Πανεπιστήμιο Θεσσαλίας

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών Τηλεπικοινωνιών και Δικτύων

Έξυπνοι και Αλληλεπιδρώμενοι πράκτορες e-Learning (Smartive e-Learning Agents)

(Smart and Interactive e-Learning Agents)

ΛΑΚΗΣ ΜΟΣΧΟΣ

Επιβλέποντες καθηγητές:
Ασπασία Δασκαλοπούλου,
Χρήστος Δ. Αντωνόπουλος

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τούς καθηγητές μου, την βασικό επιβλέποντα της διπλωματικής μου εργασίας, κυρία Ασπασία Δασκαλοπούλου, για την υποστήριξη και τις πολύτιμες συμβουλές που με βοήθησαν να ολοκληρώσω την μελέτη αυτή, και τον κύριο Χρήστο Αντωνόπουλο που με βοήθησε στην επιλογή των καταλλήλων εργαλείων που χρησιμοποίησα στην εφαρμογή του προγράμματος.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου που μου συμπαραστάθηκαν σε όλη την διάρκεια εκπόνησης της εργασίας και των σπουδών μου.

Περιεχόμενα

Σύντομή περίληψη

Σε αυτή την εργασία παρουσιάζω μία μέθοδο επιβεβαίωσης της κατανόησης συγκεκριμένης σχολικής ύλης που έχει διδαχθεί μέσα στην σχολική χρονιά. Η εφαρμογή της μεθόδου επιβεβαίωσης μπορεί να γίνει κατά τη διάρκεια ή/και στο τέλος της σχολικής χρονιάς. Η μέθοδο αποτελείται από έναν πράκτορα, ο οποίος παίρνει κάποιες αποφάσεις, και έναν αλγόριθμο αναζήτησης και εκτέλεσης αυτών των αποφάσεων.
Η υλοποίηση της μεθόδου έγινε μέσω ενός λογισμικού που γράφτηκε στις γλώσσες προγραμματισμού C/C++.

Abstract

In this thesis we developed a method to ensure that a certain school subject that has been taught during the school curriculum is fully understandable.

This method consists of a smart agent that takes certain decisions and of an algorithm that searches and executes these decisions.

The development of this method was done by a software written in C/C++ programming languages.

Αρχικά

Έξυπνοι και Αλληλεπιδρώμενοι πράκτορες e-Learning (Smartive e-Learning Agents, Smart and Interactive e-Learning Agents) είναι ο τίτλος της διπλωματικής εργασίας και σίγουρα χρωστάμε αρχικά κάποια εξήγηση για αυτόν τον περίεργο τίτλο. Πρόκειται για ένα σύστημα το οποίο χρησιμοποιεί έξυπνους πράκτορες λογισμικού για την αλληλεπίδραση με τον χρήστη.

Πιο συγκεκριμένα, χρησιμοποιούμε ένα υπολογιστικό σύστημα για να περιγράψουμε μια συγκεκριμένη συμπεριφορά όταν το σύστημα παίρνει κάποια δεδομένα από το περιβάλλον του.

Τα δεδομένα αυτά τα παίρνει μέσω της αλληλεπίδρασης με τον χρήστη. Ο χρήστης αρχικά παίρνει από την οθόνη συγκεκριμένες πληροφορίες (εικόνα, κείμενο, κτλ) και αντιδρά σύμφωνα με αυτά που το ζητά το σύστημα. Το περιβάλλον στο όποιο αναφερόμαστε αποτελείται από τον χρήστη και το σύστημα, δηλαδή το λογισμικό.

Τι είναι ένας πράκτορας

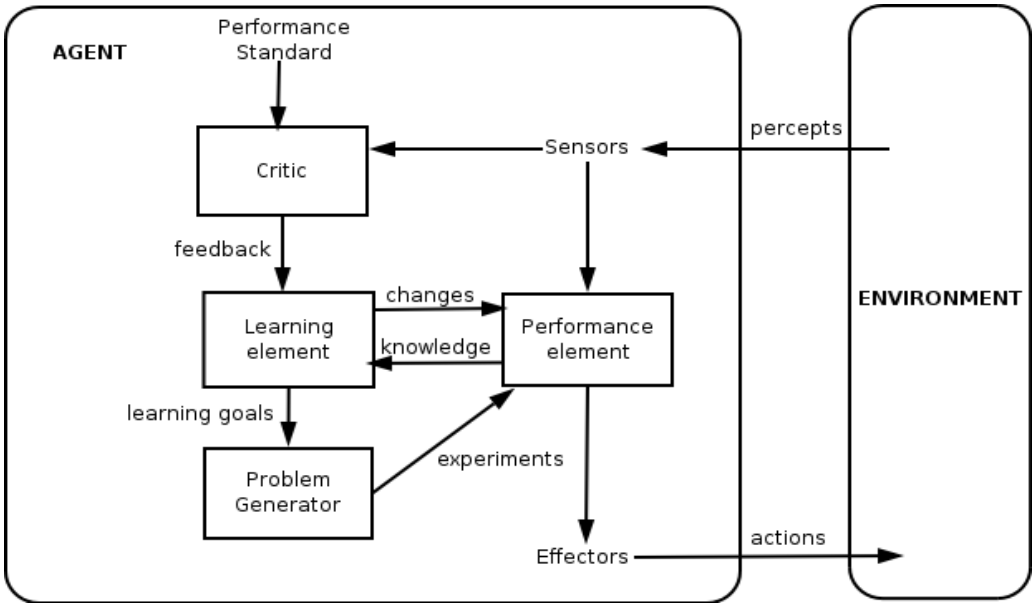
” An agent is a computer system located in a particular environment and which is capable of conducting independent operations in this environment to achieve its objectives.” © Wooldridge: *Intelligent Agents: The Key Concepts.* 2002

Πράκτορας είναι ένα υπολογιστικό σύστημα που βρίσκεται μέσα σε κάποιο περιβάλλον και είναι ικανό για αυτόνομη δράση μέσα σε αυτό, ώστε να εκπληρώσει τους στόχους για τους οποίους σχεδιάστηκε. Ανάλογα με τον τομέα εφαρμογής και των χαρακτηριστικών αυτών των εφαρμογών, διαφορετικά χαρακτηριστικά θεωρούνται σημαντικά για να χαρακτηρίζεται ένα πρόγραμμα λογισμικού ως πράκτορας. Αυτά τα χαρακτηριστικά μπορεί να είναι :

1.) **Αντιδραστικότητα** (ορθολογισμός), όπου οι νοήμονες πράκτορες μπορούν να αντιλαμβάνονται το περιβάλλον τους και να απαντούν με τον καλύτερο δυνατό τρόπο, εντός λογικού χρόνου σε αλλαγές που συμβαίνουν σε αυτό, με σκοπό να εκπληρώσουν τους στόχους τους.

2.) **Ενεργητικότητα** (αυτονομία), όπου οι νοήμονες πράκτορες μπορούν να πάρουν πρωτοβουλίες και να ενεργήσουν προς την εκπλήρωση των στόχων τους

3.) **Κοινωνικότητα,** όπου οι νοήμονες πράκτορες μπορούν να αλληλεπιδρούν με άλλους πράκτορες (και ανθρώπους) για να εκπληρώσουν τους στόχους τους. Αυτή η αλληλεπίδραση δεν περιορίζεται μόνο στην ανταλλαγή δεδομένων αλλά έχει χαρακτηριστικά που την κάνουν να μοιάζει με ανθρώπινη αλληλεπίδραση π.χ. για την σύναψη συμφωνιών, τη δημιουργία συνεργασιών, το συντονισμό ενεργειών με άλλους πράκτορες.



Οι *SMART* Agents, έξυπνοι πράκτορες, είναι μια νέα μορφή του πράκτορα λογισμικού που διασυνδέονται με άλλους παράγοντες που αποτελούν σύστημα τεχνητής νοημοσύνης. Το ακρωνύμιο "SMART" σημαίνει "σύστημα για τη διαχείριση πρακτόρων σε Πραγματικό Χρόνο". Αυτό είναι λίγο δυσνόητο εφόσον οι πράκτορες διαχειρίζονται οι ίδιοι ο ένας τον άλλον με τη συμφωνία να γίνουν μέρος της συλλογικής συνόλου. Οι SMART πράκτορες συνεργάζονται, μέσα σε ένα σύστημα SMART, ώστε να εκτελέσουν μικρότερα κομμάτια των μεγαλύτερων εργασίων προγραμματισμού, ούτως ώστε η συνδυασμένη συλλογική να μπορεί να επιτύχει σπουδαία πράγματα με σχετικά απλές δομικές μονάδες προγραμματισμού. Η βασική ιδέα του SMART είναι ότι κάθε μεμονωμένος παράγοντας δεν χρειάζεται να είναι "έξυπνος". Δουλεύοντας μαζί με έναν έξυπνο τρόπο, οι πράκτορες δημιουργούν ένα είδος αναδυόμενης νοημοσύνης που μπορεί να φαίνεται ότι εμφανίζουν έκθεμα νοημοσύνης. Ευφυείς πράκτορες υπάρχουν εδώ και πολύ καιρό σε διάφορες μορφές. Ο όρος «Ευφυής πράκτορας μπορεί να αναφέρεται σε κάθε παράγοντα που θα παρουσιάζει κάποιο ποσό της νοημοσύνης και δεν υπάρχει καμία απαίτηση ότι ο

πράκτορας έχει την ικανότητα να εργάζεται με άλλους πράκτορες. Επομένως, έξυπνοι πράκτορες είναι ένα συγκεκριμένο είδος Ευφυούς Νου, όπου η έμφαση είναι στο πώς οι πράκτορες επικοινωνούν μεταξύ τους και όχι αν κάθε πράκτορας μπορεί να επιτύχει μια σημαντική ποσότητα πληροφοριών. Το σύστημα είναι αυτό που πρέπει να επιτύχει την ευφυΐα χρησιμοποιώντας σχετικά μη- ευφυή δομικά στοιχεία.

Κάθε έξυπνος πράκτορας μπορεί να προγραμματιστεί για την επίτευξη συγκεκριμένων δοθέντων δεδομένων δίνοντας του έτσι μια μορφή λειτουργικότητας την οποία μπορούν να διοχετεύσουν σε ένα έξυπνο σύστημα, επιτρέποντας στους άλλους πράκτορες να συνεργαστούν για να επιτύχουν πράγματα που κάθε πράκτορας μόνος του δεν μπορεί.

Ένα έξυπνο σύστημα μπορεί να κατασκευαστεί χρησιμοποιώντας οποιαδήποτε γλώσσα προγραμματισμού και επικοινωνιακής υποδομής. Εμπνευσμένο από τη δημοφιλή τηλεοπτική σειρά της δεκαετίας του 1960 Get Smart, κάθε πράκτορας μπορεί να γίνει πιο έξυπνος χρησιμοποιώντας άλλους πράκτορες για την εκτέλεση διαφόρων εργασιών. Στην τηλεοπτική σειρά, ο πράκτορας 86 έπαιξε τον κύριος ρόλο και πήρε όλα τα εύσημα, ενώ ο πράκτορας 99 έκανε όλη την πραγματική δουλεία.

Ηλεκτρονική μάθηση ή e-Learning

Ηλεκτρονική μάθηση είναι η διαδικασία κατά την οποία κάποιος μαθαίνει – εκπαιδεύεται με την χρήση ηλεκτρονικών υπολογιστών.

Ευρέως διαδεδομένος είναι ο διεθνής όρος *e-learning* για την ηλεκτρονική μάθηση. Η εκπαίδευση με αυτή την διαδικασία μπορεί να χωριστεί σε εκπαίδευση με σύνδεση (online) και εκπαίδευση χωρίς σύνδεση (offline). Ηλεκτρονική μάθηση με σύνδεση είναι η προβολή εκπαιδευτικού υλικού μέσω Ιντερνέτ (από κάποιο δικτυακό τόπο) ενώ ηλεκτρονική μάθηση χωρίς σύνδεση είναι η προβολή εκπαιδευτικού υλικού αποθηκευμένου στον υπολογιστή μας, εκπαιδευτικά cdrom κ.τ.λ. Πρέπει λοιπόν αρχικά να αποφασίσουμε ποια τεχνολογία και ποια μορφή εκπαίδευσης θα χρησιμοποιήσουμε.

Για την λήψη αυτής της απόφασης πρέπει να συνυπολογίσουμε δύο παραμέτρους :

1.) Μορφή υλοποιήσης, δηλαδή εκπαίδευση με σύνδεση (online) ή εκπαίδευση χωρίς σύνδεση (offline).

2.) Τεχνογνωσία, δηλαδή τι είδους εργαλεία (γλωσσα προγραμματισμού κτλ.) θα χρησιμοποιήσω.

Quo Vadis?

Η παρούσα διπλωματική έχει ως κύριο αντικείμενο την εκπαίδευση χωρίς σύνδεση, δηλαδή ένα λογισμικό το οποίο μπορεί να είναι αποθηκευμένο στον σκληρό δίσκο του υπολογιστικού συστήματος ή σε οπτικό δίσκο που χρησιμοποιείται για αποθήκευση ψηφιακών δεδομένων.

Στόχος λοιπόν είναι να υλοποιήσουμε ένα αλληλεπιδραστικό περιβάλλον έτσι ώστε να μπορεί να συμμετέχει ο χρήστης ευχάριστα στην διαδικασία του αυτοέλεγχου, δηλαδή αν πραγματικά έχει κατανοήσει την ύλη που έχει διδαχτεί. Θέλουμε να βοηθήσουμε στην διαδικασία της αναζήτησης των προβλημάτων στην μάθηση, έτσι ώστε να εντοπίσουμε γρήγορα και εύκολα σε ποια σημεία της ύλης δυσκολεύεται ο χρήστης. Με το συγκεκριμένο λογισμικό μπορεί μόνος του να κάνει αυτή την αναζήτηση και να διαπιστώσει αν υπάρχουν ή όχι προβλήματα κατανόησης.

Εισαγωγή στην Εφαρμογή

Σε αυτήν την διπλωματική γίνεται η εφαρμογή του παραπάνω συστήματος στην εκμάθηση μαθηματικών της ΣΤ' τάξης δημοτικού. Βάση της απάντησης που δίνει ο χρήστης, αποφασίζει ένας πράκτορας αν θα προχωρήσει στην ύλη ή αν θα πρέπει να επιμείνει στο ίδιο κεφάλαιο στο οποίο βρίσκεται. Αν ο χρήστης απαντάει λάθος στις ερωτήσεις , τότε ο πράκτορας αποφασίζει ότι δεν έχει γίνει κατανοητή η ύλη και παραμένει στο ίδιο κεφάλαιο. Σε διαφορετική περίπτωση προχωρεί στο επόμενο επίπεδο. Το λογισμικό αποτελείται από έξι κεφάλαια και κάθε κεφάλαιο περιέχει έξι ερωτήσεις για το πρώτο πέρασμα και έξι για το δεύτερο. Δηλαδή ένα σετ ασκήσεων την πρώτη φορά και ένα δεύτερο την δεύτερη φορά, τα οποία είναι ξένα μεταξύ τους.

Η συνάρτηση που αναλαμβάνει τον υπολογισμό των επιτυχιών, βάση των οποίων παίρνει αποφάσεις το σύστημα (περνάει / δεν περνάει), είναι στην ουσία το κομμάτι του προγράμματος από το οποίο αποτελείται ο λογισμικός πράκτορας.

Η διεπαφή θα είναι απλή στον χρήστη, ο οποίος στην αρχή θα βλέπει τις έξι ερωτήσεις, τις οποίες καλείται να διαβάσει μια προς μια προσεκτικά, και αφού έχει βεβαιωθεί ότι έχει τις καταλάβει θα μπορέσει στη συνέχεια να δώσει τις απαντήσεις

του. Οι ερωτήσεις θα είναι πολλαπλής επιλογής και ο χρήστης θα εισάγει αριθμούς (1-3) για μία από τις 3 πιθανές απαντήσεις.

Περιγραφή

Το λογισμικό κάνει ερωτήσεις, οι οποίες βρίσκονται αποθηκευμένες σε αρχείο κειμένου. Σύμφωνα με τις απαντήσεις που παίρνει το λογισμικό πρέπει να παίρνει αποφάσεις για την επόμενη κίνηση του προγράμματος. Αν έχει "καταλάβει" ο πράκτορας ότι ο χρήστης έχει κατανοήσει την ύλη θα προχωρήσει στο επόμενο επίπεδο. Σε διαφορετική περίπτωση θα πρέπει να πάρει απόφαση είτε να επιμείνει στο ίδιο επίπεδο και να κάνει ξανά ερωτήσεις (που αυτή την φορά όμως θα είναι διαφορετικές), είτε να οπισθοδρομήσει στην άμεσα συσχετιζόμενη ύλη.

Όπως αναφέραμε προηγουμένως, ο πράκτορας είναι στην η ουσία μια συνάρτηση, με την οποία καταλαβαίνει το πρόγραμμα αν περνάει ο χρήστης ή όχι στην επόμενη φάση. Ουσιαστικά θέλουμε να αποφύγουμε τυχαίες απαντήσεις και τυχαία αποτελέσματα. Κάθε φορά επιστρέφει η συνάρτηση του κάθε επίπεδου μια boolean τιμή (0/1 equal F/T). Κάθε επίπεδο (κεφάλαιο) θα έχει 6 ερωτήσεις με διαφορετική βαρύτητα. (2 ερωτήσεις ανά 10 βαθμούς, 2 ερωτήσεις ανά 15 βαθμούς και 2 ερωτήσεις ανά 25 βαθμούς, συνολικά 100 βαθμούς). Με αυτόν τον τρόπο αποφεύγουμε τυχαίες απαντήσεις. Επομένως το σύστημα θα εξετάζει δύο παράγοντες αξιολόγησης :

1.) Συνδυασμός από bit για το αν απαντήθηκαν σωστά οι ερωτήσεις, (για παράδειγμα 0,1,1,1,0,1).

2.) Το συνολικό άθροισμα των βαθμών βαρύτητας.

Στη συνέχεια γίνεται η αριθμητική πράξη βαθμοί επί πλήθος σωστών απαντήσεων. Ανάλογα με το αποτέλεσμα θα παίρνει απόφαση για το επόμενο βήμα. Άρα θα έχουμε 3 κατηγορίες ερωτήσεων (A', B', C'). Κάθε κεφάλαιο αποτελείται από δύο σετ ερωτήσεων, ένα σετ για την πρώτη φάση και ένα δεύτερο σετ στο οποίο ανατρέχει μόνο σε περίπτωση που είτε αποτύχει ο χρήστης/μαθητής στο παρόν κεφάλαιο είτε το σύστημα τον επιστρέψει σε παλαιότερο συσχετιζόμενο κεφάλαιο.

Οι ενότητες (κεφάλαια), σχετίζονται μεταξύ τους και κάθε μια κτίζει πάνω από μια άλλη. Ξεκινώντας από την πρώτη ενότητα, χτίζουμε πάνω από αυτή την δεύτερη ενότητα και πάνω από αυτή την τρίτη μέχρι και την έκτη ενότητα. Όμως υπάρχουν και ενότητες που είναι προαπαιτούμενες για κάποιες άλλες ενότητες. Για παράδειγμα, η ενότητα που μιλάει για τον πολλαπλασιασμό έχει προαπαιτούμενη ενότητα την ενότητα με τα αθροίσματα και αυτή με την σειρά της έχει ως προαπαιτούμενη την ενότητα με τους αριθμούς. Άρα, όσο πηγαίνουμε μπροστά στα κεφάλαια, διαπερνάμε όλες τις ενότητες, αν τις έχουμε βέβαια επιλέξει όλες, όμως αν αποτύχουμε σε μια ενότητα πρέπει αναγκαστικά να προσπαθήσουμε να επιτύχουμε στην προαπαιτούμενη ενότητα.

Στο τέλος του προγράμματος φτάνουμε όταν έχουμε περάσει όλες τις ενότητες που έχουμε επιλέξει να προσπαθήσουμε ή αν αποτύχουμε σε μια ενότητα δυο φορές.

Εφαρμογή και υλοποίηση του λογισμικού

Το λειτουργικό γράφτηκε στις γλώσσες προγραμματισμού C/C++. Ως περιβάλλον ανάπτυξης (*integrated development environment*, IDE) χρησιμοποίησα το **Code::Blocks** (http://www.codeblocks.org/).

Στην συνέχεια θα δώσω μια σύντομη περιγραφή του κώδικα και κάθε μέρη του ξεχωριστά. Προσπάθησα να εξηγήσω τον τρόπο με τον οποίο σκέφτηκα να υλοποιήσω τον αλγόριθμο σε κώδικα.

Ορισμός τον ενοτήτων

Τα κεφάλαια, <<chapter_i>>, όπου i = 1-6, θα έχουν ως περιεχόμενο τα εξής:

<<chapter_**1**>>: **Αριθμοί**

<<chapter_**2**>>: **Πρόσθεση**

<<chapter_**3**>>: **Αφαίρεση**

<<chapter_**4**>>: **Πολλαπλασιασμός**

<<chapter_**5**>>: **Διαίρεση**

<<chapter_**6**>>: **Γεωμετρία**

Προαπαιτούμενες γνώσεις ανά κεφάλαιο:

1. **Αριθμοί:** Τίποτα, είναι η βασική γνώση, Δάσκαλος / Βιβλίο
2. **Πρόσθεση**: Αριθμοί
3. **Αφαίρεση**: Αριθμοί, Πρόσθεση
4. **Πολλαπλασιασμός**: Αριθμοί, Πρόσθεση
5. **Διαίρεση**: Αριθμοί, Πολλαπλασιασμός
6. **Γεωμετρία**: Αριθμοί, Πρόσθεση, Αφαίρεση, Πολλαπλασιασμός, Διαίρεση

Το πρόγραμμα θα απαρτίζεται από :

1.) την κύρια συνάρτηση, που είναι η **main**,

2.) μια συνάρτηση που θα περιέχει όλα τα αρχεία προς εκτέλεση, που θα λέγεται **files**,

3.) και μια συνάρτηση που θα περιέχει όλες τις εφαρμογές και την κλάση του προγράμματος, που θα λέγεται **implementation**.

Οι συναρτήσεις **files** και **implementation** θα είναι συναρτήσεις τύπου **header**. Όλα τα αρχεία με τις ερωτήσεις και τις απαντήσεις είναι αποθηκευμένα σε αρχεία κειμένου και βρίσκονται σε ένα φάκελο με όνομα **Chapters**.

Κύρια Συνάρτηση main (Main.cpp)

Στην main έχουμε ορίσει όλα τα τμήματα που χρειαζόμαστε για το πρόγραμμα μας. Επίσης στην main γίνεται όλη η διασύνδεση των κεφαλαίων του προγράμματος, δηλαδή ο αλγόριθμος.

Για την επιλογή και το πέρασμα των κεφαλαίων, χρησιμοποιώ vectors που είναι ένα εργαλείο της γλώσσας C++. Ουσιαστικά είναι ένας εξορισμού δυναμικός πίνακας οποίος αποθηκεύει τα αντικείμενα που έχω φτιάξει.

```
Chapters chapter;
 vector<Chapters> Lessons;
 for(int l=0; l<6; l++) {
     Lessons.push_back(chapter);
     Lessons[l].SetId(l);
     Lessons[l].trys = 0;
     Lessons[l].pass = false;
     Lessons[l].hasNext = false;
     Lessons[l].next = 1; }
```

files.h

Το files.h είναι ένα αρχείο επικεφαλίδας, το οποίο περιέχει τον μηχανισμό διαβάσματος / γραψίματος από / σε αρχείο κειμένου και τον μηχανισμό αξιολόγησης δεδομένων εισόδου.

Οι βασικές συναρτήσεις είναι οι:

```
int PrintQuestionChjSei();

int PrintScorej_i(int SumChjSei);

j=1,2,3,4,5,6,   i= 1,2
```

implementation.h

Το implementation.h είναι ένα αρχείο επικεφαλίδας, το οποίο περιέχει τον ορισμό της κλάσης και όλους τους ορισμούς των μεθόδων που περιέχουν την λειτουργικότητα του προγράμματος.

```
bool getNextChapter(vector<Chapters> &Lessons, int &chID);

int PrintQuestion(int chID, int secID);

int PrintScore(int chID, int secID, int ans);
```

```
void sort(vector<int> &array);

void delList(vector<int> &array,int &count,int select);

int* getSelects(int &lessLen, bool &exit);
```

Μενού

Στο μενού θα επιλέγει ποια κεφάλαια θέλει να δουλέψει ο χρήστης (όλα / κάποια), θα βλέπει μία λίστα με τις επικεφαλίδες των κεφαλαίων και δίπλα (αριστερά) από αυτά θα έχει νούμερα (απαρίθμηση κεφαλαίων (1-6) και έξοδο (7)). Το πρόγραμμα θα του ζητάει να εισάγει τις επιλογές του μέχρι που να εισάγει ένα σύμβολο που υποδηλώνει το τέλος τον επιλογών του. Ο χρήστης θα εισάγει τις επιλογές του, μια προς μια, και κάθε φορά θα πρέπει να επιβεβαιώνουν την επιλογή τού πατώντας <<enter>>. Άμα προσπαθήσει να εισάγει δυο φορές (ή παραπάνω) το ίδιο κεφάλαιο τότε θα πρέπει να εμφανιστεί ένα μήνυμα λάθους, δίνοντας ξανά την ευκαιρία στον χρήστη να εισάγει την επιλογή του. Άμα εισάγει κάτι διαφορετικό από τις επιλογές που υπάρχουν, τότε θα πρέπει να εμφανιστεί ένα μήνυμα λάθους, και θα έχει ξανά την ευκαιρία να εισάγει την επιλογή του. Επίσης θα του δείχνει την κατάσταση των επιλογών του που έχει κάνει έως εκείνη την στιγμή και θα πρέπει να έχει την δυνατότητα να διαγράφει μια επιλογή από την λίστα τον επιλογών του. Οπότε θα είναι τα κεφάλαια (1-6), διαγραφή κάποιας επιλογής (7), έξοδος (8) και εκτέλεση (9) ως επιλογές στο μενού.

Αυτά που θα βλέπει ο χρήστης θα είναι ορισμένα σε μία function, void ShowMenu(), και θα παρέχει τα εξής:

(1) **Αριθμοί**

(2) **Πρόσθεση**

(3) **Αφαίρεση**

(4) **Πολλαπλασιασμός**

(5) **Διαίρεση**

(6) **Γεωμετρία**

(7) **Διαγραφεί**

(8) **Έξοδος**

(9)**Εκτέλεση**

Αρχεία και τρόπος μεταβιβάσεις στις οθόνες

Τα δεδομένα εισόδου και εξόδου θα εκτυπώνονται και θα αποθηκεύονται σε αρχεία, τα οποία θα εισάγονται (<<stream>>) εξωτερικά στο πρόγραμμα. Επίσης θα υπάρχει ένα αρχείο με έτοιμες (σωστές) απαντήσεις, οι οποίες θα είναι σημείο αναφοράς όταν θα εισάγει ο χρήστης τις απαντήσεις του, δηλαδή θα γίνεται σύγκριση μεταξύ των

απαντήσεων που έδωσε ο χρήστης και των έτοιμων (σωστών) απαντήσεων που ήδη υπάρχουν.

Τα αρχεία με τις ερωτήσεις που θα εκτυπώνονται στην οθόνη τα ονομάζουμε <<**output.txt**>>.

Τα αρχεία με τις απαντήσεις που θα παίρνει από τον χρήστη τα ονομάζουμε <<**input.txt**>>.

Τα αρχεία με τις έτοιμες απαντήσεις για την σύγκριση τα ονομάζουμε <<**answers.txt**>>.

Άρα η ***σύγκριση*** θα γίνεται μεταξύ <<***input.txt***>> και <<***answers.txt***>>.

Κάθε ένα από τα έξι κεφάλαια έχει δύο σετ απαντήσεων, κάθε σετ έχει έξι ερωτήσεις. Αν αποτύχει να απαντήσει με επιτυχία στο πρώτο σετ ερωτήσεων θα δοθεί μια δεύτερη προσπάθεια στον χρήστη απαντώντας στο δεύτερο σετ ερωτήσεων. Αυτές οι ερωτήσεις είναι διαφορετικές από τις ερωτήσεις του πρώτου σετ. Άρα έχουμε για κάθε κεφάλαιο <<set_1>> και <<set_2>> ερωτήσεων. Συνολικά δώδεκα ερωτήσεις για κάθε κεφάλαιο. Επίσης υπάρχουν και δύο σετ με έτοιμες απαντήσεις για κάθε αντίστοιχο σετ ερωτήσεων, δηλαδή δώδεκα έτοιμες απαντήσεις για κάθε κεφάλαιο.

Τα **κεφάλαια** τα ονομάζουμε <<**chapter_1**>> έως <<**chapter_6**>>.

Οπότε έχουμε για κάθε <<chapter_i>>, όπου i = 1 – 6,
ένα <<output.txt>> με <<set_1>> και <<set_2>> και
ένα <<answers.txt>> με <<set_1>> και <<set_2>>.

Δηλαδή, το κεφάλαιο <<chapter_1>> θα έχει τα αρχεία
<<Ch_1_output_set_1>> και <<Ch_1_output_set_2>>, και αντίστοιχα τα αρχεία
<<Ch_1_answers_set_1>> και <<Ch_1_answers_set_2>>. κτλ....

Οι ερωτήσεις που θα περιέχει το κάθε σετ του κάθε κεφαλαίου θα είναι κατηγοριοποιημένες σε τρεις κατηγορίες. Αυτό γίνεται γιατί θέλουμε να αξιολογήσουμε πιο "καλά" το αποτέλεσμα. Κάθε κατηγορία έχει την δικιά του απόδοση σε βαθμούς. Οι κατηγορίες είναι λοιπόν A', B' και C', αντίστοιχα 10,15 και 25 πόντους για κάθε ερώτηση της κάθε κατηγορίας.

Συνοψίζοντας, κάθε κεφάλαιο θα έχει ένα φάκελο <<Chapters>>, σε αυτό τον φάκελο θα υπάρχουν έξι φάκελοι <<chapter_i>>, όπου i = 1 – 6, σε κάθε ένα από αυτούς τούς φακέλους θα υπάρχουν τρεις φάκελοι με <<chapter_i_questions>>, <<chapter_i_answers>> και <<chapter_i_input>>, όπου i = 1 – 6. Στον φάκελο <<chapter_i_questions>> θα υπάρχουν δυο φάκελοι <<chapter_i_questions_set_y>>,όπου y = 1 – 2, σε κάθε σετ θα υπάρχουν έξι αρχεία και κάθε αρχείο θα παρέχει μια ερώτηση και τρεις πιθανές απαντήσεις, είναι πολλαπλών απαντήσεων (multiple choice). Αντίστοιχα, κάθε φάκελο <<chapter_i_answers>> παρέχει τις αντίστοιχες έτοιμες απαντήσεις. Στο φάκελο

<<chapter_i_input>> θα αποθηκεύονται οι απαντήσεις του χρήστη. Το αρχείο του χρήστη θα αποθηκεύει όλα τα αποτελέσματα του κάθε σετ του κάθε κεφαλαίου. Θα είναι λοιπόν ένα αρχείο με όλες τις απαντήσεις του κάθε σετ του κάθε κεφαλαίου.

Παράδειγμα για το κεφάλαιο 1:

Μονοπάτι για την εκτύπωση στην οθόνη:
<<Chapters/chapter_1/chapter_1_questions/chapter_1_questions_set_1/Ch_1_output_set_1.txt>>

Μονοπάτι για την αποθήκευση των απαντήσεων:
<<Chapters/chapter_1/chapter_1_input/chapter_1_input_set_1/Ch_1_input_set_1.txt>>

Μονοπάτι των απαντήσεων:
<<Chapters/chapter_1/chapter_1_answers/chapter_1_answers_set_1/Ch_1_answers_set_1.txt>>

Στην οθόνη θα βλέπει ο χρήστης την ερώτηση, το κεφάλαιο και σετ στα οποία βρίσκεται, και τρεις πιθανές απαντήσεις από τις οποίες μπορεί να επιλέξει μια, πληκτρολογώντας τον αντίστοιχο αριθμό που είναι δίπλα από κάθε πιθανή απάντηση. Αν προσπαθήσει να εισάγει κάτι άλλο από τους αριθμούς (1 – 3) που αντιστοιχούν στις πιθανές απαντήσεις, θα εμφανίζεται ένα μήνυμα λάθους και θα του ξαναδοθεί η ευκαιρία να κάνει την επιλογή του. Αφού έχει επιλέξει μια από τις πιθανές απαντήσεις, επιβεβαιώνει την επιλογή του πατώντας <<enter>>. Έπειτα θα εκτυπωθεί στην οθόνη η επιλογή του και μια λίστα με επιλογές. Οι επιλογές θα είναι είτε να απαντήσει ξανά στην ερώτηση ή να προχωρήσει στην επόμενη ερώτηση. Αν είναι η τελευταία ερώτηση του εκάστοτε σετ και κεφαλαίου, θα περιμένει ο χρήστης την οθόνη με τα αποτελέσματα. Το πρόγραμμα θα πρέπει να καταλάβει πως ήταν η τελευταία ερώτηση και θα πρέπει να κάνει τώρα την σύγκριση των αρχείων << Ch_i_input_set_y.txt>> και <<Ch_i_answers_set_y.txt>> , όπου i = 1 – 6 και y = 1 – 2. Ουσιαστικά θα είναι ένα <<matching>> των αριθμών στα δυο αρχεία. Θα πρέπει να καταλάβει το πρόγραμμα πως οι πρώτοι δυο αριθμοί αντιστοιχούν στο βάρος της αντίστοιχης ερώτησης, δηλαδή οι ερωτήσεις που αφορούν την ομάδα Α' έχουν βάρος 10 πόντους, οι ερωτήσεις που αφορούν την ομάδα Β' έχουν βάρος ίσο με 15 πόντους, ενώ οι ερωτήσεις της C' ισούνται με 25 πόντους. Αν το <<matching>> για κάθε απάντηση βγάζει ως αποτέλεσμα μηδέν τότε επιστρέφει μηδέν πόντους, αν το <<matching>> επιστρέφει ως αποτέλεσμα ένα τότε επιστρέφει τους αντίστοιχους πόντους για κάθε κατηγορία. Κάθε φορά που θα παίρνει είτε μηδέν (0) είτε ένα (1) ως αποτέλεσμα του <<matching>> θα επιστρέφει σε μια μεταβλητή τα αποτελέσματα (μηδέν ή ένα) και θα γίνεται ένα άθροισμα, το αποτέλεσμα του οποίου θα αποθηκεύεται και θα είναι το <<factor>> με τον οποίο θα πολλαπλασιάζεται στην συνάρτηση μας το αποτέλεσμα του αθροίσματος των πόντων από κάθε κατηγορία.

Δηλαδή, και τους πόντους τους επιστρέφει σε μια μεταβλητή που υπολογίζει το συνολικό αποτέλεσμα της κάθε κατηγορίας.

Έτσι βγαίνει η εξής συνάρτηση:

Συνάρτηση υπολογισμού απόδοσής του χρήστη

<<factor>> f = 0/1 + 0/1 + 0/1 + 0/1 + 0/1 + 0/1, δηλαδή 0 ή 1 + 0 ή 1 +.....

A' = 10/0 + 10/0, δηλαδή 10 ή 0 + 10 ή 0,

B' = 15/0 + 15/0, δηλαδή 15 ή 0 + 15 ή 0,

C' = 25/0 + 25/0. δηλαδή 25 ή 0 + 25 ή 0,

Σ = (A' + B' + C') * f.

Το μέγιστο που μπορεί να φτάσει είναι 600 πόντοι. Άρα **max = 600.**

Το ελάχιστο που πρέπει να φτάσει είναι 105 πόντοι. Άρα **min = 105.**

Αν ξεπεράσει το min τότε πάει στην επόμενη φάση (επόμενο κεφάλαιο / τέλος (Στόχο)).

Αν δεν ξεπεράσει το min τότε πάει στο δεύτερο σετ / προηγούμενο σχετικό κεφάλαιο.

Στο προηγούμενο σχετικό κεφάλαιο επιστρέφει αν δεν καταφέρει να επιτύχει τουλάχιστον το min στο δεύτερο σετ ερωτήσεων.

Αυτό το αποτέλεσμα το εκτυπώνει στην οθόνη και αν έχει περάσει το κατώφλι περνάει στην επόμενη φάση (κεφάλαιο / τέλος προγράμματος).Αν όχι πάμε στο επόμενο σετ ή προηγούμενο κεφάλαιο που είναι προαπαιτούμενο για αυτό το κεφάλαιο που δουλεύει την τάδε χρονική στιγμή. Στην οθόνη θα εκτυπώνεται για κάθε-περίπτωσή το αντίστοιχο μήνυμα, το πρόγραμμα θα δίνει στον χρήστη την επιλογή είτε για να επιβεβαιώνει ότι έχει διαβάσει το μήνυμα πατώντας <<enter>> και να προχωράει στην επόμενη φάση είτε για να σταματήσει και να βγει από το πρόγραμμα. Οπότε θα υπάρχει είτε ένα αρχείο ακόμα με το σχετικό μήνυμα είτε απλά μια μέθοδος που θα εκτυπώνει το σχετικό μήνυμα μαζί με τις επιλογές στην οθόνη.

Λειτουργικότητα του προγράμματος:

- ⇨ Εκτυπώνει στην οθόνη ένα μενού όπου επιλέγει ο χρήστης τα κεφάλαια.
- ⇨ Εκτυπώνει στην οθόνη τις ερωτήσεις του κεφαλαίου.

- ⇨ Ο χρήστης απαντά στα ερωτήματα .
- ⇨ Το πρόγραμμα συγκρίνει τις απαντήσεις που έδωσε ο χρήστης με τις σωστές απαντήσεις.
- ⇨ Το πρόγραμμα υπολογίζει με την βοήθεια μιας συνάρτησης αν έχει επιτύχει ή όχι ο χρήστης.
- ⇨ Αν έχει πετύχει, πηγαίνει στην επόμενη φάση (κεφάλαιο / Τέλος).
- ⇨ Αν έχει αποτύχει, πηγαίνει ξανά στο ίδιο κεφάλαιο με άλλες ερωτήσεις.
- ⇨ Αν αποτύχει ξανά, πηγαίνει στο προηγούμενο κεφάλαιο.
- ⇨ Το προηγούμενο κεφάλαιο πρέπει να συσχετίζεται με το μη περασμένο κεφάλαιο, δηλαδή δεν θα πηγαίνει απλά ένα βήμα πίσω αλλά θα πρέπει να πηγαίνει στο κεφάλαιο που συσχετίζεται (που είναι προαπαιτούμενο) με αυτό.

Σειρά Εκτέλεση Προγράμματος

1. Πρέπει να διαβάσει το αρχείο (chapter_i)
2. πρέπει να εκτυπώσει στην οθόνη το αρχείο με τις ερωτήσεις
3. πρέπει να διαβάσει τις απαντήσεις και να τις αποθηκεύσει σε ένα αρχείο
4. πρέπει να διαβάσει το αρχείο με τις έτοιμες απαντήσεις και να τις αποθηκεύσει σε έναν πίνακα (answers_array) και πρέπει να διαβάσει το αρχείο με τις απαντήσεις του χρήστη και να αποθηκεύσει τις τιμές σε ένα πίνακα (input_array). Πρέπει να κάνει σύγκριση των δυο αυτών πινάκων και να επιστρέφει αν κάνει match ένα και αν δεν κάνει match μηδέν.
5. Πρέπει να αθροίσει αυτές τις τιμές και να αποθηκεύσει το αποτέλεσμα σε μια μεταβλητή. Όταν θα διαβάζει από αρχείο με τις απαντήσεις του χρήστη θα πρέπει να μετράει να διαβάζει τις τιμές και να τις αποθηκεύει σε τρεις μεταβλητές (A',B',C') που αντιστοιχούν στο βάρος της κάθε ερώτησης επί το αποτέλεσμα του matching και να αθροίζει τις δύο ερωτήσεις.
6. Θα πρέπει να εκτυπώνει το αποτέλεσμα στην οθόνη.
7. Θα πρέπει σύμφωνα με την συνάρτηση του αποτελέσματος να αποφασίσει και να πάει στο επόμενο βήμα.

Αλγόριθμος

Με πρώτη ματιά θα μπορούσε να πει κανείς πως ο αλγόριθμος είναι ένας συνδυασμός του DFS και του BFS. Όμως στην πραγματικότητα συμπεριφέρεται σαν τον DFS όταν σαρώνει το δένδρο από την ρίζα μέχρι το τελικό στοιχείο. Όταν όμως πρέπει να κάνει πίσω γυρίσματα και αρχίζει να σαρώνει το δένδρο ως προς το πλάτος, στην πραγματικότητα δεν

χρησιμοποίει τα στοιχεία του BFS άλλα μέσα στο πρόγραμμα πηδάει σε ένα άλλο σημείο εκτέλεσης και ακολουθεί στην ουσία μια λίστα μέχρι που να βρει το ζητούμενο στοιχείο.-

Στην συνέχεια περιγράφω ακριβώς τα παραπάνω και δείχνω πως συμπεριφέρονται οι αλγόριθμοι DFS και BFS σε αντίφαση με τον εκτελούμενο αλγόριθμο.

Ορισμοί αλγορίθμων

Ως πλαίσιο για την περιγραφή των αλγορίθμων αναζήτησης χρησιμοποιείται ένας γενικός αλγόριθμος, ο οποίος δε διευκρινίζει κρίσιμα χαρακτηριστικά, όπως για παράδειγμα ποια κατάσταση εξετάζεται πρώτη, πώς διευθετούνται οι επιλογές ανάμεσα σε υποψήφιες καταστάσεις, πώς γίνεται το κλάδεμα καταστάσεων, κτλ. Παρά τις διαφορές τους, όλοι οι αλγόριθμοι αναζήτησης χρησιμοποιούν ορισμένες κοινές δομές δεδομένων, που ορίζονται στη συνέχεια.

Μέτωπο της αναζήτησης (search frontier) ενός αλγορίθμου είναι το διατεταγμένο σύνολο (λίστα) των καταστάσεων που ο αλγόριθμος έχει ήδη επισκεφτεί, αλλά δεν έχουν ακόμη επεκταθεί. Σε κάθε επανάληψη (κύκλο λειτουργίας) ενός αλγορίθμου επεκτείνεται μια κατάσταση η οποία ανήκει στο μέτωπο της αναζήτησης. Στη συνέχεια, η κατάσταση αυτή παύει να ανήκει στο μέτωπο αναζήτησης και εισάγεται στο κλειστό σύνολο. Κλειστό σύνολο (closed set) ενός αλγορίθμου αναζήτησης είναι το σύνολο όλων των καταστάσεων που έχουν ήδη επεκταθεί από τον αλγόριθμο.

Κάθε επέκταση μιας κατάστασης συνοδεύεται αφενός με την εισαγωγή της κατάστασης – γονέα στο κλειστό σύνολο και αφετέρου με την εισαγωγή των καταστάσεων – παιδιών στο μέτωπο της αναζήτησης. Η κύρια χρησιμότητα του κλειστού συνόλου είναι να αποφύγει ο αλγόριθμος να επισκεφτεί καταστάσεις που έχει ήδη επισκεφτεί. Με έναν απλό έλεγχο, αν η κατάσταση προς επέκταση ανήκει ήδη στο κλειστό σύνολο, αποφεύγονται οι βρόχοι (loops).

Αναζήτηση πρώτα σε Βάθος (DFS)

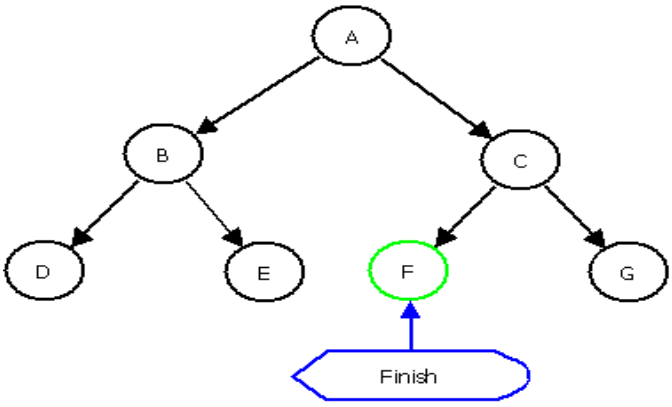
Αναζήτηση πρώτα σε βάθος(DFS) είναι ένας αλγόριθμος για την διάσχιση ή την αναζήτηση ενός δέντρου, τη δομή δέντρου, ή διάγραμμα. Ξεκινά από την ρίζα (επιλέγοντας κάποιον κόμβο, όπως η ρίζα, στην περίπτωση γράφημα) και τις διερευνά όσο το δυνατόν περισσότερο κατά μήκος κάθε κλαδιού πριν την υπαναχώρηση. Η αναζήτηση επιλέγει προς επέκταση την κατάσταση που βρίσκεται πιο βαθιά στο

δέντρο. Στην περίπτωση που υπάρχουν περισσότερες από μία καταστάσεις στο ίδιο βάθος, ο DFS επιλέγει τυχαία μία από αυτές ή, για ευκολία, επιλέγει την αυτήν που βρίσκεται στο πιο αριστερό μέρος. Η περιγραφή του αλγορίθμου είναι η ακόλουθη:

1. Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αν το μέτωπο της αναζήτησης είναι κενό τότε σταμάτησε.
3. Βγάλε την πρώτη κατάσταση από το μέτωπο της αναζήτησης.
4. Αν η κατάσταση είναι μέλος του κλειστού συνόλου τότε πήγαινε στο βήμα 2.
5. Αν η κατάσταση είναι μία από τις τελικές, τότε ανέφερε τη λύση, αλλιώς αν θέλεις και άλλες λύσεις πήγαινε στο βήμα 2.
6. Εφάρμοσε τους τελεστές μετάβασης για να βρεις τις καταστάσεις παιδιά.
7. Βάλε τις καταστάσεις – παιδιά **στην αρχή** του μετώπου της αναζήτησης.
8. Βάλε την κατάσταση – γονέα στο κλειστό σύνολο.
9. Πήγαινε στο βήμα 2.

Το μέτωπο της αναζήτησης είναι μια δομή στοίβας (Stack LIFO, Last in First Out), δηλαδή οι νέες καταστάσεις τοποθετούνται πάντα στην κορυφή της στοίβας και η αναζήτηση συνεχίζεται με μία από αυτές.

Παράδειγμα:



Μέτωπο Αναζήτησης	Κλειστό Σύνολο (Τσεκαρισμένη Κατάσταση)	Κατάσταση (Τρέχουσα κατάσταση)	Διάδοχες (κατάσταση – παιδιά)
<A>	{}	A	<B, C>
<B, C>	{A}	B	<D, E>
<D, E, C>	{A, B}	D	<δεν έχει παιδιά>
<E, C>	{A, B, D}	E	<δεν έχει παιδιά>
<C>	{A, B, D, E}	C	<F, G>
<F, G>	{A, B, D, E, C}	F	Τελική

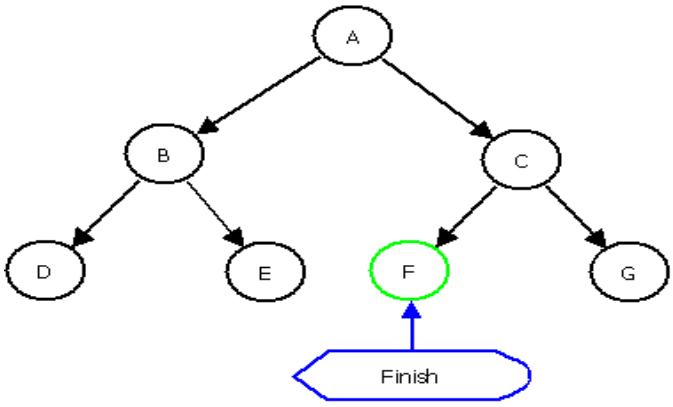
Αναζήτηση Πρώτα σε Πλάτος

Στη θεωρία γράφων, η αναζήτηση σε πλάτος (BFS - Breadth-first search) είναι ένας αλγόριθμος αναζήτησης που ξεκινάει από τον αρχικό κόμβο και διερευνά όλους τους γειτονικούς κόμβους. Στη συνέχεια, για κάθε ένα από τους πλησιέστερους κόμβους, διερευνά ανεξερεύνητους γειτονικούς κόμβους, και ούτω καθεξής, μέχρι να βρει το

στόχο. Δηλαδή εξετάζει πρώτα τις καταστάσεις που βρίσκονται στο ίδιο βάθος και μόνον όταν τις εξετάσει όλες, συνεχίζει στην επέκταση καταστάσεων στο αμέσως επόμενο επίπεδο. Η αναζήτηση μοιάζει να προχώρα κατά κύματα, όπου το πρώτο κύμα είναι οι καταστάσεις του πρώτου επιπέδου του δέντρου, το δεύτερο κύμα οι καταστάσεις του δεύτερο του επιπέδου, κτλ. Η περιγραφή του αλγορίθμου είναι:

1. Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αν το μέτωπο της αναζήτησης είναι κενό τότε σταμάτησε.
3. Βγάλε την πρώτη κατάσταση από το μέτωπο της αναζήτησης.
4. Αν η κατάσταση είναι μέλος του κλειστού συνόλου τότε πήγαινε στο βήμα 2.
5. Αν η κατάσταση είναι μια τελική τότε ανέφερε τη λύση, αλλιώς αν θέλεις και άλλες λύσεις πήγαινε στο βήμα 2.
6. Εφάρμοσε τους τελεστές μεταβάσεις για να βρεις τις καταστάσεις – παιδία.
7. Βάλε τις καταστάσεις – παιδία **στο τέλος** του μετώπου της αναζήτησης.
8. Βάλε την κατάσταση – γονέα στο κλειστό σύνολο.
9. Πήγαινε στο βήμα 2.

Παράδειγμα:



Μέτωπο Αναζήτησης	Κλειστό Σύνολο (Τσεκαρισμένη Κατάσταση)	Κατάσταση (Τρέχουσα κατάσταση)	Διάδοχες (κατάσταση – παιδιά)
<A>	{}	A	<B, C>
<B, C>	{A}	B	<D, E>
<C, D, E>	{A, B}	C	<F, G>
<D, E, F, G>	{A, B, C}	D	<δεν έχει παιδιά>
<E, F, G>	{A, B, C, D}	E	<δεν έχει παιδιά>
<F, G>	{A, B, C, D, E}	F	Τελική

Η διαφορά του BFS από τον DFS εντοπίζεται στην περιγραφή του αλγορίθμου, και συγκεκριμένα με μια μόνο λέξη, «τέλος» αντί για «αρχή».-

Ο αλγόριθμος που εφαρμόστηκε στα πλαίσια αυτής της διπλωματικής εργασίας είναι κάτι ενδιάμεσο του DFS και του BFS. χωρίς αυτό να σημαίνει όμως πώς έχει σχέση με τον αλγόριθμο επαναληπτικής εκβάθυνσης (Iterative Deepening – ID), που συνδυάζει με τον καλύτερο τρόπο τους DFS και BFS, γιατί σε αυτό τον αλγόριθμο γίνεται η αναζήτηση σε στάδια και κάθε στάδιο είναι η εφαρμογή του DFS σε κάποιο από αυτά τα στάδια, ανεξάρτητα από το συνολικό βάθος του δένδρου αναζήτησης και όταν η αναζήτηση στο προκαθορισμένο βάθος ολοκληρωθεί, ο DFS επαναλαμβάνεται με την ίδια αρχική κατάσταση, άλλα σε μεγαλύτερο βάθος, κάτι που δεν συμβαίνει σε τούτο τον αλγόριθμο. Εάν η αναζήτηση είναι η πρώτη αποτυχία σε κάποιο επίπεδο, δηλαδή σε κάποια ενότητα της ύλης, τότε η αναζήτηση επιλέγει εξορισμού προς επέκταση την κατάσταση που βρίσκεται πιο βαθιά στο δέντρο, όμως όταν βρει την πρώτη αποτυχία θα εξετάσει πρώτα τις καταστάσεις που βρίσκονται στο ίδιο βάθος, δηλαδή εκτελείται σχεδόν το BFS, και μόνον αν δεν καταλήξει σε άλλη αποτυχία θα επιστρέψει στο DFS, όπου θα συνεχίσει την εκτέλεση του χωρίς να επαναλαμβάνει την ίδια αρχική κατάσταση.

- ⇨ Στην καλύτερη περίπτωση, που όλα πάνε «καλά», ο αλγόριθμος συμπεριφέρεται σαν DFS.
- ⇨ Στην χειρότερη περίπτωση, που τίποτα δεν πάει καλά, ο αλγόριθμος συμπεριφέρεται σχεδόν σαν BFS.

Γιατί δεν είναι BFS (Breadth-first search)

Όμως επειδή τα παιδιά των κόμβων του BFS δεν μπορούν ποτέ να έχουν κάποια σχέση με τα κάτω επίπεδα, αφού κανένας κόμβος δεν έχει σχέση (ακμή) προς τους κάτω κόμβους. Άρα από την μία μεριά γίνεται μια αναζήτηση κατά πλάτος άλλα αυτή δεν γίνεται βάση τις διαφοροποίησης των δυο αυτών αλγορίθμων, δηλαδή δεν βάζει τις καινούριες καταστάσεις – παιδιά στο τέλος της αναζήτησης, κάτι που κάνει ο BFS για να βάλει την προτεραιότητα στους αδελφικούς κόμβους κατά πλάτος της αναζήτησης και όχι στους απογόνους αυτών των κόμβων. Σε αυτή την περίπτωση οι γειτονικοί κόμβοι δεν έχουν παιδιά στα κάτω επίπεδα, οπότε δεν βλέπουμε ποτέ μπροστά, αν δεν είμαστε σίγουροι πως έχουμε διαπεράσει το δέντρο στην περιοχή που βρισκόμαστε μέχρι που να έχουμε επιτυχίες οι οποίες οδηγούν εκεί που είχαμε την πρώτη αποτυχία.

Περιγραφή του αλγορίθμου:

1. Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.

2. Αν το μέτωπο της αναζήτησης είναι κενό τότε σταμάτησε.

3. Αν ο buffer δεν είναι άδειος, εκτέλεσε την πρώτη κατάσταση του buffer.

4. Βγάλε την πρώτη κατάσταση από το μέτωπο της αναζήτησης.

5. Αν η κατάσταση είναι μέλος του κλειστού συνόλου τότε πήγαινε στο βήμα 2. loop

6. Αν η κατάσταση είναι μια τελική (Exit / Finish) τότε ανέφερε τη λύση.

7. Εφάρμοσε τους τελεστές μετάβασης για να παράγεις τις καταστάσεις – παιδία.

8. Αν η κατάσταση είναι αληθής, κάνε αποκοπή του υπόδενδρου, αν η κατάσταση δεν είναι αληθής, βάλε την κατάσταση στο buffer και πήγαινε στο βήμα 2.

9. Αν ο buffer έχει καταστάσεις, βγάλε τις καταστάσεις από το buffer και βάλε τις στην αρχή του μετώπου αναζήτησης, πήγαινε στο βήμα 5.

10. Βάλε τις νέες καταστάσεις – παιδία στην αρχή του μετώπου της αναζήτησης.

11. Βάλε την κατάσταση - γονέα στο κλειστό σύνολο.

12. Πήγαινε στο βήμα 2.

Παράδειγμα εκτελέσεις του αλγορίθμου:

Μέτωπο	*Buffer Καταστάσε	Κλειστό	Κατάστα	Αποκο	Διάδοχες (κατάσταση

Αναζήτησης	ων	Σύνολο	ση	πή	– παιδιά)
<A>	<>	{}	A	<>	<B(C), C>
<B, C>	<>	{A}	B{F}=> BFS	<>	<D(E,F), E>
<C, D, E>	<>	{A, B}	C	<>	<B>
<B, D, E>	<>	{A, B, C}	B	<>	<βρόχος>
<D, E>	<>	{A, B, C, D}	D{T}=> DFS	<E,F>	<G(H,I,J,K,L),H>
< G, H>	<>	{A, B, C, D}	G{T}=> DFS	<H,I,J,K,L>	<M(N,O,P,Q,R), N>
<M(N,O,P,Q,R), N>	<>	{A, B, C, D, G}	M{F}=> BFS	<>	<S, T>
<N,O,P,Q,R, S, T>	<M>	{A,B,C, D, G}	N{F}	<>	<O, M>
<O,P,Q,R,S,T,O,M>	<N, M>	{A, B, C, D, G}	O{F}	<>	<P,G>
<P,Q,R,S,T,O, M,P,G>	<O, N, M>	{A,B,C,D,G}	P{T}	<Q,R>	<G,Q>
<O,N,M,S,T,O, M,P,G>	<>	{A,B,C,D,G,P}	O{T}	<>	<G,N>
<N,M,S,T,O,M, P,G,G,N>	<>	{A,B,C,D,G,P, O}	N{T}	<>	<M,O>
<M,S,T,O,M,P, G,G,N,M,O>	<>	{A,B,C,D,G,N, O,P}	M{T}=> DFS	<N,O, P,Q,R>	<S,T>
<S,T,S,T,M,G,G>	<>	{A,B,C,D,G,N, O,P,M}	S{T}	<T,U, V,W, X>	<Finish>

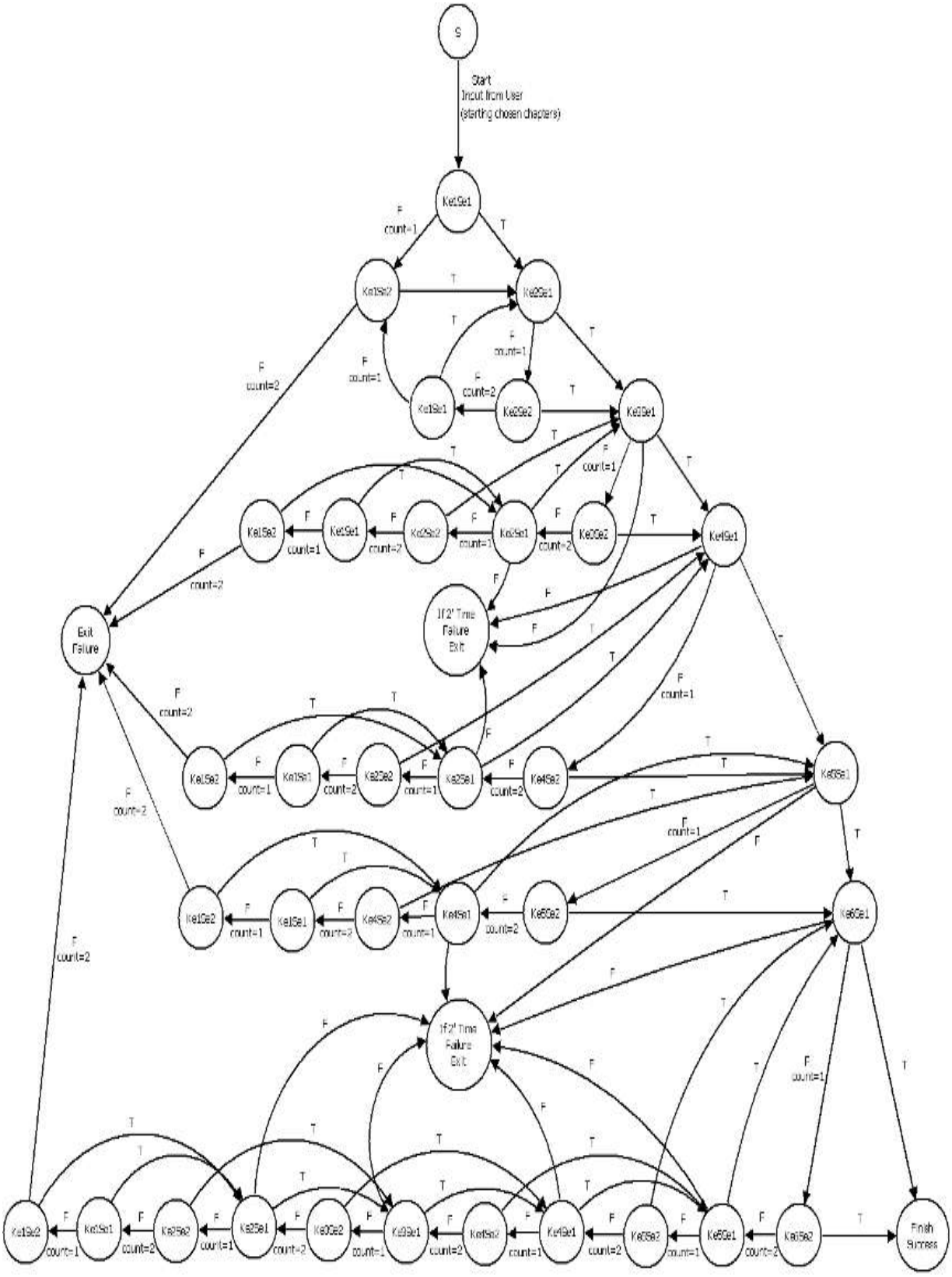
*Buffer = Δεν υπάρχει στην πραγματικότητα κάτι τέτοιο. Απλά η εκτέλεση του προγράμματος σε πηγαίνει σε ένα άλλο σημείο.-

Κλειστό Σύνολο = Είναι οι τσεκαρισμένες καταστάσεις.

Κατάσταση = είναι η τρέχουσα στην οποία βρισκόμαστε.

Αποκοπή = Γίνεται αποκοπή των καταστάσεων που δεν θα διαπεράσει ο αλγόριθμος. Μόλις έχει την πρώτη επιτυχία δεν υπάρχει περίπτωση να πάει πιο πίσω από αυτή την κατάσταση που πέτυχε.

Δένδρο του αλγορίθμου



Τρέξιμο του αλγορίθμου

Προϋποθέσεις:

Πρέπει να έχει καθοριστεί εκ των προτέρων το περιβάλλον. Το περιβάλλον αποτελείται από τις ενότητες και τους αντίστοιχους κόμβους. Δηλαδή κάθε ξεχωριστή ενότητα και υποενότητα αντιστοιχεί σε ένα κόμβο.

Πρέπει να έχει επιλέξει ο χρήστης από πόσους κόμβους θα αποτελείται το δικό τουπ περιβάλλον. Δηλαδή, πρέπει να επιλέξει αν θέλει όλες τις πιθανές προς επιλογή ενότητες ή μόνο ένα υποσύνολο από αυτό.

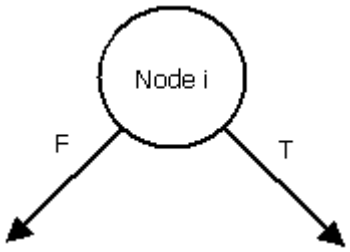
Εκτέλεση:

Όταν βρισκόμαστε σε έναν κόμβο, εξετάζουμε την είσοδο δεδομένων και παίρνουμε, με βάση την επεξεργασία και αξιολόγηση των δεδομένων, μια απόφαση. Επιτυχία (πέρασε την ενότητα) είναι αληθές (**T**, True) και αποτυχία (δεν κατάλαβε την ενότητα) είναι ψευδές (**F**, False). Αν είχαμε επιτυχία πάμε στην επόμενη ενότητα, αν είχαμε αποτυχία πάμε στην προηγούμενη ενότητα που συσχετίζεται με την τρέχουσα ενότητα.

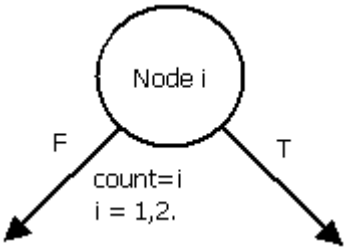
Η βέλτιστη περίπτωση είναι να επιτύχουμε σε όλες τις ενότητες και έτσι να περάσουμε το δέντρο όπως στην αναζήτηση σε βάθος.

Στην περίπτωση αποτυχίας, θυμάται ο αλγόριθμος που είχε σταματήσει την τελευταία φορά, δηλαδή θυμάται σε ποια ενότητα απέτυχε, και πάει στην προηγούμενη ενότητα που συσχετίζεται με την τρέχουσα.

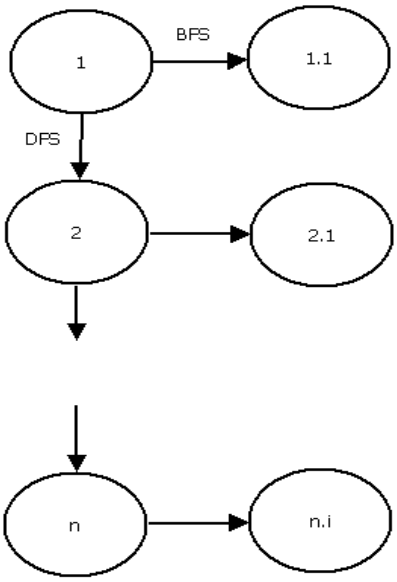
Η χείριστη περίπτωση είναι να κάνει τουλάχιστον μια αποτυχία σε κάθε ενότητα ή υποενότητα. Έτσι θα διατρέξει όλο το δέντρο μια φορά.



Κάθε ενότητα αποτελείται από δύο υποενότητες, αν αποτύχει στην πρώτη υποενότητα ο χρήστης έχει μια δεύτερη ευκαιρία να προσπαθήσει στην δεύτερη υποενότητα. Αν αποτύχει και εκεί δεν περνάει την ενότητα και πρέπει να πάει στην προηγούμενη σχετική ενότητα με αυτή την που απέτυχε. Για βοήθεια κατανόησης υπάρχει ένας μετρητής, **count**, που μετράει τον αριθμό των αποτυχιών του χρήστη σε κάθε ενότητα ξεχωριστά.



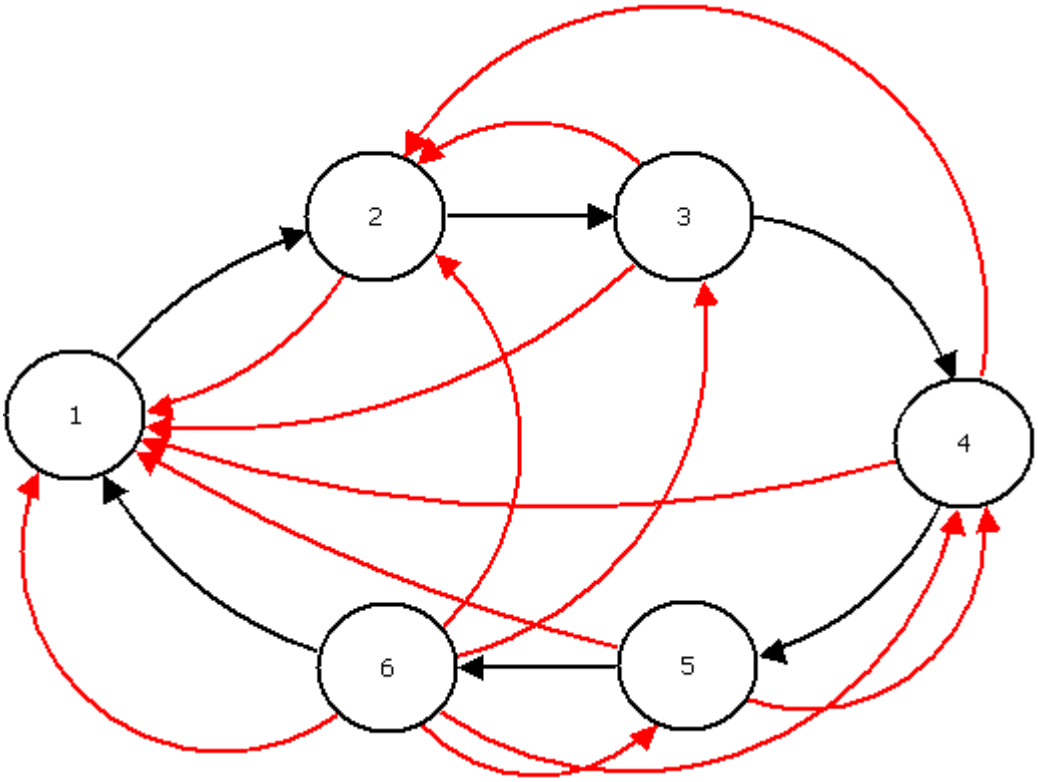
Υπάρχει επίσης ένας μετρητής, **count**, που μετράει τον αριθμό των αποτυχιών του χρήστη σε κάθε ενότητα ξεχωριστά. Αν αποτύχει την πρώτη φορά μετράει ο μετρητής την πρώτη αποτυχία και το count γίνεται **count=1**. Αν αποτύχει και στην δεύτερη υποενότητα ο χρήστης, τότε ο μετρητής καταγράφει και την δεύτερη αποτυχία, άρα **count=2**.



Κάθε ενότητα έχει και τα δικά της κεφάλαια με τα οποία συσχετίζεται. Αυτές οι ενότητες είναι οι προαπαιτούμενες γνώσεις για να μπορεί να επιτευχθεί η επόμενη ενότητα που είναι έμμεσα συνδεδεμένη. Ο χρήστης μπορεί να επιλέξει ποια είναι τα κεφάλαια με τα οποία θέλει να ασχοληθεί, και αν πετύχει τουλάχιστον μια από τις δυο υποενότητες την πρώτη φορά, θα διαπεράσει τις ενότητες έτσι όπως τις επέλεξε. Αν όμως έχει δύο αποτυχίες θα πρέπει να ασχοληθεί με την ενότητα που είναι προαπαιτούμενη σε αυτή που απέτυχε. Και αν αποτύχει και στην προαπαιτούμενη ενότητα, αλλά και στις δύο υποενότητες, θα τον πάει στην ενότητα που είναι προαπαιτούμενη στην τρέχουσα ενότητα. Όποτε τον πάει το πρόγραμμα βήμα-βήμα στο πραγματικό επίπεδο των γνώσεων του χρήστη. Άρα, κάθε ενότητα περιέχει μια λίστα με τις σχετιζόμενες ενότητες. Μπορούμε να πούμε πως πρόκειται για επίπεδα

στο δέντρο, όπου αν διατρέξουμε το δέντρο και έχουμε μια αποτυχία τότε έχουμε BFS (Breadth-first search), αναζήτηση σε πλάτος, δηλαδή διασχίζουμε οριζόντια το δέντρο μέχρι που να βρούμε την πρώτη επιτυχία ή να τελειώσουμε την αναζήτηση. Το ζητούμενο είναι να βρούμε την πρώτη αποτυχία στο DFS ακλουθώντας την πρώτη επιτυχία του BFS. Όποτε όσο ακολουθούμε με επιτυχία το DFS, σίγουρα θα μας οδηγήσει στο τέλος του δέντρου, αν όμως μπούμε σε μια αποτυχία, το BFS μπορεί να μα οδηγήσει στην έξοδο του δέντρου.

Ο γράφος τον συσχετιζόμενων ενοτήτων είναι ως εξής:

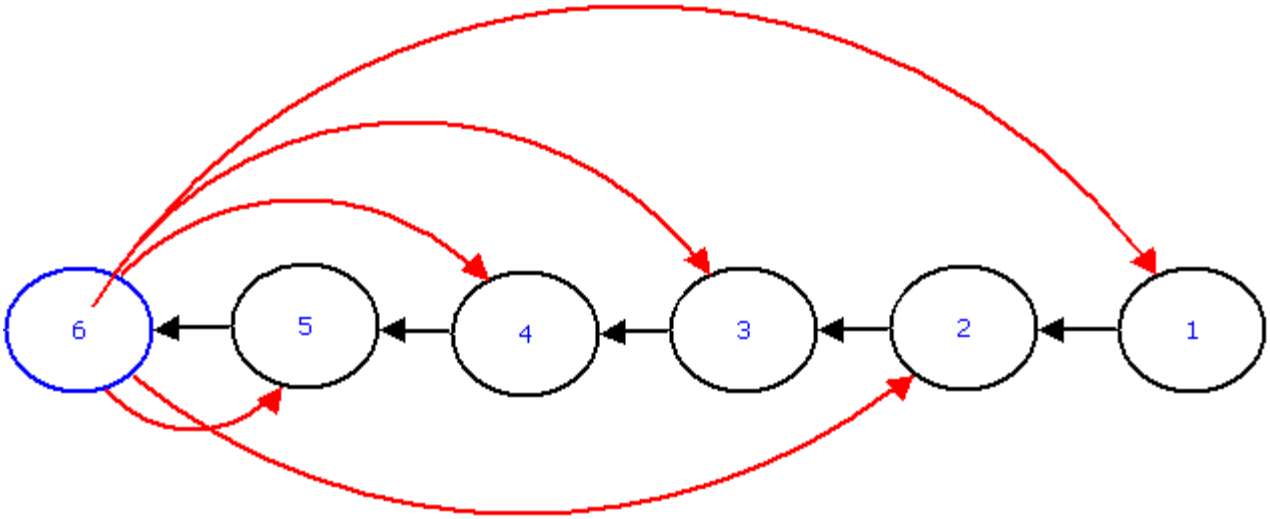


Οι κόμβοι αντιπροσωπεύουν τις ενότητες, οι μαύρες ακμές ακολουθούν την κανονική πορεία τις εκμάθησης τις μαθηματικής ύλης μιας σχολικής χρονιάς και οι κόκκινες ακμές είναι οι συσχετιζόμενες ακμές. Για παράδειγμα το κεφάλαιο 4 συσχετίζεται με τις ενότητες 2 και 1, οπότε οι ενότητες 1 και 2 είναι προαπαιτούμενες για το κεφάλαιο 4. Έτσι σχετίζεται και η ενότητα 2 με την ενότητα 1, άρα, η ενότητα 1 είναι προαπαιτούμενη για την ενότητα 2. Έτσι παράγουμε τον γράφο και τις συνδέσεις του.

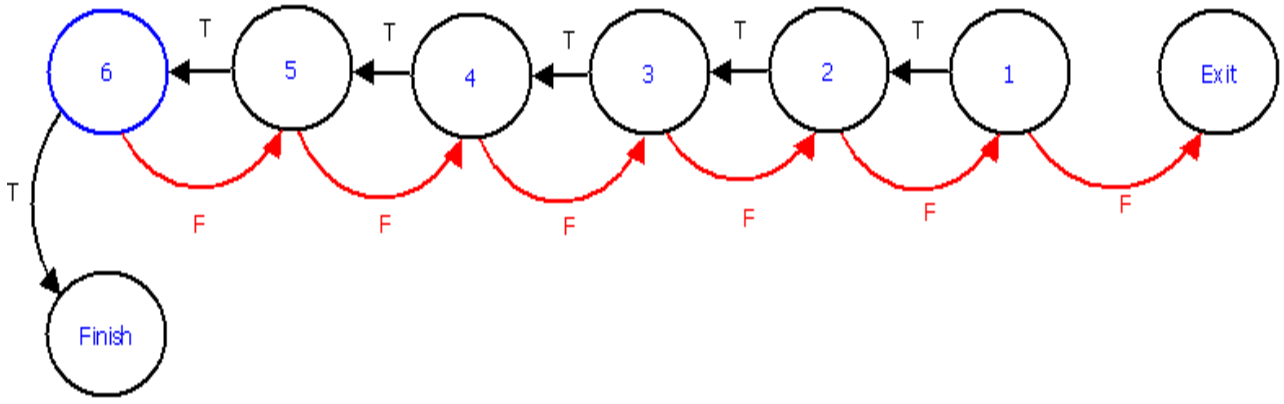
Ενότητες	Συσχετιζόμενες Ενότητες				
1					
2	1				
3	2	1			
4	2	1			
5	4	1			
6	5	4	3	2	1

Σε αυτό τον πίνακα φαίνονται πολύ καλά οι ενότητες και οι συσχετιζόμενες ενότητες. Στην πρώτη στήλη έχουμε όλες τις ενότητες και σε κάθε γραμμή τις συσχετιζόμενες ενότητες. Όπως φαίνεται και στον πίνακα, μόνο η ενότητα 6 συσχετίζεται με όλες τις υπόλοιπες ενότητες, το οποίο σημαίνει πώς όλες οι προηγούμενες ενότητες είναι προαπαιτούμενες για την ενότητα 6.

Παράδειγμα:



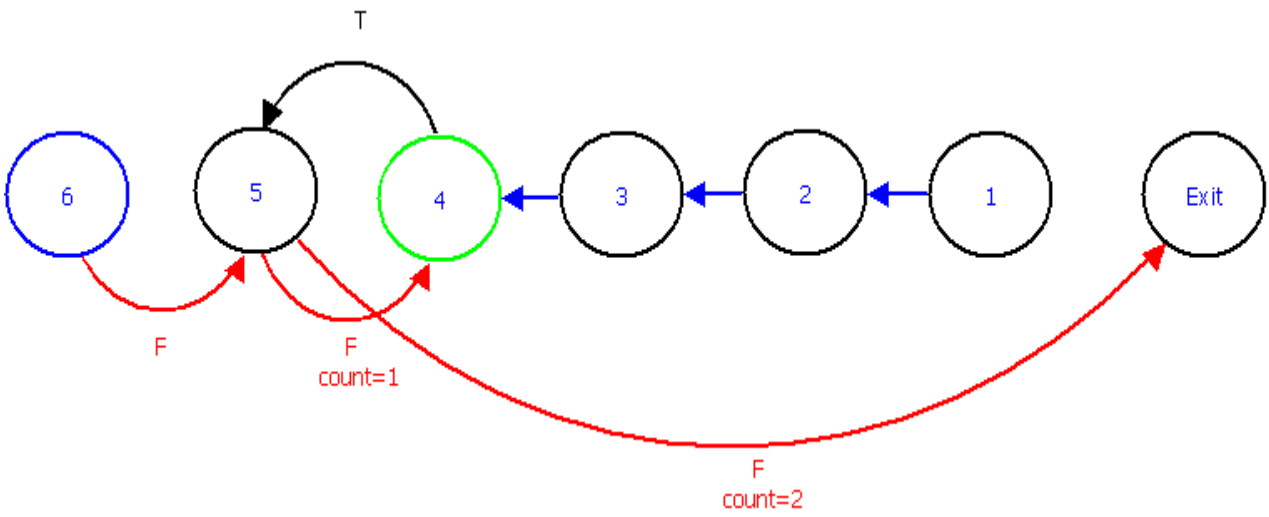
Η ενότητα 6 έχει ως προαπαιτούμενα τις ενότητες 5,4,3,2 και 1. Από την εικόνα φαίνεται, σύμφωνα με τις μαύρες ακμές, πως η ενότητα 1 είναι προαπαιτούμενη για την ενότητα 2, η ενότητα 2 είναι προαπαιτούμενη για την ενότητα 3, η ενότητα 3 είναι προαπαιτούμενη για την ενότητα 4, η ενότητα 4 είναι προαπαιτούμενη για την ενότητα 5, η ενότητα 5 είναι προαπαιτούμενη για την ενότητα 6. Άρα η ενότητα 6 συσχετίζεται με όλες τις ενότητες, που συμβολίζουν οι κόκκινες ακμές.



Αν αποτύχουμε στην ενότητα 6, πρέπει να προχωρήσουμε με την ενότητα 5, αφού είναι προαπαιτούμενη για την ενότητα 6. Αν αποτύχουμε πάλι πρέπει να

προσπαθήσουμε την ενότητα 4 αφού είναι προαπαιτούμενη για την ενότητα 5. Αν αποτύχουμε πάλι πρέπει να προσπαθήσουμε την ενότητα 3 αφού είναι προαπαιτούμενη για την ενότητα 4. Αν αποτύχουμε στην ενότητα 3 πρέπει να προσπαθήσουμε την ενότητα 2, αφού είναι προαπαιτούμενη για την ενότητα 3. Αν αποτύχουμε στην ενότητα 2 πρέπει να προσπαθήσουμε την ενότητα 1, αφού είναι προαπαιτούμενη για την ενότητα 2. Αν αποτύχουμε και στην ενότητα 1 μας πετάει έξω από το δέντρο και πρέπει να ξανά-διαβάσουμε το βιβλίο. Αυτές οι πράξεις είναι συμβολισμένες με τις κόκκινες ακμές.

Στην περίπτωση που είμαστε στην ενότητα 1, αφού έχουμε αποτύχει σε όλες τις προηγούμενες ενότητες, και έχουμε επιτυχία, πρέπει πρώτα να πετύχουμε την ενότητα από την οποία οδηγηθήκαμε στην ενότητα 1, όπου στην περίπτωση μας είναι η ενότητα 2. Μόνο αν πετύχουμε όλες τις ενότητες στις οποίες έχουμε αποτύχει μια φόρα, μπορούμε να φτάσουμε στην ενότητα από την οποία ξεκινήσαμε, δηλαδή την ενότητα 6,και να συνεχίσουμε το πρόγραμμα μας. Αν δεν βρούμε κάποια επιτυχία, σημαίνει πως δεν υπάρχει και έτσι μας πετάει από το “BFS” από το δέντρο.



Αν έχουμε αποτυχία στην ενότητα 6 και πάμε στην ενότητα 5 γιατί είναι προαπαιτούμενη για την ενότητα 6, πρέπει να προσπαθήσουμε την ενότητα 5. Αν αποτύχουμε στην ενότητα 5 πρέπει να προσπαθήσουμε την ενότητα 4, αφού η ενότητα 4 είναι προαπαιτούμενη για την ενότητα 5. Αν προσπαθήσουμε την ενότητα 4 και πετύχουμε, θα μας πάει στην τελευταία αποτυχημένη ενότητα, στην περίπτωση μας είναι η ενότητα 5. Αν όμως αποτύχουμε στην 5η για δεύτερη φορά, όπως δείχνει και ο μετρητής, count=2, τότε μας πετάει έξω από το δέντρο. Θεωρούμε πως η ενότητα 4 είναι η βάση, αφού πετύχαμε τουλάχιστον μια φορά και κάνουμε αποκοπή του υπόλοιπου δέντρου, γιατί ξέρουμε πώς σίγουρα δεν είναι εκεί αυτό που ψάχνουμε.

Ο κώδικας για την εφαρμογή του αλγορίθμου

```
/// ξεκίνα τις ενότητες που είναι επιλεγμένες

if(!exit) {

chID = lessIDs[0];

for(; c<lessLen;)  {  success = false; next = true;

        for(int s =0; s<SEC_COUNT; s++) {  secID = s + 1;

                answer = PrintQuestion(chID, secID);  PrintScore(chID, secID,
answer);

        /// αν απάντησες είναι σωστές τότε πήγαινε στην επόμενη ενότητα

                if(answer >= 100){      success = true; Lessons[chID-1].pass = true;
break;  }  }

        if(!success)     {

                if(Lessons[chID-1].trys == 0) {     cout<<Lessons[chID-1].trys;

                        Lessons[chID-1].trys = 0;

                                next = getNextChapter(Lessons, chID);        }

                else{ Lessons[chID-1].trys += 1; cout<<"\ngo to next try";     }

}

        /// πήγαινε στην επόμενη ενότητα.

else{ if(Lessons[chID-1].hasNext){ Lessons[chID-1].hasNext = false; chID =
Lessons[chID-1].next; }

else{ c += 1; if(c<lessLen) {  chID = lessIDs[c]; }}}

if(!next){break;} } }
```

ΠΑΡΑΡΤΗΜΑ – Εργαλεία που χρησιμοποιήθηκαν

A: Code::Blocks is a free and open source, cross-platform IDE which supports multiple compilers including GCC and MSVC. It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins. Currently, Code::Blocks is oriented towards C/C++.

Code::Blocks is being developed for Windows, Linux, and Mac OS X. Users have successfully built Code::Blocks under FreeBSD.

(http://www.codeblocks.org)

B: Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

Βιβλιογραφία

[1] Δασκαλοπούλου Α. (2010), Τεχνητή Νοημοσύνη Ι, Συστήματα Πρακτόρων.

[2] Μποζάνης Π.Δ. (2003), Δομές Δεδομένων: Ταξινόμηση και Αναζήτηση με Java, Τζιόλα.

[3] Βλαχάβας Ι., Κεφαλάς Π., Βασιλειάδης Ν., Κόκκορας Φ. και Σακελλαρίου Η. (2006). Τεχνητή Νοημοσύνη (Β' έκδοση), Εκδόσεις Γκιούρδα.

[4] Jesse Liberty (2004) Teach Yourself C++ In 21 Days 5th Edition.

[5] Herbert Schildt (2004), C++: The Complete Reference.

[6] Walter Savitch (2002), Complete C++.