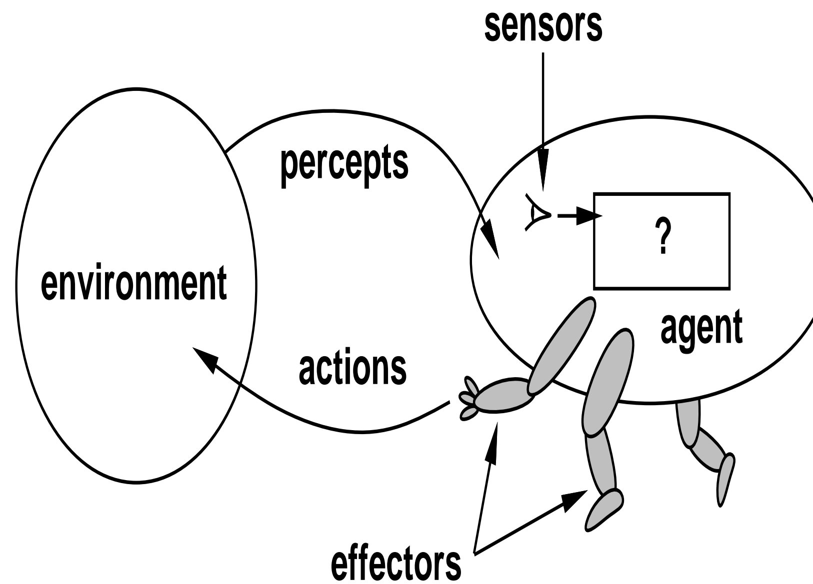
Επίλυση Προβλημάτων με Αναζήτηση

- Πράκτορες Βασισμένοι στο Στόχο, Πράκτορες Επίλυσης Προβλημάτων
- Προβλήματα Αναζήτησης
- Στρατηγικές Τυφλής (Απληροφόρητης) Αναζήτησης

Πράκτορες



Πράκτορας (agent) είναι οτιδήποτε μπορεί να θεωρηθεί ότι **αντιλαμβάνεται** το **περιβάλλον (environment)** του μέσω **αισθητήρων (sensors)** και **αλληλεπιδρά** με αυτό μέσω **μηχανισμών δράσης** (**effectors - actuators**).

Πως Πρέπει να Ενεργούν οι Πράκτορες;

Η συμπεριφορά ενός πράκτορα εξαρτάται από τα εξής:

- Το **περιβάλλον του.**
- Την **ακολουθία αντιλήψεων (percept sequence)** δηλ. το πλήρες ιστορικό για οτιδήποτε έχει αντιληφθεί ο πράκτορας.

  Ένας πράκτορας μπορεί να περιγραφεί από μια **συνάρτηση πράκτορα** που αντιστοιχεί κάθε ακολουθία αντιλήψεων σε μια **ενέργεια.**

  Η συνάρτηση πράκτορα υλοποιείται από ένα **πρόγραμμα πράκτορα**.
- Το **μέτρο απόδοσης (performance measure).** Είναι το αντικειμενικό κριτήριο της επιτυχίας της συμπεριφοράς ενός πράκτορα και καθορίζεται από το σχεδιαστή του. Μπορεί να μην είναι εύκολο να ορίσουμε το μέτρο απόδοσης.

  Παράδειγμα: Το μέτρο απόδοσης ενός αυτοματοποιημένου οδηγού ταξί είναι ...

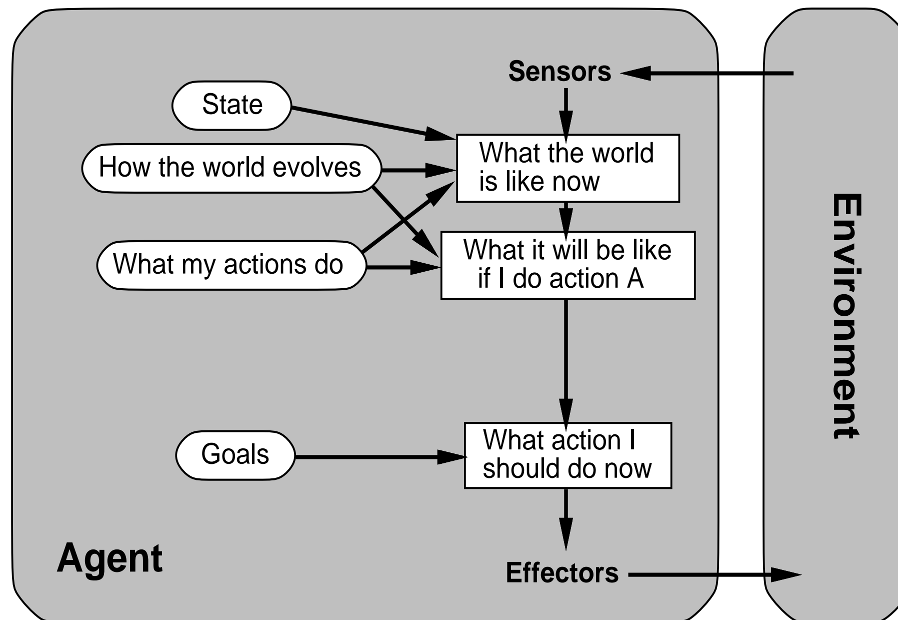
Υποθέσεις

Σ' αυτό το κεφάλαιο κάνουμε τις εξής υποθέσεις:

- Το περιβάλλον του πράκτορα μπορεί να παρατηρηθεί με ακρίβεια και είναι αιτιοκρατικό (deterministic).
- Ο πράκτορας γνωρίζει με ακρίβεια ποια είναι τα αποτελέσματα των ενεργειών του.

Αυτές οι υποθέσεις δεν ισχύουν στη γενική περίπτωση (π.χ., στον πραγματικό κόσμο).

Πράκτορες Βασισμένοι στο Στόχο (Goal-Based Agents)



Η συμπεριφορά ενός πράκτορα εξαρτάται επίσης από:

- Τους **στόχους (goals)** του πράκτορα. Ένας στόχος καθορίζει ποιες καταστάσεις του περιβάλλοντος είναι επιθυμητές για τον πράκτορα.

Πρακτορες Επίλυσης Προβλημάτων

Οι πράκτορες επίλυσης προβλημάτων (problem-solving agents) είναι μια κλάση πρακτόρων βασισμένων στο στόχο. Οι πράκτορες επίλυσης προβλημάτων αποφασίζουν τι να κάνουν επιλέγοντας ακολουθίες ενεργειών που οδηγούν σε επιθυμητές καταστάσεις.

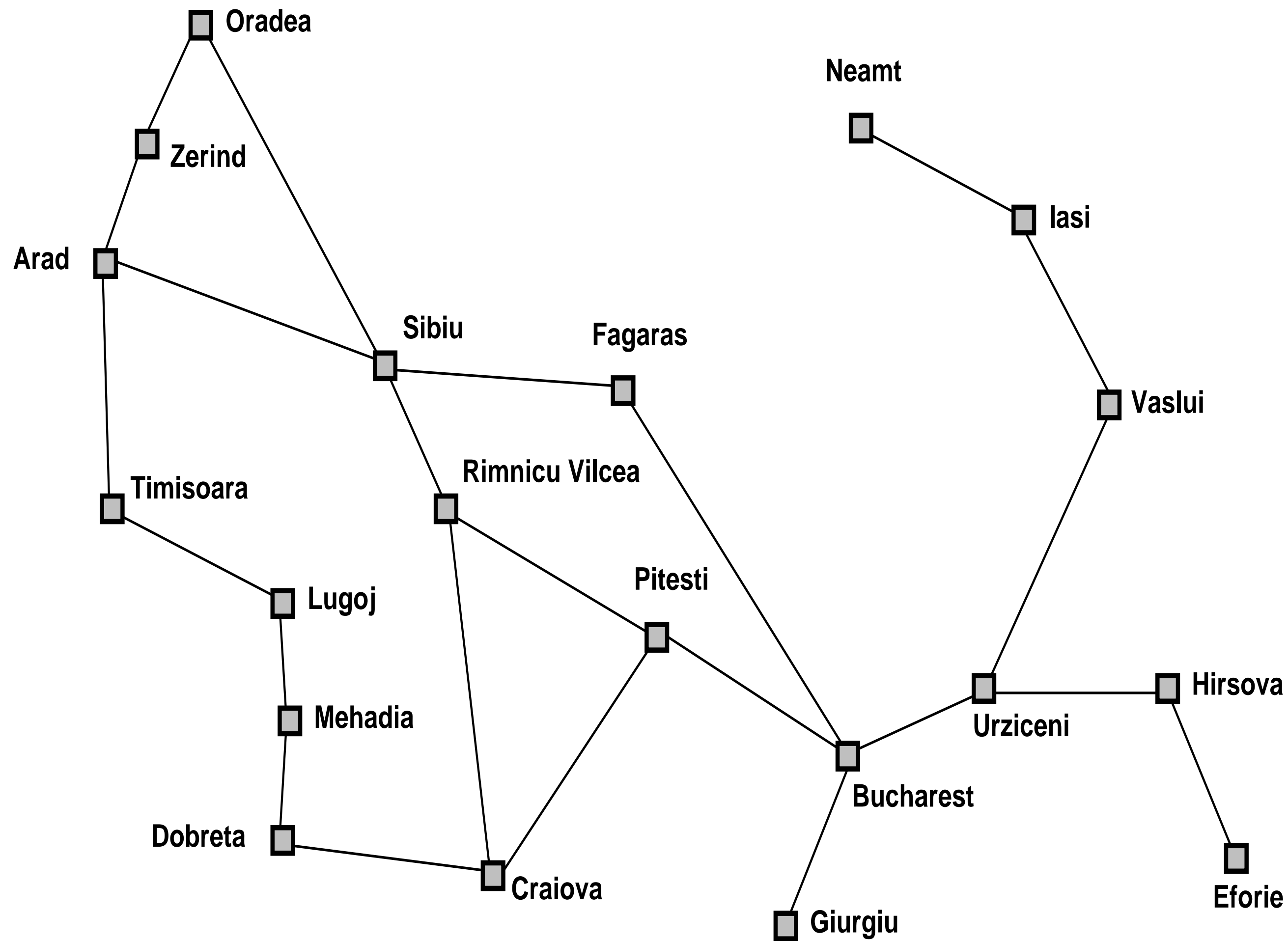
Παράδειγμα: Ένας πράκτορας βρίσκεται στην πόλη Αράντ της Ρουμανίας. Πως μπορεί να φτάσει στο Βουκουρέστι την επόμενη μέρα, ώστε να προλάβει την πτήση του;

Πράκτορες Επίλυσης Προβλημάτων

Οι **πράκτορες επίλυσης προβλημάτων** λειτουργούν εκτελώντας ξανά και ξανά τα παρακάτω:

- **Διατύπωση στόχου (goal formulation):** αποφασίζουν ποιος είναι ο επιδιωκόμενος στόχος.
- **Διατύπωση προβλήματος (problem formulation):** αποφασίζουν ποιες ενέργειες και καταστάσεις θα λάβουν υπ' όψιν τους για να πετύχουν τον επιδιωκόμενο στόχο.
- **Αναζήτηση (search):** επιλέγουν μια ακολουθία από ενέργειες που οδηγούν στο στόχο.
- **Εκτέλεση (execution):** εκτελούν την επιλεγμένη ακολουθία ενεργειών.

Παράδειγμα: Εύρεση Διαδρομής στη Ρουμανία



Το Πρώτο μας Πρόγραμμα Πράκτορα

```
function SimpleProblemSolvingAgent(percept)
returns an action
static seq, state, goal, problem

   state ← UpdateState(state, percept)
   if seq is empty then
      goal ← FormulateGoal(state)
      problem ← FormulateProblem(state, goal)
      seq ← Search(problem)
   end
   action ← First(seq)
   seq ← Rest(seq)
   return action
```

Η Δομή Ενός Πράκτορα

Πράκτορας = Αρχιτεκτονική + Πρόγραμμα

Η αρχιτεκτονική χρησιμοποιεί τους αισθητήρες για να παρέχει αντιλήψεις στο πρόγραμμα, εκτελεί το πρόγραμμα, και τροφοδοτεί τους μηχανισμούς δράσης με τις επιλεγμένες ενέργειες, καθώς αυτές παράγονται.

Σ' αυτή την ενότητα διαλέξεων θα ασχοληθούμε με **προγράμματα πρακτόρων επίλυσης προβλημάτων**.

Προβλήματα ή Προβλήματα Αναζήτησης

Τα βασικά στοιχεία ενός **προβλήματος αναζήτησης (search problem)** είναι:

- Η **αρχική κατάσταση (initial state)**.
- Το σύνολο των **διαθέσιμων ενεργειών (available actions)**. Για να καθορίσουμε τις διαθέσιμες ενέργειες χρησιμοποιούμε μια **συνάρτηση διαδοχής (successor function)** $Succ$ η οποία, δοθείσας μιας κατάστασης x, επιστρέφει ένα σύνολο από διατεταγμένα ζεύγη $(action, successor\ state)$.

  Το παραπάνω σύνολο υπαγορεύει ποιες ενέργειες είναι δυνατές όταν ο πράκτορας βρίσκεται στην κατάσταση x, και σε ποιες καταστάσεις μπορεί να φτάσει από την x εκτελώντας αυτές τις ενέργειες.

Προβλήματα Αναζήτησης

- Ο **χώρος καταστάσεων (state space).**

  Η αρχική κατάσταση και η συνάρτηση $Succ$ καθορίζουν το **χώρο καταστάσεων** ενός προβλήματος αναζήτησης.

  Ο **χώρος καταστάσεων** είναι το σύνολο όλων των καταστάσεων στις οποίες μπορούμε να φτάσουμε από την αρχική κατάσταση με μια οποιαδήποτε ακολουθία ενεργειών.

  Ένας χώρος καταστάσεων παριστάνεται συνήθως από ένα **κατευθυνόμενο γράφο με ετικέττες** που οι κόμβοι του είναι οι καταστάσεις και οι ετικέττες των ακμών είναι ενέργειες.

  Ένα **μονοπάτι** στο χώρο καταστάσεων είναι οποιαδήποτε ακολουθία καταστάσεων που συνδέονται από μια ακολουθία ενεργειών.

Προβλήματα Αναζήτησης

- Ο **στόχος (goal)** που θέλουμε να πετύχουμε. Ο στόχος είναι ένα σύνολο από καταστάσεις που ονομάζονται **καταστάσεις στόχου (goal states).** Οι στόχοι μπορούν να καθοριστούν με ακρίβεια από ένα **έλεγχο στόχου (goal test)**, δηλαδή ένα έλεγχο που μπορεί να εφαρμοστεί σε μια κατάσταση για να αποφασίσουμε αν πρόκειται για κατάσταση στόχου.
- **Συνάρτηση κόστους μονοπατιού (path cost function)** είναι μια συνάρτηση (συνήθως συμβολίζεται με g) η οποία αποδίδει ένα αριθμητικό κόστος σε κάθε μονοπάτι. Το κόστος ενός μονοπατιού είναι συνήθως το άθροισμα από τα κόστη των επιμέρους ενεργειών στο μονοπάτι.

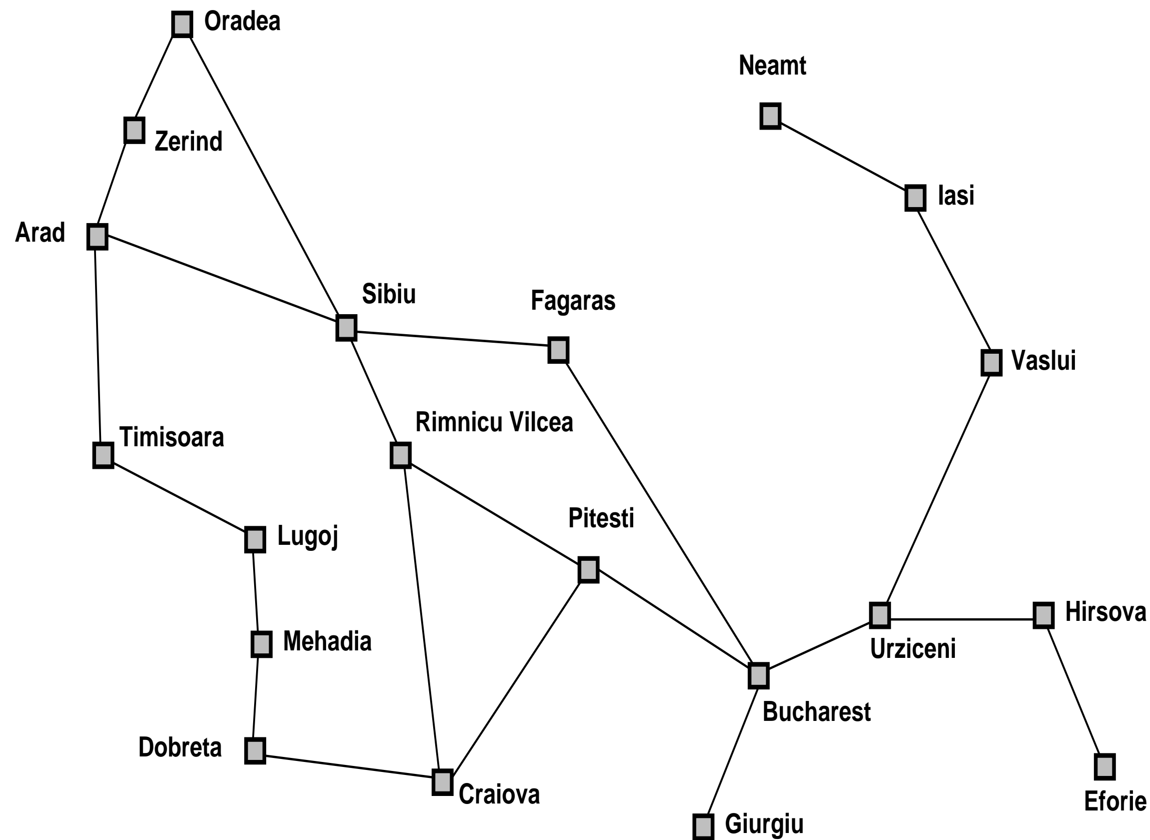
  Το κόστος βήματος (step cost) για μια ενέργεια a από μια κατάσταση x στην κατάσταση y συμβολίζεται με $c(x, a, y)$.

Προβλήματα Αναζήτησης

Μια **λύση (solution)** σε ένα πρόβλημα αναζήτησης είναι ένα μονοπάτι από την αρχική κατάσταση σε μια κατάσταση στόχου.

Μια λύση είναι **βέλτιστη (optimal)** αν έχει το ελάχιστο κόστος μεταξύ όλων των λύσεων.

Παράδειγμα: Εύρεση Διαδρομής στη Ρουμανία



Η Διατύπωση σαν Πρόβλημα Αναζήτησης

Το πρόβλημα της μετάβασης από το Αράντ στο Βουκουρέστι μπορεί να οριστεί τυπικά ως εξής:

- Οι **καταστάσεις** ορίζονται από τις πόλεις στις οποίες βρισκόμαστε π.χ., $In(Arad)$.
- Η μόνη διαθεσιμη **ενέργεια** είναι η $GoTo$ π.χ., $GoTo(Sibiu)$.
- Για κάθε κατάσταση, η **συνάρτηση διαδοχής** μας δίνει ένα σύνολο από ζεύγη $(GoTo(.), In(.))$. Για παράδειγμα:

$$Succ(In(Arad)) = \{(GoTo(Sibiu), In(Sibiu)),$$

$$(GoTo(Timisoara), In(Timisoara)), (GoTo(Zerind), In(Zerind))\}.$$

- Η **αρχική κατάσταση** είναι $In(Arad)$. Η **κατάσταση στόχου** είναι $In(Bucharest)$.
- Το **κόστος μονοπατιού** μπορεί να δίδεται από την αντίστοιχη χιλιομετρική απόσταση.

Το Πρόβλημα των 8 Πλακιδίων

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

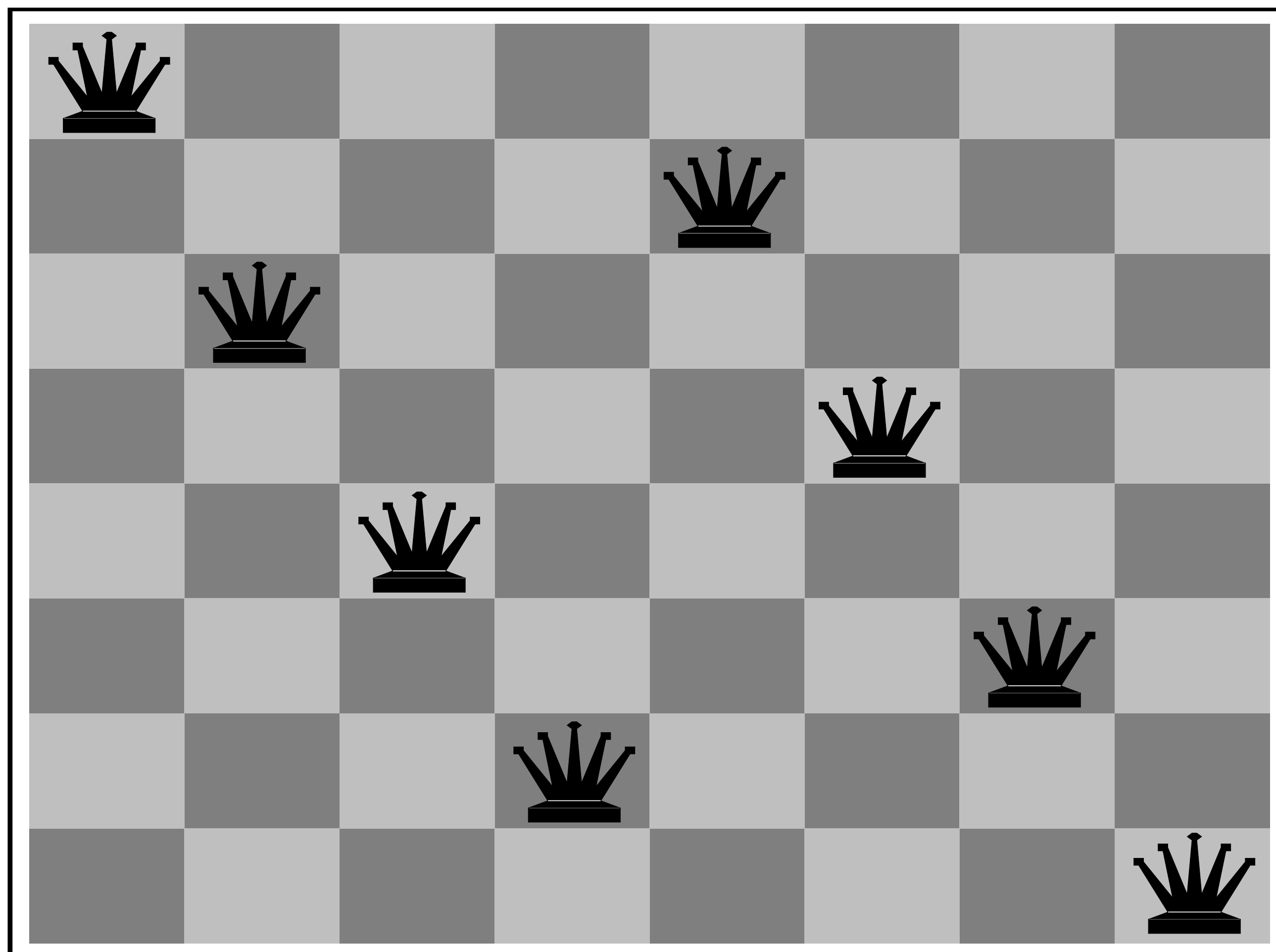
Goal State

Το Πρόβλημα των 8 Πλακιδίων (8-puzzle)

Τυπικός ορισμός:

- **Καταστάσεις**: μια περιγραφή κατάστασης καθορίζει τη θέση κάθε πλακιδίου και του κενού.
- **Ενέργειες**: Το κενό κινείται αριστερά, δεξιά, πάνω ή κάτω.
- Η **αρχική κατάσταση** και η **κατάσταση στόχου** δίνονται.
- **Κόστος μονοπατιού**: το μήκος του μονοπατιού.

Το Πρόβλημα των 8 Βασιλισσών (8-Queens)



Το Πρόβλημα των 8 Βασιλισσών

Τυπικός Ορισμός:

- **Καταστάσεις**: Οποιαδήποτε διάταξη 0 έως 8 βασιλισσών στη σκακιέρα
- **Αρχική Κατάσταση**: Καμία βασίλισσα στη σκακιέρα
- **Ενέργειες**: Πρόσθεσε μια βασίλισσα σε οποιοδήποτε κενό τετράγωνο
- **Κατάσταση Στόχου**: 8 βασίλισσες στη σκακιέρα, καμία δεν απειλείται
- **Κόστος Μονοπατιού**: Μηδέν

Μέγεθος Χώρου Καταστάσεων: 64^8

Το Πρόβλημα των 8 Βασιλισσών

Εναλλακτική Διατύπωση:

- **Καταστάσεις**: Διατάξεις n $(0 \leq n \leq 8)$ βασιλισσών, μία βασίλισσα ανά στήλη στις n πρώτες στήλες από τα αριστερά, ώστε καμία βασίλισσα δεν απειλείται.
- **Αρχική Κατάσταση**: Καμία βασίλισσα στη σκακιέρα.
- **Ενέργειες**: Τοποθετούμε μία βασίλισσα σε οποιοδήποτε κενό τετράγωνο της αριστερότερης κενής στήλης έτσι ώστε να μην απειλείται από καμία άλλη βασίλισσα.
- **Τελική Κατάσταση**: 8 βασίλισσες στη σκακιέρα, καμία δεν απειλείται.
- **Κόστος Μονοπατιού**: Μηδέν

Μέγεθος Χώρου Καταστάσεων: 8^8

Προβλήματα Αναζήτησης στον Πραγματικό Κόσμο

- Εύρεση διαδρομών σε χάρτες
- Προβλήματα δρομολόγησης π.χ., το πρόβλημα του πλανόδιου πωλητή (travelling salesman problem)
- Διάταξη κυκλωμάτων VLSI
- Πλοήγηση ρομπότ
- Αυτόματη ακολουθία συναρμολόγησης (automatic assembly sequencing)
- Σχεδίαση πρωτεϊνών (protein design)
- Προβλήματα βελτιστοποίησης ερωτοαποκρίσεων (query optimization) σε Συστήματα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων
- Αναζήτηση στο διαδίκτυο
- Αυτόματη δημιουργία ροής εργασιών

Υπολογιστική Πολυπλοκότητα

Σχεδόν όλα τα προβλήματα που παρουσιάσαμε πιο πάνω είναι **NP-δύσκολα (NP-hard) ή δυσκολότερα** από άποψη υπολογιστικής πολυπλοκότητας.

Έτσι δεν θα πρέπει να περιμένουμε οι απλοί αλγόριθμοι για τα προβλήματα αναζήτησης να είναι αποδοτικοί. Αυτό είναι ένα μεγάλο πρόβλημα για τα προβλήματα αναζήτησης. Θα προσπαθήσουμε να βρούμε τρόπους να το αντιμετωπίσουμε.

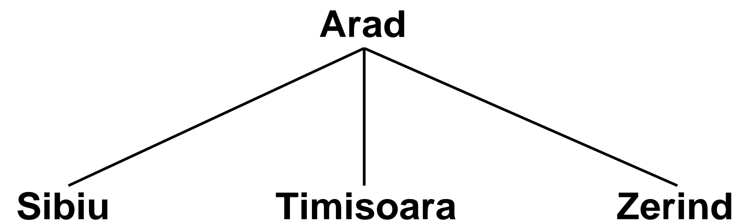
Αναζήτηση Λύσεων

Παράδειγμα:

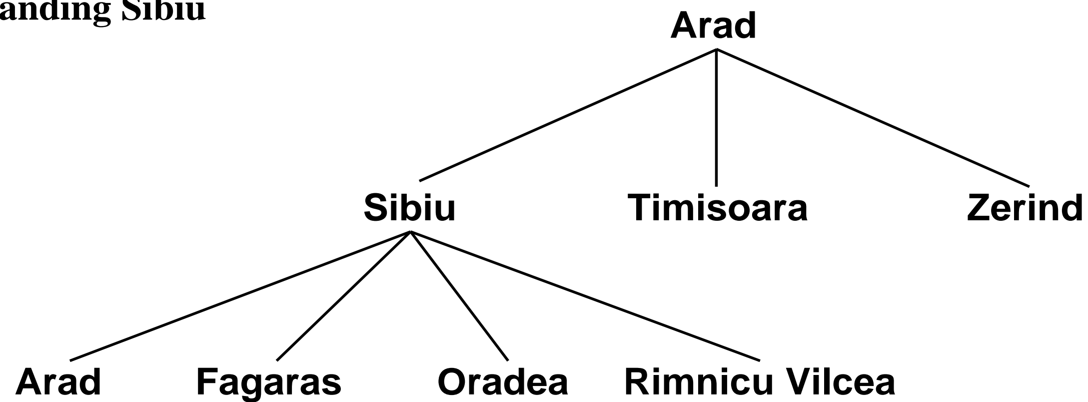
(a) The initial state

Arad

(b) After expanding Arad



(c) After expanding Sibiu



Αναζήτηση Λύσεων

- Η εύρεση μιας λύσης γίνεται ψάχνοντας στο χώρο των καταστάσεων. Η ιδέα είναι να **διατηρούμε και να επεκτείνουμε ένα σύνολο από μερικές λύσεις (partial solutions).**
- Η επιλογή της κατάστασης που θα επεκτείνουμε στη συνέχεια καθορίζεται από τη **στρατηγική αναζήτησης (search strategy)**.
- Η διαδικασία αναζήτησης κατασκευάζει ένα **δέντρο αναζήτησης (search tree)** το οποίο βρίσκεται ένα επίπεδο πάνω από το χώρο αναζήτησης.
- Είναι σημαντικό να **ξεχωρίσουμε το χώρο αναζήτησης (search space) από το δέντρο αναζήτησης.**

Ο Γενικός Αλγόριθμος Αναζήτησης σε Δένδρο

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 candidates for expansion **then**
            return failure
        choose a leaf node for expansion according to *strategy*
        if the node contains a goal state **then**
            return the corresponding solution
        else expand the node and add the resulting nodes
                to the search tree
    end

Κόμβοι του Δέντρου Αναζήτησης

Οι κόμβοι σε ένα δέντρο αναζήτησης μπορούν να παρασταθούν με μια δομή με πέντε στοιχεία:

- STATE: η κατάσταση στην οποία αντιστοιχεί ο κόμβος.
- PARENTNODE: ο κόμβος του δέντρου αναζήτησης από τον οποίο προήλθε ο τρέχων κόμβος.
- ACTION: η ενέργεια που πραγματοποιήθηκε για να παραχθεί ο τρέχων κόμβος.
- PATHCOST: το κόστος του μονοπατιού από την αρχική κατάσταση μέχρι τον τρέχοντα κόμβο.
- DEPTH: το πλήθος των κόμβων στο μονοπάτι από τη ρίζα του δέντρου μέχρι τον τρέχοντα κόμβο.

Το Σύνορο

Το σύνολο των κόμβων που πρόκειται να επεκταθούν ονομάζεται **σύνορο (frontier or fringe)**. Το σύνορο μπορεί να υλοποιηθεί ως μια **ουρά** με λειτουργίες:

- MakeQueue(*Elements*)
- Empty?(*Queue*)
- RemoveFront(*Queue*)
- QueuingFn(*Elements*, *Queue*)

Ο Γενικός Αλγόριθμος Αναζήτησης σε Δένδρο

function TreeSearch(*problem*,QueuingFn)
returns a solution, or failure
    $fringe \leftarrow$ MakeQueue(MakeNode(InitialState[*problem*]))
    loop do
        if $fringe$ is empty **then return** failure
        $node \leftarrow$ RemoveFront($fringe$)
        if GoalTest[*problem*] applied to State[*node*] succeeds **then**
            return *node*
        $fringe \leftarrow$ QueuingFn(Expand($node, problem$), $fringe$)
    end

Η συνάρτηση Expand είναι υπεύθυνη για τον υπολογισμό των στοιχείων κάθε κόμβου που δημιουργεί.

Αλγόριθμοι Αναζήτησης

Θα μελετήσουμε δύο είδη αλγόριθμων αναζήτησης:

- **Απληροφόρητους (uninformed) ή τυφλούς (blind)**
- **Πληροφορημένους (informed) ή ευρετικούς (heuristic)**

Κριτήρια αποτίμησης για έναν αλγόριθμο αναζήτησης:

- **Πληρότητα (completeness)**
- **Βέλτιστη συμπεριφορά (optimality)**
- **Χρονική πολυπλοκότητα (time complexity)**
- **Χωρική πολυπλοκότητα (space complexity)**

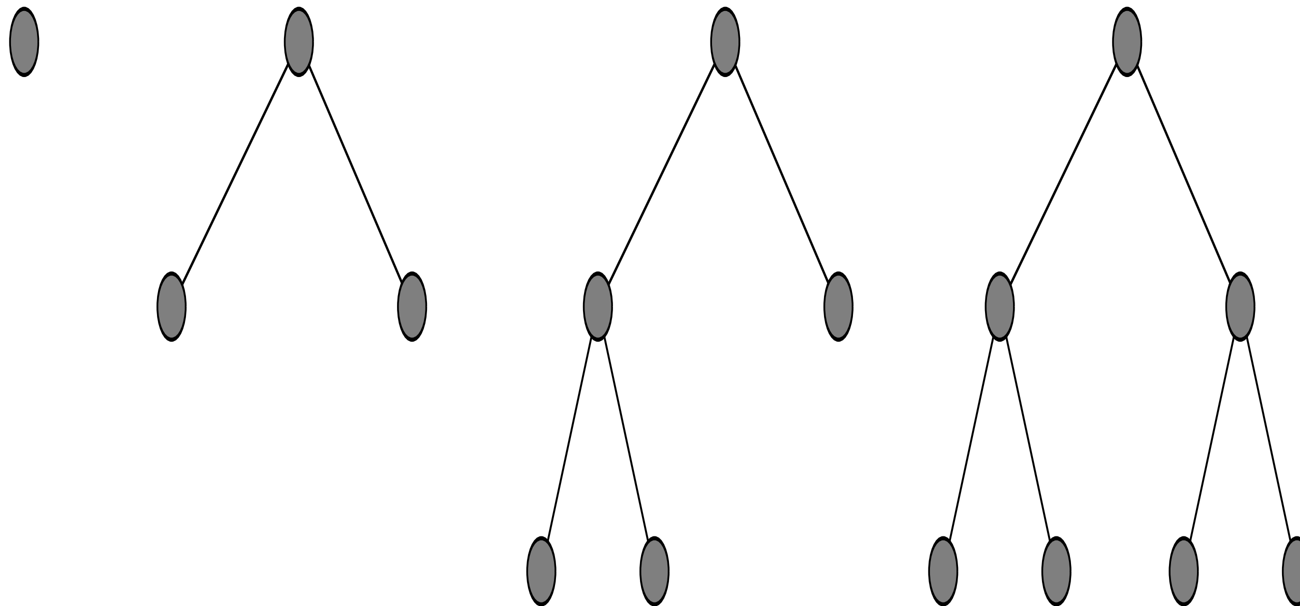
Στρατηγικές Απληροφόρητης Αναζήτησης

- Αναζήτηση πρώτα σε πλάτος (breadth-first search)
- Αναζήτηση ομοιόμορφου κόστους (uniform-cost search)
- Αναζήτηση πρώτα σε βάθος (depth-first search)
- Αναζήτηση περιορισμένου βάθους (depth-limited search)
- Αναζήτηση πρώτα σε βάθος με επαναληπτική εκβάθυνση (iterative deepening depth-first search)
- Αμφίδρομη αναζήτηση (bidirectional search)

Αναζήτηση Πρώτα σε Πλάτος (BFS)

function BREADTHFIRSTSEARCH(*problem*)
returns a solution or failure
  return TREESEARCH(*problem*,ENQUEUEATEND)

Παράδειγμα:



Αποτίμηση της Αναζήτησης Πρώτα σε Πλάτος

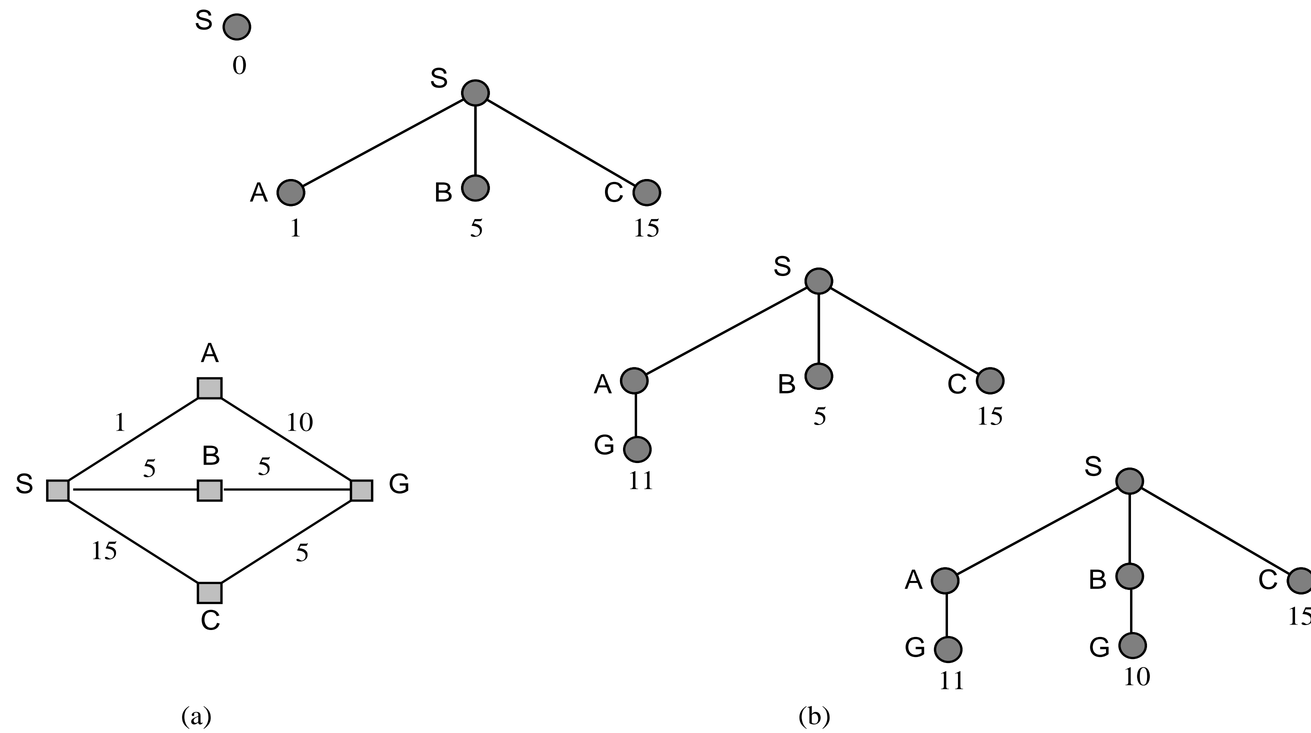
- **Πλήρης;** Ναι, αν ο **παράγοντας διακλάδωσης (branching factor)** b είναι πεπερασμένος.
- **Χρόνος:** $O(b^{d+1})$ όπου b είναι ο παράγοντας διακλάδωσης και d είναι το **βάθος (depth)** της λύσης.
- **Χώρος:** $O(b^{d+1})$. Αυτό είναι το μεγαλύτερο πρόβλημα του BFS.
- **Βέλτιστος;** Ναι, αν όλες οι ενέργειες έχουν **το ίδιο μη αρνητικό** κόστος.

Σημείωση: Ο BFS βρίσκει μία από τις **πιο ρηχές (shallow)** καταστάσεις στόχου.

Αναζήτηση Ομοιόμορφου Κόστους (UCS)

Η αναζήτηση ομοιόμορφου κόστους (UCS) είναι μια παραλλαγή του BFS η οποία επεκτείνει τον κόμβο του συνόρου που έχει το **χαμηλότερο κόστος** (υπολογισμένο με το κόστος μονοπατιού).

UCS - Παράδειγμα



(a) (b)

Αποτίμηση της Αναζήτησης Ομοιόμορφου Κόστους

- **Πλήρης;** Ναι, υπό τις συνθήκες που δίνονται παρακάτω.
- **Χρόνος:** $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ όπου b είναι ο **παράγοντας διακλάδωσης**, C^* είναι το κόστος της βέλτιστης λύσης και κάθε ενέργεια κοστίζει τουλάχιστον $\epsilon > 0$.
- **Χώρος:** το ίδιο με το χρόνο.
- **Βέλτιστος;** Ναι, υπό τις συνθήκες που δίνονται παρακάτω.

Αποτίμηση της Αναζήτηση Ομοιόμορφου Κόστους

Η **πληρότητα** και η **βέλτιστη συμπεριφορά** του UCS ισχύουν υπό τις ακόλουθες συνθήκες:

- Ο παράγοντας διακλάδωσης είναι πεπερασμένος.
- Το κόστος δε μειώνεται ποτέ καθώς προχωράμε σ' ένα μονοπάτι, δηλαδή $g(\textsc{Successor}(n)) \geq g(n)$ για κάθε κόμβο n. Αυτή η συνθήκη ισχύει π.χ., όταν κάθε ενέργεια κοστίζει τουλάχιστον $\epsilon > 0$.

Επειδή ισχύει η δεύτερη συνθήκη, ο UCS **επεκτείνει κόμβους κατά σειρά αυξανόμενου κόστους μονοπατιού. Έτσι, ο πρώτος κόμβος στόχος που επιλέγεται για επέκταση είναι η βέλτιστη λύση.**

Σύγκριση του UCS με τον BFS

- Ο BFS είναι ο UCS με $g(n) = \text{DEPTH}(n)$.
- Ο UCS λειτουργεί ακριβώς όπως ο BFS όταν το κόστος κάθε ενέργειας είναι το ίδιο $\epsilon > 0$.

Σύγκριση του UCS με τον Αλγόριθμο του Dijkstra

Ο UCS μας θυμίζει τον **αλγόριθμο του Dijkstra** για την εύρεση των συντομότερων μονοπατιών μοναδικής πηγής (single-source shortest paths) σε κατευθυνόμενους γράφους με μη αρνητικά βάρη στις ακμές.

Σημαντικές διαφορές:

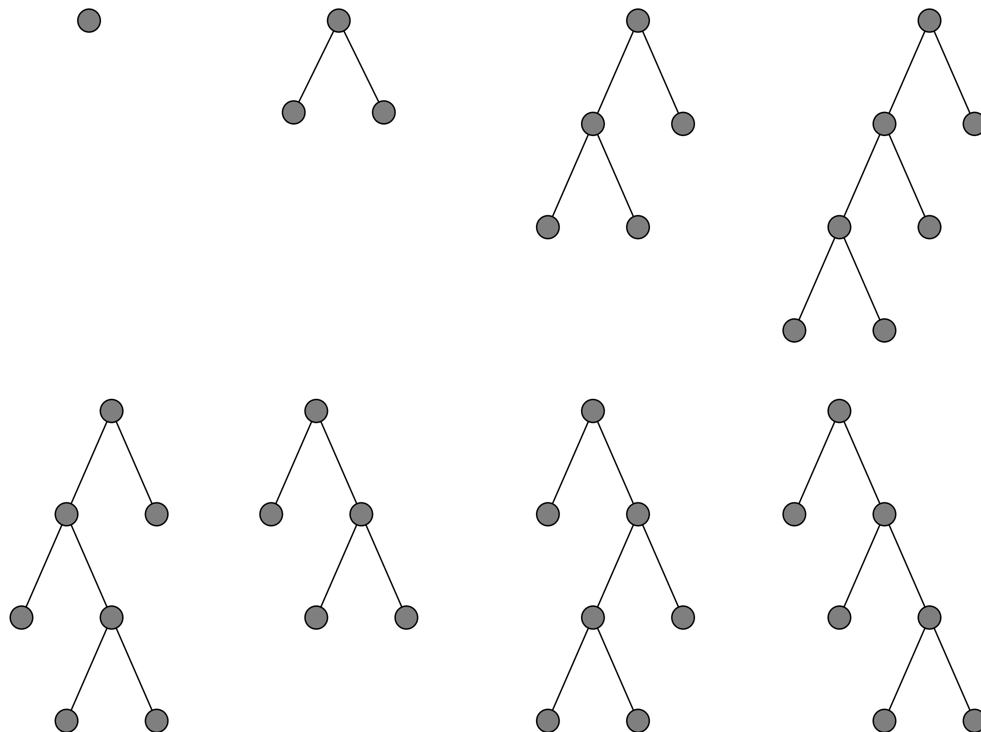
- Στον UCS η αναζήτηση διακόπτεται όταν βρεθεί ένας κόμβος στόχος.
- Στον UCS μπορεί να έχουμε αρκετούς κόμβους για την ίδια κατάσταση, αντί για αρκετές ενημερώσεις του κόστους μιας κατάστασης.
- Η χρονική και χωρική πολυπλοκότητα του UCS είναι διαφορετική διότι οι παράμετροι που μας ενδιαφέρουν είναι διαφορετικές.

Παρατηρήστε το **κλάδεμα** που γίνεται από τον UCS. Κλαδεύουμε τμήματα του δέντρου τα οποία δεν χρειάζεται να εξετάσουμε.

Αναζήτηση Πρώτα σε Βάθος (DFS)

Η αναζήτηση πρώτα σε βάθος πάντα επεκτείνει έναν από τους κόμβους στο **βαθύτερο επίπεδο** του δέντρου αναζήτησης.

Παράδειγμα:



Αναζήτηση Πρώτα σε Βάθος (DFS)

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

Αποτίμηση της Αναζήτησης Πρώτα σε Βάθος

- **Πλήρης;** Όχι
- **Χρόνος:** $O(b^m)$ όπου b είναι ο **παράγοντας διακλάδωσης** και m είναι το μέγιστο **βάθος** του δέντρου αναζήτησης.
- **Χώρος:** $O(bm)$. Αυτό είναι το μεγάλο προσόν του DFS.
- **Βέλτιστος;** Όχι

Αναζήτηση Περιορισμένου Βάθους (DLS)

Ο αλγόριθμος αυτός είναι παρόμοιος με τον DFS αλλά επιβάλλει ένα **όριο βάθους (depth limit)** στην αναζήτηση. Π.χ., για το παράδειγμα εύρεσης διαδρομής προς το Βουκουρέστι, ένα καλό όριο βάθους είναι το 19 (έχουμε 20 πόλεις).

Αποτίμηση:

- **Πλήρης;** Ναι, αν $l \geq d$ όπου l είναι το όριο βάθους και d είναι το βάθος της λύσης.
- **Χρόνος:** $O(b^l)$
- **Χώρος:** $O(bl)$
- **Βέλτιστος;** Όχι

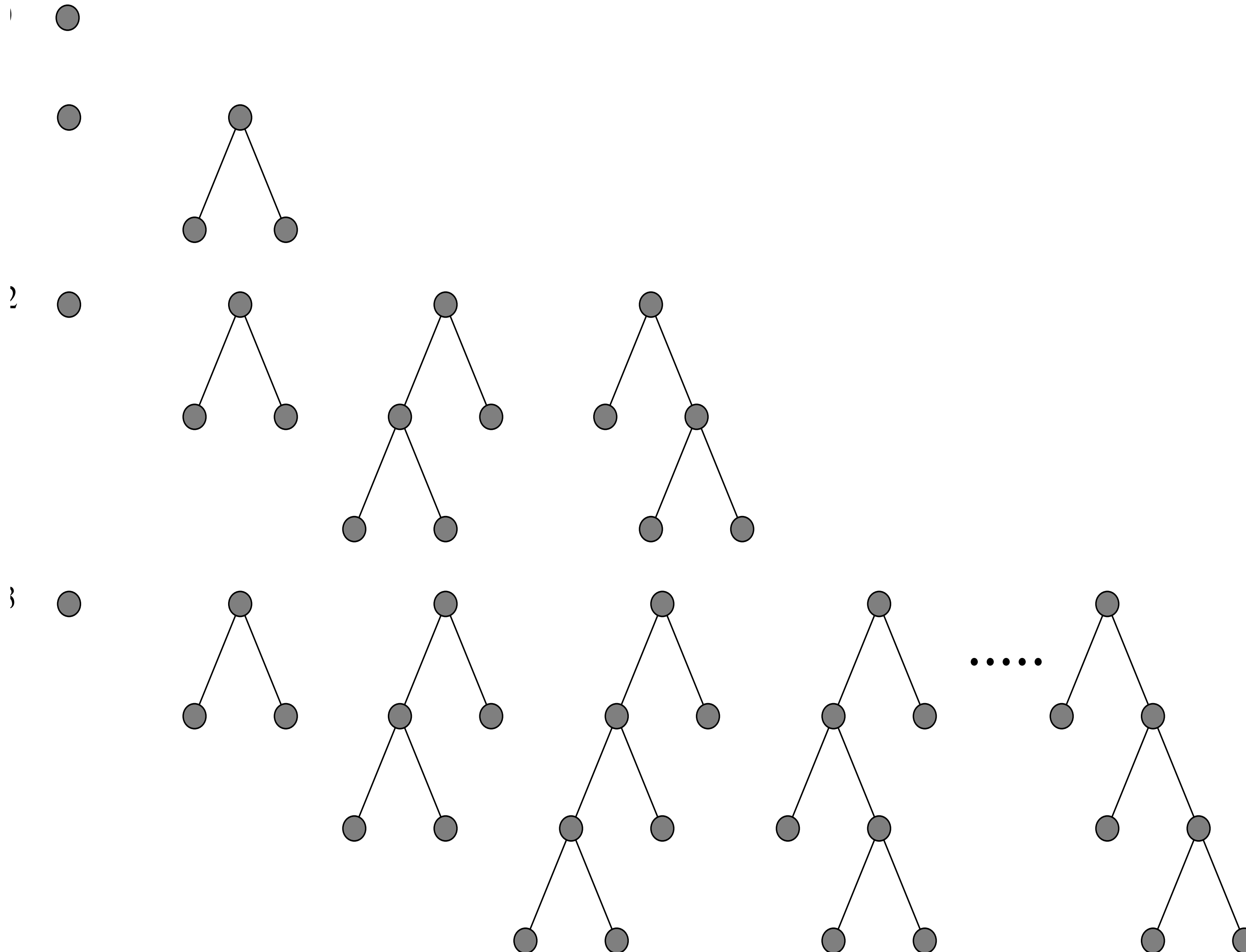
Ερώτηση: Μπορούμε πάντα να βρούμε ένα καλό όριο βάθους;

Αναζήτηση Επαναληπτικής Εκβάθυνσης (IDS)

Ο IDS παρακάμπτει το θέμα της επιλογής του καλύτερου ορίου βάθους δοκιμάζοντας **όλα τα πιθανά: 0,1,2 κλπ.**

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

IDS - Παράδειγμα



Αποτίμηση

Ερώτηση: Είναι ο IDS σπάταλος;

Απάντηση: Όχι!

Ας υποθέσουμε ότι βρίσκουμε μια λύση όταν εξετάζεται ο τελευταίος κόμβος στο επίπεδο d. Τότε ο αριθμός των κόμβων που δημιουργούνται από μια αναζήτηση με BFS μέχρι το βάθος d είναι

$$1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$$

Ο αριθμός των κόμβων που δημιουργούνται από μια αναζήτηση με τον IDS μέχρι το βάθος d είναι

$$(d+1) + db + (d-1)b^2 + \cdots + 2b^{d-1} + 1b^d$$

Αποτίμηση

Χρησιμοποιώντας τους παραπάνω τύπους μπορούμε να δούμε ότι ο BFS μπορεί ουσιαστικά να είναι πολύ πιο σπάταλος από τον IDS.

Για παράδειγμα, για $b = 10$ και $d = 5$, **ο BFS παράγει 1.111.100 κόμβους και ο IDS 123.450 κόμβους**.

Αποτίμηση

- **Πλήρης;** Ναι, υπό τις προϋποθέσεις του BFS.
- **Χρόνος:** $O(b^d)$
- **Χώρος:** $O(bd)$
- **Βέλτιστος;** Ναι, υπό τις προϋποθέσεις του BFS.

Ο IDS είναι ο πιο κατάλληλος απληροφόρητος αλγόριθμος αναζήτησης όταν ο χώρος αναζήτησης είναι μεγάλος και το βάθος αναζήτησης είναι άγνωστο.

Αμφίδρομη Αναζήτηση

Στην αμφίδρομη αναζήτηση, ψάχνουμε τόσο προς τα εμπρός ξεκινώντας από την αρχική κατάσταση, όσο και προς τα πίσω ξεκινώντας από μια κατάσταση στόχου. Σταματάμε όταν οι δύο αναζητήσεις συναντηθούν.

Κίνητρο: $b^{d/2} + b^{d/2} << b^d$

Προβλήματα:

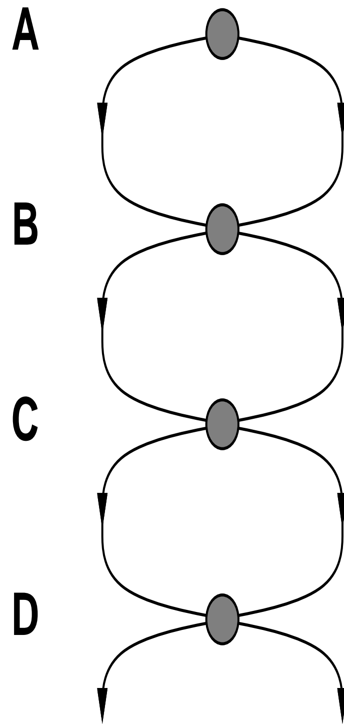
- Τι σημαίνει αναζήτηση προς τα πίσω ξεκινώντας από την κατάσταση στόχου;
- Τι γίνεται όταν έχουμε περισσότερες από μια καταστάσεις στόχου;
- Μπορούμε να ελέγχουμε αποδοτικά ότι οι δύο αναζητήσεις θα συναντηθούν;
- Τι είδος αναζήτησης εκτελούμε προς τα εμπρός και τι προς τα πίσω;

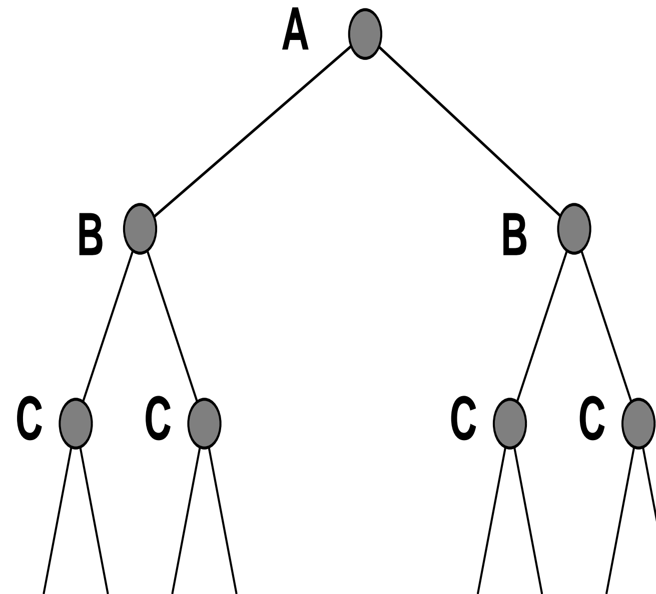
Αποτίμηση της Αμφίδρομης Αναζήτησης

- **Πλήρης;** Ναι, αν ο παράγοντας διακλάδωσης είναι πεπερασμένος και αμφότερες οι αναζητήσεις χρησιμοποιούν BFS.
- **Χρόνος:** $O(b^{d/2})$
- **Χώρος:** $O(b^{d/2})$
- **Βέλτιστος;** Ναι, αν οι αναζητήσεις χρησιμοποιούν BFS και ισχύουν οι προϋποθέσεις βέλτιστης συμπεριφοράς για τον BFS.

Αποφυγή Επαναλαμβανόμενων Καταστάσεων

Παράδειγμα:





Αποφυγή Επαναλαμβανόμενων Καταστάσεων

- Στην περίπτωση αυτή ο χώρος καταστάσεων είναι ένας **γράφος.**
- Μια λύση είναι να αποφύγουμε τη δημιουργία οποιασδήποτε κατάστασης που έχει παραχθεί προηγουμένως. Αυτό μπορεί να επιτευχθεί κρατώντας μια λίστα των καταστάσεων που έχουν ήδη παραχθεί και λέγεται **κλειστή λίστα (closed list)**. Στην περίπτωση αυτή το σύνορο των κόμβων που δεν έχουν επεκταθεί λέγεται **ανοιχτή λίστα (open list)**.

  Η κλειστή λίστα μπορεί να υλοποιηθεί με ένα **πίνακα κατακερματισμού** για ανάκτηση σε σταθερό χρόνο. Ωστόσο, δεν υπάρχει εύκολος τρόπος να αποφύγουμε την απώλεια χώρου!

Ο Γενικός Αλγόριθμος Αναζήτησης σε Γράφους

```
function GraphSearch(problem,QueuingFn)
returns a solution, or failure
    closed ← an empty set
    fringe ← MakeQueue(MakeNode(InitialState[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 ← QueuingFn(Expand(node, problem), fringe)
    end
```

Μελέτη

Κεφάλαιο 3, Ενότητες 3.1-3.5 του βιβλίου AIMA.