CSE3081 (2반): 알고리즘 설계와 분석

[HW3]

담당 교수: 임 인 성

20171682 임정호

1. **실험환경 기술**

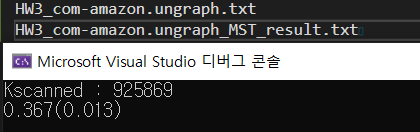
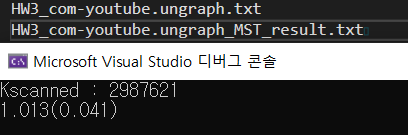
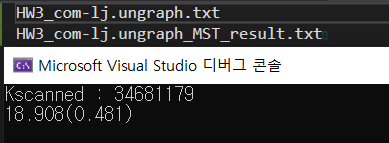
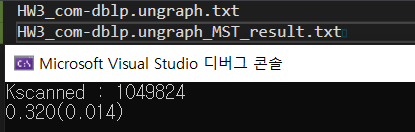
OS: Windows 10 Pro

Cpu: Intel® Core™ i5-7200U CPU @ 2.50Ghz 2.71Ghz

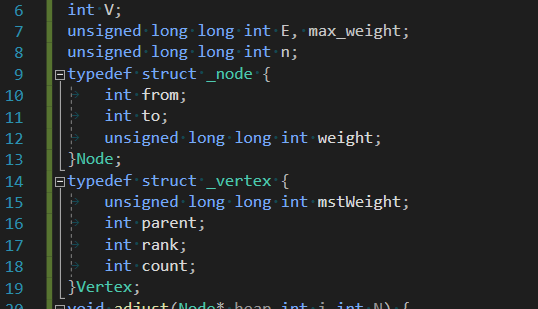
Ram: 8.00GB

Compiler: Visual Studio 19 Release Mode/x64 Platform

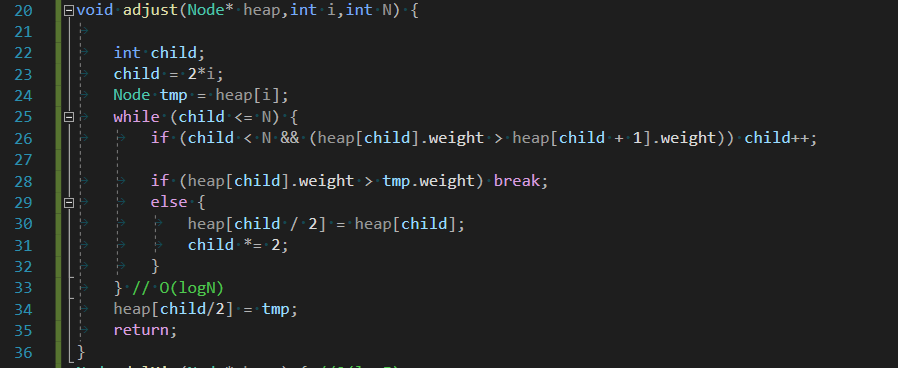
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 파일 이름 | 작동 여부 | MST weight | 수행 시간(초) | Kscanned |
| HW3\_com-amazon.ungraph.txt | YES | 2729670156 | 0.367(0.013) | 925869 |
| HW3\_com-dblp.ungraph.txt | YES | 2747895457 | 0.320(0.014) | 1049824 |
| HW3\_com-lj.ungraph.txt | YES | 28308045762 | 18.908(0.481) | 34681179 |
| HW3\_com-youtube.ungraph.txt | YES | 14578691475 | 1.013(0.041) | 2987621 |

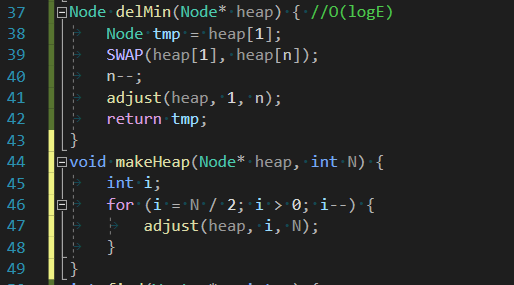
1. **코드 설명**



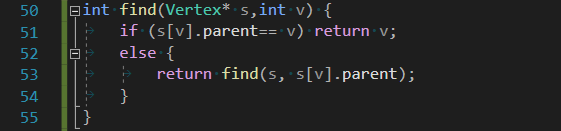
edge의 정보와 disjoint-set을 만들 때 필요한 정보를 저장하기 위한 구조체 Node와 Vertex를 선언하였다. edge는 from\_vertex와 to\_vertex 그리고 그 edge의 weight를 담고 있고, disjoint-set은 union할 때 필요한 parent와 rank 그리고 output을 출력할 때 필요한 해당 MST에 속한 vertex이 개수와 그 mstWeight를 담기 위한 변수, mstWeight, count가 구조체 안에 담겨있다. 이 때 mstWeight는 2^32까지 올 수 있어 int형의 범위를 벗어날 수 있어 mstWeight와 weight를 unsigned long long int로 선언하였다. 전역 변수 n은 힙의 크기를 저장하기 위한 전역변수이다. V와 E 그리고 max\_weight는 처음 입력 받을 때 받는 vertex개수와 edge개수 그리고 edge들 중 max\_weight를 저장하기 위한 전역변수이다.



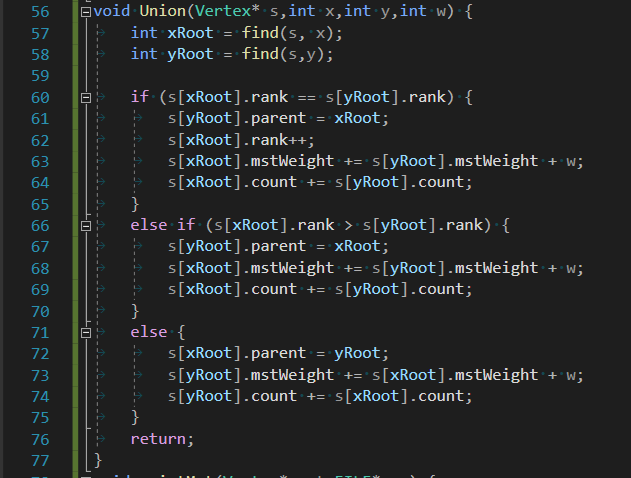
**adjust함수**는 heap에 i번째 값이 minHeap의 구조를 잘 이루고 있는지 검사하여 아니면 그 자식들과 비교해가면서 minHeap의 구조를 유지하게끔하는 함수이다. 이 때 N은 힙의 크기를 나타낸다. 이 함수의 시간복잡도는 O(logN)이다. 이 adjust함수는 minHeap을 구축하고, delete할 때 쓰인다.

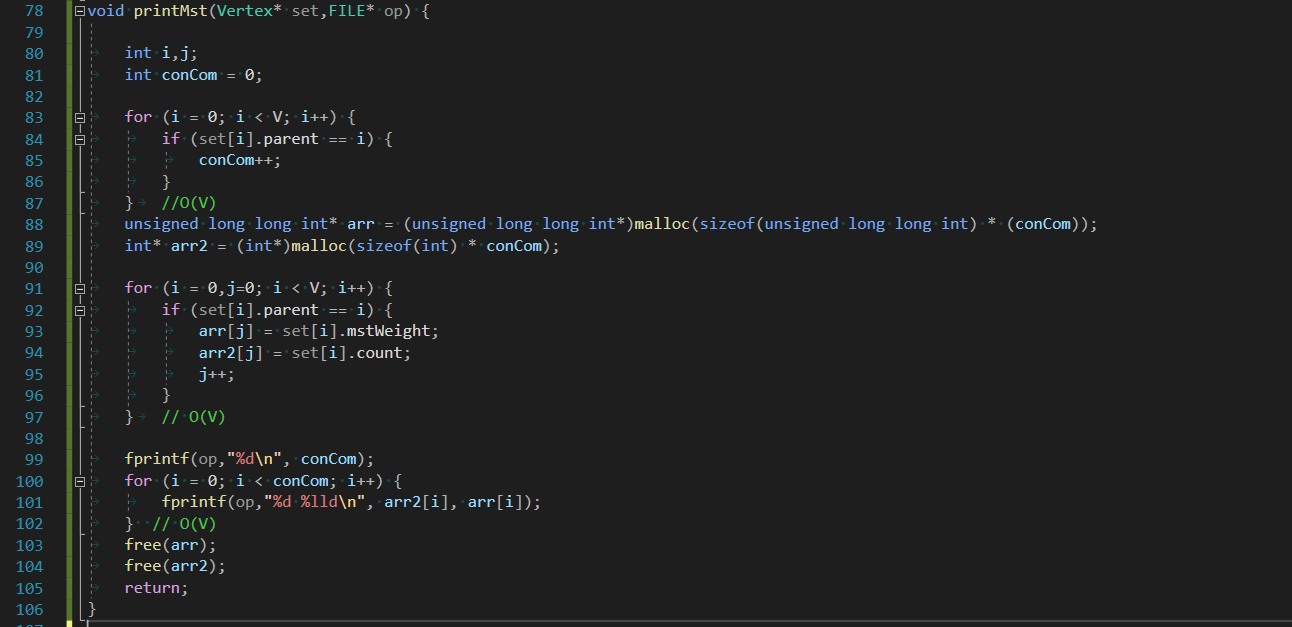


**delMin함수**와 **makeHeap**함수이다. minheap을 구축하기 위해 makeHeap함수를 호출하며, 이 때 minHeap은 edge의 weight를 key값으로 하여 구축하게 된다. 자식 노드가 있는 부모노드들만 검사하여 O(N)즉, O(E)의 시간복잡도로 구현할 수 있다. delMin함수는 kruskal 알고리즘은 edge의 weight가 가장 작은 것부터 뽑아 트리로 연결하는데, 이 때 가장 작은 것을 뽑을 때 쓰이는 deletePQ함수이다. 가장 작은 weight를 가지는 edge들을 뽑을 때 효율적으로 관리하기 위해 minheap을 사용하는 것이다. 입력을 받을 때 하나하나 힙에 insert하는 것보다 한 번에 adjust함수로 minheap 구축하는 것이 O(ElogE)에서 O(E)로 단축할 수 있다. 이 이유 때문에 heap구조를 사용하는 것이다.

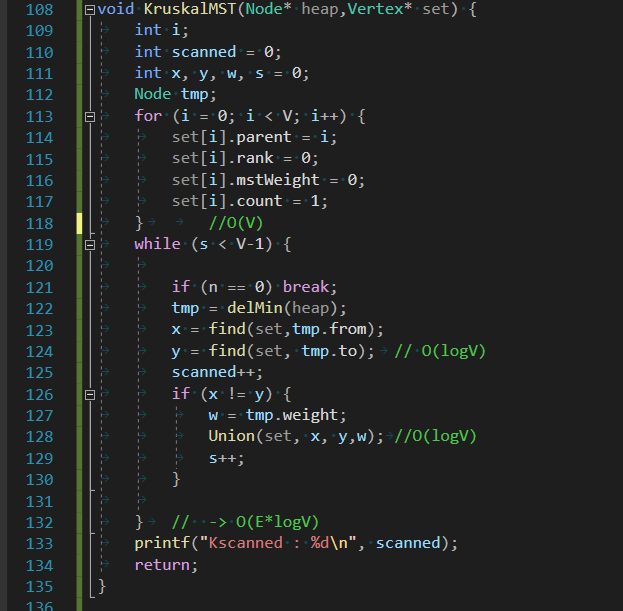


**find함수**는 disjoint-set 자료구조에서 v번째 vertex의 parent를 리턴해주는 함수이다. 시간 복잡도는 O(logV)이다. 이때 시간복잡도가 O(V)가 되지 않는 이유는 vertex를 union해줄 때 그 vertex의 rank를 비교해주면서 넣어주기 때문이다.

**union함수**는 x와 y번째의 vertex를 연결해주는 함수이다. 이 때 rank를 비교하여 rank가 같을 경우에만 rank값이 1 증가하고 다를 경우에는 작은 값을 가지는 쪽이 큰 값을 가지는 쪽에 붙는 것으로 구현하였다. 이 때 mstWeight를 저장하기 위해 rank가 큰 트리의 mstWeight에 x와 y를 잇는 edge의 weight와 rank가 작은 트리의 mstWeight를 합해주는 식으로 mstWeight를 최신화하고 있다. count는 트리의 속한 vertex의 개수이며 두 트리가 union되면 count도 합한 결과로 최신화된다.



**printMst**함수는 kruskal 함수가 종료되고 나서 disjoint-set의 vertex들을 읽어서 connected component의 개수를 카운트한다. 83번째 for문으로 O(V)로 connected component의 개수를 알 수 있고, 그 개수만큼 mstWeight를 저장할 배열 arr과 vertex의 개수를 저장할 배열 arr2를 동적할당해준 다음, O(V)의 시간복잡도로 값을 저장해줄 수 있다. printMst함수는 매개변수로 output파일의 파일포인터를 받아오고, 99번부터 102번은 output파일에 출력해주는 부분이다.

kruskalMST함수는 edge들의 minheap과 disjoint-set 자료구조를 위한 Vertex 배열 set을 매개변수로 받아온다. 113번부터 118번 코드는 set을 초기화해주는 부분이다. 초기화할 것은 parent, rank 그리고 mstWeight, count이다. parent와 rank는 disjoint-set으로 사이클 여부를 확인하기 위함이고, mstWeight와 count는 결과를 출력할 때 쓰기 위함이다. 이 부분은 O(V)의 시간 복잡도를 나타낸다. 119번 while문은 s가 V-1과 같거나, n이 0이 되면 종료하는데, MST는 edge의 개수가 V-1임을 나타낸다. 다만 이 경우 주어진 graph가 connected인 경우이고, 아닌 경우가 있을 수 있으므로 edge의 heap의 개수가 0일 때 종료되는 것으로 조건을 추가하였다. tmp에 heap에서 가장 작은 edge를 뽑아와서 그 edge가 잇는 양쪽 vertex가 같은 트리인지 확인하고 다른 트리이면 연결해준다. 이 부분이 126번째 줄이다.

**kruskalMST**함수의 시간복잡도는 while문이 dominate하기에 위에 O(V)는 무시되고, 안에 있는 시간 복잡도O(logV)의 union함수들이 while문을 돌면서 반복되는 점을 확인하면 된다. 이 때, 몇 번의 Kscanned로 while문의 종료조건이 만족되어 종료되면 좋겠지만, 최악의 경우는 heap에 있는 edge들을 다 뽑아야 종료될 때이다. 따라서 while문의 시간복잡도는 O(ElogV)이고, 이 while문이 kruskalMST함수의 시간복잡도를 dominate하기 때문에 kruskalMST함수의 시간복잡도 또한 O(ElogV)가 됨을 알 수 있다. 이번 과제에서는 MST를 출력하라고 하지 않아 따로 graph로 만들진 않았지만, 만들려고 하면 방법은 adjacecny list를 활용하면 O(ElogV)의 시간복잡도 안에 구현할 수 있다. 그 방법은 다음과 같다. 길이 V만큼 Node 포인터 배열을 할당하고, 128번 줄 다음에

1. **결과 분석**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 파일 이름 | E | V | MST weight | 수행 시간(초) | Kscanned |
| HW3\_com-amazon.ungraph.txt | 925872 | 334863 | 2729670156 | 0.367(0.013) | 925869 |
| HW3\_com-dblp.ungraph.txt | 1049866 | 317080 | 2747895457 | 0.320(0.014) | 1049824 |
| HW3\_com-lj.ungraph.txt | 34681189 | 3997962 | 28308045762 | 18.908(0.481) | 34681179 |
| HW3\_com-youtube.ungraph.txt | 2987624 | 1134890 | 14578691475 | 1.013(0.041) | 2987621 |

시간복잡도에 값을 넣어 계산하면 다음과 같다.

1. O(ElogV) -> (925872)log(334863) -> 16989751.2 (18.35)
2. O(ElogV) -> (1049866)log(317080) -> 19181051.82 (18.27)
3. O(ElogV) -> (34681189)log(3997962) -> 760558474.77 ( 21.93)
4. O(ElogV) -> (2987624)log(1134890) -> 60081118.64 (20.11)

ElogV / 수행 시간을 나타내면 다음과 같다.

16989751.2 / 0.367 => 46293600

19181051.82 / 0.32 => 59940787

760558474.77 / 19.908 => 40224163

60081118.64 / 1.013 => 59310087

이론적으로는 O(Kscanned\*LogV)를 따를 수 있지만, 이번 자료의 경우 E의 값과 Kscanned의 값이 거의 비슷해서

O(ElogV)로 얻은 것 같다.

ElogV / 수행 시간의 값이 예상 보다 차이가 조금 있었지만, 이 차이는 컴퓨터 환경과 프로그램 가동시 돌아가던 다른 프로그램이 있었는지 이런 조건에 따라 생긴 오차인 것 같다. 위에서 구한 값이 E와 V에 비례하여 차이나는 것이 아니므로 감안할 수 있는 오차로 볼 수 있을 것 같다.