알고리즘 과제(10)

201404377 진승언

1. 과제 목표 및 해결 방법

이번과제는 최소 신장 트리인 프림 알고리즘과 최단 경로 알고리즘 중 하나인 다익스트라 알고리즘을 구현하는 것이었다. 먼저 최소 신장 트리란 간선들이 가중치를 갖는 그래프에서 간선 가중치의 합이 가장 작은 트리를 말한다. 프림 알고리즘(Prim's algorithm)은 가중치가 있는 연결된 무향 그래프의 모든 꼭짓점을 포함하면서 각 변의 비용의 합이 최소가 되는 부분 그래프인 트리, 즉 최소 비용 트리를 찾는 알고리즘이다. 프림 알고리즘은 다음과 같은 순서대로 작동된다. 1. 그래프에서 하나의 꼭짓점을 선택하여 트리를 만든다. 2. 그래프의 모든 변이 들어 있는 집합을 만든다. 3. 모든 꼭짓점이 트리에 포함되어 있지 않은 동안 트리와 연결된 변 가운데 트리 속의 두 꼭짓점을 연결하지 않는 가장 가중치가 작은 변을 트리에 추가한다. 이렇게 하면 알고리즘이 종료되었을 때 만들어진 트리는 최소 신장 트리가 된다.

다익스트라 알고리즘은 입력 그래프 G = (V,E)에서 간선들의 가중치가 모두 0 이상인 경우의 최단 경로 알고리즘이다. 이 알고리즘은 프림 알고리즘과 원리가 거의 같다. 알고리즘의 골격이 거의 동일하다. 하지만 프림 알고리즘에서는 d[v]가 신장 트리에 연결하는 최소 비용을 위해 사용하는 반면, 다익스트라 알고리즘에서는 d[v]가 정점 r에서 정점 v에 이르는 최단 거리를 위해 사용된다.

2. 주요 부분 코드 설명(알고리즘 부분 코드 캡쳐)

minheap관련해서는 과거 과제를 이용하면 되었다.

<Prim>

```
graphMatrix1 = new int[num][num];
for (int i = 0; i < num; i++) {
    for (int j = 0; j < num; j++) {
        if (i == j) {
            graphMatrix1[i][j] = 0;
        } else {
            graphMatrix1[i][j] = INF;
        }
    }
}

for (int i = 0; i < G.length; i++) {
    for (int j = 0; j < G[i].length; j++) {
        graphMatrix1[i][j] = G[i][j];
    }
}
```

먼저 시작정점의 연결 비용을 0, 나머지 정점들의 연결 비용을 무한대로 초기화한다

```
for (int i = 0; i < vertexNum; i++) {
    key[i] = Integer.MAX_VALUE;
    isFinish[i] = false;
}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < vertexNum - 1; count++) {

    int u = minimunKey(key, isFinish);

    isFinish[u] = true;

    //키값과 부모 인덱스를 인접행렬로부터 뽑은 vertex로 저장해줌
    for (int v = 0; v < vertexNum; v++) {
        if (graph[u][v] != 0 && !isFinish[v] && graph[u][v] < key[v]) {

            parent[v] = u;
            key[v] = graph[u][v];
        }
    }
```

key값을 무한대로 초기화하고 isFinished는 false로 초기화한다. isFinished는 처리된 vertex인지 구분하는 역할을 한다. 그리고 처음 탐색할 0번 vertex key값을 무한대에서 0으로 바꿔주고 parent 즉 선행 vertex는 없다는 뜻으로 -1로해준다. minimumKey 메소드로 주변에 가장 가중치가 작은

vertex 번호를 갖고 오고 그 다음 vertex수만큼 for문을 돌려서 u,v vertex가 인접행렬에서 0이 아니고 v의 키 값보다작으면 v의 선행 vertex에 u를 저장하고 v에 u,v 간선의 가중치를 저장한다.

<Dijkstra>

```
//초기화
for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {
    minDist[vertexIndex] = Integer.MAX_VALUE;
    added[vertexIndex] = false;
}
```

Prim과 마찬가지로 초기화를 해준다.

```
//가장 짧은 경로 find
for (int i = 1; i < nVertices; i++) {

    int nearVertex = -1;
    int tmpMinDist = Integer.MAX_VALUE;
    for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {
        if (!added[vertexIndex] && minDist[vertexIndex] < tmpMinDist) {
            nearVertex = vertexIndex;
            tmpMinDist = minDist[vertexIndex];
        }
    }

    //한 vertex담음
    added[nearVertex] = true;

    //인접 vertex로 dist값 갱신
    for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++) {
        int edgeDistance = adjacentMatrix[nearVertex][vertexIndex];

        if (edgeDistance > 0 && ((tmpMinDist + edgeDistance) < minDist[vertexIndex])) {
            parents[vertexIndex] = nearVertex;
            minDist[vertexIndex] = tmpMinDist + edgeDistance;
        }
    }
}
```

vertex의 개수만큼 for문을 돌려서 가장 짧은 경로를 찾고 저장해준다. 해당 가장 짧은 경로의 거리, 즉 비용을 저장하고 해당 vertex를 저장한다.

그리고 해당 저장한 인접 vertex를 추가했다는 것을 알기위해서 added에 true로 저장한다. 그 후 for문을 돌려 인접한 vertex로 거리의 값을 갱신해주면된다.

```java
//아직 실행 안한 vertex경우
if (currentVertex == noParent) {
    return;
}
result += currentVertex + "";
System.out.print(currentVertex );
if(parents[currentVertex] != noParent) {
    System.out.print("<-");
    result += "<-";
}
printPath(parents[currentVertex], parents);
```

경로를 출력해주는 건데 현재 vertex에서 해당 vertex 오는데 저장된 vertex를 담은 parents를 이용해서 경로를 다 출력해주었다.

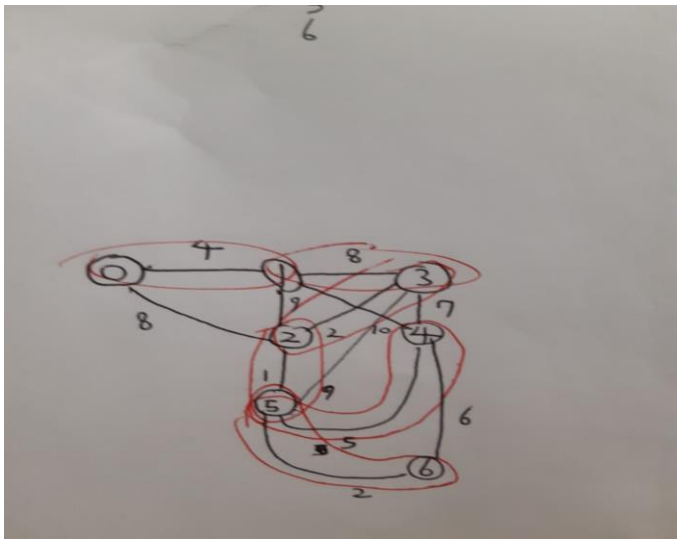
3. 결과

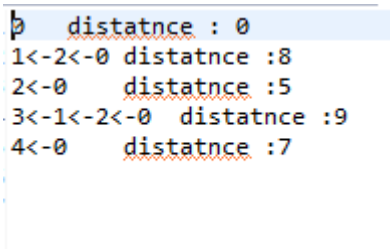
```
start | end | cost

0       1       4
3       2       2
1       3       8
5       4       5
2       5       1
5       6       2

```

Prim의 출력파일 결과 화면이다. 직접 그려서 확인해본 결과 맞게 나왔다.





Dijstra의 출력파일 결과 화면이다. 해당 vertex를 오는데 지나친 경로와 누적 distance를 보여준다.

Prim은 가장 크게 시간복잡도를 관여하는 부분은 힙에서 임의의 원소 값이 변해서 이를 반영하여 재조정하는데 O(logV)시간이 소요되고 이 수정은 최악의 경우 E번 일어날 수 있으므로 이와 관련된 비용은 총 O(ElogV)이다. 다익스트라도 마찬가지로 프림 알고리즘과 거의 같으므로 수행 시간은 동일하다. 즉 힙을 이용하면 O(ElogV)의 시간이 소요된다.