자료구조 과제 12

* –

20151523

김동현

- 실행결과

텍스트이(가) 표시된 사진

자동 생성된 설명

- 코드 및 알고리즘

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 30

#define MIN2(x,y) ((x) < (y) ? (x) : (y))

short int dfn[MAX\_VERTICES];

short int low[MAX\_VERTICES];

int num;

void init();

void bicon(int, int);

void add(int, int);

void delete(int\*, int\*);

typedef struct node \*node\_pointer;

typedef struct node {

int vertex;

node\_pointer link;

}node;

node\_pointer createNode(int);

node\_pointer graph[MAX\_VERTICES];

: 그래프의 adjacency link type

typedef struct Node {

int u;

int v;

struct Node\* next;

}Node;

Node\* top;

int n;

void main()

{

FILE\* fp = fopen("input.txt", "r");

node\_pointer tail;

int i, j, k, index, count = 0;

int input[30];

char c;

fscanf(fp, "%d", &n);

while (!feof(fp)) {

for (i = 0; i < n; i++) {

fscanf(fp, "%d", &index);

for (j = 0;; j++) {

node\_pointer temp;

fscanf\_s(fp, "%d", &input[j]);

temp = createNode(input[j]);

if (j == 0) {

graph[i] = temp;

tail = temp;

}

else {

for (k = 0; k < j; k++) {

tail->link = temp;

}

tail = temp;

}

c = fgetc(fp);

if (c == '\n') break;

if (feof(fp)) break;

}

tail = NULL;

}

}

: input.txt 읽어서 변수 저장하기

init();

bicon(3, -1);

}

node\_pointer createNode(int n) {

node\_pointer newNode = (node\_pointer)malloc(sizeof(node));

newNode->vertex = n;

newNode->link = NULL;

return newNode;

}

: 그래프를 표현하는 링크드 리스트의 새로운 노드를 생성하는 함수

void add(int a, int b) {

Node\* temp;

temp = (Node\*)malloc(sizeof(Node));

temp->u = a;

temp->v = b;

temp->next = top;

top = temp;

}

: edge 스택에 저장되는 edge stack linked list의 push 역할

void delete(int\* a, int\* b) {

Node\* temp;

temp = top;

\*a = temp->u;

\*b = temp->v;

top = top->next;

//temp->next = NULL;

free(temp);

}

: edge 스택에 저장되는 edge stack linked list의 pop 역할

void init(void) {

int i;

for (i = 0; i < n; i++) {

dfn[i] = low[i] = -1;

}

num = 0;

}

: 시작 전, dfn과 low의 값을 -1로 초기화

void bicon(int child, int parent) {

node\_pointer ptr;

int w, x, y;

dfn[child] = low[child] = num++;

: dfs의 순으로 처음에 dfn을 정해준다. Min 비교 전이므로 low도 dfn으로 해준다.

for (ptr = graph[child]; ptr; ptr = ptr->link) {

:child와 연결된 edge들을 차례로 둘러본다.

w = ptr->vertex;

if (parent != w && dfn[w] < dfn[child])

add(child, w);

: 스택에 엣지 push

if (dfn[w] < 0) {

: 방문 여부 검사

bicon(w, child);

: 후손 노드들을 호출해서 필요한 low값을 recursion을 통해 구해온다.

low[child] = MIN2(low[child], low[w]);

if (low[w] >= dfn[child]) {

printf("New biconnected component: ");

do {

delete(&x, &y);

printf("<%d, %d>", x, y);

} while (!((x == child) && (y == w)));

: biconnected component 발견. 해당 엣지들을 스택에서 pop하고 출력해준다.

printf("\n");

}

}

else if (w != parent) low[child] = MIN2(low[child], dfn[w]);

: w가 parent가 아니면 백 엣지가 존재하므로 이를 확인하고 더 작은 값을 low에 저장.

}

}