

문제해결기법 - 201921725 안성현

---

# 최적의 안테나 반경

<2021/09/11>

## ■ 문제 소개 및 접근법

### 1-1] 최적의 안테나 전송 반경 구하기

#### ① 문제

넓은 평원에 설치되어 있는  $n$ 개의 안테나를 연결하여 하나의 네트워크를 구성하고자 한다. 각 안테나의 전송 반경은 동일하게 음이 아닌 실수  $r$ 로 설정한다. 안테나의 범위는 안테나를 중심으로 한 반지름이  $r$ 인 원 내부이다. 두 안테나의 범위가 겹쳐서 공통으로 포함되는 점이 있으면 해당 안테나는 직접 통신할 수 있다. 모든 두 안테나가 통신할 수 있기 위하여 안테나가 가져야 하는 최소 전송 반경을 구하라.

(첫째 줄에 안테나의 개수를 나타내는 양수  $n$ 을 1000이하로 입력하고 이어서  $n$ 개의 줄에 음이 아니고 10억 이하인 안테나의 좌표를 입력하라.)

#### ② 접근법 (소스 간략 설명)

▶ 최적의 반경이 무엇인지 알려면 안테나를 그리고 연결해보면 된다. 안테나 두 개를 그리고 안테나를 연결하는 선의 길이가 10이라고 할 때, 최적의 반경은 5가 된다. 공통인 점이 하나가 되게끔 만들고  $r$ 을 구하는 것이다. 이번에는 세 개를 그리고 연결하는 선의 길이가 10,11,12라고 할 때 최적의 반경은 5.5가 된다. 세 개의 점을 연결하는 두 선을 결정하고 그 중 긴 선의 양끝에 위치한 안테나를 기준으로 공통인 점이 하나가 되게끔 만드는 것이다. 결국  $n$ 개의 점을 연결하는  $n-1$ 개의 선 중에서 가장 긴 값의  $1/2$ 이  $r$ 이 된다. 여기서 중요한 것은  $n-1$ 개의 선을 선택하는 것이다. 위 세 안테나 예시에서 길이가 12인 선을 선택해서  $r$ 이 6이 나왔다면 최적의 반경이라고 볼 수 없다. 즉 연결 가능한 선  $nC2$ 개 중에서  $n-1$ 개를 고를 때 가능한 한 길이가 작은 것들로 구성해야 한다. 또한  $n-1$ 개의 선은 사이클 형태로 존재해서는 안된다. 즉 연결 성분의 개수가 항상 1이어야만 한다. 다행히 이 두 조건을 만족시키는 유용한 방법이 있다. 바로 MST(최소 비용 신장 트리)를 이용하는 것이다. MST는 간선들의 가중치를 합한 값이 최소가 되기 때문에 최적의 선 선택이 보장된다. 예를 들어 정점이  $a, b, c$ 이고 간선이  $(a-b:2, b-c:3, a-c:4)$ 이라면  $(a-b, b-c)$ 로 연결된 가중치 합이 5가 되는 신장 트리를 형성한다. 필자는 MST를 구하는 알고리즘 중 Prim을 채택하였다. 그리고 안테나 각각을 정점으로 생각하고 에지의 가중치는 두 안테나 사이의 거리로 지정하였다. 결국 Prim(g)를 통해 만들어진 '최소 신장 트리'의 에지들 중 가중치가 가장 큰 값의  $1/2$ 이 답이 된다.

## 소스 코드와 실행 결과

### 2-1] 소스 코드와 실행 결과

#### ① 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#pragma warning(disable:4996)

#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 1000
#define INF 1000000001

typedef struct GraphType {
    int n; // 정점의 개수
    double weight[MAX_VERTICES][MAX_VERTICES];
}GraphType;

int selected[MAX_VERTICES]; // <테이블의 s/t열> , 선택되면 s집합에 속함
double w[MAX_VERTICES]; // <테이블의 w열> , 원소: s집합에서 해당 정점까지의 거리

// 그래프 초기화
void init(GraphType* g) {
    int r, c;
    g->n = 0;
    for (r = 0; r < MAX_VERTICES; r++) {
        for (c = 0; c < MAX_VERTICES; c++)
            g->weight[r][c] = INF;
    }
}

// 정점 삽입 연산
void insert_vertex(GraphType* g, int v) {
    if ((g->n) + 1 > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}

// 간선 삽입 연산, v를 u의 인접 리스트에 삽입한다.
void insert_edge(GraphType* g, int start, int end, double weight) {
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
}
```

```

    g->weight[start][end] = weight;
    g->weight[end][start] = weight;
}

// s집합에 속하지 않으면서 최소 w[v] 값을 갖는 정점을 반환
int get_min_vertex(int n) {
    int v, i;
    // 선택되지 않은 정점을 선택 (v정점 지정)
    for (i = 0; i < n; i++) {
        if (!selected[i]) {
            v = i;
            break;
        }
    }
    // v정점의 w값과 나머지 정점들의 w값을 비교하면서 최소 w값을 갖는 정점 구하기
    (선택된 것은 제외)
    for (i = 0; i < n; i++) {
        if (!selected[i] && (w[i] < w[v]))
            v = i;
    }
    return v;
}

// MST 구하는 prim 함수
void prim(GraphType* g, int s) {
    int i, u, v;

    // s집합에 원소가 없으면 a(0)만 인접하다고 가정하고 문제를 풀음
    for (u = 0; u < g->n; u++)
        w[u] = INF;
    w[s] = 0;

    // s집합에 원소가 한 개 이상 있을 때 반복문으로 문제 풀기 (정점을 n개 선택할
    때까지)
    for (i = 0; i < g->n; i++) {
        // w배열에서 가중치가 가장 작은 정점 선택 (s에 속하지 않은 정점들로 한정)
        u = get_min_vertex(g->n);
        selected[u] = TRUE;
        if (w[u] == INF) {
            printf("-> 선택할 정점이 없음\n");
            return;
        }

        // s집합과 인접한 정점들로 w배열 업데이트
        for (v = 0; v < g->n; v++) {
            // u정점을 기준으로 인접한 것을 찾음 <u정점 말고 다른 정점도
            s집합에 속하지만 같은 배열을 업데이트하는 개념이므로, 이미 처리가 된 상태임>
            if (g->weight[u][v] != INF) {
                if (!selected[v] && g->weight[u][v] < w[v]) // v정점이
                s집합에 속하면 처리 x, u-v의 가중치가 기존의 w배열 원소보다 작으면 업데이트
                    w[v] = g->weight[u][v];
            }
        }
    }
}

```

```

    }
}

typedef struct Antenna {
    short code;
    int x, y;
}Antenna;

typedef struct Line {
    short antenna1;
    short antenna2;
    double distance;
}Line;

// 팩토리얼 함수
int factorial(int a, int b) {
    if (a == 0)
        return 1;
    else {
        int result = 1;
        int i;
        for (i = a; i > b; i--)
            result *= i;
        return result;
    }
}

// 조합 개수 구하는 함수
int combi_num(int n, int r) {
    if (r >= 0 && r <= n) {
        int denominator, numerator, answer;
        denominator = factorial(n, n - r);
        numerator = factorial(r, 0);
        answer = denominator / numerator;
        return answer;
    }
}

// 두 점 사이의 거리 구하기
double make_distance(int x1, int x2, int y1, int y2) {
    return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}

// combination작업을 이용해서 llist 제작 (k는 항상 2)
void make_dlist(Antenna* alist, Line* llist, int n) {
    // llist 초기화를 위한 변수 선언
    int x1, x2, y1, y2;
    x1 = x2 = y1 = y2 = 0;
    int l_idx = 0;

    // combination 작업을 위한 변수 선언
    int k = 2;
    int* kset = (int*)malloc(sizeof(int)*(k));
}

```

```

int i;
for (i = 0; i < k; i++)
    kset[i] = i;
int finish = 0;

while (!finish) {
    // llist 초기화
    llist[l_idx].antenna1 = alist[kset[0]].code;
    llist[l_idx].antenna2 = alist[kset[1]].code;
    x1 = alist[kset[0]].x;
    x2 = alist[kset[1]].x;
    y1 = alist[kset[0]].y;
    y2 = alist[kset[1]].y;
    llist[l_idx++].distance = make_distance(x1, x2, y1, y2);

    // kset 업데이트 (combination 작업)
    int idx;
    for (idx = k - 1; kset[idx] == n - k + idx; --idx) {
        if (idx == 0) {
            finish = 1;
            break;
        }
    }
    if (!finish) {
        kset[idx]++;
        for (i = idx + 1; i < k; i++)
            kset[i] = kset[i - 1] + 1;
    }
}

// 배열 중 가장 큰 값 찾기
double find_biggest(double* w, int size) {
    double biggest = w[0];
    int i;
    for (i = 1; i < size; i++) {
        if (biggest < w[i])
            biggest = w[i];
    }
    return biggest;
}

int main() {
    int n = 0; // 안테나 개수
    scanf("%d", &n);

    // 안테나 개수가 1개 이하면 0출력 후 종료
    if (n <= 1) {
        printf("0\n");
        return 0;
    }
}

```

```

int line_num = combi_num(n, 2); // 두 안테나를 잇는 선의 개수

Antenna* alist = (Antenna*)malloc(sizeof(Antenna)*n); // antenna list
Line* llist = (Line*)malloc(sizeof(Line)*line_num); // line list

int i; // iteration variable

// 안테나 정보 삽입
for (i = 0; i < n; i++) {
    alist[i].code = i;
    scanf("%d %d", &(alist[i].x), &(alist[i].y));
}

// line list 제작
make_dlist(alist, llist, n);

// 안테나를 노드로 가지는 그래프 생성
GraphType* g;
g = (GraphType*)malloc(sizeof(GraphType));
init(g);
for (i = 0; i < n; i++)
    insert_vertex(g, i);

// 모든 에지 연결
for (i = 0; i < line_num; i++) {
    int a = llist[i].antenna1;
    int b = llist[i].antenna2;
    double weight = llist[i].distance;
    insert_edge(g, a, b, weight);
}

// prim알고리즘으로 최소 신장 트리 생성
prim(g, 0);

// 최소신장트리의 가중치 중 가장 큰 값의 절반을 출력
printf("%.7lf\n", find_biggest(w, n) / 2);

return 0;
}

```

## ② DEV-C++ 컴파일러 이용 실행 결과

```
C:\Users\tjdu\OneDrive\바탕 화면\test\Wantenna.exe
1000
1268419 193323019
585006099 350308103
822837725 174128995
710495407 303986052
91411268 147313412
988508474 119079732
8922664 531664159
601764134 166249481
450785892 57055898
783319170 519876157
875968220 955899831
539341501 462073831
862219531 779658313
996795644 611492688
266213905 840119132
375857318 677186774
8817100 275887234
587911717 837607619
484930583 743727674
457975742 744418587
599041297 735009335
572398722 151557162
425164882 517120054
751536424 168995888
491884263 699752953
147520433 141598666
692880341 426546337
966604100 153255530
821673085 191350643
```

입력: 안테나 1000개의 좌표 (좌표의 최대 값: 10억)

```
C:\Users\tjdu\OneDrive\바탕 화면\test\Wantenna.exe
946524081 587525201
273339200 212599960
248565182 895216882
362450066 36775561
13080395 410934268
685766327 723285430
61525247 446304954
961457376 961371885
283080382 538113680
184511275 183209270
973549013 889698684
599668158 317558074
31307497 437504465
298869187 168850116
485110411 542485769
444045001 421002890
529487724 306301758
114755392 266390299
184657501 512880221
780771767 659725664

최적의 반경은 27771137.3536661입니다.

-----
Process exited after 11.56 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

출력: '27771137.3536661'이 최적의 전송 반경