

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни
«ПІС»

«Неінформативний, інформативний та локальний пошук»

Виконав(ла)

ІТ-02 Макаров І.С.
(шифр, прізвище, ім'я, по батькові)

Перевірив

(прізвище, ім'я, по батькові)

Київ 2021

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ.....	3
2	ЗАВДАННЯ.....	4
3	ВИКОНАННЯ.....	8
3.1	ПСЕВДОКОД АЛГОРИТМІВ.....	8
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ.....	8
3.2.1	<i>Вихідний код.....</i>	<i>8</i>
3.2.2	<i>Приклади роботи.....</i>	<i>8</i>
3.3	ДОСЛІДЖЕННЯ АЛГОРИТМІВ.....	8
	ВИСНОВОК.....	11
	КРИТЕРІЇ ОЦІНЮВАННЯ.....	12

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – розглянути та дослідити алгоритми неінформативного, інформативного та локального пошуку. Провести порівняльний аналіз ефективності використання алгоритмів.

ВИКОНАННЯ

1.1 Lee Search

Код алгоритму:

```
def leeSearch(problem) -> list[Directions]:
    goalPoint: Point = problem.goal # coordinates of a destination
    startPoint: Point = problem.getStartState()

    boardOfMoves = [
        [NOT_VISITED for _ in range(problem.board.width)]
        for _ in range(problem.board.height)
    ]

    boardOfMoves[startPoint.y][startPoint.x] = 0 # make start point visited

    toVisitQueue = deque()
    toVisitQueue.append(startPoint)

    while toVisitQueue and (currentPoint := toVisitQueue.popleft()):
        possibleMoves: list[PossibleStep] = problem.getSuccessors(currentPoint) # getting all possible moves from this point
        currentPointCost: int = boardOfMoves[currentPoint.y][currentPoint.x]

        for move in possibleMoves:
            if boardOfMoves[move.coordinates.y][move.coordinates.x] != NOT_VISITED:
                continue

            boardOfMoves[move.coordinates.y][move.coordinates.x] = currentPointCost + 1
            toVisitQueue.append(move.coordinates)

            if move.coordinates == goalPoint: # check if current move is a goal
                backwardsPath = _make_backwards_path(boardOfMoves, goalPoint, startPoint)

                return reversePath(backwardsPath) # making path forward

    raise PathNotFound(f'This is impossible to build path between {startPoint} and {goalPoint}')
```

Алгоритм є неінформативним, однак в ситуації, коли під рукою більше нічого нема є досить дієвим, та зі своєю задачею справляється. Алгоритм “під капотом” оснований на BFS, що дозволяє нам стверджувати, що знайдений нами шлях є оптимальним. Однак з іншого боку, така система не може працювати на зваженому графі.



1.2 A Star

Код алгоритму:

```
def aStarSearch(
    problem,
    heuristic=manhattanDistanceHeuristics,
    raiseErrorIfNotFound: bool = False,
    showAnimation: bool = True,
    beGreedy: bool = True,
) -> list[Directions]:
    foodCoordinates: list[Point] = problem.foodCoordinates
    startPoint: Point = problem.getStartState()

    bestPathSoFar: Optional[Node] = None

    path = [startPoint]
    openList = [Node(coordinates=startPoint, path=path)]

    closedList = []

    while openList:
        nodeWithSmallestScore = sorted(openList)[0]
        currentNodePath = nodeWithSmallestScore.path

        if showAnimation:
            problem.displayPath(currentNodePath)
            # sleep(0.3)

        openList.remove(nodeWithSmallestScore)
        closedList.append(nodeWithSmallestScore.coordinates)

        if nodeWithSmallestScore.coordinates in foodCoordinates: # we found one of the goals
            nodeWithSmallestScore.passedFood.append(nodeWithSmallestScore.coordinates)

        if problem.isGoalState(nodeWithSmallestScore):
            if bestPathSoFar is None or bestPathSoFar.eatedFoodCount <
nodeWithSmallestScore.eatedFoodCount:
                bestPathSoFar = nodeWithSmallestScore

            if heuristic != foodSearchHeuristics:
                return makePathFromPoints(nodeWithSmallestScore.path)

            if bestPathSoFar.eatedFoodCount > 2/3 * len(foodCoordinates):
                return makePathFromPoints(nodeWithSmallestScore.path)

        possibleMoves: list[PossibleStep] = problem.getSuccessors(nodeWithSmallestScore.coordinates)
        # getting all possible moves from this point
        childrenNodes = [
            Node(possibleMove.coordinates, currentNodePath + [possibleMove.coordinates],
passedFood=deepcopy(nodeWithSmallestScore.passedFood))
            for possibleMove in possibleMoves
        ]
```

```

for child in childrenNodes:
    if child.coordinates in closedList:
        continue

    child.heuristic = heuristic(child.coordinates, problem, child.passedFood)

    stepCost = -1 if beGreeady else 1
    child.initialLength = nodeWithSmallestScore.initialLength + stepCost
    child.score = child.heuristic + child.initialLength

    nodeToRemove, childNodeAlreadyExists = None, False # if child node already exists in
openList we decide if we should replace it
    for node in openList:
        if node.coordinates == child.coordinates and child.score < node.score:
            nodeToRemove = node
            break
        if node.coordinates == child.coordinates and child.score > node.score:
            childNodeAlreadyExists = True
            break

    if nodeToRemove is not None:
        openList.remove(nodeToRemove)

    if not childNodeAlreadyExists:
        openList.append(child)

problem.displayPath(bestPathSoFar.path)
return makePathFromPoints(bestPathSoFar.path)

```

Код эвристики:

```

def manhetanDistanceHeuristics(point1:Point,problem,*args,**kwargs)
    point2=problem.goal
    return manhetanDistanceBetweenPoints(point1, point2)

```

Результат:



A* базується все на тому ж BFS, однак з одним дуже суттєвим покращенням. В A Star наступна в ітерації вершина обирається не випадково, а опираючись на оцінку цієї вершини (оцінка складається з евристики та відстані до цієї вершини). Таким чином на кожному кроці ми обираємо найбільш вірогідно правильний крок, що суттєво прискорює знаходження шляху.

ВИСНОВОК

Висновок щодо кожного алгоритму окремо я напиав в кінці підрозділів про кожний алгоритм.