AVL-tree를 사용한 개인형 이동수단 정렬과 검색

자료구조론(AICS201) Team07

김근호 2022270122, 인공지능사이버보안학과
김민세 2022270121, 인공지능사이버보안학과
명하준 2022270033, 응용수리과학부 데이터계산과학전공
박민찬 2022270118, 인공지능사이버보안학과
오태검 202027134, 인공지능사이버보안학과

1 Development Process

1.1 프로젝트 주제 제의; 교내 지쿠터 최단거리 회수(week 04)

개인형 이동수단 수집을 프로젝트의 목표로 제시하였을 때, 일반적인 리스트형으로 데이터셋이 주어진 상황을 가정하여, Huffman, Kruscal, Prim 등의 알고리즘으로 최소비용신장트리를 생성함을 목표로 하였다. 트리 구조의 특징상, 이진 탐색이 가능하여 시간복잡도의 측면에서 이득을 가져올 수 있음을 프로젝트의 의의로 지녔다.

1.1.1 자료구조가 사용될 상황이 제시되어야 하지만 최단거리는 알고리즘 중심 주제이다.

자료구조에 집중한 개선방식을 택했다. 개인형 이동수단에 포함되어 있는 메타데이터를 바탕으로 그래프 생성을 목표로 하여 AVL-tree 기반 트리 생성으로 결정하였다.

1.1.2 알고리즘 솔루션이 잘못되었다. 최소비용신장트리는 이 문제와 맞지 않다.

교수님께서 제시한 알고리즘 솔루션으로는 TSP가(Traveling Salesman Problem) 있었다.

1.1.3 Agile 방법론이 적절한가? 이 주제에 매번 개선이 필요한 문제가 있는가?

주제 수정이 빈번하지 않음에 동의하여, 매주 할당량 배분과 진척도 관리에 용이한 MoSCoW 방법론을 채택하기로 하였다.

1.2 프로젝트 주제 수정 및 확립; AVL-tree를 사용한 개인형 이동수단 정렬과 검색

자료구조에 집중한 정렬 및 검색 시스템의 구축을 위해 트리 구조를 조사한 결과, Red-Black tree 구조가 처음 후보로 제안되었는데, 자료의 빈번한 변경을 가정하지 않았기에 조금 더 엄격한 균형 기준을 가진 AVL-tree가 적합하다고 판단되어 적용하기로 결정했다.

MoSCoW 방법론을 제안하면서 AVL-tree를 기반으로 한 데이터 CRUD의 구성을 최우선 목표로 하며 이를 기반으로 가상의 사용자가 사용, 검색할 수 있는 시스템을 구성하는 것을 최종 목표로 하였다.

1.3 Must Have

1.3.1 Data Creation - 박민찬(data_set)

각 개인형 이동수단의 위치와 배터리 잔량, 고장 여부 등을 포함하는 데이터셋을 작성하였고, 개인형 이동수단이 특정 장소에 밀집되어 있는 특징을 적용하여 장소를 기준으로 구별할 수 있게끔 하였다.

1.3.2 AVL-tree based Sorting - 김민세(datastructure-library)

```
class LocationAVLNode:
    def __init__(self, inputData):
        self.locationId = inputData['location_id']
        self.locationName = inputData['name']
        self.location = {
            "latitude": inputData['latitude'],
            "longitude": inputData['longitude']
        }
        self.scooters = inputData['scooters']
        self.left = None
        self.right = None
        self.height = 1

class LocationAVLTree:
    def __init__(self):
        self.root = None

    def insert(self, location):
        self.root = self._insert(self.root, location)

    def _insert(self, node, location):
        if node is None:
            node = LocationAVLNode(location)
        elif location['location_id'] < node.locationId:
            node.left = self._insert(node.left, location)
            if self._height(node.left) - self._height(node.right) == 2:
                if location['location_id'] < node.left.locationId:
                    node = self._rotate_right(node)
                else:
                    node = self._rotate_left_right(node)
        elif location['location_id'] > node.locationId:
            node.right = self._insert(node.right, location)
            if self._height(node.right) - self._height(node.left) == 2:
                if location['location_id'] > node.right.locationId:
                    node = self._rotate_left(node)
                else:
                    node = self._rotate_right_left(node)

        node.height = max(self._height(node.left), self._height(node.right)) + 1
        return node
```

노드를 생성할 때, 데이터셋으로부터 장소와(location_id) 위·경도(location)를 받아오고, BF를(Balance Factor) 저장하기 위한 변수를 생성한다. 노드를 삽입할 때는, BF를 계산하여 2가 존재함을 기준으로 분기점을 생성하여 회전을(LL, LR, RL, RR) 시행한다.

1.3.3 AVL-tree Rotation- 김민세(datastructure-library)

```
def _get_balance(self, node):
    if node is None:
        return 0
    return self._height(node.left) - self._height(node.right)

def _height(self, node):
    if node is None:
        return 0
    return node.height

def _rotate_right(self, node):
    left_child = node.left
    left_grandchild = left_child.right
    left_child.right = node
    node.left = left_grandchild
    node.height = 1 + max(self._height(node.left), self._height(node.right))
    left_child.height = 1 + max(self._height(left_child.left), self._height(left_child.right))
    return left_child

def _rotate_left(self, node):
    right_child = node.right
    right_grandchild = right_child.left
    right_child.left = node
    node.right = right_grandchild
    node.height = 1 + max(self._height(node.left), self._height(node.right))
    right_child.height = 1 + max(self._height(right_child.left), self._height(right_child.right))
    return right_child

def _rotate_left_right(self, node):
    node.left = self._rotate_left(node.left)
    return self._rotate_right(node)

def _rotate_right_left(self, node):
    node.right = self._rotate_right(node.right)
    return self._rotate_left(node)
```

BF는 좌측 노드의 높이에서 우측 노드의 높이의 차로 계산 하였으며, 크기가 더 큰 쪽에 높이를 1만큼 증가함으로 회전하였다.

1.3.4 AVL-tree based Node Searching - 김민세(datastructure-library)

```
def search(self, locationId):
    return self._search(self.root, locationId)

def _search(self, node, locationId):
    if node is None:
        return None
    if node.locationId == locationId:
        return node
    elif locationId < node.locationId:
        return self._search(node.left, locationId)
    else:
        return self._search(node.right, locationId)
```

1.3.5 AVL-tree Based Deletion - 김민세(datastructure-library)

```
def delete(self, locationId):
    self.root = self._delete(self.root, locationId)

def _delete(self, node, locationId):
    if node is None:
        return None

    if locationId < node.locationId:
        node.left = self._delete(node.left, locationId)
    elif locationId > node.locationId:
        node.right = self._delete(node.right, locationId)
    else:
        if node.left is None:
            return node.right
        elif node.right is None:
            return node.left
        else:
            successor = self._find_minimum(node.right)
            node.locationId = successor.locationId
            node.locationName = successor.locationName
            node.location = successor.location
            node.scooters = successor.scooters
            node.right = self._delete(node.right, successor.locationId)

    node.height = max(self._height(node.left), self._height(node.right)) + 1

    balance = self._get_balance(node)
    if balance > 1:
        if self._get_balance(node.left) >= 0:
            return self._rotate_right(node)
        else:
            node.left = self._rotate_left(node.left)
            return self._rotate_right(node)
    elif balance < -1:
        if self._get_balance(node.right) <= 0:
            return self._rotate_left(node)
        else:
            node.right = self._rotate_right(node.right)
            return self._rotate_left(node)

    return node
```

1.4 Should Have

1.4.1 In Order of Nearest Location - 김민세(datastructure-library)

```
def _distance(self, location1, location2):
    if location1 is None or location2 is None:
        return float('inf')
    return haversine((location1['latitude'], location1['longitude']), (location2['latitude'], location2['longitude']), unit="m")

def find_nearest_location(self, user_location):
    nearest_location = None
    min_distance = float('inf')

    stack = []
    curr = self.root
    while stack or curr:
        if curr:
            stack.append(curr)
            curr = curr.left
        else:
            curr = stack.pop()
            distance = self._distance(curr.location, user_location)
            if distance < min_distance:
                min_distance = distance
                nearest_location = curr
            curr = curr.right

    return nearest_location, min_distance
```

사용자로부터 최단거리의 노드를 찾을 땐, Stack 구조를 적용하였다.

1.4.2 In Order of Battery Capacity - 김민세(datastructure-library)

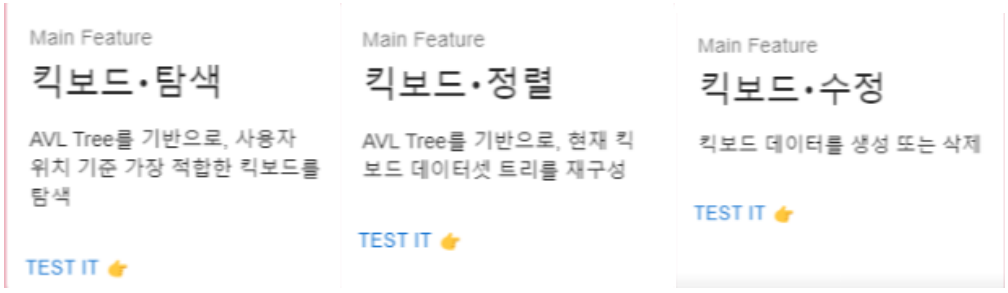
```
def heapify(scooters, n, i):
    largest = i
    l = 2*i + 1
    r = 2*i + 2

    if l < n and scooters[l]["battery"] > scooters[largest]["battery"]:
        largest = l

    if r < n and scooters[r]["battery"] > scooters[largest]["battery"]:
        largest = r

    if largest != i:
        scooters[i], scooters[largest] = scooters[largest], scooters[i]
        heapify(scooters, n, largest)

def build_heap(scooters):
    n = len(scooters)
    for i in range(n//2 - 1, -1, -1):
        heapify(scooters, n, i)

    return scooters
```

배터리 잔량을 기준으로 노드를 찾을 땐, Heap 구조를 적용하였다.
자료형에 차이가 있어 서로 다른 구조를 적용한 것은 아니지만 다양한 구조를 시도해 봄으로 Could Have의 More Various Data Structures Applying을 달성할 수 있었다.

1.5 Could Have

1.5.1 Data Manipulation Sim. (Webapp Service)

사용자가 AVL-tree를 활용하여 데이터를 조작하는 시뮬레이션을 제공하는 웹 앱 서비스를 개발하여 사용자는 데이터를 추가, 삭제하는 등의 작업을 수행할 수 있으며, AVL-tree의 균형을 유지하면서 데이터 조작 결과를 확인할 수 있다.



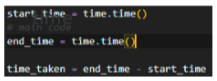
1.5.2 Viewing Tree (Webapp Service)

AVL-tree의 구조와 데이터를 시각적으로 보여주는 웹 앱 서비스를 개발하여 사용자는 AVL-tree를 시각적으로 탐색하고, 트리의 노드들과 연결 관계를 확인할 수 있다. 이를 통해 AVL-tree의 데이터의 정렬 상태를 시각적으로 확인할 수 있다.

- 13 locations
- 10~20 scooters for each location
- Random battery status (0~100)
- Random repair status (true or false)

- 5 Tests
- Target location: 조치원소방서, 학군단, 농심국제관, 중앙광장, 신정문주차장
- Nearest location (Avg): 0.000056 sec
- Best scooter (Avg): 0.00018 sec

- How to measure



1.5.3 More Various Data Structures Applying

1.4.1과 1.4.2로 달성하였다.

2 How to Access Our Data Structure Library

2.1 Setup

- Clone This Repository.

git clone https://github.com/ku-aics201-23s-team07/project-api

- Create “library” folder

- Clone This Repository in “library” folder

git clone https://github.com/ku-aics201-23s-team07/datastructure-library

- Recommends over Python 3.10
- Install dependencies in requirements.txt

2.2 Run

- go to local folder
- python3 main.py or python main.py

3 프로젝트 목표와 달성 여부

3.1 목표

정형적이지 않은 개인형 이동수단의 위치 데이터를 정렬하고, 검색 시, 보장된 시간복잡도를 가지기 위하여 AVL-tree를 사용하였다. 외에도, 팀이 여러 자료구조에 대해 균등한 이해를 가지기 위해 Stack과 Heap 구조를 사용였다.

3.2 달성 여부

양적인 측면에서 안정적인 라이브러리라 주장할 수 있다. API와 데이터셋은 차일로써 분리되어 호출에 이상이 없고, 예로 데이터셋에 포함된 개인형 이동수단 데이터를 제공했을 때 성공적으로 정렬이 가능하였다.

질적인 측면에서 데이터셋에 13개의 위치정보와 각각의 위치정보 안에 10~20개의 스쿠터를 넣어 구성한뒤 가장가까운 위치, 가장 사용하기 좋은 스쿠터를 찾을때 각각 0.000056초, 0.00018초로 빠른속도를 내었다.

3.3 성공적인 역할 분담

3.3.1 발표조

- 김근호(2022270122, 인공지능사이버보안학과)

Language & Environment: C, C++, JAVA, Python / Windows, Debian, Redhat

발표자료 및 팀 문서 작성을 총괄하며 발표자로서 기여하였고, 팀 진척도를 관리하며 각 맴버에게 할당량을 분배하였다.

- 오태검(2020271341, 인공지능사이버보안학과)

Language & Environment: C, C++ / Windows

팀의 발표자이며, 팀 회의록을 작성하였고, 발표 플롯을 제공하였다.

3.3.2 개발조

- 김민세(2022270121, 인공지능사이버보안학과)

Language & Environment: C, C++, JS Python / Windows, Mac, Linux

프로젝트 라이브러리를(AVL-tree, Heap 구조) 개발하였으며, FastAPI를 이용한 벡엔드를 구현하였고, 데이터셋 프로토타입을 제작하였으며 PPT의 내용검수 및 수정을 담당하였다.

- 명하준(2022270033, 응용수리과학부 데이터계산과학전공)

Language & Environment: C, C++, JS Python / Windows, Mac, Linux

React로 프론트앤드를 구현하였다.

- 박민찬

Language & Environment: C++, JS Python / Mac

데이터셋을 제작하였다.