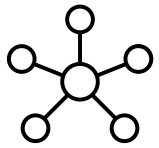

[제05주] 최소비용확장트리 (1)

충남대학교 컴퓨터공학과 알고리즘 04 분반
제출일: 2021-10-19
학번: 201701975, 이름: 구건모

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

목차

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

과제 설명

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

주요 이론 및 과제에서 해야할 일

이번 과제는 Weighted Undirected Graph를 사용자로부터 입력받아 최소비용확장트리(Minimum-Cost Spanning Tree) Tree를 찾는 것이 주요 내용이었습니다. Spanning Tree 중에서 edge의 weight의 합이 최소인 Tree를 Minimum-Cost Spanning Tree라고 하는데 이것을 찾는 알고리즘 중에 하나가 Kruskal's Algorithm 입니다.

5주차 과제에서는 최소비용확장트리를 찾는 알고리즘인 크루스칼 알고리즘을 구현하지는 않고,

Weighted Undirected Graph를 입출력하는 기능을 구현하고 그래프를 보여주는 것 까지 구현하는 것이 목표입니다.

따라서 Adjacency Matrix를 이용하여 Weighted Undirected Graph를 구현해야하므로 Edge에 cost가 있는 그래프를 구현하게 됩니다.

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

Class별 설명

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

Interface Graph

AL05_201701975_구건모 ▸ src ▸ graph ▸ Graph<E> ▸

```
1  package graph;
2
3  public interface Graph<E> {
4      public int numberOfVertices();
5
6      public int numberOfEdges();
7
8      public boolean vertexDoesExist(int aVertex);
9
0      public boolean edgeDoesExist(int aTailVertex, int aHeadVertex);
1
2      public boolean edgeDoesExist(E anEdge);
3
4      public boolean edgeIsValid(int aTailVertex, int aHeadVertex);
5
6      public boolean edgeIsValid(E anEdge);
7
8      public E edge(int aTailVertex, int aHeadVertex);
9
0      public boolean addEdge(E anEdge);
1
2  }
3
```

Inteface Graph

Graph 인터페이스는 이름과 같이 그래프를 다룰 때 사용할 공통적인 기능들을 제시합니다. 이후 구현할 그래프들에서 implements 하여 사용되게 됩니다.

WeightedEdge

AL05_201701975_구건모 ▸ src ▸ graph ▸ WeightedEdge ▸ WeightedEdge(int, int)

```java
package graph;

public class WeightedEdge extends Edge implements Comparable<WeightedEdge> {
    private static final int DEFAULT_WEIGHT = 0;
    private int _weight;

    public int compareTo(WeightedEdge otherEdge) {
        if (this.weight() < otherEdge.weight()) {
            return -1;

        } else if (this.weight() > otherEdge.weight()) {
            return +1;

        } else {
            return 0;
        }
    }

    public int weight() {
        return this._weight;

    }

    public void setWeight(int newWeight) {
        this._weight = newWeight;
    }

    public WeightedEdge(int givenTailVertex, int givenHeadVertex) {
        super(givenTailVertex, givenHeadVertex);
        this.setWeight(WeightedEdge.DEFAULT_WEIGHT);
    }

    public WeightedEdge(int givenTailVertex, int givenHeadVertex, int givenWeight) {
        super(givenTailVertex, givenHeadVertex);
        this.setWeight(givenWeight);
    }
}
```

class WeightedEdge

WeightedEdge는 이번 과제의 목표 WeightedUndirectedGraph를 구현하기 위해 필요한 클래스로 기존의 Edge를 extends 하여 구현되며, 기존의 Edge에서 weight 속성이 추가된 것으로 볼 수 있습니다. WeightedUndirectedGraph에서는 WeigtedEdge를 사용하여 그래프를 구성하게 됩니다.

UndirectedAdjacencyMatrixGraph

```java
package graph;

import app.AppView;

public class UndirectedAdjacencyMatrixGraph<E extends Edge> implements Graph<E> {
    private static final int EDGE_EXIST = 1;
    private static final int EDGE_NONE = 0;
    private int _numberOfVertices;
    private int _numberOfEdges;
    private int[][] _adjacency;

    protected UndirectedAdjacencyMatrixGraph() {

    }

    public UndirectedAdjacencyMatrixGraph(int givenNumberOfVertices) {
        this.setNumberOfVertices(givenNumberOfVertices);
        this.setNumberOfEdges(0);
        this.setAdjacency(new int[givenNumberOfVertices][givenNumberOfVertices]);
        for (int tailVertex = 0; tailVertex < this.numberOfVertices(); tailVertex++) {
            for (int headVertex = 0; headVertex < this.numberOfVertices(); headVertex++) {
                this.setAdjacencyOfEdgeAsNone(tailVertex, headVertex);
            }
        }
    }

    protected boolean adjacencyOfEdgeDoesExist(int tailVertex, int headVertex) {
        return (this.adjacencyOfEdge(tailVertex, headVertex) != UndirectedAdjacencyMatrixGraph.EDGE_NONE);
    }

    protected int adjacencyOfEdge(int tailVertex, int headVertex) {
        return this.adjacency()[tailVertex][headVertex];
    }

    protected void setAdjacencyOfEdgeAs(int tailVertex, int headVertex, int anAdjacencyOfEdge) {
        this.adjacency()[tailVertex][headVertex] = anAdjacencyOfEdge;
    }

    private void setAdjacencyOfEdgeAsExist(int tailVertex, int headVertex) {
        this.setAdjacencyOfEdgeAs(tailVertex, headVertex, UndirectedAdjacencyMatrixGraph.EDGE_EXIST);
    }

    protected void setAdjacencyOfEdgeAsNone(int tailVertex, int headVertex) {
        this.setAdjacencyOfEdgeAs(tailVertex, headVertex, UndirectedAdjacencyMatrixGraph.EDGE_NONE);
    }
```

코드가 길어 생략된 부분이 있습니다.

class UndirectedAdjacencyMatrixGraph

UndirectedAdjacecnyMatrixGrpah는 Undirected graph 를 Adjacency Matrix를 이용하여 구현한 것입니다. 최종적으로 구현해야할 WeightedUndirectedAdjacecnyMatrixGrpah 클래스에서 extends 하여 사용하게 됩니다.

WeightedUndirectedAdjacencyMatrixGraph

AL05_201701975_구건모 ▸ src ▸ graph ▸ WeightedUndirectedAdjacencyMatrixGraph<WE extends WeightedEdge> ▸

```
1  package graph;
2
3  public class WeightedUndirectedAdjacencyMatrixGraph<WE extends WeightedEdge> extends UndirectedAdjacencyMatrixGraph<WE>
4          implements SupplementForWeightedGraph<WE> {
5      private static final int WEIGHTED_EDGE_NONE = -1;
6
7      public WeightedUndirectedAdjacencyMatrixGraph(int givenNumberOfVertices) {
8          super();
9          this.setNumberOfVertices(givenNumberOfVertices);
10         this.setNumberOfEdges(0);
11
12         this.setAdjacency(new int[givenNumberOfVertices][givenNumberOfVertices]);
13         for (int tailVertex = 0; tailVertex < this.numberOfVertices(); tailVertex++) {
14             for (int headVertex = 0; headVertex < this.numberOfVertices(); headVertex++) {
15                 this.setWeightedOfEdgeAsNone(tailVertex, headVertex);
16             }
17         }
18     }
19
20     private void setWeightOfEdge(int aTailVertex, int aHeadVertex, int newWeight) {
21         this.adjacency()[aTailVertex][aHeadVertex] = newWeight;
22     }
23
24     private void setWeightedOfEdgeAsNone(int aTailVertex, int aHeadVertex) {
25         this.setWeightOfEdge(aTailVertex, aHeadVertex, WeightedUndirectedAdjacencyMatrixGraph.WEIGHTED_EDGE_NONE);
26     }
27
28     protected boolean adjacencyOfEdgeDoesExist(int aTailVertex, int aHeadVertex) {
29
30         return (this.adjacencyOfEdge(aTailVertex,
31                 aHeadVertex) != WeightedUndirectedAdjacencyMatrixGraph.WEIGHTED_EDGE_NONE);
32     }
33
34     public int weightOfEdge(WE anEdge) {
35         if (anEdge != null) {
36             return this.weightOfEdge(anEdge.tailVertex(), anEdge.headVertex());
37         }
38         return WeightedUndirectedAdjacencyMatrixGraph.WEIGHTED_EDGE_NONE;
39     }
40
41     public int weightOfEdge(int aTailVertex, int aHeadVertex) {
42         if (this.edgeDoesExist(aTailVertex, aHeadVertex)) {
43             return this.adjacencyOfEdge(aTailVertex, aHeadVertex);
44
45         }
```

코드가 길어 생략된 부분이 있습니다.

class WeightedUndirectedAdjacencyMatrixGraph

WeightedUndirectedAdjacencyMatrixGraph는 이번 과제에서 최종적으로 구현해야하는 목표로 UndirectedAdjacencyMatrixGraph와 다른점은 Edge를 WeightedEdge를 사용한다는 점과, EDGE_NONE에 대한 Constant가 기존에는 0 으로 설정했었는데, WeightedUndirectedAdjacencyMatrixGraph에서는 weight 가 0인 Edge가 존재할 수 있으므로 기존에 Edge가 존재하지 않음을 의미했던 0 과 겹치지 않게 -1로 설정하므로써 일부 메서드가 이에 맞게 변경되어 Override 됩니다. 또한 addEdge 부분도 Edge의 타입이 WeightedEdge으로 바뀌었으므로 Override 되었습니다.

AppController & AppView 추가 및 변경

```
private void showGraph() {
    AppView.outputLine("");
    AppView.outputLine("> 입력된 그래프는 다음과 같습니다: ");

    for (int tailVertex = 0; tailVertex < this.graph().numberOfVertices(); tailVertex++) {
        AppView.output("[" + tailVertex + "] ->");
        for (int headVertex = 0; headVertex < this.graph().numberOfVertices(); headVertex++) {
            if (this.graph().edgeDoesExist(tailVertex, headVertex)) {
                AppView.output(" " + headVertex);
                AppView.output("(" + this.graph().weightOfEdge(tailVertex, headVertex) + ")");

            }
        }
        AppView.outputLine("");
    }
    AppView.outputLine("");
    AppView.outputLine("> 입력된 그래프의 Adjacency Matrix 다음과 같습니다: ");
    AppView.output("        ");

    for (int headVertex = 0; headVertex < this.graph().numberOfVertices(); headVertex++) {
        AppView.output(String.format(" [%1s]", headVertex));
    }

    AppView.outputLine("");

    for (int tailVertex = 0; tailVertex < this.graph().numberOfVertices(); tailVertex++) {
        AppView.output("[" + tailVertex + "] ->");
        for (int headVertex = 0; headVertex < this.graph().numberOfVertices(); headVertex++) {
            int weight = this.graph().weightOfEdge(tailVertex, headVertex);
            AppView.output(String.format("%4d", weight));
        }
        AppView.outputLine("");
    }
}
```

코드가 길어 생략된 부분이 있습니다.

AppView

Edge의 cost를 입력받는 부분이 추가되어야 하므로 해당 기능을 담당하는 inputCost 메서드가 추가되었습니다.

AppController

이번 과제와 관련없는 코드 부분들이 제거되었고 WeightedUndirectedAdjacencyMatrixGraph를 다루게 됩니다. Edge도 Weighted Edge를 다루므로 그에 맞게 변경되었습니다. 또한 그래프를 추가하고 보여주는 부분도 Weight를 입력하고 출력해주는 부분이 추가되었습니다.

결과분석

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

결과

```
<terminated> _Main_AL05_201701975_구건모 [Java Application] C:₩Progra
<<< 최소비용 확장 트리 찾기 프로그램을 시작합니다 >>>
> 입력할 그래프의 vertex 수와 edge 수를 먼저 입력해야 합니다:
? Vertex 수를 입력하시오: 6
? Edge 수를 입력하시오: 10

> 이제부터 edge를 주어진 수 만큼 입력합니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 0
? head vertex 수를 입력하시오: 1
? cost 를 입력하시오: 16
!새로운 edge (0,1, (16)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 0
? head vertex 수를 입력하시오: 4
? cost 를 입력하시오: 19
!새로운 edge (0,4, (19)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 0
? head vertex 수를 입력하시오: 5
? cost 를 입력하시오: 21
!새로운 edge (0,5, (21)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 1
? head vertex 수를 입력하시오: 2
? cost 를 입력하시오: 5
!새로운 edge (1,2, (5)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 1
? head vertex 수를 입력하시오: 3
? cost 를 입력하시오: 6
!새로운 edge (1,3, (6)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 1
? head vertex 수를 입력하시오: 5
? cost 를 입력하시오: 11
!새로운 edge (1,5, (11)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 2
? head vertex 수를 입력하시오: 3
? cost 를 입력하시오: 10
!새로운 edge (2,3, (10)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 3
? head vertex 수를 입력하시오: 4
? cost 를 입력하시오: 18
!새로운 edge (3,4, (18)) 가 그래프에 삽입되었습니다.
```

```
!새로운 edge (3,5, (14)) 가 그래프에 삽입되었습니다.
- 입력할 edge의 두 vertex와 cost를 차례로 입력해야 합니다:
? tail vertex 를 입력하시오: 4
? head vertex 수를 입력하시오: 5
? cost 를 입력하시오: 33
!새로운 edge (4,5, (33)) 가 그래프에 삽입되었습니다.

> 입력된 그래프는 다음과 같습니다:
[0] -> 1(16) 4(19) 5(21)
[1] -> 0(16) 2(5) 3(6) 5(11)
[2] -> 1(5) 3(10)
[3] -> 1(6) 2(10) 4(18) 5(14)
[4] -> 0(19) 3(18) 5(33)
[5] -> 0(21) 1(11) 3(14) 4(33)

> 입력된 그래프의 Adjacency Matrix 다음과 같습니다:
        [0] [1] [2] [3] [4] [5]
[0] ->  -1  16  -1  -1  19  21
[1] ->  16  -1   5   6  -1  11
[2] ->  -1   5  -1  10  -1  -1
[3] ->  -1   6  10  -1  18  14
[4] ->  19  -1  -1  18  -1  33
[5] ->  21  11  -1  14  33  -1

<<< 최소비용 확장 트리 찾기 프로그램을 종료합니다 >>>
```

기존에 Edge를 추가하는 부분에 cost 를 입력하는 부분이 추가된 것을 확인할 수 있고, 그래프 삽입 메시지에도 weight 값이 반영됩니다.

또한 추가된 그래프를 프린트 할 때에 Edge의 Weight가 같이 출력되는 것을 볼 수 있으며, Show Graph 에서 추가된 AdjacencyMarix를 출력해주는 부분을 통해 Edge가 아닌 부분은 -1, Edge의 경우 Weight 값이 정상적으로 Marix에 저장되어 있음을 확인 할 수 있습니다.

이번 과제는 기존의 graph에서 Edge에 weight 속성만 부여한 것이기 때문에 구현 및 결과 상의 특별한 점은 없었던 것 같습니다.

과제 질문에 대한 답변

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

과제 안에 질문에 대한 답변

생각할 점

- 이번 주에 이어, 다음 주에는 나머지 세 번째 단계를 실행하여, "최소 경비 확장 트리" 프로그램을 완성한다.

- Java 에서 format 은 출력에만 사용하는 것이 아니라, 문자열을 형식화 (formatting) 할 필요가 있는 경우에는 언제든지 적용하여 사용할 수 있다.
  - 참고: C 언어에도 "sprintf()" 가 있다.

- 이번 과제에서는 별도의 질문이 존재하지 않았습니다!

과제를 통해 느낀점

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

과제를 통해 느낀점

이번 과제에서 실질적으로 구현한 점은 기존의 Edge에서 Weight를 부여하여 WeightedUndirectedGraph 를 구현하는 것이었습니다. 따라서 어려운 개념이나 구현상의 어려움은 없었다고 생각합니다. AdjacencyMatrix를 이용해 그래프를 구현하는 방법 또한 기존 실습과 과제에서 진행해왔던 내용이기 때문에 코드양은 많았지만 구현상의 어려움이나 구현을 이해하는데에 크게 어려운 점은 없었던 것 같습니다.

감사합니다!

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com

감사합니다

CNU, Computer Science & Engineering
StuNo. 201701975, stuName. Kugunmo
kgm9179@naver.com