1. union-find 자료구조 (≈서로 소 집합 자료구조)

서로 소 부분 집합들로 나누어진 원소들의 데이터를 처리하기 위한 자료구조. union 및 find 두 연산으로 조작할 수 있다.

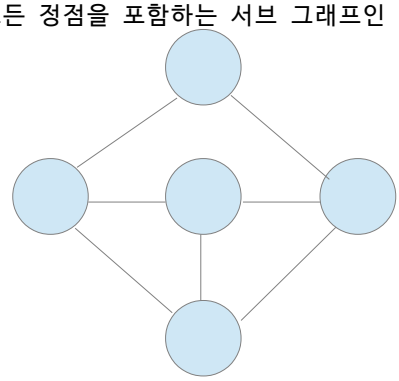
parentOf 배열: 각 노드에 대해 부모 노드를 정리해 놓은 배열. 간선이 하나도 없는 경우 각 노드의 부모 노드는 자기 자신으로 설정해 둔다. (즉, parentOf[i] = i) 간선을 하나 둘 연결하면서 특정한 방법으로 두 노드 간 친자 관계를 맺는다.

find 연산: 트리 내에서 해당 노드의 루트를 찾는 연산. 노드와 그 노드의 부모 노드가 같아질 때 까지 부모를 추적해 올라가는 방법을 사용한다. find 연산을 사용하여 그래프에서 두 노드가 (직·간접적으로) 연결되었는지 판별할 수 있다. 즉, 두 노드에 대해 루트 노드가 서로 같다면 두 노드는 연결된 것이다. 또한 그래프에 사이클이 있는지 여부를 점검할 수도 있다. 그래프에 간선을 하나씩 추가해 나갈 때 간선의 양 노드의 find 연산 결과가 같다면, 해당 간선은 그래프에 사이클을 만드는 간선이다.

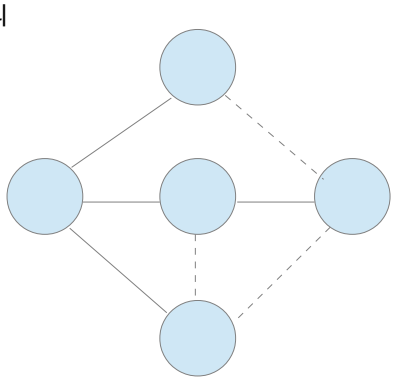
union 연산: 입력된 간선을 그래프에 포함하는 연산. union 연산 후에 분리되었던 두 그래프가 연결되었다면, 두 그래프 내 노드에 대해 find 연산을 수행 결과는 서로 같아야 한다. 이를 위해 입력할 간선의 양 노드 A, B에 대해 [A의 루트 노드]와 [B의 루트 노드]간에 친자관계를 맺는다. (대개는 [A의 루트 노드]와 [B의 루트 노드] 중 수치가 더 작은 쪽이 수치가 더 큰 쪽의 부모로 설정한다.) 입력할 간선의 양 노드 A, B 간에 친자관계를 맺는 것이 아님에 주의한다.

2. 신장트리

특정 연결그래프에 대해 모든 정점을 포함하는 서브 그래프인 트리



그리기 1: 이 연결그래프는 사이클이 존재하므로 트리가 아니다.



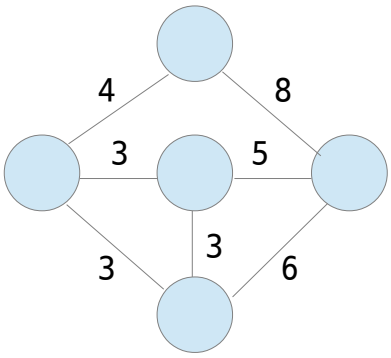
그리기 2: 왼쪽의 연결그래프에서 간선 몇 개를 지워서 트리를 만든다.

노드는 전부 보존하고 간선을 몇 개 지워서 그래프에 있던 사이클을 없게 만든다.

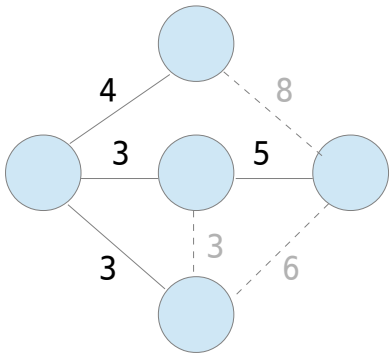
신장트리는 서브그래프이기도 해야 하므로 본래의 그래프에 없던 간선을 추가해서는 안된다.

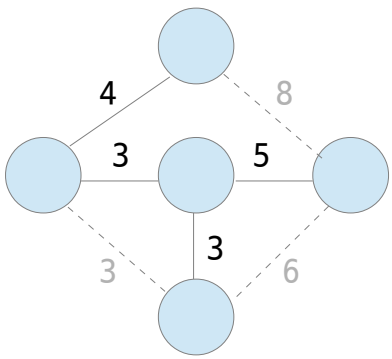
3.최소 신장 트리 (Minimum Spanning Tree: MST)

간선의 가중치 합을 최소로 하는 신장 트리



이 연결그래프로 신장 트리를 만든다면 여러가지 방법이 있고, 각 결과에 대해 간선의 가중치 합은 서로 다를 수 있다





여러 신장 트리 중 간선의 가중치 합이 최소가 되는 것은 이들이다. 이를 최소 신장 트리라고 부른다.

한 연결그래프에 대해 최소 신장 트리는 유일하지 않을 수 있다.

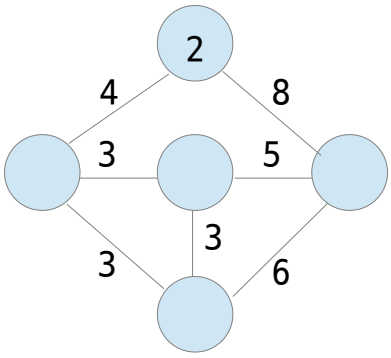
4. 크루스칼 알고리즘

연결그래프에 대해 최소 신장 트리를 생성하는 알고리즘으로, 각 단계에 대해 사이클을 생성하지 않는 한 최소 가중치의 간선을 취해 나가는 방식을 사용한다. (그리디 알고리즘의 일종)

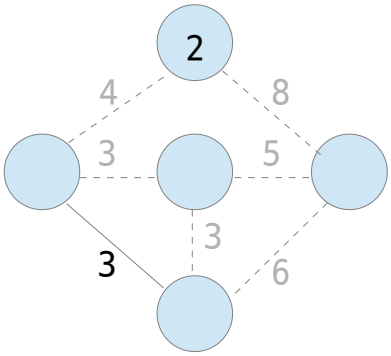
(1) 간선을 가중치 순으로 정렬하여 가장 낮은 간선을 선택

(2) 현재 선택한 간선의 두 노드가 u, v일 때 u와 v가 같은 그룹이라면 (서로 연결된 노드라면) 아무 것도 하지 않고 넘어간다. u와 v가 다른 그룹이라면 현재 선택한 간선을 최소 신장 트리에 추가

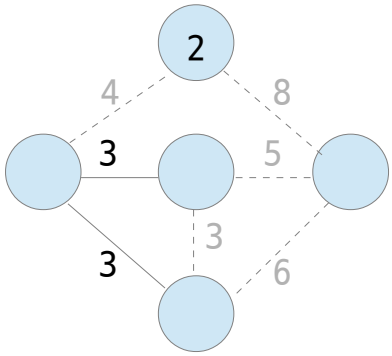
(3) 최소 신장 트리에 V-1개의 간선이 들어갔다면 알고리즘을 종료한다. 그렇지 않으면 그 다음으로 비용이 작은 간선을 선택한 후 2 번 과정을 반복



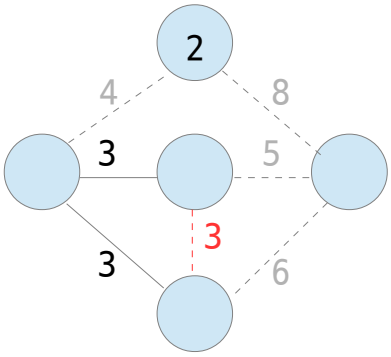
이 연결그래프에 대해 크루스칼 알고리즘을 적용해 본다.



가장 가중치가 낮은 (1, 4) 간선을 MST에 추가한다.
①과 ④는 같은 그룹이 된다.



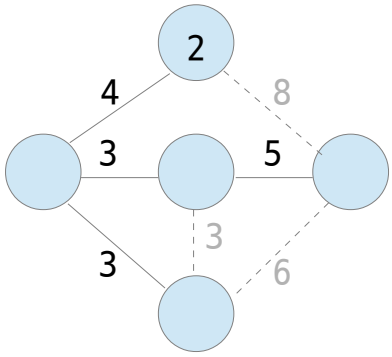
그 다음으로 가중치가 낮은 (1, 3) 간선을 MST에 추가한다.
①과 ③, ④는 서로 같은 그룹이 된다.



그 다음으로 가중치가 낮은 간선은 (3, 4)인데, 이미 ③과 ④는 같은 그룹이므로 이 간선은 MST에 추가하지 않는다. (또는, 간선 (3, 4)는 그래프에 사이클을 생성하므로 이 간선은 MST에 추가하지 않는다.)

각 단계에 대해 가중치가 가장 낮은 간선을 선택하는 것이 지역적(local)으로는 최선이지만, 전역(global)으로는 오히려 방해가 되는 경우가 있다.

=> 그리디 알고리즘에 대해 local에서의 최적이 global에서의 최적을 보장해 주지는 않는다.



간선이 4개 선택될 때 까지 이 과정을 반복한다. 트리의 특성상, 노드가 V개일 경우 간선은 V-1개이어야 한다. 이 예시의 경우 노드가 5개 있으므로 MST에는 간선이 4개 있어야 한다.

5. 크루스칼 알고리즘과 union-find 자료구조

크루스칼 알고리즘을 적용하기 위해서는 [지금 살펴보고 있는 간선의 양 노드가 이미 연결이 된 것인지] 여부를 판단할 수 있어야 한다. Flood Fill 알고리즘을 사용하면 이를 해결할 수 있으나 시간복잡도 상 손해를 보게 된다. 이 경우 union-find 자료구조를 사용하여 효율적으로 크루스칼 알고리즘을 사용할 수 있다. (union-find 자료구조를 사용할 경우 시간복잡도는 O(ElogE)가 된다. 이는 간선을 가중치 상으로 정렬하는데 필요한 시간복잡도와 같다.)

6. 기타 사항 (tuple)

크루스칼 알고리즘을 사용하기 위해서는 간선을 가중치 상으로 정렬해야 한다. 이를 위해 인접행렬이나 인접리스트 보다는 간선의 리스트를 사용하는 것이 더 편하다. 하나의 간선은 연결되는 두 노드 및 가중치 값을 가지므로 최소 세 값을 가질 수 있어야 한다. C++ 에서는 tuple을 사용하여 간편하게 값 세 개를 묶고, 정렬 시 가중치를 부여해 줄 수 있다.

tuple<long long, int, int> edges[100005]; //그래프에 있는 간선 배열. 각 간선은 값을 세 개 갖는데 순서대로 가중치, 노드, 반대쪽 노드의 값이다.

edges[i] = {weight, nodeA, nodeB}; //tuple은 중괄호로 간편하게 표현할 수 있다.

std::sort(edges, edges+numEdges); //tuple 배열을 정렬할 시, tuple의 맨 먼저 나오는 값이 오름차순이 되도록 정렬된다.

tuple<long long, int, int> currentEdge = edges[i];

int currentNodeA = get<1>(currentEdge); //tuple에서 특정 위치의 값은 get<idx> 함수를 통해 구할 수 있다.

int currentNodeB = get<2>(currentEdge);

long long currentWeight = get<0>(currentEdge);