# Τεχνητή Νοημοσύνη
# Εργασία 1
# *Pacman PDF*

Κωνσταντίνος Χαϊδεμένος
*sdi*2200262

Στις υλοποιήσεις των $DFS$, $BFS$, $UCS$, $A^*$ χρησιμοποιούμε πρώτα τον εξής κώδικα αρχικοποίησης:

```
path = []   #create a list to store the path of the nodes
state =  problem.getStartState()   #get the start state

frontier = ***   #data structure depending on which algorithm we are coding
frontier.push( (state, path) )  #initialize start state in the frontier
visited = set()    #create a set to store the visited nodes
```

Αυτό που θα αλλάξει για την υλοποίηση του κάθε αλγόριθμου θα είναι η δομή δεδομένων που θα χρησιμοποιήσουμε και ο χειρισμός του *while loop* για την επιλογή των *successor nodes*.

Μέσα στο *while loop* το πρώτο κομμάτι του κώδικα είναι επίσης κοινό σε όλες τις υλοποιήσεις των αλγορίθμων αυτών:

```
while not frontier.isEmpty():

    state, path = frontier.pop()
    #pop the top node from the frontier priority queue
    #to check if it is the goal state

    if problem.isGoalState(state):    #check if we have reached a goal state
            return path

    visited.add(state)              #if not add it to visited
```

Έπειτα από το συγκεκριμένο κομμάτι κώδικα υπάρχει ένα $for\ loop$ στο οποίο πραγματοποιείται ουσιαστικά η επιλογή των $successor\ nodes$ ανάλογα με την πολιτική του κάθε αλγορίθμου. Το συγκεκριμένο κομμάτι διαφέρει για κάθε συνάρτηση.

## $Q1$ - $DFS$

Γίνεται χρήση $Stack$ ακολουθώντας την θεωρία στις διαφάνειες του μαθήματος.

Το $for\ loop$ είναι ως εξής:

```
for s in problem.getSuccessors(state):
#check every successor of the current state

    if s[0] not in visited:
    #if the current successor is not already visited then

    frontier.push( (s[0], path + [s[1]]) )
    #add the successor and the updated path to the frontier stack
```

## $Q2$ - $BFS$

Γίνεται χρήση $Queue$ ακολουθώντας την θεωρία στις διαφάνειες του μαθήματος.

Στη συγκεκριμένη συνάρτηση ακριβώς πρίν το $for\ loop$ υπάρχει μια παραπάνω γραμμή κώδικα που θα συναντηθεί και στις επόμενες 2 συναρτήσεις. Ο ρόλος της είναι να μετατρέψει προσωρινά το $frontier$ από την δομή δεδομένων $Queue$ σε $list$ έτσι ώστε να γίνει εύκολα η προσπέλασή της για να γίνει ο έλεγχος μετά...

Το $for\ loop$ μαζί με την "έξτρα" γραμμή κώδικα είναι ως εξής:

```
frontier_list = [ t[0] for t in frontier.list ]
#turn the frontier queue into a stack for checking later

 for s in problem.getSuccessors(state):
 #check every successor of the current state

     if s[0] not in visited and s[0] not in frontier_list:
     #if the successor is not already visited
     #and if the successor is not already in the frontier

         frontier.push( (s[0], path + [s[1]]) )
         # add the successor and the updated path to the frontier queue
```

## *Q3 - UCS*

Γίνεται χρήση *PriorityQueue* ακολουθώντας την θεωρία στις διαφάνειες του μαθήματος.

Στη συγκεκριμένη συνάρτηση ρόλος της "έξτρα" γραμμής κώδικα είναι να μετατρέψει προσωρινά το *frontier* από την εκάστοτε δομή δεδομένων *PriorityQueue* σε *heap* έτσι ώστε να γίνει εύκολα η προσπέλασή των *state tuple* για να γίνει ο έλεγχος μετά...

Το *for loop* μαζί με την "έξτρα" γραμμή κώδικα είναι ως εξής:

```
frontier_heap = [ i[2][0] for i in frontier.heap ]
# frontier.heap[i][2] is the state tuple: (position, path)

 for s in problem.getSuccessors(state):

     s_path = path + [s[1]]        #successor path

     if s[0] not in visited and s[0] not in frontier_heap:
     #if the successor is not already visited
     #and if the successor is not already in the frontier
```

```
            frontier.push( (s[0], s_path), problem.getCostOfActions(s_path) )

        else:
            for i in range(0, len(frontier_heap)):
                if s[0] == frontier_heap[i]:

                    # The stored path and the new path costs have to be compared
                    updatedCost = problem.getCostOfActions(s_path)
                    storedCost = frontier.heap[i][0]
                    # frontier.heap[i] is a tuple: (cost, counter, (node, path))

                    if storedCost > updatedCost:
                    # we have to update the frontier heap
                    #with the new cost and the new path
                    #however tuples are immutable so we have to do it step by step

                        frontier.heap[i] = (storedCost, frontier.heap[i][1] ,
                        (s[0], s_path) )
                        #first update the path of the successor

                        frontier.update( (s[0], s_path), updatedCost )
                        #then we update the cost
```

## $Q4$ - $A^*$

Γίνεται χρήση *PriorityQueue* ακολουθώντας την θεωρία στις διαφάνειες του μαθήματος.

Στη συγκεκριμένη συνάρτηση είναι λίγο "πειραγμένο" όλο το *while loop* γιατί για να βάλουμε νέο κόμβο στο *visted* πρέπει να ανανεώσουμε και το κόστος του συγκεκριμένου μονοπατιού...

Το *while loop* είναι ως εξής:

```
while not frontier.isEmpty():

    state, path, cost = frontier.pop()
    #pop the top node from the frontier priority queue
    #to check if it is the goal state

    if problem.isGoalState(state):      #check if we have reached a goal state
        return path


    if (state not in visited) or (cost < visited[state]):
    #if the current node is not already visited
    #or the current cost is less than the cost of the visited state

        visited[state] = cost   #update cost

        #for every successor calculate the cost + heuristic value and
        #add it in the frontier
        for s, s_path, s_cost in problem.getSuccessors(state):

            newpath = path + [s_path]   #calculate new path
            newCost = cost + s_cost     #calculate new cost
            newNode = (s, newpath, newCost)
            #create a new node with new path and new cost

            frontier.push(newNode, newCost+heuristic(s, problem))
```

## $Q5, 6, 7$

Δεν πρόλαβα την λόγω πίεσης χρόνου να γράψω επεξήγηση στο *pdf*..

Ωστόσο σε όλες τις συναρτήσεις των ερωτημάτων έχω προσθέσει σχόλια σχεδόν για κάθε γραμμή οπότε μπορείτε να απαντήσετε τις ερωτήσεις σας από αυτά.