자료구조

실습 보고서

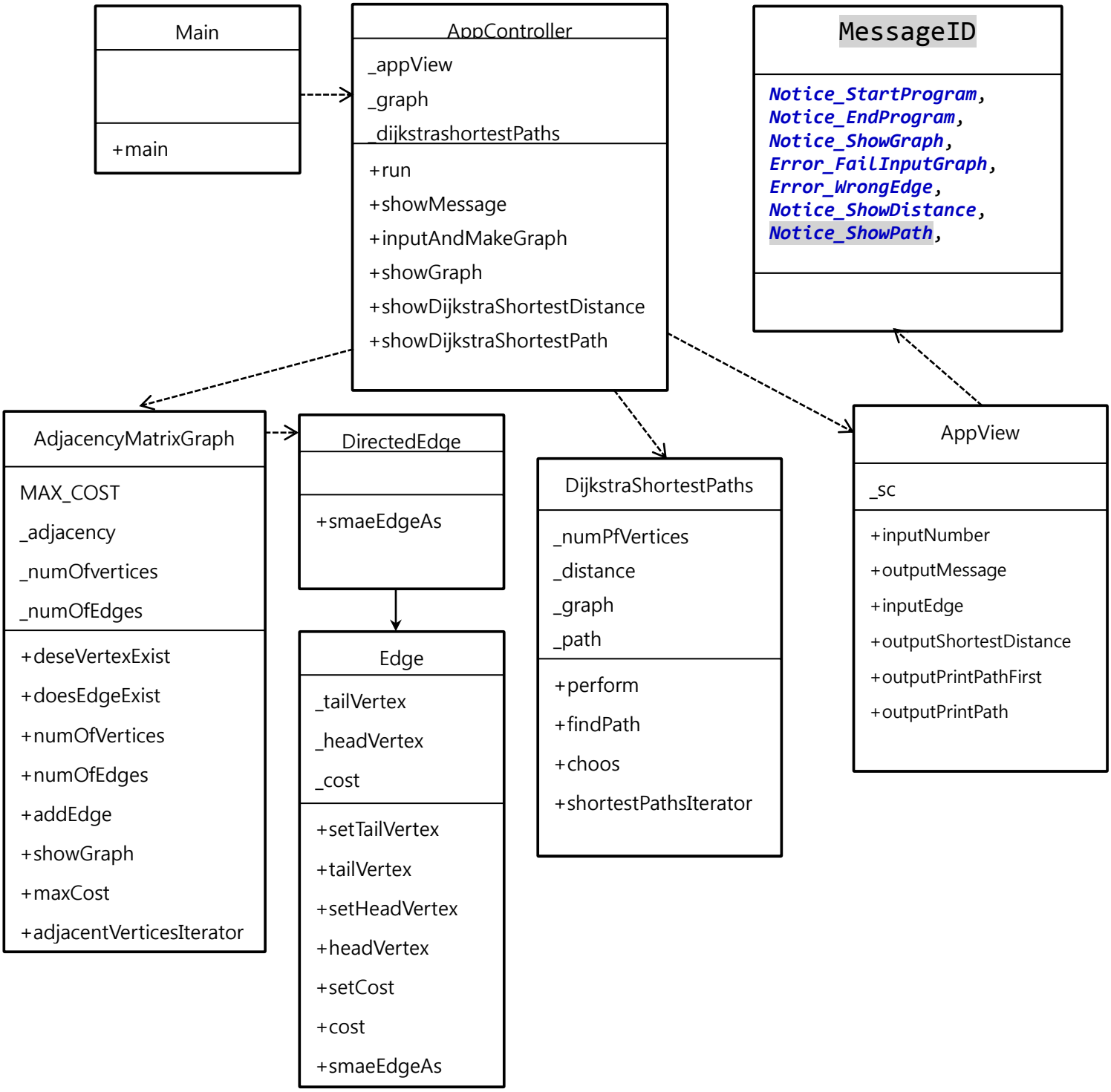
[제 03주] 최단 경로

제출일 : 2015.09.30

201402432 / 조디모데

1.프로그램설명서

1.주요알고리즘/자료구조/기타



자료 구조 : 그래프, ArrayList, 최단경로 알고리즘(Dijkstra)

2.함수설명서

Main

main : 메인함수

AppController

run : 프로그램 중심함수 (메뉴)
showMessage : AppView 를 이용하여 문자열을 출력
inputAndMakeGraph : Vertexes 와 Edge를 입력받아 그래프를 만드는 메소드
showGraph : 그래프 출력 메소드
showDijkstraShortestDistance : 최단 경로를 이용한 비용을 출력
showDijkstraShortestPath : 최단 경로를 출력

AppView

inputNumber : 정수를 입력받기위한 메소드
outputMessage() : 문자열 출력 메소드
inputEdge : 추가된 Edge의 정보를 출력하는 메소드
outputShortestDistance : 최단 경로의 비용을 출력하는 메소드
outputPrintPathFirst : 최단 경로의 첫 원소를 출력하는 메소드
outputPrintPath : 최단 경로의 둘째 원소부터 출력할 때 이용하는 메소드

AdjacencyMatrixGraph

deseVertexExist : Vertex가 이미 존재하는지 여부를 반환하는 메소드
doesEdgeExist : Edge가 이미 존재하는지 여부를 반환하는 메소드
numOfVertices : Verteices 개수를 반환하는 메소드
numOfEdges : Edge개수를 반환하는 메소드
addEdge : Edge를 추가하는 메소드
showGraph : 그래프를 출력하는 메소드
adjacentVerticesIterator : 반복자, 트리 출력시 사용
maxCost : Edge의 cost의 초기값

Edge : abstract 클래스로 추상 메소드 sameEdgeAs를 가지고 있다.

setTailVertex : 뒤쪽 Vertex를 설정하는 메소드

tailVertex : 뒤쪽 Vertex를 반환하는 메소드

setHeadVertex : 앞쪽 Vertex를 설정하는 메소드

headVertex : 앞쪽 Vertex를 반환하는 메소드

smaeEdgeAs : Edge가 같은지 판단하는 메소드

setCost : 경로의 비용을 설정하는 메소드

cost : 경로의 비용을 반환하는 메소드

DirectedEdge : Edge와 comparable을 구현하는 메소드

smaeEdgeAs : Edge가 같은지 판단하는 메소드

DijkstraShortestPaths

Perform : 최단 경로를 구하는 작업

findPath : 최단 경로를 찾는 메소드

choos : 가장 비용이 적게드는 distance를 찾아 반환함

shortestPathsIterator : Iterather of Paths

2. 탐구과제

1) findPath 메소드에서 최단경로를 만들어내는 동작 원리 설명

메소드가 처음 실행됐을 때 pahtList를 생성한 후 null인 상태에서 start를 add한다.

Start 부터 end 까지의 path를 찾았는지 여부를 path[end]!=start를 통해 확인 한 후 찾지 못했을 경우 findPath 메소드를 다시 호출한다. 재귀적으로 함수를 호출중 path를 찾았을 경우

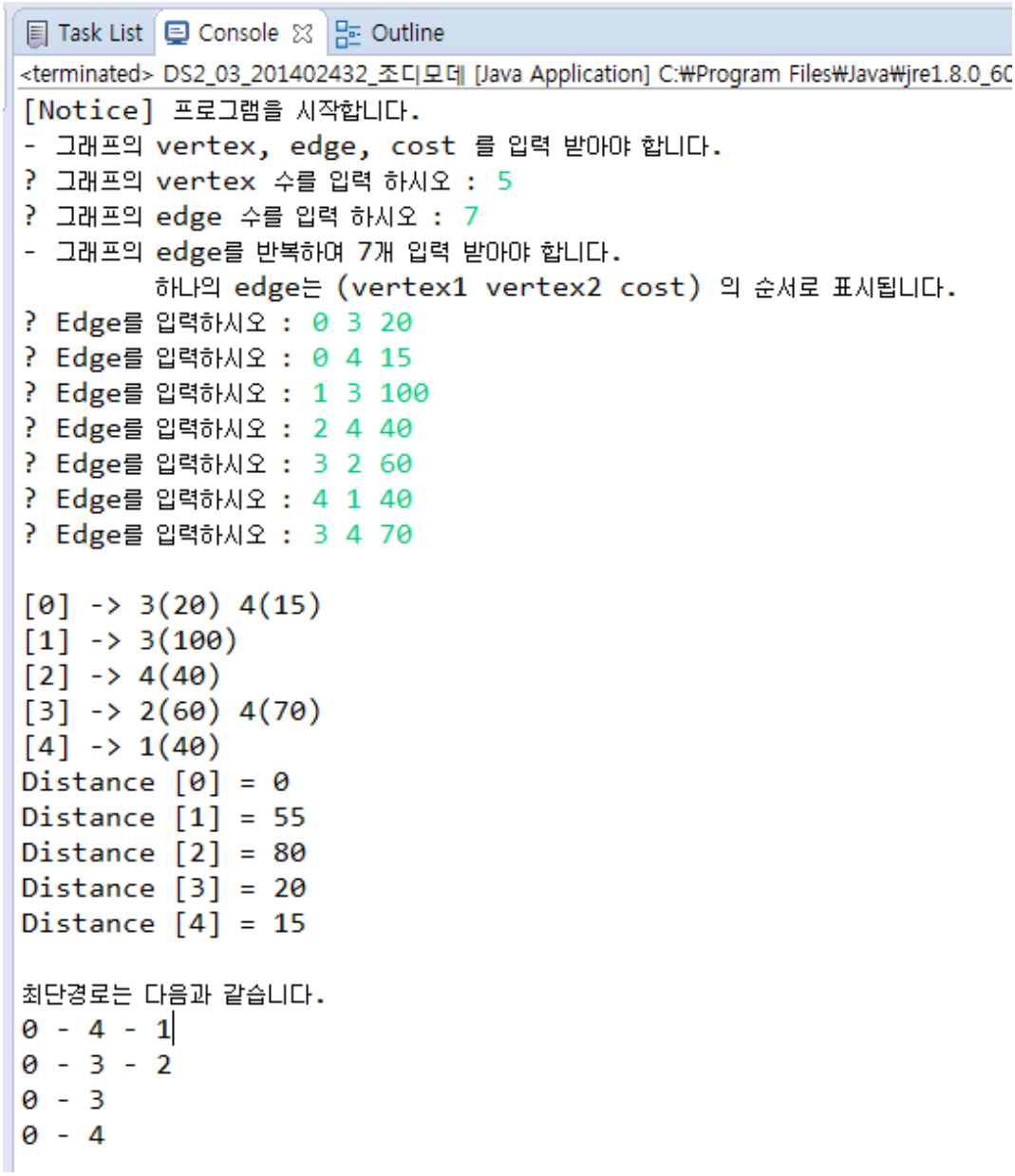
Path에 end를 add한 뒤 pathList를 반환한다.

2) 탐구과제2 : ShrotestPathsIterator를 사용하는 이유는?

반복자를 이용하여 반복여부의 진리값과 다음 원소의 호출을 추상화 시킴으로써 사용자로 하여금 프로그램의 사용을 용이케 하기위해..

3.실행 결과 분석

1.입력과출력



2.결과분석

직접 찾아본 최단 경로와 프로그램의 결과값이 일치하였다.

3.소스 코드

<main>

```
public class DS2_03_201402432_조디모데 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Application appController = new Application() ;
		appController.run();

	}

}
```

<AdjacencyMatrixGraph>

```
import java.util.Arrays;

public class AdjacencyMatrixGraph {

	private final int MAX_COST = 9999;
	private int[][] _adjacency ;
	private int _numOfVertices;
	private int _numOfEdges;


	public AdjacencyMatrixGraph(int givenNumOfVertices){

		this._numOfVertices = givenNumOfVertices ;
		this._numOfEdges = 0 ;
		this._adjacency = new
int[this._numOfVertices][this._numOfVertices] ;
		for(int i=0 ; i<this._adjacency.length ; i++)
			Arrays.fill(this._adjacency[i], MAX_COST) ;
	}

	public boolean doesVertexExist(int aVertex){
		if(aVertex>-1 && aVertex<this._numOfVertices)
			return true ;

		return false ;
	}
	public boolean doesEdgeExist(Edge anEdge){
```

```
	if( this._adjacency[anEdge.headVertex()][anEdge.tailVertex()]!
=MAX_COST)
			return true ;
		return false ;
	}

	public int numOfVertices(){
		return this._numOfVertices ;
	}

	public int numOfEdges(){
		return this._numOfEdges ;
	}

	public boolean addEdge(Edge anEdge){
		if(!this.doesVertexExist(anEdge.headVertex()))
			return false ;
		if(!this.doesVertexExist(anEdge.tailVertex()))
			return false ;
		if(this.doesEdgeExist(anEdge))
			return false ;

		this._adjacency[anEdge.headVertex()][anEdge.tailVertex()]
= anEdge.cost() ;

		return true ;
	}

	public void showGraph(){
		int i = 0 ;
		System.out.println();

		while(i<this._adjacency.length ){
			System.out.print("["+i+"] -> ");
			for(int j=0 ; j<this._adjacency[i].length ; j++ )
				if(this._adjacency[i][j]!=this.maxCost())

	System.out.print(j+"("+this._adjacency[i][j]+") ") ;
			System.out.println();
			i++ ;
		}
	}

	protected int maxCost(){
		return MAX_COST ;
	}
```

```
	public int costOfEdge(int aFromVertex, int aToVertex){
		return this._adjacency[aFromVertex][aToVertex] ;
	}

	public AdjacentVerticesIterator adjacentVerticesIterator(int
givenVertex){
		return new AdjacentVerticesIterator(givenVertex) ;
	}

	public class AdjacentVerticesIterator{
		private int _nextPosition ;
		private int _vertex ;

		private AdjacentVerticesIterator(int givenVertex){
			this._nextPosition = 0 ;
			this._vertex = givenVertex ;
		}
		public boolean hasNext(){
			while(_adjacency[this._vertex][this._nextPosition]
== MAX_COST
					&& this._nextPosition !=
numOfVertices()) ;
			this._nextPosition++ ;
			return (this._nextPosition < numOfVertices()) ;
		}

		public DirectedEdge next(){
			DirectedEdge anEdge =
					new DirectedEdge (_vertex,
this._nextPosition, _adjacency[this._vertex][this._nextPosition]) ;
			this._nextPosition++ ;
			return anEdge ;
		}

	} // Inner class


} // class
```

<Application>

```
import java.util.ArrayList;
import java.util.Iterator;

public class Application {

	private AppView _appView ;
	private AdjacencyMatrixGraph _graph ;
	private DijkstraShortestPaths
_dijkstrashortestPaths;

	public Application(){
		this._appView = new AppView() ;
	}

	public void run(){

	this.showMessage(MessageID.Notice_StartProgra
m);

		if ( this.inputAndMakeGraph() ) {
```

```
                this.showGraph() ;
                this._dijkstrashortestPaths = new
DijkstraShortestPaths(this._graph);
                this._dijkstrashortestPaths.perform();
                this.showDijkstraShortestDistance();
                this.showDijkstraShortestPath();
            }
            else {

        this._appView.outputMessage(MessageID.Error_Fai
lInputGraph);
            }

        this.showMessage(MessageID.Notice_EndProgram
);
        }

        private boolean inputAndMakeGraph(){
            // 그래프 정보를 입력받음, 마지막에 입력된
그래프를 출력하여 보여줌
            int countEdges ;
```

```
            int numOfVertices ;
            int numOfEdges ;

            this._appView.outputMessage("- 그래프의
vertex, edge, cost 를 입력 받아야 합니다.₩n") ;
            this._appView.outputMessage("? 그래프의
vertex 수를 입력 하시오 : ") ;
            numOfVertices =
this._appView.inputNumber() ;
            this._appView.outputMessage("? 그래프의
edge 수를 입력 하시오 : ") ;
           numOfEdges = this._appView.inputNumber() ;

            this._graph = new
AdjacencyMatrixGraph(numOfVertices) ;

            countEdges = 0 ;
            this._appView.outputMessage("- 그래프의
edge를 반복하여 " + numOfEdges + "개 입력 받아야
합니다.₩n");
            this._appView.outputMessage("    하나의
```

```
edge는 (vertex1 vertex2 cost) 의 순서로
표시됩니다.₩n");
            this._appView.outputMessage("? Edge를
입력하시오 : ") ;

            while(true){

                DirectedEdge anEdge = new
DirectedEdge(this._appView.inputNumber(),
                                this._appView.inputNumber(),
this._appView.inputNumber() );
                if(anEdge.cost() > 0 &&
this._graph.addEdge(anEdge)){
                    countEdges ++ ;
                }
                else

      this._appView.outputMessage(MessageID.Error_Wr
ongEdge) ;

                if(countEdges == numOfEdges)
```

```
                    break ;
                this._appView.outputMessage("? Edge를
입력하시오 : ");
            }

            return (countEdges == numOfEdges) ;

        }

        private void showDijkstraShortestDistance() {
            int i = 0 ;

        this._appView.outputMessage(MessageID.Notice_S
howDistance);
            DijkstraShortestPaths.ShortestPathsIterator
dijkstraShortestPathsIterator
            =
this._dijkstrashortestPaths.shortestPathsIterator() ;
            while(dijkstraShortestPathsIterator.hasNext()){
                this._appView.outputShortestDistance(i,
dijkstraShortestPathsIterator.next());
```

```
                i++ ;
            }
            this._appView.outputMessage("₩n") ;

        }

        private void showDijkstraShortestPath() {


        this._appView.outputMessage(MessageID.Notice_ShowPath);
            for(int i=1 ; i<this._graph.numOfVertices() ; i++){
                ArrayList<Integer> resultShortestPath = this._dijkstrashortestPaths.findPath(0, i, null) ;
                Iterator<Integer> arrayListIterator = resultShortestPath.iterator() ;


        this._appView.outputPrintPathFirst(arrayListIterator.next()) ;
```

```
                while(arrayListIterator.hasNext()){

        this._appView.outputPrintPath(arrayListIterator.nex
t()) ;
                }
                this._appView.outputMessage("₩n") ;
            }
            this._appView.outputMessage("₩n") ;

        }

        public void showGraph(){
            // Graph의 showGraph() 함수 호출
            this._graph.showGraph() ;
        }
        private void showMessage(MessageID
aMessageID) {
            this._appView.outputMessage(aMessageID);
        }
} // class End
```

<AppView>

```
import java.util.Scanner;
public class AppView {

	private Scanner _sc ;

	AppView(){
		this._sc = new Scanner(System.in) ;
	}

	public int inputNumber() {
		return this._sc.nextInt() ;
	}

	public void outputMessage(String MessageID){
		System.out.print(MessageID);
	}

	public void outputMessage(MessageID MessageID) {
		switch(MessageID) {
		case Notice_StartProgram :
			this.outputMessage("[Notice] 프로그램을 시작합니다.
\n");
			break ;
		case Notice_EndProgram :
			this.outputMessage("[Notice] 프로그램을 종료합니다.
\n");
			break ;
		case Error_WrongEdge :
			this.outputMessage("[Error] 입력 오류입니다. \n");
			break ;
		case Notice_ShowPath :
			this.outputMessage("최단경로는 다음과 같습니다.\n");
			break ;
		default:
			break;
		}

	}

	public Edge inputEdge(int v1, int v2, int cost){
		return new DirectedEdge(v1, v2, cost) ;
	}

	public void outputShortestDistance(int tailVertex, int i) {
```

```
            this.outputMessage("Distance ["+tailVertex+"] = " + i +
"\n" );
      }

      public void outputPrintPathFirst(Integer next) {
            System.out.print(next);

      }

      public void outputPrintPath(Integer next) {
            System.out.print(" - "+next);
      }


}
```

<DijkstraShortestPaths>

```
import java.util.ArrayList;
import java.util.Arrays;

public class DijkstraShortestPaths {

	private int _numOfVertices ;
	private int[] _distance;
	private AdjacencyMatrixGraph _graph ;
	private int[] _path ;

	public
DijkstraShortestPaths(AdjacencyMatrixGraph
givenGraph ){
		this._graph = givenGraph ;
		this._numOfVertices =
this._graph.numOfVertices() ;
		this._distance = new
int[this._numOfVertices] ;
		this._path = new int[this._numOfVertices] ;
	}
```

```
	public void perform(){

		int i, u, w ;
		boolean[] found = new
boolean[this._numOfVertices] ;
		Arrays.fill(found, false) ;


		for (i= 0; i< this._numOfVertices; i++) {
			this._distance[i] =
this._graph.costOfEdge(0, i) ;
		}
		found[0] = true ;
		this._distance[0] = 0 ;

		for (i=0; i< this._numOfVertices-2; i++) {
			u = this.choos(found) ;
			found[u] = true;
			for (w = 0; w < this._numOfVertices;
w++) {
```

```
                        if ( !found[w] ) {
                            if(this._graph.costOfEdge(u,
w)!=this._graph.maxCost())
                                if ( this._distance[w] >
this._distance[u] + this._graph.costOfEdge(u, w) ){
                                    this._distance[w] =
this._distance[u] + this._graph.costOfEdge(u, w);
                                    this._path[w] = u ;
                                }
                        }
                    }
                }

        }

        public ArrayList<Integer> findPath(int start, int
end, ArrayList<Integer> pathList){
            if(pathList == null){
                pathList = new ArrayList<Integer>
(this._numOfVertices) ;
                pathList.add(start) ;
```

```
            }

            if(_path[end] != start)
                pathList = findPath(start, _path[end],
pathList) ;

            pathList.add(end) ;
            return pathList ;
        }

        private int choos(boolean[] givenFound){
            int min = 0;
            int[] tmp = this._distance.clone() ;

            for(int i=0 ; i<this._distance.length ; i++)
                if(givenFound[i]==true)
                    tmp[i] = this._graph.maxCost() ;

            Arrays.sort(tmp);
            while(true){
                if(this._distance[min]==tmp[0])
```

```
                    break ;
                min++ ;
            }

            return min ;
        }

        public ShortestPathsIterator
shortestPathsIterator(){
            return new ShortestPathsIterator() ;
        }

        public class ShortestPathsIterator{
            private int _next ;

            private ShortestPathsIterator(){
                this._next = 0 ;
            }

            public boolean hasNext(){
                return (this._next != _numOfVertices) ;
```

```
        }

        public int next(){
            int next = _distance[this._next] ;
            _next ++ ;
            return next ;
        }

    }

}
```

<DirectedEdge>

```
public class DirectedEdge extends Edge {
    public DirectedEdge(int givenHeadVertex, int givenTailVertex,
            int givenCost){

        super( givenHeadVertex, givenTailVertex, givenCost);
    }

    @Override
    public boolean sameEdgeAs(Edge anEdge) {
        if(this.tailVertex() == anEdge.tailVertex() &&
anEdge.headVertex() == this.headVertex())
            return true ;
        else
            return false ;
    }

}
```

<Edge>

```
public abstract class Edge {

	private int _tailVertex ;
	private int _headVertex ;
	private int _cost ;

	public Edge(int givenHeadVertex, int givenTailVertex, int
givenCost){
		this._headVertex = givenHeadVertex ;
		this._tailVertex = givenTailVertex ;
		this._cost = givenCost ;
	}


	public void setTailVertex(int aTailVertex){
		this._tailVertex = aTailVertex ;
	}

	public int tailVertex(){
		return this._tailVertex ;
	}

	public void setHeadVertex(int aHeadVertex){
		this._headVertex = aHeadVertex ;
	}

	public int headVertex(){
		return this._headVertex ;
	}

	public void setCost(int aCost){
		this._cost = aCost ;
	}

	public int cost(){
		return this._cost ;
	}

	public abstract boolean sameEdgeAs(Edge anEdge) ;

}
```

<MessageID>

```
public enum MessageID {

	// Message IDs for Notices:
	Notice_StartProgram,
	Notice_EndProgram,
	Notice_ShowGraph,
	// MessageIDs for Errors:
	Error_FailInputGraph,
	Error_WrongEdge,
	Notice_ShowDistance,
	Notice_ShowPath,


}
```