자료구조 과제 6 – 그래프/탐색 트리 보고서

20222663 권기영

**1. 균형을 이룬 이진 탐색 트리 만들기**

**<문제 내용>**

AVL 트리를 활용하여 이진 트리의 높이를 줄이고, 높이가 얼마나 줄어드는지 확인한다. 또한 탐색 기능이 잘 동작하는지 확인한다.

**<문제 해결 - python>**

**(1) AVL 트리 클래스 선언**

1) AVLNode 클래스를 만들어 노드 객체를 선언한다. 데이터, 왼쪽 자식, 오른쪽 자식, 높이 총 4개의 정보를 갖고 있는 객체이다.

2) AVLTree 클래스를 만든다. 메소드는 다음과 같다.

getHeight : 노드의 높이를 반환하는 메소드. 높이 멤버 변수가 있지만, 멤버 변수를 사용할 때에는 항상 노드가 None인 경우에 대한 예외처리를 해줘야 하기 때문에 메소드 안에서 그 작업을 해주었다.

getDif : 노드의 왼쪽 오른쪽 높이 차 반환. 균형이 맞는지 아닌지 보기 위함이다.

updateHeight : 노드의 높이를 새로 업데이트 해준다. 트리의 균형을 맞춘 뒤에는 높이가 달라질 수 있기 때문에 업데이트를 해주어야 한다.

Insert : 트리에 노드 삽입. 트리에 있는 데이터와 비교를 하면서 내려가고, 더이상 내려갈 곳이 없으면 삽입한다.

balance : 트리의 균형을 맞추는 함수. getDif 함수가 왼쪽 높이 – 오른쪽 높이 이기 때문에 차이가 2 이상이면 LR, LL 회전을 수행하고, -2 이하면 RL, RR 회전을 수행한다. 자식 노드에서 getDif 함수를 한번 더 수행해서 이중 회전(LR, RL)인지, 한 번만 회전(RR, LL) 인지 판별한다.

left\_rotate : 왼쪽 회전을 수행한다. (RR rotate)

right\_rotate : 오른쪽 회전을 수행한다. (LL rotate)

getTreeHeight : 트리 전체의 높이를 반환한다.

print\_word : 트리를 중위 순회 하면서 모든 노드의 단어, 뜻, 높이를 출력한다.

search : 단어를 찾아 뜻을 출력한다. 트리의 데이터와 단어를 비교하면서 밑으로 내려가고, 단어와 데이터가 같으면 데이터의 뜻을 출력함.

**(2) 단어 텍스트파일에서 불러와서 트리에 삽입**

1) :을 토큰으로 : 전까지는 단어, :의 인덱스 + 3 부터는 뜻으로 저장한다.

2) 이렇게 구한 단어와 뜻을 배열 형태로 해서, [단어, 뜻]의 데이터를 갖는 노드를 트리에 삽입한다.

**(3) 단어 탐색**

1) 사용자에게서 단어를 입력받고, 그 뜻을 출력해준다. 아무것도 입력하지 않고 엔터를 누르면 종료한다.

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

AVL 트리를 활용해 사전을 구현했을 때 높이는 19가 나왔고, 그냥 이진 탐색 트리로 했을 때는 높이가 37이 나왔다. 약 2배가량 줄어들었다.

**2. 가장 낮은 높이의 탐색트리 만들기**

**<문제 내용>**

가장 높이가 낮은 트리를 만든다(높이 1, 2 제외). 탐색기능이 잘 동작하여야 한다. 또한 트리의 우수성을 설명한다.

**<문제 해결 - python>**

**(1) 핵심 아이디어**

AVL 트리를 활용하되, 트리의 구조를 바꾸는 것이 아닌 데이터의 숫자를 줄이는 방법을 사용하였다. 알파벳이 a~z 까지 총 26개이기 때문에, 알파벳에 따라 26개의 트리를 따로 만들게 되면 트리 각각에 대한 데이터 수가 적어지고, 높이가 줄어들 것이다.

**(2) 트리 만들기**

1) 텍스트 파일에서 미리 [단어, 뜻]을 전부 받아서, temp\_arr 배열에 저장하고, 단어에 따라 미리 정렬해 놓는다. 이 때, 대문자 소문자가 섞여 있으므로 lower 함수를 활용하여 모두 소문자라고 생각하고 정렬해 대소문자 구분을 없앤다.

2) 26개의 트리 객체 배열인 dicts를 선언하고, 알파벳 a로 시작하는 단어 노드는 dicts[0]에 저장하고 ~~ z로 시작하는 단어 노드는 dicts[25]에 저장한다.

**(3) 높이 및 탐색**

1) 트리의 높이는, 26개의 높이를 전부 구한 뒤 그 중 최댓값이 이 자료구조의 높이이다. 알파벳에 따라 트리를 고르는데 1번의 연산을 더 하기 때문에 실질적으로 이 자료구조의 높이는 높이 + 1 이다.

2) 사용자가 단어를 입력하면, 그 첫 번째 문자를 인덱스로 변경해서 트리 배열에서 맞는 트리를 찾아 탐색을 진행한다.

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

그냥 AVL트리와 비교했을 때 높이는 19에서 12로 7만큼 줄었고, 탐색도 잘 되는 것을 확인했다.

**(4) 이 트리의 우수성**

이 트리는(트리라고 할 수 있는지는 잘 모르겠음) 현재는 26개로 나눠져 있지만, 더 잘게 나눌 수 있고, 잘게 나눌 수록 높이가 줄어들 것이다. 잘게 나눈다고 하더라도 트리를 찾아가는 연산은 1번이기 때문에 시간적 손실이 없는 것이 장점이다.

그러나 이 자료구조가 과연 트리라고 할 수 있을지가 의문이고, 또한 트리의 구조를 바꾸지 않았기 때문에 별로 우수한 구조라고 생각이 들지 않았다.

**3. 길 찾기**

**<문제 내용>**

미로의 가장 짧은 길의 경로와 그 거리가 얼마인지를 화면에 표시한다.

**<문제 해결 - python>**

**(1) 텍스트 파일에서 미로 저장하기**

1) 미로 텍스트 파일은 이클래스에 올려진 maze1 – graph.txt 파일을 사용하였다.

2) 텍스트 파일에서 한 줄 씩 읽어 문자열로 저장한 뒤에, 문자열에서 벽과 관련된 +, -, |는 모두 O로 통일하여 저장하고, 길인 공백은 그대로 저장했다.

3) 그 문자열을 split 함수로 문자 하나씩 쪼개서 한 행을 열 만큼의 개수를 가진 배열 형태로 저장했다.

4) 이렇게 모든 행에 대해 수행하여 2차원 배열 형태로 미로를 저장하였다.

**(2) 너비 우선 탐색(BFS)을 통한 미로 길 찾기**

1) 방문 여부를 판단하는 배열 visited와 이전 위치를 저장하는 배열 prev를 선언한다. 크기는 미로 배열과 같다. 방문 여부는 갔던 길을 되돌아 가지 않기 위해서이고, 이전 위치를 저장하는 것은 막히면 되돌아가야 하기 때문이다.

2) 동서남북으로 차례대로 이동하면서 길 여부를 판단한다. 방문하지 않았고, 공백이라면, 길이다. 만약 길이라면 큐에 그 위치를 넣고, 방문여부를 True로 바꾸고 그 이전 위치를 prev에 저장한다.

3) BFS는 가까운 곳 부터 미리 방문하기 위해 큐를 활용하여 구현하기 때문에 목적지에 도달하게 되면 그 경로가 최단경로가 된다. 따라서 목적지에 도달하면 break 하고, prev에 저장된 이전 위치들을 따라가면서 경로를 표시한다.

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

최단 경로와 그 길이를 표시하였다.

**4. 지하철 짧은 길 찾기**

**<문제 내용>**

서울시 지하철 역 30 ~ 50개를 골라서 지하철 노선도를 구성하고, 이 구성을 텍스트 파일에 저장한다. 저장된 텍스트 파일에서 정보를 가져와 프로그램 안에서 새로 노선도를 구성한다. 텍스트 파일이 변경되면 소스코드에도 즉각 반영이 된다. 출발역과 도착역을 입력하면 최단거리가 얼마인지 화면에 표시한다. 환승은 고려하지 않는데, 따라서 환승 시간을 알고 있다면 어떻게 반영할 수 있을지 설명한다.

**<문제 해결 - python>**

**(1) 지하철 노선도 구성 및 텍스트 파일에 저장**

텍스트, 지도, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

1) 지하철 선정은 위 사진에서 빨간 선 안에 있는 지하철 역 전부를 선정하였다. 호선은 1~9호선에서 8호선을 제외한 모든 호선 + 중앙선 이다. 총 46개의 역을 선정하였다.

2) 이 지하철 역들을 텍스트 파일에 저장하는 방식은 다음과 같다.

‘번호’ ‘역 이름’ – ‘인접 역 번호’ ‘거리’ / ‘인접 역 번호’ ‘거리’ …..

예) 신도림역

0 신도림 – 9 1.2 / 1 1.0 / 10 1.8

이런 식으로 46개의 줄으로 텍스트 파일을 구성하였다.

3) 거리를 구할 때, 1호선과 9호선, 중앙선의 거리 정보는 이클래스에 올려진 엑셀파일에 없었기 때문에 따로 구글링을 해서 찾아 집어넣었다.

<https://ko.wikipedia.org/wiki/%EC%84%9C%EC%9A%B8_%EC%A7%80%ED%95%98%EC%B2%A0_1%ED%98%B8%EC%84%A0>

<https://ko.wikipedia.org/wiki/%EC%84%9C%EC%9A%B8_%EC%A7%80%ED%95%98%EC%B2%A0_9%ED%98%B8%EC%84%A0>

<https://ko.wikipedia.org/wiki/%EC%88%98%EB%8F%84%EA%B6%8C_%EC%A0%84%EC%B2%A0_%EC%A4%91%EC%95%99%EC%84%A0>

**(2) 노드와 그래프 클래스 선언**

1) 지하철 역에 해당하는 노드는 역 번호와 인접 역 정보(인접 역 번호, 인접 역 까지의 거리)에 대한 멤버 변수 2개로 이루어져 있다.(역 번호가 아니라 그냥 이름으로 저장했어도 됐는데, 처음에 코드를 짤 때 이름이 아니라 인덱스를 활용하면 더 쉬울 것이라고 잘못 판단함.)

2) SubwayGraph 클래스는 nodes 멤버변수 하나로 이루어져 있다. add\_node 함수를 활용하여 만든 노드를 Graph에 집어 넣는 것이다.

3) 따라서 지하철 역 하나에 해당하는 노드를 선언한 뒤, add\_adjacent\_node 함수를 통해 인접 역의 정보를 노드에 저장하고, 그렇게 모든 정보가 들어있는 노드를 add\_node 함수를 활용하여 SubwayGraph에 저장한다.

4) 데이터는 딕셔너리 형태로 저장하였다.

**(3) 텍스트 파일에서 데이터를 읽어 그래프 구성하기**

1) 일단 역을 인덱스로 접근하기 때문에 숫자로 역의 정보를 계속 사용하게 되는데, 출력을 할 때에는 다시 역의 이름을 출력해야 하므로 subway\_name 배열에 순서대로 역의 이름을 저장해놓는다.

2) 한 줄에 있는 정보는 지하철 역 번호, 지하철 역 이름, 인접 역 번호, 인접 역과의 거리 총 4개이고, 1)에서 지하철 역 번호와 지하철 역 이름은 이미 사용했으니 삭제하고, 인접 역 번호와 거리 2개의 정보만을 딕셔너리 형태로 노드에 저장한다. 저장한 정보는 삭제하는 형식으로 한 줄이 None이될 때 까지 반복한다.

3) 그렇게 만들어진 46개의 노드들을 그래프 객체에 전부 삽입하여 그래프 그리기를 완료한다.

**(4) 다익스트라 알고리즘으로 최단경로 구하기**

1) 시작 지점, 도착 지점, 그래프 총 3개를 인수로 받는 함수 Dijkstra 함수를 구현한다.

2) 시작 지점으로 부터 거리 값을 저장하기 위해 distances 배열을 선언하고, 최단 경로를 빼내기 위해서 우선순위 힙이 구현된 heapq 라이브러리를 사용한다.

3) 힙에 현재 거리, 현재 목적지, 경로 총 3개를 push한다.

4) 만약에 힙에서 뽑은 거리가 기존 거리보다 길다면 continue

5) 힙에서 뽑은 거리가 기존 거리보다 짧다면, 거리를 새로 갱신해주고, 경로도 업데이트 해준다.

6) 도착 지점에 도달하면 거리와 경로를 반환한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

0은 신도림역 이고, 17은 신용산 이다. 출발 지점과 도착 지점을 입력했을 때, 최단 거리와 그 경로를 올바르게 출력하는 것을 확인할 수 있다.

**(5) 환승 시간 반영**

환승 시간을 반영하기 위해서는 일단 최단 시간을 구하는 것으로 바꾸어야 한다. 위의 코드는 최단 거리를 구하는 것인데, 어짜피 원리는 같고 데이터만 바꾸면 되므로 이는 문제가 없다.

환승 시간이 반영되는 순간, 최단 시간의 길이 최단 시간이 아니게 될 수 있다. 왜냐하면 지하철은 내가 도착했을 때 바로 출발하는 것이 아니라, 운행 시간이 정해져 있기 때문이다. 그래서 역 사이의 시간 = 그냥 지하철 타고 가는 시간 + 환승 시간인데, 이 때 환승 시간은 내가 지하철 타는 곳에 도착하는 시간이 아니라, 도착한 뒤 지하철을 타는 시간 까지 이다. 따라서 지하철 역에 내린 뒤 환승하는 도중에 지하철이 출발하였다면, 그 다음 지하철이 오는 시간 까지가 환승 시간이라는 것이다.

그래서 지하철 역의 전철 운행 시간을 데이터로 가지고 있어야 할 것이고, 환승할 때에는 인접한 역까지의 시간 = 지하철 타는 시간 + 환승 시간으로 바꾸어서 계산하고, 환승 역까지 가는 시간에 환승역의 지하철이 출발 하는지 안 하는지에 대한 조건식도 있어야 할 것이다.