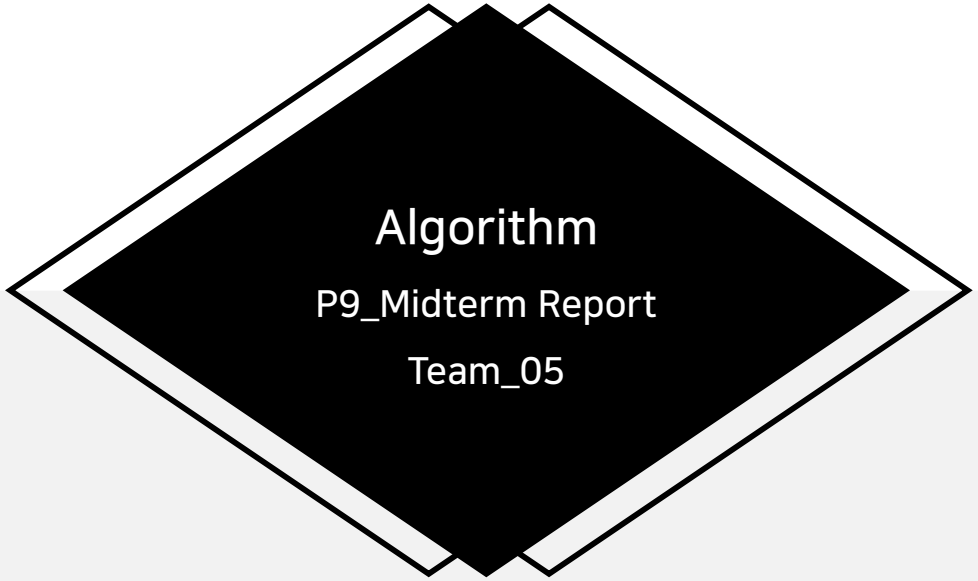
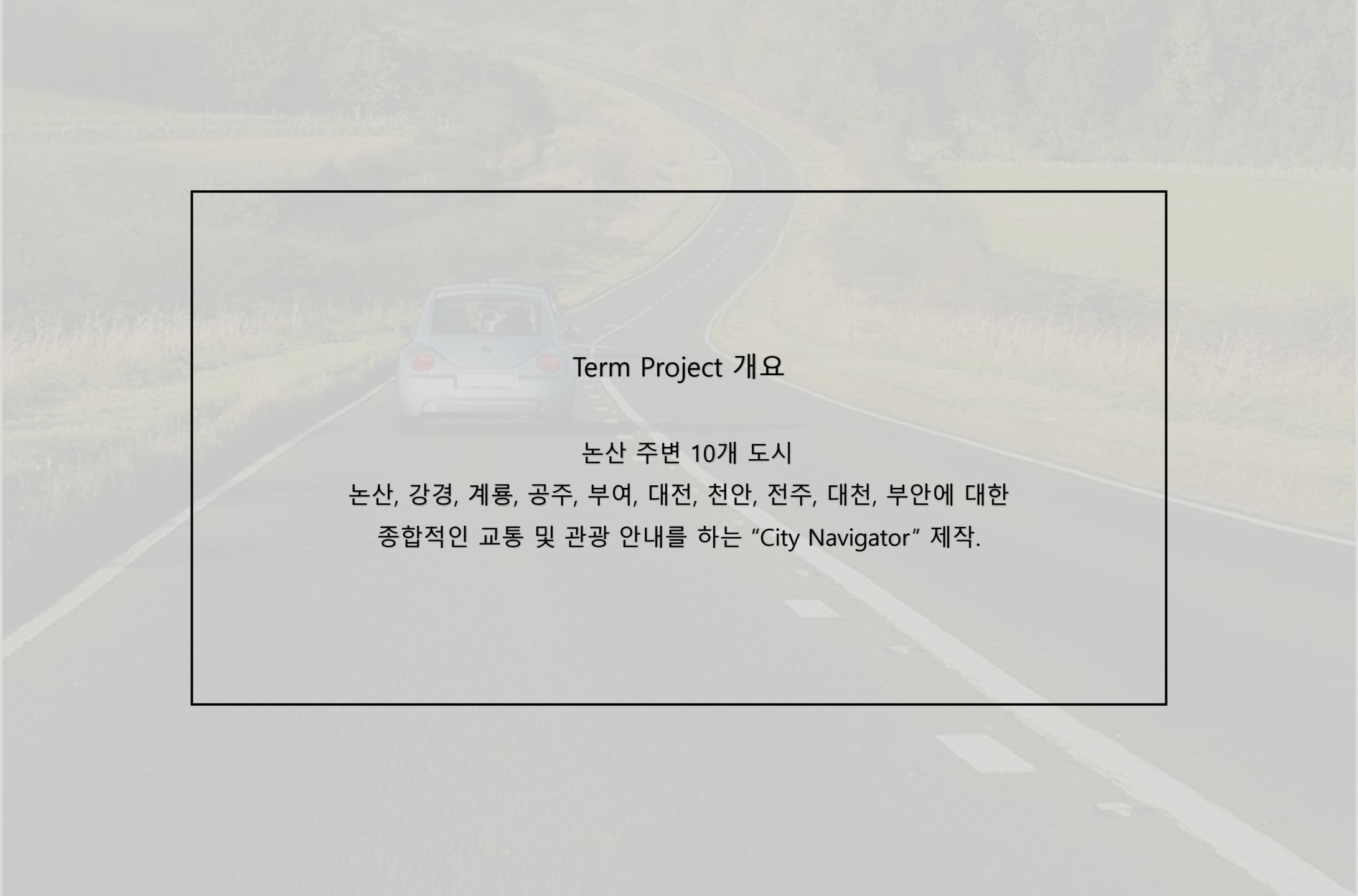

Algorithm
P9_Midterm Report
Team_05

INDEX



Term Project 개요

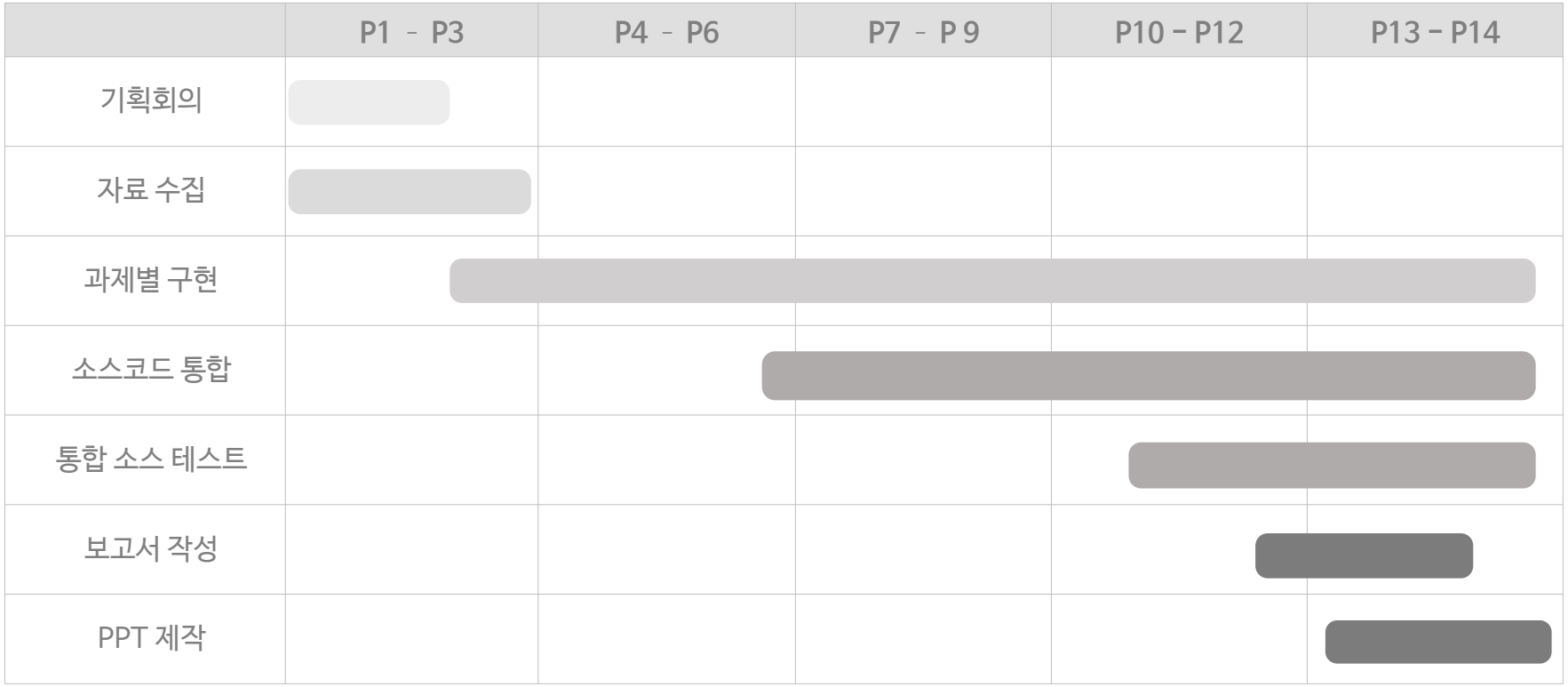
논산 주변 10개 도시
논산, 강경, 계룡, 공주, 부여, 대전, 천안, 전주, 대천, 부안에 대한
종합적인 교통 및 관광 안내를 하는 “City Navigator” 제작.

개발환경

Platform : Eclipse IDE + JVM + JAVA Language

OS : Windows 10 Pro

PROJECT TIMELINE 2019.04.29 – 2019.06.16



PROJECT SCHEDULE 2019.05

SUN	MON	TUE	WED	THR	FRI	SAT
	29	30	1	2	3	4
				Meeting		
5	6	7	8	9	10	11
	Collecting Data					
12	13	14	15	16	17	18
Term Project Assignment [P1 ~ P7]						
19	20	21	22	23	24	25
					PPT Design	
26	27	28	29	30	31	
	Term Project Assignment [P9 ~ P11]					

PROJECT SCHEDULE 2019.06

SUN	MON	TUE	WED	THR	FRI	SAT
						1
Term Project Assignment [P9 ~ P11]						
2	3	4	5	6	7	8
9	10	11	12	13	14	15
소스코드 통합			통합 소스 테스트, 수정		PPT Design	
16	17	18	19	20	21	22
최종 보고서 작성						
22	23	24	25	26	27	28
29	30					

City Navigator Base Data

논산 주변의 10개 도시 사이 지도상의 이동 거리

출발 도시	도착 도시	이동 거리
논 산	강 경	9km
	계 룡	22km
	공 주	34km
	부 여	21km
	대 전	67km
	천 안	87km
	전 주	51km
	대 천	79km
	부 안	80km

국도 및 지방도로 연결된 2차선 이상의 차량통행이 가능한 길을 이용하여 이동시,
출발 도시의 터미널에서 상대 도시의 터미널까지 측정한 거리를 Km 단위로 조사

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
논 산	# 역사 유적지	관촉사 (82 / 100) 계백장군 유적지 (80 / 100) 돈암서원 (78 / 100)
	# 자연 명승지	탑정호 (90 / 100) 대둔산 (84 / 100)
	# 레저 / 휴양	KT&G 상상마당 (86 / 100) 덕바위 농촌체험 휴양마을 (84 / 100) 논산 선샤인랜드 (80 / 100)
	# 맛집	신동회관 (84 / 100) 산애들애 (96 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
강 경	# 역사 유적지	옥녀봉 공원 (84 / 100) 죽림서원 (81 / 100) 강경 역사관 (78 / 100)
	# 자연 명승지	수락계곡 (86 / 100) 양촌 자연 휴양림 (85 / 100)
	# 레저 / 휴양	강경 황산 등대 전망대 (74 / 100) 천주교 강경성당 (90 / 100) 미내다리 (89 / 100)
	# 맛집	강경 젓갈 시장 (87 / 100) 강경 해물 칼국수 (88 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
계 룡	# 역사 유적지	무상사 (84 / 100) 사계고택 (71 / 100) 입암리 유적공원 (58 / 100)
	# 자연 명승지	계룡산 (87 / 100) 계룡 저수지 (85 / 100)
	# 레저 / 휴양	계룡대 통일탑 (81 / 100) 괴목정 무궁화 학습원 (84 / 100) 두계천 생태공원 (81 / 100)
	# 맛집	계룡면옥 (90 / 100) 콩밭가인 (88 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
공 주	# 역사 유적지	무령왕릉 (84 / 100) 공산성 (82 / 100) 석장리 박물관 (80 / 100)
	# 자연 명승지	황새바위 (78 / 100) 정안천 생태공원 (76 / 100)
	# 레저 / 휴양	공주 한옥마을 (97 / 100) 국립 공주 박물관 (84 / 100) 임립 미술관 (67 / 100)
	# 맛집	동해원 (75 / 100) 유가네 칼국수 (84 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
부 여	# 역사 유적지	정림사지 (84 / 100) 궁남지 (84 / 100) 부소산성 (81 / 100)
	# 자연 명승지	낙화암 (87 / 100) 성흥산 사랑나무 (88 / 100)
	# 레저 / 휴양	부여 국립 박물관 (78 / 100) 백제 문화 단지 (96 / 100) 서동요 테마파크 (99 / 100)
	# 맛집	서동한우 (69 / 100) 연꽃 이야기 (75 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
대 전	# 역사 유적지	둔산 선사 유적지 (78 / 100) 대덕 계족 산성 (67 / 100) 대전 시립 박물관 (87 / 100)
	# 자연 명승지	장태산 자연 휴양림 (88 / 100) 보문산 (79 / 100)
	# 레저 / 휴양	대전 시민 천문대 (81 / 100) 오월드 (94 / 100) 한밭 수목원 (84 / 100)
	# 맛집	성심당 (94 / 100) 신도 칼국수 (74 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
천 안	# 역사 유적지	독립 기념관 (98 / 100) 태조산 각원사 (88 / 100) 유관순 열사 기념관 (97 / 100)
	# 자연 명승지	광덕산 (81 / 100) 천호지 (81 / 100)
	# 레저 / 휴양	리각 미술관 (67 / 100) 쥬쥬피아 (96 / 100) 홍대용 과학관 (87 / 100)
	# 맛집	병천시장 (87 / 100) 봉봉닭강정 (67 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
전 주	# 역사 유적지	경기전 (74 / 100) 전주 한옥마을 (97 / 100) 풍남문 (81 / 100)
	# 자연 명승지	전주 자연 생태관 (67 / 100) 전주천교 (68 / 100)
	# 레저 / 휴양	덕진공원 (75 / 100) 자만 벽화마을 (76 / 100) 어진 박물관 (78 / 100)
	# 맛집	진미집 (79 / 100) 고궁 (82 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
대 천	# 역사 유적지	충청 수영성 (71 / 100) 성주사지 (84 / 100) 최고운 유적 (64 / 100)
	# 자연 명승지	죽도 관광지 (88 / 100) 무창포 (72 / 100)
	# 레저 / 휴양	대천 해수욕장 (97 / 100) 석탄 박물관 (88 / 100) 개화 예술공원 (78 / 100)
	# 맛집	화장골 맛집거리 (86 / 100) 황해원 (88 / 100)

City Navigator Base Data

10개 도시의 관광 자원

도시	#키워드	장소 (평점)
부 안	# 역사 유적지	부안 서문안당산 (87 / 100) 반계선생 유적지 (84 / 100) 선웅사 대웅전 (74 / 100)
	# 자연 명승지	채석강 (74 / 100) 적벽강 (84 / 100)
	# 레저 / 휴양	곰소염전 (63 / 100) 원숭이 학교 (95 / 100) 부안 영상테마파크 (87 / 100)
	# 맛집	씨윈드 (97 / 100) 이화자 백합죽 (91 / 100)

알고리즘 구현 List

P01 - P09

Union-Find Tree

10개 도시 데이터를 그래프로 표현하기

논산을 기점으로 다른 도시들의 최단 경로

관광 우선순위 정렬하기

관광안내계획 세우기

모든 도시 최단 경로 TSP

대전 도심의 Sky Line

P10 – P15

관광 안내 우선 순위

관광 키워드 찾기

논산 주변도시 모든 쌍 최단경로

TSP 근사해 알고리즘의 구현

[Term Project: 과제 2] 집합의 UnionFind 연산 - Tree

배타적 집합의 연산 알고리즘을 트리 표현법을 이용하여 JAVA 언어로 구현하고,
강의록에 나오는 트리로 표현된 집합 (a)와 (b)의 원소 데이터로 확인하시오.
전체의 구성은 트리 표현의 TSetNode{ }, UnionFind{ } 클래스 파일과,
응용 메소드가 있는 TreeUnionFind{ } 클래스 파일로 구성한다.

Node 클래스의 구성은 다음 코드와 같다.

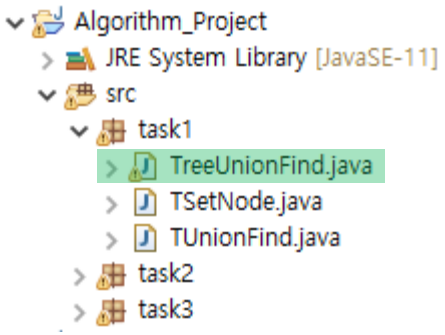
```
class TSetNode {
    int parent;  int rank;
            public TSetNode(int newParent, int newRank)
                    { parent = newParent;  rank   = newRank; }
}
```

UnionFind{ } 클래스는 세트 트리 노드를 저장할 객체배열과,
노드를 초기화 시키는 생성자,원소의 집합을 찾는 find(int i), 합집합 연산 union(int i, int j), 집합 출력
printSet(TSetNode[] a, String name[], int from, int to) 메소드를 갖는다.
응용 메소드 클래스 TreeUnionFind{ }는 강의록의 예제 집합에 대하여 출력에서 보이는
결과가 나오도록 내용을 구현하시오.

출력 예제

(1) 예제 그림의 c, b, a, h 원소로 된 집합 트리를 만든다.
[c, c] [c, b] [b, a] [c, h]
(2) 예제 그림의 e, d, f, g 원소로 된 집합 트리를 만든다.
[e, e] [e, d] [e, f] [e, g]
(3) a 원소 노드가 있는 집합을 찾아낸다.
a원소는 루트 원소가 c인 집합에 있습니다.
(4) f 원소가 있는 집합을 찾아낸다.
f원소는 루트 원소가 e인 집합에 있습니다.
(5) 두 집합의 합집합을 만들어 원소들을 출력한다.
c원소 집합에 통합되었습니다.
[c, c] [c, b] [b, a] [c, h] [c, e] [e, d] [e, f] [e, g]
(6) a 원소가 있는 집합을 찾아낸다.
a원소는 루트 원소가 c인 집합에 있습니다.

Source Code




```
public static void main(String[] argh) {
  Scanner s = new Scanner(System.in);
  String name[] = { "c", "b", "a", "h", "e", "d", "f", "g" };
  TSetNode[] A = new TSetNode[8];

  TSetNode c = new TSetNode("c", 0, 0); TSetNode b = new TSetNode("b", 0, 0);
  TSetNode a = new TSetNode("a", 1, 0); TSetNode h = new TSetNode("h", 0, 0);
  TSetNode e = new TSetNode("e", 4, 0); TSetNode d = new TSetNode("d", 4, 0);
  TSetNode f = new TSetNode("f", 4, 0); TSetNode g = new TSetNode("g", 4, 0);

  A[0] = c; A[1] = b; A[2] = a; A[3] = h;
  A[4] = e; A[5] = d; A[6] = f; A[7] = g;

  TUnionFind TF = new TUnionFind(A);
  for (int i = 0; i < 4; i++) {
  TF.printSet(A[i], name[i]);
  }

  for (int i = 4; i < 8; i++) {
  TF.printSet(A[i], name[i]);
  }

  int num, num2; // 3, 4번

  num = TF.find(2);
  System.out.println("a원소는 루트 원소가 " + TF.a[num] + "인 집합에 있습니다.\n");

  num2 = TF.find(6);
  System.out.println("f원소는 루트 원소가 " + TF.a[num2] + "인 집합에 있습니다.\n");

  TF.union(0, 4);//5번
  for(int i = 0; i < A.length; i++) {
  TF.printSet(A[i], name[i]);
  }

  int num3 = TF.find(2);
  System.out.println("a원소는 루트 원소가"+ TF.a[num3]+"인 집합에 있습니다.");
  }
}
```

c, b, a, h원소로 된 집합 트리를 만든다.

e, d, f, g 원소로 된 집합 트리를 만든다.

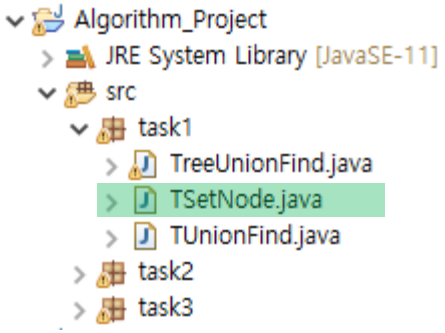
a원소 노드가 있는 집합을 찾아낸다.

f원소 노드가 있는 집합을 찾아낸다.

두 집합의 합집합을 만들어 원소들을 출력한다.

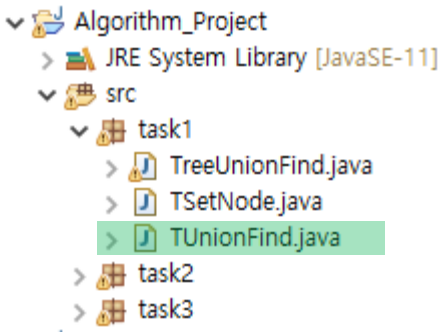
a원소가 있는 집합을 찾아낸다.

Source Code



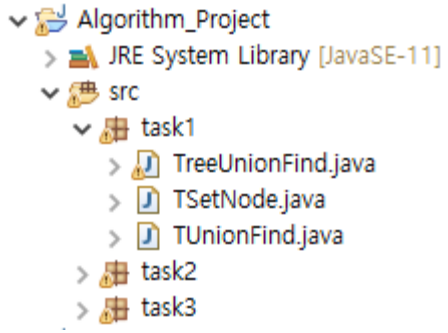

```java
package task1;

public class TSetNode {
  int parent;
  int rank;
  public String name;

  public TSetNode(String name, int newParent, int newRank) {
    parent = newParent;
    rank = newRank;
    this.name = name;
  }

  public String toString() {
    return name;
  }
}
```

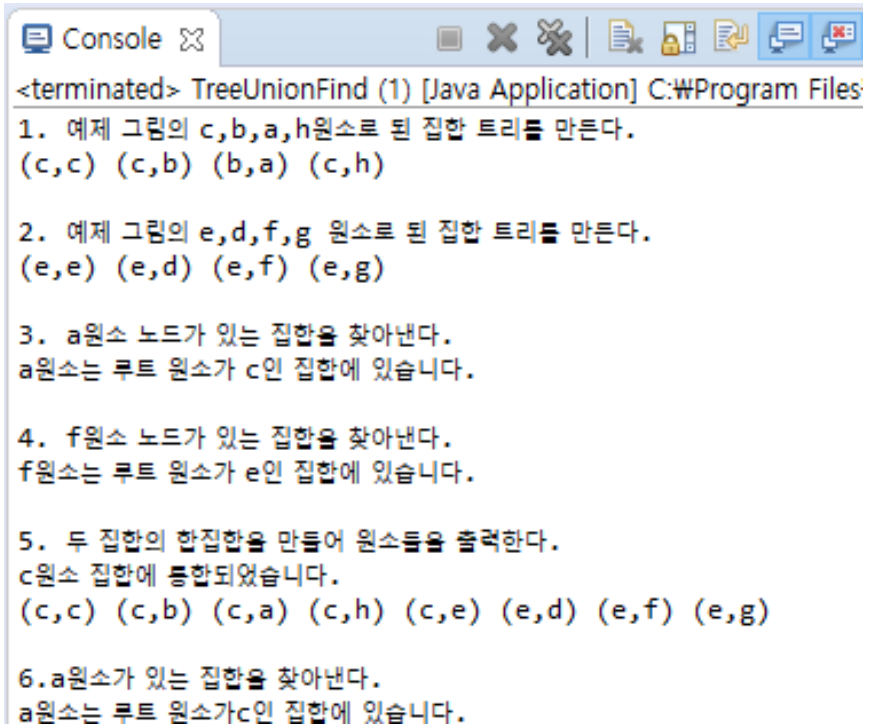
Source Code




```java
public class TUnionFind {
  public TSetNode[] a;
  public TUnionFind(TSetNode[] Tset) { // 생성자
    a = Tset;
    for (int i = 0; i < a.length; i++)
    a[i] = new TSetNode(Tset[i].name ,Tset[i].parent, 0);
  }
  protected int find(int i) { // 경로 압축
    if (i != a[i].parent)
    a[i].parent = find(a[i].parent); // 리턴하며 경로상의 각 노드의 부모가 루트가되게 만든
다.
    return a[i].parent;
  }

  public void union(int i, int j) { // Union 연산, i 부모, j 자식
    int iroot = find(i);
    int jroot = find(j);
    if (iroot == jroot)
      return; // rank가 높은 루트 노드가 승자가 된다.
    if (a[iroot].rank > a[jroot].rank) {
      a[jroot].parent = iroot; // iroot가 승자
      System.out.println(a[iroot] +"원소 집합에 통합되었습니다.");
    }else if (a[iroot].rank < a[jroot].rank) {
      if (a[iroot].rank == 0) {
      a[iroot].rank = a[jroot].rank + 1; // start of link
      a[j].parent = i;
      } else {
        a[iroot].parent = jroot; // jroot가 승자
        System.out.println(a[jroot] +"원소 집합에 통합되었습니다.");
        }
    }else {
      a[jroot].parent = iroot; // 둘중에 하나 임의로 승자
      a[iroot].rank++; // iroot의 rank 1 증가
      System.out.println(a[iroot] +"원소 집합에 통합되었습니다.");
    }
  }

  void printSet(TSetNode A, String name) {
    System.out.print("("+a[A.parent]+"," + A.name+ ") ");
  }
}
```

Result





출력 예제

(1) 예제 그림의 c, b, a, h 원소로 된 집합 트리를 만든다.

[c, c] [c, b] [b, a] [c, h]

(2) 예제 그림의 e, d, f, g 원소로 된 집합 트리를 만든다.

[e, e] [e, d] [e, f] [e, g]

(3) a 원소 노드가 있는 집합을 찾아낸다.

a원소는 루트 원소가 c인 집합에 있습니다.

(4) f 원소가 있는 집합을 찾아낸다.

f원소는 루트 원소가 e인 집합에 있습니다.

(5) 두 집합의 합집합을 만들어 원소들을 출력한다.

c원소 집합에 통합되었습니다.

[c, c] [c, b] [b, a] [c, h] [c, e] [e, d] [e, f] [e, g]

(6) a 원소가 있는 집합을 찾아낸다.

a원소는 루트 원소가 c인 집합에 있습니다.

[Term Project: 과제 3] 도시의 그래프 표현과 DFS , BFS

1. 과제 1에서 조사한 논산 주변의 10개 도시의 연결 도로망을 간선 중심의 인접리스트로 그래프를 표현하여, 인접리스트를 출력하시오.
   - 정점은 도시이며 정점에는 [도시명 , 관광 10 명소] 데이터가 저장
   - 간선은 도시간의 도로인데 Edge Node에는 [출발 도시 , 도착 도시 , 거리] 데이터를 저장

2. 도시 관광정보를 제공하기 위하여 논산을 출발 도시로 하는 도시 그래프에서 DFS 탐색으로 생성한 "도시 관광 순서도"와 경로를 모두 합한 "여행 거리"를 출력하시오.

3. 도시 관광정보를 제공하기 위하여 논산을 출발 도시로 하는 도시 그래프에서 BFS 탐색으로 생성한 "도시 관광 순서도"와 경로를 모두 합한 "여행 거리"를 출력하시오.

4. 2, 3 번 결과로 나온 도시 방문순서에는 각각 어떤 의미를 부여할 수 있는지를 비교하여 서술

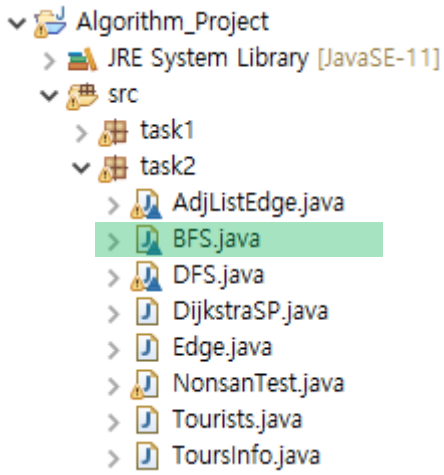
출력 예제

(1) 10개 도시의 연결도로망을 간선 중심의 인접리스트로 표현
도시 1 : 도시 2 → 도시 3 → … → 도시 10
…
…
도시 10 : 도시 1 → 도시 2 → … → 도시 9

(2) DFS 탐색으로 생성한 도시관광 순서도와 여행 거리
도시 1 → 도시 2 → … → 도시 10 / DFS 총 여행 거리 :

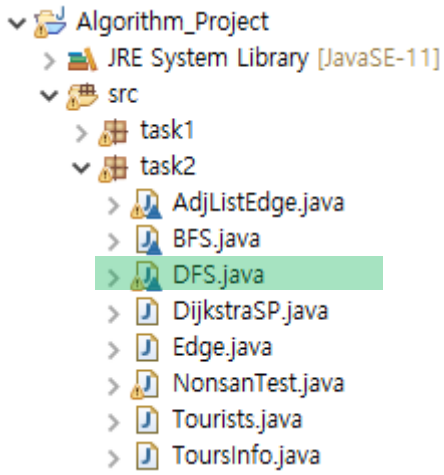
(3) BFS 탐색으로 생성한 도시관광 순서도와 여행 거리
도시 1 → 도시 2 → … → 도시 10 / BFS 총 여행 거리 :

Source Code



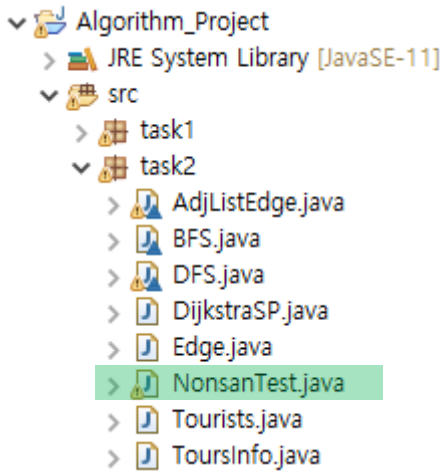

```java
class BFS {

  int N;  // 그래프 정점의 수
  List<Edge>[] graph;
  public Queue<Integer> adj;
  private boolean[ ] visited; // BFS 수행 중 방문한 정점의 원소를 true로 만든다.
  int sum  =0;

  public BFS(List<Edge>[] adjList) { // 생성자
    N = adjList.length;
    adj = new LinkedList<>();
    graph = adjList;
    visited = new boolean [N];

    for (int i = 0; i < N; i++) visited[i] = false;   // 배열 초기화
    for (int i = 0; i < N; i++) if (!visited[i]) bfs(i);
  }

  private void bfs(int i) {
    Queue<Integer> q = new LinkedList<Integer>();  // 큐 선언
    visited[i] = true;
    q.add(i); //큐에 시작 정점 s를 삽입
    while (!q.isEmpty()) {
      int j = q.remove();       // 큐에세 정점 j를 가져옴
      adj.add(j);
      System.out.print(j+" ");

      for (Edge e: graph[j]) {     // 정점 j에 인접한 정점 방문
        if (!visited[e.tv]) {
        visited[e.tv] = true;
        q.add(e.tv);  // 방문된 정점을 큐에 삽입
        sum += e.weight*2;
        }
      }
    }
  }
}
```

Source Code



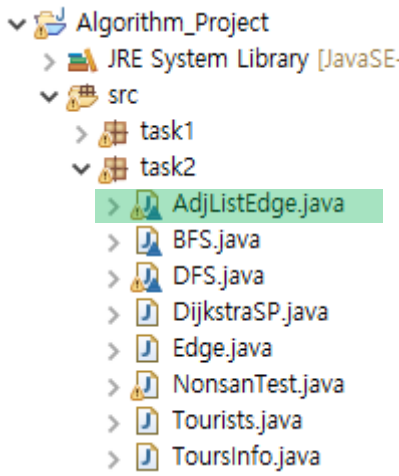

```java
class DFS {

  int N; // 그래프 정점의 수
  Queue<Integer> adj;
  List<Edge>[] graph;
  Stack<Edge> back;
  private boolean[ ] visited;// DFS 수행 중 방문한 정점을 true로 만든다.
  int sum = 0;

  public DFS(List<Edge>[] adjList) { // 생성자
    N = adjList.length;
    adj = new LinkedList<>();
    graph = adjList;
    back = new Stack();
    visited = new boolean [N];
    for (int i = 0; i < N; i++) visited[i] = false;  // 배열 초기화
    for (int i = 0; i < N; i++) if (!visited[i]) dfs(i);  // 복수 연결성분 대비
  }

  private void dfs(int i) {
    visited[i] = true;        // 점점 i가 방문되어 visited[i]를 true로 만든다.
    System.out.print(i+" ");  // 정점 i가 방문되었음을 출력한다.
    adj.add(i);

    for (Edge e: graph[i]) {
      if(!visited[e.tv]) {
      back.push(e);
      }
    }

    while(!back.isEmpty()) {
      Edge e = back.pop();   // 정점 i에 인접한 각 정점에 대해
      if (!visited[e.tv])  dfs(e.tv);
    }
  }
}
```

Source Code



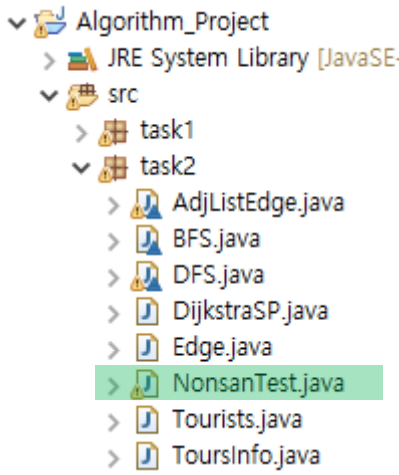

```java
public class NonsanTest {
  static String[] City = {"논산","강경","계룡","공주","부여","대전","천안","전주","대천","부안"};

  public static void main(String[] args) {
    int[][] dist = {
              {0, 9, 22, 34, 21, 67, 87, 51, 79, 80},
              {9, 0, 35, 0, 0, 74, 0, 0, 0, 0},
              {22, 35, 0, 0, 0, 22, 0, 0, 0, 0},
              {34, 0, 0, 0, 33, 0, 42, 0, 0, 0},
              {21, 0, 0, 33, 0, 0, 0, 0, 49, 0},
              {67, 74, 22, 0, 0, 0, 0, 0, 0, 0},
              {87, 0, 0, 42, 0, 0, 0, 0, 0, 0},
              {51, 0, 0, 0, 0, 0, 0, 0, 0, 41},
              {79, 0, 0, 0, 49, 0, 0, 0, 0, 0},
              {80, 0, 0, 0, 0, 0, 0, 41, 0, 0},
              };
    System.out.println("1.논산 주변의 10개 도시의 연결 도로망을 간선 중심의 인접리스트로 그래프를 표현");
    int N = dist.length;//1번
    List<Edge>[] adjList = new LinkedList[N];
    AdjListEdge test = new AdjListEdge();
    adjList = test.convertMtoL(dist, N);
    test.printAdjList(adjList);

    Tourists Nonsan = new Tourists("논산");
    Tourists Gang = new Tourists("강경");
    Tourists Gye = new Tourists("계룡");
    Tourists Gong = new Tourists("공주");
    Tourists Buyeo = new Tourists("부여");
    Tourists DaeJeon = new Tourists("대전");
    Tourists Cheon = new Tourists("천안");
    Tourists JeonJu = new Tourists("전주");
    Tourists DaeCheon = new Tourists("대천");
    Tourists Buan = new Tourists("부안");

    String k1 = "역사 유적지";
    String k2 = "자연 명승지";
    String k3 = "휴양지";
    String k4 = "맛집";
```

Source Code




```java
package task2;

import java.util.LinkedList;
import java.util.List;

class AdjListEdge{
  private int totalV;  // number of max vertex
  private List<Edge>[] adjList;  // 정점별 간선 리스트

  public List<Edge>[] convertMtoL(int dist[][] , int N) {  // 인접리스트로 바꾸기

    adjList = new List[N];

    for(int i=0; i<N; i++) {
      adjList[i] = new LinkedList<>();
      for(int j=0; j<N; j++) {
        if(dist[i][j] > 0) {
          Edge e = new Edge(i,j,dist[i][j]);
          adjList[i].add(e);

        }
      }
    }
    return adjList;
  }

  public void printAdjList(List<Edge> adjList[]) {
    NonsanTest Nt = new NonsanTest();

    for(int i = 0; i < adjList.length; i++) {
      System.out.print(""+Nt.City[i]+ "출발:");
        for(int j = 0; j < adjList[i].size(); j++) {
          System.out.print("==> "+ Nt.City[adjList[i].get(j).tv] + " " + adjList[i].get(j).weight +"km");
        } System.out.println();
    }
  }
}
```

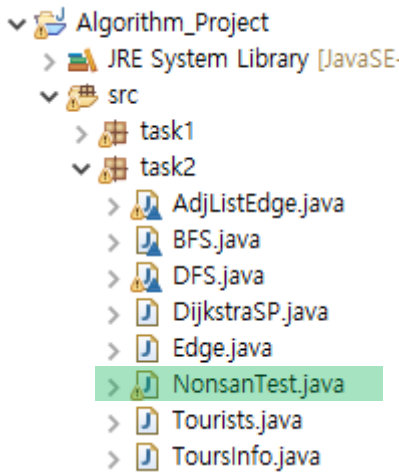
Source Code




```
// 0 논산
Nonsan.tourist[0] = new ToursInfo("관측사", k1, 82);
Nonsan.tourist[1] = new ToursInfo("계백장군", k1, 80);
Nonsan.tourist[2] = new ToursInfo("돈암서원", k1, 78);
Nonsan.tourist[3] = new ToursInfo("탑정호", k2, 90);
Nonsan.tourist[4] = new ToursInfo("대둔산", k2, 84);
Nonsan.tourist[5] = new ToursInfo("KT&G 상상마당", k3, 86);
Nonsan.tourist[6] = new ToursInfo("덕바위 농촌체험 휴양마을", k3, 84);
Nonsan.tourist[7] = new ToursInfo("논산 선샤인랜드", k3, 80);
Nonsan.tourist[8] = new ToursInfo("신동회관", k4, 84);
Nonsan.tourist[9] = new ToursInfo("산애들애", k4, 96);
```

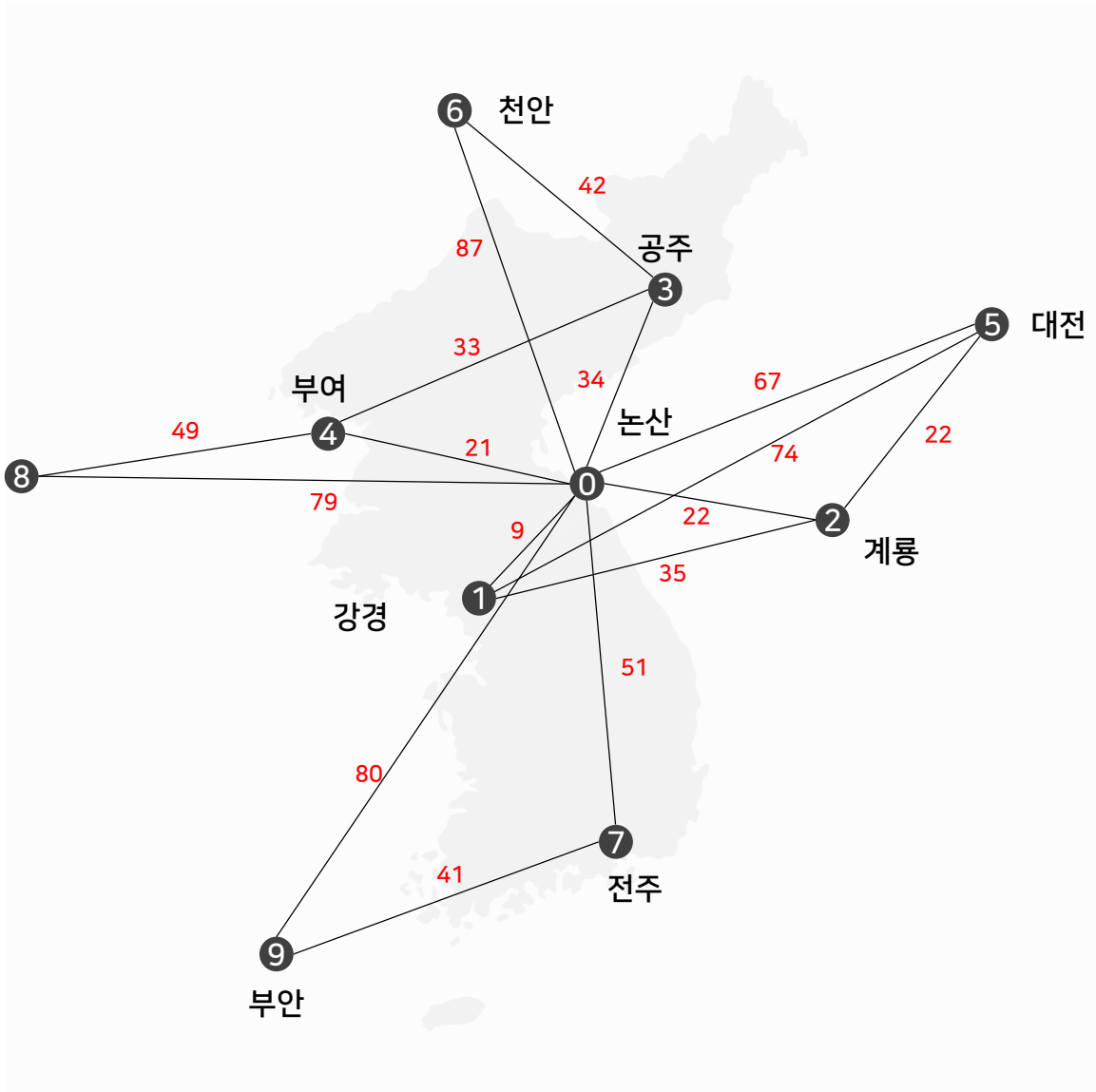
```
// 9 부안
Buan.tourist[0] = new ToursInfo("부안서문안당산", k1, 87);
Buan.tourist[1] = new ToursInfo("반계선생유적지", k1, 84);
Buan.tourist[2] = new ToursInfo("선웅사 대웅전", k1, 74);
Buan.tourist[3] = new ToursInfo("채석강", k2, 74);
Buan.tourist[4] = new ToursInfo("적벽강", k2, 84);
Buan.tourist[5] = new ToursInfo("곰소염전", k3, 63);
Buan.tourist[6] = new ToursInfo("원숭이 학교", k3, 95);
Buan.tourist[7] = new ToursInfo("부안 영상테마파크", k3, 87);
Buan.tourist[8] = new ToursInfo("씨윈드", k4, 97);
Buan.tourist[9] = new ToursInfo("이화자백합죽", k4, 91);
```

Source Code




```
    int[] d = new int[N];//dfs,bfs를 위한 배열
    int sum = 0;
    System.out.print("DFS 인덱스 순서=> ");//dfs,bfs 선언.
    DFS dfs = new DFS(adjList);//dfs
    System.out.print("\nBFS 인덱스 순서=> ");
    BFS bfs = new BFS(adjList);//bfs

    System.out.println("\n2.DFS 탐색으로 생성한 "도시 관광 순서도"와 경로를 모두 합한 "여행 거리"를 출력하시오.");
    for(int i= 0; i < N; i++) {
      d[i] = dfs.adj.remove();//dfs 도시 순회 순서
      System.out.print(City[d[i]] + "=> ");
    }

    DijkstraSP di = new DijkstraSP(adjList);//dfs 이동거리 구함 (다익스트라 사용)
    for(int i = 0; i < d.length -1; i++) {
      int[] distance = di.shortestPath(d[i]);
      sum += distance[d[i+1]];
    }
    System.out.println("DFS 총 여행거리는 "+ sum+ "입니다.\n");

    System.out.println("\n3.BFS 탐색으로 생성한 "도시 관광 순서도"와 경로를 모두 합한 "여행 거리"를 출력하시오.");
    sum = 0;

    for(int i= 0; i < N; i++) {
      d[i] = bfs.adj.remove(); //bfs 도시 순회 순서
      System.out.print(City[d[i]] + "=> ");
    }

    for(int i = 0; i < d.length -1; i++) {//bfs 이동거리 (다익스트라사용)
      int[] distance = di.shortestPath(d[i]);
      sum += distance[d[i+1]];
    }
    System.out.println("BFS 총 여행거리는 "+ sum+ "입니다.\n");

  }
}
```

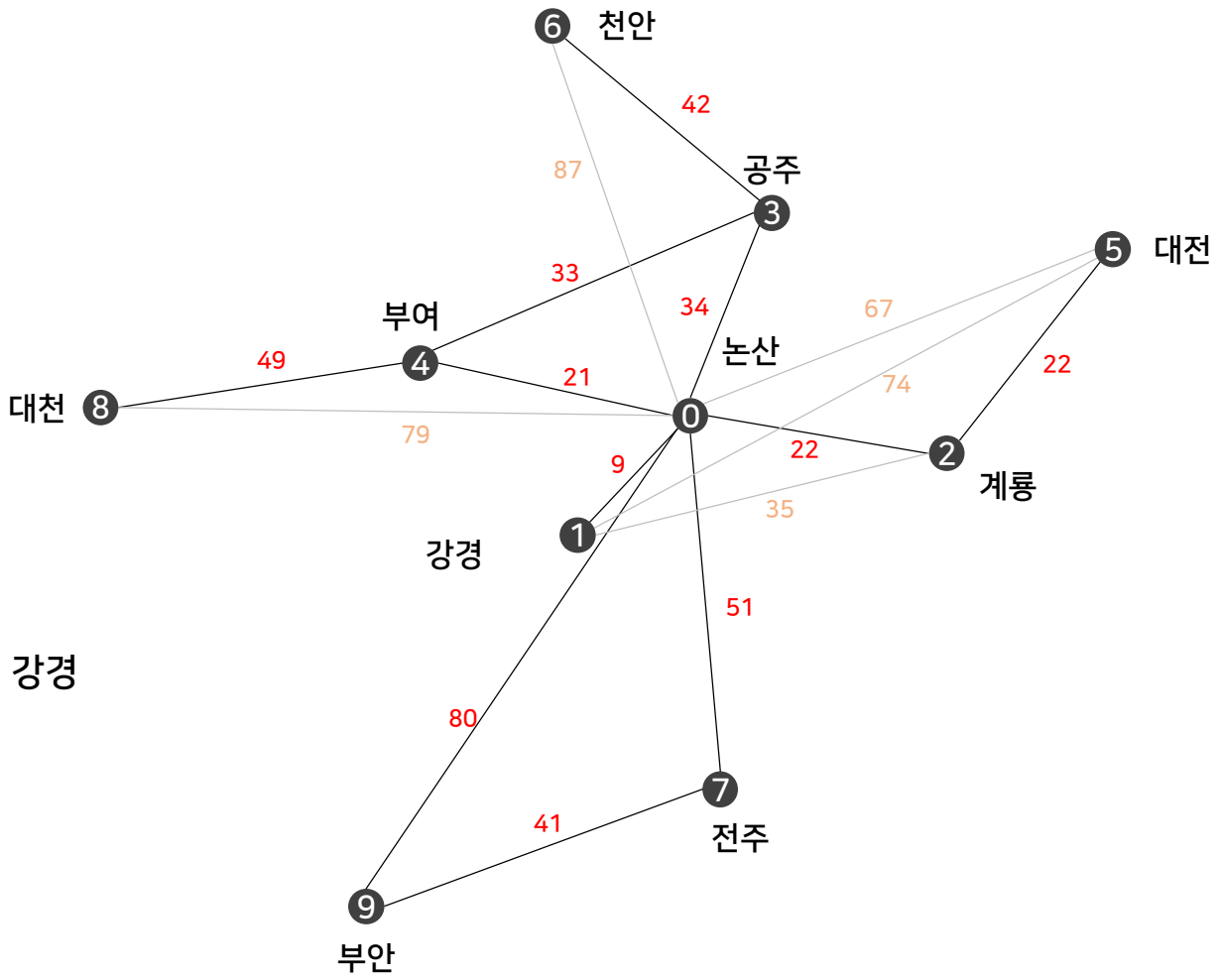
연결 그래프


6 천안
42
87
공주
3
5 대전
33
34
67
부여
4
논산
49
21
74
22
대천 8
0
79
22
9
2
계룡
35
1
강경
51
80
7
41
전주
9
부안

DFS Graph

DFS 인덱스 순서

0 → 9 → 7 → 8 → 4 → 3 → 6 → 5 → 2 → 1



DFS 탐색으로 생성한 도시 관광 순서도

논산 → 부안 → 전주 → 대천 → 부여 → 공주 → 천안 → 대전 → 계룡 → 강경

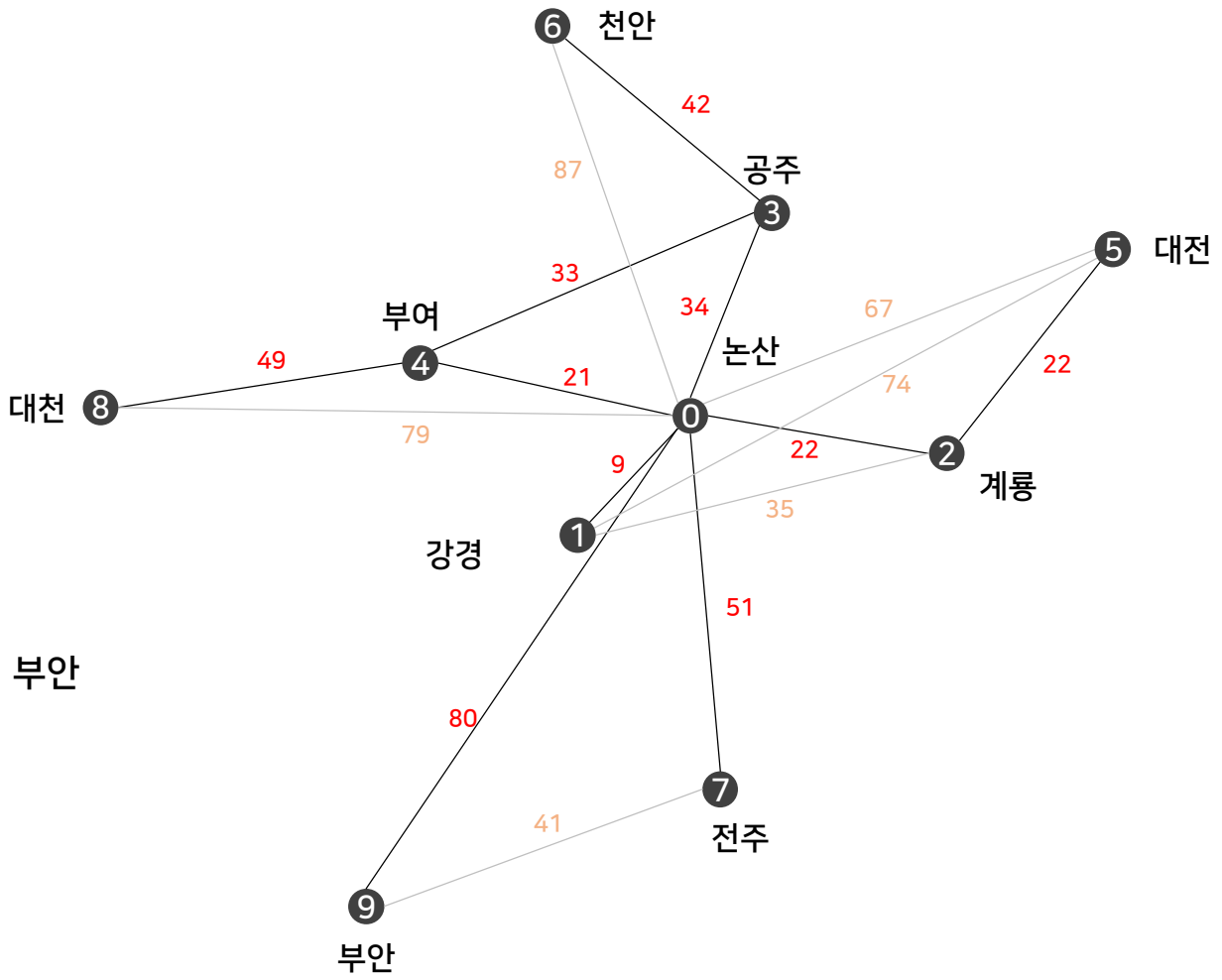
경로를 모두 합한 여행 거리

DFS 총 여행 거리 = 539km 입니다.

BFS Graph

BFS 인덱스 순서

0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9



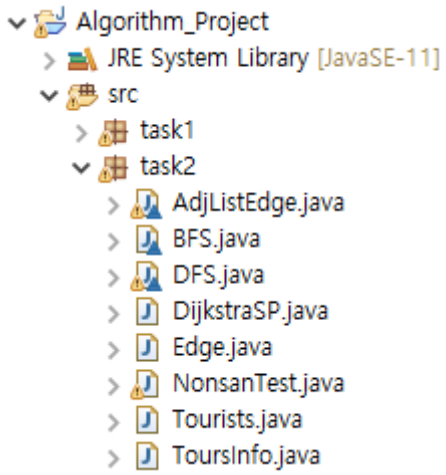
BFS 탐색으로 생성한 도시 관광 순서도

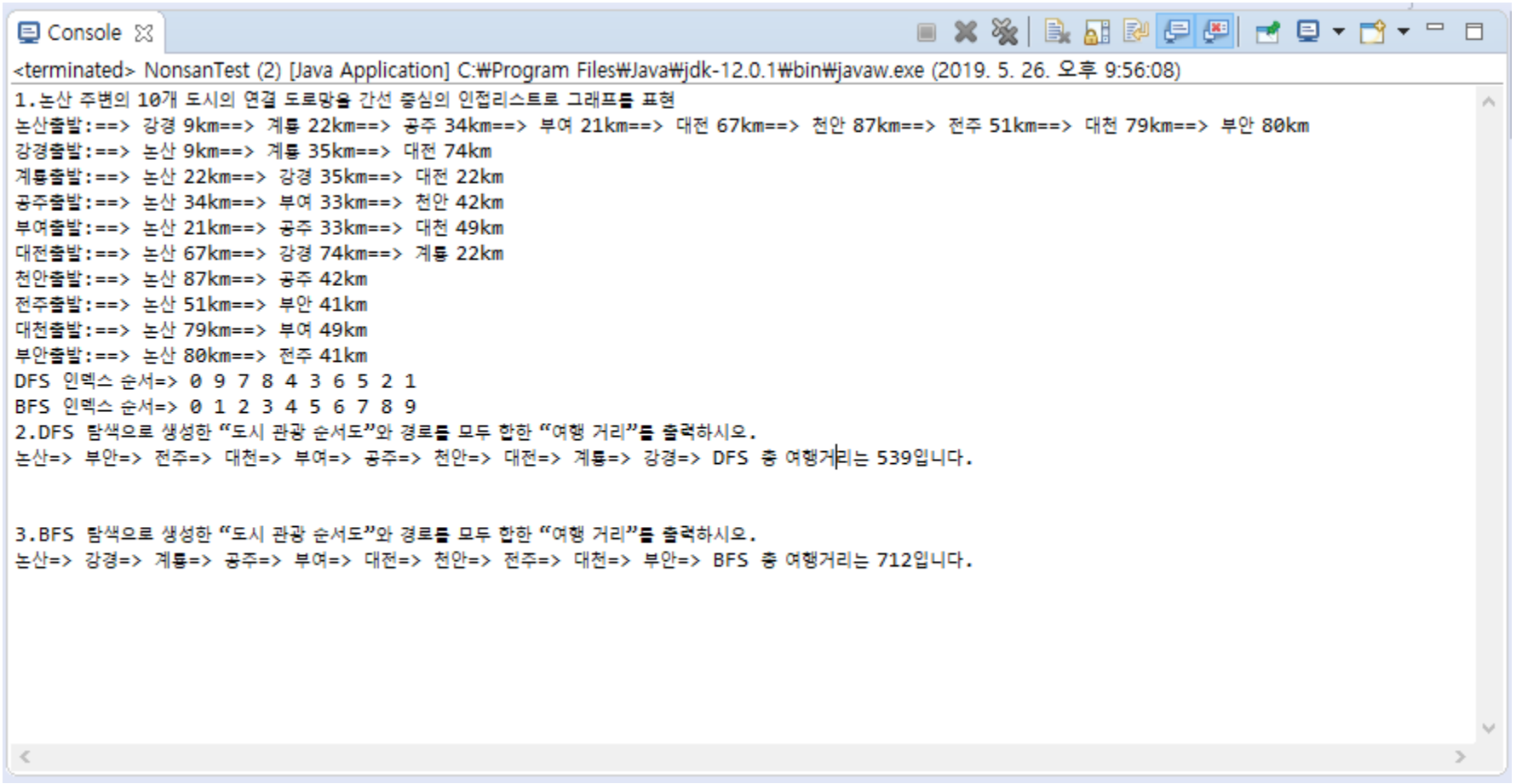
논산 → 강경 → 계룡 → 공주 → 부여 → 대전 → 천안 → 전주 → 대천 → 부안

경로를 모두 합한 여행 거리

BFS 총 여행 거리 = 712km 입니다.

Result


Algorithm_Project
JRE System Library [JavaSE-11]
src
task1
task2
AdjListEdge.java
BFS.java
DFS.java
DijkstraSP.java
Edge.java
NonsanTest.java
Tourists.java
ToursInfo.java


Console
<terminated> NonsanTest (2) [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2019. 5. 26. 오후 9:56:08)
1.논산 주변의 10개 도시의 연결 도로망을 간선 중심의 인접리스트로 그래프를 표현
논산출발:==> 강경 9km==> 계룡 22km==> 공주 34km==> 부여 21km==> 대전 67km==> 천안 87km==> 전주 51km==> 대천 79km==> 부안 80km
강경출발:==> 논산 9km==> 계룡 35km==> 대전 74km
계룡출발:==> 논산 22km==> 강경 35km==> 대전 22km
공주출발:==> 논산 34km==> 부여 33km==> 천안 42km
부여출발:==> 논산 21km==> 공주 33km==> 대천 49km
대전출발:==> 논산 67km==> 강경 74km==> 계룡 22km
천안출발:==> 논산 87km==> 공주 42km
전주출발:==> 논산 51km==> 부안 41km
대천출발:==> 논산 79km==> 부여 49km
부안출발:==> 논산 80km==> 전주 41km
DFS 인덱스 순서=> 0 9 7 8 4 3 6 5 2 1
BFS 인덱스 순서=> 0 1 2 3 4 5 6 7 8 9
2.DFS 탐색으로 생성한 "도시 관광 순서도"와 경로를 모두 합한 "여행 거리"를 출력하시오.
논산=> 부안=> 전주=> 대천=> 부여=> 공주=> 천안=> 대전=> 계룡=> 강경=> DFS 총 여행거리는 539입니다.
3.BFS 탐색으로 생성한 "도시 관광 순서도"와 경로를 모두 합한 "여행 거리"를 출력하시오.
논산=> 강경=> 계룡=> 공주=> 부여=> 대전=> 천안=> 전주=> 대천=> 부안=> BFS 총 여행거리는 712입니다.

DFS / BFS 비교

도시 방문순서에는 각각 어떤 의미를 부여할 수 있는지를 비교

DFS에서 나온 결과는 도시와 도시간의 연결점이 있으면 그 도시를 찾아가는 특징이 있기 때문에 도시들 사이에 연관성이 있지만 BFS같은 경우는 논산을 출발점으로 하기 때문에 논산과 연결되어 있는 모든 도시를 방문해야 하므로 비효율적인 방문 순서가 된다고 판단하였다.

[Term Project: 과제 4] 논산 주변 도시의 MST

1. 과제 3의 논산 주변의 10개 도시의 연결 도로망 그래프에 대한 MST를 구하여 출력.

2. Kruskal 2 알고리즘을 사용하여 논산을 출발 도시로 하는 도시 관광 그래프의 MST를 구하고, MST를 이용하여 TSP의 경로 순서도와 이 경로를 모두 합한 "여행 거리"를 출력하시오.

3. Prim 알고리즘을 사용하여 논산을 출발 도시로 하는 도시 관광 그래프의 MST를 구하고, MST를 이용하여 TSP의 경로 순서도와 이 경로를 모두 합한 "여행 거리"를 출력하시오.

4. 관광 경로를 제공하기 위하여 두 알고리즘을 사용할 경우의 장점과 단점을 비교하여 기술하시오.

5. 2, 3 번 MST의 결과를 이용하여 TSP 도시 방문 순서를 정하는데, MST는 어떤 역할을 할 수 있는지, 또 이 방법으로 하면 항상 최소 거리가 보장되는 지를 논하시오.

출력 예제

Kruskal MST =

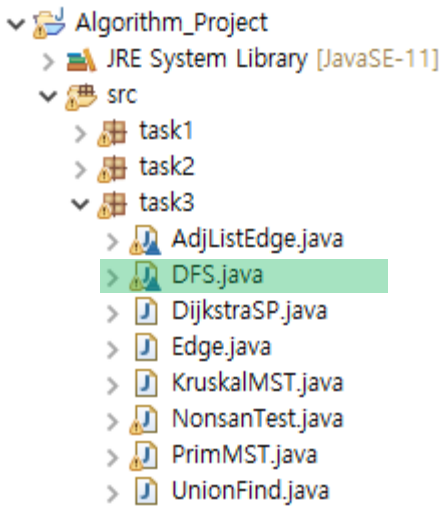
Prim MST =

MST 경로의 합 =

TSP 정점 순서도 =

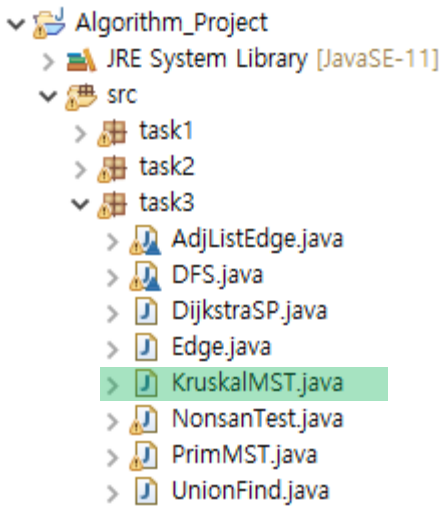
TSP 여행 거리 =

Source Code



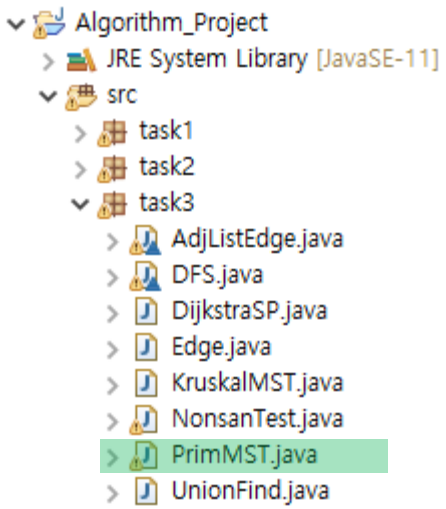

```java
package task3;

import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Stack;

class DFS {

  int N; // 그래프 정점의 수
  Queue<Integer> adj;
  List<Edge>[] graph;
  Stack<Edge> back;
  private boolean[ ] visited;// DFS 수행 중 방문한 정점을 true로 만든다.
  int g = 0;

  public DFS(List<Edge>[] adjList) { // 생성자
    N = adjList.length;
    adj = new LinkedList<>();
    graph = adjList;
    back = new Stack();
    visited = new boolean [N];
    for (int i = 0; i < N; i++) visited[i] = false;  // 배열 초기화
    for (int i = 0; i < N; i++) if (!visited[i]) dfs(i);  // 복수 연결성분 대비
  }

  private void dfs(int i) {
    visited[i] = true;        // 점점 i가 방문되어 visited[i]를 true로 만든다.
    System.out.print(i+" ");  // 정점 i가 방문되었음을 출력한다.
    adj.add(i);
    for (Edge e: graph[i]) if(!visited[e.tv]) back.push(e);
    while(!back.isEmpty()) {
      Edge e = back.pop();   // 정점 i에 인접한 각 정점에 대해
      if (!visited[e.tv]) dfs(e.tv);
    }
  }
}
```

Source Code



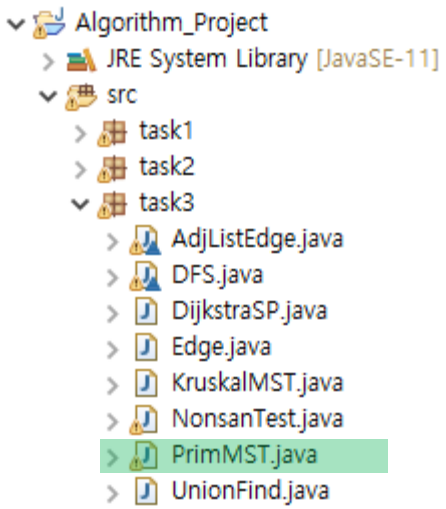

```java
public class KruskalMST {
  int N, M;//N 정점수 , M 간선수
  List<Edge>[] graph;
  UnionFind uf;
  Edge[] tree;

  static class Weight_Comparison implements Comparator<Edge> {
  //priorityQueue를 사용하기위해 어느 방식으로 비교해서 우선순위를 정할지 나타내는 메서드
    public int compare(Edge e, Edge f) {
      if (e.weight > f.weight)  return 1;
      else if (e.weight < f.weight)  return -1;
      else return 0;
  }
  public KruskalMST(List<Edge>[] adjList, int numOfEdges) {
    N = adjList.length;       //정점의 수
    M = numOfEdges;           //간선의 수
    graph = adjList;          //행렬방식에서 리스트로 바꾼 adjList를 graph에 담음
    uf = new UnionFind(N);    //Union-find 메서드를 사용하기위해 객체생성
    tree = new Edge[N - 1];   //간선의 갯수는 정점 개수 -1
  }
  public Edge[] mst() {
    Weight_Comparison BY_WEIGHT = new Weight_Comparison();  //우선순위를 정하는 객체 생성
    PriorityQueue<Edge> pq = new PriorityQueue<Edge>(M, BY_WEIGHT);  //우선순위 큐 객체 생성
    for (int i = 0; i < N; i++) {
      for (Edge e : graph[i]) {  //graph[i]에 있는 각각의 Edge형 자료들을 pq에 넣어 큐에 저장
      pq.add(e);
      }
    }
    int count = 0;
    while (!pq.isEmpty() && count < N - 1) {//pq가 empty가 아니거나 count가 간선개수를 넘지 않았다면
      Edge e = pq.poll();//최소 값을 e에 담음
      int u = e.fv;//from vertex
      int v = e.tv;//to vertex
      if (!uf.isConnected(u, v)) {//UnionFind 메서드 중 두 정점이 연결되있는지 확인하는 메서드
        uf.union(u, v);//연결이 안되있으면 연결해줌
        tree[count++] = e;//tree에 추가
      }
    }
    return tree; //Edge[] tree를 반환
  }
}
```

Source Code



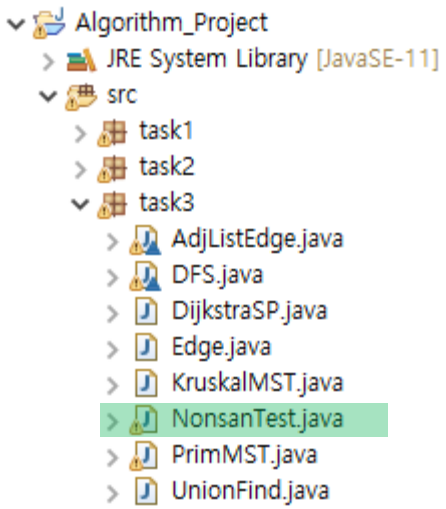

```java
public class PrimMST {
  int N;
  List<Edge>[] graph;
  List<Edge>[] prim;

  public PrimMST(List<Edge>[] adjList) {
    N = adjList.length;
    graph = adjList;
  }
  public int[] mst(int s) { // s= 시작정점
      boolean[] visited = new boolean[N];
      int[] D = new int[N];
      int[] previous = new int[N];
      for (int i = 0; i < N; i++) {
        visited[i] = false;
        previous[i] = -1;
        D[i] = Integer.MAX_VALUE;
      }
      previous[s] = 0; // 시작정점 s의 자료 초기화
      D[s] = 0;

      for (int k = 0; k < N; k++) {
        int minVertex = -1;
        int min = Integer.MAX_VALUE;
        for (int j = 0; j < N; j++) {
          if ((!visited[j]) && (D[j] < min)) {
            min = D[j];  minVertex = j;
          }
        } visited[minVertex] = true;
        for (Edge i : graph[minVertex]) {
          if (!visited[i.tv]) {
            int currentDist = D[i.tv];
            int newDist = i.weight;
            if (newDist < currentDist) {
              D[i.tv] = newDist;
              previous[i.tv] = minVertex;
            }
          }
        }
      }
      return previous;
  }
```

Source Code



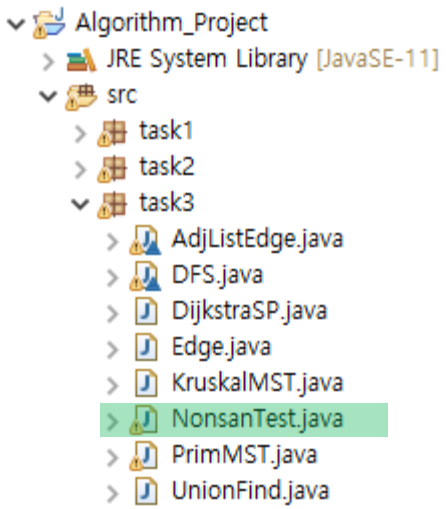

```
  public void printPrim(PrimMST a, int N) {
    int[] minTree = new int[graph.length - 1];
    int sum = 0;
    minTree = a.mst(0); // PrimMST형 변수에 출발지점을 넣고 그걸 minTree에 담음

    for (int i = 0; i < graph.length; i++) {
// previous에서 받은 tv를 adjList의 tv와 비교해 맞으면 fv값과 weight값을 가져옴.
      for (int j = 0; j < graph[i].size(); j++) {
        if (minTree[i] == graph[i].get(j).tv) {
        System.out.print("(" + graph[i].get(j).fv + "," + minTree[i] + "," + graph[i].get(j).weight + ") ");
        sum += graph[i].get(j).weight;
        }
      }
    }
    System.out.println("\n");
    System.out.println("최소신장트리의 간선 가중치 합은 " + sum + "입니다.");
  }
}
```

Source Code



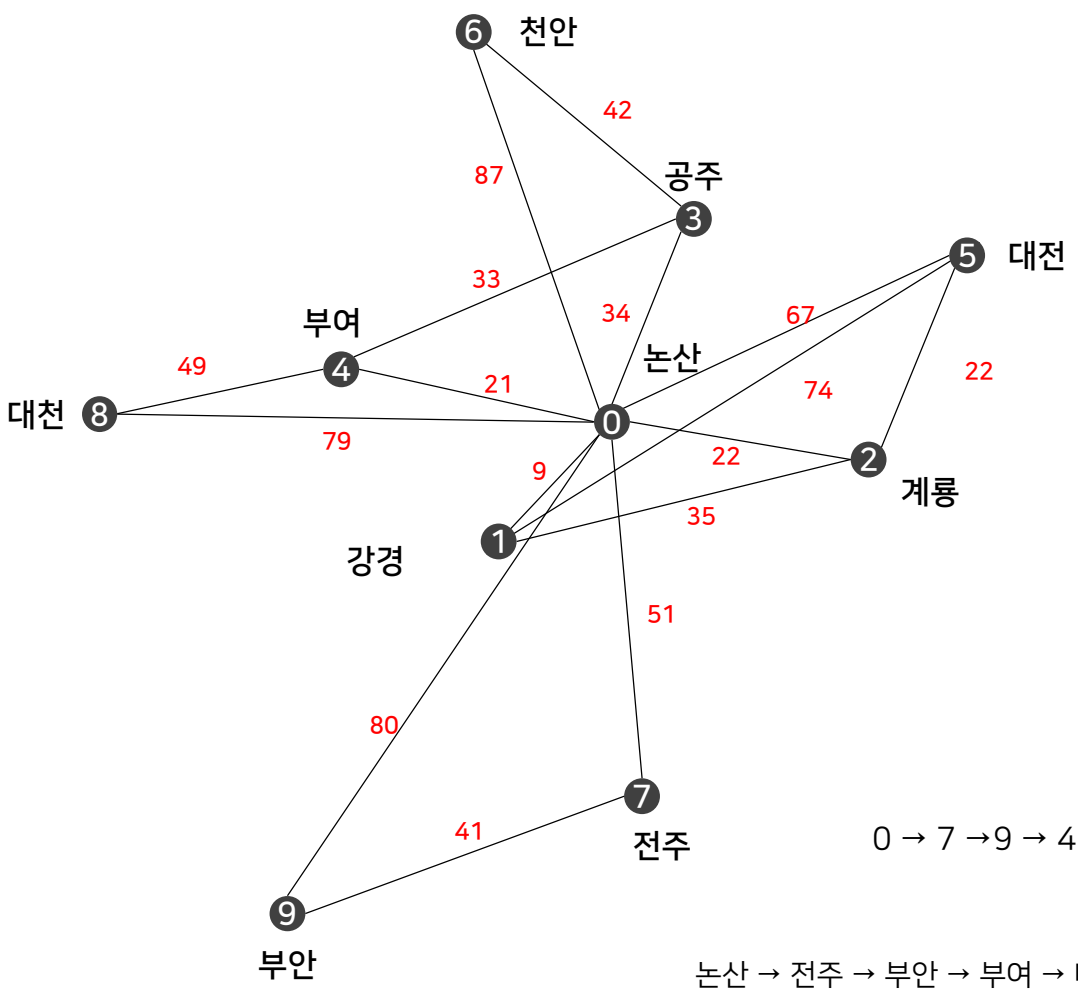

```java
public class NonsanTest {
  static String[] City = {"논산","강경","계룡","공주","부여","대전","천안","전주","대천","부안"};
  public static void main(String[] args) {
    int[][] dist = {
              {0, 9, 22, 34, 21, 67, 87, 51, 79, 80},
              {9, 0, 35, 0, 0, 74, 0, 0, 0, 0},
              {22, 35, 0, 0, 0, 22, 0, 0, 0, 0},
              {34, 0, 0, 0, 33, 0, 42, 0, 0, 0},
              {21, 0, 0, 33, 0, 0, 0, 0, 49, 0},
              {67, 74, 22, 0, 0, 0, 0, 0, 0, 0},
              {87, 0, 0, 42, 0, 0, 0, 0, 0, 0},
              {51, 0, 0, 0, 0, 0, 0, 0, 0, 41},
              {79, 0, 0, 0, 49, 0, 0, 0, 0, 0},
              {80, 0, 0, 0, 0, 0, 0, 41, 0, 0},
              };
  int N = dist.length;
  List<Edge>[] adjList = new LinkedList[N];
  AdjListEdge test = new AdjListEdge();
  adjList = test.convertMtoL(dist, N);
  //2가지 MST출력
  System.out.println("\n1. 논산 주변의 10개 도시의 연결 도로망 그래프에 대하여 MST를 구하여 출력하시오.");
  KruskalMST kmst = new KruskalMST(adjList, N);
  kmst.mst();
  System.out.print("Kruskal2 MST = ");
  for(int i = 0; i < N-1; i++) {
    System.out.print("("+kmst.tree[i].fv + "," + kmst.tree[i].tv+ ","+ kmst.tree[i].weight+") " ) ;
  }
  System.out.print("\nPrim MST = ");

  int[][] hung = new int[N][N];//prim MST 출력
  int[] pmin = new int[N];
  PrimMST pmst = new PrimMST(adjList);
  pmin = pmst.mst(0);
  pmst.printPrim(pmst, N-1);//최소신장트리 가중치

  System.out.println("Kruskal2 MST을 사용하여 논산을 출발 도시로 하는 도시 관광 그래프의 MST를 구하고 TSP를 구현하시오.”);


  System.out.println(">kruskal2 MST<");
  for(int i = 0; i < N-1; i++) {
    System.out.print("("+kmst.tree[i].fv + "," + kmst.tree[i].tv+ ","+ kmst.tree[i].weight+") “
  }
```

Source Code




```
    int[][] hang = new int[N][N];//MST를 사용한 결과를 행렬로 치환
    for(int i = 0; i < N-1; i++) {
      if(kmst.tree[i].weight != 0) {
        hang[kmst.tree[i].fv][kmst.tree[i].tv] = kmst.tree[i].weight;
        hang[kmst.tree[i].tv][kmst.tree[i].fv] = kmst.tree[i].weight;
      }
    }

    List<Edge>[] hangList = new LinkedList[N];     //MST행렬을 리스트로 바꿈
    AdjListEdge hL = new AdjListEdge();
    hangList = hL.convertMtoL(hang, N);

    System.out.print("\nTSP 정점 순서도: ");
    DFS hdfs = new DFS(hangList);    //리스트를 사용해 DFS를 통해 중복값 삭제
    hdfs.adj.add(0);   // 0 = 출발도시
    int[] d = new int[N+1];
    System.out.print(" => 0\n");
    System.out.println("순서:");
    for(int i= 0; i < N+1; i++) {
      d[i] = hdfs.adj.remove();    //dfs결과를 d배열에 넣어줌
      System.out.print("=>"+City[d[i]]);  // d배열의 인덱스로 도시들의 이름을 가져와서 출력
    }
    d[N] = 0;//도착도시을 안넣어줘도 0(논산)이지만 명시적으로 값을 넣어줌.

    DijkstraSP di = new DijkstraSP(adjList);//디작스트라에 넣는 리스트를 MST의 리스트가 아닌 처음 adjList로 입력
    int sum = 0;//이동거리
    for(int i = 0; i < d.length -1; i++) {
  //MST의 순환 경로를 따라가되 adjList의 최소거리까지 참고하여 adjList의 경로가 더 짧으면 adjList의 최소거리로 값을 출
  력
      int [] distance = di.shortestPath(d[i]);  //d배열에 저장되있는 도시순환 인덱스를 i와 i+1의 값을 순차적으로 출력
      sum += distance[d[i+1]];  //순환 하면서 이동거리의 총합.
    }
    System.out.println("\nKruskal2 MST을 이용한 TSP의 총 여행거리는 "+sum+"입니다.");
  }
}
```

adjList Graph



(0,1,9)

(0,4,21)

(0,2,22)

(2,5,22)

(3,4,33)

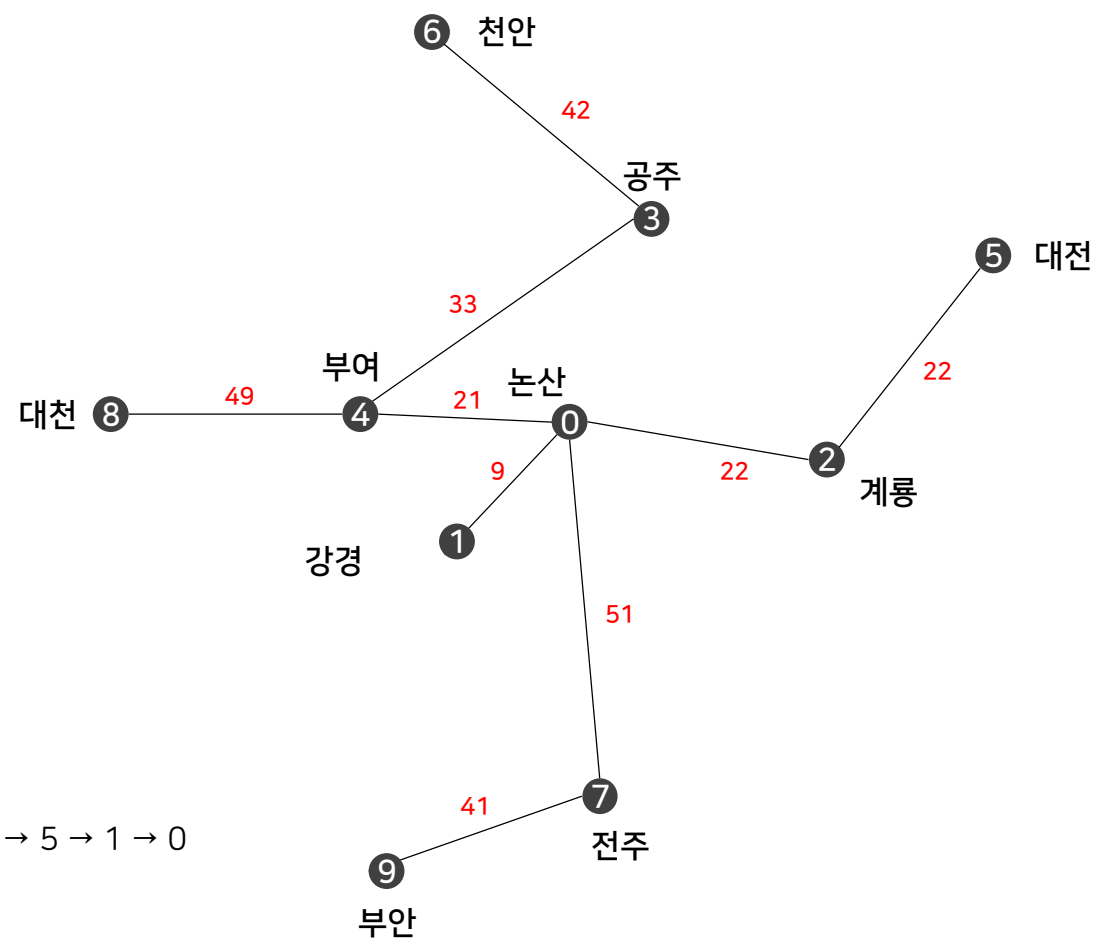
(9,7,41)

(3,6,42)

(4,8,49)

(0,7,51)

논산 주변의 10개 도시의 연결 도로망 그래프에 대한 MST



TSP 정점 순서도

0 → 7 →9 → 4 → 8 → 3 → 6 → 2 → 5 → 1 → 0

여행 순서:

논산 → 전주 → 부안 → 부여 → 대천 → 공주 → 천안 → 계룡 → 대전 → 강경 → 논산

논산 주변의 10개 도시의 연결 도로망 그래프에 대한 MST

(0,1,9)

(0,4,21)

(0,2,22)

(2,5,22)

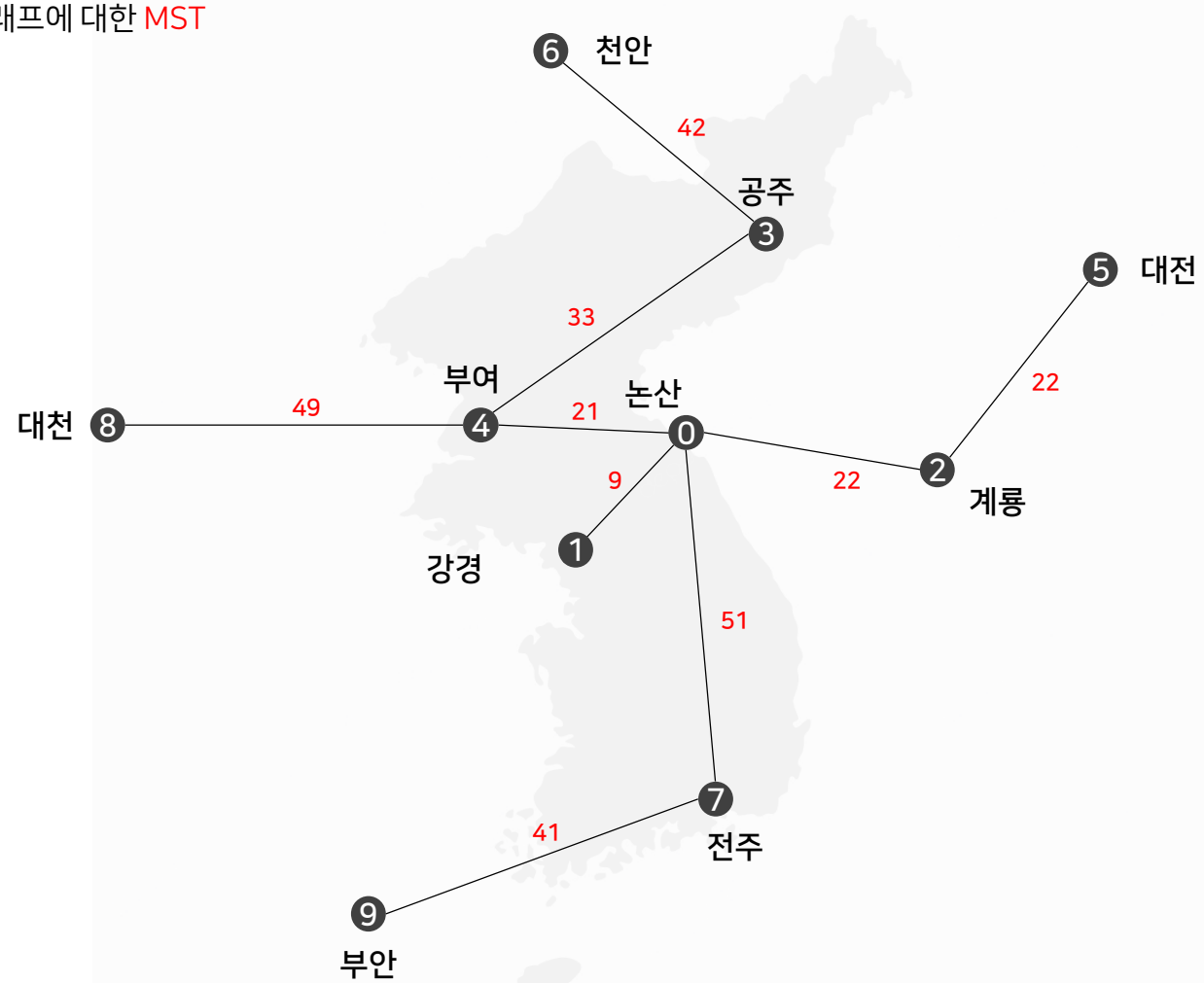
(3,4,33)

(9,7,41)

(3,6,42)

(4,8,49)

(0,7,51)



TSP 정점 순서도

0 → 7 →9 → 4 → 8 → 3 → 6 → 2 → 5 → 1 → 0

여행 순서:

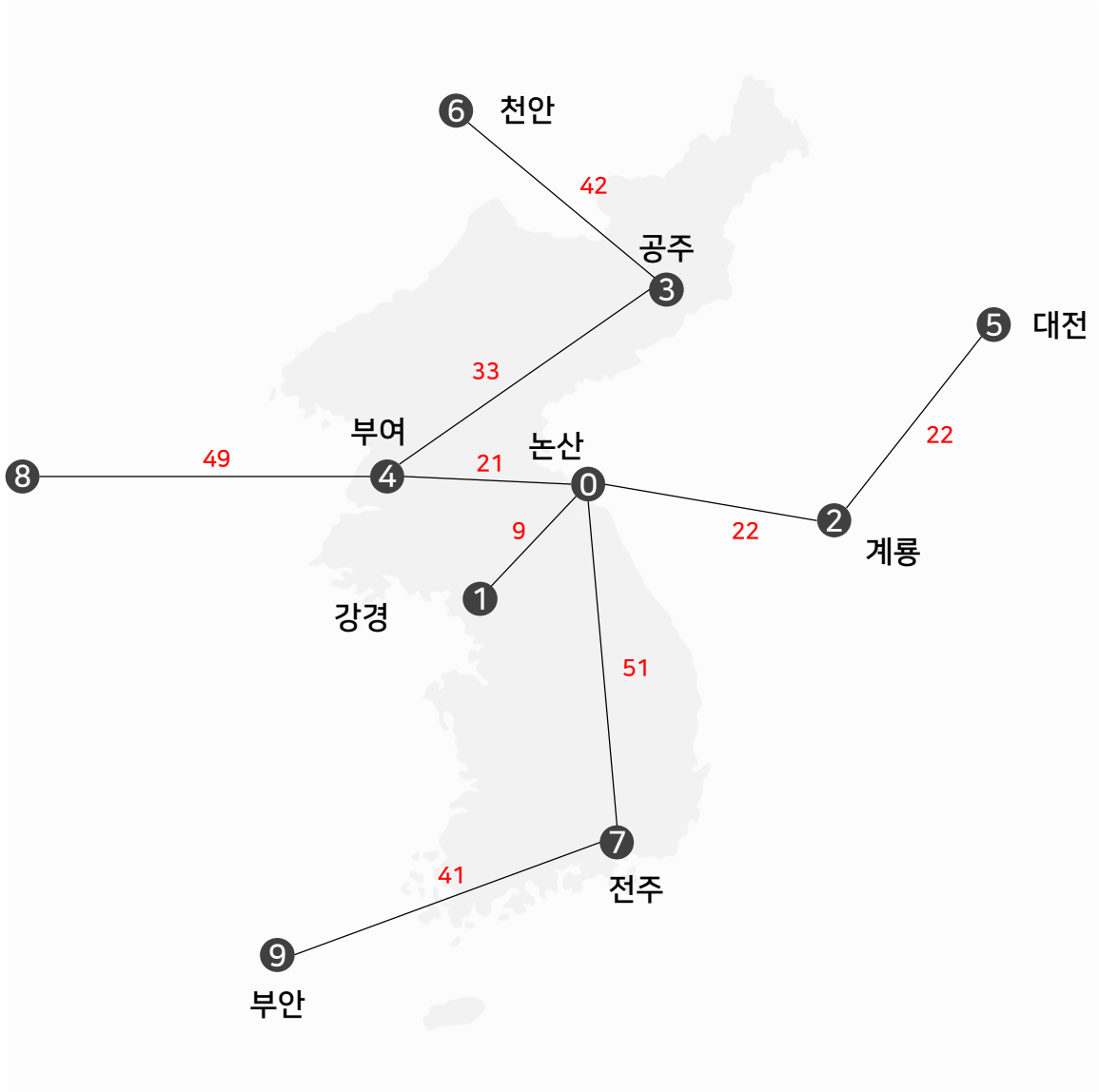
논산 → 전주 → 부안 → 부여 → 대천 → 공주 → 천안 → 계룡 → 대전 → 강경 → 논산

Kruskal2 MST을 이용한 TSP의 총 여행 거리는 548km 입니다.

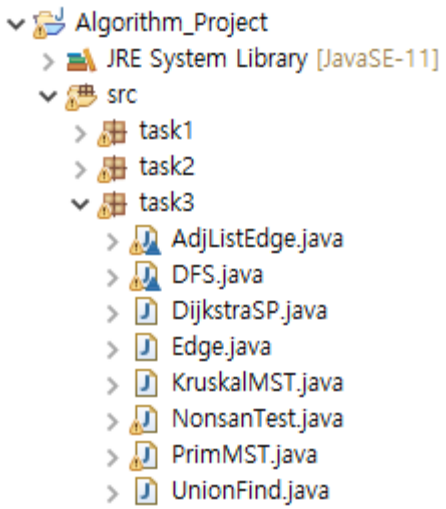
MST 간선 가중치 합 = 290

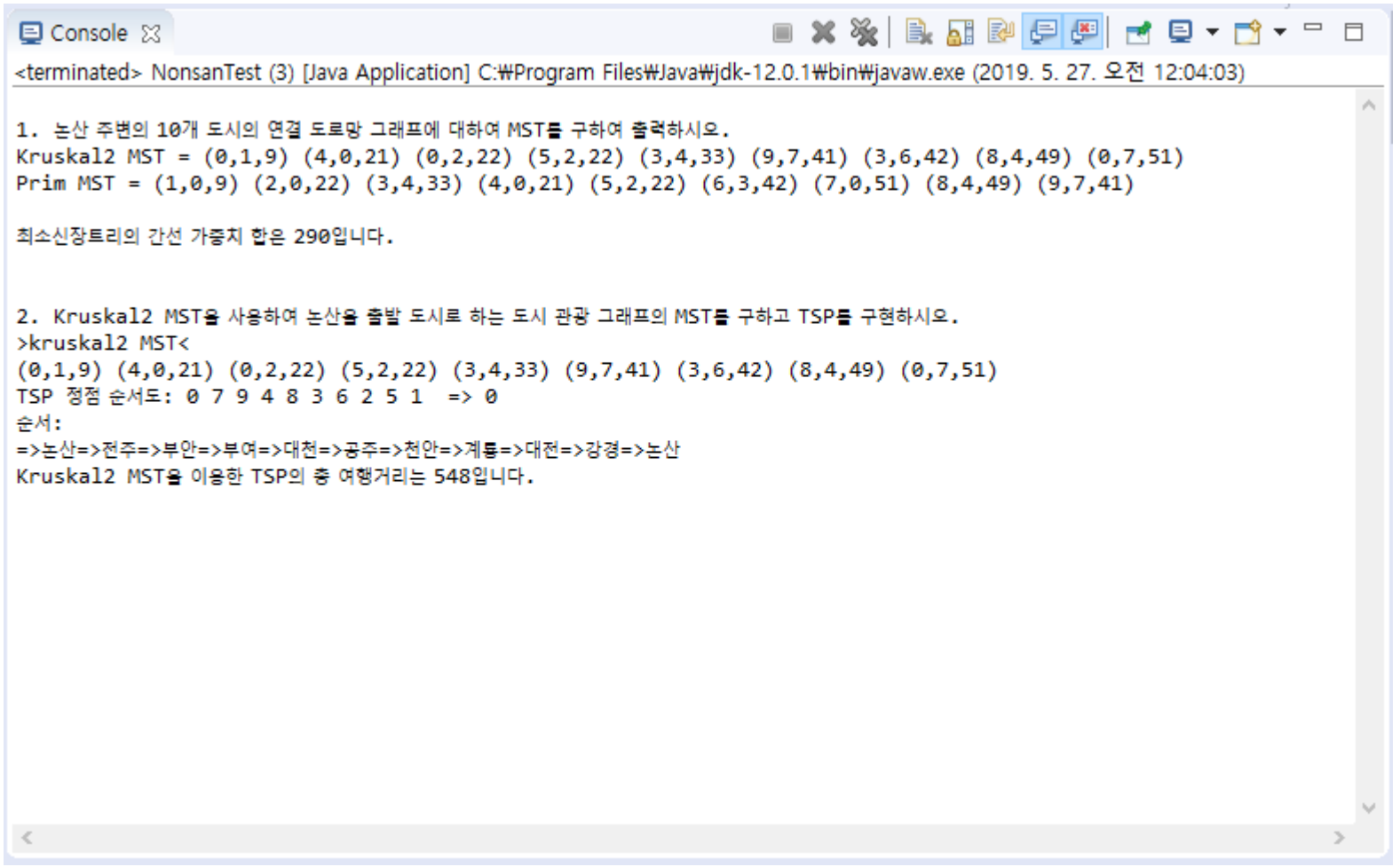
이 알고리즘의 근사 비율은 2M/M=2 보다 크지 않다.

즉, 근사해의 값이 최적해의 값의 2배를 넘지 않는다.



Result


Algorithm_Project
JRE System Library [JavaSE-11]
src
task1
task2
task3
AdjListEdge.java
DFS.java
DijkstraSP.java
Edge.java
KruskalMST.java
NonsanTest.java
PrimMST.java
UnionFind.java


Console
<terminated> NonsanTest (3) [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2019. 5. 27. 오전 12:04:03)
1. 논산 주변의 10개 도시의 연결 도로망 그래프에 대하여 MST를 구하여 출력하시오.
Kruskal2 MST = (0,1,9) (4,0,21) (0,2,22) (5,2,22) (3,4,33) (9,7,41) (3,6,42) (8,4,49) (0,7,51)
Prim MST = (1,0,9) (2,0,22) (3,4,33) (4,0,21) (5,2,22) (6,3,42) (7,0,51) (8,4,49) (9,7,41)
최소신장트리의 간선 가중치 합은 290입니다.
2. Kruskal2 MST을 사용하여 논산을 출발 도시로 하는 도시 관광 그래프의 MST를 구하고 TSP를 구현하시오.
>kruskal2 MST<
(0,1,9) (4,0,21) (0,2,22) (5,2,22) (3,4,33) (9,7,41) (3,6,42) (8,4,49) (0,7,51)
TSP 정점 순서도: 0 7 9 4 8 3 6 2 5 1 => 0
순서:
=>논산=>전주=>부안=>부여=>대천=>공주=>천안=>계룡=>대전=>강경=>논산
Kruskal2 MST을 이용한 TSP의 총 여행거리는 548입니다.

MST의 역할

MST의 결과를 이용하여 TSP 도시 방문 순서를 정하는데
MST는 어떤 역할을 할 수 있는지, 또 이 방법으로 하면 항상 최소 거리가 보장되는 지를 논하시오

TSP(Traveling Salesman Problem)는 단일 시작점 부터 시작하여
모든 도시를 방문하여 다시 시작점으로 돌아오는 최단 거리를 구하는 문제이다.

MST결과를 사용하였기 때문에 다른 지점까지 가는 거리를 최소로 정할 수 있다.

[Term Project: 과제 5] 논산 주변 도시까지의 최단거리

1. 과제 3에서 구한 논산 주변의 10개 도시의 연결 도로망 그래프에 대하여 논산을 출발점으로 9개 도시까지의 최단 거리를 구하여 출력하시오.

2. 논산을 출발점으로 주변 도시까지의 최단 거리 이동 정보를 제공하면 되므로 도시의 다른 정보는 필요 없고 도시명 대신 목록의 인덱스만 사용하여 간편화 하는데, 출발지 논산의 인덱스로 0을 배정하고 주변의 다른 도시에도 적절한 인덱스를 부여하시오.

3. 도시를 이런 인덱스로 표현한 그래프의 인접리스트를 생성하고 그래프를 출력하여 확인하시오.

4. 인덱스로 변환된 도시 그래프에 Dijakstra 알고리즘을 적용하여, 논산을 출발하여 각 도시까지 가는 최단거리 경로와 거리값을 모두 출력하시오.

출력 예제

최단거리 거리 값

0 → 1 까지의 최단 거리 :

0 → 2 까지의 최단 거리 :

…

0 →9 까지의 최단 거리 :

최단거리 경로

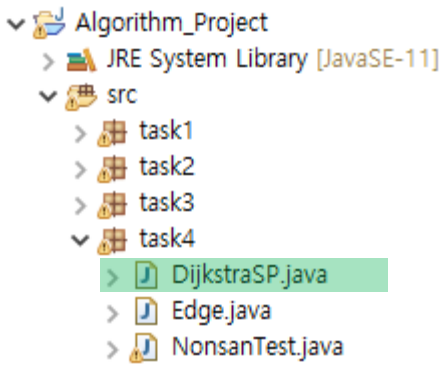
0 → … → 1

0 → … → 2

…

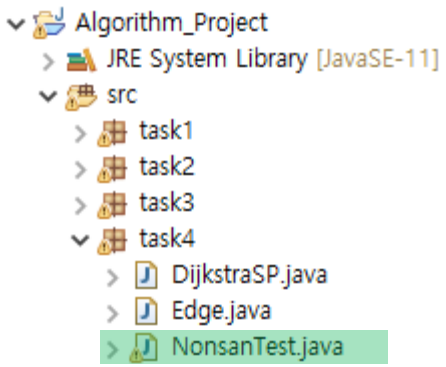
0 → … → 9

Source Code



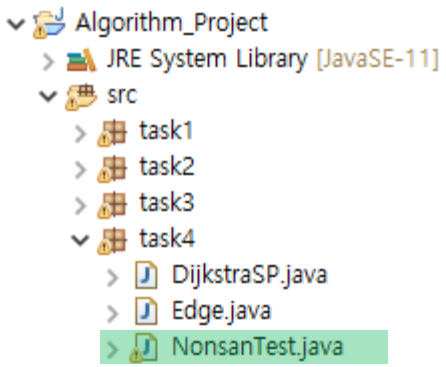

```java
public class DijkstraSP {
  public int N;//그래프 정점의 수
  List<Edge>[] graph;
  public int[] previous; //최단경로상 이전 정점을 기록함.
  public DijkstraSP(List<Edge>[] adjList) {
    N = adjList.length;
    previous = new int[N];
    graph = adjList;
  }
  public int[] shortestPath(int s) {
    boolean[] visited = new boolean[N];
    int[] D = new int[N];
    for(int i = 0; i< N; i++) {//초기화
      visited[i] = false;
      previous[i] = -1;
      D[i] = Integer.MAX_VALUE;
    }
    D[s]=0;
    for(int k = 0; k< N; k++) {
      int minVertex = -1;
      int min = Integer.MAX_VALUE;
      for(int j = 0; j < N; j++) {
        if((!visited[j]) &&(D[j] < min)) {
          min = D[j];
          minVertex = j;
        }
      }
      visited[minVertex] = true;
      for(Edge e : graph[minVertex]) { //minVertex에 인접한 정점에
        if(!visited[e.tv]) { //아직 방문 안된 정점이라면
          int currentDist = D[e.tv];
          int newDist = D[minVertex] + e.weight;
            if(newDist< currentDist) {
              D[e.tv] = newDist;
              previous[e.tv] = minVertex;//최종 최단경로를 역 방향으로 추출
            }
          }
        }
      }
      return D;
    }
}
```

Source Code



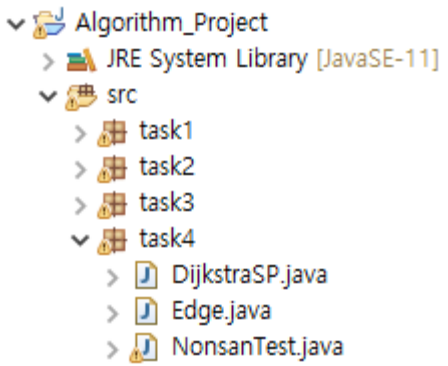

```java
public class NonsanTest {
  public static void main(String[] args) {
    int[][] dist = {
      {0, 9, 22, 34, 21, 67, 87, 51, 0, 80},
      {9, 0, 35, 0, 0, 74, 0, 0, 0, 0},
      {22, 35, 0, 0, 0, 22, 0, 0, 0, 0},
      {34, 0, 0, 0, 33, 0, 42, 0, 0, 0},
      {21, 0, 0, 33, 0, 0, 0, 0, 49, 0},
      {67, 74, 22, 0, 0, 0, 0, 0, 0, 0},
      {87, 0, 0, 42, 0, 0, 0, 0, 0, 0},
      {51, 0, 0, 0, 0, 0, 0, 0, 0, 41},
      {0, 0, 0, 0, 49, 0, 0, 0, 0, 0},
      {80, 0, 0, 0, 0, 0, 0, 41, 0, 0}};
    int N = dist.length;
    List<Edge>[] adjList = new List[N];
    for(int i =0; i<N; i++) {
      adjList[i] = new LinkedList<>();
      for(int j = 0; j< N; j++) {
        if(dist[i][j] != 0 ) {
          Edge e = new Edge(i,j, dist[i][j]);
          adjList[i].add(e);
        }
      }
    }
    DijkstraSP di = new DijkstraSP(adjList);
    int[] distance = di.shortestPath(0); //시작 인덱스 0
    for(int i = 0; i< distance.length; i++) {
      if(distance[i] == Integer.MAX_VALUE)
        System.out.println(i+"와 연결성분 없음");
      else System.out.println("(0 => "+i+") = 총 거리 합 : "+ distance[i]);
    }
      Stack<Integer> s = new Stack<>();
      System.out.println("\n정점 0(논산)으로부터의 최단경로");
      for(int i = 1; i < di.N; i++) {
      int back = i;
      s.push(back);//도착값 스택.
      while(back != 0) {
        s.push(di.previous[back]);//정점의 순서 스택.
        back = di.previous[back];
      }
      for(int q = 0; q< 10; q++) {
        if(!s.isEmpty()) {
          System.out.print("=>"+ s.pop());
        }
      }
    }
  }
}
```

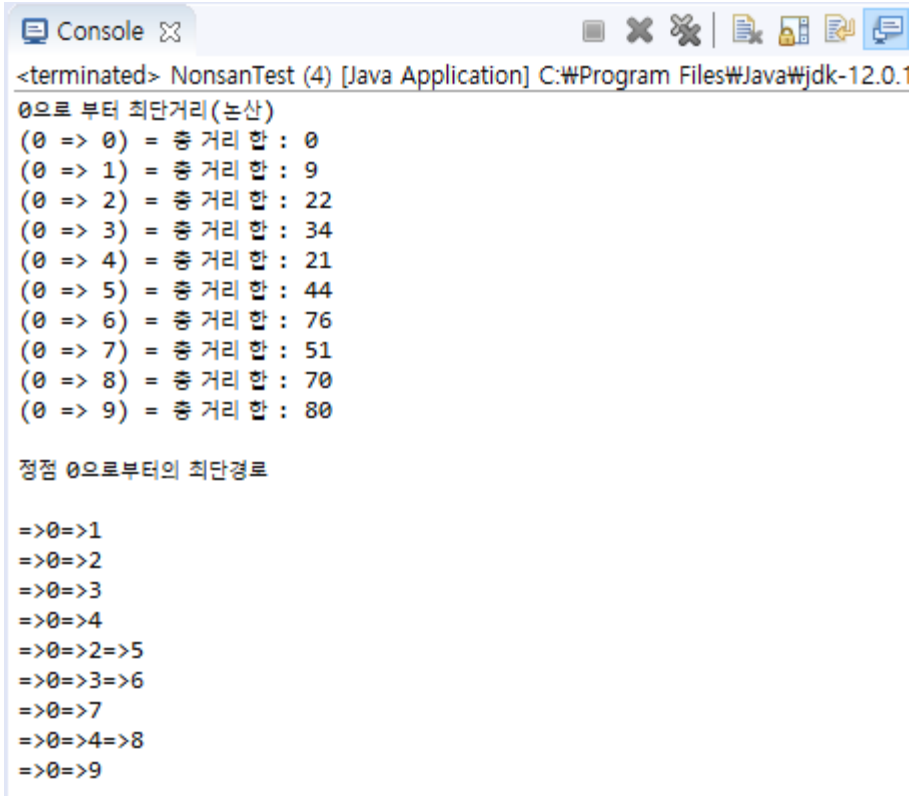
Source Code




```
        }
        Stack<Integer> s = new Stack<>();
        System.out.println("\n정점 0(논산)으로부터의 최단경로");

        for(int i = 1; i < di.N; i++) {
        int back = i;
        s.push(back);//도착값 스택.
        while(back != 0) {
          s.push(di.previous[back]);//정점의 순서 스택.
          back = di.previous[back];
        }
        System.out.println();

        for(int q = 0; q< 10; q++) {
          if(!s.isEmpty()) {
            System.out.print("=>"+ s.pop());
          }
        }
      }
    }
}
```

Result





출력 예제

최단거리 거리 값

0 → 1 까지의 최단 거리 :

0 → 2 까지의 최단 거리 :

…

0 →9 까지의 최단 거리 :

최단거리 경로

0 → … → 1

0 → … → 2

…

0 → … → 9

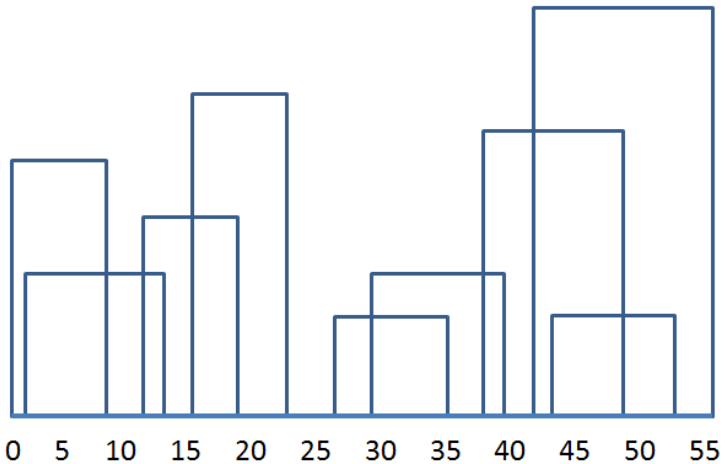
[Term Project: 과제 6] 대전 도심의 SkyLine 구하기

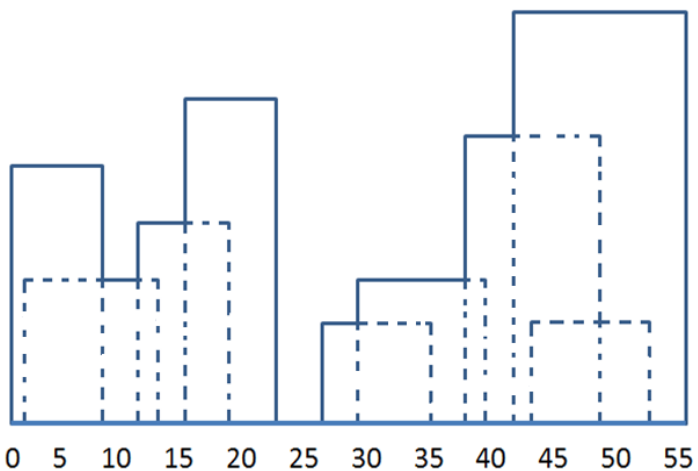
1. 대전의 어느 지역에서 바라본 도시의 빌딩들이 다음 그림과 같이 사각형 모양으로 겹쳐서 보일 때 하늘과 빌딩들의 가장자리를 오른쪽 그림과 같이 찾고자 한다.

2. 문제를 해결하기 위한 분할 정복 알고리즘의 가상코드를 제시하고, JAVA 언어로 구현하여 skyline의 좌표들을 아래의 입력 방법과 같이 출력하시오.

3. 빌딩들 외곽 윤곽선의 입력은 빌딩 풍경을 하나의 화폭(그림)에 담았다고 보고, 화폭의 좌축에서 우측으로는 x 축이고 높이를 y 축으로 표현하면 각 빌딩의 좌표 표시는 (좌측 x좌표, 높이 y좌표, 빌딩의 끝 우측 x좌표)의 3개 좌표값으로 표시할 수 있다. 예제 빌딩들의 좌표는 (0,5,9), (1,3,13), (11,4,18), (15,7,23), (27,2,35), (30,3,40), (38,6,48), (42,10,55), (43,2,53) 이다.

4. 구현한 알고리즘의 시간복잡도를 구하여 제시하시오.

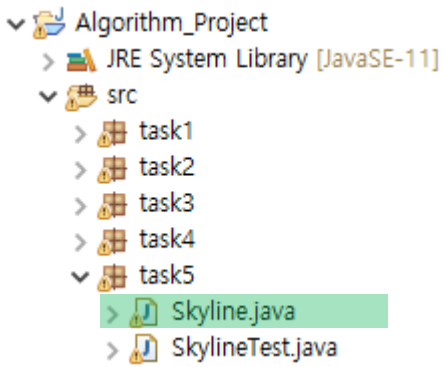




Pseudo Code

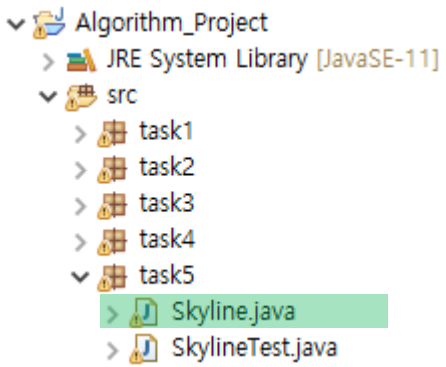
```
Skyline (B[1...n]: array of triples (start, height, end ))
1. if n=1
       Keypoints ← (B[1].start, B[1].height) (B[1].end, 0);
       return Keypoints ;
2. Keypoints ← an empty list;
3. ListA ← Skyline (B[1...n/2]);
4. ListB ← Skyline (B[n/2+1...n]);
5. CurHeightA ← 0;
6. CurHeightB ← 0;
7. WHILE ListA ≠ NULL and ListB ≠ NULL
     { IF the x field of the head of ListA < that of ListB
           Currentx ← the x field of the head of ListA,
           CurHeightA ← the y-field of the head of ListA.
           Append (Currentx, Max(CurHeightA, CurHeightB) to Keypoints,
           Delete the head of ListA.
       ELSE
           Currentx ← the x field of the head of ListB,
           CurHeightB ← the y-field of the head of ListB,
           Append (Currentx, Max(CurHeightA, CurHeightB) to Keypoints,
           Delete the head of ListB.
     }
8. IF ListA = NULL  append ListB to Keypoints
   ELSE append ListA to Keypoints
9. return Keypoints // List of points(x, y) sorted by x
```

Source Code



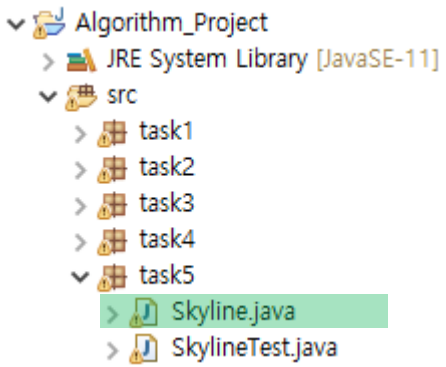

```java
package task5;

import java.lang.reflect.Array;
import java.util.ArrayList;

public class Skyline {
    Building[] building;
    public class Building {//좌표 입력
        int left, height, right;
        public Building() {
            this.left = 0;
            this.height = 0;
            this.right = 0;
        }
        public Building(int left, int height, int right) {//building 좌표 생성자
            this.left = left;
            this.height = height;
            this.right = right;
        }
    }
    public class Point {
        int x, height;

        public Point() {
            this.x = 0;
            this.height = 0;
        }
        public Point(int left, int height) {
            this.x = left;
            this.height = height;
        }
        public String toString() {//Point 자료형을 리턴할 시 그 값의 x값과 높이를 출력함.
            return this.x + ", " + this.height;
        }
    }
    public void buildArray(int n) {//배열의 크기를 지정.
        this.building = new Building[n];

        for(int i = 0 ; i < n; i++) {//배열 초기화
            this.building[i] = new Building();
        }
    }
```

Source Code



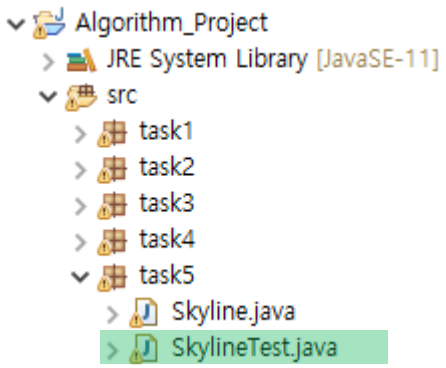

```java
public ArrayList<Point> find_skyline(Building[] build, int first, int end) {//재귀적으로 분할하는 매서드
    if( first == end ) {//시작과 끝이 같으면 skyline에 추가
        ArrayList<Point> partition_skyline = new ArrayList<Point>();
        partition_skyline.add(new Point(build[first].left, build[first].height));
        partition_skyline.add(new Point(build[end].right, 0));
        return partition_skyline;
    }
    int mid = ( first + end ) / 2;
    ArrayList<Point> sky1 = this.find_skyline(build, first, mid);//partition을 나눠 재귀 분할하는 부분
    ArrayList<Point> sky2 = this.find_skyline(build, mid+1, end);
    return merge_skyline(sky1, sky2);//나눈 부분을 합쳐주는 메사드
}
public ArrayList<Point> merge_skyline(ArrayList<Point> sky1, ArrayList<Point> sky2) {//정복 메서드
    ArrayList<Point> skyline = new ArrayList<Point>();
    int current_height1 = 0;
    int current_height2 = 0;
    int current_x, max_h;

    while(sky1.size()>0 && sky2.size()>0) {
        if(sky1.get(0).x < sky2.get(0).x) {//sky2의 x값이 더 크다면
            current_x = sky1.get(0).x;
            current_height1 = sky1.get(0).height;
            max_h = current_height1;

            if(current_height2 > max_h) {
                max_h = current_height2;
            }
            skyline.add(new Point(current_x, max_h));
            sky1.remove(0);
        }
        else {
            current_x = sky2.get(0).x;
            current_height2 = sky2.get(0).height;
            max_h = current_height1;

            if(current_height2 > max_h) {
                max_h = current_height2;
            }
            skyline.add(new Point(current_x, max_h));
            sky2.remove(0);
        }
    }
```

Source Code



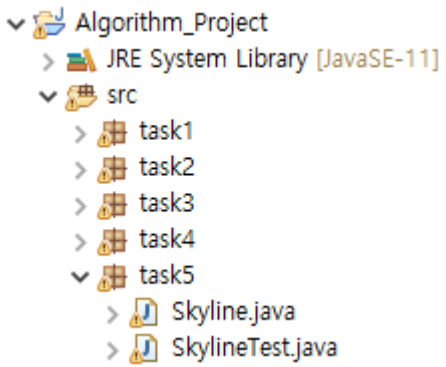

```
        while(sky1.size() > 0) {//sky2의 사이즈가 0일때 sky1의 x좌표와 높이를 skyline에 추가함.
            skyline.add(new Point(sky1.get(0).x, sky1.get(0).height));
            sky1.remove(0);
        }

        while(sky2.size() > 0) {//sky1의 사이즈 0일때 sky2의 x좌표와 높이를 skyline에 추가함
            skyline.add(new Point(sky2.get(0).x, sky2.get(0).height));
            sky2.remove(0);
        }

        return skyline;//완성된 skyline리턴
    }

    // Skyline Print
    public void print_skyline() {
        int size = this.find_skyline(this.building, 0 , this.building.length-1).size();

        for(int i = 0; i < size; i++) {
            if( i != 0 && i != size-1 && this.find_skyline(this.building, 0 , this.building.length-1).get(i-
1).height == this.find_skyline(this.building, 0 , this.building.length-1).get(i).height) {
            //높이가 같다면 높이의 변화가 없으므로 출력아무 동작도 안함
            }
            else
            if(i == size-1) System.out.print(this.find_skyline(this.building, 0 , this.building.length-
1).get(i));
            //높이가 변한다면 그 지점의 x좌표와 그지점의 높이를 출력함.
            else System.out.print(this.find_skyline(this.building, 0 , this.building.length-1).get(i)+", ");
            //Point형 자료형을 사용했고 리턴시 x값과 높이를 출력하게 함.
        }
    }

}
```

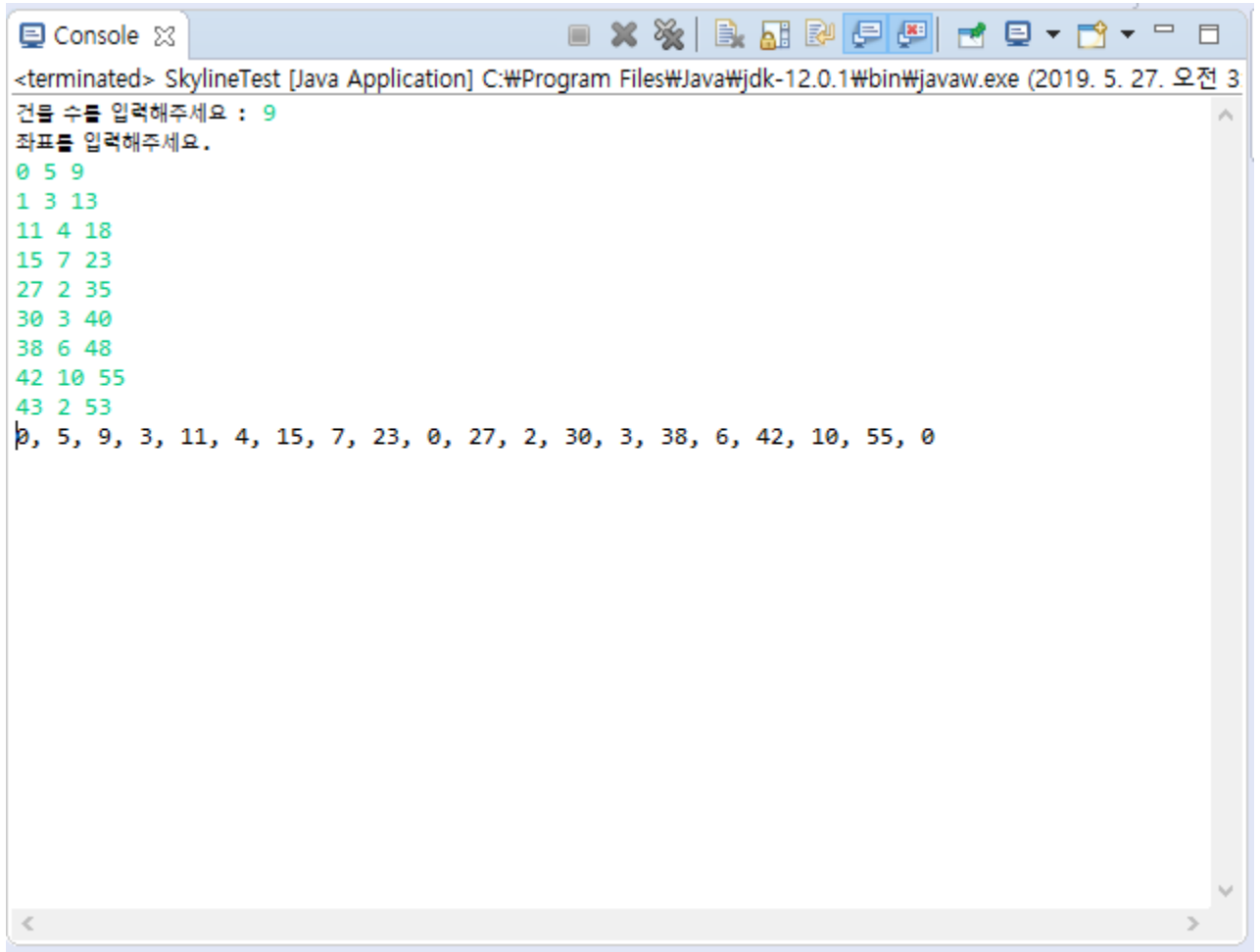
Source Code




```java
package task5;

import java.util.Scanner;


public class SkylineTest {
    public static void main(String[] args) {

        Skyline skyline = new Skyline();
        Scanner stdIn = new Scanner(System.in);

        System.out.print("건물 수를 입력해주세요 : ");//배열의 크기만큼
        int n = stdIn.nextInt();
        skyline.buildArray(n);

        System.out.println("좌표를 입력해주세요.");

        for(int i = 0; i < n; i++) {//하나의 건물의 좌표를 왼쪽x, 왼쪽y, 오른쪽x값을 입력함
        skyline.building[i].left = stdIn.nextInt();
        skyline.building[i].height = stdIn.nextInt();
        skyline.building[i].right = stdIn.nextInt();
        }



       // System.out.println(skyline.find_skyline(skyline.building,0, n-1));
        skyline.print_skyline();
    }
}
```

Result


Algorithm_Project
JRE System Library [JavaSE-11]
src
task1
task2
task3
task4
task5
Skyline.java
SkylineTest.java


Console
<terminated> SkylineTest [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2019. 5. 27. 오전 3
건물 수를 입력해주세요 : 9
좌표를 입력해주세요.
0 5 9
1 3 13
11 4 18
15 7 23
27 2 35
30 3 40
38 6 48
42 10 55
43 2 53
0, 5, 9, 3, 11, 4, 15, 7, 23, 0, 27, 2, 30, 3, 38, 6, 42, 10, 55, 0


Algorithm
P15_FInal Report
Team_05

INDEX

[Term Project: 과제 7] 관광 안내 우선순위

1. 논산 주변 도시의 관광명소를 표현하는 클래스 자료형을 정의하시오.
   - [과제1]에서 조사한 도시 별 관광명소 데이터를 class 자료형 attractions에 저장.
   - 이 클래스에는 필드 변수로 (도시명: city), (논산에서의 거리: distance),
     (관광명소이름: spot), (평가점수: rating)를 갖는다.
   - 제공하는 메소드는 클래스 객체 생성자, getters/setters 등임
2. 100개 관광명소의 데이터를 저장할 클래스 자료형 attractions인 객체배열
   tourism을 생성하고, 100개 관광명소를 이 객체배열에 저장하시오.
3. 각 도시의 관광명소들의 평가점수를 합산하여 도시의 종합 평가점수를
   계산하고, (도시명, 종합 평가점수)를 내림차순으로 정렬하여 출력하시오.
4. 논산을 출발하여 방문하려는 관광명소의 안내 순서는
   1)논산에서의 거리 50%
   2)관광명소의 평가점수 50%를 반영하여 종합 점수가 높은 순이라고 할 때, 객체배열
   attractions을 종합점수 기준으로 내림차순으로 정렬, 상위 10개 관광명소를 출력.
5. 4의 관광명소를 모두 방문하는 최단거리 경로와 총거리를 출력하시오.
   단, 한 도시내의 관광명소간 이동거리는 0으로 정의한다.

출력 예제

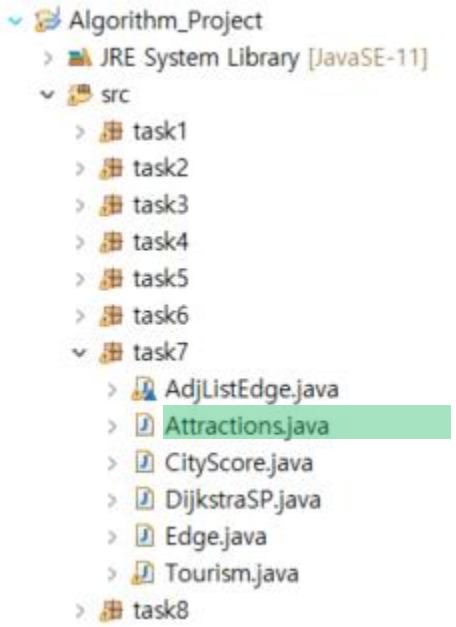
도시들의 관광명소 종합 평가 점수
도시 1 : 00점
도시 2 : 00점
…
도시 10 : 00점

상위 10개의 관광명소 출력
도시1의 관광명소 [ ] 이고 00점 입니다.
…
도시 10의 관광명소는 [ ] 이고 00점 입니다.

관광명소를 방문하는 도시는 (출발 도시 ~ 도착 도시) 이고
방문 경로의 총 거리는 00km 입니다.

과제 7 . 관광 안내 우선 순위

Source Code



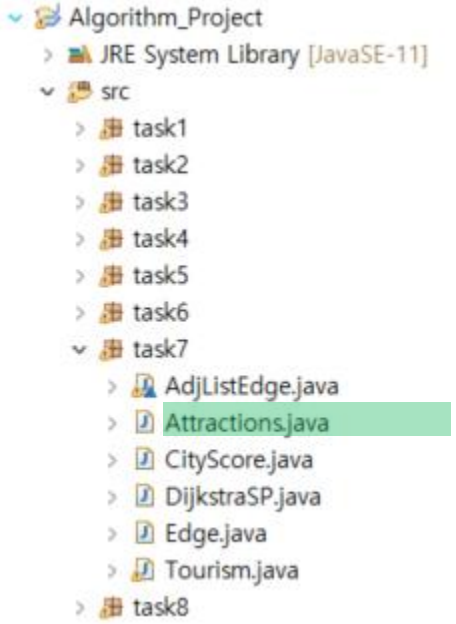

```java
package task7;

public class Attractions implements Comparable<Attractions> {
    String City;
    int distance;
    String spot;
    int rating;
    int MaxScore;
    int half = 50;

    public Attractions(String City, int distance, String spot, int rating) {
        this.City = City;
        this.distance = distance;
        this.spot = spot;
        this.rating = rating;

        if (distance >= 0 && distance <= 30) { // 거리에 따른 점수형식을 사용
            this.MaxScore = (int) (rating / 2) + half;
        } else if (distance > 30 && distance <= 40) {
            this.MaxScore = (int) (rating / 2) + half - 2;
        } else if (distance > 40 && distance <= 50) {
            this.MaxScore = (int) (rating / 2) + half - 4;
        } else if (distance > 50 && distance <= 60) {
            this.MaxScore = (int) (rating / 2) + half - 6;
        } else if (distance > 60 && distance <= 70) {
            this.MaxScore = (int) (rating / 2) + half - 8;
        } else {
            this.MaxScore = (int) (rating / 2) + half - 10;
        }

    }
```

도시별 관광명소 데이터를 class 자료형 attractions에 저장할 수 있도록 클래스 자료형을 구성함

이 클래스는 (도시명: city), (논산에서의 거리: distance), (관광명소이름: spot), (평가점수: rating)를 갖는다

과제 7 . 관광 안내 우선 순위

Source Code



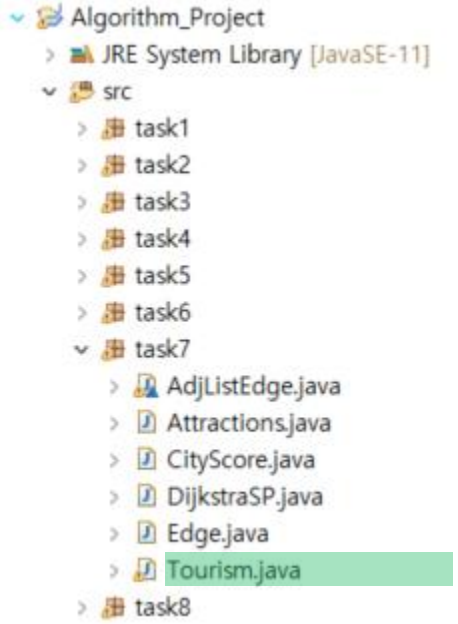

```
    public String getCity() {
        return City;
    }
    public void setCity(String city) {
        City = city;
    }
    public int getDistance() {
        return distance;
    }
    public void setDistance(int distance) {
        this.distance = distance;
    }
    public String getSpot() {
        return spot;
    }
    public void setSpot(String spot) {
        this.spot = spot;
    }
    public int getRating() {
        return rating;
    }
    public void setRating(int rating) {
        this.rating = rating;
    }
    @Override
    public String toString() {
        return City + "도시의 관광명소 " + spot + "의 종합 점수는" + MaxScore + "입니다";
    }

    @Override
    public int compareTo(Attractions o) {
        if (this.MaxScore < o.MaxScore) {
            return 1;
        } else if (this.MaxScore == o.MaxScore) {
            return 0;
        } else {
            return -1;
        }
    }

}
```

도시별 관광명소 데이터를 class 자료형 attractions에 저장할 수 있도록 클래스 자료형을 구성함

이 클래스는 (도시명: city), (논산에서의 거리: distance), (관광명소이름: spot), (평가점수: rating)를 갖는다

과제 7 . 관광 안내 우선 순위

Source Code



(1 / 2)

```
Attractions[] tourism = new Attractions[100];
// 논산으로부터 거리 distance[]
// 논산
tourism[count++] = new Attractions("논산", distance[0], "관촉사", 82);
tourism[count++] = new Attractions("논산", distance[0], "계백장군", 80);
tourism[count++] = new Attractions("논산", distance[0], "돈암서원", 78);
tourism[count++] = new Attractions("논산", distance[0], "탑정호", 90);
tourism[count++] = new Attractions("논산", distance[0], "대둔산", 84);
tourism[count++] = new Attractions("논산", distance[0], "KT&G 상상마당", 86);
tourism[count++] = new Attractions("논산", distance[0], "먹바위 농촌체험 휴양마을", 84);
tourism[count++] = new Attractions("논산", distance[0], "논산 선샤인 랜드", 80);
tourism[count++] = new Attractions("논산", distance[0], "신동회관", 84);
tourism[count++] = new Attractions("논산", distance[0], "산애들애", 96);
// 강경
tourism[count++] = new Attractions("강경", distance[1], "옥녀봉 공원", 84);
tourism[count++] = new Attractions("강경", distance[1], "죽림서원", 81);
tourism[count++] = new Attractions("강경", distance[1], "강경 역사관", 78);
touri
touri
touri
touri
touri
tourism[count++] = new Attractions("강경", distance[1], "강경 해물 칼국수", 88);
```

100개 관광명소의 데이터를 저장할 클래스 자료형 attractions인

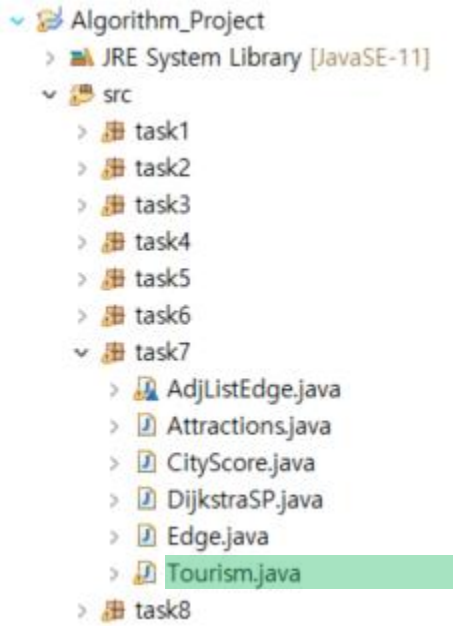
객체배열 Tourism을 생성하고, 100개 관광명소를 객체배열에 저장.

● ● ●

```
// 부안
tourism[count++] = new Attractions("부안", distance[9], "부안서문안산당", 87);
tourism[count++] = new Attractions("부안", distance[9], "반계선생유적지", 84);
tourism[count++] = new Attractions("부안", distance[9], "선운사 대웅전", 74);
tourism[count++] = new Attractions("부안", distance[9], "채석강", 74);
tourism[count++] = new Attractions("부안", distance[9], "적벽강", 84);
tourism[count++] = new Attractions("부안", distance[9], "곰소염전", 63);
tourism[count++] = new Attractions("부안", distance[9], "원숭이 학교", 95);
tourism[count++] = new Attractions("부안", distance[9], "부안 영상테마파크", 87);
tourism[count++] = new Attractions("부안", distance[9], "씨윈드", 97);
tourism[count++] = new Attractions("부안", distance[9], "이화자백합죽", 91);
```

과제 7 . 관광 안내 우선 순위

Source Code

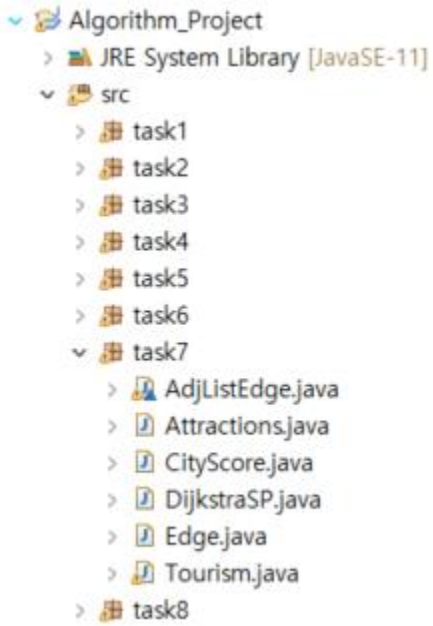


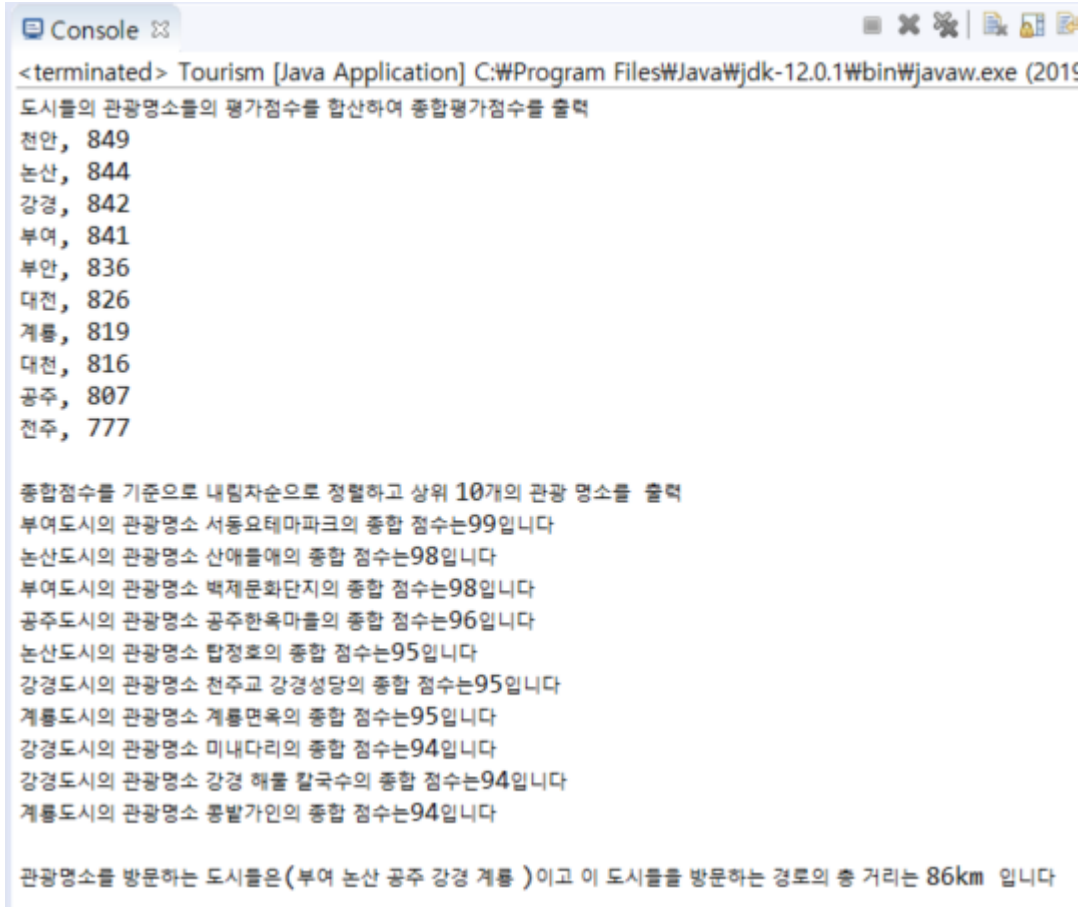

```
System.out.println("도시들의 관광명소들의 평가점수를 합산하여 종합평가점수를 출력");

int cityCount = 10;
int cityCount2 = 0;
CityScore[] CS = new CityScore[N];

for (int i = 0; i < N; i++) { // 각 도시마다 종합점수를 더함
    int SumScore = 0;
    for (int j = 0; j < cityCount; j++) {
        SumScore += tourism[cityCount2].rating;
        CS[i] = new CityScore(City[i], SumScore);
        cityCount2++;
    }
}
Arrays.sort(CS); // 3.종합점수 별 Sort
for (int i = 0; i < N; i++) {
    System.out.println(CS[i]);
}
Arrays.sort(tourism); // 4.평가점수 별 Sort
ArrayList<String> AS = new ArrayList<>();
System.out.println("\n종합점수를 기준으로 내림차순으로 정렬하고 상위 10개의 관광 명소를 출력"); // AS배열에 값이 없으면 추가.
for (int i = 0; i < N; i++) {
    System.out.println(tourism[i]);
    if (!AS.contains(tourism[i].City)) {
        AS.add(tourism[i].City);
    }
}
System.out.print("\n관광명소를 방문하는 도시들은("); // 도시점수 기준 = 키로수별 점수로 50% + 관광명소저뭇 50%
for (int i = 0; i < AS.size(); i++) {
    System.out.print(AS.get(i) + " ");
}
int SumDistance = 0;
for (int i = 0; i < AS.size(); i++) { // 원래있는 CIty배열과 찾은 도시와 비교해 일치하면 그 도시의 거리를 더함
    for (int j = 0; j < N; j++) {
        if (AS.get(i) == City[j]) {
            SumDistance += distance[j]; // distance는 이전에 구한 논산으로부터 각도시가지의 최단거리
        }
    }
}
System.out.println(")이고 이 도시들을 방문하는 경로의 총 거리는 " + SumDistance + "km 입니다");
```

각 도시의 관광명소들의 평가점수를 합산하여
도시의 종합 평가점수를 계산하고, (도시명, 종합 평가점수)를
내림차순으로 정렬하여 출력.

관광명소를 모두 방문하는 최단거리 경로와 총거리를 출력

Result





출력 예제

도시들의 관광명소 종합 평가 점수

도시 1 : 00점

도시 2 : 00점

…

도시 10 : 00점

상위 10개의 관광명소 출력

도시1의 관광명소 [ ] 이고 00점 입니다.

…

도시 10의 관광명소는 [ ] 이고 00점 입니다.

관광명소를 방문하는 도시는 (출발 도시 ~ 도착 도시) 이고

방문 경로의 총 거리는 00km 입니다.

[Term Project: 과제 8] 관광 키워드 찾기

구현하는 메소드

1) 메모장 xxxx.txt 파일을 읽고 전처리하여 병합된 문자열을 반환하는 메소드 textLines(String fileName)
2) 원시적 매칭 메소드 naiveMatch(String TextSeq, String pattern)
3) 상태변이 매트릭스를 인자로 받아 매칭하는 오토마타 메소드 automataMatch(String TextSeq, char inchar[], int TM[][])
4)패턴을 주면 KMP 상태변이 매트릭스를 스스로 만들어 매칭하는 메소드 kmpaMatch(String TextSeq, String pattern, int PIE[])
5)메인메소드의 기능: 사용하는 변수의 정의, 필요한 클래스 객체의 생성, 탐색 대상 파일명/탐색 패턴을 입력받기, 3가지 매칭 메소드 호출하기

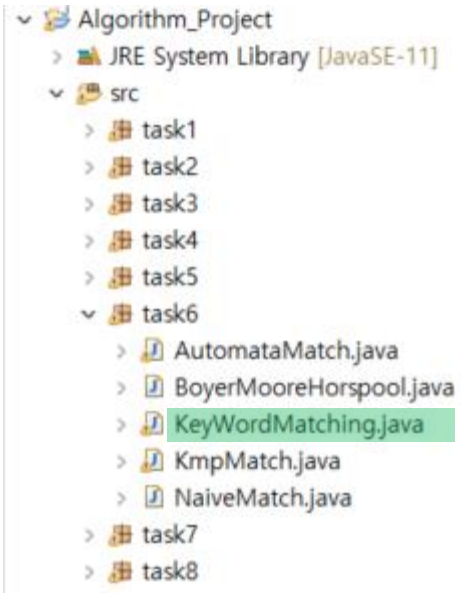
출력 예제

돈암서원txt :
안녕하세요.논산시 소셜미디어 서포터즈 이창헌입니다.
오늘 소개해드릴 논산의 명소는 국가지정 사적 제383호로 지정된 돈암서원입니다.
……
논산여행 오셨다면 교육의 핫플레이스 장소인 돈암서원은 꼭 방문하시기 바랍니다.
C:/Users/user/Desktop/돈암서원.txt 파일 읽기가 종료되었습니다.

탐색할 Key Word 패턴을 입력하세요.>>돈암서원
원시적 매칭으로 찾은 위치:
65, 75, 190, 256, 346, 483, 1175, 1246, 1554, 1651, 1847
Automata 매칭으로 찾은 위치:
65, 75, 190, 256, 346, 483, 1175, 1246, 1554, 1651, 1847
BoyerMooreHorspool 매칭으로 찾은 위치:
65, 75, 190, 256, 346, 483, 1175, 1246, 1554, 1651, 1847

과제 8 . 관광 키워드 찾기

Source Code



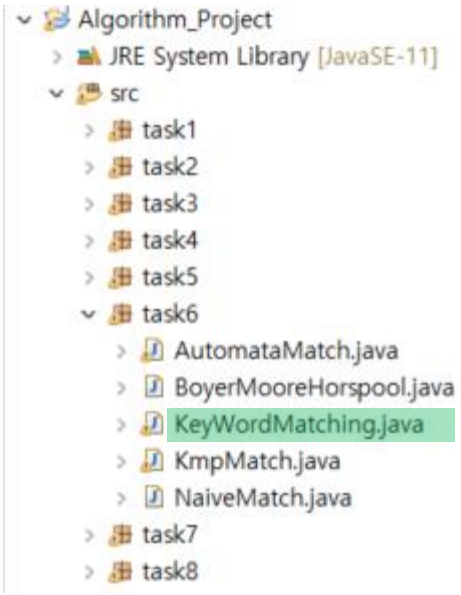

```java
public class KeyWordMatching {

    public static void main(String[] args) throws FileNotFoundException {

        File file = new File("C:\\Users\\USER\\Desktop\\2019년\\알고리즘\\돈암서원.txt");
        FileReader Freader = new FileReader(file);
        Scanner stdIn = new Scanner(file);
        String don = "";
        String ex = "";
        while (stdIn.hasNextLine()) {
            ex = stdIn.nextLine();
            don = don.concat(ex); // 따로 \n을 제거하지않아도 concat을 이용해 사라지게함.
            System.out.println(ex);
        }
        System.out.println("C:\\Users\\USER\\Desktop\\2019년\\알고리즘\\돈암서원.txt 파일읽기가 종료되었습니다.");

        System.out.println("탐색할 Key Word 패턴을 입력하세요.");
        String pattern = "";
        Scanner s = new Scanner(System.in);
        pattern = s.nextLine();
        stdIn.close();
```

돈암서원.txt 의 내용을 읽어오고

Concat 을 이용하여 전체 글이 하나의 문자열이 되도록 전처리한 텍스트 작성.

과제 8 . 관광 키워드 찾기

Source Code



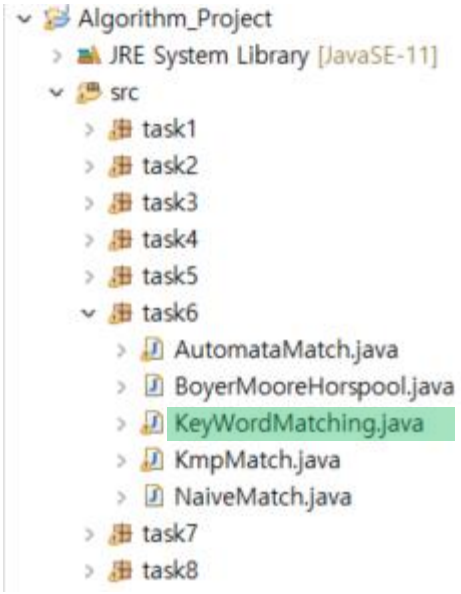
1. 원시적 매칭을 이용한 패턴 검색

```
System.out.println("원시적 매칭으로 찾은 위치: ");
NaiveMatch NM = new NaiveMatch();
System.out.print("패턴이 시작되는 위치: ");
NM.findPattern(don, pattern);
System.out.println("");
```

```
탐색할 Key Word 패턴을 입력하세요.
돈암서원

원시적 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

Automata 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

boyerMooreHorspool로 매칭으로 찾은 위치:
패턴이 시작되는 위치: 64 73 186 251 339 473 1154 1224 1527 1622 1815

KMP Automata 매칭으로 찾은 위치:
KMP automata: [  0  0  0  0  0 ]
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815
```

과제 8 . 관광 키워드 찾기

Source Code



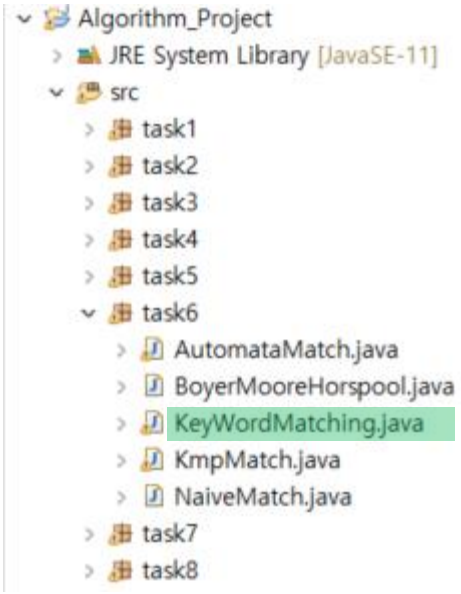
2. Automata 매칭을 이용한 패턴 검색

```
System.out.println("Automata 매칭으로 찾은 위치: ");
char inchar[] = { '돈', '암', '서', '원' }; // 입력 문자의 집합
int[][] TM = { // 돈암서원 상태 전이 함수 매트릭스
        { 1, 0, 0, 0 },
        { 0, 2, 0, 0 },
        { 0, 0, 3, 0 },
        { 0, 0, 0, 4 },
        { 0, 0, 0, 0 },
        };

System.out.print("패턴이 시작되는 위치: ");
AutomataMatch AM = new AutomataMatch();
AM.findPattern(don, inchar, TM);
System.out.println();
```

```
탐색할 Key Word 패턴을 입력하세요.
돈암서원

원시적 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

Automata 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

boyerMooreHorspool로 매칭으로 찾은 위치:
패턴이 시작되는 위치: 64 73 186 251 339 473 1154 1224 1527 1622 1815

KMP Automata 매칭으로 찾은 위치:
KMP automata: [  0  0  0  0  0 ]
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815
```

과제 8 . 관광 키워드 찾기

Source Code



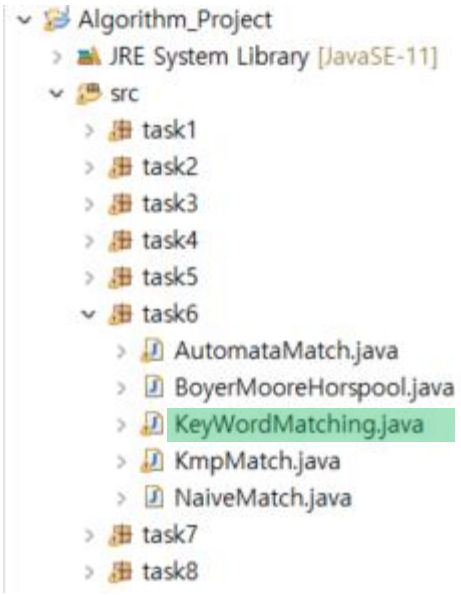
3. BoyerMooreHorspool 매칭을 이용한 패턴 검색

```
int index = 0;
int count = 0;
String pt = don;

BoyerMooreHorspool by = new BoyerMooreHorspool();
System.out.print(" \nboyerMooreHorspool로 매칭으로 찾은 위치:\n");
System.out.print("패턴이 시작되는 위치: ");
while (pt.length() >= pattern.length()) {
    index = by.BM_Search(pt, pattern);
    pt = pt.substring(index + 4);
    System.out.print(index + count + " ");
    count += index + 4; // 4는 돈암서원의 글자 길이
}
```

```
탐색할 Key Word 패턴을 입력하세요.
돈암서원

원시적 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

Automata 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

boyerMooreHorspool로 매칭으로 찾은 위치:
패턴이 시작되는 위치: 64 73 186 251 339 473 1154 1224 1527 1622 1815

KMP Automata 매칭으로 찾은 위치:
KMP automata: [  0  0  0  0  0 ]
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815
```

과제 8 . 관광 키워드 찾기

Source Code



4.KMP automata를 이용한 패턴 검색

```
System.out.println("\n\nKMP Automata 매칭으로 찾은 위치: ");

KmpMatch c = new KmpMatch();
int PIE[] = c.preprocessing(pattern);
System.out.print("패턴이 시작되는 위치: ");
c.findKMP(don, pattern, PIE);
```

```
탐색할 Key Word 패턴을 입력하세요.
돈암서원

원시적 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

Automata 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

boyerMooreHorspool로 매칭으로 찾은 위치:
패턴이 시작되는 위치: 64 73 186 251 339 473 1154 1224 1527 1622 1815

KMP Automata 매칭으로 찾은 위치:
KMP automata: [  0  0  0  0  0 ]
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815
```

Console

<terminated> KeyWordMatching [Java Application] C:₩Program Files₩Java₩jre-10.0.2₩bin₩javaw.exe (2019. 6. 14. 오

```
안녕하세요.논산시 소셜미디어 서포터즈 이창헌입니다.
오늘 소개해드릴 논산의 명소는 국가지정 사적 제383호로 지정된 돈암서원입니다.
돈암서원은 1634년 (인조12)에 지방 유림의 공의로 김장생선생의 덕행과 학문을 추모하기 위해
제자들이 창건한 후 스승의 위패를 봉안하고 제사를 지내오다가 왕이 현판을 내려줌으로써 사액서원이 되었습니다.
돈암서원 입구에는 한옥마을도 오픈을 앞두고 있어 함께 둘러보기 좋은 장소입니다.
논산에 대해서 하나씩 알아가다 보니, 돈암서원이라는 곳은 웃고 즐기는 장소가 아니었습니다.
가벼운 마음으로 사진촬영하고 가려 했던 계획에서 약간 변경을 했습니다.
이곳에 대해서 좀 더 궁금해졌고 돈암서원에 상주하고 계신 문화관광해설사님께서 알려주시는 내용을 듣고 싶어졌습니다.
해설사님께서는 논산에 대한 기본적인 지식을 알고 듣게 되면 이해하기도 빠르고 모두가 공통점이 있다는 것을 알게 된다고 합니다.
논산은 서원이 많은 지역입니다.
돈암서원은 기본이고 노강서원, 죽림서원, 충곡서원, 행림서원 등 다수의 서원들이 지어져 있습니다.
특히 서원을 배경 삼아 배롱나무꽃과 같은 아름다운 모습을 사진에 담기 위해서 방문하는 촬영 작가들이 좋아하는 장소이기도 합니다.
가장 먼저 입구에 "입덕문"이 보이고 입덕문을 통과해서 안으로 들어가면 오른쪽에 문화관광해설사가 상주하는 경회당과
왼편으로 응도당이 있고, 좀 더 안쪽으로 들어가면 "양성당"이 있습니다.
응도당은 국가 보물 제1569호로 강학공간이며 전면에 직각 방향으로 들어져 배치되어있습니다.
다른 서원과는 다른 배치를 하고 있다는 생각이 바로 드실 겁니다. 그리고 "응도당"은 현재의 학교와도 같은 장소로 선비들이 공부를 하던 장소였습니다.
그래서 그럴까요? 서원 옛 터에 남아있던 모습을 현재 이곳으로 옮기는 과정에서 기와에 작성되어있던 다수의 기록들을 발견했다고 합니다.
아마도 기왓장에 공부를 했던 건 아닐까요? 그것보다는 기와에 기록되어있던 내용들을 살펴볼 때 오래된 건물이었음을 알게 해주었다고 합니다.
양성당 건물은 정면 5칸, 측면 2칸 크기의 평면으로 되어있고
이를 만든 후 정면 5칸 중 중앙 3칸에 전퇴가 달린 우물마루의 마루방으로 만들어진 대청으로 사용하게끔 만들어져 있습니다.
이렇게 아름다운 양성당은 사계 김장생 선생님께서 살아계셨을 때 강학을 하시던 강당입니다.
양성당 앞에 생각보다 큰 "돈암서원원정비"가 있었습니다. 문화재자료 제 366호로 지정되어있는 것으로
조선시대 중기 문인 사계 김장생 선생님의 문하생들이 돈암서원을 세운 사연과 사계 선생과 그의 아들인 신독재 김집 부자의 업적과 학문에 대해 적은 비석이라고 합니다.
내용을 듣기 전에는 김장생 선생님에 대한 이야기만 있을 것으로 생각했었는데,
비석 하나로 이곳에 대한 역사를 한눈에 알 수 있어서 좋았던 것 같습니다. 비석을 자세히 보면 받침대에는 연꽃 모양이 새겨져있습니다.
비 머리는 내려간 게 아니고 올라가 있으며 현종 10년에 세워졌다는 비문이 적혀있습니다.
그리고 비문은 "송시열"이 지었고 글은 "송준길"이 작성했으며 앞면에 새겨져 있는 전서체 제목은 "김만기"가 작성했다고 기록되어있습니다.
돈암서원의 전체적인 모습은 과거 학교를 방문한 기분이 들었습니다.
수학여행을 왔다면, 이런 곳에서 기념촬영을 남겼을 것 같은 기분! 학생들에게도 배울 것이 많은 장소입니다.
돈암서원은 고정산의 기운이 이어지는 중간끝에 위치하고 있어서 배산임수 형태를 띠고 있습니다.
경치가 과거에도 좋았겠지만 현재 봐도 이렇게 아름다운 자리에 멋진 서원을 만들 생각을 했던 조상님들이 대단하다는 생각이 들었습니다.
멋진 강당과 조상님에 대한 역사 공부를 마치고 논산의 다음 여행지로 이동했습니다.
논산여행 오셨다면 교육의 핫플레이스 장소인 돈암서원은 꼭 방문하시기 바랍니다.
C:\Users\USER\Desktop\2019년\알고리즘\돈암서원.txt 파일읽기가 종료되었습니다.
탐색할 Key Word 패턴을 입력하세요.
돈암서원

원시적 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

Automata 매칭으로 찾은 위치:
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815

boyerMooreHorspool로 매칭으로 찾은 위치:
패턴이 시작되는 위치: 64 73 186 251 339 473 1154 1224 1527 1622 1815

KMP Automata 매칭으로 찾은 위치:
KMP automata: [  0  0  0  0  0 ]
패턴이 시작되는 위치:  64 73 186 251 339 473 1154 1224 1527 1622 1815
```

과제 8 Result

출력 예제

돈암서원txt :

안녕하세요.논산시 소셜미디어 서포터즈 이창헌입니다.

오늘 소개해드릴 논산의 명소는 국가지정 사적 제383호로 지정된 돈암서원입니다.

……

논산여행 오셨다면 교육의 핫플레이스 장소인 돈암서원은 꼭 방문하시기 바랍니다.

C:/Users/user/Desktop/돈암서원.txt 파일 읽기가 종료되었습니다.

탐색할 Key Word 패턴을 입력하세요.>>돈암서원

원시적 매칭으로 찾은 위치:

65, 75, 190, 256, 346, 483, 1175, 1246, 1554, 1651, 1847

Automata 매칭으로 찾은 위치:

65, 75, 190, 256, 346, 483, 1175, 1246, 1554, 1651, 1847

BoyerMooreHorspool 매칭으로 찾은 위치:

65, 75, 190, 256, 346, 483, 1175, 1246, 1554, 1651, 1847

[Term Project: 과제 9] 논산 주변 도시의 모든 쌍 최단 경로

1. 과제 3에서 확보한 논산 주변 10개 도시의 연결 그래프에 대하여, 10개 도시들을 쌍(Pair) 지워 나오는 모든 쌍 최단 경로를 찾으시오.
2. 최단 경로의 결과는,
   1) 최단 경로의 총거리
   2) 최단경로를 [출발 도시 -> 도시1 -> … -> 도착 도시] 형태로 출력
   3) 두 도시간의 최단경로의 거리가 큰 순서로 10개를 출력
3. 사용할 알고리즘을 정하고 다음 알고리즘과 비교하여 장단점을 논하시오.
   1) 다익스트라 알고리즘을 10개 도시에 번갈아 사용하기
   2) 벨만 포드 알고리즘을 10개 도시에 번갈아 사용하기
   3) Floyd-Warshall 알고리즘을 1회 사용하기
4. 구현한 프로그램의 시간복잡도를 분석하여 교재에 나오는 논리적 분석치와 같은 지를 검증하시오.
5. 2항에서 구한 10개 도시들 중 첫번째 도시에 대해, 역으로 이 도시를 출발하여 논산까지의 최단경로가 동일한지 여부를 적고 이유를 논하시오.
6. 5항의 경로로 도시를 선택하고 관광명소의 5곳씩을 방문하였을 때, 관광 내용을 대표하는 특성(키워드)을 찾아 적으시오.

출력 예제

1. 각 도시간의 최단 경로를 내림차순으로 출력

[출발 도시1 -> 도시1 -> … -> 도착 도시] : 00km

…

[출발 도시10 -> 도시1 -> … -> 도착 도시] : 00km

2. 각 경로를 역순으로 방문시 경로와 총거리 비교

[도착 도시1 -> 도시1 -> … -> 출발 도시1] : 00km

…

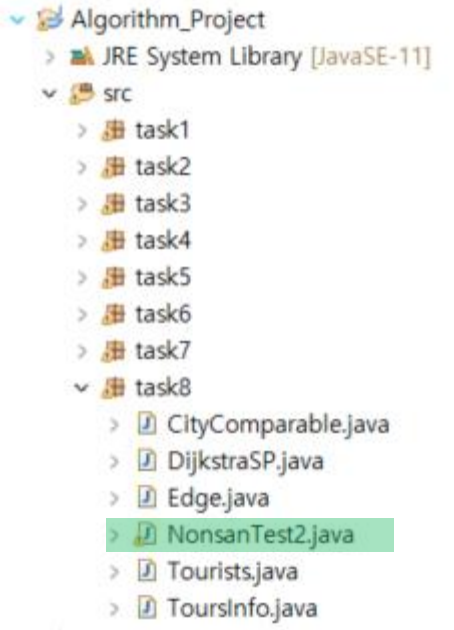
[도착 도시10 -> 도시1 -> … -> 출발 도시10] : 00km

3. 10개의 도시관광 키워드
4. 이 여행의 관광키워드는 [ ] 입니다.

과제 9

논산 주변 도시의 모든 쌍 최단 경로

Source Code



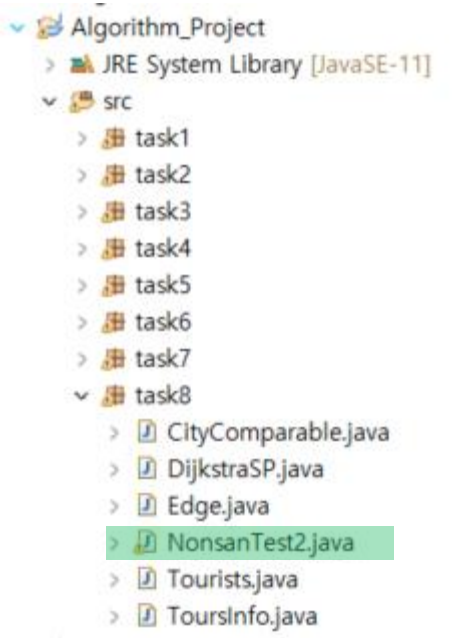

```java
public class NonsanTest2 {
    static String[] City = { "논산", "강경", "계룡", "공주", "부여", "대전", "천안", "전주", "대천", "부안" };

    public static void main(String[] args) {

        int[][] dist = {
                { 0, 9, 22, 34, 21, 67, 87, 51, 79, 80 },
                { 9, 0, 35, 0, 0, 74, 0, 0, 0, 0 },
                { 22, 35, 0, 0, 0, 22, 0, 0, 0, 0 },
                { 34, 0, 0, 0, 33, 0, 42, 0, 0, 0 },
                { 21, 0, 0, 33, 0, 0, 0, 0, 49, 0 },
                { 67, 74, 22, 0, 0, 0, 0, 0, 0, 0 },
                { 87, 0, 0, 42, 0, 0, 0, 0, 0, 0 },
                { 51, 0, 0, 0, 0, 0, 0, 0, 0, 41 },
                { 79, 0, 0, 0, 49, 0, 0, 0, 0, 0 },
                { 80, 0, 0, 0, 0, 0, 0, 41, 0, 0 }, };

        int N = dist.length;
        List<Edge>[] adjList = new List[N];
        for (int i = 0; i < N; i++) {
            adjList[i] = new LinkedList<>();
            for (int j = 0; j < N; j++) {
                if (dist[i][j] != 0) {
                    Edge e = new Edge(i, j, dist[i][j]);
                    adjList[i].add(e);
                }
            }
        }

        DijkstraSP di = new DijkstraSP(adjList);
        Stack<Integer> s = new Stack<>();
```

1. 논산 주변 10개 도시의 연결 그래프를 넣고 최단거리 구하기.

과제 9

논산 주변 도시의 모든 쌍 최단 경로

Source Code




```
DijkstraSP di = new DijkstraSP(adjList);
Stack<Integer> s = new Stack<>();

System.out.println("1.각 도시에서 도시까지 최단경로들 중 가장 큰값부터 10개를 나열하시오.");
CityComparable[] Load = new CityComparable[N];
for (int i = 0; i < N; i++) {
    int maxver = 0; // 최단경로가 가장 큰 경로
    int maxIdx = 0; // 큰경로 정점 인덱스
    int distan = 0; // 거리
    int back = 0;
    int[] distance = di.shortestPath(i);
    ArrayList<String> AS = new ArrayList<>();

    for (int j = 0; j < N; j++) { // 최단경로중 최댓값찾기
        if (maxver < distance[j]) {
            maxver = distance[j];
            distan = maxIdx = j;
        }
    }
    AS.add(City[i]);
    while (di.previous[maxIdx] != -1) {
        s.push(maxIdx);
        maxIdx = di.previous[maxIdx];
    }
    for (int st = 0; st < N; st++) { // 스택출력
        if (!s.isEmpty()) {
            AS.add(City[s.pop()]); // AS배열에 도시명을 순서대로 입력
        }
    }
    Load[i] = new CityComparable(AS, maxver); // 객체배열에 CityComparable에 도시순서와 총거리를 넣어줌
}
System.out.println("");

Arrays.sort(Load); // 내림차순으로 10개 Sort
for (int i = 0; i < N; i++) {
    Load[i].printCity();
}// 역순 인증
```

Console

```
<terminated> NonsanTest2 (1) [Java Application] C:₩Program
1.각 도시에서 도시까지 최단경로들 중 가장 큰값부터 10개를 나열하시오.

천안=>공주=>논산=>부안 총거리는 :156km
부안=>논산=>공주=>천안 총거리는 :156km
대전=>부여=>논산=>부안 총거리는 :150km
전주=>논산=>공주=>천안 총거리는 :127km
대전=>계룡=>논산=>부안 총거리는 :124km
공주=>논산=>부안 총거리는 :114km
계룡=>논산=>부안 총거리는 :102km
부여=>논산=>부안 총거리는 :101km
강경=>논산=>부안 총거리는 :89km
논산=>부안 총거리는 :80km
```

최단 경로의 결과는,

1) 최단 경로의 총거리

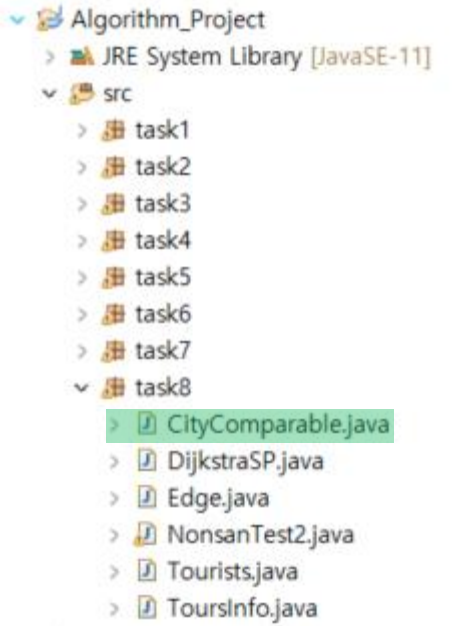
2) 최단경로 거리가 큰 순서대로

[출발 도시 -> 도시1 -> … -> 도착 도시] 형태로 출력

과제 9

논산 주변 도시의 모든 쌍 최단 경로

Source Code



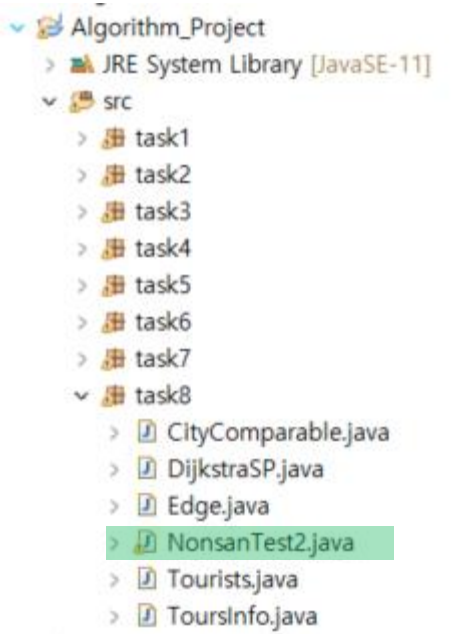

```java
package task8;

import java.util.ArrayList;

public class CityComparable implements Comparable<CityComparable> {
    ArrayList<String> City = new ArrayList<>();
    int distance;

    public CityComparable(ArrayList<String> City, int distance) {
        this.City = City;
        this.distance = distance;
    }

    @Override
    public int compareTo(CityComparable o) {
        if (this.distance < o.distance) {
            return 1;
        } else if (this.distance == o.distance) {
            return 0;
        } else {
            return -1;
        }

    }

    public void printCity() {
        System.out.print(City.get(0));
        for (int i = 1; i < City.size(); i++) {
            System.out.print("=>" + City.get(i));
        }

        System.out.println(" 총거리는 :" + this.distance + "km");
    }

    public String[] getBackVer() {
        String[] fvTv = new String[2];
        fvTv[0] = this.City.get(0);
        fvTv[1] = this.City.get(City.size() - 1);

        return fvTv;
    }

}
```

객체배열 CityComparable에
도시 순서와 총거리를 넣어
최단경로의 거리를 내림차순으로 비교

과제 9

논산 주변 도시의 모든 쌍 최단 경로

Source Code



5. 역순으로 방문시에도 경로와 거리가 동일함을 증명

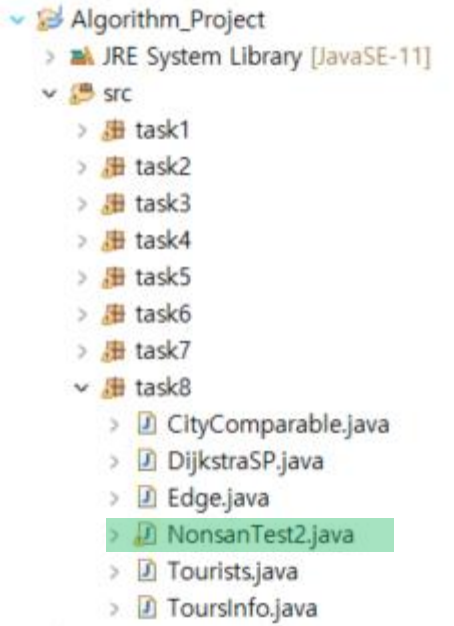
```
System.out.println("\n2.역순으로 방문시 경로와 총거리 비교\n");

for (int i = 0; i < N; i++) {
    String[] tvFv = new String[2]; // 역순으로 확인하기 위한 CityComparable의 메소드에 getBackVer의 리턴값을 받기위한 배열
    int fv = 0;
    int tv = 0;
    tvFv = Load[i].getBackVer();

    for (int j = 0; j < N; j++) {
        if (City[j] == tvFv[0]) { // fv를 구함
            fv = j;
        }
        if (City[j] == tvFv[1]) { // tv를 구함
            tv = j;
        }
    }

    int[] distance = di.shortestPath(fv);
    int back = tv;
    while (di.previous[back] != -1) {
        System.out.print(City[back] + "=>");
        back = di.previous[back];
    }
    System.out.print(City[fv] + " 총거리는 :" + distance[tv] + "km");
    System.out.println();
}
```

```
1.각 도시에서 도시까지 최단경로를 중 가장 큰값부터 10개를 나열하시오.

천안=>공주=>논산=>부안 총거리는 :156km
부안=>논산=>공주=>천안 총거리는 :156km
대전=>부여=>논산=>부안 총거리는 :150km
전주=>논산=>공주=>천안 총거리는 :127km
대전=>계룡=>논산=>부안 총거리는 :124km
공주=>논산=>부안 총거리는 :114km
계룡=>논산=>부안 총거리는 :102km
부여=>논산=>부안 총거리는 :101km
강경=>논산=>부안 총거리는 :89km
논산=>부안 총거리는 :80km

2.역순으로 방문시 경로와 총거리 비교

부안=>논산=>공주=>천안 총거리는 :156km
천안=>공주=>논산=>부안 총거리는 :156km
부안=>논산=>부여=>대전 총거리는 :150km
천안=>공주=>논산=>전주 총거리는 :127km
부안=>논산=>계룡=>대전 총거리는 :124km
부안=>논산=>공주 총거리는 :114km
부안=>논산=>계룡 총거리는 :102km
부안=>논산=>부여 총거리는 :101km
부안=>논산=>강경 총거리는 :89km
부안=>논산 총거리는 :80km
```

과제 9

논산 주변 도시의 모든 쌍 최단 경로

Source Code




```
String k1 = "역사 유적지",k2 = "자연 명승지", k3 = "휴양지", k4 = "맛집";

Tourists[] ts = new Tourists[10];

ts[0] = new Tourists("논산");
ts[1] = new Tourists("강경");
ts[2] = new Tourists("계룡");
ts[3] = new Tourists("공주");
ts[4] = new Tourists("부여");
ts[5] = new Tourists("대전");
ts[6] = new Tourists("천안");
ts[7] = new Tourists("전주");
ts[8] = new Tourists("대전");
ts[9] = new Tourists("부안");

// 논산
ts[0].tourist[0] = new ToursInfo("관촉사", k1, 82);
ts[0].tourist[1] = new ToursInfo("계백장군", k1, 80);
ts[0].tourist[2] = new ToursInfo("돈암서원", k1, 78);
ts[0].tourist[3] = new ToursInfo("탑정호", k2, 90);
ts[0].tourist[4] = new ToursInfo("대둔산", k2, 84);
ts[0].tourist[5] = new ToursInfo("KT&G 상상마당", k3, 86);
ts[0].tourist[6] = new ToursInfo("멱바위 농촌체험 휴양마을", k3, 84);
ts[0].tourist[7] = new ToursInfo("논산 선샤인랜드", k3, 80);
ts[0].tourist[8] = new ToursInfo("신동회관", k4, 84);
ts[0].tourist[9] = new ToursInfo("산애들애", k4, 96);
```

각 도시의 관광명소 목록과 평가 점수 입력

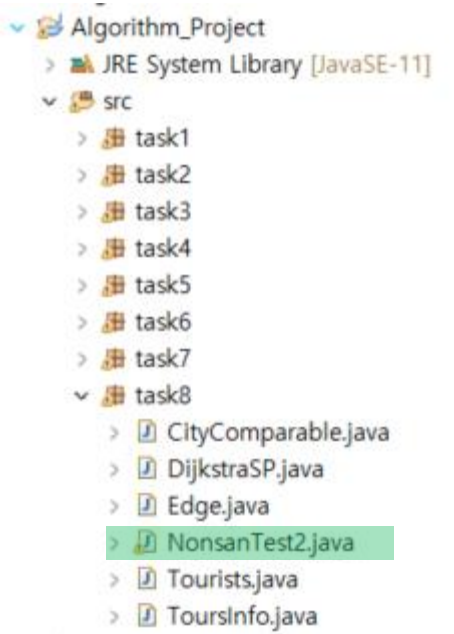
● ● ●

```
// 부안
ts[9].tourist[0] = new ToursInfo("부안서문안당산", k1, 87);
ts[9].tourist[1] = new ToursInfo("반계선생유적지", k1, 84);
ts[9].tourist[2] = new ToursInfo("선옹사 대웅전", k1, 74);
ts[9].tourist[3] = new ToursInfo("채석강", k2, 74);
ts[9].tourist[4] = new ToursInfo("적벽강", k2, 84);
ts[9].tourist[5] = new ToursInfo("곰소염전", k3, 63);
ts[9].tourist[6] = new ToursInfo("원숭이 학교", k3, 95);
ts[9].tourist[7] = new ToursInfo("부안 영상테마파크", k3, 87);
ts[9].tourist[8] = new ToursInfo("씨윈드", k4, 97);
ts[9].tourist[9] = new ToursInfo("이화자백합죽", k4, 91);
```

과제 9

논산 주변 도시의 모든 쌍 최단 경로

Source Code

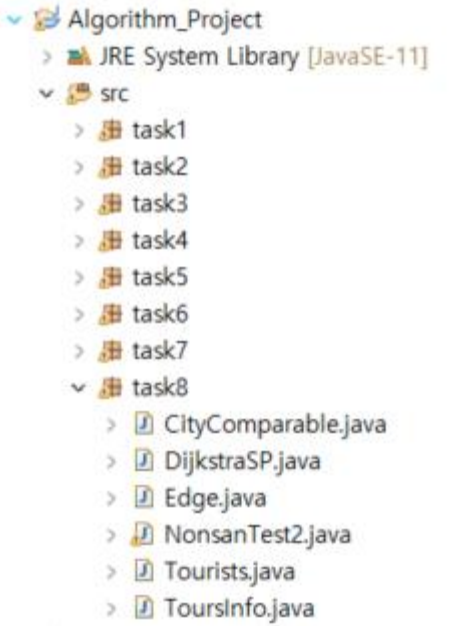



```
        String[] key = { "역사 유적지", "자연 명승지", "휴양지", "맛집" };
        int[] keypoint = new int[4];

        for (int i = 0; i < 10; i++) {

            Arrays.sort(ts[i].tourist); // 점수별로 Sorting

            for (int q = 0; q < 5; q++) { // 상위 5개의 키워드에 따라 카운트
                if (ts[i].tourist[q].keyword == k1) {
                    keypoint[0]++;
                } else if (ts[i].tourist[q].keyword == k2) {
                    keypoint[1]++;
                } else if (ts[i].tourist[q].keyword == k3) {
                    keypoint[2]++;
                } else {
                    keypoint[3]++;
                }
            }
        }

        System.out.println("\n 10개 도시의 관광키워드");
        int max = 0;
        int maxidx = 0;
        for (int i = 0; i < 4; i++) {
            if (max < keypoint[i]) {
                max = keypoint[i];
                maxidx = i;
            }
            System.out.println(key[i] + "의 키워드 개수 : " + keypoint[i] + "개");
        }

        System.out.print("\n이 여행의 관광 키워드는 " + key[maxidx] + "(" + keypoint[maxidx] + "개)입니다.");
    }
}
```

```
 10개 도시의 관광키워드
역사 유적지의 키워드 개수 : 11개
자연 명승지의 키워드 개수 : 9개
휴양지의 키워드 개수 : 17개
맛집의 키워드 개수 : 13개

이 여행의 관광 키워드는 휴양지(17개)입니다.
```

구한 경로로 도시의 관광명소 5곳씩을 방문하였을 때,
관광 내용을 대표하는 특성(키워드)을 찾는 과정.

Result



출력 예제

1. 각 도시간의 최단 경로를 내림차순으로 출력

[출발 도시1 -> 도시1 -> … -> 도착 도시] : 00km

…

[출발 도시10 -> 도시1 -> … -> 도착 도시] : 00km

2. 각 경로를 역순으로 방문시 경로와 총거리 비교

[도착 도시1 -> 도시1 -> … -> 출발 도시1] : 00km

…

[도착 도시10 -> 도시1 -> … -> 출발 도시10] : 00km

3. 10개의 도시관광 키워드

4. 이 여행의 관광키워드는 [ ] 입니다.

[Term Project: 과제 10] TSP 근사해 알고리즘의 구현

이 프로그램에서 TSP 근사해를 구하는 알고리즘으로 제시된 2개의 근사 알고리즘인
1)MST+DFS+중복회피
2)MST+최소 매칭을 이용한 오일러 사이클+중복회피 알고리즘을 JAVA 언어로 각각 구현

논산 주변 도시들의 MST는 [과제 4]에서 구한 그래프와 MST결과를 이용하시오.

논산 주변 도시들의 MST를 이 과제에서 새로 구현한 TSP 근사해 프로그램들에 입력하여,
논산을 출발하여 주변 9개 도시를 방문하고 논산으로 복귀하는 TSP 근사해의 거리와 경로를 출력.

출력 예제

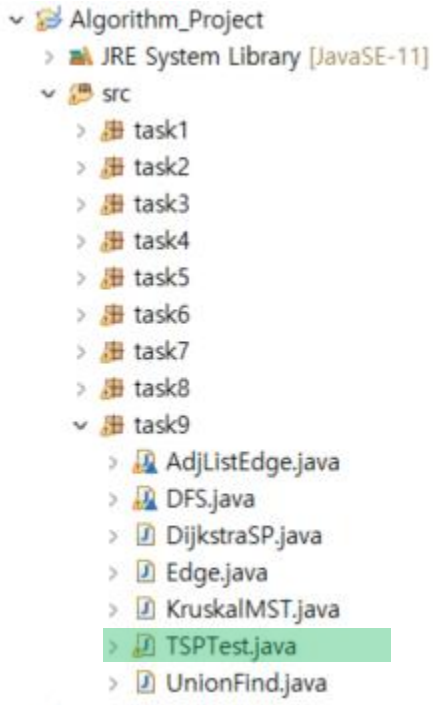
논산을 출발하여 주변 9개 도시를 방문하고 논산으로 복귀하는
TSP 근사해의 거리와 경로.

1)MST+DFS+중복회피
2)MST+최소 매칭을 이용한 오일러 사이클+중복회피

과제 10

TSP 근사해의 알고리즘 구현

Source Code



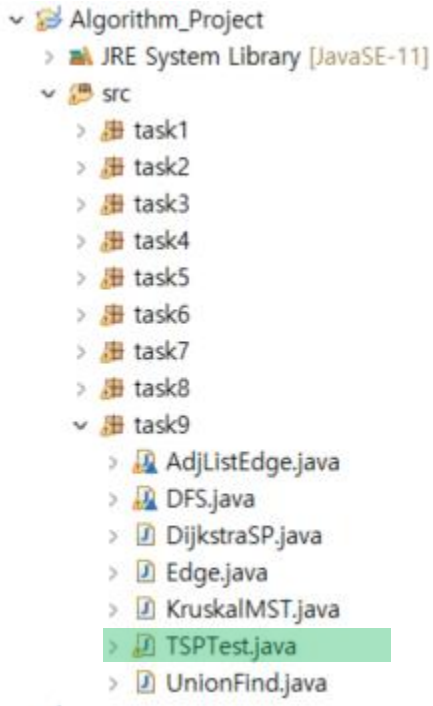

```java
public static void main(String[] args) {
    int[][] dist = {
            { 0, 9, 22, 34, 21, 43, 75, 51, 67, 80 },
            { 9, 0, 30, 0, 0, 52, 0, 0, 0, 0 },
            { 22, 30, 0, 0, 0, 22, 74, 0, 0, 0 },
            { 34, 0, 0, 0, 33, 0, 42, 0, 70, 0 },
            { 21, 0, 0, 33, 0, 0, 0, 0, 49, 79 },
            { 43, 52, 22, 0, 0, 0, 0, 0, 0, 0 },
            { 75, 0, 74, 42, 0, 0, 0, 0, 0, 0 },
            { 51, 0, 0, 0, 0, 0, 0, 0, 0, 41 },
            { 67, 0, 0, 70, 49, 0, 0, 0, 0, 0 },
            { 80, 0, 0, 0, 79, 0, 0, 41, 0, 0 }, };
    int N = dist.length;
    AdjListEdge adj = new AdjListEdge();
    List<Edge>[] adjList = new LinkedList[N];
    adjList = adj.convertMtoL(dist, N);

    KruskalMST kmst = new KruskalMST(adjList, N);
    kmst.mst();
    int[][] hang = new int[N][N]; // MST를 사용한 결과를 행렬로 치환
    for (int i = 0; i < N - 1; i++) {
        if (kmst.tree[i].weight != 0) {
            hang[kmst.tree[i].fv][kmst.tree[i].tv] = kmst.tree[i].weight;
            hang[kmst.tree[i].tv][kmst.tree[i].fv] = kmst.tree[i].weight;
        }
    }

    List<Edge>[] hangList = new LinkedList[N]; // MST행렬을 리스트로 바꿈
    AdjListEdge hL = new AdjListEdge();
    hangList = hL.convertMtoL(hang, N);

    DFS dfs = new DFS(hangList);

    int[] dfsVer = new int[11]; // dfs경로순서를 저장하는 배열
    dfsVer[0] = 0;
    dfsVer[10] = 0;
    for (int i = 1; i < N + 1; i++) { // 중복을 없앰
        dfsVer[i] = dfs.adj.pop();
    }
```

1. mst를 이용해 최소신장 트리를 만든다.
2. mst를 dfs로 방문한다.
3. 방문순서에 중복은 없애고 경로를 만든다.
4. 도착점은 시작점으로 값을 주고 순차적으로 경로를 따라간다.

과제 10

TSP 근사해의 알고리즘 구현

Source Code



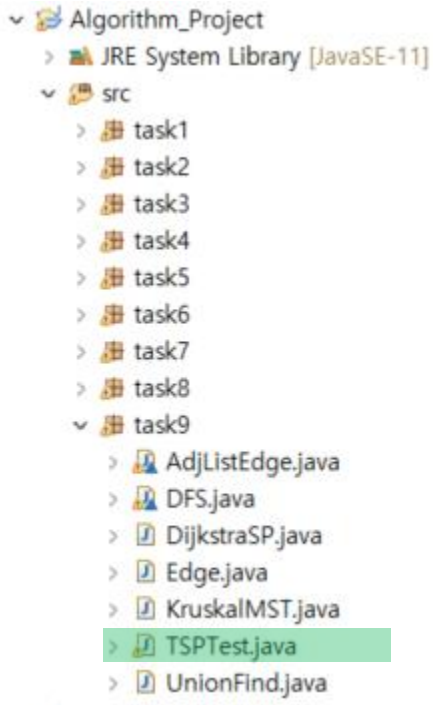

```
System.out.println("MST + DFS + 중복회피 방식의 TSP입니다.");
System.out.print("TSP :");
int SumDist = 0;
for (int i = 0; i < N; i++) { // 원래 그래프에서 TSP를 찾음
    for (int j = 0; j < adjList[dfsVer[i]].size(); j++) {
        if (adjList[dfsVer[i]].get(j).tv == dfsVer[i + 1]) {
            System.out.print("=>" + City[dfsVer[i]]);
            SumDist += adjList[dfsVer[i]].get(j).weight;
        } else if (dfsVer[i + 1] == 0) {
            System.out.print("=>" + City[0]);
        }
    }
}
System.out.print("\n TSP의 총거리의 합은 " + SumDist + "km 입니다.");
```

1. mst를 이용해 최소신장 트리를 만든다.
2. mst를 dfs로 방문한다.
3.방문순서에 중복은 없애고 경로를 만든다.
4.도착점은 시작점으로 값을 주고 순차적으로 경로를 따라간다.

과제 10

TSP 근사해의 알고리즘 구현

Source Code




```
System.out.print("\n TSP의 총거리의 합은 " + SumDist + "km 입니다.");

// 오일러 사이클 + 중복회피 방식의 TSP입니다.

DijkstraSP di = new DijkstraSP(adjList);
ArrayList<Integer> qs = new ArrayList<>();

for (int i = 0; i < N; i++) { // 연결차수가 홀수인 차수 찾기
    int j = 0;
    if (j % 2 != 0) {
        qs.add(i); // 찾은 홀수 찾수를 ArrayList에 추가
    }
}
for (int i = 0; i < qs.size(); i++) { // 홀수인 차수들의 정점들을 서로 연결하는 루프문
    int min = Integer.MAX_VALUE;
    int minIdx = 0;
    int[] distance = di.shortestPath(qs.get(i));
    for (int j = 0; j < qs.size(); j++) {
        if (distance[qs.get(j)] != 0 && distance[qs.get(j)] < min) { // 홀수 정점들 사이에 최단경로를 찾는 if문
            min = distance[qs.get(j)];
            minIdx = qs.get(j);
        }
    }
    if (dist[qs.get(i)][minIdx] != 0) { // 홀수 정점들간의 경로를 따지기 위해 원래 행렬에 그 값이 있다면 엣지를 만들고 그 엣지를 MST그래프에 추가
        Edge e = new Edge(qs.get(i), minIdx, dist[qs.get(i)][minIdx]);
        hangList[qs.get(i)].add(e);
        hangList[minIdx].add(e);
    }
}
for (int i = 0; i < N; i++) { // 윗단계에서 생긴 중복 정점을 지우는 루프문
    for (int j = 0; j < hangList[i].size() - 1; j++) {
        if (hangList[i].get(j).tv == hangList[i].get(j + 1).tv) {
            hangList[i].remove(j + 1);
        }
    }
}
DFS odfs = new DFS(hangList);
dfsVer[0] = 0;
dfsVer[10] = 0;
for (int i = 1; i < N + 1; i++) { // 찾은 MST그래프에서 중복을 없앰
    dfsVer[i] = odfs.adj.pop();
}
```

1. 입력그래프에 대해 mst를 구한다.

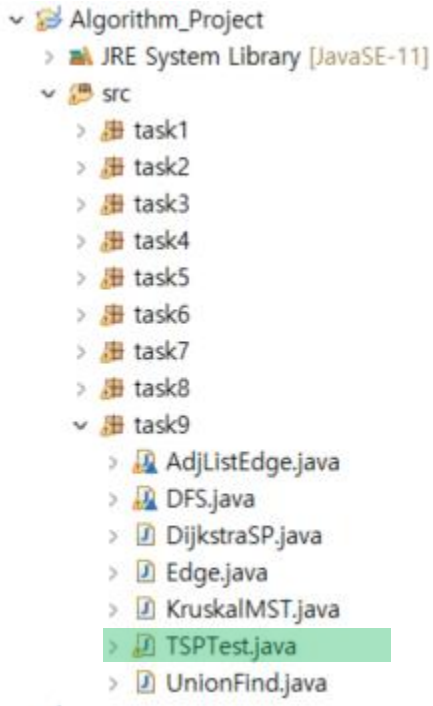
2. mst에서 차수가 홀수인 정점들을 최단경로로 연결해 준다.

3. 중복을 없애서 경로를 따라간다.

과제 10

TSP 근사해의 알고리즘 구현

Source Code



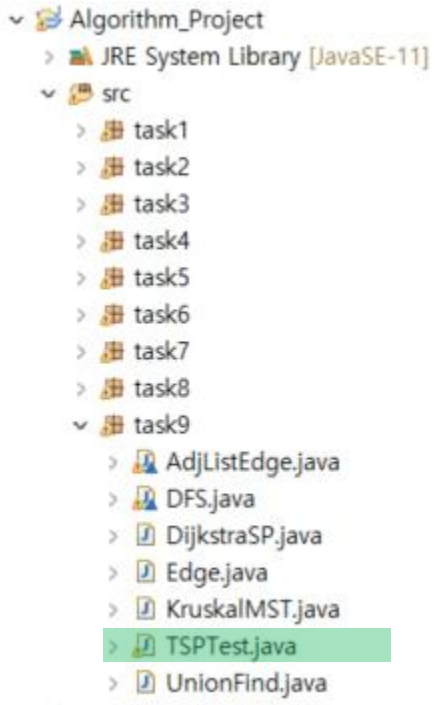

```
        System.out.println("\nMST + 오일러싸이클 + 중복회피 방식의 TSP입니다.");
        System.out.print("TSP :");
        SumDist = 0;
        for (int i = 0; i < N; i++) { // 원래 그래프에서 TSP를 찾음
            for (int j = 0; j < adjList[dfsVer[i]].size(); j++) {
                if (adjList[dfsVer[i]].get(j).tv == dfsVer[i + 1]) {
                    System.out.print("=>" + City[dfsVer[i]]);
                    SumDist += adjList[dfsVer[i]].get(j).weight;
                } else if (dfsVer[i + 1] == 0) {
                    System.out.print("=>" + City[0]);
                }
            }
        }
        System.out.print("\n TSP의 총거리의 합은 " + SumDist + "km 입니다.");

    }
}
```

1. 입력그래프에 대해 mst를 구한다.
2. mst에서 차수가 홀수인 정점들을 최단경로로 연결해 준다.
3. 중복을 없애서 경로를 따라간다.

과제 10

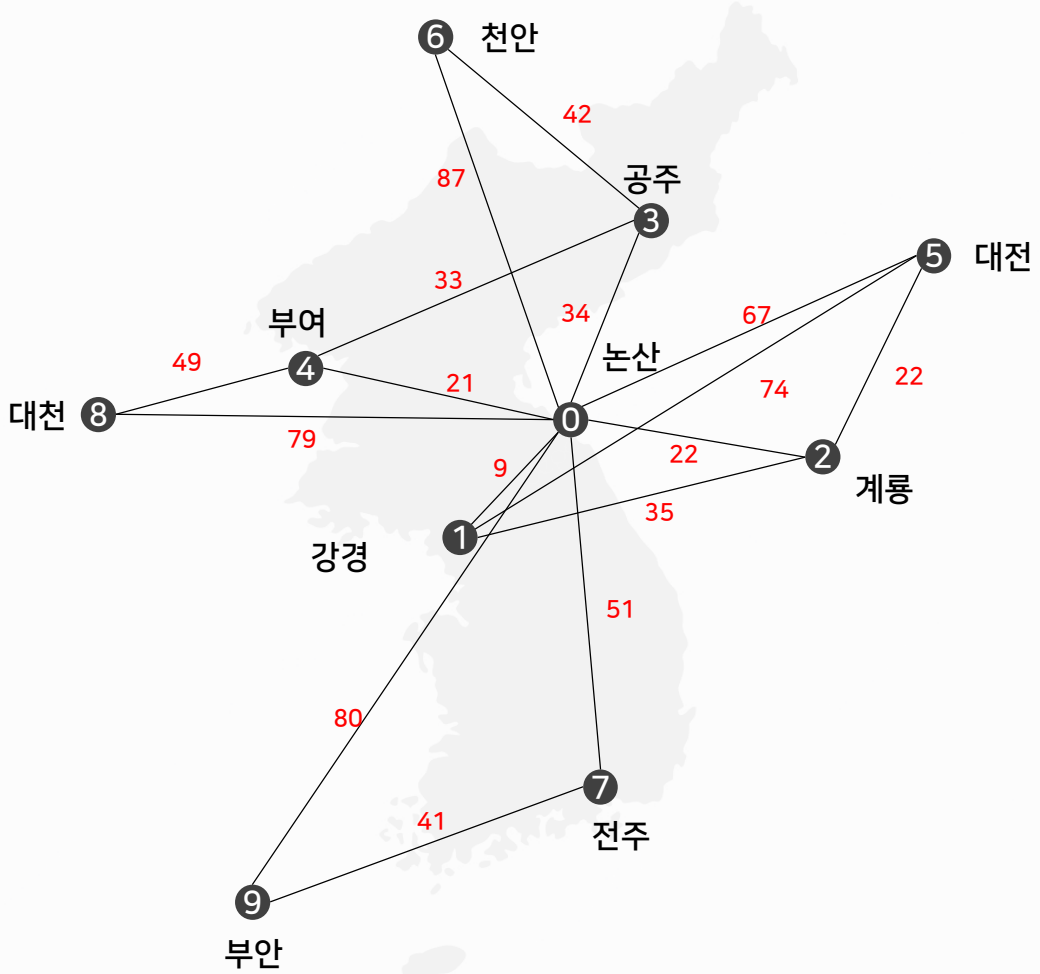
TSP 근사해의 알고리즘 구현

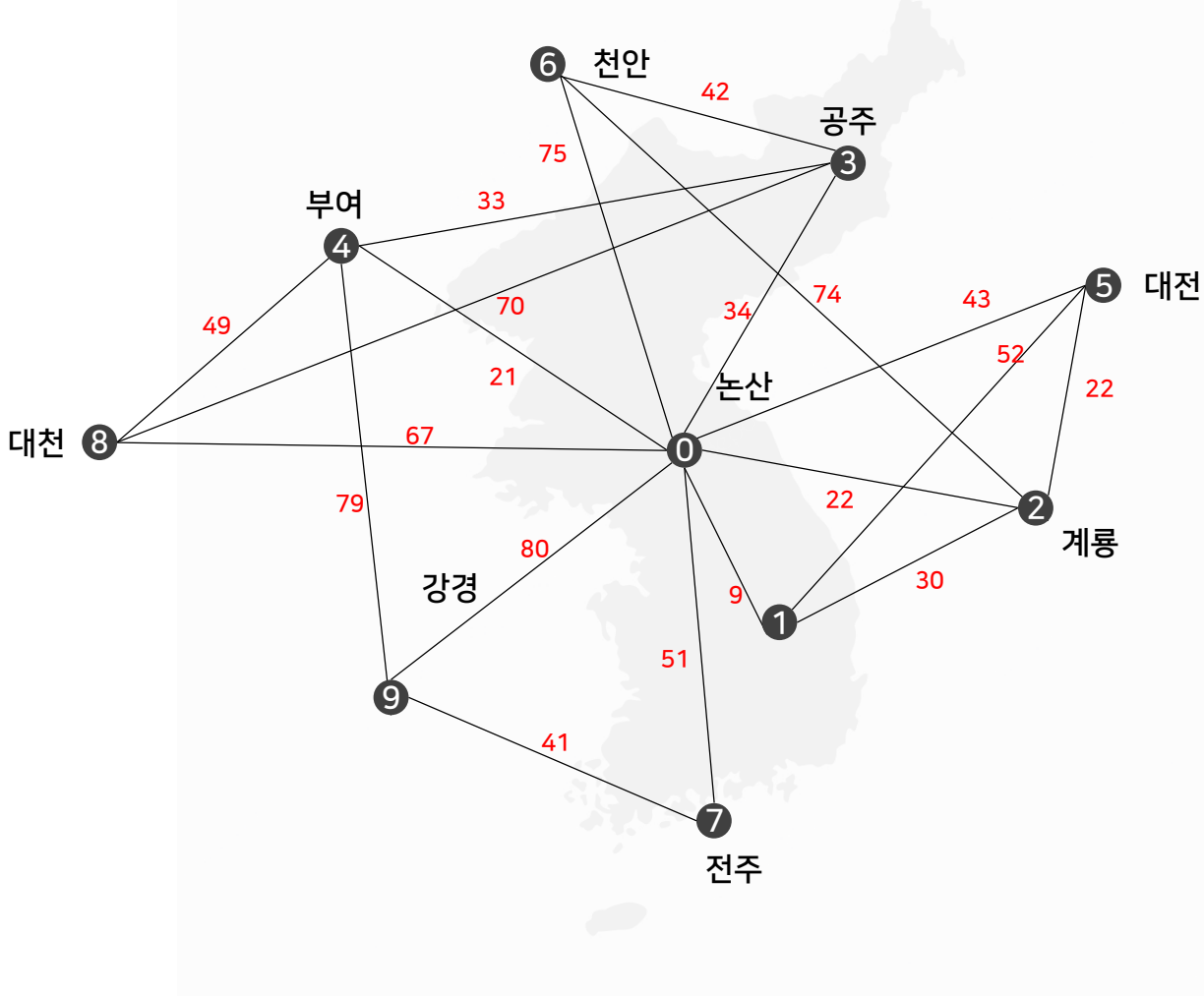
Source Code



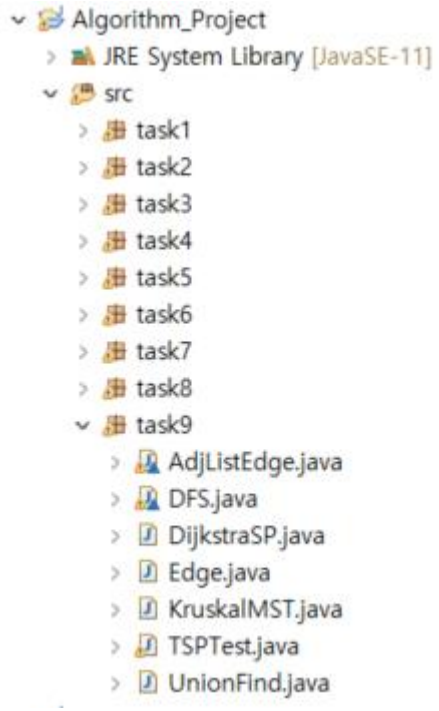

```
System.out.println("\n 1) MST + DFS + 중복회피 방식의 TSP입니다.");
System.out.print("TSP :");
int SumDist = 0;
for(int i= 0; i<N; i++) {                                    //원래 그래프에서 TSP를 찾음
    for(int j=0; j<adjList[dfsVer[i]].size(); j++) {
        if(adjList[dfsVer[i]].get(j).tv == dfsVer[i+1]) {
            System.out.print("=>"+City[dfsVer[i]]);
            SumDist += adjList[dfsVer[i]].get(j).weight;
        }else if(dfsVer[i+1] == 0){
            System.out.print("=>"+City[0]);
        }
    }
}
System.out.print("\n TSP의 총거리의 합은 "+ SumDist+"km 입니다.");
```

연결 그래프


6 천안
42
87
공주
3
33
부여
34
67
5 대전
49
4
논산
21
74
22
대천 8
79
0
9
22
2 계룡
35
1
강경
51
80
7
41
전주
9
부안


6 천안
42
공주
75
3
부여
33
4
49
70
34
74
43
5 대전
52
21
논산
22
대천 8
67
0
79
22
2
계룡
80
강경
9
30
1
51
9
41
7
전주

Result



Console

```
<terminated> TSPTest [Java Application] C:₩Program Files₩Java₩jre-10.0.2₩bin₩javaw.exe (2019. 6. 18. 오후 4:47:33)
MST + DFS + 중복회피 방식의 TSP입니다.
TSP :=>논산=>강경=>대전=>계룡=>천안=>공주=>대천=>부여=>부안=>전주=>논산
 TSP의 총거리의 합은 489km 입니다.

MST + 오일러싸이클 + 중복회피 방식의 TSP입니다.
TSP :=>논산=>강경=>대전=>계룡=>천안=>공주=>대천=>부여=>부안=>전주=>논산
 TSP의 총거리의 합은 489km 입니다.
```

출력 예제

논산을 출발하여 주변 9개 도시를 방문하고 논산으로 복귀하는
TSP 근사해의 거리와 경로.

1)MST+DFS+중복회피

2)MST+최소 매칭을 이용한 오일러 사이클+중복회피

알고리즘 구현 List

P01 - P09

- ✓ Union-Find Tree
- ✓ 10개 도시 데이터를 그래프로 표현하기
- ✓ 논산을 기점으로 다른 도시들의 최단 경로
- ✓ 관광 우선순위 정렬하기
- ✓ 관광안내계획 세우기
- ✓ 모든 도시 최단 경로 TSP
- ✓ 대전 도심의 Sky Line

P10 – P15

- ✓ 관광 안내 우선 순위
- ✓ 관광 키워드 찾기
- ✓ 논산 주변도시 모든 쌍 최단경로
- ✓ TSP 근사해 알고리즘의 구현

참고 자료

- 알기 쉬운 알고리즘, 3판, 양성봉 지음, 생능출판, 2015
- 2019 알고리즘 강의록 P01 ~ P07
- 2019 알고리즘 학습계획서 P01 ~ P07 Pseudo Code
- 근사 알고리즘 : http://ivis.kr/images/6/6f/8%EC%9E%A5%EA%B7%BC%EC%82%AC%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98.pdf
- TSP 참고자료 :

http://hochulshin.com/travelling-salesman-problem/

http://blog.naver.com/PostView.nhn?blogId=57gate&logNo=60159523081

http://ocw.snu.ac.kr/sites/default/files/NOTE/6361.pdf

https://doublezerostone.tistory.com/5

- 분할 정복 알고리즘 :

https://janghw.tistory.com/entry/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-Divide-and-Conquer-%EB%B6%84%ED%95%A0%EC%A0%95%EB%B3%B5

https://kimch3617.tistory.com/entry/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-%EB%B6%84%ED%95%A0%EC%A0%95%EB%B3%B5%EB%B2%95-Divide-and-Conquer

https://adrian0220.tistory.com/44

개인 소감

홍문기

개인과제와 텀프로젝트를 병행하면서 과제 양이 많아 힘들었던 것 같다. 하지만 수업시간에 배운 내용을 토대도 더 심화적인 과제해결을 통해 코드의 알고리즘 활용이 더욱 원활해지고 더 많은 학습이 필요하다고 생각했다.

최학준

텀프로젝트를 진행하면서 느낀점은 수업시간에 배운 것 뿐만 아니라 배우지 않았던 내용들까지 활용하면서 진행해야 했기 때문에 해결에 많은 시간이 필요했지만, 많은 시간을 투자한만큼 배운 내용들에 대해 더 심화적으로 학습할 수 있었고 한단계 성장하는데 큰 도움이 되었던 것 같다.

이재근

자료 구조도 잘 이해하고 있지 않아서 알고리즘 과목을 이해하는데 어려움이 많았다. 코드를 만드는데 시간이 많이 걸리고 소스코드의 알고리즘을 이해하기 어려웠지만 텀프로젝트를 통해 협업하고 더 많은 부분을 공부하게 되면서 이해하는 것에 큰 도움이 되었던 것 같다.

종합 소감

텀프로젝트를 진행하면서 수업시간에 배운 것 뿐만 아니라 배우지 않았던 내용들까지
활용하면서 진행해야 했기 때문에 해결에 많은 시간이 필요했지만,
많은 시간을 투자한만큼 협업하고 팀원들과 함께 배운 내용들에 대해
더 심화적으로 학습할 수 있었고 한단계 성장하는데 큰 도움이 되었던 것 같다.

팀원 역할 분담

홍문기

10개 도시 데이터를 그래프로 표현
논산 주변도시 모든 쌍 최단 경로 구하기
SkyLine 알고리즘
Kruskal2 / Prim

소스코드 통합, 수정

최학준

논산을 기점으로 다른 도시들의 최단 경로
TSP 근사해 알고리즘
관광 우선순위
관광안내계획 세우기
모든 도시 최단 경로 TSP

중간, 기말 발표자

이재근

Union-Find Tree
Kruskal2 / Prim
관광 키워드 찾기 알고리즘

자료조사, PPT