# Lab 3: Pacman agent

**00757146 許詠晴**

## 實驗內容描述:

修改 search.zip 內的兩份程式:search.py 和 searchAgent.py 來達成 8 個問題之要求。

Q1: Finding a Fixed Food Dot using Depth First Search

修改 search.py 中之 depthFirstSearch()，利用 util.py 中之 Stack 類別之資料結構來達到 深度優先之搜尋演算法，code 如下:

```
    "*** YOUR CODE HERE ***"
    stack = util.Stack()
    start = problem.getStartState()
    visited = [] # list of visited states
    stack.push((start, []))

    # while loop
    while not stack.isEmpty():
        curState, actionPath = stack.pop() # choosing a node
        if curState in visited:
            continue
        # if the node is goal state, then return
        if problem.isGoalState(curState):
            return actionPath
        # else, expand the node
        visited.append(curState)
        for nextState, action, cost in problem.getSuccessors(curState):
            if nextState not in visited:
                stack.push((nextState, actionPath+[action]))

    return [] # goal isn' t found
```

Q2: Breadth First Search

其方法與 Q1 差不多，只是改成廣度優先之搜尋演算法(BFS)，資料結構 Stack 改為 Queue

Q3: Varying the Cost Function

運用 uniform-cost graph search algorithm，資料結構改為 priority queue，code 如下:

```
    "*** YOUR CODE HERE ***"
    p_queue = util.PriorityQueue()
    start = problem.getStartState()
    visited = [] # list of visited states
    p_queue.push((start,[], 0), 0)  # the cost of build root is 0

    # while loop
    while not p_queue.isEmpty():
        curState, actionPath, costSum = p_queue.pop() # choosing a node
        if curState in visited:
            continue
        # if the node is goal state, then return
        if problem.isGoalState(curState):
            return actionPath
        # else, expand the node
        visited.append(curState)
        for nextState, action, cost in problem.getSuccessors(curState):
            if nextState not in visited:
                p_queue.push((nextState, actionPath+[action], costSum+cost), costSum+cost)

    return [] # goal isn' t found
```

若 problem.getSuccessors()之每個 item 回傳之 build cost 皆設為 1，且 root 的 build cost 為 0，則每一 node 之 total cost 即為樹的高度(root 高度為 0)

Q4: A* search

運用 A* Algo，code 如下:

```python
    """ YOUR CODE HERE """
    p_queue = util.PriorityQueue()
    start = problem.getStartState()
    visited = [] # list of visited states
    backward = 0
    forward = heuristic(start, problem)
    totalCost = backward+forward
    p_queue.push((start,[], backward), totalCost)

    # while loop
    while not p_queue.isEmpty():
        curState, actionPath, backward = p_queue.pop() # choosing a node
        if curState in visited:
            continue
        # if the node is goal state, then return
        if problem.isGoalState(curState):
            return actionPath
        # else, expand the node
        visited.append(curState)
        for nextState, action, cost in problem.getSuccessors(curState):
            if nextState not in visited:
                forward = heuristic(nextState,problem)
                totalCost = forward+(cost+backward)
                # actual cost = cost+backward
                p_queue.push((nextState, actionPath+[action], backward+cost), totalCost)

    return [] # goal isn' t found
```

actual cost(backward +step cost) + heuristc cost(forward) = Total cost 為 priority queue 之插入 weight，因為要使 total cost 小者優先展開。

Q5: Finding All the Corners

上述四題中之 problem 的 packman 所要吃的食物都在地圖的左下角。而此題要寫另外一個 Problem: 食物有 4 個，各自分布在地圖的四角。因此，需要修改 searchAgents.py 中之 Class CornersProblem 的 funciton，code 如下:

```python
def getStartState(self):
    """
    Returns the start state (in your state space, not the full Pacman state
    space)
    """
    "*** YOUR CODE HERE ***"
    return (self.startingPosition, set())
    #util.raiseNotDefined()

def isGoalState(self, state):
    """
    Returns whether this search state is a goal state of the problem.
    """
    "*** YOUR CODE HERE ***"
    # state[0]:position; state[1]:visted corner
    if state[0] in state[1]:
        if len(state[1]) == 4:
            return True
    return False
    #util.raiseNotDefined()
```

getStartState() returns a tuple(pacman 起始位置, set(已經被吃掉的食物位置))

isGoalState()returns True, if pacman 已經達成目標(吃到 4 個角落的食物)

getSuccessors() returns 可以展開的 node，也就是說 pacman 可以往哪個方向走(上下左右不撞牆)

```python
    def getSuccessors(self, state):
        """
        Returns successor states, the actions they require, and a cost of 1.

         As noted in search.py:
            For a given state, this should return a list of triples, (successor,
            action, stepCost), where 'successor' is a successor to the current
            state, 'action' is the action required to get there, and 'stepCost'
            is the incremental cost of expanding to that successor
        """

        successors = []
        for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
            # Add a successor state to the successor list if the action is legal
            # Here's a code snippet for figuring out whether a new position hits a wall:
            #   x,y = currentPosition
            #   dx, dy = Actions.directionToVector(action)
            #   nextx, nexty = int(x + dx), int(y + dy)
            #   hitsWall = self.walls[nextx][nexty]

            "*** YOUR CODE HERE ***"
            curPosition, visitedCorner = state
            x, y = curPosition
            dx, dy = Actions.directionToVector(action)
            nextx, nexty = int(x+dx), int(y+dy)
            hitsWall = self.walls[nextx][nexty]

            if not hitsWall:
                nextPosition = (nextx, nexty)
                nextVisitedCorners = set(visitedCorner)
                if (nextPosition in self.corners) and (nextPosition not in visitedCorner):
                    nextVisitedCorners.add(nextPosition)

                nextState = (nextPosition, nextVisitedCorners)
                successors.append((nextState,action,1))

        self._expanded += 1 # DO NOT CHANGE
        return successors
```

Q6: Corners Problem: Heuristic

一樣跟 Q5 是同個 problem(食物在 4 角)，但是要用 Heuristic 來估計，因為 heuristic 要滿足 admissibilty 和 consistent 所以設計的 heuristic 如下:

```python
    "*** YOUR CODE HERE ***"

    position, visitedCorners = state
    notVisitedCorners = set(corners) - visitedCorners  # different set
    heuristic = 999999
    """
    In order to keep admissible: the heuristic values must be lower bounds on the actual
    shortest path cost to the nearest goal.
    """
    for corner in notVisitedCorners:
        # manhattanDistance <= actual shortest path cost to the nearest goal
        totalDist = util.manhattanDistance(position, corner)
        notvisitedStep2 = set(notVisitedCorners)
        notvisitedStep2.discard(corner)
        corner1 = corner
        # 再找到一個corner點之後的剩餘尚未拜訪點之最小路徑總和
        while(len(notvisitedStep2)):
            closestCorner = None
            minDistance = 999999
            for corner2 in notvisitedStep2:
                distance = util.manhattanDistance(corner1, corner2)
                if distance < minDistance:
                    closestCorner = corner2
                    minDistance = distance
            corner1 = closestCorner
            notvisitedStep2.discard(closestCorner)
            totalDist += minDistance
        heuristic = min(heuristic, totalDist)
    if len(notVisitedCorners) > 0:
        return heuristic
    return 0  # Default to trivial solution
```

Q7: Eating All The Dots

此題之 problem 也是一個新的: 會有很多食物(散佈在地圖上)要吃。但因為 searchAgent.py 之 calss FoodSearchProblem 本來已經是完整的，所以只需要針對此 problem 設計 heuristic，如下:

```python
def foodHeuristic(state, problem):
    """*** YOUR CODE HERE ***"
    """
    最遠的food與現在位置的距離
    """

    foodList = foodGrid.asList()
    if(len(foodList) == 0):
        return 0  # goal state

    maxDistance = 0
    for foodPosition in foodList:
        distance = mazeDistance(position,foodPosition,problem.startingGameState)
        if distance > maxDistance:
            maxDistance = distance

    return maxDistance
```

Q8: Suboptimal Search

greedily eats the closest food，直接用 bfs 即可

```python
    def findPathToClosestDot(self, gameState):
        """
        Returns a path (a list of actions) to the closest dot, starting from
        gameState.
        """
        # Here are some useful elements of the startState
        startPosition = gameState.getPacmanPosition()
        food = gameState.getFood()
        walls = gameState.getWalls()
        problem = AnyFoodSearchProblem(gameState)

        "*** YOUR CODE HERE ***"
        # using BFS to find a path to the closest food
        return search.breadthFirstSearch(problem)
        util.raiseNotDefined()
```

## 實驗結果:

利用 autograder.py 評估程式效能

```
D:\homework\Artificial_Intelligence\search>autograder.py
D:\homework\Artificial_Intelligence\search\autograder.py:17: Deprecatio
  import imp
Starting on 6-5 at 16:38:01
```

```
Finished at 16:38:22

Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 3/3
Question q6: 3/3
Question q7: 5/4
Question q8: 3/3
------------------
Total: 26/25
```

## 結果討論與心得:

剛開始把 search.zip unzip 出來超級多份.py，瞬間不想做作業了。可是幸好這份作業只要做 serach 的部分，所以我就照者 [http://ai.berkeley.edu/search.html](http://ai.berkeley.edu/search.html) 上面的引導開始看。Q1~Q4 沒有很困難，當 Q1 寫出來，Q2-Q4 就都差不多，只是開始我還是看了很久，那個網站上面說要用 stack 寫，可是我不知道 stack 這個資料結構是從哪裡來的，不過幸好我看到了老師投影片上的一句話

> ➢ Util.py
> - Useful data structures for implementing search algorithms

不然可能要卡更久。

接者遇到的問題是 CornerProblem 的 Heuristic 到底怎麼設計，開始我寫的那個拿去 autograder 評估都會 Fail 因為會 inadmissibility

```
# this heuristic may cause inadmissible
position, visitedCorners = state
notVisitedCorners = set(corners) - visitedCorners
heuristic = 0
while len(notVisitedCorners) > 0:
    minDistance = 9999999
    for corner in notVisitedCorners:
        distance = util.manhattanDistance(position,corner)
        if minDistance > distance:
            minDistance = distance
            minCorner = corner
    heuristic += minDistance
    position = corner
    notVisitedCorners.remove(corner)

return abs(heuristic)
```

後來仔細看說明 admissibilty 的說明，才想說那我在多加一層來達到 To be *admissible*, the heuristic values must be lower bounds on the actual shortest path cost to the nearest goal.