

# 알고리즘\_과제\_9

(MST by prim, kruskal)



학부: 컴퓨터학부

학번: 20162518

출석번호 : 156번

이름 : 최승서

이번 과제는 Prim 알고리즘과 Kruskal 알고리즘을 통해 최소 신장 트리(MST)를 구해 보았다. Prim 알고리즘과 Kruskal 알고리즘은 같은 문제를 서로 다른 방식을 적용하여 해결한다.

먼저 MST(Minimum Spaning Tree), 최소 신장 트리는 한 그래프에서 모든 정점을 연결하며 순환이 생기지 않아야 하며 각 간선들의 가중치의 합이 최소가 되어야한다.

Prim algorithm은 하나의 시작 정점을 설정하고 시작 정점과 연결된 정점들에 대해 가장 작은 가중치의 간선부터 연결해 나가며 MST를 완성한다. 단 순환이 발생하는 경우 가장 작은 가중치를 가져도 무시한다.

다음으로 Kruskal algorithm은 모든 간선들 중에 가중치가 작은것 부터 차례로 연결을 하면서 MST를 만드는 방식인데 prim과 동일하게 순환이 발생하는 경우 가장 작은 가중치를 가져도 무시한다.

### <PRIM'S ALGORITHM 소스 코드>

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0

#define MAX 10 //노드의 개수
#define INF 987654321

int W[MAX][MAX]={ //해당 그래프의 인접행렬, 직접 연결되었지 않으면 INF로 표기
{ 0, 32, INF, 17, INF, INF, INF, INF, INF, INF },
{ 32, 0, INF, INF, 45, INF, INF, INF, INF, INF },
{ INF, INF, 0, 18, INF, INF, 5, INF, INF, INF },
{ 17, INF, 18, 0, 10, INF, INF, 3, INF, INF },
{ INF, 45, INF, 10, 0, 28, INF, INF, 25, INF },
{ INF, INF, INF, INF, 28, 0, INF, INF, INF, 6 },
{ INF, INF, 5, INF, INF, INF, 0, 59, INF, INF },
{ INF, INF, INF, 3, INF, INF, 59, 0, 4, INF },
{ INF, INF, INF, INF, 25, INF, INF, 4, 0, 12 },
{ INF, INF, INF, INF, INF, 6, INF, INF, 12, 0 } };

int nearest[MAX];
int distance[MAX];

int get_min_vertex(int n){ //최소 distance[v] 값을 갖는 정점을 반환
    int v = 0,i;

    for (i = 0; i <n; i++){
        if (!nearest[i]){
            v = i;
            break;
        }
    }

    for (i = 0; i < n; i++){ // 연결하지 않은 간선들 중 최소 가중치값을 갖는 정점을 찾음
        if( !nearest[i] && (distance[i] < distance[v]))
            v = i;
    }

    return (v);
}
```

```

void prim(int start)
{
    int i, j, v;
    int sum=0;

    for(j=0;j<MAX;j++){ //초기화
        distance[j]=INF;
    }
    distance[start]=0;
    for(i=0;i<MAX;i++){
        j = get_min_vertex(MAX);
        nearest[j]=TRUE; // 지나간 경로 체크

        if( distance[j] == INF )
            return;

        printf("[V%d]연결 weight : %d\n", j+1, distance[j]);
        sum+=distance[j]; //총거리를 구하기 위해 나온 거리를 계속 더해줌

        for( v=0; v<MAX; v++){
            if( W[j][v] != INF){
                if( !nearest[v] && W[j][v]< distance[v] )
                    distance[v] = W[j][v]; // 간선의 거리를 갱신
            }
        }
    }
    printf("MST by Prim's algoritm : %d\n", sum);
}

int main()
{
    prim(9);
}

```

## <출력 결과>

```

[V1]연결 weight : 0
[V4]연결 weight : 17
[V8]연결 weight : 3
[V9]연결 weight : 4
[V5]연결 weight : 10
[V10]연결 weight : 12
[V6]연결 weight : 6
[V3]연결 weight : 18
[V7]연결 weight : 5
[V2]연결 weight : 32
MST by Prim's algoritm : 107
Program ended with exit code: 0

```


All Output ↕

Filter



정점 V1부터 시작했을 때 결과

```
[V10]연결 weight : 0
[V6]연결 weight : 6
[V9]연결 weight : 12
[V8]연결 weight : 4
[V4]연결 weight : 3
[V5]연결 weight : 10
[V1]연결 weight : 17
[V3]연결 weight : 18
[V7]연결 weight : 5
[V2]연결 weight : 32
MST by Prim's algorithm : 107
Program ended with exit code: 0
```

All Output 

 Filter



정점 V10부터 시작했을 때 결과

### <KRUSKAL ALGORITHM 소스 코드>

```
#include <stdio.h>

#define TRUE 1
#define FALSE 0
#define MAX_NODE 10
#define INF 987654321
#define MAX_ELEMENT 13 // 힙의 최대크기

int parent[MAX_NODE]; // 각 정점의 부모 노드
int num[MAX_NODE]; // 각 집합의 크기

typedef struct{
    int distance; // 간선의 가중치
    int p; // 정점 1
    int q; // 정점 2
} element;

typedef struct{
    element heap[MAX_ELEMENT];
    int heap_size;
} HeapType;

void init(int n){ // 노드 초기화

    int i;
    for (i = 0; i < n; i++){
        parent[i] = -1;
        num[i] = 1;
    }
}

int find(int vertex){ // 같은 부모를 가지는지 확인하는 함수
    int p, s, i = -1; // p: 부모노드
```

```

    for (i = vertex; (p = parent[i]) >= 0; i = p);

    s = i; // 루트노드 정보를 s에 저장

    for (i = vertex; (p = parent[i]) >= 0; i = p)
        parent[i] = s; //부모노들이 s로 가게 한다

    return s; // 루트 노드 반환
}

void merge(int p, int q){
    if (num[p] < num[q]){ //더 큰쪽으로 부모노드를 합침
        parent[p] = q;
        num[p] += num[q];
    }
    else{
        parent[q] = p;
        num[p] += num[q];
    }
}

void init_heap(HeapType *h){
    h->heap_size = 0;
}

void insert_min_heap(HeapType *h, element item){
    int i = ++(h->heap_size);
    while ((i != 1) && (item.distance < h->heap[i / 2].distance)){
        h->heap[i] = h->heap[i / 2];
        i /= 2;
    }
    h->heap[i] = item;
}

element delete_min_heap(HeapType *h){
    int parent, child;
    element item, temp;

    item = h->heap[1];

    temp = h->heap[(h->heap_size)--];

    parent = 1;
    child = 2;

    while (child <= h->heap_size){
        if ((child < h->heap_size) && (h->heap[child].distance) > h->heap[child + 1].distance)
            child++;
        if (temp.distance <= h->heap[child].distance)
            break;

        h->heap[parent] = h->heap[child];
        parent = child;
        child *= 2;
    }
    h->heap[parent] = temp;
    return item;
}

void insert_heap_edge(HeapType *h, int p, int q, int weight){
    element e;
    e.p = p;
    e.q = q;
    e.distance = weight;
    insert_min_heap(h, e);
}

void insert_all_edges(HeapType *h){
    insert_heap_edge(h, 3, 7, 3);
    insert_heap_edge(h, 7, 8, 4);
    insert_heap_edge(h, 2, 6, 5);
    insert_heap_edge(h, 5, 9, 6);
    insert_heap_edge(h, 3, 4, 10);
    insert_heap_edge(h, 8, 9, 12);
    insert_heap_edge(h, 0, 3, 17);
    insert_heap_edge(h, 2, 3, 18);
    insert_heap_edge(h, 4, 7, 28);
    insert_heap_edge(h, 0, 1, 32);
    insert_heap_edge(h, 1, 4, 45);
}

```

```

    insert_heap_edge(h, 6, 7, 59);
}

void kruskal(HeapType h){
    int edge_accepted = 0; // 현재까지 연결된 간선 수
    int p, q; // 서로 연결되었는 2노드
    int sum = 0;
    element e;

    init(MAX_NODE); // 부분집합 초기화

    while (edge_accepted < (MAX_NODE - 1)){ // 간선의 수 < (n - 1)
        e = delete_min_heap(&h);
        p = find(e.p);
        q = find(e.q);

        // 두 집합이 서로 다른 집합에 속한다면
        if (p != q){
            printf("V%d, V%d 연결 weight : %2d \n", e.p+1, e.q+1, e.distance);
            edge_accepted++;
            sum+= e.distance;
            merge(p, q); // 두 개의 집합을 합친다.
        }
    }
    printf("MST by kruskal algorithm : %d\n", sum);
}

int main()
{
    HeapType h;
    init_heap(&h);
    insert_all_edges(&h);
    kruskal(h);
}

```

## <출력 결과>

```

V4, V8 연결 weight : 3
V8, V9 연결 weight : 4
V3, V7 연결 weight : 5
V6, V10 연결 weight : 6
V4, V5 연결 weight : 10
V9, V10 연결 weight : 12
V1, V4 연결 weight : 17
V3, V4 연결 weight : 18
V1, V2 연결 weight : 32
MST by kruskal algorithm : 107
Program ended with exit code: 0

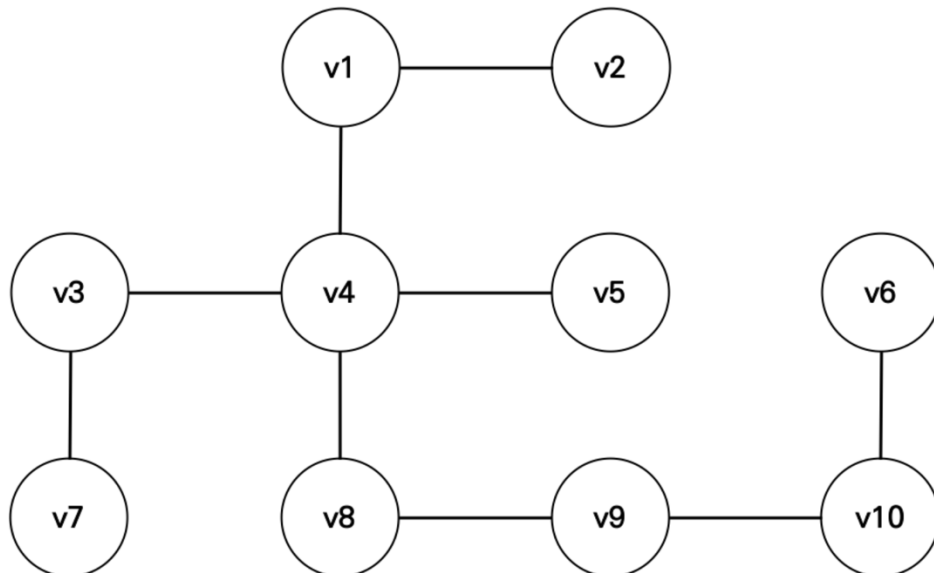
```

All Output ↕

Filter



앞서 작성한 코드를 통해 나온 결과로 그래프를 직접 그려 보면



이러한 형태의 그래프로 나타낼 수 있는데, 이번 경우에는 prim과 Kruskal 모두 같은 모양의 그래프가 나왔지만, 서로 모양을 가진 다른 그래프도 나올 수 있다.

그렇다면 “왜 같은 문제를 2가지 방식으로 해결하는가?”를 생각해보면 간선의 개수를  $E$ 라하고 정점의 개수를  $V$ 라고 할때

Kruskal algorithm은  $O(E \log E)$ 의 시간 복잡도를 가지고

prim algorithm은  $O(E \log V)$ 의 시간복잡도를 가진다.

즉 간선의 개수가 적을 경우에는 Kruskal algorithm이 더 효율적이고 간선의 개수가 많을 경우에는 Prim algorithm이 더 효율적이라는 결론이 도출된다.