```cpp
// 執行程式: Ctrl + F5 或 [偵錯] > [啟動但不偵錯] 功能表
// 偵錯程式: F5 或 [偵錯] > [啟動偵錯] 功能表
// 開始使用的提示:
//   1. 使用 [方案總管] 視窗，新增/管理檔案
//   2. 使用 [Team Explorer] 視窗，連線到原始檔控制
//   3. 使用 [輸出] 視窗，參閱組建輸出與其他訊息
//   4. 使用 [錯誤清單] 視窗，檢視錯誤
//   5. 前往 [專案] > [新增項目]，建立新的程式碼檔案，或是前往 [專案] > [新增現有項目]，將現有程式碼檔案新增至專案
//   6. 之後要再次開啟此專案時，請前往 [檔案] > [開啟] > [專案]，然後選取 .sln 檔案

// HWK1.cpp : 此檔案包含 'main' 函式。程式會於該處開始執行及結束執行。
//作者:YUDA LU
//日期:2020.03.20(1.0), 2020.03.24(1.1), 2020.04.09(1.2), 2020.04.10(2.0)

//目前 >> back_edge 錯誤運算 (low錯誤)

#include <iostream>
#include <fstream>
#include <windows.h>
#include <dos.h>
#include <conio.h>
#include <typeinfo>
#define SIZE 20
using namespace std;

class point
{
private:
    //number  >>  表格數字
    //number_line[20]  >>  數字連接到的數字
    //line_n  >>  數字連接到總數

    //老師提點 >> 可以在分支點上座分支紀錄 >> 帶入遞迴函數座判斷

    int number, number_line[SIZE], line_n,
        dfn = -1, back_edge, low, input_number;
    bool bool_number_line[SIZE] = { false },
        bool_number_end = false;

public:
    //導入檔案
    void in_number(int n);
    void in_number_line(int i, int n);
    void in_line_n(int n);
    void in_bool_number_line(int i, bool op);
    void in_dfn(int i);
    void in_low(int i);
    void in_back_edge(int i);
    void in_bool_number_end(bool i);
    void in_input_number(int i);

    //導出檔案
    int out_number();
    int out_number_line(int i);
    int out_line_n();
    int out_dfn();
    int out_low();
    int out_back_edge();
    int out_bool_total();
    int out_input_number();
    bool out_bool_number_line(int i);
    bool out_bool_number_end();

    //function at class point
    int return_number_i(int n);
    bool cheak_number(int number);
    bool cheak_number_ok(int number);
};

void read_date(string* in_number_date);
void string_to_point_array(string str_number, point point_array[SIZE], int* n);

void print_number_array(int n, point point_array[SIZE]);
void print_numbber_array_line(int n, point point_array[SIZE]);
void print_bool_array(int n, point point_array[SIZE]);
void print_articulation_point_array(int articulation_point[SIZE], int articulation_point_n);
void print_dfs_end(int n, point point_array[SIZE]);
void print_all_date(int n, point point_array[SIZE]);

void insert_number_to_point_array(int n, point point_array[SIZE]);
void insert_dfs(int n, point point_array[SIZE], int dfs_array[SIZE]);
void insert_dfs2(int n, point point_array[SIZE], int dfs_array_n, int dfs_array[SIZE]);
void insert_back_edge(int n, point point_array[SIZE], int dfs_array[SIZE]);
void insert_articulation_point(int n, point point_array[SIZE], int dfs_array[SIZE], int articulation_point[SIZE], int* articulation_point_n);

int search_next_number(int search_number, point point_array[SIZE], int dfs_array_n, int dfs_array[SIZE]);
bool ok_number(int next_number, int dfs_array_n, int dfs_array[SIZE]);

//主程式
int main(void)
{
    /*
        名詞
            articulation_point >> 關節點
        變數
            n                       總共有多少數字
            dfs_array               DFS找到的順序
            in_number_date          讀入黨案彎寫入字串中
            point_array             數字陣列
            articulation_point      關節點陣列
            articulation_point_n    關節點陣列n
```

```cpp
        思考路線
            1.讀黨
            2.將檔案轉換成Point形式
            3.使用DFS找出DFN陣列
            4.使用back_edge找出low
            5.使用low & dfn 找出關節點
    */

    int n, articulation_point_n, dfs_array[SIZE], articulation_point[SIZE];
    string in_number_date;
    point point_array[SIZE];

    read_date(&in_number_date);                                     //讀黨
    string_to_point_array(in_number_date, point_array, &n);         //轉換檔案中資料

    print_number_array(n, point_array);
    print_numbber_array_line(n, point_array);

    //建置point_array中number數字
    insert_number_to_point_array(n, point_array);

    //算出dfs
    insert_dfs(n, point_array, dfs_array);
    insert_dfs2(n, point_array, n, dfs_array);

    print_bool_array(n, point_array);
    print_numbber_array_line(n, point_array);

    //用back_edge算出low
    insert_back_edge(n, point_array, dfs_array);

    //用low and dfn 算出關節點
    insert_articulation_point(n, point_array, dfs_array, articulation_point, 
      &articulation_point_n);

    //列印檔案
    print_bool_array(n, point_array);
    print_all_date(n, point_array);
    print_articulation_point_array(articulation_point, articulation_point_n);   //列 
      印關節點

    system("pause");
    return 0;
}

//class point 程式區段 開始
void point::in_number(int n)
{
    number = n;
}

void point::in_number_line(int i, int n)
{
    number_line[i] = n;
}

void point::in_line_n(int n)
{
    line_n = n;
}

void point::in_bool_number_line(int i, bool op)
{
    bool_number_line[i] = op;
}

void point::in_dfn(int i)
{
    dfn = i;
}

void point::in_low(int i)
{
    low = i;
}

void point::in_back_edge(int i)
{
    back_edge = i;
}

void point::in_bool_number_end(bool i)
{
    bool_number_end = i;
}

void point::in_input_number(int i)
{
    input_number = i;
}

int point::out_number()
{
    return number;
}

int point::out_number_line(int i)
{
    return number_line[i];
}

int point::out_line_n()
{
    return line_n;
}

```

```cpp
int point::out_dfn()
{
    return dfn;
}

int point::out_low()
{
    return low;
}

int point::out_back_edge()
{
    return back_edge;
}

int point::out_bool_total()
{
    int sum = 0;
    for (int i = 0; i < line_n; i++)
    {
        if (bool_number_line[i] == true)
        {
            ++sum;
        }
    }
    return sum;
}

int point::out_input_number()
{
    return input_number;
}

bool point::out_bool_number_line(int i)
{
    return bool_number_line[i];
}

bool point::out_bool_number_end()
{
    return bool_number_end;
}

int point::return_number_i(int n)
{
    int out = 0;
    for (int i = 0; i < line_n; i++)
    {
        if (number_line[i] == n)
        {
            out = i;
            break;
        }
    }
    return out;
}

bool point::cheak_number(int number)
{
    bool op = false;
    for (int i = 0; i < line_n; i++)
    {
        if (number_line[i] == number) op = true;
    }
    if (op) return true;
    else return false;
}

bool point::cheak_number_ok(int number)
{
    bool op = false;
    for (int i = 0; i < line_n; i++)
    {
        if (number_line[i] == number && bool_number_line[i] == 0) op = true;
    }
    if (op == true) return true;
    else return false;
}
//class point 程式區段 結束

//開檔讀檔
void read_date(string* in_number_date)
{
    fstream InF;
    int n = 0;
    char FName[20], ch;

    cout << "輸入方程式檔名:";
    cin >> FName;
    InF.open(FName, ios::in);

    if (!InF)
    {
        cout << "檔案無法開啟\n";
        exit(1);
    }
    else
    {
        while (InF.get(ch))
        {
            *in_number_date += ch;
        }
        InF.close();
    }
}

```

```
//string to point_array

void string_to_point_array(string str_number, point point_array[SIZE], int* n)
{
    int i = 0, number_i = 0, number_n = 0, number_line_n = 0;
    bool get_n = false;
    string t;

    /*
    t >> 站存n字串
    get_n >> 判讀n讀完了嗎?
    number_i >> 紀錄number_i 的 i
    number_n >> 紀錄number_i
    number_line_n >> point_number_line 長度紀錄
    */

    while (str_number[i])
    {
        if (!get_n)
        {
            //讀入n

            if (str_number[i] != '\n')
            {
                t += str_number[i];
            }
            else
            {
                *n = atoi(t.c_str());
                get_n = true;
            }
            i++;
        }
        else
        {
            if (str_number[i] != '\n')
            {
                //如果有1才紀錄至資料中

                if (str_number[i] == '1')
                {
                    point_array[number_n].in_number_line(number_line_n, number_i);

                    number_i++;
                    number_line_n++;
                }
                else if (str_number[i] == '0')
                {
                    number_i++;
                }
                else if (str_number[i] == ' ');
                else
                {
                    exit(1);
                }
            }
            else
            {
                //紀錄當行訊息並歸零

                point_array[number_n].in_line_n(number_line_n);
                number_line_n = 0;
                number_i = 0;
                number_n++;
            }
            i++;
        }
    }

//防止字串最後無換行而無紀錄最後一行訊息

    if (str_number[i - 1] != '\n')
    {
        point_array[number_n].in_line_n(number_line_n);
    }
}

void print_articulation_point_array(int articulation_point[SIZE], int
  articulation_point_n)
{
    cout << "Articulation Point(關節點): ";
    for (int i = 0; i < articulation_point_n; i++)
    {
        cout << articulation_point[i] << " ";
    }
    cout << "\n完成運算結束囉!!\n";
}

void print_dfs_end(int n, point point_array[SIZE])
{
    for (int i = 0; i < n; i++)
    {
        cout << point_array[i].out_bool_number_end() << " ";
    }
    cout << endl;
}

void print_number_array(int n, point point_array[SIZE])
{
    cout << "n = " << n << " 矩陣：" << endl;
    printf("    ");
    for (int i = 0; i < n; i++) printf("%2d ", i);
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        printf("%2d > ", i);
```

```cpp
        for (int j = 0; j < n; j++)
        {
            if (point_array[i].cheak_number(j)) cout << "1  ";
            else cout << "0  ";
        }
        cout << endl;
    }
    cout << "\n";
}

void print_all_date(int n, point point_array[SIZE])
{
    cout << "Print all date" << endl;
    for (int i = 0; i < n; i++)
    {
        printf("number = %2d, dfs_number = %2d, low = %2d, input_number = %2d\n", i, point_array[i].out_dfn(), point_array[i].out_low(), point_array[i].out_input_number());
    }
    cout << endl;
}

void print_bool_array(int n, point point_array[SIZE])
{
    cout << "bool 連接結果狀況\n";
    for (int i = 0; i < n; i++)
    {
        printf("%2d > ", i);
        for (int j = 0; j < point_array[i].out_line_n(); j++)
        {
            cout << point_array[i].out_bool_number_line(j) << " ";
        }
        cout << " \ttotal >> " << point_array[i].out_bool_total() << endl;
    }
    cout << "\n";
}

void print_numbber_array_line(int n, point point_array[SIZE])
{
    cout << "簡單矩陣：\n";
    for (int i = 0; i < n; i++)
    {
        printf("%2d > ", i);
        for (int j = 0; j < n; j++)
        {
            if (point_array[i].cheak_number(j)) cout << j << " ";
        }
        cout << endl;
    }
    cout << "\n";
}

void insert_number_to_point_array(int n, point point_array[SIZE])
```

```cpp
{
    for (int i = 0; i < n; i++)
    {
        point_array[i].in_number(i);
    }
}

bool ok_number(int next_number, int dfs_array_n, int dfs_array[SIZE])
{
    bool op = true;
    for (int i = 0; i < dfs_array_n; i++)
    {
        if (dfs_array[i] == next_number)
        {
            op = false;
            //break;
        }
    }
    return op;
}

int search_next_number(int search_number, point point_array[SIZE], int dfs_array_n, 
  int dfs_array[SIZE])
{
    int next_number;
    /*
        尋找下一個點
        確認search_number的連接有沒有下一個點
            如果有找到點，return 點
            如果沒找到點，return NULL
    */
    for (int i = 0; i < point_array[search_number].out_line_n(); i++)
    {
        next_number = point_array[search_number].out_number_line(i);

        if (point_array[search_number].out_bool_number_line(i) == false && ok_number 
          (next_number, dfs_array_n, dfs_array) == true)
        {
            int get_number = point_array[search_number].out_number_line(i);
            int k = point_array[get_number].return_number_i(search_number);

            point_array[search_number].in_bool_number_line(i, true);
            point_array[get_number].in_bool_number_line(k, true);

            return next_number;
            break;
        }
    }
    return -1;
}

void insert_dfs(int n, point point_array[SIZE], int dfs_array[SIZE])
{
    //cursor 目前收尋位置

    int search_total = 0, next_number, cursor = 0, dfs_array_n = 0;
    bool op = false, go_break = false;
    while (cursor < n && cursor >= 0)
    {
        //利用dfs_array的n判斷是否全部點已經找到
        if (search_total == n) break;
        if (op == false)
        {
            //將起始值輸入dfs_array
            point_array[cursor].in_dfn(search_total);
            point_array[cursor].in_input_number(0);

            dfs_array[dfs_array_n] = point_array[cursor].out_number();

            dfs_array_n++, search_total++, op = true;
        }
        else if (op == true)
        {
            //search_next_number >> 尋找下一個數字 沒找到回傳NULL
            next_number = search_next_number(cursor, point_array, search_total, 
              dfs_array);
            if (next_number != -1)
            {
                if (ok_number(cursor, search_total, dfs_array) == true)
                {
                    point_array[cursor].in_dfn(search_total);
                    dfs_array[search_total++] = cursor;
                }

                point_array[next_number].in_input_number(cursor);
                cursor = next_number;
                dfs_array_n = search_total;
                go_break = true;
            }
            else if (next_number == -1)
            {
                if (go_break == true)
                {
                    dfs_array[search_total++] = cursor;
                    point_array[cursor].in_bool_number_end(true);
                    go_break = false;
                }
                cursor = dfs_array[--dfs_array_n];
            }
            else
            {
                cout << "GG!!" << endl;
                exit(1);
            }
        }
    }
```

```cpp
}

void insert_dfs2(int n, point point_array[SIZE], int dfs_array_n, int dfs_array[SIZE])
{
    for (int i = 0; i < n; i++) point_array[i].in_dfn(dfs_array[i]);
}

//back_edge >> 目前 low 錯誤運算
void insert_back_edge(int n, point point_array[SIZE], int dfs_array[SIZE])
{
    /*
        hold_number 目前最小的路徑
        find_number 找到的最短路徑
        next_op     控制有沒有找到新的(find_number)最短路進
    */

    int hold_number = n - 1, find_number = n - 1;
    bool next_op = false;

    // i >> dfn
    for (int i = n - 1; i >= 0; i--)
    {
        for (int k = 0; k < i; k++)
        {
            //利用dfs_array最後一個找回來
            if (point_array[dfs_array[i]].cheak_number_ok(dfs_array[k]) == true)
            {
                int a_line_i = 0, b_line_i = 0;                                              // a b 代提連接變數 a接b
                                                                                             //跟改bool
                a_line_i = point_array[dfs_array[i]].return_number_i(dfs_array[k]);
                b_line_i = point_array[dfs_array[k]].return_number_i(dfs_array[i]);

                point_array[dfs_array[i]].in_bool_number_line(a_line_i, true);
                point_array[dfs_array[k]].in_bool_number_line(b_line_i, true);

                find_number = k;
                next_op = true;
                break;
            }
            next_op = false;
            //find_number = -1;
        }

        point_array[0].in_low(0);

        if (find_number < hold_number && next_op == true)                //當找到其他最短路徑 比上一個小就跟新並存入
        {
            hold_number = find_number;
            point_array[dfs_array[i]].in_low(hold_number);
            next_op = false;
        }
        else if (find_number >= hold_number && next_op == true)         //找到的路徑太沒有比上一短不跟新
        {
            point_array[dfs_array[i]].in_low(hold_number);
            next_op = false;
        }
        else if (i <= n - 2 && point_array[dfs_array[i + 1]].out_low() < i && next_op == false)                                    //沒找到路徑直接紀錄
        {
            find_number = i;
            point_array[dfs_array[i]].in_low(point_array[dfs_array[i + 1]].out_low());
        }
        else if(next_op == false)
        {
            find_number = i;
            point_array[dfs_array[i]].in_low(i);
        }
        else
        {
            point_array[dfs_array[i]].in_low(i);
        }


        //當回到樹根 hold_number 跟著回到最大值
        if (point_array[dfs_array[i]].out_number() == 0 || find_number == 0 && i != 0)
        {
            hold_number = n - 1, find_number = n - 1;
        }
    }

    //強制讓樹根變成0
    point_array[0].in_low(0);
}

//articulation_point >> 關節點
void insert_articulation_point(int n, point point_array[SIZE], int dfs_array[SIZE], int articulation_point[SIZE], int* articulation_point_n)
{
    int out_n = 0;

    for (int dfn = 0; dfn < n; dfn++)
    {
        if (dfn == 0)
        {
            if (point_array[dfs_array[dfn]].out_bool_total() > 2)                                    //判斷樹根
            {
                articulation_point[out_n++] = point_array[dfs_array[dfn]].out_number();
            }
        }
        else
        {
            int a = point_array[dfs_array[dfn]].out_input_number(), b = dfs_array
              [dfn];
            if (point_array[b].out_low() >= point_array[a].out_dfn() && a != 0)
            {
                articulation_point[out_n++] = point_array[a].out_number();      //利
                  用公式找出關節點
            }
        }
    }
    *articulation_point_n = out_n;
}
```