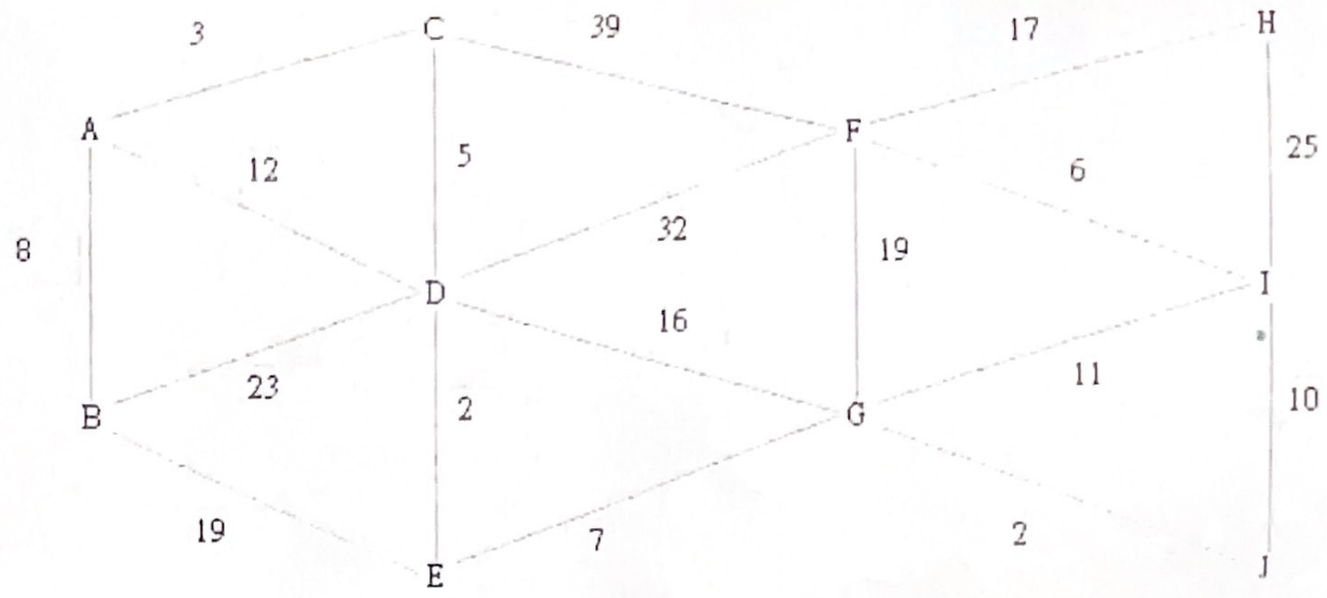
作業10有10個城市，城市間共有18條公路連通，公路上的數字為公里數，輸入起點與終點的城市，可以計算兩城市的最短距離和最短路徑




```
CLIPS> (run)
Start Vertex: B
End Vertex: H
Distance: 59  Route: (B A C D E G I F H)
```

**Dijkstra 演算法求任兩點間之最短路徑**

1) 建立每個頂點之間的最小距離矩陣（ D[i,j]代表i和j之間的最小距離）
2) 針對所有的頂點 i 和 j，每次加入一個頂點 k 來修正最小距離矩陣
   - ❖ D[i,j] = min{ D[i,j] , D[i,k]+D[k,j] }
3) 當所有的頂點都加入檢查過，則最小距離矩陣為任兩點間之最短距離

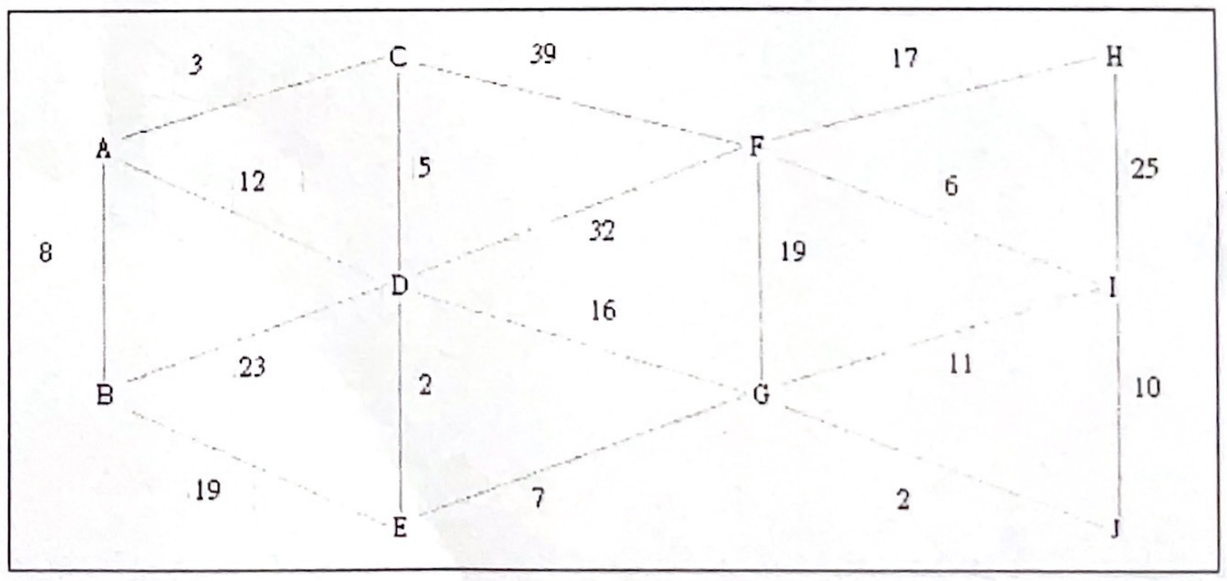
```
(defrule input-vextex
  (declare (salience 100))
  =>
  (printout t "Start Vertex: ")
  (assert (start (read)))
  (printout t "End Vertex: ")
  (assert (end (read))))
```

```
(defrule generate-path
  (start ?s)
  (end ?e)
  (shortest (v1 ?s) (v2 ?e) (distance ?d))
  =>
  (assert (path (v1 ?s) (v2 ?e) (distance ?d) (left ?d) (route ?s))))
```



```
(deftemplate edge (slot v1) (slot v2) (slot distance))
(deftemplate shortest (slot v1) (slot v2) (slot distance))
(deftemplate path (slot v1) (slot v2) (slot distance) (slot left) (multislot route))
(deffacts initial
  (vertex A)
  (vertex B)
  (vertex C)
  (vertex D)
  (vertex E)
  (vertex F)
  (vertex G)
  (vertex H)
  (vertex I)
  (vertex J)
  (edge (v1 A) (v2 B) (distance 8))
  (edge (v1 A) (v2 C) (distance 3))
  (edge (v1 A) (v2 D) (distance 12))
  (edge (v1 B) (v2 D) (distance 23))
  (edge (v1 B) (v2 E) (distance 19))
  (edge (v1 C) (v2 D) (distance 5))
  (edge (v1 C) (v2 F) (distance 39))
  (edge (v1 D) (v2 E) (distance 2))
  (edge (v1 D) (v2 F) (distance 32))
  (edge (v1 D) (v2 G) (distance 16))
  (edge (v1 E) (v2 G) (distance 7))
  (edge (v1 F) (v2 G) (distance 19))
  (edge (v1 F) (v2 H) (distance 17))
  (edge (v1 F) (v2 I) (distance 6))
  (edge (v1 G) (v2 I) (distance 11))
  (edge (v1 G) (v2 J) (distance 2))
  (edge (v1 H) (v2 I) (distance 25))
  (edge (v1 I) (v2 J) (distance 10)))
```

```
CLIPS> (run)
Start Vertex: B
End Vertex: H
Distance: 59  Route: (B A C D E G I F H)
```




請到雲端學院下載事實資料 path.txt

```
(defrule generate-anti-direction-edge
  (declare (salience 90))
  (edge (v1 ?v1) (v2 ?v2) (distance ?d))
  (not (edge (v1 ?v2) (v2 ?v1) (distance ?d)))
  =>
  (assert (edge (v1 ?v2) (v2 ?v1) (distance ?d)))
  (assert (shortest (v1 ?v1) (v2 ?v2) (distance ?d)))
  (assert (shortest (v1 ?v2) (v2 ?v1) (distance ?d))))

(defrule generate-disconnected-distance
  (declare (salience 80))
  (vertex ?v1)
  (vertex ?v2&~?v1)
  (not (edge (v1 ?v1) (v2 ?v2) (distance ?)))
  =>
  (assert (shortest (v1 ?v1) (v2 ?v2) (distance 1000))))
```