Project 1: Search

Professor : **Dr.AliShakiba**

By : **Maryam Mohammadabadi**

پروژه اول : جستجو

سوال 1 : جستجوی اول عمق(Depth First Search)

پاسخ :

1.دریافت حالت اولیه

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

2.چک کردن اینکه حالت نهایی هست یا خیر

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

3.یک پشته تولید میکنیم

```
myStack = util.Stack()
```

4.یک لیست خالی برای گره های مشاهده شده ایجاد میکنیم

```
visitedNodes = []
```

5.سپس حالت اولیه و لیست ایجاد شده را در پشته قرار میدهیم.(push)

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

6.یک حقله ایجاد میکنیم که شرط پایان آن خالی بود پشته است.

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

6.1. یک عنصر را از پشته بر میداریم(pop)

```
currentNode, actions = myStack.pop()
```

6.2. اگر آن عنصر در لیست گره های دیده شده نبود.آن را به آن لیست اضافه میکنیم.

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

6.3.سپس چک میکنیم اگر عنصر مورد نظر حالت نهایی بود لیست اقدامات را برمیگردانیم.

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

6.3.1.اگر شرط 6.2 درست بود و همچنین شرط 6.3 (حالت نهایی نباشد) برقرار بود

باید با استفاده از متد getSuccessors , گره و اقدام بعدی را مشخص کنیم و داخل پشته قرار دهیم.

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

کد کامل :

```
def depthFirstSearch(problem):

    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()
```

سوال 2 : جستجوی اول سطح(Breadth First Search)

پاسخ :

عملکرد: الگوریتم از ریشه شروع می‌کند (در گراف‌ها یا درخت‌های بدون ریشه رأس دلخواهی به عنوان ریشه انتخاب می‌شود) 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.

راه حل این سوال مثل سوال قبل است با این تفاوت که به جای پشته(Stack) از صف(Queue) استفاده میکنیم.

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

کد کامل :

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

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

سوال 3 : جستجوی هزینه یکنواخت(Varying the Cost Function)

پاسخ :

هر بار کم هزینه ترین گره گسترش نیافته را گسترش می دهد.

هزینه ی گره: هزینه ی مسیر از ریشه تا آن گره در درخت جستجو

اولویت یک گره: هزینه ی مسیر از ریشه تا آن گره.

پیاده سازی :

1.دریافت حالت اولیه

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

2.چک کردن اینکه حالت نهایی هست یا خیر

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

3.یک لیست خالی برای گره های مشاهده شده ایجاد میکنیم

```
visitedNodes = []
```

4.سپس یک صف الویت ایجاد میکنیم.

```
pQueue = util.PriorityQueue()
```

5. سپس باید initial_state را به عنوان مختصات / گره ، لیست خالی (که نشان دهنده عمل به گره فعلی است) ، هزینه به گره فعلی و اولویت در صف اولویت تولید شده قرار دهیم

((coordinate/node , action to current node , cost to current node),priority)

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

6.یک حقله ایجاد میکنیم که شرط پایان آن خالی بود صف است.

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

6.1. یک عنصر را از صف بر میداریم(pop)

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

6.2. اگر آن عنصر در لیست گره های دیده شده نبود.آن را به آن لیست اضافه میکنیم.

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

6.3.سپس چک میکنیم اگر عنصر مورد نظر حالت نهایی بود لیست اقدامات را برمیگردانیم.

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

6.3.1.اگر شرط 6.2 درست بود و همچنین شرط 6.3 (حالت نهایی نباشد) برقرار بود

باید با استفاده از متد getSuccessors , باید گره بعدی ، اقدام بعدی و اولویت بعدی را پیدا کنیم سپس آن ها را در صف الویت قرار دهیم.

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

کد کامل :

```
def uniformCostSearch(problem):
    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()
```

سوال 4 :جستجوی*A (A* search)

پاسخ :

از الگوریتم A* برای تخمین کوتاه‌ترین مسیر در مسائل جهان واقعی مانند نقشه‌ها وبازی های که امکان دارد موانع زیادی در آن‌ها باشد، استفاده می‌شود .

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

راه حل این قسمت دقیقاً همان راه حل قبلی است اما در مرحله 6.3.1 باید تغییراتی انجام دهیم:

6.3.1.اگر شرط 6.2 درست بود و همچنین شرط 6.3 (حالت نهایی نباشد) برقرار بود

باید با استفاده از متد getSuccessors , باید گره بعدی(next node) ، اقدام جدید(action,) ، هزینه بعدی(cost) را برای گره پیدا کنیم و هزینه اکتشافی(heuristic cost) را تعیین کنیم و سپس آن ها را در صف الویت قرار دهیم.

```
            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()
```

کد کامل :

```
def aStarSearch(problem, heuristic=nullHeuristic):
    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()
```