알고리즘 설계와 분석(CSE3081-02)

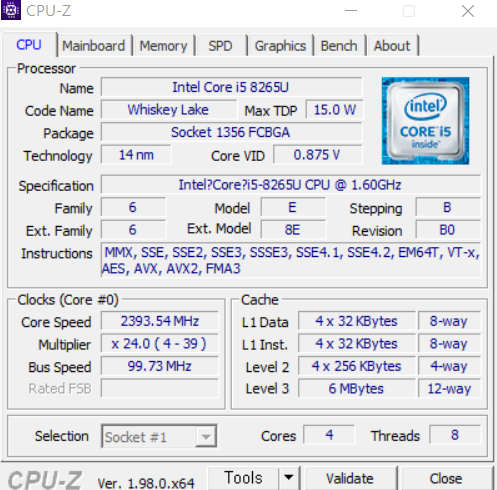
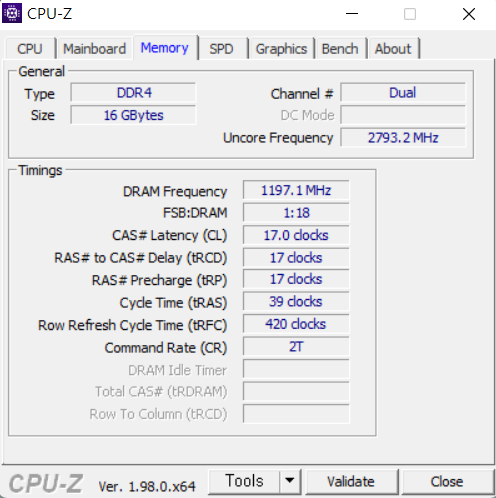
HW4

20191286 김나현

본 과제는 Kruskal의 minimum spanning tree 알고리즘과 disjoint-sets 자료구조를 사용하여 n\_vertices개의 꼭짓점과 n\_edges개의 edge 수가 있을 때, 가장 적은 비용(weight)으로 각 꼭짓점을 잇고, 즉 minimum spanning tree를 찾고 connected component 개수와 각 minimum spanning tree을 이루는 꼭짓점의 개수, 총 weight의 합 등에 대해 출력해보는 것입니다.

다음 표는 주어진 입력 파일에 따른 kruskal MST 알고리즘의 작동 여부와 가장 꼭짓점이 많은 component의 MST weight, 입출력 시간을 제외하고 MST 계산에 소요된 전체 수행 시간, 전체 edge 중 MST를 완성하기 위해 검사한 edge 수를 의미하는 Kscanned을 나타낸 것입니다.



해당 실험은 다음과 같은 CPU와 Memory 성능 환경에서 진행되었습니다.

다음은 위에서 수행한 실험에서 사용한 Kruskal MST 알고리즘을 구현하기 위해 사용한 graph representation 방법과 disjoint-set 구현 방법 등 전반적인 코드에 대해 설명해보겠습니다.

우선, Kruskal MST 알고리즘을 사용하기 위해 edge와 graph, subset을 저장하기 위해 다음과 같이 구조체를 선언하였습니다. Edge라는 구조체 안에는 edge의 각 꼭짓점을 저장하기 위한 v1, v2라는 정수형 변수가 있고, edge의 weight를 저장하기 위한 weight이라는 정수형 변수가 있습니다. Graph라는 구조체 안에는 본 과제에서 weighted undirected graph G로 입력 받는 그래프의 총 꼭짓점 개수와 edge 수를 저장하기 위한 V와 E라는 정수형 변수와 각 edge들을 저장하기 위한 struct Edge 포인터 edge가 존재합니다. 마지막으로 disjoint-sets을 이용하기 위해 선언한 Subset이라는 구조체 안에는 각 subset의 parent와 rank을 저장하기 위한 parent, rank이라는 정수형 변수와 해당 subset에 있는 총 vertex 수와 총 weight의 합을 저장하기 위한 numOfvertices라는 정수형 변수, sumOfweights라는 long long 정수형 변수가 존재합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

메인 함수에서는 struct Graph\* graph를 선언하고 input file을 통해 알게 된 n\_vertices, n\_edges, MAX\_WEIGHT을 각 변수에 저장한 후, n\_edges번의 for loop를 돌며 각 줄에 적힌 두 개의 vertex와 weight를 저장하여 graph->edge[i]에 저장하게 됩니다. 이 과정이 모두 끝나면 kruskalMST라는 함수를 호출하여 본격적으로 mst를 찾게 됩니다. kruskalMST 함수에서는 edge의 weight가 작은 순서대로 edge를 정렬해야 하는데 이때 qsort 함수를 사용하지 않고 minheap을 사용하여 구현하여야 하므로 heapsort 함수와 adjust 함수를 추가적으로 구현하였습니다. heapsort 함수와 adjust 함수는 heap sort를 배울 때 배웠던 함수로, 다음과 같이 어렵지 않게 구현이 가능합니다. Minheap이나 maxheap을 array로 표현하기 위해서는 첫 번째 index부터 사용하지 않아야 하는데 graph->edge에서는 0번째 index부터 순서대로 저장을 하였으므로 메인 함수로 돌아가 struct Edge 포인터 heapEdge를 추가적으로 선언하고 위에서 언급했던 for loop에 graph->edge[i]에 vertex와 weight을 저장한 후, heapEdge[i+1]=graph->edge[i]를 통해 heapEdge의 1번째 index부터 graph의 edge 정보가 저장될 수 있게 구현하였습니다. 그렇게 함으로써 heapsort 함수에서 heapEdge라는 포인터를 이용하여 edge의 weight 오름차순으로 배열 안에 정렬될 수 있도록 하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Heap sort를 통해 edge를 weight가 작은 순서대로 정렬한 후, disjoint-sets 자료구조를 이용하여 kruskal의 MST를 찾기 위해서 Find과 Union 함수를 다음과 같이 구현하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이때, Find 함수는 subsets과 i를 변수로 받아서 i번째 vertex가 포함된 트리의 parent를 찾아 반환하는데 이때 recursive하게 Find 함수를 호출하면서 path compression 기법을 사용하여 i의 parent를 i가 포함된 subset의 root의 vertex가 될 수 있도록 업데이트 해줍니다. Union 함수에서는 subsets과 두 개의 vertex 번호, edge의 weight를 변수로 받아 각 vertex의 parent를 찾고 두 parent 중 rank이 큰 subset에 다른 subset을 붙여주는데 rank이 큰 parent의 sumOfweights와 numOfvertices를 업데이트 해서 해당 subset에 연결된 vertex의 총 개수와 weight들의 합을 업데이트 해주어야 합니다.

kruskalMST 함수에서는 앞서 설명한 heapsort 함수, Find 함수, Union 함수 등이 호출되는, 본 과제에서 가장 중요한 함수입니다. 본 과제에서 입력으로 받는 그래프는 disconnected 그래프가 될 수도 있기 때문에 모든 vertex를 잇는 단 하나의 MST를 찾는 알고리즘이 아니라 여러 개의 connected component가 존재할 수 있다는 것을 염두에 둬야 합니다. 따라서, 해당 vertex가 다른 vertex와 connect된 것인지를 나타내기 위해 vertex 개수 크기의 정수형 포인터 connectedVertex를 선언하여 for loop을 통해 0으로 초기화해주어야 합니다. 마찬가지로 disjoint-sets을 이용하기 위해 subsets이라는 struct Subset 포인터 변수를 선언하여 for loop을 통해 parent를 자기 자신으로, rank를 0으로 sumOfweight를 0, numOfvertices를 1로 초기화해줍니다. 이제 선택된 edge의 개수가 graph->V-1보다 작고 MST를 찾기 위해 거친 edge의 개수가 graph->E보다 작을 때까지 while loop을 돌며 weight가 작은 edge부터 각 꼭짓점이 서로 다른 subset에 있으면 union하고, 그렇지 않으면 다음 edge를 살펴보는 과정을 반복합니다. While loop가 끝나면 output file에 몇 개의 connected component가 있는지 쓰기 위해 for loop을 돌며 subset의 부모와 자기 자신이 같은 vertex가 몇 개인지를 세고 다음 for loop을 돌며 자신이 subset의 부모일 때 numOfvertices와 sumOfweights을 출력하게 합니다. 해당 함수는 다음과 같이 구현할 수 있으며 이때 사용한 struct Edge 포인터 result는 하나의 MST를 구하는 것이 아니며 선택된 edge를 출력하라는 조건이 없으므로 사실상 필요없는 변수라고 할 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

kruskalMST 함수에서 heapsort 함수를 호출하면 E개의 edge이 있는 배열을 max heap으로 만드는 데에 O(E)만큼의 수행 시간이 걸리고, 하나씩 뽑아 오름차순으로 정렬하는 데에 O(ElogE)의 수행 시간이 걸립니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Find와 Union 함수는 각각 O(logV)의 시간이 걸리므로 kruskalMST 함수에서 while 문을 돌면서 MST를 완성하는 데에 최대 O(ElogV)만큼, 일반적으로 O(E+kscanned(logV))만큼의 시간이 걸립니다. 이 코드를 이용하여 수행 시간을 측정해보면 이런 시간 복잡도가 반영된 것을 확인할 수 있습니다.