# 자료구조설계

## - 실습04 : Kruskal 알고리즘 구현 -

| | |
|---|---|
| 제 출 일 | 2016.10.16. |
| 분 반 | 02 |
| 담당교수 | 박정희 |
| 학 과 | 컴퓨터공학과 |
| 학 번 | 201302423 |
| 이 름 | 신종욱 |

# 1. MST Class 필드 & 메소드 설명

필드값들 설명:
String[] TV;// 정점들을 저장하기위한 배열
int[][] E;// 정점들의 연결가중치를 저장하기위한 배열
int parent[];// union이나 find를 할때 쓰는 배열
Node[] node;//heap에 들어갈때 가지고있어야할 정보들을 저장한다
MinHeap H;//minheap을 구현하기위해 선언

**MST Class 메소드 설명 :**

MST(String[] A , int[][] B)
정점들을 저장하는 A와 Edge들의 가중치를 저장하는 A와 B를 받고 Kruskal할 때 필요한 정보들을 초기화해주는 메소드이다.

int collapsingfind(int i)
TV배열의 I번째 인덱스의 문자의 ROOT의 값을 return한다.

(그룹의 크기를 음수로 return한다)

weightedunion(int i,int j)
두 개의 그룹들을 연결하는데 크기가 큰거에 작은 크기를 연결한다.

Kruskal()
미방문 배열중에 가중치를 저장한 배열값들중에 가장 작은 것을 찾은 후

그 둘을 포함하는 그룹이 collapsingfind을 이용해서 같은지 확인후에

다르다면 둘을 weightedunion을 하고 출력을 완료한후 방문했다고 ture로 체크한다.

(cycle유무 판단하기위해서 확인)

만약 같아도 다시 방문하지 않기위해 방문했다고 체크해야한다.

class Node
heap에 들어갈때 가지고있어야할 정점과 가중치를 저장할수있게 하였다.

class MinHeap 메소드 설명 :

MinHeap(int maxsize)
heap을 구성할 때 최대 사이즈를 넣어 그에 맞게 배열을 셋팅하는 생성자이다.

int parent(int index)
해당 인덱스의 부모 인덱스를 리턴하는 메소드

int leftChild(int index)
해당 인덱스의 Leftchild의 인덱스를 리턴하는 메소드

int rightChild(int index)
해당 인덱스위 Rightchild의 인덱스를 리턴하는 메소드

boolean isLeaf(int index)
해당 인덱스가 리프노드인지 확인후 boolean값으로 리턴하는 메소드

swap(int Aindex, int Bindex)
A인덱스와 B인덱스를 서로 바꾸는 메소드

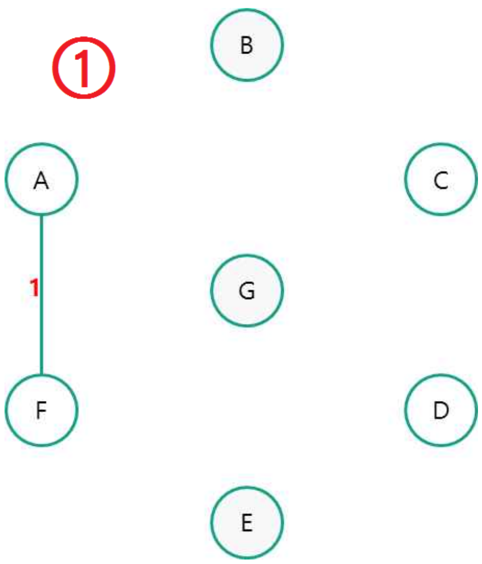
minHeapify(int index)
만약 부모의 비용값이 자식의 비용값보다 클 경우 minheap이 깨져서 자리를 바꾸고 바꾸고 난뒤에도 minheap을 만족시키는지 확인하는 메소드
insert(Node A)
MinHeap에 노드가 들어올 때 맨밑 리프에 넣은후 Minheap일 만족하는 위치에 찾아가도록하는 메소드
Node remove()
루트값 즉 제일 비용값이 낮은 Node를 삭제하며 반환하는 메소드
heap가 깨질수도 있으니 heapify를 해줘야한다

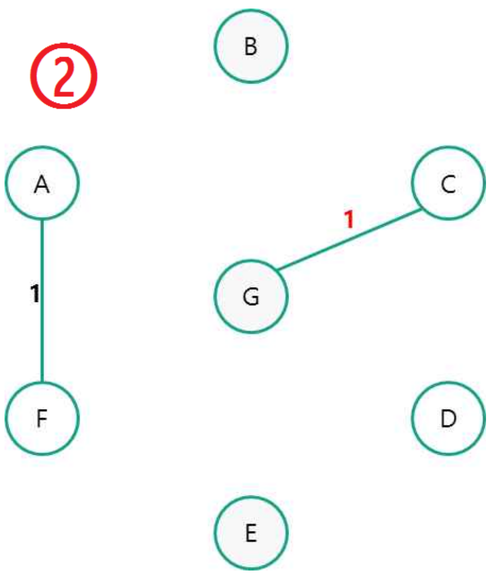
# 2. Kruskal 알고리즘 실행 결과

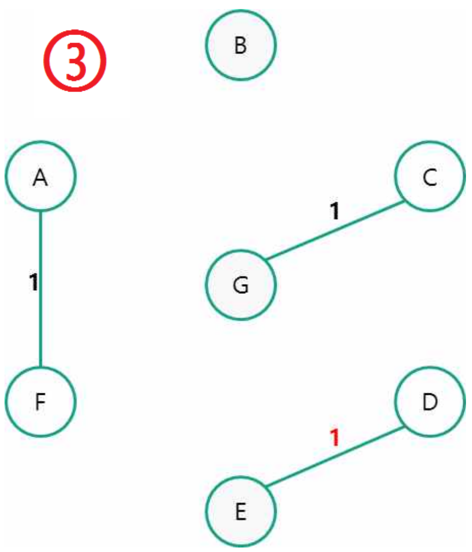
```
Problems @ Javadoc Declaration Console Debug
<terminated> TestMSTClass [Java Application] C:₩Program Files₩Java₩jre1.8.0_101₩bin₩javaw.exe (2016. 10. 13. 오후 7:33:55)
선택된 간선 : A--> F / weight = 1 추가
선택된 간선 : C--> G / weight = 1 추가
선택된 간선 : D--> E / weight = 1 추가
선택된 간선 : A--> G / weight = 2 추가
선택된 간선 : B--> C / weight = 2 추가
선택된 간선 : D--> G / weight = 2 추가
비용 합계 : 9
```

# 3. Kruskal 알고리즘 실행 결과 그래프

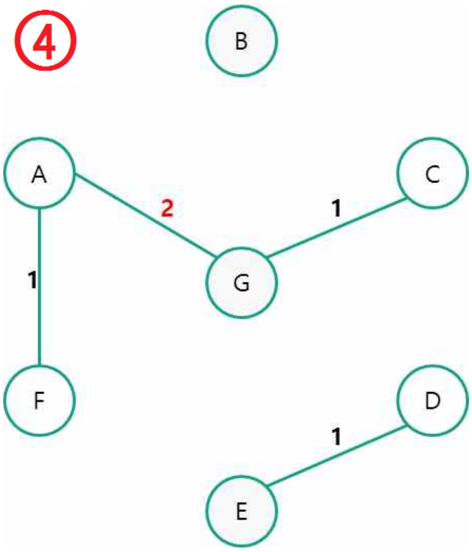
1)가중치중 가장작은값인 1들을 찾아서 연결한다(도중에 사이클은 발생하지않음)

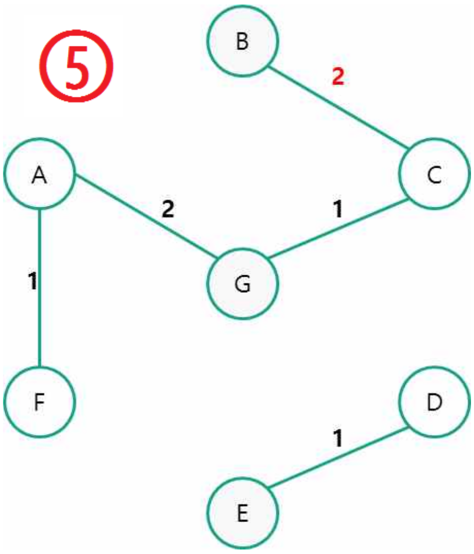


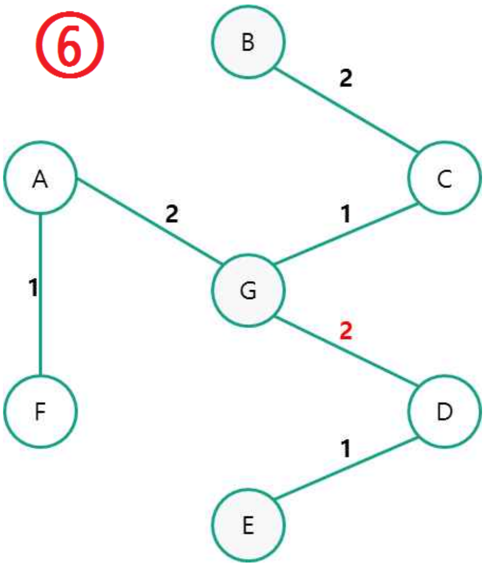



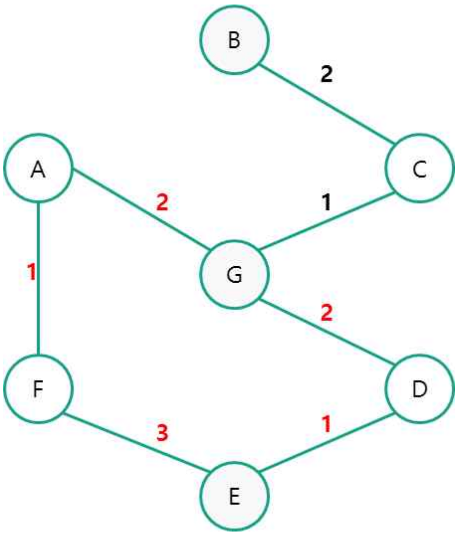
1)가중치중 가장작은값인 2들을 찾아서 연결한다.(도중에 사이클 발생하지않음)



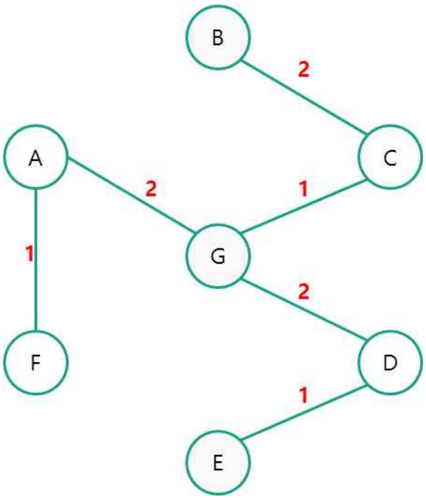



3)그다음 가중치 3인 EF를 연결하는 과정에서 사이클이 발생하여서 연결하지않는다.




4)그이후 모든 간선들은 사이클을 형성하기에 연결이 되지않고




로 완성되어서 최소신장트리가 완성되었고 비용합계는 9이다.

# 4. Kruskal 알고리즘 시간 복잡도 분석

**1)직접 코드를 보면서 분석하자**

**E는 간선의수 V는 정점의수**

```
for (int i = 0; i < TV.length; i++) {
for (int j = i+1; j < TV.length; j++) {
node[k] = new Node(i, j, E[i][j]);
H.insert(node[k]);
k++;//가중치와 정점정보들을 노드로 저장한뒤 minheap으로 구현한다.
}}
```

kruskal 메소드에서 처음나오는 명령문들이다.

일단 첫 번째 for문은 V-1만큼의 반복하고
두 번째 for문은 I가 바뀔때마다 V-1번,V-2번....1번까지 반복한다
for문 두 번은 O(V^2)이라고 할 수 있다
그리고 그안에서 일어나는 명령문인
H.insert의 경우에는 트리에 넣어서 insert를 하는건데 이 명령은 트리의 높이에 의해 비례하는데 트리의 높이는 logN(N은 트리의 크기)라고 나타낼 수 있다.
여기서 트리의 크기는 Edge들의 수라고 할수있기 때문에 O(logE)라고 할 수 있다.

전체적 분석을해보면 O(V(V(logE)))이기 때문에  이 코드의 시간복잡도는 O(V^2logE)라고 할 수 있다.
그래프의 성질에 의해 V^2>=E이기 때문에 복잡도에선 V^2대신 E라고 나타낼 수 있다.

그러므로 이코드의 시간복잡도는 O(ElogE)이다.

```
for (int i = 0; i <=TV.length; i++) {
//최소 신장트리는 TV크기 -1개 만큼만 사용하기에  for문의 횟수를 저렇게 정했다.
Node check = H.remove();
if (collapsingfind(check.i) != collapsingfind(check.j)) {
// 최고 root가같은지판단한다 즉 원소가 같은그룹인지확인한다.
weightedunion(check.i, check.j);// 두개의 그룹이 같지않다면 서로 연결한다
System.out.print("선택된 간선 : " + TV[check.i] + "--> " + TV[check.j] + " / ");
System.out.println("weight = " + check.cost + " 추가");
Cost += check.cost;// 만들때 사용된 비용을 저장될 최종비용에 저장한다.
}}
```

일단 for문 한번으로 O(V)로 시작한다
그다음은 heap을 삭제하고 heapify하는 remove insert와 마찬가지로 O(logE)라고 할 수 있다.
collapsingfind의 경우에는 O(logV)이다 왜냐하면 일반 find와 달리 구성할 때 자식이 위로 계속 올라가는 시스템이라서 제일 나쁠경우에도 logV가 된다.
wieghtedunion도 collapsingfind후 연결하는 메소드이기 때문에 똑같이 logV이다.

여기서 V를 E로 바꾼다음에 나타내면 O(1/2logE)가 나오는데 앞의 배수는 무시가능함으로 O(logE)라고 나타낼 수 있다.

각자나올걸 분석해보면
O(V(logE+logE))가되어서 O(VlogE)가된다.

kruskal의 최종복잡도는
O(ElogE+VlogE)라서 E>V이기 때문에 O(ElogE)라고 할 수 있다.