[제 4 주 실습]

최단경로

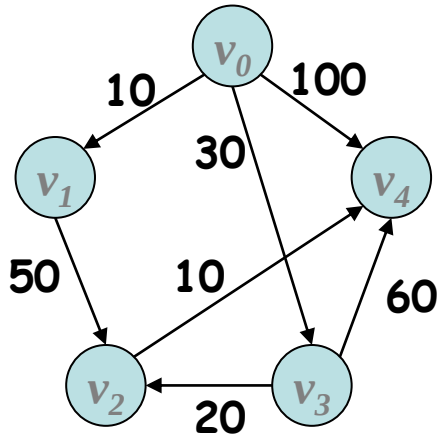
강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

[문제 4]
Dijkstra's Shortest Paths

문제 개요

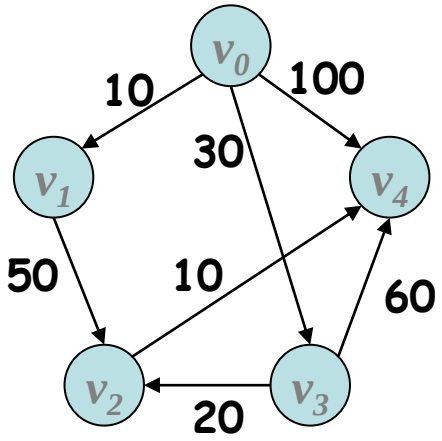
- 입력:
  - Weighted undirected graph
- 출력:
  - 주어진 그래프로부터 최단경로 (Shortest Paths)를 찾는다.



Path	Length
$v_0 - v_1$	10
$v_0 - v_3$	30
$v_0 - v_3 - v_2$	50
$v_0 - v_3 - v_2 - v_4$	60

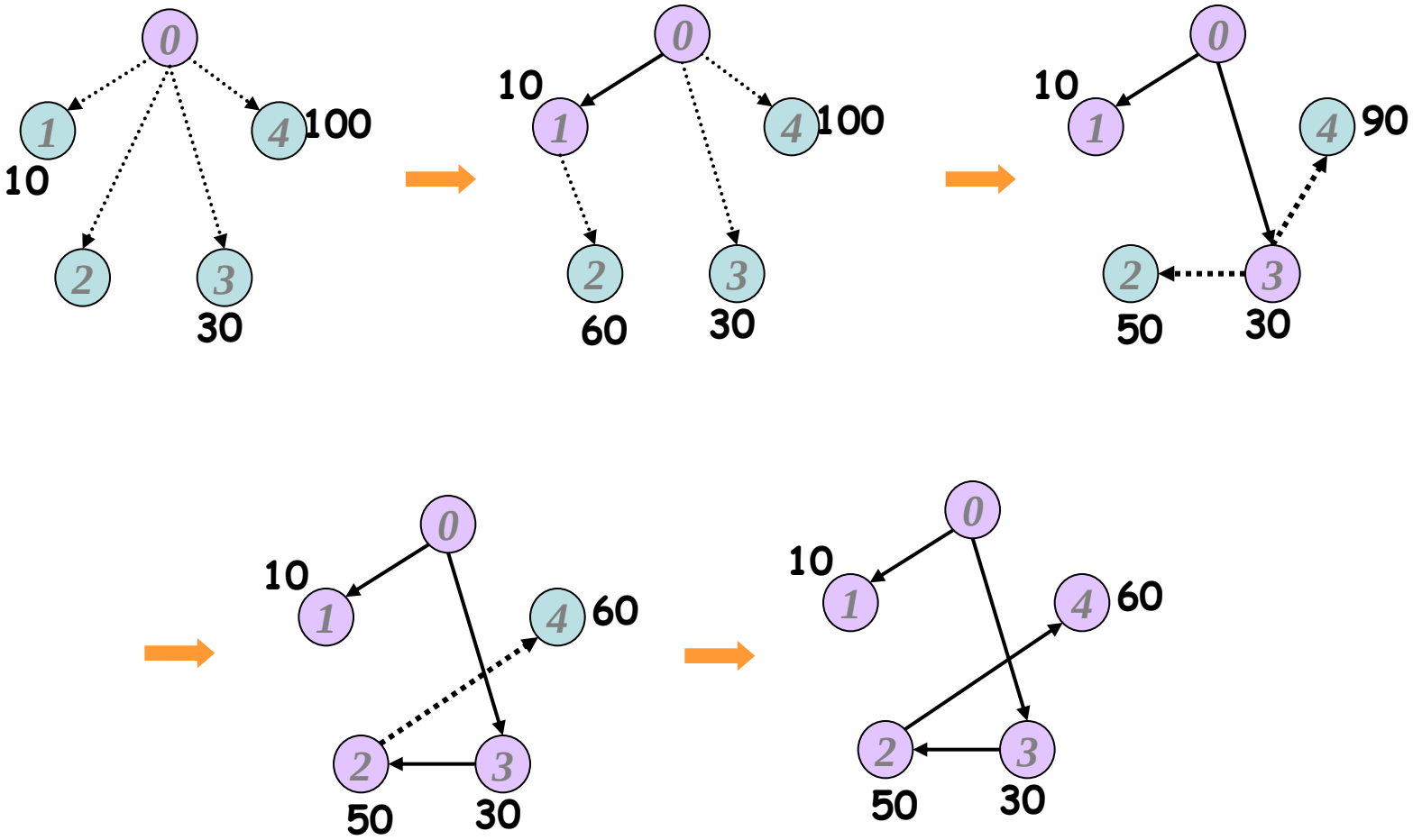
Single Source All Destinations

Dijkstra's Algorithm



Path	Length
$v_0 - v_1$	10
$v_0 - v_3$	30
$v_0 - v_3 - v_2$	50
$v_0 - v_3 - v_2 - v_4$	60

Iteration	S	u	distance[1]	distance[2]	distance[3]	distance[4]
초기화	{0}	-	**10**	∞	30	100
1	{0,1}	1	10	60	**30**	100
2	{0,1,3}	3	10	**50**	30	90
3	{0,1,3,2}	2	10	50	30	**60**
-	{0,1,3,2,4}	4	10	50	30	60


0
1
10
2
3
30
4
100
0
10
1
2
60
3
30
4
100
0
10
1
2
50
3
30
4
90
0
10
1
2
50
3
30
4
60
0
10
1
2
50
3
30
4
60

구현에서의 주요 내용

- 그래프의 표현
  - Adjacency Matrix로 구현한다.
  - 선택사항:
    별도의 동일한 프로그램으로 Adjacency List로도 구현한다. (+α)

- Shortest Paths 찾는 알고리즘
  - Dijkstra의 알고리즘

입력

- 키보드로부터 그래프 정보를 입력 받는다.
  - Vertex의 개수
    - Vertex는 0부터 시작하는 정수로 나타낸다.
  - Edge의 개수
    - 입력되는 그래프의 edge의 개수
  - Weighted Edge들
    - cost가 함께 주어져 있는 edge들
    - 미리 주어진 edge 수 만큼 입력 받는다.
    - Edge는 vertex의 쌍과 cost로 나타낸다.
      - (tailVertex, headVertex, cost)

출력

- 입력된 그래프

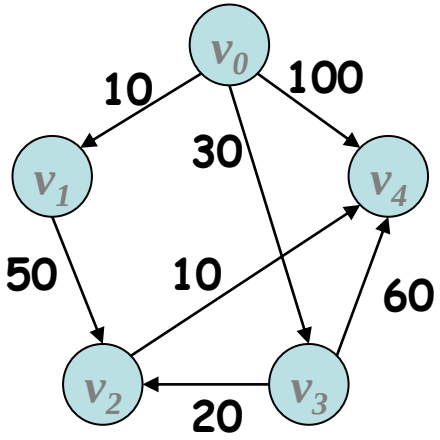
  [0] -> 1(10) 3(30) 4(100)

  [1] -> 2(50)

  [2] -> 4(10)

  [3] -> 2(20) 4(60)

  [4] ->



출력

- 찾아진 최단경로와 각 노드별 최단 경로

최단경로는 다음과 같습니다.

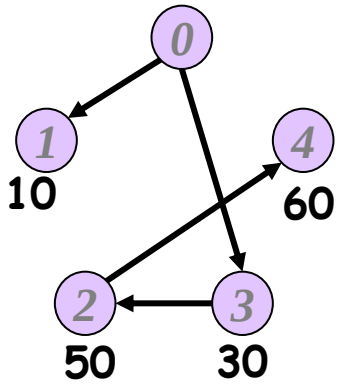
distance [0] = 0

distance [1] = 10

distance [2] = 50

distance [3] = 30

distance [4] = 60



-1	0	3	0	2

최단경로는 다음과 같습니다.

0 - 1

0 - 3 - 2

0 - 3

0 - 3 - 2 - 4

❑출력의 예

```
- 그래프의 vertex수와 edge수를 입력 받아야 합니다.
? 그래프의 vertex 수를 입력 하시오:5
? 그래프의 edge 수를 입력 하시오:7
- 그래프의 edge를 반복하여 7 개 입력 받아야 합니다.
  하나의 edge는 (vertex1 vertext2 cost)의  순서로 표시됩니다.
? Edge를 입력하시오: 0 1 10
? Edge를 입력하시오: 1 2 50
? Edge를 입력하시오: 3 2 20
? Edge를 입력하시오: 2 4 10
? Edge를 입력하시오: 3 4 60
? Edge를 입력하시오: 0 4 100
? Edge를 입력하시오: 0 3 30

! 입력된 그래프는 다음과 같습니다:
생성된 그래프 :
[0] -> 1(10)  3(30)  4(100)
[1] -> 2(50)
[2] -> 4(10)
[3] -> 2(20)  4(60)
[4] ->
```

```
최단경로는 다음과 같습니다.
distance [0] = 0
distance [1] = 10
distance [2] = 50
distance [3] = 30
distance [4] = 60

최단경로는 다음과 같습니다.
0 - 1
0 - 3 - 2
0 - 3
0 - 3 - 2 - 4
```

이 과제에서 필요한 객체는?

- Application
  - AdjacencyMatrixGraph
  - DijkstraShortestPaths
    - ArrayList
  - Edge

Application의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public void run()

DS2_04_학번_이름 Class의 구조

```
/* 항상 사용하는 main Class 구조
 * 과제 제출시 반드시 포함 되어 있어야 함*/
public class DS2_04_학번_이름 {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Application application = new Application();
        application.run ();
    }
}
```

AdjacencyMatrixGraph의 공개 함수는?

- 사용자에게 필요한 함수
  - [문제3]의 것을 복사
  - public AdjacencyMatrixGraph (int givenNumOfVertices)
  - public boolean doesVertexExist (int aVertex)
  - public boolean doesEdgeExist (Edge anEdge)
  - public int numOfVertices ()
  - public int numOfEdges ()
  - public boolean addEdge(Edge anEdge)
  - public AdjacentVerticesIterator adjacentVertircesIterator(int givenVertex)
  - public class AdjacentVerticesIterator
  - public int maxCostSize()
  - public int showVertexinfo(int afromVertex, int atoVertex)

Edge의 공개 함수는?

- 사용자에게 필요한 함수
  - [문제3]의 것을 복사
  - public Edge (int givenTailVertex, int givenHeadVertex, int givenCost)
  - public void setTailVertex (int aVertex)
  - public int tailVertex ()
  - public void setHeadVertex (int aVertex)
  - public int headVertex ()
  - public void setCost (int aCost)
  - public int cost()
  - public boolean sameEdgeAs (Edge anEdge)

DijkstraShortestPaths의 공개 함수는?

- 사용자에게 필요한 함수
  - public DijkstraShortestPaths(AdjacencyMatrixGraph givenGraph)
  - public boolean perform(int sourceVertex)
  - public ArrayList<Integer> findPath(int start,int end, ArrayList<Integer> pathList)
  - public ShortestPathsIterator shortestPathsIterator()
  - public class ShortestPathsIterator

ArrayList〈T〉의 공개 함수는?

- 사용자에게 필요한 함수
  - public ArrayList()
  - public ArrayList(int givenMaxSize)

  - public boolean isFull()
  - public boolean isEmpty()
  - public int size()

  - public boolean add(T anElement)

  - public ArrayListIterator arrayListIterator()
  - public class ArrayListIterator

Class “Application“

Application – 비공개 인스턴스 변수

```
import java.util.Scanner;

public class Application {
    private Scanner _scanner;
    private AdjacencyMatrixGraph _graph;
    private DijkstraShortestPaths _dijkstrashortestPaths;
    ......
```

Application의 공개 함수 run()의 구현

```
public void run(){
    if ( this.inputAndMakeGraph() ) {
        this.showGraph() ;

        this._dijkstrashortestPaths = new DijkstraShortestPaths(this._graph);
        this._dijkstrashortestPaths.perform(0);
        this.showDijkstraShortestDistance();
        this.showDijkstraSortestPath();
    }
    else {
        System.out.println("₩n₩n[오류]그래프 삽입 실패") ;
    }
}
```

Application의 Private Method

- Application의 Private Member function의 사용법
  - private boolean inputAndMakeGraph()
    - 키보드로부터 그래프를 입력 받아 그래프 객체로 저장한다.
    - 그래프가 정상적으로 입력되었으면 true, 아니면 false를 얻는다.
    - [문제3]에서 사용한 것에, Edge를 삽입하는 부분을 삭제
  - private void showGraph()
    - 입력된 그래프를 보여준다.
    - [문제3]과 동일
  - public void showDijkstraShortestDistance()
    - 최단경로의 Distance를 출력한다.
  - public void showDijkstraSortestPath()
    - 모든 Node의 최단 경로를 출력한다.

Member Functions의 구현

- Application의 Private Member function의 구현
  - private boolean inputAndMakeGraph()
    - [문제3]의 inputAndMakeGraph를 수정하여 사용
    - [문제3]에서 사용한 MinPriorityQ 삽입하는 부분을 삭제
  - private void showGraph()
    - [문제3]의 showGraph을 사용

Member Functions의 구현

- Application의 Private Member function의 구현
  - public void showDijkstraShortestDistance()
    - DijkstraShortestPaths의 ShortestPathsIterator형인 dijkstraShortestPathsIterator(변수명)를 _dijkstrashortestPaths의 shortestPathsIterator로 초기화한다.
    - dijkstraShortestPathsIterator가 다음 값(hasNext)이 있으면 다음 값(next)을 받아와 적절한 출력을 한다.

Member Functions의 구현

- Application의 Private Member function의 구현
  - public void showDijkstraSortestPath()

```
@SuppressWarnings("rawtypes")
public void showDijkstraSortestPath()
{
    System.out.println("최단경로는 다음과 같습니다.");
    for(int i = 1; i < this._graph.numOfVertices(); i++)
    {
        ArrayList<Integer> resultShortestPath = this._dijkstrashortestPaths.findPath(0, i, null);
        ArrayList.ArrayListIterator arrayListIterator = resultShortestPath.arrayListIterator();

        System.out.print(arrayListIterator.next() + " ");
        while(arrayListIterator.hasNext())
        {
            System.out.print(" - " + arrayListIterator.next());
        }
        System.out.println();
    }
    System.out.println();
}
```

Class “AdjacencyMatrixGraph“

AdjacencyMatrixGraph의 수정

- [문제3] 에서 사용한 것을 복사해 와서 Cost 추가
  - 맴버 변수 추가
    - private static final int MaxCostSize = 1000;
  - 기존 함수에 수정
    - public AdjacencyMatrixGraph(int givenNumOfVertices)
      - this._adjacency 초기화 값을 this.MaxCostSize으로 설정
    - public boolean addEdge(Edge anEdge)
      - 방향성을 가지는 그래프로 삽입 부분 수정
    - public boolean doesEdgeExist (Edge anEdge)
      - 현재 삽입 하려는 위치의 Edge가 this.MaxCostSize인지 확인
      - 기존 방향성을 가지지 않는 그래프에서 검사하는 부분 삭제

AdjacencyMatrixGraph의 수정

- [문제3] 에서 사용한 것을 복사해 와서 Cost 추가
  - 새로운 함수 추가
    - public int maxCostSize()
      - MaxCostSize 반환
    - public int showVertexinfo(int afromVertex, int atoVertex)
      - _adjacency의 afromVertex, atoVertex번째의 cost 반환

Class “DijkstraShortestPaths”

Dijkstra's Algorithm

```
public void  shortestPaths (int sourceVertex) {
    int i, u, w ;
    boolean[] found = new boolean[this._numOfVertices] ;
    for (i = 0; i < this._numOfVertices ; i++) {
        found[i] = false ;
        this._distance[i] = this._cost[sourceVertex][i] ;
     }
     found[sourceVertex] = true ;
     this._distance[sourceVertex] = 0 ;
     for (i=0; i < this._numOfVertices-2; i++) {
        u = choose(found);
        found[u] = true;
        for (w = 0; w < this._numOfVertices ; w++) {
            if ( !found[w] ) {
                if ( this._distance[w] > this._distance[u] + this._cost[u][w] )
                    this._distance[w] = this._distance[u] + this._cost[u][w];
            }
        }
    }
}
```

Time Complexity : $O(n^2)$

Generation of Vertex Sequences

- Use another array path[] of vertices.

  path[u] ≡ the vertex immediately before u in the shortest path

- Initialize path[u] = v for all u ≠ v (v is the source) ;

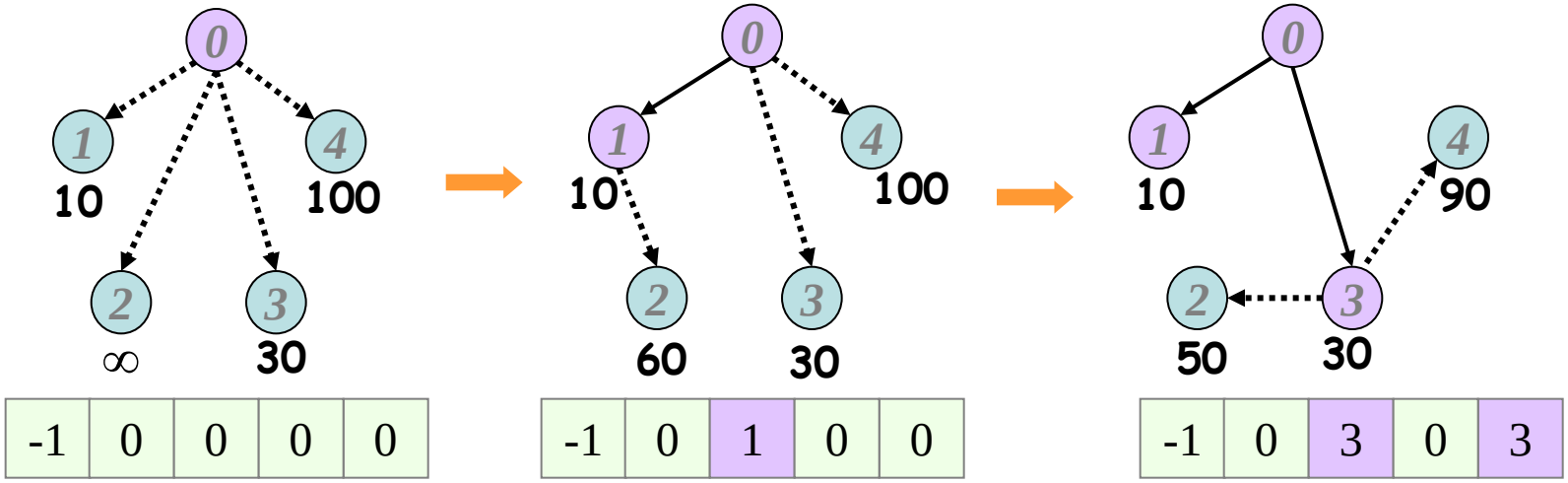
  path[v] = -1 ;

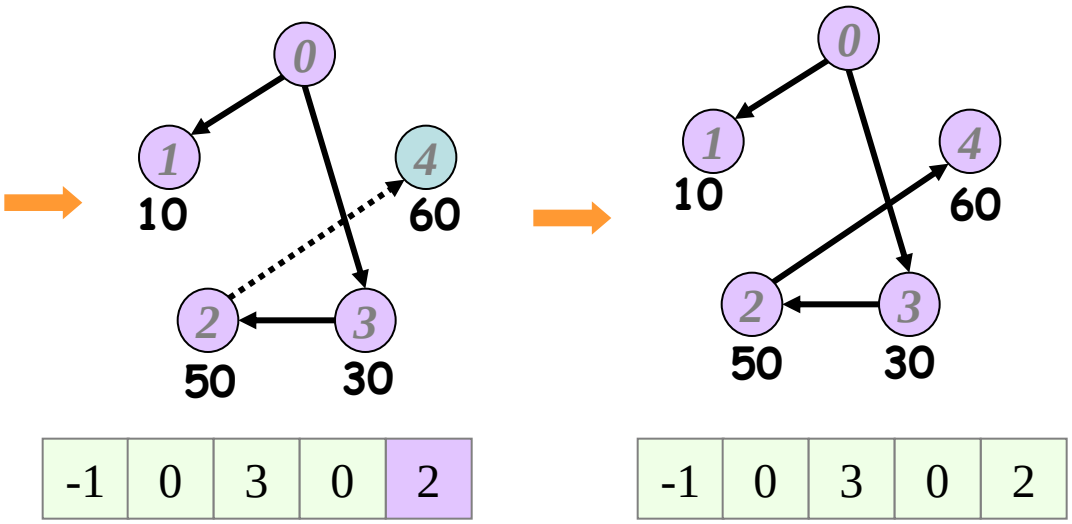
- And then,

```
If ( this._distance[w] > this._distance[u]+ this._cost[u][w] ) {
        this._distance[u] = this._distance[u]+ this._cost[u][w] ;
        this._path[w] = u ;
}
```

- Upon termination,
  the paths can be found by tracking backward.

Finding Vertex Sequences of Paths:





The shortest paths in reverse vertex order

- 1-0
- 2-3-0
- 3-0
- 4-2-3-0

All Pairs Shortest Paths

- Apply the Dijkstra's Algorithm n times : $O(n^3)$
- A simpler algorithm using Cost Adjacency Matrix.

$\text{cost}[i][i] = 0$
$\text{cost}[i][j] = \text{cost of edge } <i, j> \in \text{E}$
$\text{cost}[i][j] = \infty \text{ if } <i, j> \notin \text{E}$

  - Define $\text{A}^k[i][j]$ to be the cost of the shortest path from i to j going through no intermediate vertex of index greater than k.
  - Then $\text{A}^{n-1}[i][j]$ will be the cost of the shortest path from i to j in G.
  - $\text{A}^{-1}[i][j]$ is just $\text{cost}[i][j]$.

비공개 인스턴스 변수

```
public class DijkstraShortestPaths{
    private int _numOfVertices ;
    private int [] _distance; // 시작 노드부터의 최단 경로 거리
    private AdjacencyMatrixGraph _graph;
    private int [] _path; // 경유 노드
```

DijkstraShortestPaths 의 멤버 함수는?

- DijkstraShortestPaths 의 Public Member function의 사용법
  - public DijkstraShortestPaths(AdjacencyMatrixGraph givenGraph)
    - ShortestPaths 생성자
  - public boolean perform(int sourceVertex)
    - shortestPath를 찾는다.
  - public ArrayList<Integer> findPath(int start,int end, ArrayList<Integer> pathList)
    - Start Vertex부터 end Vertex까지의 최단 경로를 찾는다.
  - public ShortestPathsIterator shortestPathsIterator()
  - public class ShortestPathsIterator

DijkstraShortestPaths 의 멤버 함수는?

- DijkstraShortestPaths 의 Public Member function의 구현
  - public DijkstraShortestPaths(AdjacencyMatrixGraph givenGraph)
    - givenGraph를 this._graph에 저장
    - this._graph의 numOfVertices을 this._numOfVertices에 저장
    - this._distance를 this._numOfVertices만큼 생성
    - this._path를 this._numOfVertices만큼 생성

DijkstraShortestPaths 의 멤버 함수는?

- DijkstraShortestPaths 의 Public Member function의 구현
  - public void perform(int sourceVertex)
    - 앞에 설명한 Dijkstra 알고리즘을 참고하여 프로그램을 완성한다.
    - Cost를 얻어오는 것은 _graph의 showVertexinfo(u, w))를 사용한다.

DijkstraShortestPaths 의 멤버 함수는?

- DijkstraShortestPaths 의 Public Member function의 구현
  - public ArrayList<Integer> findPath(int start,int end, ArrayList<Integer> pathList)

```
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;
}
```

DijkstraShortestPaths 의 멤버 함수는?

- DijkstraShortestPaths 의 Public Member function의 구현
  - public ShortestPathsIterator shortestPathsIterator()
    - Inner Iterator class 인 ShortestPathsIterator를 생성하여 반환한다.

DijkstraShortestPaths 의 멤버 함수는?

- DijkstraShortestPaths 의 Public Member function의 구현
  - public class ShortestPathsIterator
    - private 맴버 변수
      - private int _nextElement;
    - private ShortestPathsIterator()
      - _nextElement를 0으로 초기화
    - public boolean hasNext()
      - this._nextElement이 _numOfVertices과 같은지 확인하여 반환
      - 같을 경우 false 반환
    - public int next()
      - 현재 _nextElement의 값을 임시 변수에 저장하고
      - _nextElement의 값을 하나 증가
      - 이전 _nextElement의 값을 저장 했던 임시 변수를 반환

DijkstraShortestPaths 의 멤버 함수는?

- DijkstraShortestPaths 의 private Member function의 구현
  - private int choose(boolean [] givenFound)
    - 가장 적은 값을 가지는 distance를 Vertex를 구하여 반환
    - 사용 방법은 perform을 참고하여 구현하세요.

Class “ArrayList”

비공개 인스턴스 변수

```
public class ArrayList<T> {
    private static final int DEFAULT_INITIAL_CAPACITY = 20 ;
    private int _maxSize;
    private int _size;
    private T[] _element;
```

ArrayList의 멤버 함수는?

- ArrayList의 Public Member function의 사용법
  - public ArrayList()
  - public ArrayList(int givenMaxSize)
    - ArrayList 생성자
  - public boolean isFull()
    - ArrayList가 가득 차 있는 지 확인
  - public boolean isEmpty()
    - ArrayList가 비어 있는지 확인
  - public int size()
    - ArrayList의 size를 확인
  - public boolean add(T anElement)
    - anElement를 삽입
  - public ArrayListIterator arrayListIterator()
  - public class ArrayListIterator

ArrayList의 멤버 함수는?

- ArrayList의 Public Member function의 구현
  - public ArrayList()
  - public ArrayList(int givenMaxSize)
    - DEFAULT_INITIAL_CAPACITY 혹은 받은 givenMaxSize를 this._maxSize에 저장
    - this._size를 0으로 초기화
    - this._element 를 this._maxSize만큼 생성
      - this._element = (T[]) new Object[this._maxSize];

ArrayList의 멤버 함수는?

- ArrayList의 Public Member function의 구현
  - public boolean isFull()
    - this._size와 this._maxSize를 비교하여 반환
    - 두 값이 같을 경우 true
  - public boolean isEmpty()
    - this._size와 0를 비교하여 반환
    - 0일 경우 true
  - public int size()
    - this._size를 반환

ArrayList의 멤버 함수는?

- ArrayList의 Public Member function의 구현
  - public boolean add(T anElement)
    - 가득 차 있지 않을 경우 anElement 값을 삽입하고 true를 반환
    - 가득 차 있을 경우 false를 반환

ArrayList의 멤버 함수는?

- ArrayList의 Public Member function의 구현
  - public ArrayListIterator arrayListIterator()
    - Inner Iterator class 인 ArrayListIterator를 생성하여 반환한다.

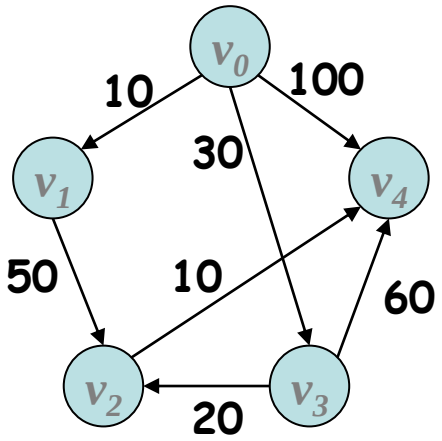
ArrayList의 멤버 함수는?

- ArrayList의 Public Member function의 구현
  - public class ArrayListIterator
    - private 맴버 변수
      - private int _nextElement;
    - private ArrayListIterator()
      - _nextElement를 0으로 초기화
    - public boolean hasNext()
      - this._nextElement이 _size와 같은지 확인하여 반환
      - 같을 경우 false 반환
    - public int next()
      - 현재 _nextElement의 값을 임시 변수에 저장하고
      - _nextElement의 값을 하나 증가
      - 이전 _nextElement의 값을 저장 했던 임시 변수를 반환

[문제 4+]
Floyd Shortest Paths

문제 개요

- 입력:
  - [문제 4]와 동일
- 출력:
  - [문제4]의 출력
  - 최소비용 신장 트리를 이루는 간선들의 집합인 Floyd의 Shortest Paths 출력



❑출력의 예 [1]

```
- 그래프의 vertex수와 edge수를 입력 받아야 합니다.
? 그래프의 vertex 수를 입력 하시오:3
? 그래프의 edge 수를 입력 하시오:4
- 그래프의 edge를 반복하여 4 개 입력 받아야 합니다.
  하나의 edge는 (vertex1 vertext2 cost)의  순서로 표시됩니다.
? Edge를 입력하시오: 0 1 8
? Edge를 입력하시오: 1 0 3
? Edge를 입력하시오: 0 2 5
? Edge를 입력하시오: 2 1 2

! 입력된 그래프는 다음과 같습니다:
생성된 그래프 :
[0] -> 1(8)  2(5)
[1] -> 0(3)
[2] -> 1(2)

최단경로의 Distance는 같습니다.
distance [0] = 0
distance [1] = 7
distance [2] = 5

최단경로는 다음과 같습니다.
0  - 2 - 1
0  - 2

모든 최단경로는 다음과 같습니다.
[0] 0 7 5
[1] 3 0 8
[2] 5 2 0
```

❑출력의 예 [2]

```
- 그래프의 vertex수와 edge수를 입력 받아야 합니다.
? 그래프의 vertex 수를 입력 하시오:5
? 그래프의 edge 수를 입력 하시오:7
- 그래프의 edge를 반복하여 7 개 입력 받아야 합니다.
  하나의 edge는 (vertex1 vertext2 cost)의  순서로 표시됩니다.
? Edge를 입력하시오: 0 1 10
? Edge를 입력하시오: 1 2 50
? Edge를 입력하시오: 3 2 20
? Edge를 입력하시오: 2 4 10
? Edge를 입력하시오: 3 4 60
? Edge를 입력하시오: 0 4 100
? Edge를 입력하시오: 0 3 30

! 입력된 그래프는 다음과 같습니다:
생성된 그래프 :
[0] -> 1(10)  3(30)  4(100)
[1] -> 2(50)
[2] -> 4(10)
[3] -> 2(20)  4(60)
[4] ->
```

```
최단경로의 Distance는 같습니다.
distance [0] = 0
distance [1] = 10
distance [2] = 50
distance [3] = 30
distance [4] = 60

최단경로는 다음과 같습니다.
0  - 1
0  - 3 - 2
0  - 3
0  - 3 - 2 - 4

모든 최단경로는 다음과 같습니다.
[0] 0 10 50 30 60
[1] -1 0 50 -1 60
[2] -1 -1 0 -1 10
[3] -1 -1 20 0 30
[4] -1 -1 -1 -1 0
```

FloydShortestPaths의 공개 함수는?

- 사용자에게 필요한 함수
  - public FloydShortestPaths(AdjacencyMatrixGraph givenGraph)
  - public void perform()
  - public ShortestPathsIterator ShortestPathsIterator(int givenVertex)
  - public class ShortestPathsIterator

Class “Application“

Application – 비공개 인스턴스 변수

```
import java.util.Scanner;

public class Application {
    private Scanner _scanner;
    private AdjacencyMatrixGraph _graph;
    private DijkstraShortestPaths _dijkstrashortestPaths;
    private FloydShortestPaths _floydShortestPaths;
    ……
```

Application의 공개 함수 run()의 구현

```
public void run(){
    if ( this.inputAndMakeGraph() ) {
        this.showGraph() ;

        this._dijkstrashortestPaths = new DijkstraShortestPaths(this._graph);
        this._dijkstrashortestPaths.perform(0);
        this.showDijkstraShortestDistance();
        this.showDijkstraSortestPath();
        this._floydShortestPaths = new FloydShortestPaths(this._graph);
        this._floydShortestPaths.perform();
        this.showFloydShortestPath();
    }
    else {
        System.out.println("₩n₩n[오류]그래프 삽입 실패") ;
    }
}
```

Application의 Private Method

- Application의 Private Member function의 사용법
  - public void showFloydShortestPath()
    - Floyd로 찾아낸 ShortestPath를 출력

Member Functions의 구현

- Application의 Private Member function의 구현
  - public void showFloydShortestPath()
    - 2차원 배열이므로 AdjacencyMatrixGraph의 출력인 showGraph()를 참고하여 작성

Class “FloydShortestPaths”

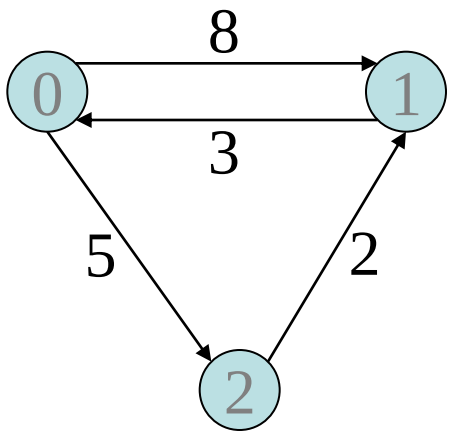
Floyd's Algorithm

```
private void  allcosts ( int[][] cost, int[][] distance,  int n)
{
        int  i,  j,  k ;
        for ( i=0 ; i<n ; i++ )
            for ( j=0 ; j<n ; j++ )
                distance[i][j] = cost[i][j] ;
        for ( k=0 ; k<n ; k++ )
            for ( i=0 ; i<n ; i++ )
                for ( j=0 ; j<n ; j++ )
                    if (distance[i][j] > distance[i][k] + distance[k][j])
                        distance[i][j] = distance[i][k] + distance[k][j];
}
```

Time Complexity : $O(n^3)$

Example of Floyd's Algorithm

$$\text{cost}[] = \begin{pmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{pmatrix}$$



A^{-1}	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

Example of Floyd's Algorithm (Cont'd)

A^0	*0*	*1*	*2*
0	0	8	5
1	3	0	8
2	∞	2	0

$A^0[1][2] = \min\{\infty, 3+5\} = 8$
$A^0[2][1] = \min\{2, \infty+8\} = 2$

A^1	*0*	*1*	*2*
0	0	8	5
1	3	0	8
2	5	2	0

$A^1[0][2] = \min\{5, 8+8\} = 5$
$A^1[2][0] = \min\{\infty, 2+3\} = 5$

A^2	*0*	*1*	*2*
0	0	7	5
1	3	0	8
2	5	2	0

$A^2[0][1] = \min\{8, 5+2\} = 7$
$A^2[1][0] = \min\{3, 8+5\} = 3$

Recovering the Paths

- path[i][j] means that the shortest path from *i* to *j* goes through path[i][j].
- Initially,

```
path[i][j] = -1;
```

- In the innermost loop,

```
if (distance[i][j] > distance[i][k] + distance[k][j]) {
    distance[i][j] = distance[i][k] + distance[k][j] ;
    path[i][j] = k;
}
```

- In order to print out the shortest path from *i* to *j* :

```
private void showPath (int i, int j)
{
    int  k ;
    k = path[i][j] ;
    if (k >= 0) {
        showPath (i, k);
        System.out.print (k);
        showPath (k, j);
    }
}
```

Example: Recovering the paths

path^{-1}	0	1	2
0	-1	-1	-1
1	-1	-1	-1
2	-1	-1	-1

path^{0}	0	1	2
0	-1	-1	-1
1	-1	-1	**0**
2	-1	-1	-1

path^{1}	0	1	2
0	-1	-1	-1
1	-1	-1	0
2	**1**	-1	-1

path^{2}	0	1	2
0	-1	**2**	-1
1	-1	-1	**0**
2	**1**	-1	-1

$A^0[1][2] = \min\{\infty, 3+5\} = 8$
path $^0[1][2] = \mathbf{0}$

$A^0[2][1] = \min\{2, \infty+8\} = 2$
path $^0[2][1]$: No change

$A^1[0][2] = \min\{5, 8+8\} = 5$
path $^1[0][2]$: No change

$A^1[2][0] = \min\{\infty, 2+3\} = 5$
path $^1[2][0] = \mathbf{1}$

$A^2[0][1] = \min\{8, 5+2\} = 7$
path $^2[0][1] = \mathbf{2}$

$A^2[1][0] = \min\{3, 8+5\} = 3$
path $^2[1][0]$: No change

path[0][1] = 2
path[0][2] = –1
path[2][1] = –1
} 0 → 2 → 1

path[1][2] = 0
path[1][0] = –1
path[0][2] = –1
} 1 → 0 → 2

path[2][0] = 1
path[2][1] = –1
path[1][0] = –1
} 2 → 1 → 0

비공개 인스턴스 변수

```
public class FloydShortestPaths {
    private int _numOfVertices ;
    private int [][] _distance;
    private AdjacencyMatrixGraph _graph;
```

FloydShortestPaths 의 멤버 함수는?

- FloydShortestPaths 의 Public Member function의 사용법
  - public FloydShortestPaths(AdjacencyMatrixGraph givenGraph)
    - FloydShortestPaths 생성자
  - public boolean perform()
    - FloydShortestPaths 를 찾는다.
  - public ShortestPathsIterator ShortestPathsIterator(int givenVertex)
  - public class ShortestPathsIterator

FloydShortestPaths 의 멤버 함수는?

- FloydShortestPaths 의 Public Member function의 구현
  - public FloydShortestPaths(AdjacencyMatrixGraph givenGraph)
    - givenGraph를 this._graph에 저장
    - this._graph의 numOfVertices을 this._numOfVertices에 저장
    - this._distance를 this._numOfVertices 2차원 배열만큼 생성
  - public boolean perform()
    - 앞에 설명한 Floyd알고리즘하여 프로그램을 완성한다.

FloydShortestPaths 의 멤버 함수는?

- FloydShortestPaths 의 Public Member function의 구현
  - public ShortestPathsIterator ShortestPathsIterator(int givenVertex)
  - public class ShortestPathsIterator
    - AdjacencyMatrixGraph의 adjacentVertircesIterator와 class AdjacentVerticesIterator를 참고하여 구현

[문제 4] 요약

- ShortestPath를 Dijkstra 알고리즘과 Floyd 알고리즘을 이용하여 구현하여 본다.

- ShortestPath가 실제 어떻게 처리 되는 지에 대하여 확인한다.

- 확인 해본 내용은 보고서에 포함이 되어야 한다.

과제 제출

과제 제출

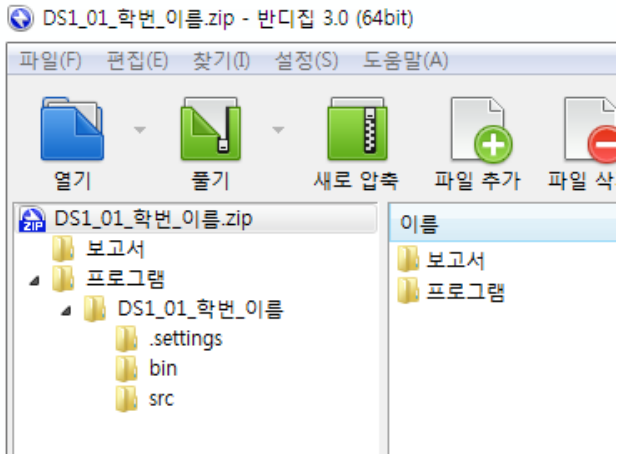
- pineai@cnu.ac.kr
  - 메일 제목 : [0X]DS2_04_학번_이름
    - 양식에 맞지 않는 메일 제목은 미제출로 간주됨
    - 앞의 0X는 분반명 (오전10시 : 00반 / 오후4시 : 01반)

- 제출 기한
  - 10월 1일(화) 23시59분까지
  - 시간 내 제출 엄수

  - 제출을 하지 않을 경우 0점 처리하고, 숙제를 50% 이상 제출하지 않으면 F 학점 처리하며, 2번 이상 제출하지 않으면 A 학점을 받을 수 없다.

과제 제출

- 파일 이름 작명 방법
  - DS2_04_학번_이름.zip
  - 폴더의 구성
    - DS2_04_학번_이름
      - 프로그램
        - 프로젝트 폴더 / 소스
        - 메인 클래스 이름 : DS2_04_학번_이름.java
      - 보고서
        - 이곳에 보고서 문서 파일을 저장한다.
        - 입력과 실행 결과는 화면 image로 문서에 포함시킨다.
        - 문서는 pdf 파일로 만들어 제출한다.



보고서 작성 방법

- 겉장
  - 제목: 자료구조 실습 보고서
  - [제xx주] 숙제명
  - 제출일
  - 학번/이름
- 내용
  1. 프로그램 설명서
     1. 주요 알고리즘 /자료구조 /기타
     2. 함수 설명서
     3. 종합 설명서 : 프로그램 사용방법 등을 기술
  2. 구현 후 느낀 점 : 요약의 내용을 포함하여 작성한다.
  3. 실행 결과 분석
     1. 입력과 출력 (화면 capture : 실습예시와 다른 예제로 할 것)
     2. 결과 분석

     ----- 표지 제외한 3번까지의 내용을 A4 세 장 내외의 분량으로 작성 할 것 -----
  4. 소스코드 : 화면 capture가 아닌 소스를 붙여넣을 것

     소스는 장수 제한이 없음.

[제 4 주 실습] 끝