알고리즘 과제 보고서

- AL06, MinCostSpanningTree 과제

충남대학교 컴퓨터공학과
분반: 알고리즘 04 분반
제출일: 2021.10.26
학번: 201701975
이름: 구건모

과제에서 해야할 일과 주요이론

*** 과제의 주요 이론**

이번 과제는 Kruskal 알고리즘을 이용하여 MinCostSpanningTree를 구하는 것이었습니다. Kruskal 알고리즘은 weight가 부여되어 있는 edge 들을 다루는 weigthed graph 에서, weight가 가장 낮은 값을 가지는 edge 부터 선택한 후, 해당 edge를 spanningTree 에 추가하였을 때 cycle을 형성하는지 형성하지 않는지 확인한 후, 형성하지 않을 경우 spanningTree에 추가하는 것이 주요 동작과정 으로, Kruskal 알고리즘을 통해 결과적으로 모든 정점들이 최소 비용으로만 연결되어 있는 MinCostSpanningTree를 얻게 됩니다.

*** 과제에서 해야할 일**

따라서 이번 과제에서 실질적으로 구현해야 하는 부분은, MinCostSpanningTree 를 Kruskal 알고리즘을 이용해 추가할 때 edge의 weight를 작은 값부터 차례대로 반환하기 위한 MinPriorityQ 클래스의 구현과,
그것을 이용한 MinCostSpanningTree 에서의 Kruskal 알고리즘 구현, 그리고 AppController 에서 만들어진 MST를 iterator로 순회 하며 출력해주는 함수를 구현하여 과제의 목표를 달성하게 됩니다.

주요 구현 내용

*** LinkedList Class의 LinkedListIterator**

```
public Iterator<T> listIterator() {
    return new LinkedListIterator<T>();
}

public class LinkedListIterator<E> implements Iterator<E> {
    LinkedNode<E> _nextNode;

    private LinkedNode<E> nextNode() {
        return this._nextNode;
    }

    private void setNextNode(LinkedNode<E> newLinkedNode) {
        this._nextNode = newLinkedNode;
    }

    private LinkedListIterator() {
        this.setNextNode((LinkedNode<E>) LinkedList.this.head());
    }

    public boolean hasNext() {
        return (this.nextNode() != null);
    }

    public E next() {
        E nextElement = this.nextNode().element();
        this.setNextNode(this.nextNode().next());
        return nextElement;
    }

}
```

* LinkedListIterator

LinkedListIterator 는 node 들로 이루어진 LinkedList를 순회할 때 사용할 Iterator 로, Iterator 클래스를 상속받아 구현합니다. LinkedListIterator 이므로 LinkedList 를 순회하기 적절하게 override 되어 구현되었습니다. next 메소드를 통해 다음 노드로 이동하고 hasNext()로 마지막 노드인지 확인합니다. 따라서 사용될때는 hasNext가 false가 반환될 때까지 Iteration 하도록 조건을 주어 next() 메서드를 통해 노드의 값을 가져오거나 사용하게 됩니다.

주요 구현 내용

*** MinPriorityQ의 코드 중 일부, min(), removeMin() 메서드**

```
    public E min() {
        if (this.isEmpty()) {
            return null;

        } else {
            return this.heap()[MinPriorityQ.HEAP_ROOT];
        }
    }

    public E removeMin() {
        if (this.isEmpty()) {
            return null;

        } else {
            E rootElement = this.heap()[MinPriorityQ.HEAP_ROOT];
            this.setSize(this.size() - 1);
            if (this.size() > 0) {
                E lastElement = this.heap()[this.size()];

                int parent = MinPriorityQ.HEAP_ROOT;
                while ((parent) * 2 <= this.size()) {
                    int smallerChild = parent * 2;
                    if ((smallerChild) < this.size()
                            && (this.heap()[smallerChild].compareTo(this.heap()[smallerChild + 1]) > 0)) {
                        smallerChild++;
                    }

                    if (lastElement.compareTo(this.heap()[smallerChild]) <= 0) {
                        break;
                    }
                    this.heap()[parent] = this.heap()[smallerChild];
                    parent = smallerChild;
                }
                this.heap()[parent] = lastElement;
            }
            return rootElement;
        }
    }
}
```

* MinPriorityQ

MinPriorityQ 는 Kruskal 알고리즘을 적용할 때에, 그래프 edge의 weight 값이 가장 작은 값을 가지는 edge 부터 spanningTreeEdge로 add 할 지에 대해 Decision을 하게 되는데 그 과정에서 MinPriorityQ 클래스를 이용하여 weight 값이 최솟값을 가지는 Edge를 하나씩 가져오게 됩니다. MinPriorityQ는 heap으로 구현되어 있으며, 위에서 구현한 removeMin을 이후 Kruskal 알고리즘을 사용할 때 가장 작은 weight 값을 가지는 edge 부터 하나씩 가져오는 과정에서 사용하게 됩니다.

주요 구현 내용

*** MinCostSpanningTree 클래스의 solve() method**

```
public List<WeightedEdge> solve(WeightedUndirectedAdjacencyMatrixGraph<WeightedEdge> aGraph) {
    this.setGraph(aGraph);
    this.initMinPriorityQ();
    this.setSpanningTreeEdgeList(new LinkedList<WeightedEdge>());

    PairwiseDisjointSets pairwiseDisjointSets = new PairwiseDisjointSets(this.graph().numberOfVertices());

    int maxNumberOfTreeEdges = this.graph().numberOfVertices() - 1;
    System.out.println("this.spanningTreeEdgeList.size() = " + this.spanningTreeEdgeList().size());
    System.out.println("maxNumberOfTreeEdges = " + maxNumberOfTreeEdges);

    while ((this.spanningTreeEdgeList().size() < maxNumberOfTreeEdges) && (!this.minPriorityQ().isEmpty())) {

        WeightedEdge edge = this.minPriorityQ().removeMin();
        int setOfTailVertex = pairwiseDisjointSets.find(edge.tailVertex());
        int setOfHeadVertex = pairwiseDisjointSets.find(edge.headVertex());
        if (setOfTailVertex == setOfHeadVertex) {
            AppView.outputLine("[DEBUG] Edge(" + edge.tailVertex() + ", " + edge.headVertex() + ", ("
                    + edge.weight() + "))는 스패닝 트리에 사이클을 생성시키므로, 버립니다.");
        } else {
            this.spanningTreeEdgeList().add(edge);
            pairwiseDisjointSets.union(setOfTailVertex, setOfHeadVertex);
            AppView.outputLine("[DEBUG] Edge(" + edge.tailVertex() + ", " + edge.headVertex() + ", ("
                    + edge.weight() + "))는 스패닝 트리의 edge로 추가됩니다.");
        }
    }
```

* MinCostSpanningTree Class의 Solve()

MinCostSpanningTree는 실질적으로 Kruskal 알고리즘을 이용하여 spanningTreeEdgeList에 edge 들을 하나씩 추가하여 최종적으로 MST를 구합니다.

solve 메소드는 Kruskal 알고리즘을 이용하여 spanningTree의 edge 리스트를 얻습니다. 이전 과제에서 구현하였던 PairwiseDisjointSet 클래스를 이용하여 추가하려는 edge가 cycle을 형성하는지 find method로 판단하고 사이클을 형성할 경우 SpanningTreeEdgeList에 추가하지 않고 만약 Cycle을 형성하지 않을 경우 추가한 후, edge가 추가되었으므로, 추가된 Edge 의 tailVertex와 headVertex를 Union 해주게 됩니다.

주요 구현 내용

* AppController 클래스의 showMinCostSpanningTree() 메소드

```
public void showMinCostSpanningTree() {
    AppView.outputLine("");
    AppView.outputLine("> 주어진 그래프의 최소비용 확장트리의 edge들은 다음과 같습니다: ");
    Iterator<WeightedEdge> listIterator = this.spanningTreeEdgeList().listIterator();
    while (listIterator.hasNext()) {
        WeightedEdge edge = listIterator.next();
        AppView.outputLine(
                "Tree Edge(" + edge.tailVertex() + ", " + edge.headVertex() + ", (" + edge.weight() + "))");
    }
}

public void run() {

    AppView.outputLine("<<< 최소비용 확장 트리 찾기 프로그램을 시작합니다 >>>");
    this.inputAndMakeGraph();
    this.showGraph();

    AppView.outputLine("");
    AppView.outputLine("> 주어진 그래프의 최소비용 확장트리 찾기를 시작합니다: ");
    AppView.outputLine("");

    this.setMinCostSpanningTree(new MinCostSpanningTree());

    this.setSpanningTreeEdgeList(this.minCostSpanningTree().solve(this.graph()));
    if (this.spanningTreeEdgeList() == null) {
        AppView.outputLine("> 주어진 그래프의 컴포넌트가 2 개 이상이어서, 최소비용 확장트리 찾기를 실패하였습니다.");

    } else {
        this.showMinCostSpanningTree();
    }

    AppView.outputLine("");
    AppView.outputLine("<<< 최소비용 확장 트리 찾기 프로그램을 종료합니다 >>>");

}
```

* AppController Class 에서 MST를 보여주는 부분

만들어진 MST를 출력해주기 위해서 Iterator를 이용하여 spanningTreeEdgeList 를 순회하며 결과로 들어간 Edge 들을 출력해주게 됩니다.

run 메서드의 경우 저번 과제에서 진행했던 부분 뒤에 showMinCostSpanningTree() 를 실행하여 결과를 보여주게 됩니다.

결과화면 1

```
<terminated> _Main_AL06_201701975_구건모 [Java Application] C:₩Prog
? 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 (3,5, (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

> 주어진 그래프의 최소비용 확장트리 찾기를 시작합니다:

[DEBUG] Edge(1, 2, (5))는 스패닝 트리의 edge로 추가됩니다.
[DEBUG] Edge(1, 3, (6))는 스패닝 트리의 edge로 추가됩니다.
[DEBUG] Edge(2, 3, (10))는 스패닝 트리에 사이클을 생성시키므로, 버립니다.
[DEBUG] Edge(1, 5, (11))는 스패닝 트리의 edge로 추가됩니다.
[DEBUG] Edge(3, 5, (14))는 스패닝 트리에 사이클을 생성시키므로, 버립니다.
[DEBUG] Edge(0, 1, (16))는 스패닝 트리의 edge로 추가됩니다.
[DEBUG] Edge(3, 4, (18))는 스패닝 트리의 edge로 추가됩니다.

> 주어진 그래프의 최소비용 확장트리의 edge들은 다음과 같습니다:
Tree Edge(3, 4, (18))
Tree Edge(0, 1, (16))
Tree Edge(1, 5, (11))
Tree Edge(1, 3, (6))
Tree Edge(1, 2, (5))

<<< 최소비용 확장 트리 찾기 프로그램을 종료합니다 >>>
```

MinCostSpanningTree 찾기결과

** 결과 이미지가 너무 커서 캡쳐 이미를 부분을 나누어 이어 붙혔습니다.*

*** 결과1**

사용자가 그래프를 추가하고 나면 추가된 그래프에 대한 MST를 찾아 보여줍니다.

위의 결과를 보면 MinCostSpanningTree 클래스에서

Solve 메소드가 실행되며 DEBUG 메시지를 출력하는 부분을 볼 수 있는데

해당 과정이 Kruskal 알고리즘이 동작하는 과정중에서 Cycle에 따라 Edge의 add를 Decision 하는

부분으로 PairwiseDisjointSet을 이용하여 사이클을 검사하고, 사이클을 생성하는

경우 discard 하고 생성하지 않는 경우 add 하는 것을 확인할 수 있습니다.

아래 최소비용확장트리가 출력된 부분을 보면 cycle을 형성하지 않는 edge들만 정상적으로 들어간

것을 확인해 볼 수 있습니다.

생각할 점

> Class "MinCostSpanningTree" 는 그래프가 Adjacency List 로 구현된 경우에도 작동할까? 또는 어떤점이 불편한가?

이번 과제의 경우 Matrix(array) 자료구조를 이용한 그래프를 다뤘기 때문에 MinCostSpanningTree 에서 MinPriorityQ를 초기화 해주는 initMinPriorityQ가 사실상 2중 for문을 통해 2차원 배열의 인덱스를 순회하며 minPriorityQ를 초기화 해주는데, Adjacency List로 바꾸게 될 경우 순회하는 방식이 배열이 인덱스를 통해 접근하는 방식과는 완전히 달라지기 때문에 해당 과정을 수정하지 않으면 작동하지 않습니다. 현재의 경우 Graph의 구현에 따라서 MinCostSpanningTree의 코드들이 종속성을 가진다고도 볼 수 있습니다. 따라서 Iterator 등을 이용하여 코드를 변경하게 된다면 작동하겠지만, initMinPriortyQ() 메서드를 그대로 사용하는 상태에서 Adjacency List로 구현된 그래프를 이용한다면 오류가 발생하게 됩니다.

> Class "MinCostSpanningTree"의 source 를 전혀 수정하지 않고도, 구현에 따라 달라지는 그래프의 class 종류와 무관하게 작동가능하게 만드는 방법은?

위의 질문에서 답변했던 것과 같은 맥락으로 어떤 자료구조로 구현을 하던, 기존에 그래프 관련 기능을 다루는 source 들이 그래프의 구현 방법이 바뀌는 것에 종속되지 않게 하기 위해서 Iterator를 이용하면 graph를 다른 자료구조로 바꾸어 구현해도 그에 맞는 Iterator를 이용하면 기존의 source 들을 수정하지 않고도 그래프의 class 종류와 무관하게 동작하도록 만들 수 있습니다.

감사합니다.

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