BÀI TẬP LÝ THUYẾT ĐỒ THỊ

Bài tập chương 3

Bài 1 Cho G là đồ thị vô hướng liên thông m cạnh, n đỉnh. Chứng minh m ≥ n-1.

n-1<= m <= n(n-1)/2

Sử dụng quy nạp, ta có:

- Với n bằng 1, đồ thị liên thông có 1 đỉnh và ít nhất 0 cạnh
- Với n bằng 2, đồ thị liên thông có 2 đỉnh và ít nhất 1 cạnh
- Với n bằng 3, đồ thị liên thông có 3 đỉnh và ít nhất 2 cạnh

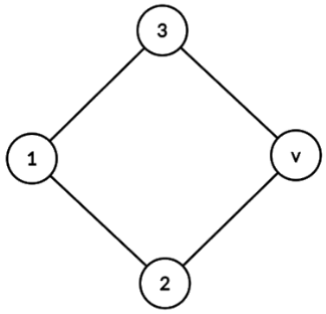
Giả sử đồ thị vô hướng liên thông G có k đỉnh, số cạnh tối thiểu của đồ thị sẽ là k – 1 (*)

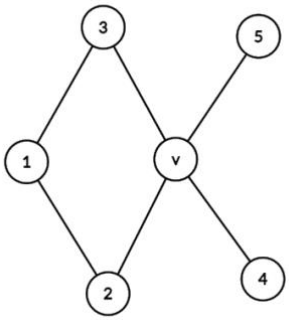
Ta cần chứng minh mệnh đề (*) cũng đúng với k + 1.

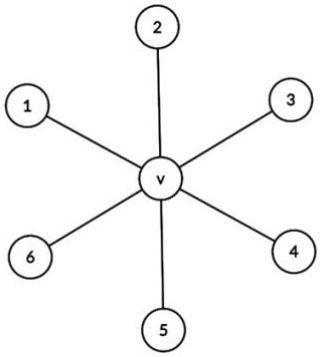
Gọi G là đồ thị vô hướng liên thông có n + 1 đỉnh, v là một đỉnh thuộc đồ thị G. Ta cần chứng minh G có tối thiểu n cạnh.

Loại đỉnh v ra khỏi đồ thị G, ta thu được G’.

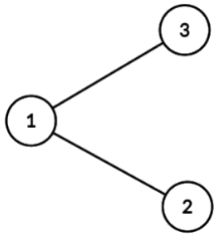
G

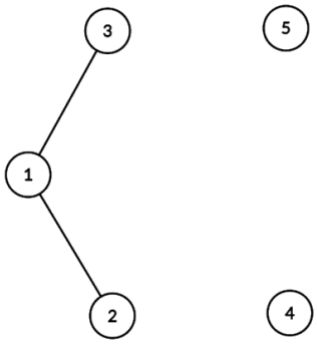


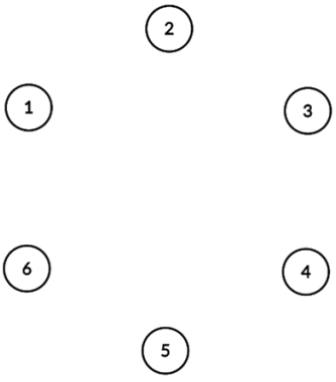




G’







Giả sử G’ có s thành phần liên thông, $1 \leq s \leq n$, áp dụng (*) cho s thành phần liên thông, ta có.

Đồ thị con liên thông 1 có k_1 đỉnh, có ít nhất $k_1 - 1$ cạnh

Đồ thị con liên thông 2 có k_2 đỉnh, có ít nhất $k_2 - 1$ cạnh

…

Đồ thị con liên thông s có k_s đỉnh, có ít nhất $k_s - 1$ cạnh

---

Đồ thị G’ có n đỉnh, có ít nhất n – s cạnh

Với mỗi thành phần liên thông trong G’, cần ít nhất 1 đỉnh liên thông với đỉnh v để G liên thông.

Để tái tạo lại đồ thị G từ đồ thị G’, ta cần ít nhất s cạnh. Suy ra, số cạnh tối thiểu để G liên thông là (n – s) + s = n => đpcm

Bài 2 Cho G là đơn đồ thị vô hướng m cạnh, n đỉnh và có p thành phần liên thông. Chứng minh rằng m ≥ n-p.

Giả sử G có s thành phần liên thông, $1 \leq s \leq n$, áp dụng câu 1 cho s thành phần liên thông, ta có.

Đồ thị con liên thông 1 có k_1 đỉnh, có ít nhất $k_1 - 1$ cạnh

Đồ thị con liên thông 2 có k_2 đỉnh, có ít nhất $k_2 - 1$ cạnh

…

Đồ thị con liên thông p có k_p đỉnh, có ít nhất $k_p - 1$ cạnh

---

Đồ thị G có n đỉnh, có ít nhất n – p cạnh

Bài 3 Cho một đồ thị có 19 cạnh và mỗi đỉnh có bậc lớn hơn hoặc bằng 3. Đồ thị này có tối đa bao nhiêu đỉnh.

Ta có số quan hệ giữa cạnh m và số bậc deg(v) của đồ thị là:

$$2m = \sum_{v \in V} \deg(v) \geq 3n$$

$$\Leftrightarrow 2m \geq 3n$$

$$\Leftrightarrow n \leq \frac{19 \times 2}{3} = \frac{38}{3} \approx 12.67$$

Vậy đồ thị có tối đa 12 đỉnh

Bài 4 Chứng minh rằng trong một đơn đồ thị luôn luôn tồn tại đường đi từ một đỉnh bậc lẻ đến một đỉnh bậc lẻ khác.

Xem xét hai trường hợp sau:

Trường hợp 1: Đồ thị G liên thông => luôn tồn tại đường đi giữa hai đỉnh bất kì. Số đỉnh bậc lẻ của đồ thị là một số chẵn. Suy ra, nếu đồ thị liên thông có đỉnh bậc lẻ thì luôn có đường đi từ một đỉnh bậc lẻ đến một đỉnh bậc lẻ khác.

Trường hợp 2: Giả sử đồ thị G không liên thông và có k thành phần liên thông. Xét G' là một đồ thị con liên thông của G. Giả sử đồ thị G' có một đỉnh bậc lẻ là u.

Ta có mối quan hệ giữa số cạnh m và số bậc deg(v) của đồ thị G' là:

$$2m = \sum_{v \in V} \deg(v)$$

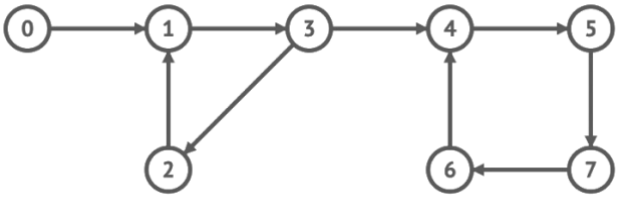
Nếu đồ thị G' chỉ có 1 đỉnh bậc lẻ là u, ta có vế trái (2m) là một số chẵn và vế phải ($\sum_{v \in V} \deg(v)$) là một số lẻ. Suy ra đẳng thức không thỏa.

Để thỏa đẳng thức trên, ta cần ít nhất 2 đỉnh bậc lẻ để số bậc ở vế phải là một số chẵn.

Suy ra, nếu đồ thị G' có đỉnh bậc lẻ, thì G' phải có ít nhất hai đỉnh bậc lẻ.

⇨ đpcm

Bài 5 Cho đồ thị G như sau:



a. Viết thuật toán tìm kiếm theo chiều sâu và biểu diễn quá trình thực hiện thuật toán tìm kiếm theo chiều sâu trên G.

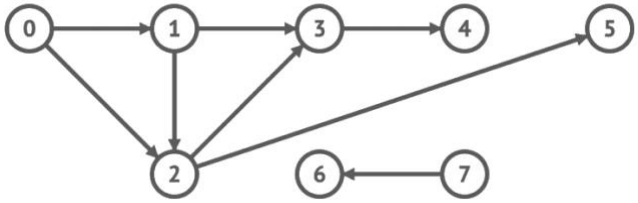
```
visited = set() # Set to keep track of visited nodes.


def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
```

b. Viết thuật toán tìm kiếm theo chiều rộng và biểu diễn quá trình thực hiện thuật toán tìm kiếm theo chiều rộng trên G bắt đầu từ đỉnh 6.

```
visited = [] # List to keep track of visited nodes.
queue = []     #Initialize a queue


def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)


  while queue:
    s = queue.pop(0)
    print (s, end = " ")
```

```
for neighbour in graph[s]:
  if neighbour not in visited:
    visited.append(neighbour)
    queue.append(neighbour)
```

Bài 6 Cho đồ thị G như sau:



a. Viết thuật toán tìm kiếm theo chiều sâu và biểu diễn quá trình thực hiện thuật toán tìm kiếm theo chiều sâu trên G bắt đầu từ đỉnh 0 và 7.

b. Viết thuật toán tìm kiếm theo chiều rộng và biểu diễn quá trình thực hiện thuật toán tìm kiếm theo chiều rộng trên G bắt đầu từ đỉnh 0 và 7.

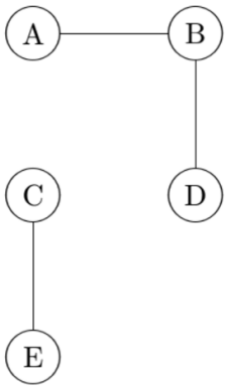
Bài 7 Viết giải thuật kiểm tra xem đồ thị vô hướng G có đường đi từ đỉnh s đến đỉnh t hay không.

```
# Use BFS to check path between s and d
def isReachable(self, s, d):
    # Mark all the vertices as not visited
    visited =[False]*(self.V)

    # Create a queue for BFS
    queue=[]

    # Mark the source node as visited and enqueue it
    queue.append(s)
    visited[s] = True

    while queue:

        #Dequeue a vertex from queue
        n = queue.pop(0)

        # If this adjacent node is the destination node,
        # then return true
        if n == d:
            return True

        #  Else, continue to do BFS
        for i in self.graph[n]:
            if visited[i] == False:
                queue.append(i)
                visited[i] = True
    # If BFS is complete without visited d
    return False
```

Bài 8 Viết giải thuật kiểm tra xem đỉnh s và đỉnh t có thuộc cùng một thành phần liên thông của đồ thị vô hướng G hay không.

Tương tự câu 7.

Bài 9: Đồ thị G liên thông khi có đường đi giữa hai đỉnh bất kỳ của đồ thị. Ví dụ: đồ thị dưới đây không liên thông vì không có đường đi từ A đến C.



Tuy nhiên, đồ thị này chứa một số đồ thị con liên thông được tạo từ các tập hợp đỉnh sau {A, B }, {B, D}, {C, E}, {A, B, D}

Một đồ thị con liên thông là cực đại (maximal) nếu không có đỉnh và cạnh nào trong đồ thị ban đầu có thể được thêm vào đồ thị con và vẫn đảm bảo tính liên thông của đồ thị con. Có hai đồ thị con liên thông cực đại ở trên, một đồ thị bao gồm các đỉnh {A, B, D} và đồ thị còn lại bao gồm các đỉnh {C, E}. Viết mã giả xác định số đồ thị con liên thông cực đại của một đồ thị đã cho.

Đáp án

Ý tưởng:

Lập danh sách các đỉnh của đồ thị.

- Với mỗi đỉnh chưa duyệt, tiến hành tìm kiếm (theo chiều rộng hoặc chiều sâu) đường đi tới tất cả các đỉnh có thể và đánh dấu các đỉnh đã đi qua. Khi quá trình tìm kiếm kết thúc, sẽ thu được một đồ thị con là liên thông cực đại.
- Lặp lại quá trình đến khi không còn đỉnh chưa duyệt.

Mã giả

MinimalSubGraph (V, n, Adj)

```
Input
   V: Danh sách các đỉnh của đồ thị.
   n: Số đỉnh của đồ thị.
   Adj: Danh sách kế cận (Adj[i] là tập các đỉnh kế cận của V[i]).

Begin
   Result = []
   Visited = [0,0,0, … ,0] #Mảng gồm n phần tử (bằng số đỉnh) ghi lại các
   đỉnh đã duyệt hay chưa. (0 là chưa duyệt, 1 là đã duyệt)
   for i := 0 → n:
      if (Visited[i] == 0):
            Visited[i] = 1
            Result.append(BFS (Visited, Adj, V[i]))

Output
   Result: Danh sách các đồ thị liên thông.
```

BFS(Visited, Adj, node)

```
Input
   Visited: Danh sách đánh dấu trạng thái duyệt/chưa duyệt của các đỉnh.
   Adj: Danh sách kế cận (Adj[i] là tập các đỉnh kế cận của V[i]).
   node: đỉnh đang xét.

Begin
   SubGraph = [] #Đồ thị con liên thông cực đại.
   SubGraph.append(node) #Thêm đỉnh node vào đồ thị
   Queue = [] #Khởi tạo hàng đợi
   for element in Adj[node]:
      if (Visited[element] == 0):
            Queue.append(element)
   while (Queue not empty):
      cur = Queue[0]
      Queue.pop(0)
      if (Visited[cur] == 0):
            SubGraph.append(cur)
            Visited[cur] = 1
            for element in Adj[cur]:
                  if (Visited[element] == 0):
                        Queue.append(element)
Output
   SubGraph: Đồ thị con liên thông cực đại chứa đỉnh node
```

Code:

```
#include<bits/stdc++.h>

using namespace std;

/**
 * @brief Tim kiem theo chieu rong duong di qua cac dinh co the
 * tu 1 dinh cho truoc tao thanh do thi con lien thong cuc dai.
 *
 * @param Visited mang 1 chieu danh dau cac dinh da duyet/chua duyet
 * @param Adj danh sach ke can
 * @param node dinh can xet
 * @return vector<int> tap hop cac dinh tao thanh do thi con lien thong cuc dai
 */
vector<int> BFS(int Visited[],vector<int> Adj[],int node)
{
    vector<int> SubGraph; //vector kieu int luu lai cac dinh cua do thi con
    SubGraph.push_back(node); //day dinh dau tien vao do thi con
    queue<int> Queue; //hang doi chua cac dinh co the duyet qua

    //duyet qua cac dinh ke cua dinh node
    for (auto element:Adj[node])
    {
        //kiem tra tung dinh da duoc duyet hay chua
        if(Visited[element] == 0)
        {
            //neu chua duyet thi day vao hang doi Queue
            Queue.push(element);
        }
    }


    //Lap lai khi ma Queue van con phan tu
    while (!Queue.empty())
    {
        //xet dinh dau trong hang doi
        int cur = Queue.front();
        Queue.pop(); //xoa dinh dau tien ra khoi hang doi
        //kiem tra dinh hien tai da duyet qua hay chua
        if (Visited[cur] == 0)
        {
            //neu chua duyet qua

            SubGraph.push_back(cur); //luu dinh hien tai vao do thi con
            Visited[cur] = 1; //danh dau dinh hien tai da duoc duyet
            //them cac dinh ke chua duyet cua dinh hien tai vao hang doi
            for (auto element:Adj[cur])
            {
                if(Visited[element] == 0)
                {
                    Queue.push(element);
                }
            }
        }
    }
    //tra ve ket qua la do thi con lien thong cuc dai
```

```
    return SubGraph;
}

/**
 * @brief Tim kiem cac do thi con lien thong cuc dai cua 1 do thi cho truoc
 *
 * @param V Mang 1 chieu chua cac dinh cua do thi
 * @param n So luong dinh cua do thi
 * @param Adj Danh sach ke can cua cac dinh trong do thi
 * @return vector<vector<int>> Danh sach cac do thi con lien thong cuc dai
 */
vector<vector<int>> MinimalSubGraph(int V[],int n,vector<int> Adj[])
{
    vector<vector<int>> Result; // Tap cac do thi con lien thong cuc dai
    int Visited[n]; // Mang danh dau trang thai
    //Khoi tao gia tri ban dau cua mang danh dau
    for (int i = 0; i < n; i++)
    {
        Visited[i] = 0;
    }

    //Duyet qua tung dinh cua do thi va tien hanh tim kiem
    for (int i = 0; i < n; i++)
    {
        //Kiem tra dinh chua duoc duyet
        if (Visited[i] == 0)
        {
            Visited[i] = 1; //Danh dau da duyet qua dinh i
            Result.push_back(BFS(Visited,Adj,i)); //Them do thi con cuc dai chua
dinh i vao ket qua
        }
    }
    //tra ve ket qua la tap cac do thi con lien thong cuc dai
    return Result;
}
int main()
{

    //Cac dinh V cua do thi
    int V[] = {0,1,2,3,4};

    //Khoi tao ma tran ke
    vector<int> Adj[5];
    Adj[0].push_back(1);
    Adj[1].push_back(0);
    Adj[1].push_back(2);
    Adj[2].push_back(1);
    Adj[3].push_back(4);
    Adj[4].push_back(3);


    //Tim Cac do thi con
    vector<vector<int>> Result = MinimalSubGraph(V,5,Adj);


    int counting = 1;
    //print Result
```

```
    for (auto subGraph : Result)
    {
        cout << "Do thi con lien thong cuc dai thu " << counting++ << ": { ";
        for (auto vertex: subGraph)
        {
            cout << vertex << " ";
        }
        cout<<"}"<<endl;
    }

    return 0;
}
```

Bài 10: **Định lý bốn màu** (còn gọi là *định lý bản đồ bốn màu*) phát biểu rằng đối với bất kỳ mặt phẳng nào được chia thành các vùng phân biệt, chẳng hạn như bản đồ hành chính của một quốc gia, chỉ cần dùng tối đa bốn màu để phân biệt các vùng lân cận với nhau. Hai vùng được coi là lân cận nếu như chúng có chung nhau một đoạn đường biên, không tính chung nhau một điểm.

Với yêu cầu đơn giản hơn, xác định xem một đồ thị liên thông có thể được tô hai màu hay không. Nghĩa là, có thể gán màu (từ một bảng hai màu) cho các đỉnh sao cho không có hai đỉnh liền kề nào có cùng màu. Để đơn giản hóa vấn đề, áp dụng các giả thiết sau:

• Không có đỉnh nào có cạnh khuyên.

• Đồ thị là vô hướng. Nghĩa là, nếu một đỉnh a liên thông với đỉnh b, thì thì đỉnh b cũng sẽ liên thông với đỉnh a.

• Đồ thị liên thông mạnh. Nghĩa là, sẽ có ít nhất một đường đi một đỉnh bất kì đến một đỉnh bất kì khác.

Viết mã giả để xác định xem một đồ thị có thể được tô bằng hai màu hay không?

Đáp án

Ý tưởng

Duyệt qua tất cả đỉnh của đồ thị và tiến hành tô màu cho các đỉnh.

Các đỉnh được đánh dấu 1 trong 3 trạng thái:

- -1: chưa tô màu
- 0: màu thứ nhất
- 1: màu thứ hai

Tại mỗi đỉnh xét:

- Nếu đỉnh không có màu, tô màu cho đỉnh khác với màu của đỉnh liền trước.
- Nếu đỉnh có màu và giống với màu của đỉnh liền trước → đồ thị không thể tô bằng 2 màu.
- Nếu đỉnh có màu và khác với đỉnh liền trước → tiếp tục tô đến khi tất cả các đỉnh đều có màu.

Nếu có thể tô màu cho tất cả đỉnh → đồ thị có thể được tô bằng 2 màu.

Mã giả

```
Input
   V: Tập hợp các đỉnh của đồ thị.
   n: Số đỉnh của đồ thị
   Adj: Danh sách kế cận (Adj[i] là tập các đỉnh kế cận của V[i])
   node: đỉnh bắt đầu.

Begin
   Color = [-1, -1, …, -1] # Color[i] là màu của đỉnh V[i]
   Visited = [0, 0, …, 0] # Đánh dấu các đỉnh đã được duyệt hay
   chưa
   curColor = 0 # Chọn màu ban đầu
   Queue = [] # Hàng đợi của các đỉnh tiếp theo
   Result = True
   Color[node] = curColor
   Visited[node] = 1
   for neighbor in Adj[node]:
     Queue.append(neighbor)
     Color[neighbor] = abs(curColor - 1)

   while (Queue not empty and Result == True):
     curNode = Queue[0]
     Queue.pop(0)
     for neighbor in Adj[curNode]:
          if (Color[neighbor] == curColor):
               Result = False
          else:
               Color[neighbor] = abs(curColor - 1)
               if (Visited[neighbor] == 0):
                    Queue.append(neighbor)
     Visited[curNode] = 1
     curColor = abs(curColor - 1)

Output
   Result : True/False Đồ thị có thể tô bằng 2 màu hay không
```

Code:

```
#include<bits/stdc++.h>

using namespace std;

/**
 * @brief Kiem tra mot do thi co the duoc to bang hai mau hay khong
 *
 * @param V Mang 1 chieu chua cac dinh cua do thi
 * @param n So luong dinh cua do thi
 * @param Adj Danh sach ke can cua cac dinh trong do thi
 * @param node Dinh duoc chon de bat dau
 * @return true Do thi co the duoc to bang hai mau
 * @return false Do thi khong the duoc to bang hai mau
 */
bool isTwoColorGraph(int V[], int n, vector<int> Adj[], int node)
{
    int Color[n]; //Mang ghi lai mau cua dinh
    int Visited[n]; //Mang ghi lai trang thai da duyet/chua duyet cua
cac dinh
    int curColor = 0; //Chon mau hien tai ban dau
    bool Result = true; //Ket qua kiem tra do thi co the to 2 mau hay
khong

    //Khoi tao cac gia tri ban dau
    for (int i = 0; i < n; i++)
    {
        Color[i] = -1;
        Visited[i] = 0;
    }
    queue<int> Queue; //Hang doi ghi lai cac dinh tiep theo
    Color[node] = curColor; //To mau cho dinh dau tien
    //Day cac dinh ke voi dinh ban dau vao hang doi
    for (auto neighbor : Adj[node])
    {
        Queue.push(neighbor);
        Color[neighbor] = abs(curColor - 1);
    }
    Visited[node] = 1; //Danh dau dinh ban dau da duoc duyet
    curColor = abs(curColor - 1); //Thay doi gia tri mau cho cac dinh
tiep theo
    //Duyet qua cac dinh theo chieu ngang va to mau
    while (!Queue.empty() && Result)
    {
        int curNode = Queue.front(); //lay ra dinh dua tien trong
hang doi
        Queue.pop(); //xoa dinh dau tien trong hang doi
```

```
        for (auto neighbor : Adj[curNode])
        {
            //kiem tra gia tri mau cua cac dinh ke can v dinh dang
xet
            if (Color[neighbor] == curColor)
            {
                //neu dinh ke can co mau giong v dinh dang xet
                //thay doi ket qua thanh false
                Result = false;
            }
            else
            {
                //neu nguoc lai to mau cho dinh ke can
                Color[neighbor] = abs (curColor - 1);
                //them dinh ke can vao hang doi neu dinh chua duoc
duyet
                if (Visited [neighbor] == 0)
                {
                    Queue.push(neighbor);
                }
            }
        }
        Visited[curNode] = 1;
        curColor = abs (curColor - 1);
    }
    //tra ve ket qua kiem tra
    return Result;
}

int main()
{
    int V[] = {0,1,2,3,4};

    //Khoi tao ma tran ke
    vector<int> Adj[5];
    Adj[0].push_back(1);
    Adj[1].push_back(0);
    Adj[0].push_back(2);
    Adj[2].push_back(0);
    Adj[1].push_back(2);
    Adj[2].push_back(1);
    Adj[2].push_back(3);
    Adj[3].push_back(2);
    Adj[3].push_back(4);
    Adj[4].push_back(3);

    if (isTwoColorGraph(V,5,Adj,0))
```

```
    {
        cout<<"Graph can be draw with 2 color"<< endl;
    }
    else
    {
        cout<<"Graph can not be draw with 2 color"<< endl;
    }

    return 0;
}
```