Σπανάκης Δημήτριος – sdi1800183

Πρόβλημα 2

Για το μεγαλύτερο αριθμό expanded κόμβων, θα πρέπει να κάνουμε IDS μέχρι το βάθος d. Ο μεγαλύτερος αριθμός κόμβων θα προκύψει από την παρακάτω σχέση:
Ο αριθμός των κόμβων που δημιουργούνται από την IDS είναι:

$$b^g + 2b^{g-1} + 3b^{g-2} + \dots + kb = b^g(1 + 2b^{-1} + 3b^{-2} + \dots + gb^{1-g})$$

That equals to: $b^g(\frac{b}{b-1})^2$ (1)

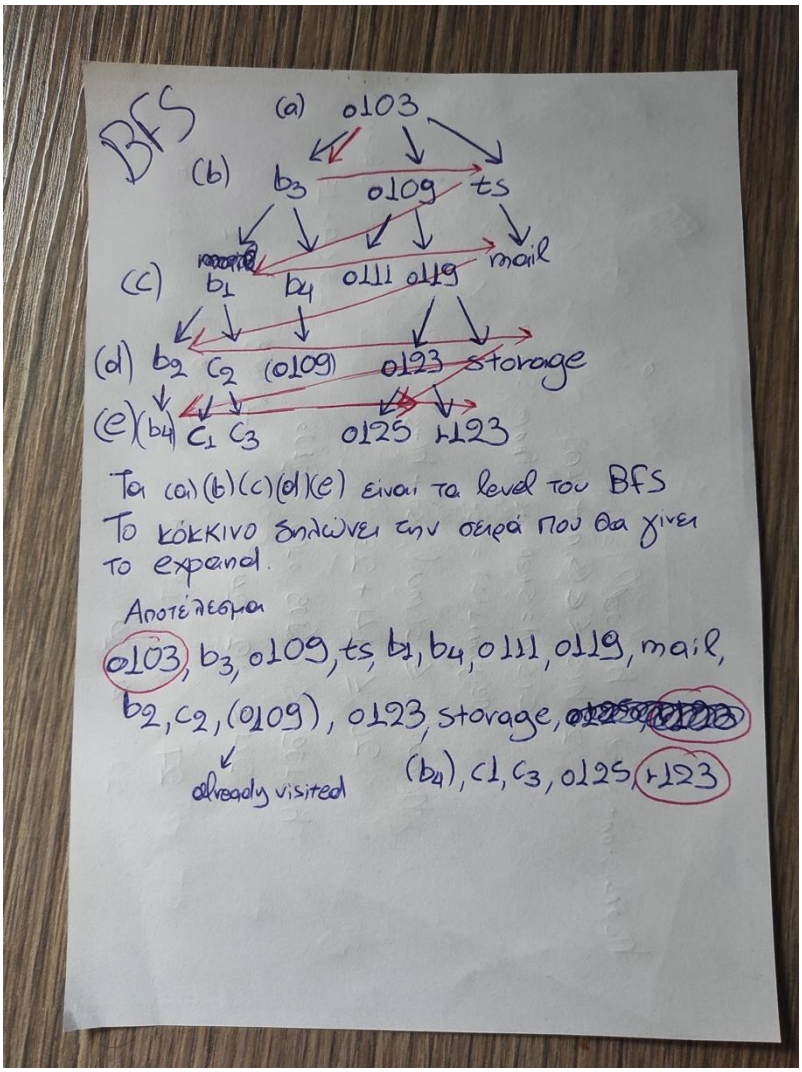
Οι λιγότεροι κόμβοι θα είναι για g = 0, όπου η παραπάνω σχέση κάνει O(1), και είναι και προφανές αφού είναι η περίπτωση η ρίζα να είναι και η λύση.
Για g = d έχουμε του περισσότερους κόμβους που θα γίνουν expand. (1)
Πολλοί κόμβοι ξανά υπολογίζονται σε κάθε loop του IDS (π.χ. αν έχουμε depth = g, τα nodes για g – 1 βάθος, θα είναι η δεύτερη φορά που θα υπολογίζονται, αφού τα υπολογίσαμε όταν είχαμε depth = g-1 και αυτό συνεχίζεται όπου στο βάθος 1 υπολογίζουμε k φορές τα ίδια nodes. Με αυτό τον τρόπο προκύπτει και η παραπάνω σχέση.

Πρόβλημα 3

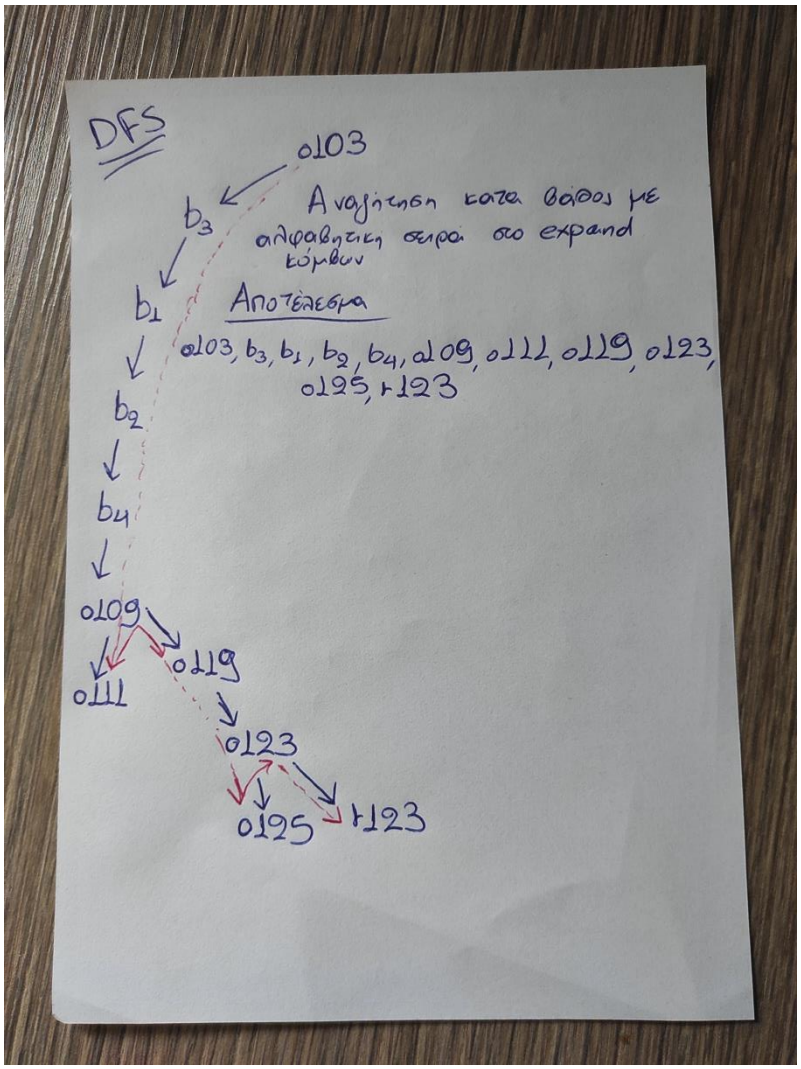
a) Αν μια συνάρτηση είναι συνεπής είναι και παραδεκτή, οπότε αρκεί να αποδείξουμε την συνέπεια της h(…). Δηλαδή αρκεί να αποδείξουμε ότι η h() δεν υπερεκτιμά ποτέ το κόστος να πάμε από ένα κόμβο σε ένα άλλο, δηλαδή αρκεί να αποδείξουμε την σχέση : $h(s) \leq c(s,n) + h(n)$ για κάθε s με n τους γειτονικούς του κόμβους και c(s,n) το κόστος από τον s στον n. Για αρχή έχουμε ότι h(Goal) = h(r123) = 0 το οποίο είναι προϋπόθεση. Έπειτα βάζοντας όλες τις διαθέσιμες τιμές και υπολογίζοντας παρατηρούμε ότι η συνθήκη επαληθεύεται πάντα, οπότε είναι συνεπής και παραδεκτή.
b) Ακολουθούν οι εικόνες με το πώς βγαίνει το κάθε node από τα σύνορα:

Θα τα βρείτε από κάτω, συγνώμη αλλά δεν μπόρεσα με τίποτα να τα φορμάρω…….

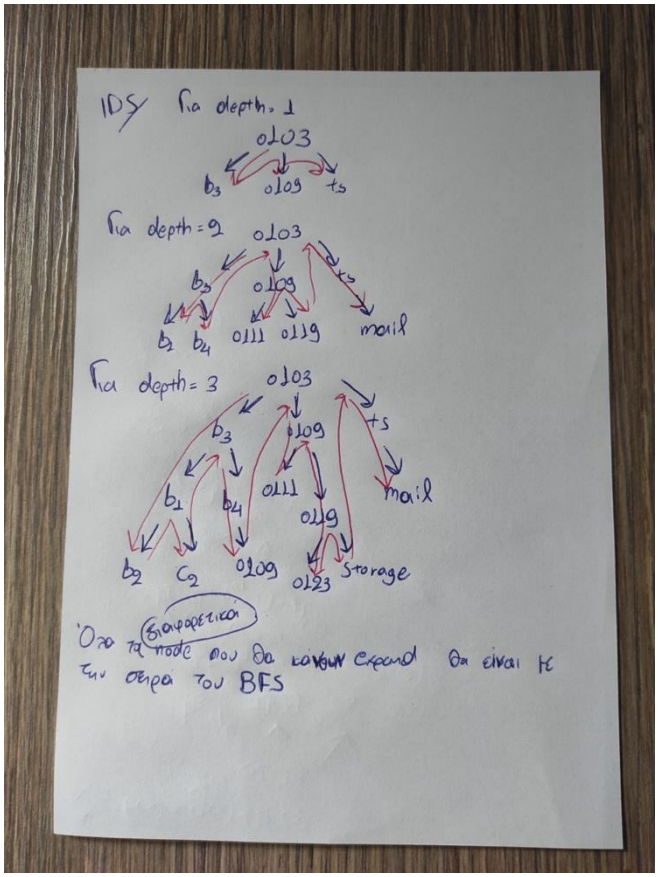
a. BFS

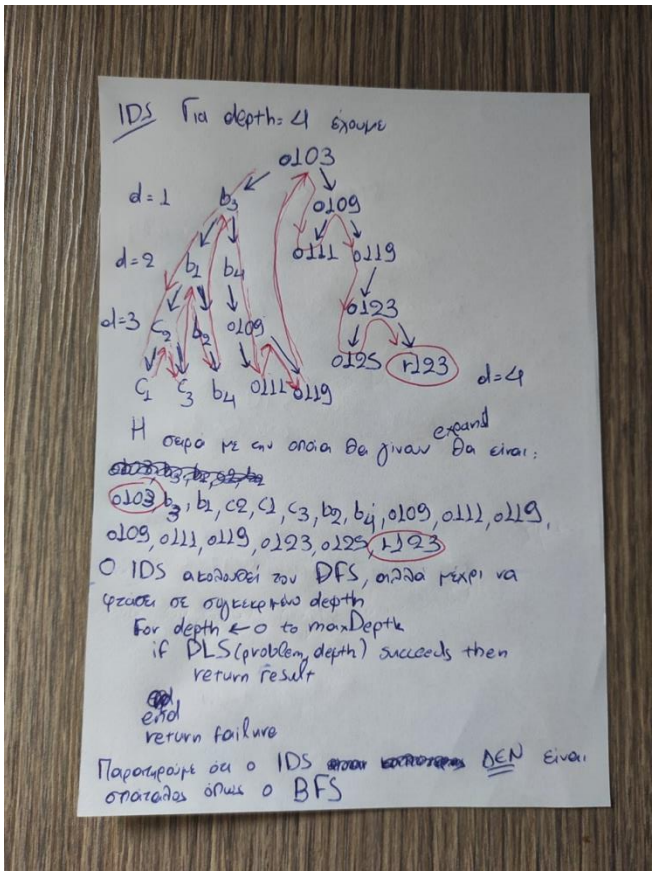

BFS
(a) o103
(b) b3 o109 ts
(c) b1 b4 o111 o119 mail
(d) b2 c2 (o109) o123 storage
(e) (b4) c1 c3 o125 +123
Τα (a)(b)(c)(d)(e) είναι τα level του BFS
Το κόκκινο σηλώνει την σειρά που θα γίνει το expand.
Αποτέλεσμα
o103, b3, o109, ts, b1, b4, o111, o119, mail,
b2, c2, (o109), o123, storage,
already visited
(b4), c1, c3, o125, +123

b. DFS

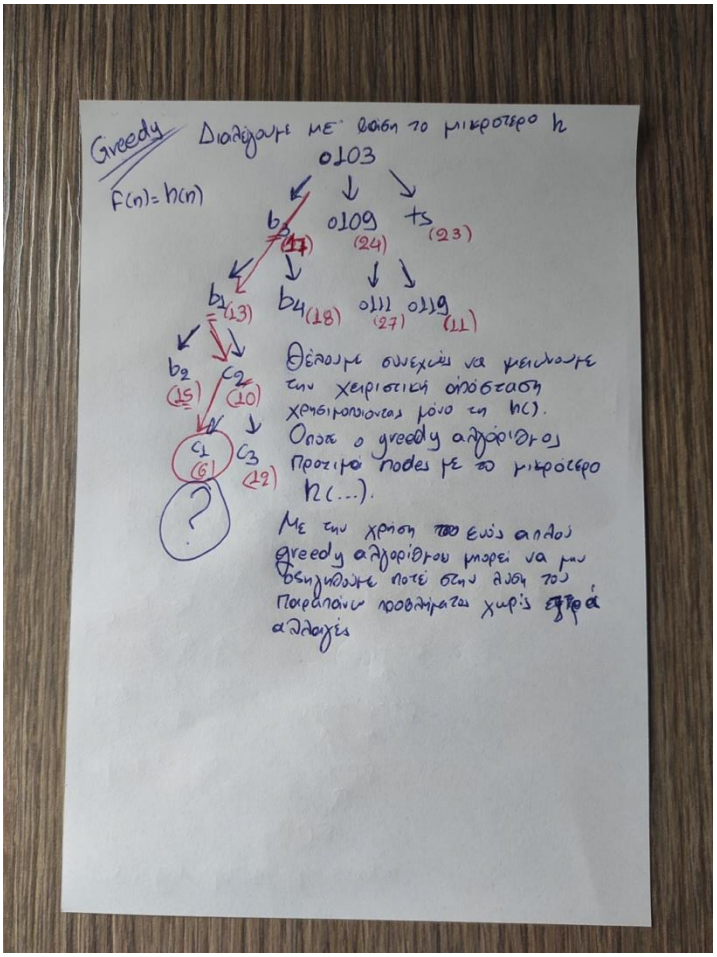

DFS
o103
b3
Αναζήτηση κατα βάθος με αλφαβητική σειρά στο expand κόμβων
b1
Αποτέλεσμα
o103, b3, b1, b2, b4, o109, o111, o119, o123, o125, r123
b2
b4
o109
o111
o119
o123
o125
r123

c. IDS


IDS Για depth=1
o103
b3
o109
ts
Για depth=2
o103
b3
o109
ts
b1
b4
o111
o119
mail
Για depth=3
o103
b3
o109
ts
b1
b4
o111
o119
mail
b2
c2
o109
o123
storage
Όλα τα nodes που θα κάνουν expand θα είναι με την σειρά του BFS
(διαφορετικά)


IDS Για depth=4 έχουμε
o103
d=1
b3
o109
d=2
b1
b4
o111
o119
d=3
c2
b2
o109
o123
c1
c3
b4
o111
o119
o125
r123
d=4
Η σειρά με την οποία θα γίνουν expand θα είναι:
o103, b3, b1, c2, c1, c3, b2, b4, o109, o111, o119, o109, o111, o119, o123, o125, r123
Ο IDS ακολουθεί τον DFS, αλλά μέχρι να φτάσει σε συγκεκριμένο depth
for depth ← 0 to maxDepth
if DLS(problem, depth) succeeds then
return result
end
return failure
Παρατηρούμε ότι ο IDS ΔΕΝ είναι σταταδιακός όπως ο BFS

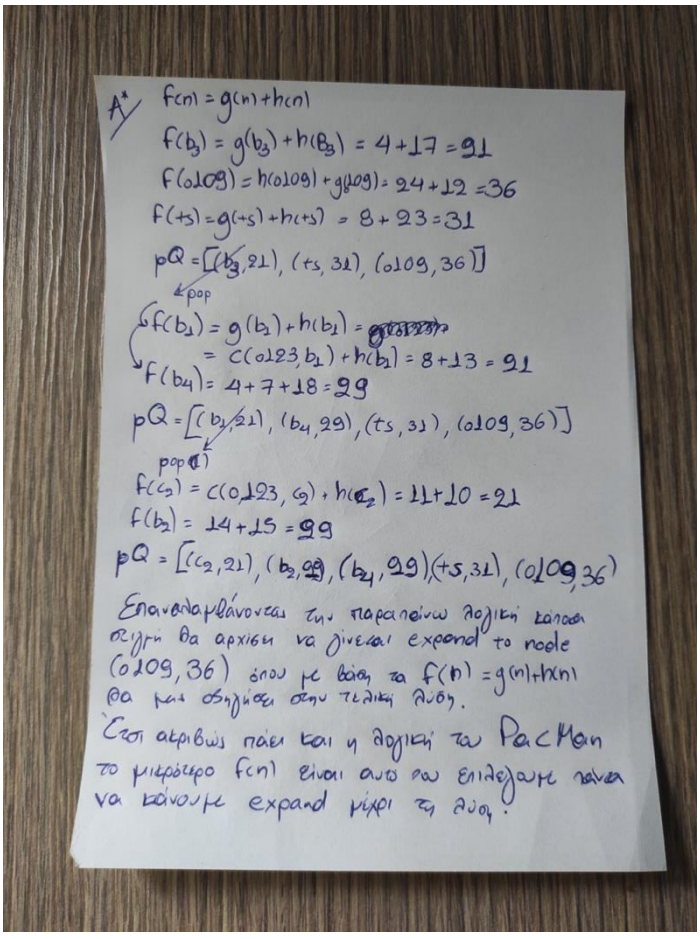
d. Greedy



Παρατηρούμε ότι η h, ξεκινώντας από τον o103 κόμβο μας κατευθύνει άπληστα προς τον c1 κόμβο όπου αφού φτάσουμε εκεί μετά πρέπει να ξεκινήσουμε από κόμβο με μεγαλύτερο h, για να φτάσουμε στην λύση.

Με κόκκινο φαίνονται τα h(n) και με τα βελάκια η προτίμηση του greedy αλγορίθμου στο ποια nodes θα γίνουν expand.

e. A*



Ο A* που παρουσιάζεται λειτουργεί όπως και του Pacman. Ορίζουμε priority queue με priority => f(n) = g(n) + h(n)
Όπου g(n) το αληθινό κόστος να πάμε από το starting point στον n. Έτσι κάνουμε pop() από την pq και ύστερα expand το node που πήραμε. Κάποια στιγμή λοιπόν θα οδηγηθούμε στη λύση, μέσω της f().

Παρατήρηση: Στο συγκεκριμένο παράδειγμα βλέπουμε ότι h μπορεί και μας απομακρύνει από τον κόμβο στόχου καθώς μας οδηγά αρχικά προς τον κόμβο c1, ο οποίος ενώ με βάση την χειριστική είναι «κοντά» στον κόμβο στόχου, στην πραγματικότητα μας απομακρύνει και άλλο από την βέλτιστη διαδρομή.

Προβλημα 4

A)Το πρόβλημα ορίζεται σαν πρόβλημα αναζήτησης με μαθηματικό τρόπο ως εξής:

Χώρος καταστάσεων: Όλα τα δωμάτια δωμάτια που μπορεί να επισκεφτεί το ρομπότ.

Αρχική κατάσταση: Αρχική θέση στο mail, και αρχικοποίηση πίνακα με όλες τις παραδώσεις σε κάθε δωμάτιο = False.

Κατάσταση στόχου: Να έχουμε curState = mail, και ο πίνακας με όλες τις παραδώσεις που έχουν γίνει για κάθε δωμάτιο, να είναι για όλα True.

Συνάρτηση διαδόχων: Όλα δωμάτια(successors) στα οποία μπορώ να «πάω» από αυτό που βρίσκομαι και να πάρω ένα δέμα. Αν το ρομπότ κουβαλάει ήδη ένα δέμα successor μπορεί να αποτελεί το μέρος που πρέπει να το αφήσει.

Συνάρτηση Κόστους: $c(s, a, s')$ Όπου s = curState, s' = nextState, και a είναι η ενέργεια στην οποία θα μεταβώ για να πάω στον s'. Πχ. Από το o103 στο ts το κόστος είναι 8.

B)Στόχος μας είναι να παραδώσουμε όλη την αλληλογραφία και να επιστρέψουμε στο mail δωμάτιο. Πρέπει να σκεφτούμε μια ευρετική που να μην υπερεκτιμά πότε το αληθινό κόστος μετάβασης από ένα κόμβο σε ένα άλλο.
Μια καλή ιδέα θα ήταν η χειριστική να υπολογίζει την απόσταση που πρέπει να διανύσει το ρομπότ για να παραδώσει το δέμα **+** την απόσταση μέχρι το κοντινότερο δέμα. Το δεύτερο μέρος του αθροίσματος κάνει τον αλγόριθμο παραδεκτό καθώς διαφοροποιεί τα h(..). Η χειριστική συνάρτηση είναι επίσης αποδεκτή αφού αδιαφορεί για τείχους και μετράει απλά την «αληθινή» απόσταση μεταξύ nodes.

Πρόβλημα 5

Η χρήση bidirectional search βοηθάει στην γρηγορότερη εύρεση μονοπατιών , και είναι optimal όταν οι αλγόριθμοι αναζήτησης που ξεκινάνε από τις δύο άκρες συναντηθούν στην μέση με λύση αφού θα έχουμε O($2b^{g/2}$) για χρόνο εύρεσης αντί για $O(b^k)$. Αυτά ισχύουν με την προϋπόθεση ότι τρέχουμε bfs και από τις δύο πλευρές.

a.) BFS & DLS
Ο παραπάνω αλγόριθμος θα είναι βέλτιστος αν κάθε φορά που κάνουμε expand ένα level του bfs , καλούμε και τον DLS για τον ίδιο αριθμό level (π.χ. αν κάνουμε expand το 3ο level του bfs, τότε καλούμε και τον DLS για depth = 3 αφού πρώτα ελέγξουμε αν ήδη υπήρχε λύση). Έτσι είναι σίγουρο πως ποτέ δεν θα προσπεράσει η μια εκτέλεση την άλλη και θα συναντηθούν αναγκαστικά αν υπάρχει λύση.

b.) IDS & DLS
Και εδώ θα εφαρμόσουμε την ίδια νοοτροπία με το (α) ερώτημα. Επειδή δεν ξέρουμε το depth της λύσης θα τρέξουμε τον IDS και στο τέλος κάθε loop * του θα τρέξουμε και τον DLS από την άλλη πλευρά , με depth ίσο με αυτό που βρίσκεται ο IDS, εξασφαλίζοντας πάλι την πληρότητα και ότι θα συναντηθούν οι δύο αλγόριθμοι.

*(αφού εξετάσουμε αν φτάσαμε σε frontier του άλλου αλγορίθμου)

c.) A* & DLS

Για κάθε pop() από την queue το astar και expand που γίνεται σε nodes, θα ελέγχεται αν υπάρχει κάποια λύση στους frontiers και αλλιώς θα καλείται η DLS σαν να ήταν η IDS, δηλαδή από depth =1 μέχρι depth = pop().count() έτσι ώστε να εξασφαλίσουμε ότι ο astar δεν θα πάρει κάποιο λάθος μονοπάτι και η DLS θα αρχίσει να κοιτάει πέρα τον μονοπατιών που θα κάνει expand μετά ο astar(βλ. c1 node από πρόβλημα 3). Ο αλγόριθμος δεν θα είναι βέλτιστος καθώς μπορεί να συναντηθεί τοπικά καλύτερο μονοπάτι του astar με μονοπάτι του DLS και να επιστραφεί η λύση.

d.) A* & A*

Εφαρμόζοντας δύο απλούς astar αλγορίθμους δεν είναι σίγουρο ότι η αλγόριθμοι μας θα συναντηθούν καθώς μπορεί ο ένας να «πέσει» σε ένα μεγάλο κύκλο και ο δεύτερος να τον «περάσει» (βλ. τον κόμβο c1 του προβλήματος 3 , όπου η h μας κατευθύνει λάθος αρχικά). Έτσι δεν έχουμε ούτε βέλτιστο bidirectional algorithm, ούτε πλήρης γιατί δεν είναι σίγουρο ότι θα συναντηθούν οι δυο astar.

Ένας τρόπος να γίνουν αποδοτικοί οι δύο astar είναι να λειτουργούν συμπληρωματικά ο ένας με τον άλλο και να επιλέγουν κάθε φορά ένα ζευγάρι από nodes με g(start,x) + h(x,y) + g(y,goal) έτσι ώστε να οδηγηθούν χειριστικά ο ένας στον άλλο.