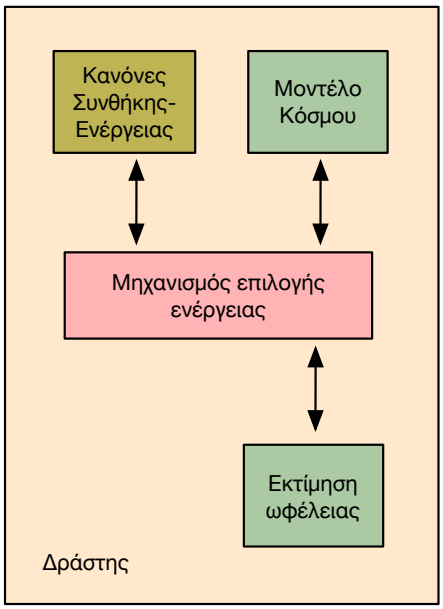
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνητή Νοημοσύνη

ΕΥΡΕΤΙΚΗ ΑΝΑΖΗΤΗΣΗ ΚΑΙ ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ

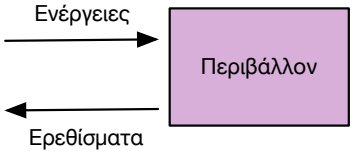
Γιώργος Στάμου

Δράστης ωφέλειας



Παρατήρηση

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



Μεθοδολογία

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

Τυπική διατύπωση προβλήματος

Κόσμος προβλήματος

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

Κατάσταση του κόσμου

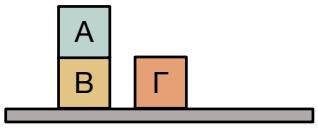
- Είναι μία *επαρκής*, *τυπική* αναπαράσταση του κόσμου σε μία δεδομένη χρονική στιγμή

Πρόβλημα και επίλυση

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

Τυπική διατύπωση προβλήματος

Κόσμος του προβλήματος: Τρεις κύβοι και ένα τραπέζι



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

Κατάσταση:

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

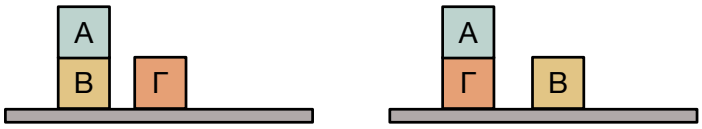
Τυπική διατύπωση προβλήματος

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

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

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

Αρχική κατάσταση και στόχος στο πρόβλημα με τους κύβους



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

Χώρος αναζήτησης λύσης

Δοθέντος ενός προβλήματος (*I,G,T,S*)

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

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

Δέντρα και χαρακτηριστικά προβλήματος

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

Αλγόριθμοι επίλυσης προβλημάτων

Αλγόριθμος επίλυσης είναι μία αυστηρά καθορισμένη ακολουθία εύρεσης ενός κόμβου στόχου

- Ονομάζεται **εξαντλητικός** (exhaustive) όταν το σύνολο των καταστάσεων που εξετάζει ο αλγόριθμος για να βρει τις απαιτούμενες λύσεις είναι ίσο με το χώρο αναζήτησης

- Ένας αλγόριθμος δεν λύνει *πάντα* το πρόβλημα, έστω και αν υπάρχει λύση. Ονομάζεται **πλήρης** (complete) αν εγγυάται ότι θα βρει μία λύση για οποιοδήποτε στόχο, αν τέτοια λύση υπάρχει. Σε αντίθετη περίπτωση, ο αλγόριθμος ονομάζεται **μη-πλήρης** (incomplete).

- Μία λύση ονομάζεται **βέλτιστη** (optimal) αν οδηγεί στην καλύτερη, σύμφωνα με τη διάταξη, τελική κατάσταση. Όταν δεν υπάρχει διάταξη, μία λύση ονομάζεται βέλτιστη αν είναι η **συντομότερη** (shortest).

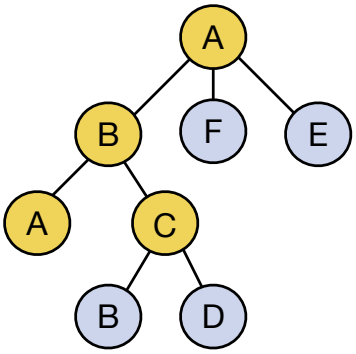
Αλγόριθμοι αναζήτησης λύσης

Βασικές δομές δεδομένων

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

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

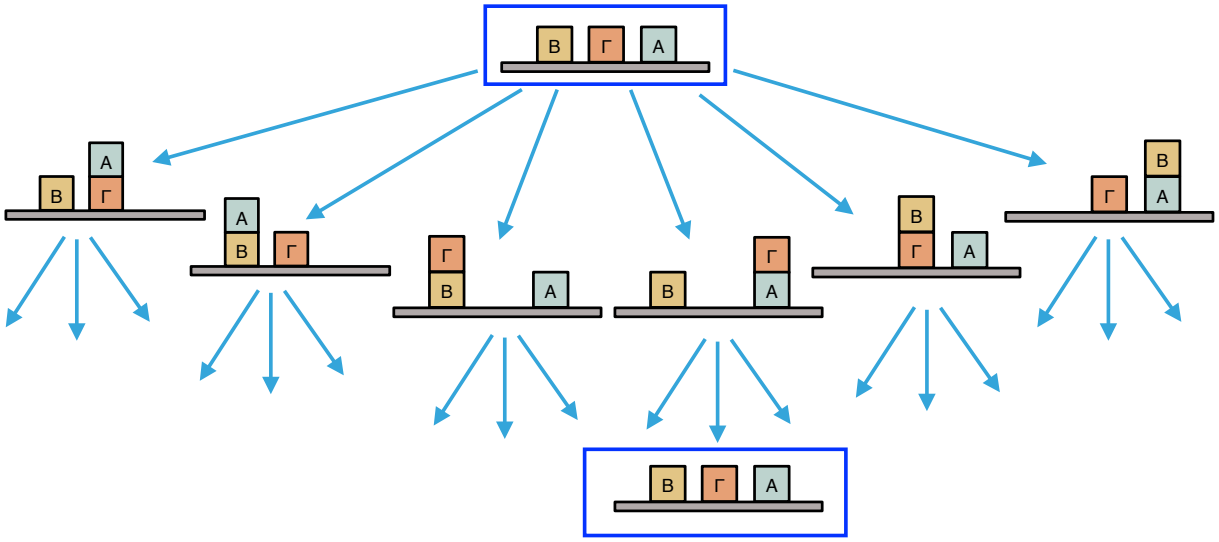
Με έναν απλό έλεγχο, αν η κατάσταση προς επέκταση ανήκει ήδη στο κλειστό σύνολο, αποφεύγονται οι βρόχοι (loops)



Αλγόριθμος αναζήτησης λύσης

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

Αλγόριθμος αναζήτησης λύσης



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

Αλγόριθμος αναζήτησης λύσης

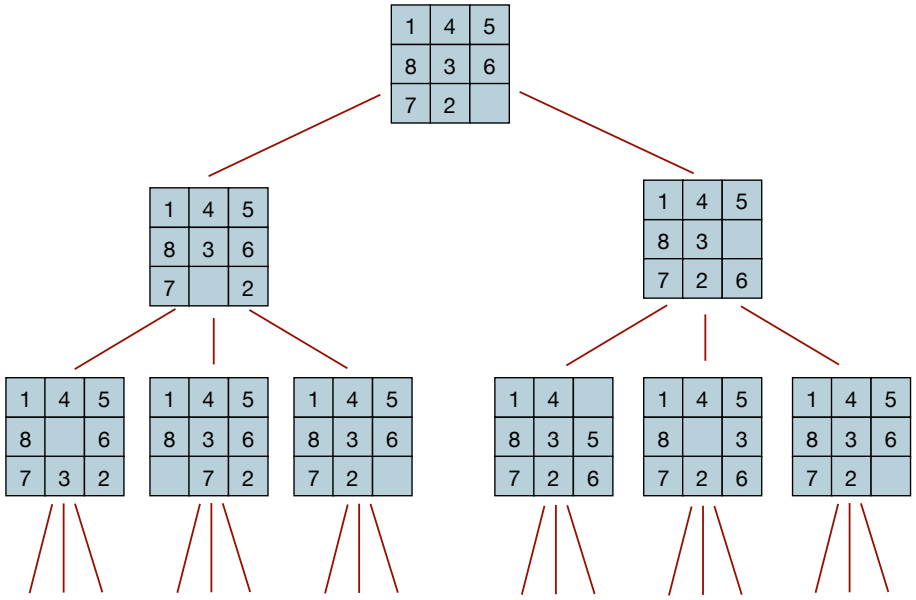
n-puzzle (Sam Loyd - 1878)

Αρχική κατάσταση

1	4	5
8	3	6
7	2	

Στόχος

1	2	3
4	5	6
7	8	



Αλγόριθμος αναζήτησης λύσης

Πλήθος καταστάσεων

- 8-puzzle: 9! = 362.880 καταστάσεις
- 15-puzzle: 16! ~ 2.09×10^{13} καταστάσεις
- 24-puzzle: 25! ~ 10^{25} καταστάσεις

Χρόνος υπολογισμού

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

Αλγόριθμος αναζήτησης λύσης

Πλήθος καταστάσεων

- 8-puzzle: 9! = 362.880 καταστάσεις
- 15-puzzle: 16! ~ 2.09×10^{13} καταστάσεις
- 24-puzzle: 25! ~ 10^{25} καταστάσεις

Προσβασιμότητα

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

Αρχική κατάσταση

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

?

Στόχος

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

Αλγόριθμοι αναζήτησης λύσης

Αλγόριθμοι τυφλής αναζήτησης λύσης

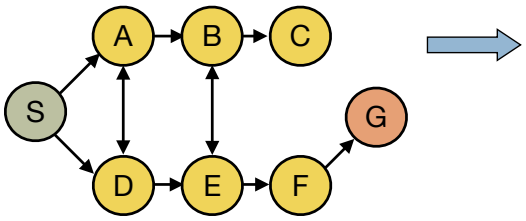
- Δεν έχουμε τρόπο να εκτιμήσουμε το κόστος των εναλλακτικών για το επόμενο βήμα (σε σχέση με τον τελικό στόχο), δηλαδή ψάχνουμε τη λύση χωρίς ένδειξη για το αν πλησιάζουμε σε κατάσταση στόχο

Αλγόριθμοι εμπεριστατωμένης (informed) αναζήτησης λύσης

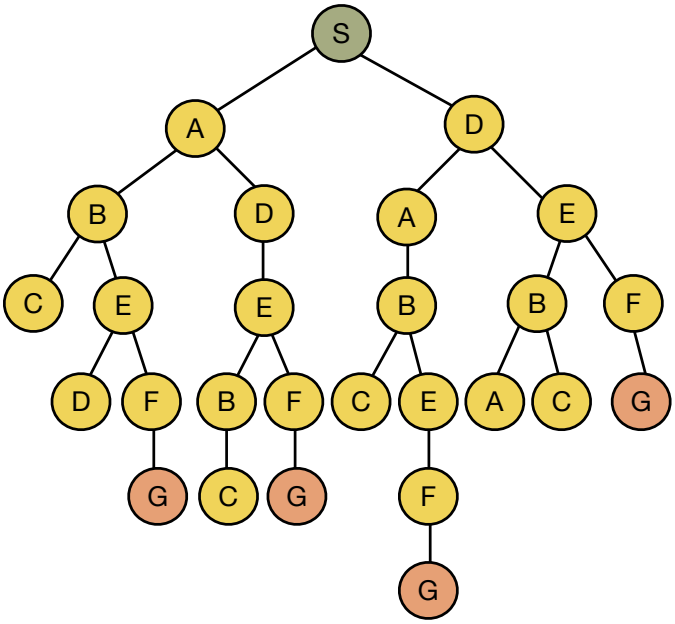
- Μπορούμε να υπολογίσουμε την πραγματική απόσταση της κατάστασης από τις επόμενες ή/και να εκτιμήσουμε την απόστασή της από τον στόχο, με βάση κάποια ευρετική συνάρτηση
  [Η ευρετική τιμή δεν είναι η πραγματική τιμή της απόστασης από μία κατάσταση στο στόχο, αλλά απλά μία εκτίμηση]

Αλγόριθμοι αναζήτησης - Πληροφορίες

Καμμία πληροφορία

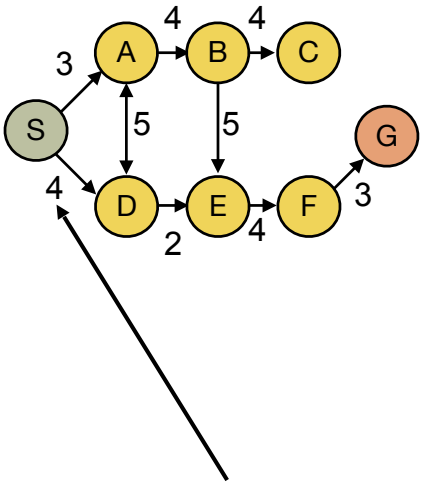


Απόσπασμα ενός δένδρου εκτέλεσης



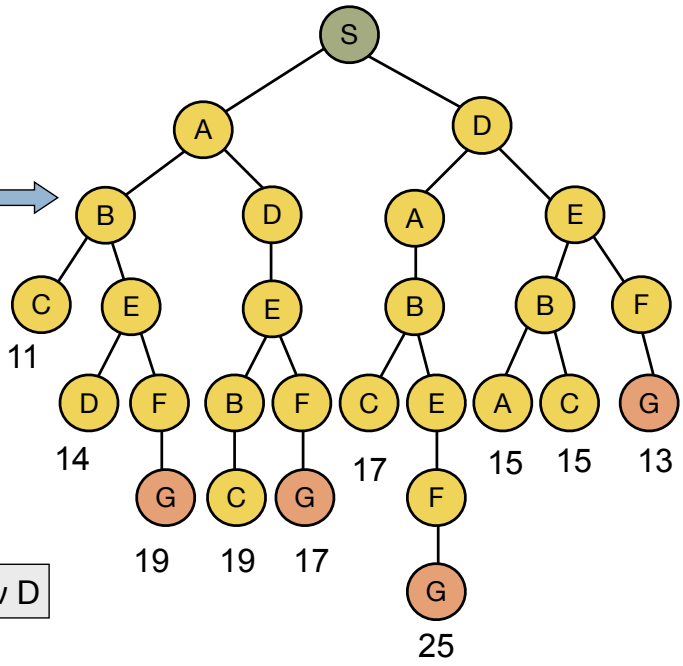
Αλγόριθμοι αναζήτησης - Πληροφορίες

Εκτίμηση κόστους μετάβασης από κατάσταση σε κατάσταση



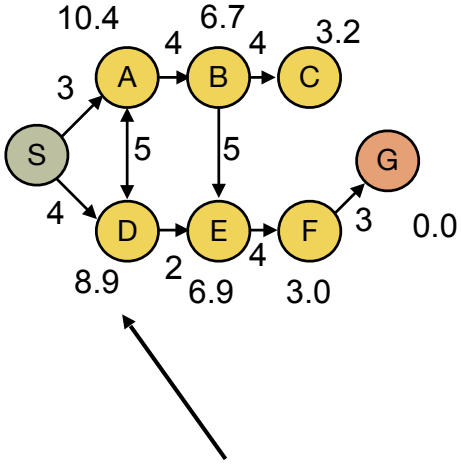
Κόστος μετάβασης από την S στην D

Απόσπασμα ενός δένδρου εκτέλεσης



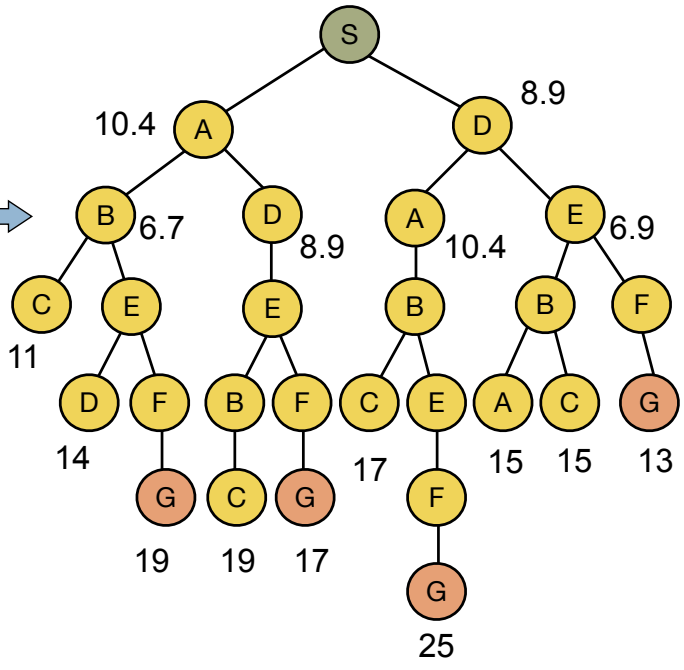
Αλγόριθμοι αναζήτησης - Πληροφορίες

Εκτίμηση κόστους μετάβασης από μία κατάσταση στο στόχο



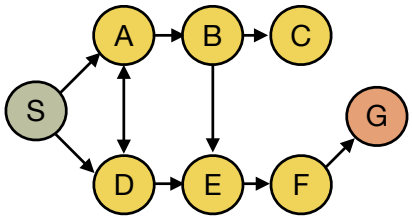
Εκτίμηση υπολειπόμενου κόστους από την D ως τη G

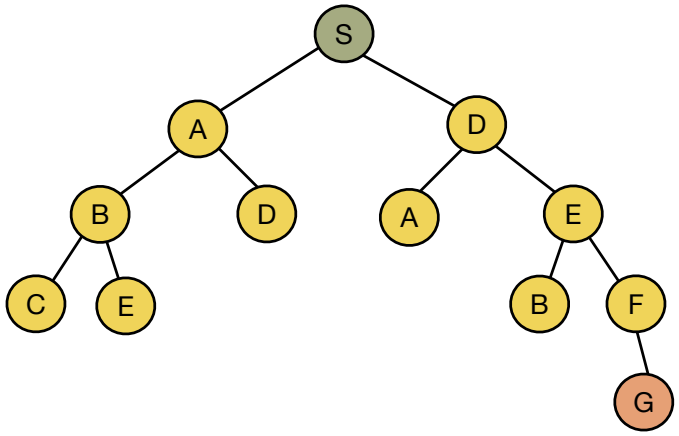
Απόσπασμα ενός δένδρου εκτέλεσης



Αλγόριθμοι τυφλής αναζήτησης

Αλγόριθμος αναζήτησης κατά πλάτος (BFS)





Μέτωπο αναζήτησης

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

Κλειστό σύνολο

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

Αλγόριθμος αναζήτησης κατά πλάτος (BFS)

Παράμετροι

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

Πολυπλοκότητα

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

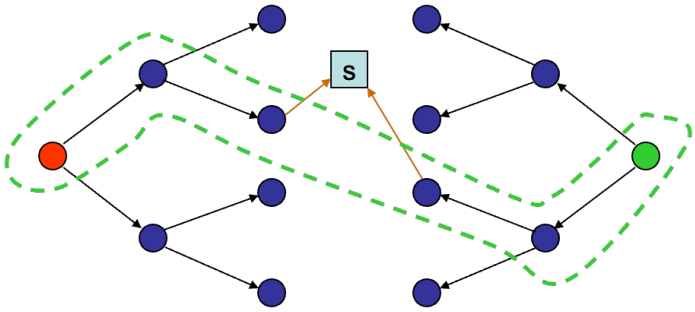
Αλγόριθμος αναζήτησης κατά πλάτος (BFS)

Υπόθεση

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

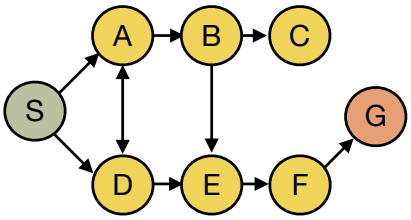
d	# Κόμβων	Χρόνος	Μνήμη
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	~10^6	1 sec	100 Mb
8	~10^8	100 sec	10 Gbytes
10	~10^{10}	2.8 hours	1 Tbyte
12	~10^{12}	11.6 days	100 Tbytes
14	~10^{14}	3.2 years	10,000 Tbytes

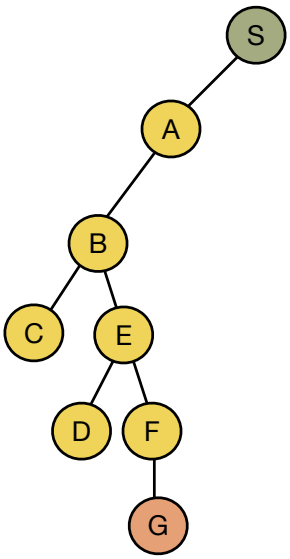
Αλγόριθμος διπλής κατεύθυνσης



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

Αλγόριθμος αναζήτησης κατά βάθος (DFS)





Μέτωπο αναζήτησης

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

Κλειστό σύνολο

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

Αλγόριθμος αναζήτησης κατά βάθος (DFS)

Παράμετροι

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

Πολυπλοκότητα

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

Αλγόριθμος επαναληπτικής εκβάθυνσης

Βήματα Αλγόριθμου (Iterative Deepening - ID)

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

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

Βήμα 3. Αν έχεις βρει λύση σταμάτησε.

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

Βήμα 5. Πήγαινε στο Βήμα 2.

- Έχει παρόμοια πολυπλοκότητα σε χώρο και χρόνο με τους DFS και BFS
- Όταν έχουμε μεγάλους χώρους αναζήτησης, παρόλο που επαναλαμβάνει άσκοπα το κτίσιμο του χώρου αναζήτησης, δεν έχει μεγάλη επιβάρυνση σε σχέση με την κατά-πλάτος αναζήτηση

Αλγόριθμος επαναληπτικής εκβάθυνσης

Παράμετροι

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

Πολυπλοκότητα

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

Αλγόριθμος επαναληπτικής εκβάθυνσης

- b = 2; d=5

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

- b = 10; d=5

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

Αλγόριθμοι εμπεριστατωμένης αναζήτησης
(αναζήτησης μίας οποιασδήποτε λύσης)

Ευρετικοί μηχανισμοί

Ευρετικός μηχανισμός και συναρτήσεις σε λαβύρινθο

Ευκλείδεια απόσταση (Euclidian distance):

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

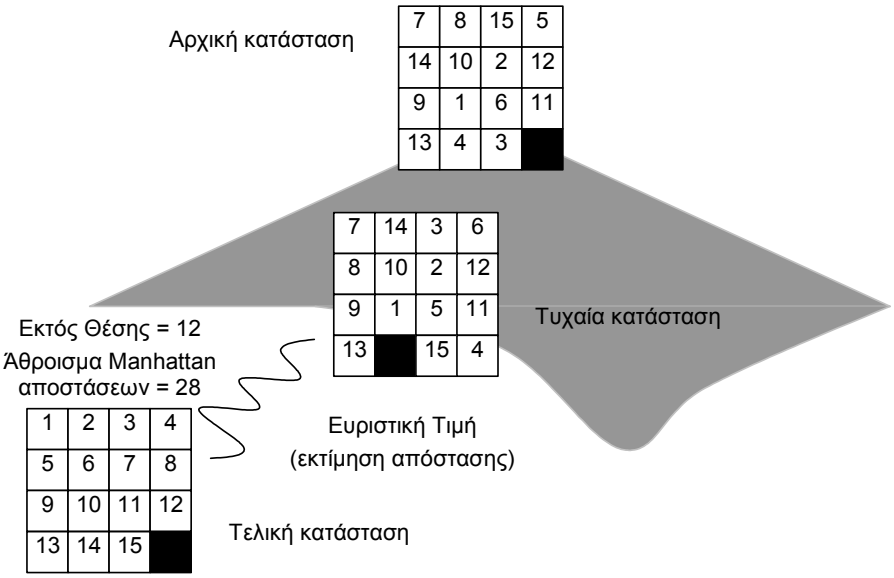
Απόσταση Manhattan (Manhattan distance):

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

Ευρετικοί μηχανισμοί

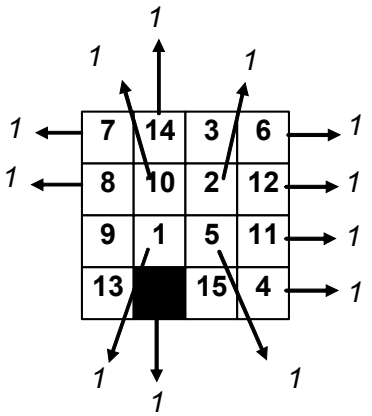
Ευρετικός μηχανισμός και συναρτήσεις στο N-Puzzle

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

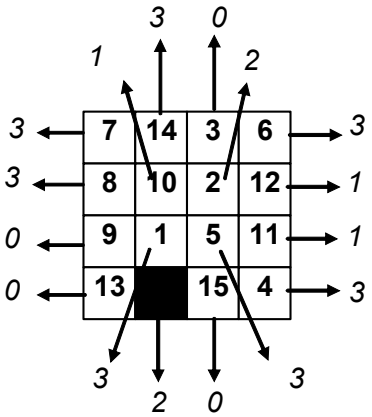


Ευρετικοί μηχανισμοί

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



Εκτός θέσης = 12



Άθροισμα αποστάσεων Manhattan = 28

Αλγόριθμος Hill climbing

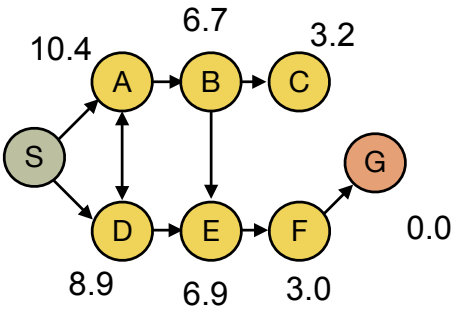
Βήμα 1: Όρισε τον τρέχοντα κόμβο ως τη ρίζα του δένδρου

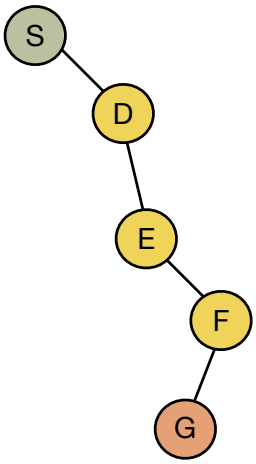
Βήμα 2: Όσο ο τρέχων κόμβος δεν είναι κόμβος στόχος, εκτέλεσε:

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

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

Αλγόριθμος Hill climbing





[S]

[D]

[E]

[F]

[G]

[S]

[D,S]

[D,E,S]

[D,E,F,S]

Αλγόριθμος Best-First

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

Αλγόριθμος Best-First

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

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

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

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

- Πώς θυμόμαστε το μονοπάτι για το στόχο????

Αλγόριθμος Best-First

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

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

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

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

Αλγόριθμοι εμπεριστατωμένης αναζήτησης (αναζήτηση βέλτιστης λύσης)

Στρατηγική αλγορίθμων εύρεσης βέλτιστης λύσης

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

Αλγόριθμος British Museum

- Βρίσκουμε όλα τα μονοπάτια από την αρχική κατάσταση σε όλες τις καταστάσεις «στόχους» και επιλέγουμε το καλύτερο μονοπάτι.
- Μπορούμε να βρούμε όλα τα μονοπάτια με αναζήτηση κατά-βάθος ή αναζήτηση κατά-πλάτος. Μόνο που εδώ δεν σταματάμε όταν βρούμε λύση. Συνεχίζουμε μέχρι να βρούμε όλες τις λύσεις για να επιλέξουμε τη καλύτερη.
- Ο αλγόριθμος αυτός δεν είναι πρακτικός στις περισσότερες περιπτώσεις. Για παράδειγμα με παράγοντα διακλάδωσης b=10 και βάθος δένδρου d=10 έχουμε 10 δισεκατομμύρια μονοπάτια

Αλγόριθμος Branch and Bound

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

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

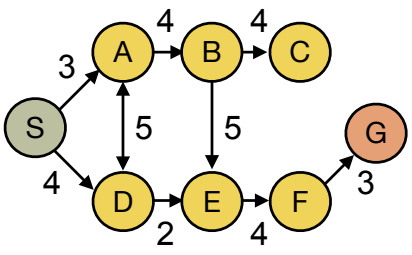
- **Βήμα 2.α**: Εάν το πρώτο μονοπάτι οδηγεί σε στόχο, κράτησέ το σαν πιθανή λύση. Εάν είναι καλύτερο από κάποια προηγούμενη λύση, κράτησέ το σαν την καλύτερη πιθανή λύση και Προχώρησε στο Βήμα 2.β

Αλγόριθμος Branch and Bound

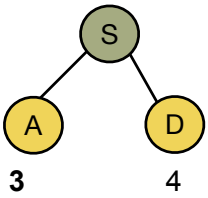
- **Βήμα 2.β**: Εάν το πρώτο μονοπάτι δεν οδηγεί σε στόχο, ή υπάρχουν άλλα μονοπάτια δεν έχουν ακόμη οδηγήσει σε στόχο και έχουν μικρότερο κόστος από το μονοπάτι που ήδη βρήκαμε
  - Βήμα 2.β.1. Βγάλε το πρώτο μονοπάτι από τη λίστα
  - Βήμα 2.β.2. Φτιάξε μονοπάτια που μπορούν να φτιαχτούν από το μονοπάτι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

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

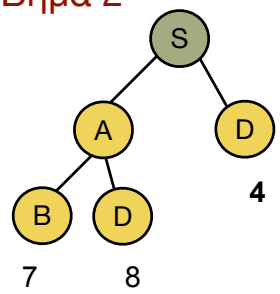
Αλγόριθμος Branch and Bound



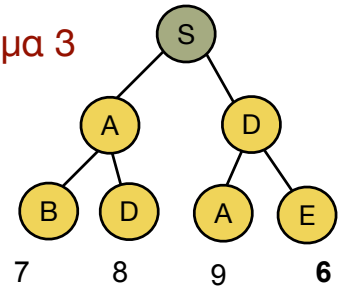
Βήμα 1



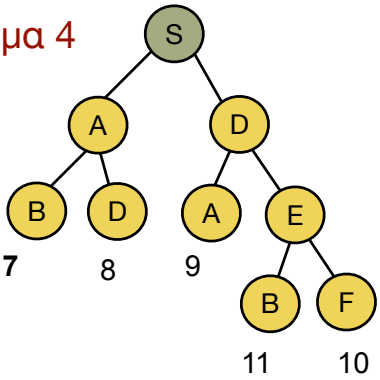
Βήμα 2



Βήμα 3



Βήμα 4

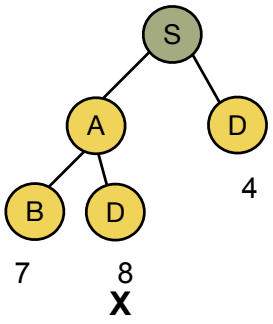


Δυναμικός προγραμματισμός (DP)

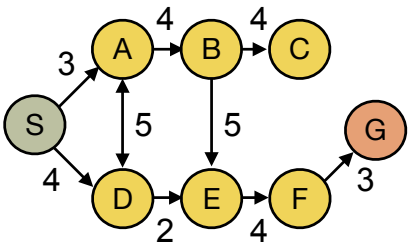
Διαισθητικά

- Επεκτείνουμε δυναμικά το όριο απόρριψης κάποιου μονοπατιού:

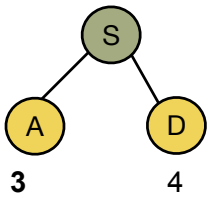
  Αν κατασκευάσουμε ένα μονοπάτι που οδηγεί σε κάποιο κόμβο, ενώ υπάρχει κάποιο άλλο μονοπάτι που οδηγεί στον ίδιο κόμβο αλλά με μικρότερο κόστος, τότε «κλαδεύουμε» το μονοπάτι με το μεγαλύτερο κόστος



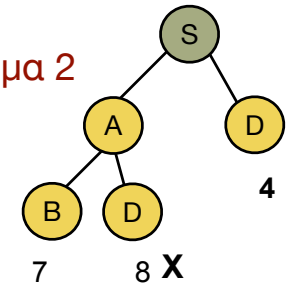
DP και Branch and Bound



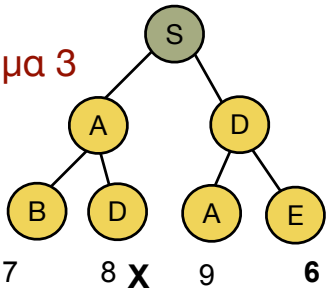
Βήμα 1



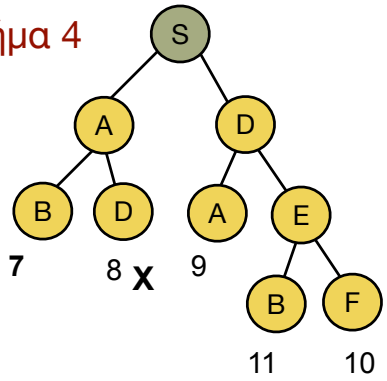
Βήμα 2



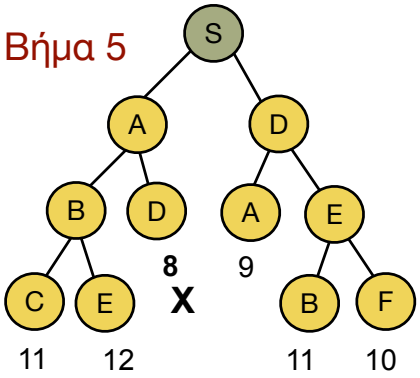
Βήμα 3



Βήμα 4



Βήμα 5



Αλγόριθμος A*

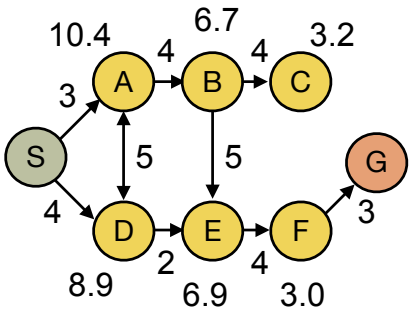
- Ακολουθούμε τη λογική του Branch and Bound
- Επεκτείνουμε το μονοπάτι με τον καλύτερο από όλους τους κόμβους που βρίσκονται στο μέτωπο αναζήτησης του δένδρου (λογική Best First).
- Για το σκοπό αυτό χρησιμοποιούμε τη σύνθετη ευρετική συνάρτηση $F(k) = g(k) + h(k)$

  $g(k)$ η απόσταση της k από την αρχική κατάσταση, η οποία είναι πραγματική και γνωστή

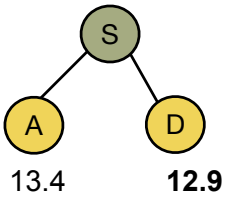
  $h(k)$ μία εκτίμηση της απόστασης της k από το στόχο (μέσω μιας ευρετικής συνάρτησης)

- Εφαρμόζουμε για οριοθέτηση (bound) δυναμικό προγραμματισμό

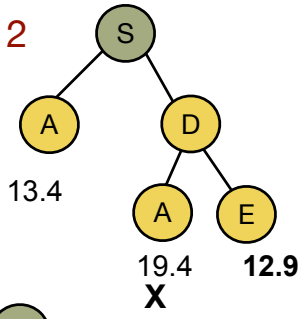
Αλγόριθμος A*



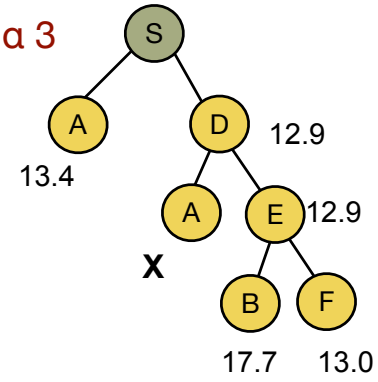
Βήμα 1



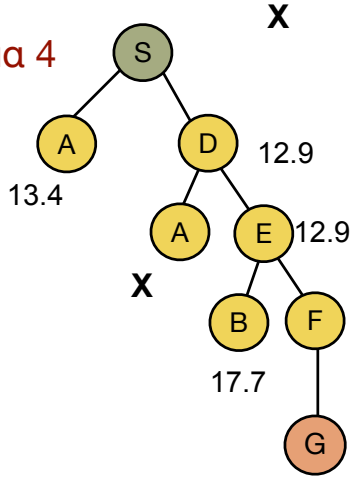
Βήμα 2



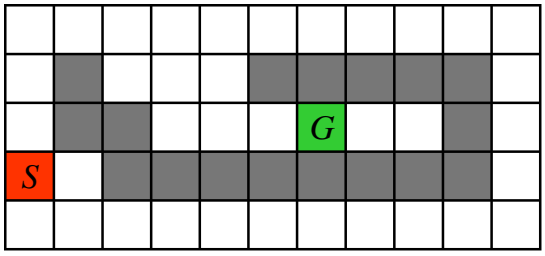
Βήμα 3



Βήμα 4



Ευρετικές συναρτήσεις



h : Απόσταση Manhattan

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

$h(k)$

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

$h(k) + g(k)$