

자료 구조 Lab 009 :

lab009.zip 파일 : LabTest.java lab009.java lab.in lab.out

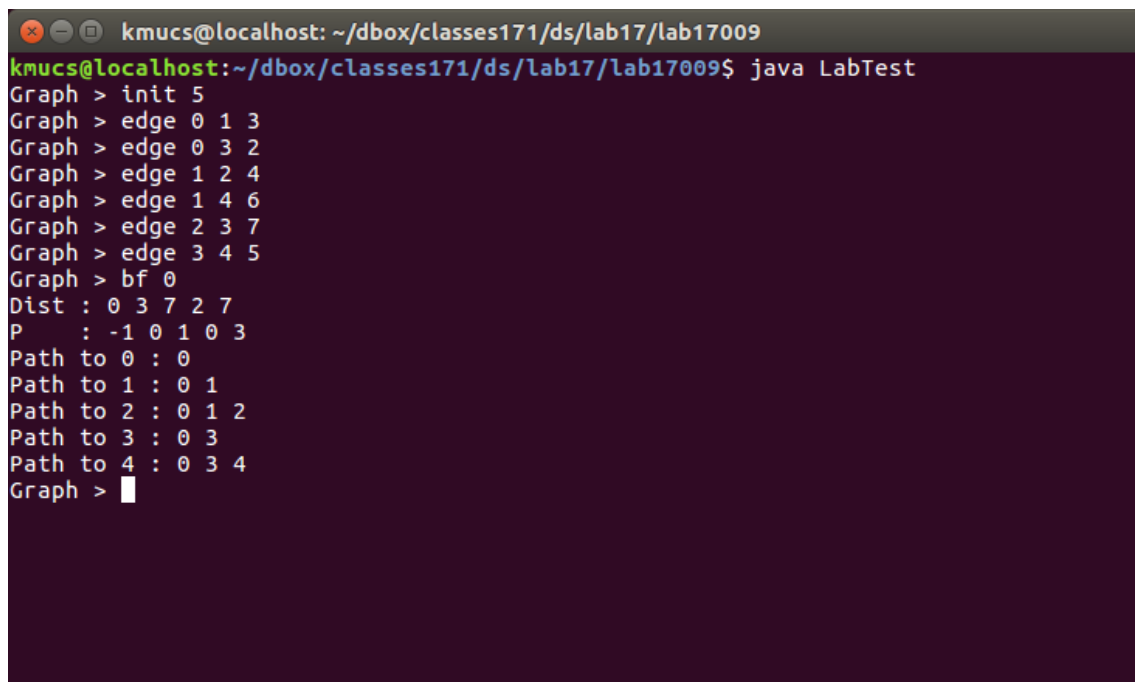
제출

lab009.java 를 학번.java 로 변경하여 이 파일 한 개만 제출할 것.

다음은 Cost-Adjacency Matrix를 이용하여 **Directed weighted** Graph에서 Bellman-Ford 알고리즘을 구현하는 내용이다.

이 프로그램에서는 우선 vertex의 개수를 입력하고, 그 다음에 edge를 구성하는 vertex pair를 edge별로 차례로 입력하여 graph를 구성한 후, **Bellman-ford** 알고리즘을 이용하여 single source-all destination shortest path problem을 해결한다.

수행 예는 다음과 같다.



```
kmucs@localhost: ~/dbbox/classes171/ds/lab17/lab17009
kmucs@localhost:~/dbbox/classes171/ds/lab17/lab17009$ java LabTest
Graph > init 5
Graph > edge 0 1 3
Graph > edge 0 3 2
Graph > edge 1 2 4
Graph > edge 1 4 6
Graph > edge 2 3 7
Graph > edge 3 4 5
Graph > bf 0
Dist : 0 3 7 2 7
P    : -1 0 1 0 3
Path to 0 : 0
Path to 1 : 0 1
Path to 2 : 0 1 2
Path to 3 : 0 3
Path to 4 : 0 3 4
Graph > █
```

사용자가 사용하는 명령어의 syntax는 다음과 같다. main() 함수에 정의되어 있다.

- init numofnodes

numofnodes는 vertex의 수를 의미하며, 각 vertex는 0부터 numofnodes -1까지의 번호를 가지게 된다.

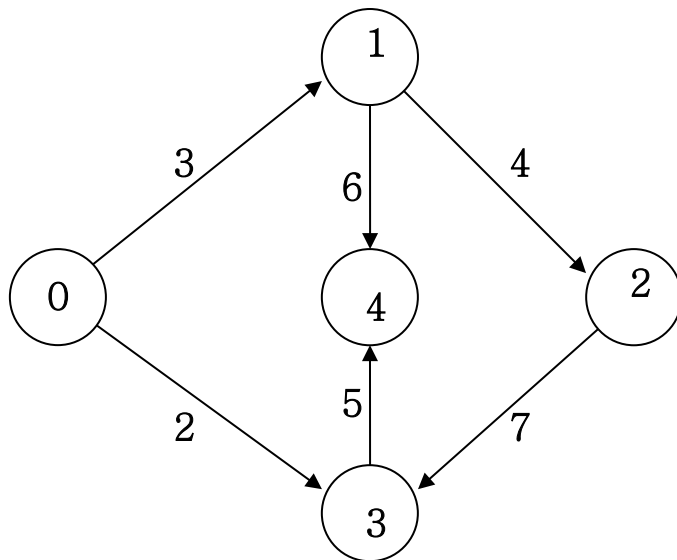
- edge v1 v2 cost

vertex v1과 vertex v2로 정의된 edge를 그래프에 추가한다. 그 edge의 cost는 이 명령어의 세 번째 파라미터인 cost가 된다.

- bf v

vertex v를 source로 하는 single source - all destination problem을 Bellman-Ford 알고리즘을 이용하여 dist와 p 배열을 계산하고, 이 두 배열과 각 destination별 경로를 보여준다.

위의 화면과 같이 입력할 경우 내부적으로 다음과 같은 cost-adjacency matrix가 구성된다.



위 그래프에 해당하는 cost-adjacency matrix는 아래와 같다.

$$Cost = \begin{bmatrix} 0 & 3 & 0 & 2 & 0 \\ 0 & 0 & 4 & 0 & 6 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

이 내용을 구현하기 위해 다음 두 가지 함수를 구현해야 한다.

- void Edge(int v1, int v2, int cost);

v1과 v2는 한 edge를 구성하는 vertex를 의미한다. 이 함수는 이 edge를 그래프에 추가하는 일을 한다. 클래스 Graph에는 Cost는 2차원 배열이 Cost-Adjacency

Matrix를 구성하는데, v1과 v2에 의해 결정되는 이차원 배열 Cost의 entry를 cost로 수정해야 한다. 이 그래프가 **Directed** Graph임에 주의한다.

- `void BellmanFord(int v);`

vertex v를 source로 하는 single source - all destination problem을 Bellman-Ford 알고리즘을 이용하여 dist와 p 배열을 계산한다.

소스 코드에 정의된 Class Graph에는 numofnodes 라는 변수가 있는데 이 변수가 노드의 수를 의미한다. 따라서 노드는 0부터 numofnodes-1 까지의 값을 가진다. 이 알고리즘을 구현하는데 있어서 2차원 배열이 아닌 **1차원 배열을 이용하여** 구현하도록 한다. 즉 아래와 같은 알고리즘을 사용하여 구현하도록 한다.

$$d(v) = \min\{d(v), \min\{d(w) + \text{length of edge } (w,v)\}\}$$

따라서 d[v,k] 대신에 d[v]를 사용하고, p[v,k] 대신에 p[v]를 사용한다.

- `String OutPath(int i);`

p배열을 이용하여 소스 노드로부터 목적지 노드 i까지 가는 경로를 리턴하는 재귀 함수이다. p[i]가 -1이면 소스 노드이기 때문에 i를 리턴하고, 그렇지 않을 경우에는 p[i]까지의 경로를 OutPath() 함수로 리턴 받은 후, 그 다음 i를 붙여서 리턴하면 된다. 노드와 노드 사이에는 공백 문자 하나가 들어간다.

프로그램 결과 테스트

```
$ diff aaa lab.out
```

또는

```
$ diff -i --strip-trailing-cr -w aaa lab.out
```