Lecture 8 : Graph Connectivity *

8.1 그래프 연결도(Connectivity)의 의미

Connectivity(연결도) 분석은 network 설계에서 가장 흔히 나타나는 문제이다. 즉 그래프가 얼마나 튼튼하게 연결되었는지, 몇 개의 edge를 제거할 경우에도 얼마나 버틸 수 있는지(fault-tolerant), 임의의 두 정점 간에 서로 독립된(mutually disjoint) 경로는 몇 개가 존재하는지 등을 분석하는 것이 연결성 분석의 주요 주제이다.

이 장에서 주로 익혀야 할 내용은 다음과 같다. (1) 특정한 그래프의 edge, vertex connectivity를 어떻게 계산하는가, (2) parameter를 가진 다양한 특징의 graph가 가지는 connectivity의 lower bound, upper bound에 대한 이해이다.

이 장에서 제시된 가장 중요한 정리는 멩거(Menger)의 정리(Menger's theorem)인데 이 정리를 바탕으로 많은 유용한 그래프 알고리즘을 유도할 수 있기 때문에 잘 익혀두어야 한다.

문제 8.1.1 다음 용어를 그래프 이론적으로 정의하시요. 단 이전에 배운 정의를 최대한 활용해서 정의해야 한다.

1. *separating set and vertex cut, cut vertex*
2. *vertex connectivity* $\kappa(G)$
3. *separating edge, edge cut*
4. *edge connectivity,* $\kappa'(G) = \lambda(G)$
5. *k-vertex connected graph*
6. *k-edge connected graph*

한 가지 주의해야할 base condition은 degenerated(퇴화된) 그래프의 경우, 즉 vertex가 1개, 2개 있는 아주 특수한 경우인데 이 경우에 connectivity를 따진다는 것이 좀 이상하지만 정의의 consistency를 위해서 한 개의 점으로 구성된 그래프에 대해서는 $\kappa(K_1) = 0$ 로 정의한다.

*, 2013년 몇 개의 그림을 추가하여 재구성함. IPE 7.1.4 를 사용함. Network flow에 관련된 내용은 11장으로 독립시킴. 2015년에 maxflow를 먼저하도록 옮김

8.2 블록(Block): 2-connected component

cut-vertex의 관점으로 보면 주어진 그래프는 block들로 구성된다. 이 경우 cut-vertex는 각 vertex에 중복되어 나타난다. Social Graph에서 block과 cut-vertex는 중요한 의미를 가진다. 이 장에서는 block의 성격에 대하여 설명하고자 한다.

문제 8.2.1 어떤 *social graph(network)*에서 *cut-vertex*는 어떤 의미를 가지는지 해석할 수 있는 다양한 예를 들어 설명하시오.

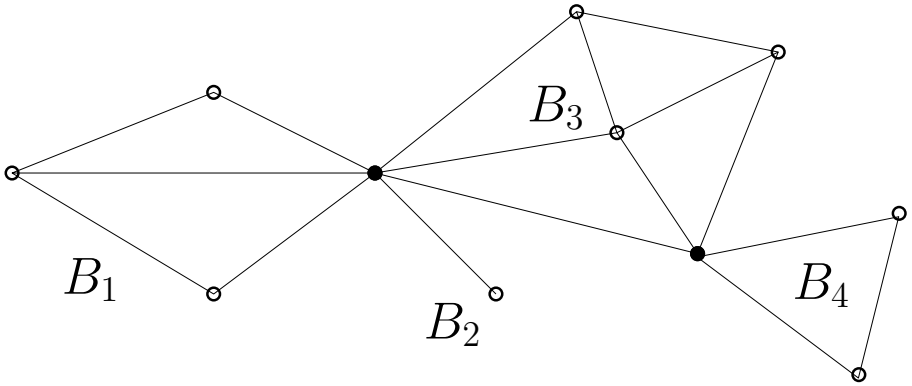


Figure 1: A graph and its 4 blocks

정의 8.2.1 *A nontrivial connected graph with no cut-vertices is called a* ***block, nonseparable graph****.*

문제 8.2.2 *Prove the following. A graph of order at least 3 is nonseparable(block) if and only if every two vertices lie on a common cycle.*

문제 8.2.3 *Prove the following statement. If G is a graph with $\rho(v) \geq n/2$ for every vertex, then G is nonseparable.*

문제 8.2.4 어떤 그래프의 *2-connected subgraph(black)*을 연결된 모양으로 그래프를 축소한 것인 *block graph*이다. 이 그래프는 항상 *Tree*임을 밝히시오.

문제 8.2.5 *Prove or disprove. If G is a connected graph with cut-vertices and u and v are vertices of G such that $d(u,v) = diam(G)$, then no block of G contains both u and v.*

문제 8.2.6 *Give an example of a graph G with the following properties.*

(a) every two adjacent vertices lie on a common cycle.
(b) there exist two adjacent edges that do not lie on a common cycle.

8.3 정점 연결도, 에지 연결도(Connectivity)의 기본 특성

가장 단순한 사실부터 먼저 증명해보자. $\kappa(G)$는 정점 연결도를 나타내고 $\kappa'(G)$은 에지 연결도를 나타낸다.

정리 8.3.1

$$\kappa(G), \kappa'(G) \leq \min_{v \in G}\{\rho(v)\}$$

즉 어떤 그래프의 minimum degree가 4이면, 이 그래프의 $\kappa(G), \kappa'(G)$는 항상 4 이하이라는 말이다.

문제 8.3.1 $\kappa(G) \geq k$임을 증명하기 위해서는 어떤 방법을 사용해야 하는지 설명하시요. □

Sol) 직접 증명을 하자면 하나의 cut set S를 선택하여 그 크기가 $|S| \geq k$임을 보인다. 또는 어떤 k개 이하의 vertex set S를 선택해서 이를 제거하여도 $G - S$는 항상 connected됨으로 보여주는 것이다.

문제 8.3.2 *HyperCube* Q_k의 *vertex connectivity, edge connectivity*를 구하시요. □

Sol.) 먼저 Q_2, Q_3의 경우를 그려본다. 만일 S가 separating set 이라고 가정한다. Q_{k+1}은 두 개의 Q_k와 각 대응되는 vertex들을 서로 연결한 것이므로 이 사실을 이용해서 증명해야 한다. 만일 Q_k에서 S를 들어내었을 때 각각의 하위 단계인 2개의 $Q_{A,k-1}, Q_{B,k-1}$의 연결성을 먼저 살펴본다. 만일 한쪽의 $Q_{A,k-1}$이 단절되었다면 S가 어떤 특성을 가지게 되는지 한번 살펴보자.

정리 8.3.2 G에서 c가 *cut-vertex*이면 $G - c$에서 서로 다른 *component*에 속한 두 *vertex* x, y를 연결하는 모든 $path(x, y)$는 반드시 c를 지난다. □

문제 8.3.3 *de Brujin Graph* dB_k의 *vertex connectivity, edge connectivity*를 구하시요. 단 사용되는 기호는 *2*개 *{ 1, 0 }* 이다. □

문제 8.3.4 Whitney Theorem 1932
$\kappa(G) \leq \kappa'(G) = \lambda(G) \leq \delta(G) = minimum\ degree$를 증명하시오. 그리고 위의 식에서 각각의 등호가 성립하는 경우와 부등호가 성립하는 경우에 해당하는 그래프를 그리시오. □

문제 8.3.5 어떤 그래프 G에서 v가 *cut-vertex*이면 G^{-1}에서는 *cut-vertex*가 아님을 보이시오. □

문제 8.3.6 다음 제시된 주장이 참이 아니라면 그 반례 *(counter example)*을 제시하시오. *(Chartrand p.110, Problem 5.5)*

1. *For G, $|G|$=13 and it is connected. and v is cut vertex of G. Then there exists a component of $G - v$ containing at least 7 vertices.*
2. *G is a connected graph containing only even vertices, then G contains no cut-vertices.*
3. *If G is a connected graph with a cut vertex, then G contains a bridge.*
4. *If G is a connected graph with a bridge, then G contains a cut-vertex.*

□

문제 8.3.7 *Prove or disprove:*

1. *If a vertex v lies on a cycle of G, then v is not a cut vertex.*
2. *If a vertex v of G does not lie on any cycle, then v is a cut vertex.*
3. *A tree of order 3 or more has more cut vertices than end-vertices.*
4. *A tree of order 3 or more has more cut-vertices than bridge*

□

문제 8.3.8 *Let G be a nontrivial connected graph. Prove that if v is an end-vertex of a spanning tree of G, the v is not a cut vertex of G.*

문제 8.3.9 *If G is a connected graph of order at least 3, then its square graph G^2 is 2-connected.* G^2는 G의 *adjacency matrix* M_G를 제곱한 *matrix* M_G^2로 표현되는 그래프이다.

문제 8.3.10 *If G is a graph of order n and $m \geq n-1$, then*

$$\kappa(G) \leq \left\lfloor \frac{2m}{n} \right\rfloor$$

Sol.) 모든 차수의 합이 $2m$이므로 평균 degree는 $2m/n$이다. 따라서 최소 degree $\delta(G) \leq 2m/n$이다. 그런데 $\delta(G)$는 integer이므로 $\delta(G) \leq \lfloor 2m/n \rfloor$이므로 앞서의 정리에 의해서 $\kappa(G) \leq \lfloor 2m/n \rfloor$가 된다.

8.4 k-Connected Graph의 주요 특성

정의 8.4.1 두 *vertex u, v*를 연결하는 두 *path*가 서로 공유하는 *vertex*가 없을 때 두 *path*는 *internally disjoint*라고 말한다. □

정리 8.4.1 *(Whitney 1932a)*
모든 *pair u, v*에 대하여 *internally disjoint path*가 있으면 *if and only if* 이 그래프는 *2-connected graph* 이다. *(Extension, k*개의 *edge disjoint path*가 존재하면 *if and only if k-connected* 이다.*)*

Proof) 왼쪽에서 오른쪽 방향으로의 증명 $\Rightarrow$는 쉽지요. u에서 v로 가는 길이 2개 이상 존재하면(모든 쌍에 대하여) 임의의 한 vertex를 제거해서 두 vertex u, v는 분리되지 않는다. Carrot ! 그리고 $\Rightarrow$ 방향은 vertex 갯수에 대한 induction으로 증명하면 된다.

문제 8.4.1 *Prove or disprove. Let G be a non-trivial graph. For every vertex v of G,*

$$\kappa(G-v) = \kappa(G), \text{ 또는 } \kappa(G-v) = \kappa(G) - 1$$

문제 8.4.2 다음에 정의된 그래프를 직접 구성해보시오. 만일 해당하는 그래프가 없을 경우에는 왜 그것이 불가능한지 설명하시오.

1. *a 2-connected graph that is not 3-connected*
2. *a 3-connected graph that is not 2-connected*
3. *a 2-edge-connected graph that is not 3-edge-connected*

4. a 3-edge-connected graph that is not 2-edge-connected

문제 8.4.3 *Give an example of a graph with following properties or explain why no such example exists.*

1. $\kappa(G) = 2, \lambda(G) = 3, \delta(G) = 4$
2. $\kappa(G) = 3, \lambda(G) = 2, \delta(G) = 4$
3. $\kappa(G) = 3, \lambda(G) = 3, \delta(G) = 2$
4. $\kappa(G) = 2, \lambda(G) = 2, \delta(G) = 3$

문제 8.4.4 *Give an example of a connected graph G such that every vertex of G belongs to a minimum vertex-cut but some edge of G belongs to no minimum edge-cut.*

8.5 맹거의 정리 (Menger's Theorem)

정리 8.5.1 어떤 두 *distinct vertex x y*의 *x,y seperator*는 x와 y를 분리하는 *(except x, y) vertex set*를 말한다. *Menger proved that the minimum size of x,y-separator* 는 *x,y*를 연결하는 *edge-disjoint path*의 *maximum size*와 같다. *(x, y*를 연결하는 *edge disjoint path*가 k 개이면 *x,y seperator*의 최소 갯수는 k 개 이다.*)* ***(Menger 1927)*** □

Proof) Case 1: G has a minimum x, y cut S other than $N(x), N(y)$ Case 2: 모든 minimum $x, y - cut$이 $N(x)$이거나 $N(y)$이 경우로 나누어서 생각하면 된다.

정의 8.5.1 *Line graph L(G)*는 그래프 *G(V,E)*의 *edge, vertex connection*의 *dual version* 이다. □

문제 8.5.1 어떤 그래프 G 의 *Line Graph* $L(G)$는 항상 존재하지만 어떤 주어진 임의의 G를 대하여 $G = L^{-1}(H)$인 H는 존재하지 않을 수 있다. 이렇게 존재할 수 없는 *Line Graph*의 예를 하나만 제시하시오. □

Sol.) Line graphs are characterized by nine forbidden subgraphs and can be recognized in linear time. 예를 들면 Line graph는 claw($K_{1,3}$-free graph이다.

정리 8.5.2 *(Dirac 1960)* 만일 G가 *k-connected* 이고 $|S| = k$인 *set* S가 있으면 G는 S로만 구성된 *cycle*를 반드시 가지고 있다. □

정리 8.5.3 *[One application of Menger theorem, Ford-Fulkerson 1958] (West Book p.172) Families* $\boldsymbol{A} = A_1, A_2, \ldots A_m$ *and* $\boldsymbol{B} = B_1, B_2, \ldots B_m$ *have a Common System of Distinct Representatives(CSDR) if and only if*

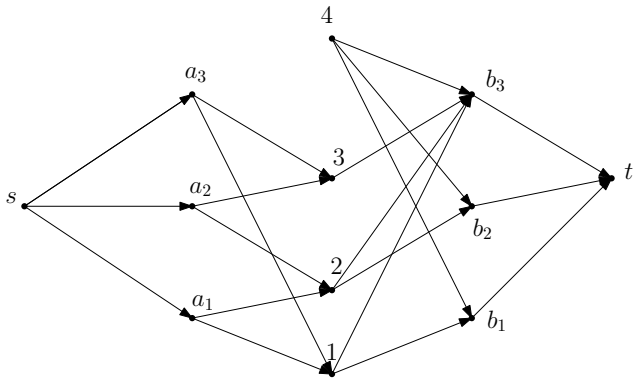
$$|(\bigcup_{i\in I} A_i) \bigcap (\bigcup_{j\in J} B_j)| \geq |I| + |J| - m$$

, for each pair $I, J \subseteq [m]$ □

이 문제는 digraph를 만들어 설명할 수 있다. $I, J \subseteq [m]$의 의미는 m개에서 임의로 I개 J를 선택한다는 말이다. 아래 그림에서 m개 이하의 s,t-cut이 없다는 말과 동일하다. s,t가 아닌 전체 집합에서 임의의 vertex set R을 선택한다. 그리고 $I = \{a_i\} - R$, $J = \{b_i\} - R$로 선택한다. 즉 R에 아닌 A쪽 vertex와 B쪽 vertex를 선택한다. 이 경우 R이 s,t-cut이면 아래가 성립해야 한다.

$$(\bigcup_{i\in I} A_i) \bigcap (\bigcup_{j\in J} B_j) \in R$$

edge, vertex connectivity는 max-flow를 정의대로 구하면 지수적, 즉 $2^{|S|-1}$ 번의 graph connectedness computing ocde를 계산해서 하므로 실용성이 없다. 그래서 이 CSDR 문제 알고리즘은 max-flow 알고리즘을 이용해서 구한다. edge connectivity는 모든 vertex 를 s,t로 설정하고 그 쌍들에 대하여 max-flow를 구해서 그 중에서 가장 낮은 값을 선택하면 구할 수 있다. 왜냐하면 그렇게 구해진 integral flow 자체가 edge-disjoint path가 되기 때문이다. 그런데 vertex connectivity를 구할 때는 문제가 좀 달라진다. 일반적인 edge disjoint한 path



$A_3 = \{1, 3\}, A_2 = \{3, 2\}, A_1 = \{1, 2\}$ $B_3 = \{1, 2, 4\}, B_2 = \{2, 4\}, B_1 = \{1, 4\}$

Figure 2: Example graph for the application of Menger theorem. 전체 4명의 사람들이 두 개의 그룹군, 예를 들어 학년별(A), 지역별(B)로 나누어져 있다. 우리는 4명의 사람중에서 서로 다른 3 사람을 선택하여 그 들이 두 군 6개 그룹의 대표가 되도록 하고자 한다.

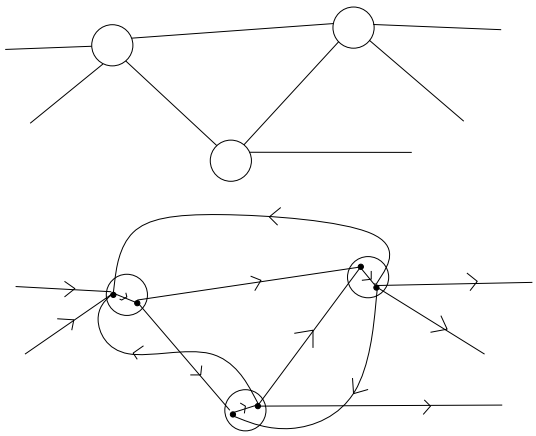


Figure 3: 그래프에서의 integral flow가 vertex disjoint한 path가 되도록 함.

는 vertex disjoint한 path는 안되므로 주어진 그래프를 좀 변형하여 그 그래프에서의 max-flow(integral flow)가 vertex disjoint하도록 되어야 한다.

먼저 G에서 s, t가 아닌 모든 vertex x는 $\overrightarrow{(x_1, x_2)}$의 edge로 쪼갠다. 이 edge의 weight는 모두 1로 둔다. 그 다음 어떤 G에서 (x, y)의 edge가 있으면 이는 G'에서는 $\overrightarrow{(y_2, x_1)}$, $\overrightarrow{(x_2, y_1)}$이 되도록 추가의 directed edge를 첨가한다. 그리고 이 edge의 capacity는 모두 충분히 큰 수로 둔다. 아래 그림을 보자.

이제 전체 그래프에서 max-flow가 vertex disjoint한 path를 찾도록 바꾸어 보도록 해보자.

문제 8.5.2 어떤 *simple graph*의 *vertex connectivity*를 구하는 방법을 *maxflow - mincut*으로 해결하려고 한다. 어떻게 하면 좋은지 그 방법을 제시해보시요. 그 때의 알고리즘을 제시하시

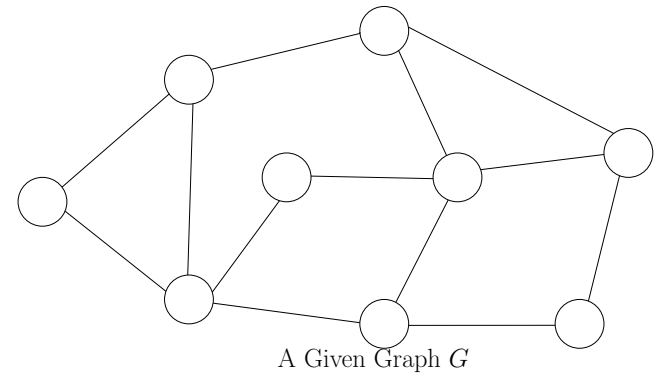


Figure 4: 주어진 그래프를 변형하여 그 그래프에서의 integral flow가 vertex disjoint한 path가 되도록 함. 모든 edge들의 방향은 s에서 나가는 edge는 모두 직진방향으로 주고, t로 들어가는 edge는 모두 들어가는 방향으로 준다. 그 외에는 한 vertex 내부는 $(v_{i,1}, v_{i,2})$로 주고, 나머지는 모두 2번째 vertex에서 첫번째 vertex로 연결되도록 방향을 준다.

요. *G(V,E)*에서 *maxflow*를 구하는데 걸리는 시간을 *M(G)*라고 한다. □

Sol.)

$$VC(G) = \min_{(u,v) \not\in E(G)} \{VC(G, u, v)\}$$

Ford-Fulkerson Method를 실제 구체적인 자료구조로 구현한 것을 Edmonds-Karp algorithm이라고 한다. 이 알고리즘의 complexity는 $O(|V||E|^2)$ 이다.

문제 8.5.3 주어진 그래프의 *edge connectivity(*가장 적은 *edge*의 수로 주어진 그래프를 *separation* 시키는 방법*)*를 *maxflow*로 구하는 방법을 제시하시요. □

Sol) 가장 간단한 Ford-Fulkerson edge relabeling 알고리즘은 이렇게 진행된다. 먼저 주어진 network flow에서 하나의 feasible flow를 구한다. 그 중에서는 0-flow도 포함된다. 그 다음 s에서 f-augmented flow를 찾아서 구한다. 이 방법은 s에서 f-augmented tree를 constructive하게 구해 나가는 과정과 같다. 즉 이미 구성된 intermediate tree를 T라고 하자. T 연결된 edge중에서 아직 visited되지 않은 edge중에서 방향이 forward이면 unsaturated edge를 첨가한다. 만일 방향이 역이고 그 flow값이 0 이상이면 이것은 모두 포함시킨다. 이 말은 반대쪽으로 들어오는 flow가 있다고 하는 것을 말한다. 이것을 역으로 되돌리는 작업을 말한다. 이렇게 해서 sink t까지 가는 길이 있는가를 찾는다. 만일 그 길이 있으면 그 edge중에 가장 작은 weight값만큼 그 값을 증가시킬 수 있으므로 f-augmented path가 존재함을 보여줄 수 있다. 문제는 그 증가분이 아주 작을 수 있다는 것이다. Bondy-Murty 책 202 page에 보면 max flow가 $2 \cdot m$ (그래프 구조는 한쪽이 m인 두 개의 arc로 구성된 cycle이 있고 그 중간에 weight 1인 edge가 있다.) 일 때 실제 증가분은 $+1$에 불과할 수 도 있다.

앞에서 제시한 deterministic VC(G)를 구하는 알고리즘의 time compelxity는 다소 높은 편이다. 이 방법은 모든 vertex pair에 대하여 하나를 source, 다른 하나를 sink로 잡아서 max flow를 구하고 그것의 minimum을 구하면 된다. 이 방식은 $O(n^2 \cdot m(n))$ 이 된다. 여기에서 $m(x)$는 maxflow algorithm의 complexity를 말한다.

8.6 그래프 연결도(connectivity) 계산 알고리즘

Network flow의 응용 중 가장 중요한 문제가 바로 주어진 그래프에의 edge connectivity와 vertex connectivity이다.

문제 8.6.1 어떤 *simple graph*의 *vertex connectivity*를 구하는 방법을 *Maxflow - Mincut*으로 해결하려고 한다. 어떻게 하면 좋은지 그 방법을 제시해보시요. 그 때의 알고리즘을 제시하시요. *G(V,E)*에서 *maxflow*를 구하는데 걸리는 시간을 *M(G)*라고 한다. □

Sol.) $VC(G) = \min_{(u,v)\notin E(G)}\{VC(G,u,v)\}$

Ford-Furkerson Method를 실제 구체적인 자료구조로 구현한 것을 Edmonds-Karp algorithm이라고 한다. 이 알고리즘의 complexity는 $O(|V||E|^2)$ 이다.

문제 8.6.2 주어진 그래프의 *edge connectivity(*가장 적은 *edge*의 수로 주어진 그래프를 *separation* 시키는 방법*)*를 *maxflow*로 구하는 방법을 제시하시요. □

Sol) 알고리즘을 이렇게 진행된다. 먼저 주어진 network flow에서 하나의 feasible flow를 구한다. 그 중에서는 0-flow도 포함된다. 그 다음 s에서 f-augmented flow를 찾아서 구한다. 이 방법은 s에서 f-augmented tree를 constructive하게 구해 나가는 과정과 같다.

즉 이미 구성된 intermediate tree를 T라고 하자. T 연결된 edge중에서 아직 visited되지 않은 edge중에서 방향이 forward이면 unsaturated edge를 첨가한다. 만일 방향이 역이고 그 flow값이 0 이상이면 이것은 모두 포함시킨다. 이 말은 반대쪽으로 들어오는 flow가 있다고 하는 것을 말한다. 이것을 역으로 되돌리는 작업을 말한다. 이렇게 해서 sink t까지 가는 길이 있는가를 찾는다. 만일 그 길이 있으면 그 edge중에 가장 작은 weight값만큼 그 값을 증가시킬 수 있으므로 f-augmented path가 존재함을 보여줄 수 있다. 문제는 그 증가분이 아주 작을 수 있다는 것이다. Bondy-Murty 책 202 page에 보면 max flow가 2m (그래프 구조는 한쪽이 m이 두 개의 arc로 구성된 cycle이 있고 그 중간에 weight 1인 edge가 있다.) 일 때 실제 증가분은 +1에 불과할 수 도 있다.

우리는 어떤 그래프 G에서 u. v의 vertex disjoint path의 수를 $\kappa(G,u,v)$로 표시하자. 앞서의 max-flow 알고리즘은 edge disjoint한 path를 구해주므로 이것을 vertex disjoint한 판으로 바꿀 필요가 있다. 일단 이렇게 바뀌어 졌다고 가정하면 그래프의 $\kappa(G)$는 모든 nonadjacent한 vertex pair에 대하여 계산한 뒤 그 최하값을 구하면 된다. 즉 다음과 같다.

$$\kappa(G) = \min_{u,v} \kappa(G,u,v)$$

이렇게 하면 모두 $n(n-1)-m$만큼의 max-flow algorithm을 불러야 한다. m은 edge 개수로서 바러 연결된 edge (u,v)에 대해서는 계산할 필요가 없다. 그런데 이것을 조금 줄일 수 있다.

만일 $\kappa(G) = S$ 일 때 S+1개의 vertex를 선택하면 그 중에는 vertex cut에 속하지 않는 vertex가 반드시 한 개 이상 존재한다. 게다가 $\kappa(G) \le \delta(G)$임을 알고 있기 때문에(여기에서 $\delta(G)$는 minimum degree이다.), $|X| \ge \delta(G)$인 vertex 집합 X를 선택하면 그 안에는 반드시 minimum vertex cut 집합에 속하지 않는 vertex가 포함되어 있다. 따라서 모든 vertex u에 대하여 모든 vertex v를 고려하여 $\kappa(G,v,u)$를 구할 것이 아니라 $\delta(G)$+1개 까지만 살펴보면 그 중에는 반드시 connectivity를 결정하는 vertex가 포함되게 된다.

예를 들어 어떤 그래프의 minimum degree가 3이라고 한다면, 4개의 vertex만 임의로 선택하여 4번의 max-flow를 부르면 된다. 그리고 같은 논리로 만일 어떤 그래프의 vertex connectivity가 5라면 6개의 vertex v만 선택하면 된다.

그런데 문제는 우리가 계하려는 그래프의 vertex connectivity는 아직 모른다는 것이다. 따라서 만일 현재까지 구해진 값 중에 최소가 $\kappa(G) = W$라고 한다면 우리는 W+1개의 vertex만 구해서 조사하면 된다. 그런데 계산 중인 $\kappa(G)$는 항상 감소하므로, 우리가 시발점으로 선택한 vertex의 갯수가 현재까지 유지된 vertex connectivity $\kappa(G)$보다 커다면 그 값까지만 max-flow를 사용하면 된다. 따라서 max-flow를 부르는 전체 횟수는 $O((n-\delta-1)\kappa$번의 max-flow call이 필요하다. 제시한 알고리즘을 두 개의 loop으로 동작하는 코드를 생각했을 때 하나는 anchor vertex이고 다른 하나는 looping vertex라고 하자. 하나의 anchor vertex에 대하여 나머지 모든 vertex를 선택하여 $\kappa(G, anchor, w)$를 계산해야하므로 선택되는 $\delta+1$개의 vertex를 제외한 모든 vertex에 대하여 max-flow를 불러야하고, 그 작업은 실제 그 그래프의 vertex connectivity만큼 해야한다.

이 방식은 Hakimi, Esfahanian 알고리즘으로 조금 더 개선된다. 그들의 관찰이란 어떤 vertex가 cut set S에 속한다면 $z \in S$의 이웃 vertex중 2개는 반드시 S에 속하지도 않으며 서로 인접하지도 않다는 사실이다. 만일 z의 모든 이웃이 모두 S에 속한다면 z를 vertex cut set에서 제거해도 주어진 그래프를 분리하는데 아무런 문제가 없으므로 그것이 최소크기의 cut set이르는 가정에 모순이 된다. 만일 이웃이 한 개만 S의 외부에 있어도 같은 방식으로 증명할 수 있다.따라서 다음 2가지 경우를 나누어 계산한다. 아래에서 v는 minimum degree vertex이다.

$$k_1 = \min\{\kappa(v, w) | w \in V - \{v\}\}$$

$$k_2 = \min\{\kappa(x, y) | x, y \in N(v)\}$$

Hakim에 의해서 개선된 알고리즘을 통하면 $O(n-\delta-1+\delta(\delta-1)/2)$번의 max-flow call을 이용할 수 있다.

문제 8.6.3 *(Algorithmic Graph Theory p.217 by James A. Machugh)*
어떤 그래프의 *VC(Vertex Connectivity)*를 *Probabilistic algorithm*으로 구하는 방식을 설명하고 그 때의 *time complexity*와 그 답이 맞을 확률을 구하시오. □

Sol.) 어떤 두 vertex s, t를 분리하는 minimal vertex cut set S를 구해서 이것이 실제 그래프의 VC가 될 가능성을 생각해보자. 만일 운이 좋아서 이것이 전체 그래프의 connectivity를 결정하는 cut vertex 집합 S'과 동일하다면 문제는 바로 해결된다. 문제는 그럴 확률이 얼마나 되는가 하는 것이다.

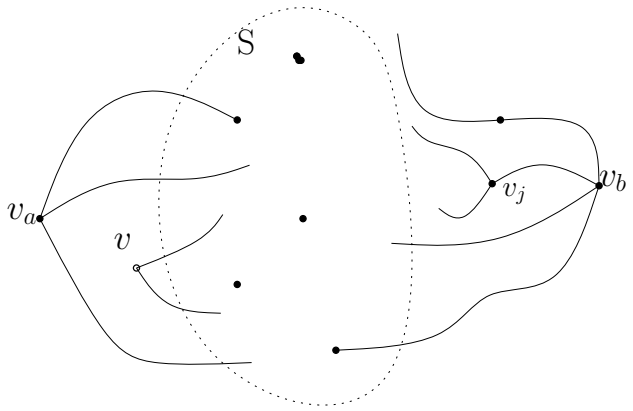


Figure 5: 주어진 그래프에서 vertex cut을 구하기 위한 확률 알고리즘의 개요

G의 vertex 모든 쌍에 대하여 max-flow를 구해보면 connectivity는 쉽게 구해진다. 그러나 한번 maxflow를 부르는 데 $O(n^{1/2}m)$이 걸리므로 이것을 $O(n^2)$번 하면 그것은 $O(n^{3.5})$까지 올라간다. 이보다 빠르고 간결한 방법을 생각해보자. 이 방법은 확률적인 방법으로 가능하다.

주어진 그래프에서 임의의 vertex set $C = \{v_1, v_2, v_3, \ldots, v_k\}$, 즉 k개의 vertex를 선택한다. 만일 C 중에서 어떤 vertex v_j가 $VC(G) = S$에 속하지 않는다고 하면 $VC(v_j, *)$), 즉 그 vertex를 중심으로 나머지 모든 vertex와의 cut set을 구하고 그 최소값을 선택하면 그것은 전체 $VC(G)$와 일치한다.

우리는 그럴 확률, 즉 적어도 하나 이상의 vertex가 cut-set S에 포함될 확률에 대하여 계산해보고자 한다. 이 확률은 당연히 $r = VC(G)/|V|$ 이다. 예를 들어 vertex connectivity가 7이고 $|V| = 30$인 그래프라면 한 vertex v_i가 cut set S에 속할 확률은 7/30이다. 따라서 이것들에 대해서 max-flow를 구하면 올바른 cut set을 찾을 수 없다. 따라서 만일 k개를 선택했을 때 이들이 모두 VC(G)에 속할 확률은 당연히 $(r = VC(G)/|V|)^k$이 되어야 한다. 따라서 집합 C에서 적어도 한 개 이상의 vertex가 cut set S에 없을 확률, 즉 S 외부에 있을 확률은 $1 - r^k$가 된다. 문제는 단 한 개라도 S에 포함되어 있지 않으면 그 답을 구할 수 있기 때문에 그 하나를 찾는 것이 중요하다.

만일 vertex가 100개 있고 $|VC(G)| = 10$ 라고 가정해보자. 우리가 무작위로 3개의 vertex를 뽑았을 때 그 중 어떤 하나가 어떤 cut set S에 포함되지 않을 확률은 $1 - 0.1^3 = 0.999$가 된다. 교재에 제시된 함수 K=RandomVC(G,k)은 그래프 G의 $VC(G)$값을 K로 대치하여 return 시켜주는 것이다. 따라서 그것이 실제의 $VC(G)$일 가능성일 확률은 $1 - VC(G)/|V|^k$이다. 그러면 어떻게 주어진 그래프의 가장 낮은 connectivity를 찾는가이다. 한가지 쉬운 방법은 가장 높은 vertex connectivity가 될 수 있는 가능성 $|V| - 1$부터 -1 씩 감소시키면서 그것에 해당하는 cut set이 있는지를 확률적으로 검사한다.

예를 들어보자. $|V| = 30$이라고 하자. 그리고 k는 5로 set한다. 그러면 우리는 주어진 G에서 5개, $v_1, v_2, v_3, v_4, \ldots v_k$를 뽑는다. 이 5개의 후보와 G의 나머지 모든 remained vertex

Decision	Authors	Year	Complexity	Comments
λ=2 or 3	Tarjan	1972	$O(m+n)$	DFS
λ	Even, Tarjan	1975	$O(m \cdot n * \min\{m^{1/2}, n^{2/3}\})$	n maxflow
λ(digraph)	Schnorr	1979	$O(\lambda \cdot mn)$	maxflow
λ	Esfahanian, Hakimi	1984	$O(\lambda \cdot mn)$	n/2 maxflow
λ(digraph)	Esfahanian, Hakimi	1984	$O(\lambda \cdot mn)$	n/2 maxflow
λ	Matula	1987	$O(mn)$	dominating set
λ(digraph)	Matula	1987	$O(k \cdot n^2)$	dominating set
λ(digraph)	Mansour, Schieber	1989	$O(mn)$	dominating set
λ	Gabow	1991	$O(m + k^2 \cdot n \cdot \log(n/k))$	uses matroids

Table 1: A chronology of edge connectivity(=λ) algorithm, $n = |V|$, $m = |E|$

r_i 에 대하여 그 쌍들 (v_i, r_j)에 대하여 cut set을 maximal flow등을 이용해서 구한다. 그 경우 실제 connectivity가 k일 가능성은 $1 - (k/30)^5$이다. 만일 우리가 set한 VC=10인데 이렇게 구한 cut set이 7이라고 하면 우리는 전체 그래프의 cut set을 다음과 같이 update한다. 즉 중간에 가정한 그래프의 connectivity보다 작은 값이 나오면 그 이하로부터 다시 시작한다.

$$CurrentVC = \min\{CurrentVC, VC(v_i, r_j)\}$$

이렇게 하면 결국에는 S에 속하지 않는 어떤 두 vertex pair (s,t)를 찾아낼 수 있고 그것으로 이 단계의 작업은 종료된다. 그 정확도는 몇 개의 vertex를 선택하는가에 전적으로 dependent 하다. (QED). 관련된 논문은 1984년 멜혼(Mehlhorn)에 의해서 IPL(information Processing Letter)에 게제된 것을 참조하면 된다. 알고리즘의 correctness에 대한 보다 자세한 분석이 제시되어 있다.

아래는 { edge , vertex } connectivity에 대하여 그 결과가 개선된 과정을 나타나고 있다. 가장 최신의 결과는 1991년에 Henzznger와 Rao에 의해서 제시된 $O(\kappa \cdot m \cdot n \log n)$ time algorithm이다.

문제 8.6.4 위에서 제시한 알고리즘을 *pseudo code*로 제시하고, 그 때의 *time complexity* 와 *probability* 를 $k, |V|, |E|$로 나타내 보시오. *LEDA*에는 직접적으로 *vertex connectivity*를 구해주는 함수가 제공되어있지 못하다. 여러분은 *LEDA*의 *max-flow code*와 위에서 설명한 확률 알고리즘을 이용해서 실제 *graph connectivity*를 확률 *0.999* 이상으로 구해보시오. □

Decision	Authors	Year	Complexity	Comments
κ=2	Tarjan	1972	$O(m+n)$	DFS
κ=3	Hopcropt, Tarjan	1973	$O(m+n)$	triconnectedness
κ	Even, Tarjan	1975	$O(\kappa\cdot(n-\delta-1)mn^{2/3})$	maxflow
κ=k	Even	1975	$O(kn^3)$	maxflow
κ	Galil	1980	$O(\min\{\kappa,n^{2/3}\}mn)$	maxflow
κ=k	Galil	1980	$O(\min\{\kappa,n^{1/2}\}kmn)$	maxflow
κ	Esfahanian, Hakimi	1984	See above	maxflow
κ=4	Kavenesky	1991	$O(n^2)$	
κ	Henzinger, Rao	1996	$O(\kappa\cdot m\cdot n\log n)$	Randomization

Table 2: A chronology of vertex connectivity(=κ) algorithm, $n=|V|,\ m=|E|$

8.7 Optimal Network Design

이 장에서는 그래프 연결도의 다양한 응용의 예에 대하여 다루고자 한다. 여기서 셜명한 것 외에도 많은 응용분야가 있으므로 인터넷 등에서 찾아보기 바란다.

문제 8.7.1 Reliable Communication Network Design
우리는 n개의 *computing server*를 가지고 있다. 그런데 좀 더 *reliable*한 통신상태를 위해서 이들을 모두 *k-connected* 되도록 만들려고 한다. 즉 $k-1$군데가 동시에 공격을 받아도 나머지들은 통신을 하는데 별 지장이 없도록 하고자 한다. 이 경우 *edge*를 아주 많이 *(hopefully complete..)* 넣으면 원하는 바를 완성하지만, 이는 경제적으로 손실이다. 이를 위해서 넣어야할 최소한의 *edge*는 몇 개인지, *Harary Graph* 구성법을 원용하여 설명하고 $n=16$, $k=4$일 때 구성을 만들어 보시요. 이 갯수를 $f(k,n)$라고 한다. 즉 $|V|=n$ 이면서 *k-connected* 인 그래프의 최소 *edge*수를 나타내자. □

Sol.) Simply saying $f(k,n)\{\geq kn/2\}$ 임을 쉽게 알 수 있다. (여기서 {x}는 x보다 큰 가장 작은 정수를 말한다. $\{3.2\}=4$이다. 이 문제에서 각 vertex는 적어도 k개의 edge는 있어야 한다, 그게 아니라면 그 놈의 incident한 edge만을 몽땅 절딴냄으로서 그것을 isolation 시킬 수 있기 때문이다. 따라서 모든 vertex의 degree는 적어도 k, 따라서 전체 edge의 수는 $kn/2$가 된다. 문제는 이런 구성으로 실제 $k-connected$ 그래프를 만들 수 있으면 땡이다.

vertex 연결도로 볼 때 흥미로운 그래프에는 해러리(Harary) 그래프 $=H_{k,n}$가 있다. 이 특정한 class의 그래프를 만드는 방법은 k,n의 parity 값에 따라서 좀 다르다. 먼저 Harary Graph의 정의를 살펴보자.

정의 8.7.1 *Given* $k<n$에 대하여 n개의 *vertex*를 원 위에 모두 등간격으로 놓는다. 만일 k가 짝수이면 $H_{k,n}$은 각 *vertex*에서 좌, 우 각각 $k/2$개의 *nearest*를 모두 연결한다. 만일 k가 홀수이고 n가 짝수이면 좌우 가까운 $(k-1)/2$개의 *vertex*를 연결한다. 그리고 추가로 반대편에 있는 *vertex*를 연결한다. 만일 k,n모두 홀수이면, $H_{k,n}$은 $H_{k-1,n}$ 에서 *edge* $(i,i+(n-1)/2)$를 추가한다. 단 $0\leq i\leq(n-1)/2$.

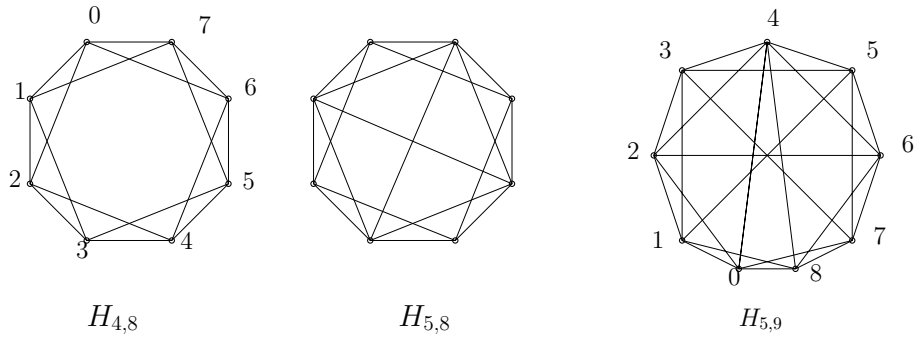


Figure 6: Structure of Harary Graph, k와 n 이 둘 중에서 하나가 짝수이면 regular 그래프이지만, 둘 모두 홀수 일 때는 정규 그래프(regular graph)가 아니다.

문제 8.7.2 *Harary Graph*를 구성하는 방법을 설명하고 $H_{4,8}$,$H_{5,8}$,$H_{5,9}$를 직접 그려보자. 그리고 이 그래프의 *vertex* 연결도가 $\kappa(H_{k,n}) = k$임을 보이시요. □

Solution) n, k에 parity에 따라서 몇 가지 경우로 나누어서 생각해보자.

1. 만일 k 가 짝수라면 ($k = 2r$)
2. 만일k 가 홀수, n이 짝수
3. 만일 k 가 홀수, n이 홀수

Proof) 일단 우리는 $k = 2r$인 경우에 대해서만 증명한다. $G = H_{k,n}$이다. minimum degree $\delta(G) = k$ 이므로 $\kappa(G) \leq k$ 이다. 따라서 $\kappa(H_{k,n}) = k$ 임을 증명하기 위해서는 우리는 $\kappa(G) \geq k$만을 증명하면 된다. 이것은 어떤 임의의 vertex 집합 $S \subseteq V(G)$, $|S| < k$에 대하여 우리는 $G-S$, 즉 S를 제거한 그래프도 항상 connected 되어 있음을 보이면 증명이 완료된다.

먼저 두 개의 임의의 vertex u, v를 선택하자. 그리고 u에서 v로 가는 두 방향에 있는 vertex 집합을 A, B로 정하자. 즉 Harary 그래프에서 시계방향으로 뽑히는 정점을 A집합이라고 하고, 반시계방향의 그래프를 집합 B라고 하자. 그런데 $|S| < k$ 이므로, 비둘기집 원리에 의해서 $\{A, B\}$ 중에 하나는 반드시 S와 $k/2$개 이하로 공통의 원소로 겹치는 vertex이 존재할 수 밖에 없다. (둘 모두가 $k/2$보다 커다면 그 갯수의 합은 k가 되고 이것은 $|S| < k$라는 가정에 모슨이 된다. 일단 S가 이 두 개의 집합 A, B중에서 circular하게 연속된 $k/2$개의 vertex가 아니라고 한다면 이 그래프는 항상 연결이 된다. 따라서 이 그래프를 분리시키려면 반드시 circular하게 연속된 r개의 인접 vertex set을 모두 제거해야만 한다.

한편 G의 모든 vertex가 한 쪽 방향으로 $k/2$개의 vertex를 연결되어있기 때문에, $k/2$개 이하의 vertex를 제거해서는 그 방향으로의 연결을 block 시킬 수는 없다. 따라서 우리는 $G - S$에서 A또는 B 중에서 $k/2$개 이하로 겹치는 집합에서 항상의 vertex를 가지는 쪽을

통하여 연결이 되는 u, v-path를 항상 찾아낼 수 있다. 즉 S에서 작게 겹치는 쪽을 통하면 항상 u와 v를 연결시킬 수 있다. (QED).

정리 8.7.1 (Harary,1962) $\kappa(H_{k,n}) = k$ 이다. 따라서 n 개의 *vertex*를 가진 k-*connected graph*가 가지는 최소 갯수의 *edge*는 $\lceil kn/2 \rceil$ 이다.

문제 8.7.3 *degree*가 모두 *3*인 *3-regular* 그래프에서 $\kappa(G) = \kappa'(G)$ 임을 증명하시요. □

Hint) vertex cut을 S라고 둔다. 그러면 전체 그래프는 H_1, S, H_2 로 분리된다. 이 경우 S에 속한 모든 vertex v는 그와 연결된 vertex 중 하나가 H_1 이나 H_2 또는 모두에 있다. 이 경우 어느 한 쪽에만 달려있는 edge를 모두 제거하면 (v하나에 대하여 하나의 edge만을 제거하면) H_1 과 H_2 를 분리할 수 있다. 이런 경우가 아니라면 교재 154에 있는 그림의 경우가 유일하다. 즉 하나씩 달려있는 edge가 양쪽에 각각 한 개씩의 경우라면 그림과 같이 걸쳐갈 수 있는 길이 없도록 한 쪽의 edge만을 없애면 된다. 따라서 $|S|$ 개 만큼의 edge만 없애면 두 component H_1 과 H_2 가 분리된다.

문제 8.7.4 *Block-cut point graph*를 정의하시요. 그리고 어떤 초등학교 *(*또는 수백명의 사원이 있는*)*에서 친구사이를 나타내는 그래프를 구성하였다고 하자. *(*가장 친한 친구 k 명을 적어내라고 하면 된다. *at most k)* 이러한 *"friendship graph"*에서 *block*는 무엇을 의미하는가 ? 그리고 이 그래프에서 *cut vertex*는 무엇을 의미하는지 말하시요. *(*또는 전화 *Calling pair* 로 그래프를 그려도 된다.*)* □

문제 8.7.5 *Connected component*를 구하는 *DFS*기반을 알고리즘을 설명하시요. *(*핵심은 *2-pass algorithm*이다. 첫번째 *pass*에서 지나가면서 모든 *vertex*의 *DFS*번호를 구하고 그 다음 *backtrack edge*를 통하여 *back-edge number*를 구하는 식으로 진행된다. 문제는 그래프 그림을 보면서 하면 쉬운 문제이지만, 실제 자료구조상에서 하면 그렇게 쉽지만은 않다. □

문제 8.7.6 G가 *simple graph*이고 *maximum degree*가 *3*이하 $\Delta(G) \leq 3$ 이면 $\kappa'(G) = \kappa(G)$ 임을 증명하시오. *(*어떻게 증명하면 되는지 그 구조부터 제시해보시요. *contradiction ? constructive proof ?* □

문제 8.7.7 모든 *edge* e에 대하여 이를 포함하는 두 개의 *cycle* C_1, C_2가 존재하고 이 둘의 공통부분은 e가 유일할 때 G는 *3-edge-connected* 임을 밝히시요. □

문제 8.7.8 *(*선인장 *(Cactus)*그래프 *).* 선인장 그래프는 모든 *block*이 에지나 *Cycle*로만 구성되어 있는 연결 그래프이다. 이 경우 가능한 *edge* $E(C)$ 의 최대 개수는 다음과 같음을 보이시오. *(Hint* $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor$*.)*

$$n - 1 \leq |E(C)| \leq \lfloor 3(n-1)/2 \rfloor$$

□

8.8 LEDA, NetworkX를 이용한 연결도 계산 프로그래밍

Graph의 edge, vertex connectivity를 위한 코드는 LEDA에서 제공해주지 아니한다. 이 문제는 해마다 조금씩 공학적으로 개선되기 때문에 최종본, 거의 최종적으로 optimization 된 코드를 확정할 수 없기 때문이다. 아래는 Graph의 vertex, edge connectivity에 관련된 알고리즘이 모두 제공되고 있는 Stony Brook Algorithm Repository 사이트이다.

`http://www.cs.sunysb.edu/~algorith/files/edge-vertex-connectivity.shtml`
실제 Graph connectivity가 implement된 각 방법은 다음과 같다.

1. Goldberg's Network Optimization Codes (C) (rating 10)
2. Boost: C++ Libraries (C++) (rating 9)
3. GOBLIN: A Graph Object Library for Network (C++) (rating 8)
4. LEDA - A Library of Efficient Data Types and Algorithms (C++) (rating 8)
5. Moret and Shapiro's Algorithms P to NP (Pascal) (rating 4)

6. Combinatorica (Mathematica) (rating 4)

이 중 하나를 선택하여 주어진 그래프의 connectivity를 구해보시오. 실험에 사용할 그래프는 다음과 같이 변형된 그래프로 하면 쉽게 그 정답의 여부를 확인할 수 있다.

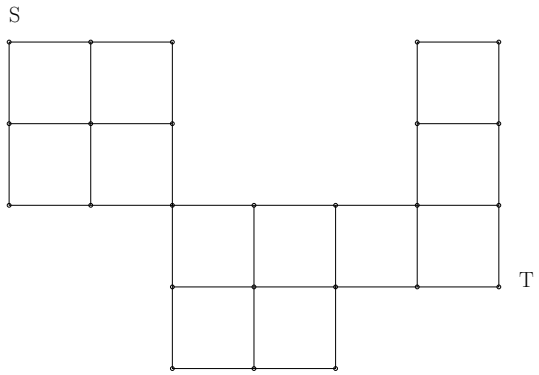


Figure 7: vertex, edge connectivity를 구하기 위한 sample graph

문제 8.8.1 아래의 *sample*그래프에서 제시된 *5*가지 *property*를 *LEDA*로 프로그래밍을 하여 채우기 바랍니다. 본인 이 그래프를 조금씩 변형하여 *(edge* 혹은 *vertex*를 추가하거나 제거*)*

```
/* Sample Graph를 돌린 결과
   Graph Name     |V|     |E|     k(G)     k'(G)      flow(1,n)
   ------------------------------------------------------------
   Graph1
   Graph2
   Graph3
   Graph4
   Graph5
   ------------------------------------------------------------
```

문제 8.8.2 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 8.8.3 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 8.8.4 여러분은 앞서 제시된 5개의 *sample graph*에서 *vertex* v_1 과 $v_{n=|V(G)|}$ 사이의 *maximal network flow*값을 구하시요. 방법은 위의 문제와 동일하다. 기말고사에서는 이와 같은 값을 계산하는 프로그래밍 시험을 칠 예정이므로 자신의 힘으로 잘 익혀두어야 한다. □