# Τεχνητή Νοημοσύνη
# Στέφανος Βάβουλας – 1115201800014

## Pacman Project

**Ερώτημα 1 - dfs:** Το πρόβλημα που αντιμετώπισα ήταν το πώς να αποθηκεύω ολόκληρο το path μέχρι το current state για κάθε current state. Η λύση που βρήκα είναι το fringe να αποτελείται από tuples με 2 στοιχεία: Το 1ο στοιχείο είναι το state (num,num) και το 2ο μια λίστα (path) η οποία κρατά όλα τα directions μέχρι το current state, και η οποία επιστρέφεται όταν βρεθεί το goal state.
Επιπλέον δημιούργησα ένα set από tuples (explored), το οποίο κρατάει όλα τα states που έχουν εξερευνηθεί ώστε να μην ερευνηθούν ξανά. (υπάρχει σε όλες τις search συναρτησεις μου)
Όσον αφορά τη λογική του αλγορίθμου, το depth first search έχει τη λογική ότι εξερευνούμε πρώτα τον κόμβο με το μεγαλύτερο βάθος (σε αναπαράσταση δέντρου θέλουμε τον κόμβο στο μεγαλύτερο μη εξερευνημένο επίπεδο).
Για αυτό το λόγο, το fringe είναι ένα stack (last in – first out), με το οποίο σε κάθε επανάληψη οι successors του current state μπαίνουν πρώτοι στη λίστα, ένας από αυτούς γίνεται “popped” από αυτήν και με τη σειρά του βρίσκουμε τους δικούς του successors ώστε να συνεχίσουμε την αναζήτηση κατά βάθος. Όταν τελικά δεν υπάρχουν άλλοι successors για κάποιο state, το fringe θα έχει και πάλι πρώτο στοιχείο τον κόμβο με το μεγαλύτερο βάθος και θα συνεχίσει από εκεί.

**Ερώτημα 2 - bfs:** Ίδια μορφή για τα στοιχεία του fringe (tuples με 2 στοιχεια) διατήρησα και στο bfs.
Όσον αφορά τη λογική του αλγορίθμου, το depth first search έχει τη λογική ότι εξερευνούμε πρώτα τον κόμβο με το μικρότερο βάθος (σε αναπαράσταση δέντρου θέλουμε τον κόμβο στο μικρότερο μη εξερευνημένο επίπεδο).
Για αυτό το λόγο, το fringe είναι ένα queue (first in – first out), με το οποίο σε κάθε επανάληψη οι successors του current state μπαίνουν τελευταίοι στη λίστα, και δεν γίνονται “popped” από αυτήν μέχρι να εξερευνηθούν όλοι οι κόμβοι του επιπέδου του current state.

**Ερώτημα 3 - ucs:** Τα στοιχεία του fringe είναι τώρα λίστες 2 στοιχείων. Το 1ο στοιχείο είναι ένα tuple με state(tuple), path(list), totalcost(int) και το 2ο είναι το totalcost(int). Αυτό συμβαίνει ώστε το totalcost να γίνεται assigned εύκολα με την fringe.pop, ενώ ταυτόχρονα χρησιμοποιείται σαν priority για το priority queue που χρησιμοποιώ σε αυτό το ερώτημα (εξηγείται παρακάτω)
Όσον αφορά τη λογική του αλγορίθμου, το uniform cost search έχει τη λογική ότι εξερευνούμε πρώτα τον κόμβο με το μικρότερο συνολικό κόστος κινήσεων από τον κόμβο εκκίνηση.
Έτσι, το fringe εδώ είναι priority queue που αποθηκεύει το συνολικό κόστος κινήσεων για να φτάσουμε σε κάθε state και τα τοποθετεί στην ουρά με φθίνουσα σειρά. Επομένως το τελευταίο στοιχείο στην ουρά που θα γίνει popped θα έχει το μικρότερο συνολικό κόστος.

**Ερώτημα 4 - astar:** Τα στοιχεία του fringe είναι τώρα λίστες 2 στοιχείων. Το 1ο στοιχείο είναι ένα tuple με state(tuple), path(list), totalcost(int) και το 2ο είναι το combined totalcost + Heuristic του current state. Αυτή τη φορά για priority έχουμε το άθροισμα heuristic + totalcost.
Όσον αφορά τη λογική του αλγορίθμου, το astar search έχει τη λογική ότι εξερευνούμε πρώτα τον κόμβο με το μικρότερο συνολικό *κόστος κινήσεων από τον κόμβο εκκίνηση + την τιμή heuristic του κόμβου* που ονόμασα CnH.
Έτσι, το fringe εδώ είναι priority queue που αποθηκεύει το CnH για κάθε state και τα τοποθετεί στην ουρά με φθίνουσα σειρά. Επομένως το τελευταίο στοιχείο στην ουρά που θα γίνει popped θα έχει το μικρότερο CnH.

**Ερώτημα 5 – Finding all the corners:** Η λογική που ακολούθησα είναι η εξής:
Το state δεν είναι πλέον μόνο συντεταγμένες x,y, αλλά είναι ένα tuple με 2 στοιχεία:
$1^{ο}$ στοιχείο είναι το παλιό state tuple με τις συντεταγμένες και $2^{ο}$ στοιχείο είναι ένα tuple που κρατά τις συντεταγμένες των εξερευνηθέντων γωνιών. Με αυτόν τον τρόπο, κάθε state κρατά σαν δεδομένο και το ποιες γωνίες έχει εξερευνήσει.
Στην isGoalState ελέγχουμε αν οι γωνίες που έχει εξερευνήσει σε ένα δεδομένο state είναι 4. Αν είναι, τότε έχει εξερευνήσει όλες τις γωνίες και επιστρέφει το μονοπάτι.
Στη συνάρτηση getSuccessors, επιπλέον από την getSuccessors του PositionSearchProblem, έχω προσθέσει έναν έλεγχο με τον οποίο αν ο successor που βρίσκω έχει συντεταγμένες γωνίας που δεν έχει εξερευνήσει ως τώρα, προσθέτω στο tuple corners visited τη γωνία που εξερευνήθηκε (μετατρέπω το corners visited (state[1]) σε list μέσω cast για την προσθήκη).

**Ερώτημα 6 – CornersHeuristic:** Η λογική του heuristic είναι η εξής:
Για κάθε state, το heuristic θα ισούται με το μικρότερο άθροισμα διαδοχικών manhattan αποστάσεων από το state στην κοντινότερη μη εξερευνημένη γωνία + από αυτή τη γωνία στην κοντινότερη μη εξερευνημένη γωνία + …
Αλλιώς, κάθε state θα έχει σαν heuristic ένα άθροισμα manhattan αποστάσεων για να εξερευνήσει διαδοχικά κάθε κοντινότερη μη εξερευνημένη γωνία.
Έτσι, όσο μικρότερο είναι αυτό το άθροισμα, από την priority queue της astar θα επιλέγεται ουσιαστικά πρώτα ένα state το οποίο έχει τη μικρότερη δυνατή απόσταση στο χώρο από την εξερεύνηση όλων των γωνιών που δεν έχει εξερευνήσει ως τότε.
Σημείωση: Για την εύρεση των μη εξερευνημένων γωνιών για κάθε state (και την τοπική αφαίρεση αυτών των γωνιών από τις μη εξερευνημένες ώστε να βρεθεί το heuristic) δημιούργησα μία λίστα CornersUnvisited, που έχει (αρχικά) για στοιχεία τις μη εξερευνημένες γωνίες για το δεδομένο state.