과목명	알고리즘
담당교수	우진운 교수님
학과	소프트웨어학과
학번	32153180
이름	이상민
제출일자	2019.05.29

<소스 코드>

MultiStageGraph

```
package homework;
import java.util.Scanner;

public class MultiStageGraph {
	int n, e;
// 정점 n과 간선 e
	int[][] c;                                                    // 이차원
배열 cost
	int[] level;                                                  // 일차원
배열 level
	int dlevel;
// 도착 정점의 level
	int[][] bcost;                                                // 이차원
배열 bcost
	int[][] bd;
// 일차원 배열 bd

	public MultiStageGraph() {                                    // MST 생성자
함수
		n = 0;
		e = 0;
		c = new int[0][0];
		level = new int[0];
		dlevel = 0;
		bcost = new int[0][0];
		bd = new int[0][0];
	}

	public void matrix() {                                        // 인접행렬 함수
		System.out.print("정점 n과 간선 e 입력 : ");
		Scanner input = new Scanner(System.in);
		n = input.nextInt();                                      // 정점 수 입력
		e = input.nextInt();                                      // 간선 수 입력
		c = new int[n+1][n+1];                                    // 이차원 배열
cost 초기화
		level = new int[n+1];                                     // 일차원 배열
level 초기화

		int a, b;
// 정점 m과 n
		int weight;
// 가중치
		for (int i = 0; i < e; i++) {
			System.out.print((i+1) + "번째 간선과 가중치 입력 : ");
			a = input.nextInt();
			b = input.nextInt();
			weight = input.nextInt();
			c[a][b] = weight;                                     // 이차원 배열
cost에 가중치 저장
		}
	}

	public void matrixPrint() {                                   // 인접행렬 출력
함수
		System.out.println("\n-- <인접행렬 출력> --");

		for (int i = 1; i <= n; i++) {
			for (int j = 1; j <= n; j++)
				System.out.print(c[i][j] + "   ");
			System.out.println();
```

```
                }
                System.out.println("\n시작 정점 : 1, 도착 정점 : " + n);
        }

        public void levelFunc(int vertex, int count) {
                level[vertex] = count;
                for (int i = 1; i <= n; i++)
                        if (c[vertex][i] != 0)
                                levelFunc(i, count+1);
        }

        public void dlevelFunc() {
                int MAX = 0;
                for (int i = 1; i <= n; i++)
                        MAX = level[i];
                dlevel = MAX;

                bcost = new int[dlevel+1][n+1];          // backward cost 초기화
                bd = new int[dlevel+1][n+1];             // backward d 초기화
        }

        public int bcostFunc(int level, int num) {
                int temp;
// 각 정점들마다 backward cost 설정
                int MIN = 9999;
                int index=0;
                for (int i = 1; i <= n; i++) {
                        if (c[i][num]!=0) {                                     // 가장
작았던 가중치와 현재 가중치를 더함
                                temp = bcostFunc(level-1,i) + c[i][num];
                                if(MIN > temp) {                  // 가장 작은
가중치를 갖는 간선
                                        MIN = temp;
                                        index = i;
                                }
                        }
                }

                if(index==0)
                        return 0;
                else {
                        bd[level][num] = index;
                        bcost[level][num] = MIN;                  // 가중치 최솟값
                        return MIN;
                }
        }

        public void costPrint() {
                bcostFunc(dlevel, n);                                   // dlevel과 n을
매개변수로 받음
                System.out.println("\n-- <각 정점 cost> --");
                for (int i = 1; i <= dlevel; i++)
                        for (int j = 1; j <= n; j++)        // 각 정점들마다 cost 출력
                                if (bcost[i][j] != 0)
                                        System.out.print(j + "번째 cost : " +
bcost[i][j] + "\n");
        }

        public void dPrint() {
                int l, vertex;
                System.out.println("\n-- <각 정점 d> --");
                for (l = 2; l <= dlevel; l++) {
                        for (vertex = 1; vertex <= n; vertex++) {
                                if (bd[l][vertex]!=0) {
```

```
                                        int path[] = new int [l+1];
                                        int level, index=vertex;
                                        for(level = l; level > 0; level--) {
                                                path[level] = bd[level][index];
// 각 레벨마다 backward d를 배열에 저장한 후
                                                index = bd[level][index];
// index를 다음 backward d로
                                                }
                                        System.out.print(vertex + "째 d : ");
                                        for (int i = 2; i <= l; i++)
// 각 정점들마다 경로 출력
                                                System.out.print(path[i] + " -> ");
                                        System.out.print(vertex + "\n");
                                }
                        }
                }
        }

        public void bcostPrint() {
        // backward cost 출력 함수
                System.out.println("\n최소 비용 : " + bcostFunc(dlevel, n));
        }

        public void bdPrint() {
        // backward d 출력 함수
                int path[] = new int [dlevel + 1];
// path 배열 초기화
                int level, index = n;
                for (level = dlevel; level > 0; level--) {
                        path[level] = bd[level][index];
                        index = bd[level][index];
                }
                System.out.print("최소 비용 경로 : ");
// 최소 비용 경로 출력
                for (int i = 2; i < dlevel; i++)
                        System.out.print(path[i] + " -> ");
                System.out.println(path[dlevel] + " -> " + n);
        }
}
```

Graph

```
package homework;

import java.util.Scanner;

public class Graph {
   public static void main(String args[]) {
            MultiStageGraph MSG = new MultiStageGraph();

            MSG.matrix();
            MSG.matrixPrint();
            MSG.levelFunc(1, 1);
            MSG.dlevelFunc();
            MSG.costPrint();
            MSG.dPrint();
            MSG.bcostPrint();
            MSG.bdPrint();
   }
}
```

<실행 화면>

```
정점 n과 간선 e 입력 : 6 9
1번째 간선과 가중치 입력 : 1 2 5
2번째 간선과 가중치 입력 : 1 3 3
3번째 간선과 가중치 입력 : 2 4 10
4번째 간선과 가중치 입력 : 2 5 3
5번째 간선과 가중치 입력 : 3 4 7
6번째 간선과 가중치 입력 : 3 2 8
7번째 간선과 가중치 입력 : 4 5 6
8번째 간선과 가중치 입력 : 4 6 3
9번째 간선과 가중치 입력 : 5 6 8

-- <인접행렬 출력> --
0  5  3  0  0  0
0  0  0  10  3  0
0  8  0  7  0  0
0  0  0  0  6  3
0  0  0  0  0  8
0  0  0  0  0  0

시작 정점 : 1, 도착 정점 : 6

-- <각 정점 cost> --
2번째 cost : 5
3번째 cost : 3
2번째 cost : 5
3번째 cost : 3
4번째 cost : 10
4번째 cost : 10
5번째 cost : 8
6번째 cost : 13

-- <각 정점 d> --
2째 d : 1 -> 2
3째 d : 1 -> 3
4째 d : 3 -> 4
4째 d : 1 -> 3 -> 4
5째 d : 1 -> 2 -> 5
6째 d : 1 -> 3 -> 4 -> 6

최소 비용 : 13
최소 비용 경로 : 1 -> 3 -> 4 -> 6
```

```
정점 n과 간선 e 입력 : 13 21
1번째 간선과 가중치 입력 : 1 2 3
2번째 간선과 가중치 입력 : 1 3 6
3번째 간선과 가중치 입력 : 1 4 7
4번째 간선과 가중치 입력 : 1 5 5
5번째 간선과 가중치 입력 : 2 6 10
6번째 간선과 가중치 입력 : 3 6 5
7번째 간선과 가중치 입력 : 4 6 2
8번째 간선과 가중치 입력 : 5 6 5
9번째 간선과 가중치 입력 : 6 7 3
10번째 간선과 가중치 입력 : 6 8 2
11번째 간선과 가중치 입력 : 6 9 2
12번째 간선과 가중치 입력 : 7 10 3
13번째 간선과 가중치 입력 : 7 11 5
14번째 간선과 가중치 입력 : 7 12 4
15번째 간선과 가중치 입력 : 8 10 3
16번째 간선과 가중치 입력 : 8 11 6
17번째 간선과 가중치 입력 : 9 11 7
18번째 간선과 가중치 입력 : 9 12 4
19번째 간선과 가중치 입력 : 10 13 5
20번째 간선과 가중치 입력 : 11 13 7
21번째 간선과 가중치 입력 : 12 13 2

-- <인접행렬 출력> --
0  3  6  7  5  0  0  0  0  0  0  0  0
0  0  0  0  0  10  0  0  0  0  0  0  0
0  0  0  0  0  5  0  0  0  0  0  0  0
0  0  0  0  0  2  0  0  0  0  0  0  0
0  0  0  0  0  5  0  0  0  0  0  0  0
0  0  0  0  0  0  3  2  2  0  0  0  0
0  0  0  0  0  0  0  0  0  3  5  4  0
0  0  0  0  0  0  0  0  0  3  6  0  0
0  0  0  0  0  0  0  0  0  0  7  4  0
0  0  0  0  0  0  0  0  0  0  0  0  5
0  0  0  0  0  0  0  0  0  0  0  0  7
0  0  0  0  0  0  0  0  0  0  0  0  2
0  0  0  0  0  0  0  0  0  0  0  0  0

시작 정점 : 1, 도착 정점 : 13

-- <각 정점 cost> --
2번째 cost : 3
3번째 cost : 6
4번째 cost : 7
5번째 cost : 5
6번째 cost : 9
7번째 cost : 12
8번째 cost : 11
9번째 cost : 11
10번째 cost : 14
11번째 cost : 17
12번째 cost : 15
13번째 cost : 17
```

```
-- <각 정점 d> --
2째 d : 1 -> 2
3째 d : 1 -> 3
4째 d : 1 -> 4
5째 d : 1 -> 5
6째 d : 1 -> 4 -> 6
7째 d : 1 -> 4 -> 6 -> 7
8째 d : 1 -> 4 -> 6 -> 8
9째 d : 1 -> 4 -> 6 -> 9
10째 d : 1 -> 4 -> 6 -> 8 -> 10
11째 d : 1 -> 4 -> 6 -> 7 -> 11
12째 d : 1 -> 4 -> 6 -> 9 -> 12
13째 d : 1 -> 4 -> 6 -> 9 -> 12 -> 13

최소 비용 : 17
최소 비용 경로 : 1 -> 4 -> 6 -> 9 -> 12 -> 13
```

```
정점 n과 간선 e 입력 : 6 7
1번째 간선과 가중치 입력 : 1 2 2
2번째 간선과 가중치 입력 : 1 3 3
3번째 간선과 가중치 입력 : 2 4 1
4번째 간선과 가중치 입력 : 2 5 3
5번째 간선과 가중치 입력 : 3 5 2
6번째 간선과 가중치 입력 : 4 6 2
7번째 간선과 가중치 입력 : 5 6 3

-- <인접행렬 출력> --
0  2  3  0  0  0
0  0  0  1  3  0
0  0  0  0  2  0
0  0  0  0  0  2
0  0  0  0  0  3
0  0  0  0  0  0

시작 정점 : 1, 도착 정점 : 6

-- <각 정점 cost> --
2번째 cost : 2
3번째 cost : 3
4번째 cost : 3
5번째 cost : 5
6번째 cost : 5
-- <각 정점 d> --
2째 d : 1 -> 2
3째 d : 1 -> 3
4째 d : 1 -> 2 -> 4
5째 d : 1 -> 2 -> 5
6째 d : 1 -> 2 -> 4 -> 6

최소 비용 : 5
최소 비용 경로 : 1 -> 2 -> 4 -> 6
```