2021 November 8

자료구조2 실습 레포트

과목명 | 자료구조2 실습

담당교수 | 홍 민 교수님

학과 | 컴퓨터소프트웨어공학과

학년 | 2학년

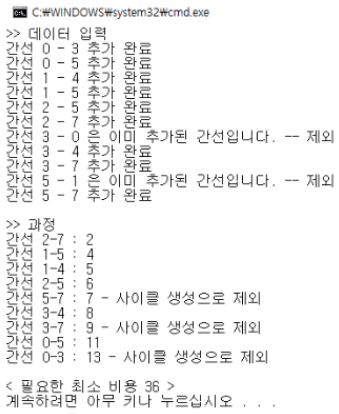
학번 | 20204059

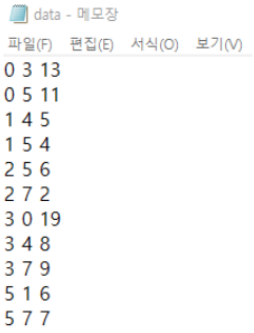
이름 | 이예빈

목 차

1. Kruskal 의 MST 알고리즘을 이용한 최소 비용 신장 트리 프로그램

- Kruskal의 MST 알고리즘
  - 405 페이지에 프로그램 11.8을 참고하여 Kruskal의 최소 비용 신장 트리 프로그램을 작성하고 아래와 같이 가장 최소 비용으로 도달할 때의 비용을 결과로 출력하시오.
    - data.txt에서 간선 및 가중치를 가져와 사용
    - 이미 입력되어있는 간선의 경우 중복되는 간선임을 출력하고 제외





1.1 문제 분석

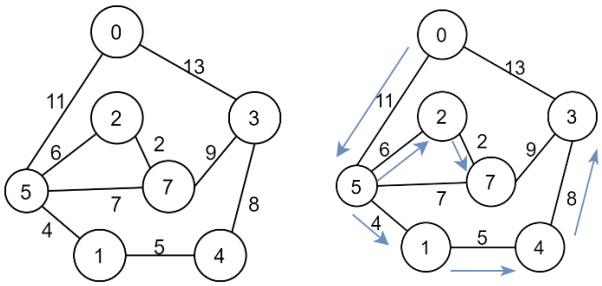
Kruskal 의 MST 알고리즘은 선택하는 그 순간 최적이라고 생각하는 것을 선택하는 탐욕적인 방법(greedy method)을 사용하여 최소 비용 신장 트리가 최소 비용의 간선으로 구성됨과 동시에 사이클을 포함하지 않는다는 조건에 근거하여, 각 단계에서 사이클을 이루지 않는 최소 비용 간선을 선택한다.

Kruskal 알고리즘은 먼저 그래프의 간선들을 가중치의 오름차순으로 정렬한다. 이후, 정렬된 간선들의 리스트에서 사이클을 형성하지 않는 간선들은 선택하기를 반복하여, 최소 비용 신장 트리의 집합에 추가한다.

Kruskal 알고리즘에서 사이클 형성 여부를 확인하기 위해서는 추가하고자 하는 간선의 양끝 정점이 같은 집합에 속하는지 검사하는 것이 중요한데, 이 검사를 위한 알고리즘을 union-find 알고리즘이라고 한다. Union-find 연산은 트리 형태 구조를 배열로 구현하여 부모 노드에 대한

인덱스를 저장하여, 동일한 집합에 속해 있는지를 검사할 수 있게 된다. 소스 코드 분석 단계를 통해 union-find 알고리즘과 kruskal 알고리즘의 과정을 자세하게 살펴보는 것으로 하자.

이제 파일의 정보를 살펴보자. 데이터 파일의 각 줄에는 두 정점, 그리고 정점 사이의 간선의 가중치 값의 정보가 나타난다. 무방향 그래프이므로, 하나의 간선을 잇는 두 정점에 대해 중복되는 데이터가 있다면, 이는 제외시키도록 하자. 따라서 추가된 간선/정점 데이터들을 모두 종합해보면, 그래프를 형성하는 정점의 개수는 총 0 부터 5, 그리고 7 까지 모두 7 개, 그리고 간선의 개수는 총 9 개이다. 그래프는 다음과 같은 그림으로 나타낼 수 있다.



그리고 그래프의 간선 데이터들(두 정점, 그리고 가중치 값)을 오름차순으로 정렬한 것을 표로 나타내면 다음과 같다.

2	1	1	2	5	3	0	0
7	5	4	5	7	4	5	3
2	4	5	6	7	8	11	13

오름차순으로 정렬된 간선들을 차례대로 확인하며, 사이클이 형성되지 않는 조건 하에서 각 간선들을 추가한다.

추가하여 선택되는 간선은 (2-7, 2), (1-5, 4), (1-4, 5), (2-5, 6) (3-4, 8), (0-5, 11) 이다. 간선 (5-7, 7)은 사이클을 형성하므로 선택되지 않게 된다. Kruskal 알고리즘은 모든 노드를 방문, 즉 (노드의 개수-1)의 값만큼 간선을 선택하면 그 이상의 간선은 살펴보지 않아도 된다. 따라서 (0-5, 11) 의 간선까지 선택을 하면, 총 6 개의 간선을 선택하게 되어 정점의 개수가 7 보다 하나 적어져서 알고리즘이 종료되는 것이다.

하지만 이번 예시 실행창에서는 (0-3, 13)의 간선까지 모두 탐색을 한 후, 알고리즘이 종료되기 때문에 이에 대해서는 "1.5 추가 소스 코드 구현"에서 다루어 보았다.

1.2 소스 코드

```c
/*
	작성자: 이예빈(20204059)
	작성일: 2021.11.07
	프로그램명: Kruskal의 MST 알고리즘을 이용한 최소 비용 신장 트리 프로그램
*/

#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 30
#define INF 1000

int parent[MAX_VERTICES]; // 부모 노드

// 초기화
void set_init(int n) {
	int i;
	for (i = 0;i < n;i++)
		parent[i] = -1;
}

// curr가 속하는 집합을 반환
int set_find(int curr) {
	if (parent[curr] == -1)
		return curr; // 루트 (자기 자신 인덱스)
	while (parent[curr] != -1) curr = parent[curr]; // 루트 인덱스
	return curr;
}

// 두개의 원소가 속한 집합을 합친다
void set_union(int a, int b) {
	int root1, root2;
	root1 = set_find(a); // 노드 a의 루트 인덱스
	root2 = set_find(b); // 노드 b의 루트 인덱스
	if (root1 != root2)
		parent[root1] = root2; // 노드b의 집합이 노드a의 집합 위에 위치하게 됨
}

// 간선을 나타내는 구조체
struct Edge {int start, end, weight;};
typedef struct GraphType { // 그래프 구조체
	int n; // 간선의 개수
	struct Edge* edges; // 간선 구조체 포인터
}GraphType;

// 그래프 초기화
void graph_init(GraphType* g, int num) {
	int i;
	g->n = 0;
	g->edges = (struct Edge*)malloc(sizeof(struct Edge) * num);
	for (i = 0;i < num;i++) {
		g->edges[i].start = 0;
		g->edges[i].end = 0;
		g->edges[i].weight = INF;
	}
}

```

```
60  //·간선·여부·확인
61  int·if_edge(GraphType*·g,·int·start,·int·end)·{
62  →   int·i;
63  →   for·(i·=·0;i·<·g->n;i++)·{
64  →   →   if·(g->edges[i].start·==·start·&&·g->edges[i].end·==·end)
65  →   →   →   return·TRUE;
66  →   →   if·(g->edges[i].start·==·end·&&·g->edges[i].end·==·start)
67  →   →   →   return·TRUE;
68  →   }
69  →   return·FALSE;
70  }
71
72  //·간선·삽입·연산
73  void·insert_edge(GraphType*·g,·int·start,·int·end,·int·w)·{
74  →   if·(if_edge(g,·start,·end)·==·TRUE)·{
75  →   →   printf("간선·%d·-·%d·은·이미·추가된·간선입니다.·--·제외·\n",·start,·end);
76  →   →   return;
77  →   }
78  →   g->edges[g->n].start·=·start;
79  →   g->edges[g->n].end·=·end;
80  →   g->edges[g->n].weight·=·w;
81  →   g->n++;
82  →   printf("간선·%d·-·%d·추가·완료·\n",·start,·end);
83  }
84
85  //·qsort()에·사용되는·비교·함수
86  int·compare(const·void*·a,·const·void*·b)·{
87  →   struct·Edge*·x·=·(struct·Edge*)a;
88  →   struct·Edge*·y·=·(struct·Edge*)b;
89  →   return·(x->weight·-·y->weight);
90  }
92  //·kruskal의·최소·비용·신상·트리·프로그램
93  void·kruskal(GraphType*·g)·{
94  →   int·i;
95  →   int·uset,·vset;→→   →   //·정점·u와·정점·v의·집합·번호
96  →   int·edge_accepted·=·0;→//·현재까지선택된·간선의·수
97  →   int·sum_of_weight·=·0;·//·선택되는·간선의·비용을·저장
98  →   struct·Edge·e;
99
100 →   set_init(g->n);→→   //·집합·초기화
101 →   qsort(g->edges,·g->n,·sizeof(struct·Edge),·compare);
102
103 →   i·=·0;
104 →   while·(edge_accepted·<·(g->n·-·1))·{→   //·종료·조건1.·간선의·수·<·(n-1)
105 →   →   e·=·g->edges[i];
106 →   →   if·(e.weight·==·INF)→   //·종료·조건2
107 →   →   →   break;
108 →   →   uset·=·set_find(e.start);→   //·정점·u의·집합·번호
109 →   →   vset·=·set_find(e.end);→→   //·정점·v의·집합·번호
110
111 →   →   if·(uset·==·vset)·//·동일한·집합이라면·사이클·생성
112 →   →   →   printf("간선·%d-%d·:·%d·-·사이클·생성으로·제외·\n",·e.start,·e.end,·e.weight);
113 →   →   else·if·(uset·!=·vset)·{→   //·서로·속한·집합이·다르면
114 →   →   →   printf("간선·%d-%d·:·%d·\n",·e.start,·e.end,·e.weight);
115 →   →   →   edge_accepted++;→   //·간선·선택
116 →   →   →   set_union(uset,·vset);·//·두·개의·집합을·합친다
117 →   →   →   sum_of_weight·+=·e.weight;
118 →   →   }
119 →   →   i++;
120 →   }
121 →   printf("\n<·필요한·최소·비용·%d·>·\n",·sum_of_weight);
122 }
123
```

```
125  int main(void) {
126      GraphType* g;
127      FILE* fp;
128      int i, u, v, w; // 정점 u, v, 가중치 w
129      int numOfEdges = 0; // 간선 개수
130
131      g = (GraphType*)malloc(sizeof(GraphType));
132
133      fp = fopen("data.txt", "rt");
134      if (fp == NULL) {
135          fprintf(stderr, "파일 열기 실패\n");
136          exit(1);
137      }
138
139      printf(">> 데이터 입력\n");
140
141      while (!feof(fp)) {
142          fscanf(fp, "%d %d %d", &u, &v, &w);
143          numOfEdges++;
144      }
145      graph_init(g, numOfEdges); // 그래프 초기화
146      rewind(fp);
147
148      // 파일 다시 읽으며 간선 삽입
149      for (i = 0;i < numOfEdges;i++) {
150          fscanf(fp, "%d %d %d", &u, &v, &w);
151          insert_edge(g, u, v, w);
152      }
153
154      printf("\n\n>> 과정\n");
155      kruskal(g);
156
157      free(g->edges);// 간선 구조체 메모리 해제
158      free(g);          // 그래프 해제
159
160      fclose(fp);
161      return 0;
162  }
```

1.3 소스 코드 분석

```
/*
    작성자: 이예빈(20204059)
    작성일: 2021.11.07
    프로그램명: Kruskal의 MST 알고리즘을 이용한 최소 비용 신장 트리 프로그램
*/

#include <stdio.h>
#include <stdlib.h>
```

1. 주석에 작성자, 작성일, 프로그램명을 적는다. 이 프로그램은 Kruskal의 MST 알고리즘을 이용하여 최소 비용 신장 트리를 만드는 프로그램이다. 이후 필요한 헤더 파일들을 추가한다.

```
#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 30
#define INF 1000

int parent[MAX_VERTICES];    //부모 노드
```

2. 프로그램에서 필요한 기호 상수들 몇 가지를 정의해준다. 1과 0의 TRUE와 FALSE를 정의하고, 미리 전역으로 선언할 배열들의 크기를 임의로 지정하기 위해, 30이라는 적당히 큰 값인 30으로 MAX_VERTICES를 정의한다. 정점들의 최대 개수를 의미하게 된다. 그리고 정점과 정점 사이에 간선이 존재하지 않는 경우의 가중치 값을 임의의 1000의 값으로 INF를 정의한다.

3. (트리 형식의) 부모 노드의 인덱스를 저장하는 parent 배열을 선언한다.

```
//초기화
void set_init(int n) {
    int i;
    for (i = 0;i < n;i++)
        parent[i] = -1;
}
```

4. 위의 parent 배열의 값을 모두 -1로 초기화하는 set_init 함수이다. 초기에는 부모 노드가 없는 단일 노드들이므로 모두 -1로 초기화하고, 어떠한 노드 집합에 속하게 된다면 해당 노드의 부모 노드 인덱스 값으로 배열 값이 변경이 된다.

```
//curr가 속하는 집합을 반환
int set_find(int curr) {
    if (parent[curr] == -1)
        return curr;    //루트 (자기 자신 인덱스)
    while (parent[curr] != -1) curr = parent[curr];//루트 인덱스
    return curr;
}
```

6. 정수형 매개변수 curr에 대하여, parent 배열에서 curr가 속하는 집합을 반환하는 set_find 함수

이다. 만약 자기 자신이 루트인 경우라면, 그대로 curr을 리턴하고, 자신이 속한 집합이 있는 경우에는 해당 집합의 루트 인덱스를 리턴한다.

```
//·두개의·원소가·속한·집합을·합친다
void·set_union(int·a,·int·b)·{
→   int·root1,·root2;
→   root1·=·set_find(a);·//·노드·a의·루트·인덱스
→   root2·=·set_find(b);·//·노드·b의·루트·인덱스
→   if·(root1·!=·root2)
→   →   parent[root1]·=·root2;·→·//·노드b의·집합이·노드a의·집합·위에·위치하게·됨
}
```

7. 두 노드가 속한 두 개의 집합을 합치는 set_union 함수이다. 매개 변수로 전달받은 정수 a와 b에 대하여, a 정점이 속한 집합의 루트와 b 정점이 속한 집합의 루트를 각각 root1과 root2에 대입한다. 그리고 두 집합이 서로 같지 않은 경우, root1의 인덱스에 해당하는 parent값에 root2의 값을 저장한다. 따라서 노드b의 집합 아래에 노드 a의 집합이 위치하게 된다.

```
//·간선을·나타내는·구조체
struct·Edge·{int·start,·end,·weight;};
typedef·struct·GraphType·{·→·//·그래프·구조체
→   int·n;·→·//·간선의·개수
→   struct·Edge*·edges;·//·간선·구조체·포인터
}GraphType;
```

8. 간선을 나타내는 구조체 Edge와 그래프 구조체 GraphType을 정의한다. Edge 구조체는 두 정점 start와 end, 그리고 간선의 가중치 weight을 멤버로 갖는다. GraphType은 간선의 개수를 의미하는 정수 n, 그리고 간선의 집합을 가리키는 edges 구조체 포인터를 멤버로 갖는다.

```
//·그래프·초기화
void·graph_init(GraphType*·g,·int·num)·{
→   int·i;
→   g->n·=·0;
→   g->edges·=·(struct·Edge*)malloc(sizeof(struct·Edge)·*·num);
→   for·(i·=·0;i·<·num;i++)·{
→   →   g->edges[i].start·=·0;
→   →   g->edges[i].end·=·0;
→   →   g->edges[i].weight·=·INF;
→   }
}
```

9. 그래프 포인터 g와 정수 num을 전달받아 그래프를 동적 할당하고 초기화하는 함수이다. 포인터 g가 가리키는 n 멤버에는 우선 0을 대입한다. 그리고 edges 포인터에는 num의 개수만큼 Edge 구조체 공간을 할당한다.

10. 그리고 각각의 edges 구조체 배열을 초기화한다. start와 end에는 0의 값을, 그리고 간선의 가중치를 의미하는 weight 멤버의 값에는 INF의 초깃값을 대입한다.

```
//·간선·여부·확인
int·if_edge(GraphType*·g,·int·start,·int·end)·{
→   int·i;
→   for·(i·=·0;i·<·g->n;i++)·{
→   →   if·(g->edges[i].start·==·start·&&·g->edges[i].end·==·end)
→   →   →   return·TRUE;
→   →   if·(g->edges[i].start·==·end·&&·g->edges[i].end·==·start)
→   →   →   return·TRUE;
→   }
→   return·FALSE;
}
```

11. 간선 여부를 확인하는 if_edge 함수이다. 중복되는 간선이 새로 추가되는 것을 방지하기 위해 구현된 함수이다. 이 프로그램에서는 무방향 그래프로 다루어지므로, 방향을 고려하지 않는다. 따라서 새로 추가되는 간선의 두 정점이 정점의 순서와 상관없이 이미 그래프에 존재하는 경우라면 TRUE를 반환하고, 아니라면 FALSE를 반환한다.

```
//·간선·삽입·연산
void·insert_edge(GraphType*·g,·int·start,·int·end,·int·w)·{
→   if·(if_edge(g,·start,·end)·==·TRUE)·{
→   →   printf("간선·%d·-·%d·은·이미·추가된·간선입니다.·---·제외·\n",·start,·end);
→   →   return;
→   }
→   g->edges[g->n].start·=·start;
→   g->edges[g->n].end·=·end;
→   g->edges[g->n].weight·=·w;
→   g->n++;
→   printf("간선·%d·-·%d·추가·완료·\n",·start,·end);
}
```

12. 그래프 포인터 g와 두 정점 start와 end, 그리고 간선의 가중치 w를 매개 변수로 전달아 새로 간선을 추가하는 insert_edge함수이다. if_edge를 호출하여 TRUE인 경우라면, 이미 추가된 간선이므로 제외를 한다.

13. if_edge가 TRUE가 아닌 경우, 간선을 그래프 g의 edges 배열에 추가하도록 한다. 추가가 완료된 후, g의 n의 값을 1 증가시킨다.

```
//·qsort()에·사용되는·비교·함수
int·compare(const·void*·a,·const·void*·b)·{
→   struct·Edge*·x·=·(struct·Edge*)a;
→   struct·Edge*·y·=·(struct·Edge*)b;
→   return·(x->weight·-·y->weight);
}
```

14. c언어의 내장 함수 중 퀵 정렬을 하는 qsort 함수에 네번째 인자로 사용되는 네번째 인자로 사용되는 compare 함수이다. 두 간선의 가중치 값을 비교한 값을 반환하는 함수이다.

다음으로 kruskal의 MST 알고리즘을 이용해 최소 비용 신장 트리를 구성해나가는 kruskal함수를 살펴보자.

```
//·kruskal의·최소·비용·신상·트리·프로그램
void·kruskal(GraphType*·g)·{
→   int·i;
→   int·uset,·vset;→→   →   //·정점·u와·정점·v의·집합·번호
→   int·edge_accepted·=·0;→//·현재까지선택된·간선의·수
→   int·sum_of_weight·=·0;·//·선택되는·간선의·비용을·저장
→   struct·Edge·e;
```

15. 제어 변수 i, 정점 u와 정점 v가 속한 집합 번호(루트의 인덱스)를 저장하는 uset과 vset, 그리고 현재까지 선택된 간선의 수를 저장하는 edge_accpeted, 선택되는 간선의 비용을 저장하는 sum_of_weight를 0으로 초기화한다. 그리고 Edge 구조체 변수 e를 선언한다.

```
→   set_init(g->n);→→   //·집합·초기화
→   qsort(g->edges,·g->n,·sizeof(struct·Edge),·compare);
```

16. 우선 그래프의 정점 개수인 g->n을 인수로 전달하여 set_init을 호출하고, 정점 개수만큼 parent배열의 값을 -1로 초기화한다. 이후 stdlib 헤더 파일에 내장되어 있는 퀵 정렬 함수 qsort를 호출하여 그래프의 edges를 가중치 weight의 순서에 따라 오름차순으로 정렬한다.

```
→   i·=·0;
→   while·(edge_accepted·<·(g->n·-·1))·{→   //·종료·조건1.·간선의·수·<·(n-1)
→   →   e·=·g->edges[i];
→   →   if·(e.weight·==·INF)→   //·종료·조건2
→   →   →   break;
→   →   uset·=·set_find(e.start);→  //·정점·u의·집합·번호
→   →   vset·=·set_find(e.end);→→   //·정점·v의·집합·번호
```

17. 제어 변수 i를 0으로 초기화한다. while 반복문은 0으로 초기화되었던 edge_accepted의 값이 g->n보다 1 작은 값인 경우 종료하게 된다. Edge 구조체 변수 e에는 edges 배열의 간선 값을 저장하고, 만약 해당 간선 값의 가중치인 weight 값이 INF인 경우에는 모든 간선을 모두 탐색했다는 뜻이므로, 반복문의 실행을 종료하도록 한다.

18. 정점 u와 정점 v의 집합 번호를 각각 uset과 vset에 저장한다.

```
→   →   if·(uset·==·vset)·//·동일한·집합이라면·사이클·생성
→   →   →   printf("간선·%d-%d·:·%d·-·사이클·생성으로·제외·\n",·e.start,·e.end,·e.weight);
```

19. 만약 두 간선의 집합 번호 uset과 vset이 동일하다면, 동일한 루트를 가지며 사이클을 생성하게 되므로, 해당 간선은 제외하도록 한다.

```
    else if (uset != vset) {   //서로 속한 집합이 나르면
        printf("간선 %d-%d : %d \n", e.start, e.end, e.weight);
        edge_accepted++;   //간선 선택
        set_union(uset, vset);//두 개의 집합을 합친다
        sum_of_weight += e.weight;
    }
    i++;
  }
  printf("\n<필요한 최소 비용 %d>\n", sum_of_weight);
}
```

20. 두 간선이 서로 속한 집합이 다르다면, 해당 간선이 선택되며 두 집합을 합치게 된다. 첫번째 정점 u의 집합이 두번째 정점 v의 집합 아래에 위치하게 된다. 해당 간선의 가중치를 sum_of_weight에 더하여 해당 값을 갱신하도록 한다.

21. i가 증가하며 위의 과정을 반복하여 실행한다.

22. 마지막으로 방문했던 정점까지의 모든 간선의 가중치의 합이 저장되어 있는 sum_of_weight을 출력한다. 필요한 최소 비용을 구하게 된 것이다.

다음으로 main 함수의 동작을 살펴보자.

```
int main(void) {
  GraphType* g;
  FILE* fp;
  int i, u, v, w;//정점 u, v, 가중치 w
  int numOfEdges = 0;//간선 개수
```

23. GraphType 그래프 포인터 g, 파일 포인터 fp, 그리고 제어 변수 i와 두 정점 u, v, 그리고 가중치의 값 w, 그리고 간선의 개수 numOfEdges를 0으로 초기화하여 선언한다.

```
  g = (GraphType*)malloc(sizeof(GraphType));

  fp = fopen("data.txt", "rt");
  if (fp == NULL) {
    fprintf(stderr, "파일 열기 실패\n");
    exit(1);
  }

  printf(">> 데이터 입력\n");

  while (!feof(fp)) {
    fscanf(fp, "%d %d %d", &u, &v, &w);
    numOfEdges++;
  }
```

24. 포인터 g에 GraphType의 크기만큼 메모리를 동적 할당한다.

24. "data.txt" 파일을 읽기 모드로 연다. 우선은 파일을 읽으며, 먼저 그래프의 간선 edges포인터에 배열을 할당하기 위해 간선의 개수 numOfEdges를 구한다.

```
    graph_init(g, numOfEdges); //·그래프·초기화
    rewind(fp);

    //·파일·다시·읽으며·간선·삽입
    for·(i·=·0;i·<·numOfEdges;i++)·{
        fscanf(fp,·"%d·%d·%d",·&u,·&v,·&w);
        insert_edge(g,·u,·v,·w);
    }

    printf("\n\n>>·과정·\n");
    kruskal(g);
```

25. 포인터 g와 처음 파일을 읽으며 구한 numOfEdges를 인수로 전달하여 graph_init 함수를 호출하여 그래프를 초기화하고 필요한 간선 edges 배열을 동적 할당 한다.

26. rewind 함수를 호출하고, 파일을 다시 읽으며 읽은 데이터를 바탕으로 간선을 삽입한다. 두 정점 u와 v, 그리고 가중치 값인 w를 읽어 insert_edge 함수에 인수로 전달하여 각 간선을 배열에 추가한다.

27. kruskal 함수를 호출하여 최소 비용 신장 트리를 구성하도록 한다.

```
    free(g->edges);//·간선·구조체·메모리·해제
    free(g);        //·그래프·해제

    fclose(fp);
    return·0;
}
```

28. 마지막으로 간선 구조체의 메모리 그리고 그래프를 해제하고, 파일을 닫아 전체 프로그램 실행을 종료한다.

1.4 실행 결과

data.txt - Window

파일(F) 편집(E) 서식

```
0 3 13
0 5 11
1 4 5
1 5 4
2 5 6
2 7 2
3 0 19
3 4 8
3 7 9
5 1 6
5 7 7
```

Microsoft Visual Studio 디버그 콘솔

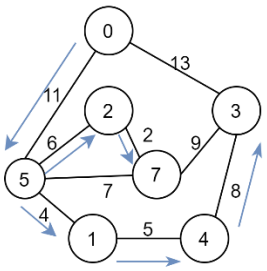
```
>> 데이터 입력
간선 0 - 3 추가 완료
간선 0 - 5 추가 완료
간선 1 - 4 추가 완료
간선 1 - 5 추가 완료
간선 2 - 5 추가 완료
간선 2 - 7 추가 완료
간선 3 - 0 은 이미 추가된 간선입니다. -- 제외
간선 3 - 4 추가 완료
간선 3 - 7 추가 완료
간선 5 - 1 은 이미 추가된 간선입니다. -- 제외
간선 5 - 7 추가 완료


>> 과정
간선 2-7 : 2
간선 1-5 : 4
간선 1-4 : 5
간선 2-5 : 6
간선 5-7 : 7 - 사이클 생성으로 제외
간선 3-4 : 8
간선 3-7 : 9 - 사이클 생성으로 제외
간선 0-5 : 11
간선 0-3 : 13 - 사이클 생성으로 제외

< 필요한 최소 비용 36 >

C:₩Users₩이예빈₩Desktop₩CSE 2021-2₩자료구조2₩code files₩
실습₩20204059-이예빈-실습9주차₩20204059-이예빈-실습9주차.
```

1.5 소스 코드 추가 구현

교재 406 페이지에 나와있는 kruskal 알고리즘 코드를 그대로 사용하며 이상하다고 생각하는 부분이 있어 추가적으로 소스 코드를 구현해 보았다. (예시 실행창과 동일하게 만들기 위해 과제로 제출한 코드는 다시 "2.2 소스 코드"의 코드로 복귀시켜 놓았다.)

교재의 코드에서 GraphType 의 멤버 n 은 간선의 개수를 의미한다. 따라서 insert_edge 함수가 호출되어 파일에서 읽은 간선이 추가될 때마다, 0 의 초깃값을 가졌던 g->n 은 1 씩 증가하게 된다. 하지만, 실제 kruskal MST 알고리즘에서 종료 조건은 선택된 간선의 수가 (정점 개수 – 1)개인 경우이다. 이 프로그램에서 정점의 개수는 총 7 개이고, 간선의 수가 총 6 개 선택되어 모든 7 개의 정점을 방문하게 되면, 더 이상의 간선이 선택될 필요가 없으므로 반복문은 종료되어야 한다. 그러나 코드에서는 g->n 의 값이 "정점의 개수"가 아닌 "간선의 개수"로 취급하기 때문에 더 많은 수의 간선을 확인하게 된다.

다음 그림과 실행창을 살펴보도록 하자. 간선 (0-5, 11)이 선택되면, 총 6 개의 간선이 선택되었고, 이는 정점의 개수 7 개보다 1 작은 값이므로 종료 조건에 의해 더 이상의 간선을 확인하지 않아도 된다. 그러나 원 코드에서는 g->n 이 정점 개수가 아닌 간선 개수로 취급되었기 때문에 간선 개수 8 보다 1 작은 7 의 값, 즉 7 개의 간선까지 모두 확인하여, 마지막 (0-3, 13) 간선까지 확인을 하게 된다는 의미이다. 이를 해결하기 위해 아래 코드를 일부 수정해보았다.




```
>> 과정
간선 2-7 : 2
간선 1-5 : 4
간선 1-4 : 5
간선 2-5 : 6
간선 5-7 : 7 - 사이클 생성으로 제외
간선 3-4 : 8
간선 3-7 : 9 - 사이클 생성으로 제외
간선 0-5 : 11
간선 0-3 : 13 - 사이클 생성으로 제외
```

```
//·간선을·나타내는·구조체
struct·Edge·{int·start,·end,·weight;};
typedef·struct·GraphType·{ → //·그래프·구조체
→   int·e,·v; → → → → //·간선의·개수,·정점·개수
→   struct·Edge*·edges;→ → //·간선·구조체·포인터
}GraphType;
```

1. 우선 GraphType 구조체에 멤버 n 을 다음과 같이 e 와 v 로 구별하였다. 간선의 개수를 의미하는 e, 그리고 정점의 개수를 의미하는 v 로 구별하였다.

```
int·vertex_list[MAX_VERTICES],·size·=·0;
```

2. main 함수에서는 정점의 개수를 확실히 구하기 위하여 다음과 같이 정점들을 중복 없이 저장하는 vertex_list 배열을 선언하고, 정점의 개수를 의미하는 size 변수를 0 으로 초기화하였다.

```
while (!feof(fp)) {
    fscanf(fp, "%d %d %d", &u, &v, &w);
    if(findIndex(u,vertex_list,size)==0) vertex_list[size++] = u;
    if (findIndex(v, vertex_list, size) == 0) vertex_list[size++] = v;
    numOfEdges++;
}
graph_init(g, size, numOfEdges);    //그래프 초기화
rewind(fp);
```

3. 이후 파일을 첫번째로 읽을 때 간선의 개수와 더불어 정점의 개수도 함께 구하였다. 아래의 findIndex 함수를 이용하여 중복되지 않게 정점의 값들을 vertex_list 에 저장하는 방식으로 정점의 개수를 구하였다.

4. 그래프를 초기화하는 graph_init 함수에도 변화가 있다. 포인터 g 와 간선의 개수 numOfEdges 와 함께, 파일을 읽으면서 구한 정점의 개수 size 도 함께 인자로 전달하게 된다.

```
//정점을 저장하는 배열에 찾고자 하는 정점이 있는 경우에는 1 반환
int findIndex(int v, int vertex_list[], int size) {
    int i = 0;
    for (i = 0;i < size;i++)
        if (v == vertex_list[i])return 1;
    return 0;
}
```

5. findIndex 함수는 위와 같다.

```
//그래프 초기화
void graph_init(GraphType* g, int numOfVertices, int numOfEdges) {
    int i;
    g->v = numOfVertices;  //그래프의 정점 개수
    g->e = 0;              //그래프의 간선 개수 (0으로 초기화)
    g->edges = (struct Edge*)malloc(sizeof(struct Edge) * numOfEdges);
    for (i = 0;i < numOfEdges;i++) {
        g->edges[i].start = 0;
        g->edges[i].end = 0;
        g->edges[i].weight = INF;
    }
}
```

6. 그래프를 초기화하는 graph_init 함수이다. 정점 개수와 간선 개수를 인수로 전달받아, g 의 v 멤버에는 정점 개수를 대입하여 초기화하고, 그래프의 간선은 0 으로 초기화한다. insert_edge 함수에서 중복되지 않게 간선을 삽입할 때 이 값을 증가시키기 위하여 0 의 값을 초깃값으로 준다. 그리고 간선의 개수만큼 edges 배열의 start 와 end 에 0 을, 그리고 weight 에는 INF 의 값으로 초기화한다.

```
//·간선·삽입·연산
void·insert_edge(GraphType*·g,·int·start,·int·end,·int·w)·{
→   if·(if_edge(g,·start,·end)·==·TRUE)·{
→   →   printf("간선·%d·-·%d·은·이미·추가된·간선입니다.·--·제외·\n",·start,·end);
→   →   return;
→   }
→   g->edges[g->e].start·=·start;
→   g->edges[g->e].end·=·end;
→   g->edges[g->e].weight·=·w;
→   g->e++;

→   printf("간선·%d·-·%d·추가·완료·\n",·start,·end);
}
```

7. insert_edge 함수 내에서 중복되지 않게 간선을 추가할 때, g->e 의 값을 증가시킨다.

8. kruskal 함수에는 큰 변화가 없다. 종료 조건이 동일하다.

```
//·kruskal의·최소·비용·신장·트리·프로그램
void·kruskal(GraphType*·g)·{
→   int·i;
→   int·uset,·vset;→→   →   //·정점·u와·정점·v의·집합·번호
→   int·edge_accepted·=·0;→//·현재까지선택된·간선의·수
→   int·sum_of_weight·=·0;//·선택되는·간선의·비용을·저장
→   struct·Edge·e;

→   set_init(g->e);→→   //·집합·초기화
→   qsort(g->edges,·g->e,·sizeof(struct·Edge),·compare);

→   i·=·0;
→   while·(edge_accepted·<·(g->v·-·1))·{→   //·종료·조건1.·간선의·수·<·(정점·개수·-·1)
→   →   e·=·g->edges[i];
→   →   uset·=·set_find(e.start);→ //·정점·u의·집합·번호
→   →   vset·=·set_find(e.end);→→   //·정점·v의·집합·번호

→   →   if·(uset·==·vset)·//·동일한·집합이라면·사이클·생성
→   →   →   printf("간선·%d-%d·:·%d·-·사이클·생성으로·제외·\n",·e.start,·e.end,·e.weight);
→   →   else·if·(uset·!=·vset)·{→   //·서로·속한·집합이·다르면
→   →   →   printf("간선·%d-%d·:·%d·\n",·e.start,·e.end,·e.weight);
→   →   →   edge_accepted++;→   //·간선·선택
→   →   →   set_union(uset,·vset);·//·두·개의·집합을·합친다
→   →   →   sum_of_weight·+=·e.weight;
→   →   }
→   →   i++;
→   }
→   printf("\n<·필요한·최소·비용·%d·>·\n",·sum_of_weight);
}
```

간선의 개수와 정점의 개수를 저장하도록 GraphType 구조체를 변경시키고, 위와 같이 코드 일부를 수정하여 실행시켜본 결과, 다음 실행창과 같이 마지막에 (0-3)간선은 확인하지 않음을 알 수 있다. 이는 kruskal 알고리즘의 본래 종료 조건인, 추가된 간선의 개수가 정점의 개수보다 1 작으면 반복문을 종료하고 더 이상의 간선을 확인하지 않는 조건을 만족하기 때문이다.

Microsoft Visual Studio 디버그 콘솔

```
>> 데이터 입력
간선 0 - 3 추가 완료
간선 0 - 5 추가 완료
간선 1 - 4 추가 완료
간선 1 - 5 추가 완료
간선 2 - 5 추가 완료
간선 2 - 7 추가 완료
간선 3 - 0 은 이미 추가된 간선입니다. -- 제외
간선 3 - 4 추가 완료
간선 3 - 7 추가 완료
간선 5 - 1 은 이미 추가된 간선입니다. -- 제외
간선 5 - 7 추가 완료


>> 과정
간선 2-7 : 2
간선 1-5 : 4
간선 1-4 : 5
간선 2-5 : 6
간선 5-7 : 7 - 사이클 생성으로 제외
간선 3-4 : 8
간선 3-7 : 9 - 사이클 생성으로 제외
간선 0-5 : 11

< 필요한 최소 비용 36 >
```

1.6 느낀점

Kruskal 알고리즘에 대하여 "1.5 추가 소스 코드 구현" 단계를 추가하기 까지 알고리즘을 이해하고 이를 코드로 구현하는 데 있어서 여러가지 난관에 거쳤기 때문에 많은 시간이 걸렸던 것 같다. 인터넷에 C 언어로 kruskal 알고리즘을 구현한 코드 예제들도 찾아보았지만, 아무래도 g->n 에 대한 값이 가장 마음에 걸렸기 때문에 결국 정점의 개수와 간선의 개수로 구분하여 코드를 조금 수정하는 것이 좋겠다고 판단하여 결국 수정한 코드 분석 단계를 추가하게 되었다. Kruskal 알고리즘의 union-find 연산을 이해하는 것에 있어서도 많은 시간이 걸렸다. 반복문이 실행되며 각 단계별로 트리를 배열로 구현한 parent 배열의 변화를 계속 살피며 각 트리 집합을 직접 그려보니 조금씩 이해를 할 수 있게 되었던 것 같다. 많은 어려움을 겪은 알고리즘인만큼, 앞으로도 최소 비용 신장 트리를 구현하는데 kruskal 알고리즘이 사용되는 프로그램을 만나면 반갑게 느껴질 것 같다.

2. Prim 의 MST 알고리즘을 이용한 최소 비용 신장 트리 프로그램

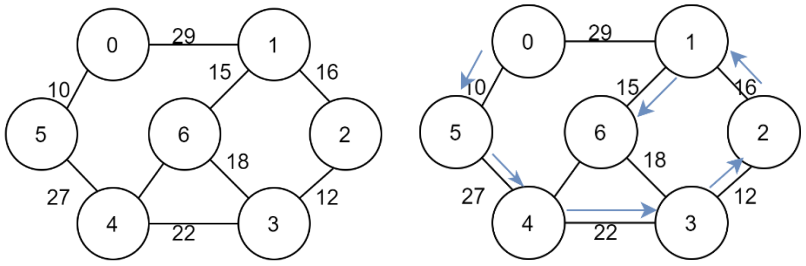
- Prim의 MST 알고리즘
  - 412 페이지에 프로그램 11.9를 참고하여 Prim의 최소 비용 신장 트리 프로그램을 작성하여 테스트 하시오.
    - data.txt에서 그래프의 정보를 가져오게 수정 (x y w -> x,y 정점 / w 가중치)
    - 동적 할당 이용

2.1 문제 분석

Prim 의 MST 알고리즘은 시작 정점에서부터 출발하여, 신장 트리 집합을 단계적으로 확장해나가는 방법이다. 간선의 가중치 오름차순으로 정렬하여, Prim 알고리즘은 시작 정점이 우선적으로 신장 트리 집합에 포함되고, 앞 단계에서 만들어진 신장 트리 집합에 인접한 정점들 중 최저 간선으로 연결된 정점을 선택하여 트리를 확장해나가는 과정을 거친다. 이 과정을 트리가 n-1 개의 간선을 가질 때까지, 즉 시장 트리의 집합에 정점의 개수가 n-1 개가 될 때까지 이 과정을 반복한다.

Kruskal 은 가중치의 값을 토대로 오름차순으로 간선 정보를 정렬하여, 사이클을 형성하지 않는 조건 하에 집합에 작은 가중치 값을 가지는 간선부터 추가하여 간선을 기반으로 하는 알고리즘이다. 반면, Prim 알고리즘은 다음 정점까지의 가중치 값을 비교하는 정점 기반의 알고리즘이며, 이전 단계에서 만들어진 신장 트리를 확장해나가는 방식이다.

Prim 알고리즘 구현을 위해서는 distance 라는 정점 개수 크기의 배열이 필요하다. 이 배열은 현재까지 알려진, 신장 트리 정점 집합에서 각 정점까지의 거리를 가지고 있다.



파일에 저장된 간선 정보들을 이용하여 그래프를 만들면 위의 그림과 같다. 간단히 그 과정을 살펴보도록 하자. 정점 0 을 신장 트리에 추가하고, 0 부터 인접한 정점에 해당하는 distance 값을 변경시킨다. 즉, 0 과 인접한 정점은 정점 5 와 정점 1 이므로 distance[5]는 10 으로, 그리고 distance[1]은 29 로 변경하는 것이다. distance 배열에 저장된 가중치 값 중 가장 작은 값인 5 를 선택하게 되고, 다음으로 5 와 인접한 정점 중 선택되지 않은 정점 4 에 해당하는 distance 배열 값을 27 로 변경한다. 이후, distance 에 저장된 값들 중 다시 가장 작은 값을 지닌 4 를 선택하고, 다음으로 정점 4 에 대해 다시 이 과정을 반복한다. 이러한 과정을 모든 정점이 선택될 때까지 반복하면 최소 비용 신장 트리가 완성된다.

자세한 구현은 다음 소스 코드 분석을 통해 살펴보자.

2.2 소스 코드

```c
/*
    작성자: 이예빈(20204059)
    작성일: 2021.11.07
    프로그램명: Prim의 MST 알고리즘을 이용한 최소 비용 신장 트리 프로그램
*/

#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 30
#define INF 1000L

typedef struct GraphType {
    int n; // 정점의 개수
    int** weight;  // 각 간선의 정보를 담은 2차원 배열
}GraphType;

// 그래프 초기화, 동적 할당
void graph_init(GraphType* g, int n) {
    int i, j;
    g->n = n;
    g->weight = (int**)malloc(sizeof(int*) * n);
    for (i = 0; i < n; i++) {
        g->weight[i] = (int*)malloc(sizeof(int) * n);
        for (j = 0; j < n; j++) {
            g->weight[i][j] = INF; // 전체 간선 가중치 값을 INF로 초기화
            if (i == j) g->weight[i][j] = 0; // 자기 자신에 대한 가중치 값은 0
        }
    }
}

// 간선 삽입 연산
void insert_edge(GraphType* g, int u, int v, int w) {
    // 무방향 그래프
    g->weight[u][v] = w;
    g->weight[v][u] = w;
}

```

```
41  int selected[MAX_VERTICES];//선택된 정점들 정보
42  int distance[MAX_VERTICES];//정점으로까지의 거리 정보
43
44  //최소 dist[v] 값을 갖는 정점을 반환
45  int get_min_vertex(int n) {
46      int v, i;
47      for (i = 0;i < n;i++)
48          //아직 선택되지 않은 정점의 번호
49          if (!selected[i]) {
50              v = i;
51              break;
52          }
53      for (i = 0;i < n;i++)
54          //선택되지 않은 정점 중, distance 값이 가장 작은 인덱스 값을 v에 저장
55          if (!selected[i] && (distance[i] < distance[v]))
56              v = i;
57      return (v);
58  }
60  void prim(GraphType* g, int s) {
61      int i, u, v;
62      int added[MAX_VERTICES]; //추가된 정점들을 차례대로 저장
63      int sum_of_weight = 0; //추가되는 간선들의 가중치 합들을 저장
64
65      printf("-- Prim의 최소 비용 신장 트리 프로그램 --\n\n");
66      printf(">> 과정 \n");
67      for (u = 0;u < g->n;u++)
68          distance[u] = INF; //distance의 모든 값을 INF로 초기화
69      distance[s] = 0;    //시작 정점의 distance 값을 0으로 초기화
70      for (i = 0;i < g->n;i++) {
71          printf("%d >> ", i + 1);
72          u = get_min_vertex(g->n);//가장 적은 가중치 값을 갖는 정점 u
73          selected[u] = TRUE;
74          if (distance[u] == INF)return;
75
76          sum_of_weight += distance[u];//추가된 간선의 가중치를 갱신하여 더함
77          added[i] = u;  //added에 선택된 정점 u를 추가
78
79          for (v = 0;v < g->n;v++)
80              if (g->weight[u][v] != INF)//정점 u와 인접한 정점 중
81                  //아직 방문하지 않은 정점이면 distance에 추가
82                  if (!selected[v] && g->weight[v][u] < distance[v]) {
83                      distance[v] = g->weight[u][v];
84                  }
85
86          //시작 정점부터 선택된 정점들을 차례대로 나열
87          for (v = 0;v <= i;v++)
88              printf("%d ", added[v]);
89          printf(": %d \n", sum_of_weight);
90      }
91      printf("\n< 필요한 최소 비용 %d > \n", sum_of_weight);//전체 가중치의 합 출력
92  }
93
94  void destroy_graph(GraphType* g) {
95      int i;
96      for (i = 0;i < g->n;i++)
97          free(g->weight[i]);
98  }
99
```

```
100  int main(void) {
101      FILE* fp;
102      GraphType* g;
103      int u, v, w;    //정점 u,v, 간선 가중치 w
104      int max_v = 0; //정점 중 가장 큰 값
105
106      g = (GraphType*)malloc(sizeof(GraphType));
107
108      fp = fopen("data.txt", "rt");
109      if (fp == NULL) {
110          fprintf(stderr, "파일 열기 실패\n");
111          exit(1);
112      }
113
114      //먼저 파일을 읽어 가장 큰 값을 기준으로 그래프 초기화 하기
115      while (!feof(fp)) {
116          fscanf(fp, "%d%d%d", &u, &v, &w);
117          //정점 중 가장 큰 정점 탐색
118          if (u > max_v)max_v = u;
119          if (v > max_v)max_v = v;
120      }
121      graph_init(g, max_v + 1);  //가장 큰 정점보다 값이 1 큰 수로 할당하도록 그래프 초기화
122      rewind(fp);
123
124      while (!feof(fp)) {
125          fscanf(fp, "%d%d%d", &u, &v, &w);
126          insert_edge(g, u, v, w);    //간선 추가
127      }
130      destroy_graph(g);  //동적 할당된 2차원 배열 메모리 해제
131      free(g);           //그래프 메모리 해제
132      fclose(fp);        //파일 닫기
133      return 0;
134  }
```

2.3 소스 코드 분석

```
/*
    작성자: 이예빈(20204059)
    작성일: 2021.11.07
    프로그램명: Prim의 MST 알고리즘을 이용한 최소 비용 신장 트리 프로그램
*/

#include <stdio.h>
#include <stdlib.h>
```

1. 작성자, 작성일, 프로그램명을 쓴다. 이 프로그램은 Prim의 MST 알고리즘을 이용한 최소 비용 신장 트리를 구하는 프로그램이다. 필요한 헤더 파일들도 추가한다.

```
#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 30
#define INF 1000L
```

2. 프로그램에서 필요한 기호 상수들 몇 가지를 정의해준다. 1과 0의 TRUE와 FALSE를 정의하고, 미리 전역으로 선언할 배열들의 크기를 임의로 지정하기 위해, 30이라는 적당히 큰 값인 30으로 MAX_VERTICES를 정의한다. 정점들의 최대 개수를 의미하게 된다. 그리고 정점과 정점 사이에 간선이 존재하지 않는 경우의 가중치 값을 임의의 1000L(long type)의 값으로 INF를 정의한다.

```
typedef struct GraphType {
    int n; //정점의 개수
    int** weight;  //각 간선의 정보를 담은 2차원 배열
}GraphType;
```

3. 그래프 구조체 GraphType이다. 정점의 개수를 저장할 정수형 변수 n과 정점과 정점 사이의 간선의 가중치 값들을 저장하는 2차원 배열을 가리킬 정수형 이중 포인터 weight을 멤버로 갖는다.

main 함수의 동작을 보기 전에, 프로그램에서 사용되는 함수들을 먼저 살펴보도록 하자.

```
//·그래프·초기화,·동적·할당
void·graph_init(GraphType*·g,·int·n)·{
    int·i,·j;
    g->n·=·n;
    g->weight·=·(int**)malloc(sizeof(int*)·*·n);
    for·(i·=·0;i·<·n;i++)·{
        g->weight[i]·=·(int*)malloc(sizeof(int)·*·n);
        for·(j·=·0;j·<·n;j++)·{
            g->weight[i][j]·=·INF;·//·전체·간선·가중치·값을·INF로·초기화
            if·(i·==·j)g->weight[i][j]·=·0;·//·자기·자신에·대한·가중치·값은·0
        }
    }
}
```

4. 그래프를 초기화하는 graph_init 함수이다. 그래프 포인터 g와 정점의 개수 n을 매개 변수로 전달 받는다. 포인터 g의 n 멤버에는 n의 값을 저장하고, weight 이중 포인터에는 우선적으로 n의 크기만큼 정수형 포인터 배열을 할당한다. 그리고 다시 각 포인터 배열에는 n의 크기만큼 int 형 공간을 동적 할당한다. 전체 간선 가중치 값을 INF로 초기화하고, 자기 자신에 대한 가중치 값으로는 0을 대입한다.

```
//·간선·삽입·연산
void·insert_edge(GraphType*·g,·int·u,·int·v,·int·w)·{
    //·무방향·그래프
    g->weight[u][v]·=·w;
    g->weight[v][u]·=·w;
}
```

5. 간선을 삽입하는 insert_edge 함수이다. 두 정점 u와 v, 그리고 정점 사이의 간선의 가중치 값인 w를 매개 변수로 전달 받는다. 프로그램에서는 무방향 그래프를 다루므로, u와 v에 대한 배열 값에 모두 w의 값을 삽입한다.

```
int·selected[MAX_VERTICES];→//·선택된·정점들·정보
int·distance[MAX_VERTICES];→//·정점으로까지의·거리·정보
```

6. 전역으로 두 정수형 배열을 MAX_VERTICES의 크기로 선언한다. 모두 0(FALSE)의 값으로 초기화된다. selected 배열은 선택된 정점들의 정보를 나타낸다. 정점이 선택되어 방문되면 해당 인덱스의 배열 값을 TRUE로 변경하게 된다. distance 배열에는 인덱스에 해당하는 정점까지의 거리(가중치 값)을 저장하게 된다.

```
// 최소 dist[v] 값을 갖는 정점을 반환
int get_min_vertex(int n) {
    int v, i;
    for (i = 0; i < n; i++)
        // 아직 선택되지 않은 정점의 번호
        if (!selected[i]) {
            v = i;
            break;
        }
    for (i = 0; i < n; i++)
        // 선택되지 않은 정점 중, distance 값이 가장 작은 인덱스 값을 v에 저장
        if (!selected[i] && (distance[i] < distance[v]))
            v = i;
    return (v);
}
```

7. distance 에 저장된 값들 중, 아직 선택되지 않은 정점까지의 가중치가 가장 작은 값에 해당하는 정점을 반환하는 get_min_vertex 함수이다. 첫번째 반복문에서는, 우선 아직 선택되지 않은 정점의 번호 중 가장 작은 정점을 v 에 대입한다.

8. 두번째 반복문에서는 아직 선택되지 않은 정점 중, distance 값이 가장 작은 정점 인덱스를 v 에 대입하고, 이 값을 반환하도록 한다.

다음은 최소 비용으로 전체 노드를 방문하는 prim 알고리즘을 구현한 함수이다.

```
void prim(GraphType* g, int s) {
    int i, u, v;
    int added[MAX_VERTICES]; //추가된 정점들을 차례대로 저장
    int sum_of_weight = 0; //추가되는 간선들의 가중치 합들을 저장
```

9. 그래프 포인터 g 와 시작 정점 s 를 매개 변수로 전달받는다. 반복문 제어 변수, 배열의 인덱스 번호로 사용할 i, u, v 변수를 선언한다. 추가되는 정점들을 차례대로 저장할 added 배열을 선언하고, 추가되는 정점까지의 간선의 가중치 합들을 저장할 sum_of_weight 을 0 으로 초기화하여 선언한다.

```
    printf("- Prim의 최소 비용 신장 트리 프로그램 -\n\n");
    printf(">> 과정 \n");
    for (u = 0; u < g->n; u++)
        distance[u] = INF; //distance의 모든 값을 INF로 초기화
    distance[s] = 0;    //시작 정점의 distance 값을 0으로 초기화
```

10. 우선 g->n (정점의 개수)의 값만큼 distance 배열의 값들을 차례대로 모두 INF 로 초기화한다. 시작 정점의 인덱스에 해당하는 distance 배열 값으로는 0 을 대입힌다.

```
for (i = 0;i < g->n;i++) {
    printf("%d >> ", i + 1);
    u = get_min_vertex(g->n);//가장 적은 가중치 값을 갖는 정점 u
    selected[u] = TRUE;
    if (distance[u] == INF)return;
```

11. 정점의 개수만큼 반복문이 실행된다. 우선 get_min_vertex 함수를 호출하여 얻게 되는, 가장 적은 가중치 값을 갖는 정점 u 를 얻는다. u 번 인덱스에 해당하는 selected 값은 TRUE 로 변경한다. 만약 u 번 인덱스에 해당하는 distance 값이 INF 이면 해당 함수를 종료한다.

```
    sum_of_weight += distance[u];//추가된 간선의 가중치를 갱신하여 더함
    added[i] = u;  //added에 선택된 정점 u를 추가

    for (v = 0;v < g->n;v++)
        if (g->weight[u][v] != INF)//정점 u와 인접한 정점 중
            //아직 방문하지 않은 정점이면 distance에 추가
            if (!selected[v] && g->weight[v][u] < distance[v]) {
                distance[v] = g->weight[u][v];
            }
```

12. sum_of_weight 값은 정점 u 에 해당하는 distance(가중치값)을 더하여 갱신한다. added 배열에는 정점 u 를 추가한다.

13. 이후 반복문 내에서 변수 v 가 0 부터 g->n 까지 증가하며, 정점 u 와 인접한 정점이면서, 동시에 아직 방문하지 않은 정점이라면 distance 배열에 해당 가중치 값을 새로 추가하도록 한다.

```
    //시작 정점부터 선택된 정점들을 차례대로 나열
    for (v = 0;v <= i;v++)
        printf("%d ", added[v]);
    printf(": %d \n", sum_of_weight);
}
printf("\n< 필요한 최소 비용 %d >\n", sum_of_weight);//전체 가중치의 합 출력
}
```

14. 마지막으로, 각 단계별로 선택된 정점들을 시작 정점부터 차례대로 출력하도록 한다. 그리고 각 단계에서 지나쳐온 정점들까지의 모든 가중치 값들도 함께 출력한다.

15. prim 알고리즘을 통해 최소 비용 신장 트리가 완성되고 나면, 전체 가중치의 합, 곧 필요한 최소 비용의 전체 합을 출력한다.

```
void destroy_graph(GraphType* g) {
    int i;
    for (i = 0;i < g->n;i++)
        free(g->weight[i]);
}
```

16. 그래프에 동적 할당된 메모리들을 해제하는 destroy_graph 함수이다. 정점의 개수만큼 할당된 포인터 배열과 할당된 정수 공간들의 메모리를 모두 해제한다.

다음으로 main 함수의 동작을 살펴보자.

```
int main(void) {
    FILE* fp;
    GraphType* g;
    int u, v, w;    //정점 u,v, 간선 가중치 w
    int max_v = 0; //정점 중 가장 큰 값

    g = (GraphType*)malloc(sizeof(GraphType));
```

17. 파일 포인터 fp, 그래프 포인터 g, 파일에서 읽을 정점 u 와 v, 간선의 가중치 w, 그리고 동적 할당을 할 때 필요한 값인 정점 들 중 가장 큰 수의 정점인 max_v 를 0 으로 초기화하여 선언한다.

18. 포인터 g 에는 우선 GraphType 크기의 메모리를 동적 할당한다.

```
    fp = fopen("data.txt", "rt");
    if (fp == NULL) {
        fprintf(stderr, "파일 열기 실패\n");
        exit(1);
    }

    //먼저 파일을 읽어 가장 큰 값을 기준으로 그래프 초기화 하기
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &u, &v, &w);
        //정점 중 가장 큰 정점 탐색
        if (u > max_v)max_v = u;
        if (v > max_v)max_v = v;
    }
    graph_init(g, max_v + 1);  //가장 큰 정점보다 값이 1 큰 수로 할당하도록 그래프 초기화
    rewind(fp);
```

19. "data.txt" 파일을 읽기 모드로 열고, 우선적으로 파일을 읽어 정점 정보와 가중치 값을 u,v,w 로 임시적으로 읽고, 정점 중 가장 큰 정점을 탐색하여 max_v 에 저장한다.

20. 그래프 포인터 g 와 max_v 에 1 을 더한 값을 graph_init 함수의 인수로 전달하여 그래프를 초기화하고 필요한 배열을 동적 할당한다. rewind 함수를 호출하여 파일을 다시 읽도록 한다.

```
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &u, &v, &w);
        insert_edge(g, u, v, w);    //간선 추가
    }
```

21. u, v, w 의 값을 읽어 insert_edge 함수에 인수로 전달하여 각 간선의 정보들을 추가하도록 한다.

```
    destroy_graph(g);  //동적 할당된 2차원 배열 메모리 해제
    free(g);           //그래프 메모리 해제
    fclose(fp);        //파일 닫기
    return 0;
}
```

22. 마지막으로 destroy_graph 함수를 호출하여 그래프에 동적 할당된 2 차원 배열의 메모리를 해제한다. 그래프 또한 메모리를 해제하고, 파일을 닫고 전체 프로그램을 종료하도록 한다.

2.4 실행 결과

data.txt 파일은 다음과 같이 각 줄에 두 정점, 그리고 두 정점 사이의 간선의 가중치 값의 정보를 담고 있다.

data.txt - Windows 메모장

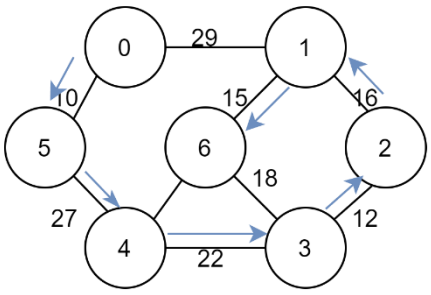
파일(F) 편집(E) 서식(O) 보기(V

```
0 1 29
0 5 10
1 2 16
1 6 15
2 3 12
3 4 22
3 6 18
4 5 27
4 6 25
```

위의 파일에 저장되어 있는 정점과 간선의 정보들을 읽어 Prim 의 MST 알고리즘을 이용하여 최소 비용 신장 트리를 만드는 과정은 다음과 같다. 정점 0 부터 시작하여 차례대로 5(0-5: 10), 4(5-4: 27), 3(4-3: 22), 2(3-2: 12), 1(2-1: 16), 6(1-6: 15)의 정점을 방문하며 각 단계별로 가중치의 값들도 갱신되는 것을 확인할 수 있다. 필요한 최소 비용은 그 합이 102 임인 것 또한 확인 가능하다.

Microsoft Visual Studio 디버그 콘솔

```
- Prim의 최소 비용 신장 트리 프로그램 -

>> 과정
1 >> 0 : 0
2 >> 0 5 : 10
3 >> 0 5 4 : 37
4 >> 0 5 4 3 : 59
5 >> 0 5 4 3 2 : 71
6 >> 0 5 4 3 2 1 : 87
7 >> 0 5 4 3 2 1 6 : 102

< 필요한 최소 비용 102 >

C:₩Users₩이예빈₩Desktop₩CSE 2021-2₩자료구
₩자료구조2실습₩20204059-이예빈-실습9주차
```



2.5 느낀점

Prim 알고리즘은 이해하는 과정에 있어 많은 어려움을 겪은 union-find 알고리즘을 포함하는 Kruskal 알고리즘보다는 조금 더 쉽게 느껴진 것 같다. 시작 정점을 기준으로, 다음 정점까지의 거리(가중치 값)을 distance 배열에 계속 저장하고, 이 값들 중 가장 작은 값을 선택한다는 점에 있어서 구현된 코드 자체를 이해하는 것이 쉽지는 않았지만, 그래프 상에서 직접 그림을 그려가며 이해하는 부분에 있어서는 kruskal 에 비해서는 조금 더 간단하게 느껴졌던 것 같다.

3. 느낀점

최소 비용으로 모든 노드들을 접근하는 MST 알고리즘을 구현하는 Kruskal과 Prim의 MST 알고리즘을 배우는 것이 쉽지만은 않았다. 두 알고리즘은 구현 방법과 접근 방식이 매우 다르기 때문에 두 알고리즘의 차이점을 명확히 이해하는 것이 중요할 것 같다. 두 그래프 모두 정점들 간에 연결된 간선이 많은 밀접 그래프가 아니었으므로 구현된 알고리즘을 이애하는 데 있어서 아주 어렵지는 않았지만, 특히나 kruskal의 경우 사이클을 형성하지 않는 조건 하에 간선을 선택하게 되므로, 밀접 그래프의 경우 그 과정이 매우 복잡해질 것 같다. 앞으로 배우게 될 많은 그래프들에 대한 탐색 알고리즘에 대한 이해도를 높이고 싶다.