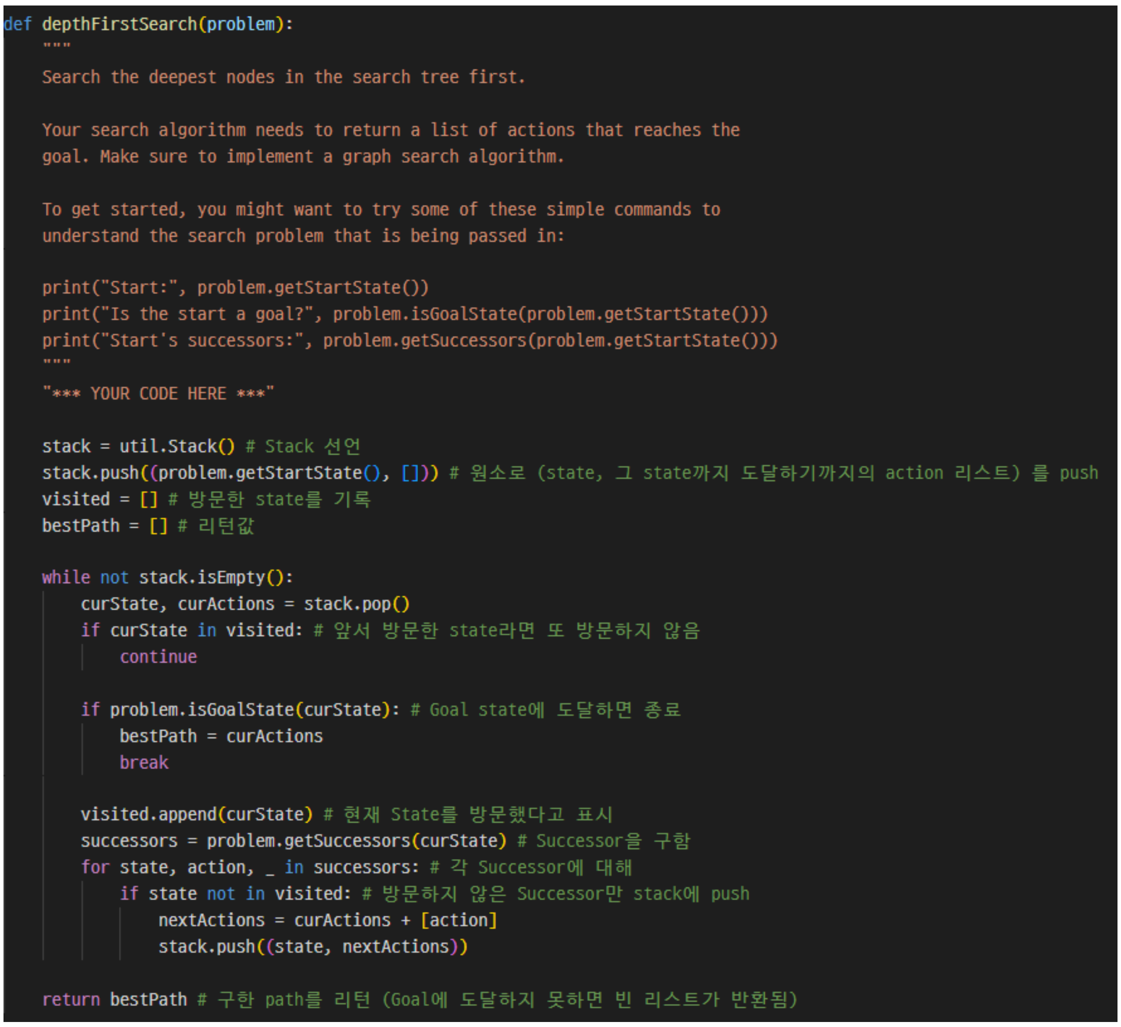
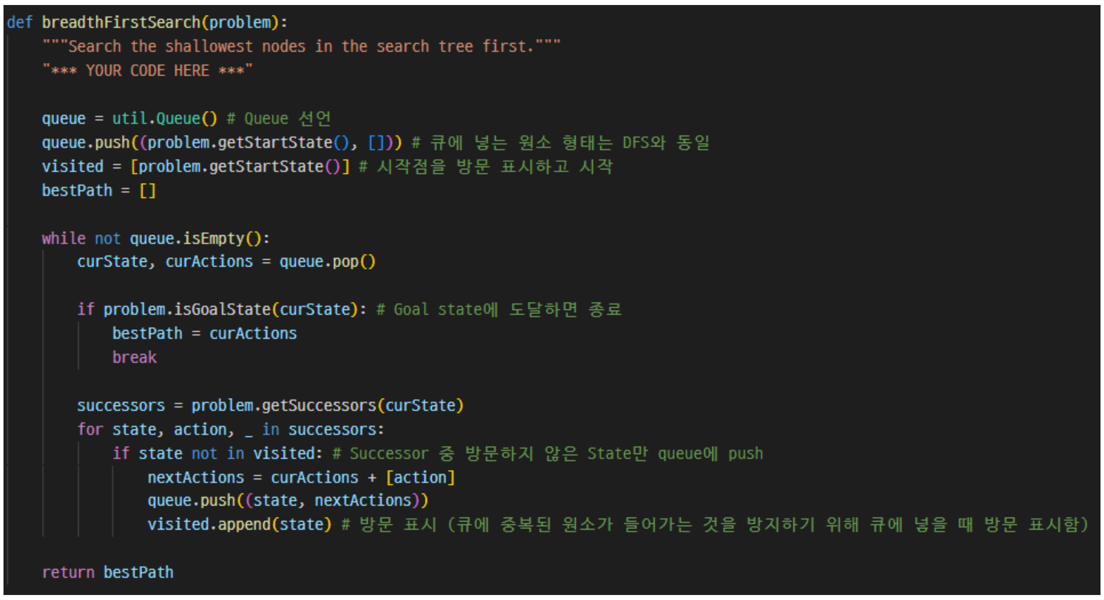
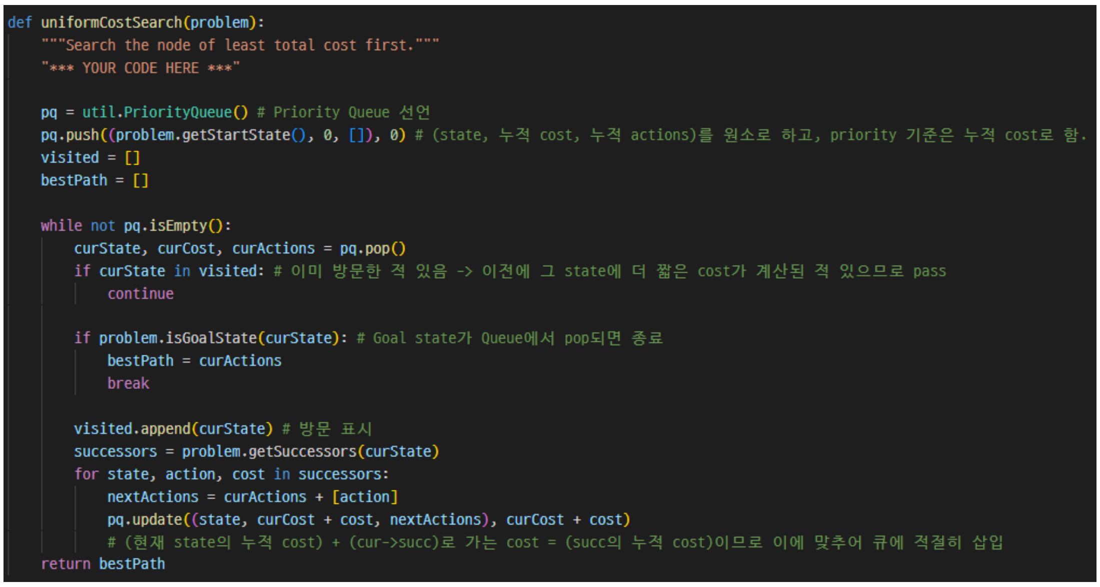
[search.py]



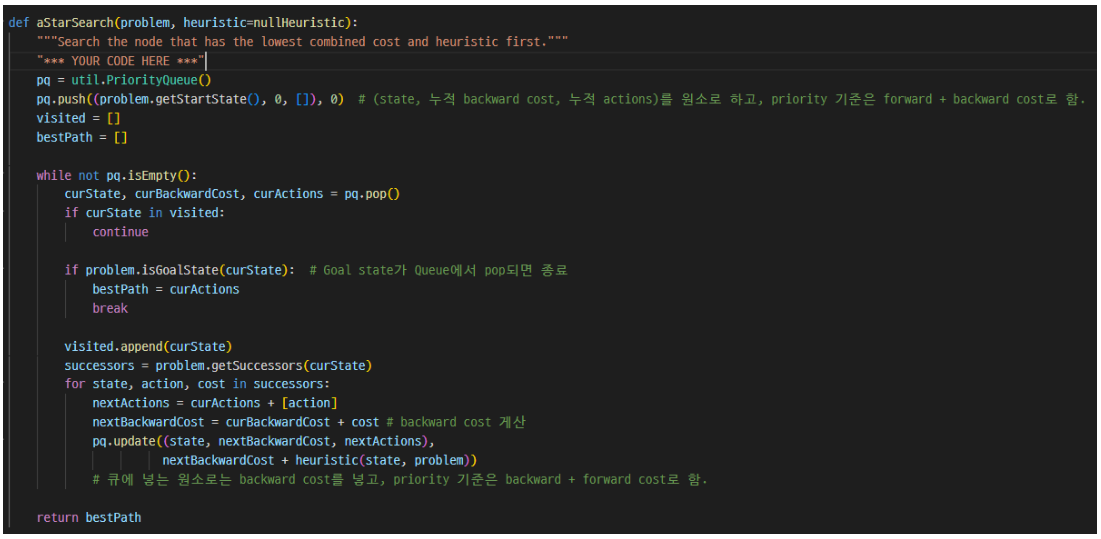
```python
def depthFirstSearch(problem):
    """
    Search the deepest nodes in the search tree first.

    Your search algorithm needs to return a list of actions that reaches the
    goal. Make sure to implement a graph search algorithm.

    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:

    print("Start:", problem.getStartState())
    print("Is the start a goal?", problem.isGoalState(problem.getStartState()))
    print("Start's successors:", problem.getSuccessors(problem.getStartState()))
    """
    "*** YOUR CODE HERE ***"

    stack = util.Stack() # Stack 선언
    stack.push((problem.getStartState(), [])) # 원소로 (state, 그 state까지 도달하기까지의 action 리스트) 를 push
    visited = [] # 방문한 state를 기록
    bestPath = [] # 리턴값

    while not stack.isEmpty():
        curState, curActions = stack.pop()
        if curState in visited: # 앞서 방문한 state라면 또 방문하지 않음
            continue

        if problem.isGoalState(curState): # Goal state에 도달하면 종료
            bestPath = curActions
            break

        visited.append(curState) # 현재 State를 방문했다고 표시
        successors = problem.getSuccessors(curState) # Successor을 구함
        for state, action, _ in successors: # 각 Successor에 대해
            if state not in visited: # 방문하지 않은 Successor만 stack에 push
                nextActions = curActions + [action]
                stack.push((state, nextActions))

    return bestPath # 구한 path를 리턴 (Goal에 도달하지 못하면 빈 리스트가 반환됨)
```



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

    queue = util.Queue() # Queue 선언
    queue.push((problem.getStartState(), [])) # 큐에 넣는 원소 형태는 DFS와 동일
    visited = [problem.getStartState()] # 시작점을 방문 표시하고 시작
    bestPath = []

    while not queue.isEmpty():
        curState, curActions = queue.pop()

        if problem.isGoalState(curState): # Goal state에 도달하면 종료
            bestPath = curActions
            break

        successors = problem.getSuccessors(curState)
        for state, action, _ in successors:
            if state not in visited: # Successor 중 방문하지 않은 State만 queue에 push
                nextActions = curActions + [action]
                queue.push((state, nextActions))
                visited.append(state) # 방문 표시 (큐에 중복된 원소가 들어가는 것을 방지하기 위해 큐에 넣을 때 방문 표시함)

    return bestPath
```

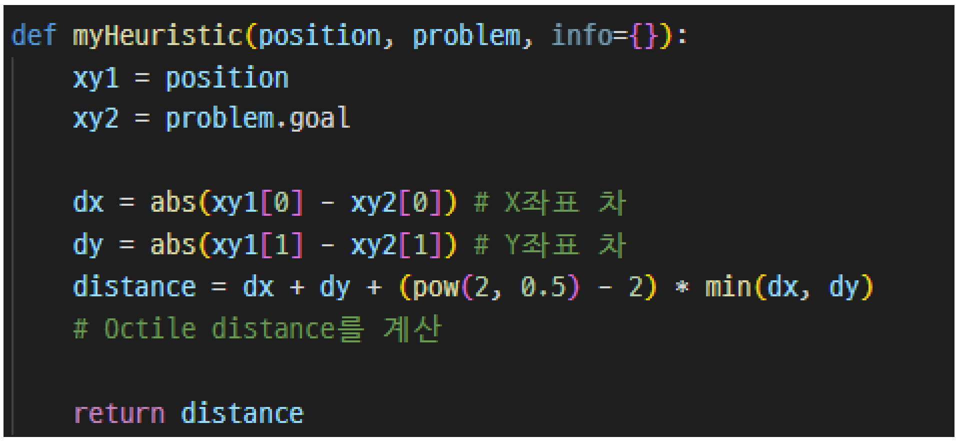


```python
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    "*** YOUR CODE HERE ***"

    pq = util.PriorityQueue() # Priority Queue 선언
    pq.push((problem.getStartState(), 0, []), 0) # (state, 누적 cost, 누적 actions)를 원소로 하고, priority 기준은 누적 cost로 함.
    visited = []
    bestPath = []

    while not pq.isEmpty():
        curState, curCost, curActions = pq.pop()
        if curState in visited: # 이미 방문한 적 있음 -> 이전에 그 state에 더 짧은 cost가 계산된 적 있으므로 pass
            continue

        if problem.isGoalState(curState): # Goal state가 Queue에서 pop되면 종료
            bestPath = curActions
            break

        visited.append(curState) # 방문 표시
        successors = problem.getSuccessors(curState)
        for state, action, cost in successors:
            nextActions = curActions + [action]
            pq.update((state, curCost + cost, nextActions), curCost + cost)
            # (현재 state의 누적 cost) + (cur->succ)로 가는 cost = (succ의 누적 cost)이므로 이에 맞추어 큐에 적절히 삽입
    return bestPath
```



```python
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"
    pq = util.PriorityQueue()
    pq.push((problem.getStartState(), 0, []), 0)  # (state, 누적 backward cost, 누적 actions)를 원소로 하고, priority 기준은 forward + backward cost로 함.
    visited = []
    bestPath = []

    while not pq.isEmpty():
        curState, curBackwardCost, curActions = pq.pop()
        if curState in visited:
            continue

        if problem.isGoalState(curState):  # Goal state가 Queue에서 pop되면 종료
            bestPath = curActions
            break

        visited.append(curState)
        successors = problem.getSuccessors(curState)
        for state, action, cost in successors:
            nextActions = curActions + [action]
            nextBackwardCost = curBackwardCost + cost # backward cost 계산
            pq.update((state, nextBackwardCost, nextActions),
                      nextBackwardCost + heuristic(state, problem))
            # 큐에 넣는 원소로는 backward cost를 넣고, priority 기준은 backward + forward cost로 함.

    return bestPath
```

[searchAgents.py]



```python
def myHeuristic(position, problem, info={}):
    xy1 = position
    xy2 = problem.goal

    dx = abs(xy1[0] - xy2[0]) # X좌표 차
    dy = abs(xy1[1] - xy2[1]) # Y좌표 차
    distance = dx + dy + (pow(2, 0.5) - 2) * min(dx, dy)
    # Octile distance를 계산

    return distance
```