알고리즘

[도시 설계 프로그램 설계 1차]

**E-mail : cary1234@naver.com**

**1. 개요**

**설계의 목적, 요구사항, 개발환경**

->설계되어있는 모든 도시들을 자동차로 왕래 할 수 있도록 도로들을 포장하는데, 최소비용으로 모든 도시를 왕래할 수 있도록 하는 프로그램 입니다. 이때 모든 도시를 왕래 할 수 있어야 하며, 최소비용으로 연결하는 것을 Kruskal 알고리즘의 Minimum Spanning Tree 를 찾는 알고리즘을 통하여 프로그램을 설계하는 것이 문제에서 원하는 것입니다.

Union -find 자료구조는 연결리스트로 구현해야 하는 것이 조건 이였으며, weighted-union heuristic을 이용해야 합니다.

Visual studio에서 C++언어로 구현하였습니다.

**2. 필요한 자료구조 및 기능**

- 필요한 자료구조

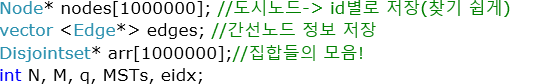
**1)Graph() :** 도시(V)에 대한 정보와, 그 도시들끼리 어떻게 연결되어 있는지 저장 되어있는 간선(E)의 정보를 저장하는 자료구조가 필요합니다.

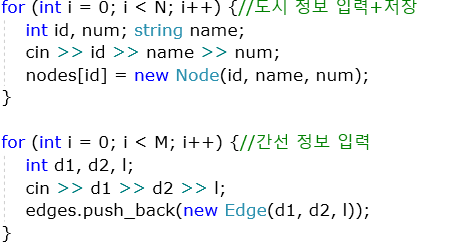
**2)Union-Find() :** 서로소 집합인 DisJoint 집합들을 분할하여 저장한 집합 자료구조입니다.두 집합을 합치는 연산인 union연산과, 어떤 원소가 어디의 집합에 속하는지 찾아내는 Find 연산이 있습니다.

**3. 기능별 알고리즘 명세**

- 각 기능에 대한 알고리즘 설명(시간 및 공간복잡도는 worst case를 기준으로 쓰겠습니다.)

**(0) 데이터 입력받기**





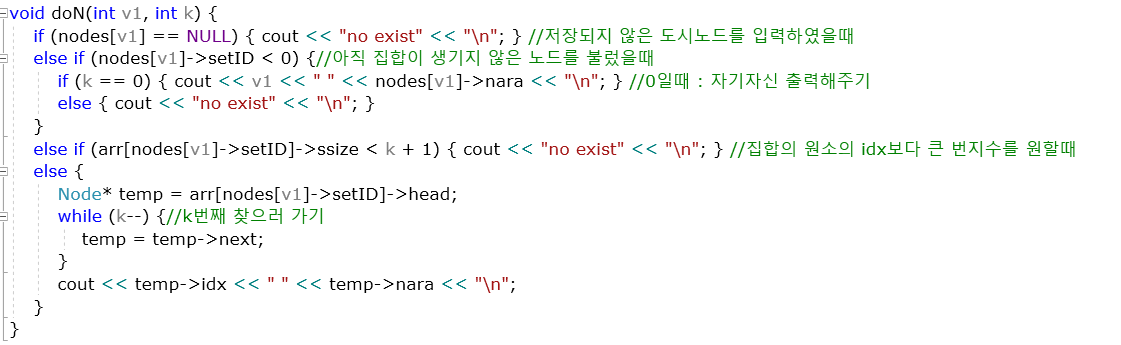
>N개의 줄을 통해 도시Node의 정보를(도시번호, 도시명 , 도시의 인구수) 입력 받을 때에는, 나중에 Node의 정보를 찾을 때 O(1) 타임이 걸리도록, Node\* 배열을 선언 한 후에, 도시번호가 Node\* 배열의 인덱스가 되도록 저장해 주었습니다. Insert의 시간복잡도 ->O(1), 공간복잡도 ->O(1000000)

>M개의 줄을 통해 간선노드 정보(도시1, 도시2, 두 도시 간의 도로 길이)는 Edge\* 벡터에 넣어서 저장해주었습니다. sort를 한 후 차례로 뽑아서 간선정보를 이용할 것이기 때문에, 굳이 array를 쓰지 않고 vector로했습니다. Insert의 시간복잡도->O(1), 벡터를 저장하는 공간의 공간복잡도->O(M)

**(1) 중간정보 출력: 특정 도시가 포함된 연결리스트의 내부 정보 파악**

N v k: 도시 v가 포함된 집합에서 헤드로부터 k번째 노드를 출력하는 질의

V : 도시번호 k: 정수



제일 먼저 , 저장되지 않은 도시노드를 입력 하였을 때는 “no exist”를 출력하였습니다.

도시 노드에 setID라는 값이 저장되어 있습니다. 이는 각 도시노드가 속한 집합의 head 를 의미하는데, 아직 지정되지 않았다면 예외처리를 해 주었습니다(k가 0일 때 자기자신의 도시번호 출력, 0이 아닌 다른 수라면 no exist출력). 또한 k번째 헤드로부터 k번째 도시 정보가 존재하지 않을 때도 “no exist”를 출력하여 예외처리 해주었습니다.

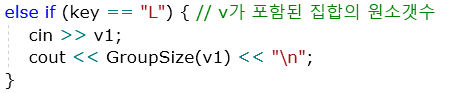
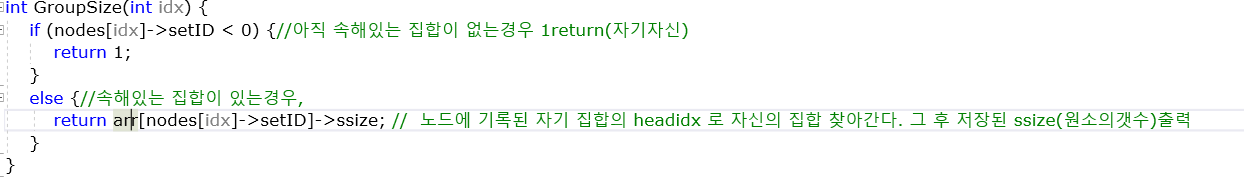
예외처리를 한 후, head로부터, k번 다음 노드로 이동하여 그 노드에 저장된 도시 번호, 도시명을 출력하였습니다. .,

V가 속한 집합의 head를 find하는 연산의 시간복잡도: O(1)

Head로부터 k번째 도시를 find 하는 연산의 시간 복잡도: O(N)

**(2) 중간정보 출력: 특정 도시가 포함된 연결리스트의 크기**

L v: 도시v가 포함된 linked-list의 총element의 개수에 대한 질의 , v: 도시번호

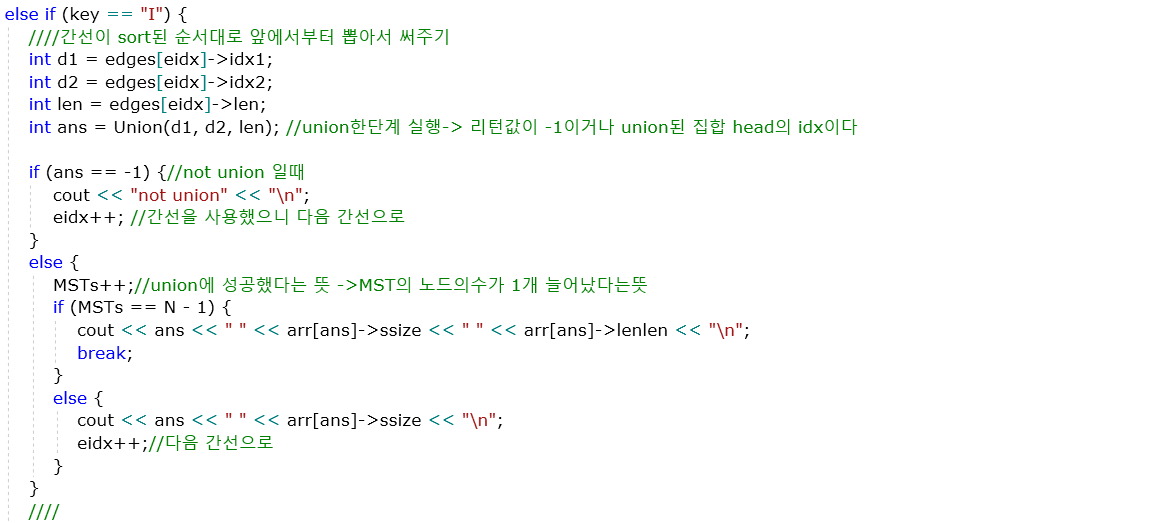
 

도시노드를 입력받으면, GroupSize(노드가 속한집합의 사이즈)를 리턴하는 함수를 호출한다. 속해있는 집합이 없는경우 1을 출력하고(자기자신), 속해있는 집합이 있는경우 도시노드에 저장된 setID로 집합에 접근하여, 집합에 저장된 size를 출력한다.

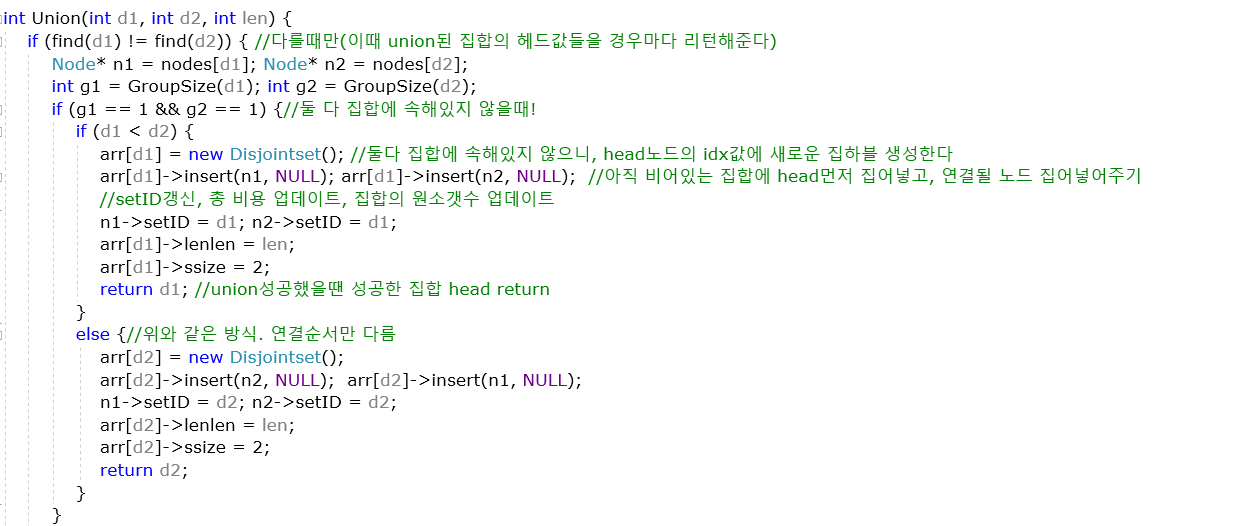
시간복잡도: O(1)

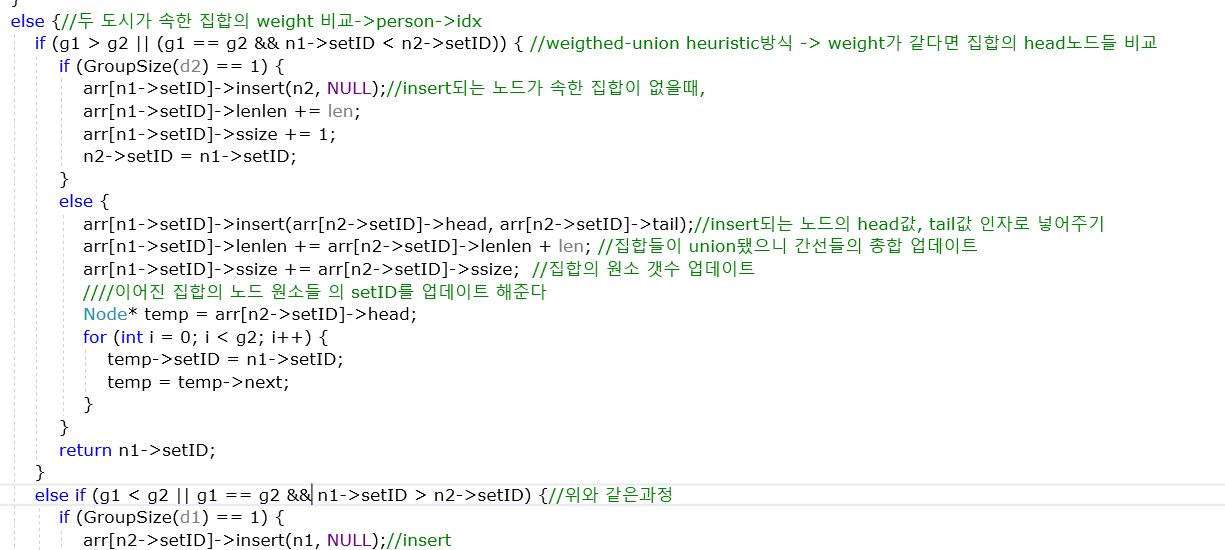
**(3) 중간단계: Kruskal 알고리즘에서 하나의 반복(iteration) 진행**

I: I가 입력되면 kruskal알고리즘에서 하나의 반복을 실행한다.



Kruskal 알고리즘은 간선의 길이가 작은 순서대로 진행을 하기 때문에, sort가 되어있는 간선 정보를 차례로 호출하여 Union함수를 실행합니다.->sort의 시간복잡도:O(MlogM)





유니온 함수의 일부입니다. 노드가 아직 집합에 속해 있지 않을 때에는 Disjointset\* 배열에 접근 할 수 없으므로, 예외처리를 해주면서 조건에 맞게 union시켜주었습니다.

노드 두개 다 집합에 속해 있지 않을 때에는, head가 될 노드번호를 새로운 Disjointset\* 배열 idx로 설정한 뒤 Disjointset을 생성해줍니다.

편의성을 위하여 모든 도시번호가 head가 될 수 있는 경우를 생각하여,

공간복잡도: O(1000000)

집합끼리 합쳐줄 때, 기존 집합의 tail에 연결시켜줄 집합의 head를 연결시켜 준 후, 기존집합의 tail을 인자값으로 받은 연결시켜줄 집합의 tail을 바꿔줍니다.

insert할때의 시간복잡도:O(1)

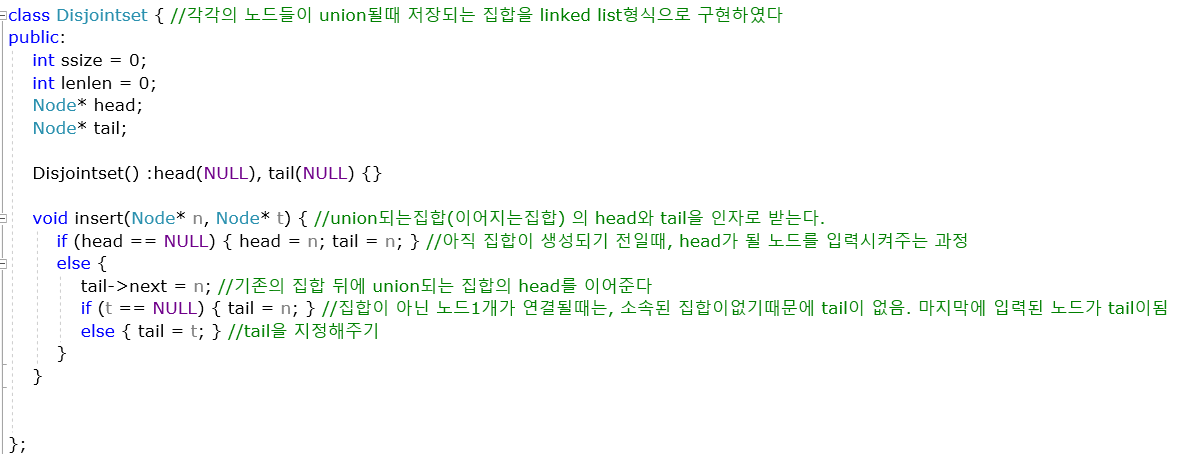
insert시켜 준 후 도로길이의 합, 집합에 속한 도시노드의 개수 를 업데이트 해주고, 각 노드들의 setID를 업데이트 시켜주었습니다.

Union()의 총 시간복잡도 :O(N) (최악의 경우 N/2 번이기 때문)

Union함수가 리턴 된 후에, I연산이 원하는 값들을 출력해줍니다.

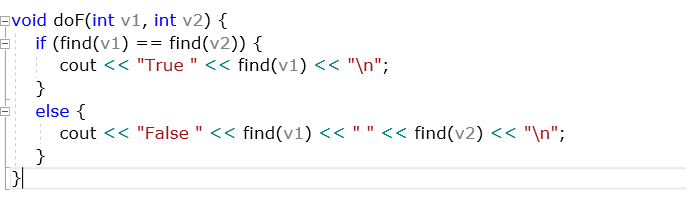
Union이 되지 않았을땐 -1을 리턴하도록 하여, not union을 출력하게 하였습니다.

Union에 성공할때마다 MSTs변수를 하나씩 늘려(0부터), N-1과 같아지면 모든 도시노드가 union되었다는 의미이므로 , MST를 찾아낸것입니다. 이에 맞게 순서대로 출력하고, 프로그램을 종료하였습니다.



**(4) 중간정보 출력: 두 도시가 같은 집합에 속하는지 확인**

F v1(도시번호) v2(도시번호): 도시v1,v2가 같은 집합인지 물어보는 질의



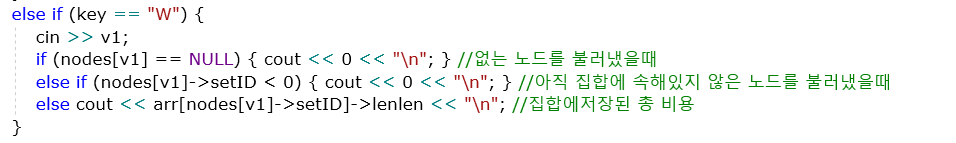


find함수를 호출하였다. find함수는 노드가 속한 집합의 head의 Id를 알려주는 함수다.(사진 속 주석 참고) . find함수의 시간복잡도 :O(1)

각각의 함수가 return한 값들이 같다면 True , 동시에 속한 집합의 head id를 출력하고, 다르다면 각각의 도시가 속한 집합의 head id를 출력한다.

**(5) 중간정보 출력: 임의의 도시를 포함하는 집합의 모든 포장도로의 길이**

W v(도시번호): 도시 v를 포함하는 집합의 모든 포장도로들의 길이의 합에 대한 질의



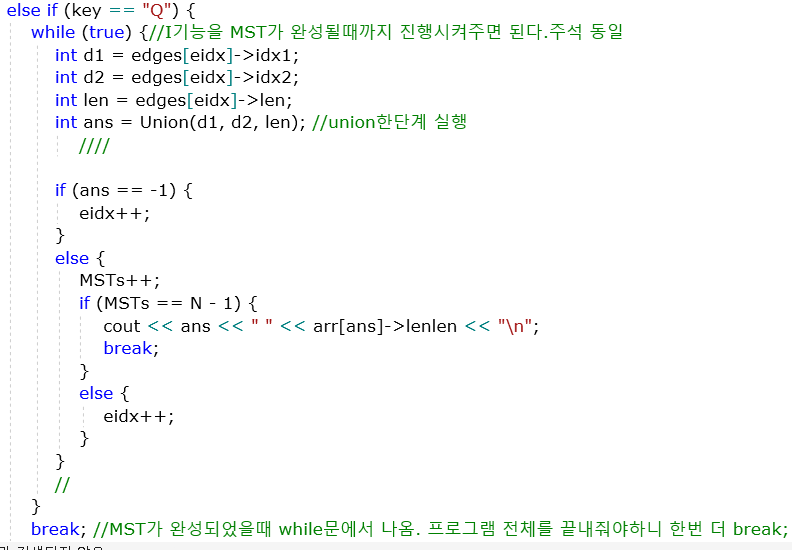
각각의 도시노드에 저장된 setID값을 통하여 소속된 집합(Disjointset) 에 저장된 lenlen(도로 길이의 합)의 값을 출력한다.

소속된 집합이 없을 경우 0을 출력한다.

시간복잡도: O(1)

**(6) 알고리즘의 나머지 단계 연속수행 및 프로그램 종료**

Q: MST를 생성할 때까지 Kruskal 알고리즘을 진행하라는 질의



MST를 생성할 때까지 I를 계속 호출하면 됩니다.(계속 Kruskal 알고리즘 한단계씩 진행)

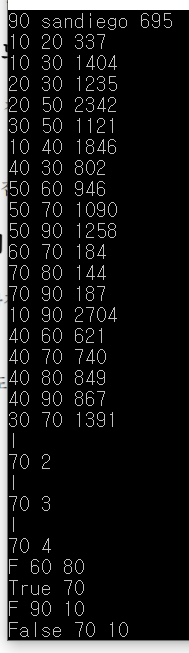
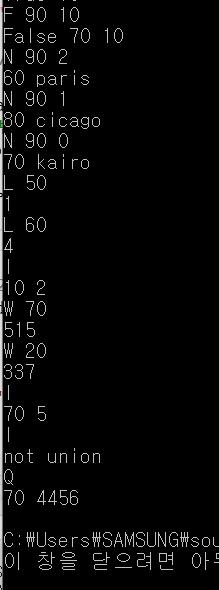
시간복잡도 : O(NlogN)

**4. 인터페이스 및 사용법**

**- 간단한 사용법 설명**

프로그램을 실행 후 첫 번째 줄에는 설립된 도시들의 개수N, 도시들 사이에 존재하는 비포장도로의 개수M, 질의의 수 q가 공백으로 구분되어 주어집니다.질의의 수 만큼 기능별 질의를 입력해주면 됩니다.

**- 실행 화면 캡쳐 포함**

 <- 프로그램 종료

**5. 평가 및 개선 방향**

**- 본 결과의 장점**

Linked list로 집합을 구현하여 집합끼리 insert하는과정에서 시간복잡도가 상수시간으로 걸립니다. 또한 도시노드나 다른 데이터값을 빠르게 찾을 수 있도록 설계하여, 전체적으로 시간이 빨리 걸리도록 만들었습니다.

**- 본 결과의 단점**

빠르고 편리성 있게 설계를 하여서, 메모리를 많이 사용하였습니다.

코드 자체의 가독성이 좋지않고, 한번만 계산해주어도 되는 것을 여러 번 계산한 것 같아 아쉬웠습니다.

**- 향후 개선 방향**

다른사람이 코드를 봐도 깔끔하다고 느낄 수 있게 알고리즘 뿐만 아니라 코드 작성시에도 효율적으로 작성해야겠다는 생각이 들었습니다.(변수 정리 등등)

메모리를 너무 많이 사용하였는데, Disjointset\* 배열 같은 경우에는 다른 효율적인 메모리로 개선해 보고 싶습니다.