오픈랩 과제 #7 : Biconnected Components

2020. 12. 04 (금) / 20181202 김수미

1. 코드 실행 결과

2. 코드 및 알고리즘 설명

이번 과제에서는 adjacency list로 표현된 그래프에서 biconnected components(이중결합 요소)를 구하는 코드를 작성해 보았다. 이중결합 요소란, articulation point(단절점)을 갖지 않는 연결그래프를 의미한다. Undirected 한 connected graph에서 이중결합 요소들은 그래프의 depth first spanning tree 를 이용해서 구할 수 있다. depth first spanning tree의 루트가 두개 이상의 children을 가지지 않으면 해당 루트노드는 단절점이다. 또한 특정 vertex의 child를 w라고 할 때, w와 w의 decendent, 또는 w에서의 하나의 back edge만을 포함하는 path를 이용해 vertex의 ancestor에 도달할 수 없다면, 이러한 child w가 해당 vertex에 하나라도 존재한다면 그 vertex 역시 단절점이다. biconnected components는 단절점을 기준으로 나누어지기 때문에 그래프의 단절점을 찾으면 biconnected components 역시 구할 수 있다.

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 30
#define MIN2(x,y) ((x) < (y) ? (x) : (y))
#define FALSE 0
#define TRUE 1

// variables, arrays and functions
short int dfn[MAX_VERTICES];
short int low[MAX_VERTICES];
short int visited[MAX_VERTICES];
int num;
void init(void);
void bicon(int, int);
```

먼저 코드 작성에 필요한 헤더파일들을 불러오고 상수와 변수, 배열, 함수들을 선언해준다.

```
16  // node for adjacency list
17  typedef struct node *node_pointer;
18  typedef struct node {
19      int vertex;
20      node_pointer link;
21  };
22  node_pointer graph[MAX_VERTICES];
23  node_pointer createnode(int data);
24
25  // node for singly linked list stack
26  typedef struct edge {
27      int data_a;
28      int data_b;
29      struct edge *next;
30  } edge;
31  edge *top = NULL;
```

그 다음 adjacency list를 구성하기 위한 노드 구조체를 선언해준다. 노드는 안에 들어갈 숫자인 vertex와 연결되어있는 다음 노드를 가리키는 link로 구성된다.

그리고 depth first spanning tree를 만들기 위해서는 DFS를 수행해야 하기 때문에, stack구조가 필요하다. 이 스택 구조를 singly linked list로 구현할 것이므로 이에 필요한 노드 역시 구조체로 선언해준다. 노드와 노드 간의 연결을 저장하는 스택이므로 edge라는 이름을 붙여 주었다.

```
33  // stack push operation
34  void push(int x, int y) {
35      struct edge *p;
36      p = (struct edge *)malloc(sizeof(struct edge));
37      p->data_a = x;
38      p->data_b = y;
39      if (top == NULL) { top = p; p->next = NULL; }
40      else { p->next = top; top = p; }
41  }
42
43  // stack pop operation
44  void pop(int *x, int *y) {
45      struct edge *p;
46      if (top == NULL) printf("stack underflow₩n");
47      else {
48          p = top;
49          top = top->next;
50          *x = p->data_a;
51          *y = p->data_b;
52          free(p);
53          p = NULL; }
54  }
```

스택의 push와 pop 함수이다.

노드와 노드간의 edge정보를 저장하기 때문에 연결된 두 노드의 숫자를 매번 push하거나 pop한다.

```
56  // create new node for adjacency list
57  node_pointer createnode(int data) {
58      node_pointer ptr;
59      ptr = (node_pointer)malloc(sizeof(struct node));
60      ptr->vertex = data;
61      ptr->link = NULL;
62      return ptr;
63  }
```

adjacency list를 만들 때 사용되는 함수이다. 하나의 노드 뒤에 새로운 노드를 만들어준다.

```
65  // create new biconnected component
66  void bicon(int u, int v) {
67      node_pointer ptr;
68      int w, x, y;
69      dfn[u] = low[u] = num++;
70
71      for (ptr = graph[u]; ptr; ptr = ptr->link) {
72          w = ptr->vertex;
73          if (v != w && dfn[w] < dfn[u]) {
74              push(u, w);
75              if (dfn[w] < 0) {
76                  bicon(w, u);
77                  low[u] = MIN2(low[u], low[w]);
78                  if (low[w] >= dfn[u]) {
79                      printf("New biconnected component: ");
80                      do {
81                          pop(&x, &y);
82                          printf("<%d,%d> ", x, y);
83                      } while (!((x == u) && (y == w)));
84                      printf("Wn");
85                  }
86              }
87              else if (w != v) low[u] = MIN2(low[u], dfn[w]);
88          }
89      }
90  }
```

biconnected component를 생성해주는 함수이다. dfn과 low를 결정해주는 부분, 그리고 그래프를 biconnected component로 분할해주는 부분으로 구성되어 있다. edge를 처음 만날 때 마다 스택에 저장해두면 biconnected component의 모든 edge들을 출력할 수 있다. x를 spanning tree의 루트라고 할 때, 함수는 bicon(x, -1)의 형태로 호출하면 된다.

```
92  // initialize dfn and low
93  void init(void) {
94      int i;
95      for (i = 0; i < MAX_VERTICES; i++) {
96          visited[i] = FALSE;
97          dfn[i] = low[i] = -1;
98      }
99      num = 0;
100 }
```

dfn과 low를 초기화 해주는 함수이다. bicon함수를 통해 biconnected component를 구한 다음 다른 작업을 수행하기 위해선 dfn과 low를 초기화 해 줄 필요가 있다.

```
102 // main function
103 void main(void) {
104     int total_vertex, vertex_num, n;
105     int flag = 0; char trash;
106     node_pointer *vertex;
107
108     // read input.txt file
109     FILE* fp;
110     fp = fopen("input.txt", "r");
111     if (fp == NULL) {
112         fprintf(stderr, "file open error ₩n");
113         exit(1); }
114     fscanf(fp, "%d", &total_vertex);
115     fscanf(fp, "%c", &trash);
116
117
118     // make adjacency list
119     for (int i = 0; !feof(fp); i++) {
120         fscanf(fp, "%d", &n);
121         if (flag == 0) {
122             vertex_num = n;
123             vertex = &graph[vertex_num];
124             flag++; }
125         else {
126             *vertex = createnode(n);
127             vertex = &((*vertex)->link); }
128
129         fscanf(fp, "%c", &trash);
130         if (trash == '₩n') flag = 0;
131     }
132
133     // create biconnected components
134     init();
135     bicon(3, -1);
136
137     system("pause");
138 }
```

메인 함수이다. input.txt 파일을 불러와서 해당 파일의 정보를 가지고 그래프를 구성하고(adjacency list), 구성된 그래프에서 biconnected components를 구하는 부분으로 구성되어 있다.