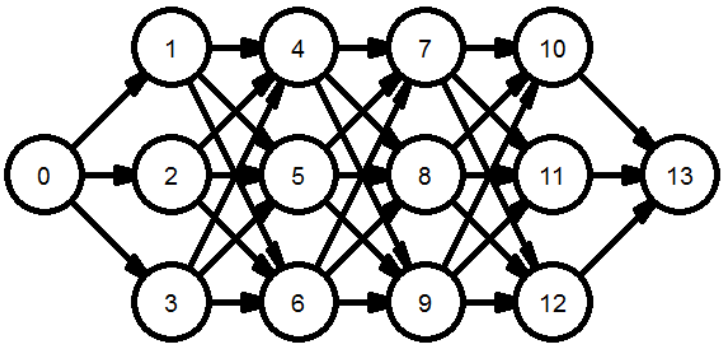
선형대수 프로젝트 보고서

Problem4

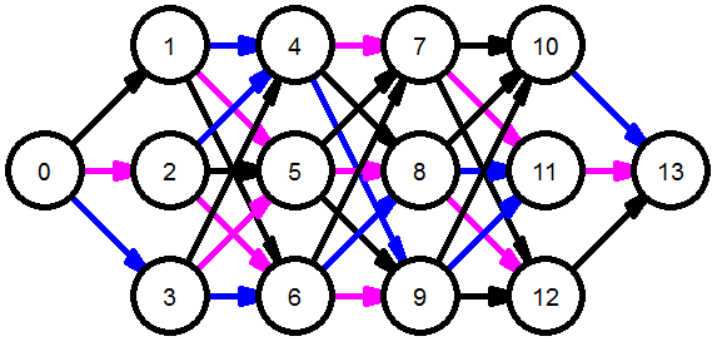
이름: 배지운

학번: 2015004584

1. Introduction



선택한 문제는 4번으로 크게 두 가지 문제가 주어진다. 공통적인 개요는 네트워크가 그래프의 형태(V, E)의 꼴로 주어지고 시작점, 끝점 (s, t) 가 주어질 때 s(0)에서 t(13)로 데이터 M을 전송하고자 할 때 각 E마다 최대 통신 가능한 데이터 량(f)이 정해져 있고 통신료(d)가 주어져 있고 데이터 f를 d의 통신료로 전송하면 d*f 만큼의 비용이 들 때 1) 모든 간선 E의 통신료가 동일 할 때, s에서 t로 데이터M 을 전송할 때 각 V에서 연결된 간선이 위에서부터 차례로 3,5,10의 전송 량을 갖고 t로 연결된 간선도 동일하게 3,5,10의 전송 량을 갖는다고 주어진다면 1초 동안 s에서 t까지의 최대 데이터 통신 량을 구하는 문제와 2) 통신료가 아래의 그림과 같이 검은색 2원, 파란색 4원, 분홍색 5원으로 주어져있고 M=9이며 f가 문제 1) 과 같다고 할 경우 데이터 전송에 드는 최소 비용을 구하는 문제이다.



문제를 처음 보았을 때 든 생각은 MCFP(minimum-cost flow problem)이었다. 최근 ACM-ICPC본선에도 진출했었고, SCPC 예선을 꽤 높은 점수로 통과했을 만큼 평소에 알고리즘 공부를 열심히 하고 있어서 다행히 알고 있는 알고리즘이었다. 하지만 대회에서 문제를 푸는 것과 알고리즘에 대해 자세히 이해하고 개선해보는 것은 다르기 때문에 좀 깊게 알아보고 해당 문제에 대해 적용할 때 개선할 수 있는 부분이 있는지 더 자세히 알아보고자 했다.

2. Related Works

먼저 주어진 문제는 유량이 주어진 그래프에서 흐를 수 있는 최대 유량을 구하는 문제1)와 유량과 비용이 주어진 그래프에서 유량을 만족하는 최소 비용을 찾는 문제2)이다.

이를 위해서는 어떠한 가중치를 기준으로 최소 비용을 검색하는 방법을 알아야 하고 Network Flow에서 Maximum flow를 구할 수 있어야 한다. 끝으로 2번 문제를 해결하기 위해서는 Minimum Cost Maximum Flow라고 불리는 문제의 유형에 대해서 알고 있는 것이 좋다.

최단경로를 구하는 Algorithm은 보통 Floyd-Warshall Algorithm이나 Bellman-Ford Algorithm혹은 Dijkstra Algorithm (가중치가 양수일 경우)을 사용하게 된다. (어떠한 사이클도 가중치의 합이 음수가 없다면 최단경로를 찾을 수 있다.)

Floyd-Warshall Algorithm은 time complexity가 $O(n^3)$으로 매우 간단한 알고리즘이다. 기본적으로 DP(Dynamic Programming)적 접근이 쓰이며, 어떤 점 V_i부터 V_j까지의 최단경로를 모두 구할 수 있다. 거쳐가는 꼭지점, 출발점, 도착점에 대해 반복을 실행하면서 최단경로를 갱신해주는 것으로 쉽게 이해할 수 있다.

Bellman-Ford Algorithm은 time complexity가 주어진 $G(V, E)$에 대해 $O(VE)$다. 시작점에서 출발하여 최단경로를 계속 갱신하는 작업을 거치면 된다.

Dijkstra Algorithm은 주어진 $G(V, E)$에서 각각의 v에 대해 s에서 v까지의 최단 경로 d[v]를 저장하면서 작동한다. Greedy Algorithm을 사용하면서 s에서의 최단거리가 확정된 정점의 집합에서 어떤 정점 v까지의 거리가 최소인 정점 v를 선택하면 다음 최단 거리가 확정된 집합에 v를 추가할 수 있고 이런 식으로 반복하다 보면 모든 정점에 대해 최단거리를 구할 수 있다는 사실을 바탕으로 한다.

유량을 어떻게 흘려 보낼 것인가에 대한 문제는 Network Flow에 해당하는 문제로 여기서는 Maximum Flow를 구한다. 기본적으로 Network Flow는 간선에 흐를 수 있는 유량은 간선의 용량보다 작아야 하고 시작 점과 끝 점을 제외한 모든 점에서의 나가는 유량과 들어오는 유량의 합은 같아야 한다는 속성을 가지고 있는다.

이 Maximum Flow(최대 유량)를 구하기 위해서는 경로를 찾은 후에 그 경로에서 제일 작은 용량만큼 각각의 유량을 더해주는 방식으로 진행된다. 이를 이용한 Ford-Fulkerson Algorithm을 사용하면 time complexity $O(Ef)$ $(f\ is\ maximum\ flow)$ 에 구할 수 있다. (BFS(Breadth-first search)를 사용하여 탐색하면 f를 검색하는데 VE이상 찾지 않는다는 것이 증명되었다고 한다. 따라서 시간 복잡도는 $O(EV^2)$이 된다.

3. Presets

코드의 구현보다 알고리즘에 무게를 두어 대부분 STL과 C++11문법을 대거 사용하여 코드의 길이를 줄이고 각 코드의 의미를 명확하게 하는데 중점을 두었다.

또한 풀이에 해당하는 알고리즘의 실행시간을 정확히 측정하기 위해 입/출력과 시간 측정하는 부분을 공통되도록 작성하고 여러 형태의 그래프를 이용하여 각각 1000번 실행하여 평균을 측정하는 것으로 알고리즘의 수행속도를 측정하도록 하였다. 동일한 조건에서 수행 속도를 측정하기 위해서 구현부분의 함수를 제외한 코드는 모두 같게 했고 실행 시간 측정에 사용되는 함수들은 첨부된 measure.h의 measure namespace에 작성되어 있고 입/출력은 STL의 입출력 표준인 fstream과 iostream을 사용하였다.

사용된 그래프는 첨부된 data폴더에 담겨있다. 문제에서 주어진 그래프는 graph0이다.

사용된 프로세서의 정보는 다음과 같다.

Intel® Core i7 4702MQ@2.2Ghz / 4C8T

4. Problem 1

A. Solution

문제 1)은 생각보다 쉽게 풀 수 있다. t와 t로 전송되는 컴퓨터를 제외한 각각의 모든 점에서 나가는 데이터 전송량의 합은 18이고 t로 들어오는 전송량의 합도 18이다. 따라서 s에서 18만큼 전송할 수 있고 중간에 연결된 어느 컴퓨터도 18보다 적게 전송할 수 있는 컴퓨터는 없으므로 t에서 18을 받을 수 있다는 것은 다음과 같은 추론을 할 수 있다.

주어진 그래프 $G((u,v) \in V,E)$에 대해 $E(u,v)$를 $u\sim v$일 때 u에서 v로 전송할 수 있는 전송량이라 하고, $O(u)$를 $\forall v\sim u \rightarrow \sum_{v}^{k} E(u,k)$, $I(u)$를 $\forall v\sim u \rightarrow \sum_{u}^{k} E(k,v)$라 하자. 그러면 그래프 G에 대해 아래의 식이 만족한다.

$$\forall u\sim s \rightarrow \sum_{u}^{k} I(k) = 18$$

$$\forall u\sim t \rightarrow \sum_{u}^{k} O(k) = 18$$

$$\forall u \neq \{s,t\}\ and\ !(u\sim\{s,t\}) \rightarrow I(u) = O(u) = 18$$

따라서 t에 연결된 컴퓨터를 제외한 s에서 t로의 전송되는 어떠한 컴퓨터도 최소한 18의 전송할 수 있고 s에서 연결된 컴퓨터를 제외한 s에서 t로 전송되는 어떠한 컴퓨터도 최소한 18을 전송 받을 수 있다. s에서 시작되는 전송량의 합은 18이므로 t에 연결된 컴퓨터들의 전송량의 합도 18이므로 t에 전송되는 전송량도 18이다.

하지만 일반적인 풀이인 Maximum Flow로 문제를 확장했을 때 알고리즘을 구상하기란 쉽지 않다.

Maximum Flow를 구하는 문제로 보통 BFS(Breadth-first search)로 증가 경로를 탐색하여 flow를 보내주는 방식으로 해결할 수 있다.

주어진 그래프 $\mathrm{G(V,E)}$에서 u에서 v로 가는 경로에 대해 $c(u,v)$가 용량(총 전송량)을 $f(u,v)$가 유량(실제 전송량)을 나타낸다고 할 때 다음이 성립함을 확인할 수 있다.

$$\forall(u,v) \in E, \qquad f(u,v) \le c(u,v)$$
$$\forall(u,v) \in E, \qquad f(u,v) = -f(v,u)$$
$$\forall u \neq \{s,t\} \sum_{v \in E} f(u,v) = \sum_{v \in E} f(v,u)$$

이를 이용해 s에서 t로 가는 어떤 경로 p에 대해 $c(u,v) - f(u,v) > 0$을 만족하는 최소의 p_c를 선택해 $f(u,v) = f(u,v) + p_c$와 $f(v,u) = f(v,u) - p_c$를 계속 갱신해 나가면서 $c(u,v) - f(u,v) > 0$을 만족하는 경로가 없을 때까지 진행하면 전송될 수 있는 모든 전송량에 대해 체크한 셈이므로 s에 연결된 u에 대해 $\sum_k^u f(s,k)$가 답이 된다.

주어진 문제에 해당하는 답은 18로 1000번 실행하는데 0.077초가 걸렸으며 평균 7.7-e005초가 걸렸다.

B. Improve

사용한 방법으로 문제를 해결하는데 불필요하게 접근하거나 갱신하는 부분은 없는지 살펴보았다.

처음으로 생각한 방법은 그래프에서 경로를 탐색하는 시간을 줄이는 것이었다. BFS로 경로를 탐색하고 있으나 퇴각검색이나 Greedy한 접근으로 해가 되지 않는 부분을 미리 제거하거나 지역 최적 해를 전역 최적해와 동치임을 보이고 싶었지만 쉽지 않았다.

그래서 기존의 방법에 DP를 응용하여 해당 경로를 역으로 추적하는 게 아니라 모든 경로에 대한 값들을 미리 저장해두고 갱신하는 방법을 사용해보기로 하였다.

C. Analysis and Discussion

여기에 분석 추가

5. Problem 2

A. Solution

문제 2)는 flow가 정해져 있으므로 최소 비용 경로를 계속 파악하여 flow를 만족시킬 때까지 탐색하면 된다.

이러한 문제는 MCMF(Minimum cost Maximum Flow)와 Hungarian Method를 사용하여 해결할 수 있는데 좀 더 익숙한 MCMF를 사용하기로 했다.

MCMF는 1번 문제였던 maximum flow와는 조금 다른데 1번 에서 고려해야 할 대상이 유량(전송량) 하나였다면 이번엔 비용을 추가적으로 고려해 줘야 한다. 이 비용을 최소로 하면서 주어진 전송량 (M=9)를 만족할 때 비용을 찾으면 된다.

주어진 그래프 $G((u,v) \in V,E)$에서 $f(u,v)$를 u에서 v로의 최대 전송량 $c(u,v)$를 u에서 v로의 비용이라고 생각했을 때 다음이 성립한다.

$$c(u,v) = -c(v,u)$$

Cost가 최소인 경로를 계속 찾아나가면서 flow가 주어진 값을 만족시킬 때까지 cost를 더해주면 된다.

보통 범용적인 이런 문제를 Minimum Cost Maximum Flow 라고 하고 최소 비용 경로를 찾을 때 Dijkstra Algorithm이 Bellman-Ford Algorithm보다 빠르지만 가중치(비용)이 음수인 간선이 생길 수 있으므로 Bellman-Ford Algorithm을 사용하는 것이 좋다.

주어진 문제에 해당하는 답은 109로 1000번 실행하는데 0.171초가 걸렸으며 평균 0.000171초가 걸렸다.

B. Improve

여기에 문제2에 대한 개선 과정 추가

C. Analysis and Discussion

여기에 문제2에 대한 결론 추가

6. Concluding Remarks

7. Review

사실 minimum cost maximum flow 문제는 기억에 잘 남을 수 밖에 없는 문제였다. 얼마 전에 치러진 ACM-ICPC 본선 대회에서 나온 문제 중 하나였고, 우리 팀은 팀 노트에 이 MCMF의 코드를 미리 적어가지 않아서 다른 문제를 풀다가 남은 시간으로는 구현할 수 없어서 아쉽게도 풀지 못했던 문제였다. (게다가 동상이랑 겨우 1문제 차이였다!)(mcmf.png)

먼저 관련된 지식을 알고 있어서 문제를 접근하는데 매우 수월했지만 오히려 문제를 알고 있었기 때문에 색다른 풀이나 개선점을 찾기 어려웠다.

Jim Orlin's algorithm은 O(VE)에 구할 수 있다는데 짐작도 안 간다.