숭실대학교 Soongsil University

알고리즘 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
   1.1. 나는 동료의 보고서를 베끼지 않았습니다.
   1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)
3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.
4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	알고리즘(나) 2021
과제명	과제 6
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20163340 강원경 (출석번호 201번)
제출일	2021년 11월 04일

차 례

1장. 문제 1 수행 결과 ---------- 3p

2장. 문제 2 수행 결과 ---------- 5p

3장. 문제 3 수행 결과 ---------- 6p

1장. 문제 1 수행 결과

문제 1은 주어진 그래프에 대해 v1부터 시작하는 최소 신장 트리를 구하는 문제이다.
이를 해결하기 위해 프림 알고리즘을 수행하는 함수를 설계하였고, 대략적인 형태는 다음과 같다.

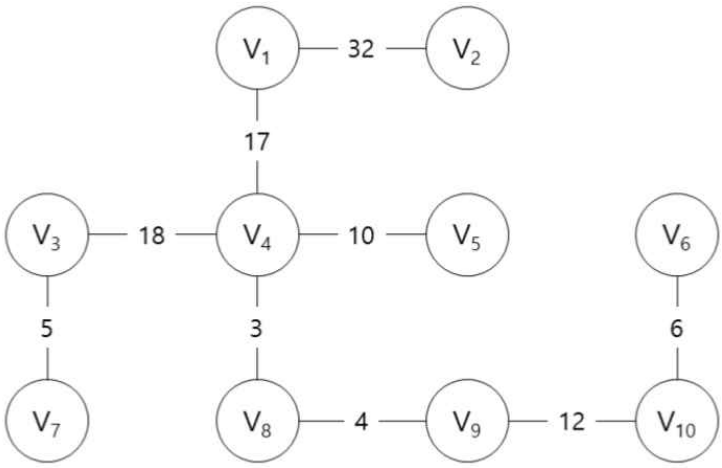
```
// 프림 알고리즘을 수행하는 함수
// start: 시작 정점 번호
// 반환값: 해당 정점 번호에서 시작해 만든 MST의 전체 비용
int prim(int start) {
    int cost[NUM][NUM]; // i->j로 가는 비용이 저장되는 배열. 연결 불가능할 시 INF
    int u,v,minDist,dist[NUM],from[NUM]; //정점의 시작점, 정점의 끝점, 최소거리, 거리의 최대
    int visitCheck[NUM],edgeCount,minCost; //각 정점을 방문했는지 여부, MST의 간선 개수, MST의 전체 비용

    //정점사이의 비용이 저장되는 배열과 집합에서의 최소비용을 저장하는 배열 초기화
    for(int i=0;i<NUM;i++){
        for(int j=0;j<NUM;j++){
            if(W[i][j]==0)
                cost[i][j]=INF;
            else
                cost[i][j]=W[i][j];
                spanning[i][j]=0;
        }
    }

    //출발 정점과 방문했던 정점과 거리를 초기화
    dist[start]=0;
    visitCheck[start]=1;

    for(int i=1;i<NUM;i++) {
        int cur=(start+i)%NUM;
        dist[cur]=cost[start][cur];
        from[cur]=start;
        visitCheck[cur]=0;
    }

    minCost=0; // 전체 비용은 0부터 시작.
    edgeCount=NUM-1; //MST의 간선 개수는 정점개수-1 이므로 이에 해당하는 값으로 초기화.
```

이 함수는 MST를 구성하고 있는 간선의 개수가 정점의 개수 –1이라는 점에서 착안하였고, 한 번씩만 방문한다는 점을 이용해 각 단계에서 최소 비용을 갖는 간선을 잇는 작업을 반복수행하여 최종적으로 MST를 구해내는 함수이다. 위 사진에 누락된 함수의 나머지 부분은 소스코드에 주석과 함께 달아두었다.

위에서 설계한 prim 함수를 이용해 v1을 시작점으로 하여 코드를 수행한 결과는 아래와 같다.

```
----- 문제 1 수행 결과 -----
[생성된 MST의 간선]
v01 - v02 : 32
v01 - v04 : 17
v03 - v04 : 18
v03 - v07 : 5
v04 - v05 : 10
v04 - v08 : 3
v06 - v10 : 6
v08 - v09 : 4
v09 - v10 : 12
생성된 MST의 총 비용 = 107
```

이 결과를 바탕으로 그래프화하여 나타내면 다음과 같다.



여기서 생성된 그래프는 사이클을 만들지 않고 모든 정점을 최소한의 비용으로 잇는다는 조건을 만족하는 MST임을 알 수 있다.

2장. 문제 2 수행 결과

문제 2는 주어진 그래프가 동일하고 프림 알고리즘을 사용한다는 점은 문제 1과 다른 점이 없지만, 시작점이 v1이 아닌 v8이라는 점이 가장 큰 차이점이다.

이 문제를 수행하기 위해 앞서 문제 1에서 작성했던 프림 알고리즘에서 시작점을 지정하는 파라미터 값을 아래와 같이 1이 아닌 8을 주어 수행하였다.

```
// 문제2
startPoint=8; // 탐색 시작 정점 번호: v8
printf("----- 문제 2 수행 결과 -----\n");
totalCost=prim(startPoint-1); // 정점 번호는 1부터 시작하지만, 인덱스는 0부터 시작하므로 (시작 정점 번호-1) 수행.
```

이 문제의 수행 결과는 아래와 같다.

```
----- 문제 2 수행 결과 -----
[생성된 MST의 간선]
v01 - v02 : 32
v01 - v04 : 17
v03 - v04 : 18
v03 - v07 : 5
v04 - v05 : 10
v04 - v08 : 3
v06 - v10 : 6
v08 - v09 : 4
v09 - v10 : 12
생성된 MST의 총 비용 = 107
```

여기서 눈여겨 볼 부분은 시작 정점이 바뀌었음에도 불구하고 MST를 구성하는 전체적인 비용이나 간선들의 구성까지도 모두 동일하다는 점이다. 동일한 비용을 같는 간선이 중복해서 있었다면 탐색에 따른 순서에 의해 약간 달라진 모양의 MST가 출력되었을 수도 있으나, 이 문제에서 입력 데이터로 주어진 그래프는 모든 간선의 비용이 다르기 때문에 이와 같은 결과가 나온 것으로 추측된다.

3장. 문제 3 수행 결과

문제 3은 앞선 문제에서 제공되는 그래프는 동일하지만 탐색 알고리즘을 Prim 이 아닌 Kruskal 알고리즘을 사용한다는 점이 가장 큰 차이점이다.

이를 위해 크루스칼 알고리즘을 수행하는 함수, 그리고 그 함수를 보조하는 여러 개의 보조적인 함수를 만들었다.

```
// 크루스칼 알고리즘 수행 함수.
int kruskal(void){
    edge eArr[NUM*(NUM-1)/2]; // 원래 그래프에서 가중치가 있는 간선들이 모두 저장되는 배열.
    // 최대 간선 개수가 nC2라는 점에서 배열의 크기 착안.
    int cnt=0;

    // 배열 초기화 부분
    p[0]=0;
    for(int i=0;i<NUM-1;i++){
        p[i+1]=i+1;
        for(int j=i+1;j<NUM;j++){ // 원래 그래프의 간선들의 정보를 배열에 저장
            if(W[i][j]!=0){
                edge tmp;
                tmp.start=i;
                tmp.end=j;
                tmp.value=W[i][j];
                eArr[cnt++]=tmp;
            }
        }
    }

    qsort(eArr,cnt,sizeof(edge),cmp); // stdlib.h에 있는 quicksort를 사용하여 정렬.

    int minCost=0;
    printf("[생성된 MST의 간선]\n");
    for(int i=0;i<cnt;i++){
        edge e=eArr[i];
        int x=findParent(eArr[i].start);
        int y=findParent(eArr[i].end);
        if(x!=y){ //x와 y가 속한 그룹이 다르다면 병합한 후 병합의 매개체가 되는 간선 출력
            unite(eArr[i].start, eArr[i].end);
            printf("v%02d - v%02d : %d\n", e.start+1, e.end+1, e.value);
            minCost+=eArr[i].value;
        }
    }
    return minCost;
```

크루스칼 알고리즘은 크게 세 부분으로 나눌 수 있다. 사용할 배열을 초기화하는 부분이 첫 번째, 간선(edge)의 비용에 따라 오름차순(작은 값 -> 큰 값)이 두 번째, 마지막으로 정렬된 간선들을 작은 값부터 선택해가며 간선의 양 끝 정점이 속한 그룹이 다를 경우(최상위 부모노드가 다를 경우) 이를 합쳐나가는 부분이 그것이다.

본인이 작성한 kruskal 함수는 이러한 핵심 3가지를 모두 포함하도록 구성하였고, 정렬 알고리즘은 stdlib.h 에 있는 C언어 라이브러리에서 제공하는 QuickSort를 사용하였다.

```c
// 구조체의 대소비교를 위한 비교 함수
// a, b: 비교할 두 구조체를 가리키는 포인터.
int cmp(const void *a, const void *b){
    edge *pa=(edge*)a;
    edge *pb=(edge*)b;
    return pa->value > pb->value? 1:-1;
}

// 자신이 속한 그룹의 최상위 부모 노드가 무엇인지 검색.
// x: 현재 노드의 번호.
// return 값: 최상위 부모 정점 번호.
int findParent(int x){
    if(x==p[x])
        return x;
    else
        return p[x]=findParent(p[x]);
}

// 두 정점의 최상위 부모 노드를 일치시킴으로써 하나의 그룹으로 병합.
// x, y: 같은 그룹에 속해있는지 판단의 대상이 되는 두 개의 정점 번호.
void unite(int x, int y){
    x=findParent(x);
    y=findParent(y);
    p[x]=y;
}
```

위 3개의 함수는 kruskal()을 수행하는데 보조적으로 사용되는 함수이다. cmp는 qsort를 이용한 간선 구조체의 정렬 부분에서 크기 비교의 지표가 되는 함수고, unite는 findParent를 호출해 두 간선의 양 끝 노드의 부모를 찾고 일치화시켜 하나의 그룹으로 묶는 역할을 수행한다.

이 함수들을 이용한 프로그램을 실행시켰을 때의 결과는 아래와 같다.

```
----- 문제 3 수행 결과 -----
[생성된 MST의 간선]
v04 - v08 : 3
v08 - v09 : 4
v03 - v07 : 5
v06 - v10 : 6
v04 - v05 : 10
v09 - v10 : 12
v01 - v04 : 17
v03 - v04 : 18
v01 - v02 : 32
생성된 MST의 총 비용 = 107
```

kruskal 알고리즘의 특성상 비용이 작은 간선들부터 탐색을 시작하고 이로 인해 결과도 비용이 작은 간선들부터 우선적으로 출력되었으나, 이 결과를 바탕으로 그래프를 그림으로 나타내면 문제 1, 2와 동일한 결과가 나왔다는 것을 알 수 있다.