<u>Dashboard</u> / My courses / <u>Graph_Theory-HK3-0405</u> / <u>Tuần 5 - Thực hành Duyệt đồ thị và Ứng dụng</u> / <u>* Bài tập 14 - Ứng dụng liên thông mạnh</u>

 Started on
 Sunday, 15 June 2025, 11:08 PM

 State
 Finished

 Completed on
 Sunday, 15 June 2025, 11:10 PM

 Time taken
 2 mins 46 secs

 Marks
 2.00/2.00

 Grade
 10.00 out of 10.00 (100%)

Question **1**Correct Mark 1.00 out of

1.00

Come and Go (nguồn: UVA Online Judge, Problem 11838)

Trong một thành phố có **N** địa điểm được nối với nhau bằng **M** con đường 1 chiều và 2 chiều. Yêu cầu tối thiểu của một thành phố là **từ địa điểm này bạn phải có thể đi đến một địa điểm khác bất kỳ**.

Hãy viết chương trình kiểm tra xem các con đường của thành phố có thoả mãn yêu cầu tối thiểu này không.

Dữ liệu đầu vào có dạng như sau:

```
4 5
1 2 1
1 3 2
2 4 1
3 4 1
4 1 2
```

Trong ví dụ này, có 4 địa điểm và 5 con đường, mỗi con đường có dạng a b p, trong đó a, b là các địa điểm; và nếu p = 1, con đường đang xét là đường 1 chiều, ngược lại nó là đường 2 chiều.

Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím (stdin) với định dạng:

- Dòng đầu tiên chứa 2 số nguyên N và M, tương ứng là số địa điểm và số con đường.
- M dòng tiếp theo mỗi dòng chứa 3 số nguyên a, b, p. Nếu p = 1, con đường (a, b) là con đường 1 chiều, ngược lại nếu p = 2, con đường (a, b) là con đường 2 chiều.

Đầu ra (Output)

- In ra màn hình OKIE nếu các con đường của thành phố có thoả mãn yêu cầu, ngược lại in ra NO.
- Xem thêm ví dụ bên dưới.

Gợi ý

- Xây dựng đồ thị có hướng từ dữ liệu các con đường và các địa điểm
 - o Địa điểm ~ đỉnh
 - Đường 1 chiều ~ cung
 - o Đường 2 chiều ~ 2 cung
- Áp dụng giải thuật kiểm tra đồ thị có liên thông mạnh hay không.

For example:

Input		ut	Result
5	7		OKIE
1	2	1	
2	3	1	
3	1	1	
2	4	1	
3	4	1	
4	5	1	
5	3	1	
8	3 10		NO
1	2	2	
1	3	1	
2	8	1	
3	4	1	
3	5	2	
4	2	1	
4	7	1	
4	8	1	
5	7	1	
6	7	2	

Answer: (penalty regime: 10, 20, ... %)

```
8 v typedef struct {
9
        ElementType data[MAX_SIZE];
10
        int top_idx;
11
    } Stack;
12
    void init_stack(Stack* pS) {
13 🔻
14
        pS \rightarrow top_idx = -1;
15
16
    int is_empty(Stack* pS) {
17 🔻
18
        return pS->top_idx == -1;
19
20
21 v
    void push(Stack* pS, int x) {
22
        pS->top_idx++;
23
        pS->data[pS->top_idx] = x;
24
25
    void pop(Stack* pS) {
26 •
27
        pS->top_idx--;
28
29
30 •
    ElementType top(Stack* pS) {
31
        return pS->data[pS->top_idx];
32
33
34
    //-----Graph-----
    typedef struct {
35 🔻
36
        int n, m;
        int A[MAX_SIZE][MAX_SIZE];
37
38
    } Graph;
39
    void init_graph(Graph* pG, int n) {
40 •
41
        pG->n = n;
42
        pG->m = 0;
43
        for (int i = 0; i <= n; i++) {</pre>
44
45 🔻
            for (int j = 0; j <= n; j++) {</pre>
46
                pG->A[i][j] = 0;
47
48
        }
49
    }
50
51 v
    void add_edge(Graph* pG, int u, int v) {
52
        pG->A[u][v]++;
53
        // if (u != v) {
54
               pG->A[v][u]++;
55
        // }
56
57
        pG->m++;
58
59
    int adj(Graph* pG, int u, int v) {
60 •
61
        return pG->A[u][v] > 0;
62
63
    //----Strong connection-----
64
65
    int num[MAX_SIZE];
66
    int min_num[MAX_SIZE];
    int k;
67
68
    Stack S;
69
    int on_stack[MAX_SIZE];
70
   int cnt;
71
72 void SCC(Graph* pG, int u) {
73
        // 1. Đánh số và đưa vào stack
74
        num[u] = k;
75
        min_num[u] = k;
76
        k++;
77
        push(&S, u);
78
        on_stack[u] = 1;
79
80
        // 2.Duyệt các đỉnh kề
81
        for (int v = 1; v <= pG->n; v++) {
82
            if (adj(pG, u, v)) {
83
                // chưa duyệt
84
                if (num[v] < 0) {</pre>
```

```
85
                      SCC(pG, v);
 86
                      min_num[u] = min_num[v] < min_num[u] ? min_num[v] : min_num[u];</pre>
                  } else if (on_stack[v]) {
 87
                      min_num[u] = num[v] < min_num[u] ? num[v] : min_num[u];</pre>
 88
 89
                  }
 90
             }
 91
         }
 92
 93
         // 3. so sánh num và min num
 94
         if (num[u] == min_num[u]) {
 95
              cnt++;
 96
              int w;
              do {
 97
 98
                  w = top(\&S);
 99
                  pop(&S);
100
                  on_stack[w] = 0;
101
              } while (w != u);
102
103
104 v int main() {
         int n, m;
105
         scanf("%d %d", &n, &m);
106
107
         Graph G;
108
109
         init_graph(&G, n);
110
         for (int i = 0; i < m; i++) {</pre>
111
112
              int u, v, t;
              scanf("%d%d%d", &u, &v, &t);
113
114
              add_edge(&G, u, v);
115
              if (t == 2) {
                  add_edge(&G, v, u);
116
117
118
         }
119
120
         init_stack(&S);
121
         k = 1;
122
123
         // chưa duyet
         for (int i = 0; i <= n; i++) {
124
125
             num[i] = -1;
126
         }
127
         cnt = 0;
128 1
         for (int i = 1; i <= G.n; i++) {
129 1
              if (num[i]< 0) {</pre>
130
                  SCC(&G, i);
131
              }
132
         }
133
         if (cnt > 1) {
134
135
              printf("NO");
136
         }else{
              printf("OKIE");
137
138
         }
139 }
```

	Input	Expected	Got	
*	5 7 1 2 1 2 3 1 3 1 1 2 4 1 3 4 1 4 5 1	OKIE	OKIE	*
	5 3 1			

	Input	Expected	Got	
~	8 10 1 2 2 1 3 1 2 8 1 3 4 1 3 5 2 4 2 1 4 7 1 4 8 1 5 7 1 6 7 2	NO	NO	*
~	3 2 1 2 2 1 3 2	OKIE	OKIE	~
~	3 2 1 2 2 1 3 1	NO	NO	~
~	4 2 1 2 2 3 4 2	NO	NO	~

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**Correct
Mark 1.00 out of

Trust group (nguồn: UVA Online Judge, Problem 11709)

Phòng nhân sự của tổ chức *Association of Cookie Monsters* (ACM) nhận thấy rằng gần đây hiệu quả làm việc của các nhân viên có chiều hướng giảm sút. Vì thế họ đã lấy ý kiến các nhân viên và phát hiện ra nguyên nhân của vấn đề này, đó là: sự tin cậy. Một số nhân viên không tin cậy vào các nhân viên khác trong nhóm làm việc của mình. Điều này làm giảm động lực và niềm vui trong công việc của các nhân viên.

Phòng nhân sự muốn giải quyết triệt để vấn đề này nên họ quyết định tổ chức lại các nhóm làm việc sao cho ổn định. Một nhóm làm việc sẽ ổn định khi mà những người trong nhóm tin cậy lẫn nhau. Họ đã hỏi các nhân viên và biết được những người mà một nhân viên tin cậy trực tiếp. Ngoài ra, sự tin cậy có tính bắt cầu: *nếu A tin cậy B và B tin cậy C thì A cũng sẽ tin cậy C*. Lẽ dĩ nhiên, một nhân viên sẽ tự tin cậy chính bản thân mình. Tuy nhiên, cần chú ý là sự tin cậy lại không có tính đối xứng: *A tin cậy B thì không nhất thiết B phải tin cậy A*.

Phòng nhân sự muốn tổ chức thành ít nhóm nhất có thể. Hãy lập trình để giúp họ.

Giả sử các nhân viên được đánh số là 1, 2, ..., n.

Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím theo định dạng:

- Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số nhân viên và số cặp tin cậy.
- m dòng tiếp theo, mỗi dòng chứa 2 số nguyên a b, nói rằng người a tin cậy vào người b.

Đầu ra (Output)

- In ra màn hình số lượng nhóm ít nhất mà những người trong nhóm đều tin cậy lẫn nhau
- Xem thêm trong phần ví dụ

For example:

Ir	put	Result
5	7	1
1	2	
2	3	
3	1	
2	4	
3	4	
4	5	
5	3	
8	13	3
1	2	
1	3	
2	1	
2	8	
3	4	
3	5	
4	2	
4	7	
4	8	
5	3	
5	7	
6	7	
7	6	

Answer: (penalty regime: 10, 20, ... %)

```
1
       #include <stdio.h>
 2
3
       #define MAX SIZE 100
4
5
       //----- Stack-----
       typedef int ElementType;
6
7
       typedef struct {
8
9
           ElementType data[MAX_SIZE];
10
           int top_idx;
       } Stack;
11
12
13 1
       void init stack(Stack* pS) {
14
           pS->top_idx = -1;
15
       }
16
```

```
int is_empty(Stack* pS) {
17
18
            return pS->top_idx == -1;
19
20
        void push(Stack* pS, int x) {
21
22
            pS->top_idx++;
23
            pS->data[pS->top_idx] = x;
24
        }
25
26
        void pop(Stack* pS) {
27
            pS->top_idx--;
        }
28
29
        ElementType top(Stack* pS) {
30
31
            return pS->data[pS->top_idx];
32
33
34
        //----Graph-----
        typedef struct {
35
36
            int n, m;
37
            int A[MAX_SIZE][MAX_SIZE];
38
        } Graph;
39
40
        void init_graph(Graph* pG, int n) {
41
            pG->n = n;
42
            pG->m = 0;
43
44
            for (int i = 0; i <= n; i++) {
                for (int j = 0; j <= n; j++) {
45
46
                    pG \rightarrow A[i][j] = 0;
47
48
            }
49
50
51
        void add_edge(Graph* pG, int u, int v) {
            pG->A[u][v]++;
52
53
            // if (u != v) {
54
                   pG->A[v][u]++;
            // }
55
56
57
            pG->m++;
58
        }
59
        int adj(Graph* pG, int u, int v) {
60
61
            return pG->A[u][v] > 0;
62
63
64
        //----Strong connection-----
        int num[MAX SIZE];
65
66
        int min_num[MAX_SIZE];
67
        int k;
68
        Stack S;
        int on_stack[MAX_SIZE];
69
70
        int cnt;
71
72
        void SCC(Graph* pG, int u) {
73
            // 1. Đánh số và đưa vào stack
            num[u] = k;
74
75
            min_num[u] = k;
76
            k++;
77
            push(&S, u);
            on_stack[u] = 1;
78
79
80
            // 2.Duyệt các đỉnh kề
81
            for (int v = 1; v \le pG -> n; v++) {
                if (adj(pG, u, v)) {
82
83
                    // chưa duyệt
84
                    if (num[v] < 0) {</pre>
85
                        SCC(pG, v);
                        min_num[u] = min_num[v] < min_num[u] ? min_num[v] : min_num[ψ];</pre>
86
87
                    } else if (on_stack[v]) {
                        min_num[u] = num[v] < min_num[u] ? num[v] : min_num[u];</pre>
88
                    }
89
90
                }
91
92
93
            // 3. so sánh num và min num
94
            if (num[u] == min num[u]) {
```

```
95
                  cnt++;
 96
                  int w;
                  do {
 97
 98
                      w = top(\&S);
 99
                      pop(&S);
100
                      on_stack[w] = 0;
101
                  } while (w != u);
102
103
104
          int main() {
105
              int n, m;
              scanf("%d %d", &n, &m);
106
107
108
              Graph G;
109
              init_graph(&G, n);
110
111
              for (int i = 0; i < m; i++) {</pre>
                  int u, v;
scanf("%d%d", &u, &v);
112
113
114
                  add_edge(&G, u, v);
115
116
              }
117
118
              init_stack(&S);
119
              k = 1;
120
121
              // chưa duyet
              for (int i = 0; i <= n; i++) {
122 🔻
123
                  num[i] = -1;
124
              }
125
              cnt = 0;
              for (int i = 1; i <= G.n; i++) {</pre>
126
127
                  if (num[i]< 0) {</pre>
128
                      SCC(&G, i);
129
130
131
              printf("%d",cnt);
132
         }
133
```

	Input	Expected	Got	
~	5 7	1	1	~
	1 2			
	2 3			
	3 1			
	2 4			
	3 4			
	4 5			
	5 3			
~	8 13	3	3	~
	1 2			
	1 3			
	2 1			
	2 8			
	3 4			
	3 5			
	4 2			
	4 7			
	4 8			
	5 3			
	5 7			
	6 7			
	7 6			

Passed all tests! ✓

Question author's solution (C):

1 Hinglude (stdie b)

```
2
    /* Khai báo CTDL Graph*/
3
    #define MAX_N 100
5 v typedef struct {
        int n, m;
6
7
        int A[MAX_N][MAX_N];
8
    } Graph;
9
10 void init_graph(Graph *pG, int n) {
11
        pG->n = n;
12
        pG->m = 0;
13
        for (int u = 1; u <= n; u++)
            for (int v = 1; v \le n; v++)
14
15
                pG->A[u][v] = 0;
16
17
18 -
    void add_edge(Graph *pG, int u, int v) {
19
        pG->A[u][v] += 1;
20
        //if (u != v)
21
              pG \rightarrow A[v][u] += 1;
22
23
        if (pG->A[u][v] > 1)
24
            printf("da cung (%d, %d)\n", u, v);
25
        if (u == v)
26
            printf("khuyen %d\n", u);
27
28
29
        pG->m++;
    }
30
31
32 v int adjacent(Graph *pG, int u, int v) {
33
        return pG->A[u][v] > 0;
    }
34
35
36
37
    #define MAX_SIZE 100
38
    typedef int ElementType;
39 v typedef struct {
        ElementType data[MAX_SIZE];
40
41
        int top_idx;
42
    } Stack;
43
    /* Hàm khởi tạo ngăn xếp rỗng */
    void make_null_stack(Stack *pS) {
44 🔻
45
        pS \rightarrow top_idx = -1;
46
47
    /* Hàm thêm phần tử u vào đỉnh ngăn xếp */
48 void push(Stack *pS, ElementType u) {
49
        pS->top_idx++;
50
        pS->data[pS->top_idx] = u;
51
52
    /* Hàm xem phần tử trên đỉnh ngăn xếp */
53 → ElementType top(Stack *pS) {
        return pS->data[pS->top_idx];
55
    /* Hàm xoá bỏ phần tử trên đỉnh ngăn xếp */
56
57 🔻
   void pop(Stack *pS) {
58
        pS->top_idx--;
59
    /* Hàm kiểm tra ngăn xếp rỗng */
60
61 v int empty(Stack *pS) {
62
        return pS->top_idx == -1;
63
64
65
66
    int min(int a, int b) {
67
68
        return a < b ? a : b;</pre>
69
70
71
72
    int num[MAX_N], min_num[MAX_N];
73
    int k;
74
   Stack S;
75
    int on_stack[MAX_N];
76
    int nb_cnt;
77
78
```

MILICIANE /2CAID*11/

1

```
//Duyer do tili bat dad tu dilili d
     void SCC(Graph *pG, int u) {
 80 •
 81
         //1. Đánh số u, đưa u vào ngăn xếp S
 82
         num[u] = min_num[u] = k; k++;
 83
         push(&S, u);
         on_stack[u] = 1;
 84
 85
         //2. Xét các đỉnh kề của u
         for (int v = 1; v \le pG -> n; v++) {
 86
              if (adjacent(pG, u, v)) {
 87
 88
                  if (num[v] < 0) {</pre>
 89
                      SCC(pG, v);
 90
                      min_num[u] = min(min_num[u], min_num[v]);
                  } else if (on_stack[v])
 91
 92
                      min_num[u] = min(min_num[u], num[v]);
 93
 94
         //3. Kiểm tra u có phải là đỉnh khớp
 95
 96
         if (num[u] == min_num[u]) {
              //printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);
 97
 98
             nb_cnt++;
             int w;
 99
100
                      //Lấy các đỉnh trong S ra cho đến khi gặp u
              do {
101
                  w = top(\&S);
102
                  pop(&S);
103
                  on_stack[w] = 0;
104
                  //printf("Lay %d.\n", w);
105
106
             } while (w != u);
107
108
109
110
111
112
113 v int main() {
114
         //1. Khai báo đồ thị G
115
         Graph G;
116
         //2. Đọc dữ liệu và dựng đồ thị
117
         int n, m, u, v;
         scanf("%d%d", &n, &m);
118
119
         init_graph(&G, n);
120
         for (int e = 0; e < m; e++) {</pre>
              scanf("%d%d", &u, &v);
121
122
              add_edge(&G, u, v);
123
         }
124
125
         for (int u = 1; u <= G.n; u++)</pre>
126
            num[u] = -1;
127
128
         //3. Duyệt toàn bộ đồ thị để kiểm tra chu trình
129
130
                              //1b. Tất cả đều chưa duyệt
         k = 1;
131
         make_null_stack(&S);
                                  //1c. Làm rỗng ngăn xếp
         //2. Duyệt toàn bộ đồ thị để tìm BPLT mạnh
132
133
         nb_cnt = 0;
134
         for (int u = 1; u <= G.n; u++)</pre>
135
             if (num[u] == -1)
                                     //u chưa duyệt
136
                  SCC(&G, u);
                                  //duyệt nó
137
138
139
         printf("%d\n", nb_cnt);
140
141
         return 0;
142
143
144
```

13

→ * Bài tập 13 - Tính liên thông mạnh

Jump to...

Bổ sung - Kiểm tra chu trình - đồ thị vô hướng ►