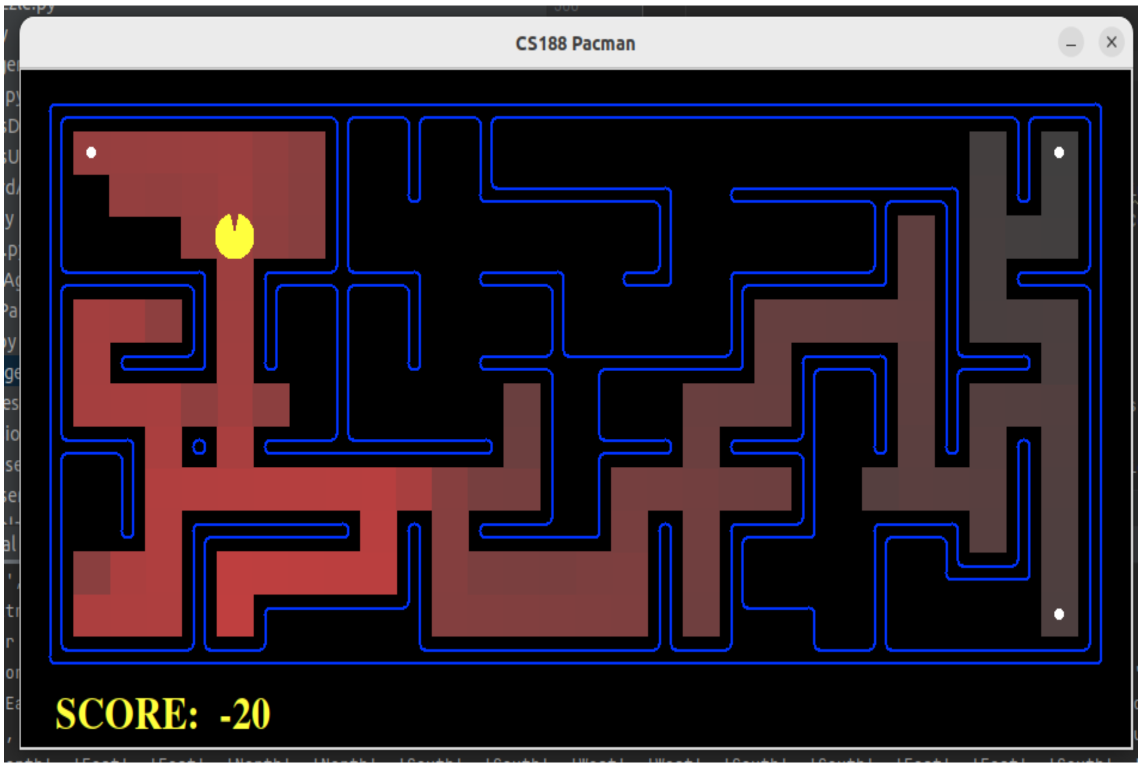
Artificial Intelligence Project 1 Search
Corner Problem heuristics, WeightedAStar variants

Andercou Alexandru

November 6, 2022



Contents

1 Introduction

Pacman is an arcade game with a long history. The main goal of the Pacman is usually to eat as much food as possible avoiding threat represented by ghosts and walls. However a sub-task that can be attributed to a Pacman is to find the special dots that transform ghosts in food, these special dots are in number of 4 and are located always in the corners. So the problem of finding the special dots turns into the https://www.overleaf.com/project/6366ae9e717c1ba80500f282Corners Problem.

2 Defining the problem and conceptual solutions

In order to solve the corners problem ,which means finding the shortest path that connects the Pacman and the 4 corners there exists many local search algorithms from the most simple like breath first search and depth first search to A*,weighted A* star ,and many other variants. As the scope of this project I choose to explore 4 variants of weighted A* with 5 heuristics starting from the simplest to more complex ones.

A* is a type of a greedy local search algorithm that uses a cost function with formula f=g+h where g represents the total cost till the current state and h represents the cost of the heuristic

Weighted A* is a simple variant of A* which complicates a bit the formula used by A*. It uses for the cost the function f=cg*g +ch*h where cg and ch are the weights attributed to the total cost and to the heuristic and it can be translated into the importance the agent offers to the total cost and to the knowledge or to his estimation, by varying the value of cg and ch we get a vast variaty of agents that act differently.If we set ch to 0 than we get a simple greedy algorithm that considers only the total cost till the current state, this variant gets outside of A* and weighted A* domains so it will not be discussed.

3 Corners Problem Heuristics

In order to get an estimation that is closest to the optimal one the heuristics must be consistent and admissible so it must not be bigger than the actual real optimal cost and it must respect the triangle law. I come up with 4 heuristics.

3.1 Heuristic 1

The most simple heuristic you can think of is the number of corners the agent managed to get to. The first heuristic is decreasing each time the Pacman manages to get in a corner.It will almost always be smaller than the total real cost, but it is a very pour estimator overall because most of

3.2 Heuristic 2

Another slightly smarter method is estimating the rest of the path by the biggest distance towards a corner using Manhattan distance that estimates better the cost of walking in a grid structure than euclidean distance. This heuristic assumes no walls.We hope that once it gets to the furthest corner it will already get trough all the other corners first.

3.3 Heuristic 3

For the 3th heuristic we make a few changes to the 2th heuristic. For starters besides the distance from Pacman to a corner we take in consideration also the distance between any two corners distance that Pacman we will anyway have to pass trough. Another important change is that we replace Manhattan distance with mazeDistance which will estimate the distance between corners using simulation with a bfs algorithm.We will consider a sum between the smallest distance between Pacman and a corner and the biggest distance between any 2 corners.

However bfs is an expansive algorithm and with not work well for a big maze,especially that we need to compute it many-many times.

3.4 Heuristic 4

As the 4th heuristic instead of considering the maximum distance between corners we will be more positive and consider the average of the distances between corners, in the rest the method stays the same.This modification brings us more to reality , there might be beteer paths than taking the longest path between corners.

3.5 Heuristic 5

For the fifth heuristic we combine the 2th , 3th and 4th. For this heuristic we will drop mazeDistance because it is too expensive and not fiting for most big problems and use a formal distance Manhattan distance. From the 3th heuristic we will keep combining the distance between corners and distance between Pacman and a corner but from the 4th heuristic we will use the average of distances between corners to not overevaluate the real distance.

4 Weighted A* variants

In weighted A* algorithm by modifying the weights associated with total cost and heursitic we get multiple possibilities. I decided to test 5 of them

4.1 Weighted A* with cg weight bigger than than ch and none 0

This method translated to a conservative Pacman but slighly open to liberalism or to free thinking , to estimations of reality.

4.2 Weighted A* with cg weight smaller than ch and none 0

This translates to a liberal who relies on estimations but still listens to the pragmatism of real cost, as it can't completly avoid reality.

4.3 A*

A* is the Variant of Weighted A* where both of weights have the same value. It will be used as a control method , to see the difference caused by algorithms. It translated to a balanced person who values both reality and estimations.

4.4 Greedy on heuristic without cost weight

If the weight of total cost is set to 0 we get an agent that considers only the heuristic , it is likely to not get great results as many positions will have the same heuristic. It represents an impulsive agent

4.5 Dynamic Weighted A*

In the previous cases we considered A* with fixed weights, if we allow change of weights as the search progresses we get a different result from any other ,likely a combination between the other behaviours.This one is more akin to a real agent as humans are able to change their method of action as they progress in life. For this heuristic we will consider the cg as 1 and ch=1+epsilon-(epsilon*depth(n))/PathEstimation Where path estimation is the lenght of path resulted from a bfs traversal. where epsilon is a random number between 0 and 1 and depth is a function that grows with each node that is taken out of the queue.

5 Implementation and results

We tested all the combinations between one given heuristic and one algorithm of search resulting in 5*4=20 different agents

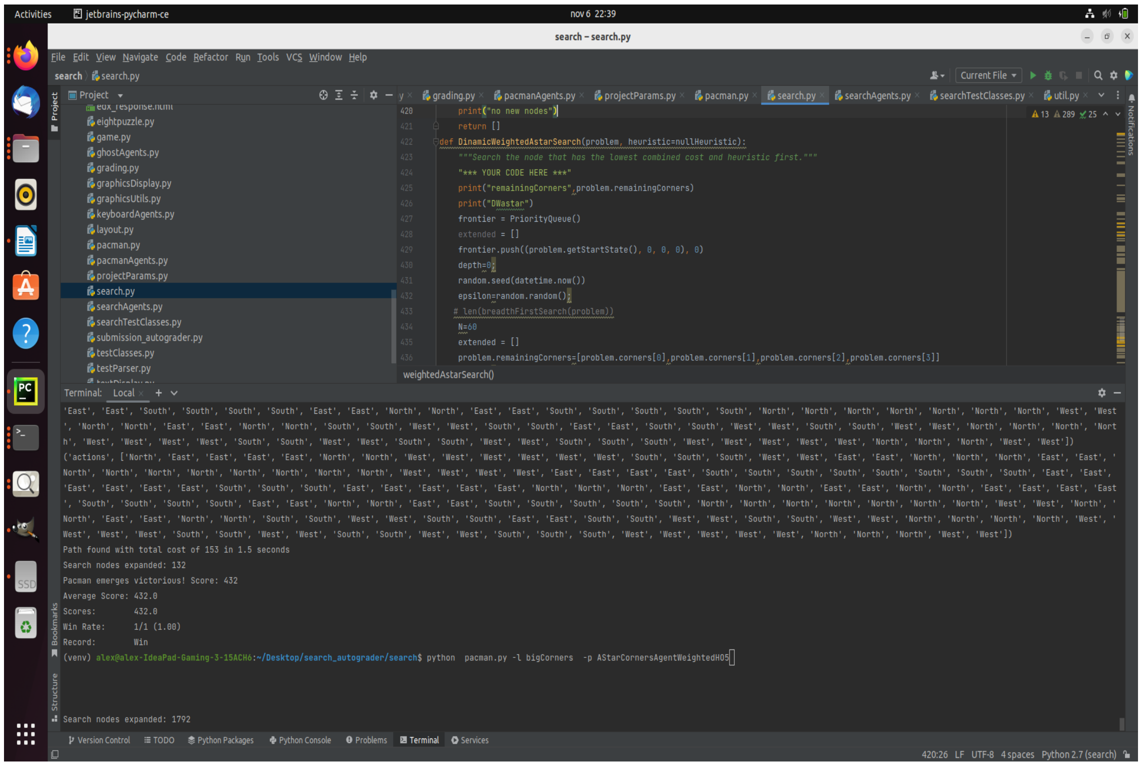
The testing was made on the grids:tinyCorners and bigCorners.

5.1 Testing

For testing we used PyCharm terminal with the command

$pythonpacman.py - llayOut_name - pAgentTypeClass$

The layOut we used where mediumCorners and bigCorners.



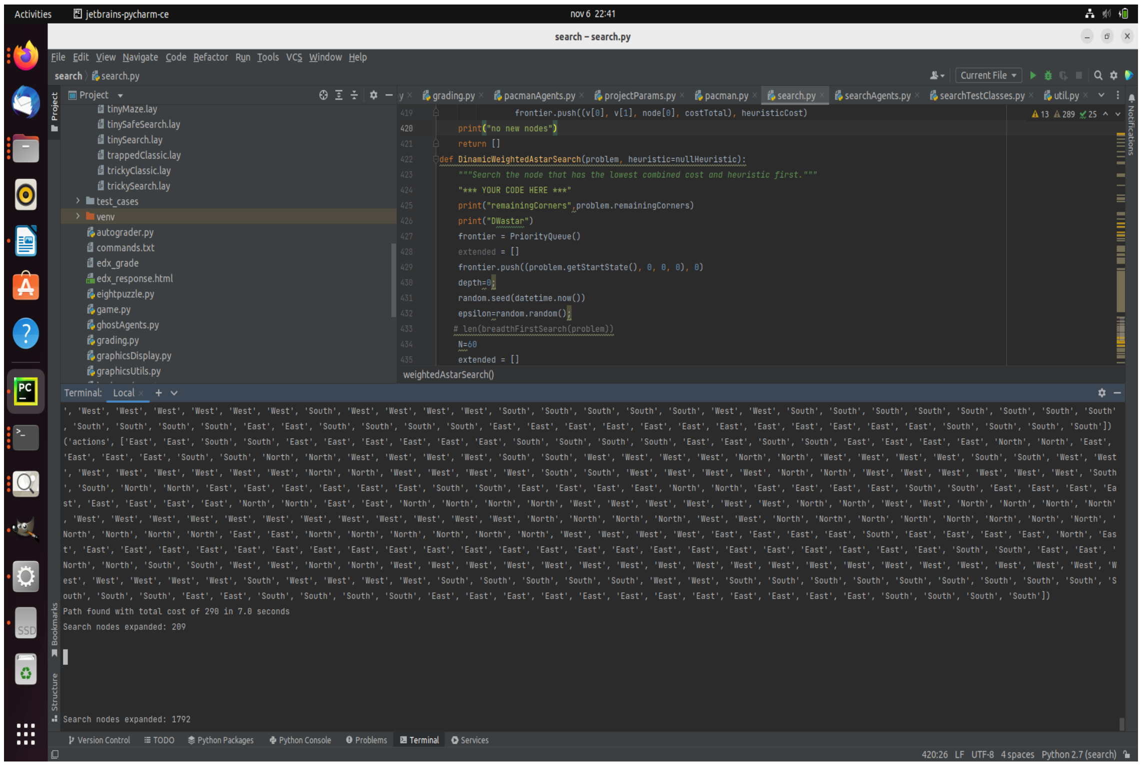
```
        print("no new nodes")
        return []
def DinamicWeightedAstarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"
    print("remainingCorners",problem.remainingCorners)
    print("DWastar")
    frontier = PriorityQueue()
    extended = []
    frontier.push((problem.getStartState(), 0, 0, 0), 0)
    depth=0;
    random.seed(datetime.now())
    epsilon=random.random();
   # len(breadthFirstSearch(problem))
    N=60
    extended = []
    problem.remainingCorners=[problem.corners[0],problem.corners[1],problem.corners[2],problem.corners[3]]
```

```
'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'North', 'North', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West
', 'North', 'North', 'East', 'East', 'North', 'North', 'South', 'South', 'West', 'West', 'South', 'South', 'East', 'East', 'South', 'South', 'West', 'West', 'South', 'South', 'West', 'West', 'North', 'North', 'North', 'Nort
h', 'West', 'West', 'West', 'West', 'South', 'South', 'West', 'West', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'West', 'West', 'West', 'West', 'West', 'North', 'North', 'North', 'West', 'West'])
('actions', ['North', 'East', 'East', 'East', 'East', 'North', 'North', 'West', 'West', 'West', 'West', 'West', 'West', 'South', 'South', 'South', 'West', 'West', 'East', 'East', 'North', 'North', 'North', 'East', 'East', '
North', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'South', 'South', 'South', 'East', 'East',
'East', 'East', 'East', 'East', 'South', 'South', 'South', 'East', 'East', 'East', 'East', 'East', 'North', 'North', 'North', 'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'East
', 'South', 'South', 'South', 'South', 'East', 'East', 'North', 'North', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'North', '
North', 'East', 'East', 'North', 'North', 'South', 'South', 'West', 'West', 'South', 'South', 'East', 'East', 'South', 'South', 'West', 'West', 'South', 'South', 'West', 'West', 'North', 'North', 'North', 'North', 'West', '
West', 'West', 'West', 'South', 'South', 'West', 'West', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'South', 'West', 'West', 'West', 'West', 'West', 'North', 'North', 'North', 'West', 'West'])
Path found with total cost of 153 in 1.5 seconds
Search nodes expanded: 132
Pacman emerges victorious! Score: 432
Average Score: 432.0
Scores:        432.0
Win Rate:      1/1 (1.00)
Record:        Win
(venv) alex@alex-IdeaPad-Gaming-3-15ACH6:~/Desktop/search_autograder/search$ python  pacman.py -l bigCorners  -p AStarCornersAgentWeightedH05



Search nodes expanded: 1792
```

5.2 Results

After Aplying the command in terminal for each eoch thec 25 combinations we made a table using the number of Node expanded and running time where was relevant. Where is put in the table medCorn means it was way to slow to be reasonably run on the bigCorners layout.

A/H	H1	H2	H3	H4	H5
WC	1062	812	365 medCorn 67.9s	232 medCorn 31.2s	497
WH	713	880	118 medCorn 44.7s	164 bigCorn 38.3s	209
A*	1062	834	171 medCorn 95.5s	202 bigCorn 41.6s	363
GH	782	1193 29.9s	126 medCorn 45.9s	173 bigCorn 77.1s	192 23.8s
DW	731	704	133 49.6s	252 big Corn 44.9s	751 29.6s



```
                frontier.push((v[0], v[1], node[0], costTotal), heuristicCost)
    print("no new nodes")
    return []
def DinamicWeightedAstarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"
    print("remainingCorners",problem.remainingCorners)
    print("DWastar")
    frontier = PriorityQueue()
    extended = []
    frontier.push((problem.getStartState(), 0, 0, 0), 0)
    depth=0;
    random.seed(datetime.now())
    epsilon=random.random();
   # len(breadthFirstSearch(problem))
    N=60
    extended = []
```

```
', 'West', 'West', 'West', 'West', 'West', 'West', 'South', 'West', 'West', 'West', 'West', 'South', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'South', 'South', 'South', 'South', 'South'
, 'South', 'South', 'South', 'South', 'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South'])
('actions', ['East', 'East', 'South', 'South', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'South', 'South', 'East', 'East', 'East', 'East', 'North', 'North', 'East',
'East', 'East', 'East', 'South', 'South', 'North', 'North', 'West', 'West', 'West', 'West', 'South', 'South', 'West', 'West', 'West', 'West', 'North', 'North', 'West', 'West', 'West', 'West', 'South', 'South', 'West', 'West
', 'West', 'West', 'West', 'West', 'West', 'West', 'North', 'North', 'West', 'West', 'West', 'West', 'South', 'South', 'West', 'West', 'West', 'West', 'North', 'North', 'West', 'West', 'West', 'West', 'West', 'West', 'South
', 'South', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'East', 'East', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'East', 'South', 'South', 'East', 'East', 'East', 'Ea
st', 'East', 'East', 'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West', 'West', 'West', 'North', 'North', 'West', 'West', 'North', 'North', 'North', 'North'
, 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'North', 'North', 'North', 'North', 'West', 'West', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'North', '
North', 'North', 'North', 'North', 'East', 'East', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West', 'East', 'East', 'East', 'East', 'South', 'East', 'East', 'East', 'East', 'North', 'Eas
t', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'East', 'East', '
North', 'North', 'South', 'South', 'West', 'West', 'North', 'North', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'W
est', 'West', 'West', 'West', 'West', 'South', 'West', 'West', 'West', 'West', 'South', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'South', 'South', 'South', 'South', 'South', 'South', 'S
outh', 'South', 'South', 'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South'])
Path found with total cost of 290 in 7.0 seconds
Search nodes expanded: 209


Search nodes expanded: 1792
```

SO after the experimentations, if we round the numbers we get that the combination that produced the best results were with GH and WH variants which put a lot of importance on the heuristic but GH proved to be the worst with the 2th heuristic. The 3th and 4th heuristics can be resonably used only on small problems in this case a medium problem , for H3 and H4 the one that performed the best was again WH.

6 Appendices

6.1 A Appendix

6.1.1 Original code

```
def weightedAstarSearch(problem, heuristic=nullHeuristic, weightE=2, weightC=1):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"

    print("Wastar")
    frontier = PriorityQueue()
    extended = []
    actions = Queue()
    frontier.push((problem.getStartState(), 0, 0, 0), 0)
    while not frontier.isEmpty():
        node = frontier.pop()
        extended.append(node)
        print("number of corners not riched",len(problem.remainingCorners))
        if problem.isGoalState(node[0]):
            print("goal riched")
            return build_path(extended,problem.visitedCorners)
        vecini = problem.getSuccessors(node[0])
        print("vecini",vecini)
        for v in vecini:
               bool = False
               for e in extended:
                  if e[0] == v[0]:
                       bool = True
               if bool == False:
                print("can add mmore")
                costTotal = node[3] + v[2]
                heuristicCost = weightE * heuristic(v[0], problem) + weightC * costTotal
                frontier.push((v[0], v[1], node[0], costTotal), heuristicCost)
    print("no new nodes")
    return []
def DinamicWeightedAstarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"
    print("remainingCorners",problem.remainingCorners)
    print("DWastar")
    frontier = PriorityQueue()
    extended = []
    frontier.push((problem.getStartState(), 0, 0, 0), 0)
    depth=0;
    random.seed(datetime.now())
    epsilon=random.random();
   # len(breadthFirstSearch(problem))
    N=60
    extended = []
    problem.remainingCorners=[problem.corners[0],problem.corners[1],problem.corners[2],problem.corner
    pause();
    while not frontier.isEmpty():
        depth=depth+1;
        weightE=1+epsilon-epsilon*depth/N;
        print("weightE",weightE);
        node = frontier.pop()
        extended.append(node)
        if problem.isGoalState(node[0]):
          print("got to goal")
          print("extended",extended)
          pause();
          return build_path(extended,problem.visitedCorners,problem.getStartState())

        vecini = problem.getSuccessors(node[0])
        for v in vecini:
            bool = False
            for e in extended:
                if e[0] == v[0]:
                    bool = True
            if bool == False:
                costTotal = node[3] + v[2]
                heuristicCost = weightE * heuristic(v[0], problem) + costTotal
                frontier.push((v[0], v[1], node[0], costTotal), heuristicCost)



def build_path(extended,visitedCorners,start_point):
    paths = []
    actions = Queue()
```

```
parent=(0,0,0,0)
for vc in visitedCorners:
    path = Stack()

    for n in extended:
     if(n[0]==vc):
       print("n0",n)
       path.push(n)
       parent = n[0]
       break


    print("extended",extended)

    while not parent == 0:
        print("parent",parent)
        for n in extended:
            if n[0] == parent:
                path.push(n)
                parent = n[2]
                break

    paths.append(path.list[1:])
print("visited corners in order are:", visitedCorners)
print("paths are:")
for path1 in paths:
    print("path:", path1)
noduri_legatura = [0,0,0,0,0,0,0]
search_intersections=[1,2,3,4,5,6]
for i in paths[0]:
    for j in paths[1]:
        for k in paths[2]:
            for l in paths[3]:
                for m in range(len(search_intersections)):
                     if i[0] == j[0] and search_intersections[m]==1:
                         search_intersections[m]=0
                         noduri_legatura[1]=i

                     if i[0] == k[0] and search_intersections[m] == 4:
                         search_intersections[m] = 0
                         noduri_legatura[4]=j
                     if i[0] == l[0] and search_intersections[m] == 5:
                         search_intersections[m] = 0
                         noduri_legatura[5]=l
                     if j[0] == k[0] and search_intersections[m] == 2:
                         search_intersections[m] = 0
                         noduri_legatura[2]=k
                     if j[0] == l[0] and search_intersections[m] == 6:
                         search_intersections[m] = 0
                         noduri_legatura[6]=j
                     if l[0] == k[0] and search_intersections[m] == 3:
                         search_intersections[m] = 0
                         noduri_legatura[3]=l
print("noduri de legatura in arbore sunt:", noduri_legatura)
destination = start_point
for i in range(0,4):
```

```
print("i=",i)
current_possition = paths[i][0]

#reverse a path:

help_stack =Stack()
help_stack2=Queue()
print("starting point", current_possition )
for n in paths[i]:
     print("destination to get to",destination)
     if n[0]==destination:

         print("destination",destination)
         help_stack=Stack()
         nod_to_go_back_to=0
         current_possiton=paths[i][0]
         if i==3:
           nod_to_go_back_to = noduri_legatura[i -1][0]
         else:
           nod_to_go_back_to = noduri_legatura[i +1 ][0]
         print("go  back to",nod_to_go_back_to)
         print("current", current_possition )
         destination= nod_to_go_back_to
    #      #you reached a point from where you will jump to an intersection so go back to that po
         while  not  (current_possition[0]==nod_to_go_back_to )and  not current_possition[2]==0:


            print("backtracking",current_possition)
           #reverse the action made to get to a corner
            if (current_possition[1] == 'North'):
               help_stack.push("South")
            if (current_possition[1]  == 'South'):
               help_stack.push("North")
            if (current_possition[1] == 'West'):
               help_stack.push("East")
            if (current_possition[1] == 'East'):
               help_stack.push('West')
           #go back to your parent
            for j in extended:
               if j[0]==current_possition[2]:
                   current_possition=j
                   break
         print(" finished backtracking")


         break
     else:

          if n[1]!=0:
            print("n not path",n)
            help_stack2.push(n[1])


actions.list=actions.list+help_stack2.list+help_stack.list
print("actions after a path",actions.list)
```

```
    print("actions",actions.list)
    return actions.list
```

```
def CostOrientedWeightAStar(problem, heuristic=nullHeuristic, weightE=2, weightC=1):
     return weightedAstarSearch(problem, heuristic, 1, 2)


def HeuristicOrientedWeightAStar(problem, heuristic=nullHeuristic, weightE=2, weightC=1):
    return weightedAstarSearch(problem, heuristic, 2, 1)
    return weightedAstarSearch(problem, heuristic, 2, 1)

def greedyWeightAStar(problem, heuristic=nullHeuristic, weightE=2, weightC=1):
    return weightedAstarSearch(problem, heuristic ,1,0)
```

```
 def isGoalState(self, state):
        x,y =state
        for i  in range(len(self.remainingCorners)):
           x1,y1=self.remainingCorners[i]
           if(x==x1 and y==y1):
                self.visitedCorners.append(self.remainingCorners[i])
                self.remainingCorners.remove(self.remainingCorners[i])
                i=i-1
                break
        return len(self.remainingCorners) == 0
```

```
def cornersHeuristic1(state, problem):
    corners = problem.remainingCorners
    return len(corners)

def cornersHeuristic2(state, problem):
    # These are the corner coordinates
    x,y= state
    corners=problem.remainingCorners
    furthestcornerIndex=0
    biggestDistance= -1
    for index in xrange(0,len(corners)):
       xg,yg= corners[index]
       dist=abs(x-xg)+abs(x-xg)
```

```
            if dist>biggestDistance:
                biggestDistance =dist
                furthestcornerIndex=index
        xg, yg = corners[furthestcornerIndex]

        return abs(x-xg)*abs(y-yg)




def cornersHeuristic3(state, problem):
    corners = problem.remainingCorners
    # These are the corner coordinates
    x,y= state
    bigestDistanceBetweenCorners=0;
    closestCorner=999999;
    print("remaining corners",len(corners))
    for i in range(len(corners)):
         xc, yc = corners[i]
         dist_corners=mazeDistance((x,y),(xc,yc),problem.startingGameState)

         if(dist_corners<closestCorner):
             closestCorner=dist_corners

         for j in (i+1,len(corners)-1):
            if j<len(corners):
              xc2,yc2=corners[j]
              dist=mazeDistance((xc,yc),(xc2,yc2),problem.startingGameState)
              if dist >  bigestDistanceBetweenCorners:
                 bigestDistanceBetweenCorners=dist

    return closestCorner+bigestDistanceBetweenCorners


def cornersHeuristic4(state, problem):
    corners = problem.remainingCorners
    # These are the corner coordinates
    x,y= state
    averageDistanceBetweenCorners=0;
    closestCorner=999999;

    for i in range(len(corners)):
         xc, yc = corners[i]
         dist_corners=mazeDistance((x,y),(xc,yc),problem.startingGameState)
         if(dist_corners<closestCorner):
             closestCorner=dist_corners

         for j in(i+1,len(corners)):
          if j < len(corners):
            xc2,yc2=corners[j]
            dist=mazeDistance((x,y),(xc,yc),problem.startingGameState)
            averageDistanceBetweenCorners +=dist

    return closestCorner+averageDistanceBetweenCorners/len(corners)
```

```
def cornersHeuristic5(state, problem):
    corners = problem.remainingCorners
    # These are the corner coordinates
    x,y= state
    averageDistanceBetweenCorners=0;
    closestCorner=999999;

    for i in range(len(corners)):
         xc, yc = corners[i]
         #=abs(x-xc)+abs(y-yc)
         dist_corners=abs(x-xc)+abs(y-yc)
         if(dist_corners<closestCorner):
             closestCorner=dist_corners

         for j in(i+1,len(corners)):
          if j < len(corners):
            xc2,yc2=corners[j]
            dist=abs(xc-xc2)+abs(yc-yc2)
            averageDistanceBetweenCorners +=dist

    return closestCorner+averageDistanceBetweenCorners/len(corners)






class AStarCornersAgent(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.aStarSearch(prob, cornersHeuristic1)
        self.searchType = CornersProblem

class AStarCornersAgentWeightedCO(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.CostOrientedWeightAStar(prob,cornersHeuristic1)
        self.searchType = CornersProblem

class AStarCornersAgentWeightedHO(SearchAgent):
     def __init__(self):
        self.searchFunction = lambda prob: search.HeuristicOrientedWeightAStar(prob,cornersHeuristic1
        self.searchType = CornersProblem


class GreedyCornersAgentHeuristic(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.greedyWeightAStar(prob, cornersHeuristic1)
        self.searchType = CornersProblem


class DinamicCornersAgentHeuristic(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.DinamicWeightedAstarSearch(prob, cornersHeuristic1)
        self.searchType = CornersProblem





class AStarCornersAgenth2(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.aStarSearch(prob, cornersHeuristic2)
        self.searchType = CornersProblem

class AStarCornersAgentWeightedCO2(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.CostOrientedWeightAStar(prob,cornersHeuristic2)
        self.searchType = CornersProblem


class AStarCornersAgentWeightedHO2(SearchAgent):
     def __init__(self):
        self.searchFunction = lambda prob: search.HeuristicOrientedWeightAStar(prob,cornersHeuristic2
        self.searchType = CornersProblem


class GreedyCornersAgentHeuristic2(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.greedyWeightAStar(prob, cornersHeuristic2)
        self.searchType = CornersProblem

class DinamicCornersAgentHeuristic2(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.DinamicWeightedAstarSearch(prob, cornersHeuristic2)
        self.searchType = CornersProblem


class AStarCornersAgenth3(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.aStarSearch(prob, cornersHeuristic3)
        self.searchType = CornersProblem

class AStarCornersAgentWeightedCO3(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.CostOrientedWeightAStar(prob,cornersHeuristic3)
        self.searchType = CornersProblem


class AStarCornersAgentWeightedHO3(SearchAgent):
     def __init__(self):
        self.searchFunction = lambda prob: search.HeuristicOrientedWeightAStar(prob,cornersHeuristic3
        self.searchType = CornersProblem
```

```
class GreedyCornersAgentHeuristic3(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.greedyWeightAStar(prob, cornersHeuristic3)
        self.searchType = CornersProblem

class DinamicCornersAgentHeuristic3(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.DinamicWeightedAstarSearch(prob, cornersHeuristic3)
        self.searchType = CornersProblem


class AStarCornersAgenth4(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.aStarSearch(prob, cornersHeuristic4)
        self.searchType = CornersProblem


class AStarCornersAgentWeightedCO4(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.CostOrientedWeightAStar(prob,cornersHeuristic4)
        self.searchType = CornersProblem


class AStarCornersAgentWeightedHO4(SearchAgent):
     def __init__(self):
        self.searchFunction = lambda prob: search.HeuristicOrientedWeightAStar(prob,cornersHeuristic4
        self.searchType = CornersProblem


class GreedyCornersAgentHeuristic4(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.greedyWeightAStar(prob, cornersHeuristic4)
        self.searchType = CornersProblem

class DinamicCornersAgentHeuristic4(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.DinamicWeightedAstarSearch(prob, cornersHeuristic4)
        self.searchType = CornersProblem

class AStarCornersAgenth5(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.aStarSearch(prob, cornersHeuristic5)
        self.searchType = CornersProblem


class AStarCornersAgentWeightedCO5(SearchAgent):

    def __init__(self):
        self.searchFunction = lambda prob: search.CostOrientedWeightAStar(prob, cornersHeuristic5)
        self.searchType = CornersProblem


class AStarCornersAgentWeightedHO5(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.HeuristicOrientedWeightAStar(prob, cornersHeuristic
        self.searchType = CornersProblem


class GreedyCornersAgentHeuristic5(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.greedyWeightAStar(prob, cornersHeuristic5)
        self.searchType = CornersProblem


class DinamicCornersAgentHeuristic5(SearchAgent):
    def __init__(self):
        self.searchFunction = lambda prob: search.DinamicWeightedAstarSearch(prob, cornersHeuristic5)
        self.searchType = CornersProblem
```