Project 1 – Artificial intelligence

Άρης Τσιλιφώνης 1115201700170

Θεωρητικές ασκήσεις

Ερώτημα 2

Θέλουμε να βρούμε τον μέγιστο και τον ελάχιστο αριθμό κόμβων που παράγονται μέσω της αναζήτησης πρώτα κατά βάθος με επαναληπτική εκβάθυνση(IDDFS).Παρατηρώντας τον αλγόριθμο βλέπουμε ότι τους κόμβους του d επιπέδου(που είναι b^d)ανακαλύπτονται 1 φορά. Με την ίδια σκέψη το d-1 επίπεδο έχει b^{d-1} κόμβους που ανακαλύπτονται 2 φορές (στο συγκεκριμένο επίπεδο και στο παρακάτω).Συνεπώς για τον μέγιστο αριθμό κόμβων προκύπτει η εξίσωση:

$b^d+2b^{d-1}+3b^{d-2}+\ldots+(d-1)b^2+bd+(d+1) = (x+a)^n = \sum_{k=0}^{n} b^d(d+1-i)$ **.(1)**

Παραγοντοποιώντας με b^d

$b^d (1+2b^{-1}+3b^{-2}+\ldots+(d-1)b^{2-d}+db^{1-d}+(d+1)\, b^{-d})$

θέτοντας με $b^{-1}=x$ προκύπτει :

$b^d (1+2x+3x^2+\ldots+(d-1)x^{d-2}+dx^{d-1}+(d+1)x^d)$

το οποίο είναι μικρότερο από την άπειρη σειρά

$b^d (1+2x+3x^2+\ldots) = b^d \sum_{n=0}^{\infty} nx^{n-1}$.Λόγω σύγκλισης της σειράς έχουμε τελικά $b^d *(1-x)^{-2}$

Ο παράγοντας $(1-x)^{-2}$ είναι ανεξάρτητος του d.Άρα $O(b^d)$χείριστη πολυπλοκότητα και max αριθμός κόμβων η σειρά στην σχέση (1).Για min αριθμό κόμβων ,είναι ο ελάχιστος αριθμός κόμβων μέχρι αυτόν τον κόμβο ο οποίος μπορεί να είναι μέχρι και 0 .

Ερώτημα 3

a)Για να είναι μια συνάρτηση παραδεκτή πρέπει h(n) <= h*(n) όπου h(n) είναι η ευρετική στον κόμβο και h*(n) είναι το κόστος για να φτάσουμε από τον κόμβο στο goal state.

Εύκολα βλέπουμε ότι είναι παραδεκτή. Αρχικά παραλείπουμε όλους τους κόμβους όπου δεν υπάρχει path για το goal state (άπειρο).Ενδεικτικά

h(o123) = 4 <= h*(o123) = 4

h(o119) = 11 <= h*(o119) = 13

h(o119) = 24 <= h*(o119) = 29

Μετά οποιαδήποτε path στο goal θα έχει τουλάχιστον 29 cost και καμία ευρετική δεν έχει μεγαλύτερη τιμή από 29.Άρα είναι παραδεκτή.

Για να ελέγξουμε αν είναι συνεπής πρέπει να ισχύει για κάθε κόμβο : h(n) <= c(n,a,n') + h(n')

h(o123) = 4 <= c(o125, o123)) + h(o125) = 4+6=10

OK

h(o123) = 4 <= c(o125, r123)) + h(r125) = 4+0=4 OK

h(o119) = 11 <= c(o119, o123)) + h(o123) = 11+9=20 OK

h(o119) = 11 <= c(o119, storage)) + h(storage) = 7+12=19 OK

h(o109) = 11 <= c(o109, o119)) + h(o119) = 16+11=19 OK

h(o109) = 24 <= c(o109, o119)) + h(o119) = 16+11=27 OK

h(o109) = 24 <= c(o109, o111)) + h(o111) = 4+27=31 OK

h(o103) = 21 <= c(o103, o109)) + h(o109) = 12+24=36 OK

h(o103) = 21 <= c(o103, ts)) + h(ts) = 8+23=31 OK

h(ts) = 23 <= c(ts, mail)) + h(mail) = 6+26=32 OK

h(o103) = 21 <= c(o103, b3)) + h(b3) = 4+17=21 OK

h(b3) = 17 <= c(b3,b1)) + h(b1) = 4+13=17 OK

h(b3) = 17 <= c(b3,b4)) + h(b4) = 7 + 18 = 25 OK

h(b4) = 18 <= c(b4,o109)) + h(o109) = 7 + 24 = 31 OK

h(b1) = 13 <= c(b1,b2)) + h(b2) = 6 + 15 = 21 OK

h(b2) = 15 <= c(b2,b4)) + h(b4) = 3 + 18 = 21 OK

h(b1) = 13 <= c(b1,c2)) + h(c2) = 3 + 10 = 13 OK

h(c2) = 10 <= c(c2,c1)) + h(c1) = 4+ 6 = 10 OK

h(c2) = 10 <= c(c2,c3)) + h(c3) = 6 + 12 = 18 OK

h(c1) = 6 <= c(c1,c3)) + h(c3) = 8 + 12 = 20 OK

Άρα τελικά είναι και συνεπής

b)Η αναζήτηση πρώτα κατά πλάτος (BFS) χρησιμοποιεί σαν fringe(σύνορο) την ουρά(queue-FIFO-first in first out).Ο αλγόριθμος επιλέγει με αλφαβητική σειρά και εξάγει με FIFO.

Η σειρά με την οποία βγαίνουν οι κόμβοι είναι οι εξής:
o103,b3,o109,ts,b1,b4,o111,o119,mail,b2,c2,o123,storage,c1,c3,o125,r123

Η αναζήτηση πρώτα κατά βάθος (DFS) χρησιμοποιεί σαν fringe(σύνορο) την στοίβα (stack-LIFO-last in first out).

o103,ts,mail,o109,o119,storage,o123,r123,o125,o111,b3,b4,b1,c2,c3,c1,b2 (αν έχουμε μαρκάρει με visited δεν χρειάζεται να επισκεφτεί ξανά τους κόμβους o109 και έπειτα από το δεύτερο μονοπάτι του b4)

η σειρά με την οποία βγαίνουν οι κόμβοι από την stack/frontier (να τονιστεί ότι μπαίνουν με αλφαβητική σειρά στο frontier – δεν το έδειξα – και βγαίνουν με LIFO – αυτός που μπήκε τελευταίος βγαίνει πρώτος).

(χρησιμοποιώ αλφαβητική σειρά για την επιλογή κόμβου στην stack και εξάγω με LIFO.)

Iterative deepening DFS(μπορεί να υλοποιηθεί με

Lists)

Για depth = 0 :

Removed from fringe: o103

Για depth = 1:

Removed from fringe: o103 ,b3,o109,ts

Για depth = 2:

Removed from fringe: o103 ,b3,b1,b4,o109,o111,o119,ts,mail

Για depth = 3:

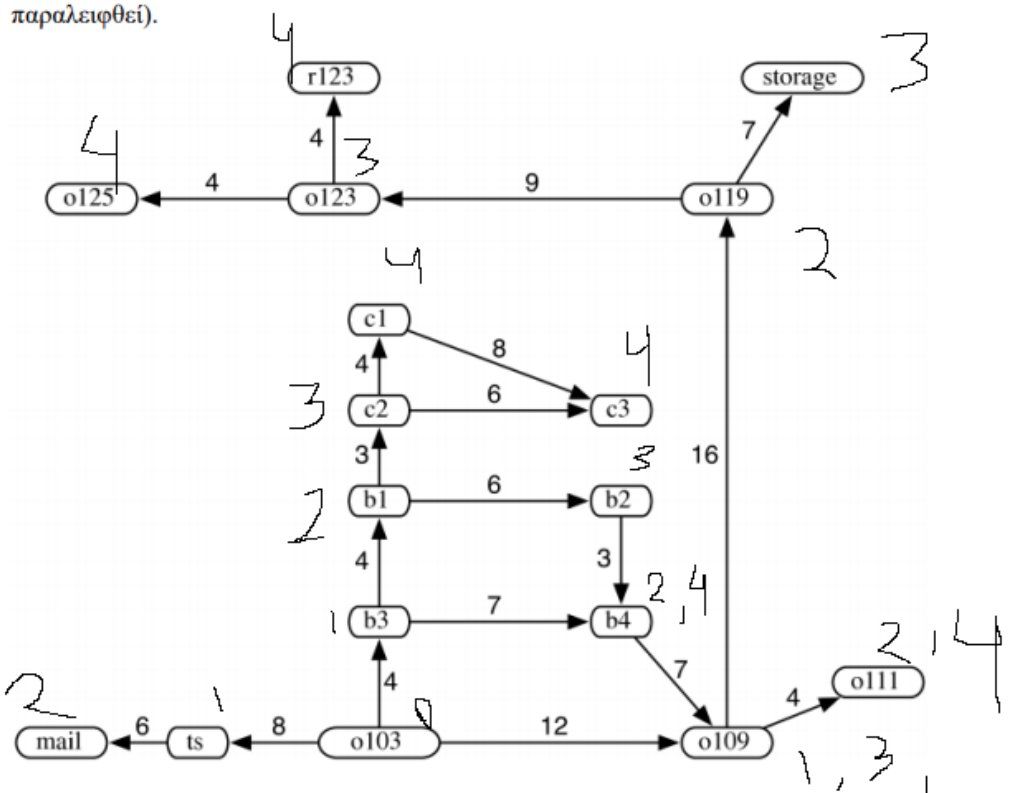
Removed from fringe: o103 ,b3,b1,b2,c2,b4,o109,(o109),o111,o119,o123,storage,ts,mail

Για depth = 4:

Removed from fringe: o103 ,b3,b1,b2, b4,c2,c1,c3,(b4),o109,o111, (,o109,o111),o119,o123,o125,r123(κανονικά εδώ σταματάει),storage,ts,mail

Από κάτω ο πίνακας με τα βάθη ,επιλογή με αλφαβητική σειρά όταν έχουν ίδιο βάθος

Θεωρήστε τώρα τον παρακάτω γράφο που παριστάνει το χώρο αναζήτησης για το ρομπότ (κάποιες τοποθεσίες και δωμάτια της προηγούμενης εικόνας έχουν παραλειφθεί).



Οι κόμβοι του γράφου αντιστοιχούν σε καταστάσεις και οι ακμές στην ενέργεια μετακίνηση του ρομπότ από μια θέση σε μια άλλη (δεν θεωρούμε προς το παρόν ότι μεταφέρει πακέττα). Οι ετικέττες στις ακμές είναι τα κόστη για κάθε ενέργεια.

Άπληστη αναζήτηση πρώτα στον καλύτερο

(Greedy best first search)

Frontier = {o103} //start

Frontier = {b3, ts,o109} // 17,23,24 ->o103

Frontier = {b1,b4, ts,o109} // 13,18,23,24 ->b3

Frontier = {c2,b2,b4, ts,o109} // 10,15,18,23,24

->b1

Frontier = {c1,c3,b2,b4, ts,o109} //
6,12,15,18,23,24

->c2

Frontier = {c3,b2,b4, ts,o109} // 12,15,18,23,24

->c1

Frontier = {b2,b4, ts,o109} // 15,18,23,24

->c3

Frontier = {b4,ts,o109} // 18,23,24

->b2

Frontier = { ts,o109} // 23,24

->b4

Frontier = {o109,mail} // 24,26

->ts

Frontier = {o119,o111,mail} // 11,27,26

->o109

Frontier = {o123,storage,o111,mail} //4,12,27,26

->o119

Frontier = {r123,o125,storage,o111,mail} // 0,6,12,27,26

->o123

Frontier = {o125,storage,o111,mail} // 6,12 ,27,26

->r123 GOAL

Χρησιμοποίησα priority queue

Η σειρά με την οποία βγαίνουν οι κόμβοι από το frontier είναι:

START o103->b3->b1->c2->c1->c3->b2->b4->ts->o109->o119->o123->r123 GOAL

A* αναζήτηση

Αλγόριθμος f(n) = g(n) + h(n)

Data structure: πάλι priority queue

Κάθε στοιχείο της ουράς θα περιέχει

(όνομα,τιμή f)

Frontier = {(o103,21)} //start

Frontier = {(b3,21), (ts,31),(o109,36)}

->o103

Frontier = {(b1,17), (b4,25),(ts,31),(o109,36)}

->b3

Frontier = {(c2,13), (b2,21)
,(b4,25),(ts,31),(o109,36)}

->b1

Frontier = {(c1,10), (c3,18),(b2,21)
,(b4,25),(ts,31),(o109,36)}

->c2

Frontier = { (c3,18),(b2,21)
,(b4,25),(ts,31),(o109,36)}

->c1

Frontier = { (b2,21) ,(b4,25),(ts,31),(o109,36)}

->c3

Frontier = {(b4,25),(ts,31),(o109,36)}

->b2

Frontier = {(ts,31),(o109,36)}

->b4

Frontier = {(mail,32),(o109,36)}

->ts

Frontier = {(o109,36)}

->mail

Frontier = {(o119,27),(o111,31) }

->o109

Frontier = { (o123,13), (storage,19) ,(o111,31) }

->o119

Frontier = {(r123,4), (o125,10) , (storage,19) ,(o111,31) }

->o123

Frontier = { (o125,10) , (storage,19) ,(o111,31) }

->r123 GOAL

Η σειρά με την οποία βγαίνουν οι κόμβοι από το frontier είναι:

o103->b3->b1->c2->c1->c3->b2->b4->ts->mail->o109->o123->r123 GOAL

Ερώτημα 4

Α)Θα ορίσω με μαθηματικό τρόπο το πρόβλημα

Αρχικά ορίζω τον χώρο καταστάσεων μου.

Επειδή τα πακέτα μπορεί να βρίσκονται οπουδήποτε στον γράφο ,ο χώρος είναι όλα τα nodes του γράφου. Κάθε κόμβος έχει ακμές προς άλλους κόμβους και μπορεί να περιέχει πακέτο προς μεταφορά. Αν έχει πακέτο τότε θα έχουμε ορίσει τοποθεσία προορισμού.

Δηλαδή state space:

{(node o103, edge A,edge B ,edge C,package,destination: b3},(node o111, edge D,edge E,NULL,destination: NULL),…},

Array packages to be delivered : [package a ,package b]

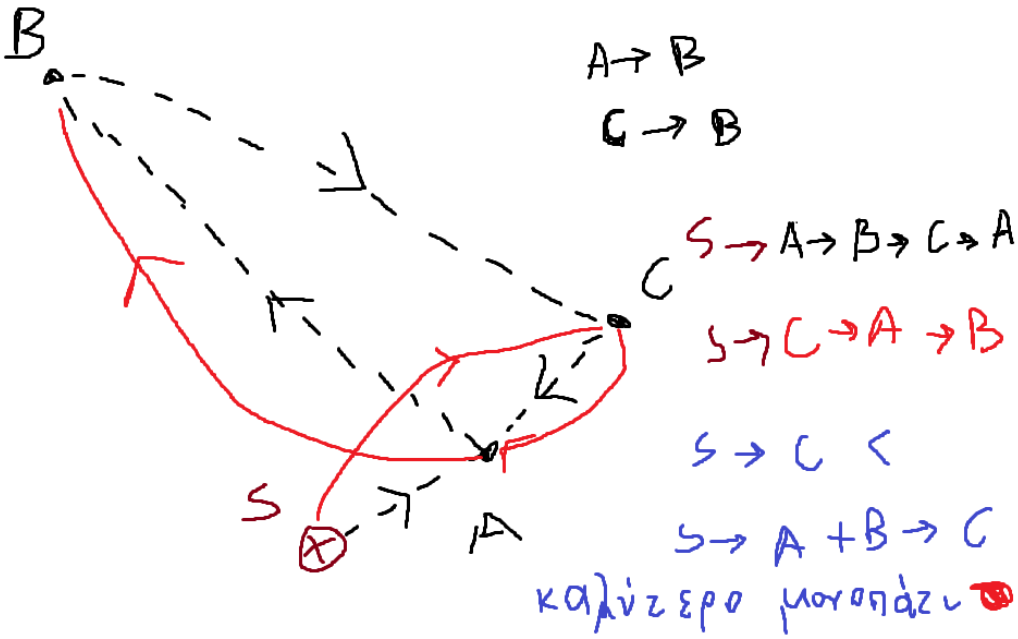
Αρχική κατάσταση μπορεί να είναι οποιοδήποτε κόμβος.

Κατάσταση στόχου είναι να αδειάσει το array με τα packages.

Η συνάρτηση διαδόχων είναι κάθε κατάσταση η οποία είναι μία «κίνηση» μετά από κάθε κατάσταση. Μπορούμε να δημιουργήσουμε αυτόν τον χώρο παρατηρώντας το τι γίνεται σε κάθε

κόμβο. Η ευρετική συνάρτηση θα οριστεί παρακάτω. Προς το παρόν γνωρίζω απλά τα κόστη των ακμών τα οποία παρατίθενται στον graph.

Β)Έστω ότι πακέτα μπορούν να υπάρχουν οπουδήποτε στον graph και ότι το ρομπότ μπορεί να μεταφέρει μόνο ένα πακέτο. Τότε μία παραδεκτή ευρετική είναι να υπολογίζει την απόσταση προς το κοντινότερο πακέτο(1) συν το άθροισμα των αποστάσεων που πρέπει να μεταφερθούν τα πακέτα (2) .Να σημειωθεί ότι η κάθε απόσταση του αθροίσματος(όρος) (2) είναι ουσιαστικά η απόσταση για να φτάσει στο πακέτο το ρομπότ συν την απόσταση για να φτάσει το πακέτο στον προορισμό του . Προφανώς δεν θα διαλέγει το κοντινό του πακέτο πάντα γιατί παίζει ρόλο και ο δεύτερος όρος .Είναι παραδεκτή γιατί ποτέ δεν υπερεκτιμά το κόστος κίνησης.(πάντα σκεφτόμαστε πρώτα το κοντινότερο πακέτο στο ρομπότ και θέλουμε να διανύσει την λιγότερη συνολική απόσταση).



Ερώτημα 5

Έχοντας μελετήσει όλους του αλγορίθμους μπορώ να βγάλω κάποια συμπεράσματα για την αμφίδρομη αναζήτηση(bidirectional search)

Για την περίπτωση (α) είναι σίγουρα πλήρης γιατί ο χώρος καταστάσεων είναι πάντα περιορισμένος (προφανώς στην DLS αν limit<= βάθος του δέντρου και b=branching factor πεπερασμένο).

Για να είναι βέλτιστος πρέπει και από τα δύο άκρα ο αλγόριθμος να είναι βέλτιστος. Στην προκείμενη περίπτωση όμως ο DLS δεν είναι πάντα βέλτιστος.

Ο BFS είναι βέλτιστος αν όλες οι ενέργειες έχουν το ίδιο μη αρνητικό κόστος.

(Β) περίπτωση IDS και DLS.

Πάλι ο χώρος καταστάσεων και για τους δύο αλγορίθμους είναι πεπερασμένος , άρα είναι πλήρης.Ο DLS μπορεί να θεωρηθεί και σαν μια ειδική περίπτωση DFS.Δεν είναι βέλτιστος ακόμα και για limit < depth(μπορεί ο goal να είναι πιο βαθιά).Ο IDS μπορεί να είναι βέλτιστος υπο τις προϋποθέσεις του BFS όμως και πάλι ο αλγόριθμος μας δεν είναι βέλτιστος λόγω DLS.

(γ)Ο αλγόριθμος A* είναι πλήρης γιατί παράγει πεπερασμένο χώρο καταστάσεων. Θα αποδείξω ότι είναι βέλτιστος με άτοπο. Έστω κάποιο άλλο path p’ είναι πιο σύντομο από κάποιο p.Ας υποθέσουμε ότι λίγο πριν επιλεγεί το p υπάρχει κάποιο μέρος p’ το οποίο επίσης υπάρχει στο frontier.Ας πούμε αυτό το part p’’.Ισχύει f(p)<f(p’’)

Επειδή p goal h(p)=0 : cost(p) <= cost(p’’) + h(p’’).

Επειδή h παραδεκτή cost(p’’) + h(p’’)<= cost(p’)

Για κάθε path p’ στο goal που επεκτείνει το p’’.

Άρα cost(p) <= cost(p’).άτοπο γιατί p’ είπαμε ότι είναι πιο σύντομο path.

(δ)Προφανώς είναι και πλήρης και βέλτιστος αφού αποδείχτηκε παραπάνω.

Όσον αφορά το δεύτερο ερώτημα , μια αποτελεσματική τερματική συνθήκη είναι όταν οι δύο αναζητήσεις διασταυρώνονται. Ωστόσο αυτό δεν είναι πάντα αποδοτικό , επειδή οι bfs, ids παράγουν πολλούς κόμβους μπορούμε να τσεκάρουμε αν η depth limited search βρήκε αποτέλεσμα με λιγότερα expansions.Για τις περιπτώσεις με τον A* , ξέρουμε ότι η ευρετική θα χρησιμοποιεί κάποια μορφή υπολογισμού της απόστασης από την source στον goal.Άρα όταν η αναζήτηση βρει την τιμή ευρετικής 0 τότε θα έχουμε κατάσταση στόχου.