Αριστοτέλειο
Πανεπιστήμιο
Θεσσαλονίκης

Τεχνητή Νοημοσύνη

Αλγόριθμοι Τυφλής Αναζήτησης

Ιώαννης Βλαχάβας

Τμήμα Πληροφορικής ΑΠΘ

Θεσσαλονίκη, 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 Ακαδημαϊκά Μαθήματα στο Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

ανοικτάμαθήματα
opencourses

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 & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Αλγόριθμοι Τυφλής Αναζήτησης

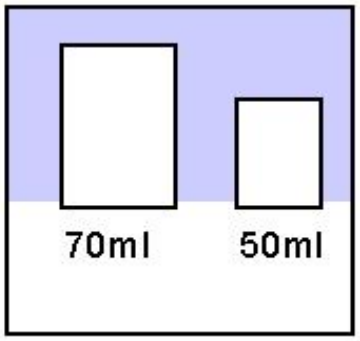
Αλγόριθμοι Τυφλής Αναζήτησης

- Οι *αλγόριθμοι τυφλής αναζήτησης* (*blind search algorithms*) εφαρμόζονται σε προβλήματα στα οποία δεν υπάρχει πληροφορία που να επιτρέπει την αξιολόγηση των καταστάσεων του χώρου αναζήτησης (Δηλαδή ψάχνουμε στα τυφλά).
- Έτσι οι αλγόριθμοι αυτοί αντιμετωπίζουν με τον ίδιο ακριβώς τρόπο οποιοδήποτε πρόβλημα καλούνται να λύσουν.
- Για τους αλγορίθμους τυφλής αναζήτησης, το τι απεικονίζει κάθε κατάσταση του προβλήματος είναι παντελώς αδιάφορο. Σημασία έχει η χρονική σειρά με την οποία παράγονται οι καταστάσεις από το μηχανισμό επέκτασης.

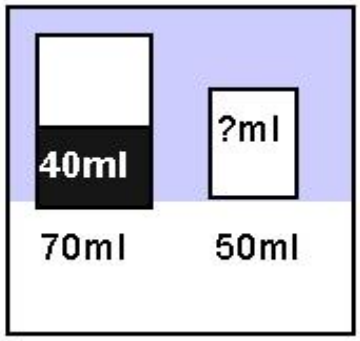
Όνομα Αλγορίθμου	Συντομογραφία	Ελληνική Ορολογία
Depth-First Search	**DFS**	Αναζήτηση Πρώτα σε Βάθος
Breadth-First Search	**BFS**	Αναζήτηση Πρώτα σε Πλάτος
Iterative Deepening	**ID**	Επαναληπτική Εκβάθυνση
Bi-directional Search	**BiS**	Αναζήτηση Διπλής Κατεύθυνσης
Branch and Bound	**B&B**	Επέκταση και Οριοθέτηση

Παράδειγμα

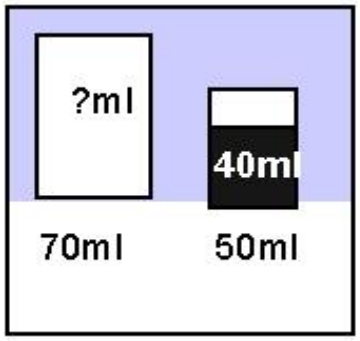
Το πρόβλημα των ποτηριών



Αρχική Κατάσταση



ή



Τελικές Καταστάσεις

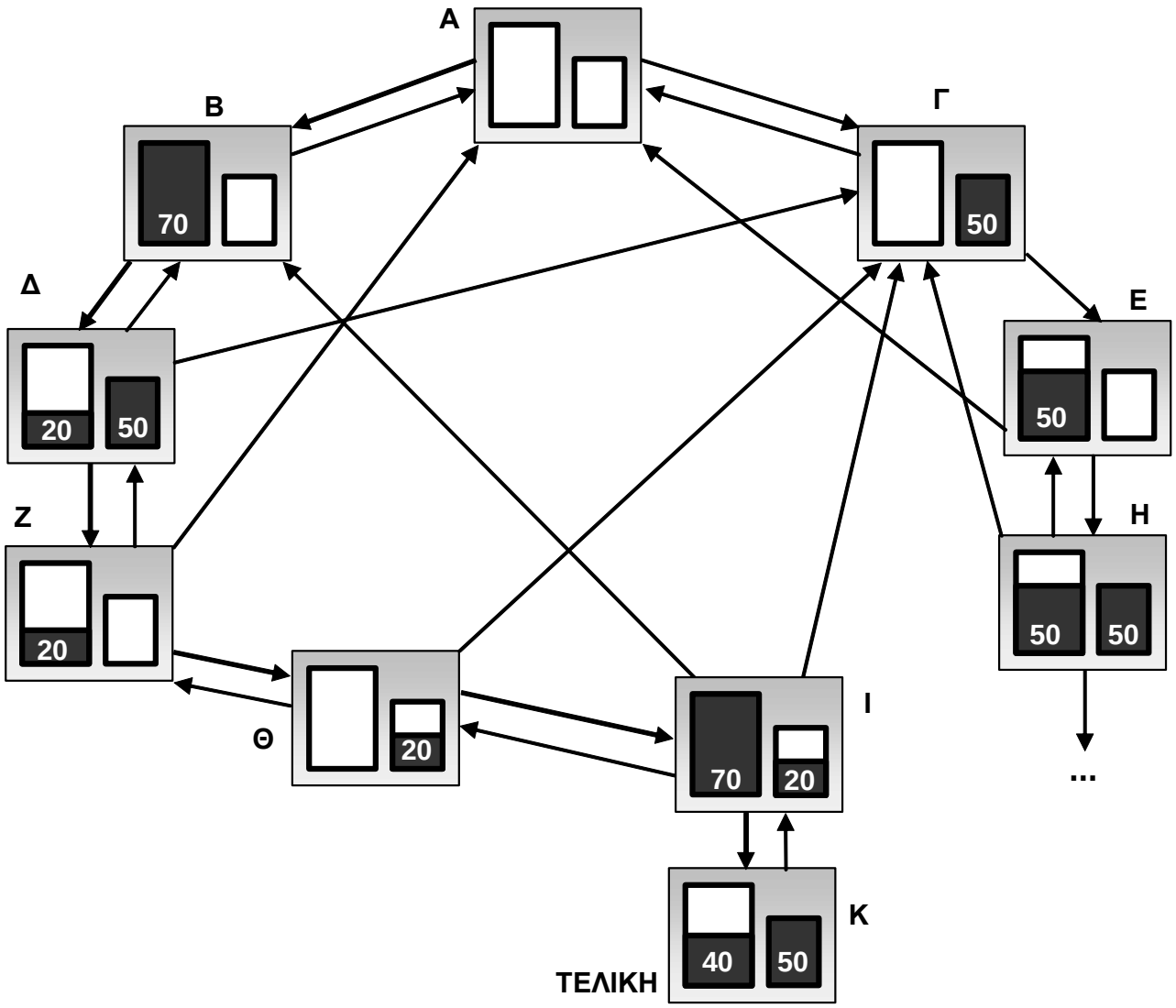
Τελεστής (1)
Γέμισε το ποτήρι των X ml μέχρι το χείλος από τη βρύση
Προϋποθέσεις
Το ποτήρι των X ml έχει 0 ml
Αποτελέσματα
Το ποτήρι των X ml έχει X ml

Τελεστής (2)
Γέμισε το ποτήρι των X ml από το ποτήρι των Y ml
Προϋποθέσεις
Το ποτήρι των X ml έχει Z ml Το ποτήρι των Y ml έχει W ml (W≠0)
Αποτελέσματα
Το ποτήρι των X ml έχει X ml και Το ποτήρι των Y ml έχει W-(X-Z), αν W≥X-Z ή Το ποτήρι των X ml έχει Z+W ml και Το ποτήρι των Y ml έχει 0, αν W<X-Z

Τελεστής (3)
Άδειασε το ποτήρι των X ml στο νεροχύτη
Προϋποθέσεις
Το ποτήρι έχει περιεχόμενο
Αποτελέσματα
Το ποτήρι των X ml έχει 0 ml

Μέρος του χώρου αναζήτησης

στο πρόβλημα με τα ποτήρια



Αναζήτηση Πρώτα σε Βάθος

Ο αλγόριθμος *πρώτα σε βάθος* (*Depth-First Search - DFS*) επιλέγει προς επέκταση την κατάσταση που βρίσκεται πιο βαθιά στο δένδρο.

- Στην περίπτωση που υπάρχουν περισσότερες από μία καταστάσεις στο ίδιο βάθος, ο DFS επιλέγει τυχαία μία από αυτές ή, για ευκολία, επιλέγει την αριστερότερη.

Ο αλγόριθμος DFS:

1. **Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.**
2. **Αν το μέτωπο της αναζήτησης είναι κενό τότε σταμάτησε.**
3. **Βγάλε την πρώτη κατάσταση από το μέτωπο της αναζήτησης.**
4. **Αν η κατάσταση ανήκει στο κλειστό σύνολο τότε πήγαινε στο βήμα 2.**
5. **Αν η κατάσταση είναι μία από τις τελικές, τότε ανέφερε τη λύση.**
6. **Αν θέλεις και άλλες λύσεις πήγαινε στο βήμα 2. Αλλιώς σταμάτησε.**
7. **Εφάρμοσε τους τελεστές μετάβασης για να βρεις τις καταστάσεις-παιδιά.**
8. **Βάλε τις καταστάσεις-παιδιά στην αρχή του μετώπου της αναζήτησης.**
9. **Βάλε την κατάσταση-γονέα στο κλειστό σύνολο.**
10. **Πήγαινε στο βήμα 2.**

Ο αλγόριθμος DFS (Ψευδοκώδικας)

```
algorithm dfs(InitialState, FinalStates)
begin
  Closed←∅;
  Frontier←<InitialState>;
  CurrentState←First(Frontier);
  while CurrentState ∉ FinalStates do
    Frontier←delete(CurrentState,Frontier);
    if CurrentState ∉ ClosedSet then
    begin
      ChildrenStates ←Expand(CurrentState);
      Frontier←ChildrenStates ^ Frontier;
      Closed←Closed∪{CurrentState};
    end;
    if Frontier=∅ then exit;
    CurrentState←First(Frontier);
  endwhile;
return success;
  end.
```

Αναζήτηση Πρώτα σε Βάθος

Σχόλια

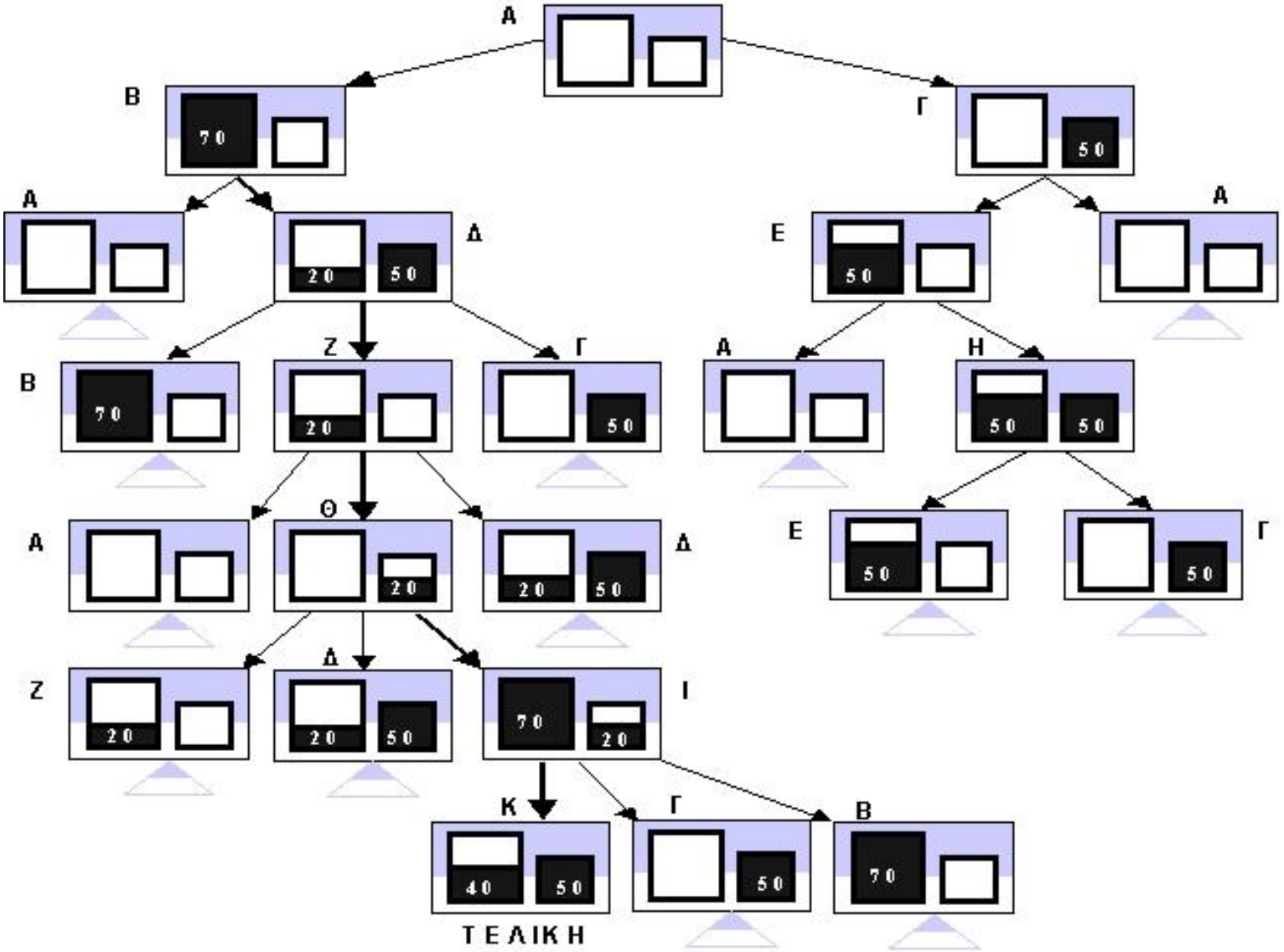
- Το μέτωπο της αναζήτησης είναι μια δομή στοίβας (Stack LIFO, Last In First Out)
- Η εξέταση αμέσως προηγουμένων (χρονικά) καταστάσεων ονομάζεται χρονική οπισθοδρόμηση (*chronological backtracking*).

- Πλεονεκτήματα:
  - Έχει μικρές απαιτήσεις σε χώρο διότι το μέτωπο της αναζήτησης δε μεγαλώνει πάρα πολύ.

- Μειονεκτήματα:
  - Δεν εγγυάται ότι η πρώτη λύση που θα βρεθεί είναι η βέλτιστη (μονοπάτι με το μικρότερο μήκος ή με μικρότερο κόστος).
  - Εν γένει θεωρείται μη-πλήρης (αν δεν υπάρχει έλεγχος βρόχων ή αν ο χώρος αναζήτησης είναι μη πεπερασμένος γιατί μπορεί να μπλεχτεί σε κλαδιά μεγάλου μήκους ή σε ατέρμονα κλαδιά).
  - Στις περιπτώσεις όμως που ο χώρος αναζήτησης είναι πεπερασμένος και χρησιμοποιείται κλειστό σύνολο, ο DFS θα βρει λύση, εάν μια τέτοια υπάρχει.

Αναζήτηση Πρώτα σε Βάθος (DFS)

Δένδρο αναζήτησης στο πρόβλημα των ποτηριών



Αναζήτηση Πρώτα σε Βάθος (DFS)

Πρόβλημα των ποτηριών

Μέτωπο της αναζήτησης	**Κλειστό Σύνολο**	**Κατάσταση**	**Παιδιά**
<Α>	{}	Α	<Β, Γ>
<Β, Γ>	{Α}	Β	<Α, Δ>
<Α, Δ, Γ>	{Α,Β}	Α	- (βρόχος)
<Δ, Γ>	{Α,Β}	Δ	<Β,Ζ,Γ>
<Β,Ζ,Γ,Γ>	{Α,Β,Δ}	Β	- (βρόχος)
<Ζ,Γ,Γ>	{Α,Β,Δ}	Ζ	<Α,Θ,Δ>
<Α,Θ,Δ,Γ,Γ>	{Α,Β,Δ,Ζ}	Α	- (βρόχος)
<Θ,Δ,Γ,Γ>	{Α,Β,Δ,Ζ}	Θ	<Ζ,Δ,Ι>
<Ζ,Δ,Ι,Δ,Γ,Γ>	{Α,Β,Δ,Ζ,Θ}	Ζ	- (βρόχος)
<Δ,Ι,Δ,Γ,Γ>	{Α,Β,Δ,Ζ,Θ}	Δ	- (βρόχος)
<Ι,Δ,Γ,Γ>	{Α,Β,Δ,Ζ,Θ}	Ι	<Κ,Γ,Β>
<Κ,Γ,Β,Δ,Γ,Γ>	{Α,Β,Δ,Ζ,Θ,Ι}	Κ	ΤΕΛΙΚΗ

Αναζήτηση Πρώτα σε Πλάτος

Ο αλγόριθμος *αναζήτησης πρώτα σε πλάτος* (*Breadth First Search - BFS*) εξετάζει πρώτα όλες τις καταστάσεις που βρίσκονται στο ίδιο βάθος και μετά συνεχίζει στην επέκταση καταστάσεων στο αμέσως επόμενο επίπεδο.

Ο αλγόριθμος BFS:

1. **Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.**
2. **Αν το μέτωπο της αναζήτησης είναι κενό τότε σταμάτησε.**
3. **Βγάλε την πρώτη κατάσταση από το μέτωπο της αναζήτησης.**
4. **Αν είναι η κατάσταση ανήκει στο κλειστό σύνολο τότε πήγαινε στο βήμα 2.**
5. **Αν η κατάσταση είναι μία τελική τότε ανέφερε τη λύση.**
6. **Αν θέλεις και άλλες λύσεις πήγαινε στο βήμα 2. Αλλιώς σταμάτησε.**
7. **Εφάρμοσε τους τελεστές μεταφοράς για να βρεις τις καταστάσεις-παιδιά.**
8. **Βάλε τις καταστάσεις-παιδιά στο τέλος του μετώπου της αναζήτησης.**
9. **Βάλε την κατάσταση-γονέα στο κλειστό σύνολο.**
10. **Πήγαινε στο βήμα 2.**

Ο αλγόριθμος BFS (Ψευδοκώδικας)

```
algorithm bfs(InitialState, FinalStates)
begin
Closed←∅;
Frontier←<InitialState>;
CurrentState←First(Frontier);
while CurrentState ∉ FinalStates do
    Frontier←delete(CurrentState,Frontier);
    if CurrentState ∉ ClosedSet
    begin
      ChildrenStates ←Expand(CurrentState);
      Frontier← Frontier ^ ChildrenStates;
      Closed←Closed∪{CurrentState};
    end;
  if Frontier= ∅ then exit;
  CurrentState←First(Frontier);
endwhile;
return success;
end.
```

Αναζήτηση Πρώτα σε Πλάτος

Σχόλια

- Το μέτωπο της αναζήτησης είναι μια δομή ουράς (Queue FIFO, δηλαδή First In First Out).
  - Έτσι, ποτέ δεν επεκτείνεται μία κατάσταση αν δεν επεκταθούν πρώτα όλες οι καταστάσεις που βρίσκονται σε μικρότερο βάθος, γιατί απλά οι τελευταίες μπήκαν στο μέτωπο της αναζήτησης νωρίτερα.

- Πλεονεκτήματα:
  - Βρίσκει πάντα την καλύτερη λύση (μικρότερη σε μήκος).
  - Είναι πλήρης.

- Μειονεκτήματα:
  - Το μέτωπο της αναζήτησης μεγαλώνει πολύ σε μέγεθος.

Αναζήτηση Πρώτα σε Πλάτος (BFS)

Πρόβλημα των ποτηριών

Μέτωπο αναζήτησης	**Κλειστό Σύνολο**	**Κατάσταση**	**Παιδιά**
<Α>	{}	Α	<Β, Γ>
<Β, Γ>	{Α}	Β	<Α, Δ>
<Γ,Α,Δ>	{Α,Β}	Γ	<Ε,Α>
<Α,Δ,Ε,Α>	{Α,Β,Γ}	Α	- (βρόχος)
<Δ,Ε,Α>	{Α,Β,Γ}	Δ	<Β,Ζ,Γ>
<Ε,Α,Β,Ζ,Γ>	{Α,Β,Γ,Δ}	Ε	<Α,Η>
<Α,Β,Ζ,Γ,Α,Η>	{Α,Β,Γ,Δ,Ε}	Α	- (βρόχος)
<Β,Ζ,Γ,Α,Η>	{Α,Β,Γ,Δ,Ε}	Β	- (βρόχος)
<Ζ,Γ,Α,Η>	{Α,Β,Γ,Δ,Ε}	Ζ	<Α,Θ,Δ>
<Γ,Α,Η,Α,Θ,Δ>	{Α,Β,Γ,Δ,Ε,Ζ}	Γ	- (βρόχος)
<Α,Η,Α,Θ,Δ>	{Α,Β,Γ,Δ,Ε,Ζ}	Α	- (βρόχος)
<Η,Α,Θ,Δ>	{Α,Β,Γ,Δ,Ε,Ζ}	Η	<Ε,Γ>
<Α,Θ,Δ,Ε,Γ>	{Α,Β,Γ,Δ,Ε,Ζ,Η}	Α	- (βρόχος)
<Θ,Δ,Ε,Γ>	{Α,Β,Γ,Δ,Ε,Ζ,Η}	Θ	<Ζ,Δ,Ι>
<Δ,Ε,Γ,Ζ,Δ,Ι>	{Α,Β,Γ,Δ,Ε,Ζ,Η}	Δ	- (βρόχος)
<Ε,Γ,Ζ,Δ,Ι>	{Α,Β,Γ,Δ,Ε,Ζ,Η}	Ε	- (βρόχος)

Μέτωπο αναζήτησης	**Κλειστό Σύνολο**	**Κατάσταση**	**Παιδιά**
<Γ,Z,Δ,I>	{A,B,Γ,Δ,E,Z,H}	Γ	- (βρόχος)
<Z,Δ,I>	{A,B,Γ,Δ,E,Z,H}	Z	- (βρόχος)
<Δ,I>	{A,B,Γ,Δ,E,Z,H}	Δ	- (βρόχος)
<I>	{A,B,Γ,Δ,E,Z,H}	I	<K,Γ,B>
<K,Γ,B>	{A,B,Γ,Δ,E,Z,H,I}	K	ΤΕΛΙΚΗ

Αλγόριθμος Επαναληπτικής Εκβάθυνσης

Ο αλγόριθμος επαναληπτικής εκβάθυνσης (Iterative Deepening - *ID*) συνδυάζει με τον καλύτερο τρόπο τους DFS και BFS.

Ο αλγόριθμος ID:

1. Όρισε το αρχικό βάθος αναζήτησης (συνήθως 1).
2. Εφάρμοσε τον αλγόριθμο DFS μέχρι αυτό το βάθος αναζήτησης.
3. Αν έχεις βρει λύση σταμάτησε.
4. Αύξησε το βάθος αναζήτησης (συνήθως κατά 1).
5. Πήγαινε στο βήμα 2.

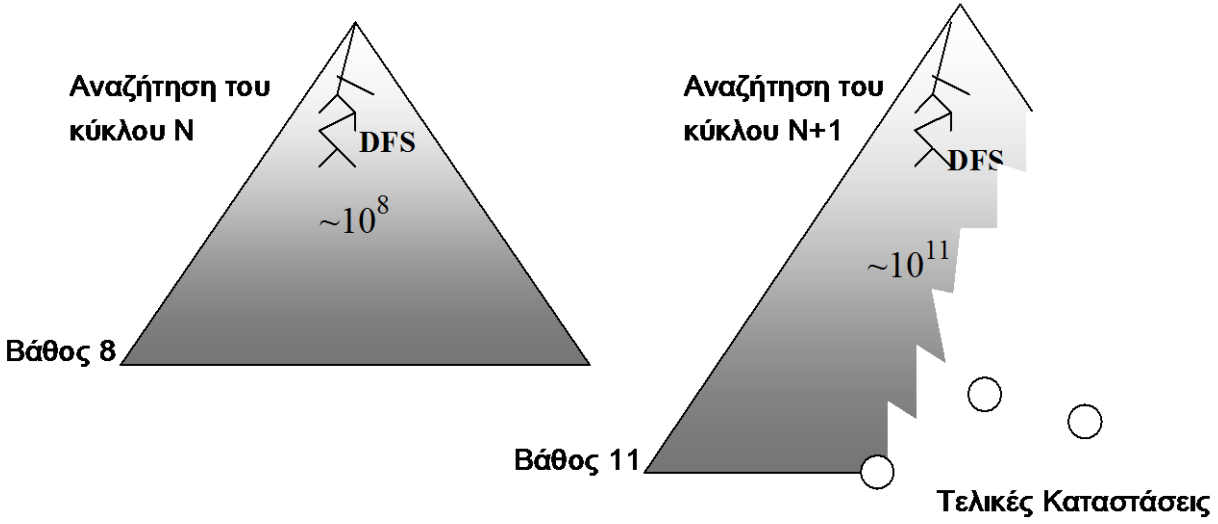
Ο αλγόριθμος ID (Ψευδοκώδικας)

```
algorithm id(InitialState, FinalStates)
begin
  depth←1
  while solution is not found do
    bounded_dfs(InitialState,FinalStates,depth);
    depth←depth+1
  endwhile;
end.
```

Αναζήτηση ID

Σχόλια

- Μειονεκτήματα:
  - Όταν αρχίζει ο DFS με διαφορετικό βάθος δε θυμάται τίποτα από την προηγούμενη αναζήτηση.
- Πλεονεκτήματα:
  - Είναι πλήρης.
  - Αν το βάθος αυξάνεται κατά 1 σε κάθε κύκλο και ο ID βρει λύση, τότε αυτή η λύση θα είναι η καλύτερη, γιατί αν υπήρχε άλλη, καλύτερη λύση, αυτή θα βρισκόταν σε προηγούμενο κύκλο αναζήτησης.



Πλεονεκτήματα (συνέχεια)

- Έχει αποδειχθεί ότι ο ID έχει την ίδια πολυπλοκότητα σε χώρο και χρόνο με τους DFS και BFS, όταν έχουμε μεγάλους χώρους αναζήτησης, παρ' όλο που επαναλαμβάνει άσκοπα το κτίσιμο του χώρου αναζήτησης,
- Για παράδειγμα:
  - Έστω ότι το δένδρο αναζήτησης έχει σταθερό παράγοντα διακλάδωσης 10. Εφαρμόζουμε τον ID σε βάθος 5.
  - Οι καταστάσεις που θα επεκταθούν είναι:
- $10^0+10^1+10^2+10^3+10^4+10^5 = 111.111$
  - Αν αυξηθεί το βάθος κατά 2 (συνολικό βάθος 7), οι καταστάσεις που θα επεκταθούν συνολικά είναι:
- $10^7+10^6+10^5+ \ldots + 10^0 = 11.111.111$
  - άρα το χάσιμο ήταν 11.111/11.111.111 = 1%

- Ο ID δεν κινδυνεύει να χαθεί σε κάποιο κλαδί απείρου μήκους. Αν βρει λύση θα είναι η καλύτερη, αν το βάθος αυξάνεται κατά 1 σε κάθε κύκλο.

Αναζήτηση Διπλής Κατεύθυνσης (1/2)

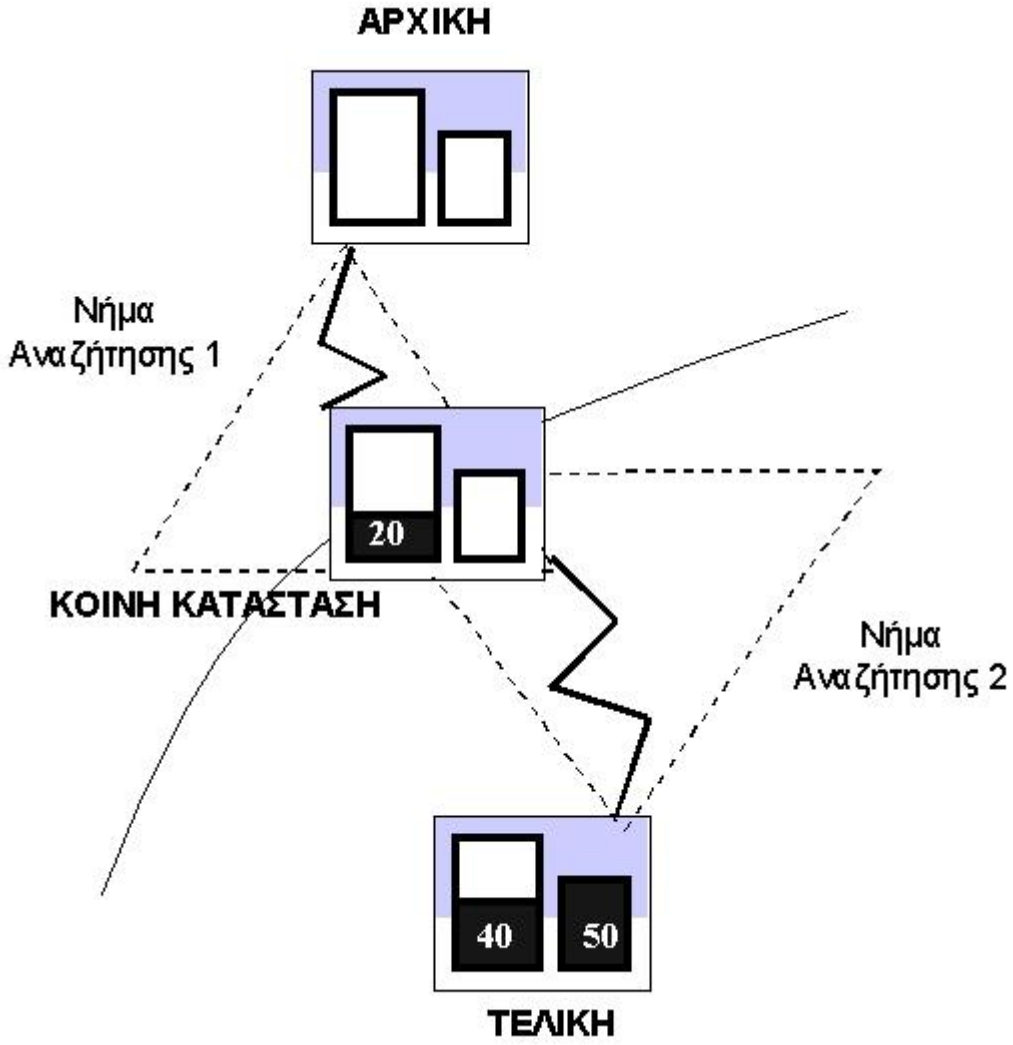
Η ιδέα της *αναζήτησης διπλής κατεύθυνσης* (Bidirectional Search - *BiS*) πηγάζει από τη δυνατότητα του παραλληλισμού (parallelism) στα υπολογιστικά συστήματα.

- Προϋποθέσεις κάτω από τις οποίες μπορεί να εφαρμοστεί:
  - Οι τελεστές μετάβασης είναι αντιστρέψιμοι (reversible), και
  - Είναι πλήρως γνωστή η τελική κατάσταση.

- Τότε
  - Μπορούμε να αρχίσουμε 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 εως την αρχική και εως την τελική κατάσταση.

- Μειονεκτήματα:
  - Υπάρχει επιπλέον κόστος που οφείλεται στην επικοινωνία μεταξύ των δύο αναζητήσεων.

Αναζήτηση Διπλής Κατεύθυνσης (2/2)



Επέκταση και Οριοθέτηση (B&B)

Ο *αλγόριθμος επέκτασης και οριοθέτησης* (Branch and Bound - *B&B*) εφαρμόζεται σε προβλήματα όπου αναζητείται η βέλτιστη λύση, δηλαδή εκείνη με το ελάχιστο κόστος.

- Η λειτουργία του B&B βασίζεται στο κλάδεμα καταστάσεων (pruning) και κατά συνέπεια στην ελάττωση του χώρου αναζήτησης.
- Αν για παράδειγμα σε ένα πρόβλημα βρούμε μια λύση με κόστος (π.χ. απόσταση) 159 και κατά την αναζήτηση για άλλες λύσεις συναντήσουμε μια κατάσταση μέχρι την οποία η διαδρομή είναι ήδη 167, δεν υπάρχει λόγος επέκτασης της γιατί θα οδηγηθούμε σε χειρότερη λύση. Άρα κλαδεύουμε αυτήν την κατάσταση καθώς και το υπόλοιπο υποδένδρο.

Ο αλγόριθμος B&B:

1. Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αρχική τιμή της καλύτερης λύσης είναι το +∞ (όριο).
3. Αν το μέτωπο της αναζήτησης είναι κενό, τότε σταμάτησε.
   Η καλύτερη μέχρι τώρα λύση είναι και η βέλτιστη.
4. Βγάλε την πρώτη σε σειρά κατάσταση από το μέτωπο της αναζήτησης.
5. Αν η κατάσταση ανήκει στο κλειστό σύνολο, τότε πήγαινε στο 3.
6. Αν η κατάσταση είναι τελική, τότε ανανέωσε τη λύση ως την καλύτερη μέχρι τώρα και ανανέωσε την τιμή του ορίου με την τιμή που αντιστοιχεί στην τελική κατάσταση. Πήγαινε στο 3.
7. Εφάρμοσε τους τελεστές μεταφοράς για να παράγεις τις καταστάσεις-παιδιά και την τιμή που αντιστοιχεί σε αυτές.
8. Βάλε τις καταστάσεις-παιδιά, των οποίων η τιμή δεν υπερβαίνει το όριο, μπροστά στο μέτωπο της αναζήτησης. **(*)**
9. Βάλε την κατάσταση-γονέα στο κλειστό σύνολο.
10. Πήγαινε στο 3.

(*) Αυτός είναι DFS-B&B γιατί οι νέες καταστάσεις μπαίνουν μπροστά στο μέτωπο αναζήτησης. Υπάρχει και BestFS-B&B.

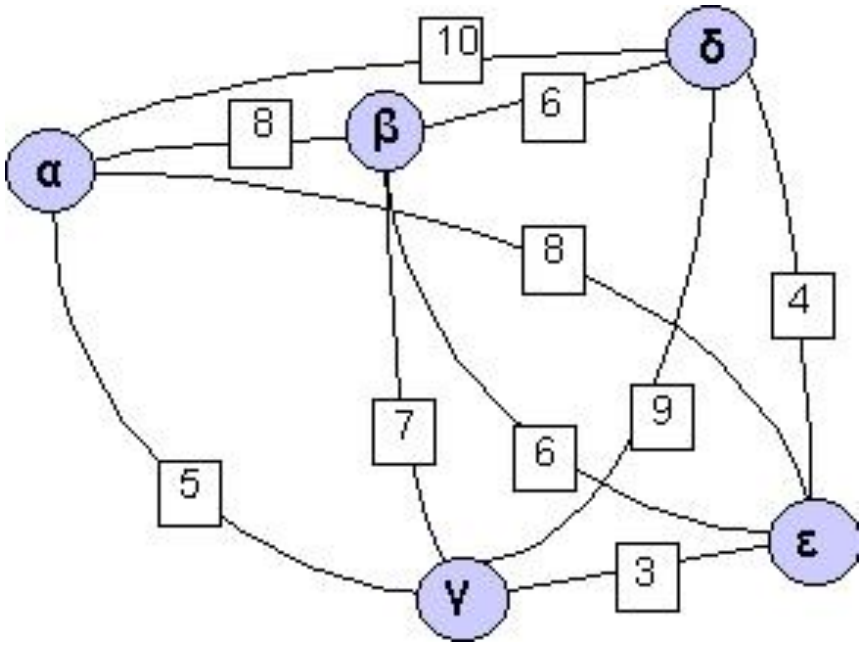
Ο αλγόριθμος B&B (Ψευδοκώδικας)

```
algorithm b&b(InitialState, FinalStates)
begin
   Closed←∅;
   Frontier←<InitialState>;
   BestCost←∝;
   BestState←null;
   while Frontier≠ ∅ do
      CurrentState←First(Frontier);
      CurrentCost←Cost(Current_State);
      Frontier←delete(CurrentState,Frontier);
      if CurrentCost < BestCost then
   if CurrentState∈ FinalStates then
            BestState←CurrentState;
            BestCost←CurrentCost;
         else
            Next←Expand(CurrentState);
            ChildrenStates←{s| s∈Next ∧ s∉Frontier ∧ s∉Closed};
            Frontier←ChildrenStates ^ Frontier;
            Closed←Closed∪{CurrentState};
         endif;
      endif;
   endwhile;
    if BestState = null
        then return fail
        else return BestState and BestCost;
end.
```

Το Πρόβλημα του Πλανόδιου Πωλητή (TSP)

- Ένα γράφημα με Ν κόμβους αντιπροσωπεύει Ν τοποθεσίες. Κάποιος πρέπει να επισκεφτεί μία φορά τον κάθε κόμβο και αν είναι δυνατόν με το λιγότερο κόστος, δηλαδή διανύοντας την ελάχιστη δυνατή απόσταση.
- Για απλότητα όλοι οι κόμβοι ενώνονται μεταξύ τους (πλήρης γράφος) και η αρχή είναι δεδομένη. Το σχήμα δείχνει και το χώρο αναζήτησης του προβλήματος (με αρχή την τοποθεσία: *α*).
  - Αν ο γράφος αυτός αναπτυσσόταν σε δένδρο, το δένδρο θα είχε 4 επιλογές από την αρχική κατάσταση, 3 επιλογές από κάθε κατάσταση που προκύπτει από την αρχική, κ.ο.κ., δηλαδή 4! διαφορετικές λύσεις με το αντίστοιχο κόστος, από τις οποίες μία θα είναι η βέλτιστη.
- Το πρόβλημα δεν είναι τόσο απλό όσο δείχνει και ανήκει στην κατηγορία προβλημάτων με λύση *μη-πολυωνυμικού χρόνου* **(***NP-complete***).**
  - Αν N=20, τότε υπάρχουν $19!=1.216\cdot10^{17}$ λύσεις, που σημαίνει ότι ακόμη και ένας υπολογιστής που εξετάζει 1 εκατομμύριο λύσεις το δευτερόλεπτο θα χρειαζόταν 385 χρόνια για να βρει τη βέλτιστη λύση.
- Το πρόβλημα είναι πρόβλημα ελαχιστοποίησης κόστους και έχει πολλές εφαρμογές.
  - Για παράδειγμα, η αυτόματη συναρμολόγηση ψηφιακών πλακετών, όπου ένα μηχανικό χέρι μεταφέρει τα ολοκληρωμένα κυκλώματα και τα τοποθετεί στη σωστή τους θέση, είναι πρόβλημα TSP. Μία μη βέλτιστη διαδρομή του χεριού μπορεί να έχει τεράστιο χρονικό και κατά συνέπεια οικονομικό κόστος σε μία εταιρία συναρμολόγησης.

Ο αλγόριθμος B&B: Το πρόβλημα TSP



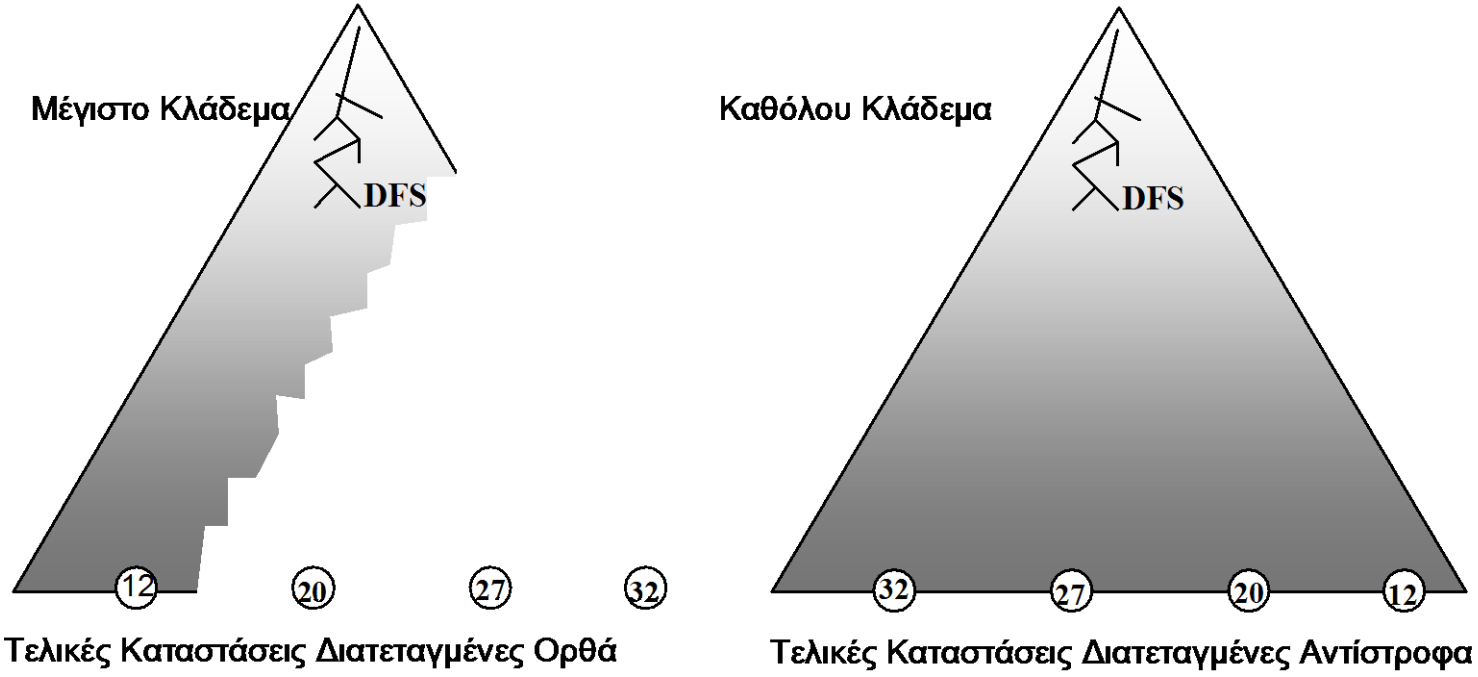
Μέτωπο της αναζήτησης	Κόστος Λύσης	Κατάσταση	Παιδιά
<α>	+∞	α	$αβ^{8}$,$αγ^{5}$,$αδ^{10}$,$αε^{8}$
<$αβ^{8}$,$αγ^{5}$,$αδ^{10}$,$αε^{8}$>	+∞	αβ	$αβγ^{15}$,$αβδ^{14}$, $αβε^{14}$
<$αβγ^{15}$,$αβδ^{14}$,$αβε^{14}$,$αγ^{5}$,...>	+∞	αβγ	$αβγδ^{24}$, $αβγε^{18}$
<$αβγδ^{24}$,$αβγε^{18}$, $αβδ^{14}$,$αβε^{14}$...>	+∞	αβγδ	$αβγδε^{28}$
<$αβγδε^{28}$,$αβγε^{18}$, $αβδ^{14}$,...>	+∞	αβγδε	$αβγδεα^{36}$
< $αβγδεα^{36}$, $αβγε^{18}$, $αβδ^{14}$,..>	36	αβγδεα	Τελική Κατάσταση

Μέτωπο της αναζήτησης	Κόστος Λύσης	Κατάσταση	Παιδιά
<αβγε^{18}, αβδ^{14},... >	36	αβγε	αβγεδ^{22}
<αβγεδ^{22},αβδ^{14},...>	36	αβγεδ	αβγεδα^{32}
< αβγεδα^{32} ,αβδ^{14},αβε^{14}...>	32	αβγεδα^{32}	Τελική Κατάσταση
...	...	...	...
<αβδεγα^{26}, ...>	26	αβδεγα	Τελική Κατάσταση
...	...	...	...
<αβεγδ^{26}, ...>	26	αβεγδ	Κλάδεμα
....	...	...	...
<αεβγδ^{30}, ...>	26	αεβγδ	Κλάδεμα
. . .	. . .	. . .	. . .
<>	ΕλάχιστηΤιμή	ΤΕΛΟΣ	

- Ο B&B εφαρμόζεται όταν υπάρχει μια πραγματική εκτίμηση του κόστους όπως στο TSP.
- Το κέρδος από το κλάδεμα καταστάσεων εξαρτάται από το πόσο γρήγορα θα βρεθεί μια καλή λύση στο πρόβλημα γιατί θα θέσει γρήγορα ένα χαμηλό όριο.
- Υπάρχει περίπτωση να μη γίνει καθόλου κλάδεμα αν οι λύσεις είναι διατεταγμένες από τη χειρότερη προς την καλύτερη.
- Στη χειρότερη περίπτωση συμπεριφέρεται σαν τον DFS.

Ο αλγόριθμος B&B

Σχόλια



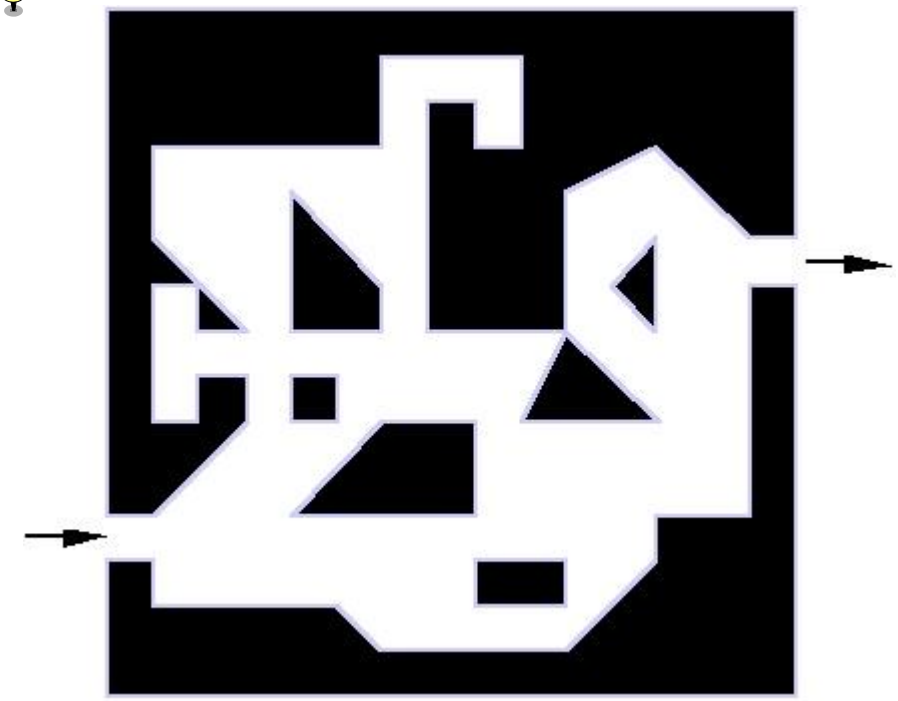
- Υπάρχουν διάφορες παραλλαγές του B&B, ανάλογα με το ποια κατάσταση επεκτείνεται πρώτη.
  - Η περιγραφή που δόθηκε αφορά έναν αλγόριθμο που είναι DFS B&B, γιατί οι νέες καταστάσεις μπαίνουν στην αρχή του μετώπου αναζήτησης και συνεπώς αυτή που βρίσκεται σε μεγαλύτερο βάθος επεκτείνεται πρώτη, όπως και στον DFS.

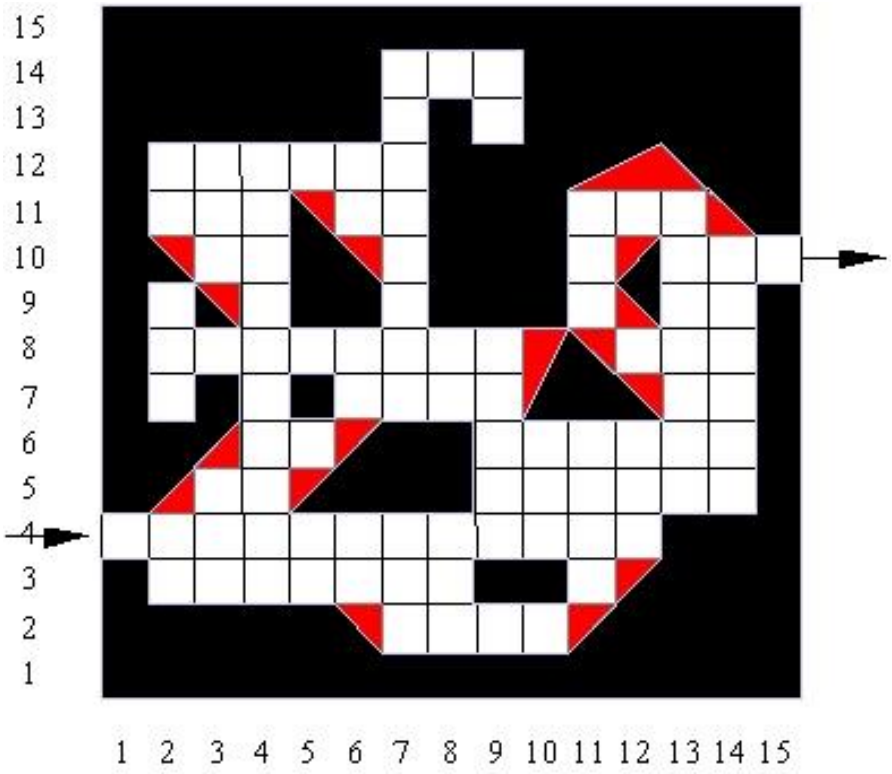
Εφαρμογή των Αλγορίθμων Τυφλής Αναζήτησης

Το πρόβλημα του Λαβύρινθου- Ορισμός.

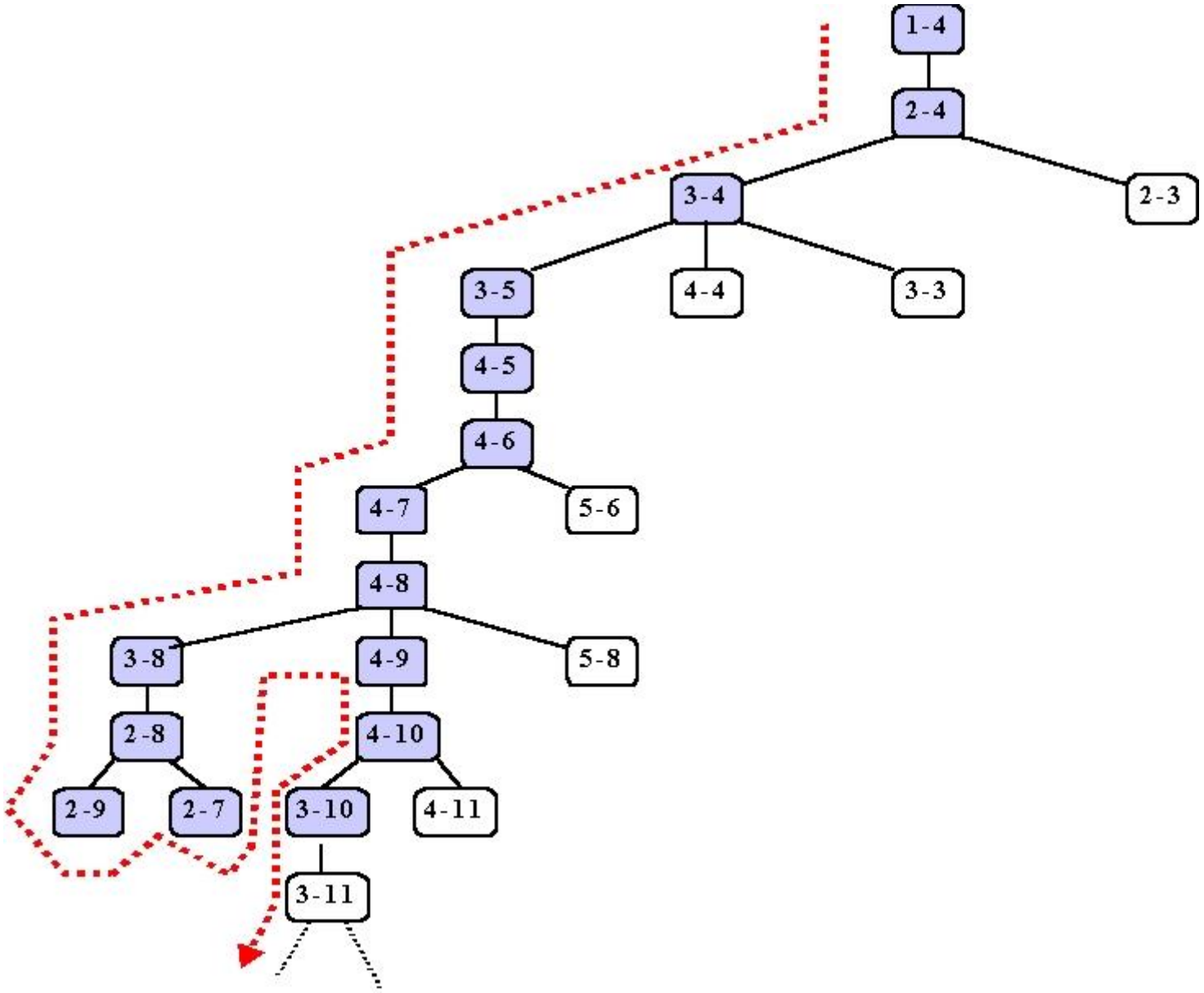
- Για να μπορέσουν να εφαρμοστούν οι αλγόριθμοι αναζήτησης, πρέπει να οριστεί το πρόβλημα και μάλιστα ο χώρος καταστάσεων.
- Αυτό γίνεται με τη βοήθεια ενός πλέγματος.
- Το πλέγμα, δίνει τη δυνατότητα να υπάρχουν συγκεκριμένες θέσεις με αντίστοιχες συντεταγμένες. Εφόσον απομακρυνθούν τα τετράγωνα του πλέγματος που εμπεριέχουν μέρος εμποδίων, το πρόβλημα εύρεσης διαδρομής γίνεται ως εξής:
- Αρχική κατάσταση είναι η θέση με συντεταγμένες (1,4).
- Το σύνολο τελικών καταστάσεων περιέχει μόνο τη θέση (15,10).
- Οι τελεστές μεταφοράς είναι οι εξής:
  - πήγαινε μία θέση αριστερά,
  - πήγαινε μία θέση επάνω,
  - πήγαινε μία θέση δεξιά,
  - πήγαινε μία θέση κάτω, εφόσον η θέση είναι ελεύθερη.

- Ο χώρος καταστάσεων είναι όλες οι ελεύθερες θέσεις, χωρίς εμπόδια, του πλέγματος.

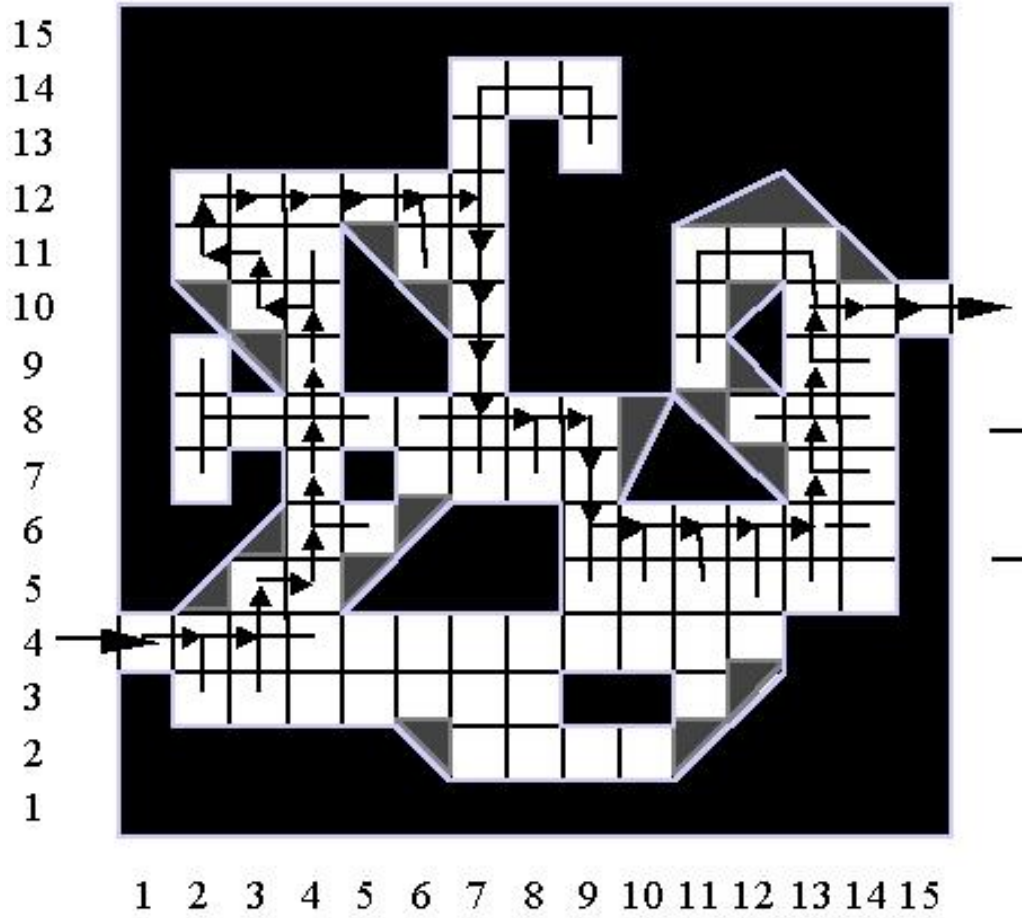



15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Εφαρμογή του αλγορίθμου DFS



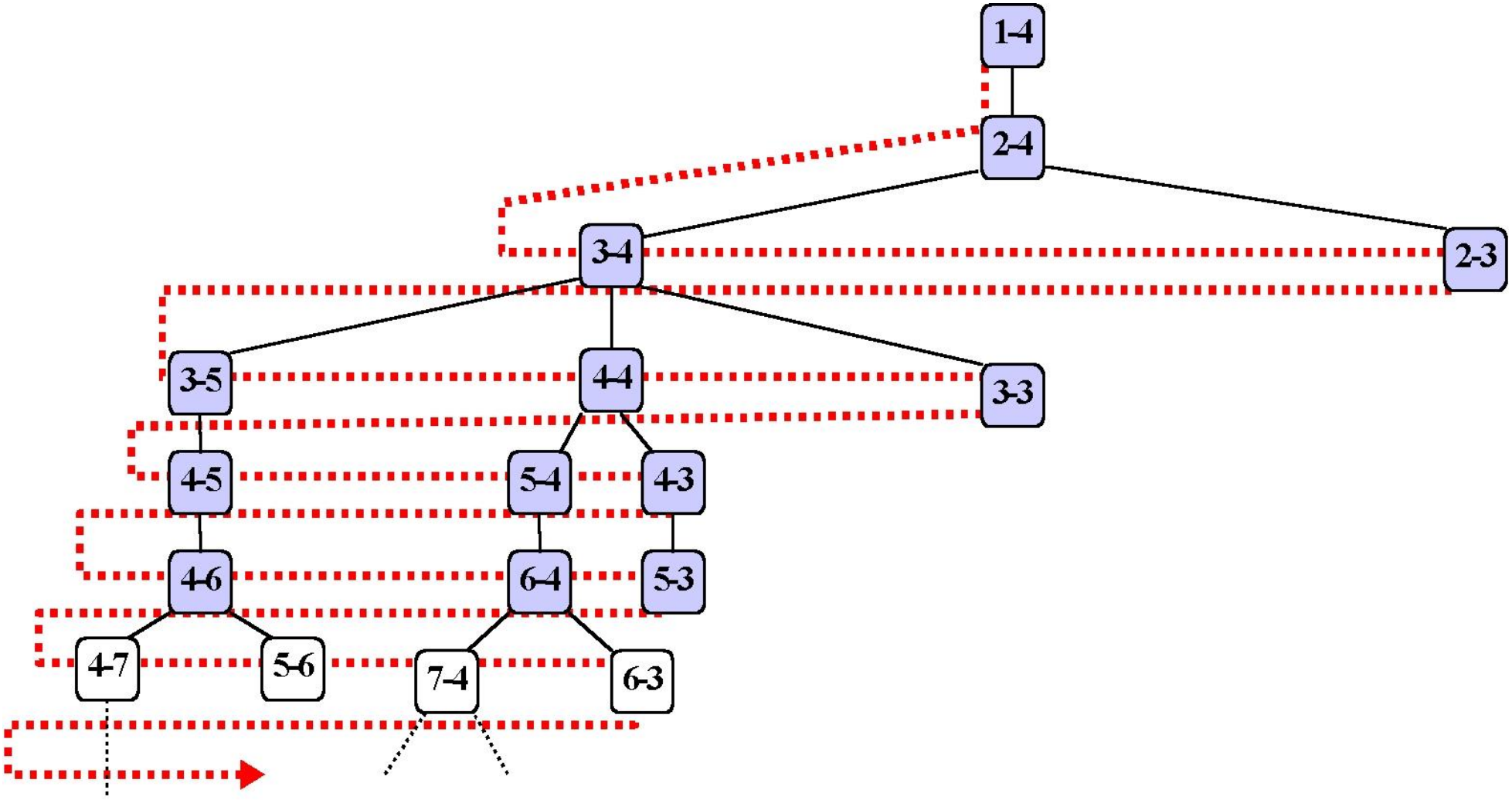
Λύση στο πρόβλημα του λαβύρινθου με χρήση DFS



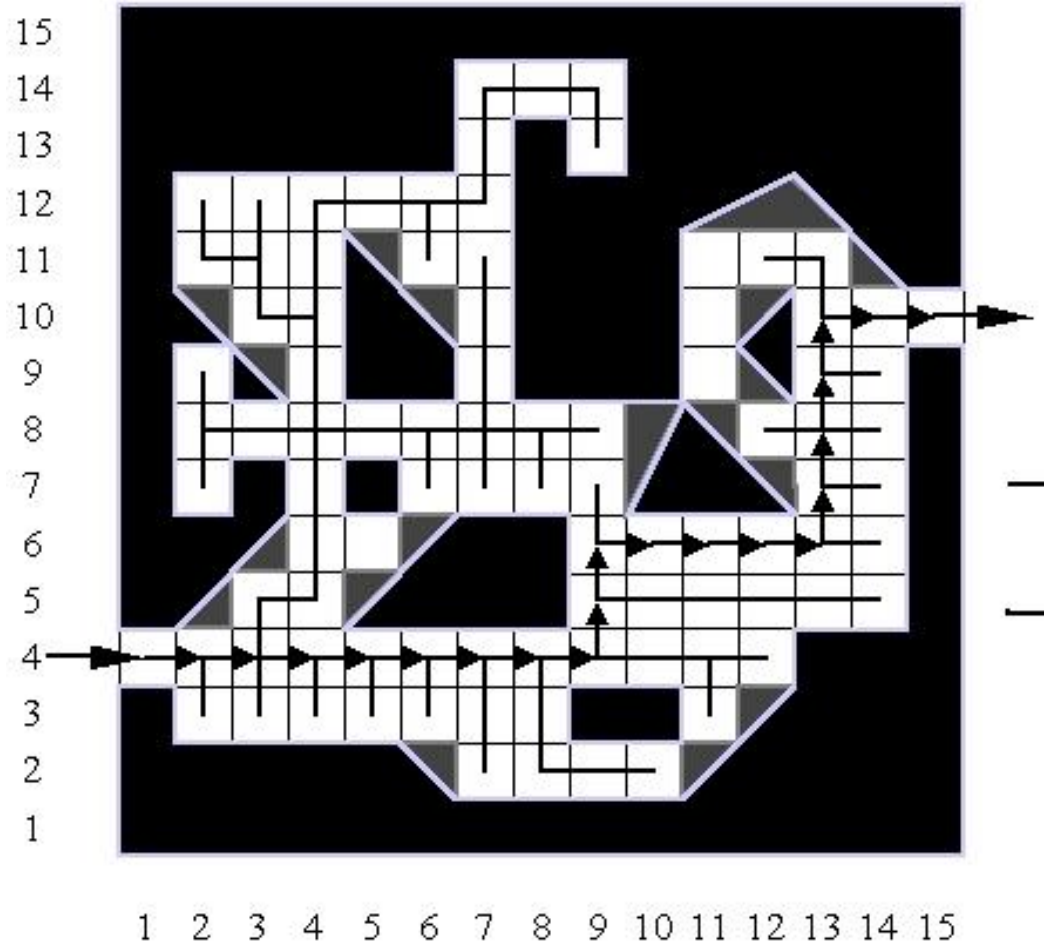
→ Τελεστής που ανήκει στη λύση

— Τελεστής που δεν ανήκει στη λύση αλλά εξετάζεται κατά την εκτέλεση του αλγορίθμου

Εφαρμογή αλγορίθμου BFS



Λύση στο πρόβλημα του λαβύρινθου με χρήση BFS

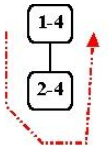


→ Τελεστής που ανήκει στη λύση

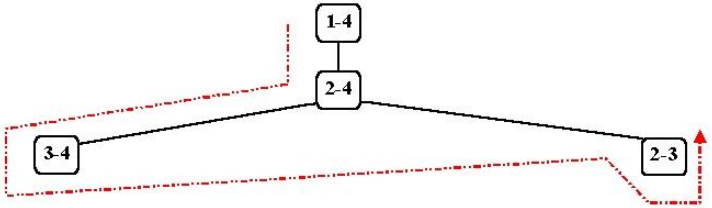
— Τελεστής που δεν ανήκει στη λύση αλλά εξετάζεται κατά την εκτέλεση του αλγορίθμου

Εφαρμογή του ID στο πρόβλημα του λαβυρίνθου

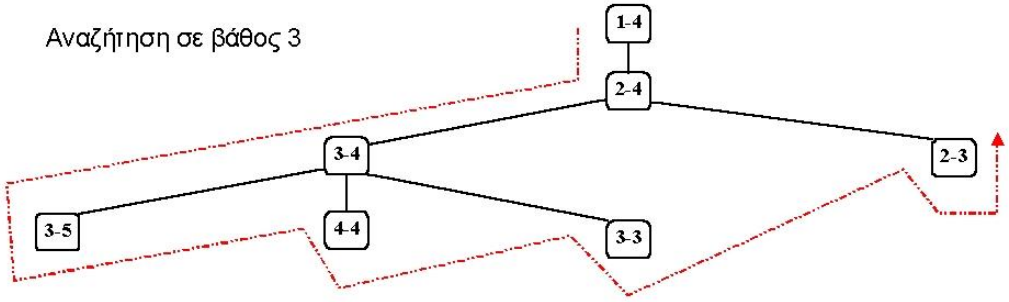
Αναζήτηση σε βάθος 1



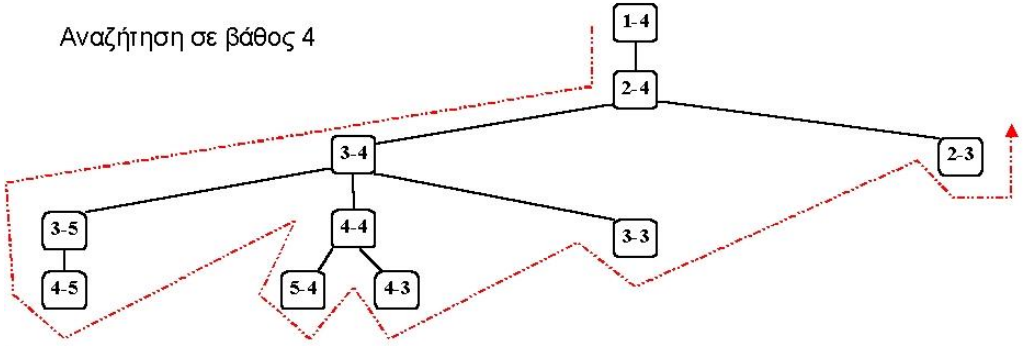
Αναζήτηση σε βάθος 2



Αναζήτηση σε βάθος 3



Αναζήτηση σε βάθος 4



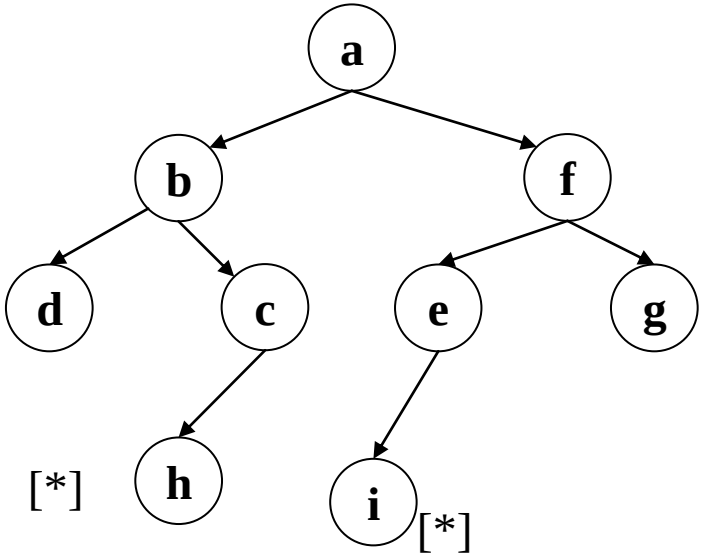
Άσκηση 3.1

- Στο διπλανό σχήμα παρουσιάζεται ένα δένδρο αναζήτησης, Οι κόμβοι που είναι σημειωμένοι με ένα * , είναι οι τερματικοί κόμβοι της αναζήτησης.
- Γράψτε την σειρά με την οποία θα εξεταστούν οι κόμβοι του δέντρου (π.χ. a,b,c,…) μέχρι να βρεθεί τερματική κατάσταση από τους ακόλουθους αλγορίθμους:

  DFS:……

  BFS:……

  ID (με βήμα 2):……



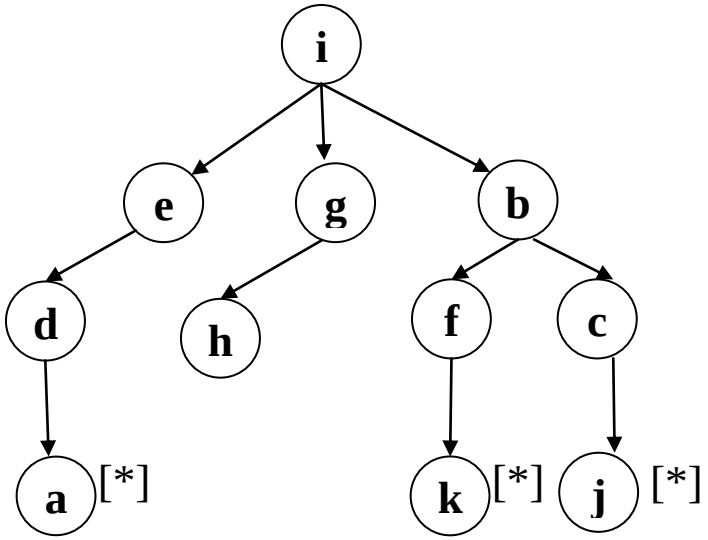
Άσκηση 3.2

- Στο διπλανό σχήμα παρουσιάζεται ένα δένδρο αναζήτησης, Οι κόμβοι που είναι σημειωμένοι με ένα * , είναι οι τερματικοί κόμβοι της αναζήτησης.
- Γράψτε την σειρά με την οποία θα εξεταστούν οι κόμβοι του δέντρου (π.χ. a,b,c,…) μέχρι να βρεθεί τερματική κατάσταση από τους ακόλουθους αλγορίθμους:

  DFS:……

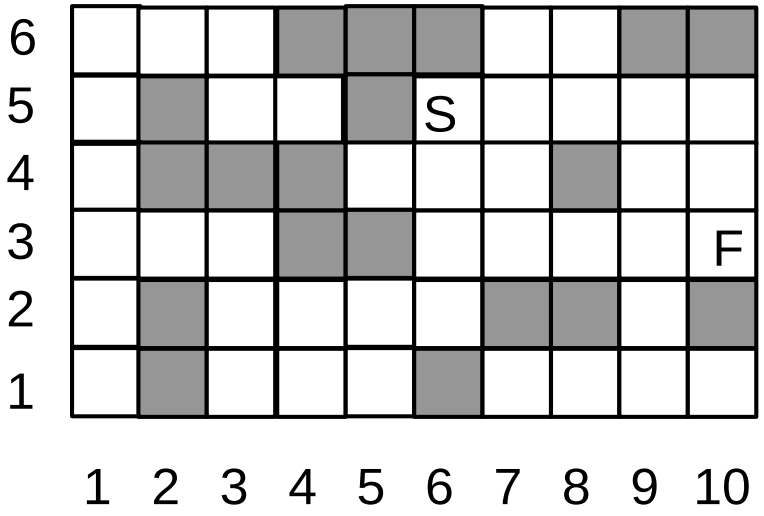
  BFS:……

  ID (με βήμα 2):……...



Άσκηση 3.3

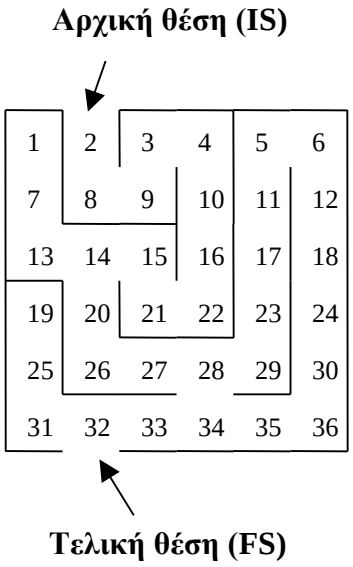
- Εφαρμόστε (μόνο 5 βήματα) τον αλγόριθμο αναζήτησης κατά βάθος (DFS) στο διπλανό πρόβλημα του λαβυρίνθου ξεκινώντας από την αρχική θέση S (τελική θέση είναι η F). Σημείωση: Δεν επιτρέπονται διαγώνιες κινήσεις.



Υλοποίηση Αλγορίθμων Τυφλής Αναζήτησης σε Prolog

Το πρόβλημα του λαβυρίνθου

Αναπαράσταση:




```
initial_state(2).

goal(32).

operator(State,Child)          :-
connect(Child,State).

operator(State,Child)          :-
connect(State,Child).
```

```
connect(1,7).  connect(2,8).  connect(3,4).
connect(3,9).  connect(4,10). connect(5,11).
connect(5,6).  connect(7,13). connect(8,9).
connect(10,16).  connect(11,17).  connect(12,18).
connect(13,14).  connect(14,15).  connect(14,20).
connect(15,21).  connect(16,22).  connect(17,23).
connect(18,24).  connect(19,25).  connect(20,26).
connect(21,22).  connect(23,29).  connect(24,30).
connect(25,31).  connect(26,27).  connect(27,28).
connect(28,29).  connect(28,34).  connect(30,36).
connect(31,32).  connect(32,33).  connect(33,34).
connect(34,35).  connect(35,36).  connect(6,12).
```

Αναζήτηση πρώτα σε βάθος (Depth First Search - DFS)

```
dfs(State, Solution, Solution):-goal(State).


dfs(State, PathSoFar, Solution):-operator(State,Child),
     not(member(Child,                                PathSoFar)),
     dfs(Child, [Child|PathSoFar], Solution).


godfs(Solution):-  initial_state(IS),dfs(IS, [IS], Sol1),
     reverse(Sol1,Solution).
```

Αναζήτηση πρώτα σε πλάτος (Breadth First Search - BFS)

```
bfs([[State|Path]|_], [State|Path]):-goal(State).

bfs([[State|Path]|RestFrontSet], Solution):-
  expand(State,Path,ChildrenStates),
  append(RestFrontSet,ChildrenStates,NewFrontSet),
  bfs(NewFrontierSet, Solution).

gobfs(Solution):-  initial_state(IS),bfs([[IS]],Sol1),
  reverse(Sol1,Solution).
```

- Το πρώτο όρισμα της `bfs/2` αποτελείται από λίστες-λιστών.
  - Κάθε "μικρή" λίστα αποτελείται από ημιτελή μονοπάτια από την αρχική έως κάποια κατάσταση.
  - Η "μεγάλη" λίστα είναι το *μέτωπο της αναζήτησης* (*frontier set*).
- Η `expand/3`
  - βρίσκει όλες τις καταστάσεις παιδιά `Child` από την τρέχουσα κατάσταση `State`, και
  - τις προσθέτει στην αρχή μονοπατιού `[State|Path]` που έχει προκύψει ως την τρέχουσα κατάσταση.

```
expand(State,Path,Children):-
 findall([Child,State|Path],operator(State,Child),Children).
```

Αναζήτηση πρώτα σε πλάτος (BFS) με κλειστό σύνολο (closed set).

- Η διαφορά με τον απλό BFS είναι ότι εκτός από το σύνορο οριοθέτησης, υπάρχει και μία άλλη λίστα (το **κλειστό σύνολο**), στο οποίο αποθηκεύονται όλες οι καταστάσεις που έχει επισκεφθεί μέχρι τώρα ο αλγόριθμος.
- Τα παιδιά της τρέχουσας κατάστασης ελέγχονται για το αν οδηγούν σε καταστάσεις που έχει επισκεφθεί ο αλγόριθμος.
    - Αν ναι, τότε αυτές "κόβονται" (prune) από τη συνέχεια της αναζήτησης.
- Σε κάθε βήμα η τρέχουσα κατάσταση μπαίνει στο κλειστό σύνολο.
- Αρχικά το κλειστό σύνολο είναι κενό.

```
bfs_cl([[State|Path]|_],_,[State|Path]):-goal(State).


bfs_cl([[State|Path]|RestFSet],ClosedSet,Solution):-
  expand(State,Path,ChildStates),
  prune(ChildStates,ClosedSet,P_ChildStates),
  append(RestFSet,P_ChildStates,NewFSet),
  bfs_cl(NewFSet,[State|ClosedSet],Solution).

prune([],_,[]) :- !.


prune([[State|Path]|RestChilds],ClosedSet,
                                  [[State|Path]|RestPChilds]) :-
  not(member(State,ClosedSet)), !,
  prune(RestChilds,ClosedSet,RestPChilds).


prune([_|RestChilds],ClosedSet,RestPChilds) :-
  prune(RestChilds,ClosedSet,RestPChilds).

gobfs_cl(Solution):initial_state(IS),bfs_cl([[IS]],[],Sol1),
  reverse(Solution1,Solution).
```

Το πρόβλημα των ιεραποστόλων και κανιβάλων (missionaries and cannibals)

Αρχικά είναι όλοι στην αριστερή όχθη

```
initial_state(state(left(3,3),right(0,0), boat_left)).
```

Σκοπός είναι να μεταφερθούν όλοι με ασφάλεια στη δεξιά όχθη

```
goal(state(left(0,0),right(3,3), boat_right)).
```

Επόμενη κίνηση όταν η βάρκα βρίσκεται στην αριστερή όχθη

```
operator(state(left(ML,CL),right(MR,CR),boat_left),
                state(left(NML,NCL),right(NMR,NCR),boat_right)) :-
    move(M, C),                                  Διάλεξε κίνηση
    M =< ML,    C =< CL,                 Υπάρχουν αρκετοί άνθρωποι?
    NML is ML-M,    NCL is CL-C,          Υπολόγισε το νέο πληθυσμό
    NMR is MR+M,    NCR is CR+C,                και στις δύο όχθες
    valid(NML, NCL),                  Είναι ασφαλείς οι ιεραπόστολοι
    valid(NMR, NCR).                             και στις δύο όχθες?
```

Επόμενη κίνηση όταν η βάρκα βρίσκεται στη δεξιά όχθη

```
operator(state(left(ML,CL),right(MR,CR),boat_right),
                state(left(NML,NCL),right(NMR,NCR),boat_left)) :-
  move(M, C),
  M =< MR,    C =< CR,
  NML is ML+M,    NCL is CL+C,
  NMR is MR-M,    NCR is CR-C,
  valid(NML, NCL), valid(NMR, NCR).
```

```
valid(M, C) :- M >= C, !.  Οι ιεραπόστολοι πρέπει να είναι περισσότεροι
valid(0, _).                        ή να μην υπάρχουν καθόλου σε μία όχθη
```

Δυνατές μετακινήσεις πληθυσμών με τη βάρκα

```
move(2,0).   move(1,0).   move(1,1).
move(0,1).   move(0,2).
```