8.1 소개

욕심쟁이 방법 ⇒ 그래프 최적화 문제
↳ 전체 비용 최소화 / 전체 이득 최대화 ⇒ 선택 필요

욕심쟁이 방법 (Greedy method) → 각 개별 선택에 따른 계산비용이 단기 기준에서 최선이 되도록 선택
↳ 한번 정한 이후에 변경 X

"모든 문제에서 최적해를 찾을수는 없다"

⇒ 몇몇 문제에서는 최적임을 보장

8.2 Prim의 최소 스패닝 트리 알고리즘

(연결) (가중치) (무향) 그래프 에서의 최소 스패닝 트리

↳ 연결 되어 있지 않은 그래프에서 각 연결 성분의 최소 스패닝 트리를 찾는 문제와 동일
↳ 선형 시간 내에 찾아짐

최소 스패닝 트리 ⇒ 가중치가 있는 무향 그래프에서 유용 $G=(V,E,w)$

· 최소 스패닝 트리 정의와 예제

〈정의 8.1〉 최소 스패닝 트리

무향연결그래프 $G=(V,E)$의 스패닝 트리는 무향 트리로 G의 모든 정점을 포함하는 G의 부그래프.

가중치 그래프 $G=(V,E,w)$ 에서 부그래프의 가중치는 부그래프에 속한 가중치의 합

가중치 그래프에서 최소 스패닝 트리 (Minimum Spanning Tree; MST)는 가중치가 최소인 스패닝 트리

최소 스패닝 트리를 구해야 하는 상황이 많음
↳ 그래프의 모든 정점을 지나는 효율적인 경로 찾기
⇒ 라우팅 알고리즘 등에서 중요.

가중치 그래프는 두 개 이상의 최소 스패닝 트리를 가질 수 있음 (여러 개가 가능하단 소리)

· 알고리즘의 개요.

연결되어 있는 무향 트리 → 어떤 정점이라도 루트가 될 수 있음 ⇒ 한 정점으로부터 에지 키워나가기

⇒ 표준 탐색 기법인 깊이 우선 탐색이나 너비 우선 탐색 시도 → 이거 적절하지 않음 (기본 탐색만으로 불가)

자연스러운 아이디어 → 욕심쟁이 방법 → 적은 단기 비용을 모두 합하면 전체 비용이 작을거라고 생각

⇒ 단기적으로 최저 비용 계속 선택

현재의 트리에 에지를 추가하여 키워나가는 방법 사용 → 현재의 트리에 꼭짓점 하나가 붙어 있는 에지 중에 가중치가 최소인 에지 선택

Prim의 알고리즘 : 임의의 출발 정점 선택 → 지금까지 만든 트리로부터 새로운 정점 하나와 에지 선택
↳ 기존의 트리에 새로운 정점 연결

알고리즘이 진행되는 동안의 정점

① 트리 정점 : 지금까지 만든 트리에 속하는 정점

② 주변 정점 : 트리에는 속하지 않고, 트리에 속한 정점과 인접한 정점

③ 미확인 정점 : 나머지 모든 정점

알고리즘에서의 핵심 단계 : 주변 정점에서 정점과 에지의 선택 ← 여기서 초점은 에지의 선택

〈알고리즘 개요〉

PrimeMST (G, n)

이거 최소 스패닝 트리를 만들 수 있을까?

모든 정점을 미확인으로 초기화

임의의 정점 S를 출발 정점으로 선택. S를 트리 정점으로 분류

S에 인접한 모든 정점을 주변 정점으로 지정

주변 정점이 존재하는 동안

트리 정점 t과 주변 정점 v 사이에 최소 가중치를 가진 에지 선택

v를 트리 정점으로 지정. 에지 tv를 트리에 추가

v에 인접한 모든 미확인 정점을 주변 정점으로 분류

• 최소 스패닝 트리 성질

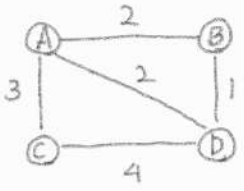
최소 스패닝 트리 → 여러 개 가질 수 있음 → 하나를 다른 것으로 단계별로 전환이 가능

〈정의 8.2〉 최소 스패닝 트리 성질

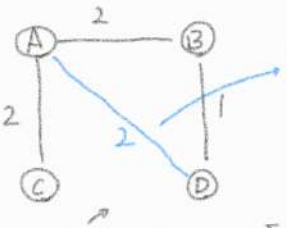
연결된 가중치 그래프 $G=(V,E,w)$에서 T가 G의 스패닝 트리. T에 포함되지 않은 임의의 에지 uv에 대해, uv를 T에 추가하면 T에 사이클 하나가 생성되고, uv가 그 사이클에서 최대 가중치 에지. 이 때 트리 T가 최소 스패닝 트리 (MST) 성질을 가진다.

무향 트리 → 사이클 X

ex)

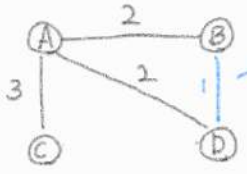


원래 그래프



이거 추가하면 사이클 생성
(사이클 내의 모든 에지가 2이하)

스패닝 트리 ⇒ 둘 다 스패닝 트리이지만 위의 것만 최소 스패닝 트리 성질을 가지고 있음.



이걸 추가해도 사이클 형성
그러나 모든 에지가 1 이하이진 않음

보조 정리 8.1

연결된 가중치 그래프 $G=(V,E,w)$ 에서 T_1, T_2 모두가 MST 성질을 가진다면 T_1, T_2의 총 가중치는 같다.

〈정리 8.2〉

연결된 가중치 그래프 $G=(V,E,w)$ 에서 트리 T가 최소 스패닝 트리일 필요 충분 조건은 T가 MST 성질을 갖는 것.

• 우선 순위 큐로 두번 정점 관리하기

알고리즘이 루프를 돌면서 두번 정점이 새로 생길수도 있고 다음에 선택한 에지의 정점이 변할수도 있음

↳ 트리 정점과 두번 정점 간의 모든 에지 고려할 필요는 없음

⇒ 각 두번 정점에 대해서 트리로부터 들어오는 에지 중 가중치가 최소인 것만 관리하면 됨 → 후보에지 (candidate edge)

우선순위 큐 ADT 이용한 알고리즘

```
Prim MST (G, n)
    우선 순위 큐 PQ가 비어있도록 초기화
    임의의 출발 정점 S 선택
    그것의 후보 에지를 (-1, S, 0)으로 두고 insert(PQ, S, 0)을 호출
    while (PQ가 비어있지 않은 동안)
        u = getmin(PQ)    deletemin(PQ)
        u의 후보에지를 트리에 추가
        updateFringe(PQ, G, u)
```

updateFringe(PQ, G, u)

u에 인접한 모든 정점 w에 대해서 netwgt = W(u,w)라고 두고

if(w가 미확인 정점이면)

그것의 후보메시지를 (u, w, netwgt)라고 둔다.

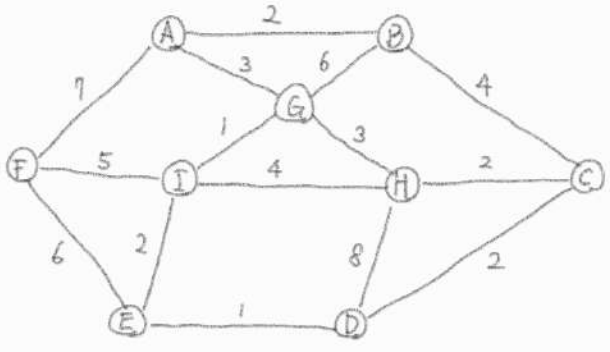
insert(PQ, w, netwgt)

else if (netwgt < w의 fringewgt 이면)

그것의 후보메시지를 (u, w, netwgt)라고 수정

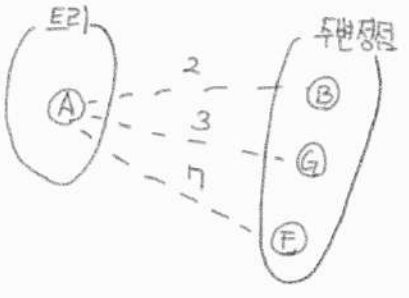
decrease key(PQ, w, netwgt)

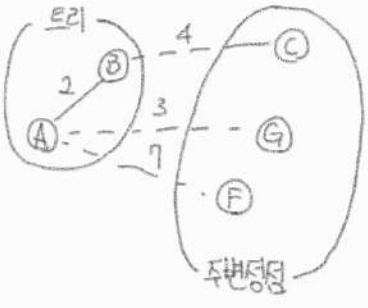
<예시>



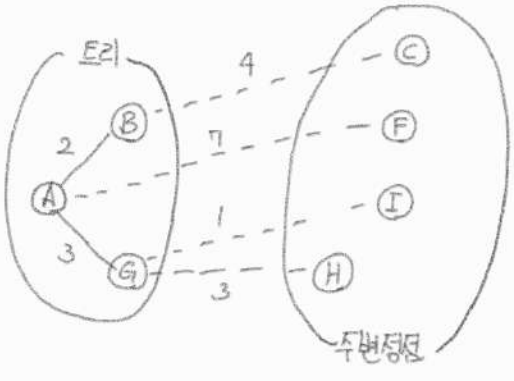
원래 그래프

(1) A에서 시작

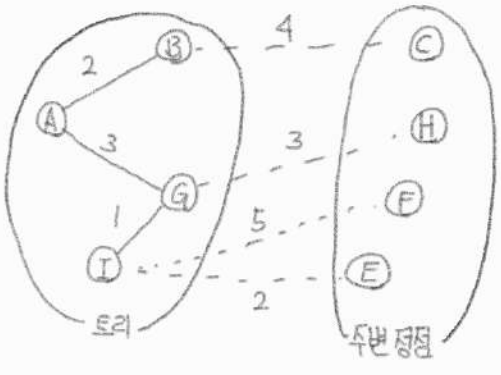




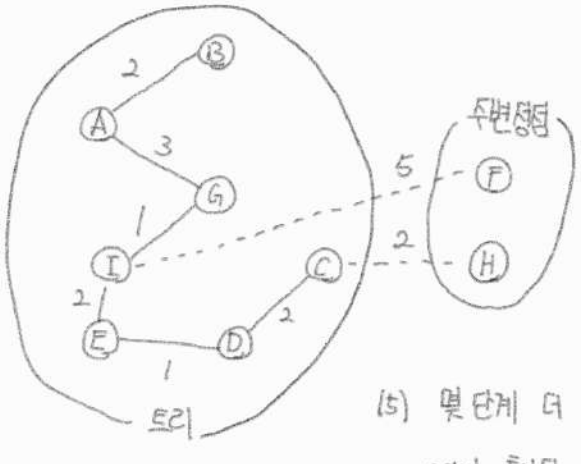
(2) BG가 고려는 되지만 후보로 AG 고려는 X



(3) AG가 선택되고 난 후 주변정점과 후보 갱신



(4) AF는 새로운 후보 IF로 교체



(5) 몇 단계 더 지난 후 : 두 에지는 트리에 추가됨

• 구현

알고리즘의 주요 자료 구조: 인덱스가 정점 번호인 배열 status, fringeWgt, parent

status: 정점의 분류 (미확인, 주변, 트리 정점을 나타내는 상수 존재) → 너비 우선 탐색에서의 색깔과 동일

↳ FIFO 큐 대신에 우선 순위 큐에 기반을 둔 탐색 ⇒ 최선 우선 탐색 (best-first search)

〈알고리즘 8.1〉 Prim 최소 스패닝 트리

input: 무향 가중치 그래프 G=(V,E,W), 정점의 수 n, 출발 정점 S

output: 최소 스패닝 트리를 내향 트리로 저장한 배열 parent

parent(v)와 v 사이의 가중치를 담은 배열 fringeWgt

주의: 배열 status[1], ··· , status(n)은 정점의 현재 탐색 상태를 나타냄

```
Void PrimMST (G, int n, int s, int [] parent, float[] fringeWgt)
    int[] status = new int [n+1]
    MinPQ pq = create(n, status, parent, fringeWgt)

    insert (pq, s, -1, 0)
    while (isEmpty (pq) == false)
        int v = getMin(pq)
        deleteMin(pq)
        updateFringe (pq, G, v)
    return

// 리스트 adjInfo에 있는 정점으로 가는 더 나은 연결이 있는지 알아보고 만약 그렇다면
   decreaseKey 실행
// 새로운 연결 위해 정점 삽입
Void updateFringe (MinPQ pq, EdgeList adjInfo, int v)
    EdgeList remAdj
    remAdj = adjInfo
    while (remAdj ≠ null)
        EdgeInfo wInfo = first(remAdj)
```

```
int w = wInfo.to (head)
float netwgt = wInfo.weight
if (PQ.status(w) = unseen)
    insert (PQ, w, v, netwgt)
else if (PQ.status(w) = fringe)
    if (netwgt < getPriority(PQ, w))
        decreaseKey(PQ, w, v, netwgt)
remAdj = rest(remAdj)
return
```

8.3 단일 소스 최단경로

두 도시 간의 최적 경로 찾기 → 직항이 나은가 경유하는게 나은가? → 같은 알고리즘 사용

주어진 두 정점쌍 s, t 사이의 최소 가중치 경로를 찾는 문제

- 최악의 경우에 s와 s로부터 도달가능한 모든 정점 사이에 최소 가중 경로를 찾는 것보다 어려움
  - 이 문제가 바로 단일 소스 최단 경로 문제 (single-source shortest path problem)

주어진 소스로부터 다른 모든 정점까지 가중치 그래프나 무향 그래프에서 최소 가중치 경로를 찾는 문제 고려

경로의 가중치 : 경로 상에 있는 에지들의 가중치 합 (가중치가 거리일 때, 최단 경로 라고 표현)

• 최단 경로의 성질

분할 정복의 방법으로 풀 수 있을까? → 떨어있는 두 정점 사이의 최단 경로에서 그 보다는 가까운 거리에 있는 정점 간의 최단 경로를 이용할 수 있을까?

〈보조정리 8.5〉 최단 경로 성질

가중치 그래프 G에서 x에서 z까지의 최단경로가 x에서 y까지 경로 P와 y에서 z까지의 경로 Q로 구성되어 있으면 P는 x에서 y까지의 최단경로이고, Q는 y에서 z까지 최단경로이다.

• Dijkstra의 최단 경로 알고리즘

〈정의 8.3〉

가중치 그래프 $G=(V,E,W)$ 에서 k개의 에지 $xv_1, v_1v_2, \cdots, v_{k-1}y$ 로 이루어진 경로를 P라고 할 때,
P의 가중치는 $W(P)$로 나타내고, 가중치 $w(xv_1), w(v_1v_2), \cdots, w(v_{k-1}y)$의 합이다.
만약 $x=y$ 이면 이를 x에서 y까지의 공백 경로라 하고, 그 가중치는 0이다
만약 x와 y 사이의 어떤 경로도 가중치가 $w(P)$보다 작지 않으면 P를 최단 경로(shortest path) 혹은
최소 가중치 경로 라고 부른다

최소 스패닝 트리와는 다름 → 최소 스패닝 트리는 트리 전체의 가중치가 최소
↳ 단일 소스는 s에서 임의의 t까지 최소

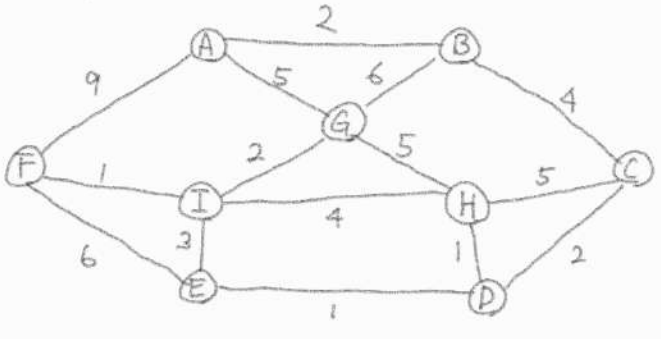
Dijkstra의 알고리즘은 s에서 다른 정점까지의 최단 경로를 s로부터 거리가 증가하는 순으로 탐색
↳ 한 정점 s에서 시작하여 새로운 정점으로 나가는 에지 선택해서 가지뻗음 ⇒ 최단 경로 트리

Dijkstra 알고리즘 : 욕심쟁이 방식 (s에서 가장 가까운 것을 선택)

트리 정점
주변 정점 으로 분류
미확인 정점

Prim 알고리즘처럼 각 주변 정점에 대해서 단 하나의 후보 에지 관리
각 주변 정점 v에 대해 최소 하나의 트리 정점 v가 존재 ⇒ vz는 G의 에지
각각의 v에 대해 트리에는 s에서 v까지 (유일한) 경로가 존재 ($s=v$일수도 있음) → $d(s,v)$는 가중치
이 경로에 에지 vz를 추가하면 s에서 z까지의 경로가 되고 그 가중치는 $d(s,v)+w(vz)$
z의 후보 에지 : 지금까지 만들어진 모든 v에 대해 $d(s,v)+w(vz)$가 최소가 되는 에지

〈예제〉 최단 경로 트리 키우기

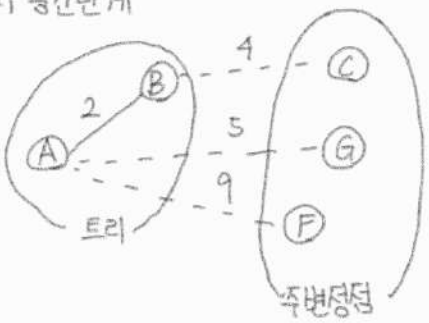


원래 그래프

초기 트리 : A만 존재

$d(A,A)=0$

(2) 중간단계



$d(A,B) + w(B,C) = 6$

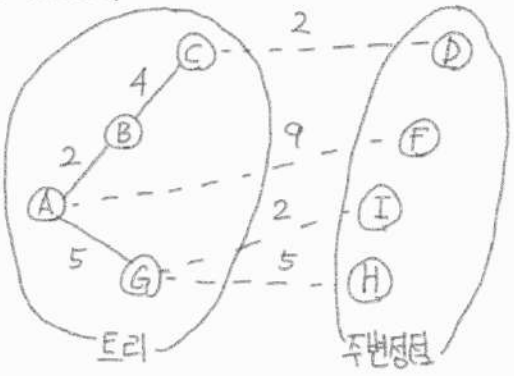
$d(A,A) + w(A,G) = 5$ → 다음 단계에서 AG 선택

$d(A,A) + w(A,F) = 9$

↳ A와 B와 연결된 모든 정점은 주변 정점이 됨

BG는 후보 에지가 아님에 주의!

(3) 중간단계



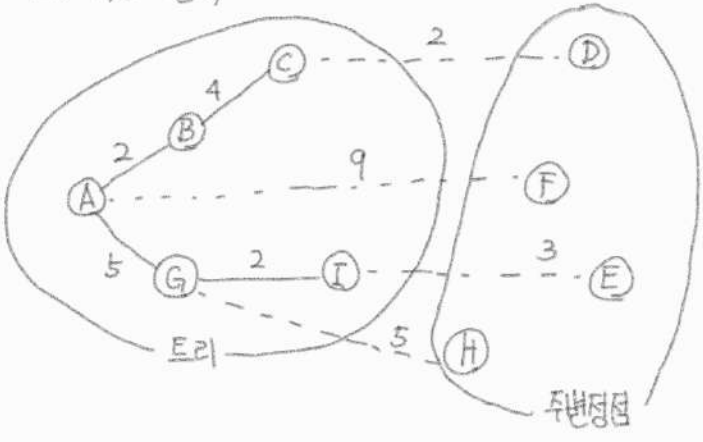
$d(A,C) + w(C,D) = 8$

$d(A,A) + w(A,F) = 9$

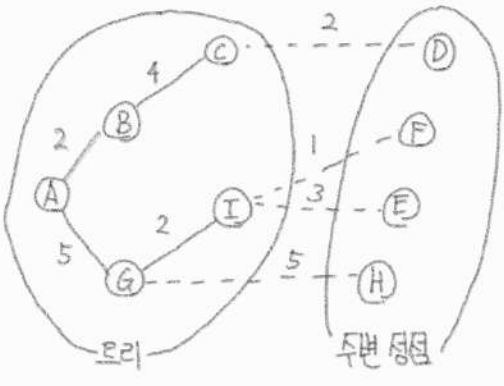
$d(A,G) + w(G,I) = 7$ → 다음 단계로 GI 선택

$d(A,G) + w(G,H) = 10$

(4) GI 선택



(5) 후보에지로서 AF는 IF로 교체



Dijkstra 알고리즘 일반적인 구조

dijkstraSSSP(G, n)

모든 정점을 미확인 정점으로 초기화

정해진 소스 S로부터 트리를 시작

$d(s,s) = 0$ 으로 정의

S에 인접한 모든 정점을 주변 정점으로 재분류

주변 정점이 존재하는 동안,

트리정점 t와 주변정점 v 사이에 $d(s,t) + w(tv)$가 최소인 에지 선택

V를 트리 정점으로 재분류, 에지 tv 트리에 추가

$d(s,v) = d(s,t) + w(tv)$

v에 인접한 모든 미확인 정점을 주변 정점으로 재분류

→ 후보에지 yz에 대해 $d(s,y) + w(yz)$는 반복사용될 수 있으니 따로 저장

〈정리 8.6〉

$G = (V, E, w)$가 가중치 그래프, $V' \subseteq V$, $s \in V'$

각 $y \in V'$에 대해 $d(s,y)$는 G에서 S로부터 y까지의 최단거리라고 가정

만약 V'에 속한 정점 y와 $V - V'$에 속한 정점 z를 잇는 모든 에지들 중에서 $d(s,y) + w(yz)$가 최소인 에지를 yz라 하면 S부터 y까지의 최단 경로에 yz를 붙여 만든 경로는 S부터 z까지의 최단경로

〈정리 8.7〉

가중치가 음이 아닌 유향 가중치 그래프 G와 소스 S가 주어질 때, Dijkstra의 알고리즘은 S에서 출발하여 S로 부터 도달 가능한 모든 정점까지의 최단 거리를 계산한다.

• 구현

Prim의 알고리즘과 같이 우선순위큐 ADT 사용가능

〈알고리즘 8.2〉 Dijkstra 단일 소스 최단 경로

input: $G = (V, E, w)$, $n = |V|$, S(소스)

output: 최단 경로를 내향 트리로 저장한 배열 parent, S부터 V까지 최단경로배열 fringeWgt

```
Void ShortestPaths(G, int n, int s, int[] parent, float[] fringeWgt)
    int[] status = new int[n+1]
    MinPQ pq = create(n, status, parent, fringeWgt)

    insert(pq, s, -1, 0)
    while (isEmpty(pq) = false)
        int v = getMin(pq)
        deleteMin(pq)
        updateFringe(pq, G, v)
    return
```

↗ G(에지)

// 리스트 adjInfo에 더 나은 연결이 있는지 알아보고 만약 그렇다면 decreaseKey 실행

```
Void updateFringe(MinPQ pq, adjInfo, v)
    float myDist = pq.fringeWgt(v)
    remAdj = v의 에지들
    while (remAdj ≠ null)
        wInfo = first(remAdj)
        int w = wInfo.to
        newdist = mydist + wInfo.wgt
        if(pq.status(w) = unseen)
            insert(pq, w, v, newdist)
        else if(pq.status(w) = fringe)
            if(newdist < getPriority(pq, w))
                decreaseKey(pq, w, v, newdist)
        remAdj = rest(remAdj)
    return
```

8.4 Kruskal의 최소 스패닝 트리 알고리즘

그래프 G의 최소 스패닝 트리 ⇒ Prim 알고리즘

↳ Kruskal의 알고리즘은 Prim 보다 더 greedy 한 방법

• 알고리즘

```
KruskalMST(G, n)
    R = E   // R은 남아있는 에지
    F = ∅   // F는 포리스트 에지
    While(R이 공집합이 아닐 때)
        R로부터 가장 가벼운 에지 uw 제거
        if ( uw가 F에서 사이클을 만들지 않으면)
            uw를 F에 추가
    return F.
```

〈정의 8.4〉 스패닝 트리 컬렉션

G=(V,E,w)인 무향 그래프. G의 스패닝 트리 컬렉션은 트리의 집합으로서 G의 연결 성분 하나에 하나씩 존재. 각 트리는 그것이 속한 연결성분의 스패닝 트리. 최소 스패닝 트리 컬렉션은 총합이 최소.

F가 임의의 포리스트 (사이클이 없는 무향 그래프) e= uw가 F에 속하지 않는 에지

e와 F에 속한 에지들로 이루어진 사이클이 존재할 필요충분조건은 u와 w가 F의 같은 연결성분

〈알고리즘 8.3〉 Kruskal 최소 스패닝 트리

input: G=(V,E,w), n=|V|, m=|E|

주의! 알고리즘의 Sets는 동치관계 ≡에 대응.

```
KruskalMST(G, n, F)
    int Count.
    가중치를 우선 순위로 하여 G의 에지로 최소 우선 순위 큐 생성
    G의 각 정점이 하나의 집합이 되도록 Union-Find 구조 Sets 초기화
    F = ∅
```

```
while ( isEmpty(pq) = false)
    uvedge = getMin(pq)
    deleteMin(pq)
    int uset = find(sets, uvedge.from)
    int wset = find(sets, uvedge.to)
    if (uset ≠ wset)
        uvedge를 T에 추가
        union(sets, uset, wset)
return.
```

↳ 에지들을 하나씩 검사하는 방법.