Data Structure

(군 도 뱃 길 구 축 사 업)

제출일	2013.12.14
이 름	박 준 영
학 번	201020268

1. Algorithm

2. Program Structure

2.1 Data Structure

2.2 Structure Chart

3. Program Complexity

3.1 Space Complexity

3.2 Time Complexity

4. Reveiw

1. Algorithm

- 본 과제에서는 “군도 뱃길 구축 사업”이다. 이는 최초에 여러 개의 섬의 정보를 입력받고, 이를 이용해서 모든 섬이 연결되어 있는지와 Articulation point등의 정보를 찾아내는 프로그램이다. 섬들의 정보는 text파일로 입력을 받고, 구현 결과도 text파일로 출력한다.

1. **파일명 입력**

   먼저 파일명을 입력받는다. 섬의 정보가 있는 파일(Load)의 이름을 입력받고 입력받은 문자열이 ‘Q’ 또는 ‘q’가 아니라면 결과를 출력할 파일(Save)의 이름을 입력받는다. ‘Q’ 또는 ‘q’가 아니라면 이후 Program 과정으로 진행되고, ‘Q’ 또는 ‘q’이라면 프로그램이 종료된다. Figure 1은 본 과정의 Flow Chart이다.

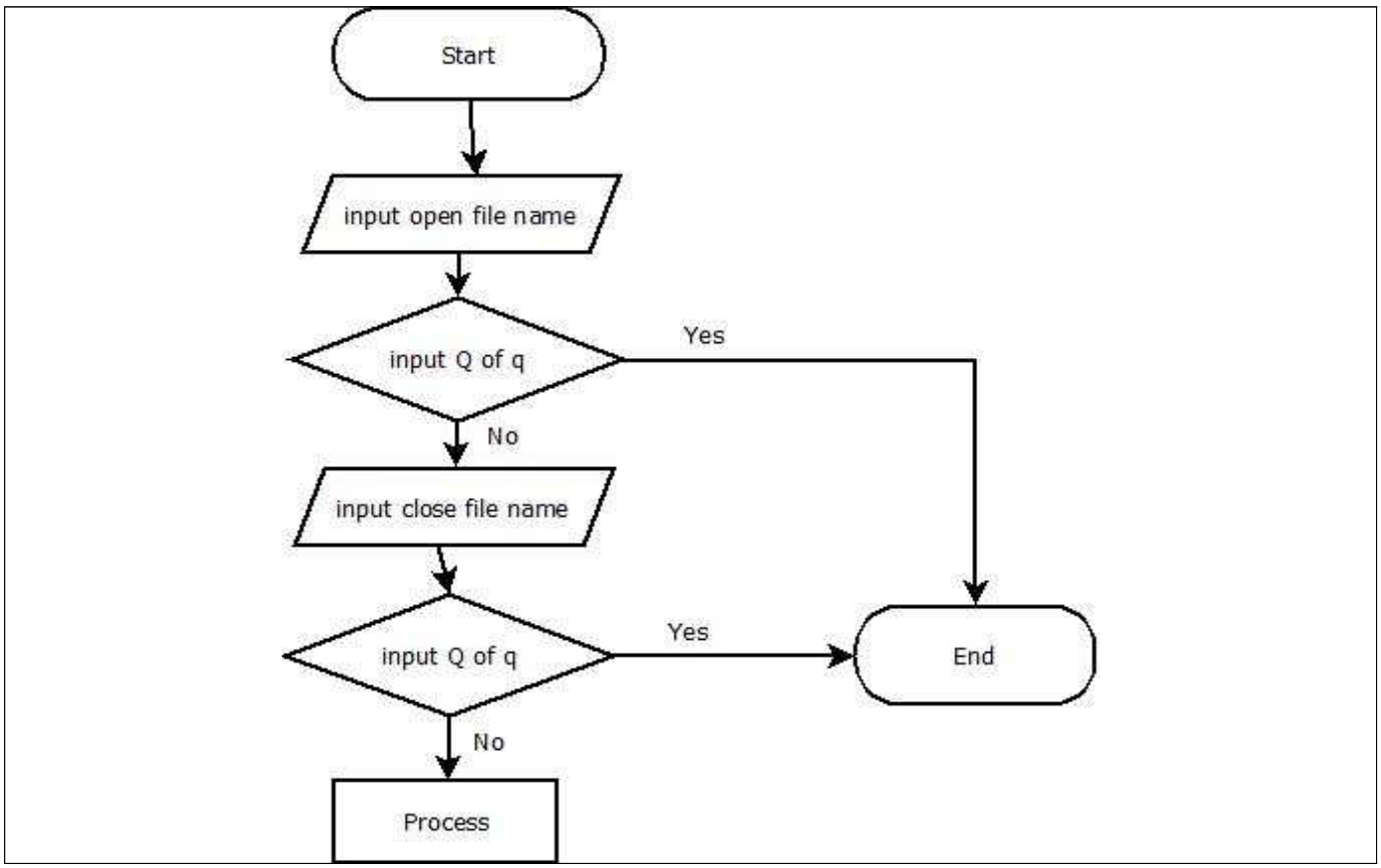

Figure 1. input file name

2. **파일 입력**

   입력받은 파일명을 갖고 있는 text 파일을 읽어온다. 입력 파일내의 구조는 첫 번째 줄, 두 번째 줄, 그 이후의 줄 각각의 불러오는 정보가 다르다. 첫 번째 줄은 섬의 총 개수를 나타내고, 두 번째 줄은 섬들의 이름들이 나열되어 있다. 세 번째 줄부터는 섬 사이에 뱃길이 존재하는 지에 대한 정보가 나열되어있다. 이를 차례대로 입력받는다. Figure 2는 그 과정을 나타낸다.

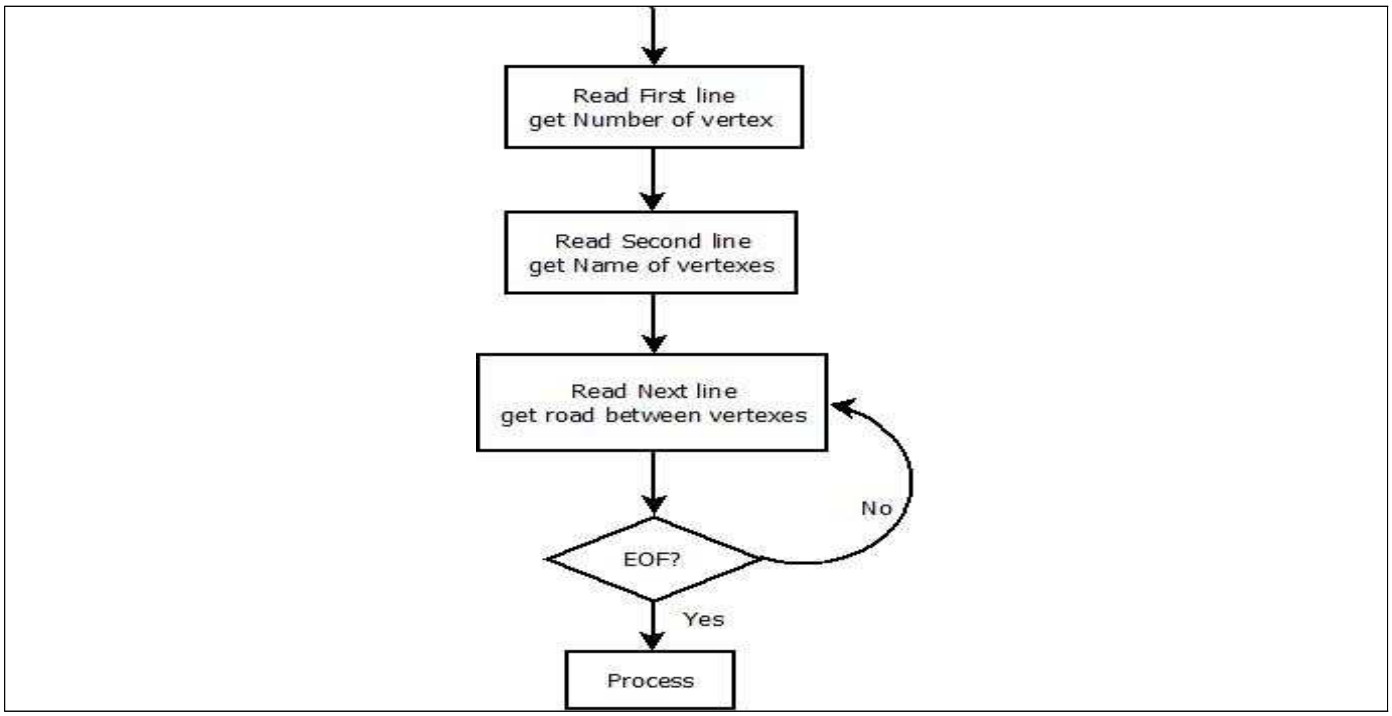

Figure 2. Loading file

3. Setting

파일에서 읽어온 정보들을 앞으로 진행될 과정에 알맞게 저장한다. 2. 파일 입력에서 첫 번째 line에서 읽어온 vertex의 개수를 Numofvertex라는 int형 변수에 저장한다. 두 번째 line에서 읽어온 vertex name들은 차례대로 vertex[MAX].name이라는 char * 형 변수에 저장한다. 이 때 입력된 순서는 앞으로 해당 vertex의 index로 사용된다. 3 번째 line부터는 vertex사이의 경로의 유무이다. 이는 int형의 2차원 배열(Adjancy matrix)에 저장된다.

4. void init (int count)

파일에서 읽어온 정보들로 프로그램 내에 변수에 저장되었다. 이 후에는 ini라는 함수를 이용해 graph배열(구조체로 선언된 형식의 pointer형)의 연결 구조를 Setting한다. 이 때 graph배열 각각의 포인터는 해당 index(위에서 언급한 해당 vertex의 index)와 연결된 vertex이 linked 되게 setting한다. 사용되는 Data는 앞서 Setting한 matrix 2차원 배열이다.

출처 : 네이버 블로그

5. void connected(int count)

본 함수는 depth first search를 통해 주어진 graph가 connected되어 있는지 판단하는 함수이다. connected되어 있다면 Connect라는 변수가 TRUE로 될 것이고, 그렇지 않다면 FALSE로 될 것이다. 또한 추가로 unconnected하다면 ConnectedList배열에 connected component의 임의의 vertex index가 차례로 하나씩 저장될 것이다.

6. void dfs(int v)

본 함수는 교재에서도 설명되어 있고, 강의 시간에도 배웠던 Algorithm이다. 이는 시작 vertex v부터 연결되어 있는 vertex를 따라간다. 이 때 각 vertex에 도달할 때마다 mark라는 1차원 배열의 해당 index에 방문여부를 표시한다. 방문하지 않은 vertex가 없다면 함수가 return된다. 본 함수는 reculsive한 특성을 이용하였다.

7. **Connected = TRUE**
   구현된 graph가 connected하다면 graph는 1개가 된다. 하나의 graph에서 articulation point를 찾고, 본 graph가 biconnected 하게 만드는 edge를 추가한다.

8. **Connected = FALSE**
   unconnected하다는 말은 곧 1개 이상의 graph가 존재한다는 것이다. 이때 각각의 graph에서 articulation point를 찾고, 1개 이상의 graph사이의 edge를 추가하여 connected하게 만든다.

9. **void bicon(int u, int v)**
   본 함수는 low 배열, dfn 배열을 setting한다. dfn은 dfs process를 시작한 u vertex부터 방문된 순서를 저장한다. low는 u vertex의 descendant vertex들 중의 최소값이 저장된다. 본 함수는 reculsive하게 진행되고, 각 함수가 호출될 때마다, dfn[v] 값이 low[u]의 값보다 작으면 v는 graph의 articulation point가 된다.

10. **insert edge**
    마지막으로 환경에 맞게 edge를 추가하는 기능이다. 7. Connected = TRUE의 경우는 articulation point를 제외한 각각의 biconnected component의 임의의 vertex끼리의 edge가 추가 되게 된다. 이를 찾는 Algorithm은 low배열을 이용하는 것이다. low배열은 biconnected component마다 low값이 일정하다. 이를 이용하여 추가할 edge를 찾는다. 먼저 low배열에서 articulation point에 해당되는 값을 구현과정에서 영향을 미치지 못하게 -1로 변경한다. 남은 배열의 값 중 같은 값의 low값중 하나를 선택해, index를 lowList라는 배열에 차례대로 추가한다. 그 후에 이를 이용하여 edge를 추가하면 이는 biconnected graph가 된다.
    8. Connected = FALSE의 경우는 unconnected graph들 사이에 edge를 추가해주면 connected graph가 되는데, 이는 앞(5. void connected(int count))에서 언급했던 ConnectedList를 이용한다. ConnectedList는 connected component의 임의의 vertex가 차례대로 저장되어 있기 때문에 이들 사이에 edge를 추가한다면, 이 graph는 connected가 된다.

2.Program structure

2.1 Data structure

- 본 과제에서는 graph라는 Data structure를 구현한다. 파일에서 읽어온 vertex의 이름을 vertex 구조체의 name 변수에 저장한다. 저장된 vertex 구조체를 기반으로 adjancy matrix를 구현한다. matrix를 구현할 때, 첫 줄에서 읽어온 N(vertex의 총 개수)을 이용하여 N × N 의 영역을 0으로 초기화한다. 그 이외의 영역은 불필요한 영역임을 명시하기 위해서 -1로 모두 초기화한다. Figure 3은 해당 구조를 나타낸다.

이렇게 구성된 matrix로 graph pointer 배열을 구성한다. 이 때 graph의 I번째 배열은 I번째 vertex가 연결되어 있는 vertex의 index를 가리키고 있다. 이렇게 프로그램 구현에 필요한 data가 구축된다.

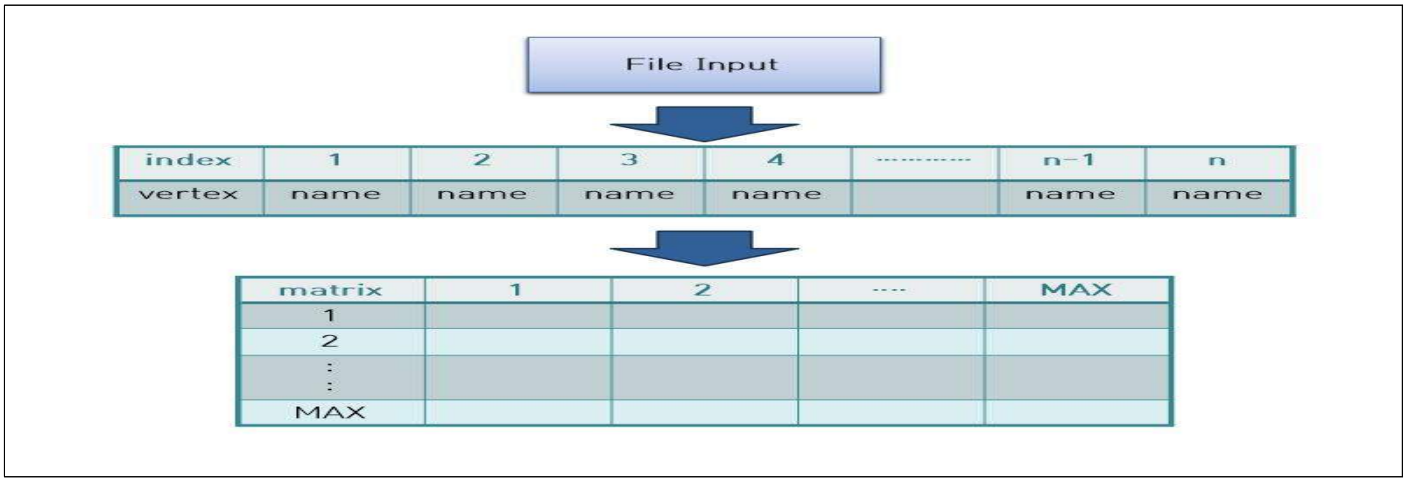

Figure 3. Data structure of input process

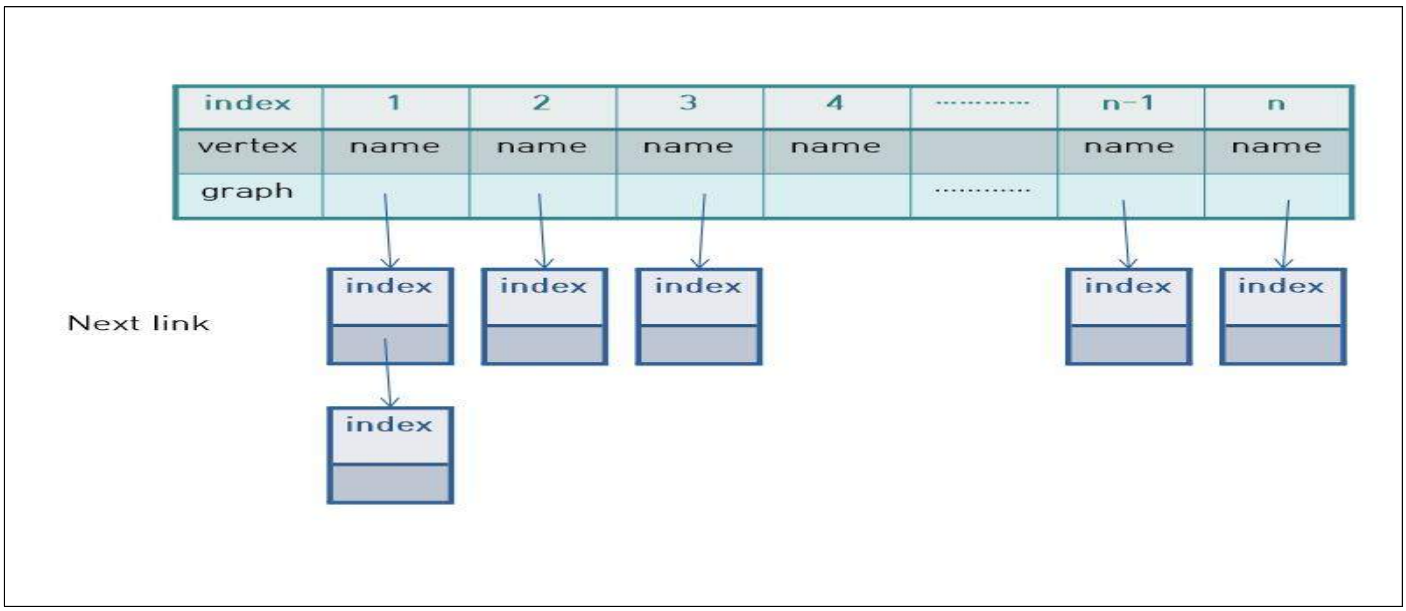

Figure 4. Data structure of graph array

이외에도 위에서 언급한 low 배열과 dfn배열, Articulation point가 차례대로 저장되는 Articulation배열, Connected component의 vertex 하나씩이 저장되는 ConnectedList 배열, 같은 low값을 이용해 biconnected component를 구성하기 위한 LowList 배열이 있다.

2.2 Structure chart

- 본 프로그램은 총 2가지 경우로 프로그램이 구분된다. 주어진 Data의 graph가 connect인 경우와 그렇지 않은 경우로 구현되는데, 이는 각 상황마다 구현 기능이 다르기 때문이다. Figure 5는 전체 프로그램의 구조이다. 최초에 Data를 불러오는 파일명, 결과를 출력할 파일명을 입력받고 이 파일명에 Q 혹은 q가 입력될 때까지 반복 시행된다.

먼저 Graph가 Connected인지 구분하고 결과에 따라 두 가지 방향으로 진행된다. Connected일 경우에는 graph의 articulation point를 찾고, 그래프가 biconnected한 성질을 갖도록 edge를 추가한다. 그렇지 않은 경우에는 graph의 각 component의 articulation point를 찾고, component사이에 edge를 추가하여 connected한 성질을 만족하도록 한다. 이를 text파일에 출력하고 다시 파일명을 입력받는다.

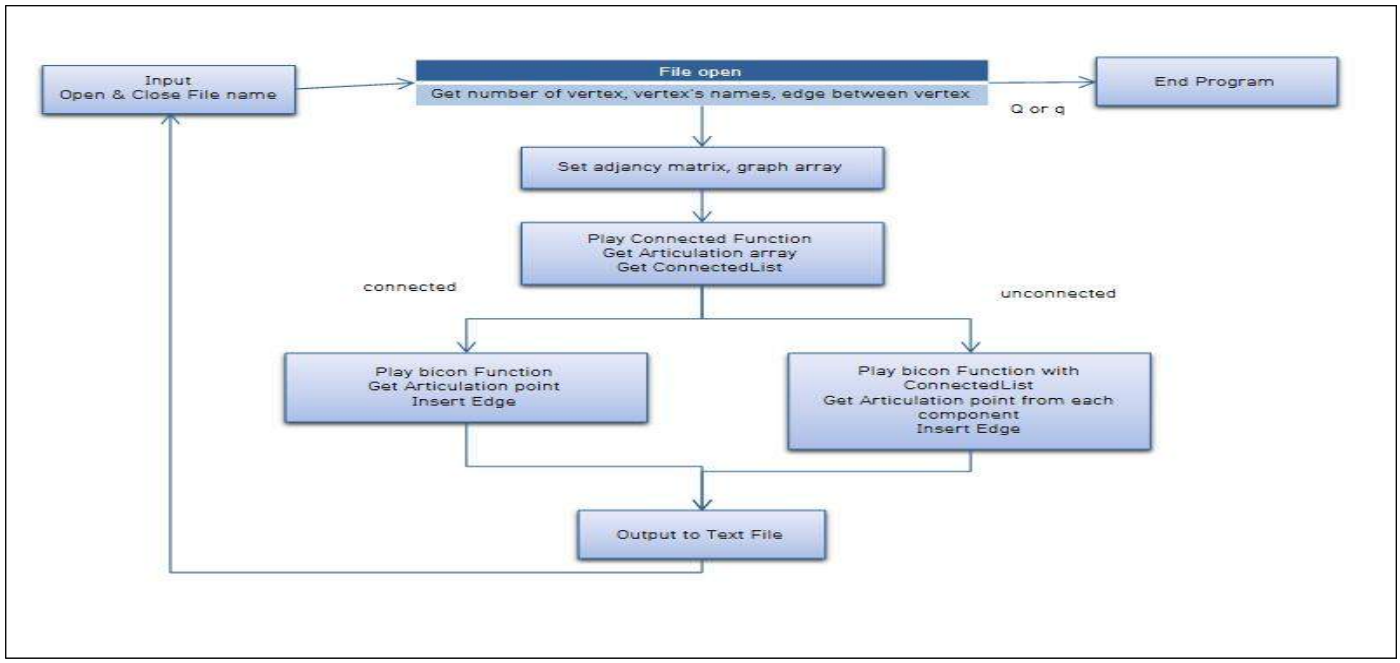

Figure 5. Structure Chart of program

3. Program Complexity Analysis

3.1 Space Complexity

function	Equation	Asymtotic Notation		
		Big-Oh	Big-Omega	Big-Theta
void bicon(int u, int v)	∞	$O(\infty)$	$\Omega(\infty)$	$\theta(\infty)$
int findVertex(char* name)	n	$O(n^2)$	$\Omega(n)$	$\theta(n^2)$
void Initialize()	n^2	$O(n^3)$	$\Omega(n^2)$	$\theta(n^3)$
void init(int count)	n^2	$O(n^3)$	$\Omega(n^2)$	$\theta(n^3)$
void connected(int count)	∞	$O(\infty)$	$\Omega(\infty)$	$\theta(\infty)$
void dfs(int v)	∞	$O(\infty)$	$\Omega(\infty)$	$\theta(\infty)$

table 1. Space compelxity of function

3.2 Time Complexity

function	Asymtotic Notation		
	Big-Oh	Big-Omega	Big-Theta
void bicon(int u, int v)	$O(n)$	$\Omega(1)$	$\theta(1)$
int findVertex(char* name)	$O(n)$	$\Omega(1)$	$\theta(1)$
void Initialize()	$O(n)$	$\Omega(1)$	$\theta(1)$
void init(int count)	$O(n)$	$\Omega(1)$	$\theta(1)$
void connected(int count)	$O(n)$	$\Omega(1)$	$\theta(1)$
void dfs(int v)	$O(n)$	$\Omega(1)$	$\theta(1)$

table 2. Time compelxity of function

4. Reveiw

- 본 과제를 수행하면서, graph의 구조를 다시 한 번 이해했다. graph의 여러 가지 특성인 Connected graph, connected component, Articulation point, Biconnected graph, Biconnected component등의 성질을 다시 한 번 정리하고 이를 본 과제에 적용함으로써, 개념을 이해했다. 그러나 low와 dfn의 부분에 대한 이해는 더디게 진행되었다. low와 dfn의 성질을 웹 사이트의 블로그, 강의노트, 교재등을 여러 번 읽고 이해를 하게 되니, Articulation point를 bicon 함수를 이용하여 알아낼 수 있게 되었고, edge 추가하는 기능까지 구현하게 되었다. 위에서 언급한 개념들이 각각 독립된 개념이 아니라, 서로 연계되어 있음을 알 수 있었다.