

선형대수 프로젝트 보고서

Problem4

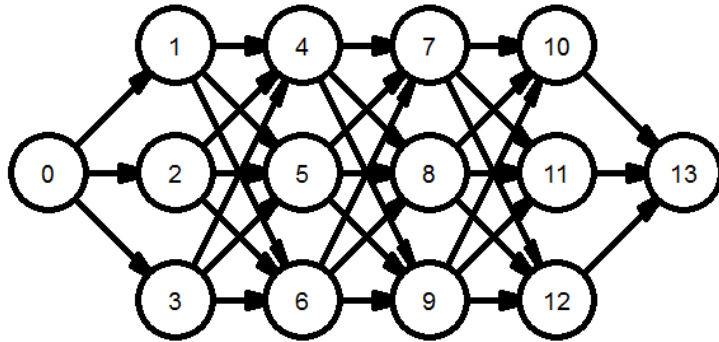
이름: 배지운

학번: 2015004584

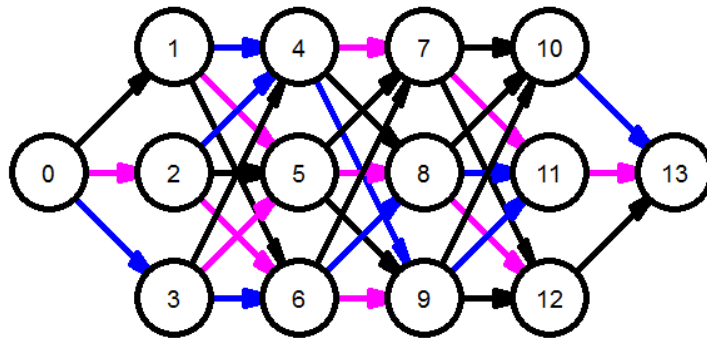
Index

1. Introduction	3
2. Related Works	4
3. Presets	5
4. Problem1	6
A. Solution		
B. Improve		
C. Analysis and Discussion		
5. Problem2	9
A. Solution		
B. Improve		
C. Analysis and Discussion		
6. Concluding Remarks	12
7. Reference	12

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 M 을 전송하고자 할 때 각 E 마다 최대 통신 가능한 데이터 량(c)이 정해져 있고 통신료(d)가 주어져 있고 데이터 f 를 d 의 통신료로 전송하면 $d \times f$ 만큼의 비용이 들 때 1) 모든 간선 E 의 통신료가 동일 할 때, s 에서 t 로 데이터 M 을 전송할 때 각 v 에서 연결된 간선이 위에서부터 차례로 3, 5, 10의 전송 량을 갖고 t 로 연결된 간선도 동일하게 3, 5, 10의 전송 량을 갖는다고 주어진다면 1초 동안 s 에서 t 까지의 최대 데이터 통신 량을 구하는 문제와 2) 통신료가 아래의 그림과 같이 검은색 2원, 파란색 4원, 분홍색 5원으로 주어져있고 $M = 9$ 이며 f 가 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 Algorithm 혹은 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_i 부터 v_j 까지의 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 $u \in V$ 인 u 에 대해 s 에서 u 까지의 최단 경로 $d[u]$ 를 저장하면서 작동한다. Greedy Algorithm을 사용하면서 s 에서의 최단거리가 확정된 정점의 집합에서 어떤 정점 v 까지의 거리가 최소인 정점 v 를 선택하면 다음 최단거리가 확정된 집합에 v 를 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 있다.

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 f 를 검색하는데 VE 이상 찾지 않는다는 것이 증명되었다고 한다. 따라서 시간 복잡도는 $O(VE^2)$ 이 된다.

3. Presets

코드의 구현보다 알고리즘에 무게를 두어 대부분 STL과 C++11문법을 대거 사용하여 코드의 길이를 줄이고 각 코드의 의미를 명확하게 하는데 중점을 두었고 대부분의 코드 위에 주석을 달아 해당 변수나 함수가 무슨 기능을 하는지 적어두었다.

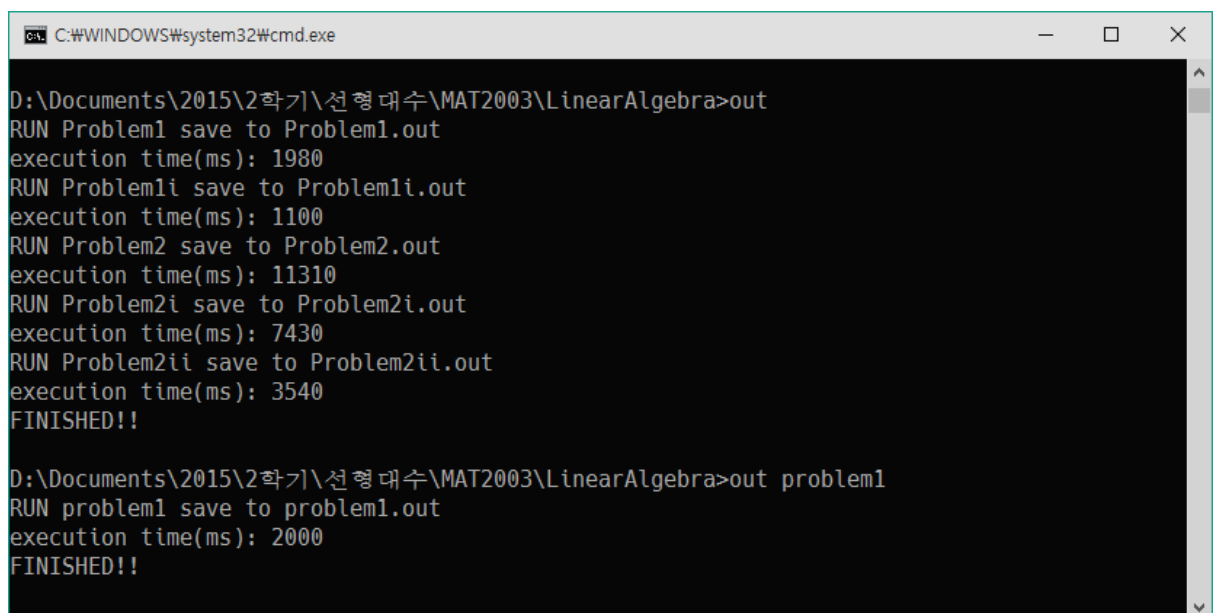
또한 풀이에 해당하는 알고리즘의 실행시간을 정확히 측정하기 위해 입/출력과 시간 측정하는 부분을 공통되도록 작성하고 여러 형태의 그래프를 이용하여 각각 10000번 실행하여 평균을 측정하는 것으로 알고리즘의 수행속도를 측정하도록 하였다. 동일한 조건에서 수행 속도를 측정하기 위해서 구현부분의 함수를 제외한 코드는 모두 같게 했고 실행 시간 측정에 사용되는 함수들은 첨부된 `measure.h`의 `measure` namespace에 작성되어 있고 입/출력은 STL의 입출력 표준인 `fstream`과 `iostream`을 사용하였다.

사용된 그래프는 첨부된 `data`폴더에 담겨있다. 문제에서 주어진 그래프는 `graph0`이다.

미리 만들어둔 `graph`는 0부터 9까지 10개가 있으며 각각 여러 형태의 그래프를 일반화하였다.

첨부된 `out.bat` 파일을 사용하여 모든 방법에 대한 로그(*.out)을 한 번에 갱신하거나 뒤에 실행파일의 이름을 덧붙여 해당 실행파일의 로그를 갱신할 수 있다.

Example)



```
C:\WINDOWS\system32\cmd.exe

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>out
RUN Problem1 save to Problem1.out
execution time(ms): 1980
RUN Problem1i save to Problem1i.out
execution time(ms): 1100
RUN Problem2 save to Problem2.out
execution time(ms): 11310
RUN Problem2i save to Problem2i.out
execution time(ms): 7430
RUN Problem2ii save to Problem2ii.out
execution time(ms): 3540
FINISHED!!

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>out problem1
RUN problem1 save to problem1.out
execution time(ms): 2000
FINISHED!!
```

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. t 와 t 로 전송되는 컴퓨터를 제외한 각각의 모든 점에서 나가는 데이터 전송량의 합은 18이고 t 로 들어오는 전송량의 합도 18이다. 따라서 s 에서 18만큼 전송할 수 있고 중간에 연결된 어느 컴퓨터도 18보다 적게 전송할 수 있는 컴퓨터는 없으므로 t 에서 18을 받을 수 있다는 것은 다음과 같은 추론을 할 수 있다.

주어진 그래프 $G(V, E)$ 에 대해 $(u, v) \in V$ 이고 $E(u, v)$ 를 $u \sim v$ 일 때 u 에서 v 로 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 G 에 대해 아래의 식이 만족한다.

$$\forall u \sim s \rightarrow \sum_u^k I(k) = 18$$

$$\forall u \sim t \rightarrow \sum_u^k O(k) = 18$$

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

따라서 t 에 연결된 컴퓨터를 제외한 s 에서 t 로의 전송되는 어떠한 컴퓨터도 최소한 18의 전송할 수 있고 s 에서 연결된 컴퓨터를 제외한 s 에서 t 로 전송되는 어떠한 컴퓨터도 최소한 18을 전송 받을 수 있다. s 에서 시작되는 전송량의 합은 18이고 t 에 연결된 컴퓨터들의 t 로의 전송량의 합도 18이므로 t 에 전송되는 전송량도 18이다.

이제 일반적인 풀이인 Maximum Flow로 문제를 해보자.

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

주어진 그래프 $G(V, E)$ 에서 u 에서 v 로 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.

$$\forall (u, v) \in E, \quad f(u, v) \leq c(u, v)$$

$$\forall (u \sim v) \in E, \quad f(u, v) = -f(v, u)$$

$$\forall u \in V \setminus \{s, t\} \sum_{u \sim v \in E} f(u, v) = \sum_{v \sim u \in E} f(v, u)$$

이를 이용해 s 에서 t 로 가는 어떤 경로 p 에 대해 $c(u, v) - f(u, v) > 0$ 을 만족하는 최소의 p_c 를 선택해 $f(u, v) = f(u, v) + p_c$ 와 $f(v, u) = f(v, u) - p_c$ 를 계속 갱신해 나가

면서 $c(u, v) - f(u, v) > 0$ 을 만족하는 경로가 없을 때까지 진행하면 전송될 수 있는 모든 전송량에 대해 체크한 셈이므로 s 에 연결된 u 에 대해 $\sum_k f(s, k)$ 가 답이 된다.

주어진 문제에 해당하는 답은 18로 1000번 실행하는데 0.078초가 걸렸다.

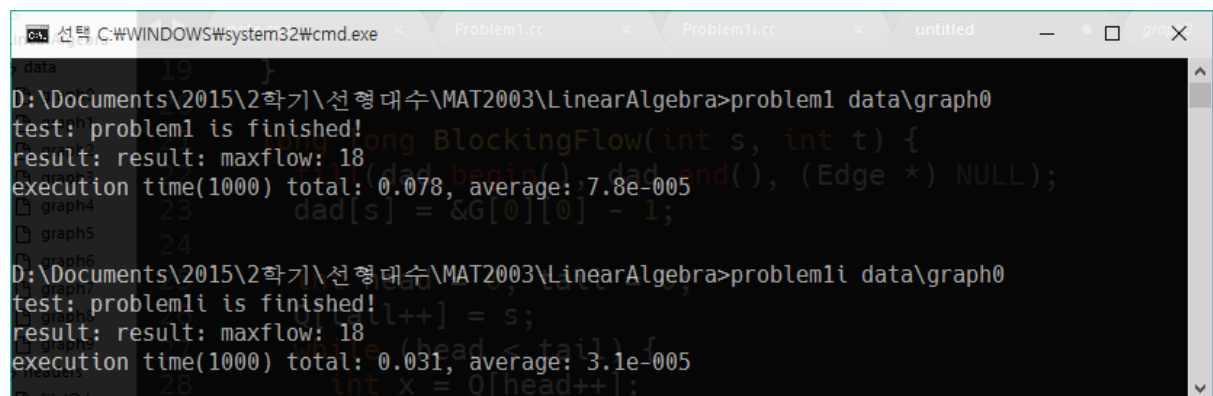
B. Improve

사용한 방법으로 문제를 해결하는데 불필요하게 접근하거나 갱신하는 부분은 없는지 살펴보았다. 경로를 탐색하는 방법은 임의의 경로로 전수 탐색을 해야 하니 더 이상 줄일 수 없을 것 같았고 flow를 갱신하는 과정에서 중복되지 않도록 처리하는 방법에 대해 생각해 보았다.

위에서 사용한 방법에 의해서 주어진 그래프에서 $\forall u, v \in V$ 에 대해 $f(u, v) = c(u, v)$ 이 성립하는 간선은 무조건 지나야 하는 것을 알 수 있다.

문제에서 위의 풀이대로라면 어떤 간선에 유량이 흐르려면 그 간선을 포함하는 경로에서 포함된 간선중 최소 값만큼만 흐를 수 있다. 현대 이 간선의 유량이 용량만큼 전부 흘렀다는 것은 이 간선을 포함하는 경로에서 이 간선을 무조건 지나야 한다는 뜻이다. 이를 이용하여 지나야만 하는 간선을 확정하는 방식으로 문제를 해결 해 볼 수 있다.

이를 이용한 풀이는 Problem1i.cc 에 작성되었으며 1000번 실행하는데 0.031초가 걸려 기존의 방법보다 빠르게 작동함을 알 수 있다.



```
선택 C:\WINDOWS\system32\cmd.exe
D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem1 data\graph0
test: problem1 is finished!
result: result: maxflow: 18
execution time(1000) total: 0.078, average: 7.8e-005

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem1i data\graph0
test: problem1i is finished!
result: result: maxflow: 18
execution time(1000) total: 0.031, average: 3.1e-005
```

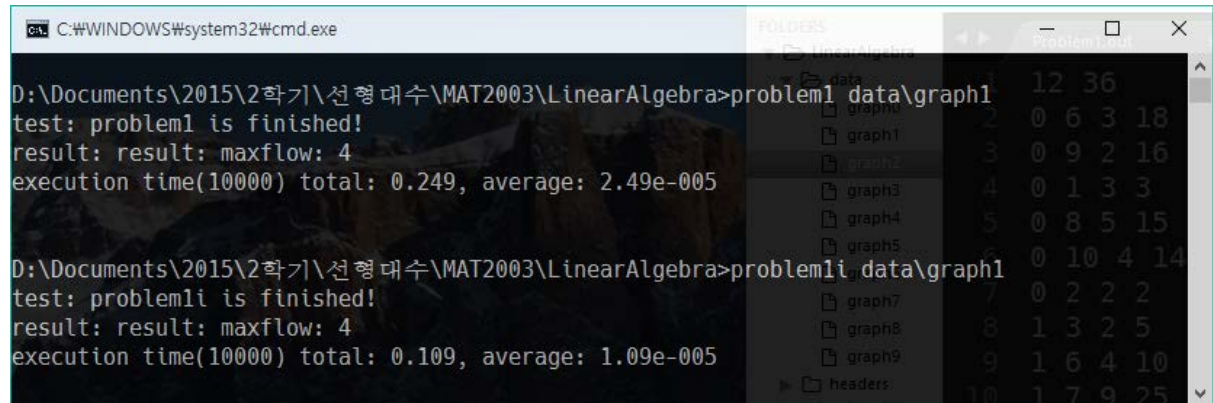
C. Analysis and Discussion

그래프를 알고리즘에 크게 영향을 미칠 수 있는 요소인 V 와 E 에 대해 $V \approx E$ 인 경우와 $V \ll E$ 인 경우로 나누어서 생각해 보기로 하였다.

Graph1은 $V = 12, E = 12$ 로 주어진 그래프이고 Graph2 $V = 12, E = 36$ 으로 $V \ll E$ 인

경우이다.

Graph1에 대해 Problem1 과 Problem1i의 실행시간은 각각 0.25, 0.11초로 결과는 다음과 같다.

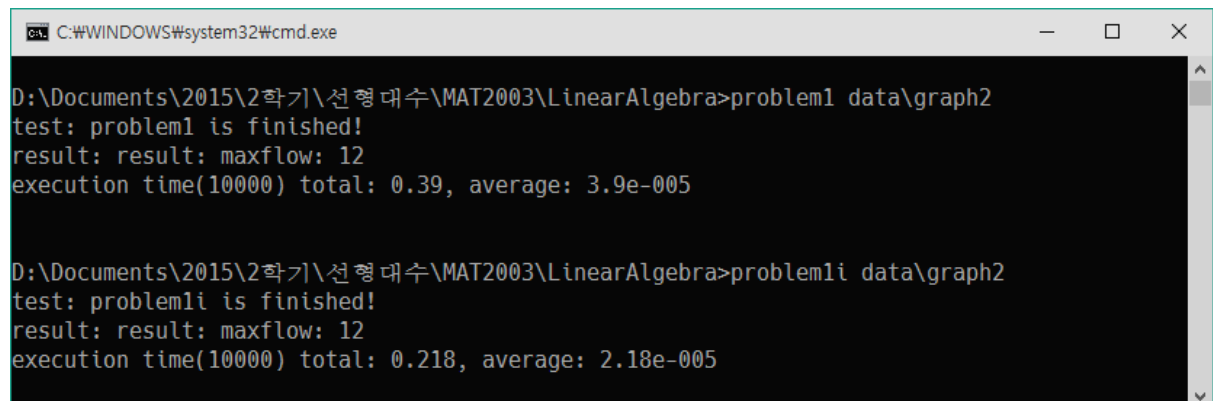


```
C:\WINDOWS\system32\cmd.exe

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem1 data\graph1
test: problem1 is finished!
result: result: maxflow: 4
execution time(10000) total: 0.249, average: 2.49e-005

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem1i data\graph1
test: problem1i is finished!
result: result: maxflow: 4
execution time(10000) total: 0.109, average: 1.09e-005
```

Graph2에 대해서는 각각의 실행시간은 0.39, 0.22초이고 실행결과는 다음과 같다.



```
C:\WINDOWS\system32\cmd.exe

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem1 data\graph2
test: problem1 is finished!
result: result: maxflow: 12
execution time(10000) total: 0.39, average: 3.9e-005

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem1i data\graph2
test: problem1i is finished!
result: result: maxflow: 12
execution time(10000) total: 0.218, average: 2.18e-005
```

하지만 Graph8, Graph9에 대해서는 Problem1i의 방식이 조금 더 느린 것을 확인할 수 있었는데 Graph8의 경우는 s에서 t로 가는 간선보다 t에서 s로 거꾸로 향하는 간선이 많은 그래프 이고, Graph9는 음수인 flow가 주어졌을 때 해당하는 그래프이다.

이 두 가지 경우에 대해서는 Problem1i의 방식이 조금 더 느린 것을 확인 할 수 있다.

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 비용 C 을 찾으려 한다.

주어진 그래프 $G((u, v) \in V, E)$ 에서 $f(u, v)$ 를 u 에서 v 로의 실제 전송량 $c(u, v)$ 를 u 에서 v 로의 전송량의 최대값, $a(u, v)$ 를 u 에서 v 로의 가격이라고 할 때 다음이 성립한다.

$$f(u, v) \leq c(u, v)$$

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

$$\forall u \in V \setminus \{s, t\} \rightarrow \sum_{w \in V} f(u, w) = 0$$

$$\sum_{w \in V} f(s, w) = \sum_{x \in V} f(x, t) M$$

Cost가 최소인 경로를 계속 찾아나가면서 flow가 주어진 값을 만족시킬 때까지 cost를 더해주면 된다. 다른 방법은 문제 1)에서와 같다.

Cost가 최소인 경로를 찾고, 그 경로에 포함되는 간선중 최소인 flow값을 선택하여 모든 간선들의 flow에서 빼주고 반대방향으로 flow가 흐르도록 새로 flow를 생성한다. 이때 새로 생성된 flow의 cost값은 원래 cost의 -1을 곱해 다음 cost가 최소인 경로를 찾을 때 끼이지 않도록 해준다.

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 좋다고 판단했다.

Bellman-Ford Algorithm을 사용하여 푼 코드는 Problem2.cc에 적혀있고 주어진 문제에 해당하는 답은 109로 1000번 실행하는데 0.331초가 걸렸다.

B. Improve

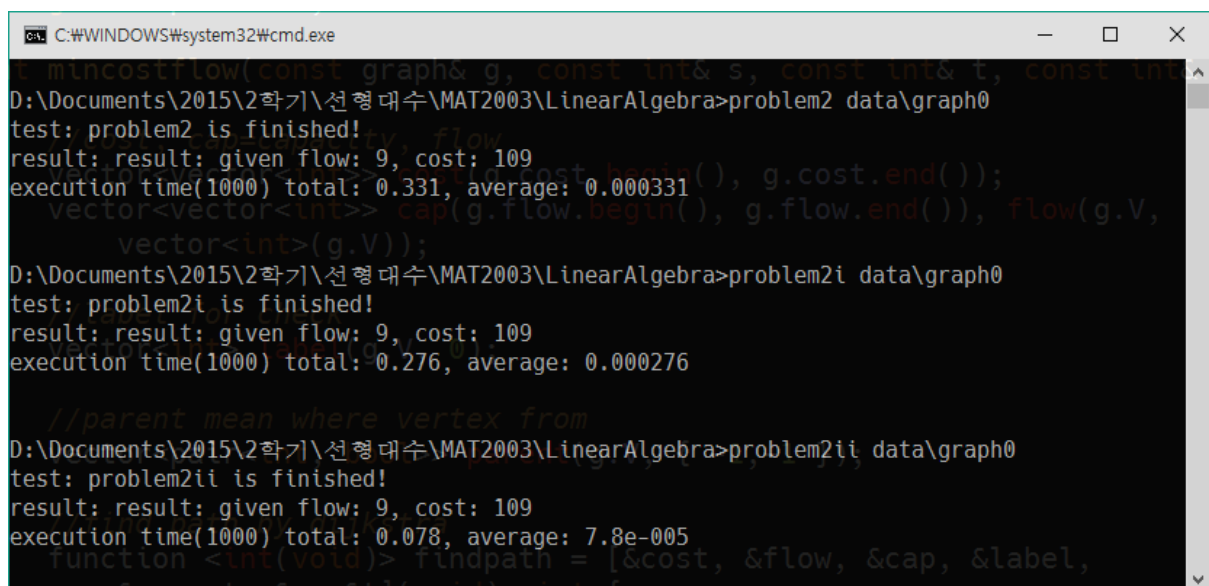
Cost가 최소인 간선을 찾는 데에는 Bellman-Ford Algorithm보다 Dijkstra Algorithm이 더 빠르다. 허나 Dijkstra Algorithm은 비용이 음수일 제대로 작동하지 않아서 Bellman-Ford를 선택했는데 조금 생각해보니 시작 비용이 음수가 없다면 진행 중에 절대로 음수인 간선이 생길 수 없다는 사실을 알게 되었다. 증명은 아래와 같이 할 수 있을 것이다.

$G(V, E)$ 와 $\forall u, v \in V$ 에서 최단경로 $p(s, t)$ 에 포함된 비용 간선 $a(u, v) \in E$ 는 포함되지 않은 간선 $b(u, v) \in E$ 에 대해 $b \geq a$ 를 만족해야 한다. 이 때 이 최단경로에 대해 flow를 처리해주면서 비용에 -1을 곱해주고 방향을 반대로 하면 $-a(v, u)$ 처럼 바뀌게 된다. 이때 음수인 사이클이 생기려면 $-a + b < 0$ 이 되어야 하는데 이는 $b < a$ 이므로 모순된다. 따라서 음수인 사이클이 존재하지 않는다.

이를 토대로 Bellman-Ford Algorithm대신 Dijkstra Algorithm을 사용할 수 있으며 이는 최단경로를 좀 더 빠르게 찾음으로써 경로를 찾는 속도에서 향상을 보일 수 있다.

Dijkstra Algorithm도 priority queue(우선순위 큐)를 간선에 따라 정리하여 최적의 경로만 선택하도록 하여 속도를 더 올릴 수 있다. 이렇게 개선하여 작성된 코드는 problem2i.cc에 작성되어있고 1000번을 실행하는데 0.276초가 걸렸다.

중복해서 방문하지 않도록 중복을 제거해주는 방식으로 Dijkstra Algorithm을 최적화 시켜보기도 하였는데 이렇게 작성된 코드는 problem2ii.cc에 작성되어 있고 1000번 실행하는데 0.078초가 걸려 가장 빠른 결과를 보여주었다.

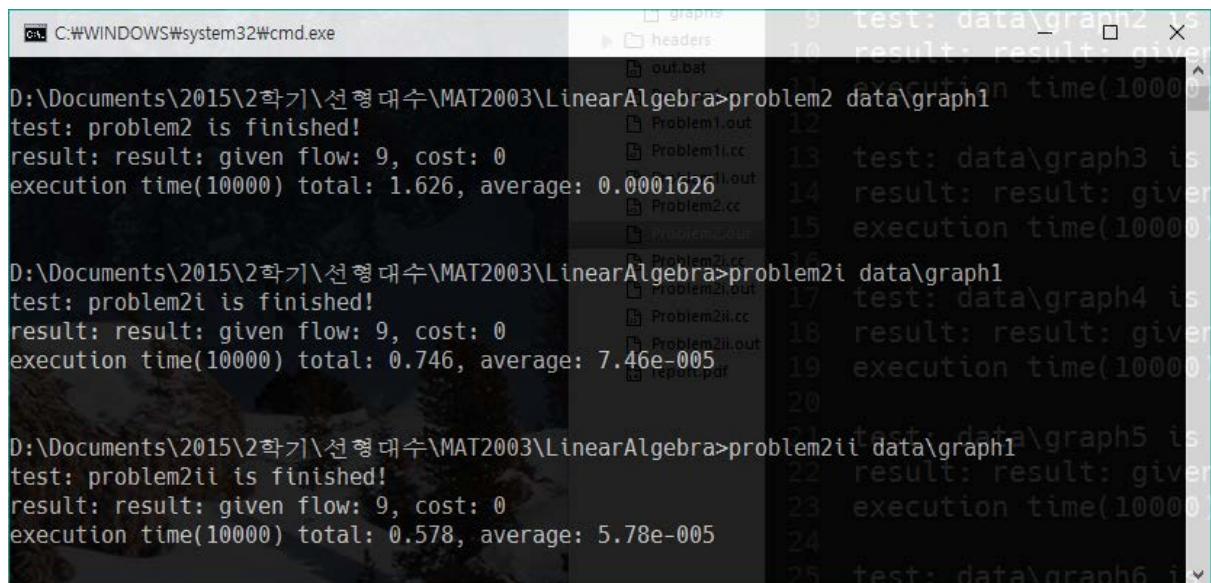


```
C:\WINDOWS\system32\cmd.exe
D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2 data\graph0
test: problem2 is finished!
result: result: given flow: 9, cost: 109
execution time(1000) total: 0.331, average: 0.000331
D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2i data\graph0
test: problem2i is finished!
result: result: given flow: 9, cost: 109
execution time(1000) total: 0.276, average: 0.000276
D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2ii data\graph0
test: problem2ii is finished!
result: result: given flow: 9, cost: 109
execution time(1000) total: 0.078, average: 7.8e-005
```

C. Analysis and Discussion

1번 문제와 마찬가지로 알고리즘에 크게 영향을 미칠 수 있는 요소인 V 와 E 에 대해 $V \approx E$ 인 경우와 $V \ll E$ 인 경우로 나누어서 생각해 보기로 하였다.

$V = 12, E = 12$ 로 주어진 Graph1 에 대한 결과로 각각 1.63, 0.75, 0.58초가 걸린 것을 확인 할 수 있으며

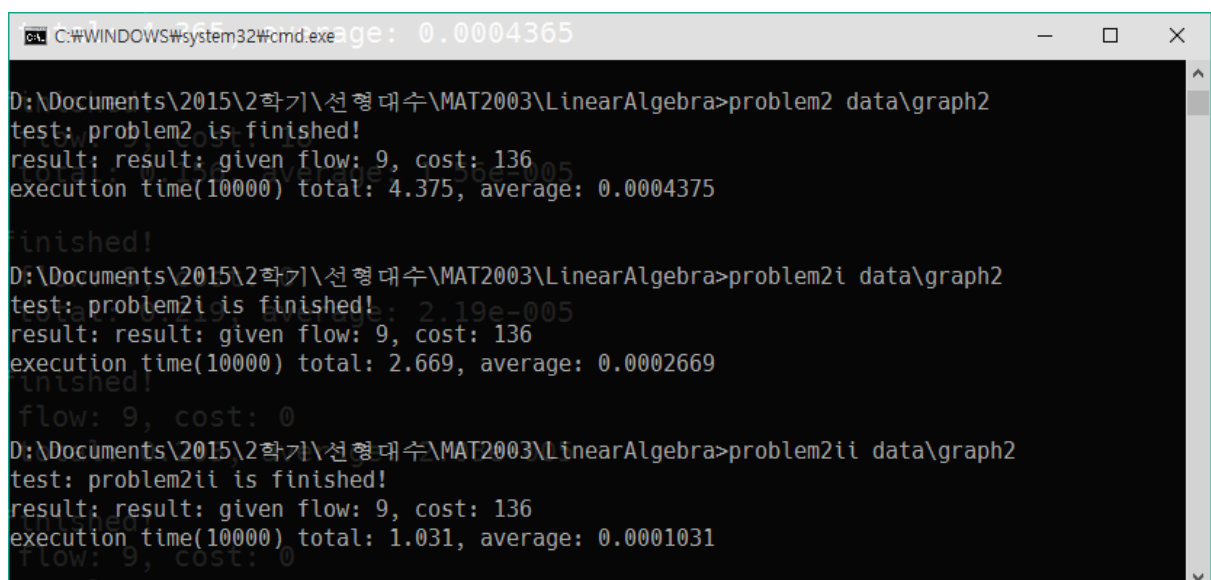


```
C:\WINDOWS\system32\cmd.exe
D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2 data\graph1
test: problem2 is finished!
result: result: given flow: 9, cost: 0
execution time(10000) total: 1.626, average: 0.0001626

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2i data\graph1
test: problem2i is finished!
result: result: given flow: 9, cost: 0
execution time(10000) total: 0.746, average: 7.46e-005

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2ii data\graph1
test: problem2ii is finished!
result: result: given flow: 9, cost: 0
execution time(10000) total: 0.578, average: 5.78e-005
```

$V = 12, E = 36$ 으로 $V \ll E$ 인 Graph2에 대해서는 4.38, 2.67, 1.03초로 월등한 향상이 있었다는 것을 알 수 있다.



```
C:\WINDOWS\system32\cmd.exe
D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2 data\graph2
test: problem2 is finished!
result: result: given flow: 9, cost: 136
execution time(10000) total: 4.375, average: 0.0004375

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2i data\graph2
test: problem2i is finished!
result: result: given flow: 9, cost: 136
execution time(10000) total: 2.669, average: 0.0002669

D:\Documents\2015\2학기\선형대수\MAT2003\LinearAlgebra>problem2ii data\graph2
test: problem2ii is finished!
result: result: given flow: 9, cost: 136
execution time(10000) total: 1.031, average: 0.0001031
```

다만 문제1과 같이 graph8, 9에 대해서는 problem2보다 problem2i의 풀이가 좋긴 했지만 problem2i와 problem2ii의 차이가 없거나 problem2ii가 더 늦었다.

6. Concluding Remarks

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!)(문제는 첨부된 mcmf.png)

먼저 관련된 지식을 알고 있어서 문제를 접근하는데 매우 수월했지만 오히려 문제를 알고 있었기 때문에 색다른 풀이나 개선점을 찾기 어려웠고 다른 학생들에 비해서 편법을 사용하는 것 같은 느낌을 받았다.

알고 있던 알고리즘에 비해 향상시켰던 방법들이 음수인 가중치와 역 간선을 가지고 있을 경우에는 미소한 차이지만 더 느리게 작동함을 확인할 수 있었는데 이 부분에 대한 개선이 필요하며 일반적인 경우에서와 특수한 경우(예를 들어 graph8, 9)가 무슨 차이가 있는지 일반화 시킬 필요가 있으며 실행시간을 Big O 방식으로 표기할 수 있도록 노력해야겠다.

7. Reference

James, B. Orlin. [*Max Flows in \$O\(nm\)\$ Time, or Better.*](#)

Jesper Larsen and Jens Clausen. [*The Max Flow Problem – Push-Relabel algorithms.*](#)