숭실대학교 Soongsil University

알고리즘 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
   1.1. 나는 동료의 보고서를 베끼지 않았습니다.
   1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)
3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.
4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	알고리즘(나) 2021
과제명	과제 4
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20163340 강원경 (출석번호 201번)
제출일	2021년 10월 14일

차 례

1장. 과제 동기/목적

이번 과제는 알고리즘 과목 수업 시간중에 배운 Dynamic Programming 분류에 속한 Floyd Algorithm과 최단경로의 출력 알고리즘(Print Shortest Path Algorithm)을 이용해 주어진 문제를 해결하는 과제이다. 수업 시간에 한 번 봤던 내용인 만큼 다시 한 번 배운 이론적인 내용을 떠올리고 정리하며 과제를 수행하는 것을 1차적인 목표로 삼았다.

매번 과제를 하면서 느끼고 보고서를 작성할 때 언급하지만, 수업 시간에 이론으로 배우는 것과 그 내용을 바탕으로 직접 코드를 작성하는 것은 별개의 영역이다. 배운 것을 완전히 내 지식으로 만드는 데에는 직접 그것을 적용해보는 것만큼 좋은 활동이 없고, 그 활동중 하나가 이런 과제를 수행하는 것이라고 생각한다. 그렇기에 이번 과제를 통해서 직접 플로이드 알고리즘을 적용해보고 코드를 작성해봄으로써 나중에 내가 어떤 프로그램을 작성할 때 배운 내용을 바탕으로 적절한 코드를 작성할 수 있는 능력을 기르는 것이 두 번째 목표였다.

또한 최적화된 이진 탐색 트리를 생성하는 문제도 주어졌는데, 이렇게 자료구조를 본인이 직접 설계하는 것 또한 앞으로 만나게 될 학업/실무과정에서의 문제들을 해결하는데 가장 기초가 되는 부분이라 생각한다. 그렇기에 직접 과제를 수행하면서 어떤 방식이나 알고리즘으로 최적화된 이진 탐색 트리를 구현하는 것이 좋을지 고민해보고 직접 구현하는 것을 목표로 하였다.

2장. 설계/구현 아이디어

1번과 3번 문제는 플로이드 알고리즘을 이용해 문제를 해결하였고, 2번과 4번 문제는 최단경로 출력 알고리즘을 이용해 문제를 해결하였다. 그리고 5번 문제는 가중치가 동일한 데이터들을 어떤 방식으로 설계하는 것이 가장 최적화된 트리를 구현하는지 수업시간에 배운 내용을 바탕으로 본인이 직접 설계하는 과정이 필요했다.

우선 1~4번 문제에서 가장 기본이 되는 그래프들에 대한 데이터는 const를 사용하여 전역변수로 두었다. 동적할당을 통해 지역변수로 그래프를 선언하고 그 선언된 메모리의 내용을 함수의 인자로 넘겨주는 방식도 있겠으나, 본인의 실력이 부족하여 이 부분은 지역변수가 아닌 전역변수의 힘을 빌리기로 하였다.

5번 문제에서는 가중치가 있는 데이터들을 탐색하는데 최적화된 최적 이진 탐색 트리를 설계하는 것이 주요한 부분이었다. 이진 탐색 트리를 기반으로 두기 때문에 데이터 값에 대한 대소비교도 필요했고, 가중치가 있기 때문에 어떤 데이터를 루트 노드에 가깝게 두어야 더 최적화된 탐색이 가능할지에 대한 부분도 필요했다.

이러한 부분들을 모두 상기하고 정리하면서 이번 과제를 수행했다.

이번 과제는 macOS 버전 11.6 (Big Sur)에서 Xcode 버전 12.5.1 버전을 이용하여 수행하였다.

3장. 문제 1 수행 결과

문제 1번은 플로이드 알고리즘을 이용하는 것이 첫 번째 조건이었고, 각각의 수행 단계를 출력하라는 두 번째 조건이 있었다. 따라서 주요 문제 해결 알고리즘에 플로이드 알고리즘을 사용하였고, 중간에 거치게 되는 정점이 바뀌게 될 때를 수행 단계가 바뀌는 시점으로 보고 그 시점에서의 행렬 D와 P를 출력하였다.

수행 결과는 아래 사진과 같다.

```
-- step 1 (정점 v1을 거칠 때) --
행렬 D :
INF   4 INF INF INF  10 INF
  3   7 INF  18 INF  13 INF
INF   6 INF INF INF INF INF
INF   5  15 INF   2  19   5
INF INF  12   1 INF INF INF
INF INF INF INF INF INF  10
INF INF INF   8 INF INF INF

행렬 P :
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
-- step 2 (정점 v2을 거칠 때) --
행렬 D :
  7   4 INF  22 INF  10 INF
  3   7 INF  18 INF  13 INF
  9   6 INF  24 INF  19 INF
  8   5  15  23   2  18   5
INF INF  12   1 INF INF INF
INF INF INF INF INF INF  10
INF INF INF   8 INF INF INF

행렬 P :
1 0 0 1 0 0 0
0 0 0 0 0 0 0
1 0 0 1 0 1 0
1 0 0 1 0 1 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
-- step 3 (정점 v3을 거칠 때) --
행렬 D :
  7   4 INF  22 INF  10 INF
  3   7 INF  18 INF  13 INF
  9   6 INF  24 INF  19 INF
  8   5  15  23   2  18   5
 21  18  12   1 INF  31 INF
INF INF INF INF INF INF  10
INF INF INF   8 INF INF INF

행렬 P :
1 0 0 1 0 0 0
0 0 0 0 0 0 0
1 0 0 1 0 1 0
1 0 0 1 0 1 0
2 2 0 0 0 2 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
-- step 4 (정점 v4을 거칠 때) --
행렬 D :
  7   4  37  22  24  10  27
  3   7  33  18  20  13  23
  9   6  39  24  26  19  29
  8   5  15  23   2  18   5
  9   6  12   1   3  19   6
INF INF INF INF INF INF  10
 16  13  23   8  10  26  13

행렬 P :
1 0 3 1 3 0 3
0 0 3 0 3 0 3
1 0 3 1 3 1 3
1 0 0 1 0 1 0
3 3 0 0 3 3 3
0 0 0 0 0 0 0
3 3 3 0 3 3 3
```

```
-- step 5 (정점 v5을 거칠 때) --
행렬 D :
  7   4  36  22  24  10  27
  3   7  32  18  20  13  23
  9   6  38  24  26  19  29
  8   5  14   3   2  18   5
  9   6  12   1   3  19   6
INF INF INF INF INF INF  10
 16  13  22   8  10  26  13

행렬 P :
1 0 4 1 3 0 3
0 0 4 0 3 0 3
1 0 4 1 3 1 3
1 0 4 4 0 1 0
3 3 0 0 3 3 3
0 0 0 0 0 0 0
3 3 4 0 3 3 3
```

```
-- step 6 (정점 v6을 거칠 때) --
행렬 D :
  7   4  36  22  24  10  20
  3   7  32  18  20  13  23
  9   6  38  24  26  19  29
  8   5  14   3   2  18   5
  9   6  12   1   3  19   6
INF INF INF INF INF INF  10
 16  13  22   8  10  26  13

행렬 P :
1 0 4 1 3 0 5
0 0 4 0 3 0 3
1 0 4 1 3 1 3
1 0 4 4 0 1 0
3 3 0 0 3 3 3
0 0 0 0 0 0 0
3 3 4 0 3 3 3
```

```
-- step 7 (정점 v7을 거칠 때) --
행렬 D :
  7   4  36  22  24  10  20
  3   7  32  18  20  13  23
  9   6  38  24  26  19  29
  8   5  14   3   2  18   5
  9   6  12   1   3  19   6
 26  23  32  18  20  36  10
 16  13  22   8  10  26  13

행렬 P :
1 0 4 1 3 0 5
0 0 4 0 3 0 3
1 0 4 1 3 1 3
1 0 4 4 0 1 0
3 3 0 0 3 3 3
6 6 6 6 6 6 0
3 3 4 0 3 3 3
```

4장. 문제 2 수행 결과

2번 문제는 앞서 1번 문제에서 플로이드 알고리즘을 이용해 구한 행렬 p를 이용해 최단경로를 탐색하는 step과 그 최종 경로를 보이는 문제이다.

따라서, 이번 코드에서는 앞선 1번 문제의 코드에 다음과 같은 함수를 추가하였다.

```
52  // 최단 경로를 출력하는 함수.
53  // p: 최단거리 경로상 경유 정점이 저장된 배열, q: 간선 시작점, r: 간선 종점
54  // res: 경유하게 되는 모든 정점이 저장된 배열, i: 경유하게 되는 정점들의 갯수가 저장된 변수를 가리키는 포인터 변수.
55  void path(int* p, int q, int r, int* res, int* i){
56      printf("path(%d,%d) = %d\n",q,r,*(p+q*7+r));
57      if(*(p+q*7+r)!=0){
58          path(p, q,*(p+q*7+r), res, i);
59          printf("%d번째 경유 정점 : v%d\n", *(i)+1, *(p+q*7+r)+1);
60          *(res+*(i))=*(p+q*7+r)+1;
61          *(i)+=1;
62          path(p, *(p+q*7+r), r, res, i);
63      }
64  }
```

이 함수를 이용해 수행 결과를 출력하면 다음과 같다.

```
path(6,2) = 4
path(6,4) = 3
path(6,3) = 0
1번째 경유 정점 : v4
path(3,4) = 0
2번째 경유 정점 : v5
path(4,2) = 0

최단경로 : v7 -> v4 -> v5 -> v3
Program ended with exit code: 0

All Output ⌃                Filter
```

5장. 문제 3 수행 결과

3번 문제는 1번 문제에서 정점의 번호를 바꾸고 다시 실행하여 행렬 D와 P의 최종 결과를 출력하는 문제이다. 중간 단계를 보여야 한다는 조건을 주지 않았으므로 문제 1의 floyd 함수 내부에 있었던 행렬 출력 코드 부분을 메인 함수로 옮겼다.

그래프의 모양은 시각화했을 때는 비슷해 보였지만, 행렬로 나타냈을 때는 많이 달라진 모양을 보였다. 이는 정점 번호의 변화로 인해 간선이 연결하고 있는 정점이 변화하면서 가중치의 위치가 변화했기 때문이라고 생각한다. 이로 인해 최종 출력 결과도 문제 1과는 꽤 달라진 양상을 보였다.

프로그램의 실행 결과는 아래와 같다.

```
-- 최종 수행 결과 --
행렬 D :
 36  20  18  32  23  26  10
 19   3   1  12   6   9   6
 18   2   3  14   5   8   5
 19  26  24  38   6   9  29
 13  20  18  32   7   3  23
 10  24  22  36   4   7  20
 26  10   8  22  13  16  13

행렬 P :
6 6 6 6 6 6 0
5 2 0 0 2 4 2
5 0 1 1 0 4 0
5 4 4 4 0 4 4
5 2 0 2 5 0 2
0 4 4 4 0 4 0
5 2 0 2 2 4 2
Program ended with exit code: 0
```

6장. 문제 4 수행 결과

4번 문제는 앞선 3번 문제에서 정점 번호를 바꾼 그래프의 최단 경로를 찾는 문제이다. 대부분의 코드는 문제 2의 코드와 동일하지만, 시작 정점번호와 끝 정점번호가 바뀌었기 때문에 그 부분을 수정 후 코드를 실행했다.

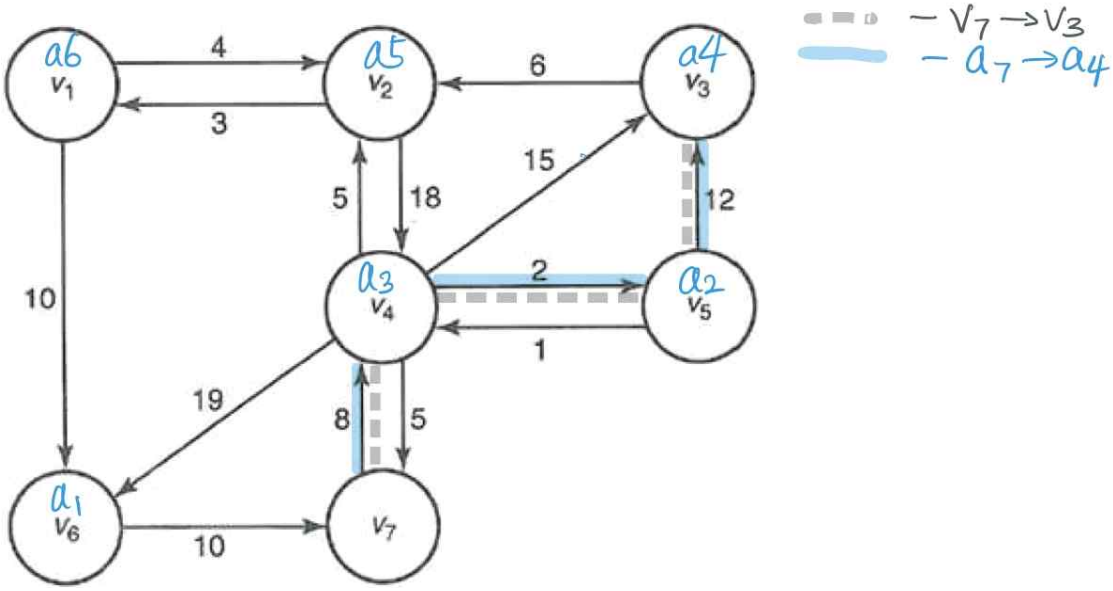
문제 4의 코드를 수행한 결과는 아래와 같다.

```
path(6,3) = 2
path(6,2) = 0
1번째 경유 정점 : a3
path(2,3) = 1
path(2,1) = 0
2번째 경유 정점 : a2
path(1,3) = 0

최단경로 : v7 -> a3 -> a2 -> a4
Program ended with exit code: 0
```

All Output ⌃ Filter

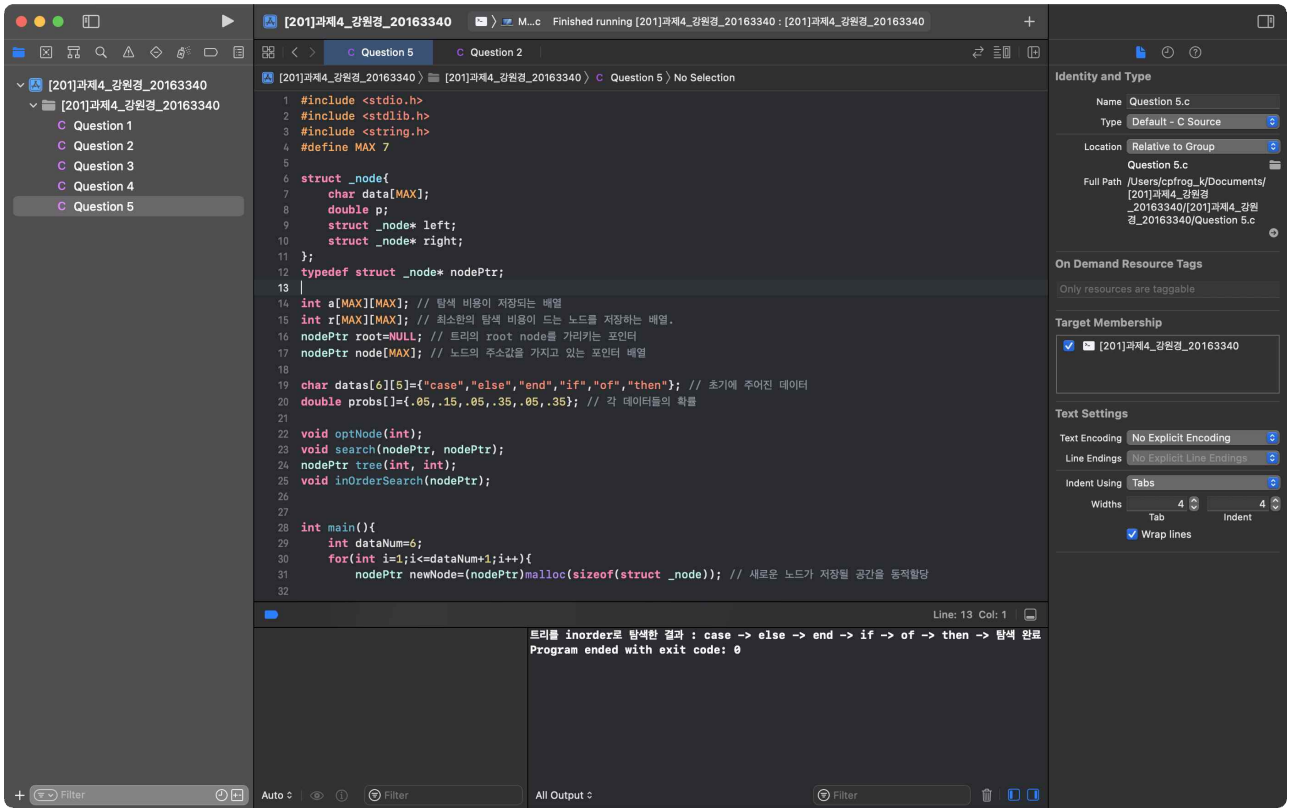
출력된 결과를 확인해보면 앞서 문제 2와 탐색 순서가 조금 변화했음을 알 수 있다. 여기서 탐색하는 경로도 바뀌었을까 하는 의문이 생겼다. 이 의문을 해결하기 위해 그래프 상에 문제 2에서 탐색한 경로와 문제 4에서의 탐색한 경로를 겹쳐서 확인해보았다.



위 그림에서 확인할 수 있듯, 경로를 탐색하는 순서만 바뀌었을 뿐 탐색하는 경로 자체는 변하지 않았음을 알 수 있다.

7장. 문제 5 수행 결과

문제 5번은 최적 이진 탐색 트리를 설계하는 문제이다. 이 문제의 경우 최저 탐색 비용이 저장되는 배열 a, 최저 탐색 비용의 노드가 저장되는 배열 r은 전역변수로 설정하고 문제를 해결하였다.



위 사진은 대략적인 코드와 수행 결과가 포함된 IDE 화면이다. 밑에 출력된 트리의 중위순회(Inorder) 탐색 결과를 바탕으로 생성된 트리를 그래프로 나타내면 다음과 같다.