Τεχνητή Νοημοσύνη
Εαρινό Εξάμηνο 2024

Τεχνητή Νοημοσύνη - Σημειώσεις

Τεχνητή Νοημοσύνη - Σημειώσεις

6 Ιανουαρίου 2026

Περιεχόμενα

1 Λογική

Στο πρώτο μέρος των σημειώσεων για την Τεχνητή Νοημοσύνη, θα εξετάσουμε τον ρόλο της Λογικής, η οποία αποτελεί θεμέλιο λίθο για την κατανόηση και εφαρμογή των μεθόδων συλλογιστικής και απόφασης. Η λογική είναι το εργαλείο μέσω του οποίου οι μηχανές μπορούν να επεξεργαστούν γνώσεις και να εξάγουν συμπεράσματα. Θα αναλύσουμε τα βασικά συστήματα λογικής που χρησιμοποιούνται στην ΤΝ, καθώς και τους μηχανισμούς που επιτρέπουν τον συλλογισμό πάνω σε προτάσεις και δεδομένα.

Η ανάλυση ξεκινά με την προτασιακή λογική, ένα σύστημα που μας επιτρέπει να εκφράζουμε απλές προτάσεις και τις σχέσεις τους μέσω λογικών τελεστών. Έπειτα, παρουσιάζεται η κανονική συζευκτική μορφή (ΚΣΜ), η οποία χρησιμοποιείται για την τυποποίηση σύνθετων λογικών προτάσεων σε μια μορφή που διευκολύνει την επεξεργασία τους από αλγορίθμους.

Συνεχίζοντας, θα αναφερθούμε στη μορφή Kowalski, μια πιο τυπική έκφραση της λογικής που διευκολύνει την αναπαράσταση προτάσεων και κανόνων συμπερασμού. Οι κανόνες συμπερασμού είναι οι θεμελιώδεις μέθοδοι μέσω των οποίων εξάγουμε νέα συμπεράσματα από ήδη γνωστές προτάσεις. Ειδική αναφορά γίνεται στην απόδειξη με ανασκευή (απαγωγή σε άτοπο), η οποία είναι μια μέθοδος απόδειξης που βασίζεται στην υπόθεση του αντιθέτου για να καταλήξουμε σε αντίφαση.

Τέλος, εισάγεται η κατηγορηματική λογική, που αποτελεί επέκταση της προτασιακής λογικής και μας επιτρέπει να εκφράζουμε πιο σύνθετες σχέσεις μεταξύ αντικειμένων και ιδιοτήτων τους, προσθέτοντας ποσοδείκτες όπως ο καθολικός και ο υπαρξιακός ποσοδείκτης. Με αυτό τον τρόπο, η λογική αποκτά μεγαλύτερη εκφραστικότητα, επιτρέποντας την επεξεργασία πιο περίπλοκων δεδομένων και την κατασκευή πιο εξελιγμένων συστημάτων ΤΝ.

1.1 Προτασιακή Λογική

Η **προτασιακή λογική (propositional logic)** αποτελεί μία γλώσσα που χρησιμοποιείται για αναπαράσταση γνώσης. Όπως κάθε άλλη γλώσσα αποτελείται από το λεξιλόγιο, το συντακτικό και τη σημασιολογία της. Το λεξιλόγιό της, αφορά τις μονάδες γνώσεις, τα στοιχεία γνώσης, ενώ το συντακτικό της αφορά την σωστή διάταξη όρων κάθε πρότασης. Τέλος, η σημασιολογία ασχολείται με το νόημα της πρότασης, δηλαδή αν ισχύει όντως ή όχι, δίνοντας της τιμή αλήθειας (truth value) **Αλήθης (True)** ή **Ψευδής (False)**.

Οι προτάσεις πρέπει να συνδέονται μεταξύ τους με λογικούς τελεστές. Οποιαδήποτε αλληλουχία προτάσεων που δεν χρησιμοποιεί λογικούς τελεστές δεν είναι έγκυρη. Έτσι, έχουμε τις απλές και τις σύνθετες προτάσεις. Μία **απλή πρόταση** (simple proposition) είναι μια πρόταση που δεν μπορεί να αναλυθεί περαιτέρω σε μικρότερες προτάσεις. Δηλαδή είναι μία πρόταση μόνη της, χωρίς λογικούς τελεστές. Μία **σύνθετη πρόταση** (compound proposition) είναι μια πρόταση που σχηματίζεται από απλές προτάσεις, συνδυασμένες με λογικούς τελεστές. Στον πίνακα 1 φαίνονται όλοι οι λογικοί τελεστές της προτασιακής λογικής.

Πίνακας 1: Παραδείγματα προτάσεων

Απλή πρόταση	Π1: βρέχει
Σύνθετη πρόταση	Π2: έχω ομπρέλα

Οι λογικοί τελεστές που είναι διαθέσιμοι στην προτασιακή λογική φαίνονται στον πίνακα 2. Όσο αναφορά αυτούς, μόνο ο λογικός τελεστής της άρνησης (NOT) μπορεί να εφαρμοστεί σε μία μεμονωμένη απλή πρόταση. Από την άλλη, οι λογικοί τελεστές της σύζευξης, διάζευξης, συνεπαγωγής και ισοδυναμίας εφαρμόζονται σε δύο προτάσεις.

Πίνακας 2: Λογικοί τελεστές προτασιακής λογικής.

Συμβολοσμός	**Ορολογία**	**Ονομασία**
$\neg$	NOT	Τελεστής Άρνησης
$\wedge$	AND	Τελεστής Σύζευξης
$\vee$	OR	Τελεστής Διάζευξης
$\Rightarrow$	IF THEN	Τελεστής Συνεπαγωγής
$\Leftrightarrow$	EQUIVALENCE	Τελεστής Ισοδυναμίας

Πίνακας 3: Πίνακας αλήθειας.

x	y	$\neg x$	$\neg y$	$x \wedge y$	$x \vee y$	$x \Rightarrow y$	$x \Leftrightarrow y$
Α	Α	Ψ	Ψ	Α	Α	Α	Α
Α	Ψ	Ψ	Α	Ψ	Α	Ψ	Ψ
Ψ	Α	Α	Ψ	Ψ	Α	Α	Ψ
Ψ	Ψ	Α	Α	Ψ	Ψ	Α	Α

Το αποτέλεσμα της χρήσης λογικών τελεστών ανάμεσα προτάσεων για την σύνδεσή τους ονομάζεται λογική έκφραση (ή μπορεί κάποιος να την σκεφτεί ως μία πιο μεγάλη λογική πρόταση). Η λογική έκφραση λοιπόν, για να αποτελείται από προτάσεις οι οποίες αντιστοιχούν σε κάποια τιμή αλήθειας, έχει και αυτή κάποια τελική τιμή αλήθειας. Αυτή η τιμή προφανώς, εξαρτάται από τις προτάσεις και τους λογικούς τελεστές που απαρτίζουν την λογική πρόταση αυτή. Στον πίνακα 3 φαίνονται οι τιμές μίας λογικής έκφρασης (λογικής πρότασης) για όλες τις δυνατές προτάσεις (μεταβλητές) και όλους τους δυνατούς τελεστές της. Ο πίνακας αυτός ονομάζεται **πίνακας αλήθειας**.

Η προτασιακή λογική αποτελεί κλάδος των μαθηματικών και όπως κάθε τι στα μαθηματικά που αφορά τελεστές υπάρχουν και οι ιδιότητες των τελεστών αυτών. Έτσι, έχουμε και τις ιδότητες των λογικών τελεστών. Στον πίνακα 4 φαίνονται οι ιδιότητες αυτές. Αξίζει να σημειωθεί ότι η ιδιότητα που ονομάζεται στον πίνακα 4 ως 'Κανόνας De Morgan' χρησιμοποιείται για μείωση πεδίου δράσεως τελεστή άρνησης και ότι η μετάφραση του Contrapositive είναι Αντίθεση, αλλά δεν βρέθηκε επίσημη βιβλιογραφία για την μετάφραση της ορολογίας αυτής και για αυτό χρησιμοποιήθηκε η αγγλική ορολογία.

Επίσης να σημειωθεί ότι δεν υπάρχει προτεραιότητα πράξεων στη λογική και γι'αυτό συνίσταται η χρήση παρενθέσεων σε λογικές προτάσεις, ώστε να υπάρξει μία σειρά, μία προτεραιότητα εκτέλεσης αυτών των πράξεων.

Παράδειγμα 1.1

Έστω οι προτάσεις Π1: βρέχει, Π2: έχω ομπρέλα και Π3: βρέχομαι και ότι θέλουμε να γράψουμε στην προτασιακή λογική το ακόλουθο:

$$(\Pi 1 \Rightarrow \Pi 2) \Leftrightarrow \neg \Pi 3 \tag{1}$$

Η έφραση 1 μεταφράζεται σε φυσική γλώσσα ως

> ΑΝ Π1 ΣΥΝΕΠΑΓΕΤΑΙ Π2 ΤΟΤΕ ΟΧΙ Π3
> (ΑΝ Π2 ΤΟΤΕ Π2)

Αν αντικαταστήσουμε τις προτάσεις έχουμε

> ΑΝ βρέχει ΣΥΝΕΠΑΓΕΤΑΙ έχω ομπρέλα ΤΟΤΕ ΔΕΝ βρέχομαι

Παρατηρούμε ότι η έκραφη 1 με τις δοσμένες προτάσεις είναι σωστή συντακτικά, αλλά όχι σημασιολογικά.

Πίνακας 4: Ιδιότητες λογικών τελεστών.

Τελεστής	**Ιδιότητα**	**Έκφραση**
$\neg$ (NOT)	Διπλή άρνηση	$\neg(\neg x) \Leftrightarrow x$
$\wedge$ (AND)	Αντιμεταθετική	$x \wedge y \Leftrightarrow y \wedge x$
	Προσεταιριστική	$(x \wedge y) \wedge z \Leftrightarrow x \wedge (y \wedge z)$
	Επιμεριστική	$x \wedge (y \vee z) \Leftrightarrow (x \wedge y) \vee (x \wedge z)$
$\vee$ (OR)	Αντιμεταθετική	$x \vee y \Leftrightarrow y \vee x$
	Προσεταιριστική	$(x \vee y) \vee z \Leftrightarrow x \vee (y \vee z)$
	Επιμεριστική	$x \vee (y \wedge z) \Leftrightarrow (x \vee y) \wedge (x \vee z)$
$\Rightarrow$ (IF THEN)	Χρυσός κανόνας	$x \Rightarrow y \Leftrightarrow \neg x \vee y$
	Contrapositive	$x \Rightarrow y \Leftrightarrow \neg x \Rightarrow \neg y$
$\Leftrightarrow$ (EQUIVALENCE)	Αντιμεταθετική	$x \Leftrightarrow y \Leftrightarrow y \Leftrightarrow x$
	Ορισμός ισοδυναμίας	$x \Leftrightarrow y \Leftrightarrow (x \Rightarrow y) \wedge (y \Rightarrow x)$
	Κανόνας De Morgan	$\neg(x \wedge y) \Leftrightarrow \neg x \vee \neg y$
	Κανόνας De Morgan	$\neg(x \vee y) \Leftrightarrow \neg x \wedge \neg y$

Τώρα έστω ότι έχουμε την ακόλουθη έκφραση:

$$(\Pi 1 \wedge \Pi 2) \Leftrightarrow \neg \Pi 3 \tag{2}$$

Με τις ίδιες προτάσεις θα έχουμε ως αποτέλεσμα:

ΑΝ Π1 ΚΑΙ Π2 ΤΟΤΕ ΔΕΝ Π3
ΑΝ βρέχει ΚΑΙ έχω ομπρέλα ΤΟΤΕ ΔΕΝ βρέχομαι

Εδώ με την έκφραση 2 έχουμε μία λογική πρόταση η οποία είναι συντακτικά και σημασιολογικά σωστή.

Μπορεί να υπάρξουν περιπτώσεις που απλές προτάσεις μίας λογικής έκφρασης θα έχουν πάντα τιμή αλήθειας Αλήθεια ή τιμή αλήθειας Ψευδής. Όταν συναντάμε προτάσεις που είναι πάντα Αλήθεια τότε έχουμε **ταυτολογία**, ενώ όταν οι προτάσεις είναι πάντα Ψευδής τότε έχουμε **αντίφαση**.

1.2 Κανονική Συζευκτική Μορφή

Μία πρόταση εκφρασμένη ως σύζευξη διαζευκτικών προτάσεων λέμε ότι είναι σε **κανονική συζευκτική μορφή**, ή ΚΣΜ, (Conjunctive Normal Form, CNF). Θα περιγράψουμε τώρα μία διαδικασία μετατροπής σε ΚΣΜ. Θα δείξουμε την διαδικασία αυτή μετατρέποντας την πρόταση $A \Leftrightarrow (B \vee \Gamma)$ σε ΚΣΜ. Τα βήματα είναι τα εξής:

1. Απαλοιφή του $\Leftrightarrow$, χρησιμοποιώντας τον Χρυσό Κανόνα (δηλαδή αντικαθιστώντας το $\alpha \Leftrightarrow \beta$ με $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$):

$$(A \Rightarrow B \vee \Gamma) \wedge ((B \vee \Gamma) \Rightarrow A)$$

2. Απαλοιφή του $\Rightarrow$, αντικαθιστώντας $\alpha \Rightarrow \beta$ με $\neg\alpha \vee \beta$:

$$(\neg A \vee (B \vee \Gamma)) \wedge (\neg(B \vee \Gamma) \vee A)$$

3. Η ΚΣΜ απαιτεί τα ¬ να εμφανίζονται μόνο σε literals, so we "move ¬ inwards" by repeated application of the following equivalences:

$$\neg(\neg\alpha) \equiv \alpha$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{(De Morgan)}$$

Οπότε έχουμε:

$$(\neg A \vee B \vee \Gamma) \wedge ((\neg B \wedge \neg\Gamma) \wedge A)$$

4. Η λογική μας πρόταση είναι σε αρκετά καλό στάδιο. Δεν υπάρχει κάποια προκαθορισμένη μεθοδολογία απο εδώ και πέρα. Έτσι, εφαρμόζουμε την επιμεριστική ιδιότητα και έχουμε:

$$(\neg A \vee B \vee \Gamma) \wedge ((\neg B \vee A) \wedge (\neg\Gamma \vee A))$$

Απαλοίφουμε τις περιτέρες παρενθέσεις:

$$(\neg A \vee B \vee \Gamma) \wedge (\neg B \vee A) \wedge (\neg\Gamma \vee A)$$

Η οποία μπορεί να σπάσει σε:

$$\neg A \vee B \vee \Gamma$$
$$\neg B \vee A$$
$$\neg\Gamma \vee A$$

Τώρα η αρχική μας πρόταση είναι σε ΚΣΜ, as a conjunction of three clauses***. It is much harder to read, but it can be used as input to a resolution procedure.

1.3 Μορφή Kowalski

Η μορφή Kowalski (ή κανονική μορφή Kowalski) είναι μια λογική αναπαράσταση που πήρε το όνομά της από τον Robert Kowalski, η οποία παρουσιάζει τη λογική σε υπολογιστική μορφή. Συνδέεται κυρίως με τον τομέα του λογικού προγραμματισμού και την ανάπτυξη της Prolog. Στο πλαίσιο μας, η μορφή Kowalski περιγράφεται ως ένας τρόπος έκφρασης της λογικής συλλογιστικής ως συνδυασμός βάσης γνώσης και μηχανισμού εξαγωγής συμπερασμάτων. Η μορφή Kowalski μπορεί να εφαρμοστεί μόνο σε προτάσεις που βρίσκονται σε ΚΣΜ.

Η μετατροπή μίας πρότασης σε ΚΣΜ σε μορφή Kowalski είναι τόσο απλή, που μπορεί να γίνει κατανοητή με ένα παράδειγμα.

Παράδειγμα 1.2

Έστω ότι έχουμε την παρακάτω πρόταση:

$$\neg A \vee B \vee \Gamma$$

Η μετατροπή της σε μορφη Kowalski θα είναι:

$$B, \Gamma \leftarrow A$$

Παρατηρούμε ότι κάθε τι που έχει τον λογική τελεστή της άρνησης ($\neg$), πηγαίνει στο δεξί μέρος με το σύμβολο του τελεστή να υπονοείται. Και άλλα παραδείγματα:

$$\text{ΚΣΜ: } \neg B \vee A \text{ σε Kowalski: } A \leftarrow B$$

$$\text{ΚΣΜ: } \neg \Gamma \vee A \text{ σε Kowalski: } A \leftarrow \Gamma$$

1.4 Κανόνες Συμπερασμού

1. Τρόπος του Θέτειν

   Αν:

   - Αριστερά έχω 2 αληθής προτάσεις (το ξέρουμε αξιωματικά)
   - Το σύνολο των προτάσεων/γνώσης είναι πάντα αλήθεια και ανήκουν στη βάση γνώσης

   Τότε μπορώ να παράξω/συμπεράνω μία νέα πρόταση που είναι και αυτή αλήθεια. Μπορεί η προκίπτουσα να συμπεριληφθεί στη βάση γνώσης. Όποια πρόταση προκύπτει από διαδικασία συμπερασμού αληθών προτάσεων, είναι αληθής.

   Παράδειγμα 1.3

   Έστω οι προτάσεις Π1: x και Π2: $x \Rightarrow y$, για τις οποίες ισχύει ότι το x είναι αληθής. Η ΚΣΜ των προτάσεων αυτών είναι:

   $$\text{Π1:}x \quad \text{Π2: } \neg x \vee y$$

   Σε μορφή Kowalski έχουμε:

   $$\text{Π1:}x \leftarrow \quad \text{Π2:}y \leftarrow x$$

   Βλέπουμε ότι τα x απαλοίφονται. Τελικά έχουμε:

   $$y \leftarrow$$

   Τελικά, όπως μας λέει ο τρόπος του Θέτειν, το y είναι αληθής.

2. Απαλοιφή σύζευξης

   Έστω ότι έχουμε μία πρόταση της μορφής: $P_1 \wedge P_2 \wedge P_3 \wedge ... \wedge P_N$, με όλες τις απλές προτάσεις να είναι αληθής. Τότε μπορούμε να συμπεράνουμε ότι οποιαδήποτε πρόταση P_i με $1 \leq i \leq N$ είναι αληθής.

3. Εισαγωγή σύζευξης

   Αν έχω μία σειρά από αληθείς προτάσεις στη βάση γνώσης, τότε μπορώ να συμπεράνω τη σύζευξή τους. Δηλαδή αν έχουμε $P_1, P_2, P_3, ..., P_N$ αληθείς προτάσεις, τότε μπορούμε να συμπεράνουμε ότι η $P_1 \wedge P_2 \wedge P_3 \wedge ... \wedge P_N$ είναι αληθής.

4. Εισαγωγή διαζεύξεων

   Αν έχω μία πρόταση στη βάση γνώσης που είναι αληθής, τότε μπορώ να βγάλω οποιεσδήποτε διαζεύξεις της προτασής μου με οποιεσδήποτε άλλες προτάσεις θέλω.

5. Ανάλυση (Resolution)

   Έστω ότι έχουμε δύο προτάσεις στη βάση γνώσης, μία διάζευξη και μία συνεπαγωγή της μορφής $P \vee Q$ και $Q \Rightarrow R$. Μετατρέποντας τις προτάσεις αυτές σε ΚΣΜ και έπειτα σε Kowalski θα καταλήξουμε σε μία νέα πρόταση που θα περιέχει τα P και Q των προηγούμενων προτάσεων. Πιο συγκεκριμένα:

   $$\text{ΚΣΜ: } P \vee Q \quad \neg Q \Rightarrow R$$

   $$\text{Kowalski: } P, Q \leftarrow \quad R \leftarrow Q$$

   Το Q απαλείφεται και έτσι έχουμε $P, R \leftarrow$

Παράδειγμα 1.4
Έστω οι παρακάτω προτάσεις στην βάση γνώσης:

1. Αν ομίχλη τότε κίνδυνος
2. Αν κίνδυνος τότε χαμηλή ταχύτητα
3. Έχει ομίχλη

Να αποδειχθεί ότι:

4. Χαμηλή ταχύτητα

Δηλαδή πρέπει να αποδειχθεί ότι η πρόταση 4 είναι αληθής.

Για να το κάνουμε αυτό, θα ακολουθήσουμε τον κανόνα της ανάλυσης, όπου πρώτα θα μετατρέψουμε αυτές τις 3 προτάσεις της βάσης γνώσης σε ΚΣΜ και έπειτα σε Kowalski. Έπειτα, θα εφαρμόσουμε κανόνες συμπερασμού ώστε τελικά να προκύψει η πρόταση 4.
Προτασιακή λογική:

1. ομίχλη $\Rightarrow$ κίνδυνος
2. κίνδυνος $\Rightarrow$ χαμηλή-ταχύτητα
3. ομίχλη

ΚΣΜ:

1. $\neg$ομίχλη $\vee$ κίνδυνος (από Χρυσό Κανόνα)
2. $\neg$κίνδυνος $\vee$ χαμηλή-ταχύτητα (από Χρυσό Κανόνα)
3. ομίχλη

Μορφή Kowalski:

1. κίνδυνος $\leftarrow$ ομίχλη
2. χαμηλή-ταχύτητα $\leftarrow$ κίνδυνος
3. ομίχλη $\leftarrow$

Έχοντας τις προτάσεις σε μορφή Kowalski, παίρνουμε τις προτάσεις 1, 2 και με τις κατάλληλες απαλοιφές έχουμε:

5. χαμηλή-ταχύτητα $\leftarrow$ ομίχλη

Παίρνουμε τις προτάσεις 3, 5 και με τις κατάλληλες απαλοιφές έχουμε:

$$\text{χαμηλή-ταχύτητα} \leftarrow$$

Η παραπάνω πρόταση είναι αληθής, άρα μόλις αποδείξαμε την πρόταση 4.

1.5 Απόδειξη με ανασκευή (εις άτοπο απαγωγή)

Ας υποθέσουμε ότι θέλουμε να δείξουμε ότι μια πρόταση είναι αληθής. Για να το κάνουμε αυτό χρησιμοποιώντας τη μέθοδο της ανασκευής, θα ξεκινήσουμε υποθέτοντας ότι η πρόταση είναι ψευδής. Αυτό σημαίνει ότι η άρνησή της είναι αληθής. Στη συνέχεια, με βάση κανόνες συμπερασμού, θα προσπαθήσουμε να δείξουμε ότι και η άρνηση της πρότασης είναι ψευδής, δηλαδή θα προσπαθήσουμε να καταλήξουμε σε αντίφαση. Εφόσον αυτό ισχύει, καταλήγουμε ότι η αρχική πρόταση είναι αληθής.

Παράδειγμα 1.5

Έστω οι παρακάτω προτάσεις στην βάση γνώσης μας:

1. Αν έχει ομίχλη τότε υπάρχει κίνδυνος
2. Αν υπάρχει κίνδυνος τότε απαιτείται χαμηλή-ταχύτητα
3. Έχει ομίχλη

Να αποδειχθεί ότι:

4. Απαιτείται χαμηλή-ταχύτητα

Όπως αναφέραμε προηγουμένως, στη μέθοδο της ανασκευής, ξεκινάμε υποθέτοντας ότι η πρόταση που θέλουμε να αποδείξουμε είναι ψευδής. Για να εφαρμόσουμε αυτή την άρνηση, πρέπει να μετατρέψουμε τις προτάσεις σε μορφή προτασιακής λογικής. Στη συνέχεια, θα έχουμε τις παρακάτω προτάσεις σε προτασιακή λογική, όπου η πρόταση 5 θα είναι η άρνηση της πρότασης που θέλουμε να αποδείξουμε.

1. ομίχλη $\Rightarrow$ κίνδυνος
2. κίνδυνος $\Rightarrow$ χαμηλή-ταχύτητα
3. ομίχλη
4. χαμηλή-ταχύτητα
5. $\neg$χαμηλή-ταχύτητα (άρνηση της 4)

Τώρα πρέπει να εφαρμόσουμε τους κανόνες συμπερασμού για να προχωρήσουμε στη διαδικασία. Πρώτα, μετατρέπουμε τις παραπάνω προτάσεις της προτασιακής λογικής σε κανονική συναρτησιακή μορφή (ΚΣΜ) και στη συνέχεια, τις εκφράζουμε σε μορφή Kowalski.

ΚΣΜ:

1. $\neg$ομίχλη $\vee$ κίνδυνος (από Χρυσό Κανόνα)
2. $\neg$κίνδυνος $\vee$ χαμηλή-ταχύτητα (από Χρυσό Κανόνα)
3. ομίχλη
4. χαμηλή-ταχύτητα
5. $\neg$χαμηλή-ταχύτητα

Μορφή Kowalski:

1. κίνδυνος $\leftarrow$ ομίχλη
2. χαμηλή-ταχύτητα $\leftarrow$ κίνδυνος

3. ομίχλη ←

4. χαμηλή-ταχύτητα ←

5. ← χαμηλή-ταχύτητα

Πράγματι, παίρνουμε τις προτάσεις 1, 3 και με τις κατάλληλες απαλοιφές έχουμε:

6. κίνδυνος ←

Παίρνουμε τις 2 και 5 και έχουμε:

7. ← κίνδυνος

Και τέλος καταλήγουμε σε αντίφαση αφού πάρουμε τις 6 και 7: ←.
Τελικά η πρόταση 4 αποδείχθηκε αληθής με την μεθοδολογία της ανασκευής.

1.6 Κατηγοριματική Λογική

Η **Κατηγορηματική Λογική (predicate logic)** αποτελεί επέκταση της προτασιακής λογικής και χρησιμοποιείται για την αναπαράσταση πιο σύνθετων μορφών γνώσης. Όπως και η προτασιακή λογική, έχει το δικό της λεξιλόγιο, συντακτικό και σημασιολογία, αλλά εμπλουτίζεται με δομές που επιτρέπουν την αναπαράσταση αντικειμένων και των ιδιοτήτων τους. Στο λεξιλόγιο της κατηγορηματικής λογικής περιλαμβάνονται μεταβλητές, συναρτήσεις, σταθερές, κατηγορήματα και ποσοδείκτες. Το συντακτικό της κατηγορηματικής λογικής αφορά τη σωστή διάταξη αυτών των στοιχείων ώστε να σχηματίζονται λογικές προτάσεις. Η σημασιολογία της ασχολείται με το νόημα αυτών των προτάσεων και αν αυτές είναι αληθείς ή ψευδείς, ανάλογα με τα αντικείμενα στα οποία αναφέρονται.

Οι προτάσεις στην κατηγορηματική λογική μπορούν να είναι πιο περίπλοκες σε σχέση με αυτές της προτασιακής λογικής, καθώς χρησιμοποιούν ποσοδείκτες, όπως **"για κάθε $\forall$" (universal quantifier)** και **"υπάρχει $\exists$" (existential quantifier)**. Αυτοί οι ποσοδείκτες προσδιορίζουν το πλήθος των αντικειμένων για τα οποία ισχύει μία δήλωση. Για παράδειγμα, η πρόταση "$\forall x$, το x είναι ζώο" είναι διαφορετική από την πρόταση "$\exists x$, το οποίο είναι ζώο", και αυτές οι δύο διακρίσεις παίζουν κρίσιμο ρόλο στην αναπαράσταση της γνώσης.

Ένα κατηγόρημα μπορεί να εκφράζει ιδιότητες, σχέσεις ή ενέργειες μεταξύ αντικειμένων. Μια **απλή πρόταση** στην κατηγορηματική λογική μπορεί να είναι μία δήλωση όπως "Ο Σωκράτης είναι άνθρωπος", η οποία μπορεί να αναπαρασταθεί ως ***Άνθρωπος(Σωκράτης)***. Μια **σύνθετη πρόταση** μπορεί να περιλαμβάνει λογικούς τελεστές, όπως στη δήλωση "Αν ο Σωκράτης είναι άνθρωπος, τότε ο Σωκράτης είναι θνητός", αναπαριστώμενη ως ***Άνθρωπος(Σωκράτης)*** $\Rightarrow$ ***Θνητός(Σωκράτης)***. Εδω να σημειωθεί ότι τα αντικείμενα μπορεί να είναι είτε μεταβλητές είτε σταθερές. Η χρήση των δύο είναι αρκετά προφανείς, αλλά αυτό που έχει σημασία είναι ότι οι σταθερές συμβολίζονται με το πρώτο τους γράμμα κεφαλαίο (Σωκράτης) ενώ οι μεταβλητές με όλα τους τα γράμματα πεζά.

Ο συνδυασμός ποσοδεικτών, κατηγορημάτων και λογικών τελεστών δημιουργεί **λογικές εκφράσεις** με μεγαλύτερη εκφραστική ισχύ σε σχέση με την προτασιακή λογική. Το αν μία τέτοια λογική έκφραση είναι αληθής ή ψευδής εξαρτάται από τις ιδιότητες των αντικειμένων στα οποία αναφέρεται και από τη χρήση των λογικών τελεστών και των ποσοδεικτών.

Όπως αναφέρθηκε και προηγουμένως, η κατηγορματική λογική αποτελεί επέκταση της προτασιακής λογικής. Έτσι, οι κανόνες συμπερασμού που είδαμε έχουν ισχύ και σε αυτό το κλάδο της λογικής. Στην κατηγορηματική λογική όμως ισχύουν άλλοι δύο κανόνες συμπερασμού:

6. Instantiation.

   Ισχύει: $\forall x P(x) \Rightarrow P(c)$, με x μεταβλητή, c σταθερά και το P ισχύει για όλα τα x ***και*** c. Με άλλα λόγια μπορούμε να πάμε από κάτι γενικό (από το $\forall x P(x)$) σε κάτι πιό ειδικό (στο $P(c)$) αν και τα δύο τους υπάγονται στην ίδια ιδιότητα (ισχύει το P).

7. Existential Generalization

   Ισχύει: $P(c) \Rightarrow \exists x P(x)$, με x μεταβλητή, c σταθερά και το P ισχύει για όλα τα x ***και*** c. Με άλλα λόγια μπορούμε να πάμε από το ειδικό (από το $P(c)$) σε κάτι πίο γενικό (στο $\exists x P(x)$) αν και τα δύο τους υπάγονται στην ίδια ιδιότητα (ισχύει το P).

Στον πίνακα 5 δίνονται παραδείγματα χρήσης κατηγορημάτων για να εκφράσουν ιδιότητα, σχέση ή ενέργεια. Η πρώτη γραμμή στην οποία το κατηγόρημα εκφράζει ιδιότητα: ***Άνθρωπος(***x) δηλώνει ότι το αντικείμενο x έχει την ιδιότητα να είναι άνθρωπος. Στην δεύτερη γραμμή όπου το κατηγόρημα εκφράζει σχέση: ***αγαπάει(***x, y***)*** δηλώνει ότι το αντικείμενο x είναι κάτι που αγαπάει και ότι το αντικείμενο y είναι κάτι που αγαπιέται. Τέλος, στην τρίτη γραμμή οπου το κατηγόρημα εκφράζει ενέργεια: ***δίνει(***x, y, z***)*** δηλώνει ότι το αντικείμενο x δίνει, το αντικείμενο y είναι αυτό που παίρνει το x και το αντικείμενο z αυτό που δίνεται.

Ο πίνακας 6 παρουσιάζει παραδείγματα προτάσεων που χρησιμοποιούν καθολικούς και υπαρξιακούς ποσοδείκτες, δείχνοντας πώς αυτοί οι ποσοδείκτες χρησιμοποιούνται για να εκφράσουν γενικές ή μεμονωμένες δηλώσεις. Οι καθολικοί ποσοδείκτες $\forall$ χρησιμοποιούνται για να δηλώσουν ότι μια ιδιότητα ισχύει για όλα τα αντικείμενα σε ένα συγκεκριμένο σύνολο, ενώ οι υπαρξιακοί ποσοδείκτες $\exists$ εκφράζουν ότι υπάρχει τουλάχιστον ένα αντικείμενο στο σύνολο για το οποίο ισχύει η ιδιότητα.

Στην πρώτη γραμμή, χρησιμοποιείται ο καθολικός ποσοδείκτης για τους ανθρώπους: $\forall x$ (***Άνθρωπος(x)*** $\Rightarrow$ ***Θνητός(x)***). Αυτή η πρόταση σημαίνει ότι "για κάθε άνθρωπο, ισχύει ότι είναι θνητός". Δηλώνει μια γενική αλήθεια που ισχύει για όλους τους ανθρώπους, κάνοντάς την μία καθολική δήλωση.

Στη δεύτερη γραμμή, υπάρχει η χρήση του υπαρξιακού ποσοδείκτη για τους ανθρώπους: $\exists x$ (***Άνθρωπος(x)*** $\wedge$ ***Γιατρός(x)***). Αυτή η πρόταση σημαίνει ότι "υπάρχει τουλάχιστον ένας άνθρωπος που είναι γιατρός". Εδώ δεν κάνουμε κάποια καθολική δήλωση για όλους τους ανθρώπους, αλλά αναγνωρίζουμε ότι υπάρχει τουλάχιστον ένα άτομο με αυτήν την ιδιότητα.

Η τρίτη γραμμή περιέχει ξανά τον καθολικό ποσοδείκτη, αυτή τη φορά για τους σκύλους: $\forall x$***Σκύλος(x)*** $\Rightarrow$ ***Θηλαστικό(x)***). Αυτό σημαίνει ότι "κάθε σκύλος είναι θηλαστικό", μια καθολική δήλωση για το σύνολο των σκύλων που εκφράζει ότι όλοι οι σκύλοι έχουν την ιδιότητα του θηλαστικού.

Στην τέταρτη γραμμή, χρησιμοποιείται ο υπαρξιακός ποσοδείκτης για τα φυτά: $\exists x$***Φυτό(x)*** $\wedge$ ***Δηλητηριώδες(x)***). Αυτό σημαίνει ότι "υπάρχει τουλάχιστον ένα φυτό που είναι δηλητηριώδες". Όπως και με την προηγούμενη υπαρξιακή πρόταση, δεν γίνεται δήλωση για όλα τα φυτά, αλλά αναγνωρίζεται ότι τουλάχιστον ένα φυτό έχει την ιδιότητα της δηλητηριώδους φύσης.

Ο πίνακας 7 παρουσιάζει τις τιμές αλήθειας για κατηγορηματικές προτάσεις που χρησιμοποιούν καθολικούς ($\forall$) και υπαρξιακούς ($\exists$) ποσοδείκτες. Κάθε πρόταση αξιολογείται ως αληθής ή ψευδής, ανάλογα με το αν πληροί τις συνθήκες που θέτει ο ποσοδείκτης και τα αντικείμενα στα οποία αναφέρεται.

Στην πρώτη πρόταση, χρησιμοποιείται ο καθολικός ποσοδείκτης για τους σκύλους: $\forall x$ (***Σκύλος(***x***)*** $\Rightarrow$ ***Θηλαστικό(***x***)***). Η πρόταση αυτή σημαίνει ότι για κάθε x, αν x είναι σκύλος, τότε πρέπει να είναι θηλαστικό. Η πρόταση είναι αληθής, γιατί όλοι οι σκύλοι είναι θηλαστικά, άρα δεν υπάρχει κανένα x που να ανήκει στην κατηγορία των σκύλων και να μην είναι θηλαστικό.

Η δεύτερη πρόταση περιέχει τον υπαρξιακό ποσοδείκτη για τους ανθρώπους: $\exists x$ (***Άνθρωπος(***x***)*** $\wedge$ ***Γιατρός(***x***)***). Αυτή η πρόταση δηλώνει ότι "υπάρχει τουλάχιστον ένας άνθρωπος που είναι γιατρός". Η πρόταση είναι αληθής, καθώς υπάρχει πράγματι τουλάχιστον ένας άνθρωπος που είναι γιατρός. Συνεπώς, η τιμή αλήθειας της πρότασης είναι αληθής.

Πίνακας 5: Κατηγορήματα.

Τι εκφράζει το κατηγόρημα	**Αναπαράσταση**
Ιδιότητα	***Άνθρωπος***(x)
Σχέση	***αγαπάει***(x, y)
Ενέργεια	***δίνει***(x, y, z)

Πίνακας 6: Παραδείγματα Καθολικών και Υπαρξιακών Ποσοδεικτών.

Ποσοδείκτης	**Παράδειγμα**
Καθολικός ($\forall$)	$\forall x$ (***Ανθρωπος(x)*** $\Rightarrow$ ***Θνητός(x)***)
Υπαρξιακός ($\exists$)	$\exists x$ (***Ανθρωπος(x)*** $\wedge$ ***Γιατρός(x)***)
Καθολικός ($\forall$)	$\forall x$ (***Σκύλος(x)*** $\Rightarrow$ ***Θηλαστικό(x)***)
Υπαρξιακός ($\exists$)	$\exists x$ (***Φυτό(x)*** $\wedge$ ***Δηλητηριώδες(x)***)

Η τρίτη πρόταση αφορά τα πτηνά και είναι της μορφής: $\forall x$ (***Πτηνό(***x***)*** $\Rightarrow$ ***Πετάει(***x***)***). Δηλώνει ότι για κάθε x, αν το x είναι πτηνό, τότε το x πρέπει να πετάει. Η πρόταση αυτή είναι ψευδής, καθώς υπάρχουν πτηνά, όπως οι πιγκουίνοι, που δεν μπορούν να πετάξουν. Επομένως, η πρόταση παραβιάζεται και η τιμή αλήθειας είναι ψευδής.

Η τελευταία πρόταση χρησιμοποιεί τον υπαρξιακό ποσοδείκτη για τα φυτά: $\exists x$ (***Φυτό(***x***)*** $\wedge$ ***Δηλητηριώδες(***x***)***). Σημαίνει ότι "υπάρχει τουλάχιστον ένα φυτό που είναι δηλητηριώδες". Αυτή η πρόταση είναι αληθής, καθώς πράγματι υπάρχουν φυτά που είναι δηλητηριώδη, όπως ο μανδραγόρας. Άρα η τιμή αλήθειας της πρότασης είναι αληθής.

Αξίζει να σημειωθεί ότι η σειρά των ποσοδεικτών επηρεάζει την τελική ερμηνεία της λογικής πρότασης. Για παράδειγμα, η πρόταση $\forall x \exists y$ ***αγαπά(***x, y***)*** έχει διαφορετική σημασία από την πρόταση $\exists y \forall x$ ***αγαπά(***x, y***)***. Η πρώτη εκφράζει ότι "για κάθε x που αγαπάει, υπάρχει τουλάχιστον ένα που αγαπιέται", δηλαδή για κάθε άτομο x που εκφράζει αγάπη, υπάρχει τουλάχιστον ένα άτομο y που είναι αντικείμενο της αγάπης του. Αντίθετα, η δεύτερη πρόταση δηλώνει ότι "υπάρχει τουλάχιστον ένα y που αγαπιέται από όλους", δηλαδή υπάρχει κάποιο άτομο που όλοι οι άλλοι αγαπούν.

Παρακάτω είναι οι νόμοι De Morgan για τους ποσοδείκτες:

- $\forall x(\neg P(x)) \Leftrightarrow \neg \exists x P(x)$
- $\exists x(\neg P(x)) \Leftrightarrow \neg \forall x P(x)$
- $\forall x \forall y(\alpha(x, y)) \Leftrightarrow \forall y \forall x \alpha(x, y)$
- $\forall x \alpha(x) \wedge \forall x \beta(x) \Leftrightarrow \forall x(\alpha(x) \wedge \beta(x))$
- $\exists x \alpha(x) \wedge \exists x \beta(x) \Leftrightarrow \exists x(\alpha(x) \wedge \beta(x))$
- $\forall x \alpha(x) \vee \forall x \beta(x) \Rightarrow \forall x \forall y(\alpha(x) \vee \beta(x))$
- $\exists x \alpha(x) \vee \exists x \beta(x) \Rightarrow \exists x \exists y(\alpha(x) \vee \beta(x))$

Πίνακας 7: Τιμές Αλήθειας για Κατηγορηματικές Προτάσεις.

Ποσοδείκτης	**Αντικείμενο**	**Πρόταση**	**Τιμή Αλήθειας**
$\forall$	Σκύλοι	$\forall x$ (***Σκύλος(***x***)*** $\Rightarrow$ ***Θηλαστικό(***x***)***)	Αληθής
$\exists$	Άνθρωποι	$\exists x$ (***Ανθρωπος(***x***)*** $\wedge$ ***Γιατρός(***x***)***)	Αληθής
$\forall$	Πτηνά	$\forall x$ (***Πτηνό(***x***)*** $\Rightarrow$ ***Πετάει(***x***)***)	Ψευδής
$\exists$	Φυτά	$\exists x$ (***Φυτό(***x***)*** $\wedge$ ***Δηλητηριώδες(***x***)***)	Αληθής

- $\forall x P(x) \Leftrightarrow \forall y P(y)$
- $\exists x P(x) \Leftrightarrow \exists y P(y)$

Παράδειγμα 1.6

Εδώ θα δούμε την κατηγορηματική μορφή κάποιων προτάσεων φυσικής γλώσσας. Έστω η πρόταση: "Mary loves everyone". Σε κατηγορηματική λογική θα αναπαρασταθεί ως: $\forall y$ ***loves(Mary,*** y***)***. Τώρα η πρόταση "Every student who walks talks" θα αναπαρασταθεί σε κατηγορηματική λογική ως: $\forall x($***student(***x***)*** $\wedge$ ***walk(***x***))*** $\Rightarrow$ ***talk(***x***)***, ενώ, η πρόταση "John does not love everyone" θα γίνει: $\forall y \neg$***love(John,*** y***)***. Τέλος οι προτάσεις "Someone walks and talks" και "No one who runs walks" θα αναπαραστιθούν ως $\exists x$***walk(***x***)*** $\wedge$ ***talk(***x***)***, $\neg \exists x$***run(***x***)*** $\wedge$ ***walk(***x***)***.

Παράδειγμα 1.7

Τώρα θα χρησιμοποιήσουμε την μεθοδολογία της απόδειξης με ανασκευή που είδαμε προηγουμένως στην κατηγορηματική μορφή προτάσεων. Έστω λοιπόν οι ακόλουθες προτάσεις:

1. Τα beletus είναι μανιτάρια
2. Τα μανιτάρια είναι άσπρα ή γκρίζα
3. Τα γκρίζα μανιτάρια είναι δηλητηριώδη
4. Κανένα beletus δεν είναι άσπρο

Να αποδείξετε ότι:

5. Τα beletus είναι δηλητηριώδη

Όπως ξέρουμε πρώτα θα μετατρέψουμε τις προτάσεις σε κατηγορηματική λογική:

1. ***μανιτάρι(BELETUS)***
2. $\forall x$***μανιτάρι(***x***)*** $\Rightarrow$ ***(άσπρο(***x***)*** $\vee$ ***γκρίζο(***x***))***
3. $\forall x$***(μανιτάρι(***x***)*** $\wedge$ ***γκρίζο(***x***))*** $\vee$ ***δηλητηριώδη(***x***)***
4. $\neg$***άσπρο(BELETUS)***
5. ***δηλητηριώδη(BELETUS)***
6. $\neg$***δηλητηριώδη(BELETUS)*** (άρνηση της 5)

Έπειτα θα μετατρέψουμε αυτές τις προτάσεις σε ΚΣΜ (όσες χρειάζονται):

1. ***μανιτάρι(BELETUS)***
2. $\neg$***μανιτάρι(***x***)*** $\vee$ ***άσπρο(***x***)*** $\vee$ ***γκρίζο(***x***)***
3. $\neg$***μανιτάρι(***x***)*** $\vee \neg$***γκρίζο(***x***)*** $\vee$ ***δηλητηριώδη(***x***)***
4. $\neg$***άσπρο(BELETUS)***
5. ***δηλητηριώδη(BELETUS)***
6. $\neg$***δηλητηριώδη(BELETUS)***

Μετά σε μορφή Kowalski:

1. ***μανιτάρι(BELETUS)*** $\leftarrow$
2. ***άσπρο(***x***), γκρίζο(***x***)*** $\leftarrow$ ***μανιτάρι(***x***)***
3. ***δηλητηριώδη(***x***)*** $\leftarrow$ ***μανιτάρι(***x***), γκρίζο(***x***)***

4. ← ***άσπρο(BELETUS)***

Και:

6. ← ***δηλητηριώδη(BEETUS)***

Τώρα θα πάρουμε τις προτάσεις 1 και 2 και με απαλοιφές θα έχουμε:

7. ***άσπρο(BELETUS), γκρίζο(BELETUS)*** ←

Τις 3 και 6:

8. ← ***μανιτάρι(BELETUS), γκρίζο(BELETUS)***

Τις 4 και 7:

9. ***γκρίζο(BELETUS)*** ←

Τώρα τις 8 και 9:

10. ← ***μανιτάρι(BELETUS)***

Και τέλος τις 1 και 10: ← όπου και καταλήγουμε σε άτοπο.

2 Αναζήτηση

Η αναζήτηση λύσης αποτελεί έναν θεμελιώδη μηχανισμό επίλυσης προβλημάτων στην Τεχνητή Νοημοσύνη. Τα προβλήματα που αντιμετωπίζει ένας αλγόριθμος αναζήτησης συνήθως μπορούν να περιγραφούν ως διαδικασίες μετάβασης από μία αρχική κατάσταση σε μία τελική (ή στόχο) μέσα από μία σειρά ενδιάμεσων καταστάσεων. Η διαδικασία αυτή καθοδηγείται από τους τελεστές μεταβάσεων, οι οποίοι καθορίζουν τις επιτρεπτές κινήσεις ή ενέργειες που μπορούν να μετατρέψουν μία κατάσταση σε κάποια άλλη. Ένα **μονοπάτι** σε αυτό το πλαίσιο είναι η αλληλουχία εφαρμογής των τελεστών μετάβασης. Επίσης να σημειωθεί ότι το μονοπάτι που δημιουργήθηκε μέχρι να βρεθεί η τελική λύση, μπορεί να διαφέρει από το μονοπάτι που θα μας δίνει την λύση (μονοπάτι που συνδέει αρχική με τελική κατάσταση) του προβλήματος. Αυτό γιατί πολύ απλά ένας αλγόριθμος μπορεί να εξετάσει διάφορες διαδρομές, διάφορους κόμβους, (ανάλογα τους μηχανισμούς του) μέχρι τελικά να βρεί την τελική κατάσταση. Το μονοπάτι που μας δίνει την λύση του προβήματος λέγεται πολύ απλά **μονοπάτι λύσης (solution path)**, ενώ αυτό που δημιουργήθηκε μέχρι τελικά να βρούμε το στόχο που ψάχνουμε λέγεται **ακολουθία ελέγχου**.

Κάθε πρόβλημα μπορεί να θεωρηθεί ως ένας χώρος αναζήτησης, όπου η **αρχική κατάσταση** είναι το σημείο εκκίνησης και η **τελική κατάσταση** ο στόχος που επιδιώκουμε να φτάσουμε. Η **λύση** του προβλήματος είναι η ακολουθία εφαρμογής **τελεστών μεταβάσεων** ή κινήσεων που θα οδηγήσει από την αρχική στην τελική κατάσταση.

Οι τελεστές μεταβάσεων είναι οι κανόνες που ορίζουν πώς μπορεί κανείς να προχωρήσει από μία κατάσταση σε μια άλλη. Κάθε τέτοια μετάβαση μπορεί να έχει ένα κόστος, ανάλογα με το πλαίσιο του προβλήματος, όπως για παράδειγμα η απόσταση μεταξύ δύο σημείων σε έναν χάρτη ή ο χρόνος που απαιτείται για μια κίνηση.

Οι αλγόριθμοι αναζήτησης διακρίνονται σε δύο μεγάλες κατηγορίες: τους **μη πληροφορημένους** και τους **πληροφορημένους (ή ευρετικούς)** αλγόριθμους. Οι μη πληροφορημένοι αλγόριθμοι αναζήτησης είναι εκείνοι που δεν διαθέτουν καμία πληροφορία για το χώρο αναζήτησης εκτός από την ίδια τη δομή του προβλήματος, ενώ οι πληροφορημένοι αλγόριθμοι αναζήτησης

είναι εκείνοι που εκμεταλλεύονται μία ευρετική συνάρτηση, η οποία τους βοηθά να εκτιμήσουν το κόστος ή την απόσταση από τον στόχο.

Η **ευρετική συνάρτηση** αποτελεί βασικό στοιχείο των πληροφοριμένων αλγορίθμων αναζήτησης. Παρέχει μία εκτίμηση του πόσο κοντά βρίσκεται μία κατάσταση στον στόχο και χρησιμοποιείται για να καθοδηγήσει τη διαδικασία αναζήτησης με τον πιο αποδοτικό τρόπο. Μια καλά σχεδιασμένη ευρετική συνάρτηση μπορεί να μειώσει δραστικά τον αριθμό των βημάτων που απαιτούνται για την εύρεση της λύσης, καθιστώντας την αναζήτηση πιο αποτελεσματική.

Το **δένδρο αναζήτησης (search tree)** είναι μια δομή δεδομένων που χρησιμοποιείται για την οπτικοποίηση και ανάλυση της διαδικασίας αναζήτησης λύσης σε ένα πρόβλημα. Κάθε κόμβος του δένδρου αναπαριστά μια κατάσταση του προβλήματος, ενώ οι ακμές (κλαδιά) αναπαριστούν τις μεταβάσεις (ή ενέργειες) που μπορούν να εφαρμοστούν για τη μετάβαση από μία κατάσταση σε μια άλλη. Η ρίζα του δένδρου αντιπροσωπεύει την **αρχική κατάσταση** του προβλήματος, και οι τερματικοί κόμβοι (φύλλα) αντιστοιχούν σε πιθανές λύσεις (**τερματικές καταστάσεις**) ή σε καταστάσεις όπου δεν υπάρχουν άλλες μεταβάσεις (**αδιέξοδα**). Στόχος της αναζήτησης (πάνω στο δένδρο αναζήτησης) είναι να βρεθεί η διαδρομή από τη ρίζα μέχρι έναν κόμβο που αντιπροσωπεύει μια λύση. Το μονοπάτι λοιπόν, που συνδέει μια τερματική κατάσταση με την ρίζα του δένδρου αποτελεί την λύση του προβλήματος.

Συνολικά, η αναζήτηση αποτελεί έναν ισχυρό μηχανισμό για την επίλυση προβλημάτων που μπορεί να εφαρμοστεί σε ένα ευρύ φάσμα πεδίων, από τη ρομποτική και τα παιχνίδια, μέχρι τη διαχείριση πόρων και την πλοήγηση σε σύνθετους χώρους.

2.1 Πληροφοριμένες Τεχνικές Αναζήτησης - Informed Search Algorithms

Οι **πληροφοριμένες τεχνικές αναζήτησης (ευρετικές τεχνικές - heuristic search)** βασίζονται στη χρήση ευρετικών συναρτήσεων, οι οποίες παρέχουν εκτιμήσεις για το πόσο κοντά βρίσκεται μία κατάσταση στον στόχο. Αυτό επιτρέπει στους αλγορίθμους να επικεντρώνονται σε πιο υποσχόμενες κατευθύνσεις, βελτιώνοντας την αποδοτικότητα της αναζήτησης. Εμείς θα εξετάσουμε τρεις βασικούς πληροφοριμένους αλγορίθμους: Best-First, Hill Climbing, και A*.

2.1.1 Best-First

Ο αλγόριθμος **Best-First** είναι μια πληροφορημένη τεχνική αναζήτησης που επιλέγει να εξετάσει πρώτα τον κόμβο με την καλύτερη εκτίμηση από μια ευρετική συνάρτηση. Σε κάθε βήμα, αξιολογεί τους γειτονικούς κόμβους και επιλέγει εκείνον με τη μικρότερη τιμή της ευρετικής συνάρτησης $h(n)$, η οποία υπολογίζει την "απόσταση" του κόμβου από τον τελικό στόχο. Άλλωστε το λέει και το όνομα του αλγορίθμου: οι καλύτεροι πρώτα, best-first.

Πιο αναλυτικά, η διαδικασία αναζήτησης του αλγόριθμου Best First μπορεί να περιγραφεί ως εξής:

1. Αρχική Κατάσταση: Ξεκινάμε από μια αρχική κατάσταση (κόμβο) του προβλήματος. Ο αλγόριθμος ελέγχει αν αυτή η κατάσταση είναι η τελική ή στόχος. Αν ναι, έχουμε βρει τη λύση και η αναζήτηση τερματίζεται.

2. Αξιολόγηση Κόμβων: Ο αλγόριθμος αξιολογεί όλους τους γειτονικούς κόμβους και υπολογίζει μια τιμή ευρετικής συνάρτησης ($h(n)$) για καθέναν από αυτούς. Αυτή η τιμή υποδεικνύει την εκτίμηση της απόστασης από τον κάθε κόμβο στον στόχο.

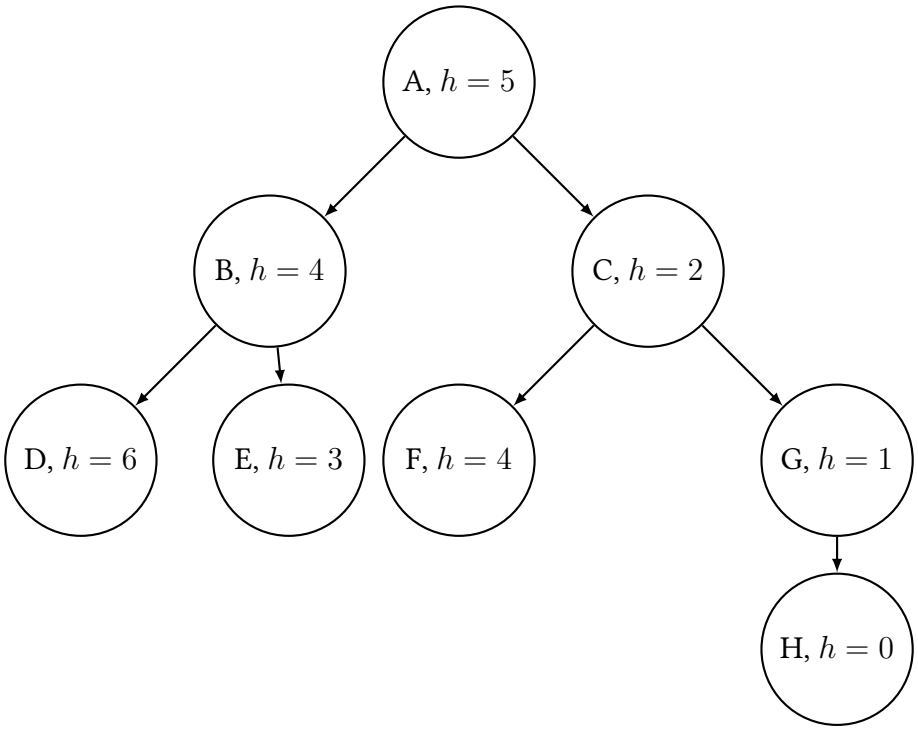
3. Επιλογή Διαδρομής: Ο αλγόριθμος επιλέγει τον κόμβο με την καλύτερη εκτίμηση (μικρότερη τιμή $h(n)$) για να εξερευνήσει πρώτος. Προχωρά στην εξερεύνηση αυτού του κόμβου και ελέγχει αν είναι ο στόχος.

4. Επέκταση Κόμβων: Εάν ο επιλεγμένος κόμβος δεν είναι ο στόχος, ο αλγόριθμος επεκτείνει τους γειτονικούς του κόμβους και υπολογίζει τις αντίστοιχες τιμές ευρετικής συνάρτησης για αυτούς.

5. Επαναλαμβανόμενη Διαδικασία: Η διαδικασία αυτή επαναλαμβάνεται, επιλέγοντας πάντα τον κόμβο με την καλύτερη εκτίμηση, μέχρι να βρεθεί η λύση ή να μην υπάρχουν άλλοι διαθέσιμοι κόμβοι.

6. Τερματισμός: Η διαδικασία τερματίζεται όταν βρεθεί η λύση ή όταν όλοι οι διαθέσιμοι κόμβοι έχουν εξερευνηθεί χωρίς να βρεθεί λύση.

Έστω ότι στο παρακάτω δέντρο αναζήτησης, ο Best-First αναζητά την καλύτερη πορεία από τον κόμβο A προς τον κόμβο H, εξετάζοντας πάντα τον κόμβο με την καλύτερη (μικρότερη) τιμή $h(n)$. Ξεκινάμε από τον κόμβο A ($h = 5$). Στη συνέχεια, ο αλγόριθμος εξετάζει τους κόμβους B (h=4) και C (h=2). Επειδή το $h(C) = 2$ είναι μικρότερο από το $h(B) = 4$, ο Best-First επιλέγει τον κόμβο C για να επεκτείνει. Από τον κόμβο C, υπάρχουν δύο επιλογές: F (h=4) και G (h=1). Επειδή το $h(G) = 1$ είναι μικρότερο από το $h(F) = 4$, ο αλγόριθμος επιλέγει τον κόμβο G. Τέλος, από τον κόμβο G, ο αλγόριθμος καταλήγει στον κόμβο H (h = 0). Η λύση του προβλήματος, όπως και η ακολουθία ελέγχου σε αυτή τη περίπτωση, (δηλαδή το Μονοπάτι που συνδέει την ρίζα με την τελική κατάσταση) είναι: A ⟶ C ⟶ G ⟶ H.



Σχήμα 1: Δένδρο Αναζήτησης για τον αλγόριθμο Best-First

2.1.2 Hill Climbing

Ο αλγόριθμος Hill-Climbing είναι μια πληροφορημένη τεχνική αναζήτησης. Η βασική ιδέα πίσω από αυτόν τον αλγόριθμο είναι να αναζητήσει λύσεις ξεκινώντας από μια αρχική κατάσταση και προχωρώντας προς μια καλύτερη κατεύθυνση, δηλαδή προς την κατεύθυνση όπου η ευρετική συνάρτηση βελτιώνεται, με στόχο να φτάσει σε μια κορυφή (λύση). Ο αλγόριθμος

εξετάζει τις γειτονικές καταστάσεις και επιλέγει εκείνη με την καλύτερη ευρετική τιμή. Ως εγκυκλοπεδικές γνώσεις, αλγόριθμος Hill Climbing είναι χρήσιμος για γρήγορες αναζητήσεις. Τέλος με αυτόν τον αλγόριθμο αναζητάμε κάθε φορά το τοπικό βέλιστο και ο ίδιος δεν είναι πλήρες.

Η διαδικασία αναζήτησης του αλγόριθμου Hill Climbing μπορεί να περιγραφεί ως εξής:

1. Αρχική Κατάσταση: Ξεκινάμε από μια αρχική κατάσταση (κόμβο) του προβλήματος. Ο αλγόριθμος ελέγχει αν αυτή η κατάσταση είναι η τελική ή στόχος. Αν ναι, έχουμε βρει τη λύση και η αναζήτηση τερματίζεται.

2. Εξερεύνηση Γειτονικών Κόμβων: Ο αλγόριθμος εξερευνά τους γειτονικούς κόμβους του τρέχοντος κόμβου και υπολογίζει την αξία (ή το κόστος) του καθενός.

3. Επιλογή Καλύτερης Κατάστασης: Με βάση την αξία των γειτονικών κόμβων, ο αλγόριθμος επιλέγει τον κόμβο με την καλύτερη αξία για να συνεχίσει την αναζήτηση. Αν η αξία αυτή είναι καλύτερη από την τρέχουσα κατάσταση, προχωρά στον νέο κόμβο.

4. Τερματισμός ή Επιστροφή: Εάν δεν υπάρχει γείτονας με καλύτερη αξία από την τρέχουσα κατάσταση, τότε ο αλγόριθμος τερματίζει την αναζήτηση, θεωρώντας ότι έχει φτάσει σε μια τοπική βέλτιστη λύση. Διαφορετικά, επαναλαμβάνει τη διαδικασία.

5. Εξέταση Τοπικών Βελτιστοτήτων: Η στρατηγική αυτή μπορεί να οδηγήσει σε τοπικές βελτιστοποιήσεις, αλλά δεν εγγυάται πάντα την εύρεση της καλύτερης (overall) λύσης.

Μπορεί κάποιος να το δει και αλλιώς. Ο Hill-Climbing λειτουργεί όπως η ανάβαση σε ένα βουνό: σε κάθε βήμα προσπαθεί να κινηθεί προς τα "πάνω", δηλαδή προς καλύτερες καταστάσεις σύμφωνα με την ευρετική συνάρτηση. Ο στόχος είναι να φτάσει στην κορυφή του βουνού, δηλαδή στην καλύτερη δυνατή λύση. Ωστόσο, αν δεν υπάρχουν γειτονικές καταστάσεις με καλύτερη ευρετική τιμή, ο αλγόριθμος σταματά, ακόμη κι αν δεν έχει βρεθεί η βέλτιστη λύση. Στην ουσία οι υπόλοιπες εναλλακτικές "καίγονται". Αυτή είναι άλλωστε και η διαφορά του με τον Best-First. Ο Hill-Climbing σε αντίθεση με τον Best-First, αν δεν βρεί κάποια καλύτερη ευρετική, δεν μπορεί να κάνει backtrack με σκοπό να βρεί κάτι καλύτερο στο προηγούμενο επίπεδο και έτσι εγκλωβίζεται χωρίς να βρεί λύση.

Παράδειγμα 2.1

Στο παραπάνω παράδειγμα, έχουμε ένα δέντρο αναζήτησης με διάφορες καταστάσεις και τις αντίστοιχες ευρετικές τιμές τους (h). Έστω ότι θέλουμε να φτάσουμε στον στόχο G.

Ο αλγόριθμος Hill-Climbing ξεκινά από την A με ευρετική τιμή 7 και εξετάζει τις γειτονικές καταστάσεις:

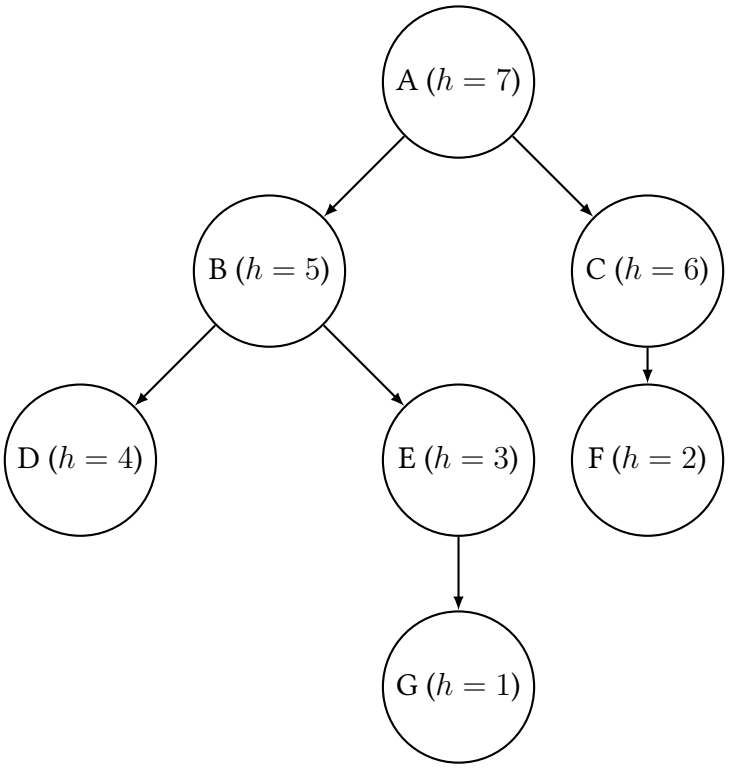
- Η B έχει ευρετική τιμή 5.
- Η C έχει ευρετική τιμή 6.

Ο Hill-Climbing επιλέγει τον κόμβο B καθώς έχει την χαμηλότερη ευρετική τιμή. Στη συνέχεια, εξετάζει τα παιδία κόμβους του B:

- Η D έχει ευρετική τιμή 4.
- Η E έχει ευρετική τιμή 3.

Ο αλγόριθμος επιλέγει τον κόμβο E και συνεχίζει ελέγχοντας τους κόμβους παιδιά του:

- Η G έχει ευρετική τιμή 1.



Σχήμα 2: Δένδρο Αναζήτησης για τον αλγόριθμο Hill-Climbing

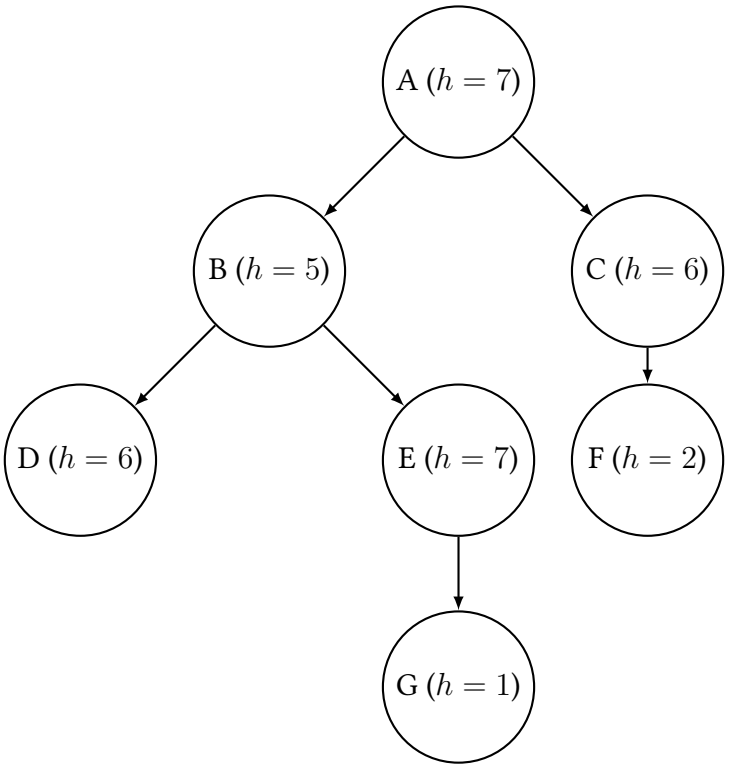
Τέλος ο Hill Climbing επιλέγει τον κόμβο G, ο οποίος είναι και ο στόχος που αναζητούμε.

Παράδειγμα 2.2

Ας δούμε τώρα ένα παράδειγμα εκτέλεσης του Hill Climbing στο οποίο ο αλγόριθμος "παγιδεύεται" σε ένα κόμβο, χωρίς να μπορεί να κάνει backtrack. Πάλι η αρχική κατάσταση μας είναι ο κόμβος A και η τελική ο κόμβος G.

1. Ξεκινάμε από τον κόμβο A με $h = 7$. Ο αλγόριθμος εξετάζει τους γειτονικούς κόμβους του A, δηλαδή τους B και C.
   - Ο κόμβος B έχει $h = 5$.
   - Ο κόμβος C έχει $h = 6$.
   - Ο αλγόριθμος επιλέγει τον κόμβο B, επειδή έχει τη μικρότερη τιμή h.
2. Μεταβαίνουμε στον κόμβο B με $h = 5$. Οι γειτονικοί κόμβοι του B είναι οι D και E.
   - Ο κόμβος D έχει $h = 6$.
   - Ο κόμβος E έχει $h = 7$.
   - Ο αλγόριθμος παρατηρεί ότι καμία από τις δύο επιλογές (D και E) δεν έχει μικρότερη τιμή από το h του B ($h = 5$). Άρα, παγιδεύεται σε τοπικό μέγιστο και σταματάει την αναζήτηση.

Αυτό είναι το κλασικό πρόβλημα του Hill Climbing: μπορεί να "κολλήσει" σε τοπικά μέγιστα, χωρίς να φτάσει στον τελικό στόχο. Στο συγκεκριμένο δένδρο, ο κόμβος G ($h = 1$) είναι η λύση, αλλά δεν μπορεί να επιλεχθεί λόγω της φύσης του αλγορίθμου, καθώς το Hill Climbing δεν εξετάζει επιστροφή σε προηγούμενους κόμβους ή αποθήκευση πολλών μονοπατιών.



Σχήμα 3: Δένδρο Αναζήτησης για τον αλγόριθμο Hill-Climbing: η αναζήτηση παγιδεύεται

2.1.3 A*

Ο αλγόριθμος A* (A-Star) είναι μία πληροφορημένη τεχνική αναζήτησης που χρησιμοποιείται για την εύρεση της βέλτιστης λύσης σε προβλήματα διαδρομών ή γενικότερα σε προβλήματα που μπορούν να αναπαρασταθούν ως δένδρα αναζήτησης. Ο A* συνδυάζει τις στρατηγικές της απληστίας (best-first) και του πραγματικού κόστους με σκοπό να επιτύχει αποδοτικότητα. Τέλος, ο A* δεν εγγυάται από μόνος του την εύρεση βέλτιστης λύσης. Για να εγγυηθεί ο A* βέλτιστη λύση θα πρέπει η ευρετική συνάρτηση να είναι **αποδεκτή**. Αποδεκτή λέγεται η ευρετική συνάρτηση που για όλους τους κόμβους δεν υπερεκτιμά την πραγματική απόσταση από το τερματικό κόμβο. Αντιθέτως, **μη αποδεκτή** λέγεται η ευρετική που όταν έστω για ένα κόμβο (μία κατάσταση), υπερεκτιμά την πραγματική του απόσταση από το τέρμα. Με πιο απλά λόγια, όταν η ευρετική είναι μικρότερη από το πραγματικό κόστος τότε είναι αποδεκτή και όταν έστω για έναν κόμβο το πραγματικό κόστος είναι μεγαλύτερο από την ευρετική του τότε είναι μη αποδεκτή.

Η διαδικασία αναζήτησης του αλγόριθμου A* μπορεί να περιγραφεί ως εξής:

1. Αρχική Κατάσταση: Ξεκινάμε από μια αρχική κατάσταση (κόμβο) του προβλήματος. Ο αλγόριθμος ελέγχει αν αυτή η κατάσταση είναι η τελική ή στόχος. Αν ναι, έχουμε βρει τη λύση και η αναζήτηση τερματίζεται.

2. Αξιολόγηση Κόμβων: Ο αλγόριθμος αξιολογεί τους γειτονικούς κόμβους, υπολογίζοντας την τιμή $f(n) = g(n) + h(n)$, όπου $g(n)$είναι το (πραγματικό) κόστος για να φτάσουμε στον κόμβο n και $h(n)$ είναι η ευρετική τιμή του κόστους από τον κόμβο n στον στόχο.

3. Επιλογή Διαδρομής: Ο αλγόριθμος επιλέγει τον κόμβο με την μικρότερη τιμή $f(n)$ για εξερεύνηση.

4. Επέκταση Κόμβων: Εάν ο επιλεγμένος κόμβος δεν είναι ο στόχος, ο αλγόριθμος επεκτείνει τους γειτονικούς του κόμβους, υπολογίζοντας τις αντίστοιχες τιμές $f(n)$ για αυτούς.

5. Επαναλαμβανόμενη Διαδικασία: Η διαδικασία αυτή επαναλαμβάνεται, επιλέγοντας πάντα τον κόμβο με την καλύτερη εκτίμηση $f(n)$, μέχρι να βρεθεί η λύση ή να μην υπάρχουν άλλοι διαθέσιμοι κόμβοι.

6. Τερματισμός: Η διαδικασία τερματίζεται όταν βρεθεί η λύση ή όταν όλοι οι διαθέσιμοι κόμβοι έχουν εξερευνηθεί χωρίς να βρεθεί λύση.

Παράδειγμα 2.3

Παρακάτω ακολουθεί ένα παράδειγμα εκτέλεσης του αλγορίθμου A*. Όπως είπαμε προηγουμένως, ο A*χρησιμοποιεί τη συνάρτηση κόστους $f(n) = g(n) + h(n)$, όπου:

- $g(n)$ είναι το πραγματικό κόστος από τον αρχικό κόμβο έως τον κόμβο n.
- $h(n)$ είναι η ευρετική εκτίμηση του κόστους από τον κόμβο n έως τον στόχο.

Οι κόμβοι που εξετάζονται στο παράδειγμα είναι:

- A (Αρχικός κόμβος)
- B, C (Παιδιά του A)
- D, E (Παιδιά του B)
- F, G (Παιδιά του C, με το G να είναι ο στόχος)

Πίνακας 8: Κόστος και ευρετική εκτίμηση κόμβων

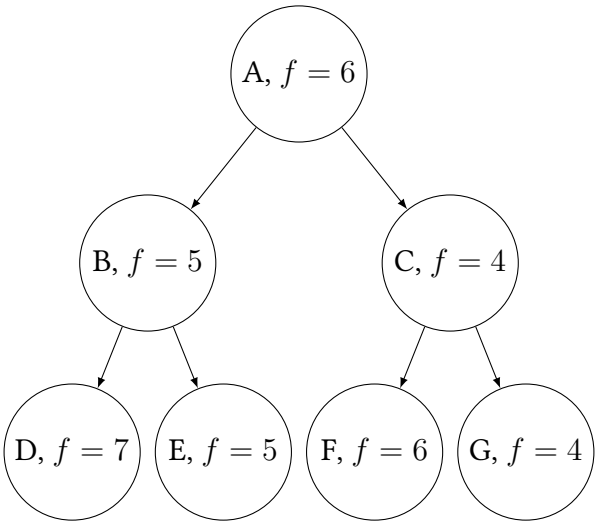
Κόμβος	**g(n)**	**h(n)**	**f(n) = g(n) + h(n)**
A	0	6	6
B	1	4	5
C	2	2	4
D	2	5	7
E	2	3	5
F	3	1	4
G (Στόχος)	4	0	4

Στο παρακάτω σχήμα παρουσιάζεται το δέντρο αναζήτησης με τον αλγόριθμο A*. Ξεκινώντας από τον κόμβο A, ο A* εξετάζει κάθε κόμβο με βάση τη συνάρτηση κόστους $f(n)$ και επιλέγει τον κόμβο με τη χαμηλότερη τιμή. Η διαδικασία συνεχίζεται μέχρι να βρεθεί ο στόχος.

Βήματα επίλυσης:

- Αρχικός Κόμβος (A): Ξεκινάμε από τον κόμβο A. Το πραγματικό του κόστος είναι $g(A) = 0$ και η ευρετική του εκτίμηση είναι $h(A) = 6$, άρα $f(A) = 0 + 6 = 6$.
- Επιλογή Κόμβων B και C: Ο A* επεκτείνει τα παιδιά του A, δηλαδή τους κόμβους B και C.
  - Για τον κόμβο B: $g(B) = 1$, $h(B) = 4$, άρα $f(B) = 1 + 4 = 5$.
  - Για τον κόμβο C: $g(C) = 2$, $h(C) = 2$, άρα $f(C) = 2 + 2 = 4$.

  Από τους κόμβους B και C, ο αλγόριθμος επιλέγει τον C επειδή έχει το μικρότερο $f(n)$.
- Επιλογή Κόμβων F και G: Ο A* επεκτείνει τα παιδιά του C, δηλαδή τους κόμβους F και G.
  - Για τον κόμβο F: $g(F) = 3$, $h(F) = 3$, άρα $f(F) = 3 + 3 = 6$.



Σχήμα 4: Δέντρο αναζήτησης για τον αλγόριθμο A*

– Για τον κόμβο G (στόχος): $g(G) = 4$, $h(G) = 0$, άρα $f(G) = 4 + 0 = 4$.

Ο αλγόριθμος επιλέγει τον κόμβο G, αφού είναι ο στόχος και έχει το μικρότερο $f(n)$.

- Λύση: Το μονοπάτι που βρέθηκε από τον A στον G είναι: $A \to C \to G$ με συνολικό κόστος $f(G) = 4$.

Αν παρατηρήσουμε το πίνακα 8 θα δούμε ότι η ευρετική συνάρτηση $h(n)$ είναι μη αποδεκτή, καθώς υπερεκτιμά την πραγματική απόσταση , όχι μόνο ενός, αλλά αρκετών κόμβων. Σε αυτή τη περίπτωση ο A* δεν μας εγγυάται βέλτιστη λύση. Το ότι τελικά μας βρήκε την βέλτιστη λύση είναι κάτι εντελώς το διαφορετικό όμως. Άλλο η εκτίμηση, αλλά το τελικό αποτέλεσμα.

2.2 Μη Πληροφορημένες Τεχνικές Αναζήτησης - Uninformed Search Algorithms

Οι **μη πληροφοριμένες τεχνικές αναζήτησης (τυφλές τεχνικές - blind search)** είναι αλγόριθμοι αναζήτησης οι οποίοι δεν χρησιμοποιούν προηγούμενες γνώσεις σχετικές με το πρόβλημα. Ως εκ τούτου, έχοντας στην διάθεσή τους μόνο τον ορισμό του προβλήματος, εξετάζουν κάθε πιθανή λύση συστηματικά. Εμείς θα δούμε τις τυφλές τεχνικές: DFS, BFS, Uniform Cost και iterative deepening.

2.2.1 Αναζήτηση κατά Βάθος - Depth First Search (DFS)

Ο αλγόριθμος **Αναζήτησης κατά Βάθος (Depth-First Search, DFS)** είναι μία από τις πιο διαδεδομένες τεχνικές αναζήτησης που χρησιμοποιούνται στην Τεχνητή Νοημοσύνη για την επίλυση προβλημάτων που μπορούν να απεικονιστούν ως δέντρα αναζήτησης. Η βασική ιδέα πίσω από τον αλγόριθμο DFS είναι ότι εξερευνά όσο το δυνατόν πιο βαθιά έναν κόμβο προτού επιστρέψει και εξετάσει άλλους γείτονες. Αυτό επιτυγχάνεται μέσω μιας στοίβας, που μπορεί να είναι είτε ρητή, είτε υλοποιημένη μέσω αναδρομής. Όμως πρακτικά σχετικά με το μάθημα που θα εξεταστείς, δεν θα ζητηθεί η "στοίβα" αυτή.

Η διαδικασία αναζήτησης του αλγόριθμου DFS (Depth-First Search) μπορεί να περιγραφεί ως εξής:

1. Αρχική Κατάσταση: Ξεκινάμε από μια αρχική κατάσταση (κόμβο) του προβλήματος. Ο αλγόριθμος ελέγχει αν αυτή η κατάσταση είναι η τελική ή στόχος. Αν ναι, έχουμε βρει τη λύση και η αναζήτηση τερματίζεται.

2. Εξερεύνηση Κόμβων: Αν ο τρέχων κόμβος δεν είναι ο στόχος, ο αλγόριθμος προχωρά στην εξερεύνηση των γειτονικών κόμβων (γονέων). Ξεκινά από τον πρώτο γείτονα και συνεχίζει να προχωρά σε βάθος. Δηλαδή θα εξερευνήσει τα παιδιά του. Για κάθε κόμβο παιδό, ελέγχει αν είναι ο στόχος. Τώρα, ένας κόμβος είναι πολύ πιθανό να έχει περισσότερους από έναν κόμβους παιδιά. Το ποιόν κόμβο παιδί θα διαλέξεις πρώτα θα το αποφασίσεις με βάση το κριτήριο επιλογής που θα δίνεται απο την άσκηση. Μπορεί αυτό το κριτήριο να σου λέει να εξερευνήσεις τους κόμβους παιδιά απο αριστερά στα δεξιά, ή με αλφαβητική σειρά. Εμείς σε όλα τα παραδείγματά μας θα εξερευνούμε τους κόμβους με αλφαβητική σειρά.

3. Επιλογή Διαδρομής: Ο αλγόριθμος συνεχίζει να κινείται σε βάθος, επιλέγοντας κάθε φορά τον επόμενο διαθέσιμο και προχωρώντας στην εξερεύνηση.

4. Backtracking: Εάν ο αλγόριθμος φτάσει σε έναν κόμβο που δεν έχει διαθέσιμους κόμβους (ή αν όλους τους έχει επισκεφθεί) και δεν έχει βρει τον στόχο, επιστρέφει στον προηγούμενο κόμβο και συνεχίζει την αναζήτηση με τους υπόλοιπους. Αυτή η διαδικασία συνεχίζεται έως ότου είτε βρει τη λύση είτε εξερευνήσει όλους τους δυνατούς κόμβους χωρίς επιτυχία.

5. Τερματισμός: Η διαδικασία τερματίζεται είτε όταν βρεθεί η λύση (στόχος) είτε όταν όλοι οι διαθέσιμοι κόμβοι έχουν εξερευνηθεί χωρίς να βρεθεί λύση.

Να σημειωθεί ότι στον DFS υπάρχει κίνδυνος να εγκλωβιστεί σε κύκλους ή να επισκεφθεί το ίδιο μονοπάτι πολλές φορές, αν δεν υπάρχει κάποιος μηχανισμός για την αποφυγή επαναλήψεων. Θα δούμε στην συνέχεις μία εκδοχή του DFS με "στεροειδή", όπου υπάρχει μηχανισμός για την αποφυγή επαναλήψεων.

Παράδειγμα 2.4

Έστω ότι έχουμε το δένδρο αναζήτησης στο Σχήμα 5, όπου ο στόχος μας είναι να βρούμε την τελική κατάσταση, που αντιπροσωπεύεται από τον κόμβο G. Αρχικά, η αναζήτηση ξεκινά από τον κόμβο A, την αρχική κατάσταση του προβλήματος. Προσθέτουμε τον A στο μονοπάτι:

- Μονοπάτι: [A]
- Μονοπάτι αναζήτησης: [A]

Εξερευνούμε με αλφαβητική σειρά, οπότε πρώτα έχουμε το πρώτο παιδί του A τον κόμβο B. Προσθέτουμε τον B στο μονοπάτι:

- Μονοπάτι: [A, B]
- Ακολουθία ελέγχου: [A, B]

Έπειτα από τον B, μεταβαίνουμε στο πρώτο παιδί του, τον κόμβο D. Προσθέτουμε τον D στο μονοπάτι:

- Μονοπάτι: [A, B, D]
- Μονοπάτι αναζήτησης: [A, B, D]

Ο D δεν είναι η τελική κατάσταση και δεν έχει παιδιά, άρα επιστρέφουμε (backtrack) στον κόμβο B. Επιστρέφουμε λοιπόν στον B και εξερευνούμε το επόμενο παιδί του, τον κόμβο E. Προσθέτουμε τον E στο μονοπάτι:

- Μονοπάτι: [A, B, E]
- Ακολουθία ελέγχου: [A, B, D, E]

Μετά από τον κόμβο E, προχωρούμε στο παιδί του, τον G, ο οποίος είναι η τελική κατάσταση (στόχος). Προσθέτουμε τον G στο μονοπάτι:

- Μονοπάτι: [A, B, E, G]
- Ακολουθία ελέγχου: [A, B, D, E, G]

Η λύση βρέθηκε, καθώς φτάσαμε στον κόμβο G. Η αναζήτηση ολοκληρώθηκε. Το τελικό μονοπάτι που ακολουθήσαμε για να φτάσουμε στη λύση είναι:

- Μονοπάτι λύσης: [A, B, E, G]

Αλλά η ακολουθία ελέγχου που ακολουθήσαμε μέχρι να βρούμε την τελική λύση είναι:

- Ακολουθία ελέγχου: [A, B, D, E, G]

2.2.2 Αναζήτηση κατά Πλάτος - Breast First Search (BFS)

Ο αλγόριθμος **Αναζήτησης κατά Πλάτους (Breadth-First Search, BFS)** είναι μία τεχνική αναζήτησης που εξερευνά συστηματικά κάθε επίπεδο ενός δένδρου ή γράφου, πριν προχωρήσει στο επόμενο επίπεδο. Αντί να πηγαίνει όσο το δυνατόν πιο βαθιά σε κάθε μονοπάτι, όπως κάνει ο DFS, ο BFS εξετάζει πρώτα όλους τους κόμβους που βρίσκονται στο ίδιο επίπεδο από την αρχική κατάσταση, κι έπειτα κατεβαίνει στο επόμενο επίπεδο.

Η διαδικασία εκτέλεσης του BFS ξεκινάει με την αναζήτηση από την ρίζα του δένδρου (ή αρχική κατάσταση). Αρχικά, εξετάζει όλους τους γειτονικούς κόμβους της ρίζας (το πρώτο επίπεδο). Στη συνέχεια, προχωράει στο επόμενο επίπεδο του δένδρου και εξετάζει τους γειτονικούς κόμβους των κόμβων του πρώτου επιπέδου. Αυτή η διαδικασία συνεχίζεται έως ότου βρεθεί η τελική κατάσταση. Η αναζήτηση σταματά μόλις εντοπιστεί η τελική κατάσταση (λύση). Η διαδικασία ολοκληρώνεται με την επιστροφή του μονοπατιού που συνδέει την αρχική κατάσταση με την τελική.

Τα βήματα της αναζήτησης του BFS περιγράφονται ως εξής:

- Ξεκινάει από την αρχική κατάσταση, η οποία αναπαρίσταται από τη ρίζα του δένδρου αναζήτησης.
- Εξερευνά όλους τους κόμβους που είναι άμεσα συνδεδεμένοι με την αρχική κατάσταση.
- Προχωράει στο επόμενο επίπεδο και εξερευνά τους κόμβους που είναι συνδεδεμένοι με τους κόμβους του προηγούμενου επιπέδου.
- Η διαδικασία συνεχίζεται μέχρι να βρεθεί η τελική κατάσταση.
- Μόλις βρεθεί η τελική κατάσταση, η αναζήτηση ολοκληρώνεται και έτσι έχουμε το μονοπάτι που οδηγεί από την αρχική κατάσταση στο στόχο.

Παράδειγμα 2.5
Έστω ότι έχουμε το δένδρο αναζήτησης στο Σχήμα 5, όπου ο στόχος μας είναι να βρούμε την τελική κατάσταση, που αντιπροσωπεύεται από τον κόμβο G. Αρχικά, η αναζήτηση ξεκινά από την ρίζα του δένδρου, τον κόμβο A, την αρχική κατάσταση. Προσθέτουμε τον κόμβο A στο μονοπάτι:

- Μονοπάτι: [A]
- Ακολουθία ελέγχου: [A]

Στη συνέχεια, εξετάζουμε διαδοχικά όλα τα παιδιά του A. Πρώτα επισκεπτόμαστε τον κόμβο B και τον προσθέτουμε στο μονοπάτι:

- Μονοπάτι: [A, B]
- Ακολουθία ελέγχου: [A, B]

Έπειτα, επισκεπτόμαστε τον κόμβο C, το δεύτερο παιδί του A, και τον προσθέτουμε στο μονοπάτι:

- Μονοπάτι: [A, B, C]
- Ακολουθία ελέγχου: [A, B, C]

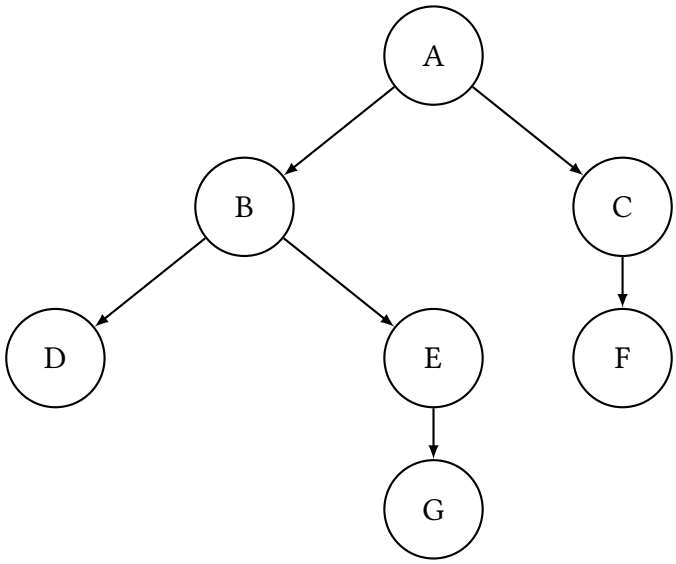
Αφού εξετάσαμε όλους τους κόμβους στο πρώτο επίπεδο, προχωρούμε στο δεύτερο επίπεδο. Πρώτα εξετάζουμε τον κόμβο D, που είναι παιδί του B. Προσθέτουμε τον D στο μονοπάτι:

- Μονοπάτι: [A, B, C, D]
- Ακολουθία ελέγχου: [A, B, C, D]

Ο κόμβος D δεν έχει παιδιά και δεν είναι η τελική κατάσταση, επομένως προχωράμε στον επόμενο κόμβο. Εξετάζουμε τον κόμβο E, που είναι παιδί του B, και τον προσθέτουμε στο μονοπάτι:

- Μονοπάτι: [A, B, C, D, E]
- Ακολουθία ελέγχου: [A, B, C, D, E]

Στη συνέχεια, εξετάζουμε τον κόμβο F, που είναι παιδί του C, και τον προσθέτουμε στο μονοπάτι:



Σχήμα 5: Δένδρο Αναζήτησης για τους αλγορίθμους DFS και BFS

- Μονοπάτι: [A, B, C, D, E, F]
- Ακολουθία ελέγχου: [A, B, C, D, E, F]

Στο επόμενο επίπεδο, εξετάζουμε τον κόμβο G, παιδί του E, ο οποίος είναι η τελική κατάσταση. Προσθέτουμε τον G στο μονοπάτι:

- Μονοπάτι: [A, B, C, D, E, F, G]
- Ακολουθία ελέγχου: [A, B, C, D, E, F, G]

Η λύση βρέθηκε όταν φτάσαμε στον κόμβο G. Η αναζήτηση ολοκληρώθηκε και το τελικό μονοπάτι που ακολουθήσαμε για να φτάσουμε στη λύση είναι:

- Μονοπάτι λύσης: [A, B, E, G]

Η ακολουθία των κόμβων που εξετάστηκαν για να βρεθεί η τελική κατάσταση είναι:

- Ακολουθία ελέγχου: [A, B, C, D, E, F, G]

2.2.3 Uniform Cost Search

Ο αλγόριθμος **Uniform Cost Search (UCS)** είναι μία τεχνική αναζήτησης που βασίζεται στο κόστος των μεταβάσεων για να βρει τη λύση με το μικρότερο συνολικό κόστος. Σε αντίθεση με τον BFS, ο οποίος εξετάζει όλα τα επίπεδα του δένδρου αναζήτησης ισότιμα, ο UCS προχωρά εξετάζοντας πάντα τον κόμβο με το χαμηλότερο σωρευτικό κόστος από την αρχική κατάσταση.

Η διαδικασία εκτέλεσης του UCS ξεκινά από τον αρχικό κόμβο και προχωρά εξερευνώντας διαδρομές με το μικρότερο κόστος. Ο UCS προσθέτει τους κόμβους σε μια προτεραιότητα βάσει του συνολικού κόστους της διαδρομής από την αρχική κατάσταση μέχρι τον κόμβο αυτό. Όσο εξερευνά, ενημερώνει το συνολικό κόστος κάθε διαδρομής και προχωρά εξετάζοντας πρώτα τους κόμβους με το μικρότερο συνολικό κόστος.

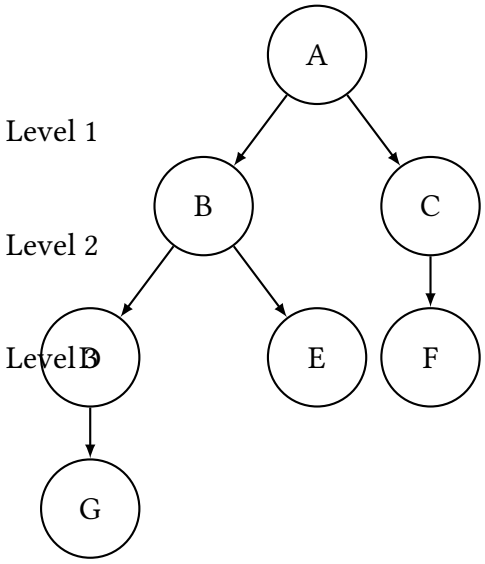
Ο UCS εξασφαλίζει ότι θα βρεθεί πάντα η λύση με το χαμηλότερο κόστος. Αντί να εξετάζει απλώς το βάθος ή τα επίπεδα του δένδρου, όπως ο DFS ή ο BFS, ο UCS είναι σχεδιασμένος για προβλήματα όπου το κόστος μεταβάσεων μας ενδιαφέρει, και το ζητούμενο είναι προφανώς η οικονομικότερη διαδρομή.

Παράδειγμα 2.6

Έστω ότι έχουμε το παρακάτω δένδρο αναζήτησης, όπου κάθε μετάβαση έχει το δικό της κόστος, και ο στόχος μας είναι να βρούμε την τελική κατάσταση, που αντιπροσωπεύεται από τον κόμβο G. Τα κόστη των μεταβάσεων φαίνονται στον πίνακα 9.

Πίνακας 9: Κόστος μετάβασης κόμβων

Κόμβος	**Κόστος**
A	-
A $\rightarrow$ B	2
A $\rightarrow$ C	4
B $\rightarrow$ D	3
B $\rightarrow$ E	5
C $\rightarrow$ F	3
D $\rightarrow$ G (Στόχος)	1



Σχήμα 6: Δένδρο Αναζήτησης για τον αλγόριθμο UCS

Η αναζήτηση ξεκινά από την αρχική κατάσταση A. Το κόστος μετάβασης από τον A στον B είναι 2 και από τον A στον C είναι 3. Ξεκινάμε με το μονοπάτι:

- Μονοπάτι: [A]
- Ακολουθία ελέγχου: [A]
- Κόστος: 0

Εξετάζουμε το μονοπάτι με το μικρότερο κόστος, που είναι το A -> B (κόστος 2):

- Μονοπάτι: [A, B]
- Ακολουθία ελέγχου: [A, B]
- Κόστος: 2

Στη συνέχεια, εξετάζουμε τα παιδιά του B. Από το B, η μετάβαση στο D κοστίζει 3 και στο E κοστίζει 5. Προχωράμε στον κόμβο D, καθώς έχει το μικρότερο κόστος (2+3=5):

- Μονοπάτι: [A, B, D]
- Ακολουθία ελέγχου: [A, B, D]
- Κόστος: 5

Μετά από τον κόμβο D, η μετάβαση στον G κοστίζει 1 μονάδα. Προχωράμε στον κόμβο G, που είναι η τελική κατάσταση:

- Μονοπάτι: [A, B, D, G]
- Ακολουθία ελέγχου: [A, B, D, G]
- Κόστος: 6

Η λύση βρέθηκε, καθώς φτάσαμε στον κόμβο G με συνολικό κόστος 6 μονάδες. Το τελικό μονοπάτι που ακολουθήσαμε για να φτάσουμε στη λύση είναι:

- Μονοπάτι λύσης: [A, B, D, G]
- Συνολικό κόστος: 6 μονάδες

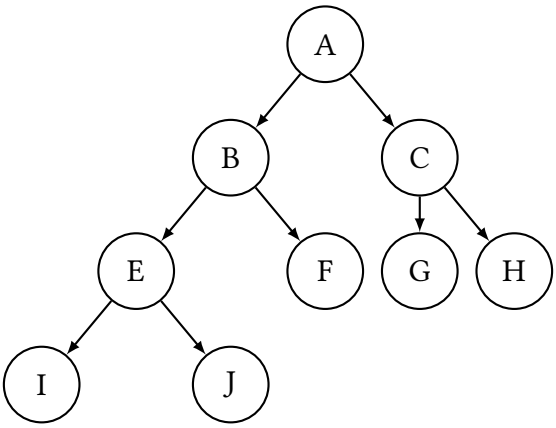
2.2.4 Επαναληπτική Εκβάθυνση - Iterative Deepening

Ο αλγόριθμος **Επαναληπτικής Εκβάθυνσης (Iterative Deepening)** είναι μια τεχνική αναζήτησης που συνδυάζει τα πλεονεκτήματα της Αναζήτησης Κατά Βάθος (DFS) και της Αναζήτησης Κατά Πλάτος (BFS). Ο αλγόριθμος αυτός λειτουργεί σαν DFS με όριο βάθους, αλλά επαναλαμβάνεται πολλές φορές, αυξάνοντας το όριο κάθε φορά, έως ότου βρεθεί λύση. Ο αλγόριθμος Επαναληπτικής Εκβάθυνσης ξεκινά εκτελώντας DFS με όριο βάθους $d = 0$. Εάν δεν βρει η λύση, αυξάνει το όριο βάθους κατά 1 και επαναλαμβάνει τη διαδικασία από την αρχή. Αυτό συνεχίζεται έως ότου βρεθεί η λύση. Ακόμη, ο αλγόριθμος Επαναληπτικής Εκβάθυνσης μπορεί να χρησιμοποιηθεί να προκαθορισμένο όριο βάθους, για παράδειγμα $d = 3$. Όταν αυτό συμβεί ο αλγόριθμος θα φτάσει στο 3 + 1 = 4 (το πρώτο επίπεδο αριθμήται ως 0) επίπεδο του δένδρου αναζήτησης και θα τερματίσει εκεί. Με απλά λόγια, ο αλγόριθμος της Επαναληπτικής Εκβάθυνσης αξιοποιεί τα πλεονεκτήματα του DFS, αλλά χωρίς να υπάρχει ο κίνδυνος να παγιδευτεί σε μη βέλτιστες λύσεις, αποτελώντας τον έτσι μία εκδοχή του DFS σε "στεροειδή".

Παράδειγμα 2.7

Έστω ότι έχουμε το παρακάτω δένδρο αναζήτησης, όπου ο στόχος μας είναι να βρούμε τον κόμβο G, που είναι η τελική κατάσταση (λύση):

Ας υποθέσουμε ότι ο στόχος μας είναι να βρούμε τον κόμβο G.



Σχήμα 7: Δένδρο Αναζήτησης για τον αλγόριθμο Iterative Deepening

1. Πρώτη επανάληψη (όριο βάθους $d = 0$):
   - Εξετάζουμε τον κόμβο A.
   - Ο κόμβος A δεν είναι η λύση, αλλά δεν μπορούμε να προχωρήσουμε σε παιδιά λόγω του ορίου βάθους.
   - Αποτέλεσμα: Δεν βρέθηκε λύση.
2. Δεύτερη επανάληψη (όριο βάθους $d = 1$):
   - Εξετάζουμε τον κόμβο A.
   - Προχωράμε στους κόμβους B και C.
   - Ούτε ο B ούτε ο C είναι η λύση.
   - Αποτέλεσμα: Δεν βρέθηκε λύση.
3. Τρίτη επανάληψη (όριο βάθους $d = 2$):
   - Ξεκινάμε από τον A.
   - Προχωρούμε στους B και C.
   - Από τον B, προχωράμε στους κόμβους E και F.
   - Από τον C, προχωράμε στους κόμβους G και H.
   - Ο κόμβος G είναι η λύση.
   - Αποτέλεσμα: Η λύση βρέθηκε.

Το μονοπάτι που ακολουθήθηκε για να φτάσουμε στη λύση είναι το εξής:

$$\text{Μονοπάτι λύσης: } [A, C, G]$$

Ενώ η ακολουθία αναζήτησης είναι η εξης:

$$\text{Ακολουθία αναζήτησης: } [A, B, C, E, F, G]$$

2.3 MiniMax

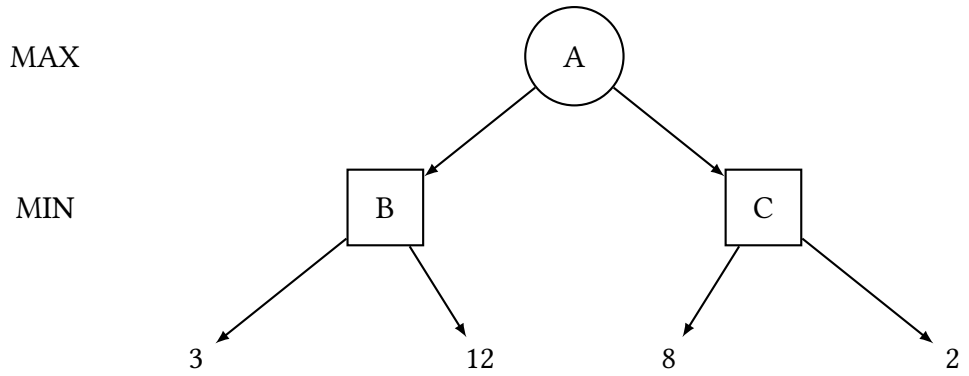
Ο αλγόριθμος **Minimax** χρησιμοποιείται κυρίως στη θεωρία παιγνίων και σε σενάρια λήψης αποφάσεων, όπως σε παιχνίδια δύο παικτών (π.χ., σκάκι, τρίλιζα). Στόχος του είναι να βρει την καλύτερη κίνηση για έναν παίκτη, υποθέτοντας ότι ο αντίπαλος θα παίξει και αυτός με τον βέλτιστο τρόπο. Ο Minimax εξερευνά το δένδρο του παιχνιδιού προσομοιώνοντας κάθε πιθανή κίνηση και αντι-κίνηση, έως ότου το παιχνίδι φτάσει σε μια τερματική κατάσταση. Η διαδικασία εναλλάσσεται μεταξύ του επιπέδου του παίκτη που μεγιστοποιεί (Max) και του παίκτη που ελαχιστοποιεί (Min), και στόχος είναι να βρεθεί η κίνηση που θα οδηγήσει στο καλύτερο αποτέλεσμα για τον παίκτη Max (ή για τον παίχτη Min).

Έτσι λοιπόν, Για κάθε φύλλο του δένδρου, ο παίχτης που έχει σειρά (Max/Min) θα προσπαθήσει να μεγιστοποιήσει ή να ελαχιστοποιήσει (αναλόγως τον παίχτη) την τιμή αξιολόγησης από τα φύλλα του δένδρου. Μετά γίνεται οπισθοδιάδωση των τιμών της συνάρτησης αξιολόγησης στα προηγούμενα επίπεδα, όσπου μία τιμή να καταλήξει στην ρίζα του δένδρου.

Σημαντικό αρνητικό του MiniMax είναι η μεγάλη χρονική του πολυπλοκότητα, αλλά για αυτό θα δούμε μία παραλλαγή του, το A - B κλάδεμα. Το A - B κλάδεμα (ή A - B prooning), παραμένει στην ίδια λογική με τον Minimax, ελαχιστοποιώντας την χρονική πολυπλοκότητα κανοντάς τον μία εκδοχή του Minimax σε "στεροειδή".

Παράδειγμα 2.8

Έστω ότι έχουμε ένα απλοποιημένο δένδρο παιχνιδιού, όπου ο παίκτης Max κάνει την πρώτη κίνηση και ο παίκτης Min ανταποκρίνεται. Στόχος μας σε αυτό το παράδειγμα είναι να βρούμε την βέλτιστη κίνηση για τον Max χρησιμοποιώντας τον αλγόριθμο Minimax.



Σχήμα 8: Απλοποιημένο Δένδρο Παιχνιδιού για το Παράδειγμα του Minimax

Στο παράδειγμα μας, το πρώτο επίπεδο αντπροσωπεύει την "σειρά" (ή πιο επίσημα, το σημείο απόφασης) του Max, ενώ το δεύτερο αντιπροσωπεύει αυτό του Min. Αυτό το δένδρο απόφασης θα μπορούσε να συνεχιστεί, με περισσότερα επίπεδα στα οποία θα εναλλάσονται Max και Min όπως είναι λογικό. Σκοπός μας σε τέτοιου τύπου ασκήσεις είναι, αν βρισκόμαστε σε επίπεδο του Max να επιλέγουμε την κίνηση που μεγιστοποιεί το "άθροισμα" ενώ αν βρίσκομαστε σε επίπεδο του Min να επιλέγουμε την κίνηση που ελαχιστοποιεί το "άθροισμα".

Ξεκινάμε λοιπόν από τους κόμβους-φύλλα και ανεβαίνουμε προς τα επάνω. Οι κόμβοι-φύλλα αντιπροσωπεύουν τις τερματκές καταστάσεις του παιχνιδιού οι οποίες θα είναι προφανώς αποτέλεσμα των επιλογών/κινήσεων των παιχτών. Ξεκινάμε λοιπόν προς τα πάνω συναντόντας πρώτα επίπεδο στο οποίο παίζει ο Min. Έτσι λοιπόν για τους κόμβους (οι οποίοι κόμβοι αντιπροσωπεύουν αποφάσεις) B και C θα επιλέξουμε το μικρότερο κόμβο-παιδί τους. Ως αποτέλεσμα αυτού, πλέον οι κόμβοι B και C θα έχουν τις τιμές 3 και 2 αντιστοίχως. Τέλος, φτάνουμε στο τελικό επίπεδο που σειρά έχει ο Max, ο οποίος επιλέγει το μεγαλύτερο εκ των δύο παιδιών-κόμβων, δηλαδή το 3. Αυτός ο αριθμός υποδηλώνει την δυναμική του Max υπερ του Min, αν ο καθένας του παίξει τίμια.

Πρέπει να σημειωθεί ότι οι φορές και σειρά που θα παίξουν οι παίχτες μπορεί να διαφέρει. Στο συγκεκριμένο παράδειγμα έχουμε 2 κινήσεις παιχνιδιού μια για κάθε έναν παίχτη, με πρώτο να παίζει ο Max και τελευταίο πραφανώς τον Min. Θα μπορούσε να έπαιζε πρώτος ο Min, με σειρά κινήσεων Min-Max και τελικό αποτέλεσμα 8. Επίσης θα μπορούσε να είχαμε 5 επίπεδα και πρώτο τον Max, με σειρά κινήσεων Max-Min-Max-Min-Max. Ακόμη, σε ότι ασκήσεις σου δωθούν, οι κόμβοι (οι κινήσεις) του παιχνιδιού, δεν θα έχουν ονόματα, όπως B, C, A και ούτο κάθε εξής. Αντιθέτως θα είναι κενοί κόμβοι που θα υποδηλώνουν την κίνηση ενός παίχτη όπου εσύ θα πρέπει να βρείς ποια τιμή θα καταλήξει και ποια θα φύγει από αυτόν.

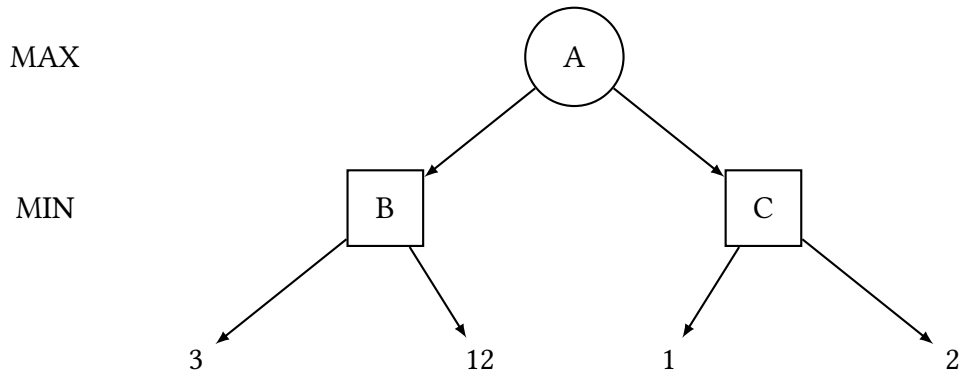
2.4 Α - Β κλάδεμα

Ο αλγόριθμος **Α - Β κλάδεμα** (ή **Α - Β pruning**) είναι μία βελτίωση του Minimax που επιτρέπει τη μείωση της χρονικής πολυπλοκότητας μέσω του κλαδέματος κάποιων κλάδων του δένδρου παιχνιδιού. Όπως και ο Minimax, χρησιμοποιείται κυρίως σε παιχνίδια δύο παικτών και σενάρια λήψης αποφάσεων, αλλά με τη διαφορά ότι αποφεύγει να εξετάζει κλάδους που δεν θα επηρεάσουν την τελική απόφαση.

Η βασική ιδέα του Alpha-Beta κλαδέματος είναι η εισαγωγή δύο παραμέτρων, ***alpha*** και ***beta***, που καθορίζουν το καλύτερο και το χειρότερο αποτέλεσμα που μπορεί να επιτύχει αντίστοιχα ο παίκτης Max και ο παίκτης Min σε κάθε δεδομένο σημείο του δένδρου. Όταν ο Max προσπαθεί να μεγιστοποιήσει το σκορ, και βρίσκει μια κίνηση που είναι ήδη χειρότερη από μια προηγούμενη κίνηση του Min, τότε δεν υπάρχει λόγος να εξερευνήσει περαιτέρω κλάδους αυτής της κίνησης, αφού ο Min δεν θα επιλέξει ποτέ αυτή την πορεία. Αντίστοιχα, όταν ο Min προσπαθεί να ελαχιστοποιήσει το σκορ, και βρίσκει μια κίνηση που είναι ήδη χειρότερη για τον Max από μια προηγούμενη κίνηση, τότε αυτός ο κλάδος κλαδεύεται. Αυτή η διαδικασία μπορεί να εξοικονομήσει σημαντικό χρόνο, καθώς πολλές πιθανές διαδρομές στο δένδρο δεν εξετάζονται καν, αφού είναι φανερό ότι δεν θα επηρεάσουν το αποτέλεσμα.

Παράδειγμα 2.9

Έστω ότι έχουμε ένα απλοποιημένο δένδρο παιχνιδιού, όπου ο Max και ο Min εναλλάσσονται σε κάθε επίπεδο για να επιλέξουν την καλύτερη στρατηγική. Το δένδρο έχει την παρακάτω δομή:



Σχήμα 9: Απλοποιημένο Δένδρο Παιχνιδιού για το Παράδειγμα του Alpha-Beta Κλαδέματος

Στο συγκεκριμένο δένδρο παιχνιδιού, ο Max προσπαθεί να βρει την καλύτερη κίνηση στο επίπεδο του, ενώ ο Min προσπαθεί να ελαχιστοποιήσει το σκορ. Κατά την πορεία της αναζήτησης, αν ο Min εντοπίσει ότι μία επιλογή του Max οδηγεί σε αποτέλεσμα χειρότερο από το ήδη υπάρχον ***beta***, θα σταματήσει να εξετάζει άλλες διαδρομές, κάνοντας κλάδεμα.

Αν υποθέσουμε ότι ο κόμβος Β έχει ήδη αξιολογηθεί με τιμή 3, όταν φτάσουμε στον κόμβο C, βλέπουμε ότι ο πρώτος κόμβο-παιδί του έχει τιμή 1, η οποία είναι μικρότερη από 3. Αυτό σημαίνει ότι ο κόμβος C δεν μπορεί να δώσει μια καλύτερη κίνηση για τον Max, επομένως μπορούμε να κλαδέψουμε τον κόμβο C χωρίς να εξετάσουμε τις υπόλοιπες επιλογές του (δηλαδή, τον δεύτερο γιο του με τιμή 2). Έτσι, εξοικονομούμε χρόνο χωρίς να αλλάξει το τελικό αποτέλεσμα.

3 Μηχανική Μάθηση

Στην Τεχνητή Νοημοσύνη, το **μοντέλο** είναι ένα σύνολο από μαθηματικούς κανόνες ή αλγόριθμους που χρησιμοποιούμε για να κάνουμε προβλέψεις ή να πάρουμε αποφάσεις. Μπορούμε να το δούμε σαν ένα "εργαλείο" που προσπαθούμε να εκπαιδεύσουμε για να λύνει συγκεκριμένα προβλήματα. Για να εκπαιδεύσουμε ένα μοντέλο, χρησιμοποιούμε ένα **δείγμα**, δηλαδή ένα υποσύνολο δεδομένων που παρέχει τις πληροφορίες που χρειάζεται το μοντέλο για να "μάθει".

Υπάρχουν δύο κύριοι τρόποι εκπαίδευσης ενός μοντέλου. Στην **επιβλεπόμενη μάθηση (supervised learning)**, το μοντέλο μαθαίνει να συνδέει δεδομένα εισόδου με τις σωστές απαντήσεις, αφού έχουμε ήδη τις απαντήσεις αυτές στα δεδομένα μας. Αντίθετα, στη **μη επιβλεπόμενη μάθηση (unsupervised learning)**, το μοντέλο προσπαθεί να βρει μόνο του μοτίβα ή σχέσεις στα δεδομένα χωρίς να γνωρίζει την απάντηση εκ των προτέρων. Τέλος το **reinforcement learning** είναι μια μέθοδος εκπαίδευσης όπου το μοντέλο μαθαίνει μέσω δοκιμής και λάθους, λαμβάνοντας ανταμοιβές ή ποινές για τις ενέργειές του, με στόχο να αναπτύξει στρατηγικές που μεγιστοποιούν την επιτυχία του μακροπρόθεσμα.

Για να δούμε πόσο καλά λειτουργεί το μοντέλο, διαχωρίζουμε το σύνολο των δεδομένων σε δύο μέρη: το **εκπαιδευτικό σύνολο (training split)**, που χρησιμοποιούμε για να "εκπαιδεύσουμε" το μοντέλο, και το **δοκιμαστικό σύνολο (testing split)**, που χρησιμοποιούμε για να ελέγξουμε την απόδοσή του. Αυτός ο διαχωρισμός βοηθά να βεβαιωθούμε ότι το μοντέλο μαθαίνει πραγματικά και δεν απλά "θυμάται" τα δεδομένα.

3.1 K-NN

Ο αλγόριθμος **K-Nearest Neighbors(K-NN, ή k-πλησιέστρερου γείτονα)** είναι ένας πολύ απλός αλγόρθμος μηχανικής μάθησης ο οποίος μπορεί να περιγραφτεί με τα εξής βήματα:

1. Αναπαράστησε όλα τα training δεδομένα σε ένα n διαστάσεων σύστημα αξόνων. Όπου n είναι ο αριθμός των χαρακτηριστικών (attributes) κάθε δείγματος.

2. Αναπαράστησε όλα τα testing δεδομένα στο ίδιο σύστημα αξόνων

3. Υπολόγισε την απόσταση αυτών των training δεδομένων από τα testing δεδομένα. Η απόσταση υπολογίζεται χρησιμοποιώντας τον τύπο της Ευκλείδιας ή της Manhattan απόστασης

4. Κατηγοριοποίησε τα testing δεδομένα στην κλάση των K πλησιέστερων training δεδομένων.

Όσο αναφορά την παράμετρο K, αυτή θα δίνεται πάντα από την εκφώνηση. Αν τρέχουμε τον παραπάνω αλγόριθμο εξετάζοντας μόνο τους 3 κοντινότερους γείτονες, τότε τρέχουμε έναν 3-NN. Αν τον τρέχουμε για 4 τότε 4-NN, για 5 5-NN και πάει λέγοντας. Επίσης, θα μπορούσε

Πίνακας 10: Dateset για τον K-NN

#	Κολυμπάει	Θηλαστικό	Πνεύμονες	Ψάρι
1	ΝΑΙ	ΟΧΙ	ΟΧΙ	ΝΑΙ
2	ΟΧΙ	ΟΧΙ	ΝΑΙ	ΟΧΙ
3	ΝΑΙ	ΝΑΙ	ΝΑΙ	ΝΑΙ
4	ΝΑΙ	ΝΑΙ	ΟΧΙ	ΝΑΙ
5	ΝΑΙ	ΟΧΙ	ΝΑΙ	;

κάποιος να ρωτήσει, ποιά είναι η καλύτερη τιμή για αυτή τη παράμετρο Ν. Δεν υπάρχει καποια καλύτερη τιμή για αυτή τη παράμτερο. Το μόνο σίγουρο είναι ότι η τιμή πρέπει να είναι περιττός αριθμός, για να αποφευχθούν τυχόν ισοβαθμίες μταξύ των Κ κοντινότερων κλάσεων. Τέλος να σημειωθεί κάτι το θεωρητικό, ο αλγόριθμος K-NN δεν παράγει κάποιο μοντέλο, αλλά ο ίδιος αποτελεί lazy classifier.

Παραπάνω αναφέρθηκαν οι τύποι της Ευκλείδιας και Manhattan απόστασης. Αυτοί λοιπόν είναι:

- Ευκλείδια$(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- Manhattan$(A, B) = |x_2 - x_1| + |y_2 - y_1|$

Γενικά, προτείνεται να χρησιμοποιείται ο τύπος της Manhattan, καθώς είναι απλούστερος για στον υπολογισμό μιάς και δεν χρησιμοποιεί ρίζες.

Παράδειγμα 3.1

Έστω τα δείγματα στον πίνακα 10. Τα δείγματα 1 έως 4 αποτελούν δείγματα των οποίων γνωρίζουμε την κλάση, ενώ το 5ο αποτελεί νεό δείγμα προς κατηγοριοποίηση. Θέλουμε να το κατηγοριοποιήσουμε χρησιμοποιώντας τον 1-NN με τον τύπο της απόστασης Manhattan. Έτσι λοιπόν έχουμε:

- Man(5,1) = |ΝΑΙ - ΝΑΙ| + |ΟΧΙ - ΟΧΙ| + |ΝΑΙ - ΟΧΙ| = 0 + 0 +1 = 1
- Man(5,2) = |ΝΑΙ - ΟΧΙ| + |ΟΧΙ - ΟΧΙ| + |ΝΑΙ - ΝΑΙ| = 1 + 0 + 0 = 1
- Man(5,3) = |ΝΑΙ - ΝΑΙ| + |ΟΧΙ - ΝΑΙ| + |ΝΑΙ - ΝΑΙ| = 0 + 1 + 0 = 1
- Man(5,4) = |ΝΑΙ - ΝΑΙ| + |ΟΧΙ - ΝΑΙ| + |ΝΑΙ - ΟΧΙ| = 0 + 1 + 1 = 2

Έχοντας τον 1-NN, ψάχνουμε τον ένα πλησιέστερο γείτονα του 5ου δείγματο. Παρατηρούμε ότι τα δείγματα 1,2 και 3 είναι το ίδιο κοντά με το δείγμα που θέλουμε να κατηγοριοποιήσουμε, με απόσταση 1. Το 1ο και το 3ο δείγμα έχουν κλάση ΝΑΙ, ενώ το 2ο κλάση ΟΧΙ. Συνεπώς μίας και η κλάση ΝΑΙ κερδίζει λόγω πλειοψηφίας, το νεό 5ο δείγμα κατηγοριοποιήται με τον 1-NN ως Ψάρι = ΝΑΙ. Το ίδιο αποτέλεσμα θα έχουμε αν χρησιμοποιήσουμε 3-NN.

3.2 Decision Trees - id3

Ένα **δέντρο απόφασης (decision tree)** αποτελείται από κόμβους οι οποίοι αντιπροσωπεύουν χαρακτηριστικά και ακμές που αντιπροσωπεύουν τις δυνατές τιμές που μπορούν να λάβουν αυτά τα χαρακτηριστικά (σε προγραμματιστικό λεξιλόγιο αυτές μπορούμε να τις πούμε και συνθήκες). Τα φύλλα του δένδρου αντιστοιχούν στις κατηγορίες που θα κατηγοριοποιήσουν το νέο δείγμα.

Πίνακας 11: Dataset για τον ID3

#	ΧΡΩΜΑΤΑ	ΑΡΙΘ. ΜΑΤΙΩΝ	ΕΞΟΔΟΣ(ΖΩΝΤΑΝΟΣ)
1	ΜΑΥΡΟ	6-15	ΟΧΙ
2	ΜΑΥΡΟ	>15	ΟΧΙ
3	ΚΟΚΚΚΙΝΟ	6-15	ΝΑΙ
4	ΠΡΑΣΙΝΟ	0-5	ΝΑΙ
5	ΜΑΥΡΟ	6-15	ΟΧΙ
6	ΚΟΚΚΙΝΟ	6-15	;

Το πως κατασκευάζουμε ένα δένδρο απόφασης για εμάς θα είναι πολύ απλό. Απλά, βρίσκουμε ένα μοτίβο που ακολουθεί το dataset και σχεδιάζουμε κατάλληλους κόμβους και συνθήκες στις ακμές που να αντιπροσοπεύουν αυτό το μοτίβο. Το μοτίβο αυτό μπορούμε να το διακρίνουμε από τις τιμές των χαρακτηριστικών που οδηγούν σε κάποια συγκεκριμένη κλάση. Στην πράξη σε πιο περίπλοκα μοντέλα χρησιμοποιούμε εντροπία για να ξεχωρίσουμε χαρακτηριστικά εκείνα που δημιουργούν το μοτίβο, αλλά για τους σκοπούς του μαθήματος δεν θα χρειαστεί κάτι τέτοιο.

Ένα παράδειγμα μοτίβου είναι η τιμή ενός προϊόντος στην αγορά του, ή το φύλο ενός ατόμου στην εκδήλωση αυτοάνοσου. Αυτά τα χαρακτηριστικά θα μπούν ψηλά στο δένδρο απόφασης, αφού γνωρίζοντας την τιμή τους μπορούμε να είμαστε αρκετά σιγούροι για την τελική κλάση του δείγματος. Αν η τιμή ενός προϊόντος είναι πάνω από 100ευρώ περιμένουμε να μην το αγοράσουν πολλά άτομα κατηγοριοποιόντας το προϊόν ως ΟΧΙ ΑΓΟΡΑ, ενώ αν το φύλο ενός ατόμου είναι θηλύ ξέρουμε ότι είναι 9 φορές πιο πιθανό να εκδηλώσει αυτοάνοσο από ενά άτομο φύλο αρρέν.

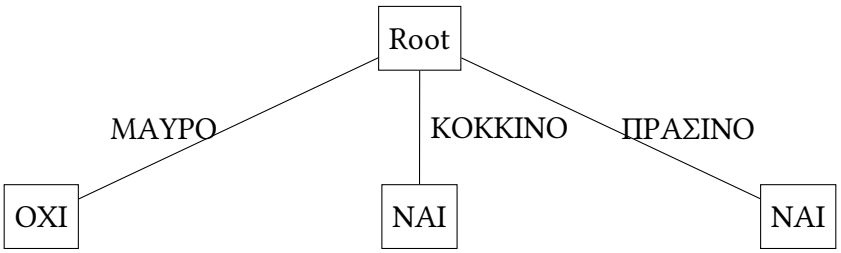
Προφανώς μπορούμε να γίνουμε πιο λεπτομερείς με την κατηγοριοποίηση και να πούμε ότι τελικά πιο άλλα σημαντικά χαρακτηριστικά είναι το είδος του προϊόντος, η αγοραστική δύναμη των καταναλωτών, η γεωγραφική τοποθεσία και η ηλικία των ασθενών. Όμως, στα πλαίσια του μαθήματος δεν θα σου χρειαστεί να κατασκευάσεις μοντέλο με πόσα περίπλοκα χαρακτηριστικά, όποτε θα σου είναι αρκετά εύκολο να διακρίνεις τα απολύτως σημαντικά.

Παράδειγμα 3.2

Έστω το σύνολο δεδομένων στον πίνακα 11, ρεαλιστικό ή όχι δεν έχει σημασία. Για να κατασκευάσουμε το δένδο απόφασης με τον id3, θυμίζω ότι πρέπει να βρούμε εκείνα τα χαρακτηριστικά που η τιμή τους ευθύνεται για την κατηγοριοποίηση του δείγματος σε κάποια κλάση. Αυτό το χαρακτηριστικό που η τιμή του ευθύνεται πιο πολύ απο τα άλλα (χαρακτηριστικά) για την κατηγοριοποίηση του δείγματος θα μπεί ως ρίζα του δένδρου απόφασης, ενώ τα υπόλοιπα θα ακολουθήσουν ως κόμβοι.

Έστω ότι θέλουμε να εφαρμόσουμε τον αλγόριθμο δένδρου απόφασης id3 από τα δεδομένα του πίνακα 11, ποιό χαρακτηριστικό από τα δύο θα επιλέξει ο αλγόριθμος να βάλει ρίζε στο δένδρο και γιατί (στον πίνακα βάλαμε 6-15 γιατί το χαρακτηριστικό αφορά εύρος, range);

Στην ρίζα λοιπόν θα μπει το χαρακτηριστικό 'ΧΡΩΜΑ' γιατί από αυτό εξαρτάται η τιμή της εξόδου για κάθε δείγμα (σχήμα 10). Για ένα νεό με 'ΧΡΩΜΑΤΑ'=ΚΟΚΚΙΝΟ, 'ΑΡΙΘ.ΜΑΤΙΩΝ'=7 για τις τιμές των χαρακτηριστικών, το παραγώμενο δένδο απόφασης θα κατηγριοποιούσε αυτό το δείγμα ως 'ΖΩΝΤΑΝΟΣ'=ΝΑΙ.



Σχήμα 10: Δένδρο απόφασης αλγόριθμου id3