ARTIFICIAL INTELLIGENCE

GV Nguyễn Thành An

DANH SÁCH THÀNH VIÊN

Tên nhóm

52100243 - TUẤN KIỆT
52100210 - KHÁNH HUÂN
52100017 - QUANG ĐÃNG
52100171 - VĂN CƯỜNG
52100369 - ĐÌNH VĂN

PHÂN CÔNG CÔNG VIỆC

BÀI 1,2: CƯỜNG + HUÂN
BÀI 1: VĂN
BÀI 2: KIỆT
BÀI 1,2,3: ĐÃNG
PPT: KIỆT
THUYẾT TRÌNH: VĂN

MỨC ĐỘ HOÀN THÀNH:

ARTIFICIAL INTELLIGENCE

CÂU 1
UNINFORMED
SEARCH

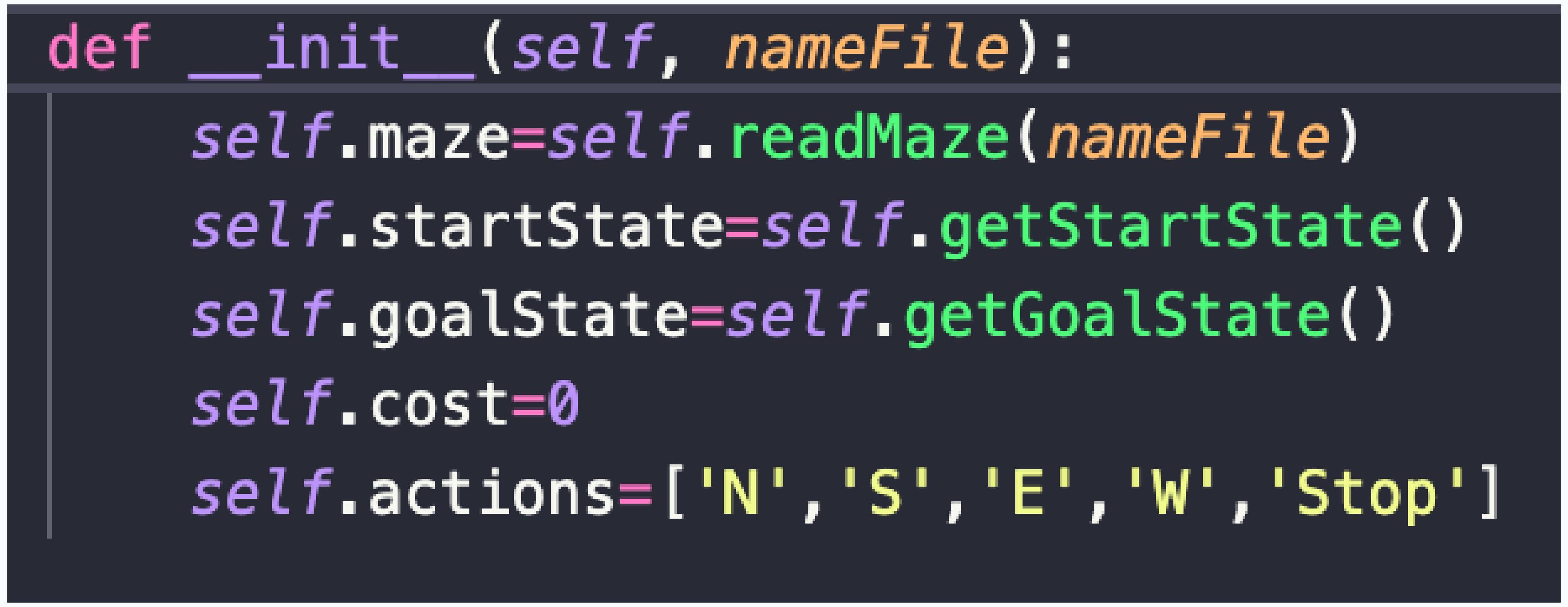
CÂU 2
BEST-FIREST
SEARCH

CÂU 3
LOCAL
SEARCH

CÂU 1: UNINFORMED SEARCH

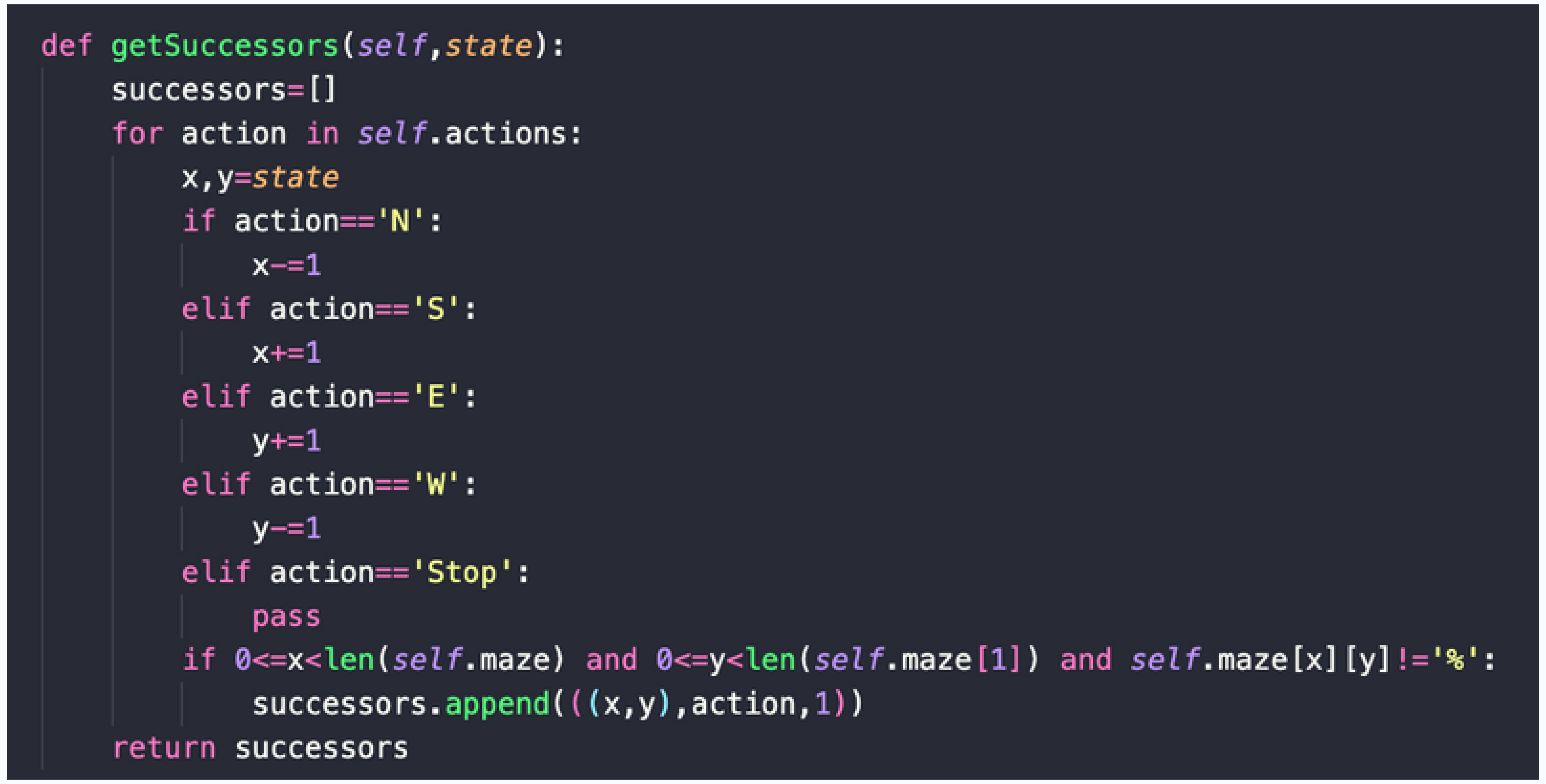
Y/C 1-1

STATE, NODE, INITIAL STATE



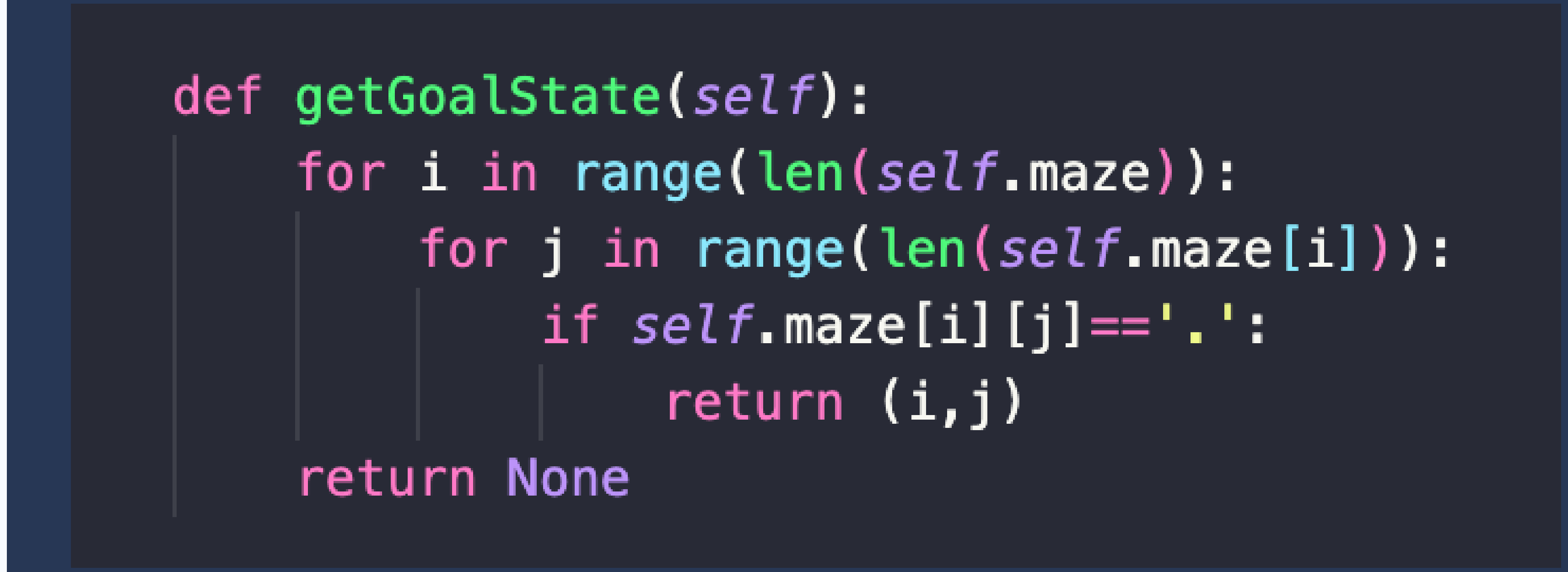
```python
def __init__(self, nameFile):
    self.maze=self.readMaze(nameFile)
    self.startState=self.getStartState()
    self.goalState=self.getGoalState()
    self.cost=0
    self.actions=['N','S','E','W','Stop']
```

SUCCESSOR FUNCTION



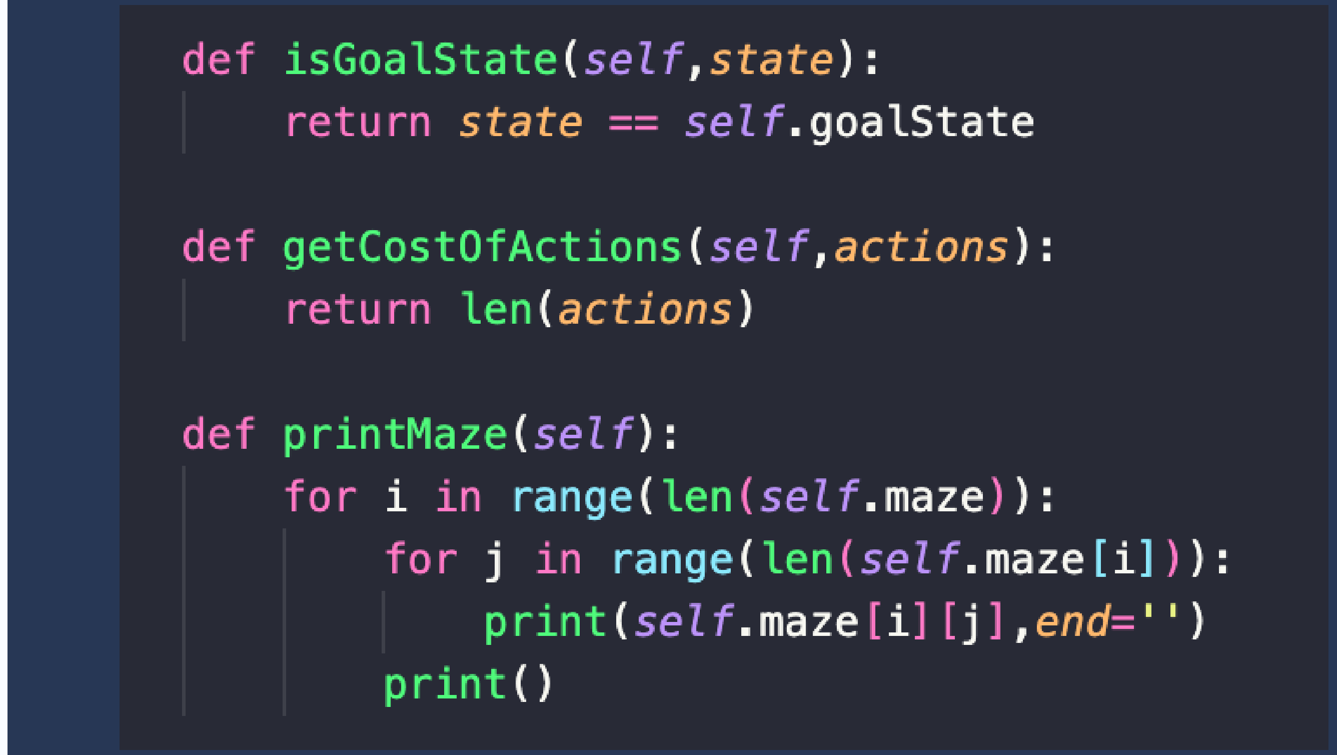
```python
def getSuccessors(self,state):
    successors=[]
    for action in self.actions:
        x,y=state
        if action=='N':
            x-=1
        elif action=='S':
            x+=1
        elif action=='E':
            y+=1
        elif action=='W':
            y-=1
        elif action=='Stop':
            pass
        if 0<=x<len(self.maze) and 0<=y<len(self.maze[1]) and self.maze[x][y]!='%':
            successors.append(((x,y),action,1))
    return successors
```

GOAL-TEST FUNCTION



```python
def getGoalState(self):
    for i in range(len(self.maze)):
        for j in range(len(self.maze[i])):
            if self.maze[i][j]=='.':
                return (i,j)
    return None
```

PATH-COST FUNCTION



```python
def isGoalState(self,state):
    return state == self.goalState

def getCostOfActions(self,actions):
    return len(actions)

def printMaze(self):
    for i in range(len(self.maze)):
        for j in range(len(self.maze[i])):
            print(self.maze[i][j],end='')
        print()
```

CÂU 1: UNINFORMED SEARCH

Y/C 1-2

BFS:

```
FUNCTION BFS(PROBLEMS) RETURNS [ACTIONS]
FRINGE = QUEUE(NEWSTATE, [ACTIONS TO CURSTATE])
FRINGE.ENQUEUE(PACMAN.STARTSTATE, ACTIONS FROM START TO CURSTATE)
VISITED = SET(VISITED STATE)
TEMP = ARRAY(ACTIONS)
WHILE FRINGE NOT EMPTY:
    CURSTATE, ACTIONS = FRINGE.DEQUEUE()
    IF CURSTATE IS NOT VISITED
        VISITED <- CURSTATE
        FOR CHILD, ACTION, COST IN GETSUCCESSORS(CURSTATE):
            X,Y = CHILD
            IF CHILD IS NOT VISITED THEN FRINGE.ENQUEUE(CHILD, ACTIONS + [ACTION])
        IF CURSTATE == GOALSTATE:
            IF PROBLEM IS SINGLEFOOD THEN RETURN ACTIONS
            ELSE THEN TEMP <- ACTIONS, FRINGE.CLEAR, VISITED.CLEAR, FRINGE.PUSH(CURSTATE,[])
                IF NUMFOOD == 0 THEN RETURN TEMP
```

CÂU 1: UNINFORMED SEARCH

Y/C 1-2

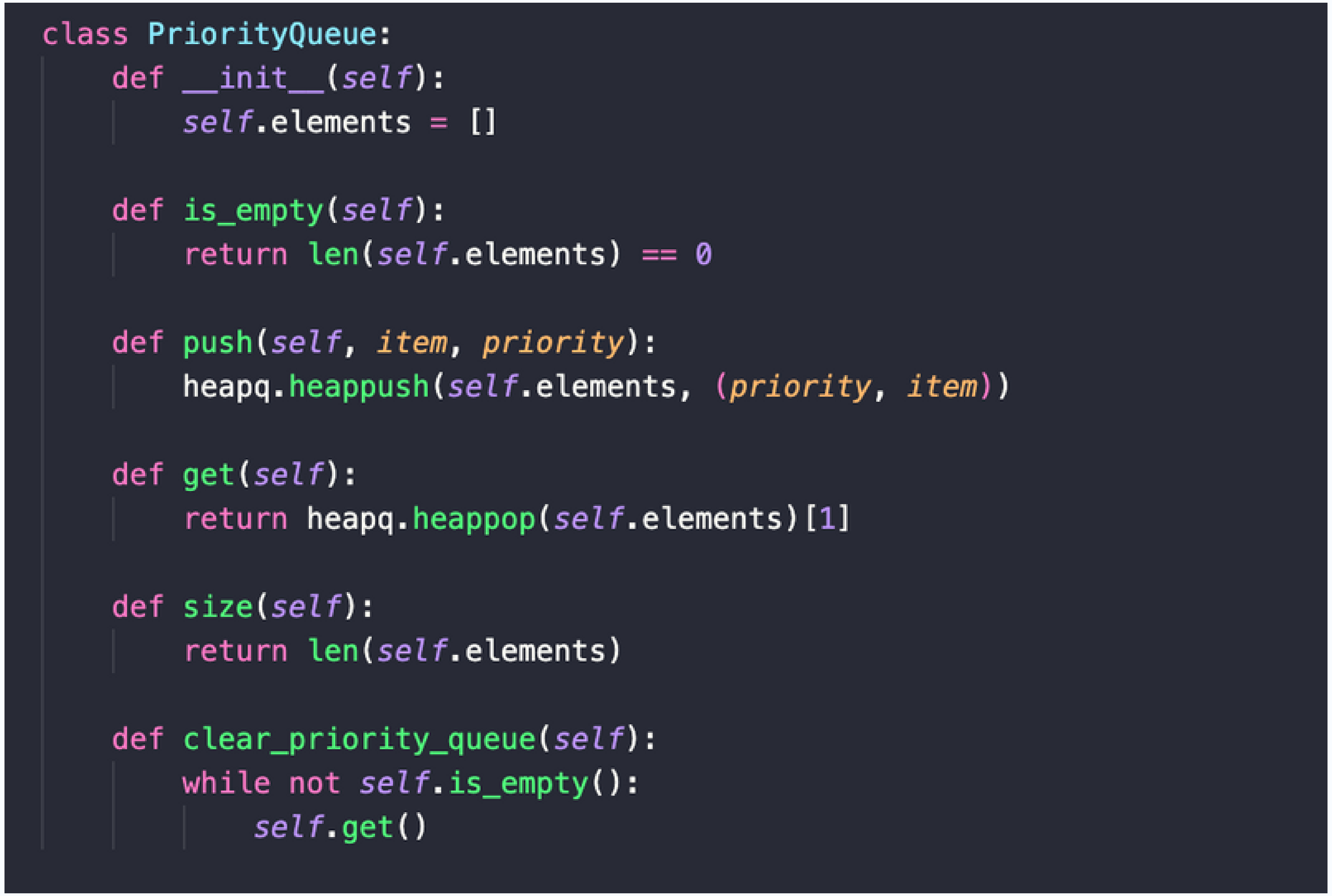
DFS:

```
FUNCTION BFS(PROBLEMS) RETURNS [ACTIONS]
FRINGE = STACK(NEWSTATE, [ACTIONS TO CURSTATE])
FRINGE.PUSH(PACMAN.STARTSTATE, ACTIONS FROM START TO CURSTATE)
VISITED = SET(VISITED STATE)
TEMP = ARRAY(ACTIONS)
WHILE FRINGE NOT EMPTY:
    CURSTATE, ACTIONS = FRINGE.POP()
    IF CURSTATE IS NOT VISITED
        VISITED <- CURSTATE
        FOR CHILD, ACTION, COST IN GETSUCCESSORS(CURSTATE):
            X,Y = CHILD
            IF CHILD IS NOT VISITED THEN FRINGE.PUSH(CHILD, ACTIONS + [ACTION])
    IF CURSTATE == GOALSTATE:
        IF PROBLEM IS SINGLEFOOD THEN RETURN ACTIONS
        ELSE THEN TEMP <- ACTIONS, FRINGE.CLEAR, FRINGE.PUSH(CURSTATE,[])
            IF NUMFOOD==0 THEN RETURN TEMP
```

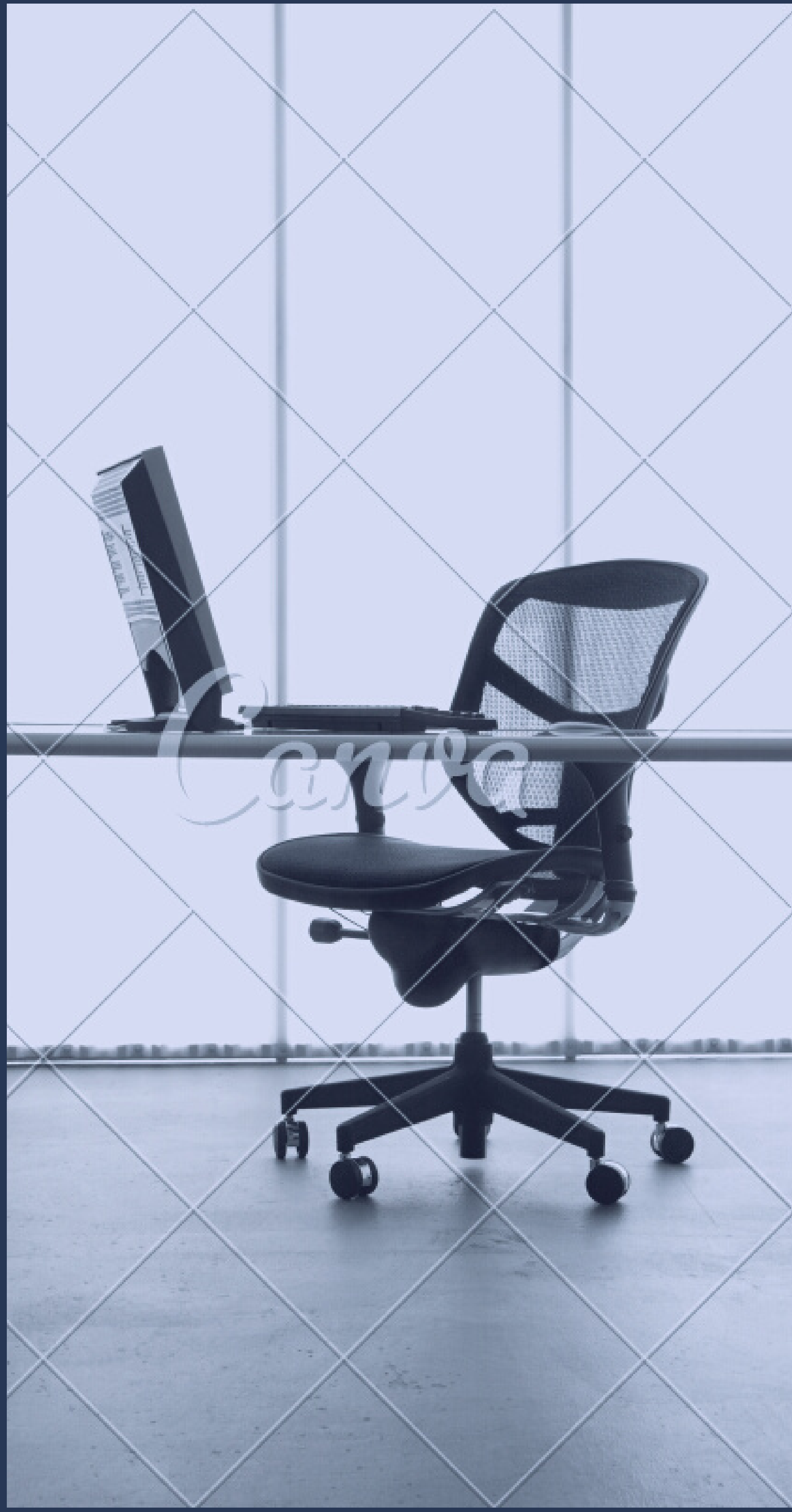
CÂU 1: UNINFORMED SEARCH

Y/C 1-2

PRIORITYQUEUE



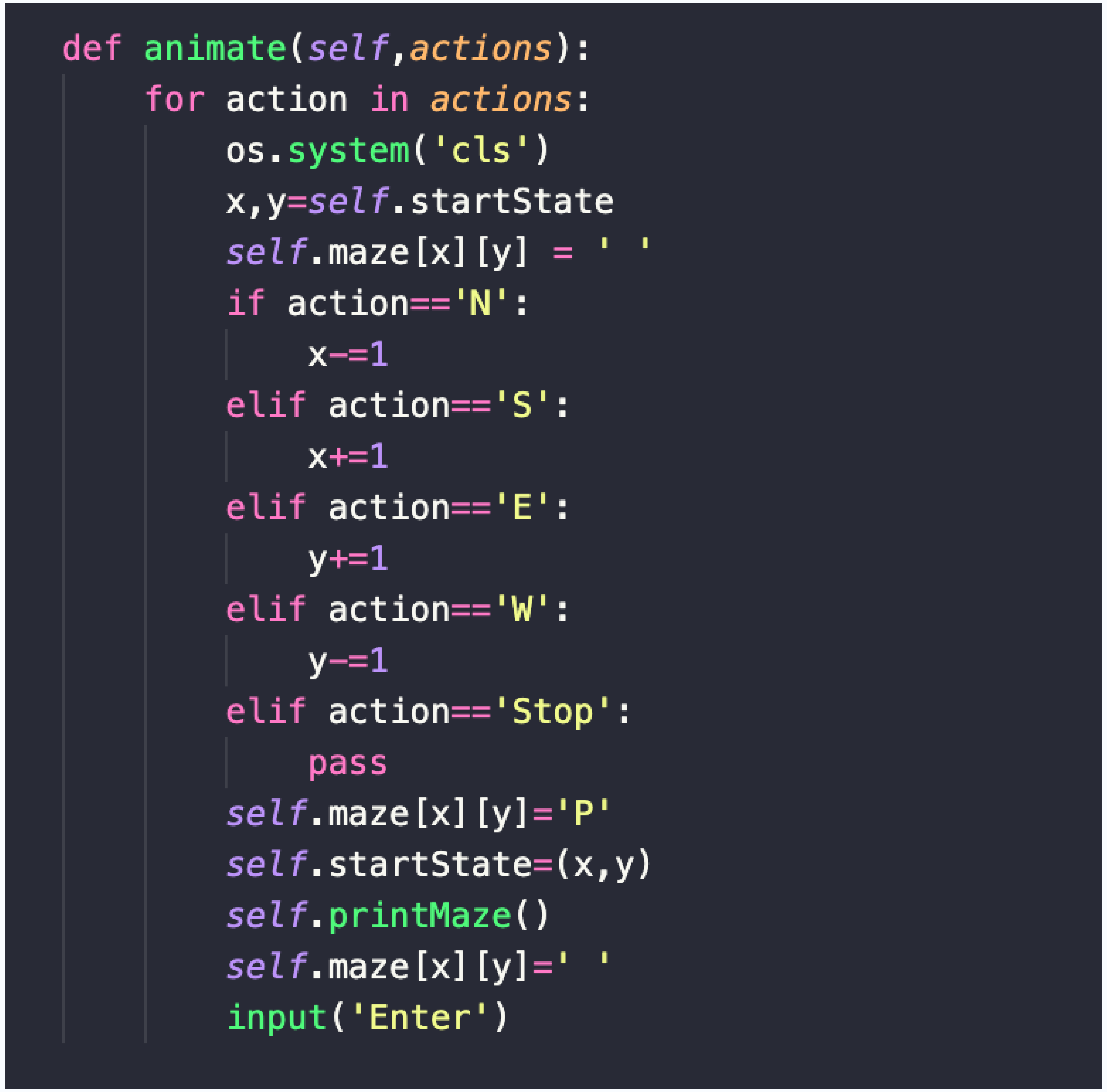
```python
class PriorityQueue:
    def __init__(self):
        self.elements = []

    def is_empty(self):
        return len(self.elements) == 0

    def push(self, item, priority):
        heapq.heappush(self.elements, (priority, item))

    def get(self):
        return heapq.heappop(self.elements)[1]

    def size(self):
        return len(self.elements)

    def clear_priority_queue(self):
        while not self.is_empty():
            self.get()
```



CÂU 1: UNINFORMED SEARCH

Y/C 1-4

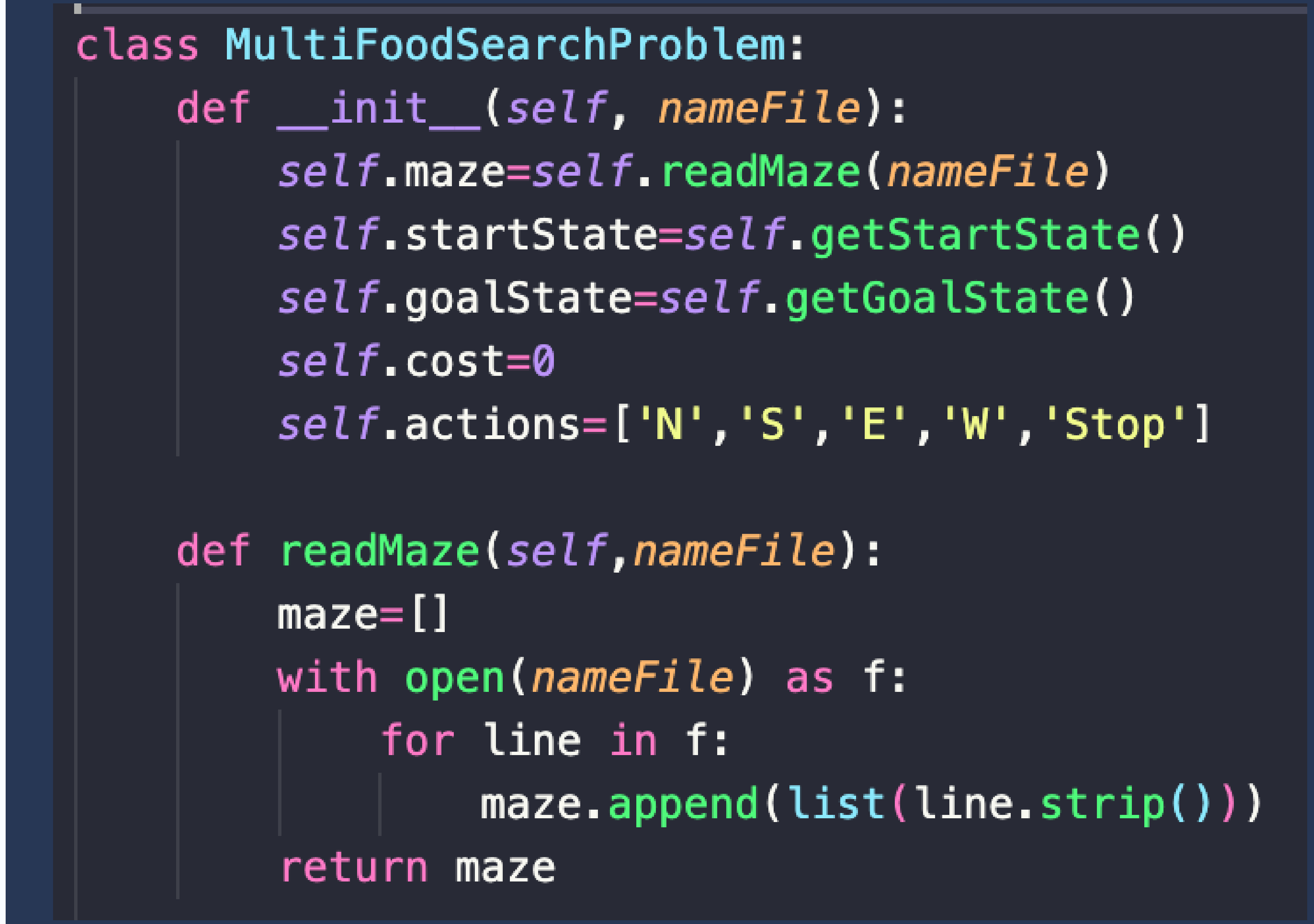
ANIMATE(SELF, ACTIONS)



```
def animate(self,actions):
    for action in actions:
        os.system('cls')
        x,y=self.startState
        self.maze[x][y] = ' '
        if action=='N':
            x-=1
        elif action=='S':
            x+=1
        elif action=='E':
            y+=1
        elif action=='W':
            y-=1
        elif action=='Stop':
            pass
        self.maze[x][y]='P'
        self.startState=(x,y)
        self.printMaze()
        self.maze[x][y]=' '
        input('Enter')
```

Y/C 1-5

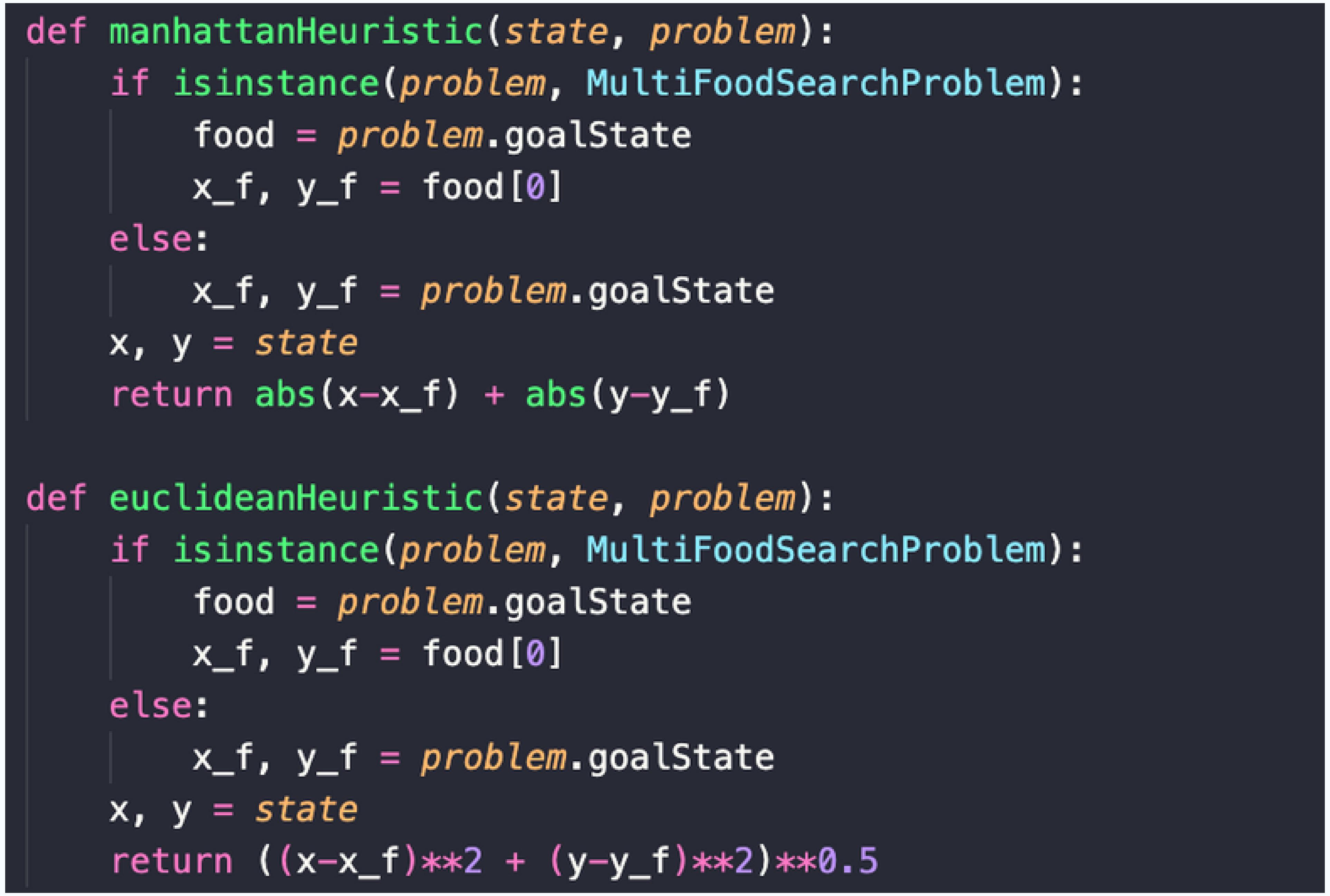
CLASS MULTIFOODSEARCHPROBLEM



```
class MultiFoodSearchProblem:
    def __init__(self, nameFile):
        self.maze=self.readMaze(nameFile)
        self.startState=self.getStartState()
        self.goalState=self.getGoalState()
        self.cost=0
        self.actions=['N','S','E','W','Stop']

    def readMaze(self,nameFile):
        maze=[]
        with open(nameFile) as f:
            for line in f:
                maze.append(list(line.strip()))
        return maze
```

CÂU 2: BEST-FIREST SEARCH

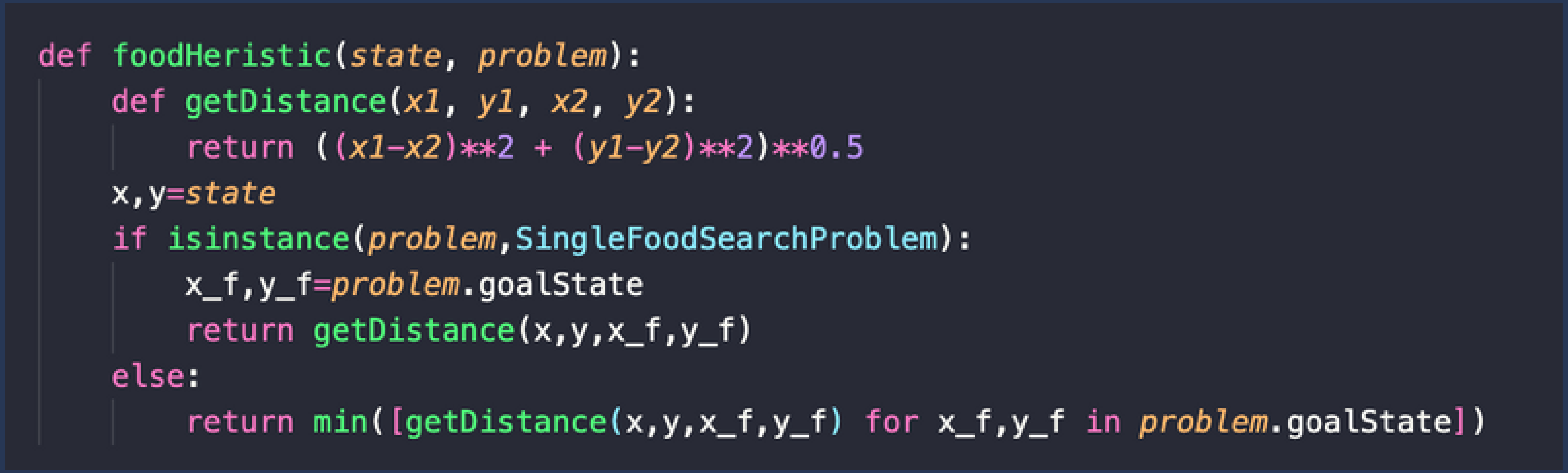
Y/C 2-1



```python
def manhattanHeuristic(state, problem):
    if isinstance(problem, MultiFoodSearchProblem):
        food = problem.goalState
        x_f, y_f = food[0]
    else:
        x_f, y_f = problem.goalState
    x, y = state
    return abs(x-x_f) + abs(y-y_f)

def euclideanHeuristic(state, problem):
    if isinstance(problem, MultiFoodSearchProblem):
        food = problem.goalState
        x_f, y_f = food[0]
    else:
        x_f, y_f = problem.goalState
    x, y = state
    return ((x-x_f)**2 + (y-y_f)**2)**0.5
```

CÂU 2: BEST-FIREST SEARCH

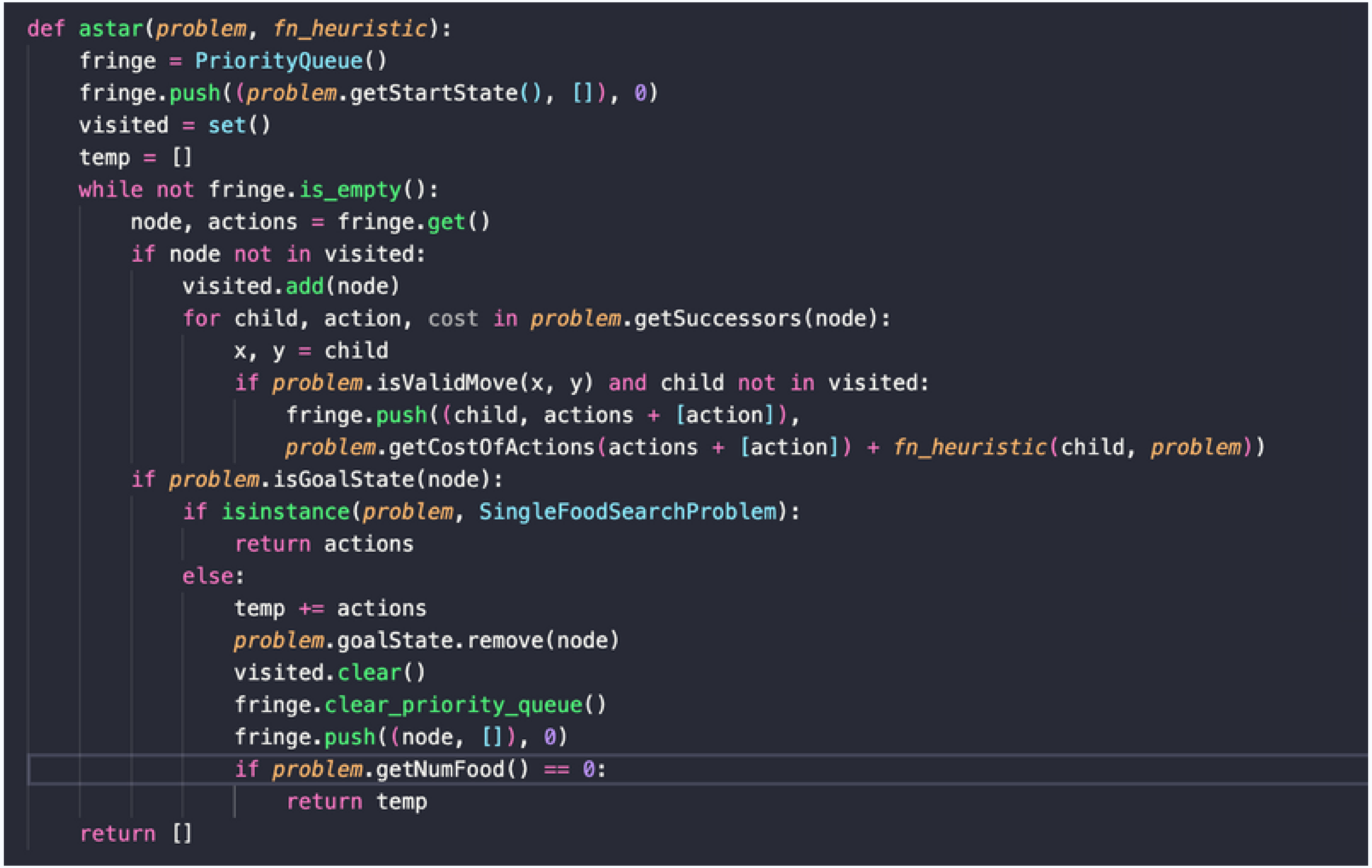
Y/C 2-2



```python
def foodHeristic(state, problem):
    def getDistance(x1, y1, x2, y2):
        return ((x1-x2)**2 + (y1-y2)**2)**0.5
    x,y=state
    if isinstance(problem,SingleFoodSearchProblem):
        x_f,y_f=problem.goalState
        return getDistance(x,y,x_f,y_f)
    else:
        return min([getDistance(x,y,x_f,y_f) for x_f,y_f in problem.goalState])
```

CÂU 2: BEST-FIREST SEARCH

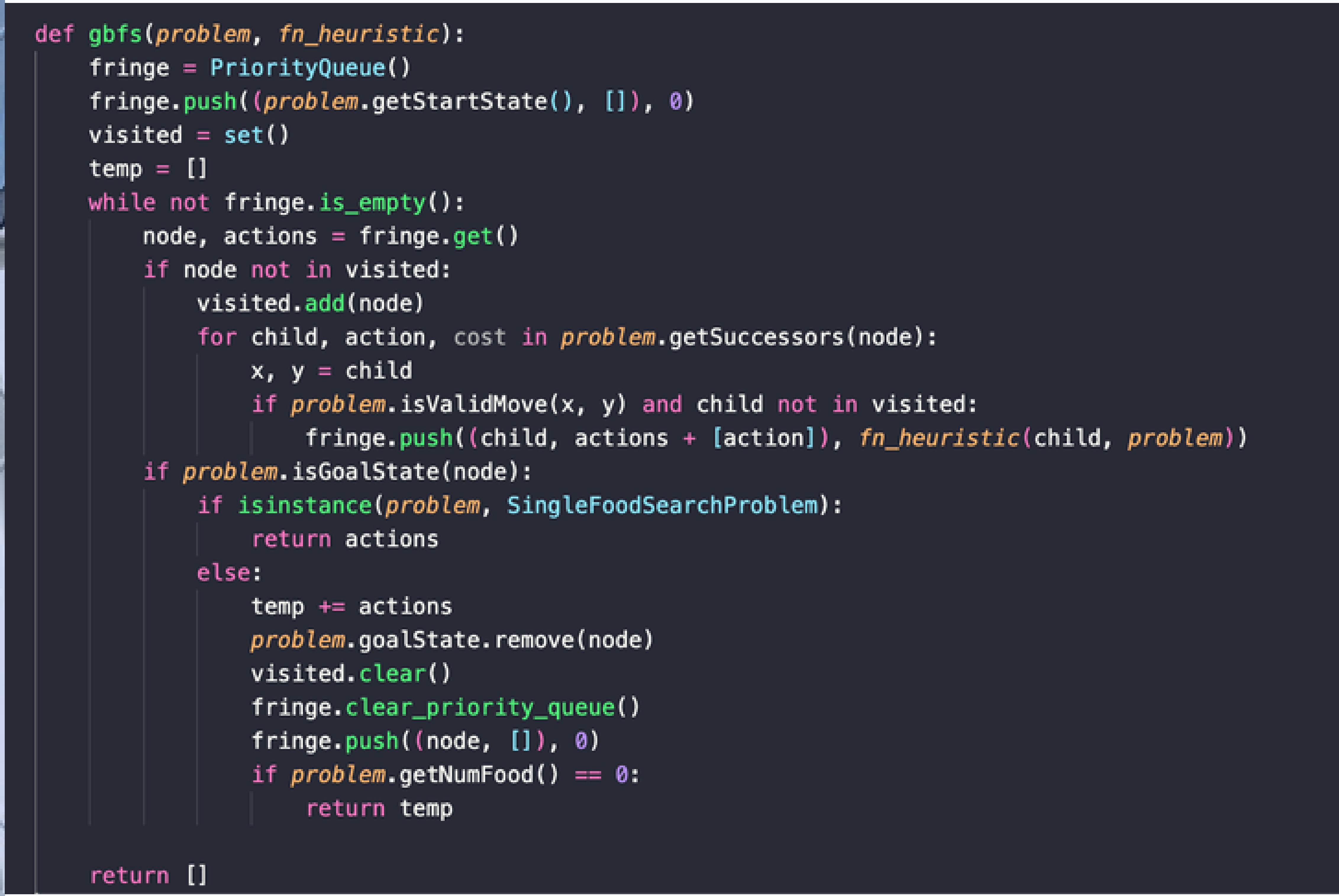
Y/C 2-3



```
def astar(problem, fn_heuristic):
    fringe = PriorityQueue()
    fringe.push((problem.getStartState(), []), 0)
    visited = set()
    temp = []
    while not fringe.is_empty():
        node, actions = fringe.get()
        if node not in visited:
            visited.add(node)
            for child, action, cost in problem.getSuccessors(node):
                x, y = child
                if problem.isValidMove(x, y) and child not in visited:
                    fringe.push((child, actions + [action]),
                    problem.getCostOfActions(actions + [action]) + fn_heuristic(child, problem))
        if problem.isGoalState(node):
            if isinstance(problem, SingleFoodSearchProblem):
                return actions
            else:
                temp += actions
                problem.goalState.remove(node)
                visited.clear()
                fringe.clear_priority_queue()
                fringe.push((node, []), 0)
                if problem.getNumFood() == 0:
                    return temp
    return []
```

CÂU 2: BEST-FIREST SEARCH

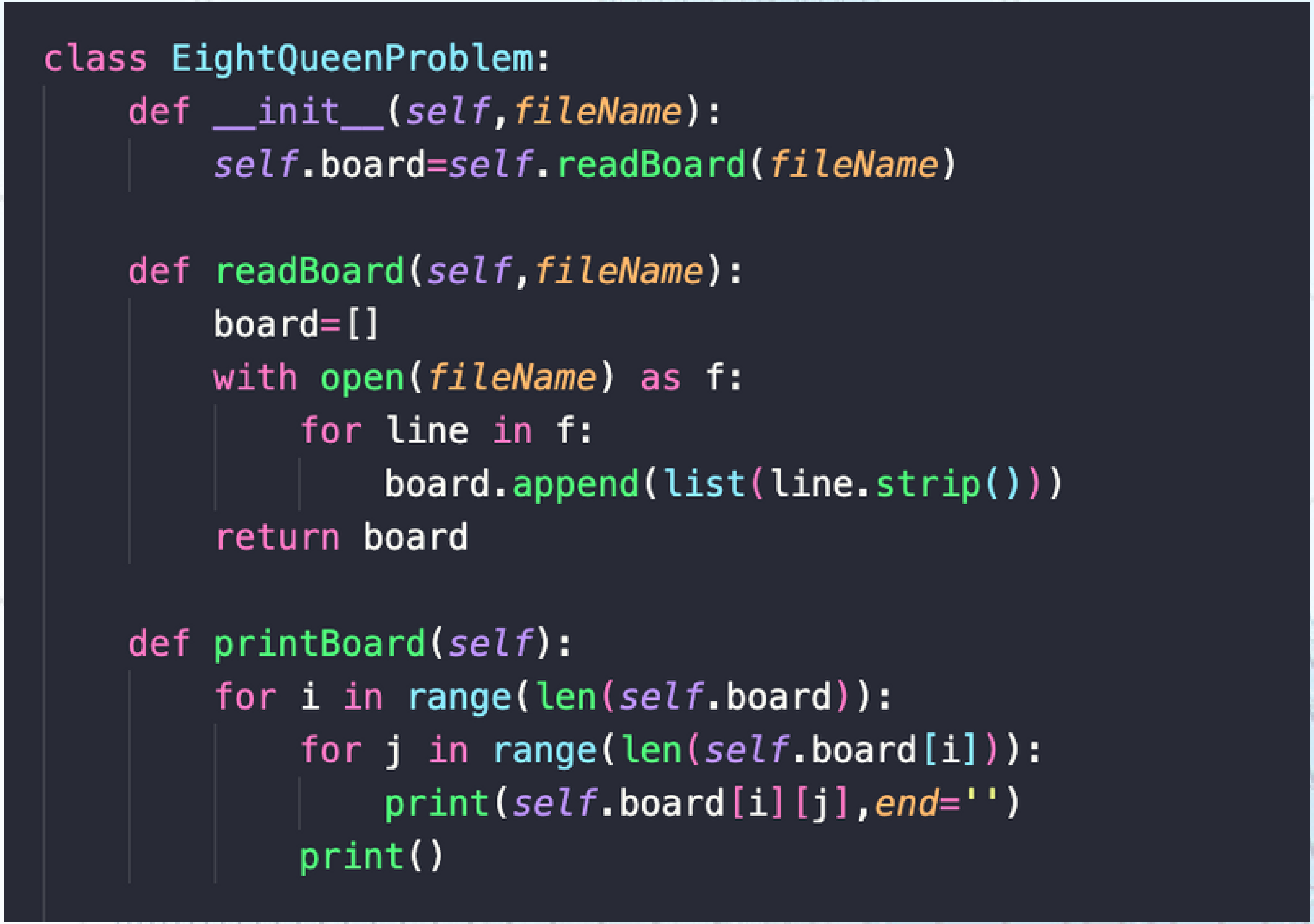
Y/C 2-5



```
def gbfs(problem, fn_heuristic):
    fringe = PriorityQueue()
    fringe.push((problem.getStartState(), []), 0)
    visited = set()
    temp = []
    while not fringe.is_empty():
        node, actions = fringe.get()
        if node not in visited:
            visited.add(node)
            for child, action, cost in problem.getSuccessors(node):
                x, y = child
                if problem.isValidMove(x, y) and child not in visited:
                    fringe.push((child, actions + [action]), fn_heuristic(child, problem))
        if problem.isGoalState(node):
            if isinstance(problem, SingleFoodSearchProblem):
                return actions
            else:
                temp += actions
                problem.goalState.remove(node)
                visited.clear()
                fringe.clear_priority_queue()
                fringe.push((node, []), 0)
                if problem.getNumFood() == 0:
                    return temp

    return []
```

CÂU 3: LOCAL SEARCH

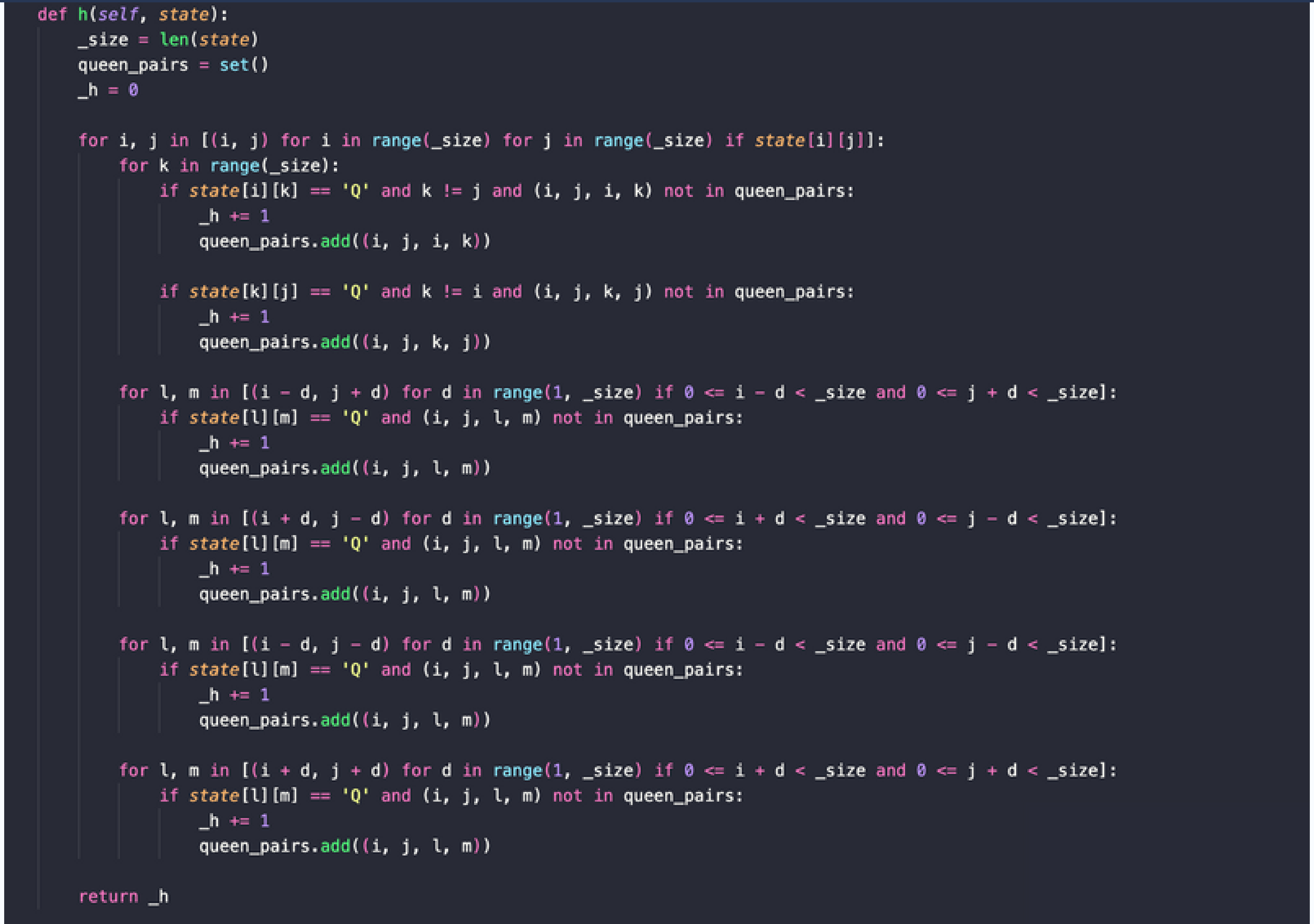
Y/C 3-1



```
class EightQueenProblem:
    def __init__(self,fileName):
        self.board=self.readBoard(fileName)

    def readBoard(self,fileName):
        board=[]
        with open(fileName) as f:
            for line in f:
                board.append(list(line.strip()))
        return board

    def printBoard(self):
        for i in range(len(self.board)):
            for j in range(len(self.board[i])):
                print(self.board[i][j],end='')
            print()
```

CÂU 3: LOCAL SEARCH

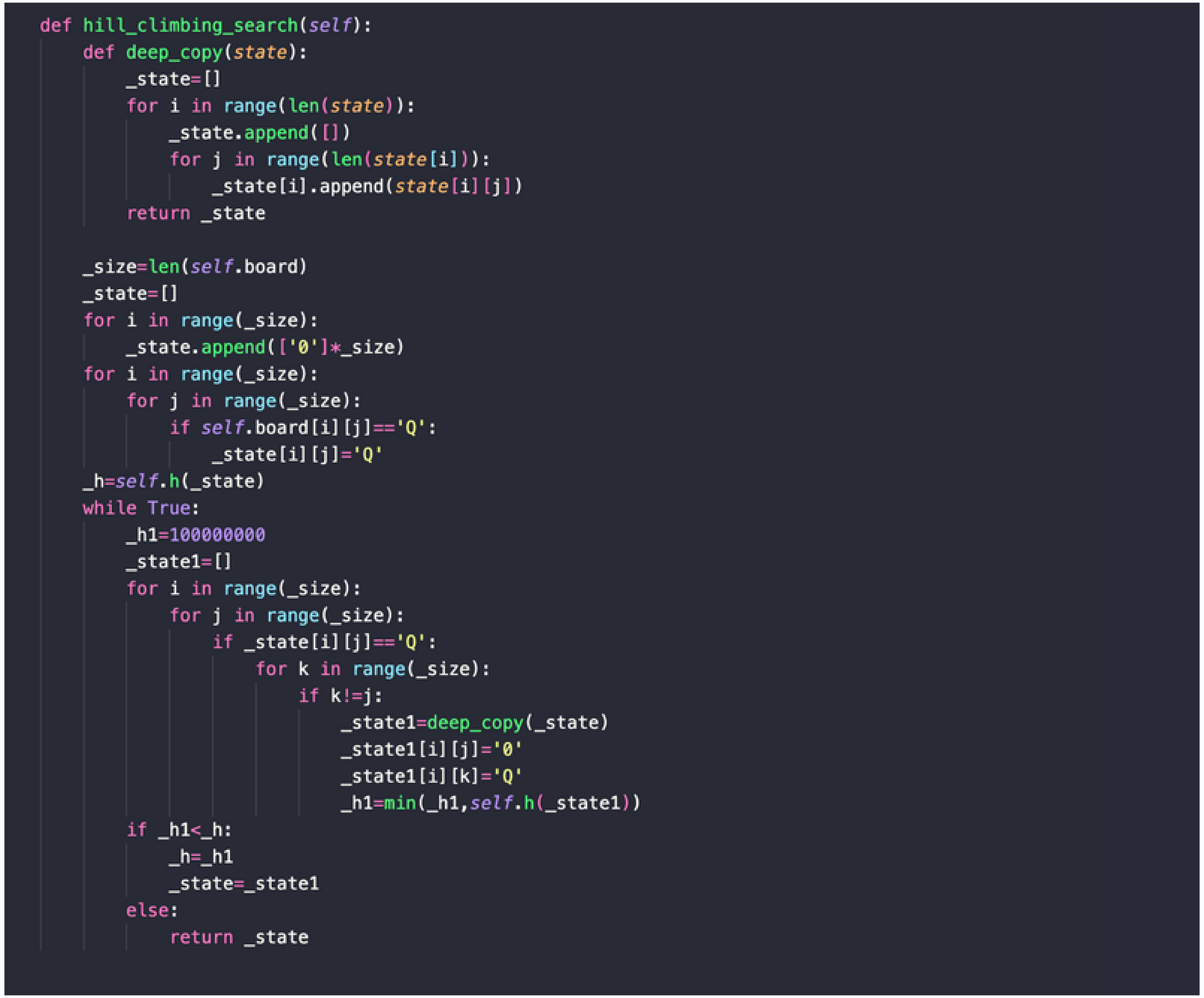
Y/C 3-1



```python
def h(self, state):
    _size = len(state)
    queen_pairs = set()
    _h = 0

    for i, j in [(i, j) for i in range(_size) for j in range(_size) if state[i][j]]:
        for k in range(_size):
            if state[i][k] == 'Q' and k != j and (i, j, i, k) not in queen_pairs:
                _h += 1
                queen_pairs.add((i, j, i, k))

            if state[k][j] == 'Q' and k != i and (i, j, k, j) not in queen_pairs:
                _h += 1
                queen_pairs.add((i, j, k, j))

        for l, m in [(i - d, j + d) for d in range(1, _size) if 0 <= i - d < _size and 0 <= j + d < _size]:
            if state[l][m] == 'Q' and (i, j, l, m) not in queen_pairs:
                _h += 1
                queen_pairs.add((i, j, l, m))

        for l, m in [(i + d, j - d) for d in range(1, _size) if 0 <= i + d < _size and 0 <= j - d < _size]:
            if state[l][m] == 'Q' and (i, j, l, m) not in queen_pairs:
                _h += 1
                queen_pairs.add((i, j, l, m))

        for l, m in [(i - d, j - d) for d in range(1, _size) if 0 <= i - d < _size and 0 <= j - d < _size]:
            if state[l][m] == 'Q' and (i, j, l, m) not in queen_pairs:
                _h += 1
                queen_pairs.add((i, j, l, m))

        for l, m in [(i + d, j + d) for d in range(1, _size) if 0 <= i + d < _size and 0 <= j + d < _size]:
            if state[l][m] == 'Q' and (i, j, l, m) not in queen_pairs:
                _h += 1
                queen_pairs.add((i, j, l, m))

    return _h
```

CÂU 3: LOCAL SEARCH

Y/C 3-2



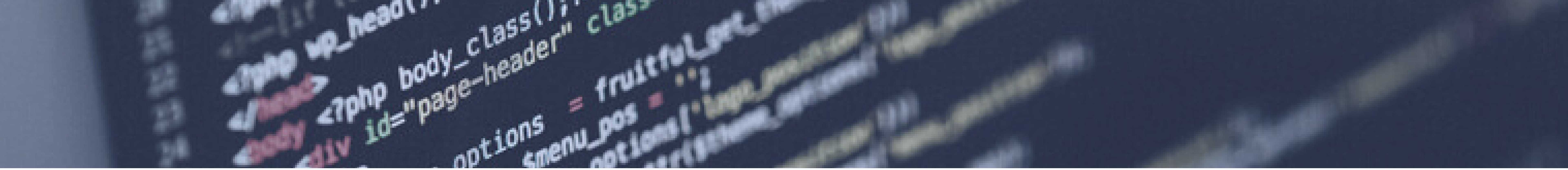
```
def hill_climbing_search(self):
    def deep_copy(state):
        _state=[]
        for i in range(len(state)):
            _state.append([])
            for j in range(len(state[i])):
                _state[i].append(state[i][j])
        return _state

    _size=len(self.board)
    _state=[]
    for i in range(_size):
        _state.append(['0']*_size)
    for i in range(_size):
        for j in range(_size):
            if self.board[i][j]=='Q':
                _state[i][j]='Q'
    _h=self.h(_state)
    while True:
        _h1=100000000
        _state1=[]
        for i in range(_size):
            for j in range(_size):
                if _state[i][j]=='Q':
                    for k in range(_size):
                        if k!=j:
                            _state1=deep_copy(_state)
                            _state1[i][j]='0'
                            _state1[i][k]='Q'
                            _h1=min(_h1,self.h(_state1))
        if _h1<_h:
            _h=_h1
            _state=_state1
        else:
            return _state
```

THUẬN LỢI

ĐƯỢC LÀM QUEN VỚI CÁC THUẬT TOÁN TÌM KIẾM THÔNG QUA CÁC BÀI TẬP LAB, CÁC BẠN TRONG NHÓM ĐỀU ĐÒNG LÒNG, GÓP SỨC ĐỂ HOÀN THÀNH BÀI BÁO CÁO

KHÓ KHĂN

NHÓM CHỈ CÓ 1 BẠN THÀNH THẠO NGÔN NGỮ PYTHON CÁC BẠN KHÁC THÌ KHÁ MỚI MẺ NÊN CẦN PHẢI TÌM HIỂU, THAM KHẢO NHIỀU HƠN



BẢNG ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH CỦA NHÓM

Câu 1	Câu 2	Câu 3
90%	90%	90%

CẢM ƠN THẦY CÔ VÀ CÁC BẠN ĐÃ LẮNG NGHE !