

# Τεχνητή Νοημοσύνη

## Στρατηγικές Απληροφόρητης Αναζήτησης

Δρ. Δημήτριος Κουτσομητρόπουλος

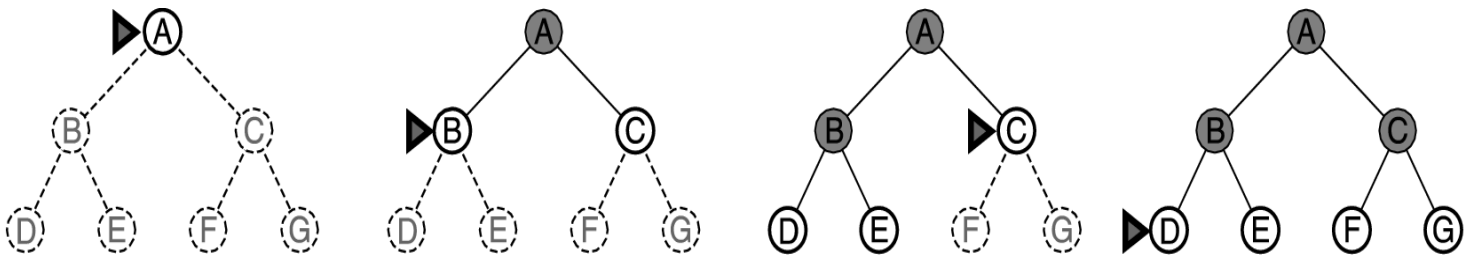
## Αλγόριθμοι αναζήτησης λύσης

---

### Απληροφόρητοι Αλγόριθμοι Αναζήτησης

- ▶ **Breadth-first Search (BFS)**  
Αναζήτηση Πρώτα σε πλάτος
- ▶ **Uniform-cost Search (UCS)**  
Αναζήτηση Ομοιόμορφου κόστους
- ▶ **Depth-first Search (DFS)**  
Αναζήτηση Πρώτα σε βάθος
- ▶ **Iterative deepening Search (IDS)**  
Αναζήτηση επαναληπτικής εκβάθυνσης
- ▶ **Bidirectional Search**  
Αμφίδρομη αναζήτηση

# Αναζήτηση πρώτα κατά πλάτος (Breadth-first search)



- ▶ Στιγμιότυπο του Graph-Search
  - ▶ Με ποια σειρά επεκτείνονται οι κόμβοι;
- ▶ **Επέκταση πρώτα του ρηχότερου** μη εκτεταμένου κόμβου
- ▶ Μέτωπο αναζήτησης: **Ουρά (FIFO)**
  - ▶ Οι νέοι κόμβοι προστίθενται στο τέλος
- ▶ Ο έλεγχος στόχου γίνεται κατά τη δημιουργία του κόμβου (όχι κατά την επέκτασή του)

3

## BFS Ψευδοκώδικας

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

# Ιδιότητες BFS

- ▶ **Πλήρης;** Ναι (αν το  $b$  πεπερασμένο)
  - ▶ Κάποια στιγμή θα βρει τον ρηχότερο κόμβο-στόχο
- ▶ **Βέλτιστος;** Ναι (αν το κόστος είναι ίδιο σε κάθε βήμα)
  - ▶ Είναι ο ρηχότερος κόμβος ο βέλτιστος;
  - ▶ Κόστος αύξουσα συνάρτηση του βάθους  $d$
- ▶ **Χρόνος;**  $O(b^d)$ 
  - ▶ Αν η λύση είναι σε βάθος  $d$  θα δημιουργηθούν και θα ελεγχθούν  $1+b+b^2+b^3+\dots+b^d$  κόμβοι
  - ▶ Έλεγχος πριν την επέκταση (αλλιώς:  $+b^{d+1}$ )
- ▶ **Χώρος;**  $O(b^d)$  (κρατά όλους τους κόμβους στη μνήμη) **μειονέκτημα**
  - ▶  $O(b^d)$  στο μέτωπο και  $O(b^{d-1})$  στο κλειστό σύνολο

## ▶ Tree-Search εκδοχή;

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	$10^6$	1.1 seconds	1 gigabyte
8	$10^8$	2 minutes	103 gigabytes
10	$10^{10}$	3 hours	10 terabytes
12	$10^{12}$	13 days	1 petabyte
14	$10^{14}$	3.5 years	99 petabytes
16	$10^{16}$	350 years	10 exabytes

**Figure 3.13** Time and memory requirements for breadth-first search. The numbers shown assume branching factor  $b = 10$ ; 1 million nodes/second; 1000 bytes/node.

▶ 5

# Αναζήτηση ομοιόμορφου κόστους (Uniform-cost search)

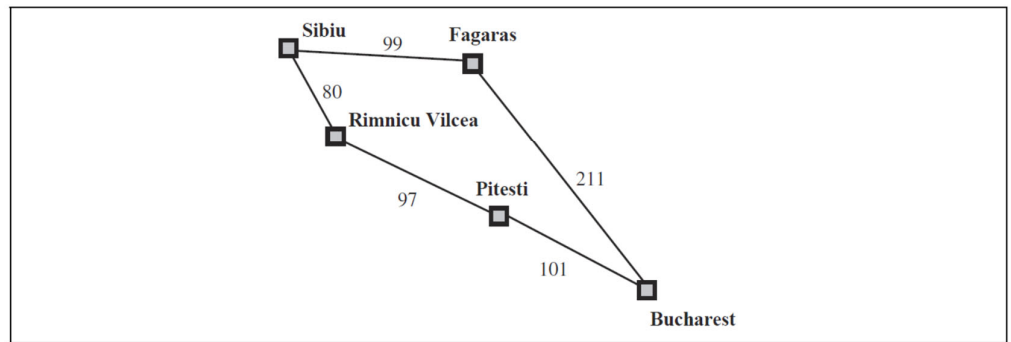
- ▶ Παραλλαγή του BFS
  - ▶ Στον BFS οι κόμβοι επεκτείνονται με σειρά βάθους (πρώτα ο μικρότερος)
  - ▶ Έτσι βρίσκει τον ρηχότερο κόμβο-στόχο
  - ▶ Αν όμως το κόστος ανεξάρτητο του βάθους;
- ▶ UCS
  - ▶ Στον UCS οι κόμβοι επεκτείνονται με σειρά κόστους μονοπατιού  $g(n)$
- ▶ Μέτωπο Αναζήτησης: **Διάταξη**
  - ▶ με βάση το κόστος μονοπατιού μέχρι τον κόμβο
- ▶ **Επέκταση πρώτα του φθηνότερου** μη εκτεταμένου κόμβου
- ▶ **Όμως:**
  - ▶ Έλεγχος στόχου κατά την **επιλογή για επέκταση** του κόμβου (Όχι κατά τη δημιουργία του)
    - ▶ Ο πρώτος κόμβος-στόχος που δημιουργείται μπορεί να βρίσκεται σε μη βέλτιστο μονοπάτι (στον BFS τι ισχύει;;)
  - ▶ Προσθήκη ελέγχου αν υπάρχει καλύτερο μονοπάτι για τον τρέχοντα κόμβο του μετώπου

▶ 6

# UCS Ψευδοκώδικας

```

function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
    
```

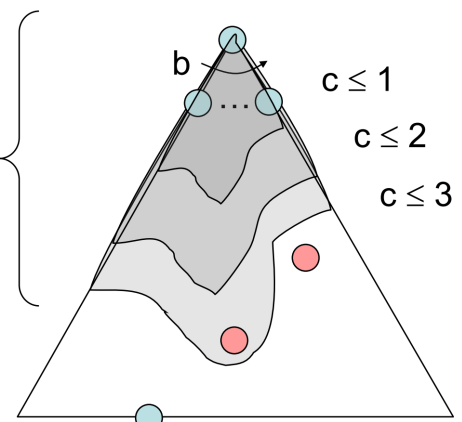


▶ 7

## Ιδιότητες UCS

- ▶ **Βέλτιστος;** Ναι (αν το κόστος θετικό σε κάθε βήμα)
  - ▶ Ένας κόμβος προς επέκταση στο μέτωπο βρίσκεται ήδη στο φθηνότερο μονοπάτι του
- ▶ **Πλήρης;** Ναι
  - ▶ Εκτός αν υπάρχουν ατέρμονα μονοπάτια μηδενικών βημάτων (NoOp)
- ▶ **Χρόνος;**  $O(b^{1+ \lfloor C^*/\epsilon \rfloor})$ 
  - ▶  $C^*$  το κόστος της βέλτιστης λύσης
  - ▶  $\epsilon$  το κόστος του φθηνότερου βήματος
  - ▶  $C^*/\epsilon$  το «βάθος» του μονοπατιού στη χειρότερη περίπτωση (Μπορεί  $\gg d$ )
  - ▶ Σε κάθε βήμα, έχει επεκτείνει τον γονέα του και έχουν δημιουργηθεί όλα τα αδέρφια του ( $b$ )
  - ▶ Μπορεί να ξαναγυρνά πίσω σε μονοπάτια με πολλά μικρά βήματα
  - ▶ Γιατί  $+1$ ; Ποια η σχέση με BFS;
- ▶ **Χώρος;**  $O(b^{1+ \lfloor C^*/\epsilon \rfloor})$

$C^*/\epsilon$  "tiers"

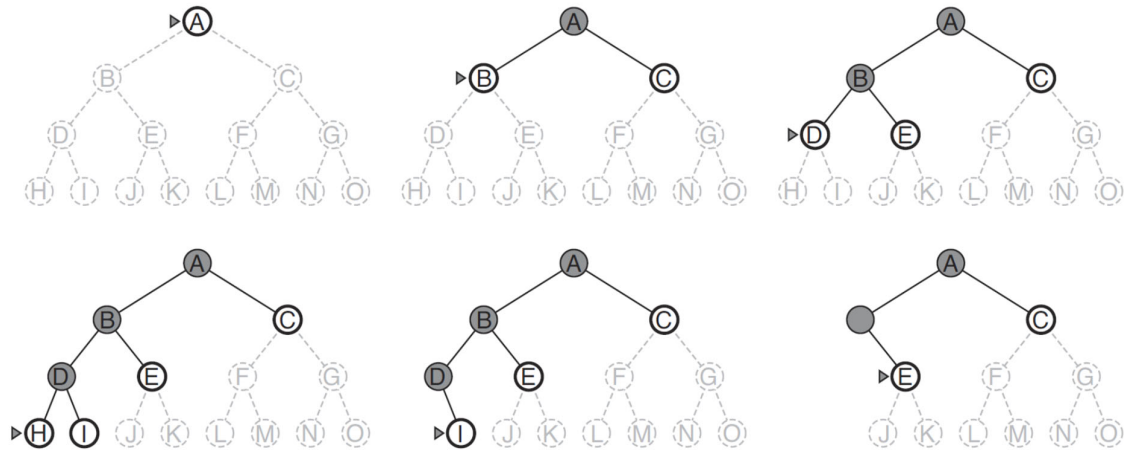


Πηγή: ai.berkeley.edu

▶ 8

# Αναζήτηση πρώτα κατά βάθος (Depth-first search)

- ▶ Στιγμιότυπο του Graph-Search ή του Tree-Search
- ▶ **Επέκταση πρώτα του βαθύτερου** κόμβου στο μέτωπο
  - ▶ Επεκτείνεται αυτός που δημιουργήθηκε τελευταία
  - ▶ Είναι ο βαθύτερος γιατί είναι πιο κάτω από τον γονέα του, ο οποίος ήταν πριν ο βαθύτερος κόκ
- ▶ Μέτωπο αναζήτησης: **Στοιίβα (LIFO)**
  - ▶ Οι νέοι κόμβοι προστίθενται στην αρχή



▶ 9

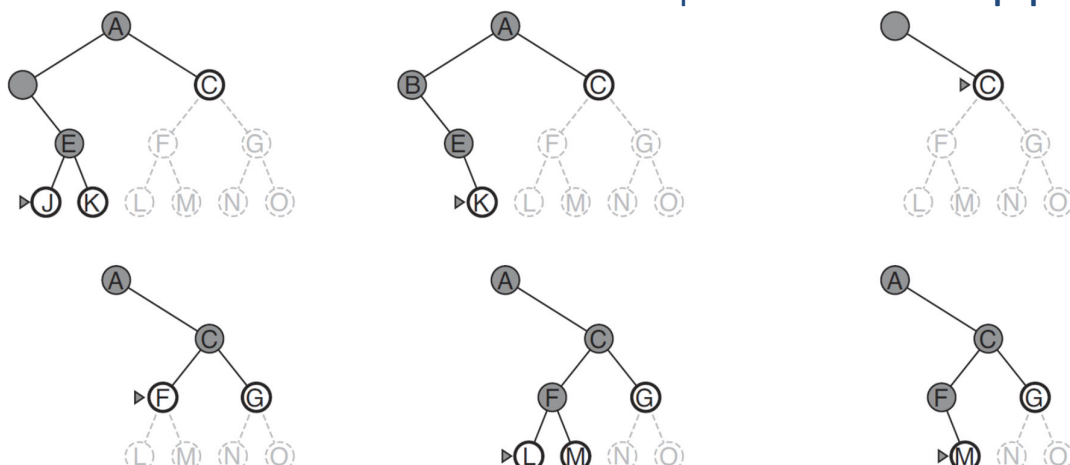
# Αναζήτηση πρώτα κατά βάθος (DFS)

## Graph-Search εκδοχή

- ▶ **Πλήρης; Ναι**
  - ▶ Στο τέλος θα επεκτείνει όλους τους κόμβους
- ▶ **Βέλτιστος; Όχι**
  - ▶ Θα προτιμήσει τη βαθύτερη λύση πρώτα (π.χ. J vs. C)
- ▶ **Χρόνος; Το μέγεθος του χώρου καταστάσεων,  $|S|$**

## Tree-Search εκδοχή

- ▶ **Πλήρης; Όχι**
  - ▶ Μπορεί να εγκλωβιστεί σε κυκλικά μονοπάτια
- ▶ **Βέλτιστος; Όχι**
  - ▶ Θα προτιμήσει τη βαθύτερη λύση πρώτα (π.χ. J vs. C)
- ▶ **Χρόνος;  $O(b^m)$** 
  - ▶  $m$  μέγιστο βάθος στο δέντρο. Μπορεί  $m \gg d$  και  $b^m \gg |S|$



▶ 10

## Αναζήτηση πρώτα κατά βάθος (DFS) – Χώρος

---

- ▶ Graph-Search εκδοχή: Όλοι οι κόμβοι στη μνήμη
  - ▶ Όπως και BFS (μέτωπο + κλειστό σύνολο)
- ▶ Tree-Search εκδοχή
  - ▶ Δεν υπάρχει κλειστό σύνολο!
  - ▶ Όταν ένας κόμβος επεκταθεί, αφαιρείται από το μέτωπο
  - ▶ Κάθε κόμβος κρατά δείκτη στον γονέα του κοκ μέχρι τη ρίζα
  - ▶ Αν όλοι οι απόγονοι αφαιρεθούν, αφαιρείται από τη μνήμη
    - ▶ Κανένας δε δείχνει σε αυτόν
  - ▶ **Χώρος:**  $O(bm)$ 
    - ▶  $b$  κόμβοι σε κάθε βήμα του μονοπατιού προς τη ρίζα
  - ▶ Το  $m$  μπορεί να είναι **άπειρο**
    - ▶ Κυκλικά μονοπάτια
    - ▶ Άπειρος χώρος καταστάσεων

---

▶ 11

## Αναζήτηση περιορισμένου βάθους (Depth-limited search, DLS)

---

- ▶ Τίθεται όριο βάθους  $L < m$ 
  - ▶ Λύνεται το πρόβλημα των άπειρων διαδρομών
- ▶ Αν  $L < d$ :
  - ▶ Δεν είναι **πλήρης**: Δεν θα φτάσει ποτέ στη λύση
- ▶ Αν  $L > d$ :
  - ▶ Μπορεί να μην είναι **βέλτιστος**: Θα προτιμήσει βαθύτερη λύση
- ▶ Χρόνος:  $O(b^L)$ , Χώρος:  $O(bL)$
- ▶ Επιλογή  $L$  με γνώση του προβλήματος (π.χ. διάμετρος του χώρου καταστάσεων)
  - ▶ 9 βήματα μέγιστο προς οποιαδήποτε άλλη πόλη

---

▶ 12

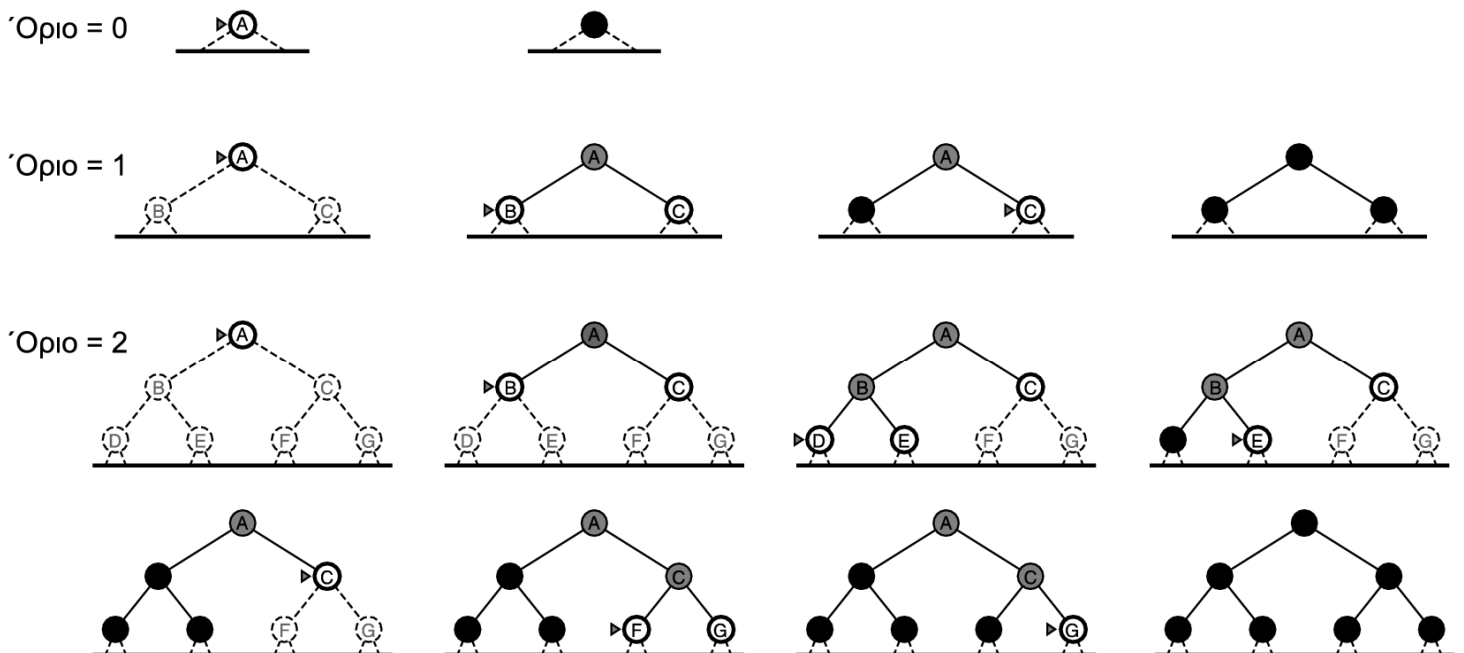
# Επαναληπτική εκβάθυνση (1/3)

## (Iterative-deepening search)

- ▶ **Ιδέα:** Σταδιακή αύξηση του ορίου βάθους
  - ▶ Πρώτα 0, μετά 1, ..., μέχρι  $\infty$
  - ▶ Μέχρι να βρει λύση
- ▶ Πλεονεκτήματα **BFS+DFS** μαζί
- ▶ **Πλήρης:** Ναι, εφόσον  $b$  πεπερασμένο (όπως **BFS**)
  - ▶ Κυκλικά μονοπάτια OK!
- ▶ **Βέλτιστος:** Ναι, εφόσον κόστος ανάλογο του βάθους
  - ▶ Θα βρει την ρηχότερη λύση (όπως **BFS**), εξ ορισμού
- ▶ **Χώρος:**  $O(bd)$ ,  $d$  το βάθος της λύσης (όπως **DFS**)
- ▶ **Χρόνος:**  $O(b^d)$ 
  - ▶ Αν η λύση είναι σε βάθος  $d$  θα τρέξει  $d$  φορές
    - ▶ Πόση σπατάλη γίνεται; Τα μικρότερα βάθη θα δημιουργηθούν πολλές φορές, ενώ το επίπεδο  $d$  μόνο μία.
    - ▶ Οι «πολλοί» κόμβοι είναι στα βαθιά επίπεδα:
    - ▶  $N(IDS) = (d)b + (d-1)b^2 + \dots + (1)b^d = O(b^d)$

# Επαναληπτική εκβάθυνση (2/3)

## (Iterative-deepening search)

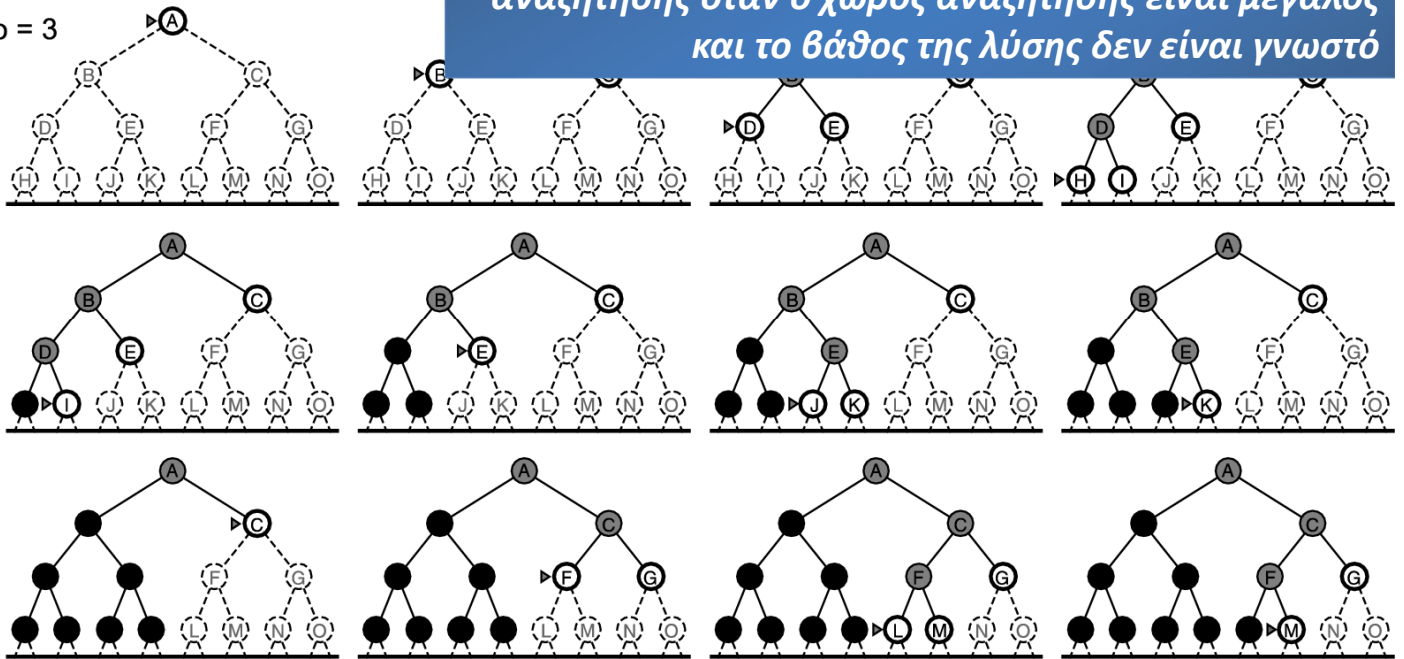


# Επαναληπτική εκβάθυνση (3/3)

## (Iterative-deepening search)

είναι η προτιμότερη μέθοδος απληροφόρητης αναζήτησης όταν ο χώρος αναζήτησης είναι μεγάλος και το βάθος της λύσης δεν είναι γνωστό

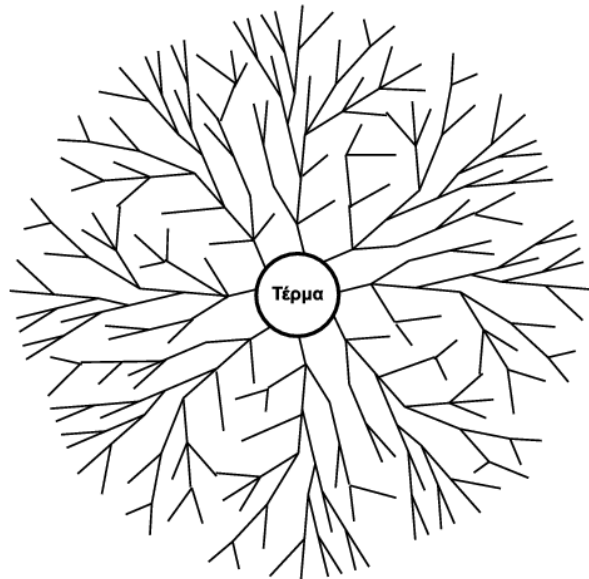
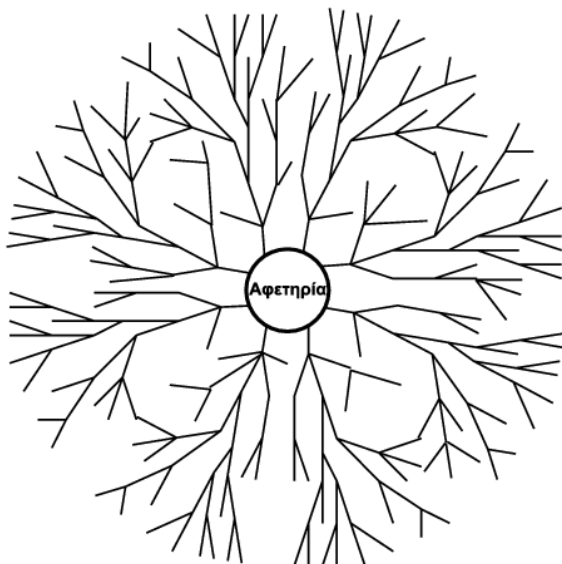
Όριο = 3



▶ 15

# Αμφίδρομη αναζήτηση (1/2)

- ▶ Ιδέα: BFS αναζήτηση από την αρχή και το τέλος **ταυτόχρονα**
- ▶ Αν τα δύο μέτωπα **τέμνονται** τότε βρέθηκε λύση
- ▶ Στη χειρότερη περίπτωση, θα συναντηθούν **στη μέση** (γιατί;)
- ▶  $b^{d/2} + b^{d/2} \ll b^d$
- ▶ Επιπλέον (σταθερός) χρόνος αν αυτό είναι το βέλτιστο σημείο τομής



▶ 16



## Αμφίδρομη αναζήτηση (2/2)

- ▶ Χρονική πολυπλοκότητα:  $O(b^{d/2})$
- ▶ Χωρική πολυπλοκότητα:  $O(b^{d/2})$  (μειονέκτημα)
  - ▶ Ακόμα και με DFS, ένα από τα δύο μέτωπα θα πρέπει να παραμένει στη μνήμη για να βρεθεί η κοινή κατάσταση
- ▶ Προβλήματα:
  - ▶ Εύρεση προκατόχων καταστάσεων (predecessors)
    - ▶ Οι τελεστές μετάβασης είναι αντιστρέψιμοι;
  - ▶ Πώς υλοποιείται αναζήτηση προς τα πίσω από το στόχο;
    - ▶ Έλλειψη καλά καθορισμένων στόχων (π.χ. n-queens)

▶ 17

## Αλγόριθμοι απληροφόρητης αναζήτησης

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>	No	No	Yes <sup>a</sup>	Yes <sup>a,d</sup>
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes <sup>c</sup>	Yes	No	No	Yes <sup>c</sup>	Yes <sup>c,d</sup>

a) Πλήρης αν  $b$  πεπερασμένο

b) Πλήρης αν κόστη βημάτων θετικά

c) Βέλτιστος αν κόστος διαδρομής είναι αύξουσα συνάρτηση του βάθους

d) όταν και οι δύο κατευθύνσεις είναι BFS