HW4 결과보고서

20190258 김혜린

## 실험 결과

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 파일 이름 | 작동 여부 | MST weight | 수행시간(초) | kscanned |
| HW4\_com-amazon.ungraph.txt | Yes | 2729670156 | 0.215 | 925863 |
| HW4\_com-dblp.ungraph.txt | Yes | 2747895457 | 0.196 | 1049824 |
| HW4\_com-lj.ungraph.txt | Yes | 28308045762 | 10.906 | 34681189 |
| HW4\_com-youtube.ungraph.txt | Yes | 14578691475 | 0.639 | 2987623 |
| HW4\_email-Eu-core.txt | Yes | 3110161 | 0.002 | 25571 |
| HW4\_wiki-topcats.txt | Yes | 5351181035 | 8.538 | 28416106 |

## 실험 목적

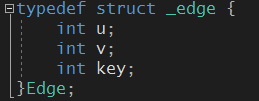
이번 과제에서는 min heap과 disjoint sets을 이용해 kruskal MST algorithm을 구현하는 것을 목표로 한다. 또한 disjoint sets를 이용한 kruskal MST algorithm이 최악의 경우 O(|E|log|V|)의 시간복잡도를 갖고 주어진 graph의 edges를 minheap을 이용한 priority queue를 이용해 저장하고자 할 때 edge를 weight의 크기에 따라 나열하는 연산이 O(|E|+kscanned∙log|V|)의 시간복잡도를 갖는지 확인하고자한다.

## 실험 과정

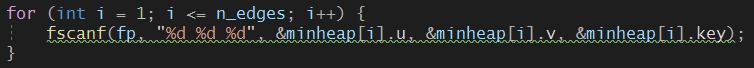
실험 과정은 크게 1) 입력 데이터로 들어온 graph edges의 min heap 구현, 2) min heap과 disjoint sets를 이용한 kruskal MST algorithm,. 3) 결과 저장 세 가지 과정으로 나뉜다.

1. min heap 구현

먼저 입력 파일 첫 줄에 주어진 그래프의 n\_vertices, n\_edges, MAX\_WEIGHT 값을 저장하고 n\_edges 크기를 가지는 minheap이라는 이름의 배열을 할당해 입력 파일의 그래프 edges 정보를 저장하였다.



입력 파일의 그래프 edges 정보는 위와 같은 struture를 이용해 저장하였다. 각각 from-vertex-id, to-vertex-id, edge weight 값을 나타내며 edge weight가 minheap의 key값이 되기 때문에 key라는 변수에 저장해주었다.

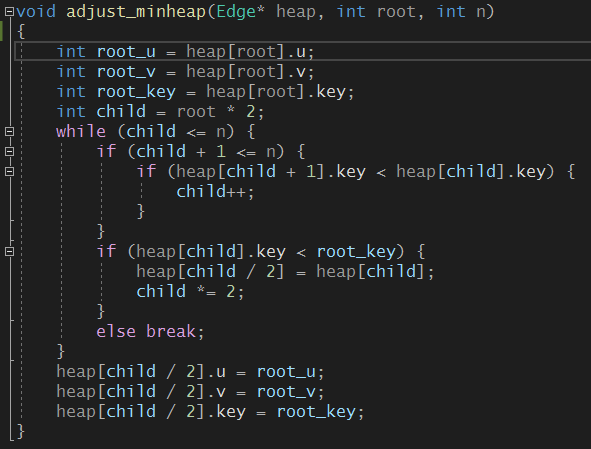


minheap 배열에 edges 정보를 index 1번부터 저장해 child와 parent의 index 관계가 2n, 2n+1의 관계를 가질 수 있도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

n\_edges/2부터 1번 원소까지 돌며 adjust\_minheap 함수를 실행해 i번째 원소를 root로 하는 subtree가 minheap을 만족할 수 있도록 하였다.



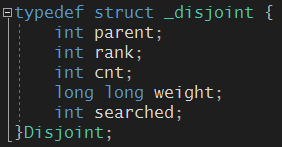
adjust\_minheap 함수를 따로 구현해 unordered 상태의 minheap 배열을 minheap 상태로 바꾸어주었다. |E|개의 원소로 이뤄진 minheap 배열에 대해 |E|/2번 adjust\_minheap 함수를 실행하였고 adjust\_minheap 함수의 경우 depth d를 갖는 원소에 대해 최악의 경우 log2(log2|E|-d)의 시간 복잡도를 가지는 child와의 비교연산을 수행하기 때문에 unordered 상태의 배열을 minheap으로 바꾸어주는 연산은 이론상 O(|E|)의 시간복잡도를 가짐을 알 수 있다.

1. min heap과 disjoint sets를 이용한 kruskal MST algorithm
2. disjoint sets

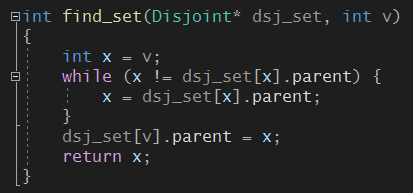
kruskal algorithm에서 매번 minheap 기반 priority queue에서 edge를 가져오고 해당 edge가 MST의 부분 edge가 될 수 있는지 확인하는 과정에서 disjoint sets가 사용된다.

priority queue에서 pop된 edges의 from-vertex와 to-vertex의 parent를 통해 해당 edge가 MST에 포함될 수 있는지 확인하는데 이미 포함된 subMST의 edges로 연결되어있을 경우 두 vertices의 parent가 동일하고 아닐 경우 해당 edge를 subMST에 포함시키며 두 vertices를 union하는 과정을 거친다.

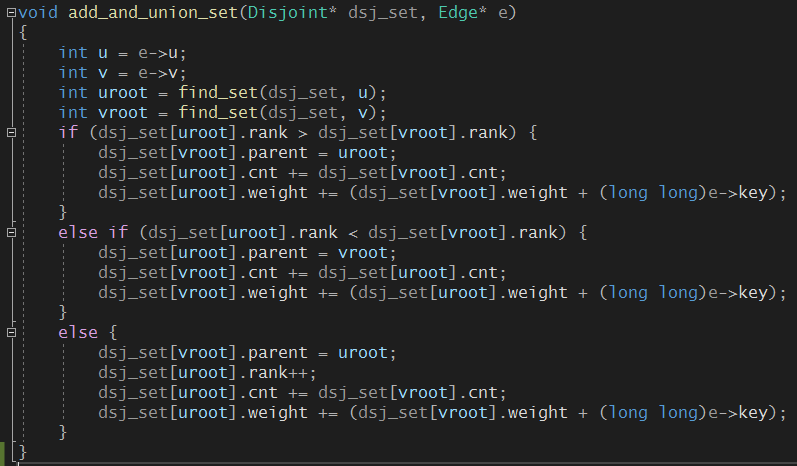
따라서 먼저 n\_vertices의 크기를 가지는 dsj\_set 배열을 할당하고 배열의 각 index에는 해당 id의 vertex에 대한 disjoint set 정보를 저장할 수 있도록 하였다.



dsj\_set의 각 vertex는 위와 같은 structure를 가지고 있으며 parent를 통해 자신의 포함된 set의 대표 원소를 저장하고 rank를 통해 set의 depth를 저장한다. cnt와 weight, searched의 경우 MST 와 관련된 정보를 저장하는 변수로 cnt의 경우 자신이 set의 대표값일 경우 자신을 포함한 set의 총 원소 개수를 나타내고, weight의 경우 지금까지 union된 vertices의 edge들의 weight 총합을 나타낸다.



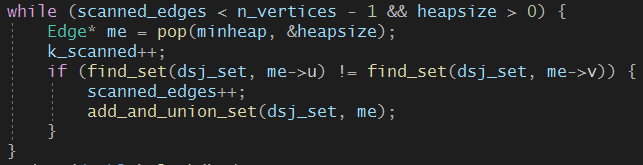
disjoint의 find 함수의 경우 path compression 기법을 이용해 구현하였다. find가 실행될 때마다 while문을 통해 찾은 자신이 속한 set의 대표값을 자신의 parent로 저장하는 방식으로 path compression을 구현하였다. 이로써 같은 parent를 가지는 vertices끼리 하나의 connected component를 만드는 것을 의미하도록 하였다.



disjoint sets의 union의 경우 rank를 기준으로 union 될 수 있도록 구현하였다. find\_set을 통해 edge의 from-vertex와 to-vertex의 parent vertex를 찾은 뒤, rank가 더 높은 set으로 나머지 set이 union될 수 있도록 구현하였다. 두 rank가 같을 경우, 앞 vertex set에 합쳐지고 해당 set의 rank가 1 커지도록 하였다.

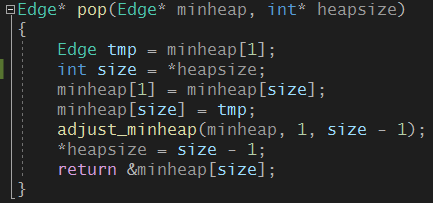
선택된 edge의 weight와 합쳐지는 disjoint의 weight를 union parent의 weight 변수에 더하고 cnt의 경우도 합쳐지는 disjoint set parent의 cnt값을 더함으로써 최종적으로 parent vetex의 cnt와 weight에 각각 connected components의 vertices 개수와 total weight 정보를 저장할 수 있도록 하였다.

1. kruskal algorithm



connected graph의 경우 graph의 connected components가 1개이고 |V|-1개의 edges로 이루어지지만 입력 파일의 graph가 connected graph가 아닐 수도 있기 때문에 heapsize 변수를 추가해 pop 함수가 실행될 때마다 heapsize를 감소시켜 모든 edge를 탐색하였는지도 확인하였다.

pop 함수를 통해 1부터 heapsize 사이의 minheap 원소중 가장 작은 weight를 가지는 edge를 반환받고 edge의 from-vertex와 to-vertex의 find 결과 두 parent가 다를 경우, scanned\_edges를 1추가하고 add\_and\_union\_set함수를 실행해 해당 edge를 추가하고 두 vertices를 union하는 연산을 실행하였다.



pop 함수의 경우 minheap 배열의 1번 index에 저장된 edge를 반환하는 함수로 현재 heapsize의 index를 가지는 원소와 1번 index에 저장된 edge를 swap 하는 연산을 실행함으로써 pop된 함수를 배열의 뒤로 배치해 minheap[size]의 주소값을 반환하도록하였다. 1부터 heapsize-1까지의 minheap에 대해 adjust\_minheap 함수를 실행해 minheap 상태를 유지할 수 있도록 하였다.

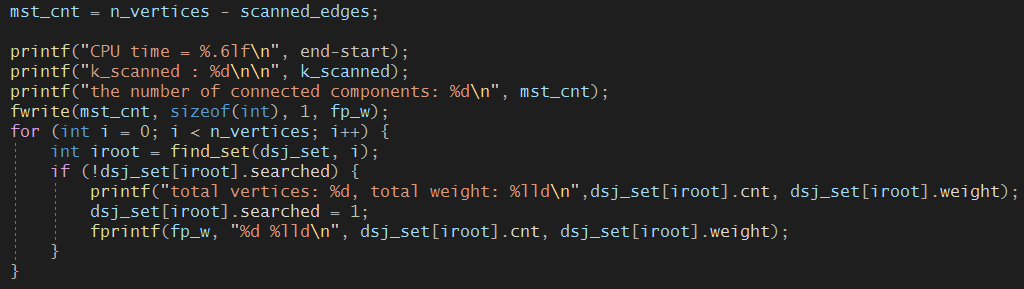
이때 pop된 edge의 개수를 kscanned라고 한다면 kruskal algorithm은 최악의 경우 kscanned의 union-find 연산을 실행하게 되고 최종적으로 O(kscanned∙log|E|)의 시간복잡도를 가지게된다.

1. MST 결과 저장

입력 파일에 저장된 graph가 disconnected graph일 경우, kruskal algorithm 실행 결과 해당 그래프는 2개 이상의 connected components를 가지게된다. 하지만 단순히 weight가 작은 edge를 고르고 두 vertices가 disjoint set에서 연결되었는지 확인하는 과정을 통해서는 해당 그래프가 몇 개의 connected components를 가지는지 알 수 없다.

몇 개의 edge가 kruskal 결과 포함되었는지를 저장하는 변수 scanned\_edges의 값은 connected components가 1개 일 경우 n\_vertices-1의 값을 가지지만 아닐 경우, m개의 connected components를 가지는 그래프에 대해 n\_vertices-m의 값을 가지게된다. 따라서 (n\_vertices-scanned\_edges) 연산을 통해 해당 그래프에 생성된 connected components의 개수를 구하였다.

m개의 connected compoents를 가지는 그래프에 대해 find-union 결과 각 vertex는 자신이 속한 connected component가 어디냐에 따라 다른 parent를 가질 수 밖에 없고 n\_vertices의 dsj\_set에는 m개의 parent가 존재하게 된다.



따라서 n\_vertices 만큼의 dsj\_set 배열을 순회하면서 해당 vertex의 parent의 searched가 0일 경우 parent vertex에 저장된 cnt와 weight 값을 결과 파일에 저장하고 searched를 1로 만들어 dsj\_set에 저장된 connected components에 대한 정보를 저장하였다.

## 결과 분석

실제 실험결과 복잡도가 O(|E|+kscanned∙log|V|)의 복잡도를 가지는지 분석해보면 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 파일 이름 | 수행시간(초) | |E| | |V| | kscanned | |E|+kscannedlog|V| |
| HW4\_email-Eu-core.txt | 0.002 | 25571 | 1005 | 25571 | 102339.388 |
| HW4\_com-dblp.ungraph.txt | 0.196 | 1049866 | 317080 | 1049824 | 6825125.086 |
| HW4\_com-amazon.ungraph.txt | 0.215 | 925872 | 334863 | 925863 | 6041142.086 |
| HW4\_com-youtube.ungraph.txt | 0.639 | 2987624 | 1134890 | 2987623 | 21077543.145 |
| HW4\_wiki-topcats.txt | 8.538 | 28511807 | 1791489 | 28416106 | 46280977.836 |
| HW4\_com-lj.ungraph.txt | 10.906 | 34681189 | 3997962 | 34681189 | 263640803.381 |

|E|+kscannedlog|V|의 값과 연산시간이 비례함을 알 수 있다.

minheap을 생성하는 과정은 |E|만큼의 수행시간이 들고 kruskal 수행시간은 |E|+kscannedlog|V|의 수행시간을 가지는데 , E는 V2에 비례하므로 지수를 고려하지 않았을 때 최종 수행시간은 kscannedlog|V|가 됨을 알 수 있다.