Coputer Algorithms
Project - 04

이름 김태연
학번 201511058
이메일 kim77ty@dgist.ac.kr

요 약

본 프로젝트를 통해서 몇가지의 예제 문제를 풀어보면서 그래프 알고리즘에 대해서 이해하는 것을 목표로 한다. 각 문제들에 대해서 어떠한 알고리즘을 써서 문제를 해결하였는지 설명을 하고, 그 문제에서 사용한 메모리, 시간 복잡도 등에 대해서 설명을 한다.

1. 서론

컴퓨터 알고리즘 수업시간에 배운 그래프 알고리즘들을 실습해보는데 목적을 두고 있다. 그래프란, 컴퓨터의 자료구조 중에 하나로, 각각 edge 와 node 들을 정의하여 지점사이의 관계를 나타낸 것이다. 비슷한 자료구조로 tree 가 있지만, 그래프는 loop 를 만들 수 있다는 점에서 tree 와 차별화된다.

2. 본문

2.1 problem 1 구현

1 번 문제의 경우는 Minimum Spanning Tree 의 대표적인 예시이다. 이를 풀기 위해서 Kruskal's algorithm 을 사용하였다. 코드의 구현에 대해서 간단히 소개하면, 처음에는 set 과 Union 의 구조를 사용하지 않고, 다이나믹 프로그래밍을 통해서 뽑힌 edge 가 다른 edge 들과 loop 를 만드는지 여부를 확인하는 함수를 만들고자 했지만, 다이나믹 프로그래밍을 추가하면서 코드가 너무 복잡해지고, 함수 구조상 재귀호출이 되는데 그 재귀호출의 깊이가 너무 깊어져서 실행시간이 매우 오래 걸릴 것으로 예측하여, 수업시간에 배운 방법대로 구현을 해주었다. 일단 인풋으로 각 edge 의 node 에 해당하는 computer 정보들과 비용을 각각 벡터로 받아줬다. 그리고 cost 값에 대해서 모든 벡터들을 quicksort 알고리즘을 이용해서 sorting 해준 뒤에 cost 가 작은 원소부터 확인해주었으며, 풀이 방법은 Kruskal's algorithm 와 똑같이 각 node 들의 Set 을 만들어준 뒤에, node 가 서로 이어질 때마다 node 들의 Set 을 Union 해주는 방법을 이용하여 구현하였다.

이 알고리즘의 시간 복잡도를 생각해보면, 실제로 계산을 할 때 Union 과정과 같은 다른 부분들은 거의 constant 시간으로 생각해 줄 수 있고, 처음에 인풋으로 받은 cost 들을 sorting 해주기 위해 O(NlogN) (N 은 line 수)의 시간이 소요되며, edge 의 node 들에 대해서 이 node 들이 어떠한 Set 에 들어있는지 찾는데 각각 약 O(n) (n 은 컴퓨터 수)의 시간이 소요된다. 후자의 경우는 이 작업을 line 의 수만큼 반복해야 하기 때문에 O(nN)의 총 시간이 소요된다. 따라서, 총 시간복잡도는 약 O(nN)이 된다.

다음으로, 이 알고리즘의 메모리 사용량에 대해서 생각해보면, 일단 quickSort 에서의 pivot 과 같이 간단한 integer 형 인자를 저장해주기 위한 메모리는 제외를 하면, 기본적으로 인풋 값들을 저장해주기 위해서 line 의 수 * 3*4(int Byte)의 메모리가 필요하며, 다음으로 연결관계를 저장하기 위한 벡터인 Set 에 약 컴퓨터의 수 *4 (int Byte)의 메모리가 필요하다. 하지만, 이번 문제풀이에서는 구현의 편의를 위해서 STL 의 vector library 를 사용했기 때문에, vector 의 구현을 위해 여유 capacity 할당 등 추가적인 메모리가 필요할 것으로 예상된다.

또한, 그래프의 구현에 있어서는 자료구조에 대한 지식이 많이 부족하고, 이번 학기에 C 언어도 처음 배우는 단계라서 많은 한계점을 느꼈다. 일단은 코드가 정상적으로 돌아갈 수는 있도록 구현을 했지만, 처음에 의도했던 대로 다이나믹 프로그래밍을 이용한 코드는 완전히 구현하지 못하고 다른 방법을 사용하였으며, 그 과정에서 비효율적인 부분이 많은 것 같다. 예를 들면, 인풋으로 세 줄에 대한 정보를 각각의 벡터로 받아주어서, 이를 cost 에 대해서 정렬을 시키고자 할 때 내장 sort 함수를 쓰지 못하고, 컴퓨터 알고리즘 프로젝트 1 에서 구현했던 quicksort 알고리즘을 이용해서 정렬되는 리스트가 한 번 swap 될 때 나머지 list 들도 다 같이 swap 이 되도록 구현을 하였는데, 이 부분에 대해서는 시간이 좀 더 있으면 이에 맞는 적절한 자료구조를 찾거나, 또는 간단하게 integer 형 인자 3 개를 member function 으로 갖는 class 를 만들고, 그 class 의 vector 를 이용하는 등 했으면 더욱 효율적으로 코드를 만들 수 있었을 것 같다.

2.1 problem 2 구현

2 번 문제의 경우는, 모든 경로의 shortest path 를 계산하는 문제이다. 모든 경로를 계산해주기 위해서는 floyd warshall 알고리즘을 쓸 수도 있지만, 이번 문제의 경우는 vertiex 의 수가 200 개로 비교적 적은 숫자로 제한되어 있었기 때문에 Dijkstra 알고리즘을 이용해서 한 시작지점에 대한 shortest path 를 구하고 이를 vertex 개수만큼 반복해주도록 구현했다. 사실 이번 문제의 경우는 어느 구조체나 class 를 써야 할지도, 어떻게 구현할지도 감이 안 와서 알고리즘의 구현 부분은 외부의 코드를 참고하였고, 이번 프로젝트를 통해 이러한 알고리즘의 구현이 어떻게 되는지에 대해서 알아보고, 보고서를 통해 그 알고리즘의 효율성에 대해서 생각해보는 것에 초점을 맞추었다. 그래서 어떻게 이 함수를 구현했는지에 대해서 자세히 쓰지는 않고, 다만 priority_queue 를 사용하는 Dijkstra 모델을 사용했으며, 이 알고리즘을 통해 보통은 최단거리와 바로 이전에 들리는 node 의 정보를 얻게 되지만, 문제에서는 최단경로로 가기 위해서 어떠한 node 를 가장 먼저 들려야 하는지를 물어 봤기 때문에, 바로 이전의 정보, parent 라는 벡터를 이용해서 계속 역으로 올라가서 시작점 직후에 어떤 node 를 지나는지 확인해주었고, 이 정보들의 vector 를 반환해주었다.

다음으로는 이 알고리즘의 시간 복잡도에 대해서 생각해보면, Dijkstra 알고리즘에서 최단경로를 계산해주는 부분에서 이 알고리즘은 priority_queue 를 이용했기 때문에 O(Elog(V))의 시간 복잡도가 걸릴 것으로 예측이 되며, 이후에 정답을 출력하기 위해 계산하는 부분이 O(V) 정도의 시간이 추가로 걸릴 것이다. 따라서 총 약 O(Elog(V) + V) 정도의 시간 복잡도가 걸린다는 것을 확인 할 수 있다.

다음으로 이 알고리즘이 어느 정도의 메모리를 사용하는지를 알아보면, Edge 라는 구조체의 vector 를 가지는 graph 라는 벡터와, 결과값을 저장해주는 dis(최단거리 저장), parent(직전에 들리는 node 저장), result(출력될 값 저장)의 int 형의 벡터가 필요하다. 또한, 이 세 개의 벡터들을 dijkstra 알고리즘이 총 vertex 개수만큼 불리므로, 이 횟수만큼의 메모리가 필요하게 된다. 그러면 일단 graph 를 저장해주기 위한 메모리로 egde 수 * 16 (Byte)정도가 들 것이며, 결과를 저장하기 위해서는 vertex 수 * vertex 수 * 3 * 4 (Byte) 정도의 메모리가 필요할 것이라고 생각할 수 있다. 이 부분 역시 STL 과 구조체에서 정확히 어느 정도의 메모리를 사용하는지 정확히 알지 못하기 때문에 오차가 있을 수 있다.

그래프 구현에 있어서는 그 알고리즘을 직접 구현을 해보지는 못해서 아쉽지만, 알고리즘을 직접 사용하고 분석해보면서 그 알고리즘에 대해서 더욱 명확하게 알 수 있는 좋은 기회였던 것 같다.

참고문헌

[1] 본 레포트 양식은 국내 학술대회 논문을 참고하였습니다.

[2] Introduction to Algorithms (Thomas H. Cormen)

[3] 컴퓨터 알고리즘 강의노트

[4] Dijkstra 알고리즘 C++ 코드 참고 https://github.com/cannalee90