Video bài giảng phần 1: https://youtu.be/NcK0hsAf1Uk
Video bài giảng phần 2: https://youtu.be/3u3HKqgRIp0

Khớp và cầu

Định nghĩa

Cho một đồ thị vô hướng (có thể không liên thông):
+ Một cạnh của đồ thị được gọi là “cầu” khi và chỉ khi nếu xóa cạnh này khỏi đồ thị, số TPLT của đồ thị tăng lên (việc xóa cạnh này chia một TPLT có sẵn thành nhiều phần nhỏ hơn).
+ Một đỉnh của đồ thị được gọi là “khớp” khi và chỉ khi nếu xóa đỉnh này khỏi đồ thị, số TPLT của đồ thị tăng lên.

Cây DFS

Thuật toán DFS có dạng như sau:

```
int numNode, numEdge;
vector<int> adj[MAX]; // danh sách kề
bool visited[MAX]; // đánh dấu đỉnh đã thăm

void dfs(int u) {
      visited[u] = true;
      for (int v : adj[u]) if (!visited[v]) dfs(v);
}
for (int i = 1; i <= numNode; i++) if (!visited[i]) dfs(i);
```

Từ thuật toán DFS, ta định nghĩa khái niệm “cây DFS" như sau: Do mỗi đỉnh v bất kì trên đồ thị chỉ được thăm một lần, hàm dfs được gọi chính xác 1 lần với mỗi đỉnh v. Để gọi ra hàm DFS(v), có hai cách gọi: gọi từ vòng for (i: 1 -> n) ở bên ngoài hoặc gọi thông qua một hàm DFS(u) nào đó. Nếu hàm DFS(v) được gọi đệ quy từ một hàm DFS(u), ta gọi u là cha của v. Nếu hàm DFS(v) được gọi từ vòng for bên ngoài, ta coi v là gốc của cây.

Với việc định nghĩa quan hệ cha con như trên, ta có một được một rừng các cây. Mỗi TPLT trong đồ thị gốc cho ta chính xác một cây DFS.

Nhận xét: Theo cách xây dựng quan hệ cha con: u là cha của v chỉ khi có cạnh nối giữa u và v. Thật vậy, để u là cha của v thì hàm dfs(v) phải được gọi đệ quy từ bên trong hàm dfs(u). Mặt khác, hàm dfs(u) chỉ gọi đệ quy tới các đỉnh v kề với u, vì vậy để u là cha của v, chắc chắn giữa u và v phải có cạnh.

Do đó, một cây DFS (hoặc rừng DFS) là một tập hợp các cạnh trên đồ thị ban đầu (tập hợp các cạnh thuộc cây DFS là một tập con của tập các cạnh trong đồ thị ban đầu. Như đã nói ở trên, số cây DFS có được = số TPLT của đồ thị. Tức là một TPLT trong đồ thị ban đầu cho ta một cây DFS. Do đó, tập đỉnh trên cây DFS == tập đỉnh trong TPLT. Như vậy, tập cạnh thuộc cây DFS nối các đỉnh thuộc một TPLT với nhau. Do đó, cây DFS là một cây khung của TPLT.

Ngoài tính chất liên thông (các cạnh của cây DFS kết nối tất cả các đỉnh với nhau), ta có một tính chất rất quan trọng của cây DFS như sau: Nếu u và v được nối với nhau bởi một cạnh và cạnh này không thuộc cây DFS, chắc chắn hoặc u là tổ tiên của v hoặc v là tổ tiên của u.

Bài toán tìm cầu

Trong bài toán này, ta quan tâm tới việc “thử xóa đi một cạnh khỏi đồ thị" và “số TPLT của đồ thị sau khi xóa". Như đã phân tích ở trên: cây DFS là cây khung. Nói cách khác, nếu chỉ giữ lại các cạnh thuộc cây DFS, tính liên thông của đồ thị là không đổi (nếu có tất cả các cạnh thuộc đồ thị ban đầu hay chỉ chữ lại các cạnh trên cây DFS, tính liên thông cũng giống nhau). Điều đó có nghĩa là, ngay cả khi đã xóa hết các cạnh màu đen (những cạnh không thuộc cây DFS), số TPLT của đồ thị không tăng. Do đó, việc chỉ xóa đi một cạnh màu đen nào đó không thể làm thay đổi tính liên thông của đồ thị. Vì vậy, **mọi cạnh màu đen không thể là cầu**.

Xét một trường hợp, giả sử là đồ thị không có cạnh màu đen nào. Khi đó, mọi cạnh của đồ thị đều thuộc cây DFS. Hệ quả, đồ thị chính là cây. Nếu đồ thị là cây, tất cả các cạnh trên cây đều là cầu. Do đó, sự xuất hiện của những cạnh không thuộc cây DFS làm cho những cạnh thuộc cây DFS không thể là cầu.

Từ đây, để phân tích kĩ hơn về bài toán liệt kê cầu, ta giả sử đồ thị ban đầu là đồ thị liên thông. Nói cách khác, ta chỉ có duy nhất một cây DFS. Việc có nhiều cây DFS không làm thay đổi thuật toán, vì các cây là độc lập nhau và được xử lý theo cách hoàn toàn tương tự nhau.

Bây giờ, ta đi tìm câu trả lời cho câu hỏi: xét một cạnh (u, v) thuộc cây DFS, hỏi cạnh này có là cầu hay không? Nhắc lại kiến thức cũ: Một cạnh trên cây luôn nối giữa một cha và một con. Nói cách khác, nếu u và v kề nhau và cạnh (u, v) thuộc cây DFS, một trong hai điều sau đây đúng: u là cha của v và v là cha của u. KMTQ, ta giả sử u là cha của v.

Nếu đồ thị ban đầu là cây, chắc chắn cạnh (u, v) là cầu. Việc xóa đi cạnh (u, v) là cho cây bị chia ra làm 2 TPLT: cây con gốc v và phần bù cây con gốc v.

Như đã phân tích ở trên, sự xuất hiện của các cạnh màu đen (cạnh không thuộc cây DFS) khiến cho cạnh (u, v) có thể không phải là cầu. Nếu cạnh (u, v) không phải là cầu, điều đó có nghĩa là bên trong cây con gốc v có cạnh màu đen nối ra bên ngoài cây con gốc v. Theo một nhận xét khác ở trên, cạnh màu đen (không thuộc cây DFS) luôn nối giữa một con cháu và một tổ tiên. Từ đây, ta có kết luận: Nếu một cạnh phá hoại làm cho (u, v) không phải là cầu, cạnh phá hoại đấy phải thỏa mãn các yêu cầu sau đây:
+ Đây là cạnh màu đen (không thuộc cây DFS)
+ Cạnh này phải nối giữa một đỉnh bên trong cây con gốc v tới một tổ tiên của v (tổ tiên của v bao gồm u và các tổ tiên của u).

Như vậy, để kiểm tra cạnh (u, v) thuộc cây DFS có phải cầu hay không, ta cần xem xét các cạnh màu đen (đối tượng phá hoại) nối từ bên trong cây con gốc v. Vì vậy, ta định nghĩa hai giá trị là low và num.
Giống như khi DFS ở trên cây, khi DFS trên đồ thị, ta cũng có thể đánh số các đỉnh theo thứ tự thăm DFS. Vì vậy, giá trị num[u] được hiểu là số thứ tự của đỉnh u (thứ tự đỉnh u được thăm). low[v] là số thứ tự nhỏ nhất của một đỉnh mà từ bên trong cây con gốc v có thể nối tới nó thông qua các cạnh màu đen.

Sau khi tính được các giá trị low và num, ta có điều kiện cần và đủ để cạnh (u, v) là cầu: low[v] > num[u]. Nhắc lại tính chất về số thứ tự khi thăm DFS: đỉnh cha được thăm trước đỉnh con, do đó num của đỉnh cha < num của đỉnh con.low[v] là STT nhỏ nhất của đỉnh mà nối từ bên trong cây con gốc v thông qua các cạnh màu đen. Việc low[v] > num[u] có nghĩa là số thứ tự nhỏ nhất > num[u], tức là trong các đỉnh nối được, không có đỉnh nào là u hoặc là các tổ tiên của u (vì các tổ tiên của u chắc chắn có STT < num[u]). Như vậy, thông qua các cạnh màu đen, từ bên trong cây con gốc v không thể nối được tới các tổ tiên của v. Tức là cạnh phá hoại không tồn tại, do đó (u, v) là cầu.

Ngược lại, trong tr Tarjan (cho đồ thị có hướng), khớp và cầu đều sử dụng ý tưởng DFS với các mảng low và num. Trong cả 3 thuật toán này, mảng low và num được tính theo cách tương tự nhau: num[u] = số thứ tự của

đỉnh u khi thăm DFS. Quá trình DFS có dạng: Xét các v kề với u, nếu v chưa thăm (num[v] == 0): thăm v và minimize(low[u], low[v]); ngược lại (nếu v đã thăm) minimize(low[u], ường hợp low[v] <= num[u], điều đó có nghĩa là từ trong cây con gốc v có nối được tới một đỉnh nào đó là u hoặc tổ tiên của u, vì vậy cạnh (u, v) không phải là cầu.

```
int numNode, numEdge;
vector<int> adj[MAX]; // danh sách kề
int low[MAX], num[MAX], cnt;

void dfs(int u) {
	low[u] = numNode + 1; // dương vô cùng
	num[u] = ++cnt; // đánh số theo thứ tự thăm dfs

	for (int v : adj[u]) {
		if (num[v] == 0) {
			// đỉnh v chưa được thăm, do đó bây giờ ta sẽ gọi dfs(v)
			// như vậy, v là con của u trên cây DFS và cạnh (u, v) thuộc cây DFS
			dfs(v);
			if (low[v] > num[u]) cạnh (u, v) là cầu
			minimize(low[u], low[v]);
		} else {
			// đỉnh v đã được thăm trước đó, do đó cạnh (u, v) ở trường hợp này
			// không thuộc cây DFS
			minimize(low[u], num[v]);
		}
	}
}
for (int i = 1; i <= numNode; i++) if (!visited[i]) dfs(i);
```

Chú ý: Cả 3 thuật toán Tarjan (cho đồ thị có hướng), khớp và cầu đều sử dụng ý tưởng DFS với các mảng low và num. Trong cả 3 thuật toán này, mảng low và num được tính theo cách tương tự nhau: num[u] = số thứ tự của đỉnh u khi thăm DFS. Quá trình DFS có dạng: Xét các v kề với u, nếu v chưa thăm (num[v] == 0): thăm v và minimize(low[u], low[v]); ngược lại (nếu v đã thăm) minimize(low[u], num[v]).

Tuy nhiên, đoạn code trên cần phải chỉnh sửa thêm để thực sự đạt được ý tưởng mà nó muốn thể hiện. Ý tưởng nó muốn thể hiện ở đây là: Khi xét các đỉnh v kề u (bên trong hàm dfs), nếu v chưa thăm (if (num[v] == 0)) thì cạnh (u, v) thuộc cây DFS, ngược lại (else) thì cạnh (u, v) không thuộc cây DFS. Nhưng đoạn code trên chưa thể hiện được ý tưởng này. Cụ thể, xét một đồ thị nào đó, mà ở trong đó, đỉnh số 5 là cha của đỉnh số 6 trên cây DFS, tức là cạnh (5, 6) là cạnh thuộc cây DFS). Chú ý rằng, do 5 kề với 6 trên đồ thị gốc, trong danh sách kề của 5 có 6 mà trong danh sách kề của 6 cũng có 5.

Diễn biến của thuật toán DFS diễn ra như sau:
- Đầu tiên, hàm DFS(5) được gọi ra: Trong quá trình xét các đỉnh kề với 5, có đỉnh 6; lúc này đỉnh 6 chưa được thăm, nên đỉnh 6 được xếp là con của 5 trên cây DFS và hàm dfs(6) được gọi ra, đồng thời 6 được đánh dấu là đã thăm.
- Tiếp theo, hàm DFS(6) được gọi ra: Trong quá trình xét các đỉnh kề với 6, có đỉnh 5; lúc này đỉnh 5 hiển nhiên đã được thăm từ trước (hàm dfs(6) được gọi ra sau hàm dfs(5)); vì vậy trong lệnh if, nhánh else được thực thi. Chú ý, ở thời điểm này, lệnh minimize(low[u], num[v]) được thực hiện với v = 6 và u = 5, tức là gán num[5] cho low[6].
- Cuối cùng, sau khi hàm DFS(6) kết thúc, chuỗi đệ quy lại trở về hàm DFS(5), khi đó điều kiện cạnh (5, 6) là cầu được kiểm tra: lúc này ta có u = 5 và v = 6, điều kiện để cạnh (5, 6) là cầu là low[6] > num[5]. Tuy nhiên, do đã có

lệnh gán num[5] cho low[6] ở trên, chắc chắn điều kiện này không bao giờ xảy ra, tức là cạnh (5, 6) không thể nào là cầu.

Như vậy, ta cần chỉnh sửa lại đoạn code này một chút để sao cho trong hàm DFS(6), ta không coi 5 là đỉnh kề của 6 nữa, để ngăn chặn việc num[5] được gán cho low[6]. Để làm được điều này, ta cần dùng một phương pháp đánh dấu cạnh đã thăm (không chỉ là đỉnh đã thăm).

Thao tác này còn được gọi là “định chiều cây DFS" bởi vì mỗi một cạnh (u, v) thông thường được xét hai lần (lần 1: trong hàm dfs(u) xét đỉnh v kề với u. lần 2: trong hàm dfs(v) xét đỉnh u kề với v). Thao tác này đảm bảo cạnh chỉ được xét ở 1 lần (nói cách khác, nếu đã xét theo hướng u -> v thì không xét theo hướng v -> u nữa). Do đó, nó giống như một thao tác biến từ cạnh hai chiều trở thành cạnh một chiều.

```
#define fi   first
#define se   second

#define MAX   300300
int numNode, numEdge; // số đỉnh và số cạnh
pair<int, int> edges[MAX]; // danh sách các cạnh
vector<int> adj[MAX]; // danh sách kề
// used là mảng đánh dấu cạnh đã dùng, bridge là mảng đánh dấu cạnh là cầu.
bool used[MAX], bridge[MAX];
int low[MAX], num[MAX], cnt;

// đọc đồ thị
scanf(“%d%d”, &numNode, &numEdge);
for (int i = 1; i <= numEdge; i++) { // đọc vào các cạnh
      int u, v; scanf(“%d%d”, &u, &v);
      edges[i] = make_pair(u, v);
      // lưu ý, danh sách kề không lưu các đỉnh kề với nó mà lưu chỉ số của các cạnh kề
      // với nó.
      adj[u].push_back(i);
      adj[v].push_back(i);
}

void dfs(int u, int p) {
      low[u] = numNode + 1;
      num[u] = ++cnt;

      for (int id : adj[u]) if (!used[id]) { // xét các cạnh kề với u mà chưa được sử dụng
            used[id] = true; // đánh dấu cạnh (u, v) là đã sử dụng
            // lấy ra đỉnh còn lại trong hai đỉnh kề với u
            int v = edges[id].fi + edges[id].se - u.
            if (num[v] == 0) {
                  dfs(v);
                  minimize(low[u], low[v]);
                  if (low[v] > num[u]) bridge[id] = true;
            } else minimize(low[u], num[v]);
      }
}
```

```
for (int i = 1; i <= numNode; i++) dfs(i);
```

Chú ý, ở một số tài liệu, trong trường hợp này thay vì dùng mảng đánh dấu cạnh đã thăm, người ta làm thao tác: Nếu v là cha của u thì bỏ qua v. Cách làm này là sai vì giữa u và v có thể có nhiều cạnh nối, trong trường hợp có nhiều cạnh nối, cách làm này khiến cho 1 cạnh không được xét lần nào thay vì mỗi cạnh được xét 1 lần, do đó, thay vì thấy tất cả các cạnh (u, v) đều không phải là cầu, thuật toán tìm ra một cạnh là cầu.

Sau khi đã biết cạnh (u, v) chắc chắn là cầu, câu hỏi tiếp theo sẽ là: Nếu xóa cạnh (u, v) các TPLT bị tách ra có dạng như thế nào. Như ta đã biết, nếu cạnh (u, v) là cầu, chắc chắn cạnh (u, v) thuộc cây DFS. Do một cạnh trên cây luôn nối giữa một cha và một con, ta có thể giả sử u là cha của v trên cây DFS. Khi đó, việc xóa cạnh (u, v) ra khỏi đồ thị cũng giống như xóa trên cây: Đồ thị/TPLT bị tách ra làm đôi (số TPLT của đồ thị luôn luôn tăng 1). Hai TPLT có dạng: cây con gốc v và phần bù cây con gốc v.

Bài toán tìm khớp

Cách tiếp cận và giải bài toán tìm khớp giống hệt với việc giải bài toán tìm cầu. Tuy nhiên, ta cần chú ý một số sự khác biệt giữa việc xóa đỉnh và xóa cạnh để tìm ra các sự khác biệt này.

Ta sẽ tiếp cận bài toán tìm khớp thông qua việc “biến đổi" một số nhận xét mà ta đã có ở bài toán tìm cầu.

Nhận xét 1: Để là cầu, một cạnh phải thuộc cây DFS. Nếu một cạnh không thuộc cây DFS, nó chắc chắn không phải là cầu
Biến đổi: Để một đỉnh là khớp, đỉnh này phải thuộc cây DFS. Nhận xét này hơi vô dụng đỉnh nào cũng thuộc cây DFS do thuật toán DFS kiểu gì cũng thăm hết tất cả các đỉnh.

Nhận xét 2: Nếu không có các cạnh “phá hoại" (nếu không có các cạnh không thuộc cây DFS), mọi cạnh của đồ thị đều là cầu, bởi khi đó, đồ thị biến thành cây.
Biến đổi: Nếu không có các cạnh “phá hoại", đồ thị cũng trở thành cây. Tuy nhiên, ngay cả trên, không phải đỉnh nào cũng là khớp. Một đỉnh chỉ là khớp khi nó có bậc > 1. Như vậy, có hai loại đỉnh trên cây không phải là khớp: Lá và gốc (nếu gốc có bậc là 1). Mọi đỉnh loại khác (khác lá và gốc có bậc > 1) thì đều là khớp.

Nhận xét 3: Sự xuất hiện của các cạnh “phá hoại" khiến cho các cạnh thuộc cây DFS không còn là cầu nữa.
Biến đổi: Sự xuất hiện của các cạnh “phá hoại" khiến cho một đỉnh trên đồ thị không thể là khớp.

Nhận xét 4: (nhắc lại) Một cạnh không thuộc cây DFS luôn nối giữa một tổ tiên với một con cháu của nó.

Nhận xét 5: (cơ chế “phá hoại" của các cạnh không thuộc cây DFS). So với việc “phá game" (làm cho một cạnh không phải là cầu), việc khiến một đỉnh không phải là khớp khó khăn hơn rất nhiều. Cụ thể, khó khăn hơn ở hai điểm sau đây:
- Việc xóa đi một cạnh (u, v) (với u là cha của v trên cây DFS) chỉ khiến đồ thị bị tách ra làm hai phần (cây con gốc v bị tách ra khỏi đồ thị). Do đó, để khiến cạnh (u, v) không phải là cầu, ta chỉ cần một cạnh màu đen nối từ bên trong cây con gốc v ra bên ngoài là được. Tuy nhiên, một đỉnh u có thể có nhiều con trên cây DFS (ta gọi các con là v1 v2 ... vk); khi đó, việc xóa đỉnh u khỏi đồ thị sẽ khiến các cây con gốc v1, cây con gốc v2, ... cây con gốc vk đều bị tách rời nhau ra và tách rời khỏi phần còn lại của đồ thị. Do đó, để làm cho u đỉnh u không còn là khớp, ta cần những cạnh màu đen (cạnh không thuộc cây DFS) nối từ bên trong mỗi cây con gốc v1 v2 ... vk ra ngoài. Chỉ cần một cây con gốc Vi nào đó không có cạnh nối ra bên ngoài, chắc chắn khi đó u là khớp. Nhắc lại, do một cạnh “phá hoại" chỉ nối được từ một tổ tiên tới một con cháu, các cạnh “phá hoại" giúp cho u không phải là khớp bắt buộc phải nối từ bên trong **tất cả** các cây con gốc v1 v2 ... vk lên các tổ tiên của u.

- Để khiến một cạnh không phải là cầu, ta có thể nối từ bên trong cây con gốc v tới đỉnh u hoặc một tổ tiên thực sự của u; bởi khi xóa cạnh, đỉnh u vẫn còn, do đó, việc nối tới đỉnh u có thể “phá hoại" thành công. Tuy nhiên, để khiến đỉnh u không còn là khớp, việc nối cạnh tới đỉnh u là không hợp lý, vì khi xóa đỉnh u thì mọi cạnh kề với u cũng bị xóa theo. Do đó, các cạnh “phá hoại" đỉnh u bắt buộc phải nối trực tiếp tới một tổ tiên thực sự của u.

Từ nhận xét 5 ở trên, ta có điều kiện để đỉnh u là khớp là như sau: Giả sử v1 v2 ... vk là các con của u. Khi đó, u là khớp khi và chỉ khi tồn tại một nhánh con vi sao cho low[vi] >= num[u]. Nói cách khác, nếu với **mọi** vi, ta luôn có low[vi] < num[u], khi đó u không còn là khớp.

Điều kiện trên là rút ra được từ nhận xét “làm sao để “phá hoại" khiến một đỉnh không phải là khớp". Tuy nhiên, trong nhận xét 2, ta thấy có một loại đỉnh “tự phá hoại" (kể cả khi không có cạnh nào không thuộc cây DFS, nó vẫn không phải là khớp). Có hai trường hợp “tự phá hoại": lá và gốc chỉ có 1 con. Với trường hợp lá, do điều kiện ở trên “là khớp khi tồn tại một cây con vi ...”, ta thấy: nếu đỉnh u là lá, đỉnh u không có con, và do đó, chắc chắn đỉnh u không thỏa mãn điều kiện này. Như vậy, điều kiện trên tự bao hàm trường hợp tự phá hoại là lá. Còn trường hợp “tự phá hoại" là gốc của cây chỉ có 1 con, ta phải xét riêng: đếm số con của gốc, nếu số con = 1, xoá đánh dấu.

Ở trong bài toán tìm khớp, ta vẫn có các mảng low và num với ý nghĩa hoàn toàn giống hệt như trong bài toán tìm cầu.

```
#define fi   first
#define se   second

#define MAX   300300
int numNode, numEdge; // số đỉnh và số cạnh
pair<int, int> edges[MAX]; // danh sách các cạnh
vector<int> adj[MAX]; // danh sách kề
// used là mảng đánh dấu cạnh đã dùng, bridge là mảng đánh dấu cạnh là cầu.
bool used[MAX], bridge[MAX];
int low[MAX], num[MAX], cnt;
int numChild[MAX]; // numChild[u] = số con của đỉnh u trên cây DFS
bool isCut[MAX]; // isCut là mảng đánh dấu đỉnh là khớp

// đọc đồ thị
scanf(“%d%d”, &numNode, &numEdge);
for (int i = 1; i <= numEdge; i++) { // đọc vào các cạnh
      int u, v; scanf(“%d%d”, &u, &v);
      edges[i] = make_pair(u, v);
      // lưu ý, danh sách kề không lưu các đỉnh kề với nó mà lưu chỉ số của các cạnh kề
      // với nó.
      adj[u].push_back(i);
      adj[v].push_back(i);
}

void dfs(int u) {
      low[u] = numNode + 1;
      num[u] = ++cnt;

      for (int id : adj[u]) if (!used[id]) { // xét các cạnh kề với u mà chưa được sử dụng
            used[id] = true; // đánh dấu cạnh (u, v) là đã sử dụng
```

```
            // lấy ra đỉnh còn lại trong hai đỉnh kề với u
            int v = edges[id].fi + edges[id].se - u.
            if (num[v] == 0) {
                  numChild[u]++; // v là con của u, nên số con của u tăng 1
                  dfs(v);
                  minimize(low[u], low[v]);
                  if (low[v] > num[u]) bridge[id] = true;
                  if (low[v] >= num[u]) isCut[u] = true;
            } else minimize(low[u], num[v]);
      }
}

for (int i = 1; i <= numNode; i++) {
      dfs(i); // khi một đỉnh có hàm dfs được gọi qua đây, nó là gốc của cây dfs
      if (numChild[i] == 1) isCut[i] = false; // gốc chỉ có 1 con thì không phải là khớp.
}
```

Chú ý: Với bài toán tìm khớp, ta không nhất thiết phải đánh dấu lại các cạnh đã sử dụng. Sở dĩ như vậy là vì ở bài toán tìm cầu, khi u là cha của v, ta cần ngăn chặn việc num[u] được gán vào low[v] khi chỉ có duy nhất một cạnh giữa u và v, để điều kiện low[v] > num[u] có thể xảy ra. Tuy nhiên, với bài toán tìm khớp, do điều kiện là low[v] >= num[u], việc gán num[u] cho low[v] không làm mất tính khớp của đỉnh, nên ta không bắt buộc phải ngăn chặn nó. Nhưng kể cả có đánh dấu cạnh thì cũng không sao cả.