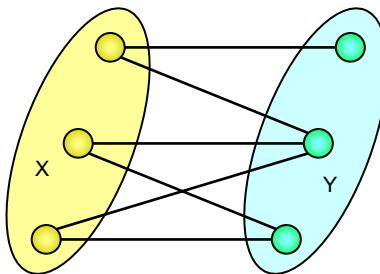


§11. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI TRÊN ĐỒ THỊ HAI PHÍA

11.1. ĐỒ THỊ HAI PHÍA (BIPARTITE GRAPH)

Các tên gọi đồ thị hai phía một dạng đơn đồ thị vô hướng $G = (V, E)$ mà tập đỉnh của nó có thể chia làm hai tập con X, Y rời nhau sao cho bất kỳ cạnh nào của đồ thị cũng nối một đỉnh của X với một đỉnh thuộc Y . Khi đó người ta còn ký hiệu G là $(X \cup Y, E)$ và gọi một tập (chẳng hạn tập X) là **tập các đỉnh trái** và tập còn lại (chẳng hạn tập Y) là **tập các đỉnh phải** của đồ thị hai phía G . Các đỉnh thuộc X còn gọi là các $X_đỉnh$, các đỉnh thuộc Y gọi là các $Y_đỉnh$.



Hình 85: Đồ thị hai phía

Để kiểm tra một đồ thị liên thông có phải là đồ thị hai phía hay không, ta có thể áp dụng thuật toán sau:

Với một đỉnh v bất kỳ:

```

 $X := \{v\}; Y := \emptyset;$ 
repeat
     $Y := Y \cup K_{\bar{e}}(X);$ 
     $X := X \cup K_{\bar{e}}(Y);$ 
until  $(X \cap Y \neq \emptyset)$  or  $(X \text{ và } Y \text{ là tối đại - không bổ sung được nữa});$ 
if  $X \cap Y \neq \emptyset$  then
    (Không phải đồ thị hai phía)
else
    (Đây là đồ thị hai phía,
     $X$  là tập các đỉnh trái: các đỉnh đến được từ  $v$  qua một số chẵn cạnh
     $Y$  là tập các đỉnh phải: các đỉnh đến được từ  $v$  qua một số lẻ cạnh);

```

Đồ thị hai phía gặp rất nhiều mô hình trong thực tế. Chẳng hạn quan hệ hôn nhân giữa tập những người đàn ông và tập những người đàn bà, việc sinh viên chọn trường, thầy giáo chọn tiết dạy trong thời khoá biểu v.v...

11.2. BÀI TOÁN GHÉP ĐÔI KHÔNG TRỌNG VÀ CÁC KHÁI NIỆM

Cho một đồ thị hai phía $G = (X \cup Y, E)$ ở đây X là tập các đỉnh trái và Y là tập các đỉnh phải của G . $X = \{x[1], x[2], \dots, x[m]\}$, $Y = \{y[1], y[2], \dots, y[n]\}$

Một bộ ghép (matching) của G là một tập hợp các cạnh của G đôi một không có đỉnh chung.

Bài toán ghép đôi (matching problem) là tìm một bộ ghép lớn nhất (nghĩa là có số cạnh lớn nhất) của G

Xét một bộ ghép M của G.

- ❖ Các đỉnh trong M gọi là các đỉnh đã ghép (matched vertices), các đỉnh khác là chưa ghép.
- ❖ Các cạnh trong M gọi là các cạnh đã ghép, các cạnh khác là chưa ghép
- ❖ Nếu định hướng lại các cạnh của đồ thị thành cung, những cạnh chưa ghép được định hướng từ X sang Y, những cạnh đã ghép định hướng từ Y về X. Trên đồ thị định hướng đó: Một đường đi xuất phát từ một X_đỉnh chưa ghép gọi là đường pha, một đường đi từ một X_đỉnh chưa ghép tới một Y_đỉnh chưa ghép gọi là đường mở.

Một cách dễ hiểu, có thể quan niệm như sau:

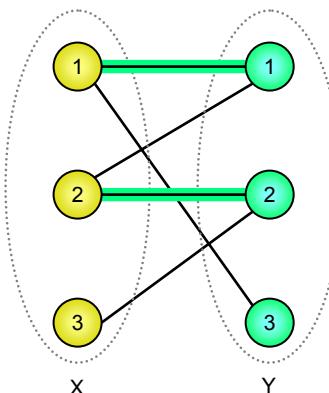
- ❖ Một đường pha (alternating path) là một đường đi đơn trong G bắt đầu bằng một X_đỉnh chưa ghép, đi theo một cạnh **chưa ghép** sang Y, rồi đến một cạnh **đã ghép** về X, rồi lại đến một cạnh **chưa ghép** sang Y... cứ xen kẽ nhau như vậy.
- ❖ Một đường mở (augmenting path) là một đường pha. Bắt đầu từ một X_đỉnh chưa ghép kết thúc bằng một Y_đỉnh chưa ghép.

Ví dụ: với đồ thị hai phía trong hình Hình 86 và bộ ghép $M = \{(x[1], y[1]), (x[2], y[2])\}$

$x[3]$ và $y[3]$ là những đỉnh chưa ghép, các đỉnh khác là đã ghép

Đường $(x[3], y[2], x[2], y[1])$ là đường pha

Đường $(x[3], y[2], x[2], y[1], x[1], y[3])$ là đường mở.



Hình 86: Đồ thị hai phía và bộ ghép M

11.3. THUẬT TOÁN ĐƯỜNG MỞ

Thuật toán đường mở để tìm một bộ ghép lớn nhất phát biểu như sau:

Bước 1:

Bắt đầu từ một bộ ghép bất kỳ M (thông thường bộ ghép được khởi gán bằng bộ ghép rỗng hay được tìm bằng các thuật toán tham lam)

Bước 2:

Tìm một đường mở

Bước 3:

- ❖ Nếu bước 2 tìm được đường mở thì mở rộng bộ ghép M: Trên đường mở, loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép. Sau đó lặp lại bước 2.
- ❖ Nếu bước 2 không tìm được đường mở thì thuật toán kết thúc

(Khởi tạo một bộ ghép M);

```
while (Có đường mở xuất phát từ x tới một đỉnh y chưa ghép ∈ Y) do
    {Đọc trên đường mở, xoá bỏ khỏi M các cạnh đã ghép và thêm vào M những cạnh chưa ghép};
    {Sau thao tác này, đỉnh x và y trở thành đã ghép, số cạnh đã ghép tăng lên 1}
```

Như ví dụ trên, với bộ ghép hai cạnh $M = \{(x[1], y[1]), (x[2], y[2])\}$ và đường mở tìm được gồm các cạnh:

$$(x[3], y[2]) \notin M$$

$$(y[2], x[2]) \in M$$

$$(x[2], y[1]) \notin M$$

$$(y[1], x[1]) \in M$$

$$(x[1], y[3]) \notin M$$

Vậy thì ta sẽ loại đi các cạnh $(y[2], x[2])$ và $(y[1], x[1])$ trong bộ ghép cũ và thêm vào đó các cạnh $(x[3], y[2]), (x[2], y[1]), (x[1], y[3])$ được bộ ghép 3 cạnh.

11.4. CÀI ĐẶT

11.4.1. Biểu diễn đồ thị hai phía

Giả sử đồ thị hai phía $G = (X \cup Y, E)$ có các X _đỉnh ký hiệu là $x[1], x[2], \dots, x[m]$ và các Y _đỉnh ký hiệu là $y[1], y[2], \dots, y[n]$. Ta sẽ biểu diễn đồ thị hai phía này bằng ma trận A cỡ $m \times n$. Trong đó:

$$A[i, j] = \text{TRUE} \Leftrightarrow \text{có cạnh nối đỉnh } x[i] \text{ với đỉnh } y[j].$$

11.4.2. Biểu diễn bộ ghép

Để biểu diễn bộ ghép, ta sử dụng hai mảng: $\text{matchX}[1..m]$ và $\text{matchY}[1..n]$.

- ❖ $\text{matchX}[i]$ là chỉ số của đỉnh thuộc tập Y ghép với đỉnh $x[i]$
- ❖ $\text{matchY}[j]$ là chỉ số của đỉnh thuộc tập X ghép với đỉnh $y[j]$.

Tức là nếu như cạnh $(x[i], y[j])$ thuộc bộ ghép thì $\text{matchX}[i] = j$ và $\text{matchY}[j] = i$.

Quy ước rằng:

- ❖ Nếu như $x[i]$ chưa ghép với đỉnh nào của tập Y thì $\text{matchX}[i] = 0$
- ❖ Nếu như $y[j]$ chưa ghép với đỉnh nào của tập X thì $\text{matchY}[j] = 0$.

Suy ra

- ❖ Thêm một cạnh $(x[i], y[j])$ vào bộ ghép \Leftrightarrow Đặt $\text{matchX}[i] := j$ và $\text{matchY}[j] := i$;
- ❖ Loại một cạnh $(x[i], y[j])$ khỏi bộ ghép \Leftrightarrow Đặt $\text{matchX}[i] := 0$ và $\text{matchY}[j] := 0$;

11.4.3. Tìm đường mở như thế nào.

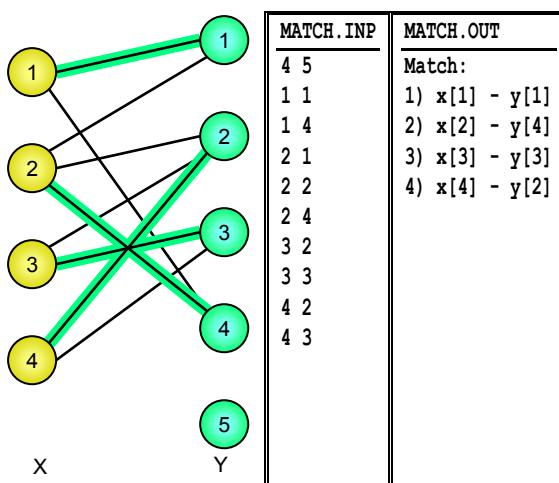
Vì đường mở bắt đầu từ một $X_{\text{đỉnh}}$ chưa ghép, đi theo một cạnh chưa ghép sang tập Y , rồi theo một đã ghép để về tập X , rồi lại một cạnh chưa ghép sang tập $Y \dots$ **cuối cùng là cạnh chưa ghép** tới một $Y_{\text{đỉnh}}$ chưa ghép. Nên có thể thấy ngay rằng độ dài đường mở là lẻ và trên đường mở số cạnh $\in M$ ít hơn số cạnh $\notin M$ là 1 cạnh. Và cũng dễ thấy rằng giải thuật tìm đường mở nên sử dụng thuật toán tìm kiếm theo chiều rộng để đường mở tìm được là đường đi ngắn nhất, giảm bớt công việc cho bước tăng cấp ghép.

Ta khởi tạo một hàng đợi (Queue) ban đầu chứa tất cả các $X_{\text{đỉnh}}$ chưa ghép. Thuật toán tìm kiếm theo chiều rộng làm việc theo nguyên tắc lấy một đỉnh v khỏi Queue và lại đẩy Queue những nút từ v chưa được thăm. Như vậy nếu thăm tới một $Y_{\text{đỉnh}}$ chưa ghép thì tức là ta tìm đường mở kết thúc ở $Y_{\text{đỉnh}}$ chưa ghép đó, quá trình tìm kiếm dừng ngay. Còn nếu ta thăm tới một đỉnh $y[j] \in Y$ đã ghép, dựa vào sự kiện: **từ $y[j]$ chỉ có thể tới được $\text{match}_Y[j]$** theo duy nhất một cạnh đã ghép định hướng ngược từ Y về X , nên ta có thể **đánh dấu thăm $y[j]$, thăm luôn cả $\text{match}_Y[j]$, và đẩy vào Queue phần tử $\text{match}_Y[j] \in X$** (Thăm liền 2 bước).

Input: file văn bản MATCH.INP

- ❖ Dòng 1: chứa hai số m, n ($m, n \leq 1000$) theo thứ tự là số $X_{\text{đỉnh}}$ và số $Y_{\text{đỉnh}}$ cách nhau ít nhất một dấu cách
- ❖ Các dòng tiếp theo, mỗi dòng ghi hai số i, j cách nhau ít nhất một dấu cách thể hiện có cạnh nối hai đỉnh $(x[i], y[j])$.

Output: file văn bản MATCH.OUT, ghi bộ ghép cực đại tìm được



P_4_11_1.PAS * Thuật toán đường mở tìm bộ ghép cực đại

```

{$MODE DELPHI} (*This program uses 32-bit Integer [-231..231 - 1]*)
program Finding_the_Maximum_Matching;
const
  InputFile = 'MATCH.INP';
  OutputFile = 'MATCH.OUT';
  max = 1000;
var
  m, n: Integer;
  a: array[1..max, 1..max] of Boolean;
  matchX, matchY: array[1..max] of Integer;
  Trace: array[1..max] of Integer;

```

```

procedure Enter;
var
  i, j: Integer;
  f: Text;
begin
  Assign(f, InputFile); Reset(f);
  FillChar(a, SizeOf(a), False);
  ReadLn(f, m, n);
  while not SeekEof(f) do
    begin
      ReadLn(f, i, j);
      a[i, j] := True;
    end;
  Close(f);
end;

procedure Init; {Khởi tạo bộ ghép rỗng}
begin
  FillChar(matchX, SizeOf(matchX), 0);
  FillChar(matchY, SizeOf(matchY), 0);
end;

{Tim đường mở, nếu thấy trả về một Y_dinh chưa ghép là đỉnh kết thúc đường mở, nếu không thấy trả về 0}
function FindAugmentingPath: Integer;
var
  Queue: array[1..max] of Integer;
  i, j, Front, Rear: Integer;
begin
  FillChar(Trace, SizeOf(Trace), 0); {Trace[j] = X_dinh liền trước y[j] trên đường mở}
  Rear := 0; {Khởi tạo hàng đợi rỗng}
  for i := 1 to m do {Đây tất cả những X_dinh chưa ghép vào hàng đợi}
    if matchX[i] = 0 then
      begin
        Inc(Rear);
        Queue[Rear] := i;
      end;
  {Thuật toán tìm kiếm theo chiều rộng}
  Front := 1;
  while Front <= Rear do
    begin
      i := Queue[Front]; Inc(Front); {Lấy một X_dinh ra khỏi Queue (x[i])}
      for j := 1 to n do {Xét những Y_dinh chưa thăm kề với x[i] qua một cạnh chưa ghép}
        if (Trace[j] = 0) and a[i, j] and (matchX[i] <> j) then
          begin {lệnh if trên hơi thừa dk matchX[i] <> j, điều kiện Trace[j] = 0 đã bao hàm luôn điều kiện này rồi}
            Trace[j] := i; {Lưu vết đường đi}
            if matchY[j] = 0 then {Nếu j chưa ghép thì ghi nhận đường mở và thoát ngay}
              begin
                FindAugmentingPath := j;
                Exit;
              end;
            Inc(Rear); {Đây luôn matchY[j] vào hàng đợi}
            Queue[Rear] := matchY[j];
          end;
    end;
  FindAugmentingPath := 0; {Ở trên không Exit được tức là không còn đường mở}
end;

{Nối rỗng bộ ghép bằng đường mở kết thúc ở fεY}
procedure Enlarge(f: Integer);
var
  x, next: Integer;
begin

```

```

repeat
  x := Trace[f];
  next := matchX[x];
  matchX[x] := f;
  matchY[f] := x;
  f := next;
until f = 0;
end;

procedure Solve; {Thuật toán đường mở}
var
  finish: Integer;
begin
  repeat
    finish := FindAugmentingPath; {Đầu tiên thử tìm một đường mở}
    if finish <> 0 then Enlarge(finish); {Nếu thấy thì tăng cặp và lặp lại}
  until finish = 0; {Nếu không thấy thì dừng}
end;

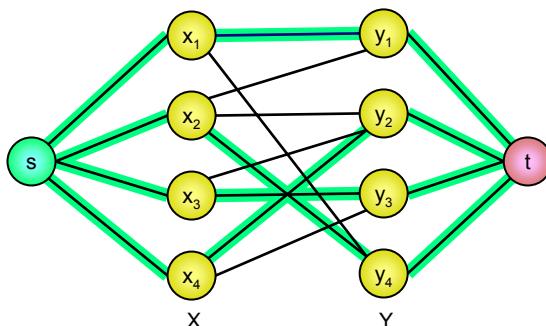
procedure PrintResult; {In kết quả}
var
  i, Count: Integer;
  f: Text;
begin
  Assign(f, OutputFile); Rewrite(f);
  WriteLn(f, 'Match: ');
  Count := 0;
  for i := 1 to m do
    if matchX[i] <> 0 then
      begin
        Inc(Count);
        WriteLn(f, Count, ') x[' , i, '] - y[' , matchX[i], ']');
      end;
  Close(f);
end;

begin
  Enter;
  Init;
  Solve;
  PrintResult;
end.

```

Khảo sát tính đúng đắn của thuật toán cho ta một kết quả khá thú vị:

Nếu ta thêm một đỉnh s và cho thêm m cung từ s tới tất cả những đỉnh của tập X, thêm một đỉnh t và nối thêm n cung từ tất cả các đỉnh của Y tới t. Ta được một mạng với đỉnh phát s và đỉnh thu t.



Hình 87: Mô hình luồng của bài toán tìm bộ ghép cực đại trên đồ thị hai phía

Nếu đặt khả năng thông qua của các cung đều là 1 sau đó tìm luồng cực đại trên mạng thì theo định lý về tính nguyên, luồng dương tìm được trên các cung đều phải là số nguyên (tức là bằng 1 hoặc 0). Khi đó dễ thấy rằng những cung có luồng dương bằng 1 từ tập X tới tập Y sẽ cho ta một bộ ghép lớn nhất. Để chứng minh thuật toán đường mở tìm được bộ ghép lớn nhất sau hữu hạn bước, ta sẽ chứng minh rằng số bộ ghép tìm được bằng thuật toán đường mở sẽ bằng giá trị luồng cực đại nói trên, điều đó cũng rất dễ bởi vì nếu để ý kỹ một chút thì đường mở chẳng qua là đường tăng luồng trên mạng thăng dư mà thôi, ngay cái tên augmenting path đã cho ta biết điều này. Vì vậy thuật toán đường mở ở trường hợp này là một **cách cài đặt hiệu quả trên một dạng đồ thị đặc biệt**, nó làm cho chương trình sáng sủa hơn nhiều so với phương pháp tìm bộ ghép dựa trên bài toán luồng thuần túy.

Người ta đã chứng minh được chi phí thời gian thực hiện giải thuật này trong trường hợp xấu nhất sẽ là $O(n^3)$ đối với đồ thị dày và $O(n(n + m)\log n)$ đối với đồ thị mỏng. Tuy nhiên, cũng giống như thuật toán Ford-Fulkerson, trên thực tế phương pháp này hoạt động rất nhanh.

Bài tập

Bài 1

Có n thợ và m công việc ($n, m \leq 100$). Mỗi thợ cho biết mình có thể làm được những việc nào, hãy phân công các thợ làm các công việc đó sao cho:

- ❖ Mỗi thợ phải làm ít nhất hai việc
- ❖ Một việc chỉ giao cho một thợ thực hiện
- ❖ Số việc thực hiện được là nhiều nhất có thể.

Hướng dẫn: Dựng đồ thị hai phía $G = (X \cup Y, E)$, X tập các thợ và Y là tập các công việc. Sử dụng thuật toán đường mở với cách hiểu đường mở là đường pha xuất phát từ 1 thợ chưa được phân đú hai việc và kết thúc ở một việc chưa được phân công.

Bài 2

Có n thợ và m công việc ($n, m \leq 100$). Mỗi thợ cho biết mình có thể làm được những việc nào, hãy phân công thực hiện các công việc đó sao cho:

- ❖ Mỗi công việc chỉ giao cho một thợ thực hiện
- ❖ Số công việc phân cho người thợ làm nhiều nhất là cực tiểu.

Hướng dẫn: Dựng đồ thị hai phía $G = (X \cup Y, E)$, X là tập các công việc và Y là tập thợ. Với một số nguyên k ($1 \leq k \leq m$), tìm cách phân công thực hiện các công việc sao cho không thợ nào làm quá k việc. Có thể sử dụng thuật toán đường mở với cách hiểu đường mở là đường pha xuất phát từ một việc chưa được phân công và kết thúc ở một thợ chưa làm đủ k việc.

Số k sẽ được thử lần lượt từ 1 tới m , xác định số k đầu tiên cho phép tìm ra phép phân công thực hiện hết các việc và in kết quả ra phép phân công tương ứng với số k đó. Có hai điều quan trọng phải lưu ý: Thứ nhất là phép thử với $k = k_0$ có thể tận dụng kết quả của phép phân công với $k = k_0 - 1$ chứ không cần thực hiện lại từ đầu. Thứ hai là việc “cải tiến” bằng áp dụng thuật toán tìm kiếm nhị phân để chỉ ra số k nhỏ nhất thỏa mãn yêu cầu phân công toàn

bộ các công việc sẽ không giúp ích gì mà sẽ chỉ làm chương trình chậm đi, lý do là bởi mỗi lần tăng cặp dựa trên đường mở thì số việc được phân công cũng chỉ tăng lên 1 mà thôi.

Bài 3

Xem lại bài toán Minimum Path Cover ở §10. Cài đặt lại theo cách bớt công kèn hơn bằng cách sử dụng những kỹ thuật đã học được trong bài toán tìm bộ ghép cực đại.

Bài 4

Cho đồ thị hai phía $G = (X \cup Y, E)$. Hãy chỉ ra một tập S gồm ít nhất các đỉnh sao cho mỗi cạnh $\in E$ đều liên thuộc với ít nhất một đỉnh thuộc S .

Hướng dẫn: Có thể đưa về mô hình bài toán luồng với khả năng thông qua của cả các cung và các đỉnh: Đưa một đỉnh phát giả s nối tới mọi đỉnh $\in X$, một đỉnh thu giả t nối từ mọi đỉnh $\in Y$. Các cạnh trong E được định chiều từ X sang Y , khả năng thông qua của các cung được đặt bằng $+\infty$. Khả năng thông qua của các đỉnh $\in X \cup Y$ đặt bằng 1. Sau đó tìm luồng cực đại từ s tới t và lát cắt hẹp nhất, lát cắt này chắc chắn cắt tại các đỉnh, những đỉnh bị cắt sẽ được chọn vào tập S . Tuy nhiên, bằng một số suy luận, ta có thể đưa về mô hình bài toán bộ ghép cực đại để dễ dàng hơn trong việc cài đặt bằng phương pháp sau:

- ❖ Tìm bộ ghép cực đại trên G
- ❖ Khi tìm xong bộ ghép, tức là thủ tục tìm đường mở không tìm ra đường mở, ta xác định được:
 - Tập $Y^* = \{Tập các Y_đỉnh đi đến được từ một X_đỉnh chưa ghép qua một đường pha\}$
 - Tập $X^* = \{x \in X | x \text{ đã ghép và đỉnh ghép với } x \notin Y^*\}$
- ❖ Đặt $S = X^* \cup Y^*$

§12. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI VỚI TRỌNG SỐ CỰC TIỂU TRÊN ĐỒ THỊ HAI PHÍA - THUẬT TOÁN HUNGARI

12.1. BÀI TOÁN PHÂN CÔNG

Đây là một dạng bài toán phát biểu như sau: Có m người (đánh số 1, 2, ..., m) và n công việc (đánh số 1, 2, ..., n), mỗi người có khả năng thực hiện một số công việc nào đó. Để giao cho người i thực hiện công việc j cần một chi phí là $c[i, j] \geq 0$. Cần phân cho mỗi thợ một việc và mỗi việc chỉ do một thợ thực hiện sao cho số công việc có thể thực hiện được là nhiều nhất và nếu có ≥ 2 phương án đều thực hiện được nhiều công việc nhất thì chỉ ra phương án chi phí ít nhất.

Dựng đồ thị hai phía $G = (X \cup Y, E)$ với X là tập m người, Y là tập n việc và $(u, v) \in E$ với trọng số $c[u, v]$ nếu như người u làm được công việc v . Bài toán đưa về tìm bộ ghép nhiều cạnh nhất của G có trọng số nhỏ nhất.

Gọi $k = \max(m, n)$. Bổ sung vào tập X và Y một số đỉnh giả để $|X| = |Y| = k$.

Gọi M là một số dương đủ lớn hơn chi phí của mọi phép phân công có thể. Với mỗi cặp đỉnh (u, v) : $u \in X$ và $v \in Y$. Nếu $(u, v) \notin E$ thì ta bổ sung cạnh (u, v) vào E với trọng số là M .

Khi đó ta được G là một đồ thị hai phía đầy đủ ($\text{Đồ thị hai phía mà giữa một đỉnh bất kỳ của } X \text{ và một đỉnh bất kỳ của } Y \text{ đều có cạnh nối}$). Và nếu như ta tìm được bộ ghép đầy đủ k cạnh mang trọng số nhỏ nhất thì ta chỉ cần loại bỏ khỏi bộ ghép đó những cạnh mang trọng số M vừa thêm vào thì sẽ được kế hoạch phân công 1 người \leftrightarrow 1 việc cần tìm. Điều này dễ hiểu bởi bộ ghép đầy đủ mang trọng số nhỏ nhất tức là phải ít cạnh trong số M nhất, tức là số phép phân công là nhiều nhất, và tất nhiên trong số các phương án ghép ít cạnh trọng số M nhất thì đây là phương án trọng số nhỏ nhất, tức là tổng chi phí trên các phép phân công là ít nhất.

12.2. PHÂN TÍCH

Input: $\text{Đồ thị hai phía đầy đủ } G = (X \cup Y, E); X = \{x[1], \dots, x[k]\}, Y = \{y[1], \dots, y[k]\}$. Được cho bởi ma trận vuông C cỡ $k \times k$, $c[i, j] = \text{trọng số cạnh nối đỉnh } x[i] \text{ với } y[j]$. Giả thiết $c[i, j] \geq 0 (\forall i, j)$

Output Bộ ghép đầy đủ trọng số nhỏ nhất.

Hai định lý sau đây tuy rất đơn giản nhưng là những định lý quan trọng tạo cơ sở cho thuật toán sẽ trình bày:

Định lý 1: Loại bỏ khỏi G những cạnh trọng số > 0 . Nếu những cạnh trọng số 0 còn lại tạo ra bộ ghép k cạnh trong G thì đây là bộ ghép cần tìm.

Chứng minh: Theo giả thiết, các cạnh của G mang trọng số không âm nên bất kỳ bộ ghép nào trong G cũng có trọng số không âm, mà bộ ghép ở trên mang trọng số 0, nên tất nhiên đó là bộ ghép đầy đủ trọng số nhỏ nhất.

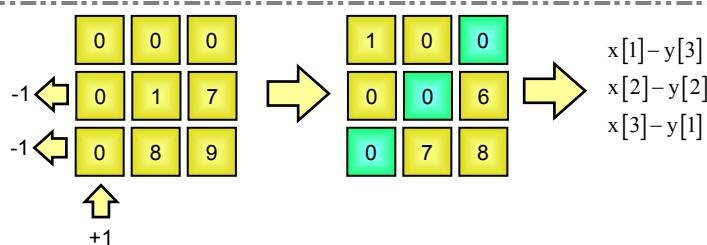
Định lý 2: Với đỉnh $x[i]$, nếu ta cộng thêm một số Δ (dương hay âm) vào tất cả những cạnh liên thuộc với $x[i]$ (tương đương với việc cộng thêm Δ vào tất cả các phần tử thuộc hàng i của ma trận C) thì không ảnh hưởng tới bộ ghép đầy đủ trọng số nhỏ nhất.

Chứng minh: Với một bộ ghép đầy đủ bất kỳ thì có một và chỉ một cạnh ghép với $x[i]$. Nên việc cộng thêm Δ vào tất cả các cạnh liên thuộc với $x[i]$ sẽ làm tăng trọng số bộ ghép đó lên Δ . Vì vậy nếu như ban đầu, M là bộ ghép đầy đủ trọng số nhỏ nhất thì sau thao tác trên, M vẫn là bộ ghép đầy đủ trọng số nhỏ nhất.

Hệ quả: Với đỉnh $y[j]$, nếu ta cộng thêm một số Δ (dương hay âm) vào tất cả những cạnh liên thuộc với $y[j]$ (tương đương với việc cộng thêm Δ vào tất cả các phần tử thuộc cột j của ma trận C) thì không ảnh hưởng tới bộ ghép đầy đủ trọng số nhỏ nhất.

Từ đây có thể nhận ra tư tưởng của thuật toán: *Từ đồ thị G, ta tìm chiến lược cộng / trừ một cách hợp lý trọng số của các cạnh liên thuộc với từng đỉnh để được một đồ thị mới vẫn có các cạnh trọng số không âm, mà các cạnh trọng số 0 của đồ thị mới đó chứa một bộ ghép đầy đủ k cạnh.*

Ví dụ: Biến đổi ma trận trọng số của đồ thị hai phía 3 đỉnh trái, 3 đỉnh phải:



Hình 88: Phép xoay trọng số cạnh

12.3. THUẬT TOÁN

12.3.1. Các khái niệm:

Để cho gọn, ta gọi những cạnh trọng số 0 của G là những 0_cạnh.

Xét một bộ ghép M chỉ gồm những 0_cạnh.

- ❖ Những đỉnh $\in M$ gọi là những đỉnh đã ghép, những đỉnh còn lại gọi là những đỉnh chưa ghép.
- ❖ Những 0_cạnh $\in M$ gọi là những 0_cạnh đã ghép, những 0_cạnh còn lại là những 0_cạnh chưa ghép.
- ❖ Nếu ta định hướng lại các 0_cạnh theo cách: Những 0_cạnh chưa ghép cho hướng từ tập X sang tập Y, những 0_cạnh đã ghép cho hướng từ tập Y về tập X. Khi đó:
- ❖ Đường pha (Alternating Path) là một đường đi cơ bản xuất phát từ một X_đỉnh chưa ghép đi theo các 0_cạnh đã định hướng ở trên. Như vậy dọc trên đường pha, các 0_cạnh chưa

ghép và những 0_cạnh đã ghép xen kẽ nhau. Vì đường pha chỉ là đường đi cơ bản trên đồ thị định hướng nên việc xác định những đỉnh nào có thể đến được từ $x \in X$ bằng một đường pha có thể sử dụng các thuật toán tìm kiếm trên đồ thị (BFS hoặc DFS). Những đỉnh và những cạnh được duyệt qua tạo thành một cây pha gốc x

- ❖ Một đường mở (Augmenting Path) là một đường pha đi từ một $X_{\text{đỉnh}} \neq \emptyset$ chưa ghép tới một $Y_{\text{đỉnh}} \neq \emptyset$ chưa ghép.

Như vậy:

- ❖ Đường đi trực tiếp từ một $X_{\text{đỉnh}} \neq \emptyset$ chưa ghép tới một $Y_{\text{đỉnh}} \neq \emptyset$ chưa ghép qua một 0_cạnh chưa ghép cũng là một đường mở.

- ❖ Dọc trên đường mở, số 0_cạnh chưa ghép nhiều hơn số 0_cạnh đã ghép đúng 1 cạnh.

12.3.2. Thuật toán Hungari

Bước 1: Khởi tạo:

Một bộ ghép $M := \emptyset$

Bước 2: Với mọi đỉnh $x^* \in X$, ta tìm cách ghép x^* :

Bắt đầu từ đỉnh x^* , thử tìm đường mở bắt đầu ở x^* bằng thuật toán tìm kiếm trên đồ thị. Có hai khả năng có thể xảy ra:

- ❖ Hoặc tìm được đường mở thì dọc theo đường mở, ta loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép, ta được một **bộ ghép mới nhiều hơn bộ ghép cũ 1 cạnh và đỉnh x^* trở thành đã ghép**.

- ❖ Hoặc không tìm được đường mở thì có thể xác định được:

$\text{VisitedX} = \{\text{Tập những } X_{\text{đỉnh}} \text{ có thể đến được từ } x^* \text{ bằng một đường pha}\}$

$\text{VisitedY} = \{\text{Tập những } Y_{\text{đỉnh}} \text{ có thể đến được từ } x^* \text{ bằng một đường pha}\}$

Gọi Δ là trọng số nhỏ nhất của các cạnh nối giữa một đỉnh thuộc VisitedX với một đỉnh không thuộc VisitedY . Dễ thấy $\Delta > 0$ bởi nếu $\Delta = 0$ thì tồn tại một 0_cạnh (x, y) với $x \in \text{VisitedX}$ và $y \notin \text{VisitedY}$. Vì x^* đến được x bằng một đường pha và (x, y) là một 0_cạnh nên x^* cũng đến được y bằng một đường pha, dẫn tới $y \in \text{VisitedY}$, điều này vô lý.

Biến đổi đồ thị G: Với $\forall x \in \text{VisitedX}$, trừ Δ vào trọng số những cạnh liên thuộc với x , Với $\forall y \in \text{VisitedY}$, cộng Δ vào trọng số những cạnh liên thuộc với y .

Lặp lại thủ tục tìm kiếm trên đồ thị thử tìm đường mở xuất phát ở x^* cho tới khi tìm ra đường mở.

Bước 3: Sau bước 2 thì mọi $X_{\text{đỉnh}}$ đều được ghép, in kết quả về bộ ghép tìm được.

Mô hình cài đặt của thuật toán có thể viết như sau:

```
M := ∅;
for (x* ∈ X) do
begin
repeat
```

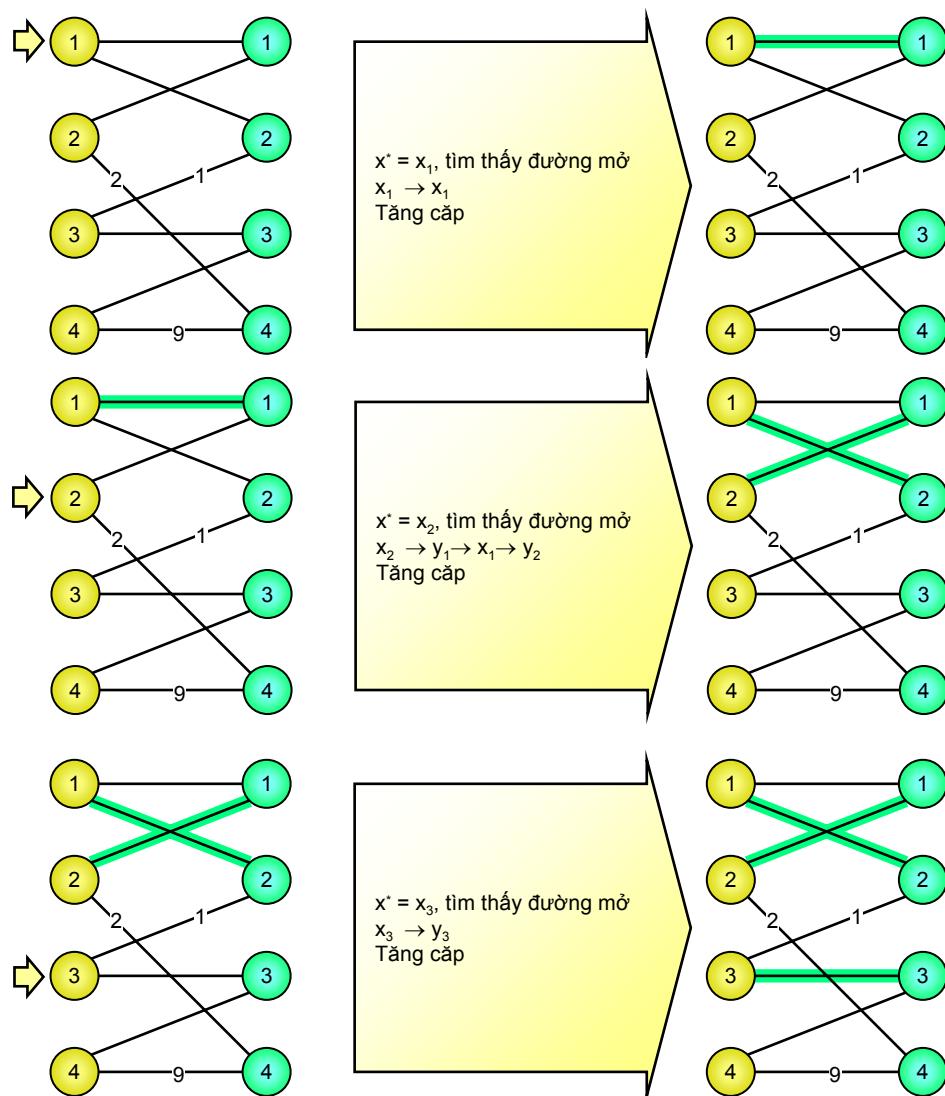
```

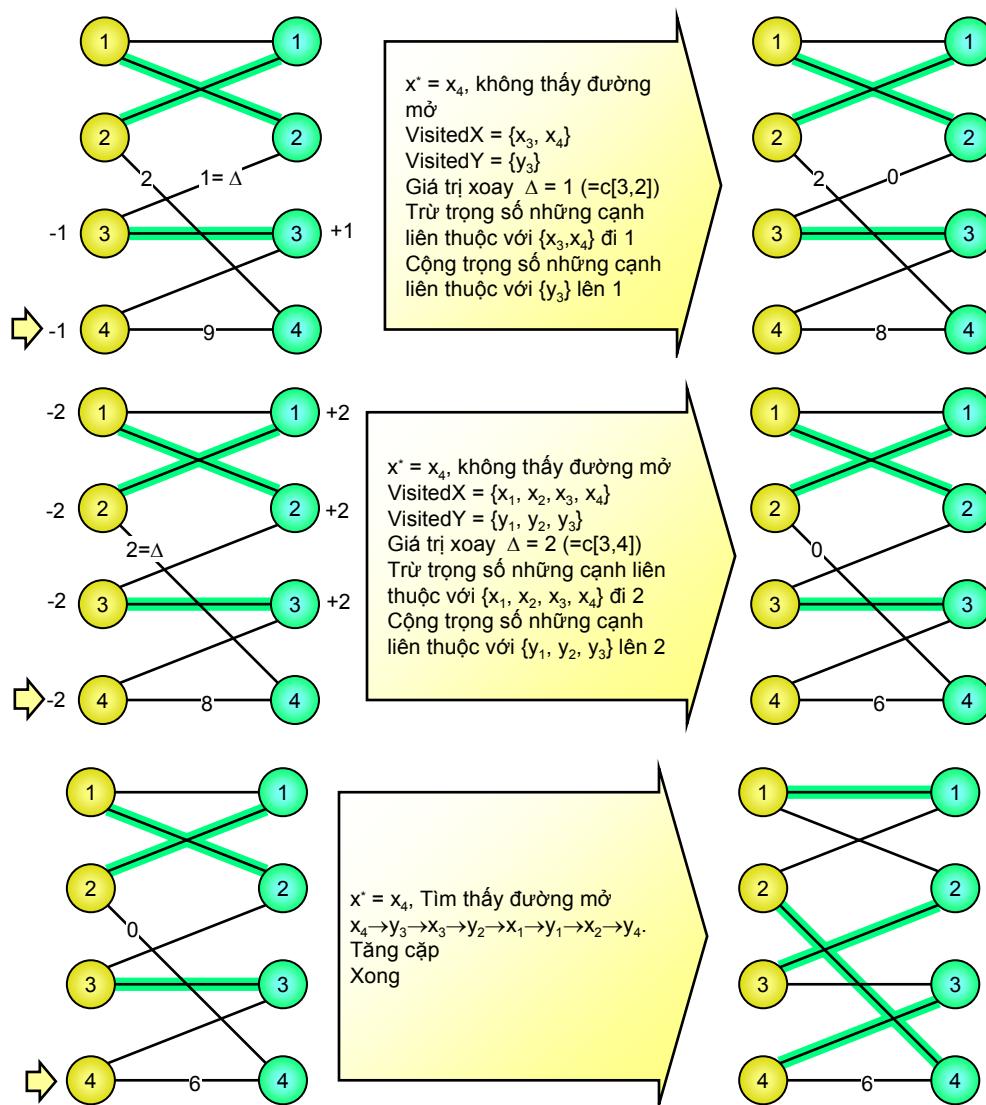
<Tim đường mở xuất phát ở  $x^*$ >;
if <Không tìm thấy đường mở> then <Biến đổi đồ thị G: Chọn  $\Delta := ...$ >;
until <Tim thấy đường mở>;
<Đọc theo đường mở, loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép>;
end;
<Output: M là bộ ghép cần tìm>;

```

Ví dụ minh họa:

Để không bị rối hình, ta hiểu những cạnh không ghi trọng số là những 0_cạnh, những cạnh không vẽ mang trọng số rất lớn trong trường hợp này không cần thiết phải tính đến. Những cạnh nét đậm là những cạnh đã ghép, những cạnh nét thanh là những cạnh chưa ghép.



**Hình 89: Thuật toán Hungari**

Để ý rằng nếu như không tìm thấy đường mở xuất phát ở x^* thì quá trình tìm kiếm trên đồ thị sẽ cho ta một cây pha gốc x^* . Giá trị xoay Δ thực chất là trọng số nhỏ nhất của cạnh nối một $X_{\text{đỉnh}}$ trong cây pha với một $Y_{\text{đỉnh}}$ ngoài cây pha (cạnh ngoài). Việc trừ Δ vào những cạnh liên thuộc với $X_{\text{đỉnh}}$ trong cây pha và cộng Δ vào những cạnh liên thuộc với $Y_{\text{đỉnh}}$ trong cây pha sẽ làm cho cạnh ngoài nối trên trở thành 0_cạnh, các cạnh khác vẫn có trọng số ≥ 0 . Nhưng quan trọng hơn là **tất cả những cạnh trong cây pha vẫn là 0_cạnh**. Điều đó đảm bảo cho quá trình tìm kiếm trên đồ thị lần sau sẽ xây dựng được cây pha mới lớn hơn cây pha cũ. Vì tập các $Y_{\text{đỉnh}}$ đã ghép là hữu hạn nên sau quá k bước, cây pha sẽ quét tới một $Y_{\text{đỉnh}}$ chưa ghép, tức là tìm ra đường mở

12.3.3. Phương pháp đối ngẫu Kuhn-Munkres

Phương pháp Kuhn-Munkres để tìm hai dãy số $Fx[1..k]$ và $Fy[1..k]$ thoả mãn:

- ❖ $c[i, j] - Fx[i] - Fy[j] \geq 0$
- ❖ Tập các cạnh $(x[i], y[j])$ thoả mãn $c[i, j] - Fx[i] - Fy[j] = 0$ chứa trọn một bộ ghép đầy đủ k cạnh, đây chính là bộ ghép cần tìm.

Rõ ràng nếu tìm được hai dãy số thỏa mãn trên thì ta chỉ việc thực hiện hai thao tác:

- ❖ Với mỗi đỉnh $x[i]$, trừ tất cả trọng số của những cạnh liên thuộc với $x[i]$ đi $Fx[i]$
- ❖ Với mỗi đỉnh $y[j]$, trừ tất cả trọng số của những cạnh liên thuộc với $y[j]$ đi $Fy[j]$

(Hai thao tác này tương đương với việc trừ tất cả trọng số của các cạnh $(x[i], y[j])$ đi một lượng $Fx[i] + Fy[j]$ tức là $c[i, j] := c[i, j] - Fx[i] - Fy[j]$)

Thì dễ thấy đồ thị mới tạo thành sẽ gồm có các cạnh trọng số không âm và những 0_cạnh của đồ thị chứa trọn một bộ ghép đầy đủ.

	1	2	3	4	
1	0	0	M	M	$Fx[1] = 2$
2	0	M	M	2	$Fx[2] = 2$
3	M	1	0	M	$Fx[3] = 3$
4	M	M	0	9	$Fx[4] = 3$

$Fy[1] = -2 \quad Fy[2] = -2 \quad Fy[3] = -3 \quad Fy[4] = 0$

(Có nhiều phương án khác: $Fx = (0, 0, 1, 1)$; $Fy = (0, 0, -1, 2)$ cũng đúng)

Vậy phương pháp Kuhn-Munkres đưa việc biến đổi đồ thị G (biến đổi ma trận C) về việc biến đổi hay dãy số Fx và Fy . Việc trừ Δ vào trọng số tất cả những cạnh liên thuộc với $x[i]$ tương đương với việc tăng $Fx[i]$ lên Δ . Việc cộng Δ vào trọng số tất cả những cạnh liên thuộc với $y[j]$ tương đương với giảm $Fy[j]$ đi Δ . Khi cần biết trọng số cạnh $(x[i], y[j])$ là bao nhiêu sau các bước biến đổi, thay vì viết $c[i, j]$, ta viết $c[i, j] - Fx[i] - Fy[j]$.

Sơ đồ cài đặt phương pháp Kuhn-Munkres có thể viết như sau:

Bước 1: Khởi tạo:

- ❖ $M := \emptyset$;
- ❖ Việc khởi tạo các Fx , Fy có thể có nhiều cách miễn sao $c[i, j] - Fx[i] - Fy[j] \geq 0$, đơn giản nhất có thể đặt tất cả các $Fx[.]$ và $Fy[.]$ bằng 0

Bước 2: Với mọi đỉnh $x^* \in X$, ta tìm cách ghép x^* như sau:

Bắt đầu từ đỉnh x^* , thử tìm đường mở bắt đầu ở x^* bằng thuật toán tìm kiêm trên đồ thị (BFS hoặc DFS). Có hai khả năng xảy ra:

- ❖ Hoặc tìm được đường mở thì đọc theo đường mở, ta loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép.
- ❖ Hoặc không tìm được đường mở thì xác định được:

$VisitedX = \{Tập nhũng X_đỉnh có thể đến được từ x^* bằng một đường pha\}$

$VisitedY = \{Tập nhũng Y_đỉnh có thể đến được từ x^* bằng một đường pha\}$

$\Delta := \min \{c[i, j] - Fx[i] - Fy[j] \mid \forall x[i] \in VisitedX; \forall y[j] \notin VisitedY\}$

Với $\forall x[i] \in VisitedX$: đặt $Fx[i] := Fx[i] + \Delta$;

Với $\forall y[j] \in VisitedY$: đặt $Fy[j] := Fy[j] - \Delta$;

Lặp lại thủ tục tìm đường mở xuất phát tại x^* cho tới khi tìm ra đường mở.

Bước 3:

Lưu ý rằng bước 2 luôn tìm ra đường mở vì đồ thị đã được làm cho trở nên cân bằng ($|X|=|Y|$) và đầy đủ, ta chỉ việc trả về đường mở tìm được.

Đáng lưu ý ở phương pháp Kuhn-Munkres là phương pháp này không làm thay đổi ma trận C ban đầu. Điều đó thực sự hữu ích trong trường hợp trọng số của cạnh $(x[i], y[j])$ không được cho một cách tường minh bằng giá trị $c[i, j]$ mà lại cho bằng hàm $c(i, j)$: trong trường hợp này, việc trừ hàng/cộng cột trực tiếp trên ma trận chi phí C là không thể thực hiện được.

12.3.4. Cài đặt

a) Biểu diễn bộ ghép

Để biểu diễn bộ ghép, ta sử dụng hai mảng: $matchX[1..k]$ và $matchY[1..k]$.

- ❖ $matchX[i]$ là chỉ số của đỉnh thuộc tập Y ghép với đỉnh $x[i]$
- ❖ $matchY[j]$ là chỉ số của đỉnh thuộc tập X ghép với đỉnh $y[j]$.

Tức là nếu như cạnh $(x[i], y[j])$ thuộc bộ ghép thì $matchX[i] = j$ và $matchY[j] = i$.

Quy ước:

- ❖ Nếu như $x[i]$ chưa ghép với đỉnh nào của tập Y thì $matchX[i] = 0$
- ❖ Nếu như $y[j]$ chưa ghép với đỉnh nào của tập X thì $matchY[j] = 0$

Suy ra:

- ❖ Thêm một cạnh $(x[i], y[j])$ vào bộ ghép \Leftrightarrow Đặt $matchX[i] := j$ và $matchY[j] := i$;
- ❖ Loại một cạnh $(x[i], y[j])$ khỏi bộ ghép \Leftrightarrow Đặt $matchX[i] := 0$ và $matchY[j] := 0$;

b) Tìm đường mở như thế nào

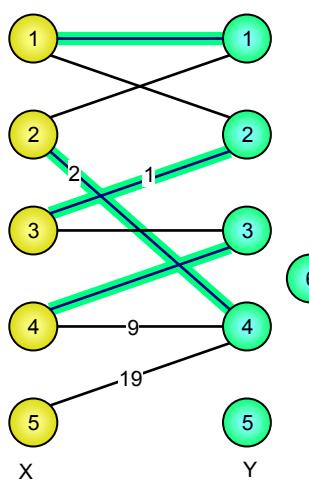
Ta sẽ tìm đường mở và xây dựng hai tập $VisitedX$ và $VisitedY$ bằng thuật toán tìm kiếm theo chiều rộng, chỉ xét những 0_cạnh định hướng như đã nói trong phần đầu:

Khởi tạo một hàng đợi (Queue) ban đầu chỉ có một đỉnh x^* . Thuật toán tìm kiếm theo chiều rộng làm việc theo nguyên tắc lấy một đỉnh v khỏi Queue và lại đẩy Queue những nối từ v chưa được thăm. Như vậy nếu thăm tới một $Y_{\text{đỉnh}}$ chưa ghép thì tức là ta tìm đường mở kết thúc ở $Y_{\text{đỉnh}}$ chưa ghép đó, quá trình tìm kiếm dừng ngay. Còn nếu ta thăm tới một đỉnh $y[j] \in Y$ đã ghép, dựa vào sự kiện: từ $y[j]$ chỉ có thể tới được $matchY[j]$ theo duy nhất một 0_cạnh định hướng, nên ta có thể đánh dấu thăm $y[j]$, thăm luôn cả $matchY[j]$, và đẩy vào Queue phần tử $matchY[j] \in X$.

Input: file văn bản ASSIGN.INP

- ❖ Dòng 1: Ghi hai số m, n theo thứ tự là số thợ và số việc cách nhau 1 dấu cách ($m, n \leq 1000$)
- ❖ Các dòng tiếp theo, mỗi dòng ghi ba số $i, j, c[i, j]$ cách nhau 1 dấu cách thể hiện thợ i làm được việc j và chi phí để làm là $c[i, j]$ ($1 \leq i \leq m; 1 \leq j \leq n; 0 \leq c[i, j] \leq 1000$).

Output: file văn bản ASSIGN.OUT, mô tả phép phân công tối ưu tìm được.



ASSIGN.INP	ASSIGN.OUT
5 6	Optimal assignment:
1 1 0	1) x[1] - y[1] 0
1 2 0	2) x[2] - y[4] 2
2 1 0	3) x[3] - y[2] 1
2 4 2	4) x[4] - y[3] 0
3 2 1	Cost: 3
3 3 0	
4 3 0	
4 4 9	
5 4 19	

P_4_12_1.PAS * Thuật toán Hungari

```

{$MODE DELPHI} (*This program uses 32-bit Integer [-231..231 - 1]*)
program Finding_the_Best_Assignment;
const
  InputFile = 'ASSIGN.INP';
  OutputFile = 'ASSIGN.OUT';
  max = 1000;
  maxEC = 1000
  maxC = max * maxEC + 1;
var
  c: array[1..max, 1..max] of Integer;
  Fx, Fy, matchX, matchY, Trace: array[1..max] of Integer;
  m, n, k, start, finish: Integer;

procedure Enter; {Nhập dữ liệu}
var
  i, j: Integer;
  f: Text;
begin
  Assign(f, InputFile); Reset(f);
  ReadLn(f, m, n);
  if m > n then k := m else k := n;
  for i := 1 to k do
    for j := 1 to k do c[i, j] := maxC;
  while not SeekEof(f) do ReadLn(f, i, j, c[i, j]);
  Close(f);
end;

procedure Init; {Khởi tạo bộ ghép rỗng và các giá trị Fx[.], Fy[.]}
begin
  FillChar(matchX, SizeOf(matchX), 0);
  FillChar(matchY, SizeOf(matchY), 0);
  FillChar(Fx, SizeOf(Fx), 0);
  FillChar(Fy, SizeOf(Fy), 0);
end;

function GetC(i, j: Integer): Integer; {Hàm trả về trọng số cạnh (x[i], y[j])}
begin
  GetC := c[i, j] - Fx[i] - Fy[j];
end;

procedure FindAugmentingPath; {Thủ tục tìm đường mò xuất phát ở x[start]}
var
  Queue: array[1..max] of Integer; {Hàng đợi dùng cho BFS, chỉ chứa chỉ số các đỉnh ∈ X}
  i, j, Front, Rear: Integer;

```

```

begin
  FillChar(Trace, SizeOf(Trace), 0);
  Queue[1] := start;
  Front := 1; Rear := 1;
  repeat
    i := Queue[Front]; Inc(Front); {Lấy i ra khỏi Queue, xét x[i]}
    for j := 1 to k do
      if (Trace[j] = 0) and (GetC(i, j) = 0) then {Nếu y[j] chưa thăm và kè với x[i] qua 0_cạnh}
        begin
          Trace[j] := i; {Lưu vết đường đi}
          if matchY[j] = 0 then {Nếu y[j] đã ghép thì ghi nhận và thoát ngay}
            begin
              finish := j;
              Exit;
            end;
          Inc(Rear); Queue[Rear] := matchY[j]; {Không thì đẩy matchY[j] vào Queue, chờ duyệt tiếp}
        end;
    until Front > Rear;
end;

```

```
procedure SubX_AddY; {Phép xoay trọng số cạnh}
```

```

var
  i, j, t, Delta: Integer;
  VisitedX, VisitedY: set of Byte;
begin
  {Trước hết tìm hai tập VisitedX và VisitedY chứa chỉ số các đỉnh đến được từ x[start] qua một đường pha}
  VisitedX := [start];
  VisitedY := [];
  for j := 1 to k do
    if Trace[j] <> 0 then
      begin
        Include(VisitedX, matchY[j]);
        Include(VisitedY, j);
      end;
  {Tính Delta := min(GetC(i, j)|i ∈ VisitedX và j ∉ VisitedY)}
  Delta := maxC;
  for i := 1 to k do
    if i in VisitedX then
      for j := 1 to k do
        if not (j in VisitedY) and (GetC(i, j) < Delta) then
          Delta := GetC(i, j);
  {Xoay}
  for t := 1 to k do
    begin
      if t in VisitedX then Fx[t] := Fx[t] + Delta;
      if t in VisitedY then Fy[t] := Fy[t] - Delta;
    end;
end;

```

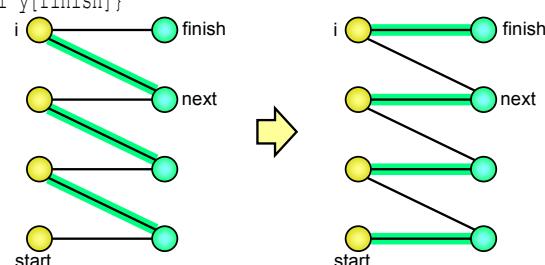
```
procedure Enlarge; {Nới rộng bộ ghép bằng đường mờ kết thúc tại y[finish]}
```

```

var
  i, next: Integer;
begin
  repeat
    i := Trace[finish];
    next := matchX[i];
    matchX[i] := finish;
    matchY[finish] := i;
    finish := Next;
  until finish = 0; {finish = 0 ⇔ i = start}
end;

```

```
procedure Solve; {Thuật toán Hungari}
```



```

var
  i: Integer;
begin
  for i := 1 to k do
    begin
      start := i; finish := 0;
      repeat {Tim cách ghép x[start]}
        FindAugmentingPath;
        if finish = 0 then SubX_AddY; {Nếu không tìm ra đường mờ xuất phát từ x[start] thì xoay các trọng số cạnh}
        until finish <> 0;
      Enlarge; {Khi đã tìm ra đường mờ thì chỉ cần tăng cặp theo đường mờ}
    end;
end;

procedure Result; {In kết quả}
var
  i, j, Count, W: Integer;
  f: Text;
begin
  Assign(f, OutputFile); Rewrite(f);
  WriteLn(f, 'Optimal assignment:');
  W := 0; Count := 0;
  for i := 1 to m do
    begin
      j := matchX[i];
      if c[i, j] < maxC then {Chỉ in quan tâm tới những cạnh trọng số < maxC}
        begin
          Inc(Count);
          WriteLn(f, Count:3, ') x[', i, ', ', j, ', ', c[i, j]);
          W := W + c[i, j];
        end;
    end;
  WriteLn(f, 'Cost: ', W);
  Close(f);
end;

begin
  Enter;
  Init;
  Solve;
  Result;
end.

```

Nhận xét:

- ❖ Có thể giải quyết bài toán phân công nếu như ma trận chi phí C có phần tử âm bằng cách sửa lại một chút trong thủ tục khởi tạo (Init): Với $\forall i$, thay vì gán $Fx[i] := 0$, ta gán $Fx[i] :=$ giá trị nhỏ nhất trên hàng i của ma trận C, khi đó sẽ đảm bảo $c[i, j] - Fx[i] - Fy[j] \geq 0$ ($\forall i, j$).
- ❖ Sau khi kết thúc thuật toán, tổng tất cả các phần tử ở hai dãy Fx, Fy bằng trọng số cực tiểu của bộ ghép đầy đủ tìm được trên đồ thị ban đầu.
- ❖ Một vấn đề nữa phải hết sức cẩn thận trong việc ước lượng độ lớn của các phần tử $Fx[.]$ và $Fy[.]$ khi khai báo mảng, các giá trị này có thể lớn hơn rất nhiều lần so với giá trị lớn nhất của các $c[i, j]$. Hãy tự tìm ví dụ để giải thích tại sao.

12.4. BÀI TOÁN TÌM BỘ GHÉP CỰC ĐẠI VỚI TRỌNG SỐ CỰC ĐẠI TRÊN ĐỒ THỊ HAI PHÍA

Bài toán tìm bộ ghép cực đại với trọng số cực đại cũng có thể giải nhờ phương pháp Hungari bằng cách đổi dấu tất cả các phần tử ma trận chi phí.

Khi cài đặt, ta có thể sửa lại đôi chút trong chương trình trên để giải bài toán tìm bộ ghép cực đại với trọng số cực đại mà không cần đổi dấu trọng số. Cụ thể như sau:

Bước 1: Khởi tạo:

- ❖ $M := \emptyset$;
- ❖ Khởi tạo hai dãy Fx và Fy thỏa mãn: $\forall i, j: Fx[i] + Fy[j] \geq c[i, j]$; chặng hạn ta có thể đặt $Fx[i] :=$ Phần tử lớn nhất trên dòng i của ma trận C và đặt các $Fy[j] := 0$.

Bước 2: Với mọi đỉnh $x^* \in X$, ta tìm cách ghép x^* :

Bắt đầu từ đỉnh x^* , thử tìm đường mở bắt đầu ở x^* . Có hai khả năng xảy ra:

- ❖ Hoặc tìm được đường mở thì dọc theo đường mở, ta loại bỏ những cạnh đã ghép khỏi M và thêm vào M những cạnh chưa ghép.
- ❖ Hoặc không tìm được đường mở thì xác định được:

$$\text{VisitedX} = \{\text{Tập những } X\text{-đỉnh có thể đến được từ } x^* \text{ bằng một đường pha}\}$$

$$\text{VisitedY} = \{\text{Tập những } Y\text{-đỉnh có thể đến được từ } x^* \text{ bằng một đường pha}\}$$

$$\text{Đặt } \Delta := \min \{Fx[i] + Fy[j] - c[i, j] \mid \forall x[i] \in \text{VisitedX}, \forall y[j] \notin \text{VisitedY}\}$$

Xoay trọng số cạnh:

$$\text{Với } \forall x[i] \in \text{VisitedX}: Fx[i] := Fx[i] - \Delta;$$

$$\text{Với } \forall y[j] \in \text{VisitedY}: Fy[j] := Fy[j] + \Delta;$$

Lặp lại thủ tục tìm đường mở xuất phát tại x^* cho tới khi tìm ra đường mở.

Bước 3: Sau bước 2 thì mọi X -đỉnh đều đã ghép, ta được một bộ ghép đầy đủ k cạnh với trọng số lớn nhất.

Dễ dàng chứng minh được tính đúng đắn của phương pháp, bởi nếu ta đặt:

$$c'[i, j] = -c[i, j]; F'x[i] := -Fx[i]; F'y[j] = -Fy[j].$$

Thì bài toán trở thành tìm cặp ghép đầy đủ trọng số cực tiểu trên đồ thị hai phía với ma trận trọng số $c'[1..k, 1..k]$. Bài toán này được giải quyết bằng cách tính hai dãy đối ngẫu $F'x$ và $F'y$. Từ đó bằng những biến đổi đại số cơ bản, ta có thể kiểm chứng được tính tương đương giữa các bước của phương pháp nêu trên với các bước của phương pháp Kuhn-Munkres ở mục trước.

12.5. NÂNG CẤP

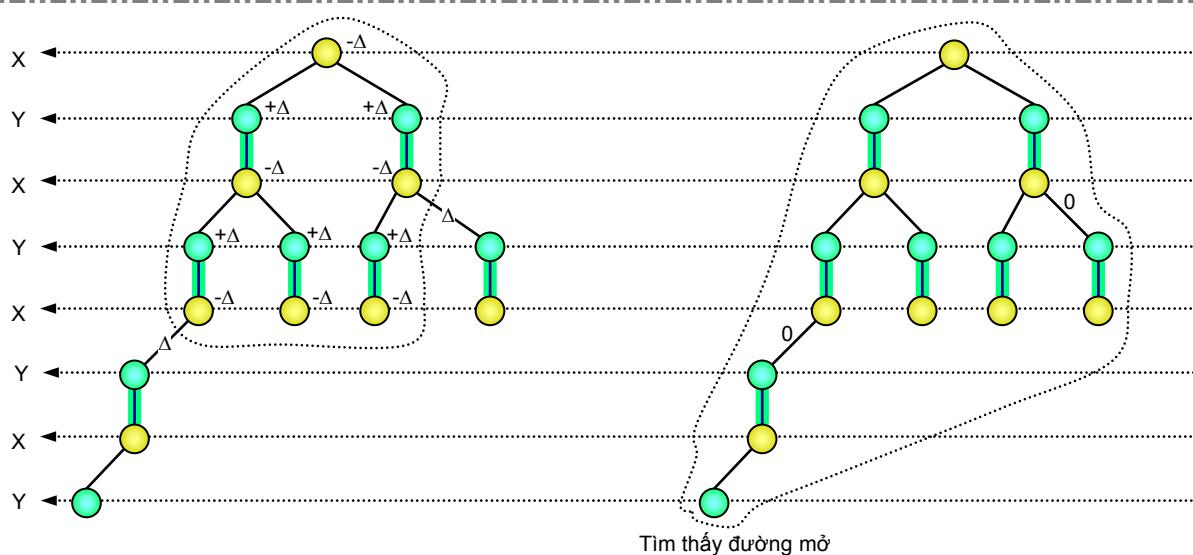
Dựa vào mô hình cài đặt thuật toán Kuhn-Munkres ở trên, ta có thể đánh giá về độ phức tạp tính toán lý thuyết của cách cài đặt này:

Thuật toán tìm kiếm theo chiều rộng được sử dụng để tìm đường mở có độ phức tạp $O(k^2)$, mỗi lần xoay trọng số cạnh mất một chi phí thời gian cỡ $O(k^2)$. Vậy mỗi lần tăng gấp, cần tối đa k lần dò đường và k lần xoay trọng số cạnh, mất một chi phí thời gian cỡ $O(k^3)$. Thuật toán cần k lần tăng gấp nên độ phức tạp tính toán trên lý thuyết của phương pháp này cỡ $O(k^4)$.

Có thể cải tiến mô hình cài đặt để được một thuật toán với độ phức tạp $O(k^3)$ dựa trên những nhận xét sau:

12.5.1. Nhận xét 1

Quá trình tìm kiếm theo chiều rộng bắt đầu từ một đỉnh x^* chưa ghép cho ta một cây pha gốc x^* . Nếu tìm được đường mở thì dừng lại và tăng gấp ngay, nếu không thì xoay trọng số cạnh và bắt đầu tìm kiếm lại để được một cây pha mới lớn hơn cây pha cũ (Hình 90):



Hình 90: Cây pha “mọc” lớn hơn sau mỗi lần xoay trọng số cạnh và tìm đường

12.5.2. Nhận xét 2

Việc xác định trọng số nhỏ nhất của cạnh nối một $X_{\text{đỉnh}}$ trong cây pha với một $Y_{\text{đỉnh}}$ ngoài cây pha có thể kết hợp ngay trong bước dựng cây pha mà không làm tăng gấp phúc tạp tính toán. Để thực hiện điều này, ta sử dụng kỹ thuật như trong thuật toán Prim:

Với mọi $y[j] \in Y$, gọi $d[j] :=$ khoảng cách từ $y[j]$ đến cây pha gốc x^* . Ban đầu $d[j]$ được khởi tạo bằng trọng số cạnh $(x^*, y[j])$ (cây pha ban đầu chỉ có đúng một đỉnh x^*).

Trong bước tìm đường bằng BFS, mỗi lần rút một đỉnh $x[i]$ ra khỏi Queue, ta xét những đỉnh $y[j] \in Y$ chưa thăm và đặt lại $d[j]_{\text{mới}} := \min(d[j]_{\text{cũ}}, \text{trọng số cạnh } (x[i], y[j]))$ sau đó mới kiểm tra xem $(x[i], y[j])$ có phải là 0_cạnh hay không để tiếp tục các thao tác như trước. Nếu quá trình BFS không tìm ra đường mở thì giá trị xoay Δ chính là giá trị nhỏ nhất trong các $d[j]$ dương. Ta bót được một đoạn chương trình tìm giá trị xoay có độ phức tạp $O(k^2)$. Công việc tại mỗi bước xoay chỉ là tìm giá trị nhỏ nhất trong các $d[j]$ dương và thực hiện phép cộng, trừ trên hai dãy đối ngẫu F_x và F_y , nó có độ phức tạp tính toán $O(k)$. Tối đa có k lần xoay để tìm đường mở nên tổng chi phí thời gian thực hiện các lần xoay cho tới khi tìm ra đường mở cỡ $O(k^2)$. Lưu ý rằng đồ thị đang xét là đồ thị hai phần đủ nêu sau khi xoay các trọng số cạnh