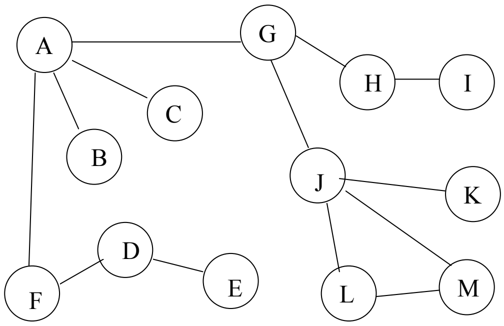
ALGORITHM

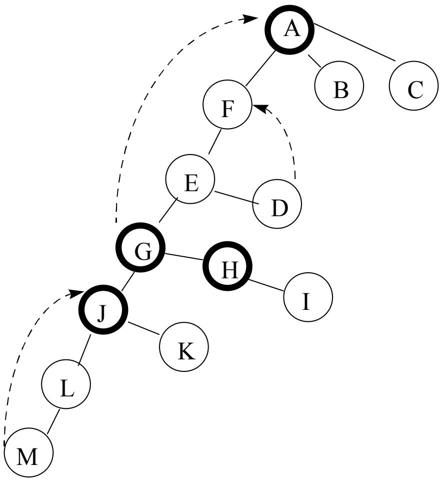
컴퓨터 공학과 김진홍 (964524)

“Search Articulation Point”

Articulation Point를 찾기 위해 DFS를 기반으로 하였다.
우선 내가 실험용으로 사용한 그래프는 아래와 같다.



위의 그래프에서 루트를 A로 잡고 DFS를 적용해 다시 그리면...



이렇다. 위에서 점선은 Back Edge이고 진하게 그려진 Circle은 Articulation Point를 의미한다.

생각보다 프로그램이 길어졌다. 출력은 다음과 같다.

```
A is an articulation point.
G is an articulation point.
J is an articulation point.
H is an articulation point.
```

소스는 아래와 같다.

```
#include <stdio.h>
#include <alloc.h>

#define  MAX    26
#define MAX_BE          50
#define TRUE    1
#define FALSE   0

struct queue {
        char vtx;               /* vertex */
        struct queue *link;
        int id;                 /* sequence number */
} *adj[MAX];

int VISIT[MAX];
int havechild[MAX];
int idcnt=0;
int elim_cnt=0;
int ourgrp=0;
int seq_cnt=1;                  /* zero is reserved for ROOT */

char SL[MAX];                   /* Sequence List */

struct {
  int cnt;
  char be[MAX_BE];              /* back edge */
} BEL[MAX];                     /* Back Edge List */

int a2i(char);                  /* convert ascii to integer number */
void addq(char, char);
void initvisit();
void initq();
void initlist();
```

```
int findseq(char);              /* find sequence number. internal use. */
void dfs_visit(char);           /* Depth First Search */
void inourgrp(char, char);      /* if a vertex is in the group, return true */
void count_elim(char);          /* count eliment in the group */
int searchanc(char);            /* if relative root has any ancestor, return true */
void isAP(char);
void checkAP();


void main()
{
    initq();

    addq('A','F'); addq('A','B'); addq('A','C'); addq('A','G');
    addq('B','A');
    addq('C','A');
    addq('D','F'); addq('D','E');
    addq('E','D'); addq('E','F'); addq('E','G');
    addq('F','E'); addq('F','D'); addq('F','A');
    addq('G','A'); addq('G','E'); addq('G','J'); addq('G','H');
    addq('H','G'); addq('H','I');
    addq('I','H');
    addq('J','G'); addq('J','L'); addq('J','M'); addq('J','K');
    addq('K','J');
    addq('L','J'); addq('L','M');
    addq('M','J'); addq('M','L');

    initlist();
    VISIT[0]=TRUE;
    SL[0]='A';          /* ROOT */
    dfs_visit(SL[0]);

    checkAP();
}

int a2i(char c)
{
  return c-'A';
}

void addq(char c, char vtx)
{
  struct queue *tmp, *ptr;
  int n;

  tmp=(struct queue *)malloc(sizeof(struct queue));
  n=a2i(c);

  tmp->vtx=vtx;
  tmp->link=NULL;
  tmp->id=0;

  if (adj[n]==NULL) adj[n]=tmp;
  else {
          for (ptr->link=adj[n]; ptr->link != NULL; ptr=ptr->link);
          ptr->link=tmp;
        }
}

void initvisit()
{
  int t;

  for (t=0; t<MAX; t++) VISIT[t]=FALSE;
}

void initq()
{
  int t;

  for (t=0; t<MAX; t++) havechild[t]=TRUE;

  initvisit();
}

int findseq(char c)
{
  int start=0;

  while (SL[start] != c) start++;

  return start;
}

void dfs_visit(char root)
{
  struct queue *ptr;

  for (ptr=adj[a2i(root)]; ptr; ptr=ptr->link)
      if (VISIT[a2i(ptr->vtx)] == TRUE) {
          BEL[a2i(root)].be[BEL[a2i(root)].cnt]=ptr->vtx;
          BEL[a2i(root)].cnt++;
      } else {
        SL[seq_cnt]=ptr->vtx;
        seq_cnt++;

        VISIT[a2i(ptr->vtx)]=TRUE;
        adj[a2i(ptr->vtx)]->id=idcnt++;  /* set sequence number */
        havechild [a2i(ptr->vtx)]=(adj[a2i(ptr->vtx)]->link) ? (1):(0);

        dfs_visit(ptr->vtx);
      }
}

void initlist()
{
  int t, i;

  for (t=0; t<MAX; t++) {
    BEL[t].cnt=0;
    for (i=0; i<MAX_BE; i++) BEL[t].be[i]='|';
  }

  for (t=0; t<MAX; t++) SL[t]=0;
}

void inourgrp(char grp, char kid)
{
  struct queue *ptr;
  int t, i;

  if (grp == kid) ourgrp=1;

  for (ptr=adj[a2i(grp)]; ptr; ptr=ptr->link)
      if (VISIT[a2i(ptr->vtx)] == FALSE)
        if (adj[a2i(ptr->vtx)]->id > adj[a2i(grp)]->id)
          inourgrp(ptr->vtx, kid);
}

void count_elim(char grp)
{
  struct queue *ptr;
  int t, i;

  elim_cnt++;

  for (ptr=adj[a2i(grp)]; ptr; ptr=ptr->link)
```

```
      if (VISIT[a2i(ptr->vtx)] == FALSE)
        if (adj[a2i(ptr->vtx)]->id > adj[a2i(grp)]->id)
          count_elim(ptr->vtx);
}

int searchanc(char root)
{
  int t, i;
  int bool=0;

  for (t=findseq(root)+1; t<seq_cnt; t++) {
    initvisit();
    ourgrp=0;
    inourgrp(root, SL[t]);

    if (!ourgrp) continue;

    for (i=0; i<BEL[t].cnt; i++) {
      if (BEL[a2i(SL[t])].be[i] == '|') break;

      if (findseq(root) > findseq(BEL[a2i(SL[t])].be[i])) bool=1;
    }
  }

  return bool;
}

void isAP(char root)
{
  if (!searchanc(root) && havechild[a2i(root)]) {
    initvisit();
    elim_cnt=0;
    count_elim(root);
    if (elim_cnt > 1) printf("%c is an articulation point.\n", root);
  }
}

void checkAP()
{
  int t;

  for (t=0; t<seq_cnt; t++) isAP(SL[t]);
}
```