

알고리즘 과제 3

Kruskal's Algorithm

Dijkstra's Algorithm

제출 마감일 : 2016.12.01

담당교수 : 최지웅 교수님

소속학부 : 컴퓨터학부

학번 : 20142577

이름 : 홍상원

1. Kruskal's Algorithm

1.1. 크루스칼 알고리즘 개요

- 최소비용 신장트리를 풀기 위한 알고리즘
- 각 정점마다 하나씩 그 정점만 포함하는 V의 서로소 부분집합들을 만든다.
- 가중치가 작은 것부터 차례로 이음선을 검사한다.
- 만약 어떤 이음선이 서로소 부분집합들에 있는 두 정점을 연결하면, 그 이음선을 추가하고, 두 부분집합을 합친다.
- 부분집합들이 모두 집합 하나로 합쳐질 때까지 이 과정을 반복한다.

1.2. 구현내용

- 문제: 최소비용 신장트리를 구한다.
- 입력: 정수 $n(n \geq 2)$, 양의 정수 m , n 개의 정점과 m 개의 이음선을 가진 연결된 가중치포함 비방향 그래프. 이 그래프는 가중치와 함께 이음선이 포함된 E로 표현된다.
- 출력: 최소비용 신장 트리에서 이음선의 집합 F
- 소스코드: kruskal.cpp

```
#include <iostream>
#include <queue>
#define N 5
using namespace std;

//이음선
struct edge {
    int v1;
    int v2;
};

//이음선 집합
struct set_of_edge {
    edge e;
    int weight;
};

//disjoint set data structure 2 시작
struct nodetype {
    int parent;
    int depth;
};

typedef nodetype universe[100];
universe U;
```

```

void makeset(int i) {
    U[i].parent = i;
    U[i].depth = 0;
}

int find(int i) {
    int j;
    j = i;
    while(U[j].parent != j)
        j = U[j].parent;
    return j;
}

void merge(int p, int q) {
    if(U[p].depth == U[q].depth) {//트리의 깊이가 증가한다.
        U[p].depth = U[p].depth + 1;
        U[q].parent = p;
    }
    else if(U[p].depth < U[q].depth)//깊이가 작은 트리를 자식노드로 만든다
        U[p].parent = q;
    else
        U[q].parent = p;
}

bool equal(int p, int q) {
    if(p == q)
        return true;
    else
        return false;
}

void initial(int n) {
    int i;
    for(i = 1; i <= n; i++)
        makeset(i);
}

//disjoint set data structure 2 끝

//우선순위큐에서 비교 기준(weight) 설정하는 구조체
struct compare {
    bool operator()(const set_of_edge& l, const set_of_edge& r)

```

```

{
    return l.weight > r.weight;
}
};

//F에 저장된 원소개수 계산
int size(set_of_edge F[], int n) {
    int cnt = 0;
    for(int i = 1; i <= n-1; i++) {
        if(F[i].weight == 0) {
            break;
        }
        cnt++;
    }
    return cnt++;
}

//크루스칼 알고리즘
void kruskal(int n, int m, set_of_edge E[], set_of_edge F[]) {
    int i, j;
    int p, q;
    edge e;

    //E에 속한 m 개의 이음선을 가중치가 작은 것부터 차례대로 정렬
    priority_queue<set_of_edge, vector<set_of_edge>, compare > pq;
    for(int s = 0; s < m; s++)
        pq.push(E[s]);

    //n 개의 서로소 부분집합을 초기화
    initial(n);
    int s = 1;
    while(size(F, n) < n - 1) {
        e = pq.top(); // 아직 고려되지 않은 이음선 중에서 가중치가 최소인
        이음선
        pq.pop();

        //e로 연결된 정점의 인덱스
        i = e.v1;
        j = e.v2;
        p = find(i);
        q = find(j);
        if(!equal(p, q)) {

```

```

        merge(p, q);
        set_of_edge res;
        res.e = e;
        F[s++] = res;//e 를 F 에 추가
    }
}

}

```

```

int main(void) {
    set_of_edge E[] = {
        {{1, 2}, 1},
        {{1, 3}, 3},
        {{2, 3}, 3},
        {{2, 4}, 6},
        {{3, 4}, 4},
        {{3, 5}, 2},
        {{4, 5}, 5}
    };

    int m = sizeof(E) / sizeof(E[0]);
    set_of_edge F[N] = {0};

    kruskal(N + 1, m, E, F);
    for(int i = 1; i <= N - 1; i++)
        cout << "(" << F[i].e.v1 << " - " << F[i].e.v2 << ")" << endl;
    return 0;
}

```

1.3. 결과

- 책의 그림을 입력으로 넣었을 때의 이음선 출력 결과

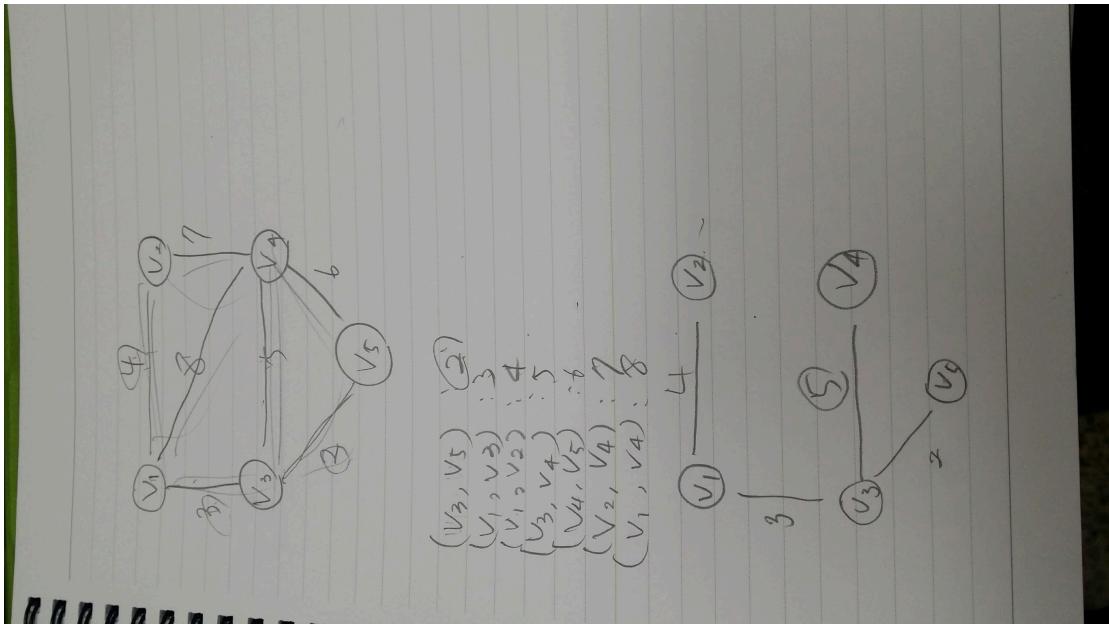
```

<1 - 2>
<3 - 5>
<1 - 3>
<3 - 4>

```

- 새로 만든 그래프를 입력으로 넣었을 때의 이음선 출력 결과와 손 계산 비교

Mongsung 1
<3 - 5>
<1 - 3>
<1 - 2>
<3 - 4>



2. Dijkstra's Algorithm

2.1. 다익스트라 알고리즘 개요

- 가중치가 있는 방향성 그래프에서 한 특정 정점에서 다른 모든 정점으로 가는 최단경로를 구하는 문제

2.2. 구현내용

- 문제: 가중치포함 방향그래프에서 v1에서 다른 모든 정점으로 가는 최단경로를 구한다.
- 입력: 정수 n($n \geq 2$)과 정점이 n개 있는 연결된 가중치포함 방향 그래프. 이 그래프는 2차원 배열 W로 표현되며, 행과 열의 인덱스는 각각 1부터 n까지이다. 여기서 $W[i][j]$ 는 i번째 정점에서 j번째 정점을 잇는 이음선상의 가중치가 된다.
- 출력: 최단경로상에 놓여있는 이음선의 집합 F
- 소스코드 : dijkstra.cpp

```
#include <iostream>
#define N 5
#define INF 1000//무한대 표현
using namespace std;

//이음선
struct edge {
    int v1;
    int v2;
};

//이음선 집합
struct set_of_edge {
    edge e;
    int weight;
};

//다익스트라 알고리즘
void dijkstra(int n, int (*W)[N + 1], set_of_edge *F) {
    int i, vnear, min;
    edge e;
    int touch[N + 1];
    int length[N + 1];

    //각 정점에 대해서 v1에서 출발하는 현재 최단경로의 마지막 정점을 v1으로 초기화한다.
    //그 경로의 길이는 v1에서의 이음선 상의 가중치를 초기화한다.
    for(i = 2; i <= n; i++) {
        touch[i] = 1;
```

```

        length[i] = W[1][i];
    }

    int repeat = 1;
    int s = 1;
    //n-1 개 정점을 모두 Y에 추가
    while(repeat <= n - 1) { //최단 경로를 갖는지 각 정점을 점검
        min = INF;
        for(i = 2; i <= n; i++) {
            if(0 <= length[i] && length[i] < min) {
                min = length[i];
                vnear = i;
            }
        }

        e.v1 = touch[vnear];
        e.v2 = vnear;
        set_of_edge res;
        res.e = e;
        F[s++] = res;

        for(i = 2; i <= n; i++) {
            if(length[vnear] + W[vnear][i] < length[i]) {
                length[i] = length[vnear] + W[vnear][i];
                touch[i] = vnear;
            }
        }
        length[vnear] = -1;
        repeat++;
    }

}

int main(void) {
    int W[N + 1][N + 1] = {
        {-1,-1,-1,-1,-1,-1},
        {-1, 0, 7, 4, 6, 1},
        {-1, INF, 0, INF, INF, INF},
        {-1, INF, 2, 0, 5, INF},
        {-1, INF, 3, INF, 0, INF},
        {-1, INF, INF, INF, 1, 0}
    };
}

```

```

set_of_edge F[N] = {0};

dijkstra(N, W, F);

for(int i = 1; i <= N - 1; i++)
    cout << "<" << F[i].e.v1 << " -> " << F[i].e.v2 << ">" << endl;

return 0;
}

```

2.3. 결과

- 책의 그래프를 입력으로 넣었을 때의 이음선 출력 결과

```

<1 -> 5>
<5 -> 4>
<1 -> 3>
<4 -> 2>

```

- 새로 만든 그래프를 입력으로 넣었을 때의 이음선 출력 결과와 손 계산 비교

```

<1 -> 5>
<1 -> 4>
<1 -> 3>
<1 -> 2>

```

