# Lecture 07 : Network Flow와 그 응용 *

네트워크 플로우 모형(model)은 선형계획(Linear programming)과 더불어 현실적인 여러 문제, 특히 optimization문제를 푸는 매우 유용한 접근법이다. 특히 산업공학의 최적화 과정에서 가장 중요하게 다루는 주제이다. 게다가 고급 수준의 프로그래밍 대회에서 항상 빠지지 않고 나오는 문제다. 예를 들어 ICPC같은 대회인데, 대륙별 본선대회에서는 거의 빠지지 않고 network flow로 해결하는 문제가 제시된다.

제일 중요한 것은 전혀 네트워크 문제로 보이지는 않는듯한 문제를 network flow 문제로 모델링하는 것인데 이 과정은 상당히 까다롭고 많은 경험이 필요하다. 가장 좋은 방법은 매우 다양한 예제 문제의 모형화를 통하여 해보는 것이다. 특히 계산문제는 보통 그래프 응영분야에서 많이 다루는 주제인데, 가장 중요한 것은 maxflow 알고리즘을 이해하는 것이며 그것을 실제 문제에 적용하여 다양한 문제를 flow 문제로 변형하여 풀어내는 능력이다. 특히 network flow는 graph partition, combinatorial optimization 등에 활용되는 예에서 볼 수 있듯이 매우 다양한 분야에서 유용하게 사용되고 있다. 전혀 다르게 보이는 문제라도 이것을 network flow 문제로 변환하게 되면 쉽게 이미 구성된 알고리즘을 풀 수 있다. 예를 들어 bipartite graph의 matching문제도 maxflow문제로 쉽게 해결할 수 있다. 여러분은 이 maxflow mincut 모형을 잘 이해하고 이것을 자신의 연구영역에 어떻게 적용하여 응용할 수 있는지를 항상 생각하고 있어야 한다. 그리고 실제 현장에서는 본인이 직접 코드를 만들지말고 LEDA에 있는 function을 잘 활용해서 실제 적용할 수 있는 능력을 키우는 것이 가장 중요하다.

## 7.1 네트워크 흐름의 기초 정의(Preliminary)

**정의 7.1.1** *network is a graph or multi-graph in which any or all of the following additional structure can exist.*

1. 모든 *edge*는 방향을 가진다. 하나 이상의 방향 *(two way) edge*도 가능하다.
2. 특별한 두 *vertex*인 시작점 *source*와 종착점 *sink node*가 존재한다.
3. 모든 *edge*에 어떤 의미를 나타내는 값이 지정되어 있다.
4. 모든 *vertex*에 어떤 의미를 나타내는 값이 지정될 수 있다.

*2013년부터 network flow 장으로 독립. 2019년에 survey design과 lower bound, minimum cost model 추가함.

이 문제에서 우리는 각 edge의 최대용량(capacity)를 $c(u,v)$라고 한다. 그리고 어떤 시점에 그 edge $(u,v)$를 흐르는 유량을 $f(u,v)$라고 한다. 이 때 각 edge마다 특정한 성질을 가지고 있어야 한다. 먼저 용량제한 속성으로다음과 같다.

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

그 다음 성질로서는 유량의 대칭성이다. 이 특성은 문제를 일관되게 만들어주며 아주 매끈한 알고리즘을 가능하게 한다.

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

끝으로 필요한 성질은 유량의 보존성으로 들어온 양과 나간 양은 항상 일치하여야 하는 것이다. $u \in V - \{s,t\}$에 대하여 아래 식이 성립한다.

$$\sum_{v \in V} f(u,v) = 0$$

다중 출발지와 다중 종점을 가지는 네트워크를 다루어야 하는 경우도 있다. 이 경우는 추가의 super source와 super sink를 추가하여 표준형 네트워크로 만들 수 있다.

## 7.2 Network flow의 행렬(Matrix) 기반 해석

**정의 7.2.1** *Path flow*는 교차점이 없는 *simple path*를 따라 흘러가는 *flow*를 나타낸다. 그리고 모든 *edge*의 *flow weight*는 동일하며 그 중에서도 모든 *edge weight*가 *1*인 것을 *unit flow path* 라고 한다.

**문제 7.2.1** 다음 제시된 두 *network* $N_a$와 $N_b$를 *path flow*의 *sum*으로 나타내 보시오. 단 시작점과 끝은 반드시 *source* $U$와 *sink* $V$가 되어야 한다.

**문제 7.2.2** 위의 네트워크 $N_b$에서 *cycle flow*의 개념을 도출하여 설명해보시오.

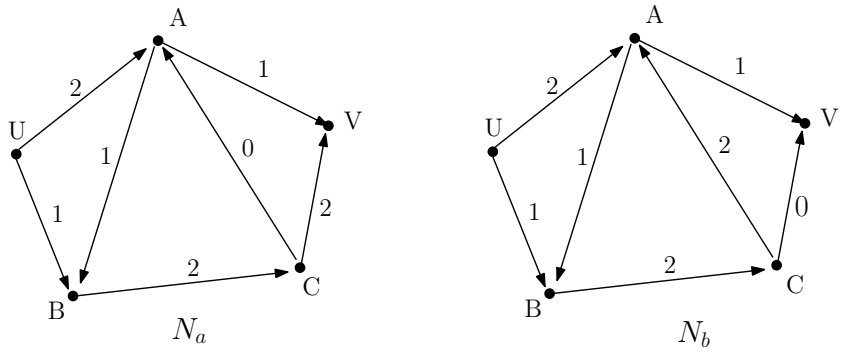


Figure 1:

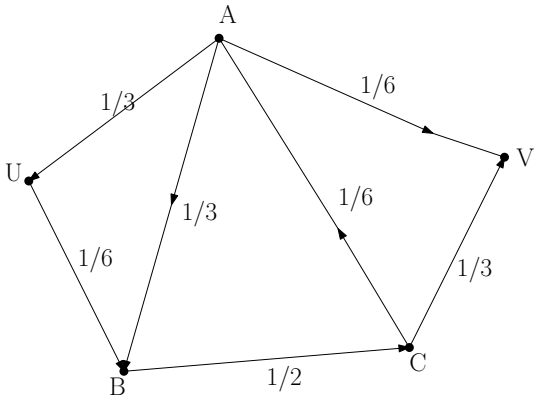


Figure 2: edge weight가 분수로 나타난 경우 $P_3$.

**문제 7.2.3** 다음은 각 *edge*의 *weight*가 분수로 표시된 어떤 *network*이다. 여기에서 전체를 *path flow*의 *sum*으로 나타내 보시오. 단 시작점과 끝은 반드시 *source*와 *sink*가 되어야 한다.

**정의 7.2.2** *network flow* $\phi$는 주어진 *network*에서 *source*에서 *sink*로 이르는 *flow*의 *vlaue*을 말한다.

앞의 $P_1$은 다음과 같은 matrix로도 표시할 수 있다.

| | U | A | B | C | V |
|---|---|---|---|---|---|
| U | 0 | 2 | 1 | 0 | 0 |
| A | -2 | 0 | 1 | 0 | 1 |
| B | -1 | -1 | 0 | 2 | 0 |
| C | 0 | 0 | -2 | 0 | 2 |
| V | 0 | -1 | 0 | -2 | 0 |

**문제 7.2.4** 주어진 행렬에서 각 *row*과 *column sum*은 무엇을 말하는지 설명하시오. 그리고 주어진 *network*에서 *flow*는 어디에 표시가 되는지 설명하시오.

**정의 7.2.3** *two flow* $\phi_1, \phi_2$는 *equivalent* 하다. *if and only if* 대응하는 *flow*의 *matrix*가 동일하다. 즉 $[\phi_1] = [\phi_2]$. 이 *notation*은 다음과 같이 표시된다.

$$\phi_1 \sim \phi_2 \textit{ if and only if } [\phi_1] \sim [\phi_2]$$

**정의 7.2.4** 어떤 두 노드 $A$와 $B$에 대하여 *flow*가 *reduced*되었다는 것은 $\phi(AB) \cdot \phi(BA) = 0$ , 즉 둘 중 하나는 *0*이 경우를 말한다.

**정리 7.2.1** *(1)* 모든 *flow*는 유일한 *flow*로 *reduced*가 된다.
*(2) distinct reduced flow*는 *distinct matrix*와 대응된다.

**정의 7.2.5** 어떤 *network*의 한 *edge* $(A, B)$의 용량 *(capacity)*는 $c(A, B)$로 표시된다. 어떤 *flow*가 *"*가능 *(feasible)"*하다는 것은 모든 *arc*에 대하여 $\phi(A, B) \leq c(A, B)$ 인 경우를 말한다.

이제 남은 문제는 주어진 network에 대하여 각 arc마다 capacities가 주어져 있을 때 이를 하나의 feasible flow를 구한 다음에 이를 조금씩 개선(증가) 시켜서 이것이 궁극에는 maximal flow가 되도록 만드는 것이다. 이 방법은 Ford-Fulkerson algorithm을 이용해서 해결할 수 있다.

## 7.3 최소컷-최대흐름 정리 (Min-Cut Max-Flow Theorem)

어떤 네트워크에서 cut이란 S와 T를 완전히 양분하는 edge의 집합을 말한다. 즉 그것을 모두 제거 했을 때 S에서 T로 가는 path가 없는 경우이다. 이것이 cut set의 formal한 정의다. 단순한 cut은 S에서 나가는 모든 edge set이나 T로 들어가는 모든 set이 될 수 있다. 이때 그 cut에 표시된 용량의 합을 cut의 용량이라고 한다. 이제 min cut은 C 중에서 어떤 하나라도 빠지면 cut이 될 수 없는 집합을 말한다고 하자. 즉 Cut중에서 갯수가 가장 작은 cut을 말한다. 그림-3에서 보라색 cut은 네트워크을 S와 T와 양분하고 있고 그 중 어떤 것이라도 빠지면 cut이 될 수 없기 때문에 한 min cut이라고 말할 수 있다.

따라서 cut은 우리가 어떤 edge를 선택하는 가에 따라서 값이 달라질 수 있지만 min cut을 유일하다. 그리고 이것이 max flow와 같다는 것이 이 장의 핵심이다. 극단적으로 볼 때 모든 edge를 cut으로 볼 수 있다. 왜냐하면 이들을 제거하면 S와 T가 분리되기 때문이다. 네트워크에서 어떤 min cut set C의 용량 capacity( C )은 해당 edge의 capacity의 합으로 정의된다. 즉 아래와 같다.

$$capacity(C) = \sum_{e \in C} capacity(e)$$

이제 주어진 flow $\Phi$가 제시된 네트워크에서 특정 cut에 흐르는 유량(flow(C)를 확인해보자. 이것은 다음과 같이 정의된다.

$$flow(C) = \sum_{e \in C} flow(e)$$

그런데 어떤 유의미한 flow가 주어지면 이 flow에 적용된 모든 cut C의 $flow(C)$는 동일하다. 즉 모든 flow $\phi$가 있을 때 이것에 적용된 $flow([s, t-cut])$는 동일하다. 왜냐하면 S에서 T로 가는 전체 양이 동일하기 때문이다. 그러나 cut의 capacity(C)는 각각 다를 수 있다. 즉 cut에 따라서 각 edge에 처음 주어진 capacity의 합은 다를 수 있다. 즉 이것이 flow cut 중에서 maximum이기 때문이다.

아래 그림-3이라면 $20+7+8$이 제시된 어떤 cut의 용량이다. 그리고 이 때 어떤 유량 flow는 실제 그 양분된 그래프의 경계를 넘어가는 유량이 된다.[1] 즉 $13+0-3+7=17$이 된다. 따라서 모든 cut의 flow(C)는 어떤 cut의 capacity보다 작은 것은 당연하다. 따라서 아래의 식이 성립하고 이것이 의미하는 바가 바로 minimum cut은 maximum flow와 동일하다는 것이다.

$$flow(s,t) \leq capacity(s, t-cut)$$

위 정리를 Mincut maxflow Theorem이라고 한다. 이것을 직관적으로 설명해보자. 일단 어떤 cut $C_a$의 $flow(C_a)$는 그 cut의 $capacity(C_a)$보다는 항상 같거나 작다. 그럴 수 밖에 없다.

$$flow(C_a) \leq capacity(C_a)$$

[1] 넘어갔다가 다시 오는 것은 상쇄된다.

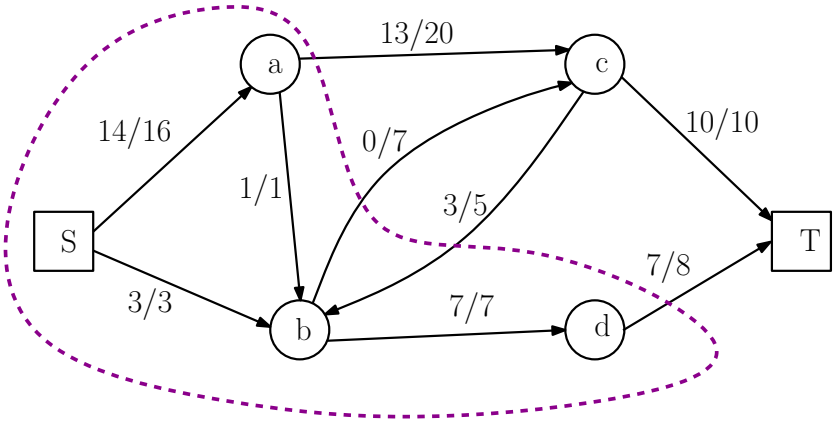


Figure 3: 네트워크의 모든 cut C에 대하여 $flow(C)$은 동일하고, 그 값의 최대치 maximum flow는 어떤 minimum cut $C^*$의 $capacity(C^*)$과 동일하다.

그런데 모든 $flow(C_a)$은 어떤 cut을 선택하더라도 동일하기 때문에 임의의 cut $C^*$의 flow는 다음의 정리가 성립한다.

$$flow(C^*) \leq capacity(C_a)$$

따라서 우리가 $capacity(C)$중에 minimum인 $C_m$을 선택해도 이 사실이 성립하므로 우리는 다음의 사실을 알 수 있다.

$$flow(C^*) \leq capacity(C_m)$$

그리고 이것이 모든 flow $\Phi$에 대하여 성립하므로 우리가 원하는 아래 식을 얻을 수 있다.

$$max\{flow(N)\} \leq min\{capacity(C)\}$$

따라서 우리는 maxflow를 구해도 되고 mincut을 구해도 된다. 그런데 아무래도 mincut을 구하면 cut edge를 알 수 있고 동시에 maxflow도 알 수 있기 때문에 이것을 구하는 것이 더 유리하다. 아래 설명할 Ford-Fulkerson 알고리즘은 오직 maxflow만 알려주기 때문에 이것을 이용해서 mincut을 구하는 것을 고안해야 한다. 그러면 mincut은 어떤 특성이 있을 지 생각해보자.

만일 어떤 mincut이 존재하고 그것의 capacity가 $c_a$인데 residual flow가 가능하다고 해보자. 예를 들어 capacity가 10인데 배정된 flow가 8인 경우라고 생각해보자. 그러면 이 경우 +2의 flow에 대한 여유가 생긴다. 그런데 이 경우 우리는 일단 이 edge에 +1 이상의 여유가 있어 flow를 추가할 가능성이 있다. 그런데 이 일이 불가능할 경우에는 이 edge로 들어오는 모든 edge가 "포화(saturated)"가 되어 추가가 불가능한 경우이다. 만일 추가가 가능하다면 maxflow가 증가하게 되고 이 때문에 mincut 역시 증가하므로 원래 가정에 모순이 된다. 따라서 이 edge 대신 앞서의 edge를 모두 막으면 같은 cut

으로 mincut을 달성할 수 있고, 그런 edge들은 모두 saturated된 edge가 된다. 따라서 mincut edge set을 찾는 방법은 S에서 나가는 모든 flow 중에서 saturated된 edge set를 모으면 된다. 즉 S에서 flow를 따라 DFS를 수행하고 그 경우 어떤 edge가 saturation이 되면 멈추고 해당 edge를 제거한다. 이 작업을 S와 T가 분리될 때까지 계속하고 그 때 제거된 모든 edge set을 모으면 이것이 바로 mincut set이 된다.

**문제 7.3.1** 다음 *network* 그래프에서 $(s,t)$-*maximum flow*를 구하시오.

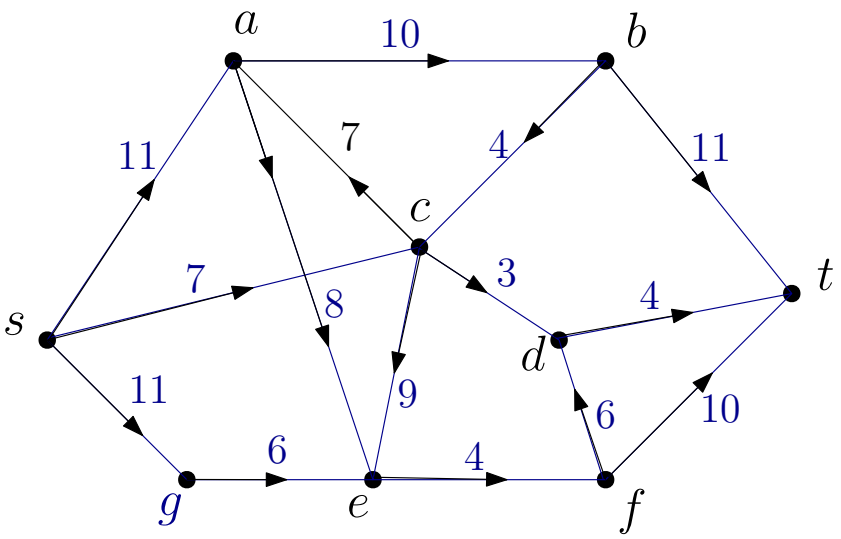


Figure 4: 네트워크의 예.s에서 t로 가는 최대 flow를 구해보시오.

**정리 7.3.1** *[Min-cut max-flow Theorem] Ford-Fulkson(1956)*
모든 네트워크에서 최대유통량 *(maximum value of feasible flow)*는 최소 *source/sink capacity*이다.

## 7.4 포드-풀커슨(Ford-Fulkerson) 알고리즘

포드-풀커슨 알고리즘은 단순한다. 일단 모든 에지의 유량을 0으로 두고 조금씩 흘려보낼 수 있을 때 까지 계속 흘려보내는 것이다. 그러다 더 이상 보낼 수 없다고 판단이 되면 그 때 중지하는 일종의 그리디(Greedy) 방법이다.

**문제 7.4.1** 다음 *network*에서 *unit flow path*를 하나씩 들어내는 방법으로 *maximal flow*를 구해보자. 즉 시작 점에서 남은 유량으로 $T$까지 갈 수 있는 길을 찾는다. 만일 그러한 길이 없으면 작업을 중단하고 *max flow*를 보고하는 방식의 알고리즘을 수행해 보자.

이 방식을 일반적인 단순 네트워크에서 적용한 단계가 아래와 같이 표시되어 있다.

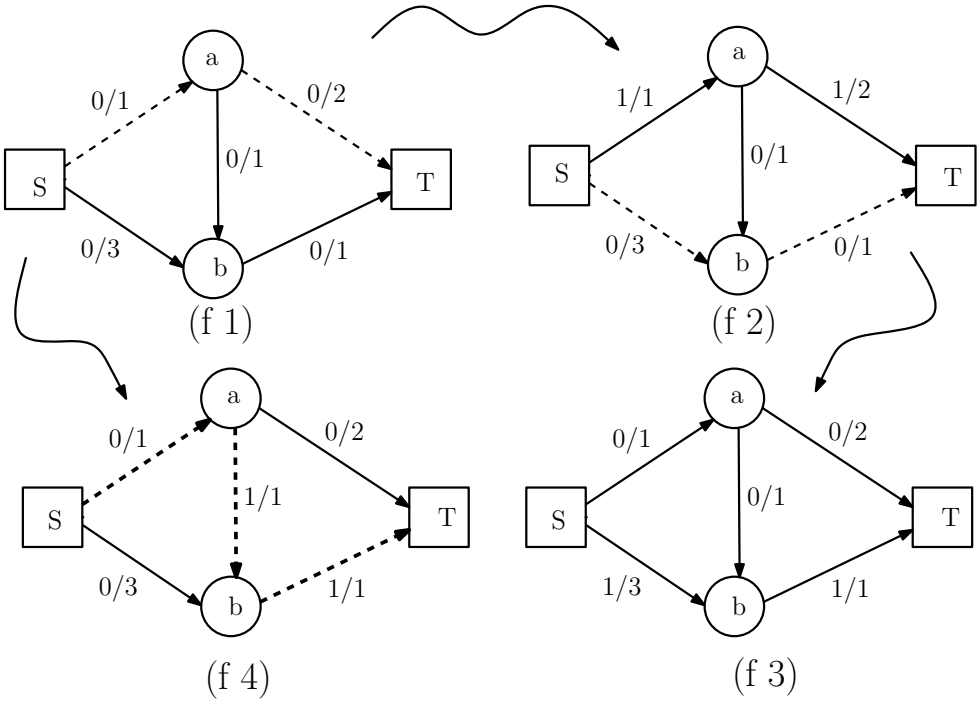


Figure 5: 직관적인 방법으로 풀 때 틀리는 경우.

즉 어떤 경로를 선택하는가에 따라서 우리는 새로운 flow를 찾을 수도 있고 그렇지 못한 경우도 있다. 먼저 흘려보낸 flow의 방향을 뒤집어 보자. 그러면 이제 새로운 flow를 추가할 수 있다.

b에서 a로 가는 edge가 없기 때문에 $c(b,a) = 0$이다. 그러너 유량의 대칭성을 이용하면 그 값은 $f(b,a) = -1$임을 알 수 있다. 이제 잔여 용량 (residue)를 계산하면 다음과 같다.

$$r(b,a) = c(b,a) - f(b,a) = 0 - (-1) = 1$$

즉 b에서 a로 $+1$을 흘려보낼 수 있다는 것을 의미한다. 따라서 edge $(a,b)$에 대한 용량은 주거니 받거니 하면서 도두가 상쇄된다. 동시에 용량의 보존성도 유지되므로 새 유량을 보내는 것은 기존의 유량을 깍아서 상쇄하는 것과 동일한 작업이 된다. 즉

Ford-Fulkerson 알고리즘은 모든 간선에 대하여 잔여용량이 남아있을 까지 계속해서 (대칭성까지 고려하여) 흘려보내는 방식으로 진행된다.

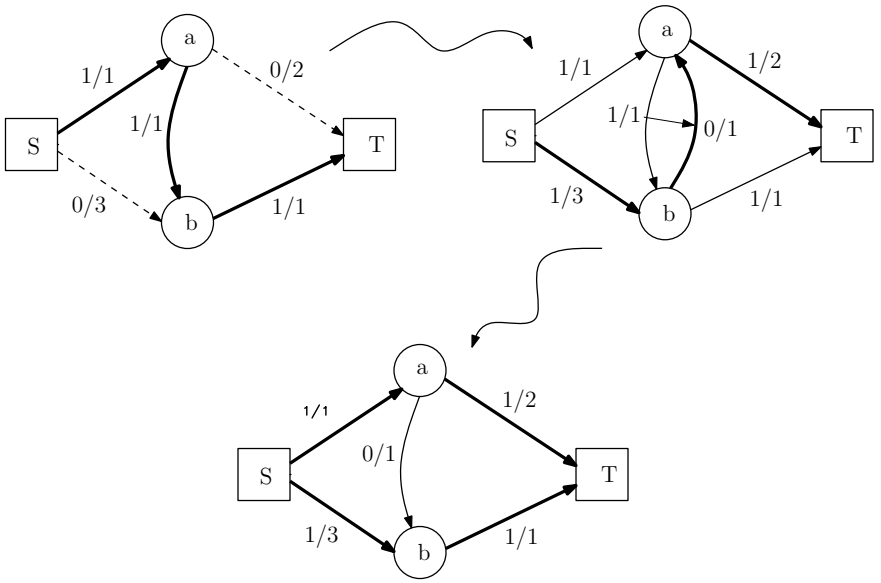


Figure 6: 유량 상쇄 작업을 통한 새로은 adding flow 찾아내기. 처음에는 S-a-b-T를 찾음. 그 다음에 S-b-a-T로 가는 path를 찾음. 결국 (a,b) 의 flow는 +1, -1이 되어 0/1 로 고정됨.

문제는 Ford-Fulkerson 알고리즘을 사용할 때 시간에 매우 오래 걸리는 경우가 있다는 점이다. 아래가 그 경우인데 작은 edge가 계속해서 선택되어 최악의 경우 $\epsilon$ 만큼씩 총 유량의 증가하여 경우에 따라서는 유량이 점진적으로 추가되는 전체 단계를 $O(N/\epsilon)$ 번 정도로 매우 많이 수행해야하는 경우가 생긴다.

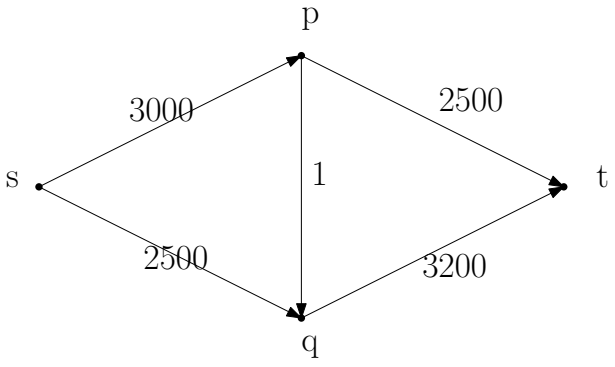


Figure 7: Ford-Flukerson 알고리즘을 활용할 때 최악의 경우

문제는 이 방법으로 최대 유량을 찾아낼 수 있는가 하는 것을 증명하는 것이다. 이것은 Min-Cut Max-FLow라는 위대한 정리를 이용해서 증명가능하다. 이 알고리즘의

복잡도를 생각해보자. 이 알고리즘은 매단계에서 DFS를 수행하기 때문에 $O(|V|+|E|)$ 이간이 든다. 따라서 최대유량이 $f$ 라면 한번에 적어도 $+1$은 증가하기 때문에 $O(f|E|)$ 의 시간이 듭니다. 그런데 BFS를 사용하면 가장 짧은 경로만을 찾아내기 때문에 위에서 제시한 최악의 경우는 생기지 않는다. 즉 $O(|V||E|)$ 개의 증가 경로만을 사용해서 max flow를 찾을 수 있음이 알려져 있다. 따라서 FF algorithm의 복잡도는 다음과 같다.

$$\min\{O(|E|f), O(|V||E|^2)\}$$

**문제 7.4.2** 다음 *network*에서 *unit flow path*를 하나씩 들어내는 방법으로 *maximal flow*를 구해보자.

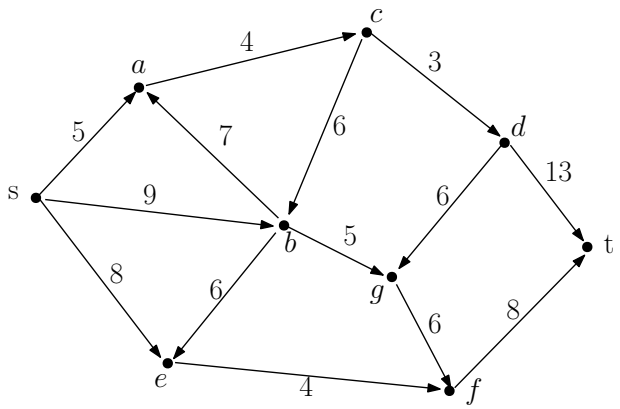


Figure 8: Ford-Flukerson 알고리즘으로 위 network에서 maximum flow를 구해보자.

## 7.5 Circulation with Demand/Supply Problem

어떤 directed network이 있다. 이 네트워크에서 뭔가를 공급하는 Supply node가 있고 받기만 하는 Demand node가 있다. 즉 각 노드마다 요구 정도 $d(x)$가 정의되는데 $d(v) > 0$이면 demand node, $d(v) > 0$이면 supply node, $d(v) = 0$ 이면 transhipment node이다. 그리고 각 directed edge는 일반적인 (S,t) network과 같이 capacity가 정해져 있다. 물론 capacity의 bound가 $[l, u]$로 각각 주어질 수 있다.

이 상황에서 circulation은 어떤 flow를 지정하는 함수(각 에지의 flow를 지정하는)로서 다음을 만족해야 한다.

1. 모든 edge에 대하여 지정된 flow는 capacity 조건을 만족해야 한다. 즉 $0 \leq flow(e) \leq capacity(e)$
2. 모든 노드 $v$에 대하여 +flow와 -flow의 차이는 $d(v)$가 되어야 한다.
3. demand의 합과 supply의 합은 동일해야 한다. $\sum_{x \in Demand} d(x) = \sum_{x \in Supply} d(x)$.

Circulation problem 문제란 이런 network이 주어질 때 이것을 만족하는 함수가 존재하는지를 yes, no로 판단하는 decision problem 이다.

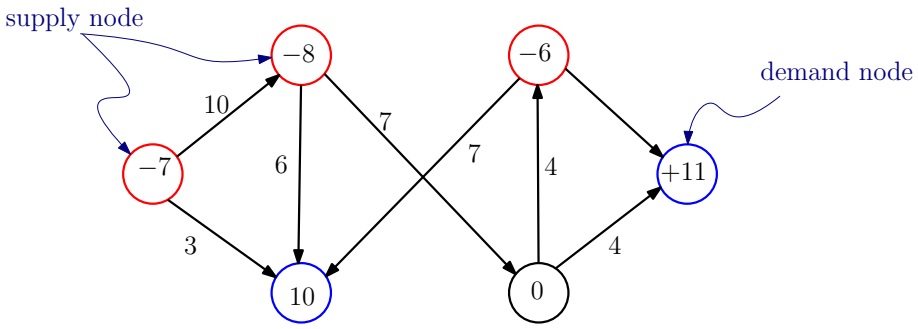


Figure 9: Supply-Demand Network

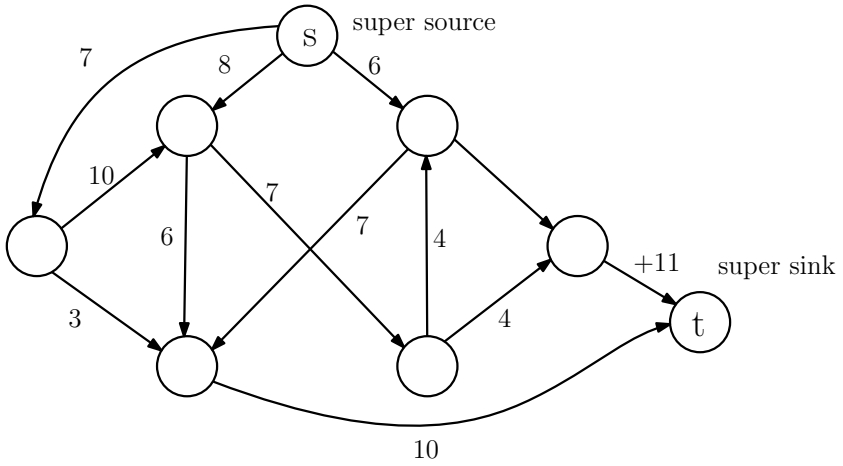


Figure 10: 새로운 source S와 t를 추가하여 circulation 문제는 network flow로 바꿈.

## 7.6 Circulation with Lower Bound

우리는 앞 서 maxflow의 기본유형에 대하여 설명하였고, circulation network을 maxflow로 해결하는 방법을 제시하였다. 이제 각 edge capacity에 대하여 lower bound가 존재하는 경우에 대하여 살펴본다. 이전 일반모형에서는 $0 \leq f(e) \leq capacity(e)$ 인데 비하여 그 하한이 0이 아니라 $l(e) \leq f(e) \leq capacity(e)$가 되는 것이다.

Circulation with demands with Lower bounds문제는 앞서의 제한조건과 동일하며 flow(e)의 제한만 $l(e) \leq f(e) \leq capacity(e)$ 만족하는 것이 추가된다.

아이디어는 항상 $l(e)$ units을 edge $e$로 항상 보내도록 **강제 (forcing)**하는 것이다.

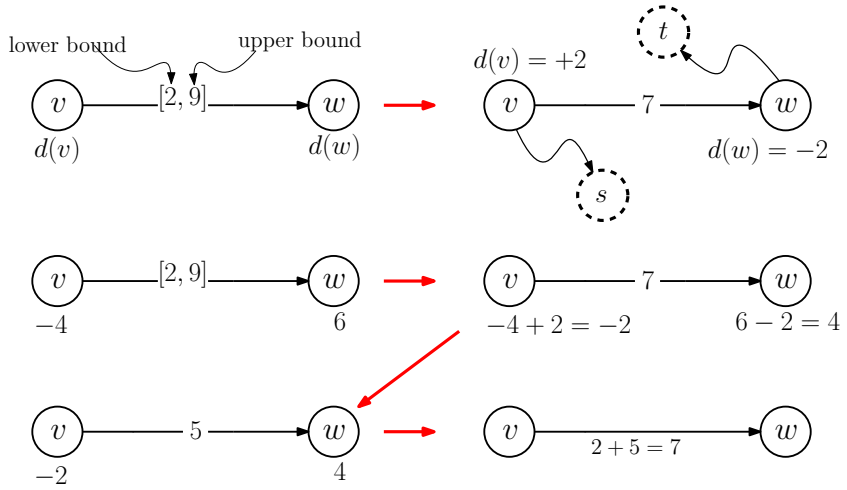


Figure 11: capacity의 하한치가 있는 경우 node를 demand, supply node로 바꿔서 변환함.

이와 같이 $G$를 변환시킨 network $G'$ 에서는 새로운 supply demand node가 만들어져서 이 문제는 변환된 network에서 circulation이 존재하는지를 검사하는 문제로 변환된다. 아래 그림은 왼쪽 lower bound가 있는 네트워크를 일반적인 $[0, w]$ capacity로 바꾸는 과정을 보여주고 있다.

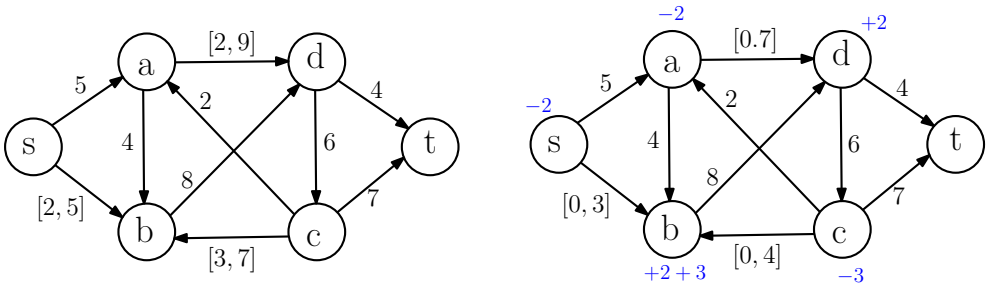


Figure 12: lower bound가 있는 네트워크 모형을 일반적인 macflow로 바꾸는 과정

따라서 $G'$에서 circulation이 존재하면 lower bound가 설정된 network에서도 circulation이 존재함이 if and only if로 증명된다.

## 7.7 Network Flow의 다양한 응용

네트워크 흐름은 앞서 설명과 같이 매우 다양하게 활용된다. 즉 주어진 문제를 Network flow모형으로 변환하면 쉽게 문제가 해결된다. 이 변화기술이 네트워크 흐름 응용의 가장 중요한 내용이다. 이 변환에는 어떤 규칙이 존재하는 것은 아니며 다양한 문제를 직접 다뤄보는 방식으로 익혀야 한다.

**문제 7.7.1** 하나 이상의 *source*와 *sink*를 가진 네트워크에서 *max flow(min cut)*을 구하려고 한다. 주어진 네트워크를 어떻게 변환하면 될지 설명해 보시오. □

### 7.7.1 마일리지와 상품권

우리는 몇 종류의 마일리지를 가지고 있다. 그리고 각 상품($b_i$), 예를 들면 항공권이나 여행숙박, 식당에 필요한 비용을 어떤 마일리지로 결제할 수 있으며 그 가능한 최대값이 얼마인지 정해져 있다. 예를 들어 마일리지의 종류에는 $\{m_1, m_2, m_3, m_4, m_5\}$ 종류가 있으며 어떤 사용자는 각 마일리지에 대한 포인트를 가지고 있다. 그리고 각 상품의 가격이 지정되어 있다. 동시에 그 상품에 사용되는 마일리지의 종류와 활용할 수 있는 최대 가격이 정해져 있다. 우리는 해당 상품을 모두 구매하려고 한다. 가장 쉬운 방법은 모든 상품을 현금으로 사는 것인데 이보다는 마일리지를 활용하면 가장 적은 돈을 사용해서 상품을 구매할 수 있다. 이때 가장 적게 돈을 쓰도록 마일리지를 상품별로 분배하는 과정을 해결하는 것이 이 문제이다.

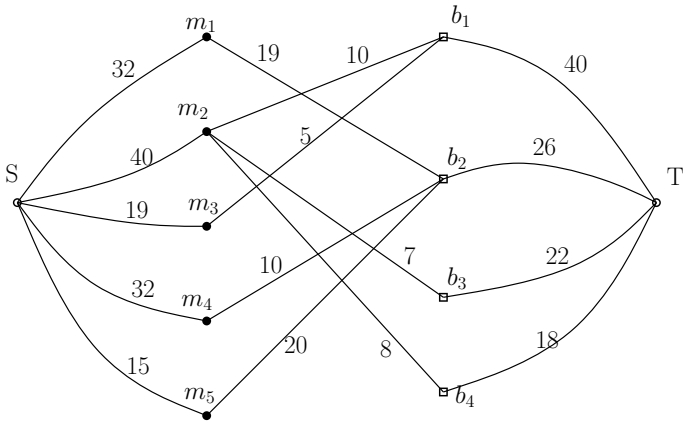


Figure 13: 다양한 마일리지가 있을 때 가장 효율적으로 상품을 구매하는 방법

### 7.7.2 최소 경로 커버 (Minimum Path Cover)

그림과 같이 directed acyclic graph가 있다. 예를 들어 어떤 프로그램의 data flow graph라고 생각하면 된다. 이 그래프의 모든 정점을 서로 겹치지 않는 vertex disjoint path로서 cover하고 싶어 한다. 즉 모든 vertex는 길이가 0 이상인 서로 다른 vertex disjoint path cover에 반드시 포함되어 있어야 한다. 즉 하나의 vertex가 하나 이상의 path cover에 중복으로 포함되는 안된다.

이 문제는 software testing과도 연관이 있다. 한 시스템이 여러 함수로 구성되어 있을 때 하나의 독립된 함수를 vertex로 매핑하고 한 함수 $f(x)$에서 다른 함수 $g(x)$를 호출하면 하나의 directed edge $\overrightarrow{(f,g)}$를 추가한다. 그런데 하나의 testing case를 실행하면 는 하나의 path를 cover하기 때문에 최소의 disjoint path cover의 수는 최소 개수의 test set과 동일하다고 할 수 있다. 따라서 하나의 DAG에서 가장 작은 수의 path cover를 찾는 일은 매우 의미있는 일이라고 할 수 있다.

우리는 주어진 방향 DAG $G$를 변형하여 모든 directed bipartite graph로 만든다. 모든 $G$의 vertex $i$에 대하여 각 두 개의 vertex $X = \{x_i\}$와 $Y = \{y_i\}$로 이루어진

bipartite graph $B(G)$인 $B(X, Y, E)$를 만든다. 만일 DAG $G$에서 $(\overrightarrow{i, j})$가 존재하면 $B$에서 directed edge $(x_i, y_j)$를 설정한다. 그리고 새로운 source s를 추가하여 X와 모두 연결하고 sink T를 추가하여 Y의 모든 vertex와 연결한다.

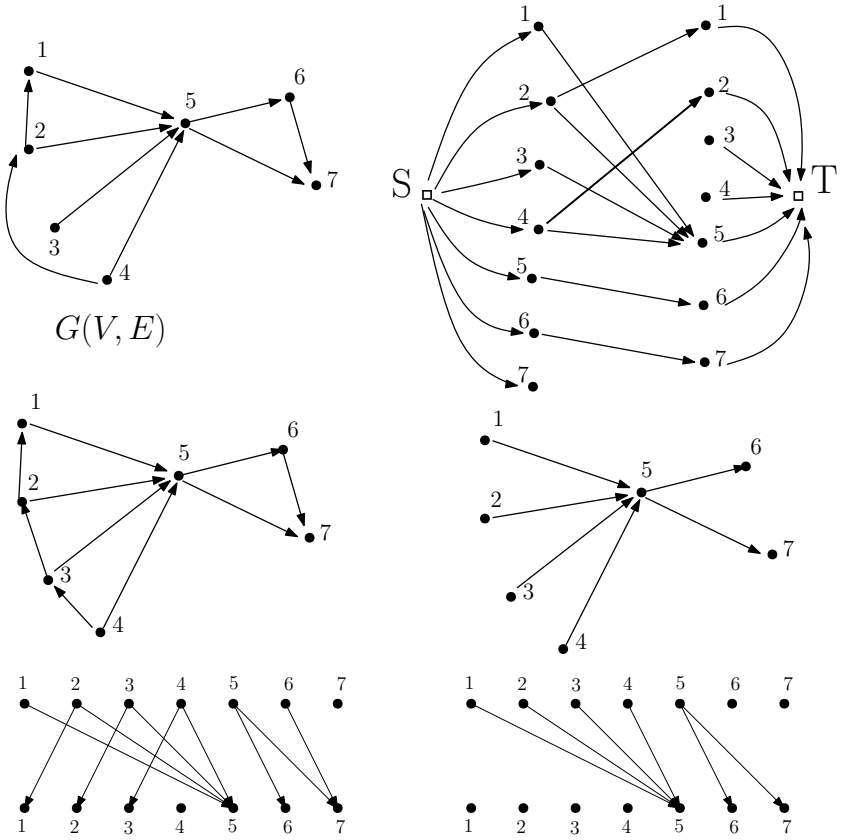


Figure 14: 주어진 acyclic graph를 최소 갯수의 disjoint directed path로 cover하는 문제

이 경우 maximum matching의 갯수와 path cover와의 관계가 어떻게 성립하는지를 볼 필요가 있다. 만일 $N$개의 vertex로 구성된 path cover가 있다고 하면 이 path상의 edge들은 maximum matching이 포함되어 있다. 즉 N-1개의 matching을 찾아낼 수 있다. 따라서 M개의 vertex가 존재할 때 maximum matching의 갯수가$K$개라고 한다면 path cover의 최소갯수는 $M - K$가 된다. 그림-10에서 (2,2) 위치의 subgraph를 B(X,Y,E) 그래프로 만들어 max matching을 구하면 2개가 나온다. 따라서 전체 정점의 수가 7개이므로 7-2=5이다.

**문제 7.7.2** 만일 *cycle*에 존재하는 그래프라면 왜 위 이 방법이 적용되지 않는지 예를 들어 설명하시오. $K_3$인 *directed cycle*를 생각해보자.

### 7.7.3 정점 독립경로(Vertex Disjoint Path) 구하기

방향이 없는 일반적인 그래프는 edge의 방향이 존재하는 네트워크와 다르기 때문에 network flow를 바로 적용할 수 없다. 그렇지만 이 일반 그래프를 교묘하게 변환하면 이 그래프에서의 network flow를 이용해서 vertex disjoint path의 갯수와 그 경로를 찾아낼 수 있다.

무방향 그래프 $G$의 $(v_a, v_b)$가 있을 경우 각 vertex중 source S와 sink T를 제외한 모든 vertex을 2개로 쪼갠다. 즉 $v_a \rightarrow \{v_{a1}, v_{a2}\}$로 만들고 이 둘 사이에 inner directed edge $(\overrightarrow{v_{a1}, v_{a2}})$를 설정한다. 그리고 source와 인접한 edge는 모두 나가는 방향을 주고, 반대로 sink로 들어가는 edge는 모두 들어오는 쪽으로 edge를 준다. 그 다음 $G$에 존재하는 edge $(x, y)$에 대해서 두 개의 directed edge를 준다. 주는 방향은 다음과 같다.

$$(\overrightarrow{x2, y1}), (\overrightarrow{y_1, x_2}), (\overrightarrow{x1, x_2}), (\overrightarrow{y1, y2})$$

그림으로 나타내면 다음과 같다. 먼저 그림-16에서 edge $(x, y)$의 경우 x에서 y로도 갈 수 있고, 반대로 y에서 x로도 올 수 있다. 어떤 경우이든 이런 path가 존재하면 다시는 y나 x를 거치는 path가 존재하지 않도록 x,y의 모든 vertex를 occupy해야 한다. 어떤 경우라도 x나 y의 subvertice를 모두 사용하기 때문에 다른 path가 다시 $x$나 $y$를 거쳐 지나갈 수가 없다. 왜냐하면 $x$나 $y$를 사용한다는 말은 그 안의 inner edge $(x1, x2)$나 $(y1, y2)$를 반드시 지나가야하기 때문에 이것을 한번 지나가면 모든 subvertex를 사용하기 때문이다.

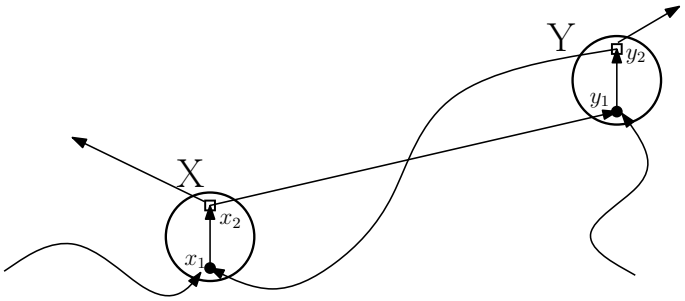


Figure 15: 단순 무방향 그래프 edge를 vertex 중복없는 path flow가 되는 directed edge로 만들기

이 그림은 어떤 무방향 그래프를 방향있는 network으로 만든 예이고, 이 network에서 maxflow를 찾으면 그것은 당연히 가장 많은 수의 directed unit flow가 된다. 그런데 이 unit flow는 모두 vertex disjoint 하므로 이것은 변환전 무방향 그래프의 vertex disjoint path가 되기 때문이다. 이 변환 기법은 NP-complete problem transform이나 graph connectivity 계산에 사용도는 아주 중요한 방법이기 때문에 잘 이해하고 있어야 한다.

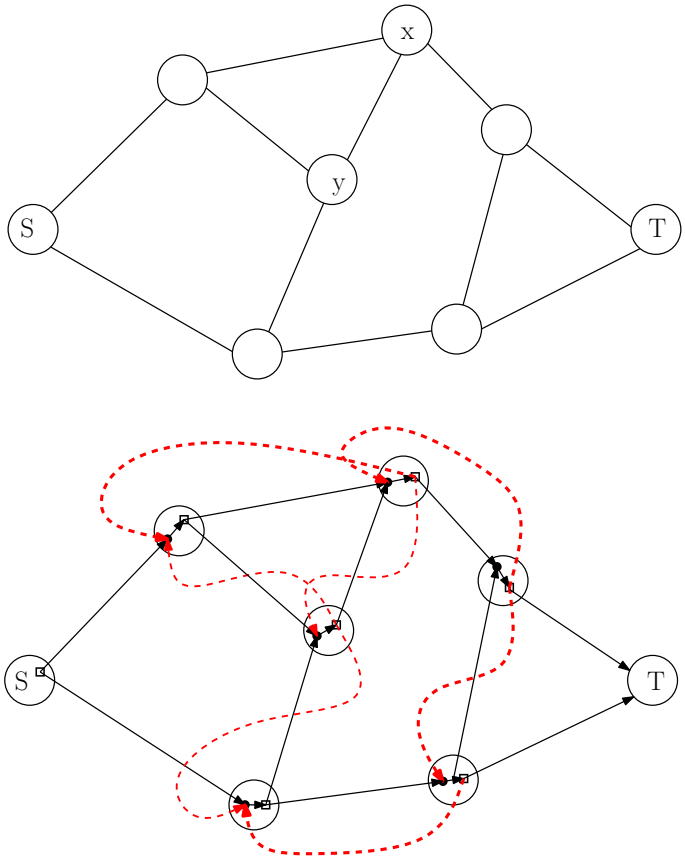


Figure 16: 단순 무방향 그래프를 directed network으로 변형 시키기

### 7.7.4 수영 단체전 선수 배정

어떤 코치는 200 미터 혼계영 수영팀을 책임지고 있다. 이 수영팀에는 5명의 선수가 있는데 코치는 이 선수를 잘 선발하여 4종류의 영법을 각 50미터마다 사용하는 200미터 계영에서 가장 좋은 기록을 만들어 내고 싶어 한다. 그런데 5명의 선수는 모두 특징이 있어서 각 영법(stroke)마다 다른 기록을 보이고 있다. 아래 표는 각 선수가 50 미터에서 사용한 영법마다의 기록(seconds)이다. 여러분은 이 중 4명을 각 종목 선수로 선발하여 그 전체 시간의 합이 최소가 되도록 하는 것이다. 단 한 종목에는 한 사람만 선정할 수 있으며 아래 기록은 50미터 1회 전력을 했을 때의 기록이다.

| Stroke | Abraham | Bernard | Colley | David | Ericson |
|---|---|---|---|---|---|
| Backstroke | 37.7 | 32.9 | 33.8 | 37.0 | 35.4 |
| Breaststroke | 43.4 | 33.1 | 42.2 | 34.7 | 41.8 |
| Butterfly | 33.3 | 28.5 | 38.9 | 30.4 | 33.6 |
| Free-style | 29.2 | 26.4 | 29.6 | 28.5 | 31.1 |

### 7.7.5 이미지 분할 문제 (Image Segmentation)

어떤 이미지에서 전경(A)과 후경(B)을 pixel 단위로 분리하고자 한다. 그런데 인접한 pixel간 관계도 존재하여 어떤 pixel을 둘러싼 다른 이웃이 전(후)경이면 그 pixel의 그 주위의 무리와 같아질 가능성이 있다. 그 다음에는 확실히 foreground인 Pixel (F), background (B)인 몇 개의 Pixel을 선택한다.

그리고 두 이웃 pixel $x, y$가 같은 그룹에 속할 가능성(likelihood)를 $s(x, y)$라고 하면 우리는 다음 값을 최대가 되도록 각 pixel을 labelling(A or B)하려고 한다. 즉 같은 그룹에 속한 인접 pixel의 weight의 값을 가능한 크고, 경계를 이루는 Pixel들의 weight의 합은 최대한 적도록한다.

$$maximize \sum_{v \in A} pf(v) + \sum_{v \in B} fb(v) - \sum_{(x,y) \in (A,B) or (B,A)} s(x, y)$$

즉 이 말은 전(후)경에 속한 Pixel의 likelihood의 합은 높을수록 좋고, 두 경계에 속한 edge의 $s(x, y)$ 값은 낮을수록 좋아야 한다는 말이다. 다시 말해서 전경과 후경의 경계면의 값은 낮으수록 (뚜렷할수록) 좋아야 한다는 것이다. 그런데 이것을 maximize하는 문제는 다른 minimization 문제로 바꿀 수 있다. 이것은 전체 edge weight에서 이 값을 뺀 K를 minimize하는 것과 동일하다. 즉 $Q = \sum_{v \in V} pf(v) + pb(v)$ 일 때 아래와 같다.

$$K = Q - \left( \sum_{u \in A} pf(u, v) + \sum_{v \in B} pb(v) - \sum_{(x,y) \in (A,B) or (B,A)} s(x, y) \right)$$

결국 우리는 다음을 minimize하면 된다.

$$minimize \sum_{v \in B} pf(v) + \sum_{v \in A} pb(v) + \sum_{(x,y) \in (A,B) or (B,A)} s(x, y)$$

이것은 결국 super node S에서 T로 가는 min-cut을 찾아내는 문제와 동일해진다. super source, super sink $S, T$ 를 모든 pixel $(i, j)$와 edge를 주고 각각의 capacity를 $pf((i, j))$, $pb((i, j))$ 로 지정한다. 따라서 우리는 이미지 grid pixel graph에서s에서 t에 이르는 maximum flow를 찾고, 그것에 바탕하여 minimum cut을 찾으면 그 cut을 중심으로 S와 T에 달려있는 component로 전경과 후경으로 구분할 수 있다. 즉 s에서 pixel로 가는 edge가 cut되지 않으며 같은 방식으로 pixel에서 t로 가는 edge도 cut이 되지 않는다. 모든 cut edge는 image grid 상에서만 존재하게 된다.

### 7.7.6 위원회 선발 (Committee)

**문제 7.7.3** 다음 *5*개의 집합 에서 *distinct representative*를 *unit edge weight graph*의 *network Flow*로 설명하시오

$A_1 = \{1, 3, 5, 7\}$, $A_2 = \{2, 4, 4\}$, $A_3 = \{7, 8\}$, $A_4 = \{2, 7\}$, $A_5 = \{1, 3, 4\}$, $A_6 = \{5, 6\}$,

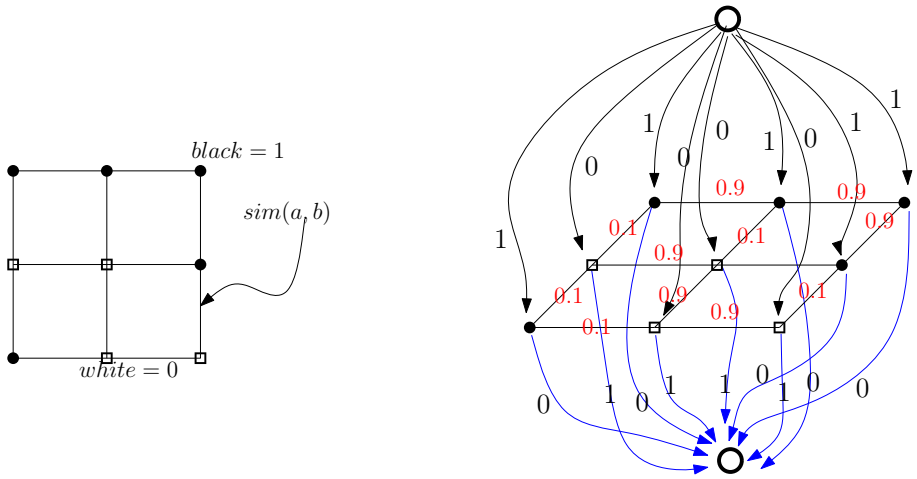


Figure 17: grid 구조의 pixel space에서 background 이미지와 foreground 이미지를 분리하기

**문제 7.7.4** 다음 각 *5*개의 집합으로 된 각각 두 종류의 집합 $A_i$와 $B_i$가 있다. 이 두 집합 $A$, $B$의 *common distinct representative*를 구하시요. 즉 $A$의 *Distinct Representative*와 $B$의 *Distinct Representative*가 집합으로 같아야 한다. 즉 $A$는 *10*명의 사람을 출신지별로 나눈 것이고, $B$는 나이별로 나누 것이다. 우리는 출신지별 *5*명의 대표를 선택하고, 나이별로도 *5*명의 대표자를 선택하고자 한다. 단 이 대표자들은 모두 같아야 한다. 다시 말해서 *5*명의 대표자가 각각 나이별, 출신지별 대표가 될 수 있는지를 판별하는 문제이다.

$$A_1 = \{1, 3, 5, 7\},\ A_2 = \{2, 4, 9, \},\ A_3 = \{7, 8, 9\},\ A_4 = \{4, 7, 10\},\ A_5 = \{1, 3, \}$$
$$B_1 = \{2, 5\},\ B_2 = \{3, 6, 7\},\ B_3 = \{2, 4, 7\},\ B_4 = \{1, 2, 10\},\ B_5 = \{1, 4, 6\}$$

**문제 7.7.5** *200*명의 교수로 구성된 $A$ 공과대학에는 *15*개의 학과 *(department)*가 있다. 각 교수는 서로 *disjoint*하게 정교수, 부교수, 조교수로 구분되어 있다. 이 공과대학에는 교수연구업적 평가위원회를 구성하려고 하는데 이 위원회는 각 학과 소속에서 학과를 대표하는 **서로 다른** *15*명의 교수로 구성되어야 한다.

그런데 어떤 교수는 *2*개 학과 이상의 학과에서 겸임으로 재직하기도 한다. 따라서 각 학과 교수님의 총합은 공대 재직 교수님인 *200*명보다 많을 수 있다. *(*겸직 교수 때문에*)*. 그리고 이 위원회는 형평성을 위해서 조교수 *5*인, 부교수 *5*인, 정교수 *5*인으로 구성하고자 한다. 이 경우 어떻게 교수님들을 선발하면 되는지 이 문제를 *network flow* 문제로 해결해보시요. □

Sol) 먼저 하나의 unit flow가 한 교수가 committee에 assign되도록 network graph를 구성해야 한다. 그리고 조교수, 부교수, 정교수는 반드시 5인이 되도록 constraint가 되어야 한다.

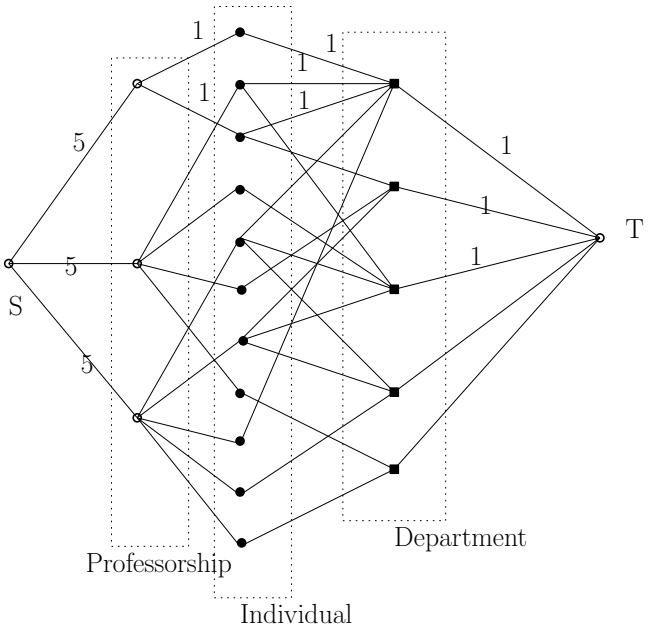


Figure 18: A network Graph for Committee member Selection

**문제 7.7.6** *(**위 문제의 발전형** )*
위 위원회 선발 문제에서 새로운 조건이 추가된다. 교수님마다 각각 연구업적 점수가 최대 *100*점 만점으로 지정되어 있다. 우리는 평가 위원회를 구성하되 그 위원회 소속 교수님들의 연구업적 점수의 합이 최대가 되도록 하고자 한다. 취지는 연구업적이 뛰어난 교수들이 다른 교수들의 업적을 제대로 평가할 수 있다고 믿어지기 때문이다. 어떻게 하면 이런 취지로 평가 위원회를 구성할 수 있는지 이를 위한 *Network Flow*을 제시하고 그 방법을 설명해보시요. □

### 7.7.7 도미노 패킹 (Domino Packing)

도미노란 두 개의 unit square가 합쳐진 $2 \times 1$ 직사각형을 말한다. 우리는 임의의 격자 공간을 이 도미노로 모두 채울 수 있는지를 알아내고자 한다.

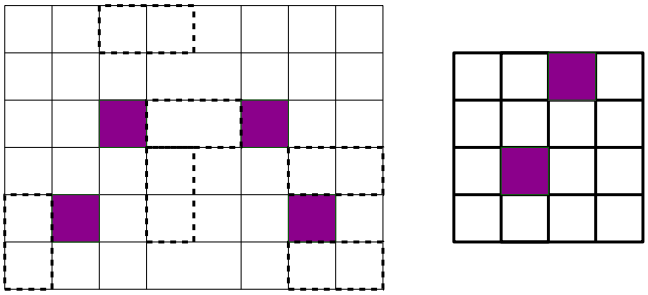

Figure 19: 격자에서 놓을 수 없는 자리가 있을 때 domino로 packing할 수 있는지의 여부

이 문제를 maxflow로 해결할 수 있다. 일단 각 비어있는 Cell과 인접한 cell은 domino로 커버가 가능하므로 이를 이분그래프로 만들고 그 그래프에서 perfect matching이 존재하는지를 살펴보면 된다.

### 7.7.8 스포츠 팀 우승 가능성 (Sport Team Elimination)

어떤 야구리그가 있다. 예를 들어 5개의 팀으로 구성된 자갈치 시리즈가 있다고 하자. 특정 팀, 예를 들어 $x_0$가 우승할 가능성이 있는가를 알아보려고 한다. 즉 다른 팀들이 얽혔을 때의 경우를 고려하는 것이다. 예를 들어 롯데가 남은 게임 25게임에서 모두 이겨서 총 승수가 87승이라고 할 때, 나머지 팀들 중에서 최고의 승수를 가진 팀의 승수가 87승 이하가 될 수 있는가를 따져보는 문제이다.

간단하게 생각하면 롯데만 계속 모든 경기를 이기고, 다른 팀은 하나의 승리로 없어 모두 패가 기록한다고 가장 하면 (만일 롯데가 현재 65승이고 다른 팀들이 70승, 71승, 64승.. 이라고 할 때) 롯데의 승리가 보장될 것 같지만 이런 상황은 불가능하다. 모든 게임에서는 반드시 이기는 팀이 있고 지는 팀이 있기 때문에, 모두가 하나의 승리를 추가할 수 없는 상황을 나타나지 않는다.

일단 롯데가 남은 게임에서 모두 이겨서 W 승을 한다고 하자. 그러면 나머지 팀들 간의 시합에서 그들이 얽혀서 모두 $W$ 이하의 승수를 얻을 수 있는가 하는 것이다. 일단 이 자료에서 network 을 구성해보자.

아래 그림에서 $x_i$는 팀을 나타낸다. 먼저 $x_i$를 한 stage에 모두 세우고, 그 모든 가능한 쌍들 즉 $\binom{n}{2}$ 개의 $y_{i,j}$로 구성된 다른 stage를 또 세운다. $y_{i,j}$는 그리고 추가로 s(start)와 t(sink)를 만들어 $x_i$는 모두 s와 $y_{i,j}$는 T와 연결한다.

각 팀이 지금까지 이긴 게임을 $w_i$라고 한다. 그리고 $x_i$와 $x_j$ 팀간 남아있는 게임의 수를 $a_{i,j}$라고 표시한다. 우리는 이 network 그래프에서 각 edge의 weight를 다음과 같이 지정한다.

1. $weight(s, x_i) = W - w_i$

2. $weight(x_i, y_{i,j}) = \infty$
3. $weight(y_{i,j}, T) = a_{i,j}$

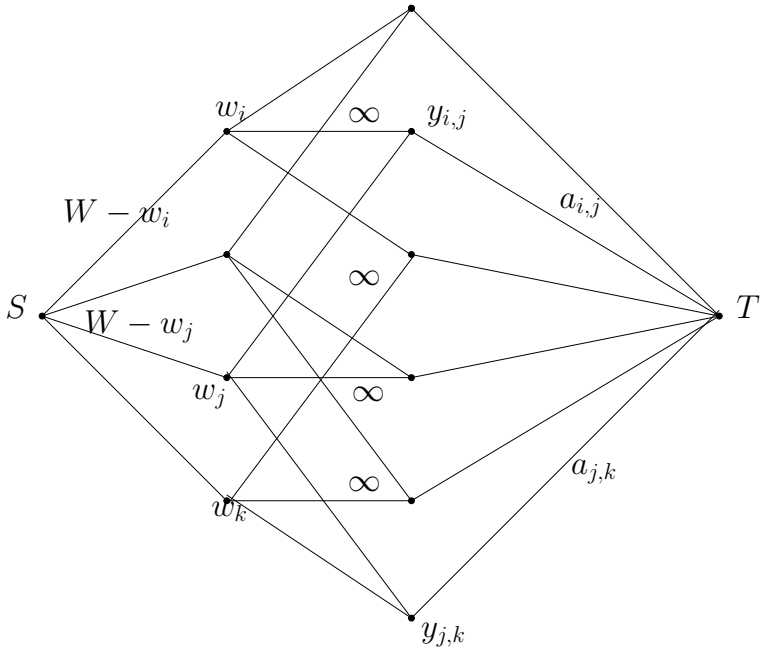


Figure 20: 야구 리그에서 현재의 성적으로 우승팀이 될 수 있는지 여부 알아보기

하나의 integral flow network은 하나의 게임(game)을 나타냅니다. 만일 $x_i$가 20 게임이 남은 상황에서 첫 stage의 edge $(S, x_i) = W - w_i$의 값은 승수 W를 달성하기 위하여 남은 게임에서 이겨야하는 게임의 수를 나타낸다. 예를 들어 롯데가 현재 승수 45/90 게임에서 (전체 게임은 130게임이라고 하자) 나머지 게임이 40게임을 모두 이긴다고 하면, $W = 45 + 40 = 80$이 된다. 그리고 두산이 현재 이긴 승수가 53이라고 한다면 $W - x_j = 80 - 53 = 27$이 된다. 이런 식으로 삼성, KIA 등의 첫번째 edge값을 구한다. 이 네트워크에서 $S = \sum a_{i,j}$의 flow를 가지는 경우는 if and only if 모든 남은 게임들이 절묘하게 엮여서 어떤 팀도 W를 넘지 않는 경우이다. Min-Cut max-Flow 정리에 의해서 S가 되기 위해서는 $a_{i,j}$가 모두 saturated되는 경우이다.

s에서 나가는 edge weight의 합은 각 팀들이 W를 넘지 않는 상황에서 최대로 나눠서 이길 수 있는 게임의 들의 총 합니다. 그렇게 각 팀이 W를 넘지않는 상황에서 "흘려보낸" 게임의 합이 만일 $S = \sum a_{i,j}$이라면 $x_0$가 우승하는 시나리오가 존재한다. 역으로 $x_0$가 승수 W로 우승하는 길이 있다면 이 network에서의 maximal flow는 반드시 $S = \sum a_{i,j}$가 되어야 한다.

**문제 7.7.7** 실제 예를 들어 보자. *4*개의 팀 *A, B, C, D*가 있다. 각 팀간 시합은 모두 *15*번씩 한다. 그래서 한 팀은 모두 *45*번의 시합을 한다. 현재까지 진행된 각 팀별 승패는

다음과 같다고 하자. *w-l*는 두 팀간의 승-패를 나타낸다.

| *()* | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| *A* | □ | *2-6* | *2-8* | *3-7* |
| *B* | *6-2* | □ | *5-5* | *4-7* |
| *C* | *8-2* | *5-5* | □ | *3-6* |
| *D* | *7-3* | *7-4* | *6-3* | □ |

현재 *A*는 모두 *8 + 10 + 10 = 28*의 게임을 치룬 상태이고, 승수는 모두 *7*게임, 패한 게임은 모두 *21*게임이다. *A*팀이 만일 남아있는 *45-28=17*게임을 기적과 같이 다 이기면 모두 *24*게임을 이기게 되고 승률은 $24/45 = .533$이 된다. 그리고 나머지 팀들이 지금의 상황에서 자기들끼리 묘하게 치고받고 해서 *A*팀보다 적은 게임을 가져갈 가능성이 있는지를 알아보고자 한다. 그렇다면 현재 *B*틈의 승수*(*게임수*)*는 *15*승*(29*게임*)*, *C*는 *16*승*(29*게임*)*, *D*팀은 *20*승*(30*게임*)*이다. 따라서 각 팀의 남은 게임의 수는 *45-29, 29, 30*이 남아있다. 모든 경우를 감안해서라도 이것이 가능한지를 *max-flow*로 알아보자. 이 문제에서 *W=24*가 된다.

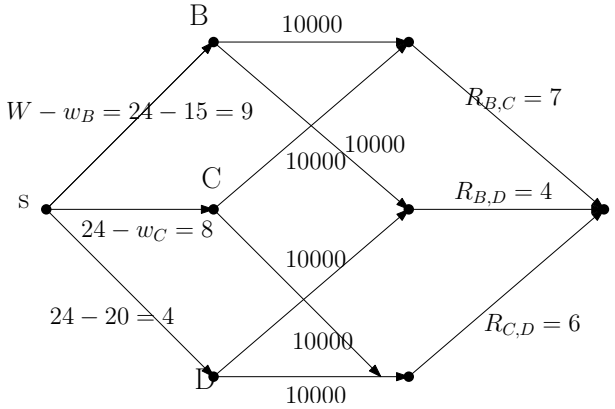


Figure 21: 4개의 팀이 있는 리그에서 A팀이 모든 이후의 게임을 이겨서 최종적으로 24승을 하게 될 경우 나머지팀들의 성적이 따라서 우승할 가능성이 있는지를 max-flow로 해결하기. 위 그림에서 $R_{b,c} = 5$가 되어야 한다. typo이지만 그림을 수정할 수 없이 여기에 부기한다.

### 7.7.9 Consistent Rounding

또 다른 응용 문제인 주어진 행렬의 원소값을 적절히 rounding하는 matrix consistent rounding 문제를 생각해보자. 이 문제는 각 matrix의 원소 float $m_{i,j}$ 의 값을 rounding down, 또는 up을 하되 그것이 모두 up 또는 down이 되면 행렬의 전체 특성이 바뀌므로, 그렇게 올리고 내려서 정수로 만든 각 원소의 값의 합이 원래 row sum, column의 sum과 차이가 1이상 나지 않도록 하는 것이다. 아래 예를 살펴보자.

$$\begin{pmatrix} 6.3 & 7.6 & 4.6 \\ 4.7 & 2.3 & 2.8 \\ 5.5 & 4.5 & 3.5 \end{pmatrix} \tag{1}$$

**문제 7.7.8** 위에 제시된 행렬에 대응하는 아래의 아래 *network FLow*를 보고 이것이 어떻게 만들어 졌는지, 그리고 여기에서의 *maximal flow*가 어떻게 주어진 문제의 답이 되는지를 설명하시오. □

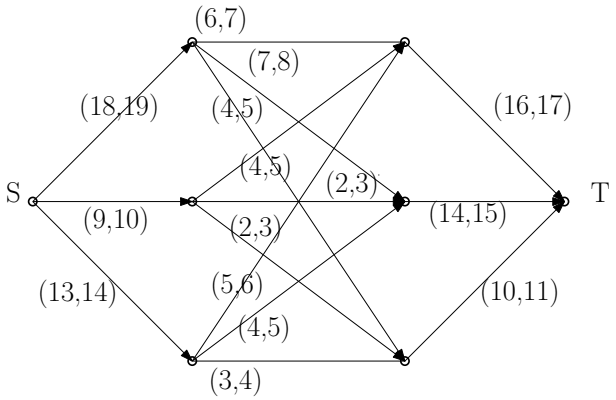


Figure 22: 행렬 matrix의 올림(round on) 또는 내림(round off)을 통하여 가장 적절한 integer matrix 만들기

## 7.8 설문지 구성하기 (Survey Design)

$n$ 명의 소비자와 $p$ 개의 제품이 있다. 각 사용자 (user) $u_i$ 는 $\{p_i, p_j ... p_w\}$ 의 물픔을 가지고 있다. 우리는 각 사용자에게 어떤 설문을 통하여 그 사용자가 가지고 있는 제품에 대하여 알아보고자 한다. 단 사용자 $u_i$ 마다 물어보아야 할 질문의 수는 $[c_i, c_i']$ 범위의 안으로 결정되어 있다. 예를 들어 4명의 사용자에 대하여 각 사람에게 물어볼 질문이 [2,3], [0,1], [1,4], [3,3] 으로 한정할 수 있다. 즉 $u_1$ 에게는 최소 2개, 최대 3개까지 물어볼 수 있다. 그리고 각 제품 $p_i$ 에 대해서도 최소, 최대까지 질문, 즉 $[w_i, w_j']$ 범위로 물어볼 수 있다. 예를 들어 어떤 제품에 대해서는 적게 물어볼 수도 있고 특정 제품에 대해서는 많이 물어볼 수 있다. 만약 특정 제품의 선전비를 많이 지급 받는 경우 중복으로 많은 질문을 던져볼 수 있다. 예를 들어 설명해보자.

4명의 사용자가 있으며 7개의 제품이 있다. 4명의 사용자에게는 각각 [2,3], [0,1], [1,4], [3,3] 개의 질문을 할 수 있다. 또 7개의 제품에 대해서는 범위 [1,2] 에 대한 질문을 할 수 있다. 즉 최대 2개 최소 1개의 질문을 할 수 있다. 각 사용자별 소유하고 있는 제품에 대한 정보는 이미 주어져있다. 일단 이 정보에서 network을 만들어야 한다. 만드는 방법은 다음과 같다. 먼저 bipartite graph $B(U, P)$ 를 만든다.

1. 각 사용자 $x$ 가 소유하고 있는 제품 $y$ 에 대하여 edge $(x, y)$ 를 추가한다.

2. 시작점 source S에서 각 사용자에 대하여 edge를 넣고 flow의 capacity를 각 사용자에게 제한된 질문의 수 $[c_i, c_i']$를 부여한다.
3. 각 물품에 대하여 sink $t$와 edge를 연결하고 그 flow의 capacity를 $[w_i, w_j']$로 제한한다.
4. 마지막으로 $t$에서 $s$로 circulation edge를 지정하고 그 capacity를 $[0, \infty]$로 제한한다.

만일 이 조건을 만족하는 설문지가 가능하면 아래 demand-supply network에서 circulation이 가능함을 보이는 것으로 해결된다.

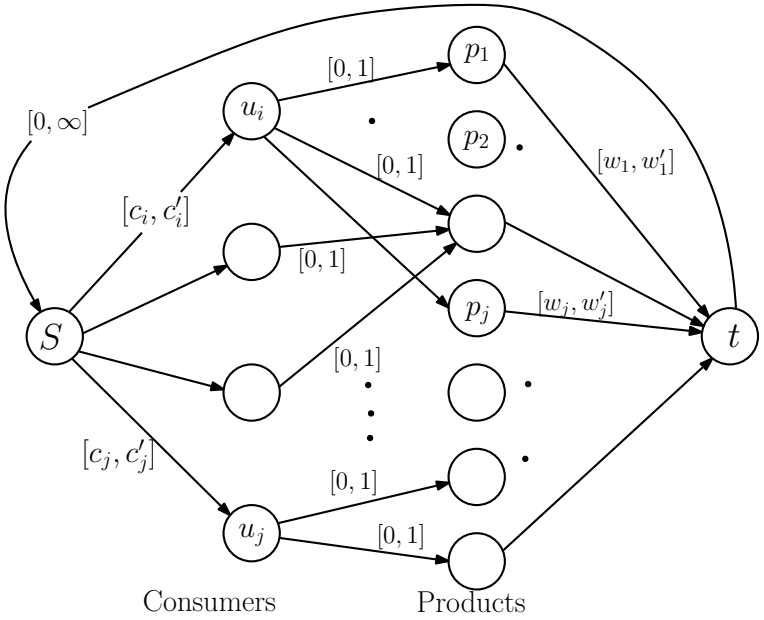


Figure 23: 사용자가 소유한 제품에 대하여 다양한 조건을 가진 survey 구성하기

### 7.8.1 지하철 시간표 (Subway Time-Table)

제시한 행렬 rounding 문제와 그 학샘이 유사한 문제가 2009 스톡홀름에서 열린 ACM 국제대학생 프로그래밍 대회(ICPC 2009)에 나왔다. 문제는 아래 그림으로 표시된다. 문제는 내용은 다음과 같다. 스톡홀름에는 전철이 있는데 각 역구간을 지나는 전철의 운행시간은 초단위로 계산된다. 그러나 실제 시민들에게 초단위로는 제시할 수 없어, 분 단위로 rounding을 하여 제시한다. 각 노선의 양 터미널에서 터미널까지 가는 시간은 반드시 분 단위 안의 여유로 지켜져야 한다. 즉 그 시간이 32분 15초이면 32분, 또는 33분이 되어야 한다. 이렇게 했을 때 아래 그림과 같이 3개의 노선이 있고 그 처음과 끝이 terminal node에 있을 때, 여러분은 각 구간의 시간을 rounding on 또는 round off해서 각 노선의 운행시간을 맞출 수 있겠는지를 판별하는 것이, 이 문제의 내용이다.

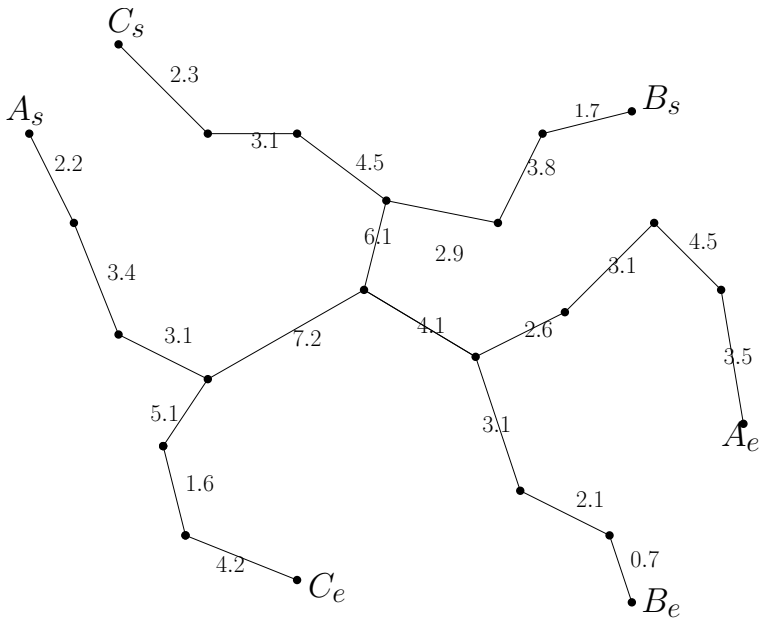


Figure 24: 구간별 전철 시간표(분단위) 만들기. 각 edge의 소숫점이 있는 분단위 값이다.

위 그림에서 전철의 A, B, C 3개의 전철선으로 구성되어 있다. 각 전철의 시작은 $X_s$, 끝은 $X_e$로 표시되어 있다. 시작과 끝은 사실상 별 의미는 없다. 그리고 각 단위 역 구간을 어떻게 rounding을 하든지, 그 전체의 값은 전체 구간을 지나는 시간의 분 단위이상 어긋나면 안된다. 이 문제를 직접 한번 풀어보자.

## 7.9 안전한 백업망 구성

컴퓨터 또는 작은 단위의 소자로 구성된 네트워크 망이 있다. 이 망에서 각 서버들은 안전을 위하여 인접한 서버쪽으로 항상 백업을 보내서 서로의 안전을 도모한다. 그 백업의 개수는 $k$로 주어진다. 만일 $k = 3$이라면 항상 인접한 3개의 이웃 서버로 자료를 보내서 백업을 저장한다. 그런데 이렇게 백업을 구성할 때 한 서버가 지나치게 많은 이웃 서버의 백업을 관리하게 할 수는 없다. 극단적으로 모든 서버가 하나의 서버를 지정해서 그곳에만 백업을 한다면 만일의 상황에 대비할 수 없기 때문이다. 따라서 어떤 경우라도 $b$개 이하의 백업본만을 저장하도록 강제한다.

다른 상황이라면 $n$개의 IoT device $\{d_i\}$가 공간에 배치된다. 일반적으로 비행기를 통하여 뿌린다. 그런데 이 소자는 파워나 계산능력이 약해서 충전량(태양광의 경우)이 일정 이하가 되면 거리 $D$이하인 이웃의 소자에게 자신이 수집한 자료를 인접한 소자로 백업을 실시한다. 이런 경우도 동일하다.

어떤 그래프가 주어지고 $k$와 $b$가 지정될 때 이 조건을 만족하는 백업 사이트, 즉 각 서버 $s_i$에 대하여 이것과 인접한 $k$개 백업 사이트를 지정하는 방법을 생각해보자. 단 이 경우에 어떤 서버도 $b$개 이상의 백업본을 받으면 안된다. 이 문제를 max flow로

풀어보자.

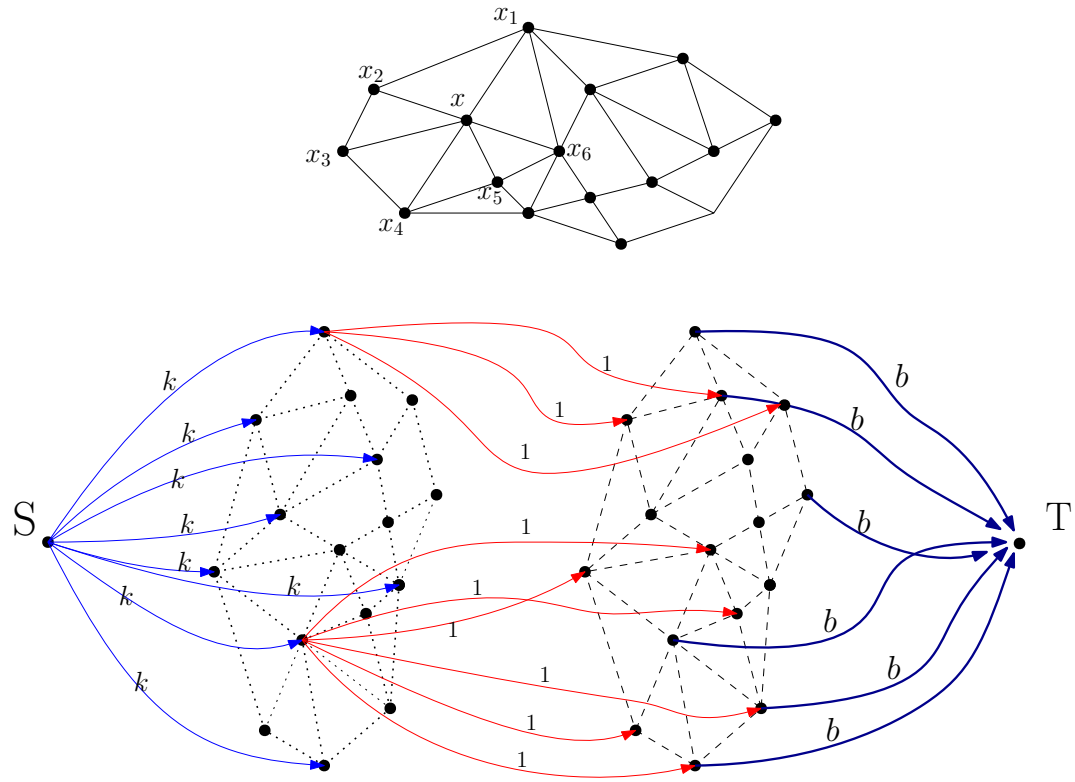


Figure 25: 네트워크망에서 Backup 서버 설정하기

## 7.10 프로그래밍 과제 : Maxflow 구현하기

이번에는 NetworkX를 이용해서 실제 프로그래밍을 해본다. 그래프 입력은 앞서의 문제와 같이 첫 줄에 그래프 이름이 주어진다. 그 다음 줄에는 vertex의 갯수 N과 edge의 갯수 M이 주어진다. 그 다음 이어지는 M개의 줄에는 각 directed edge의 정보 $(u, v)$와 해당 edge의 capacity가 다음과 같이 3개의 정수로 주어진다. 그리고 항상 source node의 번호는 1번이고 sink node의 번호는 N의 고정되어 있다.

$$u\ v\ capacity(u, v)$$

**문제 7.10.1** 여러분은 *5*개의 *sample graph G{1,2,3,4,5}.inp*를 이용해서 실제 이 그래프의 *vertex, edge connectivity* 가 얼마인지 계산하는 프로그램을 *algorithm repository*의 프로그램을 이용해서 작성하시요. *(Mathematica*에 있는 *Combinatorica*를 이용해도 좋다.*)* 입력 형식은 이전 *maximal weighted matching*에 사용한 양식으로 그대로 사용하면 된다. 여하간 제시된 *5*개의 그래프에 대하여 이 값을 구하면 된다. 단 모든 그래프의 *edge weight*는 정수로 주어진다. *(edge connectivity* 계산할 때*)* □

**문제 7.10.2** 위의 문제에서 우리는 실제적인 *minimal cut vertex set*과 *minimal edge cut*을 찾아내려고 한다. 단 이러한 *cut set*의 많이 존재하기 때문에 이 들을 사전식으로 나열했을 때 *lexicographically* 가장 빠른 것을 출력하면 된다. 즉 만일 복수개의 *cut set*이 존재하면, 즉 $\{1, 4, 6, 7\}$이 있고, $\{2, 3, 5, 6\}$ 이 있으면 여러분은 전자 *(former)*가 사전식으로 더 빠른 것이므로 $\{1, 4, 6, 7\}$을 출력해야 한다. □

**문제 7.10.3** [ **NetworkX 프로그래밍 문제 : network Flow** ]
여러분은 앞서 제시된 *5*개의 *sample graph*에서 *vertex* $v_)1$ 과 $v_{n=|V(G)|}$사이의 *maximal network flow*와 *minimum cut*을 *NetworkX* 모듈을 이용해서 구하시요.[2] 방법은 위의 문제와 동일하다.
□

[2]2019년부터 NetwprkX를 이용해서 구현 함.