Q1 : Depth First Search

1)ابتدا باید حالت شروع را ایجاد کنیم

```
initial_state = problem.getStartState()
```

2)برسی کنیم ایا حالت شروع حالت نهایی است؟ بنابراین ما باید لیست actionهای خالی را برگردانیم

```
if problem.isGoalState(initial_state):
    return []
```

3) برای قسمت بعدی ، یک پشته و یک لیست خالی برای گره های بازدید شده ایجاد می کنیم.

```
myStack = util.Stack()
visitedNodes = []
```

و سپس pushانجام میدهیم.

```
myStack.push((initial_state, [])
```

(در پشته قرار میدهیم)

4)یک عنصر از پشته pop میکنیم.

```
currentNode, actions, prevCost = pQueue.pop()
```

5)بررسی کنیم
که اگرعنصر در VisitNodes نبوده آن را در انتهایVisitNodes
وارد کنیم

```
if currentNode not in visitedNodes:
    visitedNodes.append(currentNode)
```

6) اگر شرط قبلی درست بود ، بررسی کنیم که آیا عنصر حالت نهایی
است یا خیر ، لیست actionها را برگردانیم.

```
if problem.isGoalState(currentNode):
    return actions
```

7) اگر شرط 5درست بود و اگر عنصرpop حالت هدف نبود ، مطابق
با متد getuccessors ، باید گره بعدی و action بعدی را پیدا کنیم
سپس آنها راpush کنیم در mystack

```
for nextNode, action, cost in problem.getSuccessors(currentNode):
    newAction = actions + [action]
    myStack.push((nextNode, newAction))
```

8) مراحل بالا را تکرار کنید تا زمانی که mystack هنوز خالی نباشد.

```
while not myStack.isEmpty():
```

کد کامل:

```
    initial_state = problem.getStartState()
    if problem.isGoalState(initial_state):
        return []

    myStack = util.Stack()
    visitedNodes = []

    myStack.push((initial_state, []))

    while not myStack.isEmpty():
        currentNode, actions = myStack.pop()
        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                myStack.push((nextNode, newAction))

    util.raiseNotDefined()
```

Q2:Breadth First Search

الگوریتم از ریشه شروع می‌کند و آن را در سطح یک قرار می‌دهد. سپس در هر مرحله همهٔ همسایه‌های رئوس آخرین سطح دیده شده را که تا به حال دیده نشده‌اند بازدید می‌کند و آنها را در سطح بعدی می‌گذارد. این فرایند زمانی متوقف می‌شود که همهٔ همسایه‌های رئوس آخرین سطح قبلاً دیده شده باشند .

پیاده‌سازی این الگوریتم مشابه پیاده‌سازی جستجوی عمق اول است با این تفاوت که به جای پشته از صف استفاده می‌شود.

```
myStack = util.Queue()
```

کد کامل:

```
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""
    "*** YOUR CODE HERE ***"

    initial_state = problem.getStartState()
    if problem.isGoalState(initial_state):
        return []

    myStack = util.Queue()
    visitedNodes = []
    # (node,actions)
    myStack.push((initial_state, []))

    while not myStack.isEmpty():
        currentNode, actions = myStack.pop()
        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                myStack.push((nextNode, newAction))
    util.raiseNotDefined()
```

Q3: Uniform Cost Search

برای اجرای def uniformCostSearchما باید کم هزینه ترین گره را جستجو کنیم

الگوریتم جستجو باید لیستی ازactionهایی را که به هدف رسیده است بازگرداند.

راه حل:

1) اول از همه ما برای شروع مشکل باید حالت شروع ایجاد کنیم

```
initial_state = problem.getStartState()
```

2) بررسی کنید آیا حالت شروع حالت هدف است یا خیر ، باید لیست عملکرد خالی را برگردانیم

```
if problem.isGoalState(initial_state):
    return []
```

3) برای قسمت بعدی یک صف اولویت ویک لیست خالی
برای گره های بازدید شده ایجاد می کنیم

```
pQueue = util.PriorityQueue()

visitedNodes = []
```

برای push, startNode به عنوان مختصات / گره ، لیست خالی (که نشان دهنده عملکرد به گره فعلی است) ، هزینه
به گره فعلی و اولویت در صف اولویت تولید شده

```
pQueue.push((initial_state, [], 0), 0)
```

4) قسمت اصلی را در pQueue خود پیاده کنیم و راه حل را پیدا کنیم:
1) یک عنصر از pQueue راpop کنید.

```
currentNode, actions, prevCost = pQueue.pop()
```

2) بررسی کنید که آیا عنصر pop در گره های دیده شده نبوده سپس آن را در انتهایVisitNodes وارد کنید

```
if currentNode not in visitedNodes:
    visitedNodes.append(currentNode)
```

3) اگر شرط قبلی درست بود ، بررسی کنید آیا عنصر حالت هدف است یا خیر ، لیست اقدامات را برگردانید

```
if problem.isGoalState(currentNode):
    return actions
```

4) اگر شرط قبلی2 درست بود و اگر عنصر pop حالت هدف نبود ، مطابق با متد getuccessors ، باید گره بعدی ، action بعدی و اولویت بعدی را پیدا کنیم سپس pushکنیم آنها را در pQueue

```
for nextNode, action, cost in problem.getSuccessors(currentNode):
    newAction = actions + [action]
    priority = prevCost + cost
    pQueue.push((nextNode, newAction, priority),priority)
```

5) مراحل بالا را تکرار کنید تا زمانی که pQueue هنوز خالی نباشد.

```
while not myStack.isEmpty():
```

کدکامل:

```
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    "*** YOUR CODE HERE ***"
    initial_state = problem.getStartState()
    if problem.isGoalState(initial_state):
        return []

    visitedNodes = []

    pQueue = util.PriorityQueue()
    #((coordinate/node , action to current node , cost to current node),priority)
    pQueue.push((initial_state, [], 0), 0)

    while not pQueue.isEmpty():

        currentNode, actions, prevCost = pQueue.pop()
        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                priority = prevCost + cost
                pQueue.push((nextNode, newAction, priority),priority)
    util.raiseNotDefined()
```

Q4: A* Search

برای اجرای

def astarsearch(problem, heuristic=nullheuristic):

ما باید گره را جستجو کنیم

که کمترین هزینه و heuristicرا دارد.

الگوریتم جستجو باید لیستی ازactionها که به هدف می رسد.

راه حل این قسمت دقیقاً همان راه حل قبلی است اما در مرحله 4) در قسمت 4 ما باید تغییراتی را ایجاد کنیم:

مرحله 4)

4) اگر شرط 2 قبلی درست بود و اگر عنصر pop حالت هدف نبود ، طبق روش getuccessors ، ما باید گره بعدی ، action جدید ، هزینه بعدی گره را پیدا کنیم و تعیین کنیم

هزینهheuristic سپس آنها را در pQueue, pushکنیم

```
        for nextNode, action, cost in problem.getSuccessors(currentNode):
            newAction = actions + [action]
            newCostToNode = prevCost + cost
            heuristicCost = newCostToNode + heuristic(nextNode,problem)
            pQueue.push((nextNode, newAction, newCostToNode),heuristicCost)
util.raiseNotDefined()
```

کدکامل:

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"
    initial_state = problem.getStartState()
    if problem.isGoalState(initial_state):
        return []

    visitedNodes = []

    pQueue = util.PriorityQueue()
    #((coordinate/node , action to current node , cost to current node),priority)
    pQueue.push((initial_state, [], 0), 0)

    while not pQueue.isEmpty():

        currentNode, actions, prevCost = pQueue.pop()

        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                newCostToNode = prevCost + cost
                heuristicCost = newCostToNode + heuristic(nextNode,problem)
                pQueue.push((nextNode, newAction, newCostToNode),heuristicCost)
    util.raiseNotDefined()
```

فاطمه خدادادی_97143013