2019 봄학기 자료구조 과제3

<Kruskal's algorithm 구현해서 Minimum Cost Spanning Tree 찾기>

#2017320233 김채령

- **과제 환경**

  OS: Windows 10

  Coding environment: Visual Studio 2017 Community version

- **과제 분석**

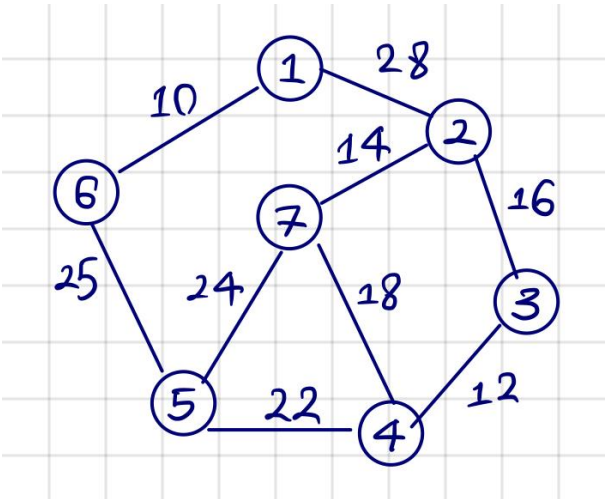
  : Kruskal's algorithm 구현하기

  - Input: undirected & weighted(cost) graph
  - Output: Minimum Cost Spanning Tree (앞으로는 MST라고 간단히 부른다.)

  ➔ **Input 구현**

  : 노드(vertices)의 개수와 cost adjacency matrix를 입력 받는데, cost adjacency matrix란 각 노드를 나타내는 행과 열에 entry로 edge의 cost(weight)를 넣는 adjacency matrix를 말하며, 두 노드 사이에 edge가 없으면 0, 있으면 cost 값을 입력 받는다.

  예를 들면, 우리 교재 한글판 p.309쪽 그림6.22의 그래프는 다음과 같이 입력을 준다.



-노드의 개수: 7

-cost adjacency matrix

	1	2	3	4	5	6	7
1	0	28	0	0	0	10	0
2	28	0	16	0	0	0	14
3	0	16	0	12	0	0	0
4	0	0	12	0	22	0	18
5	0	0	0	22	0	25	24
6	10	0	0	0	25	0	0
7	0	14	0	18	24	0	0

위의 input을 가지고 output인 MSP를 도출하기 위해서 사용할 Kruskal's algorithm의 pseudo code와 구현 계획은 다음과 같다.

Kruskal(G, W) // W = weight information

1. A = empty set // for result MST
2. For each node x in V(G)

   Make-set(x) // disjoint set
3. Sort all edges in non-decreasing order

   ⇨ 가장 cost가 적은 edge 순서로 다룸으로써 sorting을 대신한다.
4. For each edge (u, v) in this order

   If find-set(u) != find-set(v) // If two edges not in same set

   then A<-A union {(u,v)} // add to result set

   ⇨ 교재 Chapter5의 find와 union 함수를 사용한다.
5. Return A

   ⇨ MST의 edge를 바로 출력한다.

- **코드 설명**

```
13  void main()
14  {
15      printf("₩n₩t Kruskal's Algorithm으로 MST 찾기! ₩n");
16      printf("₩n₩t₩t 2017320233김채령 ₩n");
17      printf("₩n노드의 개수를 입력하세요. (최대 9 개) : ");
18      scanf("%d", &n);
19      printf("₩ncost adjacency matrix를 입력하세요.₩n(edge가 없는 노드 사이의 cost는 0으로 입력하고, cost는 998이하로 입력하세요.) :₩n");
20      for (i = 1; i <= n; i++)
21      {
22          for (j = 1; j <= n; j++)
23          {
24              scanf("%d", &cost[i][j]); //n*n adjacency matrix에 cost를 입력받는다.
25              if (cost[i][j] == 0) //cost가 없는 edge는 연결된 edge가 아니니까 매우 큰 가중치를 부여한다.
26                  cost[i][j] = 999;
27          }
28      }
```

Main 함수의 앞부분에서 노드의 개수와 cost adjacency matrix를 입력 받음으로써 input 그래프에 대한 정보를 입력 받는다. 이때 cost는 989 이하의 값을 입력 받는데, 그 이유는 두 노드 사이에 edge가 없을 때 (cost가 0) 그 두 노드의 entry에 999를 부여해서 Kruskal's algorithm을 적용시켜 edge를 고를 때 그 edge가 선택되지 않도록 하였다. 다음은 교재의 find, union 함수 구현이다.

```
69  int find(int i)
70  {
71      //printf("parent of %d: %d₩n", i, parent[i]);
72      while (parent[i]) {
73          //printf("hi₩n");
74          i = parent[i];
75      }
76      return i;
77  }
```

```
79  int uni(int i, int j)
80  {
81      if (i != j)
82      {
83          //printf("dif₩n");
84          parent[j] = i;
85          return 1;
86      }
87      return 0;
88  }
```

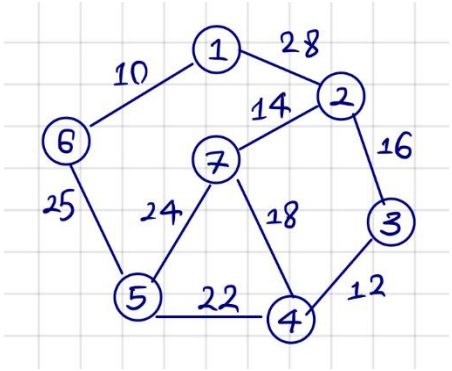
```
41      while (ne < n) //edge들이 n-1개 선택되도록
42      {
43          min = 999;
44          for (i = 1; i <= n; i++)
45          {
46              for (j = 1; j <= n; j++)
47              {
48                  if (cost[i][j] < min) //앞서 cost가 없는 edge에 부여한 매우 큰 값은 여기서 걸러진다.
49                  { //cost adjacency matrix에서 가장 적은 cost의 edge를 찾는다.
50                      min = cost[i][j];
51                      a = u = i;
52                      b = v = j;
53                  }
54              }
55          }
56          u = find(u); //printf("find result %d₩n", u);
57          v = find(v); //printf("find result %d₩n", v);
58          if (uni(u, v))
59          {// 두 개가 다른 집합의 노드들이어서 합쳐질 때
60              //printf("uni result %d₩n", uni(u, v));
61              printf("%d edge (%d,%d) =%d₩n", ne++, a, b, min);
62              mincost += min;
63          }
64          cost[a][b] = cost[b][a] = 999;
65      }
66      printf("₩n₩tMinimum cost = %d₩n", mincost);
67  }
```

Input 그래프에 대한 정보를 입력 받은 뒤에는 MST의 edge들을 고르는데, cost adjacency matrix를 탐색하면서 가장 작은 cost의 edge를 골라 MST의 edge로 추가할지를 검사한다. Find 함수를 통해 각 노드들의 부모 노드가 다르면 두 노드를 합쳐주어 한 tree로 만들고 그 edge를 MST의 edge로 선택함을 출력해주는데, 결국은 두 노드가 다른 집합(tree)에 속하는지 여부를 검사하는 것이다. 처음에 parent 배열은 다 0으로 초기화 되어 있기 때문에, edge의 두 노드를 탐색해 나가면서 parent 배열에 노드를 채우게 된다. 이 과정을 마치면 이 edge에 대해서는 다시 검사가 이뤄지지 않도록 999라는 cost가 0인 edge에 부여한 값을 부여함으로써 edge가 없어지는 효과를 준다. 이때, cost adjacency matrix는 undirected graph에서는 대칭이므로 대칭인 두 entry에 대해서 모두 999를 대입해준다. 그래서 다음 while 문을 통해서는 cost가 그 다음으로 작은 값이 골라진다. 이때 while 문은 최대 #node-1만큼 도는데, 그 이유는 이 이상 edge가 선택되면 cycle이 생겨 tree의 조건을 위배하기 때문이다. edge들을 다 검사하면 이 MST의 cost를 합산한 값을 출력함으로써 종료한다.

- **실행 예시**

  위의 input 구현에서 첨부한 것과 같은 예제를 사용해서 알고리즘이 잘 구현되어 작동함을 확인할 수 있다.

  Input: #node=7



cost adjacency matrix

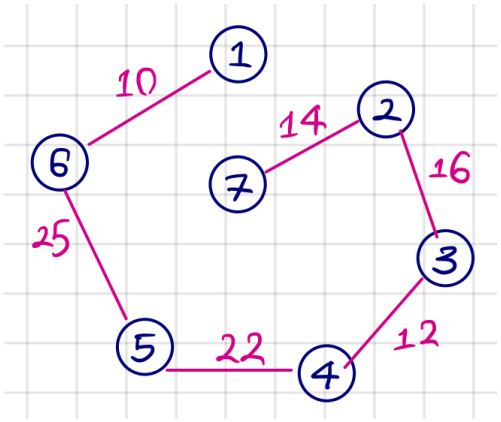
	1	2	3	4	5	6	7
1	0	28	0	0	0	10	0
2	28	0	16	0	0	0	14
3	0	16	0	12	0	0	0
4	0	0	12	0	22	0	18
5	0	0	0	22	0	25	24
6	10	0	0	0	25	0	0
7	0	14	0	18	24	0	0

실행 창)

```
C:₩Users₩chae₩source₩repos₩kruskals_alg₩Debug₩kruskals_alg.exe
        Kruskal's Algorithm으로 MST 찾기!

                2017320233김채령

노드의 개수를 입력하세요. : 7

cost adjacency matrix를 입력하세요.
(edge가 없는 노드 사이의 cost는 0으로 입력하세요.) :
0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24
10 0 0 0 25 0 0
0 14 0 18 24 0 0

MSP의 edge들은 다음과 같습니다.
1 edge (1,6) =10
2 edge (3,4) =12
3 edge (2,7) =14
4 edge (2,3) =16
5 edge (4,5) =22
6 edge (5,6) =25

        Minimum cost = 99
```

2019 봄학기 자료구조 과제3

위의 실행창에서 출력된 edge들을 연결하여 그래프로 그려보았다.



교재의 예제에서 제시한 output과 동일한 input에 대한 동일한 output이 출력됨으로써 Kruskal's algorithm이 잘 작동함을 확인할 수 있다.

-끝입니다. 고맙습니다☺