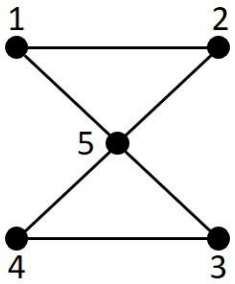
ĐỒ THỊ EULER

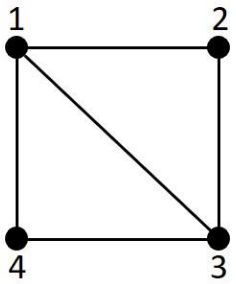
1. Khái niệm

Đồ thị vô hướng được gọi là Euler nếu tồn tại 1 chu trình đơn đi qua tất cả cạnh của đồ thị, mỗi cạnh qua đúng 1 lần. Một đồ thị được gọi là nửa Euler nếu tồn tại 1 đường đi đơn qua tất cả cạnh của đồ thị, mỗi cạnh đúng 1 lần.

Chu trình đi qua tất cả các cạnh của đồ thị, mỗi cạnh qua đúng 1 lần gọi là chu trình Euler. Đường đi qua tất cả các cạnh của đồ thị, mỗi cạnh qua đúng 1 lần gọi là đường đi Euler.



Đồ thị Euler



Đồ thị nửa Euler xuất phát từ đỉnh 1 hoặc 3

1.1. Định lý

Bổ đề: Xét đồ thị G vô hướng, liên thông. Nếu bậc mỗi đỉnh của G không nhỏ hơn 2 thì G chứa chu trình.

Chứng minh bổ đề: Ta xây dựng quy nạp đường đi như sau:

$$u \to x_1 \to x_2 \to \cdots \to x_i \to x_{i+1} \to \cdots$$

Trong đó u kề với x_1 và x_i kề với $x_{i+1} (\forall i \geq 1)$. Vì $d(x_i) \geq 2$ nên ta có $x_{i+1} \neq x_{i-1}$. Vì số đỉnh là hữu hạn nên sau một số hữu hạn bước sẽ gặp lại đỉnh x_j đã xuất hiện trước đó. Vậy khi đó ta tìm được 1 chu trình.

Định lý: Đồ thị vô hướng G được gọi là Euler khi và chỉ khi G liên thông và bậc của mọi đỉnh đều chẵn.

Chứng minh định lý:

Điều kiện cần: đồ thị G có chu trình Euler thì bậc của mọi đỉnh là chẵn

- Mỗi đỉnh đều có cạnh nối vào và cạnh đi ra nên số bậc phải là chẵn

Điều kiện đủ: đồ thị G liên thông có bậc của mọi đỉnh chẵn là đồ thị Euler

- Vì G liên thông và $d(x)$ chẵn $\forall x \in V$ nên $d(x) \geq 2$. Theo bổ đề thì đồ thị G phải chứa chu trình. Nếu tất cả cạnh của G đều thuộc chu trình thì G là đồ thị Euler.
- Trường hợp chu trình C tìm được không đi qua tất cả cạnh của G. Khi đó nếu xóa tất cả cạnh thuộc C ra khỏi G ta được một đồ thị mới G'. Vì các đỉnh của G' vẫn có bậc chẵn nên sẽ tìm được một chu trình C' trong G'. Vì đồ thị G liên thông nên C và C' có ít nhất 1 đỉnh chung. Giả sử đỉnh chung đó là x, ta có $C = x \to x_1 \to \cdots \to x_k \to x$ và $C' = x \to y_1 \to \cdots \to y_l \to x$. Khi đó ta có chu trình mới $x \to x_1 \to \cdots \to x_k \to x \to y_1 \to \cdots \to y_l \to x$.
- Theo quy nạp thì chu trình tìm được sẽ chứa tất cả cạnh của G và G là đồ thị Euler.

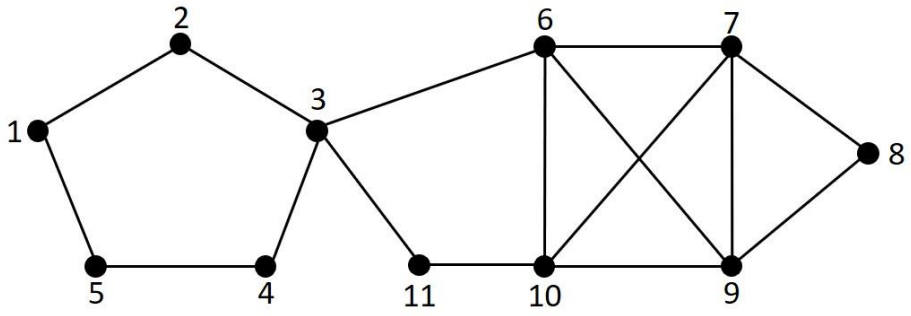
Hệ quả: Đồ thị vô hướng liên thông G có đường đi Euler khi và chỉ khi G *có đúng 2 đỉnh bậc lẻ*. Đường đi Euler xuất phát tại 1 đỉnh có bậc lẻ và kết thúc ở đỉnh có bậc lẻ còn lại.

1.2. Thuật toán tìm chu trình Euler

Chứng minh định lý cho ta thuật toán tìm chu trình Euler như sau:

- Chọn một đỉnh bất kỳ, từ đỉnh đó đi qua các cạnh, qua cạnh nào thì xóa cạnh đó.
- Nếu không còn đi được nữa thì lần ngược lại những đỉnh đã qua để tìm đỉnh còn cạnh chưa đi (tìm đỉnh chung của 2 chu trình).
- Nếu tồn tại đỉnh như thế thì đổi đỉnh này làm đỉnh xuất phát của chu trình và tiếp tục hành trình.

Ví dụ: Áp dụng thuật toán để tìm chu trình Euler cho đồ thị như hình minh họa



-

- Thuật toán xuất phát từ đỉnh 1. Các đỉnh kề được xét theo thứ tự từ nhỏ đến lớn.
- Thuật toán đi và xóa các cạnh như sau: $(1,2), (2,3), (3,4), (4,5), (5,1)$, các dãy đỉnh đi qua: 1,2,3,4,5,1.
- Thuật toán lần ngược lại những đỉnh vừa đi qua để tìm đỉnh còn cạnh chưa đi. Đỉnh 3 còn cạnh chưa đi qua nên chu trình đổi thành: 3,4,5,1,2,3.
- Hành trình tiếp tục đi qua và xóa các cạnh như sau: $(3,6), (6,7), (7,8), (8,9), (9,6), (6,10), (10,7), (7,9), (9,10), (10,11), (11,3)$. Các đỉnh đi qua: 3,6,7,8,9,6,10,7,9,10,11,3.
- Ghép 2 chu trình ta có chu trình: 3,4,5,1,2,3,6,7,8,9,6,10,7,9,10,11,3.

2. Cài đặt thuật toán

Để thực hiện thao tác xóa cạnh đã đi qua, tổ chức mảng `deleted[id]` để đánh dấu xóa cạnh có thứ tự `id`. Mỗi cạnh vô hướng (u, v) của đồ thị được tách thành 2 cạnh ngược hướng (u, v) và (v, u) có vị trí tương ứng là id và $2 \times m - id + 1$ trên mảng `deleted`. Do vậy trên danh sách kề biểu diễn đồ thị, ta cần lưu thêm thông tin thứ tự của cạnh.

Đoạn chương trình minh họa tổ chức CTDL biểu diễn đồ thị và hàm xây dựng đồ thị.

```
#define maxN 100000
#define maxM 1000000

struct edge
{
    int v, id;
};

typedef vector<edge> vii;

vii g[maxN+1];
bool deleted[2*maxM+1] = {false};

void BuildGraph()
{
    int u, v;
    cin>>n>>m;
    for (int i = 1; i <= m; ++i)
    {
        cin>>u>>v;
        g[u].push_back({v, i}); //cung (u, v) có vị trí i
        g[v].push_back({u, 2*m-i+1});//cung (v, u) có vị trí 2m-i+1
    }
}
```

2.1. Tìm chu trình bằng cách lặp không đệ quy

```
int d[maxN+1] = {0}; //lưu bậc của các đỉnh
vector<int> Cycle;//lưu các đỉnh thuộc chu trình Euler

void Flip(int lo, int hi)
{
    while (lo < hi)
        swap(Cycle[lo++], Cycle[hi--]);
}

void Flip(int id)
{
    int n = Cycle.size()-1, x = Cycle[id];
    Flip(0, id-1);
    Flip(id, n);
    Flip(0, n);
    Cycle.push_back(x);
}

void Euler()
{
    Cycle.push_back(1);
    bool ok = true;
    while (ok)
    {
        int u = Cycle.back();
        ok = false;
        for (int i = 0; d[u] > 0 && i < g[u].size(); ++i)
        {
            int v = g[u][i].v, id = g[u][i].id;
            if (mark[id] == false)
            {
                Cycle.push_back(v);
                mark[id] = mark[2*m-id+1] = ok = true;
                --d[u], --d[v];
                break;
            }
        }
        if (!ok) //lần ngược để tìm đỉnh còn cạnh chưa đi
        {
            Cycle.pop_back();
            for (int i = C.size()-1; i >= 0; --i)
            {
                if (d[Cycle[i]] > 0) //đỉnh Cycle[i] còn cạnh chưa đi
                {
```

```
                    Flip(i);
                    ok = true;
                    break;
                }
            }
        }
    }
    Cycle.push_back(Cycle[0]);
    for (int i = 0; i < Cycle.size(); ++i)
        cout<<Cycle[i]<<" ";
}
```

2.2. Tìm chu trình bằng phương pháp DFS

Sử dụng đệ quy

```
void Euler(int u)
{
    for (int i = 0; i < g[u].size(); ++i)
    {
        int v = g[u][i].v, id = g[u][i].id;
        if (mark[id] == false)
        {
            mark[id] = mark[2*m-id+1] = true;
            Euler(v);
        }
    }
    Cycle.push(u);
}
```

Khử đệ quy

```
void Euler()
{
    stack<int> S;
    S.push(1);
    while (!S.empty())
    {
        int u = S.top();
```

```
        bool ok = false;
        for (int i = 0; i < g[u].size(); ++i)
        {
            int v = g[u][i].v, id = g[u][i].id;
            if (mark[id] == false)
            {
                S.push(v);
                mark[id] = mark[2*m-id+1] = ok = true;
                break;
            }
        }
        if (!ok)
        {
            C.push(S.top());
            S.pop();
        }
    }
}
```