**알고리즘설계와분석**

**HW4**

학번 : 20191617

전공 : 컴퓨터공학

이름 : 이규호

**A. 보고서의 가장 앞에 다음의 예처럼 실험에 사용한 CPU의 속도 및 메인 메모리의 용량 등의실험 환경을 기술하라.**

OS : Windows 11 Home

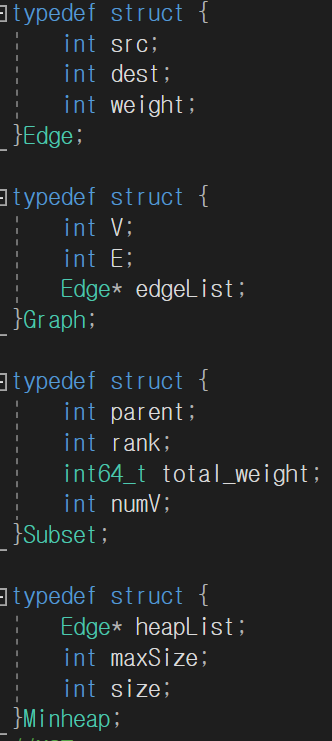
CPU : Intel(R) Core(TM) i5-1035G7 CPU @ 1.20GHz 1.50 GHz

RAM : 8.00GB(7.72GB 사용 가능)

Compiler : Visual Studio 2022 Release Mode

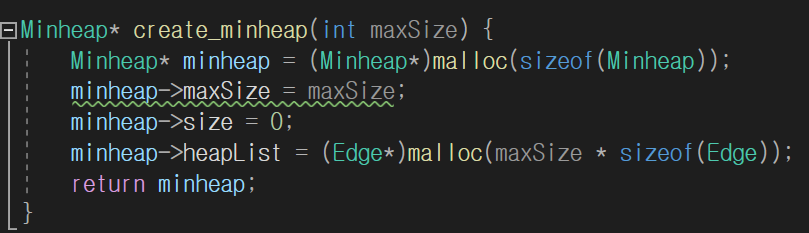
**B. 위에서 요구한 알고리즘들을 올바르게 구현하였다는 것을 밝히기 위하여, 보고서에 자신이작성한 원시코드의 주요 부분을 아래의 Figure 1과 같이 출력한 후 그 위에 코드 구현에 대한 설명을 기술하라. 특히 그러한 시간 복잡도를 달성하는데 필요한 핵심 부분에 대한 자신만의 설명이 있어야 한다.**

**- Struct**

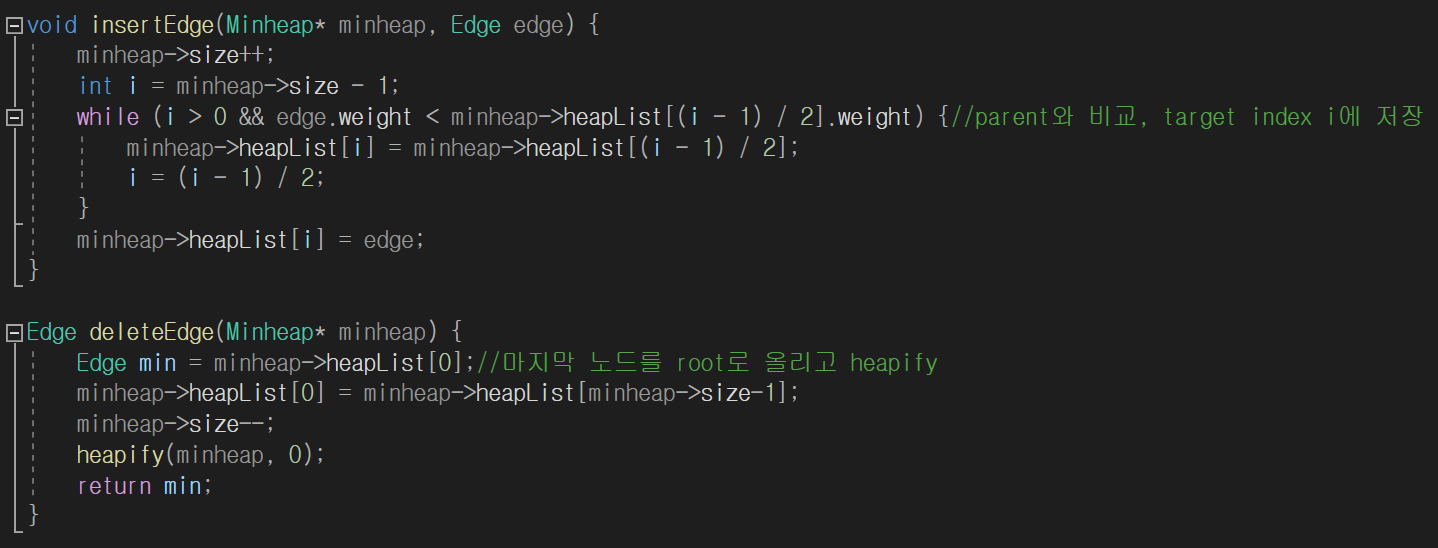


먼저 edge의 양옆 vertex와 weight를 저장하기 위한 Edge struct, 모든 그래프 정보를 저장하기 위한 Graph struct를 선언하였다. text file로 부터 정보를 입력받아 해당 구조체에 저장한다. 또한 disjoint set을 위한 Subset struct를 선언하였다. 먼저 parent와 rank, 그리고 이후 connected commponent 처리를 위해 해당 집합의 edge weight의 총합인 total\_weight와 해당 집합에 포함된 vertex의 개수를 저장하기 위한 numV 또한 선언하였다. 마지막으로 edge를 ascending order로 뽑기 위해 minheap을 사용해야하므로 이를 위한 Minheap struct 까지 선언하였다.

**- Minheap**

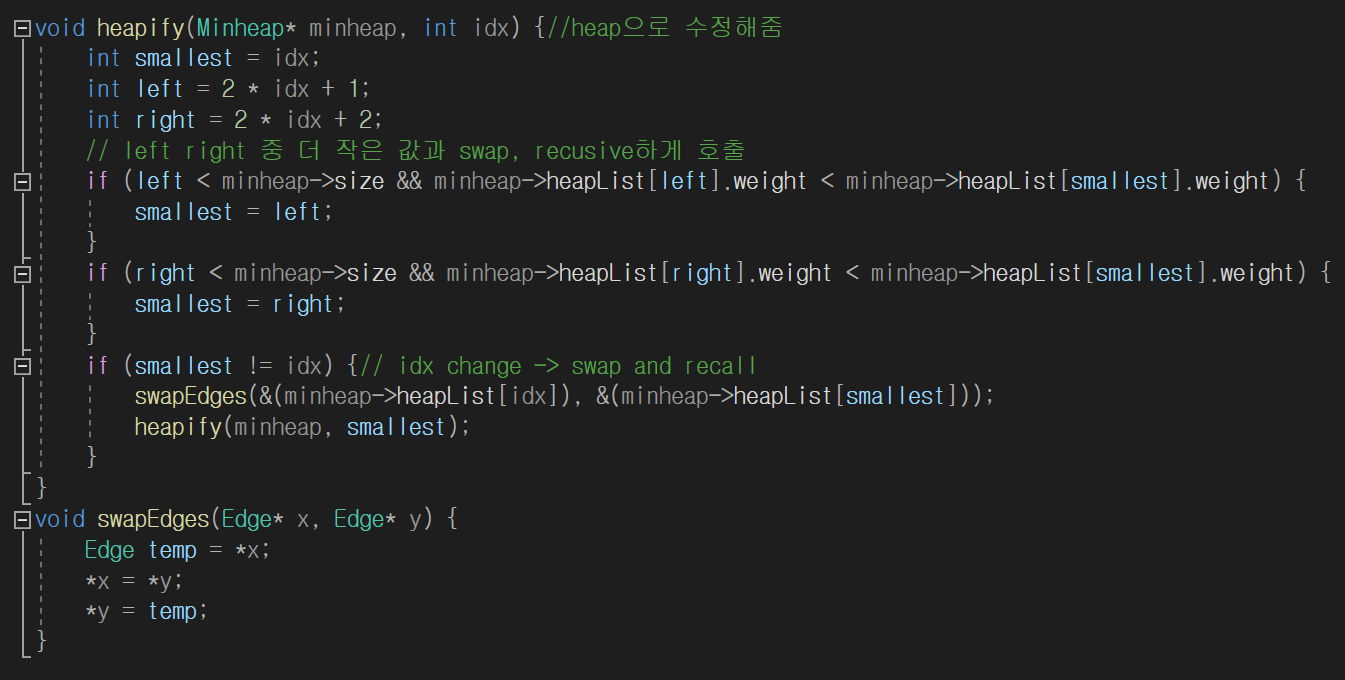


create\_minheap 함수를 통해 먼저 minheap을 동적할당하고, minheap의 list를 paramter 만큼 동적할당하여 edge를 넣을 배열을 생성한 후 minheap을 return한다.



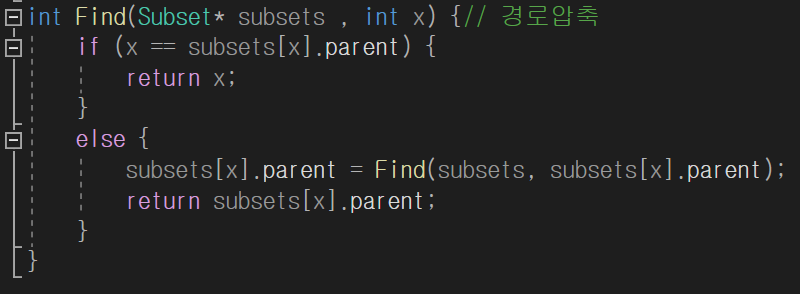
insertEdge 함수를 통해 새로 추가된 edge를 heap list의 마지막에 두고, parent와 비교하며 edge의 weight가 더 작을 경우 계속해서 값을 swap한다. while문을 통해 맞는 자리를 찾으면 해당 index에 edge값을 넣고 함수가 종료된다.

deleteEdge 함수를 통해 root에 있는 node를 저장하고 heaplist의 가장 마지막에 있는 node를 root로 올린다. size를 update하고 heapify함수를 통해 minheap을 수정한 후, root에 있던 node 값을 return한다.

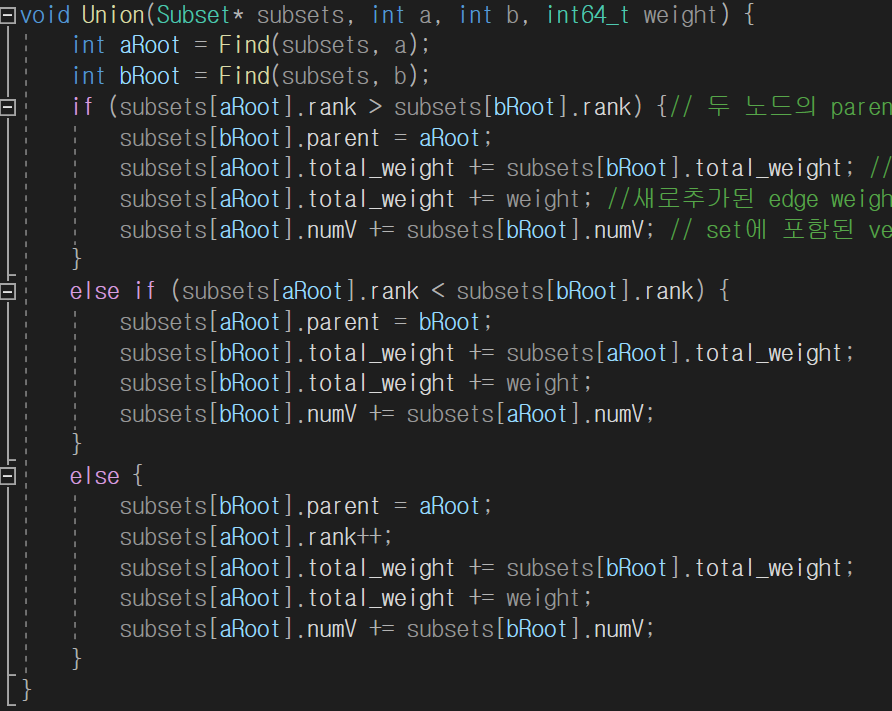


마지막으로 heapify 함수이다. 이는 deleteEdge를 실행할 때 minheap이 유지되도록 adjust하는 함수이다. smallest node를 parameter로 받아 왼쪽 자식노드와 먼저 비교하고 오른쪽 자식 노드와 비교하여 둘중 더 작은 값이 있는 쪽으로 내려간다. 이는 minheap을 유지하기 위함이다. 그 후 recursive하게 heapify를 호출하며 새로 들어온 node에 대한 수정을 진행한다. swapEdges 함수는 Edge를 swap하는 함수이다.

**- disjoint set**

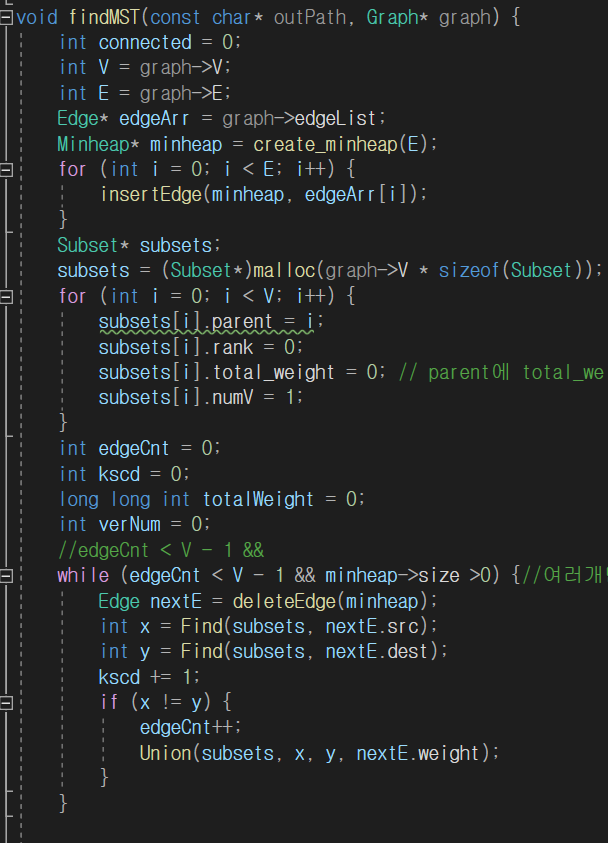
****

disjoint set의 Find함수이다. 시간 효율을 위해 자식 노드의 parent 값을 root로 지정시키도록 하는 경로압축된 Find함수로 수정하였다.

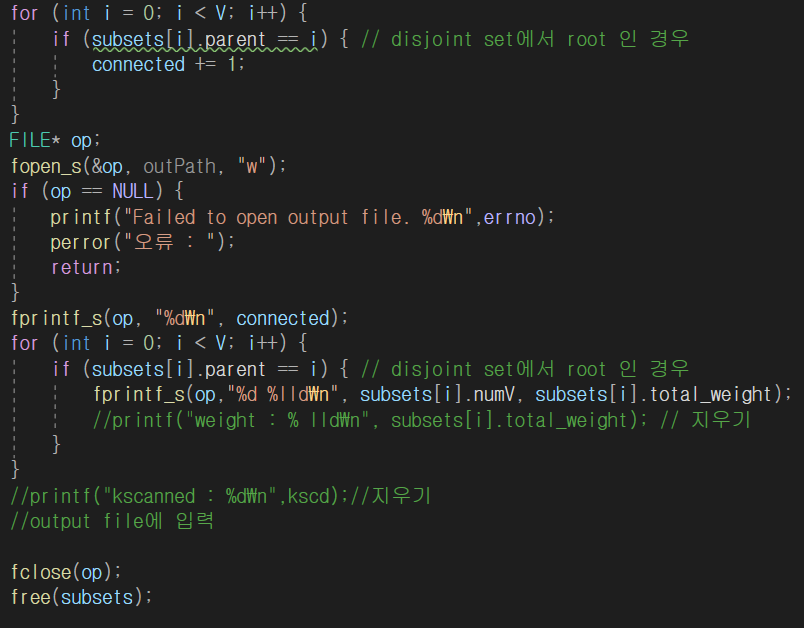


그 다음은 Union함수이다. 기존 Union 함수에서 두 set을 합칠 때, rank와 parent 뿐만 아니라 set에 포함된 vertex개수와 total\_weight까지 전부 root node에 update되도록 구현하였다.

**- Kruskal**



MST를 찾는 findMST 함수이다. 먼저 minheap을 생성하여 Edge를 전부 삽입하고, subsets 배열을 선언하여 초기화 해준다. while문을 통해 edge count가 vertex의 개수 -1 이 되거나, minheap의 size가 0이 되는 순간 break 되도록 구현하였고, edge를 deleteEdge 하여 최소값을 가지는 edge를 뽑아 양 옆 vertex가 같은 set에 포함되어있는지 확인하여 아닌 경우 Union 함수를 호출한다. 그 외의 kscd 변수는 k scanned를 count 하는 변수이다.

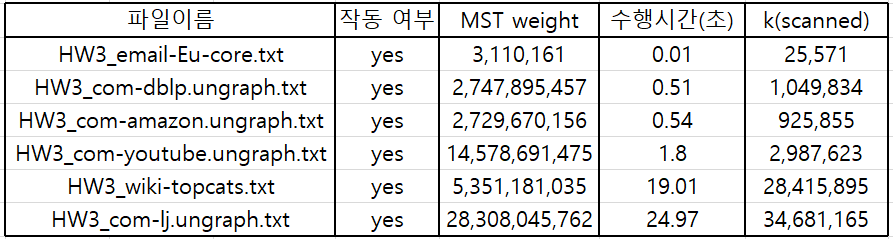


마지막으로 Vertex 개수만큼 한번 subset 배열을 순회하여 connected component 개수를 계산하고, output file에 결과를 출력한다.

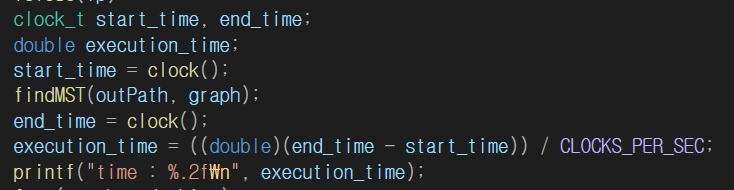
위와 같은 구현을 통해 Kruskal을 구현한 findMST 함수에서 각 edge에 대하여 반복문을 돌리고 vertex의 개수만큼 Find, Union 연산을 진행하기 때문에 O(|E|log|V|) 의 시간복잡도를 가진다. 또한, while문 내에서 edge의 개수만큼 loop을 돌고, deleteEdge 함수는 O(log|E|)만큼의 시간복잡도를 가지고 결국 시간복잡도로 표현하면 O(log|V|)와 같다. 이에 따라 여전히 해당 알고리즘은 O(|E|log|V|)의 시간복잡도를 가진다는 것을 알 수 있다. subsets 초기화, connected count하는 for문은 전부 O(|V|)의 시간복잡도를 가지기 때문에 알고리즘의 시간복잡도에 영향을 끼치지 않는다고 판단하였다.

또한 findMST함수 초반에 minheap에 edge를 insert하는 연산은 O(|E|log|E|)의 시간복잡도를 가지고 이는 O(|E|log|V|)와 같기때문에 해당 알고리즘의 시간복잡도와 동일하여 결론적으로 findMST 함수의 시간복잡도는 O(|E|log|V|)라고 판단하였다.

**C. 보고서에 Kruskal 방법에 대한 자신의 실험 결과를 아래와 같은 양식의 표에 요약하여 제출하라. 조교는 이 내용에 따라 여러분의 프로그램을 컴파일 한 후 그 결과를 확인할 예정이므로 각방법에 대한 구현 여부 및 해당 데이터 처리 여부를 명확히 밝힐 것**

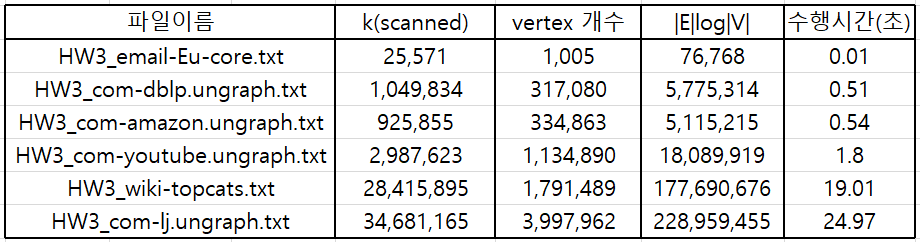


먼저 수행 시간을 측정한 방법은 다음과 같다.



시간을 측정한 기준은 findMST함수가 호출되어 끝날 때까지의 시간을 kruskal 알고리즘이 적용되는 시간이라고 판단하여 측정 후, print하도록하였다. k scanned를 계산한 방법은 B의 findMST 함수 부분에서 설명하였다. 또한 가장 큰 MST\_weight는 결과 파일에서 가장 큰 connected component의 값을 확인하여 표를 완성하였다.

**D. 이어서 여러분이 측정한 수행 시간이 과연 위의 이론적인 시간 복잡도를 반영한다고 할 수 있는지 자신의 분석 결과를 명확히 기술하라**

****

위의 표는 각 파일 별로 k scanned와 vertex의 개수 그리고 마지막으로 |E|log|V|를 계산한 값을 표로 표현하여 수행시간과 비교한 표이다. 계산 후 |E|log|V|와 수행시간을 비교했을 때, 두 값이 비례하는 것을 정확하게 확인할 수 있다. 이를 통해, 구현한 코드에서 while문을 통해 확인하는 edge개수와 O(log|V|)의 시간복잡도를 가지는 연산들을 통해 O(|E|log|V|)의 시간복잡도를 가질 것이라 이론적으로 분석한 결과를 확인할 수 있었다.

**E. 또한, 필요할 경우 채점 시 조교가 알아야 할 것이 있을 경우 명확히 기술하라.**

코드 제출 시에 시간을 측정하여 출력하는 부분과 k scanned를 출력하는 부분은 //주석처리 라는 주석과 함께 주석 처리 해놓았습니다.