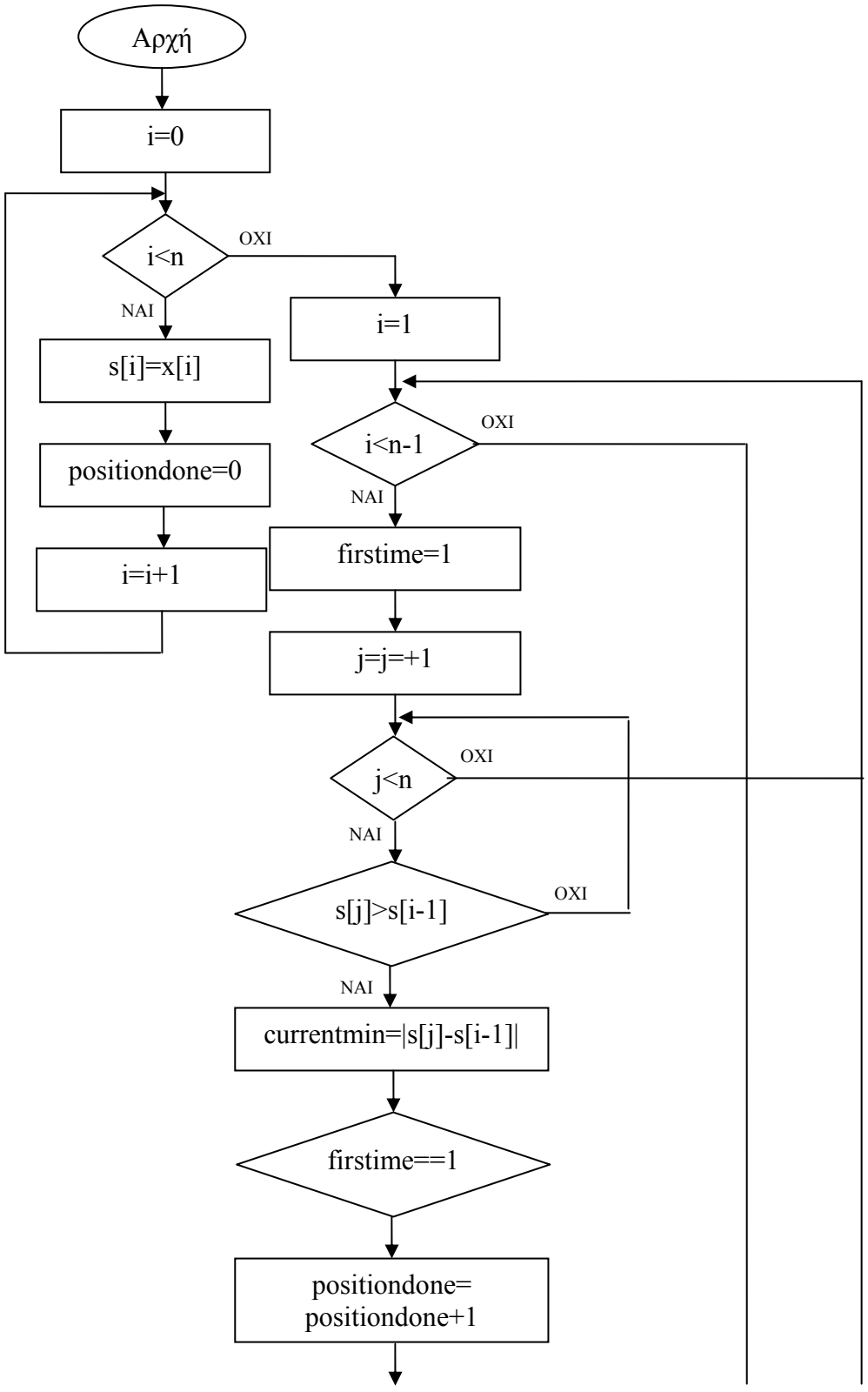
Κώδικας:

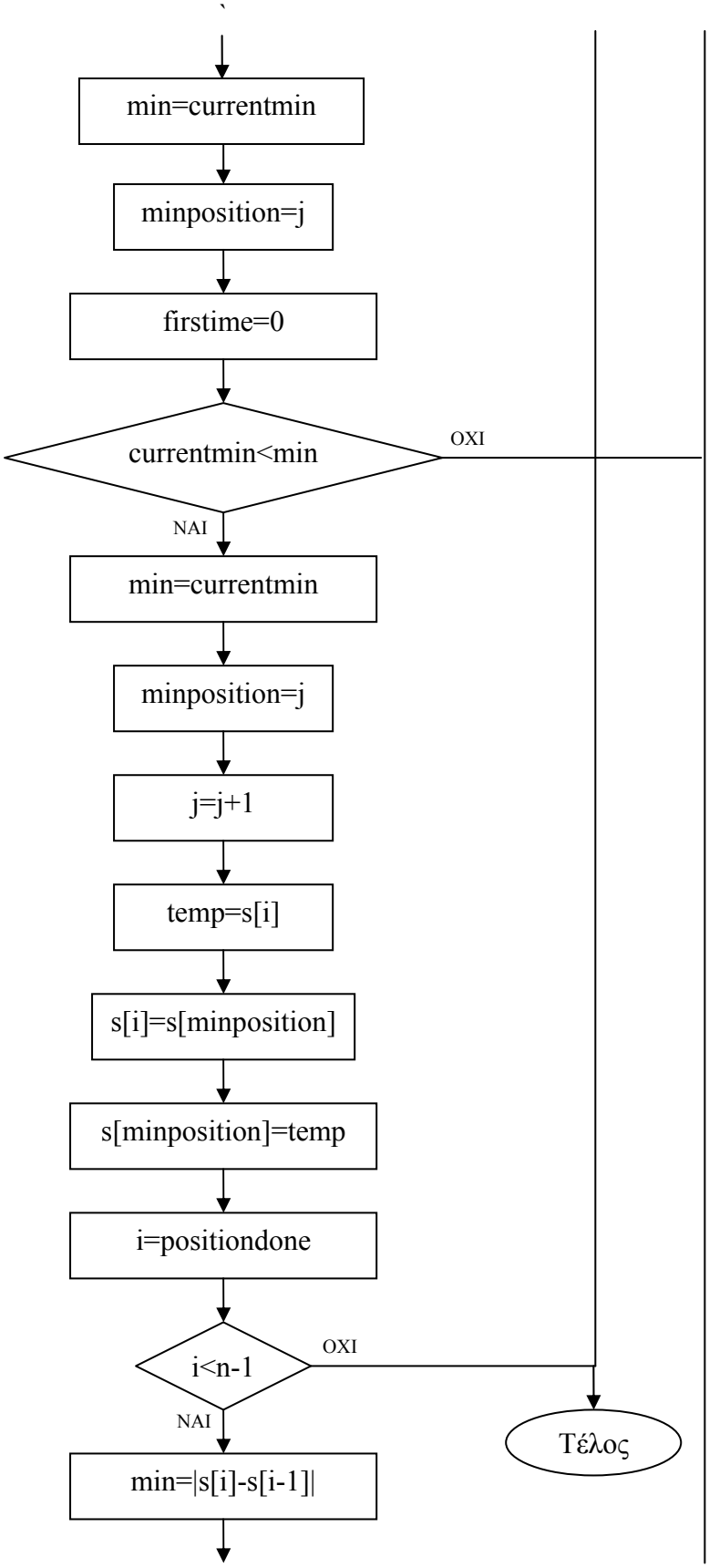
```
void orderSPF(int *s, int *x, int n)
{
     int i, j;
     int min, minposition, currentmin;
     int temp;
     // αρχικοποεί τον πίνακα s ίδιο με τον πίνακα x
     for (i = 0; i < n; i++)
          s[i] = x[i];
     // κάνει τις κατάλληλες αντιμεταθέσεις στον πίνακα s
     για να δημιουργηθεί η σειρά SPF
     for (i = 1; i < n-1; i++)
     {
          min = abs(s[i] - s[i-1]);
          minposition = i;
          for (j = i + 1; j < n; j++)
          {
               currentmin = abs(s[j] - s[i-1]);
               if (currentmin < min)
               {
                    min = currentmin;
                    minposition = j;
               }
          }
          temp = s[i];
          s[i] = s[minposition];
          s[minposition] = temp;
     }
}
```

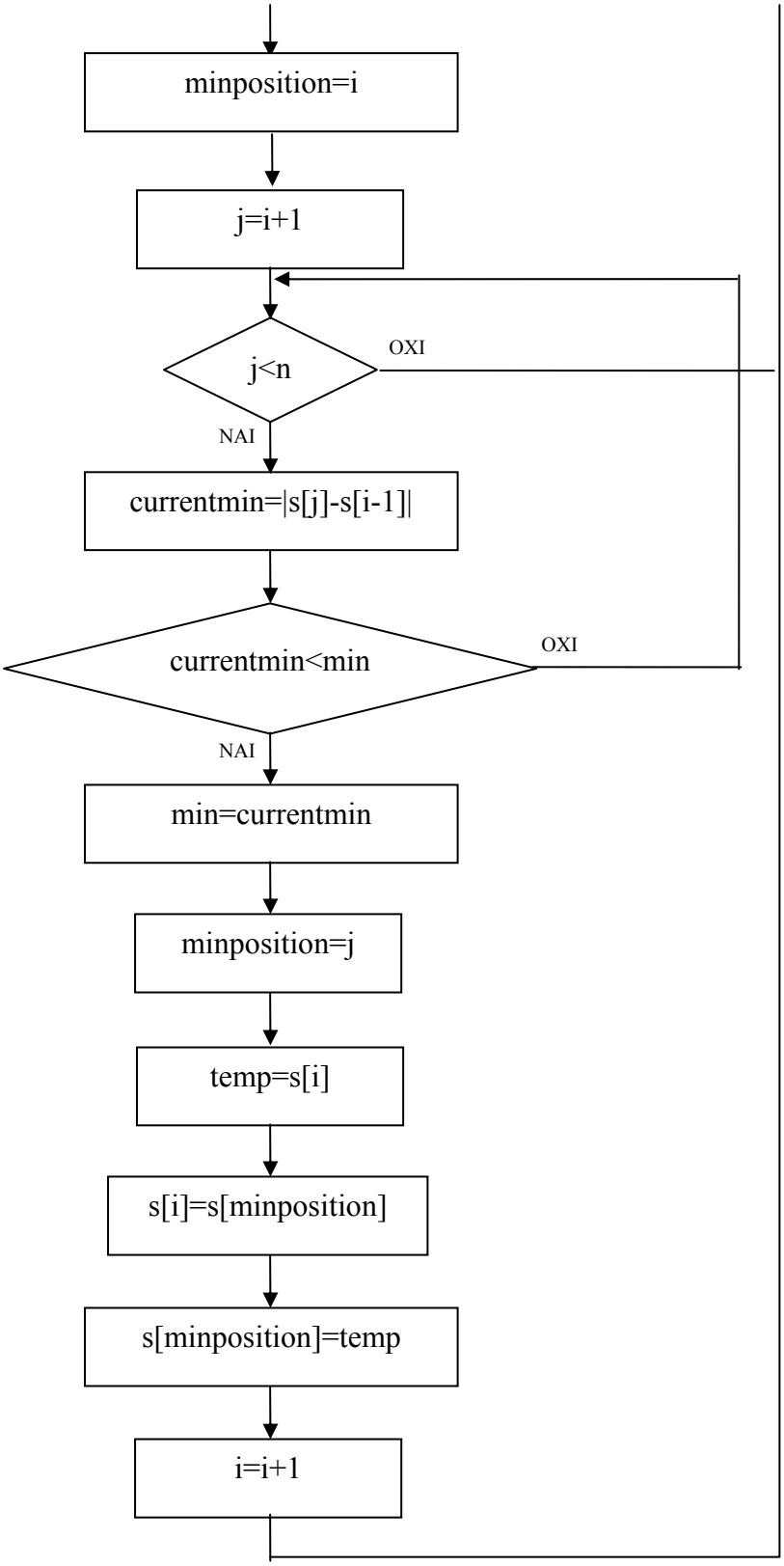
4.3 Υλοποίηση SCAN

Ο τρόπος λειτουργίας αυτού του αλγορίθμου ταιριάζει αρκετά με τον προηγούμενο. Αυτό που συμβαίνει εδώ είναι να εξυπηρετούνται όλες οι αιτήσεις προς τη μια κατεύθυνση, ξεκινώντας, χωρίς βλάβη της γενικότητας, με πορεία προς τα πάνω και έπειτα κατεβαίνοντας να εξυπηρετεί και τις υπόλοιπες αιτήσεις με τη σειρά που συναντάει τους ορόφους. Και πάλι αρχικοποιούμε τον πίνακα s δίνοντας του τις ίδιες τιμές με τον πίνακα x. Η τιμή της πρώτης θέσης παραμένει πάλι η ίδια και αυτή τη φορά ο αλγόριθμος ψάχνει ποια αίτηση ορόφου βρίσκεται κοντινότερα αλλά με την προϋπόθεση να έχει τιμή μεγαλύτερη του τρέχοντος ορόφου. Αφού βρει αυτή τη τιμή την αντιμεταθέτει με την τιμή της δεύτερης θέσης. Κατ' αυτόν τον τρόπο κάνει και την σύγκριση της τιμής της δεύτερης θέσης μέχρι να βρει την επόμενη κοντινότερη αλλά ταυτόχρονα και μεγαλύτερη τιμή και την αντιμεταθέτει με την τρίτη θέση. Συνεχίζει έτσι μέχρι να φτάσει στην υψηλότερη τιμή αίτησης. Από κει ξεκινάει η καθοδική πορεία του ανελκυστήρα με παρόμοιο τρόπο. Ο αλγόριθμος εντοπίζει ποια αίτηση, που δεν έχει ικανοποιηθεί ακόμη, βρίσκεται ποιο κοντά στον υψηλότερο όροφο και την αντιμεταθέτει στη διπλανή θέση (είναι αυτονόητο ότι θα έχει τιμή μικρότερη της πρώτης θέσεως του πίνακα μας). Και αυτή η διαδικασία γίνεται μέχρι να φτάσουμε στον χαμηλότερο όροφο, που έχει αιτηθεί, δηλαδή στην μικρότερη τιμή του πίνακα s.

Διάγραμμα ροής:




min=currentmin
minposition=j
firstime=0
currentmin<min
OXI
NAI
min=currentmin
minposition=j
j=j+1
temp=s[i]
s[i]=s[minposition]
s[minposition]=temp
i=positiondone
i<n-1
OXI
NAI
Τέλος
min=|s[i]-s[i-1]|


minposition=i
j=i+1
j<n
OXI
NAI
currentmin=|s[j]-s[i-1]|
currentmin<min
OXI
NAI
min=currentmin
minposition=j
temp=s[i]
s[i]=s[minposition]
s[minposition]=temp
i=i+1

Κώδικας:

```
void orderSCAN(int *s, int *x, int n)
{
     int i, j;
     int min, minposition, currentmin;
     int temp;
     int firsttime;
     int positiondone;
     // αρχικοποεί τον πίνακα s ίδιο με τον πίνακα x
     for (i = 0; i < n; i++)
          s[i] = x[i];
     // κάνει τις κατάλληλες αντιμεταθέσεις στον πίνακα s
     για να δημιουργηθεί η σειρά SCAN
     // στο ανέβασμα (καθώς ο ανελκυστήρας ανεβαίνει)
     positiondone = 0;
     for (i = 1; i < n-1; i++)
     {
          firsttime = 1;
          for (j = i + 1; j < n; j++)
          {
               if (s[j] > s[i-1])
               {
                    currentmin = abs(s[j] - s[i-1]);
                    if (firsttime == 1)
                    {
                    positiondone = positiondone + 1;
                    min = currentmin;
                    minposition = j;
                    firsttime = 0;
               }
               else
               {
                    if (currentmin < min)
                    {
```

```
                        min = currentmin;
                        minposition = j;
                    }
                }
            }
        }
        temp = s[i];
        s[i] = s[minposition];
        s[minposition] = temp;
}
// στο κατέβασμα (καθώς ο ανελκυστήρας κατεβαίνει)
for (i = positiondone; i < n-1; i++)
{
        min = abs(s[i] - s[i-1]);
        minposition = i;
        for (j = i + 1; j < n; j++)
        {
            currentmin = abs(s[j] - s[i-1]);
            if (currentmin < min)
            {
                min = currentmin;
                minposition = j;
            }
        }
        temp = s[i];
        s[i] = s[minposition];
        s[minposition] = temp;
        }
}
```

4.4 Σύγκριση των αλγορίθμων

Έχοντας ολοκληρώσει την δημιουργία των τριών αλγορίθμων που μας απασχολούν, το επόμενό μας βήμα είναι να διαπιστώσουμε ποιος είναι ο πιο αποδοτικός. Ποιος, δηλαδή, θα ανταποκριθεί στις ίδιες κλήσεις συντομότερα. Αυτόματα οδηγούμαστε στη σύγκριση τους. Για να βγάλουμε ασφαλές αποτέλεσμα αυτό που έχουμε να κάνουμε είναι να μετρήσουμε τη διαδρομή που πρόκειται να διανύσει ο ανελκυστήρας μέχρι να εξυπηρετήσει όλες τις αιτήσεις. Αυτό πρέπει να γίνει για τον καθένα αλγόριθμο ξεχωριστά. Το κυριότερο όμως είναι να τους ελέγξουμε για κάθε πιθανό συνδυασμό ορόφων που μπορεί να προκύψει. Όλοι οι συνδυασμοί προκύπτουν από τον τύπο n^n, όπου n = αριθμός των ορόφων. Αν δηλαδή εξετάζουμε μια τετραώροφη πολυκατοικία έχουμε $4^4 = 256$ συνδυασμούς.

Έτσι ακριβώς λειτουργεί και ο συγκεκριμένος κώδικας. Πραγματοποιεί όλους τους συνδυασμούς για n=2 ως n=8 και το πρόγραμμα εμφανίζει το μέσο όρο των ορόφων που χρειάστηκε να διανύσει για να εκπληρώσει όλους τους συνδυασμούς, το μέσο όρο αναμονής των ορόφων μέχρι να εξυπηρετηθούν και το μέσο όρο της μέγιστης αναμονής ενός ορόφου.

Κώδικας:

```
#include <stdio.h>
#include <malloc.h>
#include <math.h>

int *define(int n)
{
     int *x;
     x = (int *) malloc(n * sizeof(int));
     return x;
}

void initialization(int *x, int n)
{
     int i;
```

```
     for (i = 0; i < n; i++)
          x[i] = 0;
}

void orderFIFO(int *f, int *x, int n)
{
     int i;
     for (i = 0; i < n; i++)
          f[i] = x[i];
}

void orderSPF(int *s, int *x, int n)
{
     int i, j;
     int min, minposition, currentmin;
     int temp;

     for (i = 0; i < n; i++)
          s[i] = x[i];

     for (i = 1; i < n-1; i++)
     {
          min = abs(s[i] - s[i-1]);
          minposition = i;
          for (j = i + 1; j < n; j++)
          {
               currentmin = abs(s[j] - s[i-1]);
               if (currentmin < min)
               {
                    min = currentmin;
                    minposition = j;
               }
          }
          temp = s[i];
```

```
            s[i] = s[minposition];
            s[minposition] = temp;
        }
    }

    void orderSCAN(int *s, int *x, int n)
    {
        int i, j;
        int min, minposition, currentmin;
        int temp;
        int firsttime;
        int positiondone;
        int updown;

        for (i = 0; i < n; i++)
            s[i] = x[i];

        if (s[1] > s[0])
        {
            updown = 1;
        }
        else
        {
            updown = 2;
        }

        positiondone = 0;
        for (i = 1; i < n-1; i++)
        {
            minposition = i;
            firsttime = 1;
            for (j = i; j < n; j++)
            {
```

```
                if ((updown == 1 && s[j] > s[i-1]) ||
(updown == 2 && s[j] < s[i-1]))
                {
                     currentmin = abs(s[j] - s[i-1]);
                     if (firsttime == 1)
                     {
                          positiondone = positiondone + 1;
                          min = currentmin;
                          minposition = j;
                          firsttime = 0;
                     }
                     else
                     {
                          if (currentmin < min)
                          {
                               min = currentmin;
                               minposition = j;
                          }
                     }
                }
           }
           temp = s[i];
           s[i] = s[minposition];
           s[minposition] = temp;
     }

     for (i = positiondone; i < n-1; i++)
     {
           min = abs(s[i] - s[i-1]);
           minposition = i;
           for (j = i + 1; j < n; j++)
           {
                currentmin = abs(s[j] - s[i-1]);
                if (currentmin < min)
```

```
                {
                    min = currentmin;
                    minposition = j;
                }
            }
            temp = s[i];
            s[i] = s[minposition];
            s[minposition] = temp;
        }
}

int distance(int *x, int n)
{
        int i;
        int d = 0;

        for (i = 0; i < n-1; i++)
        {
            d = d + abs(x[i] - x[i+1]);
        }
        return d;
}

int delaySum(int *x, int n)
{
        int i;
        int d = abs(x[0] - x[1]);
        int delay = 0;

        for (i = 1; i < n; i++)
        {
            delay = delay + d;
            d = abs(x[i] - x[i+1]);
        }
```

```
     return delay;
}

int search(int *x, int n, int key)
{
     int index = 0;

     while (x[index] != key && index < n)
     {
          index++;
     }
     return index;
}

int max(int *x, int n)
{
     int i;
     int m = x[0];

     for (i = 1; i < n; i++)
          if (m < x[i])
               m = x[i];
     return m;
}

int delayMax(int *array, int *x, int n)
{
     int i;
     int *delays;
     int d = abs(x[0] - x[1]);
     int xxx;

     delays = (int *) malloc(n * sizeof(int));
```

```
     delays[0] = 0;
     for (i = 1; i < n; i++)
     {
          delays[i] = n - search(array, n, x[i]) +
search(x, n, x[i]);
     }

     xxx = max(delays, n);

     free(delays);
     delays = NULL;

     return xxx;
}

void print(int *x, int n)
{
     int i;
     for (i = 0; i < n; i++)
          printf("%d\n", x[i]);
}

int nextOrder(int *array, int n)
{
     int increased = 0;
     int index = n - 1;
     int end = 0;

     while (increased == 0 && end != 1)
     {
          if (array[index] < n-1)
          {
               array[index]++;
               increased = 1;
```

```
            }
            else
            {
                  array[index] = 0;
                  index--;
                  if(index == -1)
                  {
                        end = 1;
                  }
            }
      }
      return end;
}

void statistics(int *array, int *FIFO, int *SPF, int
*SCAN, int n)
{
      double sumFIFO = 0, sumSPF = 0, sumSCAN = 0;
      double delayFIFO = 0, delaySPF = 0, delaySCAN = 0;
      double delayMaxFIFO = 0, delayMaxSPF = 0,
delayMaxSCAN = 0;
      int times = 0;
      int end = 0;

      while (end != 1)
      {
            orderFIFO(FIFO, array, n);
            orderSPF(SPF, array, n);
            orderSCAN(SCAN, array, n);

            times++;
            sumFIFO += distance(FIFO, n);
            sumSPF += distance(SPF, n);
            sumSCAN += distance(SCAN, n);
```

```
            delayFIFO += delaySum(FIFO, n) / (double)n;
            delayMaxFIFO += delayMax(array, FIFO, n) /
(double)n;

            delaySPF += delaySum(SPF, n) / (double)n;
            delayMaxSPF += delayMax(array, SPF, n) /
(double)n;

            delaySCAN += delaySum(SCAN, n) / (double)n;
            delayMaxSCAN += delayMax(array, SCAN, n) /
(double)n;

            end = nextOrder(array, n);
        }

        printf("FIFO sum: %f\taverage:%f\n", sumFIFO,
sumFIFO / times);
        printf("SPF  sum: %f\taverage:%f\n", sumSPF, sumSPF
/ times);
        printf("SCAN sum: %f\taverage:%f\n", sumSCAN,
sumSCAN / times);
        printf("\n");
        printf("FIFO delay: %f\t\taverage:%f\n", delayFIFO,
delayFIFO / times);
        printf("SPF  delay: %f\t\taverage:%f\n", delaySPF,
delaySPF / times);
        printf("SCAN delay: %f\t\taverage:%f\n", delaySCAN,
delaySCAN / times);
        printf("\n");
        printf("FIFO delay max: %f\t\t\taverage:%f\n",
delayMaxFIFO, delayMaxFIFO / times);
        printf("SPF  delay max: %f\t\t\taverage:%f\n",
delayMaxSPF, delayMaxSPF / times);
```

```
     printf("SCAN delay max: %f\t\t\taverage:%f\n",
delayMaxSCAN, delayMaxSCAN / times);
     printf("\n\n\n");
}

int main()
{
     int i;
     int *array;
     int *FIFO;
     int *SPF;
     int *SCAN;

     for (i = 2; i <= 8; i++)
     {
          printf("n = %d\n", i);

          array = define(i);
          FIFO = define(i);
          SPF = define(i);
          SCAN = define(i);

          initialization(array, i);
          statistics(array, FIFO, SPF, SCAN, i);

          free(array);
          array = NULL;
          free(FIFO);
          FIFO = NULL;
          free(SPF);
          SPF = NULL;
          free(SCAN);
          SCAN = NULL;
     }
     return 0;
}
```

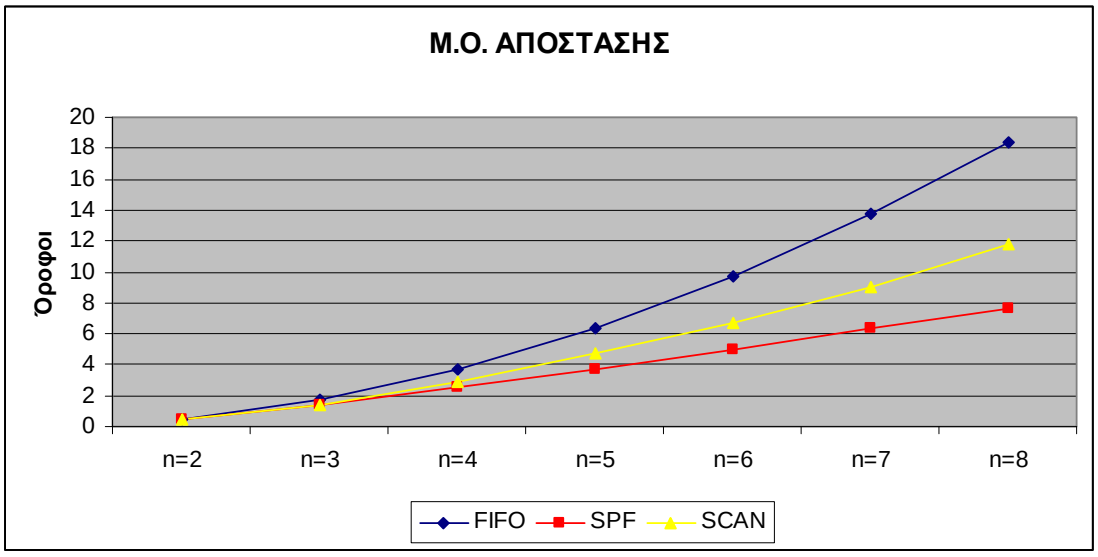
4.5 Στατιστική ανάλυση αποδόσεων

Παρακάτω θα διαπιστώσουμε ποιος αλγόριθμος είναι ο αποδοτικότερος. Αυτό θα προκύψει από τη σύγκριση των αλγορίθμων σε δύο διαφορετικά κριτήρια. Το ένα κριτήριο είναι ο μέσος όρος της απόστασης που διανύει ο ανελκυστήρας και το άλλο ο μέσος όρος της μέγιστης αναμονής, δηλαδή η αναμονή του ορόφου που άργησε περισσότερο από τους άλλους να εξυπηρετηθεί.

Για n=1 δε χρειάζεται να δοκιμάσουμε, γιατί για έναν όροφο οι συνδυασμοί είναι εντελώς περιορισμένοι και θα λάβουμε τα ίδια αποτελέσματα και για τους τρεις αλγόριθμους. Επίσης ο χρόνος μέγιστης αναμονής για τον αλγόριθμο FIFO είναι πάντα ο ίδιος, μιας και εκτελεί τις κλήσεις με τη σειρά, οπότε δε χρειάζεται να τον θέσουμε σε σύγκριση με τους άλλους. Θα αρχίσουμε από μια τριώροφη πολυκατοικία και θα εξετάσουμε όλους τους συνδυασμούς μέχρι να φτάσουμε τους οκτώ ορόφους. Θα ξεκινήσουμε τον έλεγχο με τη σύγκριση των αποστάσεων και έπειτα θα ελέγξουμε και τον χρόνο μέγιστης αναμονής.

Σύγκριση αποστάσεων:

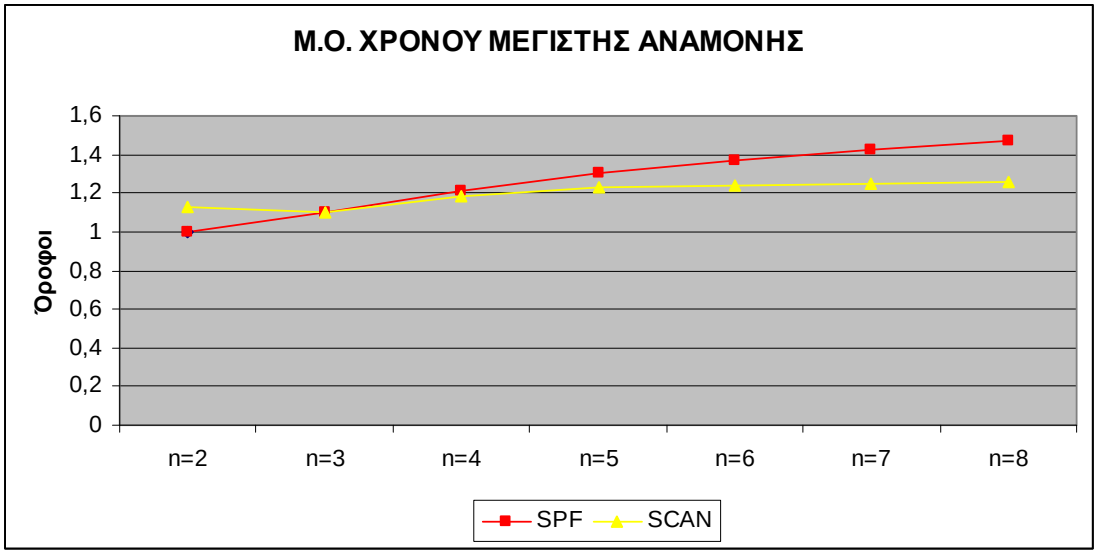
	n=2	n=3	n=4	n=5	n=6	n=7	n=8
FIFO	0,5	1,777	3,75	6,4	9,722	13,714	18,375
SPF	0,5	1,407	2,492	3,697	4,974	6,304	7,67
SCAN	0,5	1,407	2,851	4,702	6,725	9,071	11,749



Σαν ένα πρώτο συμπέρασμα προκύπτει ότι ο αλγόριθμος που εξυπηρετεί όλους τους ορόφους κάνοντας τη μικρότερη απόσταση είναι ο SPF. Επόμενος έρχεται ο Scan αφήνοντας τελευταίο τον FIFO.

Σύγκριση χρόνου μέγιστης αναμονής:

	n=2	n=3	n=4	n=5	n=6	n=7	n=8
SPF	1	1,098	1,214	1,304	1,372	1,424	1,466
SCAN	1,125	1,098	1,185	1,227	1,238	1,246	1,262



Από αυτή τη σύγκριση φαίνεται ότι ο αλγόριθμος SPF μόνο στη χαμηλότερη τιμή του n έχει μικρότερη καθυστέρηση, ενώ από το κοινό τους σημείο κι έπειτα ο Scan είναι ο αλγόριθμος που εξυπηρετεί με τη μικρότερη καθυστέρηση τις κλήσεις.

ΚΕΦΑΛΑΙΟ 5^{Ο}

5.1 Συμπεράσματα

Έχοντας ελέγξει την απόδοση των αλγορίθμων για οκτώ ορόφους γίνεται κατανοητό ότι μπορούμε να βγάλουμε ασφαλέστατα συμπεράσματα. Με βάση τα παραπάνω αποτελέσματα έχουμε μια σαφή εικόνα για τις αποδόσεις των αλγορίθμων σε κάθε όροφο. Αρχικά θα αναλύσουμε τα αποτελέσματα τους με βάση τη σύγκριση των αποστάσεων που έγινε στο προηγούμενο κεφάλαιο και έπειτα θα ασχοληθούμε με την ανάλυση των αποτελεσμάτων της σύγκρισης του μέγιστου χρόνου αναμονής.

Στη σύγκριση των αποστάσεων είναι εμφανές ότι ο First In First Out υστερεί από τους άλλους δύο σε οποιαδήποτε τιμή του n. Σε όλες τις τιμές που ερευνήσαμε αυτός είναι πάντα ο αλγόριθμος που εμφανίζει το μεγαλύτερο μέσο όρο εξυπηρέτησης, έχοντας μάλιστα στις περισσότερες των περιπτώσεων μεγάλη διαφορά από τους υπόλοιπους. Ο επόμενος στην κατάταξη μας είναι ο αλγόριθμος Scan. Είναι σαφέστατα πολύ πιο βελτιωμένος σε σχέση με τον FIFO. Όμως κι αυτός σε καμία τιμή δεν έχει χαμηλότερο μέσο όρο από τον αλγόριθμο Shortest Path First. Ενώ ο Scan είναι αρκετά ανταγωνιστικός, ο SPF είναι σε όλες τις περιπτώσεις ο αλγόριθμος που εμφανίζει τον χαμηλότερο μέσο όρο εξυπηρέτησης.

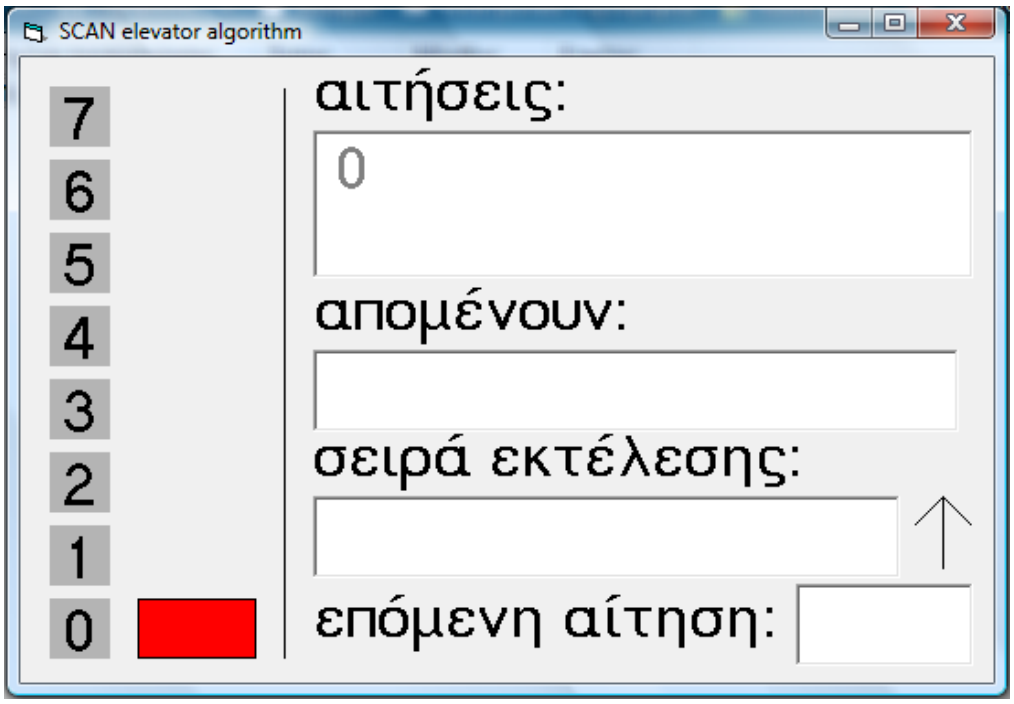
Στην περίπτωση της σύγκρισης του μέγιστου χρόνου αναμονής τα αποτελέσματα διαφοροποιούνται. Ο χρήστης που περίμενε την περισσότερη ώρα μέχρι να εξυπηρετηθεί είναι αυτός του οποίου ο ανελκυστήρας λειτουργεί με βάση τον αλγόριθμο SPF. Ο SPF είναι αποδοτικότερος του Scan μόνο για n=2 και από κει και πέρα ο χρόνος αναμονής διαρκώς αυξάνεται. Ο Scan από n=2 για n=3 παρουσιάζει μια μείωση του χρόνου αναμονής, ενώ από n=3 και μετά εμφανίζει μια μικρή άνοδο του χρόνου αναμονής μικρότερη όμως από την αντίστοιχη του SPF.

Μιας και το ζητούμενό μας είναι η άμεση εξυπηρέτηση του χρήστη αποδοτικότερος κρίνεται ο αλγόριθμος Scan, καθώς αυτός σε σχέση τόσο με τον SPF όσο και με τον FIFO έχει και τον μικρότερο μέσο όρο αναμονής και έναν σχετικά καλό χρόνο συνολικής εξυπηρέτησης των κλήσεων που έχει δεχτεί.

5.2 Ανάπτυξη εφαρμογής

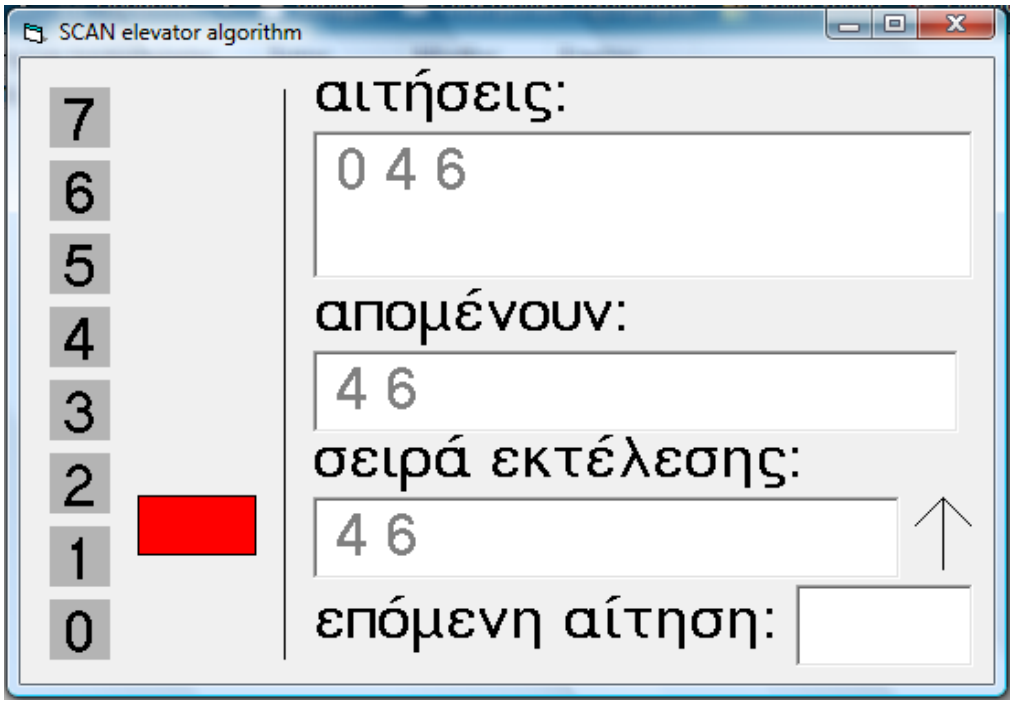
Έχοντας επιχειρηματολογήσει γιατί ο αλγόριθμος Scan είναι ο αποδοτικότερος, δημιούργησα μια εφαρμογή, η οποία λειτουργεί σύμφωνα με αυτόν. Αυτή η εφαρμογή υλοποιήθηκε στο προγραμματιστικό περιβάλλον της Visual Basic. Έτσι θα γίνει ακόμα περισσότερο κατανοητός ο τρόπος με τον οποίο διαχειρίζεται τις κλήσεις και τις φέρνει εις πέρας. Ας δούμε λοιπόν πως λειτουργεί αυτή η εφαρμογή.

Η αρχική της μορφή φαίνεται στην παρακάτω εικόνα.

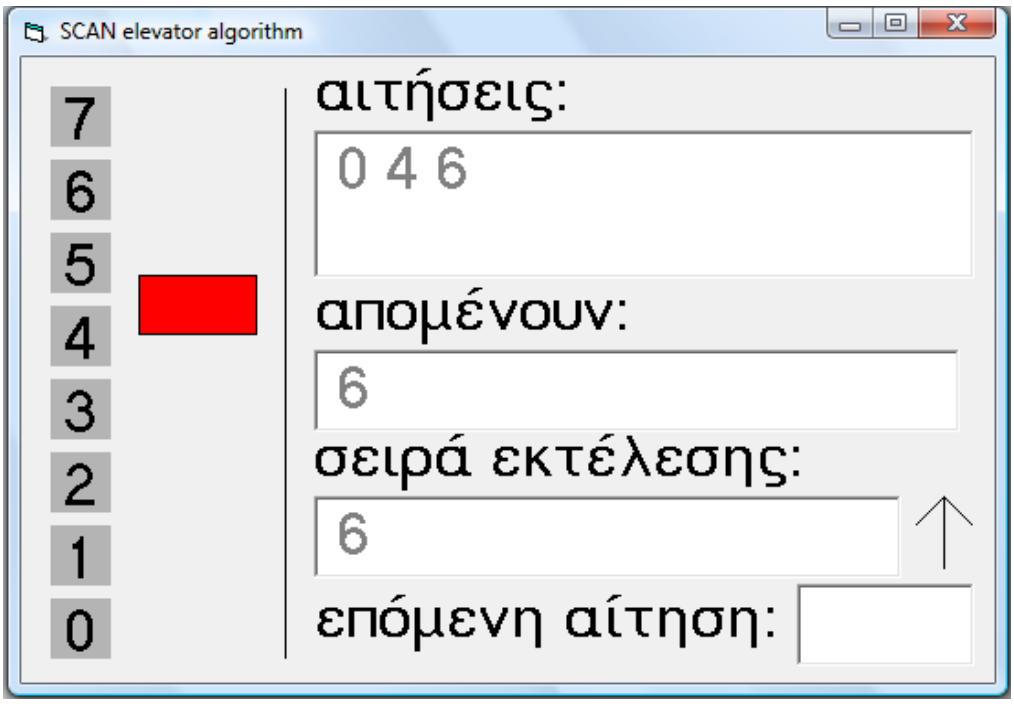


Στο αριστερό μέρος έχουμε ορίσει με πόσους ορόφους θα ασχοληθούμε. Όπως φαίνεται θα αναφερθούμε σε οκτώ ορόφους ή αλλιώς σε μια επταώροφη πολυκατοικία. Το κόκκινο «κουτάκι» ουσιαστικά αποτελεί τον θάλαμο του ανελκυστήρα και θα κινείται προς τα πάνω ή προς τα κάτω ανάλογα με τις αιτήσεις που θα έχει στην αναμονή. Τις αιτήσεις αυτές θα τις δέχεται από το πληκτρολόγιο και θα αναγράφονται στο κάτω δεξιά πλαίσιο με την ονομασία «επόμενη αίτηση». Εάν οι κλήσεις που εκκρεμούν είναι περισσότερες από μια, στο πλαίσιο «σειρά εκτέλεσης» θα αναγράφεται η σειρά με την οποία αυτές θα εξυπηρετηθούν. Στο πλαίσιο με την ονομασία «απομένουν» θα φαίνεται ο αριθμός του κάθε ορόφου, που έχει καλεστεί ο ανελκυστήρας, με τη χρονική σειρά που έγινε η κλήση. Τέλος το πλαίσιο «αιτήσεις» θα δείχνει όλες τις αιτήσεις που έχουν ζητηθεί από τον ανελκυστήρα.

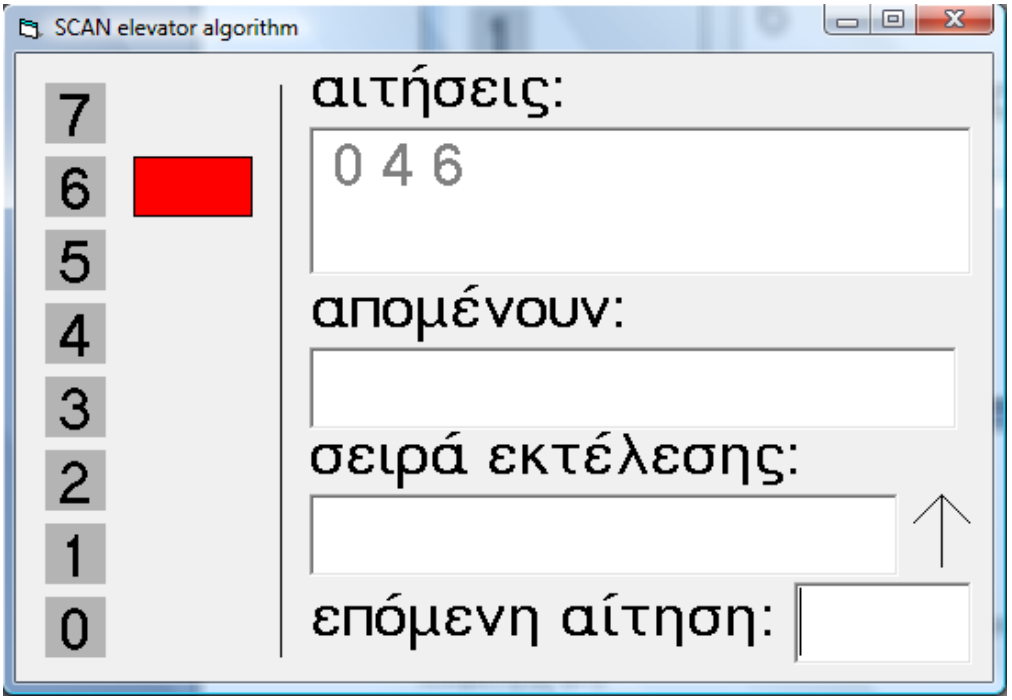
Ας προχωρήσουμε και σε ένα παράδειγμα. Καθώς ο ανελκυστήρας βρίσκεται στο ισόγειο δέχεται αιτήσεις για τον τέταρτο και τον έκτο όροφο. Αμέσως ξεκινάει την πορεία του προς τα πάνω με σκοπό να εξυπηρετήσει τον τέταρτο όροφο.



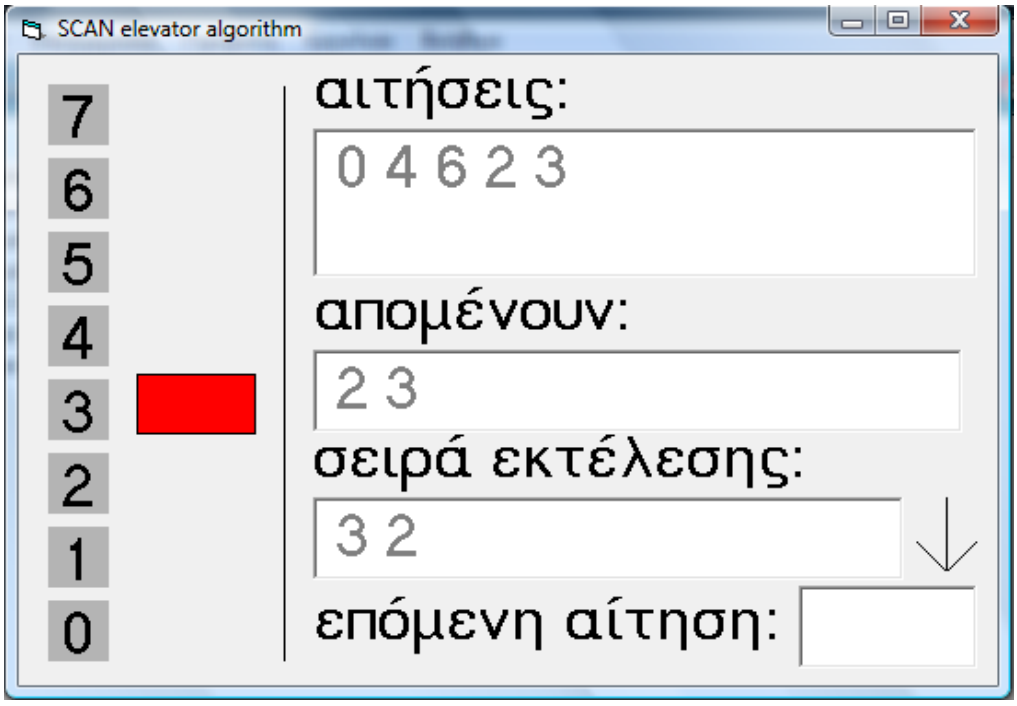
Αμέσως μόλις εξυπηρετήσει την αίτηση στον τέταρτο συνεχίζει την ίδια πορεία για να φτάσει και στον έκτο.



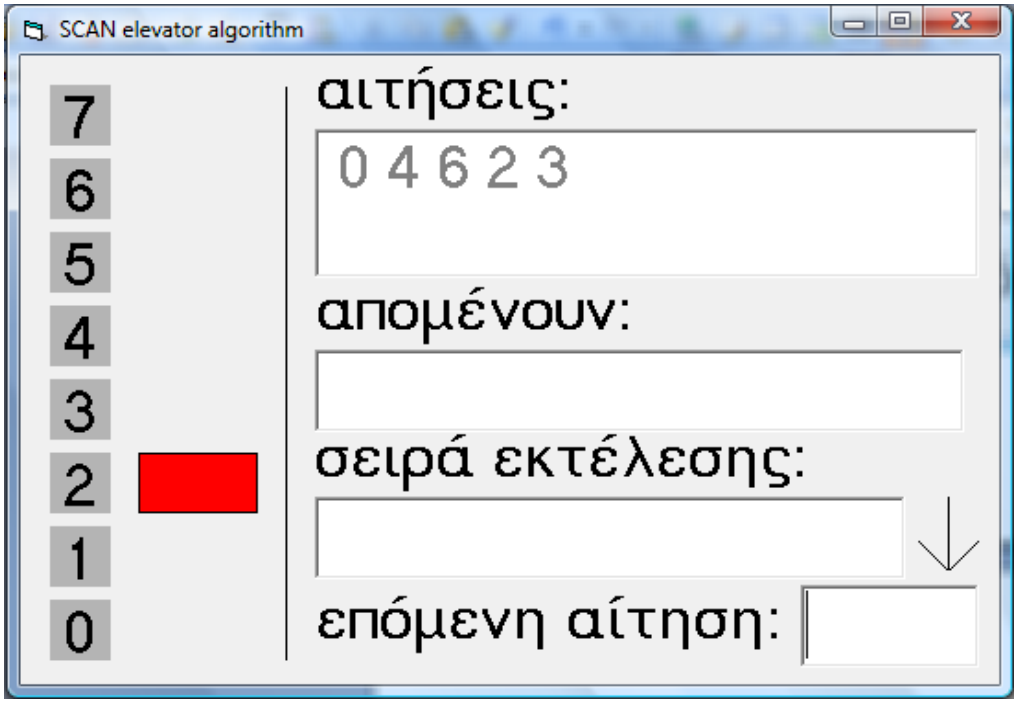
Μόλις εκπληρώσει και την αίτηση του έκτου παραμένει εκεί αναμένοντας την επόμενη κλήση.



Αν η επόμενη αίτηση αφορά χαμηλότερο όροφο τότε ο ανελκυστήρας θα αλλάξει φορά και θα κατευθυνθεί προς τα κάτω. Ας τον καλέσουμε στο δεύτερο για να δούμε και τη καθοδική του πορεία και πριν προλάβει να φτάσει εκεί θα τον καλέσουμε και στον τρίτο. Έτσι θα δούμε ότι θα εξυπηρετηθεί αρχικά ο τρίτος κι έπειτα ο δεύτερος άσχετα αν αυτός δηλώθηκε πρότερα (όπως φαίνεται και από το πλαίσιο της σειράς εκτέλεσης).



Μόλις ολοκληρώσει κι αυτήν την διαδικασία θα παραμείνει σε αδράνεια στον δεύτερο όροφο περιμένοντας τις επόμενες αιτήσεις.



Αυτό ήταν ένα μικρό, αλλά και κατατοπιστικό παράδειγμα για την λειτουργία τις συγκεκριμένης εφαρμογής που βασίζεται στον αλγόριθμο Scan. Κατ' αυτόν τον τρόπο θα λειτουργούσε και σε έναν πραγματικό ανελκυστήρα, εάν κάναμε φυσικά τις κατάλληλες συνδέσεις.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία – Άρθρα

- Michael Halvorson, (2004). Microsoft Visual Basic 6.0 Professional Βήμα Βήμα. Αθήνα: Εκδόσεις Κλειδάριθμος.
- Εφημερίδα «Έθνος» (30/11/2006).

Διαδίκτυο

www.disabled.gr

www.geocities.com

www.livepedia.gr

www.kleemann.gr

users.att.sch.gr/estoikos

infoman.teikav.edu.gr

www.wikipedia.org

www.el.wikipedia.org