



## Θεμελιώδη Θέματα Επιστήμης Υπολογιστών

### Τεχνητή Νοημοσύνη - Ευρετική επίλυση προβλημάτων



## Τεχνητή Νοημοσύνη



### GENERATIVE ARTIFICIAL INTELLIGENCE

The GRAMMY Award recognizes creative excellence. **Only human creators are eligible to be submitted for consideration for, nominated for, or win a GRAMMY Award.** A work that contains no human authorship is not eligible in any Categories. A work that features elements of A.I. material (i.e., material generated by the use of artificial intelligence technology) is eligible in applicable Categories; however: (1) the human authorship component of the work submitted must be meaningful and more than de minimis; (2) such human authorship component must be relevant to the Category in which such work is entered (e.g., if the work is submitted in a songwriting Category, there must be meaningful and more than de minimis human authorship in respect of the music and/or lyrics; if the work is submitted in a performance Category, there must be meaningful and more than de minimis human authorship in respect of the performance); and (3) the author(s) of any A.I. material incorporated into the work are not eligible to be nominees or GRAMMY recipients insofar as their contribution to the portion of the work that consists of such A.I. material is concerned. De minimis is defined as lacking significance or importance; so minor as to merit disregard.



## Τεχνητή Νοημοσύνη

Τεχνητή Νοημοσύνη είναι ο κλάδος της Επιστήμης Υπολογιστών που ασχολείται με την **αυτοματοποίηση** της επίλυσης *δύσκολων προβλημάτων* που ο άνθρωπος επιλύει καλύτερα από ότι οι υπολογιστές



## Τεχνητή Νοημοσύνη

### Πρόβλημα

- ▶ Συναντά ο Α τον Β στο δρόμο, παλιοί φίλοι, έχουν να συναντηθούν πολλά χρόνια. Αφού λένε τα νέα τους, λέει ο Β στον Α είμαι καθηγητής μαθηματικών, γενικά λύνω προβλήματα. Του λέει ο Α, ωραία θα σου βάλω ένα πρόβλημα!
- ▶ Λέει ο Α, σήμερα έχουν γενέθλια οι τρεις κόρες μου. Ξέρεις ποιες είναι οι ηλικίες τους;
- ▶ Λέει ο Β, χρειάζομαι κάποια δεδομένα ακόμα.
- ▶ Λέει ο Α, το γινόμενο των ηλικιών τους είναι 36.
- ▶ Λέει ο Β, χρειάζομαι κάποια δεδομένα ακόμα.
- ▶ Λέει ο Α, το άθροισμα των ηλικιών τους είναι όσα τα παράθυρα σε αυτό το κτίριο. (Ο Β βλέπει το κτίριο και μετράει τα παράθυρα)
- ▶ Λέει ο Β, χρειάζομαι κάποια δεδομένα ακόμα.
- ▶ Λέει ο Α, η μεγάλη μου κόρη έχει γαλάζια μάτια.
- ▶ Λέει ο Β, ξέρω τις ηλικίες τους.

Ποιες είναι οι ηλικίες τους;

## Τεχνητή Νοημοσύνη



- ▶ Σήμερα έχουν γενέθλια οι τρεις κόρες μου.

1	1	1
1	1	2
1	1	3
.	.	.

- ▶ Το γινόμενο των ηλικιών τους είναι 36.

36	1	1
18	2	1
12	3	1
9	4	1
9	2	2
6	6	1
6	3	2
4	3	3

## Τεχνητή Νοημοσύνη



- ▶ Το άθροισμα των ηλικιών τους είναι όσα τα παράθυρα σε αυτό το κτίριο.

36	1	1	38
18	2	1	21
12	3	1	16
9	4	1	14
9	2	2	13
6	6	1	13
6	3	2	11
4	3	3	10

- ▶ Λέει ο Β, χρειάζομαι κάποια δεδομένα ακόμα.

9	2	2	13
6	6	1	13

- ▶ Η μεγάλη μου κόρη έχει γαλάζια μάτια.

9 2 2

## Τεχνητή Νοημοσύνη



### Πρόβλημα

- ▶ Συναντά ο Α τον Β στο δρόμο, παλιοί φίλοι, έχουν να συναντηθούν πολλά χρόνια. Αφού λένε τα νέα τους, λέει ο Β στον Α είμαι καθηγητής μαθηματικών, γενικά λύνω προβλήματα. Του λέει ο Α, ωραία θα σου βάλω ένα πρόβλημα!
- ▶ Λέει ο Α, σήμερα έχουν γενέθλια οι τρεις κόρες μου. Ξέρεις ποιες είναι οι ηλικίες τους;
- ▶ Λέει ο Β, χρειάζομαι κάποια δεδομένα ακόμα.
- ▶ Λέει ο Α, το γινόμενο των ηλικιών τους είναι 36.
- ▶ Λέει ο Β, χρειάζομαι κάποια δεδομένα ακόμα.
- ▶ Λέει ο Α, το άθροισμα των ηλικιών τους είναι όσα τα παράθυρα σε αυτό το κτίριο. (Ο Β βλέπει το κτίριο και μετράει τα παράθυρα)
- ▶ Λέει ο Β, χρειάζομαι κάποια δεδομένα ακόμα.
- ▶ Λέει ο Α, η μεγάλη μου κόρη έχει γαλάζια μάτια.
- ▶ Λέει ο Β, ξέρω τις ηλικίες τους.

Ποιες είναι οι ηλικίες τους;

## Τεχνητή Νοημοσύνη



A meets B on the street. They are old friends who haven't seen each other in many years. After catching up on their news, B tells A, "I'm a math professor; I generally solve problems." A responds, "Great, I'll give you a problem!"

A says, "Today is my three daughters' birthday. Can you figure out their ages?"

B replies, "I need more information."

A says, "The product of their ages is 36."

B replies, "I need more information."

A says, "The sum of their ages is the same as the number of windows in that building." (B looks at the building and counts the windows.)

B replies, "I need more information."

A says, "My eldest daughter has blue eyes."

B responds, "Now I know their ages."

## Τεχνητή Νοημοσύνη



This is a classic logic puzzle involving reasoning through constraints. Let's solve it step by step:

### Step 1: Analyze the first clue

- The product of the three daughters' ages is 36.
- List all possible sets of three positive integers whose product is 36:

Ages (sorted)	Product	Sum
1, 1, 36	36	38
1, 2, 18	36	21
1, 3, 12	36	16
1, 4, 9	36	14
1, 6, 6	36	13
2, 2, 9	36	13
2, 3, 6	36	11
3, 3, 4	36	10

## Τεχνητή Νοημοσύνη



### Step 2: Analyze the second clue

- The sum of the ages equals the number of windows in the building.
- If the number of windows were unique among the sums above, B would immediately know the answer.
- However, 13 appears twice (for the sets 1, 6, 6 and 2, 2, 9), so B needs more information.

### Step 3: Analyze the third clue

- A says, "My eldest daughter has blue eyes."
- This implies there is a **distinct eldest daughter**, ruling out the set 1, 6, 6 because there is no single "eldest" (the two eldest are twins).
- The only remaining set is 2, 2, 9, where there is a clear eldest daughter (aged 9).

### Conclusion

The ages of the daughters are 2, 2, and 9.

Michalewicz, Z., & Fogel, D. B. (2004). *How to Solve It: Modern Heuristics* (2nd ed.). Springer.

🔍 📄 🗑️ 🔄 📌

## Τεχνητή Νοημοσύνη



Iokasti has a son who is patricide. Oidipous is a patricide. Oidipous is Iokasti's son. Polyneikis is Iokasti's son. Polyneikis is Oidipous's son. Thersandros is Polyneikis's son. Thersandros is not a patricide. Is there anyone that has a child that is a patricide and has a child that is not a patricide?

Yes, there is such a person. Polyneikis fits this description. Polyneikis's father, Oidipous, is a patricide, as he killed his father, King Laius. However, Polyneikis himself is not a patricide. Additionally, Polyneikis has a son named Thersandros, who is also not a patricide. Therefore, Polyneikis is the person who has a child that is a patricide (Oidipous) and has a child that is not a patricide (Thersandros).

We are not sure that Polyneikis is not a patricide.

## Τεχνητή Νοημοσύνη



Απρίλιος 2024

You are correct; my apologies for the oversight. Let's reevaluate the situation:

- Iokasti has a son who is a patricide.
- Oidipous is a patricide.
- Oidipous is Iokasti's son.
- Polyneikis is Iokasti's son.
- Polyneikis is Oidipous's son.
- Thersandros is Polyneikis's son.
- Thersandros is not a patricide.

Given this information, it is unclear whether Polyneikis is a patricide. However, based on the information provided, we know that Thersandros, Polyneikis's son, is not a patricide.

So to answer the question: No, based on the provided information, there is no explicit confirmation of a person who has a child that is a patricide and a child that is not a patricide.

## Τεχνητή Νοημοσύνη



Νοέμβριος 2024

lokasti has a son who is patricide. Oidipous is a patricide. Oidipous is lokasti's son. Polyneikis is lokasti's son. Polyneikis is Oidipous's son. Thersandros is Polyneikis's son. Thersandros is not a patricide. Is there anyone that has a child that is a patricide and has a child that is not a patricide?

To determine whether someone has a child that is a patricide and a child that is not a patricide, let's analyze the relationships step by step:

### Given Relationships:

- lokasti has a son who is a patricide.
- Oidipous is a patricide.
- Oidipous is lokasti's son.
- Polyneikis is lokasti's son.
- Polyneikis is Oidipous's son.
- Thersandros is Polyneikis's son.
- Thersandros is not a patricide.

### Observations:

- From (1), (2), and (3), lokasti has a child (Oidipous) who is a patricide.
- From (4), lokasti has another child, Polyneikis. The information does not specify whether Polyneikis is a patricide.
- From (6) and (7), Polyneikis has a child (Thersandros) who is not a patricide.

## Τεχνητή Νοημοσύνη



Νοέμβριος 2024

lokasti has a son who is patricide. Oidipous is a patricide. Oidipous is lokasti's son. Polyneikis is lokasti's son. Polyneikis is Oidipous's son. Thersandros is Polyneikis's son. Thersandros is not a patricide. Is there anyone that has a child that is a patricide and has a child that is not a patricide?

### Conclusion:

The information provided is insufficient to definitively determine if lokasti has both a patricide and a non-patricide child. However, if we assume that Polyneikis is not a patricide (which would align with the fact that his son, Thersandros, is not one), then lokasti would meet the condition of having both a patricide (Oidipous) and a non-patricide (Polyneikis) child.

### Key Question:

Does lokasti have both:

- A child that is a patricide (Oidipous), and
- A child that is not a patricide?

From the information provided:

- lokasti's child Polyneikis is not explicitly stated to be a patricide or not.
- We know that Thersandros, Polyneikis's son, is not a patricide, but that doesn't tell us about Polyneikis's status as a patricide.

## Τεχνητή Νοημοσύνη



### Πρόβλημα

SAT problem

Δίνεται μία λογική συνάρτηση, πχ. η  $F(\vec{x})$  με:

$$F(\vec{x}) = (x_{17} \vee \neg x_{37} \vee x_{73}) \wedge (\neg x_{11} \vee x_{56}) \wedge \dots \wedge (x_2 \vee x_{43} \vee \neg x_{77} \vee \neg x_{97})$$

Υπάρχει αντιστοίχιση των  $x_i$  ( $i = 1, \dots, 100$ ) σε TRUE ή FALSE

τέτοια ώστε  $F(\vec{x}) = \text{TRUE}$ ?

### Πιθανές λύσεις

$$2^{100} \approx 10^{30}$$

Αν εξετάζουμε 100.000 πιθανές λύσεις/sec χρειαζόμαστε > 1.5 δισεκατομμύρια χρόνια για να εξετάσουμε όλες τις πιθανές λύσεις

Έχουμε ιδέες για να περιορίσουμε-οδηγήσουμε την αναζήτηση στο SAT;

## Τεχνητή Νοημοσύνη



### Πρόβλημα

Φτάνεις σε ένα εστιατόριο για φαγητό.

Αποφάσισε αν θα περιμένεις για τραπέζι ή θα φύγεις.

### Παράγοντες που επηρεάζουν την απόφαση

Εναλλακτικές: υπάρχει κάποιο εστιατόριο κοντά;

Ποιότητα αναμονής: υπάρχει άνετος χώρος αναμονής (π.χ. μπαρ);

Ημέρα: Είναι σήμερα ΣΚ ή κάποια γιορτή;

Πείνα: Είσαι πολύ πεινασμένος;

Πληρότητα: Πόσοι τρώνε στο εστιατόριο (κανείς, αρκετοί, είναι γεμάτο);

Τιμή: πόσο ακριβό είναι το εστιατόριο (φτηνό, μέσο, ακριβό);

Καιρικές συνθήκες: Βρέχει έξω;

Κράτηση: Έχεις κάνει κράτηση;

Είδος εστιατορίου: Ινδικό, πιτσαρία, γκουρμέ;

Εκτίμηση αναμονής: εκτίμηση σε λεπτά (0-10, 10-30, 30-60, >60);



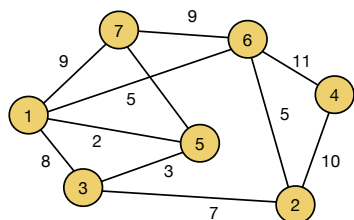
## Τεχνητή Νοημοσύνη



### Πρόβλημα (TSP)

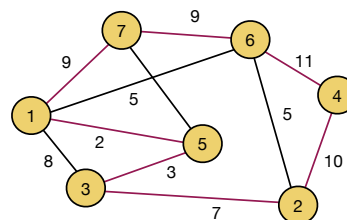
Δίνεται ένας γράφος που οι κόμβοι του συμβολίζουν πόλεις, οι ακμές του δρόμους και το βάρος της ακμής το μήκος της διαδρομής. Ένας πλανόδιος πωλητής επιθυμεί να επισκεφτεί όλες τις πόλεις και πρέπει να βρούμε για αυτόν την μικρότερη δυνατή διαδρομή.

### Πρόβλημα 7 πόλεων



1-2-3-4-5-6-7  
 1-3-2-4-5-6-7  
 1-3-4-2-5-6-7  
 ⋮ ⋮ ⋮

## Τεχνητή Νοημοσύνη



1-7-6-4-2-3-5  
 7-6-4-2-3-5-1  
 5-3-2-4-6-7-1  
 ⋮ ⋮ ⋮

Κάθε διαδρομή μπορεί να αναπαρασταθεί με  $2n$  διαφορετικούς τρόπους

Ο χώρος αναζήτησης είναι  $\frac{n!}{2n} = \frac{(n-1)!}{2}$

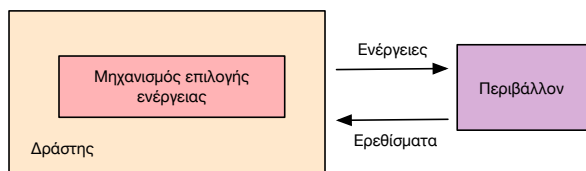
Για μεγάλες τιμές του  $n$  είναι πολύ μεγαλύτερος αριθμός από το  $2^n$

$n = 6$  : |TSP| = 60, |SAT| = 64       $n = 7$  : |TSP| = 360, |SAT| = 128

$n = 50$  : |TSP|  $\approx 10^{61}$ , |SAT|  $\approx 10^{15}$

Παρότι στην περίπτωση αυτή έχουμε ξανά απίστευτα μεγάλο χώρο, έχουμε ιδέες για να περιορίσουμε-οδηγήσουμε την αναζήτηση στο TSP;

## Ορθολογικός δράστης



Περιβάλλον, Αισθητήρες, Δράσεις, Επίδοση

### Αφαίρεση

Καθορισμός σημαντικών στοιχείων



Τυπικός ορισμός προβλήματος  
(αναπαράσταση)

Πρόβλημα → Μοντέλο → Λύση

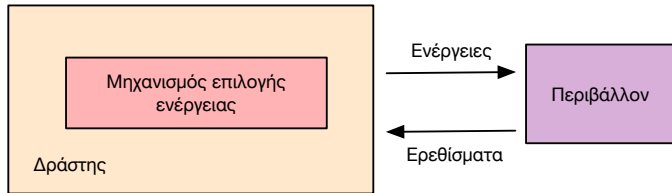
## Ορθολογικός δράστης



### Σύστημα υποστήριξης ιατρικής διάγνωσης

- ▶ Περιβάλλον: Ασθενής, ιατροί, πληροφοριακό σύστημα νοσοκομείου,...
- ▶ Αισθητήρες: Δεδομένα που αφορούν τον ασθενή, βάσεις ιατρικών δεδομένων,...
- ▶ Δράσεις: Διάλογοι με ιατρικό προσωπικό, αυτόματη ενημέρωση βάσεων δεδομένων,...
- ▶ Δείκτες Επίδοσης: Ανάρρωση ασθενούς, εξαγωγή ιατρικής γνώσης,...

## Δράστες βασισμένοι σε πίνακα



### Μεθοδολογία

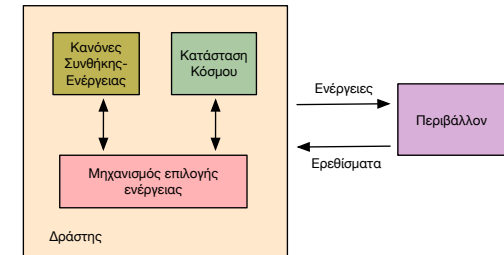
- Χρήση πινάκων για αντιστοίχιση ερεθισμάτων που λαμβάνει ο δράστης από το περιβάλλον με την κατάσταση του δράστη με τις ενέργειες που πρέπει να εκτελεστούν

## Απλοί ανακλαστικοί δράστες



### Μεθοδολογία

- Αναγνώριση συχνών και επαναλαμβανόμενων σχέσεων μεταξύ ερεθισμάτων και ενεργειών
- Μοντελοποίηση των σχέσεων με τη μορφή κανόνων συνθήκης-ενέργειας (condition-action rules)
- Επιλογή του κατάλληλου κανόνα, και στη συνέχεια εφαρμογή της αντίστοιχης ενέργειας



```

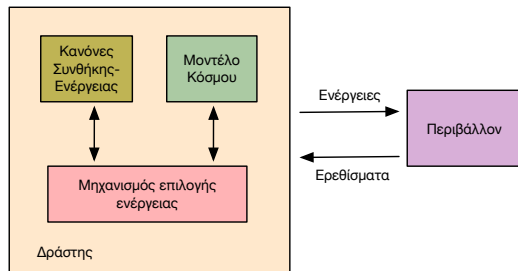
Function Simple-Reflex-Agent(percept)
  static: rules, /*condition-action rules*/
  state ← Intercept_input(percept)
  rule ← Rule_match(state, rules)
  action ← Rule_Action(rule)
  return (action)
    
```

## Δράστες με μοντέλο



### Παρατήρηση

Υπάρχουν μηχανισμοί διαφορετικών καταστάσεων του περιβάλλοντος που μπορεί να παράγουν τα ίδια ερεθίσματα



```

Function Reflex-Agent-With-State(percept)
  static: state, /*description of the current state*/
           rules // set of condition-action rules //
  state ← Update_State(state, percept)
  rule ← Rule_Match(state, rules)
  action ← Rule_Action(rule)
  state ← Update_State(state, action)
  return (action)
    
```

### Μεθοδολογία

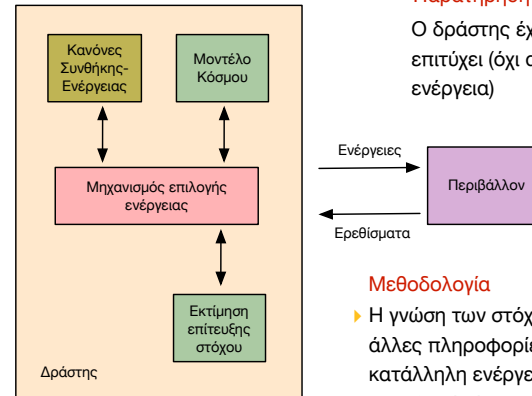
- Διατήρηση ενός μοντέλου εσωτερικής κατάστασης που επιτρέπει την διαφοροποίηση των περιπτώσεων και την επιλογή των δράσεων

## Δράστες επίδιξης στόχου



### Παρατήρηση

Ο δράστης έχει στόχους που πρέπει να επιτύχει (όχι απλά να εκτελέσει μια ενέργεια)



### Μεθοδολογία

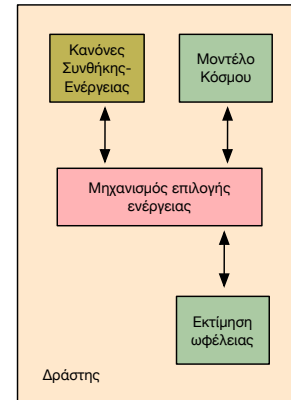
- Η γνώση των στόχων συνδυάζεται με όλες τις άλλες πληροφορίες ώστε να επιλεγεί η κατάλληλη ενέργεια
- Η επιλογή είναι αποτέλεσμα μιας διαδικασίας έρευνας (search) και αλγόριθμων σχεδιασμού ενεργειών (planning)

## Δράστης επιδίωξης στόχου



```
Function Goal-Based-Agent(percept)
  static: state, /* description of the current world state */
           rules /* set of condition-action rules */
           goal /* set of specific success states */
  state ← Update_State(state, percept)
  rule ← Rule_Match(state, rules)
  action ← Rule_Action(rule)
  state ← Update_State(state, action)
  if (state in goal) then
    return (action)
  else
    percept ← Obtain_Percept(state, goal)
    return (Goal-Based-Agent(percept))
```

## Δράστης ωφέλειας



### Παρατήρηση

Στους δράστες επίτευξης στόχου οι καταστάσεις χαρακτηρίζονται ως επιτυχείς (τελική επίτευξη στόχου) ή ανεπιτυχείς (ο στόχος δεν έχει ακόμη επιτευχθεί)

### Μεθοδολογία

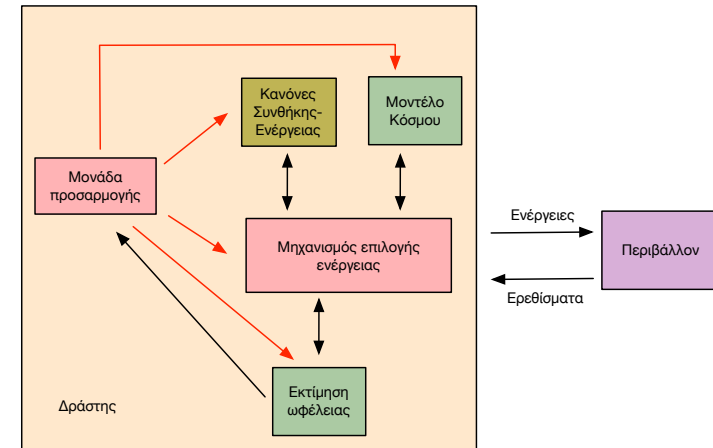
- ▶ Ορίζουμε μια μέθοδο για να διακρίνουμε πόσο είναι το κέρδος ή το σκορ κάθε κατάστασης σε σχέση με το στόχο
- ▶ Χρησιμοποιούμε μία συνάρτηση που ονομάζουμε συνάρτηση ωφέλειας (utility function)
- ▶ Αξιολογούμε με αυτές τις εναλλακτικές ενέργειες

## Δράστης ωφέλειας



```
Function Goal-Based-Agent(percept)
  static: state, /* description of the current world state */
           rules /* set of condition-action rules */
           goal /* set of specific success states */
  state ← Update_State(state, percept)
  rule ← Rule_Match(state, rules)
  action ← Rule_Action(rule)
  state ← Update_State(state, action)
  score ← Obtain_Score(state)
  if (state in goal) and Best_Score(score) then
    return (action)
  else
    percept ← Obtain_Percept(state, goal)
    return (Goal-Based-Agent(percept))
```

## Δράστης με Μάθηση



## Τυπική διατύπωση προβλήματος



### Κόσμος προβλήματος

- ▶ Ένα σύνολο από αντικείμενα, οι ιδιότητές τους και οι σχέσεις που τα συνδέουν  
(αποτελεί υποσύνολο του πραγματικού κόσμου, έχει γίνει αφαίρεση των στοιχείων και των λεπτομερειών που δεν εμπλέκονται στην επίλυση του προβλήματος)

### Κατάσταση του κόσμου

- ▶ Είναι μία *επαρκής, τυπική* αναπαράσταση του κόσμου σε μία δεδομένη χρονική στιγμή

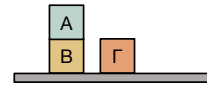
### Πρόβλημα και επίλυση

- ▶ Δεδομένης μίας αρχικής κατάστασης και μίας επιθυμητής τελικής κατάστασης, ζητείται μία ακολουθία από ενέργειες που πρέπει να γίνουν ώστε να φτάσουμε από την αρχική στην τελική κατάσταση

## Τυπική διατύπωση προβλήματος



Κόσμος του προβλήματος: Τρεις κύβοι και ένα τραπέζι



Αντικείμενα	Ιδιότητες	Σχέσεις
Κύβος Α	Κύβος Α ελεύθερος	Κύβος Α πάνω στον κύβο Β
Κύβος Β	Κύβος Γ ελεύθερος	Κύβος Β πάνω στο Τ
Κύβος Γ	Τ έχει αρκετό ελεύθερο χώρο	Κύβος Γ πάνω στο Τ
Τραπέζι Τ	Κύβος Β δεν είναι ελεύθερος	

Κατάσταση:

Κύβος Α πάνω στον κύβο Β
Κύβος Β πάνω στο Τ
Κύβος Γ πάνω στο Τ
Κύβος Α ελεύθερος
Κύβος Γ ελεύθερος

## Τυπική διατύπωση προβλήματος



**Τελεστής μετάβασης (transition operator) ή ενέργεια (action)** είναι μια αντιστοίχιση μίας κατάστασης του κόσμου σε μία άλλη

- ▶ Στον κόσμο των κύβων, τελεστές μετάβασης είναι πχ.:
  - ▶ Βάλε τον κύβο Α πάνω στον κύβο Γ
  - ▶ Βάλε τον κύβο Α πάνω στον κύβο Β

Σε κάθε πρόβλημα ξεκινούμε από μία αρχική κατάσταση  $I$  και (πιθανά) καταλήγουμε σε μία τελική κατάσταση (στόχο)  $G$

Αρχική κατάσταση και στόχος στο πρόβλημα με τους κύβους



**Λύση (solution)** σε ένα πρόβλημα, είναι μία ακολουθία  $t_1, t_2, \dots, t_n$  από τελεστές μετάβασης, τέτοια ώστε  $G = t_n(\dots(t_2(t_1(I))))\dots$

## Χώρος αναζήτησης λύσης



Δοθέντος ενός προβλήματος  $(I, G, T, S)$

- ▶ Χώρος αναζήτησης (search space)  $SP$  είναι το σύνολο όλων των καταστάσεων που είναι προσβάσιμες από την αρχική κατάσταση
- ▶ Μία κατάσταση  $s$  ονομάζεται *προσβάσιμη (accessible)* αν υπάρχει μια ακολουθία τελεστών μετάβασης  $t_1, t_2, \dots, t_k$  τέτοια ώστε  $s = t_k(\dots(t_2(t_1(I))))\dots$
- ▶ Ο χώρος αναζήτησης είναι υποσύνολο του χώρου καταστάσεων, δηλαδή  $SP \subseteq S$

Ο χώρος αναζήτησης μπορεί γενικά να αναπαρασταθεί με *γράφο* που συνήθως μετατρέπεται σε *δέντρο αναζήτησης (OR-δέντρο)* (με μονοπάτια πιθανά απείρου μήκους αν ο γράφος έχει κύκλο)

## Δέντρα και χαρακτηριστικά προβλήματος



Δέντρο	Πρόβλημα
Κόμβος (node)	Κατάσταση (state)
Ρίζα (root)	Αρχική κατάσταση (initial state)
Φύλλο (tip, leaf) ή τερματικός κόμβος	Στόχος (goal) ή αδιέξοδο (dead node), δηλαδή κατάσταση στην οποία δεν μπορεί να εφαρμοστεί κανένας τελεστής μετάβασης
Κλαδί (branch)	Τελεστής μετάβασης που μετατρέπει μια κατάσταση-γονέα (parent state) σε μία άλλη κατάσταση-παιδί (child state)
Μονοπάτι (path) που ενώνει τη ρίζα με ένα φύλλο-στόχο	Λύση (solution)
Επέκταση (expansion)	Διαδικασία παραγωγής όλων των καταστάσεων-παιδιών ενός κόμβου
Παράγοντας διακλάδωσης (branching factor)	Αριθμός των καταστάσεων-παιδιών που προκύπτουν από την επέκταση μιας κατάστασης [Επειδή δεν είναι σταθερός αριθμός, αναφέρεται και ως μέσος παράγοντας διακλάδωσης (average branching factor)]

## Αλγόριθμοι επίλυσης προβλημάτων



**Αλγόριθμος επίλυσης** είναι μία αυστηρά καθορισμένη ακολουθία εύρεσης ενός κόμβου στόχου

- ▶ Ονομάζεται **εξαντλητικός** (exhaustive) όταν το σύνολο των καταστάσεων που εξετάζει ο αλγόριθμος για να βρει τις απαιτούμενες λύσεις είναι ίσο με το χώρο αναζήτησης
- ▶ Ένας αλγόριθμος δεν λύνει *πάντα* το πρόβλημα, έστω και αν υπάρχει λύση. Ονομάζεται **πλήρης** (complete) αν εγγυάται ότι θα βρει μία λύση για οποιοδήποτε στόχο, αν τέτοια λύση υπάρχει. Σε αντίθετη περίπτωση, ο αλγόριθμος ονομάζεται **μη-πλήρης** (incomplete).
- ▶ Μία λύση ονομάζεται **βέλτιστη** (optimal) αν οδηγεί στην καλύτερη, σύμφωνα με τη διάταξη, τελική κατάσταση. Όταν δεν υπάρχει διάταξη, μία λύση ονομάζεται βέλτιστη αν είναι η **συντομότερη** (shortest).

## Αλγόριθμοι αναζήτησης λύσης

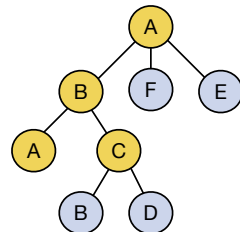


### Βασικές δομές δεδομένων

- ▶ **Μέτωπο αναζήτησης** (search frontier)  
Το διατεταγμένο σύνολο (λίστα) των καταστάσεων που ο αλγόριθμος έχει ήδη επισκεφτεί, αλλά δεν έχουν ακόμη επεκταθεί
- ▶ **Κλειστό σύνολο** (closed set)  
Το σύνολο όλων των καταστάσεων που έχουν ήδη επεκταθεί από τον αλγόριθμο

Κάθε επέκταση μιας κατάστασης συνοδεύεται από την εισαγωγή της κατάστασης γονέα στο κλειστό σύνολο και των καταστάσεων παιδιών στο μέτωπο αναζήτησης

Με έναν απλό έλεγχο, αν η κατάσταση προς επέκταση ανήκει ήδη στο κλειστό σύνολο, αποφεύγονται οι βρόχοι (loops)

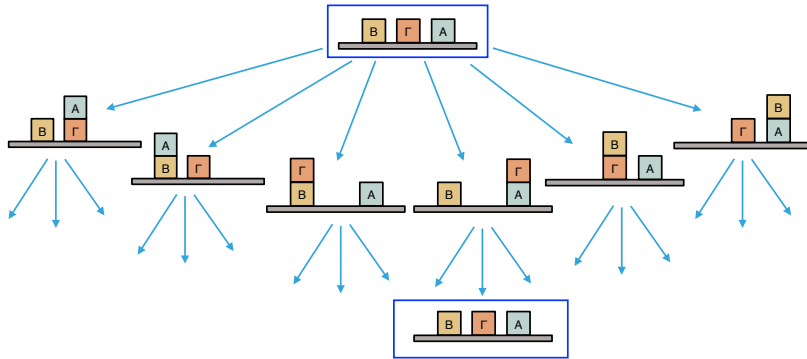


## Αλγόριθμος αναζήτησης λύσης



1. Βάλτε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αν το μέτωπο αναζήτησης είναι άδειο τότε σταμάτησε.
3. Πάρε την πρώτη σε σειρά κατάσταση του μετώπου της αναζήτησης.
4. Αν η κατάσταση είναι μέρος του κλειστού συνόλου τότε πήγαινε στο βήμα 2.
5. Αν η κατάσταση είναι τελική κατάσταση τότε τύπωσε τη λύση και πήγαινε στο βήμα 2.
6. Εφάρμοσε τους τελεστές μετάβασης για να παράγεις τις καταστάσεις-παιδιά.
7. Βάλτε τις νέες καταστάσεις-παιδιά στο μέτωπο αναζήτησης.
8. Κλάδεψε τις καταστάσεις που δεν χρειάζονται (σύμφωνα με κάποιο κριτήριο), και διέγραψε τις από το μέτωπο αναζήτησης.
9. Κάνε αναδιάταξη στο μέτωπο αναζήτησης (σύμφωνα με κάποιο κριτήριο).
10. Βάλτε την κατάσταση-γονέα στο κλειστό σύνολο.
11. Πήγαινε στο βήμα 2.

## Αλγόριθμος αναζήτησης λύσης



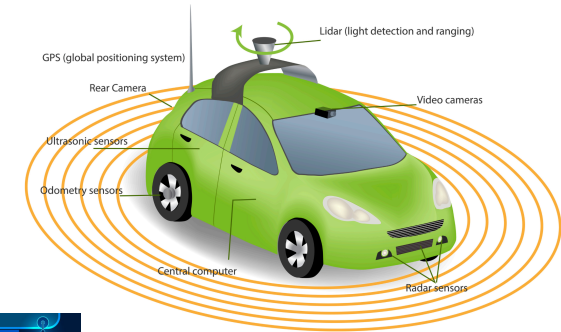
- ▶ Οι πιθανές καταστάσεις είναι πολλές και τα ζευγάρια αρχικής - τελικής κατάστασης ακόμα περισσότερα
- ▶ Οι ακολουθίες δράσεων που θα πρέπει να εξεταστούν είναι πολύ περισσότερες, ίσως και πρακτικά άπειρες
- ▶ Μπορεί περισσότερες από μία ακολουθίες δράσεων να οδηγούν σε λύση

## Αλγόριθμος αναζήτησης λύσης



### Αυτόνομα οχήματα

Source: [IEEE Innovation at work](#)



## Αλγόριθμος αναζήτησης λύσης



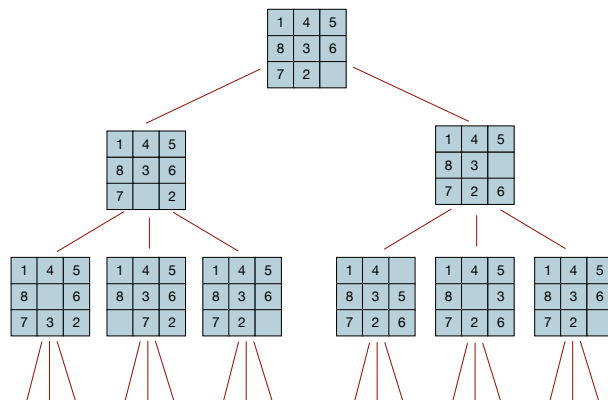
n-puzzle (Sam Loyd - 1878)

Αρχική κατάσταση

1	4	5
8	3	6
7	2	

Στόχος

1	2	3
4	5	6
7	8	



## Αλγόριθμος αναζήτησης λύσης



Πλήθος καταστάσεων

- ▶ 8-puzzle:  $9! = 362.880$  καταστάσεις
- ▶ 15-puzzle:  $16! \sim 2.09 \times 10^{13}$  καταστάσεις
- ▶ 24-puzzle:  $25! \sim 10^{25}$  καταστάσεις

Χρόνος υπολογισμού

- ▶ Έστω ότι μπορούμε να επεξεργαστούμε  $10^6$  καταστάσεις/sec
- ▶ Για όλες τις καταστάσεις του 8-puzzle χρειαζόμαστε 0.036 sec
- ▶ Για όλες τις καταστάσεις του 15-puzzle χρειαζόμαστε ~55 ώρες
- ▶ Για όλες τις καταστάσεις του 24-puzzle χρειαζόμαστε  $10^9$  χρόνια
  - ▶ Το πρόβλημα είναι NP-hard

## Αλγόριθμος αναζήτησης λύσης



### Πλήθος καταστάσεων

- ▶ 8-puzzle:  $9! = 362.880$  καταστάσεις
- ▶ 15-puzzle:  $16! \sim 2.09 \times 10^{13}$  καταστάσεις
- ▶ 24-puzzle:  $25! \sim 10^{25}$  καταστάσεις

### Προσβασιμότητα

- ▶ Δεδομένης μιας αρχικής κατάστασης, μπορούμε να φτάσουμε σε όλες τις άλλες καταστάσεις;

#### Αρχική κατάσταση

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

?

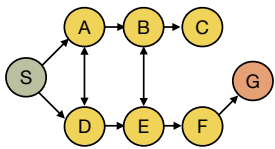
#### Στόχος

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

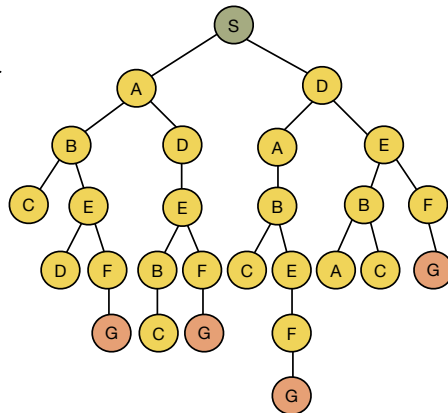
## Αλγόριθμοι αναζήτησης - Πληροφορίες



### Καμμία πληροφορία



### Απόσπασμα ενός δένδρου εκτέλεσης



## Αλγόριθμοι αναζήτησης λύσης



### Αλγόριθμοι τυφλής αναζήτησης λύσης

- ▶ Δεν έχουμε τρόπο να εκτιμήσουμε το κόστος των εναλλακτικών για το επόμενο βήμα (σε σχέση με τον τελικό στόχο), δηλαδή ψάχνουμε τη λύση χωρίς ένδειξη για το αν πλησιάζουμε σε κατάσταση στόχο

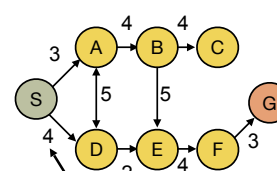
### Αλγόριθμοι εμπειριστατωμένης (informed) αναζήτησης λύσης

- ▶ Μπορούμε να υπολογίσουμε την πραγματική απόσταση της κατάστασης από τις επόμενες ή/και να εκτιμήσουμε την απόστασή της από τον στόχο, με βάση κάποια ευρετική συνάρτηση [Η ευρετική τιμή δεν είναι η πραγματική τιμή της απόστασης από μία κατάσταση στο στόχο, αλλά απλά μία εκτίμηση]

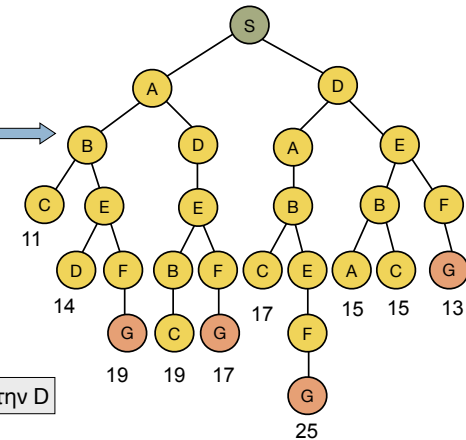
## Αλγόριθμοι αναζήτησης - Πληροφορίες



### Εκτίμηση κόστους μετάβασης από κατάσταση σε κατάσταση



### Απόσπασμα ενός δένδρου εκτέλεσης



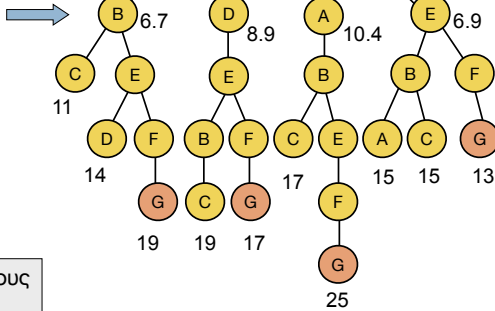
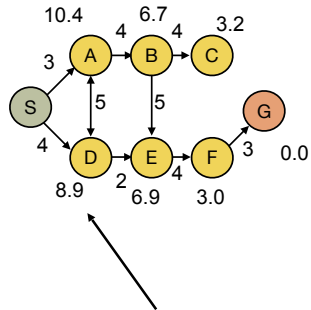
Κόστος μετάβασης από την S στην D

## Αλγόριθμοι αναζήτησης - Πληροφορίες



Εκτίμηση κόστους μετάβασης από μία κατάσταση στο στόχο

Απόσπασμα ενός δένδρου εκτέλεσης



Εκτίμηση υπολειπόμενου κόστους από την D ως τη G

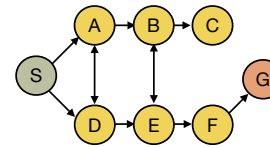
## Αλγόριθμος αναζήτησης λύσης



1. Βάλτε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αν το μέτωπο αναζήτησης είναι άδειο τότε σταμάτησε.
3. Πάρε την πρώτη σε σειρά κατάσταση του μετώπου της αναζήτησης.
4. Αν η κατάσταση είναι μέρος του κλειστού συνόλου τότε πήγαινε στο βήμα 2.
5. Αν η κατάσταση είναι τελική κατάσταση τότε τύπωσε τη λύση και πήγαινε στο βήμα 2.
6. Εφάρμοσε τους τελεστές μετάβασης για να παράγεις τις καταστάσεις-παιδιά.
7. Βάλτε τις νέες καταστάσεις-παιδιά στο μέτωπο αναζήτησης.
8. Κλάδεψε τις καταστάσεις που δεν χρειάζονται (σύμφωνα με κάποιο κριτήριο), και διέγραψε τις από το μέτωπο αναζήτησης.
9. Κάνε αναδιάταξη στο μέτωπο αναζήτησης (σύμφωνα με κάποιο κριτήριο).
10. Βάλτε την κατάσταση-γονέα στο κλειστό σύνολο.
11. Πήγαινε στο βήμα 2.

## Αλγόριθμοι τυφλής αναζήτησης

## Αλγόριθμος αναζήτησης κατά πλάτος (BFS)

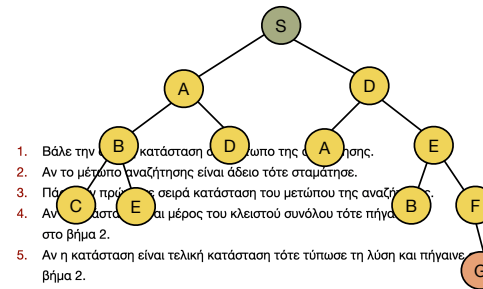


Μέτωπο αναζήτησης

[S]  
[A,D]  
[D,B,D]  
[B,D,A,E]  
[D,A,E,C,E]  
[A,E,C,E]  
[E,C,E]  
[C,E,B,F]  
[E,B,F]  
[B,F]  
[G]

Κλειστό σύνολο

[S]  
[A,S]  
[A,D,S]  
[A,B,D,S]  
[A,B,D,S]  
[A,B,D,S]  
[A,B,D,E,S]  
[A,B,C,D,E,S]  
[A,B,C,D,E,S]  
[A,B,C,D,E,S]  
[A,B,C,D,E,S]  
[A,B,C,D,E,F,S]



1. Βάλτε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αν το μέτωπο αναζήτησης είναι άδειο τότε σταμάτησε.
3. Πάρε την πρώτη σε σειρά κατάσταση του μετώπου της αναζήτησης.
4. Αν η κατάσταση είναι μέρος του κλειστού συνόλου τότε πήγαινε στο βήμα 2.
5. Αν η κατάσταση είναι τελική κατάσταση τότε τύπωσε τη λύση και πήγαινε στο βήμα 2.
6. Εφάρμοσε τους τελεστές μετάβασης για να παράγεις τις καταστάσεις-παιδιά.
7. Βάλτε τις νέες καταστάσεις-παιδιά στο μέτωπο αναζήτησης.
8. Βάλτε την κατάσταση-γονέα στο κλειστό σύνολο.
9. Πήγαινε στο βήμα 2.



## Αλγόριθμος αναζήτησης κατά πλάτος (BFS)



### Παράμετροι

- ▶ Μέγιστος παράγοντας διακλάδωσης:  $b$
- ▶ Βάθος κοντινότερου στόχου:  $d$

### Πολυπλοκότητα

- ▶ Μέγιστος αριθμός κόμβων που επισκέπτεται:
  - ▶  $1 + b + b^2 + \dots + b^d = (b^{d+1}-1)/(b-1) = O(b^d)$
- ▶ Απαιτήσεις χώρου  $O(b^d)$

## Αλγόριθμος αναζήτησης κατά πλάτος (BFS)



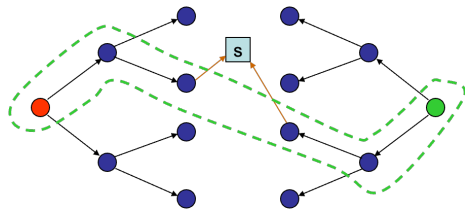
### Υπόθεση

- ▶  $b = 10$ ; 1,000,000 κόμβοι/sec; 100 bytes/κόμβο

$d$	# Κόμβων	Χρόνος	Μνήμη
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	$\sim 10^6$	1 sec	100 Mb
8	$\sim 10^8$	100 sec	10 Gbytes
10	$\sim 10^{10}$	2.8 hours	1 Tbyte
12	$\sim 10^{12}$	11.6 days	100 Tbytes
14	$\sim 10^{14}$	3.2 years	10,000 Tbytes

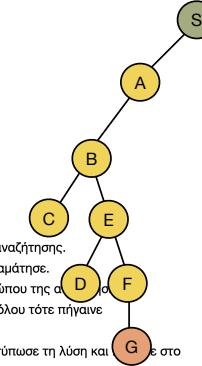
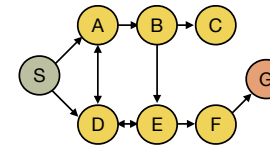
Μπορώ να σκεφτώ κάποια βελτιστοποίηση (για παράδειγμα να υπολογίζω παράλληλα);

## Αλγόριθμος διπλής κατεύθυνσης



- ▶ Αρχίζουμε την αναζήτηση από την αρχική και τελική κατάσταση ταυτόχρονα
- ▶ Αν κάποια κατάσταση που επεκτείνεται είναι κοινή και από τις 2 πλευρές, τότε βρέθηκε λύση
- ▶ Λύση είναι η ένωση των μονοπατιών από την κοινή κατάσταση ως την αρχική και ως την τελική κατάσταση
- ▶ Πολυπλοκότητα:  $O(b^{d/2}) \ll O(b^d)$

## Αλγόριθμος αναζήτησης κατά βάθος (DFS)



### Μέτωπο αναζήτησης

[S]  
[A,D]  
[B,D,D]  
[C,E,D,D]  
[E,D,D]  
[D,F,D,D]  
[F,D,D]  
[G,D,D]

### Κλειστό σύνολο

[S]  
[A,S]  
[A,B,S]  
[A,B,C,S]  
[A,B,C,E,S]  
[A,B,C,D,E,S]  
[A,B,C,D,E,F,S]

1. Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αν το μέτωπο αναζήτησης είναι άδειο τότε σταμάτησε.
3. Πάρε την πρώτη σε σειρά κατάσταση του μετώπου της αναζήτησης.
4. Αν η κατάσταση είναι μέρος του κλειστού συνόλου τότε πήγαινε στο βήμα 2.
5. Αν η κατάσταση είναι τελική κατάσταση τότε τύπωσε τη λύση και πήγαινε στο βήμα 2.
6. Εφάρμοσε τους τελευταίους μεταβάσεις για να παράγεις τις καταστάσεις-παιδιά.
7. Βάλε τις νέες καταστάσεις-παιδιά στο μέτωπο αναζήτησης.
8. Βάλε την κατάσταση-γονέα στο κλειστό σύνολο.
9. Πήγαινε στο βήμα 2.

## Αλγόριθμος αναζήτησης κατά βάθος (DFS)



### Παράμετροι

- ▶ Μέγιστος παράγοντας διακλάδωσης:  $b$
- ▶ Βάθος κοντινότερου στόχου:  $d$
- ▶ Βάθος πιο απομακρυσμένου φύλλου:  $m$

### Πολυπλοκότητα

- ▶ Μέγιστος αριθμός κόμβων που επισκέπτεται:
  - ▶  $1 + b + b^2 + \dots + b^m = O(b^m)$
- ▶ Απαιτήσεις χρόνου  $O(b^m)$
- ▶ Απαιτήσεις χώρου  $O(bm)$

## Αλγόριθμος επαναληπτικής εκβάθυνσης



### Βήματα Αλγόριθμου (Iterative Deepening - ID)

Βήμα 1. Όρισε το αρχικό βάθος αναζήτησης (συνήθως 1).

Βήμα 2. Εφάρμοσε τον αλγόριθμο Κατά-Βάθος μέχρι αυτό το βάθος αναζήτησης.

Βήμα 3. Αν έχεις βρει λύση σταμάτησε.

Βήμα 4. Αύξησε το βάθος αναζήτησης (συνήθως κατά 1).

Βήμα 5. Πήγαινε στο Βήμα 2.

- ▶ Έχει παρόμοια πολυπλοκότητα σε χώρο και χρόνο με τους DFS και BFS, όταν έχουμε μεγάλους χώρους αναζήτησης, παρόλο που επαναλαμβάνει άσκοπα το κτίσιμο του χώρου αναζήτησης

## Αλγόριθμος επαναληπτικής εκβάθυνσης



### Παράμετροι

- ▶ Μέγιστος παράγοντας διακλάδωσης:  $b$
- ▶ Βάθος κοντινότερου στόχου:  $d$
- ▶ Βάθος πιο απομακρυσμένου φύλλου:  $m$

### Πολυπλοκότητα

- ▶ Απαιτήσεις χρόνου:
  - ▶  $(d+1)(1) + db + (d-1)b^2 + \dots + (1) b^d = O(b^d)$
- ▶ Απαιτήσεις χώρου  $O(bd)$

## Αλγόριθμος επαναληπτικής εκβάθυνσης



- ▶  $b = 2; d=5$

BFS	ID
1	$1 \times 6 = 6$
2	$2 \times 5 = 10$
4	$4 \times 4 = 16$
8	$8 \times 3 = 24$
16	$16 \times 2 = 32$
32	$32 \times 1 = 32$
63	120

- ▶  $b = 10; d=5$

BFS	ID
1	6
10	50
100	400
1,000	3,000
10,000	20,000
111,111	123,456

## Αλγόριθμοι εμπειριστατωμένης αναζήτησης (αναζήτησης μίας οποιασδήποτε λύσης)

## Ευρετικοί μηχανισμοί



Ευρετικός μηχανισμός και συναρτήσεις σε λαβύρινθο  
Ευκλείδεια απόσταση (Euclidian distance):

$$d(S, F) = \sqrt{(X_S - X_F)^2 + (Y_S - Y_F)^2}$$

Απόσταση Manhattan (Manhattan distance):

$$Md(S, F) = |X_S - X_F| + |Y_S - Y_F|$$

## Ευρετικοί μηχανισμοί



Ευρετικός μηχανισμός και συναρτήσεις στο N-Puzzle

12	9	3	14
5	6	11	8
2	10	7	1
13	4	15	

15	2	3	13
5	10	11	14
12	6	7	9
4	8	1	

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

## Ευρετικοί μηχανισμοί



Ευρετικός μηχανισμός και συναρτήσεις στο N-Puzzle

- Πόσα πλακίδια βρίσκονται εκτός θέσης
- Το άθροισμα των αποστάσεων Manhattan κάθε πλακιδίου από την τελική του θέση

Αρχική κατάσταση

7	8	15	5
14	10	2	12
9	1	6	11
13	4	3	

7	14	3	6
8	10	2	12
9	1	5	11
13		15	4

Τυχαία κατάσταση

Εκτός Θέσης = 12  
Άθροισμα Manhattan  
αποστάσεων = 28

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

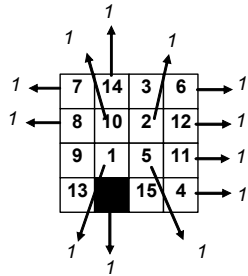
Ευριστική Τιμή  
(εκτίμηση απόστασης)

Τελική κατάσταση

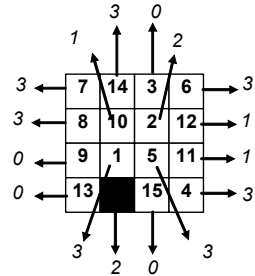
## Ευρετικοί μηχανισμοί



Αναλυτικός υπολογισμός ευρετικής τιμής για μία τυχαία κατάσταση του 15-puzzle



Εκτός θέσης = 12



Άθροισμα αποστάσεων Manhattan = 28

## Αλγόριθμος Hill climbing



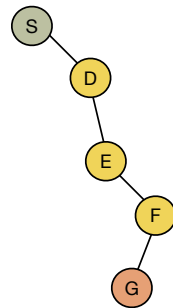
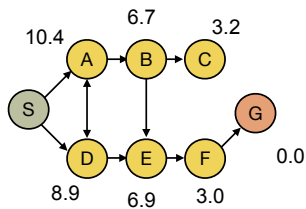
**Βήμα 1:** Όρισε τον τρέχοντα κόμβο ως τη ρίζα του δένδρου

**Βήμα 2:** Όσο ο τρέχων κόμβος δεν είναι κόμβος στόχος, εκτέλεσε:

- **Βήμα 2.α:** Βρες τα παιδιά του τρέχοντος κόμβου, και στη συνέχεια βρες αυτό με την ελάχιστη υπολογιζόμενη υπόλοιπη απόσταση από το στόχο
- **Βήμα 2.β:** Εάν ο τρέχων κόμβος δεν έχει παιδιά ή το παιδί που βρέθηκε στο βήμα 2.α έχει μεγαλύτερη τιμή ευρετικής συνάρτησης από τον τρέχων κόμβο πήγαινε στο βήμα 3.
- **Βήμα 2.γ:** Όρισε τον κόμβο που βρέθηκε στο Βήμα 2.α ως τρέχων κόμβο.

**Βήμα 3:** Εάν βρήκαμε ένα κόμβο στόχο τότε ανακοινώνουμε επιτυχία αλλιώς ανακοινώνουμε αποτυχία

## Αλγόριθμος Hill climbing



- [S]
- [D]
- [E]
- [F]
- [G]
  
- [S]
- [D,S]
- [D,E,S]
- [D,E,F,S]

## Αλγόριθμος Best-First



- ▶ Ο Best First μοιάζει με τον Hill Climbing, μόνο που εδώ επεκτείνουμε όχι τον καλύτερο κόμβο από τα παιδιά του κόμβου που είμαστε, αλλά τον καλύτερο κόμβο από όλους τους κόμβους που βρίσκονται στο μέτωπο αναζήτησης του δένδρου (συνεπώς κρατάμε στο μέτωπο αναζήτησης όλα τα παιδιά του κόμβου που επεκτείνουμε)
- ▶ Ο αλγόριθμος αναζήτησης Best First Search είναι πιθανότερο να παράγει μικρότερα μονοπάτια από την αρχική κατάσταση στο στόχο
- ▶ Η βασική διαφορά από τον Hill Climbing είναι ότι αντί να κρατάμε το καλύτερο από τα παιδιά του κόμβου που επεκτείνουμε, εισάγουμε τα παιδιά του κόμβου που επεκτείνουμε στη λίστα, και μετά ταξινομούμε όλη τη λίστα. Οπότε:
  - ▶ Ο κόμβος με τη συνολικά μικρότερη τιμή ευρετικής, ανεβαίνει στην κορυφή της λίστας για επέκταση στο επόμενο βήμα

## Αλγόριθμος Best-First



**Βήμα 1:** Κατασκευάσε το μέτωπο αναζήτησης που περιέχει τη ρίζα του δένδρου (αρχική κατάσταση)

**Βήμα 2:** Μέχρι που το μέτωπο αναζήτησης να αδειάσει ή να βρεθεί ένας στόχος, εξέτασε εάν ο πρώτος κόμβος στη λίστα είναι κόμβος στόχος

- **Βήμα 2.α:** Εάν ο πρώτος κόμβος είναι κόμβος στόχος τότε πήγαινε στο Βήμα 3
- **Βήμα 2.β:** Εάν ο πρώτος κόμβος δεν είναι κόμβος στόχος, τότε πάρε βρες τα παιδιά του, προσέθεσέ τα στο μέτωπο αναζήτησης, και **ταξινόμησε το μέτωπο αναζήτησης σε αύξουσα σειρά ευρετικής**
- **Βήμα 2.γ:** Εάν ο πρώτος κόμβος δεν έχει παιδιά απλά αφάιρεσέ τον από το μέτωπο αναζήτησης και πήγαινε στο βήμα 2

**Βήμα 3:** Εάν βρήκαμε ένα κόμβο στόχο τότε ανακοινώνουμε επιτυχία αλλιώς ανακοινώνουμε αποτυχία

► Πώς θυμόμαστε το μονοπάτι για το στόχο????

## Αλγόριθμος Best-First



**Βήμα 1:** Κατασκευάσε το μέτωπο αναζήτησης από μονοπάτια (που αρχικά είναι κενή)

**Βήμα 2:** Μέχρι που το μέτωπο αναζήτησης να είναι άδειο ή το πρώτο μονοπάτι να έχει ουρά (τελευταίο στοιχείο) ένα στόχο

- Βήμα 2.1. Βγάλε το πρώτο μονοπάτι από το μέτωπο αναζήτησης
- Βήμα 2.2. Φτιάξε μονοπάτια που μπορούν να φτιαχτούν από το μονοπάτι που βγάλαμε επεκτείνοντας το κατά ένα βήμα
- Βήμα 2.3. Βάλε τα νέα μονοπάτια στο μέτωπο αναζήτησης
- Βήμα 2.4. Ταξινόμησε σε αύξουσα σειρά ευρετικής όλα τα μονοπάτια με βάση την ευρετική τιμή της ουράς

**Βήμα 3:** Εάν βρήκαμε ένα μονοπάτι που οδηγεί σε κόμβο στόχο τότε ανακοινώνουμε επιτυχία αλλιώς ανακοινώνουμε αποτυχία

## Αλγόριθμοι εμπεριστατωμένης αναζήτησης (αναζήτηση βέλτιστης λύσης)

## Στρατηγική αλγορίθμων εύρεσης βέλτιστης λύσης



- Αποθηκεύουμε στο μέτωπο αναζήτησης **μονοπάτια** από την αρχική κατάσταση σε όλες τις καταστάσεις «στόχους»
- Επιλέγουμε ένα από τα μονοπάτια για να επεκτείνουμε με μετάβαση σε μία προσβάσιμη κατάσταση
- Οργανώνουμε την επιλογή και επέκταση των μονοπατιών έτσι ώστε ή να ελέγξουμε τελικά όλα τα μονοπάτια ή όσα αποκλείσουμε, **διασφαλισμένα** να μην οδηγούν σε βέλτιστη λύση

## Αλγόριθμος Branch and Bound



**Βήμα 1:** Κατασκευάσε μια λίστα από μονοπάτια (που αρχικά είναι κενή)

**Βήμα 2:** Μέχρι που η λίστα να είναι άδεια ή το πρώτο μονοπάτι της λίστας να οδηγεί σε «στόχο» και όλα τα άλλα μονοπάτια που δεν έχουν ακόμη οδηγήσει σε στόχο έχουν μεγαλύτερο κόστος

- **Βήμα 2.α:** Εάν το πρώτο μονοπάτι οδηγεί σε στόχο, κράτησέ το σαν πιθανή λύση. Εάν είναι καλύτερο από κάποια προηγούμενη λύση, κράτησέ το σαν την καλύτερη πιθανή λύση και Προχώρησε στο Βήμα 2.β

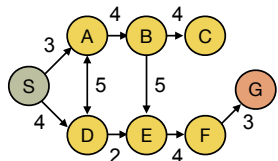
## Αλγόριθμος Branch and Bound



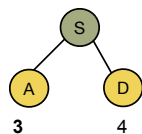
- **Βήμα 2.β:** Εάν το πρώτο μονοπάτι δεν οδηγεί σε στόχο, ή υπάρχουν άλλα μονοπάτια που δεν έχουν ακόμη οδηγήσει σε στόχο και έχουν μικρότερο κόστος από το μονοπάτι που ήδη βρήκαμε
  - Βήμα 2.β.1. Βγάλε το πρώτο μονοπάτι από τη λίστα
  - **Βήμα 2.β.2.** Φτιάξε μονοπάτια που μπορούν να φτιαχτούν από το μονοπάτι που βγάλαμε επεκτείνοντας το κατά ένα βήμα (branch)
  - Βήμα 2.β.3. Βάλε τα νέα μονοπάτια στη λίστα
  - Βήμα 2.β.4. Ταξιλόγησε σε αύξουσα σειρά όλα τα μονοπάτια στη λίστα σύμφωνα με το κόστος του κάθε μονοπατιού (από την αρχική κατάσταση στο τελευταίο κόμβο του μονοπατιού)
  - **Βήμα 2.β.5.** Κλάδεψε τα μονοπάτια που έχουν κόστος μεγαλύτερο από ένα *όριο* που διασφαλισμένα δεν μπορεί να οδηγήσει σε βέλτιστη λύση (bound) (π.χ. το κόστος της καλύτερης λύσης που έχουμε βρει μέχρι τότε)
  - Βήμα 2.β.6. Εάν βρήκαμε ένα μονοπάτι που οδηγεί σε κόμβο στόχο τότε ανακοινώνουμε μερική επιτυχία αλλιώς ανακοινώνουμε αποτυχία

**Βήμα 3:** Εάν βρήκαμε ένα μονοπάτι που οδηγεί σε κόμβο στόχο τότε ανακοινώνουμε επιτυχία αλλιώς ανακοινώνουμε αποτυχία

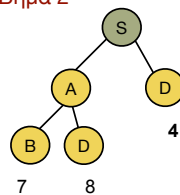
## Αλγόριθμος Branch and Bound



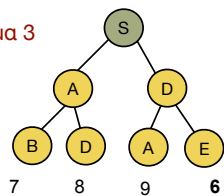
Βήμα 1



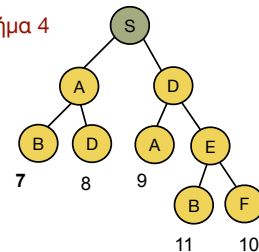
Βήμα 2



Βήμα 3



Βήμα 4

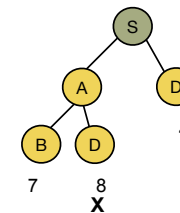


## Δυναμικός προγραμματισμός (DP)

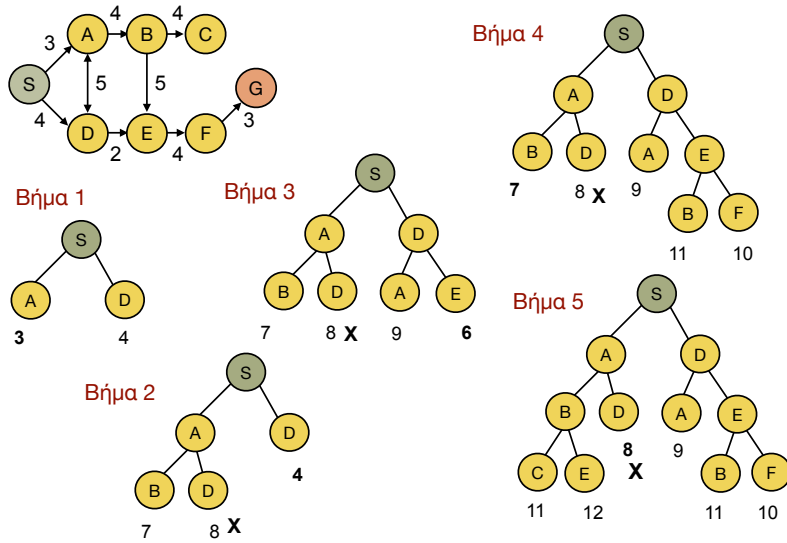


### Διαισθητικά

- ▶ Επεκτείνουμε δυναμικά το όριο απόρριψης κάποιου μονοπατιού:  
*Αν κατασκευάσουμε ένα μονοπάτι που οδηγεί σε κάποιο κόμβο, ενώ υπάρχει κάποιο άλλο μονοπάτι που οδηγεί στον ίδιο κόμβο αλλά με μικρότερο κόστος, τότε «κλαδεύουμε» το μονοπάτι με το μεγαλύτερο κόστος*



## DP και Branch and Bound

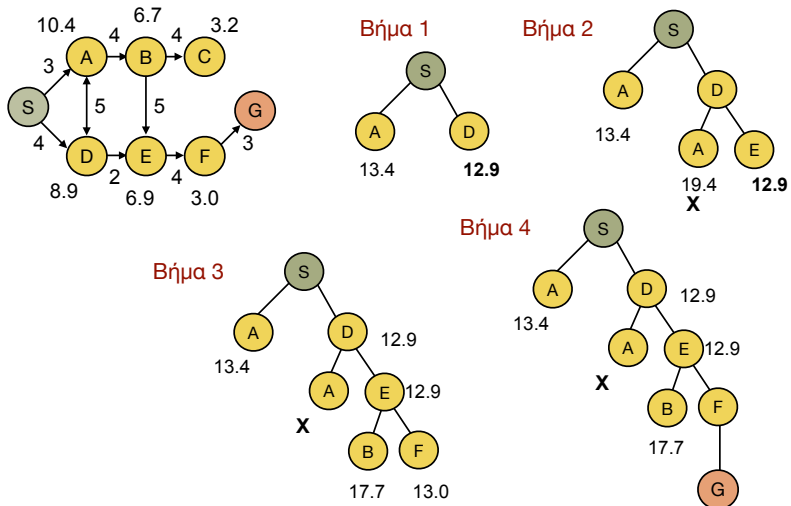


## Αλγόριθμος A\*

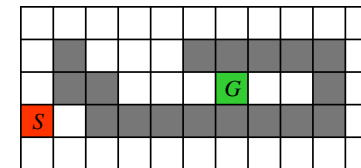


- ▶ Ακολουθούμε τη λογική του Branch and Bound
- ▶ Επεκτείνουμε το μονοπάτι με τον καλύτερο από όλους τους κόμβους που βρίσκονται στο μέτωπο αναζήτησης του δένδρου (λογική Best First).
- ▶ Για το σκοπό αυτό χρησιμοποιούμε τη σύνθετη ευρετική συνάρτηση  $F(k) = g(k) + h(k)$ 
  - $g(k)$  η απόσταση της  $k$  από την αρχική κατάσταση, η οποία είναι πραγματική και γνωστή
  - $h(k)$  μία εκτίμηση της απόστασης της  $k$  από το στόχο (μέσω μιας ευρετικής συνάρτησης)
- ▶ Εφαρμόζουμε για οριοθέτηση (bound) δυναμικό προγραμματισμό

## Αλγόριθμος A\*



## Ευρετικές συναρτήσεις



$h$  : Απόσταση Manhattan

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

$h(k)$

8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

$h(k) + g(k)$

## Αποδεκτές ευρετικές



### Αποδεκτές (admissible) ευρετικές

Έστω  $h^*(k)$  το κόστος του βέλτιστου μονοπατιού από την  $k$  στο στόχο  $G$

Θα λέμε την  $h^*(k)$  βέλτιστη ευρετική

Μία ευρετική  $h(k)$  λέγεται *αποδεκτή* όταν:

$$0 \leq h(k) \leq h^*(k)$$

- ▶ Οι αποδεκτές ευρετικές είναι βέλτιστες σε ένα απλούστερο πρόβλημα (σε ένα πρόβλημα που αγνοούμε κάποιους περιορισμούς)
  - ▶ Η Ευκλείδεια απόσταση είναι η βέλτιστη ευρετική όταν δεν έχω εμπόδια
  - ▶ Η απόσταση Manhattan είναι η βέλτιστη ευρετική όταν δεν έχω εμπόδια και μπορώ να κινηθώ μόνο οριζοντίως ή καθέτως

## Αποδεκτές ευρετικές



### Θεώρημα

- ▶ Ο αλγόριθμος  $A^*$  είναι πλήρης και βρίσκει τη βέλτιστη λύση για κάθε αποδεκτή ευρετική

### Απόδειξη

Για κάθε κόμβο  $k$  στο μέτωπο αναζήτησης έχουμε:

$$f(k) = g(k) + h(k) \geq g(k) \geq d(k) \cdot \epsilon$$

όπου  $d(k)$  το βάθος του  $k$  και  $\epsilon$  το ελάχιστο κόστος μετάβασης

Κάθε κόμβος  $k$  που βρίσκεται στο μέτωπο αναζήτησης θα επεκταθεί

εκτός αν βρεθεί μικρότερο από  $d(k) \cdot \epsilon$  μονοπάτι για το στόχο

Συνεπώς ο  $A^*$  τερματίζει και επιστρέφει μία λύση

## Αποδεκτές ευρετικές



Έστω  $c^*$  το κόστος του βέλτιστου μονοπατιού προς το στόχο  $G$

Έστω  $c'$  το κόστος ενός άλλου μονοπατιού προς το στόχο  $G'$

Τότε, θα ισχύει:

$$f(G') = g(G') + h(G') = g(G') > c^*$$

ενώ για κάθε κόμβο  $k$  του βέλτιστου μονοπατιού θα ισχύει:

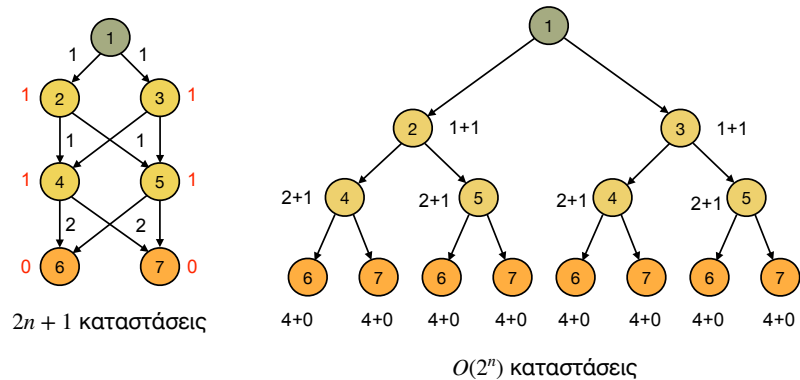
$$f(k) = g(k) + h(k) \leq c^*$$

Συνεπώς ο  $G'$  δεν θα επιλεγεί ποτέ για επέκταση

Άρα, όταν ο  $A^*$  επιλέγει για επέκταση ένα κόμβο στόχο αυτός είναι στο βέλτιστο μονοπάτι

Συνεπώς ο  $A^*$  βρίσκει το βέλτιστο μονοπάτι προς το στόχο

## Πολυπλοκότητα ευρετικών



- ▶ Ο  $A^*$  μπορεί να κατασκευάσει εκθετικά μεγάλα δέντρα



## Συνεπείς ευρετικές



Συνεπείς (consistent) ή μονότονες (monotone) ευρετικές

Έστω  $h(k)$  μία ευρετική

Έστω  $k$  ένας κόμβος και  $k'$  ένας επόμενός του

Η  $h(k)$  λέγεται *συνεπής* ή *μονότονη* όταν:

$$h(k) \leq c(k, k') + h(k')$$

όπου  $c(k, k')$  το κόστος μετάβασης από την  $k$  στην  $k'$

- ▶ Οι συνεπείς ευρετικές γίνονται πιο ακριβείς όσο πλησιάζουμε το στόχο
- ▶ Ισχύει ότι οι συνεπείς ευρετικές είναι αποδεκτές

## Συνεπείς ευρετικές



Θεώρημα

- ▶ Ο αλγόριθμος  $A^*$  που χρησιμοποιεί *συνεπείς* ευρετικές, όταν επεκτείνει ένα μονοπάτι, αυτό είναι το βέλτιστο μονοπάτι προς το τελευταίο στοιχείο του

Απόδειξη

Έστω  $k$  ένας κόμβος και  $k'$  ένας επόμενός του

Αφού η  $h(k)$  είναι συνεπής, θα ισχύει ότι:

$$h(k) \leq c(k, k') + h(k')$$

Συνεπώς:  $f(k) = g(k) + h(k) \leq g(k) + c(k, k') + h(k') = f(k')$

Άρα η  $f$  είναι αύξουσα σε κάθε μονοπάτι

## Συνεπείς ευρετικές



Όταν ένας κόμβος  $k$  επιλέγεται για επέκταση

για κάθε άλλο κόμβο  $k'$  του μετώπου αναζήτησης ισχύει ότι:

$$f(k') \geq f(k)$$

Συνεπώς, αν υπάρχει άλλο μονοπάτι προς τον  $k'' = k$  μέσω του  $k'$

δηλαδή  $[\dots, k', \dots, k'' = k]$

τότε  $f(k'') \geq f(k') \geq f(k)$

και επειδή  $h(k'') = h(k)$  ισχύει ότι:

$$g(k'') \geq g(k)$$

## Ακρίβεια ευρετικών



Ακρίβεια (accuracy) ευρετικών

Έστω  $h_1, h_2$  δύο συνεπείς ευρετικές τέτοιες ώστε για κάθε  $k$ :

$$h_1(k) \leq h_2(k)$$

Τότε η  $h_2(k)$  θα λέγεται πιο ακριβής (accurate) από την  $h_1(k)$

Θεώρημα

Έστω  $h_1, h_2$  δύο συνεπείς ευρετικές με  $h_2$  πιο ακριβή από την  $h_1$

Έστω  $A_1^*$  ο αλγόριθμος  $A^*$  που χρησιμοποιεί την  $h_1$

και  $A_2^*$  ο αλγόριθμος  $A^*$  που χρησιμοποιεί την  $h_2$

Αν υπάρχει λύση, όλοι οι κόμβοι που επεκτείνονται από τον  $A_2^*$

επεκτείνονται και από τον  $A_1^*$

(εκτός ίσως από ορισμένους για τους οποίους  $f_1(k) = f_2(k) = c^*$ )

## Ευρετικές συναρτήσεις



Αποτελεσματικός παράγοντας διακλάδωσης (effective branching factor) ευρετικής

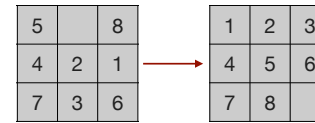
Έστω  $n$  το πλήθος των κόμβων που επεκτείνονται και  $d$  το βάθος του στόχου

Ο αποτελεσματικός παράγοντας διακλάδωσης  $b^*$  ορίζεται από τη σχέση:

$$n = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

- ▶ Είναι ένα μέτρο της αποτελεσματικότητας της ευρετικής

## Ευρετικές συναρτήσεις

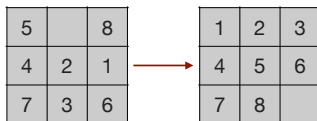


$h_1$ : αριθμός πλακιδίων εκτός θέσης

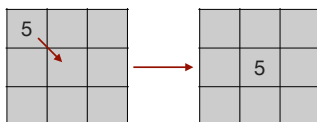
$h_2$ : απόσταση πλακιδίων από τελική θέση

d	IDS	A <sub>1</sub> *	A <sub>2</sub> *
2	2.45	1.79	1.79
6	2.73	1.34	1.30
12	2.78 (3,644,035)	1.42 (227)	1.24 (73)
16	--	1.45	1.25
20	--	1.47	1.27
24	--	1.48 (39,135)	1.26 (1,641)

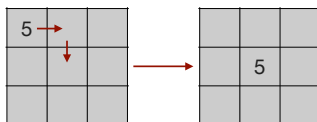
## Ευρετικές συναρτήσεις



Ευκολότερο πρόβλημα

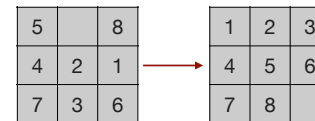


Αγνώ τα υπόλοιπα πλακίδια  
Μετακινώ τα πλακίδια σε ένα βήμα

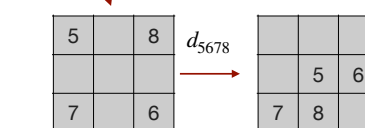
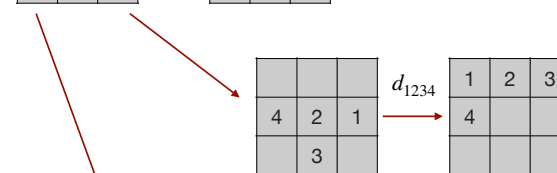


Αγνώ τα υπόλοιπα πλακίδια

## Ευρετικές συναρτήσεις



Ευκολότερο πρόβλημα



Ευρετική  $h = d_{1234} + d_{5678}$

- ▶ Αρκετές τάξεις μεγέθους πιο αποδοτική για >15-puzzle

## Αλγόριθμοι αναζήτησης για παίγνια

## Εισαγωγή



- Σε πολλά προβλήματα της ΤΝ, η εξέλιξη των καταστάσεων εξαρτάται από δύο διαφορετικά σύνολα τελεστών μετάβασης που εφαρμόζονται εναλλάξ από δύο ενεργά συστήματα (πράκτορες-agents)
- Αυτά τα προβλήματα αναφέρονται και ως ανταγωνιστικά παίγνια ή παίγνια δύο αντιπάλων (*adversary ή two-person games*)
  - Επιλύονται με ειδικούς αλγορίθμους αναζήτησης, βασικότεροι των οποίων είναι ο minimax και ο alpha-beta
- Ο όρος "παίγνιο" αφορά την περιγραφή του τρόπου με τον οποίο παίζεται το παιχνίδι και περιλαμβάνει:
  - τα αντικείμενα που υπάρχουν (για παράδειγμα, τα πούλια, το ταμπλώ, κτλ.) καθώς και
  - το σύνολο των κανόνων (για παράδειγμα, των επιτρεπτών κινήσεων, των συνθηκών νίκης, κτλ.) που το διέπουν
- Αντίθετα, με τον όρο "παιχνίδι" χαρακτηρίζεται μία συγκεκριμένη παρτίδα του παιχνιδιού
  - Γνωστά παραδείγματα παιχνιδιών είναι το σκάκι, η τρίλιζα, το τάβλι, κτλ.

## Εισαγωγή



- Σε ένα παίγνιο δύο ατόμων το πρόβλημα ορίζεται ως εξής:
  - Μια κατάσταση παριστάνει τη διάταξη των πιονιών σε κάποια χρονική στιγμή.
  - Ο χώρος καταστάσεων αποτελείται από όλες αυτές τις πιθανές επιτρεπτές καταστάσεις.
  - Οι τελεστές μετάβασης είναι οι επιτρεπτές κινήσεις που καθορίζονται από τους κανόνες του παιχνιδιού. Οι κανόνες του παιχνιδιού παίζουν και το ρόλο των προϋποθέσεων αυτών των τελεστών.
  - Οι τελικές καταστάσεις του παιχνιδιού έχουν γνωστά χαρακτηριστικά (π.χ. ματ στο σκάκι).
- Έστω ότι κάποιος συμμετέχει σε ένα παιχνίδι και είναι η σειρά του να κάνει μία κίνηση.
  - Αν θέλει να κερδίσει, θα επιδιώξει να κάνει την καλύτερη κίνηση για αυτόν, η οποία αντίστοιχα θα είναι και η χειρότερη για τον αντίπαλο.
  - Ο τρόπος που πιθανά σκέφτεται κάποιος για να παίξει είναι ο εξής: "Αν κάνω αυτή την κίνηση, τότε ο αντίπαλος θα κάνει εκείνη, αν όμως κάνω την άλλη κίνηση, τότε θα κάνει αυτήν, ..." κ.ο.κ.
  - Ο συλλογισμός αυτός αντιστοιχεί στη δημιουργία ενός δένδρου με όλες τις εναλλακτικές κινήσεις που πηγάζουν από την παρούσα κατάσταση.
  - Το δένδρο αυτό ονομάζεται **δένδρο του παιχνιδιού (game tree)**.

## Εισαγωγή



- Το χαρακτηριστικό του είναι ότι οι κινήσεις δύο διαδοχικών επιπέδων ανήκουν σε διαφορετικό παίκτη, γιατί οι παίκτες παίζουν εναλλάξ.
  - Το δένδρο αυτό αναπτύσσεται νοερά από τον παίκτη του οποίου είναι η σειρά να παίξει.
  - Αυτή η διαδικασία μπορεί να συνεχιστεί μέχρι κάποιο βάθος, συνήθως μικρό, και να σταματήσει εκεί όπου αρχίζει η αξιολόγηση των καταστάσεων με βάση κάποια κριτήρια.
  - Ο παίκτης αξιολογεί τις καταστάσεις που θα προκύψουν από πιθανές διαφορετικές εξελίξεις του παιχνιδιού και αποφασίζει ποια από τις εναλλακτικές κινήσεις θα τον οδηγήσει σταδιακά στην πιο ευνοϊκή για αυτόν εξέλιξη.
- Έστω ότι ένας από τους δύο αντιπάλους είναι ο υπολογιστής.
  - Το πρόγραμμα αναζήτησης και ο άνθρωπος πρέπει να εξετάσουν και να θυμούνται όλες τις πιθανές κινήσεις που παράγονται από μία (αρχική) κατάσταση.
  - Ένα τέτοιο πρόγραμμα μπορεί να προβλέψει την έκβαση του παιχνιδιού μετά από  $n$  κινήσεις (βάθος  $n$  του δένδρου παιχνιδιού), ενώ ο άνθρωπος σταματά συνήθως μετά την πρόβλεψη 2 ή 3 κινήσεων.

## Εισαγωγή



- Παραδόξως, οι διαφορές αυτές δεν κάνουν κάποιο πρόγραμμα κυρίαρχο οποιουδήποτε παιχνιδιού.
  - Π.χ., το Deep Blue (συνδυασμός λογισμικού και υλικού) μπορεί, κάποιες φορές, να νικήσει τον παγκόσμιο πρωταθλητή στο σκάκι.
  - Ο λόγος για την ανωτερότητα των ανθρώπων στο σκάκι είναι από τη μία μεριά απλώς να περιγραφεί, αλλά από την άλλη πολύ δύσκολος να υλοποιηθεί σαν πρόγραμμα.
- Η ανωτερότητα τους έγκειται στους εξής παράγοντες:
  - Οι μαίτρ του σκάκι έχουν τη δυνατότητα να επιλέγουν με κάποιον ευρετικό τρόπο τις εναλλακτικές κινήσεις, αλλά δεν μπορούν να μετατρέψουν αυτόν τον τρόπο σε αντίστοιχη ευρετική τιμή.
- Οι άνθρωποι διαθέτουν κάποια διαίσθηση για την κατάληξη του παιχνιδιού αν επιλεγεί κάποια από τις εναλλακτικές κινήσεις η οποία επίσης δε μπορεί να αποδοθεί αριθμητικά ώστε να υπάρξει ευρετική κλάδεμα του χώρου αναζήτησης.
- Τέλος, με την εμπειρία που αποκτούν, μπορούν να σκέφτονται πολλές φορές εντελώς μηχανικά, ιδίως στα πρώτα και τελευταία στάδια του παιχνιδιού.

## Εισαγωγή



- Το κύριο μέλημα των προγραμμάτων σκάκι είναι να αναζητήσουν το δένδρο του παιχνιδιού σε όσο το δυνατόν μεγαλύτερο βάθος.
  - Αν υπήρχε η δυνατότητα να φτάσουν μέχρι τις τελικές καταστάσεις του παιχνιδιού τότε τα προγράμματα θα ήταν ανίκητα.
  - Ο λόγος που κάτι τέτοιο είναι αδύνατο πηγάζει από τους χρονικούς περιορισμούς μιας παρτίδας και από το πρόβλημα της συνδυαστικής έκρηξης.
- Αν υποθέσουμε ότι από κάθε κατάσταση στο σκάκι παράγονται άλλες 10 καταστάσεις (Μ.Ο.), και ότι μία παρτίδα τελειώνει σε 40 κινήσεις (20 για κάθε παίκτη), τότε οι πιθανές τελικές καταστάσεις θα ήταν της τάξης του  $10^{40}$ .
  - Αν ένας υπέρ-υπολογιστής ξοδεύει 1 nanosecond ( $=10^{-9}$  sec) για να εξετάσει κάθε μία από αυτές, τότε χρειάζεται  $10^{40} \times 10^{-9} = 10^{31}$  sec συνολικά για να τις εξετάσει όλες.
  - Έχοντας υπόψη πως ένα έτος έχει  $3 \times 10^7$  sec περίπου, ο υπέρ-υπολογιστής αυτός θα χρειαζόταν  $3 \times 10^{24}$  χρόνια για να διαλέξει την επόμενη κίνησή του.
  - Αυτό είναι περίπου και το 1/4 της ηλικίας του σύμπαντος!
- Αφού λοιπόν δεν μπορούμε να περιμένουμε τόσο πολύ, οι αλγόριθμοι αναζήτησης που υπάρχουν εξετάζουν το δένδρο του παιχνιδιού μέχρι ένα ορισμένο βάθος, τόσο όσο επιτρέπει ο χρόνος για τη κίνηση.

## Ο Αλγόριθμος Min-Max



- Δεδομένης μίας κατάστασης του παιχνιδιού, ο αλγόριθμος *αναζήτησης μεγίστου-ελαχίστου (Minimax)* καλείται να αποφασίσει ποια θα είναι η επόμενη κίνησή του έναντι του αντιπάλου.
- Η εξαντλητική αναζήτηση των δένδρων αναζήτησης είναι ανέφικτη.
- Το ζητούμενο είναι:
  - Να χτιστεί το δένδρο μέχρι κάποιο βάθος
  - Να βρεθεί η καλύτερη κίνηση από την παρούσα κατάσταση.
- Το μέτρο της υπεροχής του ενός ή του άλλου αντιπάλου δίνεται από μία ευριστική συνάρτηση που καλείται συνάρτηση αξιολόγησης (evaluation function) και η οποία εφαρμόζεται στα φύλλα του δένδρου του παιχνιδιού.
- Ο ένας παίκτης (πρόγραμμα) ονομάζεται max και ο άλλος (άλλο πρόγραμμα ή άνθρωπος) ονομάζεται min.
- Οι κόμβοι του δένδρου διακρίνονται σε:
  - Αυτούς που σειρά έχει να παίξει ο max
  - Αυτούς που σειρά έχει να παίξει ο min

## Ο Αλγόριθμος Min-Max



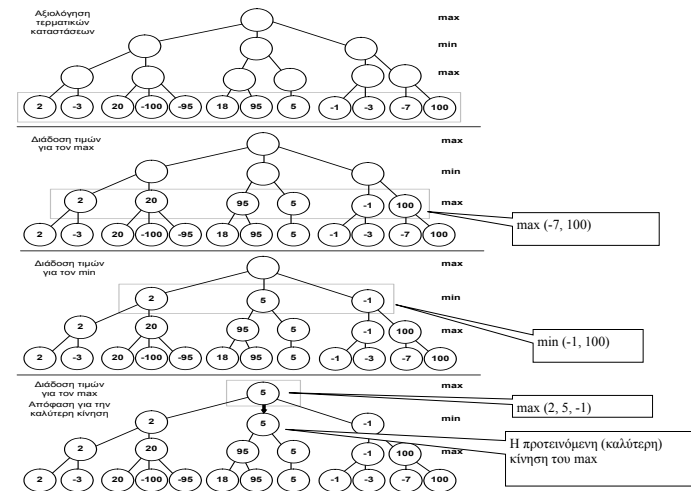
- Βήμα 1.** Εφάρμοσε τη συνάρτηση αξιολόγησης σε όλους τους κόμβους-φύλλα του δένδρου.
  - Βήμα 2.** Έως ότου η ρίζα του δένδρου αποκτήσει τιμή, επανέλαβε:
  - Βήμα 3.** Αρχίζοντας από τα φύλλα του δένδρου και προχωρώντας προς τη ρίζα, μετέφερε τις τιμές προς τους ενδιάμεσους κόμβους του δένδρου ως εξής:
    - Βήμα 3.1** Η τιμή κάθε κόμβου Max είναι η μέγιστη (maximum) των τιμών των κόμβων-παιδιών του.
    - Βήμα 3.2** Η τιμή κάθε κόμβου Min είναι η ελάχιστη (minimum) των τιμών των κόμβων-παιδιών του.
  - Βήμα 4.** Καλύτερη κίνηση είναι η κίνηση που οδηγεί στον κόμβο που έδωσε την πιο συμφέρουσα στη ρίζα τιμή (μέγιστη για το Max, ελάχιστη για το Min).
- Ο αλγόριθμος εγγυάται την πιο συμφέρουσα εξέλιξη μετά από κάποιες κινήσεις, έστω και αν ο αντίπαλος διαλέγει τις καλύτερες για αυτόν κινήσεις.

## Ο Αλγόριθμος Min-Max



- Κατά σύμβαση, ο παίκτης που βρίσκεται στη ρίζα θεωρείται πως είναι ο Max.
- Οι καταστάσεις-φύλλα του δένδρου ονομάζονται και τερματικές καταστάσεις, οι οποίες όμως δεν είναι απαραίτητα τελικές καταστάσεις, απλά αποτελούν το όριο της αναζήτησης.
- Οι τιμές των τερματικών καταστάσεων υπολογίζονται από τη συνάρτηση αξιολόγησης ενώ οι άλλες προκύπτουν από τη διάδοση αυτών.

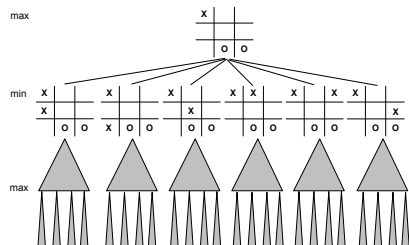
## Παράδειγμα Min-Max



## Εφαρμογή Min-Max στην Τρίλιζα



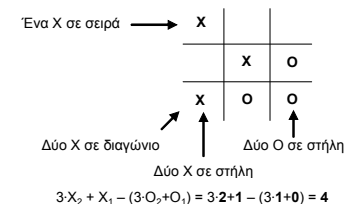
- Το παίγνιο τρίλιζα είναι ένα πρόβλημα με μικρό χώρο αναζήτησης, της τάξης των  $9!$  καταστάσεων.
- Όλες οι καταστάσεις μπορεί κάλλιστα να κωδικοποιηθούν σε κάποιο πρόγραμμα το οποίο σαν αποτέλεσμα δε θα χάνει ποτέ.
- Αποτελεί καλό παράδειγμα εφαρμογής του αλγορίθμου Minimax.
- Έστω ότι ο παίκτης Max είναι ένα πρόγραμμα που σημειώνει X και αντιμετωπίζει έναν άνθρωπο ή ένα άλλο πρόγραμμα, τον Min.
- Έστω ότι ο Max αποφασίζει να δει μέχρι βάθος τριών κινήσεων.
- Στο βάθος αυτό εφαρμόζει τη συνάρτηση αξιολόγησης για όλες τις καταστάσεις που έχουν προκύψει.



## Εφαρμογή Min-Max στην Τρίλιζα



- Μία συνάρτηση αξιολόγησης στην τρίλιζα θα μπορούσε να είναι η:
 
$$3 \cdot X_2 + X_1 - (3 \cdot O_2 + O_1)$$
  - $X_2$  ο αριθμός γραμμών, στηλών ή διαγωνίων με δύο X και χωρίς κανένα O.
  - $X_1$  ο αριθμός γραμμών, στηλών ή διαγωνίων με ένα X και χωρίς κανένα O.
  - $O_2$  ο αριθμός γραμμών, στηλών ή διαγωνίων με δύο O και χωρίς κανένα X.
  - $O_1$  ο αριθμός γραμμών, στηλών ή διαγωνίων με ένα O και χωρίς κανένα X.
- Προφανώς, θετικές τιμές επιστρέφουν οι καταστάσεις που ωφελούν αυτόν που σημειώνει X, ενώ αρνητικές τιμές αυτές που ωφελούν αυτόν που σημειώνει O.
- Για παράδειγμα, στο επόμενο Σχήμα απεικονίζεται μία κατάσταση της οποίας η συνάρτηση αξιολόγησης επιστρέφει 4, δηλαδή αυτός που σημειώνει το X έχει εμφανές πλεονέκτημα.

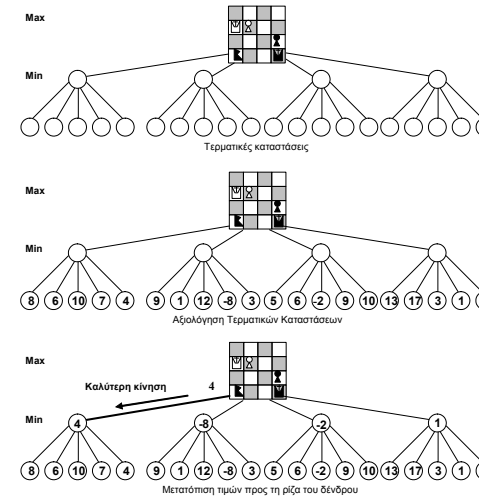


## Εφαρμογή Min-Max στο Σκάκι



- Το κύριο μέλημα των προγραμμάτων σκάκι είναι να αναζητήσουν το δένδρο του παιχνιδιού σε όσο το δυνατόν μεγαλύτερο βάθος. Αν υπήρχε η δυνατότητα να φτάσουν μέχρι τις τελικές καταστάσεις του παιχνιδιού, τότε τα προγράμματα θα ήταν ανίκητα.
- Μία συνάρτηση αξιολόγησης στο σκάκι
  - Υπεροχή κομματιών:** Κάθε κομμάτι έχει κάποια αξία, π.χ. Βασιλιάς=10, Άλογο=5, Πιόνι=1 κλπ. Η αξία όλων των κομματιών κάθε χρώματος προστίθεται.
  - Υπεροχή θέσης:** Κάθε κομμάτι που βρίσκεται στα 4 κεντρικά τετράγωνα παίρνει επιπλέον 2 πόντους.
  - Απειλές:** Για κάθε απειλή που προβάλλει ένας παίκτης παίρνει 3 επιπλέον πόντους, εκτός αν απειλεί το βασιλιά του άλλου παίκτη, οπότε παίρνει 20 πόντους.

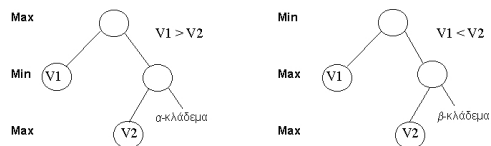
## Εφαρμογή Min-Max στο Σκάκι



## Ο Αλγόριθμος Alpha-Beta



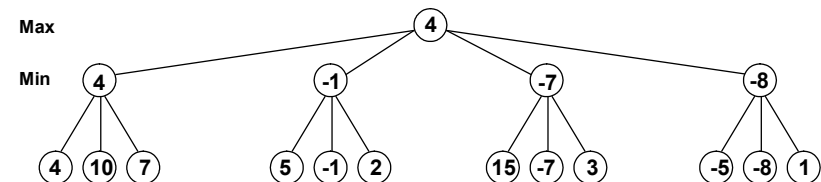
- Ο αλγόριθμος *Άλφα-Βήτα* (*Alpha-Beta - AB*) αποφεύγει την αξιολόγηση καταστάσεων που ικανοποιούν ορισμένες συνθήκες.
- Ο AB είναι όμοιος με τον Minimax, αλλά με κλάδεμα υποδένδρων, όπως κατά αντιστοιχία ο B&B με τον DFS.
- Το κλάδεμα που κάνει ο AB, όπως άλλωστε και ο B&B, δεν είναι ευρετικό διότι χρησιμοποιεί τα πραγματικά κόστη.



## Σύγκριση του AB με τον Min-Max



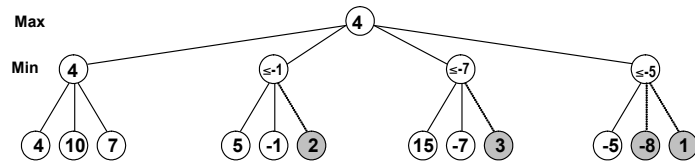
### Παράδειγμα Min-Max



## Σύγκριση του AB με τον Min-Max



### Παράδειγμα Alpha-Beta



#### Εξηγήσεις

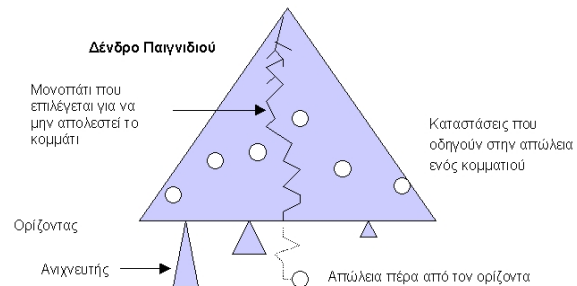
- Αξιολογούνται οι πρώτοι αριστεροί κόμβοι (4, 10, 7) και δίνουν την τιμή 4 στον κόμβο πατέρα.
- Η αξιολόγηση προχωρά στο δεύτερο κλαδί και βρίσκει 5 και -1 οπότε και σταματά γιατί η τιμή του είναι μικρότερη από τη μεγαλύτερη τιμή του ανωτέρου επιπέδου (δηλ. το 4).
- Όμοια γίνεται και στους επόμενους κόμβους.

## Σύγκριση του AB με τον Min-Max



- Κατά προσέγγιση ο AB εξετάζει τερματικούς κόμβους, όπου  $N$  είναι οι τερματικοί κόμβοι που εξετάζει ο αλγόριθμος Minimax.
- Η απόδοσή του βελτιώνεται με διάφορες μεθόδους, όπως:
  - Ευρετικό κλάδεμα του δένδρου παιχνιδιού.
  - Δυναμική αντί στατικής συνάρτησης αξιολόγησης.
  - Αποθήκευση τιμών των τερματικών καταστάσεων (transposition tables).
  - Προκαθορισμένες κινήσεις (χωρίς αναζήτηση) σε αρχικές και τελικές φάσεις του παιχνιδιού (*Openings, End Game moves*).

## Το Πρόβλημα του Ορίζοντα



#### Μειονεκτήματα:

Το φαινόμενο του ορίζοντα (horizon effect)  
Αντιμετωπίζεται με ανιχνευτές (scouts)