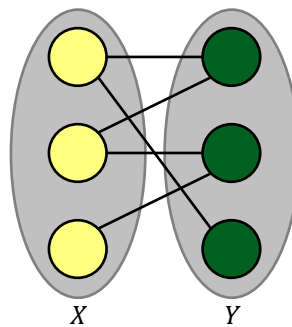


Chương 1. Bộ ghép cực đại trên đồ thị hai phía

1.1. Đồ thị hai phía

Đồ thị vô hướng $G = (V, E)$ được gọi là đồ thị hai phía nếu tập đỉnh V của nó có thể chia làm hai tập con rời nhau: X và Y sao cho mọi cạnh của đồ thị đều nối một đỉnh thuộc X với một đỉnh thuộc Y . Khi đó người ta còn ký hiệu $G = (X \cup Y, E)$. Để thuận tiện trong trình bày, ta gọi các đỉnh thuộc X là các X _đỉnh và các đỉnh thuộc Y là các Y _đỉnh.



Hình 1-1. Đồ thị hai phía

Một đồ thị vô hướng là đồ thị hai phía nếu và chỉ nếu từng thành phần liên thông của nó là đồ thị hai phía. Để kiểm tra một đồ thị vô hướng liên thông có phải đồ thị hai phía hay không, ta có thể sử dụng một thuật toán tìm kiếm trên đồ thị (BFS hoặc DFS) bắt đầu từ một đỉnh s bất kỳ. Đặt:

$X = \{\text{tập các đỉnh đến được từ } s \text{ qua một số chẵn cạnh}\}$

$Y = \{\text{tập các đỉnh đến được từ } s \text{ qua một số lẻ cạnh}\}$

Nếu tồn tại cạnh của đồ thị nối hai đỉnh $\in X$ hoặc hai đỉnh $\in Y$ thì đồ thị đã cho không phải đồ thị hai phía, ngược lại đồ thị đã cho là đồ thị hai phía với cách phân hoạch tập đỉnh thành hai tập X, Y ở trên.

Đồ thị hai phía gặp rất nhiều mô hình trong thực tế. Chẳng hạn mô hình phân công giữa những người thợ và những công việc, việc sinh viên chọn trường, thầy giáo chọn tiết dạy trong thời khoá biểu v.v...

1.2. Bài toán

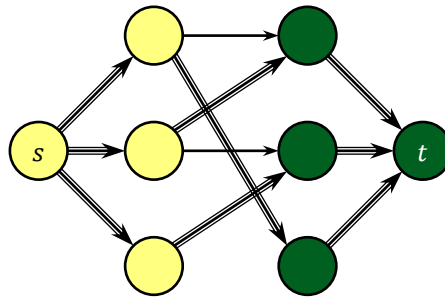
Cho đồ thị hai phía $G = (X \cup Y, E)$. Một *bộ ghép* (*matching*) của G là một tập các cạnh đôi một không có đỉnh chung. Có thể coi một bộ ghép là một tập $M \subseteq E$ sao cho trên đồ thị $(X \cup Y, M)$, mỗi đỉnh có bậc không quá 1.

Vấn đề đặt ra là tìm *một bộ ghép lớn nhất* (*maximum matching*) (có nhiều cạnh nhất) trên đồ thị hai phía cho trước.

1.3. Mô hình luồng

Ta xây dựng mạng G' từ đồ thị G bằng cách định hướng các cạnh của G thành cung từ X sang Y . Thêm vào đỉnh phát s và các cung nối từ s tới các X _đỉnh, thêm vào đỉnh thu t và các cung nối từ các Y _đỉnh tới t , sức chứa của tất cả các cung được đặt bằng 1.

Xét một luồng trên mạng G' có luồng trên các cung là số nguyên, khi đó có thể thấy rằng những cung có luồng bằng 1 từ X sang Y sẽ tương ứng với một bộ ghép trên G . Bài toán tìm bộ ghép cực đại trên G có thể giải quyết bằng cách tìm luồng nguyên cực đại trên G' và chọn ra các cung mang luồng 1 nối từ X sang Y .



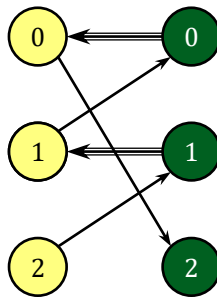
Hình 1-2. Mô hình luồng của bài toán bộ ghép cực đại trên đồ thị hai phía

Mô hình luồng và NEEDREF cho thấy rằng tất cả những thuật toán tìm luồng cực đại đã trình bày đều có thể áp dụng để tìm bộ ghép cực đại nếu luồng/tiền luồng được khởi tạo trên các cung là số nguyên (chẳng hạn luồng/tiền luồng 0). Ta sẽ phân tích một số đặc điểm của đường tăng luồng trong trường hợp này để tìm ra một cách cài đặt đơn giản hơn.

Xét đồ thị hai phía $G = (X \cup Y, E)$ và một bộ ghép M trên G .

- ✿ Những đỉnh thuộc M gọi là những *đỉnh đã ghép* (*matched vertices*), những đỉnh không thuộc M gọi là những *đỉnh chưa ghép* (*unmached vertices*).
- ✿ Những cạnh thuộc M gọi là những cạnh *đậm*, những cạnh không thuộc M được gọi là những cạnh *nhạt*.
- ✿ Nếu định hướng lại những cạnh của đồ thị thành cung: Những cạnh *nhạt* định hướng từ X sang Y , những cạnh *đậm* định hướng ngược lại từ Y về X . Trên đồ thị định hướng đó, một đường đi được gọi là *đường pha* (*alternating path*) và một đường đi từ một X _đỉnh chưa ghép tới một Y _đỉnh chưa ghép gọi là một *đường tăng* (*augmenting path*).

Dọc trên một đường pha, các cạnh đã ghép và chưa ghép xen kẽ nhau. Đường tăng là một đường pha đi qua một số lẻ cạnh, trong đó số cạnh *nhạt* nhiều hơn số cạnh *đậm* đúng một cạnh.



Hình 1-3. Đồ thị hai phía và các cạnh được định hướng theo bộ ghép

Hình 1-3 là ví dụ về một đồ thị hai phía với bộ ghép $\{(x_0, y_0), (x_1, y_1)\}$. Trên đồ thị này, đường đi $\langle x_2, y_1, x_1, y_0 \rangle$ là một đường pha, đường đi $\langle x_2, y_1, x_1, y_0, x_0, y_2 \rangle$ là một đường tăng.

1.4. Thuật toán đường tăng

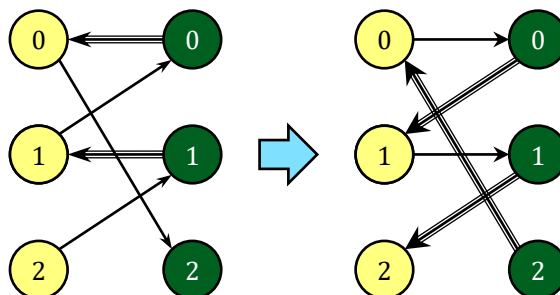
1.4.1. Ý tưởng của thuật toán

Đường tăng thực chất là đường tăng luồng với dư lượng 1 trên mô hình luồng. NEEDREF đã chỉ ra rằng điều kiện cần và đủ để một bộ ghép M là bộ ghép cực đại là không tồn tại đường tăng ứng với M .

Nếu tồn tại đường tăng P ứng với bộ ghép M , ta mở rộng bộ ghép bằng cách: dọc trên đường P loại bỏ những cạnh đậm khỏi M và thêm những cạnh nhạt vào M . Bộ ghép mới thu được sẽ có lực lượng nhiều hơn bộ ghép cũ đúng một cạnh. Đây thực chất là phép tăng luồng dọc trên đường P trên mô hình luồng.

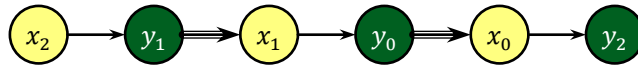
Thuật toán khởi tạo một bộ ghép bất kỳ trước khi bước vào vòng lặp chính. Tại mỗi bước lặp, đường tăng (thực chất là một đường đi từ một X _đỉnh chưa ghép tới một Y _đỉnh chưa ghép) được tìm bằng BFS hoặc DFS và bộ ghép sẽ được mở rộng dựa trên đường tăng tìm được.

```
M = «Một bộ ghép bất kỳ, chẳng hạn  $\emptyset$ »;
while (Tìm được đường tăng P)
  «Dọc trên đường P:
    - Loại bỏ những cạnh đậm khỏi M
    - Thêm những cạnh nhạt vào M
  »
```



Hình 1-4. Mở rộng bộ ghép

Ví dụ với đồ thị trong Hình 1-5 và bộ ghép $M = \{(x_0, y_0), (x_1, y_1)\}$, thuật toán sẽ tìm được đường tăng:



Dọc trên đường tăng này, ta loại bỏ hai cạnh đậm (y_1, x_1) và (y_0, x_0) khỏi bộ ghép và thêm vào bộ ghép ba cạnh nhạt (x_2, y_1) , (x_1, y_0) , (x_0, y_2) , được bộ ghép mới 3 cạnh. Đồ thị với bộ ghép mới không còn đường tăng (do không còn đỉnh chưa ghép) nên đây chính là bộ ghép cực đại.

1.4.2. Cài đặt

Ta sẽ cài đặt thuật toán tìm bộ ghép cực đại trên đồ thị hai phía $G = (X \cup Y, E)$, trong đó $|X| = nx$, $|Y| = ny$ và $|E| = m$. Các X _đỉnh được đánh số từ 0 tới $nx - 1$ và các Y _đỉnh được đánh số từ 0 tới $ny - 1$.

Khuôn dạng Input/Output như sau:

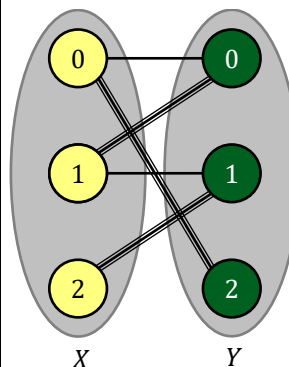
Input

- ✿ Dòng 1 chứa ba số nguyên dương nx, ny, m lần lượt là số X _đỉnh, số Y _đỉnh và số cạnh của đồ thị hai phía. ($nx, ny \leq 10^5; m \leq 10^6$).
- ✿ m dòng tiếp theo, mỗi dòng chứa hai số nguyên dương i, j tương ứng với một cạnh (x_i, y_j) của đồ thị.

Output

Bộ ghép cực đại trên đồ thị.

Sample Input	Sample Output
3 3 5	x[1] - y[0]
0 0	x[2] - y[1]
0 2	x[0] - y[2]
1 0	
1 1	
2 1	



✿ Biểu diễn đồ thị hai phía và bộ ghép

Đồ thị hai phía $G = (X \cup Y, E)$ sẽ được biểu diễn bằng cách danh sách kề của các X _đỉnh. Cụ thể là mỗi đỉnh $x \in X$ sẽ được cho tương ứng với một danh sách $adj[x]$ gồm chỉ số các Y _đỉnh kề với x .

Bộ ghép trên đồ thị hai phía được biểu diễn bởi mảng $match[0 \dots ny - 1]$, trong đó $match[j]$ là chỉ số của X _đỉnh ghép với đỉnh y_j . Nếu y_j là đỉnh chưa ghép, ta gán $match[j]$ bằng một giá trị đặc biệt (chẳng hạn $NoMatch = -1$).

✱ Tìm đường tăng

Đường tăng thực chất là một đường đi từ một X _đỉnh chưa ghép tới một Y _đỉnh chưa ghép trên đồ thị đã định hướng. Ta sẽ tìm đường tăng tại mỗi bước bằng thuật toán DFS:

Bắt đầu từ một đỉnh $x \in X$ chưa ghép, trước hết ta đánh dấu các Y _đỉnh bằng mảng số nguyên $lab[0 \dots ny)$ trong đó $lab[j] \neq visited$ nếu đỉnh $y_j \in Y$ chưa thăm và $lab[j] == visited$ nếu đỉnh $y_j \in Y$ đã thăm (chỉ cần đánh dấu các Y _đỉnh). Lý do ta không dùng kiểu đánh dấu true/false là vì thuật toán DFS sẽ phải thực hiện nhiều lần, nếu mỗi lần nếu phải khởi tạo lại cả mảng đánh dấu sẽ khá tốn thời gian. Ta dùng mảng số nguyên để khi muốn khởi tạo các Y _đỉnh chưa thăm chỉ cần gán một giá trị *visited* khác với mọi phần tử trong mảng $lab[0 \dots ny)$ là đủ.

Thuật toán DFS để tìm đường tăng xuất phát từ x được thực hiện bằng một hàm đệ quy $DFS(x)$, hàm này sẽ quét tất cả những đỉnh $y \in Y$ chưa thăm nối từ x (dĩ nhiên qua một cạnh nhát), với mỗi khi xét đến một đỉnh $y \in Y$, trước hết ta đánh dấu thăm y ($lab[y] = visited$). Sau đó:

- ✱ Nếu y đã ghép, dựa vào sự kiện từ y chỉ đi đến được $match[y]$ qua một cạnh đã ghép hướng từ Y về X , thuật toán DFS tiếp tục với lời gọi $Visit(match[y])$ để thăm luôn đỉnh $match[y] \in X$ (thăm liền hai bước).
- ✱ Ngược lại nếu y chưa ghép, tức là thuật toán DFS tìm được đường tăng kết thúc ở y , ta thoát khỏi dây chuyền đệ quy. Quá trình thoát dây chuyền đệ quy thực chất là lần ngược đường tăng, ta sẽ lợi dụng quá trình này để mở rộng bộ ghép dựa trên đường tăng.

Để thuật toán hoạt động hiệu quả hơn, ta sử dụng liên tiếp các pha xử lý lô: Ký hiệu S là tập các X _đỉnh chưa ghép, mỗi pha sẽ cố gắng mở rộng bộ ghép dựa trên không chỉ một mà nhiều đường tăng không có đỉnh chung xuất phát từ các đỉnh khác nhau $\in S$. Cụ thể là một pha bắt đầu với các Y _đỉnh chưa thăm, sau đó quét tất cả những đỉnh $x \in S$, với mỗi x đó thử tìm đường tăng xuất phát từ x và mở rộng bộ ghép nếu tìm ra đường tăng. Trong một pha có thể có nhiều X _đỉnh được ghép thêm, những X _đỉnh đã ghép sẽ được loại khỏi S ngay lập tức và thuật toán sẽ kết thúc khi trong một pha không có X _đỉnh nào được ghép thêm.

```



bool found; //Biến toàn cục với vai trò cờ báo đã tìm ra đường tăng

void Visit(x∈X) //Thuật toán DFS
{
    for (∀y ∈ adj[x])
        if (lab[y] ≠ visited) //y chưa thăm
        { //Thuật toán đảm bảo chắc chắn cạnh (x, y) là cạnh nhạt
            lab[y] = visited; //Đánh dấu thăm y
            if (match[y] == NoMatch) //y chưa ghép
                found = true; //Dựng cờ báo tìm thấy đường tăng
            else
                Visit(match[y]); //y đã ghép, gọi đệ quy tiếp tục DFS
            if (found) // ngay khi đường tăng được tìm thấy
            {
                match[y] = x; //Chỉnh lại bộ ghép theo đường tăng, cạnh (x, y) trở thành đậm
                return; //Thoát luôn, lệnh return đặt ở đây sẽ thoát cả dây chuyền đệ quy
            }
        }
}

//Thuật toán tìm bộ ghép cực đại trên đồ thị hai phía
«Khởi tạo một bộ ghép bất kỳ, chẳng hạn ∅»;
S = «Các X_đỉnh chưa ghép»;
lab[0 ... ny) = 0; //Nhãn đánh dấu được khởi tạo bằng 0
visited = 0; //Giá trị nhãn được coi là đã thăm
do
{
    oldsize = S.size(); //Lưu số đỉnh chưa ghép khi bắt đầu pha
    ++visited; //Đánh dấu mọi Y_đỉnh chưa thăm: lab[0...ny) ≠ visited
    for (∀x∈S)
    {
        found = false; //Cờ báo chưa tìm thấy đường tăng
        Visit(x); //Tìm đường tăng bằng DFS
        if (found) «x đã được ghép, loại bỏ x khỏi L»;
    }
}
while (oldsize != L.size()); //Lặp lại nếu có thêm đỉnh được ghép trong pha

```

Tập S chứa các X _đỉnh chưa ghép trong chương trình này được cài đặt bằng một mảng động (vector). Trong một pha, khi duyệt các đỉnh $x \in S$ ta duyệt từ cuối vector lên đầu, nếu tìm được đường tăng xuất phát từ x và mở rộng bộ ghép theo đường tăng đó, ta loại bỏ x khỏi S bằng cách đưa phần tử cuối vector S vào thế chỗ và hủy phần tử cuối vector S (pop_back).

 **BMATCH.CPP** ✓ Tìm bộ ghép cực đại trên đồ thị hai phía 

```

#include <iostream>
#include <vector>
using namespace std;
const int maxN = 1e5;
const int NoMatch = -1;
int nx, ny, m; //Số X_đỉnh, số Y_đỉnh và số cạnh
int match[maxN]; //match[y] = chỉ số X_đỉnh ghép với y ∈ Y, = NoMatch nếu y chưa ghép
int visited, lab[maxN]; //đánh dấu: lab[y] ≠ visited ↔ y chưa thăm
bool found; //Cờ báo tìm ra đường tăng

```

```

vector<int> adj[maxN]; //adj[x]: Danh sách chỉ số các Y_đỉnh kề với x
vector<int> S; //Danh sách các X_đỉnh chưa ghép

void Enter() //Nhập dữ liệu
{
    cin >> nx >> ny >> m;
    while (m-- > 0)
    {
        int x, y;
        cin >> x >> y;
        adj[x].push_back(y);
    }
}

void Init() //Khởi tạo bộ ghép rỗng
{
    fill(match, match + ny, NoMatch); //mọi Y_đỉnh chưa ghép
    S.reserve(nx);
    for (int x = 0; x < nx; ++x) S.push_back(x); //S chứa các X_đỉnh chưa ghép
}

```

```

void DFS(int x) //Thuật toán DFS từ x
{
    for (int y: adj[x]) //Xét các Y_đỉnh kề x
        if (lab[y] != visited) //y chưa thăm
        {
            lab[y] = visited; //Đánh dấu thăm y
            if (match[y] == NoMatch) //y chưa ghép
                found = true; //tìm ra đường tăng
            else
                DFS(match[y]); //DFS tiếp
            if (found) //Đã tìm ra đường tăng
            {
                match[y] = x; //Cho x ghép với y
                return; //Thoát dây chuyền đệ quy
            }
        }
}

```

```

void Solve()
{
    vector<int>::size_type oldsize;
    fill(lab, lab + ny, 0);
    visited = 0;
    do
    { //Một pha xử lý
        ++visited; //Các Y_đỉnh đều chưa thăm, visited ≠ lab[0..ny)
        oldsize = S.size(); //Lưu lại số lượng X_đỉnh chưa ghép
        for (int i = S.size() - 1; i >= 0; --i) //Duyệt ngược vector S
        {
            found = false; //Đặt cờ báo chưa tìm ra đường tăng
            DFS(S[i]); //Thử tìm đường tăng xuất phát từ S[i] và tăng cặp
            if (found) //Nếu tìm thấy, loại S[i] khỏi S
            {
                S[i] = S.back();
                S.pop_back();
            }
        }
    } while (oldsize != S.size());
}

```

```

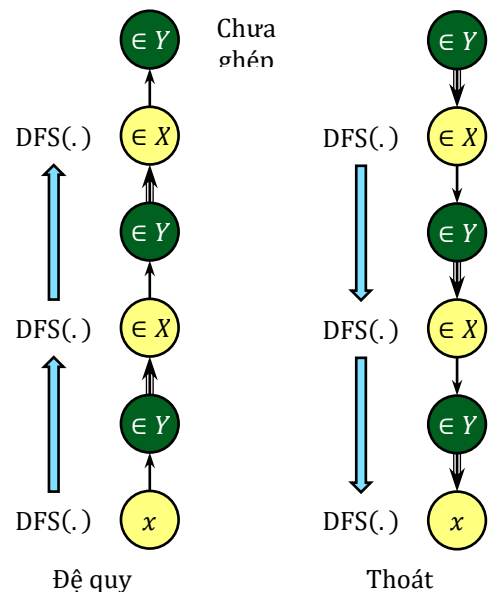
void Print() //In kết quả
{
    for (int y = 0; y < ny; ++y)
        if (match[y] != NoMatch)
            cout << "x[" << match[y] << "] - y[" << y << "] \n";
}

```

```

int main()
{
    Enter();
}

```



```

Init();
Solve();
Print();
}

```

1.4.3. Đánh giá

Nếu đồ thị có n đỉnh ($n = n_x + n_y$) và m cạnh, thời gian thực hiện của một pha sẽ bằng $O(n + m)$ (suy ra từ thời gian thực hiện giải thuật của DFS).

Các pha sẽ được thực hiện lặp cho tới khi một pha thực hiện xong mà không ghép thêm được đỉnh nào. Thuật toán cần không quá n_x pha, nên thời gian thực hiện giải thuật tìm bộ ghép cực đại trên đồ thị hai phía là $O(n^2 + nm)$ trong trường hợp xấu nhất.

Cần lưu ý rằng đây chỉ là đánh giá về cận trên thời gian thực hiện giải thuật, trên thực tế thuật toán này thực hiện rất nhanh trên các bộ dữ liệu ngẫu nhiên. Trường hợp xấu nhất cũng rất khó chỉ ra được khi mà ta có thể trộn ngẫu nhiên các đỉnh trong các danh sách kề cũng như các đỉnh trong tập S .

1.5. Thuật toán Hopcroft-Karp

1.5.1. Ý tưởng của thuật toán

Kỹ thuật tìm một lúc nhiều đường tăng rời nhau trình bày ở thuật toán trên thực chất chỉ là một mẹo cài đặt. Thuật toán Hopcroft-Karp [1] trình bày trong phần này mới thực sự là một cải tiến nhằm giảm độ phức tạp tính toán lý thuyết.

Thuật toán được lặp qua nhiều pha, tại mỗi pha:

- ✿ Xác định độ dài đường tăng ngắn nhất đi từ một X _đỉnh chưa ghép tới một Y _đỉnh chưa ghép, nếu không tồn tại đường tăng thì thuật toán dừng.
- ✿ Tìm một tập tối đại những đường tăng ngắn nhất rời nhau và mở rộng bộ ghép theo những đường tăng đó. Tính rời nhau ở đây nghĩa là các đường tăng trong tập đôi một không có đỉnh chung. Tính tối đại ở đây không có nghĩa là nhiều nhất, chỉ là nếu bổ sung thêm bất kỳ đường tăng ngắn nhất nào vào tập thì sẽ vi phạm tính rời nhau.

Các chi tiết kỹ thuật trong một pha:

- ✿ Thuật toán BFS được sử dụng để đo độ dài đường tăng ngắn nhất đi từ một X _đỉnh chưa ghép tới một Y _đỉnh chưa ghép, độ dài đường đi có thể tính bằng số cạnh nhạt (vì số cạnh đậm ít hơn số cạnh nhạt đúng 1 đơn vị). Với mỗi đỉnh $v \in X \cup Y$, thuật toán BFS tính luôn $level(v)$ là số cạnh nhạt trên đường pha ngắn nhất từ một X _đỉnh chưa ghép tới v .
- ✿ Việc tìm tập tối đại những đường tăng ngắn nhất rời nhau được thực hiện bằng DFS (tương tự như trong thuật toán trước, mục 1.4) chỉ có điều từ một đỉnh $x \in X$, ta duyệt danh sách kề của nó và chỉ xét những đỉnh $y \in Y$ mà $level(y) = level(x) + 1$. Điều này đảm bảo cho thuật toán DFS khi tìm ra đường tăng sẽ là đường tăng ngắn nhất tới một Y _đỉnh chưa ghép.

1.5.2. Cài đặt



Thuật toán Hopcroft-Karp sẽ được cài đặt với khuôn dạng Input/Output như trong thuật toán đường tăng. Cách tổ chức dữ liệu giống như trong thuật toán đường tăng, chỉ có thêm thao tác BFS để tính các giá trị $level[y]$ với $\forall y \in Y$. Chú ý rằng ta không lưu trữ các giá trị $level[x]$ với $\forall x \in X$.

Thuật toán DFS tiếp theo sẽ được thực hiện bởi:

for ($\forall x$ chưa ghép) DFS($x, 0$);

Vì ta không lưu trữ các giá trị $level[x \in X]$, hàm đệ quy DFS(x, lv) có nhiệm vụ tìm kiếm đường tăng theo chiều sâu bắt đầu từ x với $level(x) == lv$: Quét tất cả các đỉnh y chưa thăm kề với x mà $level[y] = lv + 1$. Nếu y chưa ghép thì tìm ra đường tăng ngắn nhất, thuật toán mở rộng bộ ghép theo đường tăng tìm được. Nếu y đã ghép, thuật toán sẽ tiếp tục với lời gọi đệ quy DFS($match[y], lv + 1$).

Thuật toán Hopcroft-Karp sử dụng cơ chế phân tầng cho các đỉnh khá giống thuật toán Dinic (NEEDREF) trong đó từ đỉnh $x \in X$ ta chỉ được phép thăm đỉnh $y \in Y$ nếu $level(y) == level(x) + 1$. Vì vậy để đánh dấu đỉnh $y \in Y$ đã thăm, ta không cần phải sử dụng một cơ chế đánh dấu riêng mà có thể gán luôn $level[y] = 0$.

 HOPCROFTKARP.CPP ✓ Thuật toán Hopcroft-Karp 

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
const int maxN = 1e5;
const int NoMatch = -1;
int nx, ny, m;
int match[maxN];
int level[maxN];
bool found;
vector<int> adj[maxN];
vector<int> S;

void Enter() //Nhập dữ liệu
{
    cin >> nx >> ny >> m;
    for (; m > 0; --m)
    {
        int x, y;
        cin >> x >> y;
        adj[x].push_back(y);
    }
}

void Init() //Khởi tạo bộ ghép rỗng
{
    fill(match, match + ny, NoMatch);
    S.reserve(nx);
    for (int x = 0; x < nx; ++x) S.push_back(x);
}

bool BFS() //Đo level[y] = độ dài đường pha ngắn nhất từ một X_đỉnh chưa ghép tới y
{
    queue<int> Q; //Hàng đợi dùng cho BFS
    fill(level, level + ny, 0); //level[.] = 0: Chưa thăm
    for (int x: S) //Xét mọi x ∈ X chưa ghép
        for (int y: adj[x]) //Xét mọi y ∈ Y kề x
            if (level[y] == 0) //y chưa thăm
            {
                level[y] = 1; //y đến được từ một X_đỉnh chưa ghép qua 1 cạnh nhạt
```

```

        Q.push(y);
    }
    while (!Q.empty()) //BFS
    {
        int ypop, x;
        ypop = Q.front(); Q.pop(); //Lấy ypop∈Y khỏi hàng đợi
        if ((x = match[ypop]) == NoMatch) //Đường tăng ngắn nhất qua level[ypop] cạnh nhạt
            return true; //dừng BFS, không cần đo tiếp những level[y] > level[ypop]
        for (int y: adj[x]) //BFS tiếp từ x = match[y]
            if (level[y] == 0) //Duyệt các y∈Y chưa thăm kề x
            {
                level[y] = level[ypop] + 1; //Gán nhãn level[.]
                Q.push(y);
            }
    }
    return false; //Không tồn tại đường tăng
}

void DFS(int x, int lv) //Tìm đường tăng theo chiều sâu bắt đầu từ x, level(x) = lv - 1
{
    for (int y: adj[x])
        if (level[y] == lv + 1) //Xét mọi y∈Y kề x chưa thăm và level[y] = lv + 1
        {
            level[y] = 0; //Đánh dấu y đã thăm
            if (match[y] == NoMatch) found = true; //Tìm ra đường tăng, dừng chờ báo.
            else DFS(match[y], lv + 1); //Tiếp tục DFS từ match[y] với level(match[y]) = lv + 1
            if (found) //Tìm ra đường tăng
            {
                match[y] = x; //Tăng cặp
                return; //Thoát dây chuyền đệ quy
            }
        }
}

void Solve() //Thuật toán Hopcroft-Karp
{
    while (BFS()) //Còn đường tăng, tìm các đường tăng ngắn nhất rời nhau và tăng cặp...
        for (int i = S.size() - 1; i >= 0; --i) //Xét các X_đỉnh chưa ghép
        {
            found = false;
            DFS(s[i], 0); //Thử tìm đường tăng từ s[i]
            if (found) //Tìm thấy đường tăng, loại s[i] khỏi S
            {
                S[i] = S.back();
                S.pop_back();
            }
        }
}

void Print() //In kết quả
{
    for (int y = 0; y < ny; ++y)
        if (match[y] != NoMatch)
            cout << "x[" << match[y] << "] - y[" << y << "]\n";
}

int main()
{
    Enter();
    Init();
    Solve();
    Print();
}

```

1.5.3. Đánh giá

Một số kết quả trong phần này có thể suy ra từ mô hình luồng của bài toán bộ ghép cực đại, tuy nhiên các chứng minh chỉ dựa vào mô hình đồ thị hai phía để làm rõ những tính chất riêng của bài toán. Hướng của các cung cũng không được quan tâm nữa, ta coi như đồ thị gồm các cạnh vô hướng.

Để tiện lợi cho việc trình bày các chứng minh, ta sử dụng một phép toán tập hợp là hiệu đối xứng (*symmetric difference*). Với A và B là hai tập hợp, hiệu đối xứng của A và B , ký hiệu $A \oplus B$, được định nghĩa bởi:

$$A \oplus B = (A - B) \cup (B - A)$$

là tập những phần tử chỉ thuộc A hoặc chỉ thuộc B .

Hiệu đối xứng có thể định nghĩa qua phép toán logic *hoặc loại trừ* (*eXclusive OR -XOR*) tương tự như phép hợp có thể định nghĩa qua phép tuyển logic hay phép giao có thể định nghĩa qua phép hội logic:

$$x \in A \cup B \Leftrightarrow (x \in A) \text{ or } (x \in B)$$

$$x \in A \cap B \Leftrightarrow (x \in A) \text{ and } (x \in B)$$

$$x \in A \oplus B \Leftrightarrow (x \in A) \text{ xor } (x \in B)$$

Dễ dàng suy ra được tính chất sau từ định nghĩa:

$$A \oplus B = C \Leftrightarrow A = B \oplus C$$

Với M là một bộ ghép và P là một đường tăng, nếu coi M và P là những tập cạnh thì phép tăng cặp ghép dựa vào đường P để được bộ ghép M' có thể viết thành:

$$M' = M \oplus P$$

Bổ đề 1-1

Xét đơn đồ thị vô hướng G mà bậc của mỗi đỉnh không quá 2, khi đó mỗi thành phần liên thông của G sẽ là đỉnh cô lập, đường đi đơn, hoặc chu trình đơn.

Chứng minh

Ban đầu đồ thị chưa có cạnh nào, mỗi đỉnh coi như một đường đi từ nó tới chính nó. Khi thêm một cạnh vào đồ thị, dựa vào giả thiết bậc mỗi đỉnh không quá 2, cạnh này sẽ chỉ nối hai đỉnh đầu mút của cùng một đường đi hoặc hai đường đi khác nhau. Nếu cạnh nối hai đầu mút của cùng một đường đi ta có một chu trình đơn, nếu cạnh nối hai đầu mút của hai đường đi khác nhau ta có một đường đi mới. Từ đó suy ra ĐPCM.

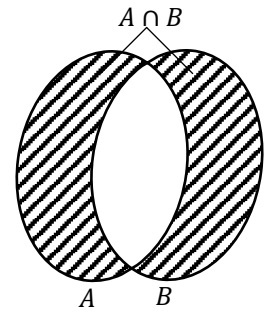
Bổ đề 1-2

Gọi M và N là hai bộ ghép trong đó $|M| < |N|$, khi đó $M \oplus N$ chứa ít nhất $|N| - |M|$ đường tăng ứng với bộ ghép M mà các đường tăng này đôi một không có đỉnh chung.

Chứng minh

Xét đồ thị $\bar{G} = (V, M \oplus N)$, vì M và N là các bộ ghép nên bậc của các đỉnh trong \bar{G} không quá 2 (mỗi đỉnh chỉ có tối đa hai cạnh liên thuộc, một cạnh $\in M$ và một cạnh $\in N$). Theo Bổ đề 1-1, các thành phần liên thông của \bar{G} sẽ thuộc một trong ba dạng:

- ⚙ Thành phần liên thông là đỉnh cô lập: Không chứa cạnh nào.



- ✿ Thành phần liên thông là chu trình đơn: Vì mỗi đỉnh có đúng hai cạnh liên thuộc, một cạnh $\in M$ và một cạnh $\in N$, chu trình đơn này chứa các cạnh $\in M$ và $\in N$ xen kẽ nhau. Do đó số cạnh $\in M$ và số cạnh $\in N$ trên chu trình là bằng nhau.
- ✿ Thành phần liên thông là đường đi đơn: Vì mỗi đỉnh có tối đa hai cạnh liên thuộc, một cạnh $\in M$ và một cạnh $\in N$, đường đi đơn này chứa các cạnh $\in M$ và $\in N$ xen kẽ nhau. Do đó số cạnh $\in M$ và số cạnh $\in N$ trên đường đi này hơn kém nhau tối đa 1 đơn vị.

Do $|M| < |N|$ nên có ít nhất $|N| - |M|$ thành phần liên thông là đường đi đơn mà số cạnh $\in N$ nhiều hơn số cạnh $\in M$ đúng 1 cạnh, mỗi thành phần liên thông này chính là một đường tăng đối với bộ ghép M , hiển nhiên những đường tăng đó đôi một không có đỉnh chung.

Bổ đề 1-3

Cho M là một bộ ghép, P là một đường tăng ngắn nhất ứng với M và P' là một đường tăng ứng với bộ ghép $M \oplus P$, khi đó:

$$|P'| \geq |P| + 2|P \cap P'|$$

(Trong ký hiệu này ta đồng nhất đường đi P và P' với tập cạnh của chúng)

Chứng minh

Gọi $N = M \oplus P \oplus P'$ khi đó $|N| = |M| + 2$. Bổ đề 1-2 cho biết $M \oplus N$ chứa hai đường tăng đối với bộ ghép M , gọi hai đường tăng đó là P_1 và P_2 . Mặt khác, từ biểu thức $N = M \oplus P \oplus P'$ ta có $N \oplus M = P \oplus P'$, suy ra:

$$|P \oplus P'| \geq |P_1| + |P_2|$$

Do P là đường tăng ngắn nhất ứng với bộ ghép M nên $|P_1| \geq |P|$ và $|P_2| \geq |P|$:

$$\Rightarrow |P \oplus P'| \geq 2|P|$$

$$\Rightarrow |P| + |P'| - 2|P \cap P'| \geq 2|P| \text{ (từ định nghĩa của hiệu đối xứng)}$$

$$\Rightarrow |P'| \geq |P| + 2|P \cap P'| \text{ (ĐPCM)}$$

Quá trình thuật toán Hopcroft-Karp thực hiện có thể tóm tắt qua lược đồ sau: Bắt đầu với bộ ghép $M_0 = \emptyset$, thuật toán xây dựng các bộ ghép $M_1, M_2, \dots, M_i, \dots$ bằng cách mở rộng bộ ghép tại mỗi bước bởi đường tăng ngắn nhất. Gọi P_i là một đường tăng ngắn nhất ứng với bộ ghép M_i và $M_{i+1} = M_i \oplus P_i$.

Bổ đề 1-3 chỉ ra rằng $|P_i| \leq |P_{i+1}|$ tức là độ dài đường tăng ngắn nhất sau mỗi bước mở rộng bộ ghép không giảm đi.

Bổ đề 1-4

Với $\forall i, j: i \neq j$ và $|P_i| = |P_j|$, thì P_i và P_j không có đỉnh chung.

Chứng minh

Giả sử kết luận này không đúng, chọn chỉ số j bé nhất thỏa mãn: $\exists i < j$ sao cho $|P_i| = |P_j|$ và P_i có đỉnh chung với P_j . Nếu có nhiều chỉ số i như vậy ta chọn chỉ số i lớn nhất. Khi đó các đường đi P_i, P_{i+1}, \dots, P_j có cùng độ dài theo Bổ đề 1-3:

$$|P_i| = |P_{i+1}| = \dots = |P_j|$$

Hơn nữa do cách chọn i và j , các đường đi $P_{i+1}, P_{i+2}, \dots, P_{j-1}$ đều không có đỉnh chung với nhau và cũng không có đỉnh chung với P_i cũng như P_j . Suy ra P_j là một đường tăng ứng với bộ ghép M_i . Cũng từ Bổ đề 1-3:

$$|P_j| \geq |P_i| + 2|P_i \cap P_j|$$

Ta có $|P_i \cap P_j| = \emptyset$ hay hai đường đi này không có cạnh chung. Tuy nhiên hai đường đi này có đỉnh chung, gọi là v , đỉnh v chắc chắn là đỉnh đã ghép đối với bộ ghép $M_i \oplus P_i$ nên cạnh đã đậm liên thuộc v trong bộ ghép này sẽ thuộc cả P_i và P_j . Mâu thuẫn này cho ta ĐPCM.

Định lý 1-5

Có thể cài đặt thuật toán Hopcroft-Karp chạy trong thời gian $O(\sqrt{n} \cdot m)$ với n là số đỉnh và m là số cạnh của đồ thị hai phía.

Chứng minh

Ta có thể coi như thuật toán thực hiện qua nhiều pha, mỗi pha có một lượt BFS và một lượt DFS thực hiện thời gian trong $O(n + m)$.

Độ dài đường tăng ngắn nhất tăng gấp đôi sau mỗi pha, điều này được chỉ ra trong Bổ đề 1-3: Độ dài đường tăng ngắn nhất không giảm đi sau mỗi pha và Bổ đề 1-4: Nếu độ dài đường tăng ngắn nhất không tăng lên ở pha sau thì đường tăng này đã được tìm thấy bởi thuật toán DFS ở pha trước.

Xét tình trạng sau \sqrt{n} pha đầu tiên, ta có một bộ ghép M mà độ dài mọi đường tăng ứng với bộ ghép đó đều $\geq \sqrt{n}$. Giả sử khi kết thúc tất cả các pha ta thu được bộ ghép cực đại M^* , khi đó theo Bổ đề 1-1 và Bổ đề 1-2, $M \oplus M^*$ tách thành các thành phần liên thông thuộc một trong ba dạng:

- ✿ Đỉnh cô lập
- ✿ Chu trình pha ứng với M
- ✿ Đường pha ứng với M

Xét những thành phần liên thông là đường tăng, không thể có quá \sqrt{n} thành phần liên thông như vậy (do mọi đường tăng đều chứa $> \sqrt{n}$ đỉnh và $M^* \oplus M$ chứa không quá n đỉnh). Vậy thì cùng lắm là chỉ thêm \sqrt{n} pha nữa, tất cả các đường tăng này sẽ được tìm và xử lý. Vậy tổng số pha cần thực hiện trong thuật toán là $O(\sqrt{n})$, tức là thuật toán Hopcroft-Karp thực hiện trong thời gian $O(\sqrt{n}(n + m))$.

Bài tập 1-1

Có p thợ và q việc. Mỗi thợ cho biết mình có thể làm được những việc nào, và mỗi việc khi giao cho một thợ thực hiện sẽ được hoàn thành xong trong đúng 1 đơn vị thời gian. Tại một thời điểm, mỗi thợ chỉ thực hiện không quá một việc.

Hãy phân công các thợ làm các công việc sao cho:

- ✳ Mỗi việc chỉ giao cho đúng một thợ thực hiện.
- ✳ Thời gian hoàn thành tất cả các công việc là nhỏ nhất. Chú ý là các thợ có thể thực hiện song song các công việc được giao, việc của ai người nấy làm, không ảnh hưởng tới người khác.

Bài tập 1-2

Một bộ ghép M trên đồ thị hai phía gọi là tối đại nếu việc bổ sung thêm bất cứ cạnh nào vào M sẽ làm cho M không còn là bộ ghép nữa.

- a) Chỉ ra một ví dụ về bộ ghép tối đại nhưng không là bộ ghép cực đại trên đồ thị hai phía
- b) Tìm thuật toán $O(|E|)$ để xác định một bộ ghép tối đại trên đồ thị hai phía
- c) Chứng minh rằng nếu A và B là hai bộ ghép tối đại trên cùng một đồ thị hai phía thì $|A| \leq 2|B|$ và $|B| \leq 2|A|$. Từ đó chỉ ra rằng nếu thuật toán đường tăng được khởi tạo bằng một bộ ghép tối đại thì số lượt tìm đường tăng giảm đi ít nhất một nửa so với việc khởi tạo bằng bộ ghép rỗng.

Bài tập 1-3 (Phủ đỉnh – Vertex Cover)

Cho đồ thị hai phía $G = (X \cup Y, E)$. Bài toán đặt ra là hãy chọn ra một tập C gồm ít nhất các đỉnh sao cho mọi cạnh $\in E$ đều liên thuộc với ít nhất một đỉnh thuộc C .

Bài toán tìm phủ đỉnh nhỏ nhất trên đồ thị tổng quát là NP-đầy đủ, hiện tại chưa có thuật toán đa thức để giải quyết. Tuy vậy trên đồ thị hai phía, phủ đỉnh nhỏ nhất có thể tìm được dựa trên bộ ghép cực đại.

Dựa vào mô hình luồng của bài toán bộ ghép cực đại, giả sử các cung (X, Y) có sức chứa $+\infty$, các cung (s, X) và (Y, t) có sức chứa 1. Gọi (S, T) là lát cắt hẹp nhất của mạng. Đặt $C = \{T \cap X\} \cup \{S \cap Y\}$.

- a) Chứng minh rằng C là một phủ đỉnh
- b) Chứng minh rằng C là phủ đỉnh nhỏ nhất
- c) Giả sử ta tìm được M là bộ ghép cực đại trên đồ thị hai phía, khi đó chắc chắn không còn tồn tại đường tăng tương ứng với bộ ghép M . Đặt:

$$Y^* = \{y \in Y: \exists x \in X \text{ chưa ghép, } x \text{ đến được } y \text{ qua một đường pha}\}$$

$$X^* = \{x \in X: x \text{ đã ghép và đỉnh ghép với } x \text{ không thuộc } Y^*\}$$

Chứng minh rằng $X^* \cup Y^*$ là phủ đỉnh nhỏ nhất.

Bài tập 1-4 (Tập độc lập cực đại)

Cho đồ thị hai phía $G = (X \cup Y, E)$. Bài toán đặt ra là hãy chọn ra một tập I gồm nhiều đỉnh sao cho hai đỉnh bất kỳ của I không kề nhau.

- Chứng minh rằng nếu I là tập độc lập cực đại thì $|I| = |X| + |Y| - |M|$ với $|M|$ là số cạnh của bộ ghép cực đại
- Xây dựng thuật toán tìm tập độc lập cực đại trên đồ thị hai phía (Gợi ý: Quy về bài toán tìm phủ đỉnh)

Bài tập 1-5

Cho M là một bộ ghép trên đồ thị hai phía $G = (X \cup Y, E)$. Gọi k là số X _đỉnh chưa ghép. Chứng minh rằng ba mệnh đề sau đây là tương đương:

- ✿ M là bộ ghép cực đại.
- ✿ G không có đường tăng tương ứng với bộ ghép M .
- ✿ Tồn tại một tập con A của X sao cho $|N(A)| = |A| - k$. Ở đây $N(A)$ là tập các Y _đỉnh kề với một đỉnh nào đó trong A (Gợi ý: Chọn A là tập các X _đỉnh đến được từ một X _đỉnh chưa ghép bằng một đường pha)

Bài tập 1-6 (Định lý Hall)

Cho $G = (X \cup Y, E)$ là đồ thị hai phía có $|X| = |Y|$. Chứng minh rằng G có bộ ghép đầy đủ (bộ ghép mà mọi đỉnh đều được ghép) nếu và chỉ nếu $|A| \leq |N(A)|$ với mọi tập $A \subseteq X$.

Bài tập 1-7 (Phủ đường tối thiểu)

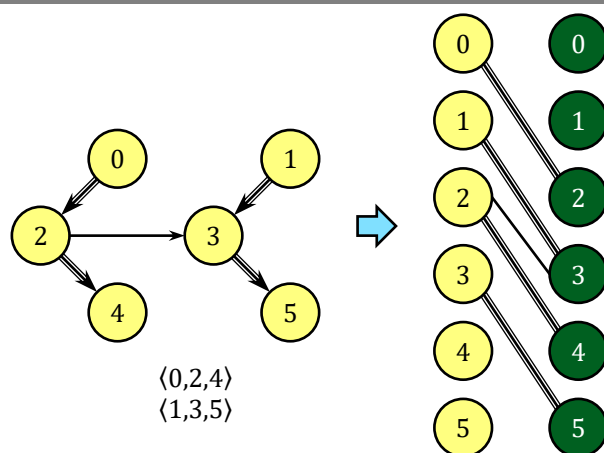
Cho $G = (V, E)$ là đồ thị có hướng không có chu trình. Một *phủ đường* (*path cover*) là một tập P các đường đi trên G thỏa mãn: Với mọi đỉnh $v \in V$, tồn tại duy nhất một đường đi trong P chứa v . Đường đi có thể bắt đầu và kết thúc ở bất cứ đâu, tính cả đường đi độ dài 0 (chỉ gồm một đỉnh). Bài toán đặt ra là tìm *phủ đường tối thiểu* (*minimum path cover*): Phủ đường gồm ít đường đi nhất.

Gọi n là số đỉnh của đồ thị, ta đánh số các đỉnh thuộc V từ 0 tới $n - 1$. Xây dựng đồ thị hai phía $G' = (X \cup Y, E')$ trong đó:

$$X = \{x_0, x_1, \dots, x_{n-1}\}$$

$$Y = \{y_0, y_1, \dots, y_{n-1}\}$$

Tập cạnh E' được xây dựng như sau: Với mỗi cung $(i, j) \in E$, ta thêm vào một cạnh $(x_i, y_j) \in E'$ (Hình 1-5)



Hình 1-5. Tìm phủ đường tối thiểu qua mô hình bộ ghép cực đại

Gọi M là một bộ ghép trên G' . Khởi tạo P là tập n đường đi, mỗi đường đi chỉ gồm một đỉnh trong G , khi đó P là một phủ đường. Xét lần lượt các cạnh của bộ ghép, mỗi khi xét tới cạnh (x_i, y_j) ta đặt cạnh (i, j) nối hai đường đi trong P thành một đường... Khi thuật toán kết thúc, P vẫn là một phủ đường.

a) Chứng minh tính bất biến vòng lặp: Tại mỗi bước khi xét tới cạnh $(x_i, y_j) \in M$, cạnh $(i, j) \in E$ chắc chắn sẽ nối hai đường đi trong P : một đường đi kết thúc ở i và một đường đi khác bắt đầu ở j . Từ đó chỉ ra tính đúng đắn của thuật toán. (Gợi ý: mỗi khi xét tới cạnh $(x_i, y_j) \in M$ và đặt cạnh (i, j) nối hai đường đi của P thành một đường thì $|P|$ giảm 1. Vậy khi thuật toán trên kết thúc, $|P| = n - |M|$, tức là muốn $|P| \rightarrow \min$ thì $|M| \rightarrow \max$).

b) Viết chương trình tìm phủ đường cực tiểu trên đồ thị có hướng không có chu trình.

c) Chỉ ra ví dụ để thấy rằng thuật toán trên không đúng trong trường hợp G có chu trình.

d) Chứng minh rằng nếu tìm được thuật toán giải bài toán tìm phủ đường cực tiểu trên đồ thị tổng quát trong thời gian đa thức thì có thể tìm được đường đi Hamilton trên đồ thị đó (nếu có) trong thời gian đa thức.

Bài tập 1-8 (Bộ ghép cực đại trên đồ thị chính quy hai phía)

Một đồ thị vô hướng $G = (V, E)$ gọi là đồ thị chính quy bậc k (k -regular graph) nếu bậc của mọi đỉnh đều bằng k . Đồ thị chính quy bậc 0 là đồ thị không có cạnh nào, đồ thị chính quy bậc 1 thì các cạnh tạo thành bộ ghép đầy đủ, đồ thị chính quy bậc 2 có các thành phần liên thông là các chu trình đơn.

a) Chứng minh rằng đồ thị hai phía $G = (X \cup Y, E)$ là đồ thị chính quy thì $|X| = |Y|$.

b) Chứng minh rằng luôn tồn tại bộ ghép đầy đủ trên đồ thị hai phía chính quy bậc k ($k \geq 1$).

c) Tìm thuật toán $O(|E| \log |E|)$ để xác định một bộ ghép đầy đủ trên đồ thị chính quy hai phía bậc $k \geq 1$.