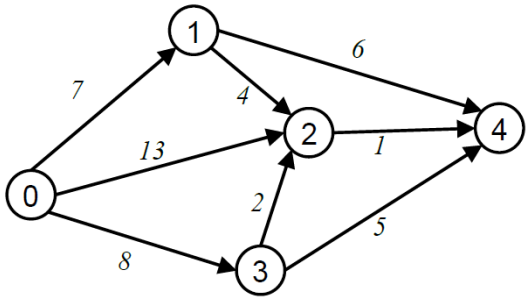
알고리즘 Quiz03

충남대학교 컴퓨터공학과
알고리즘 04 분반
학번: 201701975
이름: 구건모
제출일: 2021.11.05

#문제 1

** 문제에서 제시된 G2 graph 이미지*



/* 변화 상태를 보여주는 Table 에서 변화를 시각적으로 편하게 보실 수 있도록 이전 iteration 과 달라진(변경된) 부분을 highlight 하였습니다. */

반복 직전 초기 상태					
	[0]	[1]	[2]	[3]	[4]
distance	0	7	13	8	∞
found	TRUE	FALSE	FALSE	FALSE	FALSE
path	-1	0	0	0	0

> 반복 직전 초기 상태

최초에는 출발 Vertex 인 0의 found 값을 True로 설정하고, 출발 Vertex는 이전 Vertex가 존재하지 않으므로 path를 -1로 초기화 합니다. 나머지 1 ~ 4 Vertex는 Found 에 출발 Vertex만 True인 상태이므로 0번 Vertex만을 이용하여 distance를 계산한 값을 넣어줍니다. 1,2,3번 Vertex는 출발 vertex와 이어져 있는 상태이기 때문에 distance 값을 각각의 Edge의 Weight 값으로 초기화하고, 4번 Vertex는 출발 Vertex 에서 접근할 수 없기 때문에 ∞ 값으로 초기화 합니다. 현재 상태에서 1 ~ 4 번 Vertex 이전 Vertex는 출발 Vertex인 0번 Vertex 이므로 0 값으로 초기화 합니다.

i=0, u=1					
	[0]	[1]	[2]	[3]	[4]
distance	0	7	11	8	13
found	TRUE	TRUE	FALSE	FALSE	FALSE
path	-1	0	1	0	1

> i = 0, u = 1

첫번째 Iteration(i=0)에서는 Found가 false인 Vertex 중에서 가장 distance 값이 작은 Vertex를 Choose 하므로, 1 ~ 4 번 Vertex 중 1번 Vertex가 선택되어 Found[1] = true 로 설정되고, 1번 Vertex가 추가되므로 인해 기존의 distance 값이 변경될 수 있으므로 기존의 distance와 1번을 이용하여 destination까지 갔을 때의 distacne를 비교하여 더 짧은(작은) 값을 distance로 업데이트 합니다. 2번 Vertex의 경우 0번에서 2번으로 갈 경우 13의 distance를 가지지만 1번을 거쳐 2번에 도달할 경우 7+4 = 11 의 distance를 가지므로 기존의 distance인 13보다 더 짧은 11의 distance로 업데이트 합니다. 따라서 distance[2] = 11로 업데이트 되고 이와 동시에 이전 Vertex도 1번 Vertex로 변경되므로 path[2] = 1 으로 변경됩니다. 4번 Vertex 또한 기존에는 도달할 수 없었지만, 1번 Vertex가 추가되면서 도달할 수 있게 되므로, 같은 원리로 distance[4] = 13, path[4] = 1 로 업데이트 됩니다.

i=1, u=3					
	[0]	[1]	[2]	[3]	[4]
distance	0	7	10	8	13
found	TRUE	TRUE	FALSE	TRUE	FALSE
path	-1	0	3	0	1

> i = 1, u = 3

두번째 Iteration(i=1) 에서는 found가 false인 Vertex 중에서 가장 distance가 작은 값인 3번 Vertex가 추가됩니다. 따라서 found[3] = true 로 설정되고 3번 Vertex를 거쳐 2번 Vertex에 갈 수 있는 경로를 10으로 줄일 수 있으므로 distance[2] = 10 으로 변경합니다. 또한 2번 Vertex의 이전 Vertex도 3번 Vertex로 바뀌었으므로 path[2] = 3 으로 변경합니다.

i=2, u=2					
	[0]	[1]	[2]	[3]	[4]
distance	0	7	10	8	11
found	TRUE	TRUE	TRUE	TRUE	FALSE
path	-1	0	3	0	2

> i = 2, u = 2

3번째 Iteration(i=2) 에서는 found가 false인 Vertex중에서 가장 작은 distance를 가지는 2번 Vertex가 u가 되므로, found[2] = true로 설정해주고, 2를 거쳐 4번 Vertex로 갈 경우 기존 distance 보다 더 작은 값인 11 값을 가지므로, distance[4] = 11 로 설정해주고 4번 Vertex의 이전 Vertex도 2번 Vertex가 되므로 path[4] = 2 로 변경됩니다.

#문제 2

```
public void showPath(int destinationVertex) {
    if (!(destinationVertex > this.graph().numberOfVertices()-1 || destinationVertex < 0)) {
        /* 유효하지 않은 Vertex 입력을 방지, 0 <= destinationVertex < graph().numberOfVertices()-1 를 만족해야 유효한 Vertex로 판단함.  */
        if (destinationVertex != this.sourceVertex()) {
            /* destinationVertex가 sourceVertex 일 경우 loop 할 필요가 없으므로 sourceVertex가 아닌 경우에만 loop */
            int currentVertex = destinationVertex;
            /* loop에서 현재 Vertex를 나타내기 위한 currentVertex, 초기에 destinationVertex 부터 시작 */
            while (currentVertex != this.sourceVertex()) {
                /* currentVertex가 sourceVertex가 될때까지 path를 이용해 이전 Vertex를 currentVertex로 설정하는 식으로 이동할 것임. */
                System.out.print(currentVertex + "-");
                /* currentVertex + "-" 출력 */
                currentVertex = this.path()[currentVertex]; /* 이전 Vertex를 path를 통해 가져와 이전 Vertex에 대해 다음 loop 때 수행하게 됨 */
                if (currentVertex == this.sourceVertex()) {
                    /* 이동한 이전 Vertex가 sourceVertex 일 경우 sourceVertex 출력 후 while문의 조건에 따라 loop가 종료됨 */
                    System.out.print(currentVertex);
                }
            }
        } else if (destinationVertex == this.sourceVertex()) {
            /* destinationVertex가 sourceVertex일 경우 바로 출력 */
            System.out.print(destinationVertex);
        }
    } else {
        /* 유효하지 않은 Vertex를 입력했을 경우 */
        System.out.print("[!] 유효하지 않은 destinationVertex 입니다.");
    }
}
```

/* 문제에서 MVC 설계를 고려하지 않아도 된다고 명시해주셔서, MVC 설계를 고려하지 않고 기능적인 측면 자체만을 구현하는 것을 목적으로 하였습니다. */

우선 destinationVertex가 유효한지 판단 후 유효한 Vertex이면 destinationVertex가 sourceVertex와 같은지 판단합니다. 같을 경우 loop 할 필요가 없으므로 바로 sorceVertex를 출력후 method를 종료하고, 같지 않을 경우 currentVertex 라는 변수를 선언하여 loop를 돌면서 sourceVertex 까지 역으로 출력합니다. currentVertex는 loop에서 현재

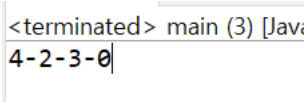
Vertex를 나타내기 위해 사용한 변수로 destinationVertex 로부터 시작해서 path를 이용해 sourceVertex가 나올때까지 이전 Vertex들로 이동하며 출력하는 식의 구조입니다.

>> [참고] 문제 2의 코드 실행결과 예시

예시로 문제에 제시된 G2 graph를 이용해 테스트 해보면 다음과 같이 작동합니다.

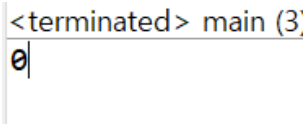
- showPath(4) 실행 결과

: destinationVertex가 4번이므로 path를 destinationVertex ~ sourceVertex 까지 역으로 출력합니다.



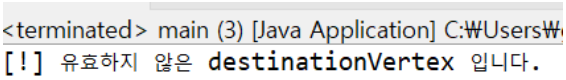
- showPath(0) 실행 결과 (SourceVertex를 destinationVertex로 넣은 경우)

: 현재 0이 sourceVertex 이므로 0을 출력하고 끝내게 됩니다.



- showPath(-1) 또는 showPath(5) 실행 결과 (예외처리)

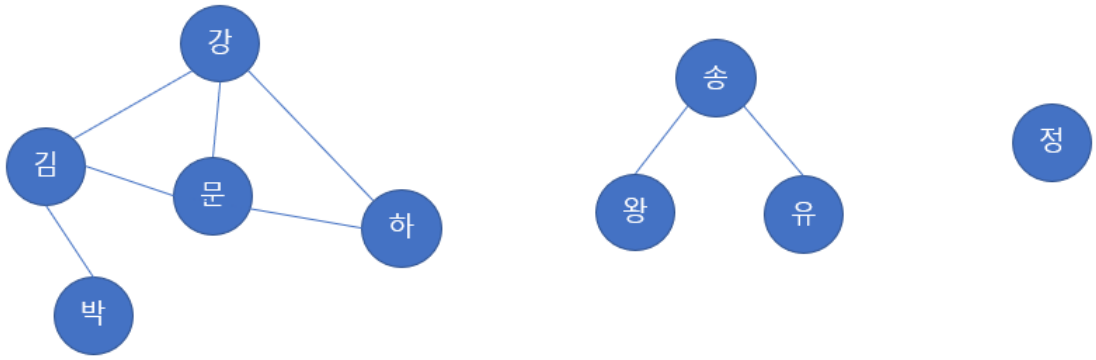
: -1 과 5와 같이 유효하지 않은 Vertex를 인자로 받은 경우 오류 문구를 출력합니다.



#문제 3

(Equivalance relation은 Transitive, Symmetric, Reflexive 세 가지 성질을 만족해야 합니다. 동문관계를 나타내는 집합 E는 Equivalance Relation 이므로 집합 E의 원소로 직접 표현되어 있지 않아도 Transitive, Reflexive, Symmetric pairs가 암묵적으로 존재한다고 가정합니다. 하지만 수업시간 내용 중 Graph에 relation을 추가할 때, Symmetric Pairs 만 표현하고 Transitive / Reflexive Pairs 는 그래프 표현 과정에서 생략해도 된다고 하셔서 집합 E에 있는 관계만 직접적으로 표현하고 E에 직접 명시되지 않은 경우, 생략된 경우가 있습니다.)

V와 E를 이용하여 E에 명시된 관계만을 Undirected Graph를 그려보면 학생들의 집합 V와 동문관계 E 가 의미하는 Undirected Graph는 다음과 같이 나타낼 수 있습니다.



#문제 4

학생들의 집합 V 와 동문관계 E 로부터 찾을 수 있는 동등 클래스는 다음과 같습니다. 동문관계(E)를 가지는 학생들(V)을 같은 클래스로 묶어 classify 한 결과입니다.

동등클래스 => {강,김,하,문,박}, {왕,송,유}, {정}

감사합니다.

충남대학교 컴퓨터공학과 알고리즘 04분반

2017019755학번 구건모