Τεχνητή Νοημοσύνη

Ενότητα 2: Αναζήτηση (Search)

Αν. καθηγητής Στεργίου Κωνσταντίνος
kstergiou@uowm.gr

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

CC BY NC ND
Ευρωπαϊκή Ένωση
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης
ΕΣΠΑ 2007-2013

Άδειες Χρήσης

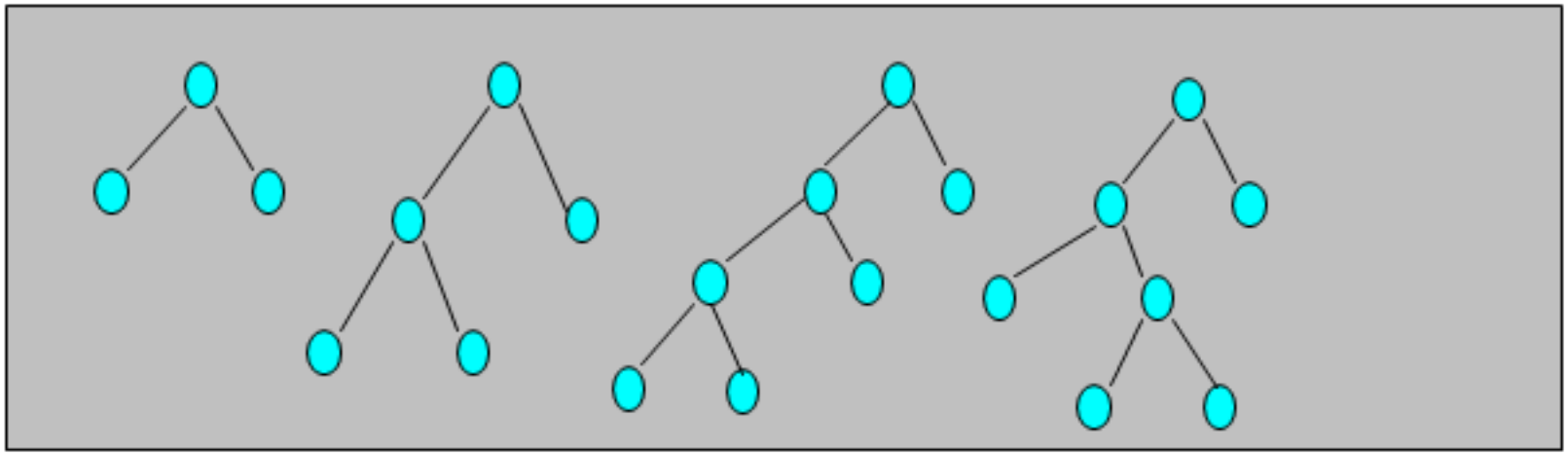
- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Αναζήτηση (search)



Αλγόριθμοι και Πολυπλοκότητα

- Ας υποθέσουμε ότι έχουμε δύο διαφορετικούς αλγόριθμους για την επίλυση ενός προβλήματος.
  Πως θα βρούμε ποιος είναι ο καλύτερος;
  - Ποιος τρέχει πιο γρήγορα;
  - Ποιος καταλαμβάνει λιγότερη μνήμη;
- **Δύο τρόποι να απαντήσουμε:**
  - Πειράματα.
  - Μαθηματική ανάλυση.
- Πως μπορεί να γίνει μαθηματική ανάλυση της χρονικής / χωρικής πολυπλοκότητας ενός αλγορίθμου;

Χρονική Πολυπλοκότητα (1/2)

- Συνήθως η χρονική πολυπλοκότητα προσδιορίζεται ως συνάρτηση του μεγέθους των δεδομένων *(input).*

- **Ορισμός:** Η **χρονική πολυπλοκότητα χειρότερης περίπτωσης** *(worst-case time complexity)* ενός αλγόριθμου είναι η συνάρτηση **T(*n*)**, η οποία είναι η μέγιστη, για όλα τα inputs μεγέθους *n*, των αθροισμάτων χρόνου που ξοδεύεται από κάθε **βασική πράξη** *(primitive operation).*

- Οι βασικές πράξεις στους αλγόριθμους είναι αναθέσεις τιμών, συγκρίσεις, εντολές εισόδου/εξόδου, εντολές return, εντολές διάθεσης μνήμης, κτλ.

Χρονική Πολυπλοκότητα (2/2)

- Μπορούμε να υποθέτουμε *(για λόγους απλότητας)* ότι κάθε βασική πράξη εκτελείται σε μια μονάδα χρόνου.

- Η θεωρία δεν αλλάζει αν κάθε βασική πράξη “κοστίζει” c μονάδες χρόνου ή αν οι διαφορετικές βασικές πράξεις έχουν διαφορετικό κόστος σε μονάδες χρόνου.

- Υπάρχει και πολυπλοκότητα μέσης περίπτωσης:
  - Απαιτεί υποθέσεις για πιθανοτική κατανομή του input.

Μέγεθος input

- Με ποια παράμετρο μπορούμε να αναπαραστήσουμε το μέγεθος του input ενός αλγορίθμου;

- **Παραδείγματα:**
  - Διάταξη λίστας ακέραιων αριθμών:
    - ✓ **Το μέγεθος της λίστας.**
  - Εύρεση του αν ένας γράφος έχει κύκλους:
    - ✓ **Το πλήθος των κόμβων και ακμών του γράφου.**
  - Ανάθεση μαθημάτων σε διδάσκοντες:
    - ✓ **Το πλήθος των μαθημάτων και των διδασκόντων.**

Παράδειγμα (1/2)

- **Function** Summation (*sequence, n*=Length(*sequence*))
  returns an integer
  sequence: array of integers, *sum*: integer
  sum ← 0
  for i ← 0 to *n* **do**
  sum ← *sum* + *sequence*[*i*]
  end
  return *sum*

- **Input size:** *n* the length of the sequence.
- **Time complexity:** $T(n) = c_1 n + c_2$.

Παράδειγμα (2/2)

- **Function** Find_13(*sequence, n*=Length(*sequence*)) **returns** the position in the sequence containing the number 13

  sequence: array of integers, *pos*: integer.

  pos ← 1

  while *sequence*[*pos*]≠13 **and** *pos*≤Length(*sequence*) **do**

  pos ← *pos* + 1

  end

  if *pos* ≤ Length(*sequence*) **then return** *pos*

  else return -1
- **Input size:** *n* the length of the sequence.
- **Time complexity:** $T(n) = c_1 n + c_3$.

Ανάλυση Αλγορίθμων

- Όταν υπολογίζουμε τη πολυπλοκότητα χειρότερης περίπτωσης ενός αλγόριθμου αυτό που μας ενδιαφέρει είναι ο **ρυθμός αύξησης** ή **η τάξη μεγέθους αύξησης** του χρόνου.
- Για παράδειγμα στην πολυπλοκότητα $T(n)=3n^2+5n+2$ ο **πιο σημαντικός όρος** είναι το $3n^2$.
  - Για μεγάλες τιμές του n οι όροι $5n$ και 2 είναι σχετικά ασήμαντοι σε σύγκριση με τον $3n^2$.
  - Μπορούμε επίσης να αγνοήσουμε τη σταθερά 3 και να θεωρήσουμε το n^2 ως τον πιο σημαντικό όρο στο $T(n)$.
- **Ασυμπτωτική ανάλυση** αλγορίθμων.

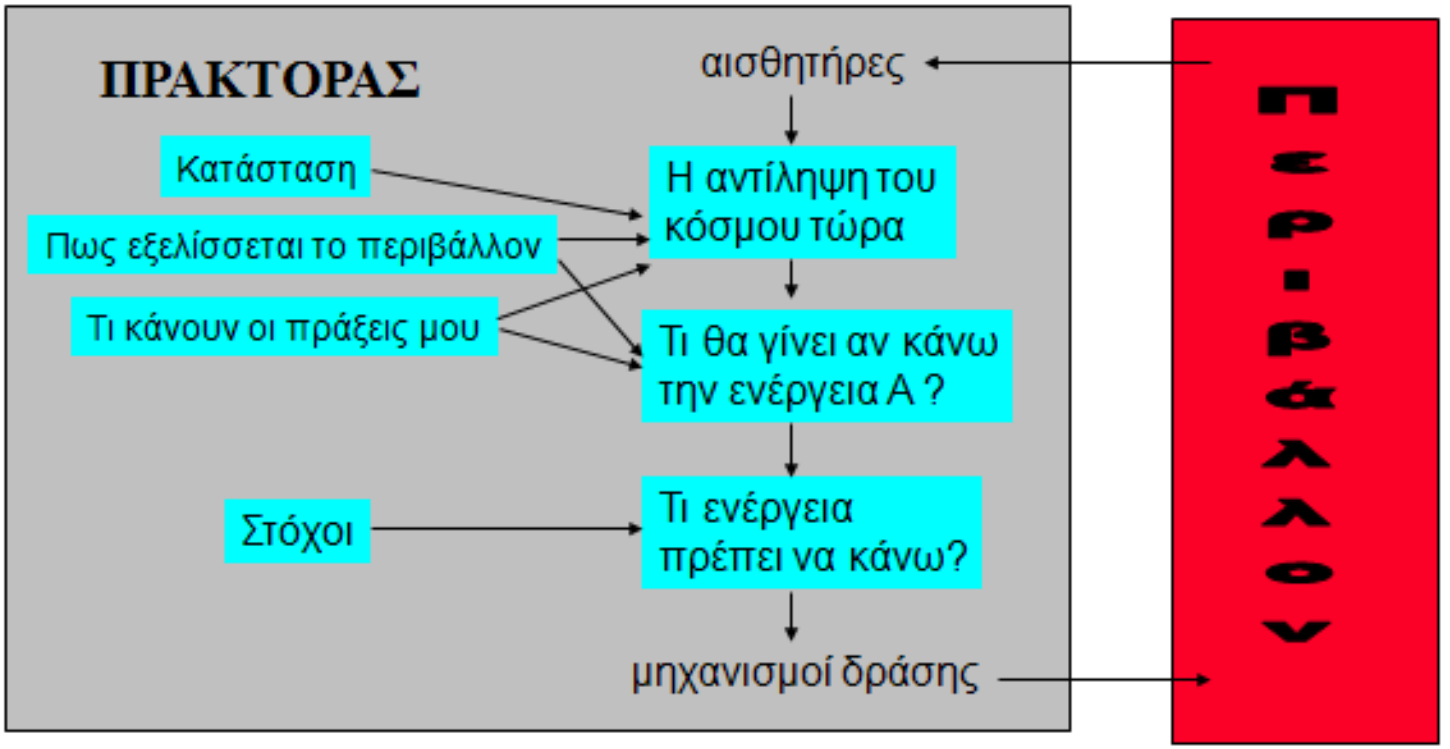
Συμβολισμός Ο

- Ο συμβολισμός Ο (μεγάλο όμικρον) είναι πολύ χρήσιμος.
- **Ορισμός:** Η συνάρτηση Τ(n) είναι (τάξης) Ο(f(n)) αν υπάρχει μια σταθερά k τέτοια ώστε $T(n) \leq kf(n)$ για όλα τα $n>n_0$.
- **Παραδείγματα:**
  - $2n^2 + 5n$ είναι $O(n^2)$ γιατί $2n^2 + 5n \leq 2n^2 + 5n^2 \leq 7n^2$ για όλα τα n.
  - Γενικά κάθε πολυώνυμο της μορφής $a_n x^n + a_{n-1}x^{n-1} +\ldots+a_1 x + a_0$,όπου $a_n > 0$ είναι $O(x^n)$.
  - $2^n + n^3$ $O(2^n)$ γιατί $2^n + n^3 < 2^n + 2^n \leq 2^{n+1}$ για όλα τα $n>9$.

Χωρική Πολυπλοκότητα

- Η χωρική πολυπλοκότητα προσδιορίζεται επίσης ως συνάρτηση του μεγέθους των δεδομένων (input).
- **Ορισμός:** Η **χωρική πολυπλοκότητα χειρότερης περίπτωσης** (worst-case space complexity) ενός αλγόριθμου είναι η συνάρτηση S(*n*), η οποία είναι η μέγιστη, για όλα τα inputs μεγέθους *n*, των αθροισμάτων χώρου μνήμης από κάθε **βασική πράξη** (primitive operation).
- Αν είναι εκθετική υπάρχει σοβαρό πρόβλημα!

Πράκτορες βασισμένοι σε στόχους *(Goal-based Agents)*



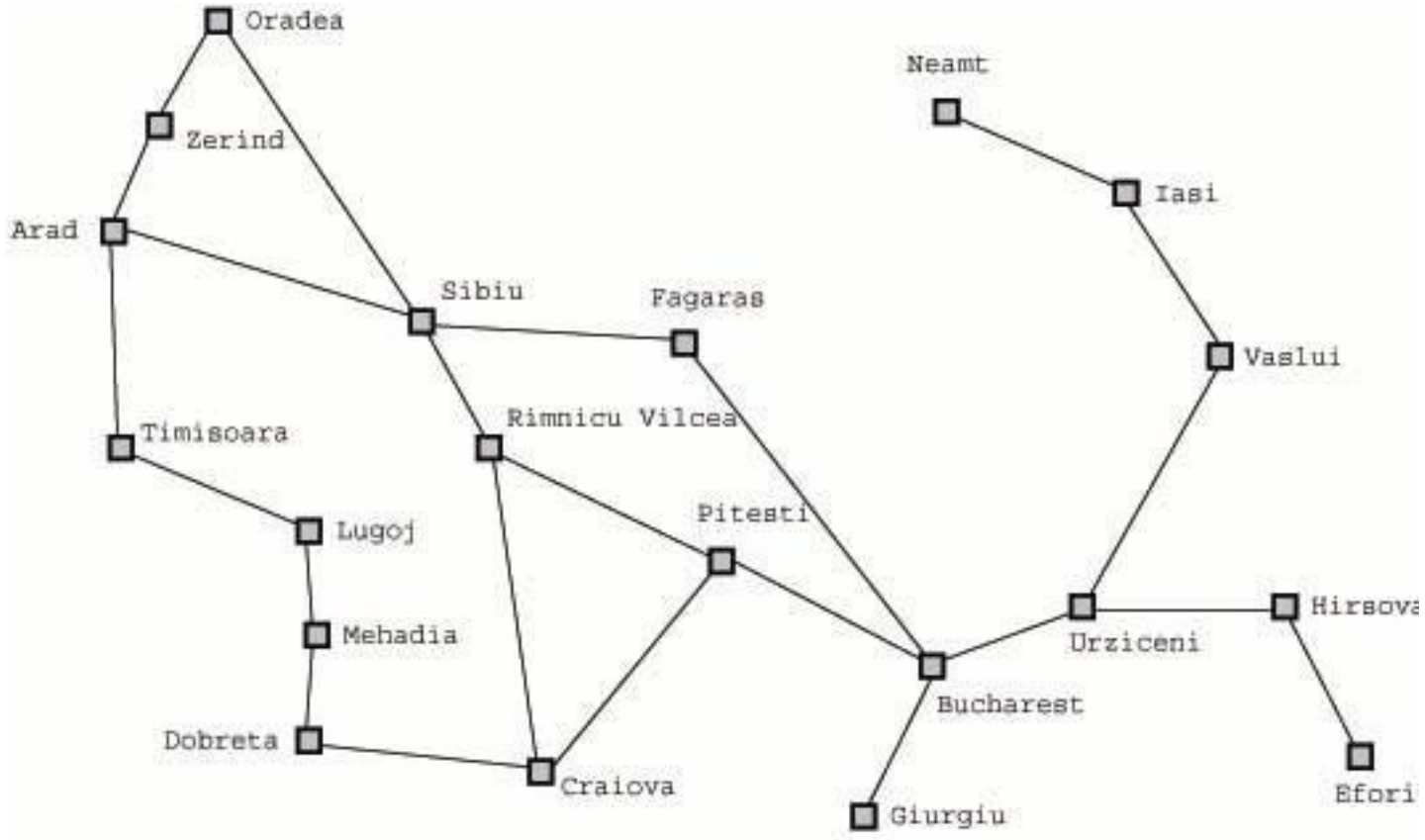
Επίλυση Προβλημάτων με Αναζήτηση

- **Πράκτορες επίλυσης προβλημάτων** *(Problem Solving Agents).*
- **Προβλήματα Αναζήτησης** *(Search Problems).*
- **Στρατηγικές Τυφλής Αναζήτησης** *(Blind Search Strategies).*
- **Στρατηγικές Ευριστικής Αναζήτησης** *(Heuristic Search Strategies).*
- **Προβλήματα Ικανοποίησης Περιορισμών** *(Constraint Satisfaction Problems).*

Πράκτορες Επίλυσης Προβλημάτων (1/3)

- Οι **πράκτορες επίλυσης προβλημάτων** είναι πράκτορες που βασίζονται σε στόχους. Λειτουργούν πράττοντας τις ακόλουθες διεργασίες:
  - **Σχηματισμός στόχων** *(goal formulation):*
    - Αποφάσισε τι σκοπεύεις να κάνεις.
  - **Σχηματισμός του προβλήματος** *(problem formulation):*
    - Αποφάσισε τι ενέργειες και καταστάσεις απαιτούνται για να επιτευχθεί ο στόχος.
  - **Αναζήτηση** *(Search):*
    - Βρες μια ακολουθία ενεργειών που να επιτυγχάνει το στόχο *(δηλ. μια λύση).*
  - **Εκτέλεση** *(Execution):*
    - Εκτέλεσε την επιλεγμένη ακολουθία ενεργειών.

Παράδειγμα: Εύρεση διαδρομών στη Ρουμανία



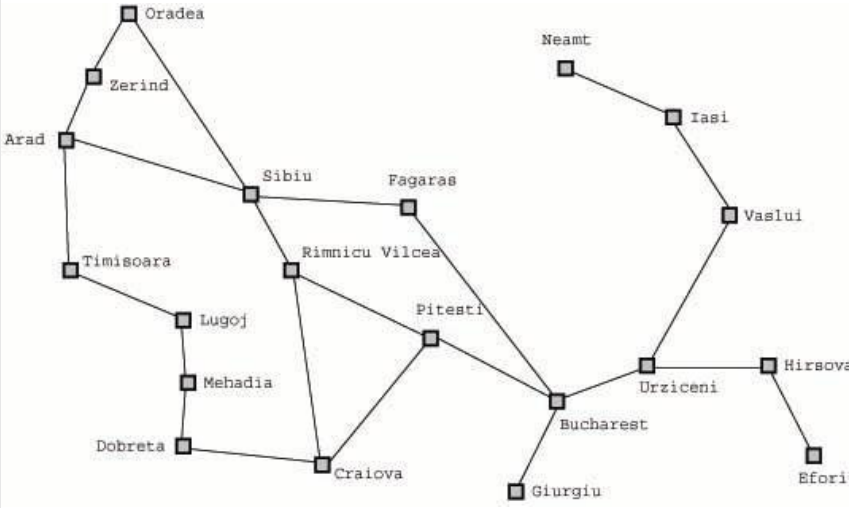
Πράκτορες Επίλυσης Προβλημάτων (2/3)

function SimpleProblemSolvingAgent (*p*) **returns** an *action*
    static *aseq*, *state*, *goal*, *problem*
    state ← UpdateState (*state,p*)
    if *aseq* is empty **then**
        goal ← FormulateGoal (*state*)
        problem ← FormulateProblem (*state*, *goal*)
        aseq ← Search (*problem*)
    endif
    action ← First (*aseq)*
    aseq ← Remainder (*aseq)*
    return action

Πράκτορες Επίλυσης Προβλημάτων (3/3)

π.χ. στην εύρεση διαδρομών στη Ρουμανία.

- Σε τι είδους περιβάλλον μπορεί να δράσει ένας πράκτορας επίλυσης προβλημάτων;
  - **στατικό.**
    - γιατί;
  - **πλήρως παρατηρήσιμο.**
    - γιατί;
  - **διακριτό.**
    - γιατί;
  - **αιτιοκρατικό.**
    - γιατί;
  - **μονοπρακτορικό.**



Προβλήματα Αναζήτησης (1/3)

- Τα βασικά στοιχεία ενός προβλήματος αναζήτησης είναι:
  - **Η αρχική κατάσταση.**
  - **Το σύνολο των διατιθέμενων ενεργειών** *(actions).*
  - Για να προσδιορίσουμε τις διαθέσιμες ενέργειες συχνά χρησιμοποιούμε μια συνάρτηση **διαδοχής** *(successor function)* ***Succ***, η οποία για κάθε δεδομένη κατάσταση *x* επιστρέφει ένα σύνολο διατεταγμένων ζευγών *(action,* ***successor state****).*
  - Αυτό μας λέει ποιες ενέργειες είναι δυνατές στην κατάσταση *x* και σε ποιες καταστάσεις μπορούμε να βρεθούμε εκτελώντας αυτές τις ενέργειες.
  - Η αρχική κατάσταση και η συνάρτηση διαδοχής ορίζουν τον **χώρο καταστάσεων** *(state space)* σε ένα πρόβλημα αναζήτησης. Δηλαδή το σύνολο όλων των καταστάσεων στις οποίες μπορούμε να βρεθούμε ξεκινώντας από την αρχική με οποιαδήποτε ακολουθία ενεργειών.
  - Ένα **μονοπάτι** *(path)* στο χώρο καταστάσεων είναι μια ακολουθία καταστάσεων που συνδέεται με μια ακολουθία ενεργειών.

Προβλήματα Αναζήτησης (2/3)

- **Ο στόχος που πρέπει να επιτευχθεί.**
  - Ο στόχος είναι ένα σύνολο από καταστάσεις του περιβάλλοντος που ονομάζονται **καταστάσεις στόχου** *(goal states)*. Οι στόχοι μπορούν να προσδιοριστούν έμμεσα χρησιμοποιώντας ένα **τεστ στόχου** *(goal test),* δηλ. Ένα τεστ που μπορούμε να εφαρμόσουμε σε μια κατάσταση για να ελέγξουμε αν είναι κατάσταση στόχου
- **Μια συνάρτηση κόστους μονοπατιού**.
  - Η συνάρτηση αυτή *(αναπαρίσταται ως g)* αναθέτει ένα αριθμητικό κόστος σε κάθε μονοπάτι. Το κόστος ενός μονοπατιού συνήθως είναι το άθροισμα του κόστους των μεμονωμένων ενεργειών σε αυτό το μονοπάτι
  - Το κόστος μιας ενέργειας *a* που μας μεταφέρει από την κατάσταση *x* στην κατάσταση *y* συνήθως αναπαρίσταται ως *c* (*x*, *a*, *y*).
- Μια **λύση** σε ένα πρόβλημα αναζήτησης είναι ένα μονοπάτι από την αρχική κατάσταση σε μια κατάσταση στόχου. Μια λύση είναι **βέλτιστη** *(optimal)* αν έχει το μικρότερο κόστος από όλες τις λύσεις.

Διατύπωση Προβλημάτων

- Η διατύπωση του προβλήματος εύρεσης διαδρομής παραλείπει πολλές απόψεις του πραγματικού κόσμου:
  - Η περιγραφή κατάστασης *In (Arad)* αγνοεί πράγματα όπως το ακριβές σημείο στο Arad όπου βρισκόμαστε, τη διαθέσιμη βενζίνη, τις καιρικές συνθήκες, τους πιθανούς συνεπιβάτες, κτλ.

- Η διαδικασία παράλειψης λεπτομερειών από μια αναπαράσταση ονομάζεται **αφαίρεση** *(abstraction).*
  - αφαίρεση χρησιμοποιούμε στην περιγραφή *καταστάσεων κι ενεργειών:*
    - ✓ δε μας απασχολούν ενέργειες όπως “στρίψε το τιμόνι”, δε μας απασχολεί ο χρόνος εκτέλεσης μιας ενέργειας, ούτε τα δευτερεύοντα αποτελέσματα της.

- Ποιο είναι το κατάλληλο επίπεδο αφαίρεσης;
  - **εγκυρότητα** και **παράλειψη λεπτομερειών** ταυτόχρονα
    - ✓ μια αφαίρεση είναι ***έγκυρη*** αν μπορούμε να αναπτύξουμε κάθε αφηρημένη λύση σε μια λύση στον πραγματικό λεπτομερή κόσμο.

Προβλήματα Αναζήτησης (3/3)

- **Πως μετράμε την απόδοση μιας μεθόδου επίλυσης προβλημάτων αναζήτησης;**
  - Βρίσκει λύση;
  - Είναι καλή λύση;(Έχει χαμηλό κόστος;)
  - Πόσο χρόνο κάνει και πόση μνήμη καταναλώνει για να βρει λύση;
- **Ο πράκτορας πρέπει να αποφασίσει πόσο χρόνο θα ξοδέψει στην αναζήτηση της λύσης και πόσο στην εκτέλεση της:**
  - όταν ο χώρος αναζήτησης είναι μικρός είναι εύκολο να βρεθεί η καλύτερη λύση,
  - όταν ο χώρος αναζήτησης είναι μεγάλος μπορεί να αρκεί μια απλώς καλή λύση.

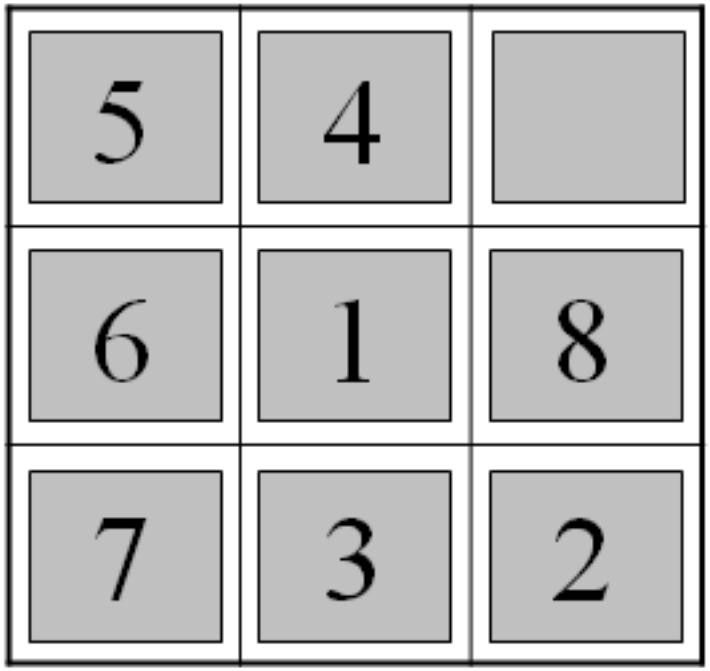
Ένα Παράδειγμα – Εύρεση Διαδρομής

- Το πρόβλημα εύρεσης μιας διαδρομής από το Arad στο Bucharest μπορεί να διατυπωθεί ως εξής:
  - Οι **καταστάσεις** προσδιορίζουν την πόλη στην οποία είμαστε.
    - π.χ. *In* (Arad) - **Ο χώρος καταστάσεων ταυτίζεται με τον γράφο!**
  - Η μόνη διαθέσιμη **ενέργεια** είναι η *GoTo.*
    - π.χ. *GoTo* (Sibiu).
  - Για κάθε πόλη η **συνάρτηση διαδοχής** δίνει ένα σύνολο ζευγών (*GoTo*(x), *In*(x)).

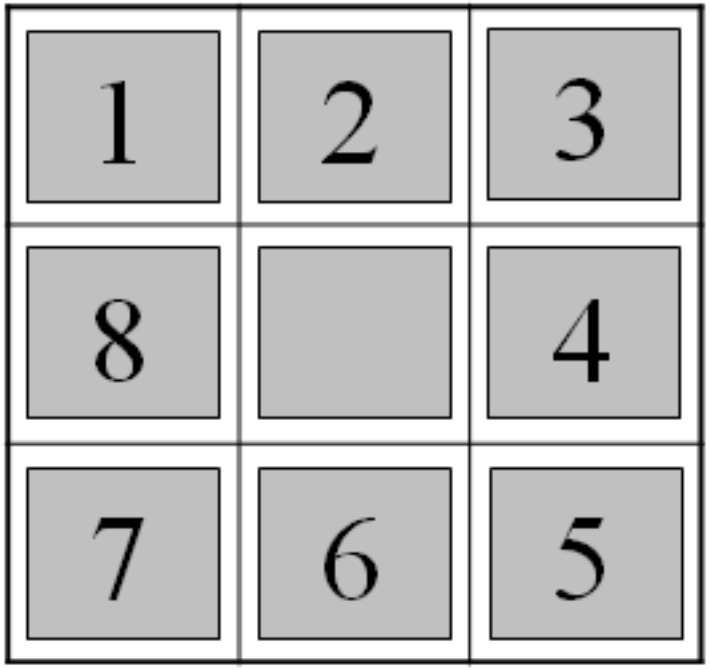
π.χ.*S ucc* (Arad)= {(*GoTo*(Sibiu),*In*(Sibiu)), (*GoTo*(Timisoara),*In*(Timisoara)), (*GoTo*(Zerind),*In*(Zerind))}.

  - Η **αρχική κατάσταση** είναι *In*(Arad). Η **κατάσταση στόχου** είναι *In*(Bucharest).
  - Το **κόστος μονοπατιού** μπορεί να είναι η χιλιομετρική απόσταση.

Το 8-puzzle (1/2)



Αρχική Κατάσταση



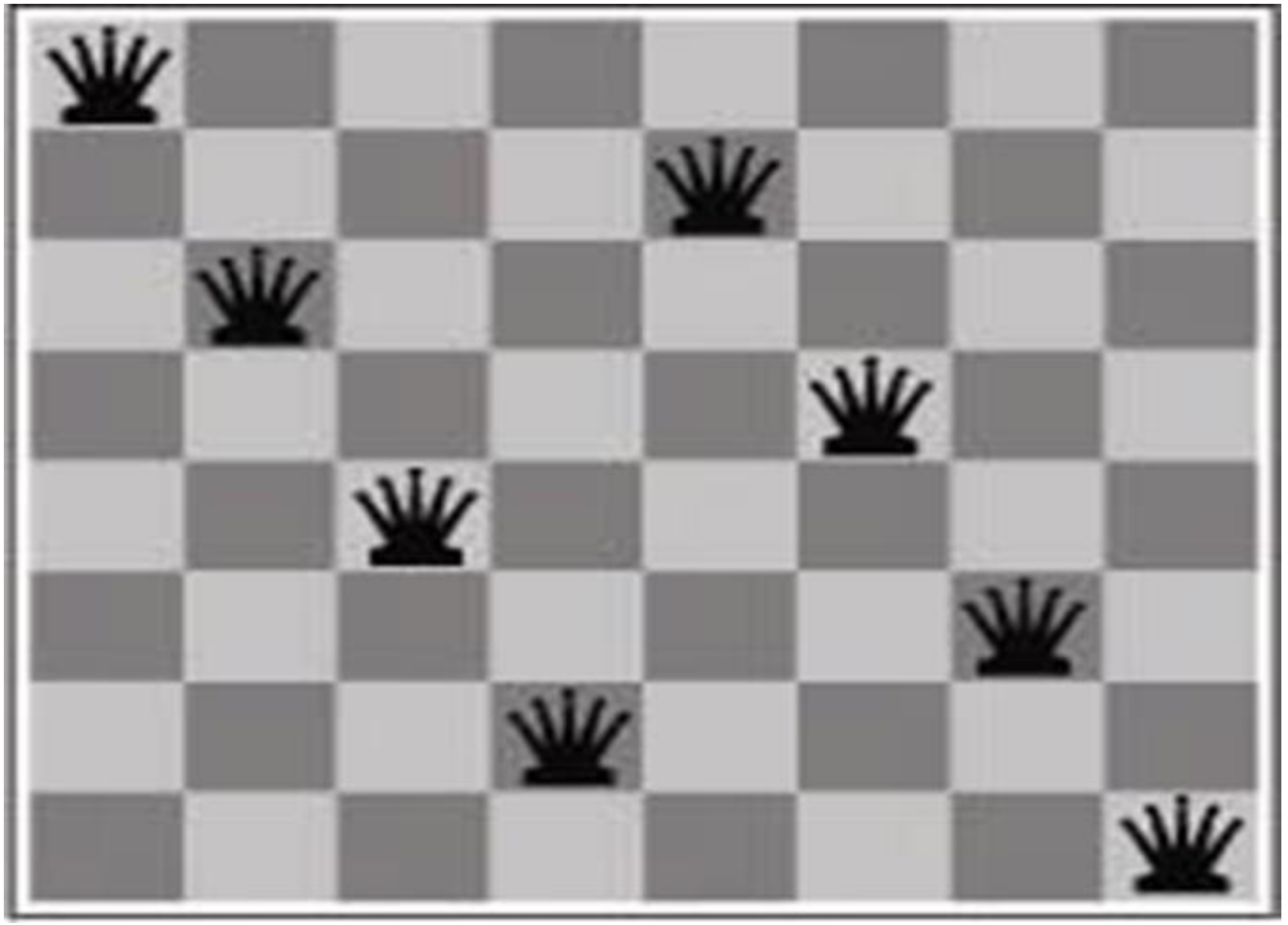
Κατάσταση Στόχου

Το 8-puzzle (2/2)

- Επίσημη περιγραφή:
  - **Καταστάσεις:**
    - ✓Η κάθε κατάσταση περιγράφεται προσδιορίζοντας την τοποθεσία κάθε αριθμού καθώς και του κενού.
  - **Ενέργειες:**
    - ✓ Το κενό μετακινείται Π (πάνω), Κ (κάτω), Δ (δεξιά), Α (αριστερά).
  - **Κατάσταση Στόχου:**
    - ✓ Το κενό στη μέση, οι αριθμοί σε διάταξη σύμφωνα με τους δείκτες του ρολογιού.
  - **Κόστος Μονοπατιού:**
    - ✓ Το μήκος του μονοπατιού, δηλ. πόσες μετακινήσεις γίνονται.

- Τι μορφή έχει ο χώρος καταστάσεων;

Το πρόβλημα των 8 βασιλισσών (8-queens) (1/3)



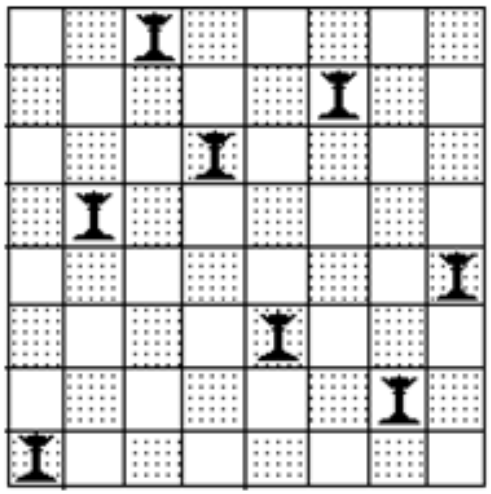
Το πρόβλημα των 8 βασιλισσών (8-queens) (2/3)

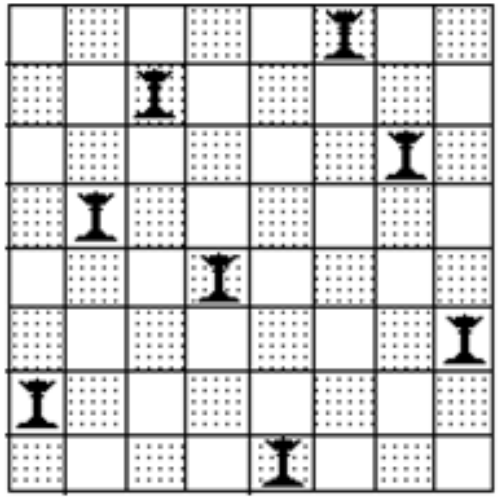
- Επίσημη περιγραφή:
  - **Καταστάσεις:**
    - Κάθε τοποθέτηση από 0 ως 8 βασιλισσών στη σκακιέρα περιγράφει μια κατάσταση.
  - **Ενέργειες:**
    - Πρόσθεσε μια βασίλισσα στη σκακιέρα.
  - **Τεστ Στόχου:**
    - Υπάρχουν 8 βασίλισσες στη σκακιέρα και καμία δε μπορεί να επιτεθεί σε άλλη.
  - **Κόστος Μονοπατιού:**
    - 0 (γιατί ;)
- Υπάρχουν $64x63x...x57 \approx 3x10^4$ καταστάσεις.

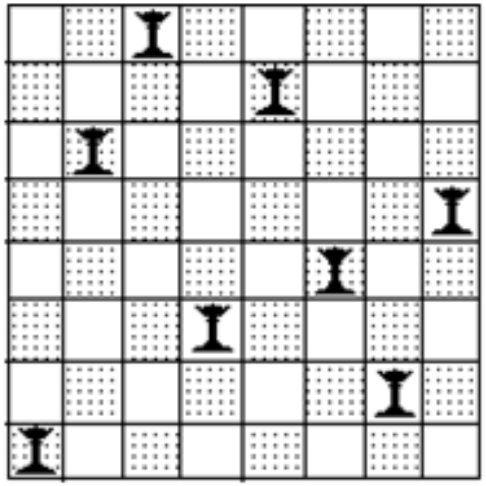
Το πρόβλημα των 8 βασιλισσών (8-queens) (3/3)

- Εναλλακτική περιγραφή:
  - **Καταστάσεις:**
    - ✓ Κάθε τοποθέτηση από 0 ως 8 βασιλισσών στη σκακιέρα έτσι ώστε καμία να μην μπορεί να επιτεθεί σε άλλη περιγράφει μια κατάσταση.
  - **Ενέργειες:**
    - ✓ Πρόσθεσε μια βασίλισσα στην πιο αριστερή άδεια στήλη έτσι ώστε να μη μπορεί να της επιτεθεί καμία άλλη.
  - **Τεστ Στόχου:**
    - ✓ Υπάρχουν 8 βασίλισσες στη σκακιέρα και καμία δε μπορεί να επιτεθεί σε άλλη.
  - **Κόστος Μονοπατιού:**
    - ✓ 0 (γιατί ;)
- Υπάρχουν 2057 καταστάσεις.

Μερικές Λύσεις







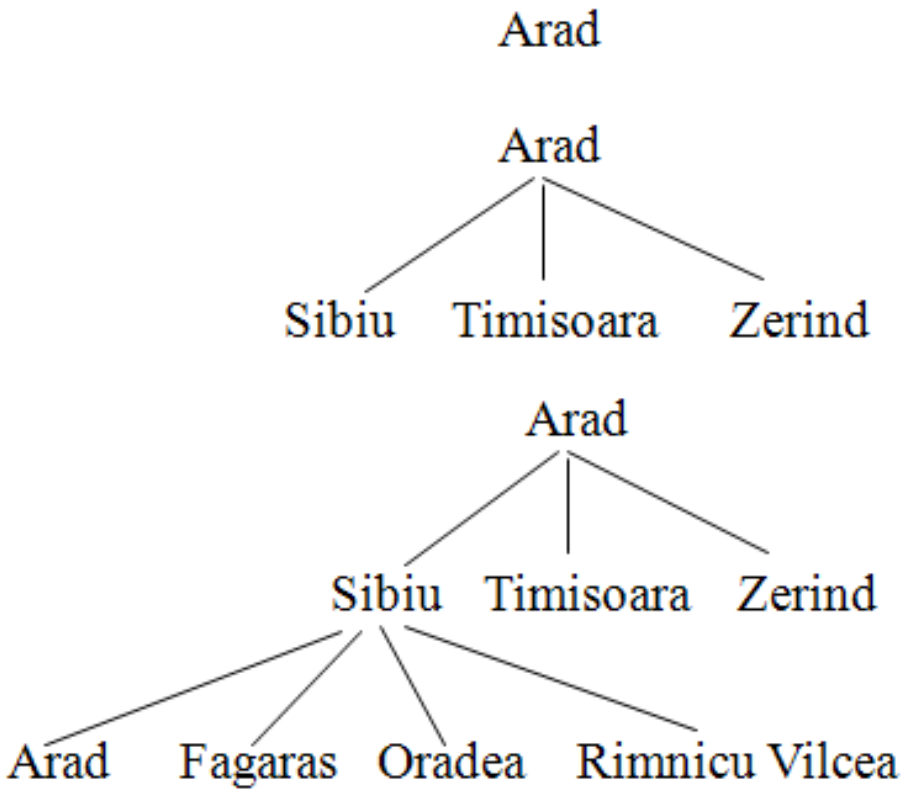
- Υπάρχουν πολλές περισσότερες λόγω **συμμετρίας!**

Προβλήματα Αναζήτησης στον Πραγματικό Κόσμο

- **Εύρεση Διαδρομών** (route finding).
- **Προβλήματα Περιοδείας** (e.g. traveling salesman).
- **Καθοδήγηση Robot** (robot navigation).
- **Σχεδιασμός Πρωτεϊνών** (protein design).
- **VLSI σχεδιασμός.**
- **Ανάθεση εργασιών/πόρων σε εργαζόμενους** (resource allocation).
- **Βελτιστοποίηση Επερωτήσεων σε ΣΔΒΔ** (query optimization).
- **Αναζήτηση στο Internet** (internet search engines).
- **Κατασκευή προγράμματος εξεταστικής** (timetabling).

Αναζήτηση Λύσεων (1/3)

- **Α) Αρχική κατάσταση.**
- **Β) Επεκτείνοντας το Arad.**
- **Γ) Επεκτείνοντας το Sibiu.**



Αναζήτηση Λύσεων (2/3)

- **Γενικά σχόλια:**
  - Η εύρεση λύσης επιτυγχάνεται με αναζήτηση μέσα στο χώρο καταστάσεων.
    - Αποθηκεύουμε και επεκτείνουμε (expand) ένα σύνολο μερικών λύσεων.
  - Η επιλογή της επόμενης κατάστασης που θα επεκτείνουμε εξαρτάται από τη **στρατηγική** (δηλ. **αλγόριθμο**) **αναζήτησης** (search strategy).
  - Κατά τη διαδικασία αναζήτησης χτίζεται ένα **δέντρο αναζήτησης** (search tree) που περιλαμβάνει καταστάσεις του χώρου καταστάσεων.
  - *Είναι σημαντικό να διαφοροποιηθούν οι έννοιες δέντρο αναζήτησης και χώρος καταστάσεων.*
    - Παράδειγμα;

Αναζήτηση Λύσεων (3/3)

function TreeSearch (*problem, strategy*)
returns a *solution* or *failure*
initialize the search tree using the initial state of *problem*
loop do
if there are no candidate states for expansion **then**
return *failure*
else choose a leaf node for expansion according to *strategy*
if the node contains a goal state
return the corresponding solution
else expand the node and add the resulting nodes to the search tree
end

Κόμβοι του δέντρου αναζήτησης

- Οι **κόμβοι** του δέντρου αναζήτησης *(search tree nodes)* μπορούν να αναπαρασταθούν με μια δομή δεδομένων με 5 συστατικά:
  - **ΚΑΤΑΣΤΑΣΗ** (STATE):
    - ✓ η κατάσταση στην οποία αντιστοιχεί ο κόμβος.
  - **ΠΑΤΡΙΚΟΣ ΚΟΜΒΟΣ** (PARENT NODE):
    - ✓ ο κόμβος που δημιούργησε τον συγκεκριμένο κόμβο.
  - **ΕΝΕΡΓΕΙΑ** (ACTION):
    - ✓ η ενέργεια που εφαρμόστηκε για τη δημιουργία του κόμβου.
  - **ΚΟΣΤΟΣ ΜΟΝΟΠΑΤΙΟΥ** (PATH COST):
    - ✓ το κόστος του μονοπατιού από την αρχική κατάσταση ως τον συγκεκριμένο κόμβο.
  - **ΒΑΘΟΣ** (DEPTH):
    - ✓ το πλήθος των κόμβων στο μονοπάτι από τη ρίζα ως τον συγκεκριμένο.

Το Μέτωπο Αναζήτησης *(fringe or frontier)*

- Το σύνολο των κόμβων που περιμένουν επέκταση ονομάζεται **μέτωπο αναζήτησης**. Μπορεί να υλοποιηθεί με μια ουρά *(queue)* με τις παρακάτω λειτουργίες:
  - **MakeQueue** (*Elements*):
    - ✓ Δημιουργεί μια ουρά με τα δεδομένα στοιχεία.
  - **Empty;** (*Queue*):
    - ✓ Επιστρέφει true αν δεν υπάρχουν στοιχεία στην ουρά.
  - **RemoveFron**t (*Queue*):
    - ✓ Βγάζει το πρώτο στοιχείο της ουράς.
  - **Queuing-Fn** (*Elements*, *Queue*):
    - ✓ Συνάρτηση που εισάγει ένα σύνολο στοιχείων στην ουρά.
    - ✓ Διαφορετικές παραλλαγές της συνάρτησης δίνουν διαφορετικούς αλγόριθμους αναζήτησης.

Γενικός αλγόριθμος αναζήτησης (1/2)

```
function TreeSearch (problem, Queuing-Fn)
returns a solution or failure
    fringe ← MakeQueue(MakeNode(ininitialState[problem]))
    loop do
       if fringe is empty then return failure
       node ←  RemoveFront(fringe)
       if GoalTest[problem] applied to State[node] succeeds
           then return node
       fringe ← Queuing-Fn(fringe,Expand(node, problem))
    end
```

- Η συνάρτηση Expand υπολογίζει τους κόμβους που παράγονται από μια επέκταση.

Αλγόριθμοι Αναζήτησης

- Θα ασχοληθούμε με δύο είδη αλγορίθμων αναζήτησης:
  - **Αλγόριθμοι τυφλής αναζήτησης** ή **μη ενημερωμένοι αλγόριθμοι** *(blind or uninformed search algorithms).*
  - **Ευρετικοί** ή **ενημερωμένοι αλγόριθμοι** *(heuristic or informed search algorithms).*

- Κριτήρια αξιολόγησης αλγορίθμων:
  - **Πληρότητα** (Completeness).
  - **Δυνατότητα εύρεσης βέλτιστης λύσης** (Optimality).
  - **Χρονική Πολυπλοκότητα** (Time Complexity).
  - **Χωρική Πολυπλοκότητα** (Space Complexity).

Πολυπλοκότητα Αλγορίθμων Αναζήτησης (1/2)

- Σε περίπτωση που το γράφημα χώρου καταστάσεων είναι επακριβώς ορισμένο (π.χ. χάρτης Ρουμανίας) *το μέγεθος* του είναι το μέτρο υπολογισμού της πολυπλοκότητας.
- Αν ο χώρος καταστάσεων αναπαριστάται έμμεσα από την αρχική κατάσταση και τη συνάρτηση διαδοχής, χρησιμοποιούνται οι εξής ποσότητες:
  - παράγοντας διακλάδωσης ***b.***
  - βάθος του πιο ρηχού κόμβου στόχου ***d.***
  - μέγιστο μήκος διαδρομής στο χώρο καταστάσεων ***m.***
- Ο χρόνος μετρείται βάση του *πλήθους των κόμβων που παράγονται* και ο χώρος με βάση το *πλήθος των κόμβων που αποθηκεύονται στη μνήμη.*

Πολυπλοκότητα Αλγορίθμων Αναζήτησης (2/2)

- Για να εκτιμήσουμε την αποτελεσματικότητα ενός αλγορίθμου αναζήτησης εξετάζουμε:
  - το ***κόστος αναζήτησης*** (που εξαρτάται από την πολυπλοκότητα).
  - ή και το ***κόστος λύσης*** (δηλ. το κόστος διαδρομής της λύσης που βρέθηκε).
    - Σε αυτή την περίπτωση κοιτάμε το ***ολικό κόστος.***
  - Η βαρύτητα που προσδίδουμε στο κάθε κόστος εξαρτάται από την εφαρμογή.
    - Μερικές φορές θέλουμε μια πολύ καλή (ή και τη βέλτιστη) λύση χωρίς να μας ενδιαφέρει πολύ πόσο χρόνο θα ξοδέψουμε για να τη βρούμε.
    - Άλλες φορές θέλουμε μια (οποιαδήποτε) λύση γρήγορα.

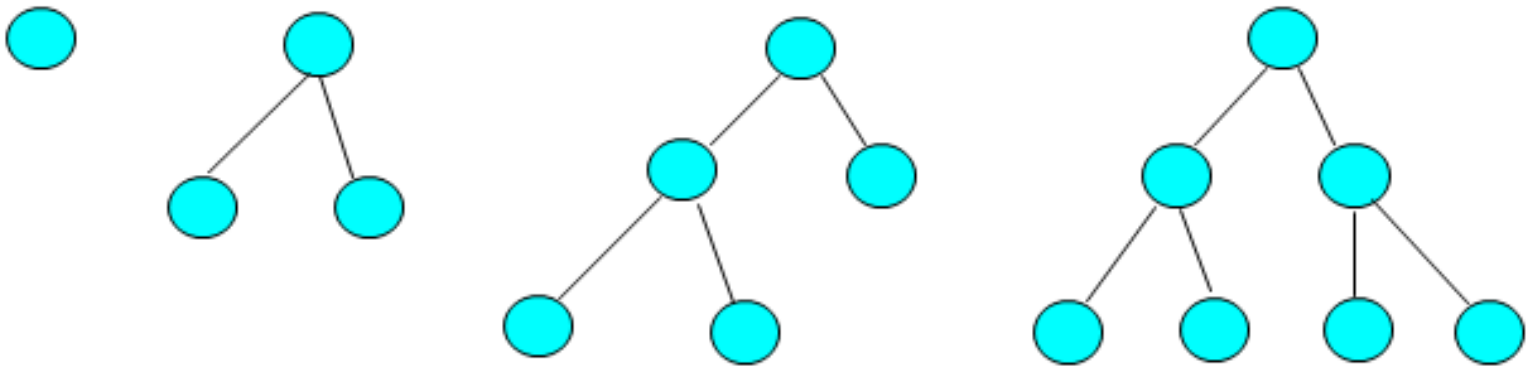
Αλγόριθμοι Τυφλής Αναζήτησης

- **Αναζήτηση Πρώτα σε Πλάτος** (breadth-first search).
- **Αναζήτηση Ενιαίου Κόστους** (uniform-cost search).
- **Αναζήτηση Πρώτα σε Βάθος** (depth-first search).
- **Αναζήτηση Οριοθετημένου Βάθους** (depth-limited search).
- **Αναζήτηση Επαναληπτικής Εκβάθυνσης** (iterative deepening search).
- **Αναζήτηση Διπλής Κατεύθυνσης** (bidirectional search).

Αναζήτηση Πρώτα σε Πλάτος (BFS) (1/3)

function BreadthFirstSearch (*problem*)
returns a *solution* or *failure*
    return TreeSearch *(problem,* EnQueueAtEnd*)*

Παράδειγμα:



Αναζήτηση Πρώτα σε Πλάτος (BFS) (2/3)

- Αξιολόγηση:
  - **Πλήρης;** Ναι.
  - **Χρονική πολυπλοκότητα** – O (b^{d+1}).
    - b είναι ο **παράγοντας διακλάδωσης** (branching factor) και d το **βάθος** της πιο ρηχής λύσης.
  - **Χωρική πολυπλοκότητα** – O (b^{d+1}).
    - το μεγαλύτερο πρόβλημα του BFS.
  - **Βρίσκει την βέλτιστη λύση;**
    - Ναι αν ο παράγοντας διακλάδωσης είναι πεπερασμένος και όλες οι ενέργειες έχουν το ίδιο κόστος.
- Η αναζήτηση πρώτα σε πλάτος βρίσκει πάντα την πιο “ρηχή” λύση στο δέντρο αναζήτησης.

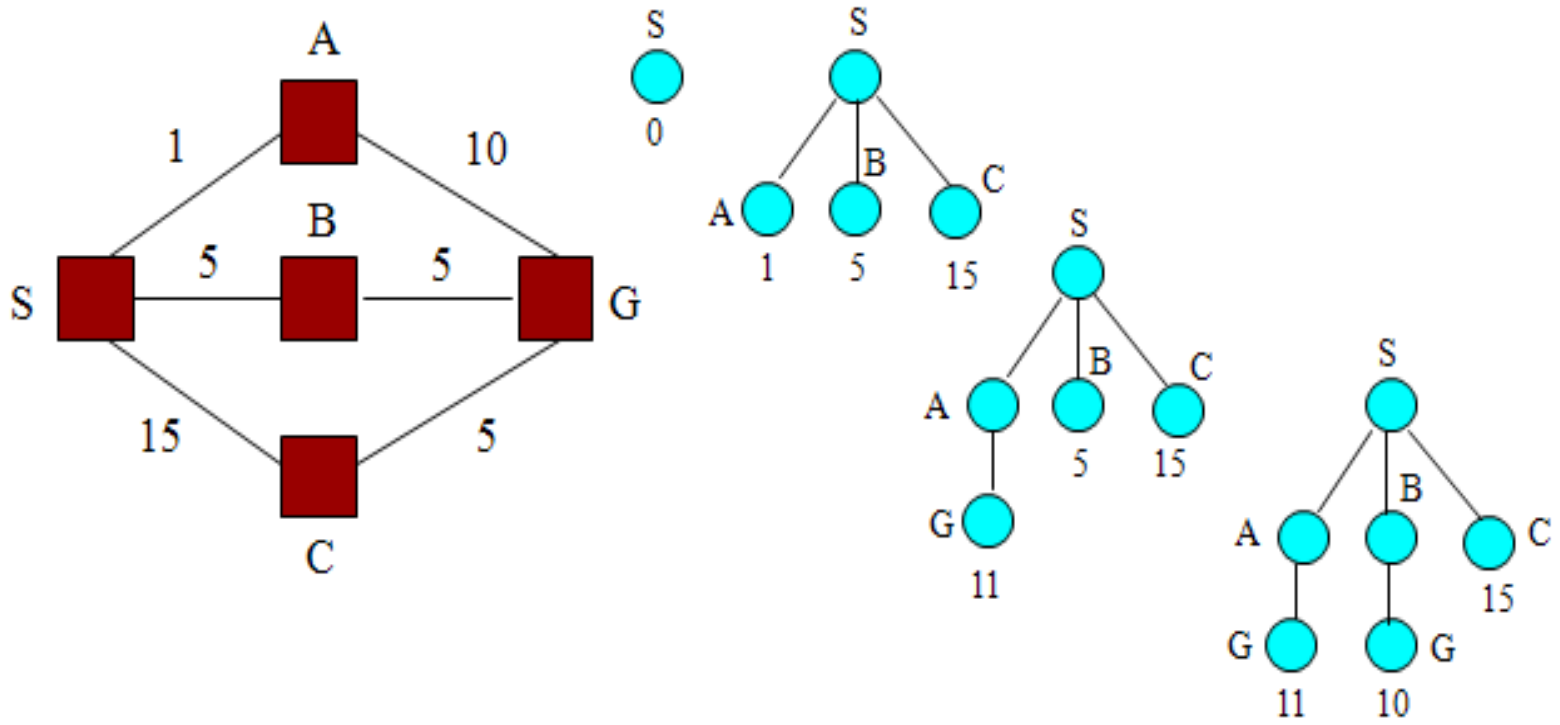
Αναζήτηση Πρώτα σε Πλάτος (BFS) (3/3)

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

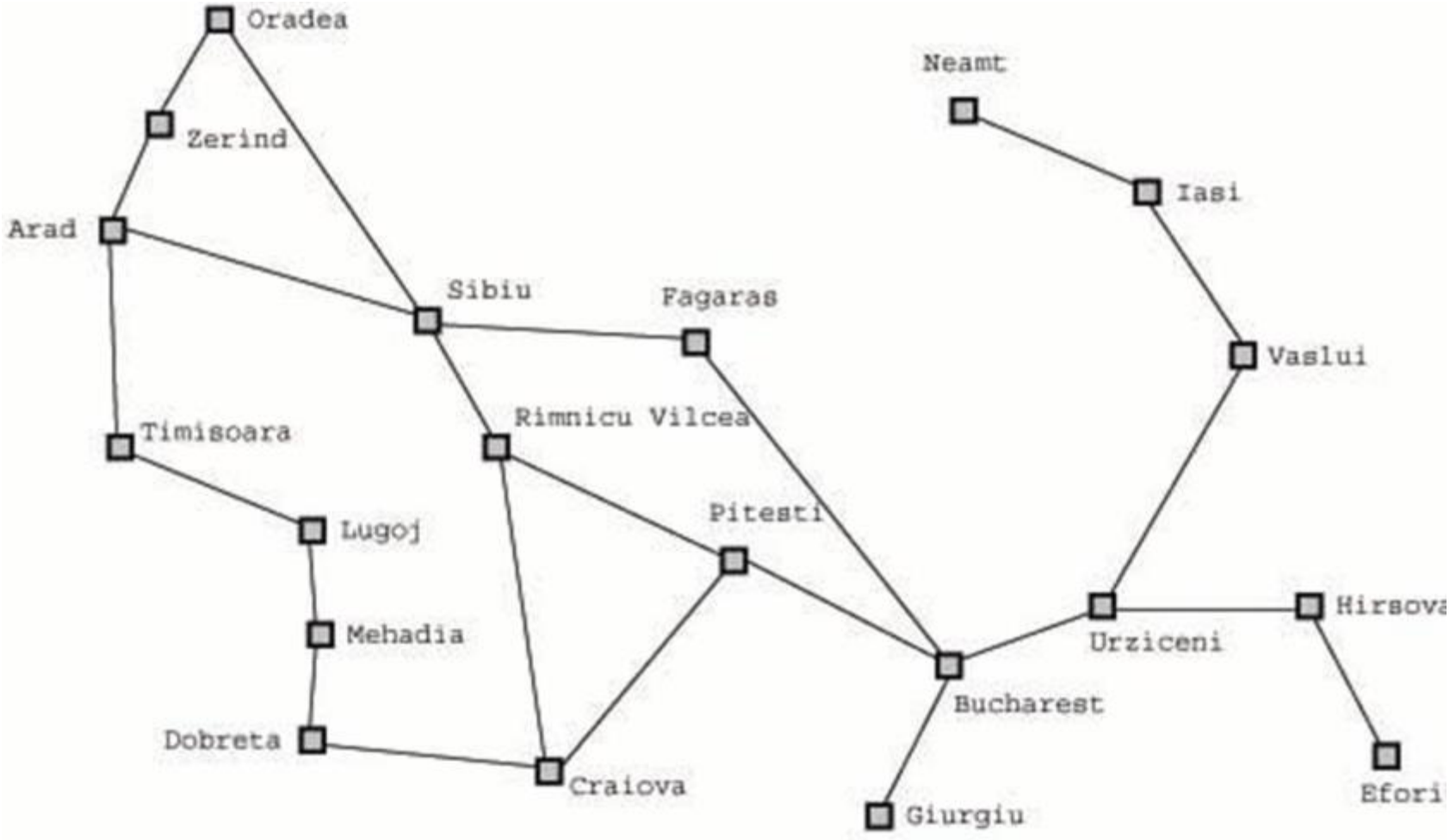
- Απαιτήσεις σε χρόνο και μνήμη για BFS.
- Υποθέτουμε παράγοντα διακλάδωσης 10, έλεγχο 1000 nodes/sec και 100 bytes/node.

Αναζήτηση Ενιαίου Κόστους (UCS) (1/2)

- Παραλλαγή του BFS που πάντα επεκτείνει τον κόμβο με το **μικρότερο κόστος** (σύμφωνα με το κόστος μονοπατιού).



Παράδειγμα


Oradea
Neamt
Zerind
Iasi
Arad
Sibiu
Fagaras
Vaslui
Timisoara
Rimnicu Vilcea
Pitesti
Lugoj
Hirsova
Urziceni
Mehadia
Bucharest
Dobreta
Craiova
Efori
Giurgiu

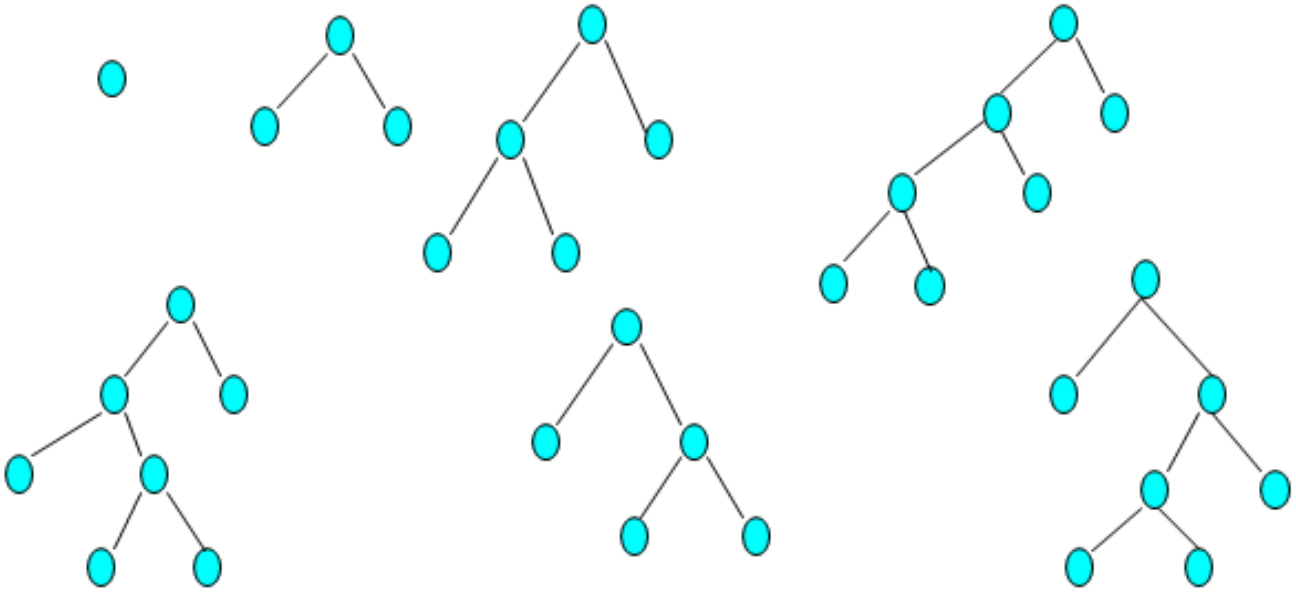
Αναζήτηση Ενιαίου Κόστους (UCS) (2/2)

- Αξιολόγηση:
  - **Πλήρης;** Ναι.
  - **Χρονική πολυπλοκότητα** – O ($b^{\lceil C^*/\varepsilon \rceil}$).
    - b είναι ο **παράγοντας διακλάδωσης** (branching factor), C* το κόστος της βέλτιστης λύσης και κάθε ενέργεια κοστίζει τουλάχιστον ε,
    - μπορεί να είναι πολύ μεγαλύτερο από O (b^{d+1}). Πότε;
  - **Χωρική πολυπλοκότητα** – ίδια με τη χρονική:
    - πάλι έχουμε πρόβλημα.
  - **Βρίσκει την βέλτιστη λύση;**
    - Ναι αν το κόστος ποτέ δε μειώνεται καθώς προχωράμε σε ένα μονοπάτι, δηλ. g(Successor (n)) ≥ g(n) για κάθε κόμβο n.
- Το BFS είναι UCS με g(n) = depth(n).

Αναζήτηση Πρώτα σε Βάθος (DFS) (1/2)

- Η αναζήτηση πρώτα σε βάθος πάντα επεκτείνει έναν κόμβο στο βαθύτερο επίπεδο του δέντρου αναζήτησης.

Παράδειγμα:



Αναζήτηση Πρώτα σε Βάθος (DFS) (2/2)

function DepthFirstSearch (*problem*)
returns a *solution* or *failure*
    return TreeSearch *(problem,* EnQueueAtFront*)*

- Αξιολόγηση:
  - **Πλήρης ;** Όχι.
  - **Χρονική πολυπλοκότητα** – $O(b^m)$.
    - *b* είναι ο **παράγοντας διακλάδωσης** (branching factor), και *m* το μέγιστο βάθος του δέντρου αναζήτησης.
  - **Χωρική πολυπλοκότητα** – $O(bm)$
  - **Βρίσκει την βέλτιστη λύση;**
    - Όχι
- Παραλλαγή: **Αναζήτηση με υπαναχώρηση** (backtracking search):
  - $O(m)$ χωρική πολυπλοκότητα.

Αναζήτηση Οριοθετημένου Βάθους (DLS)

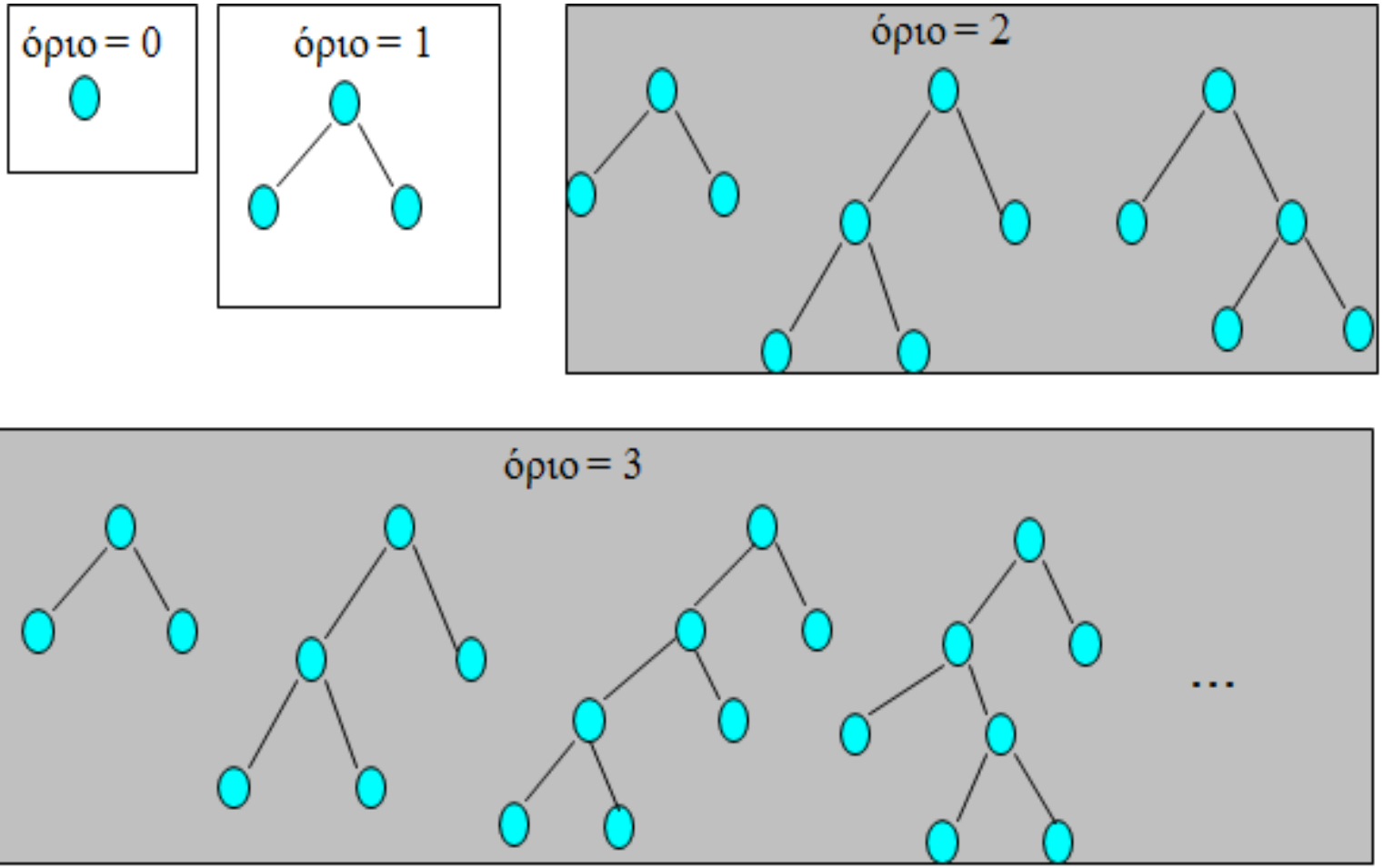
- Η αναζήτηση οριοθετημένου βάθους είναι όπως το DFS με τη διαφορά ότι υπάρχει ένα όριο στο βάθος της αναζήτησης:
  - Στο παράδειγμα της διαδρομής για Bucharest ένα καλό όριο είναι το 19.
    - ✓ Και το 9 καλύτερο.
- Αξιολόγηση:
  - **Πλήρης;** Ναι αν l ≥ d, όπου l είναι το όριο και d το βάθος μιας λύσης.
  - **Χρονική πολυπλοκότητα** – $O(b^l)$.
  - **Χωρική πολυπλοκότητα** – O(bl).
  - **Βρίσκει την βέλτιστη λύση;**
    - Όχι
- Πρόβλημα:
  - ✓ Μπορούμε πάντα να βρίσκουμε ένα καλό όριο;

Αναζήτηση Επαναληπτικής Εκβάθυνσης (IDS) (1/4)

- Η αναζήτηση επαναληπτικής εκβάθυνσης αντιμετωπίζει το πρόβλημα της εύρεσης σωστού ορίου για το βάθος δοκιμάζοντας όλα τα πιθανά: 0, 1, 2, κτλ.

function IterativeDeepeningSearch (*problem*)
returns a *solution sequence* or *failure*
    for *depth* ← 0 **to** ∞ **do**
        if DepthLimitedSearch *(problem, depth)* succeeds
        then return its result
    endfor
    return failure

Αναζήτηση Επαναληπτικής Εκβάθυνσης (IDS) (2/4)



Αναζήτηση Επαναληπτικής Εκβάθυνσης (IDS) (3/4)

- **Ερώτηση:** Είναι το IDS σπάταλος αλγόριθμος;

  Απάντηση: Όχι!
- Ας υποθέσουμε ότι βρίσκουμε μια λύση όταν επεκτείνεται ο τελευταίος κόμβος στο επίπεδο *d.*
  - Ο αριθμός των κόμβων που παράγονται από BFS ως το βάθος *d* είναι: $b + b^2 + \ldots + b^d + (b^{d+1} - b)$
  - Ο αριθμός των κόμβων που παράγονται από IDS ως το βάθος *d* είναι: $db + (d-1)b^2 + \ldots + 2b^{d-1} + b^d$
  - Μπορούμε να δούμε από τους τύπους ότι το BFS μπορεί να είναι πολύ πιο σπάταλο από το IDS.

Αναζήτηση Επαναληπτικής Εκβάθυνσης (IDS) (4/4)

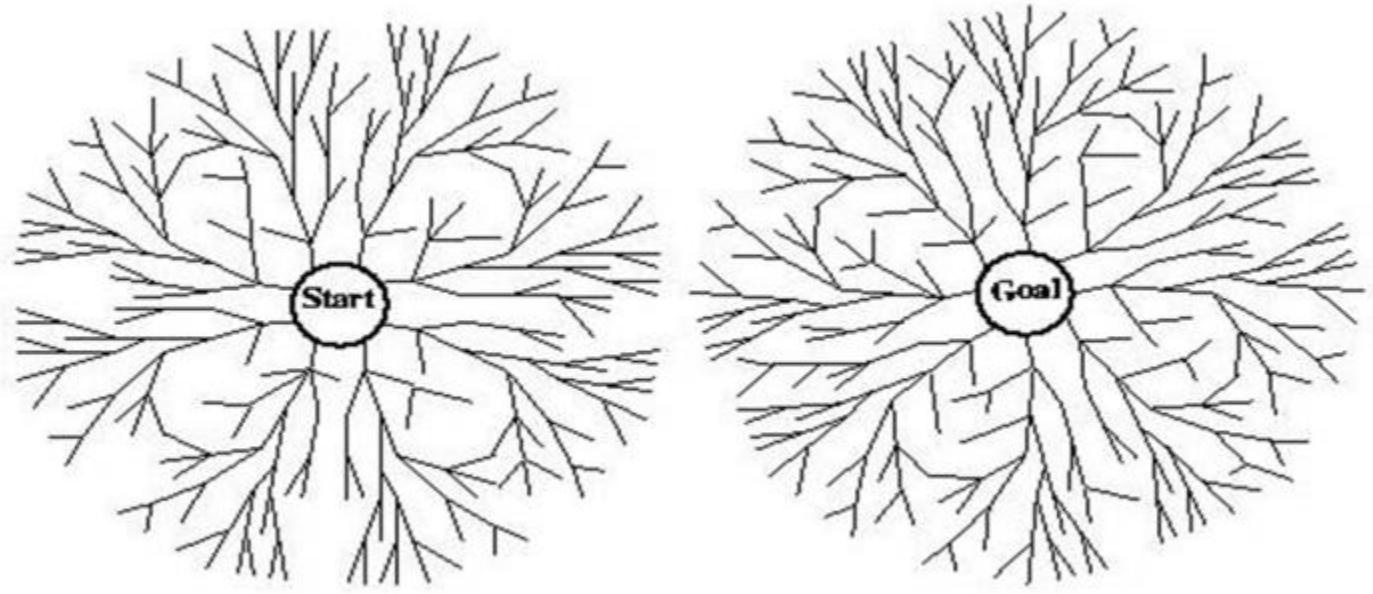
- Για b=10, d=5 έχουμε 123.450 κόμβους για το IDS και 1.111.100 κόμβους για το BFS.
- Αξιολόγηση:
  - **Πλήρης;** Ναι.
  - **Χρονική πολυπλοκότητα** – $O(b^d)$.
  - **Χωρική πολυπλοκότητα** – $O(bd)$.
  - **Βρίσκει την βέλτιστη λύση;** Ναι.
- Το IDS είναι η καλύτερη επιλογή όταν ο χώρος αναζήτησης είναι μεγάλος και το βάθος της αναζήτησης δεν είναι γνωστό.
- Μπορούμε να συνδυάσουμε την ιδέα με το UCS;

Αναζήτηση Διπλής Κατεύθυνσης (1/2)

- **Βασική Ιδέα:**
  - Αναζήτηση ταυτόχρονα από την αρχική κατάσταση προς το στόχο και από τον στόχο προς την αρχική κατάσταση.
    - ✓ εφαρμόζεται όταν οι ενέργειες είναι ***αναστρέψιμες.***
  - Σταμάτα όταν οι δύο αναζητήσεις συναντηθούν.

- **Προβλήματα:**
  - Τι σημαίνει αναζήτηση από τον στόχο προς τα πίσω στην αρχική κατάσταση;
  - Τι γίνεται όταν έχουμε πολλές πιθανές καταστάσεις στόχου;
  - Μπορούμε να ελέγξουμε αποδοτικά πότε συναντώνται οι δύο αναζητήσεις;
  - Τι είδους αναζήτηση κάνουμε σε κάθε μια;

Αναζήτηση Διπλής Κατεύθυνσης (2/2)



- Σχηματική αναπαράσταση αναζήτησης διπλής κατεύθυνσης.

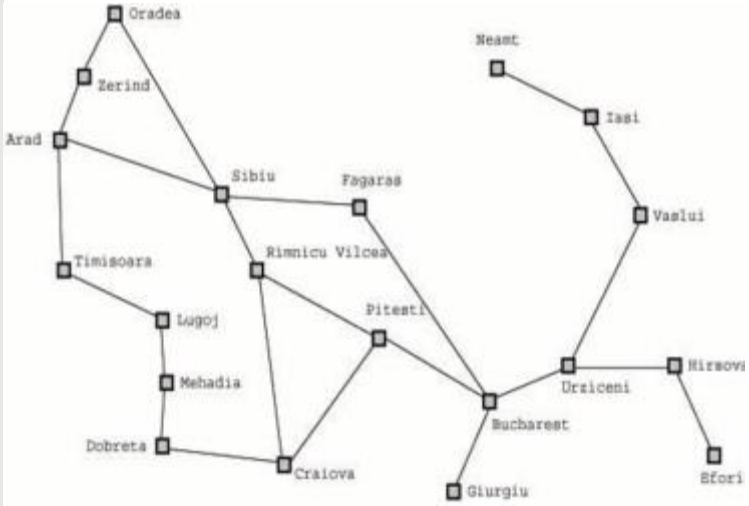
Σύγκριση Αλγορίθμων Αναζήτησης

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

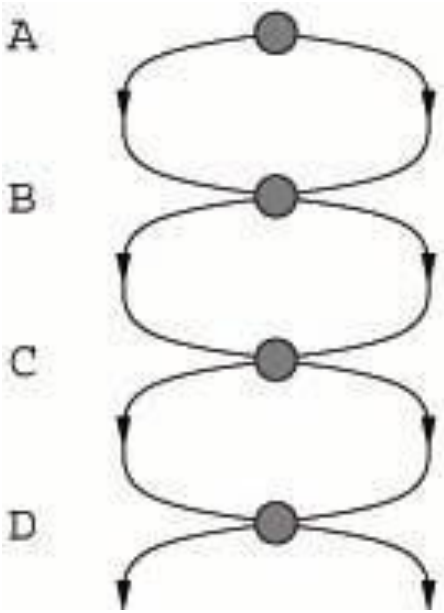
- ***b***: παράγοντας διακλάδωσης.
- ***d***: το βάθος της λύσης.
- ***m***: το μέγιστο βάθος του δέντρου αναζήτησης.
- ***l***: το όριο βάθους αναζήτησης.

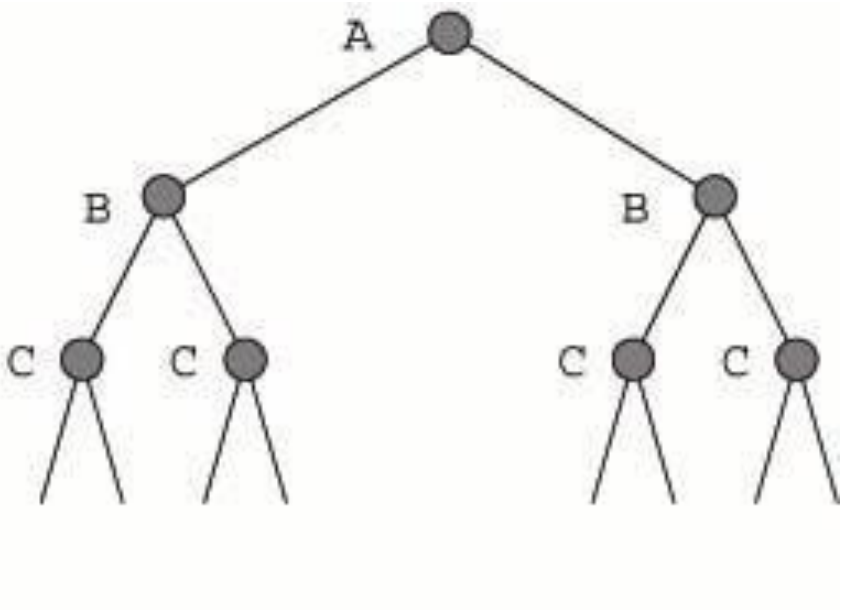
Επαναλαμβανόμενες Καταστάσεις

- Τι γίνεται όταν κατά την αναζήτηση συναντήσουμε καταστάσεις που έχουμε ήδη συναντήσει και επεκτείνει;
  - *οι αλγόριθμοι τυφλής αναζήτησης που είδαμε δε μπορούν να αντιμετωπίσουν αυτό το πρόβλημα.*
- Όταν ο χώρος καταστάσεων είναι δέντρο το πρόβλημα αυτό δεν εμφανίζεται:
  - *όταν οι ενέργειες είναι αναστρέψιμες τότε οι επαναλαμβανόμενες καταστάσεις είναι αναπόφευκτες.*



Αποφυγή Επαναλαμβανόμενων Καταστάσεων (1/2)





Πως μπορούμε να αποφύγουμε την επέκταση καταστάσεων που έχουμε ήδη συναντήσει και επεκτείνει;

Αποφυγή Επαναλαμβανόμενων Καταστάσεων (2/2)

- Σε αυτή την περίπτωση ο χώρος καταστάσεων είναι **γράφος.**
- Μια λύση είναι να αποφεύγουμε να παράγουμε καταστάσεις που έχουν παραχθεί προηγουμένως:
  - Αυτό μπορεί να πραγματοποιηθεί κρατώντας μια λίστα των παραχθέντων καταστάσεων που ονομάζεται **κλειστή λίστα** *(closed list).*
  - Σε αυτή την περίπτωση το σύνορο των κόμβων που δεν έχουν επεκταθεί ονομάζεται **ανοιχτή λίστα** *(open list).*
  - Η κλειστή λίστα μπορεί να υλοποιηθεί με ένα πίνακα κατακερματισμού για πρόσβαση σε σταθερό χρόνο.
  - Αν ο τρέχων κόμβος βρίσκεται στην κλειστή λίστα απορρίπτεται:
    - ✓ υπάρχει πιθανότητα να χάσουμε τη βέλτιστη λύση;

Γενικός Αλγόριθμος Αναζήτησης (2/2)

function TreeSearch (*problem*, Queuing-Fn)
returns a *solution* or *failure*
closed ← an empty set
fringe ← MakeQueue(MakeNode(IninitialState[problem]))
loop do
if *fringe* is empty **then return** *failure*
node ← RemoveFront(*fringe*)
if GoalTest[*problem*] applied to State[*node*] succeeds
then return node
if State[*node*] is not in *closed* **then**
add State[*node*] to *closed*
fringe ← Queuing-Fn(*fringe*,Expand(*node*, *problem*))
end

Ανακεφαλαίωση

- **Πράκτορες επίλυσης προβλημάτων.**
- **Προβλήματα αναζήτησης.**
- **Στρατηγικές τυφλής αναζήτησης:**
  - Αναζήτηση Πρώτα σε Πλάτος.
  - Αναζήτηση Ενιαίου Κόστους.
  - Αναζήτηση Πρώτα σε Βάθος.
  - Αναζήτηση Οριοθετημένου Βάθους.
  - Αναζήτηση Επαναληπτικής Εκβάθυνσης.
  - Αναζήτηση Διπλής Κατεύθυνσης.

Τέλος Ενότητας

Σημείωμα Αναφοράς

- Copyright Πανεπιστήμιο Δυτικής Μακεδονίας, Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών, Στεργίου Κωνσταντίνος. «**Τεχνητή Νοημοσύνη**». Έκδοση: 1.0. Κοζάνη 2015. Διαθέσιμο από τη δικτυακή διεύθυνση:

  https: //eclass.uowm.gr/courses/ICTE103/

Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Όχι Παράγωγα Έργα Μη Εμπορική Χρήση 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[1] h t t p ://creativecommons.org/licenses/by-nc-nd/4.0/

Ως Μη Εμπορική ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

Εικόνες/Σχήματα/Διαγράμματα/Φωτογραφίες

- Τεχνητή Νοημοσύνη, Μια σύγχρονη προσέγγιση, S. Russel, P. Norvig, Εκδόσεις Κλειδάριθμος