



HỘI THẢO KHOA HỌC CÁC TRƯỜNG THPT CHUYÊN KHU VỰC DUYÊN HẢI VÀ ĐỒNG BẮC BỘ NĂM 2021

Môn: Tin học

BÀI TOÁN LUỒNG CỰC ĐẠI THUẬT TOÁN FORD – FULKERSON và NHỮNG CẢI TIẾN



Tháng 09/2021

MỤC LỤC

1.	Giới thiệu.....	5
2.	Một số khái niệm	5
2.1.	Mạng	5
2.2.	Luồng trên mạng	6
2.3.	Hình ảnh thực tế	7
2.4.	Bài toán luồng cực đại tổng quát.....	7
3.	Thuật toán Ford – Fulkerson và những cải tiến	8
3.1.	Thuật toán gốc và một số khái niệm	8
3.1.1.	Ý tưởng thuật toán	8
3.1.2.	Mạng thặng dư (Residual Network)	8
3.1.3.	Ví dụ Minh họa thuật toán Ford – Fulkerson	10
3.1.4.	Độ phức tạp.....	14
3.1.5.	Bài toán Luồng cực đại cụ thể	14
3.1.6.	Link tải bộ test	14
3.2.	Cài đặt thuật toán Ford – Fulkerson và những thuật toán cải tiến.....	15
3.2.1.	Thuật toán Ford – Fulkerson phiên bản gốc	15
3.2.2.	Thuật toán Edmonds – Karp: Shortest path.....	16
3.2.3.	Thuật toán Edmonds – Karp: Fattest Path	19
3.2.4.	Thuật toán Edmonds – Karp, Dinitz: Capacity scaling	21
3.2.5.	Thuật toán Dinic	24
3.2.6.	Bảng tổng hợp các thuật toán	28
3.3.	Lát cắt:	29
3.3.1.	Khái niệm	29
3.3.2.	Khả năng thông qua của lát cắt	29
3.3.3.	Bài toán tìm lát cắt nhỏ nhất	29
4.	Bài tập ứng dụng	33
4.1.	AFINDFLOW.....	33
4.1.1.	AFINDFLOW_SOLUTION	35
4.1.2.	Độ phức tạp.....	35
4.1.3.	Code tham khảo.....	35
4.1.4.	Link tải bộ test	37
4.2.	BDISJCON.....	37
4.2.1.	BDISJCON _SOLUTION	39
4.2.2.	Độ phức tạp.....	40

4.2.3. Code tham khảo.....	40
4.2.4. Link tải bộ test	41
4.3. CSCHEWAR.....	42
4.3.1. CSCHEWAR_SOLUTION	43
4.3.2. Độ phức tạp.....	43
4.3.3. Code tham khảo.....	43
4.3.4. Link tải bộ test	45
4.4. CBADGES.....	46
4.4.1. “Cặp ghép cực đại trên đồ thị hai phía (Bipartite Matching)”	47
4.4.2. Chứng minh:	48
4.4.3. CBADGES_SOLUTION	48
4.4.4. Độ phức tạp.....	49
4.4.5. Code tham khảo.....	49
4.4.6. Link tải bộ test	51
4.5. CODSPORTS	53
4.5.1. CODSPORTS_SOLUTION	54
4.5.2. Độ phức tạp.....	55
4.5.3. Code tham khảo.....	55
4.5.4. Link tải bộ test	58
4.6. CEDGEDES.....	58
4.6.1. CEDGEDES_SOLUTION	60
4.6.2. Độ phức tạp.....	60
4.6.3. Code tham khảo.....	60
4.6.4. Link tải bộ test	63
4.7. DDELETEOWER	64
4.7.1. DDELETEOWER_SOLUTION.....	65
4.7.2. Định lý Konig	65
4.7.3. Độ phức tạp.....	67
4.7.4. Code tham khảo.....	67
4.7.5. Link tải bộ test	70
4.8. DGRIDS	71
4.8.1. DGRIDS_SOLUTION.....	72
4.8.2. Độ phức tạp.....	74
4.8.3. Code tham khảo.....	74
4.8.4. Link tải bộ test	76

4.9.	ECOVERCHESS	77
4.9.1.	ECOVERCHESS_SOLUTION	78
4.9.2.	Độ phức tạp	83
4.9.3.	Code tham khảo	83
4.9.4.	Link tải bộ test	85
5.	Một số bài tập luyện tập:	85
6.	Tài liệu tham khảo	85
7.	Kết luận:	86

BÀI TOÁN LUỒNG CỰC ĐẠI

THUẬT TOÁN FORD – FULKERSON và NHỮNG CẢI TIẾN

1. Giới thiệu

Bài toán “Luồng cực đại” là một bài toán không mới. Tuy nhiên, đây là một nội dung khó, đòi hỏi học sinh phải tư duy tưởng tượng nhiều.

Để học phần này, học sinh cần được trang bị một số kiến thức như là: Đồ thị cơ bản, duyệt trên đồ thị với DFS và BFS, tìm đường đi ngắn nhất trên đồ thị, xử lý đồ thị dạng mảng 2 chiều, kỹ thuật duỗi mảng 2 chiều, cấu trúc dữ liệu (priority_queue, vector...)

Trong khuôn khổ chuyên đề, tôi xin trình bày thuật toán Ford – Fulkerson kinh điển, cùng một số thuật toán cải tiến quan trọng như: Edmonds – Karp (Shortest path), Edmonds – Karp (Fattest path), Edmonds – Karp (Capacity scaling), và thuật toán Dinic. Với những thuật toán này, ta có thể giải quyết các bài toán về luồng cực đại khá tốt với cấu hình tương đối lớn.

Chuyên đề cũng tổng hợp một số các công trình nghiên cứu khác về bài toán luồng cực đại trong nhiều giai đoạn lịch sử. Ngoài ra, chuyên đề cũng trình bày các bài toán liên quan rất quan trọng như: bài toán “Lát cắt cực tiểu” (Min – Cut), bài toán “Cặp ghép cực đại” (Maximum Matching) trên đồ thị hai phía. Bên cạnh đó cũng giới thiệu một số định lý quan trọng như: Konig, Menger.

Chuyên đề hướng đến đối tượng là học sinh giỏi lớp 11, ôn tập chuẩn bị tham gia kỳ thi học sinh giỏi quốc gia hàng năm.

Hệ thống bài tập trong chuyên đề được chia làm các mức độ A, B, C, D, E tương ứng với các mức: cơ bản, trung bình, khá, khó, và rất khó. Với mỗi bài, ký tự đầu trong tên bài cho biết mức độ của bài đó. Giáo viên có thể dựa vào các cấp độ để triển khai giảng dạy cho học sinh.

Rất mong chuyên đề sẽ mang đến những kiến thức bổ ích, cần thiết cho quý Thầy, Cô đồng nghiệp trong hội thảo lần này. Sau đây sẽ là nội dung chi tiết.

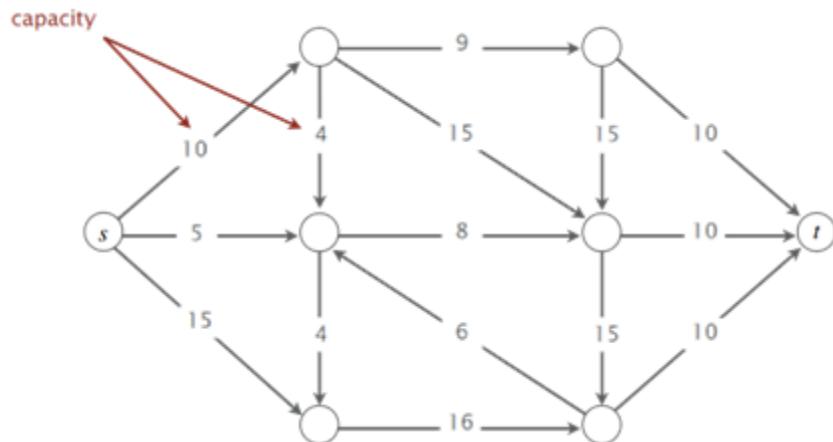
2. Một số khái niệm

2.1. Mạng

Là một đồ thị có hướng $G = (V, E)$. Trong đó:

- Có duy nhất một đỉnh s chỉ có cung đi ra (đỉnh phát – Source).
- Có duy nhất một đỉnh t chỉ có cung đi vào (đỉnh thu - Sink).
- Các cung e được gán với một giá trị $c(e) \geq 0$ gọi là khả năng thông qua (Capacity) của cung e.

Ví dụ:



2.2. Luồng trên mạng

Cho mạng $G(V, E)$. Luồng f trong mạng G là một cách gán cho mỗi cung e một số thực (các bài toán ta xét thường là số nguyên) không âm $f(e)$.

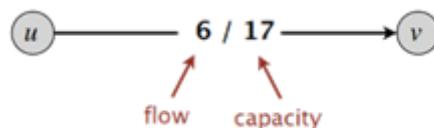
- $f(e)$ gọi là luồng trên cung e .
- $f(e)$ phải thỏa các điều kiện sau:
 - a. Luồng trên mọi cung phải nhỏ hơn khả năng thông qua của cung đó ($0 \leq f(e) \leq c(e)$)
 - b. Tổng luồng trên các cung đi vào (inflow) đỉnh v ($v \neq s, t$) bằng tổng luồng trên các cung đi ra (outflow) khỏi v .

$$\sum_{(w,v) \in E} f(w,v) = \sum_{(v,z) \in E} f(v,z)$$

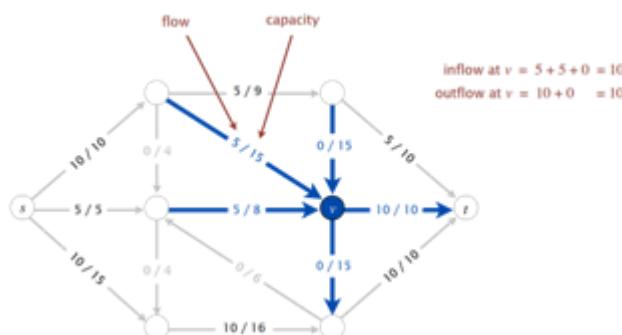
- c. Giá trị của luồng f là số $\text{val}(f)$ với

$$\text{val}(f) = \sum_{(s,w) \in E} f(s,w) = \sum_{(z,t) \in E} f(z,t)$$

Ví dụ:



Hình ảnh một luồng ($f(u,v) = 6$) trên cung (u,v) có khả năng thông qua $c(u,v) = 17$



Hình ảnh về một mạng với luồng cực đại

Nhìn hình trên ta thấy:

- Trên mỗi cạnh e , tồn tại 1 luồng $f(e) \leq c(e)$
- Tại mọi đỉnh $v \in V - \{s,t\}$, ta đều thấy inflow tại $v = \text{outflow}$ tại $v = 10$

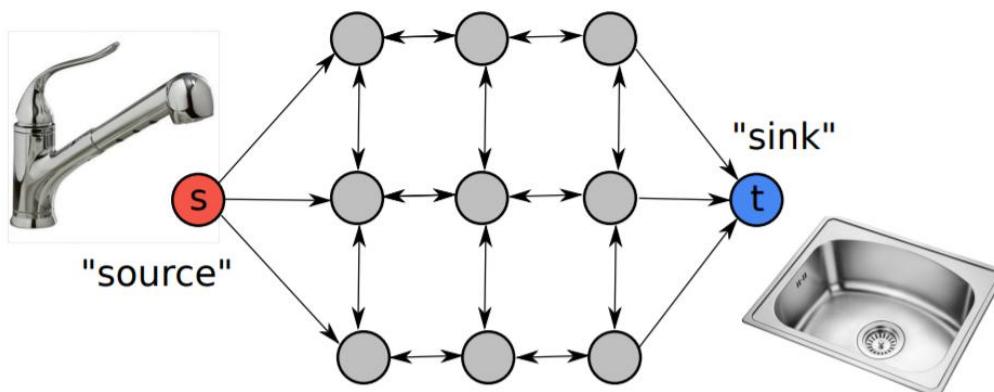
Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

- Giá trị $\text{val}(f^*) = \text{outflow tại } s = \text{inflow tại } t = 25$

2.3. Hình ảnh thực tế

Ví dụ: Nước chảy từ nguồn chứa đến bồn rửa thông qua các ống nước.

- Ta có thể hình dung mỗi cung là một ống dẫn nước. Khả năng thông qua của ống là lượng nước tối đa có thể chảy qua ống mỗi giây. Và luồng là lượng nước thực tế chảy qua ống mỗi giây. Điều này tương ứng với điều kiện a ở trên. Rõ ràng ta không thể cho chảy 1 lượng nước nhiều hơn khả năng thông qua của ống.
- Các đỉnh sẽ đóng vai trò là điểm nối. Nơi nước sẽ chảy ra những ống khác. Đây chính là hình ảnh ứng với điều kiện b ở trên. Tại mỗi điểm nối, lượng nước chảy ra phải được phân bổ đến các ống nước khác nhau. Và hiển nhiên tổng lượng nước phân bổ vào các ống khác phải bằng với lượng nước đi vào điểm nối. Chúng không thể tự biến mất được.
- Đỉnh s là nơi chứa nguồn nước (Source), và nước chỉ có thể thoát đi tại đỉnh t (bồn rửa - Sink). Đây chính là hình ảnh tương ứng với điều kiện c ở trên.



2.4. Bài toán luồng cực đại tổng quát

Cho mạng $G(V,E)$. Hãy tìm luồng f^* trong mạng G sao cho giá trị $\text{val}(f^*)$ của luồng f^* là lớn nhất.

Lưu ý: Từ đây ta dùng ký hiệu f^* để chỉ luồng cực đại trên mạng

3. Thuật toán Ford – Fulkerson¹ và những cải tiến

3.1. Thuật toán gốc và một số khái niệm

Được đề xuất bởi L. R. Ford và D. R. Fulkerson năm 1956

3.1.1. Ý tưởng thuật toán

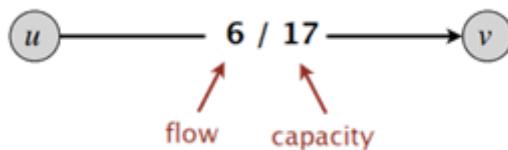
- 1/ Bắt đầu với khởi tạo luồng $f^* = 0$
- 2/ Trong khi tồn tại đường tăng luồng (augmenting path) từ s đến t , ta thực hiện tăng luồng f dọc theo đường tăng luồng tìm được.
- 3/ Trả về giá trị luồng f^*

3.1.2. Mạng thặng dư (Residual Network)

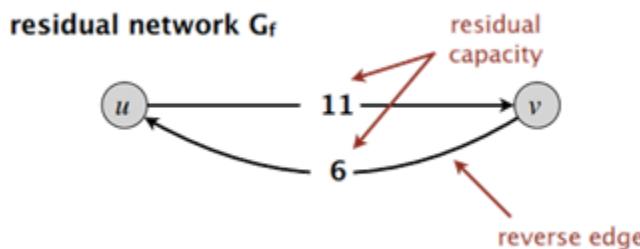
Là mạng G_f có số đỉnh là V , số cung là $2*E$. G_f cho biết khả năng có thể tăng thêm cho luồng f trên mỗi cung. Một số đặc điểm

- Mỗi cung $e(u,v)$ trong mạng thặng dư có một cung ngược $e'(v,u)$ tương ứng. Mỗi cung có một giá trị thặng dư thông qua (Residual Capacity - $RC(e)$). Đây chính là khả năng tăng thêm cho luồng $f(e)$ trên cung e tương ứng. Giá trị $RC(e)$ được tính:
 - ✓ Cung thuận $e(u, v)$: $RC(e) = C(e) - f(e)$
 - ✓ Cung ngược $e'(v, u)$: $RC(e') = f(e)$

Ví dụ: Cung $e(u,v)$ như hình sau



Trong mạng thặng dư tương ứng, ta có cung thuận $e(u,v)$, và cung ngược $e'(v,u)$ lần lượt có $RC(e) = c(e) - f(e) = 17 - 6 = 11$, $RC(e') = f(e) = 6$ như sau:



- Nếu có một đường đi từ s đến T trên mạng thặng dư sao cho các cung trên đường đi đều có giá trị $RC(e) > 0$, thì đây chính là **đường tăng luồng**.
 - ✓ Nếu không còn tìm được đường tăng luồng thì đó luồng là cực đại.
 - ✓ Để tìm đường tăng luồng ta có thể sử dụng các thuật toán DFS, BFS, PFS (Priority First Search) trên mạng thặng dư.
- Nếu gọi B là giá trị thặng dư thông qua nhỏ nhất trên đường tăng luồng tìm được (hay còn gọi B là "**bottleneck capacity**" theo một số sách).

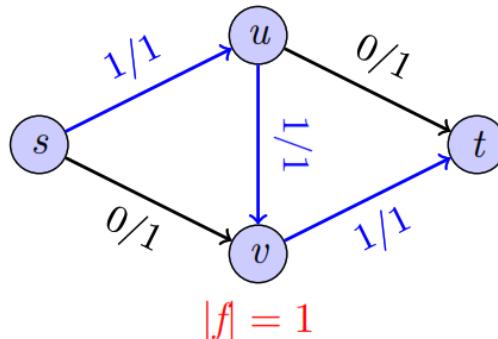
¹ <https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/>

- ✓ Ta có: $B = \min(RC(e))$, với e là cung thuộc đường tăng luồng tìm được. Ta tiến hành cập nhật luồng trên các cung của G thuộc đường tăng luồng như sau:

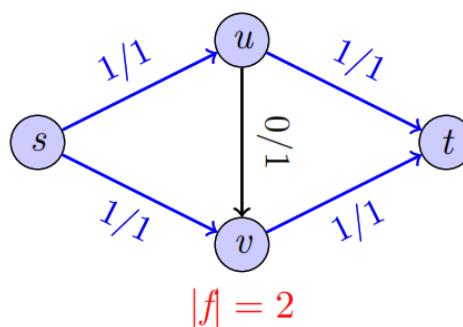
$$f((u,v)) += B \text{ và } f((v,u)) -= B$$

Câu hỏi: Tại sao lại tồn tại cung ngược trên mạng thặng dư?

Ta xét một trường hợp mạng G và đường tăng luồng f là các cung màu xanh như sau:



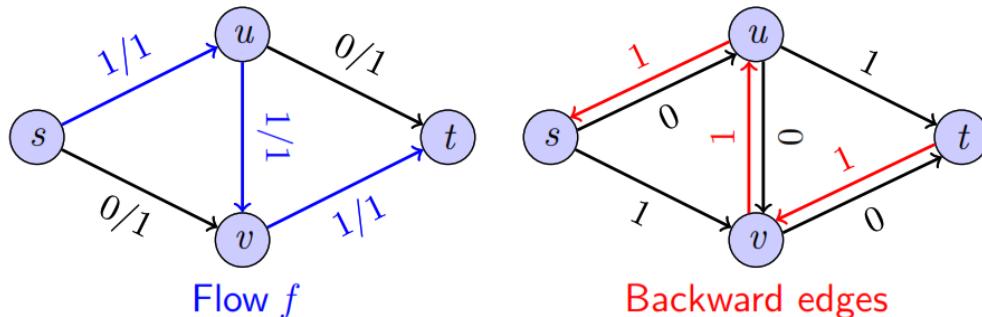
Rõ ràng: ta không thể tăng thêm luồng f lần nào nữa nếu chọn đường tăng luồng như trên. Trong khi giá trị luồng cực đại với đồ thị trên phải là $|f| = 2$ với 2 đường tăng luồng như sau:



Vậy: Có cách nào giúp ta giảm luồng f (trên mỗi cung) đi một lượng bằng với lượng đã tăng trước đó hay không?

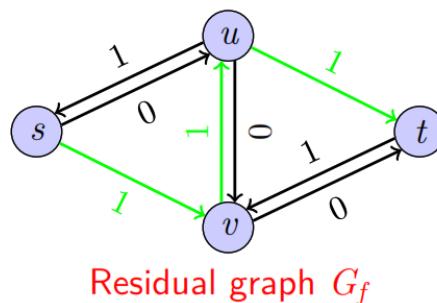
➔ Câu trả lời chính là ta sẽ thêm mỗi cung trong mạng thặng dư một cung ngược (backward edge) tương ứng

Cụ thể trong trường hợp chọn sai đường tăng luồng như hình đầu. Ta có mạng thặng dư khi đã thêm cung ngược như sau:

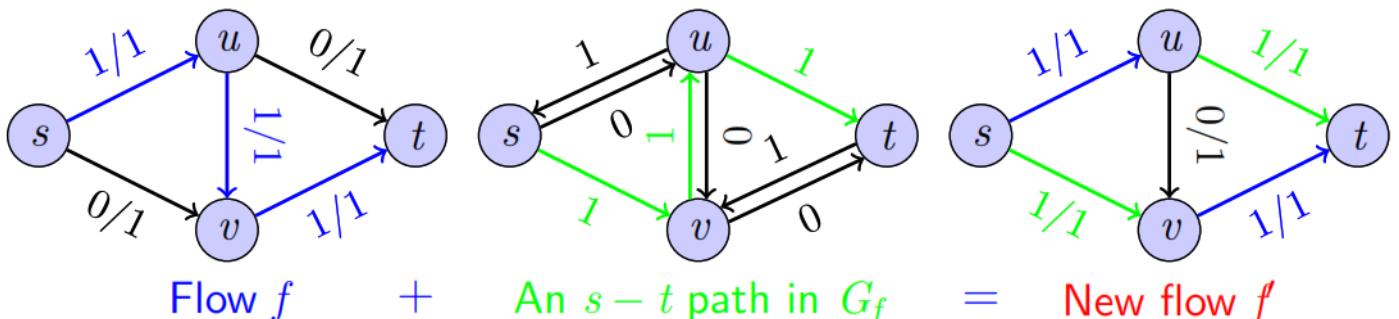


Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

Tiếp theo, ta có thể chọn đường tăng luồng như đường màu xanh lá trong hình sau:



Kết quả ta có quá trình tăng luồng và kết quả như sau:

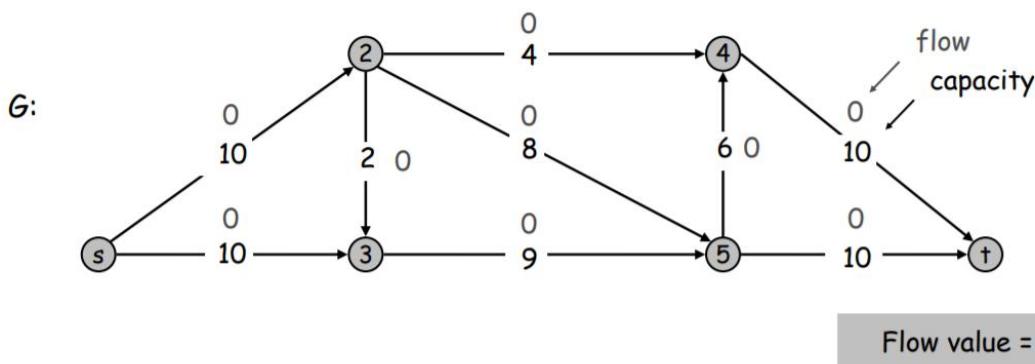


Hình thứ 3 là kết quả sau khi tăng luồng f lần thứ 2. Và khi này luồng f đã đạt cực đại là 2
 $|f^*| = f(s,u) + f(s,v) = 2$

→ Kết luận: Các cung ngược giúp ta có thể giảm luồng $f(e)$ đi 1 lượng đã tăng trước đó.
Trong trường hợp này là cung (u,v)

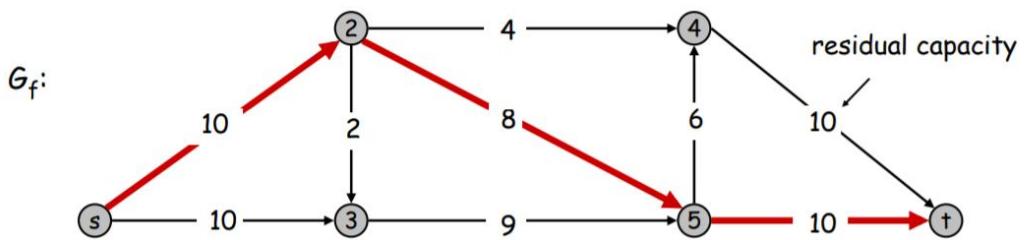
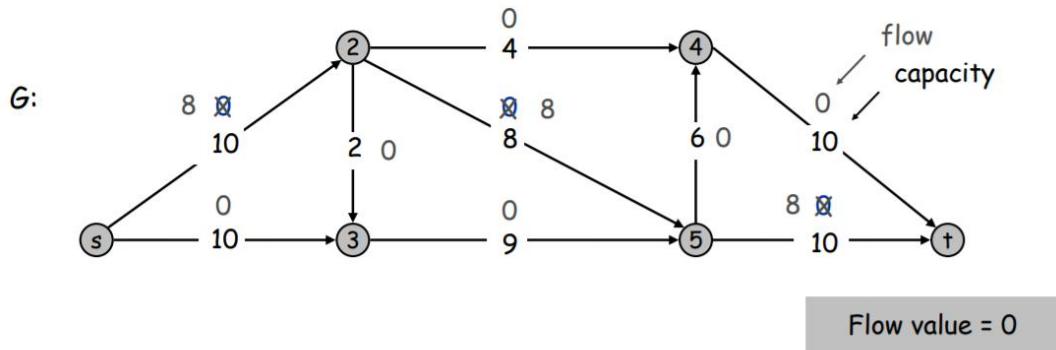
3.1.3. Ví dụ Minh họa thuật toán Ford – Fulkerson

Cho mạng $G(V,E)$ như hình sau:



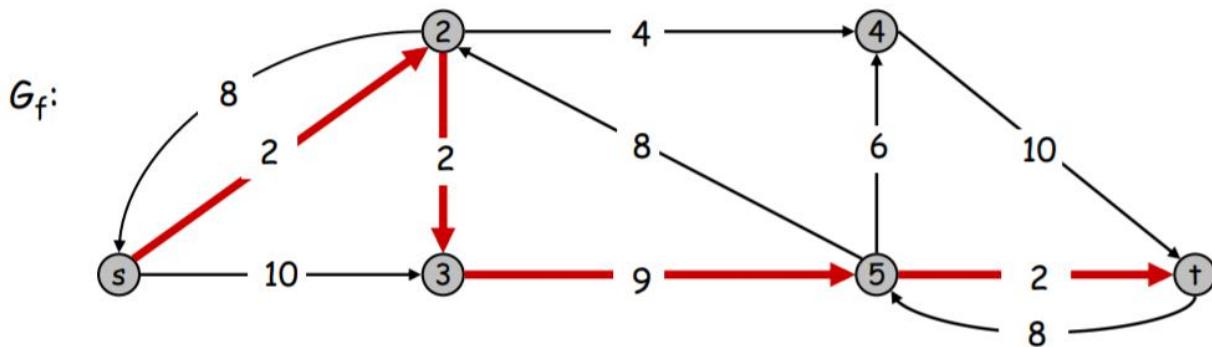
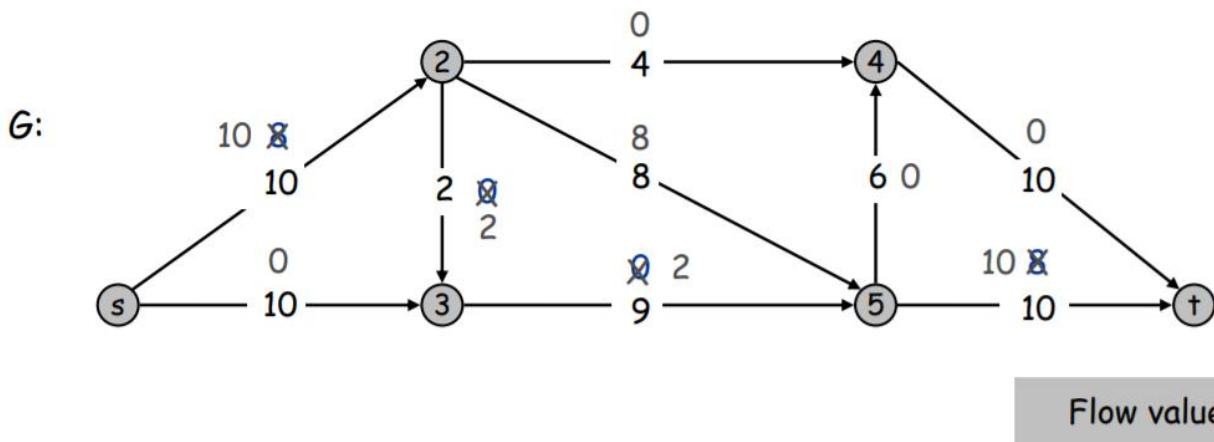
Tăng luồng lần 1:

Ta có mạng thặng dư, đường tăng luồng là: $s, 2, 5, t \rightarrow B = 8 \rightarrow f = 0 + B = 0 + 8 = 8$.



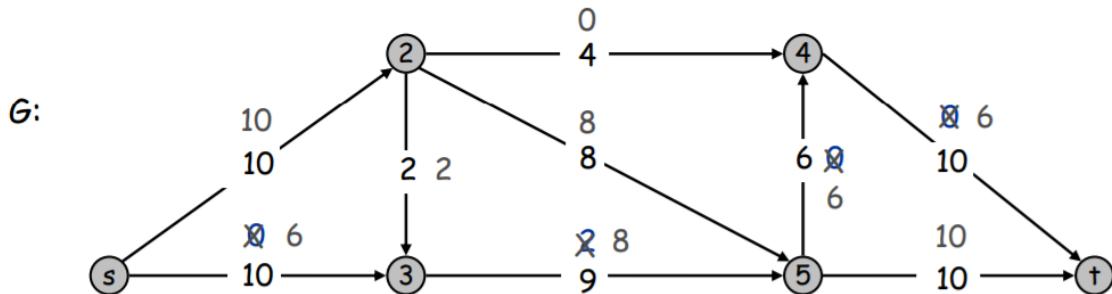
Tăng luồng lần 2:

Ta có mạng thặng dư, đường tăng luồng là: $s, 2, 3, 5, t \rightarrow B = 2 \rightarrow f = 8 + 2 = 10$

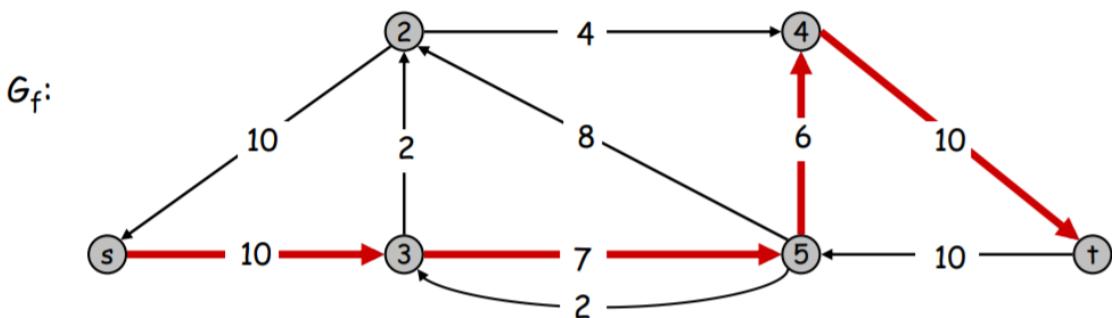


Tăng luồng lần 3:

Ta có mạng thặng dư, đường tăng luồng là: $s, 3, 5, 4, t \rightarrow B = 6 \rightarrow f = 10 + 6 = 16$

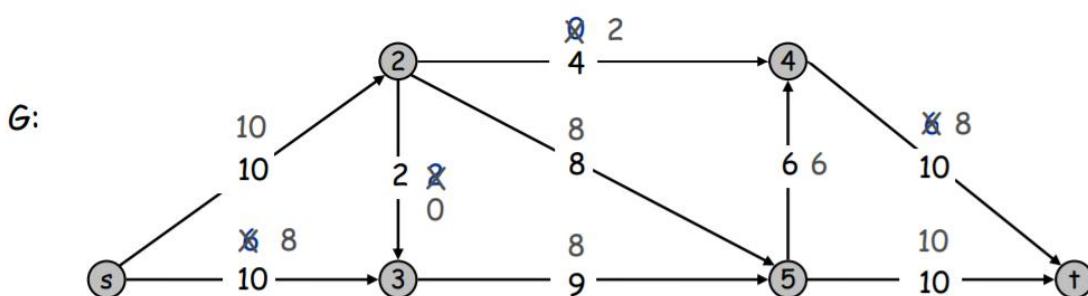


Flow value = 10

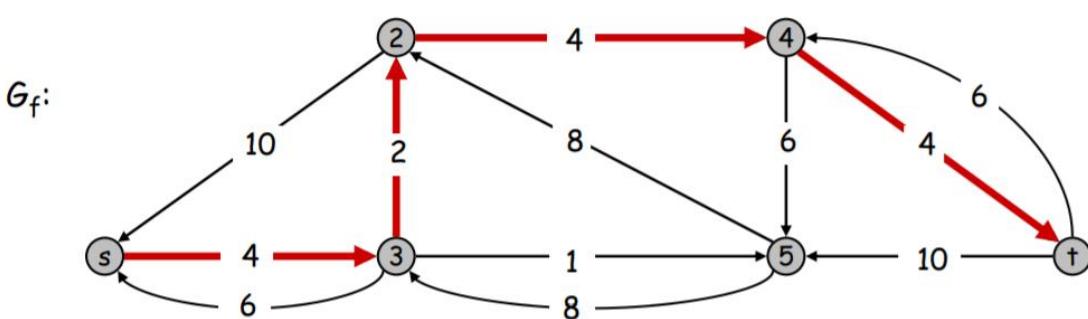


Tăng luồng lần 4:

Ta có mạng thặng dư, đường tăng luồng là: $s, 3, 2, 4, t \rightarrow B = 2 \rightarrow f = 16 + 2 = 18$

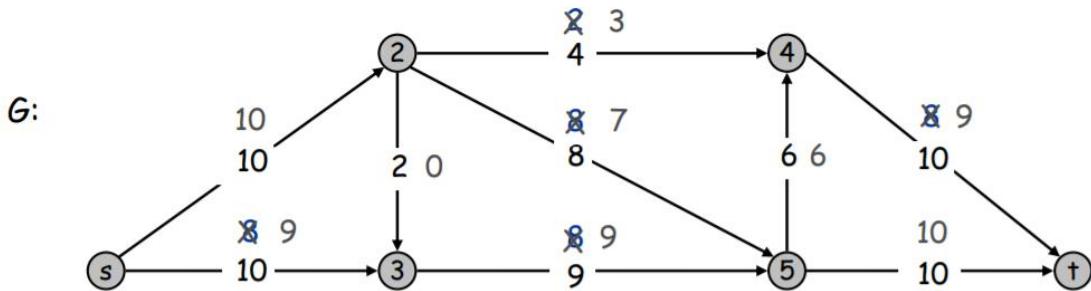


Flow value = 16

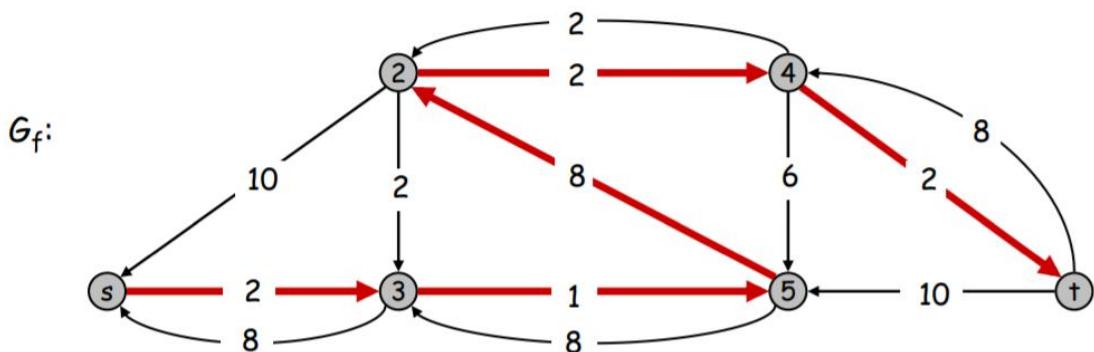


Tăng luồng lần 5:

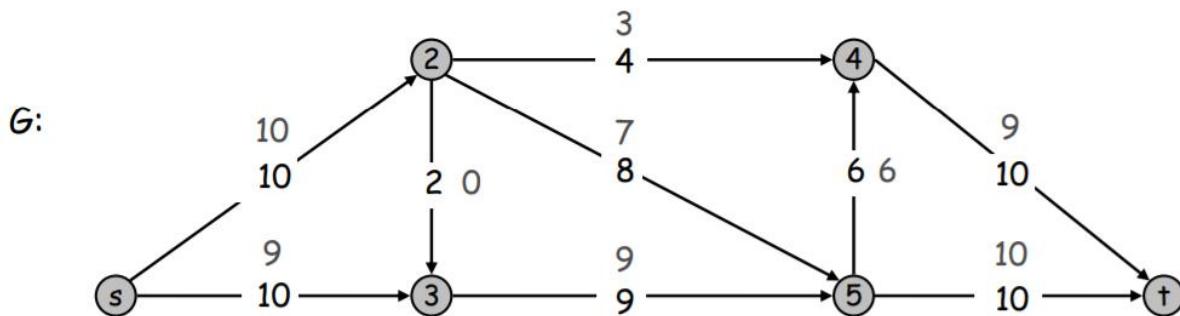
Ta có mạng thặng dư, đường tăng luồng là: $s, 3, 5, 2, 4, t \rightarrow B = 1 \rightarrow f = 18 + 1 = 19$



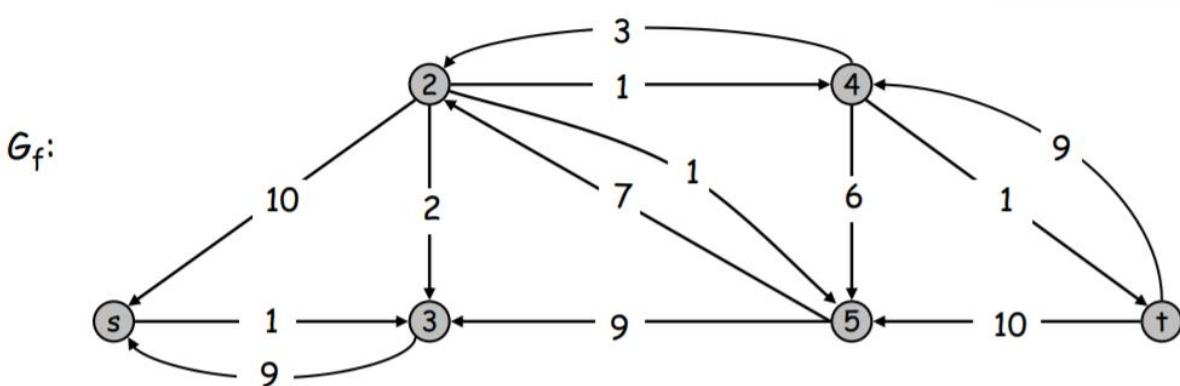
Flow value = 18



Ta thấy không thể tìm thấy đường tăng luồng trên mạng thặng dư G_f



Flow value = 19



→ Luồng đã đạt giá trị cực đại $|f^*| = 19$

3.1.4. Độ phức tạp

Ta phải thực hiện 1 vòng lặp while nhiều lần để xác định đường tăng luồng. Việc xác định đường tăng luồng có thể dùng DFS hoặc BFS, cả 2 đều cho độ phức tạp là $O(E)$. Và trong trường hợp xấu nhất ta sẽ tăng f thêm 1 đơn vị cho mỗi lần lặp cho đến khi đạt cực đại f^* . Vì vậy độ phức tạp thuật toán Ford – Fulkerson là $O(E|f^*|)$.

Thông thường, để giảm số lần tăng luồng, người ta hay tìm cách tăng tốc ở một trong hai bước: số lượng đường tăng luồng và số lần thực hiện thực hiện thuật toán. Vấn đề này ta sẽ tìm hiểu ở phần tiếp theo

3.1.5. Bài toán Luồng cực đại cụ thể

Cho mạng G có n đỉnh và m cạnh, đỉnh phát là 0, và đỉnh thu là $n-1$. Hãy tìm luồng f^* trong mạng sao cho giá trị $\text{val}(f^*)$ của luồng f^* là lớn nhất.

Input: AMAXFLOW.INP

- Dòng 1: Ghi 2 số nguyên n, m cho biết số đỉnh và số cung trong mạng G .
- m dòng tiếp theo: Mỗi dòng ghi 3 số nguyên u, v, c cho biết cung nối từ u đến v có trọng số là c

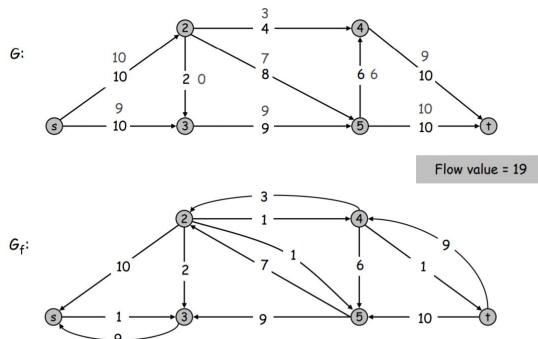
Output: AMAXFLOW.OUT

- Ghi một số nguyên dương cho biết giá trị luồng cực đại tìm được

Ví dụ:

AMAXFLOW.INP	AMAXFLOW.OUT	Mình họa
6 9 1 2 10 1 3 10 2 3 2 2 4 4 2 5 8 3 5 9 4 6 10 5 4 6 5 6 10	19	

Giải thích: Ta có luồng trên mạng trong test mẫu là:



Ràng buộc: $0 < n \leq 500$; $0 < m \leq 1000$; $0 < c \leq 10^9$

3.1.6. Link tải bộ test

<https://drive.google.com/drive/folders/1AHL0TEK0sAWbD42U1xU6zbvkLLfjw3WT?usp=sharing>

3.2. Cài đặt thuật toán Ford – Fulkerson và những thuật toán cải tiến

3.2.1. Thuật toán Ford – Fulkerson phiên bản gốc

Qui ước:

- Gọi $\text{par}[v]$: lưu trữ đỉnh u là cha trực tiếp của v . Khởi tạo ban đầu các $\text{par}[u] = -1$ ($u \neq s$), riêng $\text{par}[s] = 0$
- Gọi $a[u]$: lưu trữ danh sách các đỉnh kề với u .
- Gọi $RC[u][v]$: Giá trị khả năng tăng dư thông qua trên mạng tăng dư của cung (u,v) . Khởi tạo ban đầu với các cung (u,v) ta có: $RC[u][v] = C[u][v]$, $RC[v][u] = 0$;

Sử dụng DFS để tìm đường tăng luồng: Cách cài đặt này có ưu điểm là dễ cài đặt nhưng thông thường số lần tăng luồng là khá lớn nên thời gian xử lý chậm hơn BFS.

Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define N 205

vector<int> adj[N];
int RC[N][N];
int par[N], n, m, s, t, add_flow;
//-----
void nhap()
{
    cin >> n >> m;
    s = 1; t = n;
    for (int i = 1; i <= m; i++)
    {
        int u, v, c;
        cin >> u >> v >> c;
        adj[u].push_back(v);
        adj[v].push_back(u);
        RC[u][v] = c;
    }
}
//-----
void dfs(int u)
{
    for (auto v : adj[u])
        if (par[v] == -1 && RC[u][v])
        {
            par[v] = u;
            add_flow = min(add_flow, RC[u][v]);
            dfs(v);
        }
}
//-----
int Maxflow(int s, int t, int n)
{
    int flow = 0;
    while (1)
    {
        for (int i = 1; i <= n; i++)
```

```

        par[i] = -1;
par[s] = 0;
add_flow = INT_MAX;
dfs(s);
if(par[t]==-1)
    break;
flow += add_flow;
int v = t;
while(v!=s)
{
    int u = par[v];
    RC[v][u]+=add_flow;//Cung ngược
    RC[u][v]-=add_flow;//Cung xuôi
    v = u;
}
return flow;
}
//-----
int main()
{
    freopen("AMAXFLOW.INP","r",stdin);
    freopen("AMAXFLOW.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    nhap();
    cout<<Maxflow(s,t,n);
    return 0;
}

```

3.2.2. Thuật toán Edmonds – Karp: Shortest path

Thuật toán được đề xuất bởi Jack Edmonds và Richard Karp vào năm 1970.

Ý tưởng

Edmonds – Karp sử dụng BFS để tìm đường tăng luồng ngắn nhất (với ít cạnh nhất).

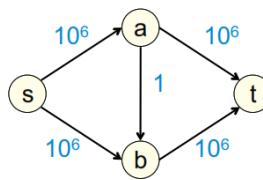
Độ phức tạp²

- ✓ Mỗi lần tăng luồng sẽ có 1 cung (u,v) đạt độ bão hòa $(f(u,v) = c(u,v))$. Cung này sẽ không xuất hiện (mà cung ngược của nó sẽ xuất hiện) trong mạng thặng dư. Ta nói cung (u,v) bị biến mất khỏi mạng thặng dư.
- ✓ Cung bị biến mất có thể sẽ xuất hiện lại trong mạng thặng dư ở các bước sau. Nghĩa là nó sẽ không tham gia vào các đường tăng luồng lần nào nữa cho đến khi độ dài đường tăng luồng ngắn nhất trên mạng thặng dư được tăng lên (khi này là lúc cung ngược của nó tham gia vào đường tăng luồng). Mỗi cạnh (u,v) sẽ biến mất không quá $O(V/2)$ lần. Vì vậy, số lần tăng luồng không quá $O(VE^2)$.
- ✓ Chi phí tìm đường tăng luồng với BFS tốn $O(E)$
- ➔ Độ phức tạp của thuật toán đạt $O(VE^2)$.

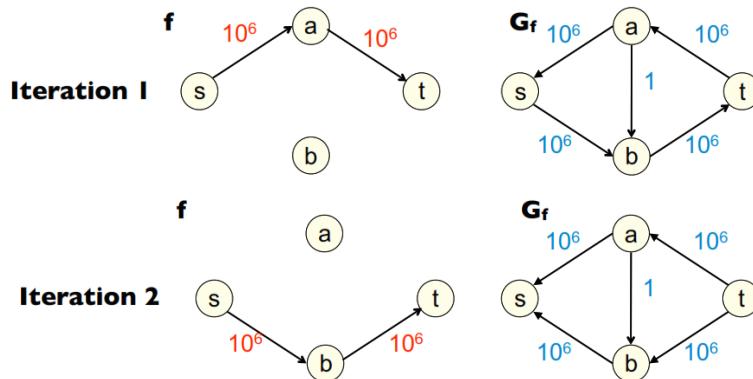
² <https://www.giaithuatlaptrinh.com/?p=1539>

Ví dụ

Xét đồ thị như hình:



- Thật tế nếu thứ tự chọn đường tăng luồng là: $s - a - b - t$, $s - b - a - t$, $s - a - b - t$, ... Với cách chọn các đường tăng luồng thế này mỗi lần luồng chỉ tăng thêm 1 cho đến khi đạt cực đại.
- Vì vậy thuật toán Edmond – Karp được đề xuất tìm đường tăng luồng ngắn nhất. Cụ thể các lần lặp như sau:



Thuật toán kết thúc sau 2 lần lặp với luồng $f^* = 2 \cdot 10^6$

Code tham khảo

```

#include<bits/stdc++.h>
using namespace std;
#define N 205
vector<int> adj[N];
int RC[N][N];
int par[N], n, m, s, t, add_flow;
//-----
void nhap()
{
    cin>>n>>m;
    s=1; t=n;
    for(int i=1; i<= m; i++)
    {
        int u, v, c;
        cin>>u>>v>>c;
        adj[u].push_back(v);
        adj[v].push_back(u);
        RC[u][v] = c;
    }
}
//-----
bool bfs(int s, int t, int n)
{
  
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
for (int i = 1 ; i <= n ; i++)
    par[i] = -1;
par[s] = 0;
add_flow = INT_MAX;

queue< int > q;
q.push(s);
while (!q.empty())
{
    int u = q.front();
    q.pop();
    for (auto v:adj[u])
        if (par[v] == -1 && RC[u][v])
        {
            par[v] = u;
            add_flow = min(add_flow,RC[u][v]);
            if (v == t)
                return true;
            q.push(v);
        }
}
return false;
}

int Maxflow(int s, int t, int n)
{
    int flow = 0;
    while(bfs(s,t,n))
    {
        flow += add_flow;
        int v = t;
        while (v!=s)
        {
            int u = par[v];
            RC[v][u] += add_flow;
            RC[u][v] -= add_flow;
            v = u;
        }
    }
    return flow;
}

int main()
{
    freopen("AMAXFLOW.INP","r",stdin);
    freopen("AMAXFLOW.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    nhap();
    cout<<Maxflow(s,t,n);
    return 0;
}
```

3.2.3. Thuật toán Edmonds – Karp: Fattest Path

Thuật toán này tìm ra đường tăng luồng lớn nhất (fattest path) và khá giống với thuật toán Dijkstra tìm đường đi ngắn nhất vì thay vì sử dụng BFS để tìm đường tăng luồng, thì ta sử dụng hàng đợi ưu tiên priority_queue.

Ý tưởng: Thay vì lưu nhẫn khoảng cách các đỉnh vào priority_queue như trong Dijkstra, ta sẽ lưu các giá trị max “bottleneck capacity” ($B_{\max}[u]$) với từng đỉnh. Trong quá trình tìm đường tăng luồng, mỗi lần ta chọn đỉnh u có giá trị $B_{\max}[u]$ lớn nhất để đi tiếp, dùng đỉnh u để tối ưu các giá trị $B_{\max}[v]$ (với v là đỉnh kề với u)

Độ phức tạp³

- Số lần tăng luồng là $O(E \ln(|f^*|) + 1)$
- Chi phí tìm đường tăng luồng là $O(E \log V)$.

→ Độ phức tạp thuật toán đạt xấp xỉ $O(E^2 \log(V) \ln(|f^*|))$. Cũng có nhiều tài liệu chứng minh thuật toán có độ phức tạp $O(E \log |f^*| * (E + V \log V))$

Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define N 205
typedef pair<int, int> node;

vector<int> adj[N];
int RC[N][N];
int par[N], n, m, s, t, add_flow;
//-----
void nhap()
{
    cin >> n >> m;
    s = 1; t = n;
    for (int i = 1; i <= m; i++)
    {
        int u, v, c;
        cin >> u >> v >> c;
        adj[u].push_back(v);
        adj[v].push_back(u);
        RC[u][v] = c;
    }
}
//-----
bool pfs(int s, int t, int n)
{
    int Bmax[n+1] = {0};
    for (int i = 0; i <= n; i++)
        par[i] = -1;
    par[s] = 0; Bmax[s] = INT_MAX;
    add_flow = INT_MAX;
    priority_queue<node> pq;
    pq.push({Bmax[s], s});
    while (!pq.empty())
    {
        int curNode = pq.top().second;
        pq.pop();
        if (curNode == t)
            break;
        for (int i : adj[curNode])
        {
            if (par[i] == -1 || Bmax[i] < Bmax[curNode])
            {
                par[i] = curNode;
                Bmax[i] = min(Bmax[i], Bmax[curNode]);
                pq.push({Bmax[i], i});
            }
        }
    }
    if (par[t] == -1)
        return false;
    else
        return true;
}
```

³ Thuật toán đường béo (fattest path): <https://www.giaithuatlaptrinh.com/?p=1539>

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
    pq.push({Bmax[s], s});
    while (!pq.empty())
    {
        pair<int, int> tmp;
        tmp = pq.top();
        pq.pop();
        int B = tmp.first;
        int u = tmp.second;
        add_flow=min(add_flow,B);
        if(u==t) return true;
        for(auto v:adj[u])
            if(par[v]==-1 && min(B,RC[u][v]) > Bmax[v])
            {
                Bmax[v] = min(B,RC[u][v]);
                par[v] = u;
                pq.push({Bmax[v], v});
            }
        }
        return false;
    }
//-----
int Maxflow(int s, int t, int n)
{
    int flow = 0;
    while(pfs(s,t,n))
    {
        flow += add_flow;
        int v = t;
        while(v!=s)
        {
            int u = par[v];
            RC[v][u]+=add_flow;
            RC[u][v]-=add_flow;
            v = u;
        }
    }
    return flow;
}
//-----
int main()
{
    freopen("AMAXFLOW.INP", "r", stdin);
    freopen("AMAXFLOW.OUT", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    nhap();
    cout<<Maxflow(s,t,n);
    return 0;
}
```

3.2.4. Thuật toán Edmonds – Karp, Dinitz: Capacity scaling

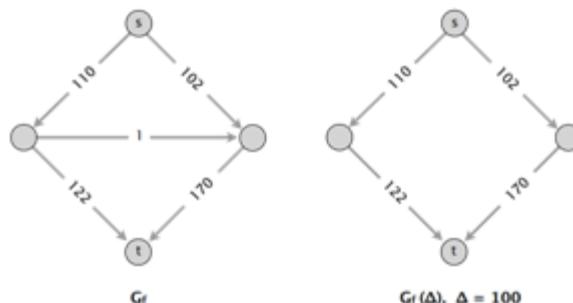
Ý tưởng

Thuật toán cải tiến giúp tìm các đường tăng luồng có giá trị bottleneck capacity "lớn" thông qua các bước điều chỉnh mức độ khả năng thông qua từ lớn đến nhỏ.

Một số khái niệm

- Gọi Δ là: giới hạn độ lớn khả năng thông qua của các cạnh trong mạng thặng dư (RC(e))
- Gọi $G_f(\Delta)$: là đồ thị con của mạng thặng dư. Trong đó các cạnh đều có giá trị $RC(e) \geq \Delta$
- ➔ Nghĩa là bất cứ đường tăng luồng nào trên $G_f(\Delta)$ sẽ có giá trị bottleneck capacity $\geq \Delta$

Ví dụ:



Thuật toán

Khởi tạo giá trị $f(e) = 0$ cho mỗi cạnh

Δ = giá trị lũy thừa 2 lớn nhất $\leq C_{\max}$. Với $C_{\max} = \text{Max}(c(e))$

While ($\Delta \geq 1$)

{

Mạng thặng dư cần xét: $G_f(\Delta)$

While (Tồn tại đường tăng luồng P từ S → T trong $G_f(\Delta)$)

{

$f \leftarrow f + B$ //giá trị bottleneck capacity, $B \geq \Delta$

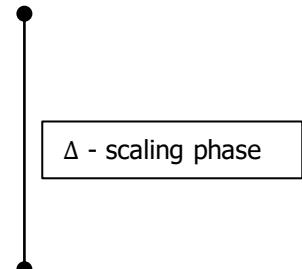
cập nhật $G_f(\Delta)$

}

$\Delta \leftarrow \Delta/2$

}

return f;



Độ phức tạp⁴

- Vòng lặp while bên ngoài sẽ thực hiện $1 + \log C_{\max}$ lần
- Mỗi lần thực hiện Δ - scaling phase tốn $O(E)$
- Chi phí tìm đường tăng luồng với BFS tốn $O(E)$
- ➔ Tổng độ phức tạp $O(E^2 \log C_{\max})$

⁴ <http://people.csail.mit.edu/moitra/docs/6854lec8.pdf>

Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define N 205

vector<int> adj[N];
int RC[N][N];
int par[N], n, m, s, t, add_flow, delta;
//-----
int largest_power(int x)//Tính lũy thừa 2 cao nhất <= N
{
    //changing all right side bits to 1.
    x = x | (x>>1);
    x = x | (x>>2);
    x = x | (x>>4);
    x = x | (x>>8);
    x = x | (x>>16);

    return (x+1)>>1;
}
//-----
void nhap()
{
    cin>>n>>m;
    s=1; t=n;
    for(int i=1; i<= m; i++)
    {
        int u,v,c;
        cin>>u>>v>>c;
        adj[u].push_back(v);
        adj[v].push_back(u);
        RC[u][v] = c;
        delta = max(delta,c);
    }
    delta = largest_power(delta);
}
//-----
bool bfs(int s, int t, int n)
{
    for (int i = 1 ; i <= n ; i++)
        par[i] = -1;
    par[s] = 0;
    add_flow = INT_MAX;
    queue< int > q;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (auto v:adj[u])
        {
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
//Xét các cạnh trên mạng thặng dư có giá trị RC(e) từ delta trở lên
if (par[v] == -1 && RC[u][v]>=delta)
{
    par[v] = u;
    add_flow = min(add_flow,RC[u][v]);
    if (v == t)
        return true;
    q.push(v);
}
}
return false;
}
//-----
int Maxflow(int s,int t, int n)
{
    int flow = 0;
    while(delta>=1)
    {
        while(bfs(s,t,n))
        {
            flow += add_flow;
            int v = t;
            while(v!=s)
            {
                int u = par[v];
                RC[v][u]+=add_flow;
                RC[u][v]-=add_flow;
                v = u;
            }
        }
        delta /= 2;
    }
    return flow;
}
//-----
int main()
{
    freopen("AMAXFLOW.INP","r",stdin);
    freopen("AMAXFLOW.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    nhap();
    cout<<Maxflow(s,t,n);
    return 0;
}
```

3.2.5. Thuật toán Dinic

Do Y. A. Dinitz đề xuất năm 1970, đây là một thuật toán cải tiến tốc độ nhanh hơn thuật toán Edmonds – Karp thông qua việc tìm luồng cực đại trên toàn bộ mạng.

Một số khái niệm:

- **Blocking flow:** Một “blocking flow” của một mạng G nào đó, là một luồng f từ s đến t , sao cho có ít nhất một cạnh trên đường đi đạt độ bão hòa. Luồng f không yêu cầu phải lớn nhất.
- **Level graph:** là một đồ thị mà mỗi đỉnh đều được gán cấp. Cấp của mỗi đỉnh u là khoảng cách ngắn nhất từ s đến u (số cạnh ít nhất).

Tương tự thuật toán Edmond Karp, Dinic cũng có các đặc điểm:

- Luồng đạt cực đại nếu không có đường đi từ S đến T trên mạng thặng dư
- BFS được sử dụng trong vòng lặp. Tuy nhiên, cách cài đặt có sự khác biệt.

Điểm khác biệt trong cách sử dụng BFS như sau:

- Trong thuật toán Edmond Karp, sử dụng BFS tìm đường tăng luồng và cập nhật luồng dọc theo con đường tìm được.
- Trong thuật toán Dinic, ta dùng BFS để kiểm tra xem có thể tăng luồng thêm không? và xây dựng đồ thị cấp (level graph). Mỗi lần level graph được xây dựng, ta sẽ thử gửi nhiều luồng (send multiple flow) sử dụng level graph. Những luồng đạt blocking flow sẽ được cộng vào luồng chính. Đây là lý do thuật toán hoạt động nhanh hơn Edmond Karp (chỉ xét 1 luồng dọc theo đường BFS tìm được)

Ý tưởng thuật toán Dinic như sau:

- 1/ Khởi tạo mạng thặng dư (residual graph) theo mạng G đã cho
- 2/ Dùng BFS trên G để xây dựng level graph (gán cấp cho mỗi đỉnh), đồng thời kiểm tra xem có thể tăng thêm cho luồng hay không:
 - a) Nếu không thể tăng thêm cho luồng thì kết thúc, luồng đạt cực đại.
 - b) Ngược lại, thực hiện gửi thử nhiều luồng (dùng DFS) trên mạng thặng dư sử dụng level graph cho đến khi đạt được blocking flow. Mỗi luồng đạt blocking flow sẽ được cộng dồn vào luồng cực đại cần tìm. Việc sử dụng level graph ở đây nghĩa là đường đi qua các đỉnh nên theo thứ tự cấp $0, 1, 2, \dots$ từ S đến T .

Độ phức tạp⁵:

- Thuật toán thực hiện tối đa V lần.
- Việc gửi thử nhiều luồng trên level graph dùng DFS tốn $O(V)$. Để tìm được blocking flow nhanh chóng, ta sẽ bỏ đi những cạnh trong mạng thặng dư đã đạt bão hòa khi xây

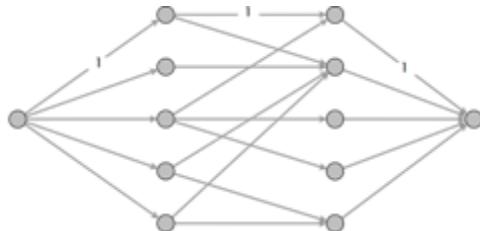
⁵ <https://cp-algorithms.com/graph/dinic.html#toc-tgt-5>

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

dụng level graph. Các cạnh này sẽ bị xóa đi nhiều nhất E lần. Vì vậy chi phí cho mỗi lần thuật toán thực hiện là $O(EV)$

→ Tổng độ phức tạp thuật toán là $O(EV^2)$

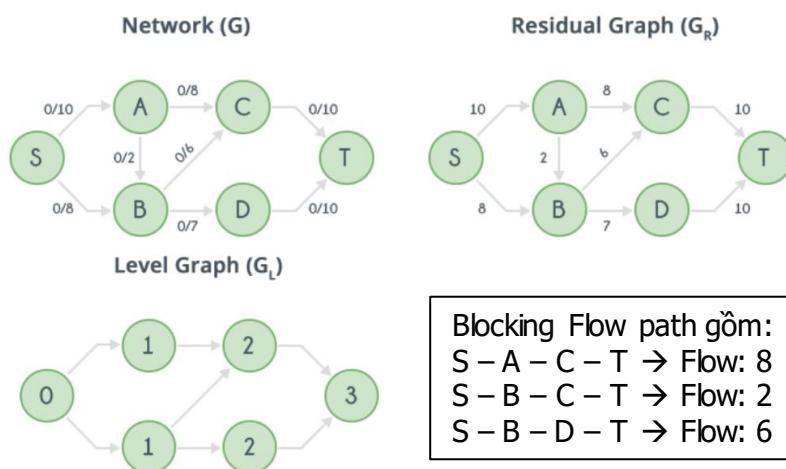
Unit networks: Là mạng mà tất cả các cạnh có cùng khả năng thông qua. Bất kỳ đỉnh nào khác s, t đều có duy nhất 1 cung vào và 1 cung ra. Đây chính xác là trường hợp ứng dụng trong việc tìm cặp ghép cực đại với luồng.



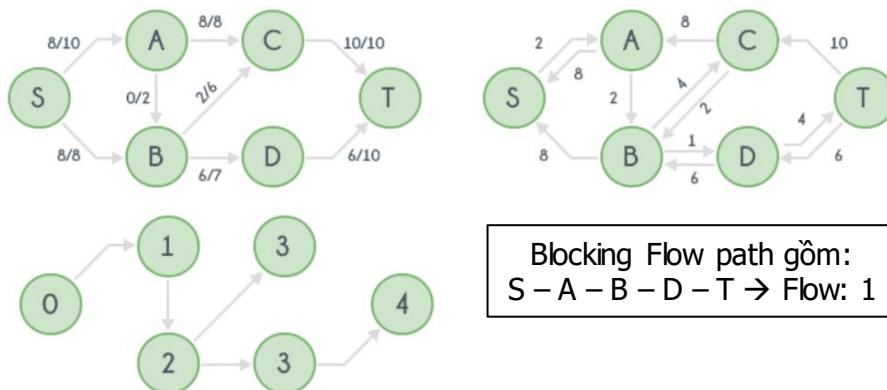
- Trong trường hợp này, độ phức tạp của thuật toán đạt $O(E\sqrt{V})$

Demo:

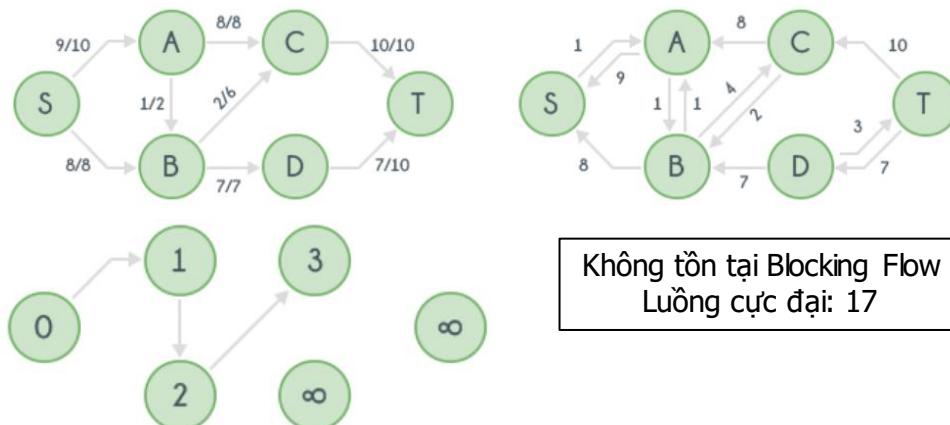
Lượt 1:



Lượt 2:



Lượt 3:



Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define N 205

int n,m,s,t,level[N],par[N], RC[N][N];
vector<int> adj[N];
//-----
void nhap()
{
    cin>>n>>m;
    s=1; t=n;
    for(int i=1; i<=m; i++)
    {
        int u,v,c;
        cin>>u>>v>>c;
        adj[u].push_back(v);
        adj[v].push_back(u);
        RC[u][v] = c;
    }
}
//-----
// Kiểm tra xem có thể tăng thêm luồng được không?
// Xây dựng level graph
bool BFS(int s, int t, int n)
{
    for (int i = 1 ; i <= n ; i++)
        level[i] = -1,par[i] = -1;

    level[s] = 0; // Level của đỉnh phát s
    par[s] = 0;

    queue< int > q;
    q.push(s);

    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for(auto v:adj[u])
            if (level[v] < 0 && RC[u][v])

```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
{  
    level[v] = level[u] + 1;  
    par[v] = u;  
    q.push(v);  
}  
  
}  
//Kết quả kiểm tra có tăng thêm luồng được không?  
return level[t] < 0 ? false : true;  
}  
-----  
// Hàm sendFlow được cài đặt tương tự DFS, dùng để gửi các luồng trên đồ thị thặng dư  
// sau khi BFS đã kiểm tra có thể tăng thêm luồng và xây dựng level graph,  
// cho đến khi đạt được blocking flow thì dừng  
// Các tham số hàm gồm:  
int sendFlow(int u, int flow)  
{  
    if (u == t)  
        return flow;  
    //Duyệt qua lần lượt từng cạnh kề với u, tăng số lượng cạnh đã duyệt  
    for(auto v:adj[u])  
    {  
        // Lấy cạnh kề tiếp theo trong danh sách kề của u  
        if (level[v] == level[u]+1 && RC[u][v])  
        {  
            int curr_flow = min(flow, RC[u][v]); //Tìm luồng nhỏ nhất trên đường đi từ u --> t  
            int add_flow = sendFlow(v, curr_flow); //Tiếp tục gửi luồng nhỏ nhất tìm được sang đỉnh kề v  
  
            //Nếu luồng trả về > 0  
            if (add_flow > 0)  
            {  
                //Thêm luồng vào cung ngược trên đồ thị thặng dư  
                RC[v][u] += add_flow;  
                //Giảm luồng tại cung thuận trên đồ thị thặng dư  
                RC[u][v] -= add_flow;  
                return add_flow;  
            }  
        }  
    }  
  
    return 0;  
}  
-----  
int DinicMaxflow(int s, int t, int n)  
{  
    if (s == t)  
        return -1;  
    int flow = 0;  
    while (BFS(s, t, n)) //Trong khi còn tăng luồng được  
        while (int add_flow = sendFlow(s, INT_MAX)) //Trong khi tìm được luồng tăng  
            flow += add_flow;  
    return flow;  
}  
-----  
int main()  
{
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```

freopen("AMAXFLOW.INP","r",stdin);
freopen("AMAXFLOW.OUT","w",stdout);
ios_base::sync_with_stdio(false);
cin.tie(0);
nhap();
cout << DinicMaxflow(s,t,n);
return 0;
}

```

3.2.6. Bảng tổng hợp các thuật toán

Năm	Người đề xuất	Cách thức xử lý	Độ phức tạp
1951	Dantzig	Simplex	EV^2C
1955	Ford, Fulkerson	Augmenting path	EVC
1970	Edmond – Karp	Shortest path	E^2V
1970	Edmond – Karp	Max capacity (Fattest path)	$E^2\log(V)\ln(f^*)$
1970	Dinitz	Improved shortest path	EV^2
1972	Edmonds – Karp, Dinitz	Capacity scaling	$E^2\log C$
1974	Karzanov	Preflow – push	V^3
1983	Sleator – Tarjan	Dynamic trees	$EV\log V$
1985	Dinitz – Gabow	Improved capacity scaling	$EV\log C$
1988	Goldberg – Tarjan	FIFO preflow – push	$EV\log(V^2/E)$
1998	Goldberg – Rao	Binary blocking flows	$E^{3/2}\log(V^2/E)\log C$
2013	Orlin	Compact networks	EV
2014	Lee – Sidford	interior – point methobs	$EV^{1/2}\log C$
2016	Madry	Electrical flows	$E^{10/7}C^{1/7}$
...			

Thông lượng của 1 cung trên G thuộc đoạn $[1, C]$

Trong khuôn khổ chuyên đề, tôi đã trình bày chi tiết về 5 thuật toán được tô đỏ ở bảng trên. Thầy, Cô có thể mở rộng tìm hiểu thêm các thuật toán cải tiến gần đây để đạt độ phức tạp tốt hơn.

3.3. Lát cắt:

3.3.1. Khái niệm

Lát cắt (A, B) của mạng $G = (V, E)$ là một cách phân hoạch tập V thành hai tập A và $B = V \setminus A$. Trong đó:

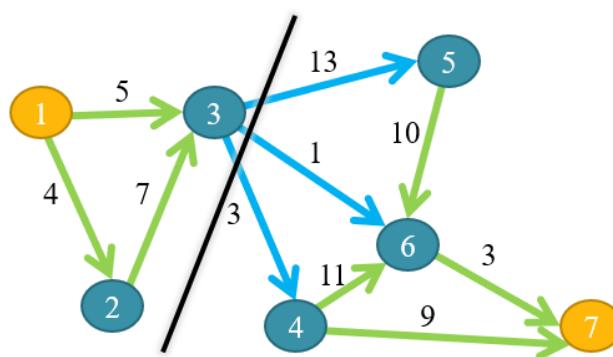
$$s \in A, t \in B$$

3.3.2. Khả năng thông qua của lát cắt

Khả năng thông qua của lát cắt $s-A, B$ là giá trị:

$$cap(A, B) = \sum_{\substack{v \in A \\ w \in B}} c(v, w)$$

Ví dụ: Với $s = 1$, $t = 7$ và mạng $G(V, E)$. Xét lát cắt (A, B), với $A = \{1, 2, 3\}$, $B = \{4, 5, 6, 7\}$, ta có hình minh họa sau:



Khả năng thông qua của lát cắt trên là:

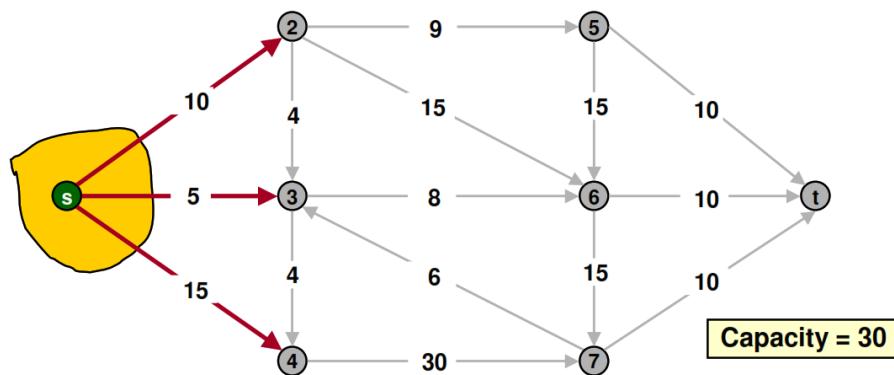
$$cap(A, B) = c(3, 5) + c(3, 6) + c(3, 4) \Rightarrow c(A, B) = 13 + 1 + 3 = 17$$

3.3.3. Bài toán tìm lát cắt nhỏ nhất

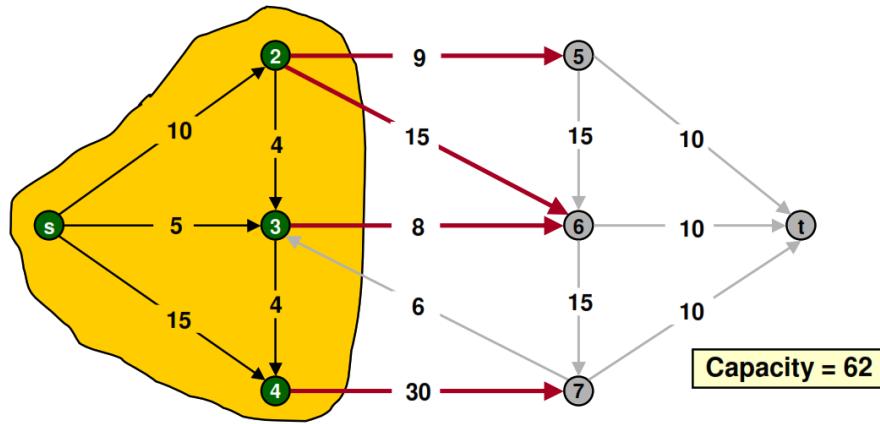
Bài toán: Có nhiều lát cắt khác nhau trên một mạng G . Mỗi lát cắt cho ta một cách phân hoạch tập đỉnh V của G thành 2 phần A, B như đã nói ở trên. Trong số các lát cắt, hãy tìm lát cắt có khả năng thông qua ($cap(A, B)$) nhỏ nhất.

Ta quan sát một số ví dụ về lát cắt sau:

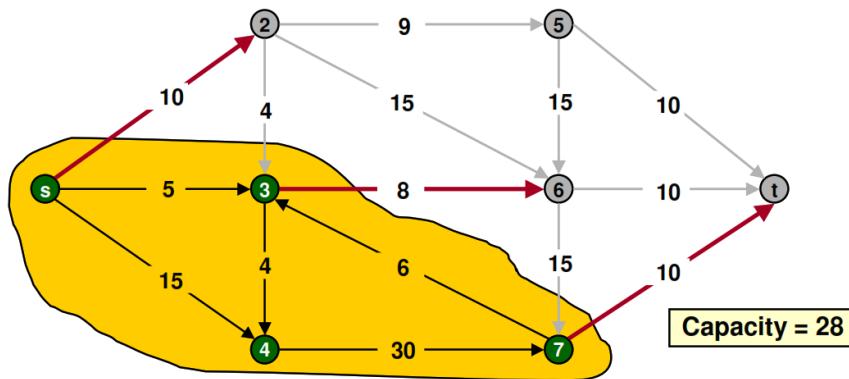
Ví dụ 1: $A = \{s\}$, $B = \{2, 3, 4, 5, 6, 7, t\} \rightarrow cap(A, B) = 30$



Ví dụ 2: A = {s, 2, 3, 4}, B = {5, 6, 7, t} $\rightarrow \text{cap}(A, B) = 62$



Ví dụ 3: A = {s, 3, 4, 7}, B = {2, 5, 6, t} $\rightarrow \text{cap}(A, B) = 28$.



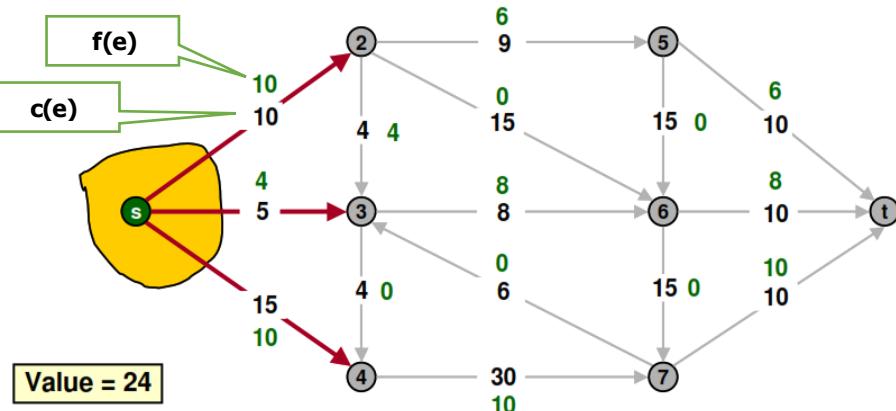
Định lý 1:

Cho f là một luồng trên mạng G . Gọi (A, B) là một lát cắt trên G . Thì tổng luồng thông qua lát cắt bằng với tổng luồng đi ra khỏi đỉnh phát s , và cũng bằng với tổng luồng đi vào đỉnh thu t . Ta có biểu thức:

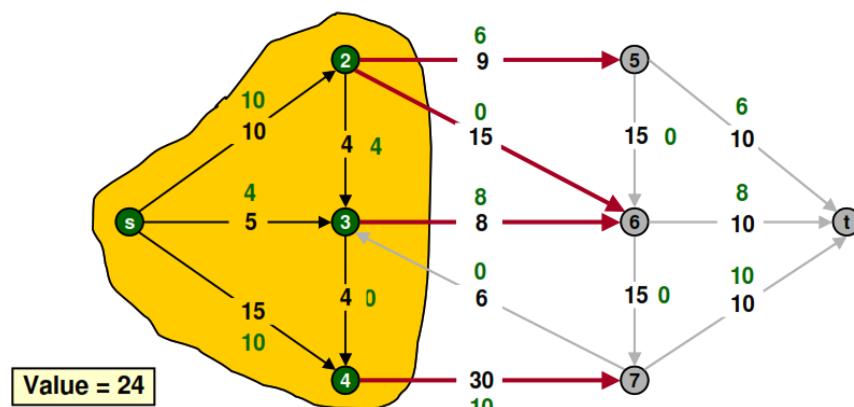
$$\sum_{e \text{ ra khỏi } A} f(e) - \sum_{e \text{ đi vào } A} f(e) = \sum_{e \text{ ra khỏi } s} f(e) = \sum_{e \text{ đi vào } t} f(e) = \text{val}(f)$$

Thật vậy, ta quan sát vài ví dụ sau:

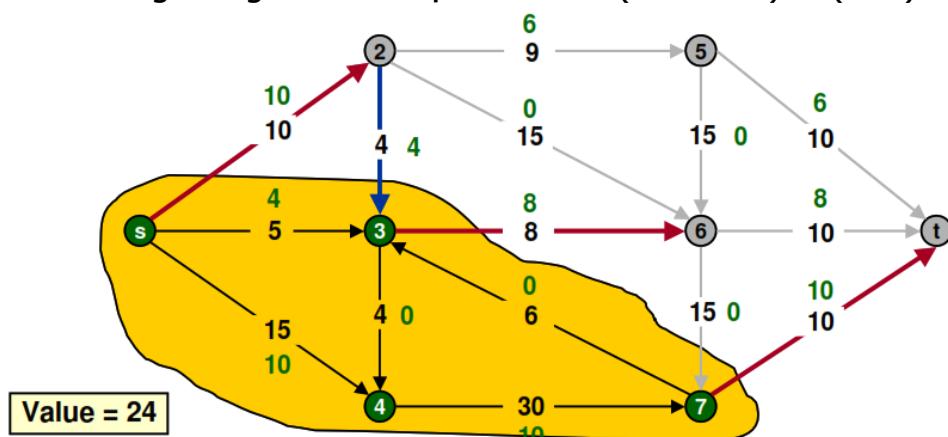
Ví dụ 1: A = {s}, B = {2, 3, 4, 5, 6, 7, t} \rightarrow Theo định lý 1, ta có: $\text{val}(f) = \text{tổng luồng trên các cạnh ra khỏi } A - \text{tổng luồng trên các cạnh vào } A = (10 + 4 + 10) - 0 = 24$



Ví dụ 2: A = {s, 2, 3, 4}, B = {5, 6, 7, t} \rightarrow Theo định lý 1, ta có: val(f) = tổng luồng trên các cạnh ra khỏi A – tổng luồng trên các cạnh vào A = (6+0+8+10) – 0 = 24



Ví dụ 3: A = {s, 3, 4, 7}, B = {2, 5, 6, t} \rightarrow Theo định lý 1, ta có: val(f) = tổng luồng trên các cạnh ra khỏi A – tổng luồng trên các cạnh vào A = (10+8+10) – (4+0) = 24



Chứng minh định lý 1

Ta sẽ chứng minh bằng qui nạp biểu thức:

$$\sum_{e \text{ ra khỏi } A} f(e) - \sum_{e \text{ đi vào } A} f(e) = \text{val}(f)$$

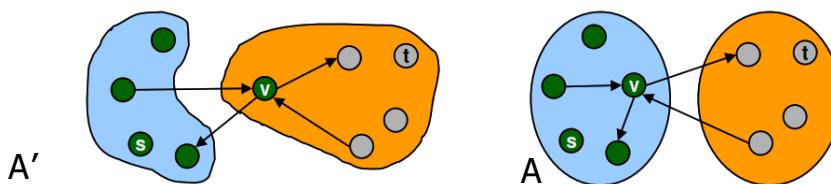
Ta có:

- Biểu thức đúng với trường hợp cơ sở A = {s}
- Giả sử biểu thức đúng với trường hợp $|A| < k$
- Ta sẽ chứng minh biểu thức cũng đúng với trường hợp $|A| = k$.
- Xét lát cắt (A, B) với $|A| = k$. Ta có: $A = A' \cup \{v\}$, với $v \neq s, t$; $|A'| = k-1$ và thỏa định lý 1
- Khi ta thêm đỉnh v vào tập A', thì tổng lưu lượng luồng tăng thêm là:

$$\sum_{e \text{ ra khỏi } v} f(e) - \sum_{e \text{ đi vào } v} f(e) = 0 \quad (*)$$

(*) là theo khái niệm luồng đã trình bày ở đầu chuyên đề.

Hình ảnh minh họa



→ Kết luận: rõ ràng lưu lượng luồng qua lát cắt không tăng thêm khi thêm v. Nên vẫn thỏa định lí 1 khi $|A| = k$.

Định lý 2:

Cho f là một luồng trên mạng G . Gọi (A, B) là một lát cắt trên G . Thì ta có: $\text{val}(f) \leq \text{cap}(A, B)$

Chứng minh:

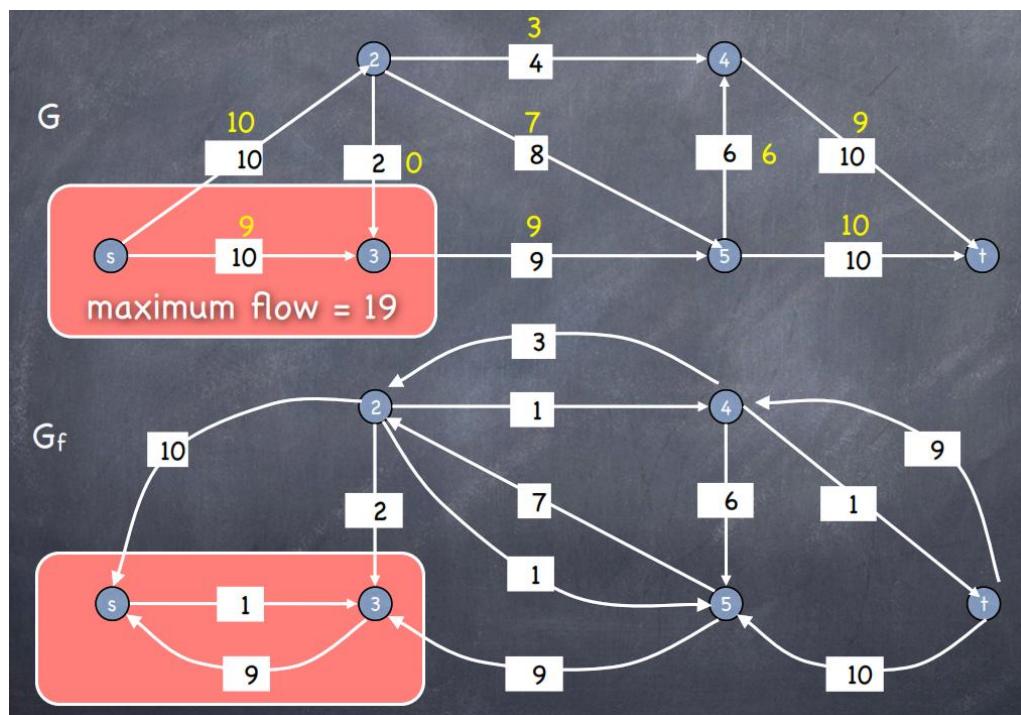
$$\text{Val}(f) = \sum_{e \text{ ra khỏi } A} f(e) - \sum_{e \text{ đi vào } A} f(e) \leq \sum_{e \text{ ra khỏi } A} f(e) \leq \sum_{e \text{ ra khỏi } A} c(e) \leq \text{cap}(A, B)$$

Hệ quả:

Cho f là một luồng trên mạng G . Gọi (A, B) là một lát cắt trên G . Nếu $\text{val}(f) = \text{cap}(A, B)$, thì f đạt cực đại và lát cắt (A, B) là lát cắt nhỏ nhất.

- Điều này có nghĩa là để giải bài toán tìm lát cắt nhỏ nhất, ta sẽ đi giải bài toán tìm luồng cực đại đã trình bày ở trên.
- Sau khi tìm thấy luồng cực đại, những đỉnh u đến được từ s trên đồ thị thặng dư khi luồng f đạt cực đại sẽ thuộc tập đỉnh A trong lát cắt. Các đỉnh còn lại và đỉnh t sẽ thuộc tập đỉnh B trong lát cắt.

Ví dụ:



Trong ví dụ trên, ta thấy

- Tập $A = \{s, 3\}$ vì 3 là đỉnh đến được từ s trên đồ thị thặng dư sau khi f đạt cực đại.
- Tập $B = \{2, 4, 5, t\}$.

- $\text{val}(f^*) = \text{cap}(A, B) = 19$

4. Bài tập ứng dụng

Các thuật toán được trình bày ở trên, trong thực tế có tốc độ xử lý nhanh hơn lý thuyết. Vì thế Các Thầy, Cô khi triển khai giảng dạy có thể lựa chọn thuật toán phù hợp với đội của mình.

Trong các bài toán được trình bày sau đây, tôi chủ yếu sử dụng phiên bản thuật toán “Fattest Path” và “Dinic” để trình bày trong đoạn code tham khảo. Tuy nhiên, các phiên bản như “Shortest Path” và “Capacity Scale” trong thực tế vẫn có thể giải quyết được bài toán.

Sau đây là hệ thống bài tập được tôi lựa chọn trình bày trong khuôn khổ chuyên đề, rất mong sẽ giúp ích cho các Thầy, Cô trong giảng dạy:

4.1. AFINDFLOW

Cho một mạng G có n đỉnh và m cung. Mỗi cung e của G có một khả năng thông qua $c(e)$.
Mạng G được mô tả như sau:

- Chỉ có một đỉnh phát ký hiệu là ký tự 'S', và một đỉnh thu ký hiệu là 'T'.
- Các đỉnh khác S, T được ký hiệu bởi một chữ cái in hoa từ 'A' → 'O'
- G không chứa cạnh vòng
- G không có các đỉnh kết thúc khác ngoài T
- Giữa mọi cặp đỉnh chỉ có một cung duy nhất

Input: AFINDFLOW.INP gồm

- Dòng 1: Ghi số nguyên m cho biết số lượng cung trên G
- m dòng tiếp theo: Mỗi dòng ghi 3 giá trị u, v, c cho biết có cung nối từ u đến v có giá trị thông qua là c. Trong đó u,v là các ký tự hoa từ 'A' → 'O'

Output: AFINDFLOW.OUT gồm

- Ghi một số nguyên duy nhất là luồng cực đại tìm được

Ví dụ:

AFINDFLOW.INP	AFINDFLOW.OUT	Minh họa
10 S A 8 S B 10 A B 4 A C 8 A D 5 B D 2 B C 5 C T 4 C D 3 D T 12	14	<pre> graph TD S((S)) -- 8 --> A((A)) S((S)) -- 10 --> B((B)) A((A)) -- 4 --> B((B)) A((A)) -- 8 --> C((C)) A((A)) -- 5 --> D((D)) B((B)) -- 5 --> C((C)) B((B)) -- 2 --> D((D)) C((C)) -- 3 --> D((D)) C((C)) -- 4 --> T((T)) D((D)) -- 12 --> T((T)) </pre>

Ràng buộc:

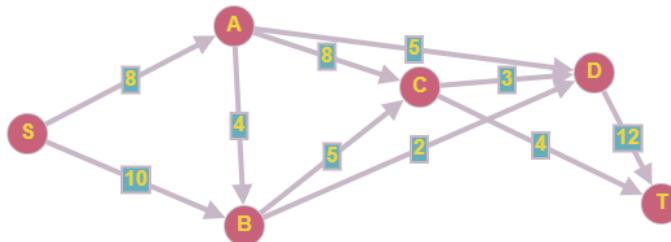
$1 \leq m \leq 50$

$1 \leq c \leq 50$

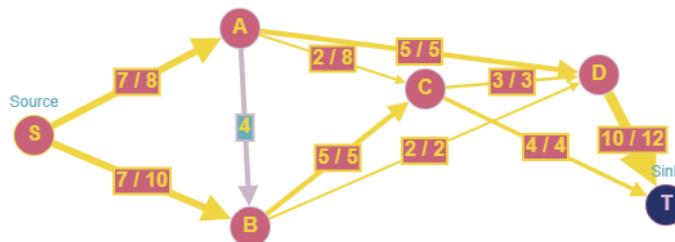
4.1.1. AFINDFLOW_SOLUTION

Đây là bài toán giúp học sinh ôn tập lại code cơ bản của luồng cực đại. Với mỗi đỉnh, học sinh chuyển sang dạng số nguyên sau đó code tương tự các thuật toán đã trình bày ở trên.

Lưu ý: Giáo viên có thể yêu cầu học sinh code theo nhiều thuật toán để kiểm chứng. Với test mẫu, ta sẽ có mô hình mạng G như sau:



Luồng trên mạng G có $\text{val}(f^*) = 14$:



4.1.2. Độ phức tạp

Trong bài toán này tôi dùng thuật toán “Fattest path” $\rightarrow O(E^2 \log(V) \ln(|f^*|))$

4.1.3. Code tham khảo

```

#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> node;
vector<int> adj[1001];
int RC[1001][1001];
int par[1001], n, m, s, t, add_flow;
//-----
void reading()
{
    cin >> m;
    s = 16; t = 17;
    for (int i = 1; i <= m; i++)
    {
        int u, v, c;
        string a, b;
        cin >> a >> b >> c;
        u = (a == "S") ? 16 : (a[0] - 65 + 1);
        v = (b == "T") ? 17 : (b[0] - 65 + 1);
        adj[u].push_back(v);
        adj[v].push_back(u);
        RC[u][v] = c;
    }
    n = 17;
}
//-----
bool pfs(int s, int t, int n)
{
    
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
int Bmax[n+1]={0};
for (int i = 0 ; i <= n ; i++)
    par[i] = -1;
par[s] = 0; Bmax[s] = INT_MAX;
add_flow = INT_MAX;
priority_queue <node> pq;

pq.push({Bmax[s],s});
while (!pq.empty())
{
    pair<int, int> tmp;
    tmp = pq.top();
    pq.pop();
    int B = tmp.first;
    int u = tmp.second;
    add_flow=min(add_flow,B);
    if(u==t) return true;
    for(auto v:adj[u])
        if(par[v]==-1 && min(B,RC[u][v]) > Bmax[v])
    {
        Bmax[v] = min(B,RC[u][v]);
        par[v] = u;
        pq.push({Bmax[v],v});
    }
}
return false;
}
-----
int Maxflow(int s, int t)
{
    int flow = 0;
    while(pfs(s,t,n))
    {
        flow += add_flow;
        int v = t;
        while(v!=s)
        {
            int u = par[v];
            RC[v][u]+=add_flow;
            RC[u][v]-=add_flow;
            v = u;
        }
    }
    return flow;
}
-----
int main()
{   freopen("AFINDFLOW.INP","r",stdin);
    freopen("AFINDFLOW.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    readinp();
    cout<<Maxflow(s,t);
    return 0;
}
```

4.1.4. Link tải bộ test

https://drive.google.com/drive/folders/1rBT5sUC5jFF0n-2jtFyUOcN_r6jN99fe?usp=sharing

4.2. BDISJCON

Cho mạng G có n đỉnh và m cung, đỉnh phát là s = 1, và đỉnh thu là t = n. Mỗi cung đều có khả năng thông qua là 1.

Hai đường đi gọi là phân biệt từ s → t nếu giữ chúng không có cung chung nào.

Gọi F là một tập cung trọng yếu nhỏ nhất khi mọi đường đi từ s → t trên G phải đi qua ít nhất một trong các cung thuộc F.

Yêu cầu: Hãy tìm số lượng đường đi phân biệt từ s đến t, và số lượng cạnh trọng yếu trong tập F.

Input: BDISJCON.INP

- Dòng 1: Ghi 2 số nguyên n, m cho biết số đỉnh và số cung trong mạng G.
- m dòng tiếp theo: Mỗi dòng ghi 2 số nguyên u, v cho biết có cung nối từ u đến v

Output: BDISJCON.OUT

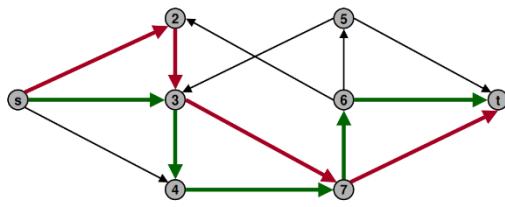
- Ghi 2 số nguyên dương. Số thứ 1 cho biết số lượng đường đi phân biệt tìm được. Số thứ 2 cho biết số lượng cạnh trọng yếu tìm được.

Ví dụ:

BDISJCON.INP	BDISJCON.OUT	Mình họa
8 14 1 2 1 3 1 4 2 3 3 4 3 7 4 7 5 3 5 8 6 2 6 5 6 8 7 6 7 8	2 2	

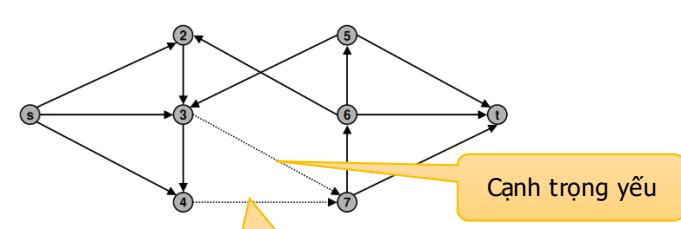
Giải thích:

Trong G tồn tại 2 đường đi phân biệt từ s đến t, và 2 cạnh trọng yếu như hình sau:



Ràng buộc:

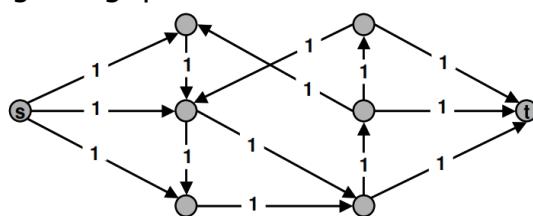
- $0 < n \leq 10^4$



- $0 < m \leq 10^6$

4.2.1. BDISJCON _SOLUTION

Ta có mỗi cạnh đều có khả năng thông qua là 1



Nhận xét: Ta sẽ có k đường đi phân biệt từ $s \rightarrow t \Leftrightarrow \text{val}(f^*) = k$

Chứng minh:

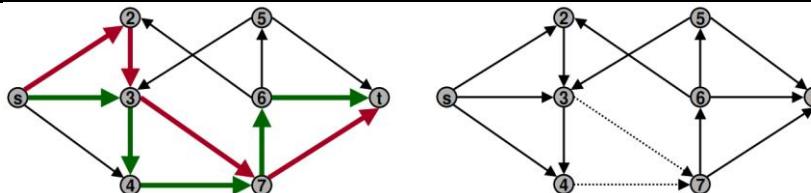
(1) Nếu ta có k đường đi phân biệt từ $s \rightarrow t \Rightarrow \text{val}(f^*) = k$

- Giả sử ta có k đường đi phân biệt P_1, P_2, \dots, P_k
 - Đánh dấu $f(e) = 1$ nếu e thuộc một đường đi P_i nào đó. Ngược lại $f(e) = 0$;
 - Vì các đường đi là phân biệt nên ta có $\text{val}(f) = k$
- (2) Nếu $\text{val}(f^*) = k \Rightarrow$ Ta có k đường đi phân biệt từ $s \rightarrow t$
- Theo khái niệm luồng thì sẽ tồn tại các luồng $f \{0,1\}$ trên các cạnh sao cho $\text{val}(f) = k$
 - Ta sẽ cung (s,v) với $f(s,v) = 1 \Rightarrow$ tồn tại 1 cung (v,w) mà $f(v,w) = 1$ (sự bảo tồn luồng vào, ra tại mỗi đỉnh v trong mạng). Và điều này cũng xảy ra tương tự với đỉnh w tiếp theo cho đến khi gặp đỉnh t .
- Từ 2 điều trên cho ta sự tồn tại k đường đi từ $s \rightarrow t$

Từ (1) và (2) → Điều phải chứng minh

Định lý Menger (1972)

Số lượng đường đi ($s \rightarrow t$) phân biệt trên mạng G bằng với số lượng cung trọng yếu ít nhất trên G



Chứng minh:

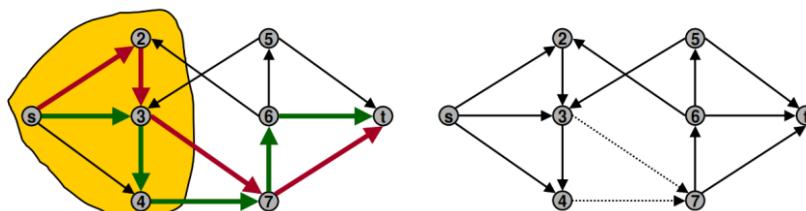
(1) Nếu $|F| = \text{val}(f^*) = k \Rightarrow$ số lượng đường đi ($s \rightarrow t$) phân biệt trên $G = k$

Dễ thấy rằng mỗi đường đi phân biệt từ $s \rightarrow t$ đều phải qua ít nhất 1 cung thuộc F

(2) Nếu số lượng đường đi ($s \rightarrow t$) phân biệt trên $G = k$, và $\text{val}(f^*) = k \Rightarrow |F| = k$

Ta có:

- Lát cắt nhỏ nhất (A,B) (với $s \in A$ và $t \in B$) có $\text{cap}(A,B) = k$
- Gọi F là tập những cạnh đi từ $A \rightarrow B$. Theo định nghĩa mối quan hệ giữa lát cắt và luồng → $|F| = k$



Từ (1) và (2) ta suy ra định lý Menger

4.2.2. Độ phức tạp

Các bài toán trên đồ thị dạng Unit Network và đồ thị 2 phía ta nên dùng thuật toán **Dinic** với độ phức tạp đã trình bày là $O(E\sqrt{V})$ với E, V là số cung, số đỉnh trong G . Tuy nhiên, trong một số trường hợp, thuật toán fattest path (đường béo) tỏ ra hiệu quả hơn.

→ Trong bài này tôi dùng thuật toán Dinic → $O(E\sqrt{V})$.

4.2.3. Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define N 10005

int n,m,s,t,level[N],par[N], RC[N][N];
vector<int> adj[N];
//-----
void nhap()
{
    cin>>n>>m;
    s=1; t=n;
    for(int i=1; i<=m; i++)
    {
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
        adj[v].push_back(u);
        RC[u][v] = 1;
    }
}
//-----
bool BFS(int s, int t, int n)
{
    for (int i = 1 ; i <= n ; i++)
        level[i] = -1, par[i] = -1;

    level[s] = 0;
    par[s] = 0;

    queue< int > q;
    q.push(s);

    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for(auto v:adj[u])
            if (level[v] < 0 && RC[u][v])
            {
                level[v] = level[u] + 1;
                par[v] = u;
                q.push(v);
            }
    }
    return level[t] < 0 ? false : true;
}
//-----
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
int sendFlow(int u, int flow)
{
    if (u == t)
        return flow;
    for(auto v:adj[u])
    {
        if (level[v] == level[u]+1 && RC[u][v])
        {
            int curr_flow = min(flow, RC[u][v]);
            int add_flow = sendFlow(v, curr_flow);
            if (add_flow > 0)
            {
                RC[v][u] += add_flow;
                RC[u][v] -= add_flow;
                return add_flow;
            }
        }
    }
    return 0;
}
-----
int DinicMaxflow(int s, int t, int n)
{
    if (s == t)
        return -1;
    int flow = 0;
    while (BFS(s, t, n))
        while (int add_flow = sendFlow(s, INT_MAX))
            flow += add_flow;
    return flow;
}
-----
int main()
{
    freopen("BDISJCON.INP","r",stdin);
    freopen("BDISJCON.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    nhap();
    int ans = DinicMaxflow(s,t,n);
    cout <<ans<<" "<<ans;
    return 0;
}
```

4.2.4. Link tải bộ test

https://drive.google.com/drive/folders/1Fonw_I-RJ-IwKcINq2zcuxTBdI_1BmK6?usp=sharing

4.3. CSCHEWAR

Nam và Hùng cùng làm chung một công ty. Hiện cả hai thực hiện hai dự án khác nhau có mức độ ưu tiên như nhau. Cả hai cần chạy một số lô xử lý trên các hệ thống của công ty.

Mỗi lô có một số xử lý cần chạy trên một số hệ thống của công ty. Mỗi hệ thống có thể chạy một số lượng xử lý bất kỳ. Hiện tại các hệ thống của công ty chỉ dành để chạy các xử lý của Nam và Hùng mà thôi.

Quản lý của họ đã cho phép Nam chạy các lô xử lý của anh trước Hùng. Hùng cảm thấy buồn và phàn nàn với Quản lý trưởng. Quản lý trưởng thông cảm và đưa cho Hùng một điều kiện nếu muốn sử dụng các hệ thống xử lý. Đó là bằng cách thay thế một số hoặc toàn bộ các xử lý đang chạy của Nam, Hùng phải cho thấy số lượng xử lý của mình trên các hệ thống của công ty là nhiều hơn.

Hùng muốn chạy các lô xử lý của mình theo một thứ tự sao cho tổng số lượng xử lý trên các hệ thống là nhiều nhất có thể. Hiện tại, Hùng hơi rối, em hãy giúp anh ấy nhé.

Một số lưu ý:

- Một hệ thống có thể chạy số lượng xử lý bất kỳ từ nhiều lô khác nhau tại cùng một thời điểm nhưng phải là những lô xử lý của Nam và Hùng.
- Các xử lý của Nam được chạy trên một số hoặc tất cả các hệ thống.
- Một lô xử lý của Hùng hoặc là chạy hết tất cả các qui trình, hoặc không chạy qui trình nào.
- Nếu Hùng thay thế một hệ thống bằng các xử lý của anh ấy, thì hệ thống đó có thể chạy các xử lý thuộc các lô khác của Hùng.
- Mỗi lô cần tổng cộng b hệ thống và chạy p xử lý trong bất kỳ ngữ cảnh nào.

Input: CSCHEWAR.INP gồm:

- Dòng 1: Ghi số nguyên n là số lượng hệ thống đang chạy các xử lý của Nam
- Dòng 2: Ghi n số nguyên cách nhau bởi 1 dấu cách. Số thứ i cho biết số lượng xử lý đang chạy tại hệ thống thứ i.
- Dòng 3: Ghi số m cho biết số lượng lô xử lý của Hùng.
- m dòng tiếp theo: Dòng thứ i gồm số thứ 1 là q, cho biết số lượng các hệ thống cần để thiết để xử lý lô thứ i. Tiếp theo là b số nguyên a₁, a₂, ..., a_q cho biết số hiệu của các hệ thống cần thiết. Cuối cùng là giá trị p_i cho biết số lượng xử lý trong lô thứ i.

Output: CSCHEWAR.OUT gồm:

- Một số nguyên duy nhất cho biết số lượng xử lý tăng thêm khi Hùng thay thế một số hoặc tất cả các xử lý của Nam đang chạy.

Ví dụ:

CSCHEWAR.INP	CSCHEWAR.OUT
4	
1 7 8 9	1
2	
3 1 2 3 15	
2 3 4 11	
4	
3 1 5 2	2
3	
2 1 2 1	
2 2 3 7	
1 4 3	

Giải thích:

Test 1:

Hùng có thể chạy lô 1 bằng cách thay thế các xử lý của Nam tại các hệ thống 1, 2, 3. Số xử lý sẽ giảm đi 1. Tuy nhiên, Hùng vẫn có thể chạy tiếp lô xử lý 2 tại các hệ thống 3, 4. Nghĩa là Hùng sẽ sử dụng hệ thống 3 đang có và thay thế tiếp tục các xử lý của Nam tại hệ thống 4. Theo đó tổng số xử lý trên hệ thống sẽ là 26, tăng 1 so với 25 xử lý của Nam.

Test 2:

Hùng sẽ chạy lô 2 để thay thế các xử lý tại hệ thống 2, 3. Số qui trình tăng lên 1. Tiếp theo Hùng sẽ chạy lô 3 để thay thế các xử lý tại hệ thống 4. Số qui trình tăng lên 1. Tổng qui trình tăng thêm là 2.

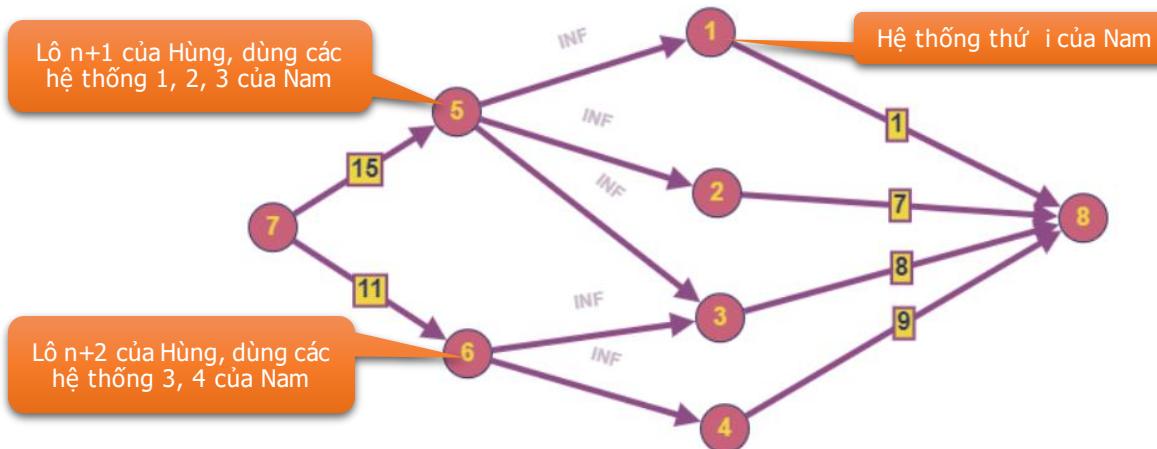
Ràng buộc:

- $1 \leq n, m \leq 100$
- $0 \leq s_i < 70$
- $0 \leq q < 100$
- $1 \leq p \leq 50$

4.3.1. CSCHEWAR_SOLUTION

- Để làm được các bài toán phần luồng, học sinh cần hình dung được các đỉnh, đỉnh phát s, đỉnh thu t, các cung và giá trị thông qua trên mỗi cung.
- Đối với bài này ta sẽ xây dựng mạng G gồm:
 - ✓ Đỉnh phát s có số hiệu là $n + m + 1$
 - ✓ Đỉnh thu t có số hiệu là $s + 1$
 - ✓ Tổng số đỉnh N = $n+m+2$
 - ✓ Với hệ thống thứ i trong danh sách Nam dùng, xây dựng 1 cung $i \rightarrow t$, với $c(i,t) = a_i$
 - ✓ Với lô xử lý thứ j ($j=1,2,\dots,m$) của Hùng, ta đánh số hiệu đỉnh là $n+j$, ta xây dựng:
 - o Một cung $s \rightarrow n+j$ với $c(s,n+j) = p$
 - o Với mỗi hệ thống có số hiệu i (i là số hiệu hệ thống mà nam đang dùng) trong danh sách lô thứ j của Hùng ta xây dựng 1 cung từ $n+j \rightarrow i$, với $c(n+j,i) = \text{INF}$ (∞)

Ví dụ: Trong test 1 ta sẽ xây dựng mạng G với đỉnh phát s = 7, đỉnh thu t = 8



Sau khi xây dựng được mạng G như trên ta thực hiện:

- Gọi sum là tổng số xử lý các lô của Hùng đạt được nếu chạy trên các hệ thống
- Tìm luồng cực đại từ $s \rightarrow t$: $\text{val}(f^*)$. Đây là số xử lý lớn nhất nếu chạy các lô xử lý của Hùng trên các hệ thống nhưng vẫn giữ nguyên số xử lý tối đa trên mỗi hệ thống bằng với Nam
- Kết quả bài toán: $\text{sum} - \text{val}(f^*)$

4.3.2. Độ phức tạp

Bài toán sử dụng thuật toán “Fattest path” $\Rightarrow O(E^2 \log(V) \ln(|f^*|))$

4.3.3. Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> node;
vector<int> adj[1001];
int RC[1001][1001];
int par[1001], a[1001], n, m, N, s, t, add_flow;
//-----
bool pfs(int s, int t, int n)
{
    int Bmax[n+1]={0};
    for (int i = 0 ; i <= n ; i++)
    {
        if (par[i] == -1)
        {
            queue<int> q;
            q.push(i);
            while (!q.empty())
            {
                int u = q.front();
                q.pop();
                for (int v : adj[u])
                {
                    if (RC[u][v] > 0 && Bmax[v] < RC[u][v] && par[v] == -1)
                    {
                        par[v] = u;
                        Bmax[v] = RC[u][v];
                        q.push(v);
                    }
                }
            }
        }
    }
    if (par[t] == -1)
        return false;
    else
        return true;
}
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
    par[i] = -1;
par[s] = 0; Bmax[s] = INT_MAX;
add_flow = INT_MAX;
priority_queue <node> pq;

pq.push({Bmax[s], s});
while (!pq.empty())
{
    pair<int, int> tmp;
    tmp = pq.top();
    pq.pop();
    int B = tmp.first;
    int u = tmp.second;
    add_flow=min(add_flow,B);
    if(u==t) return true;
    for(auto v:adj[u])
        if(par[v]==-1 && min(B,RC[u][v]) > Bmax[v])
        {
            Bmax[v] = min(B,RC[u][v]);
            par[v] = u;
            pq.push({Bmax[v], v});
        }
    }
    return false;
}
//-----
int Maxflow(int s, int t)
{
    int flow = 0;
    while(pfs(s,t,N))
    {
        flow += add_flow;
        int v = t;
        while(v!=s)
        {
            int u = par[v];
            RC[v][u]+=add_flow;
            RC[u][v]-=add_flow;
            v = u;
        }
    }
    return flow;
}
//-----
void themcanh(int u, int v, int c)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
    RC[u][v] = c;
}
//-----
void solve() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
}
```

```
cin >> m;
s = n + m + 1;
t = s + 1;
N = n+m+2;
int sum = 0; int i,q;
for (int j = 1; j <= m; j++) {
    cin >> q;
    while (q--) {
        cin >> i;
        themcanh(n + j, i, INT_MAX);
    }
    int w;
    cin >> w;
    themcanh(s, n + j, w);
    sum += w;
}
for (int i = 1; i <= n; i++) {
    themcanh(i, t, a[i]);
}
cout << sum - Maxflow(s,t);
}

-----
int main() {
    freopen("CSCHEWAR.INP","r",stdin);
    freopen("CSCHEWAR.OUT","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(NULL);

    solve();
    return 0;
}
```

4.3.4. Link tải bộ test

<https://drive.google.com/drive/folders/14XeJgWaIaFxBSHRslAuPHnkJ95iNiuyz?usp=sharing>

4.4. CBADGES

Hoa được Thầy hiệu trưởng giao nhiệm vụ thiết kế huy hiệu cho cả trường. Một huy hiệu được tạo bằng cách xếp chồng 2 hình vuông lên nhau. Hình vuông bên dưới sẽ có màu xanh. Hình vuông bên trên thì có màu vàng. Hai hình vuông được xoay sao cho có thể nhìn thấy 8 đỉnh phân biệt.

Thầy hiệu trưởng giao cho Hoa n hình vuông với kích thước cạnh khác nhau thuộc một trong 2 màu xanh và vàng. Hiện tại bạn ấy đang rất cần biết với những hình vuông này ta sẽ tạo được tối đa bao nhiêu huy hiệu. Em hãy giúp Hoa nhé!

Input: CBADGES.INP

- Dòng 1: Ghi số nguyên n là số lượng hình vuông.
- n dòng tiếp theo: Mỗi dòng ghi 2 số nguyên c_i và d_i mô tả một hình vuông. Trong đó $c_i = 1$ nếu là màu vàng, $c_i = 2$ thì là màu xanh; d_i là chiều dài cạnh của hình vuông.

Output: CBADGES.OUT

- Ghi một số nguyên là số huy hiệu tối đa tạo được.

Ví dụ:

CBADGES.INP	CBADGES.OUT
10	4
1 3	
1 17	
2 13	
2 13	
2 6	
1 10	
2 8	
1 7	
1 8	
2 8	

Ràng buộc:

- $1 \leq n \leq 100000$
- $1 \leq d_i \leq 1000$
- $1 \leq c_i \leq 2$

Trước khi đi vào giải bài này, ta sẽ tìm hiểu 1 chút về bài toán

4.4.1. “Cặp ghép cực đại trên đồ thị hai phía (Bipartite Matching)”

Khái niệm đồ thị hai phía (Bipartite Graph):

Một đồ thị đơn vô hướng $G=(V,E)$ là một đồ thị hai phía nếu như tồn tại một cách phân hoạch tập đỉnh V thành hai tập V_1, V_2 sao cho mỗi cạnh thuộc E đều có dạng (v_1, v_2) , với $v_1 \in V_1, v_2 \in V_2$.

Cặp ghép trên đồ thị (Matching):

Tập cạnh $M \subseteq E$ là một cặp ghép hay tập cạnh độc lập của một đồ thị là một tập các cạnh không có đỉnh chung. Bài toán ghép cặp thường được quan tâm trong trường hợp đồ thị hai phía.

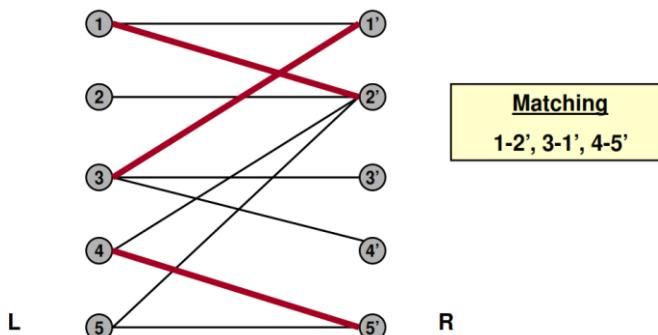
Một ví dụ đó là bài toán sắp xếp công việc. Giả sử có P người và J công việc, mỗi người có thể làm một số công việc nào đó. Ta mô hình bài toán bằng một đồ thị hai phía với $P+J$ đỉnh. Nếu người p_i có thể làm công việc j_i thì có một cạnh giữa hai đỉnh p_i và j_i trên đồ thị.

Một cặp ghép cực đại (còn được gọi là cặp ghép có số lượng cạnh lớn nhất) trên một đồ thị hai phía G .

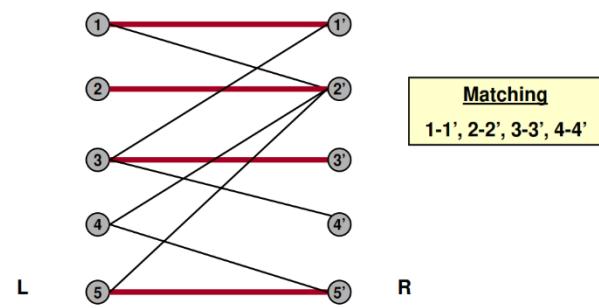
Ví dụ:

Input: Đồ thị hai phía, vô hướng $G= (L \cup R, E)$, với L là tập đỉnh bên trái, R là tập đỉnh bên phải

Output: Tìm cặp ghép cực đại trên G



Matching theo cách thứ 1

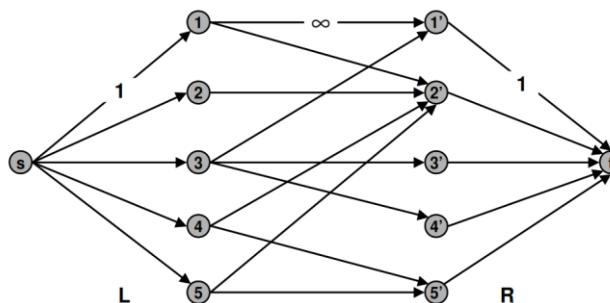


Matching theo cách thứ 2

→ Cách thứ 2 có số lượng cạnh thuộc cặp ghép lớn hơn

Ta sẽ giải bài toán này với ứng dụng luồng cực đại như sau:

- 1/ Ta sẽ xây dựng đồ thị có hướng $G' = (L \cup R \cup \{s, t\}, E')$
- 2/ Tất cả các cung e có chiều từ $L \rightarrow R$ được cho giá trị $c(e) =$ vô cùng
- 3/ Thêm đỉnh phát ảo s , và các cung từ $s \rightarrow$ đỉnh trong tập L , mỗi cung e có $c(e) = 1$
- 4/ Thêm đỉnh thu ảo t , và các cung từ đỉnh trong $R \rightarrow t$, mỗi cung e có $c(e) = 1$

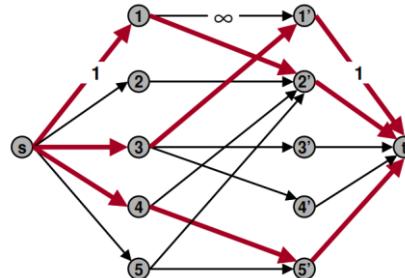
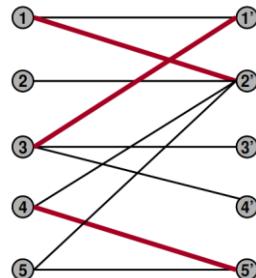


Kết quả: cặp ghép cực đại = giá trị luồng cực đại trên G'

4.4.2. Chứng minh:

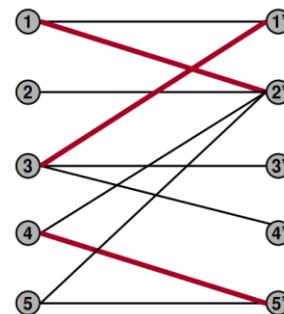
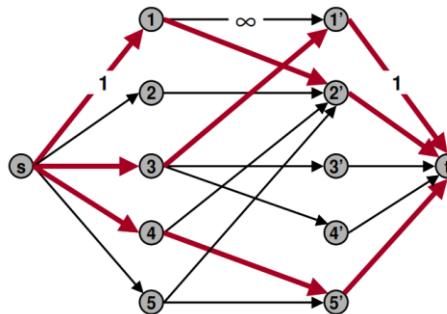
→ (1) Nếu $|M| = k$ thì $\text{val}(f) = k$

- Giả sử ta có tập $M = \{1 - 2', 3 - 1', 4 - 5'\}$ hay $|M| = 3$
- Khi đó ta có thể gửi các luồng giá trị 1 dọc theo 3 con đường độc lập là:
 $s - 1 - 2' - t$, $s - 3 - 1' - t$, $s - 4 - 5' - t$
- Ta có f là luồng có $\text{val}(f) = 3$



← (2) Nếu $\text{val}(f) = k$ thì $|M| = k$

- Theo khái niệm luồng đã trình bày ta sẽ có các luồng $\{0,1\}$ đi ra từ f sao cho $\text{val}(f) = k$
- Xét M là tập cung từ L đến R có $f(e) = 1$ và thỏa điều kiện: Mỗi đỉnh trong L và R chỉ tham gia vào tối đa 1 cung của M
- Xét độ lớn lát cắt $(L \cup s, R \cup t) \rightarrow |M| = k$



4.4.3. CBADGES SOLUTION

Tóm tắt: Ta có 2 loại hình vuông, hình vuông màu xanh và màu vàng. Mỗi hình vuông có một chiều dài cạnh. Nhiệm vụ của Hoa là thiết kế huy hiệu cho trường. Mỗi huy hiệu gồm 2 hình vuông xếp chồng lên nhau. Hình vuông dưới màu xanh, hình vuông trên màu vàng. Xếp làm sao thấy được 8 đỉnh phân biệt. Hoa cần bạn cho biết số huy hiệu lớn nhất có thể tạo từ n hình vuông đã cho.

Nhận xét:

Ta có thể hình dung dưới dạng một đồ thị hai phía với hai tập đỉnh.

- Tập đỉnh bên trái là các hình vuông màu vàng. Số hiệu đỉnh được đánh theo giá trị độ dài cạnh. Vì thế các đỉnh ứng với hình vuông màu vàng được đánh số từ $1 \rightarrow N$ (1005)
- Tập đỉnh bên phải là các hình vuông màu xanh. Được đánh số từ $N + 1 \rightarrow 2 * N$
- Gọi $\text{cnt}[1][i]$ là số hình vuông màu vàng có cạnh dài i
- Gọi $\text{cnt}[2][i]$ là số hình vuông màu xanh có cạnh dài i

Ta thực hiện mô hình hóa thành mạng G' như sau:

- Thêm đỉnh phát $s = 2 * N + 1$.
- Thêm đỉnh thu $t = s + 1$
- Với mỗi hình vuông u màu vàng, ta thêm cung $e'(s, u)$ với $c(e') = \text{cnt}[1][u]$

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

- Với mỗi hình vuông v màu xanh, ta thêm cung e'(v, t) với $c(e') = \text{cnt}[2][v]$
- Với mỗi hình vuông màu vàng u, và hình vuông màu xanh v thỏa điều kiện tạo huy hiệu (*) thì ta thêm cung e' (u,v) với $c(e') = \text{vô cùng (INF)}$

Điều kiện (*):

- ✓ Giả sử ta có hình vuông màu vàng nằm trên, có cạnh b, hình vuông màu xanh có cạnh a, nằm dưới. Thì điều kiện để 2 hình vuông tạo thành huy hiệu là:

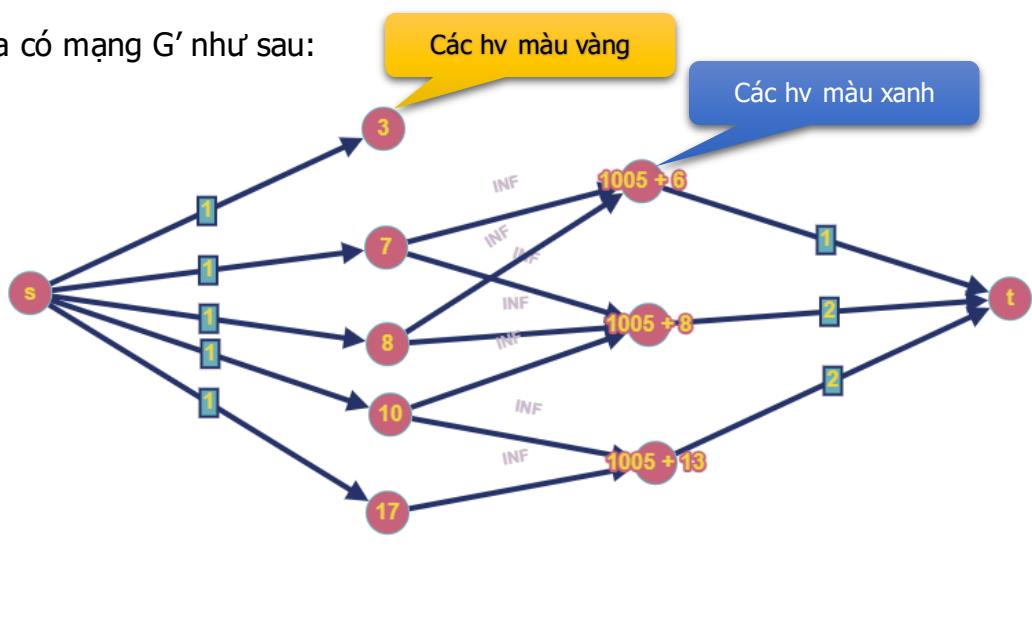
Đường chéo của hình vuông có cạnh nhỏ > cạnh của hình vuông còn lại

Giả sử $b \leq a$, ta sẽ có: $2*b*b > a*a$. Trường hợp $a > b$ thì ta hoán đổi giá trị a, b

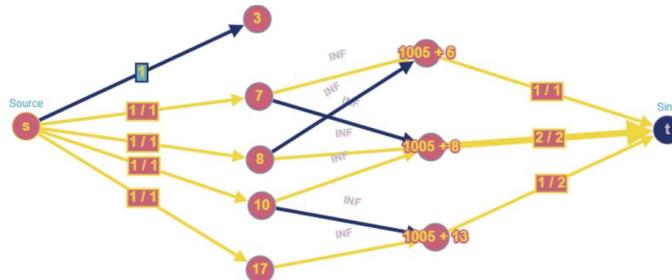
- Theo bài toán cặp ghép cực đại đã tìm hiểu ở trên, thì kết quả bài toán là giá trị luồng cực đại trên mạng G'

Theo đó với test mẫu ta có mạng G' như sau:

CBADGES.INP	
10	
1 3	
1 17	
2 13	
2 13	
2 6	
1 10	
2 8	
1 7	
1 8	
2 8	



Luồng cực đại trên mạng G' có $\text{val}(f^*) = 4$ như hình sau:



4.4.4. Độ phức tạp

Bài toán sử dụng thuật toán Dinic tìm cặp ghép cực đại trên đồ thị 2 phía, tương đương trên Unit Network $\Rightarrow O(E\sqrt{V})$

4.4.5. Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> node;
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
vector<int> adj[2050];
int RC[2050][2050];
int par[2050], cnt[3][1050], level[2050], n, N, m, s, t, add_flow;
//-----
bool BFS(int s, int t, int n)
{
    for (int i = 0 ; i <= n ; i++)
        level[i] = -1, par[i] = -1;
    level[s] = 0;
    par[s] = 0;
    queue< int > q;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for(auto v:adj[u])
            if (level[v] < 0 && RC[u][v])
            {
                level[v] = level[u] + 1;
                par[v] = u;
                q.push(v);
            }
    }
    return level[t] < 0 ? false : true;
}
//-----
int sendFlow(int u, int flow)
{
    if (u == t)
        return flow;
    for(auto v:adj[u])
    {
        if (level[v] == level[u]+1 && RC[u][v])
        {
            int curr_flow = min(flow, RC[u][v]);
            int add_flow = sendFlow(v, curr_flow);
            if (add_flow > 0)
            {
                RC[v][u] += add_flow;
                RC[u][v] -= add_flow;
                return add_flow;
            }
        }
    }
    return 0;
}
//-----
int DinicMaxflow(int s, int t, int n)
{
    if (s == t)
        return -1;
    int flow = 0;
    while (BFS(s, t, n))
        while (int add_flow = sendFlow(s, INT_MAX))
            flow += add_flow;
    return flow;
}
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
bool check(int a, int b) {
    if (a < b) { swap(a, b); }
    return 2 * b * b > a * a;
};

//-----
void themcung(int u, int v, int c)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
    RC[u][v] = c;
}

//-----
void solve() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int c,d;
        cin >> c >> d;
        cnt[c][d]++;
    }
    N = 1005;
    s = 2*N + 1;
    t = s+1;
    for (int i = 1; i <= N; i++) {
        if (cnt[1][i] == 0) {//không có hình vuông màu vàng độ dài i
            continue;
        }
        themcung(s, i, cnt[1][i]);
        for (int j = 1; j <= N; j++) {
            if (cnt[2][j] == 0 || !check(i, j)) {
                continue;
            }
            themcung(i, N + j, INT_MAX);
            //i là hình vuông vàng, N+j là hình vuông xanh, cả 2 thỏa điều kiện
        }
    }
    for (int i = 1; i <= N; i++) {
        if (cnt[2][i] == 0) {//không có hình vuông màu xanh độ dài i
            continue;
        }
        themcung(N + i, t, cnt[2][i]);
    }
    cout << DinicMaxflow(s,t, 2*N+2);
}

//-----
int main() {
    freopen("CBADGES.INP", "r", stdin);
    freopen("CBADGES.OUT", "w", stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(NULL);
    solve();
    return 0;
}
```

4.4.6. Link tải bộ test

https://drive.google.com/drive/folders/1UtjfDkn9C682Wj2d_YHQEFJINeYae18N?usp=sharing

4.5. CODSPORTS

Ghi chú: Bài của Thầy Đỗ Đức Đông (Trại hè Phương Nam 2019)

Sau buổi giao lưu giữa các đoàn, Ban tổ chức nhận thấy các thí sinh rất hào hứng với các trò chơi tập thể. Đây là một tín hiệu tốt trong bối cảnh điện thoại thông minh hay mạng xã hội đang ngày càng ảnh hưởng và làm thay đổi đến giới trẻ. Vì vậy, Ban tổ chức quyết định tổ chức một môn thể thao ngoài trời như sau:

Trên một mặt sân phẳng rộng, vẽ một lưới ô vuông kích thước $n \times n$. Các hàng của lưới đánh số từ 1 đến n từ trên xuống, các cột của lưới được đánh số từ 1 đến n từ trái sang. Ô nằm giao giữa hàng i và cột j được gọi là ô (i,j) . Chọn một nhóm gồm n thí sinh, xếp các thí sinh đứng vào các ô trên lưới sao cho mỗi hàng có đúng một thí sinh. Sau đó, mỗi thí sinh được phát một chiếc hộp có khối lượng là m kg. Nhiệm vụ của nhóm tham gia như sau:

Mỗi thí sinh có thể di chuyển chiếc hộp của mình sang các ô cùng hàng để sau khoảng thời gian 5 phút, khi quan sát lưới theo cột, mỗi cột chỉ có đúng một hộp. Đây là một môn thể thao đòi hỏi thể lực cũng như sự tính toán hợp lý của nhóm. Nếu thể lực của thí sinh ở hàng i là s_i thì trong vòng 5 phút thí sinh đó có thể di chuyển chiếc hộp sang một ô khác cùng hàng cách ô cũ không quá $\lfloor \frac{s_i}{m} \rfloor$, khoảng cách giữa hai ô (i,j_1) và (i,j_2) được tính bằng $|j_1 - j_2|$.

Yêu cầu: Cho biết vị trí đứng trên lưới và thể lực của từng thí sinh, với chiếc hộp có khối lượng m , hãy giúp Ban tổ chức trả lời câu hỏi có hay không tồn tại cách di chuyển các hộp để nhóm có thể di chuyển các chiếc hộp thỏa mãn yêu cầu.

Input: Vào từ file văn bản **CODSPORTS.INP** theo khuôn dạng:

- Dòng đầu tiên chứa số nguyên dương n ;
- Dòng thứ hai gồm n số v_1, v_2, \dots, v_n là vị trí cột mà thí sinh ở hàng i đứng;
- Dòng thứ ba gồm n số s_1, s_2, \dots, s_n là thể lực của thí sinh đứng ở hàng i ;
- Dòng thứ tư là một số nguyên dương Q ;
- Dòng cuối cùng gồm Q số nguyên dương m_1, m_2, \dots, m_Q tương ứng Q câu hỏi mà
- Ban tổ chức muốn biết có tồn tại hay không phương án di chuyển hộp thỏa mãn yêu cầu.

Output: Ghi ra file văn bản **CODSPORTS.OUT** gồm Q số nguyên trên một dòng, số thứ k là câu trả lời cho câu hỏi mk trong file dữ liệu vào, ghi số 1 nếu tồn tại và 0 nếu không tồn tại.

Ví dụ:

CODSPORTS.INP	CODSPORTS.OUT
3	1 1
1 2 1	
5 5 2	
2	
5	
3	

Ràng buộc:

Sub1: Có 20% số test ứng với 20% số điểm của bài có $Q = 1$; $n \leq 10$;

Sub2: Có 20% số test khác ứng với 20% số điểm của bài có $Q \leq 10$; $n \leq 10$;

Sub3: Có 10% số test khác ứng với 10% số điểm của bài có $Q \leq 10$; $n \leq 20$;

Sub4: Có 20% số test khác ứng với 20% số điểm của bài có $Q \leq 10$; $n \leq 200$;

Sub5: Có 10% số test khác ứng với 10% số điểm của bài có $Q \leq 10$; $n \leq 2000$;

Sub6: Có 20% số test còn lại ứng với 20% số điểm của bài có Q , $n \leq 10^5$

4.5.1. CODSPORTS_SOLUTION

Tóm tắt:

- Chọn một nhóm gồm n thí sinh, xếp các thí sinh đứng vào các ô trên lưới sao cho mỗi hàng có đúng một thí sinh. Hàng i thí sinh sẽ đứng tại cột $v[i]$ và có một thể lực $s[i]$.
- Cho giá trị m . Mỗi thí sinh có thể di chuyển từ cột $i \rightarrow$ cột j sao cho: $|i-j| \leq \lfloor \frac{s_i}{m} \rfloor$
- Hãy cho biết có hay không cách di chuyển các thí sinh thỏa điều kiện trên sau khi di chuyển thì mỗi cột trong lưới chỉ có 1 thí sinh.
- Có Q truy vấn với các giá trị m_1, m_2, \dots, m_Q . Với mỗi giá trị m_i ta trả lời cho câu hỏi ở trên. Nếu tồn tại cách di chuyển ta ghi 1, ngược lại ta ghi 0.

Nhận xét:

- Cụ thể ta có n dòng mỗi dòng 1 thí sinh hãy tìm cách ghép n thí sinh vào các cột j thỏa yêu cầu đề bài.
- Với thuật toán luồng cực đại và bài toán cặp ghép cực đại ta sẽ giải quyết được 5 Sub đầu của bài. Sub 6 ta có thể sử dụng Priority_queue cùng với chặt nhị phân để giải quyết (Trong chuyên đề mình không trình bày sub 6)

Thuật toán:

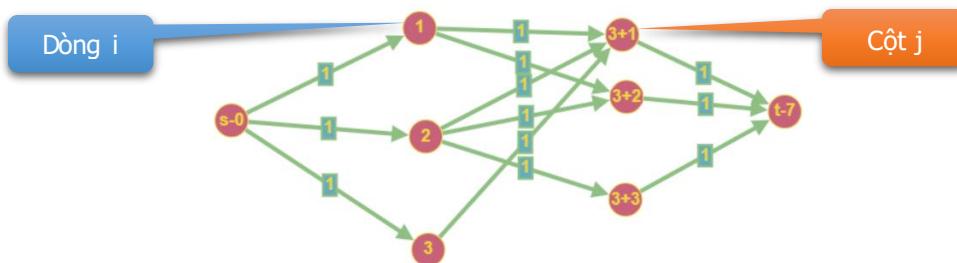
Ta sẽ mô hình hóa bài toán dưới dạng một đồ thị hai phía với 2 tập đỉnh:

- Tập đỉnh thứ 1: là các đỉnh đánh số từ 1 → n tương ứng với n dòng
- Tập đỉnh thứ 2: là các cột j ($j=1..n$) đánh số từ $n+1$ → $n+n$

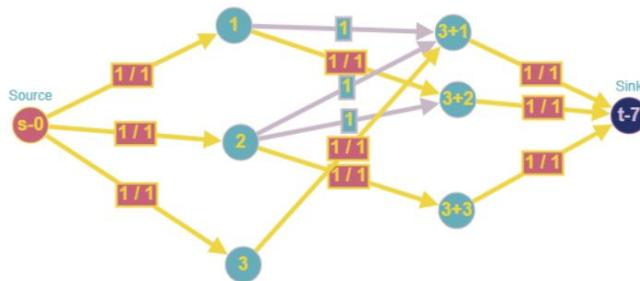
Ta sẽ xây dựng mạng G với khả năng thông qua các cạnh đều là 1 (Unit Network) như sau:

- Số đỉnh: $2*n + 1$
- Đỉnh phát $s = 0$
- Đỉnh thu $t = 2*n + 1$
- Với mỗi dòng i , ta thêm cung $e(s,i)$ với $c(e) = 1$
- Với mỗi cột j , ta thêm cung $e(n+j,t)$ với $c(e) = 1$
- Với mỗi cặp (i,j) thỏa yêu cầu đề bài, ta sẽ thêm cung $e(i, n+j)$ với $c(e) = 1$

Theo mô tả trên và test mẫu đề bài với $n = 3$, ta có mạng G như sau:



Sau khi tìm luồng cực đại ta được hình sau:



Nhìn vào hình minh họa luồng cực đại trên, quan sát các cung $e(i, n+j)$ ứng với dòng i , cột j , nghĩa là thí sinh trên dòng i sẽ di chuyển đến cột j theo điều kiện đề bài. Ta thấy:

- Cung $(1, 3+2)$ có luồng đi qua \rightarrow Thí sinh tại dòng 1 sẽ di chuyển đến cột 2
 - Cung $(2, 3+3)$ có luồng đi qua \rightarrow Thí sinh tại dòng 2 sẽ di chuyển đến cột 3
 - Cung $(3, 3+1)$ có luồng đi qua \rightarrow Thí sinh tại dòng 3 sẽ di chuyển đến cột 1
- \Rightarrow Đây là một phương án để lấp đầy các cột, mỗi cột 1 thí sinh. Không thể có 2 luồng từ các dòng i vào cùng 1 cột j được (vì từ cột $j \rightarrow t$ chỉ có khả năng thông qua là 1). Hay nói cách khác nếu $\text{val}(f^*) = n$ thì ta xuất 1 (tồn tại phương án), ngược lại ta xuất 0.

Chi tiết thuật toán:

1/ Nhập dữ liệu và xây dựng mạng G với các cung (s,i) và $(n+j,t)$

2/ Với mỗi $m[k]$ trong q truy vấn, ta thực hiện:

- Xây dựng các cung $(i, n+j)$ thỏa điều kiện đề bài
- Tìm luồng cực đại tên mạng G và xuất kết quả
- khôi phục các cung (s,i) và $(n+j, t)$ như lúc đầu, và xóa các cung $(i,n+j)$ đã tạo với $m[k]$

4.5.2. Độ phức tạp

Nhận xét: Với các bài toán nhiều truy vấn, đòi hỏi các thao tác hủy bỏ, khôi phục luồng trên một số cung. Mặc dù là bài toán về một đồ thị dạng “Unit Network”, nhưng thuật toán Fatest Path tỏ ra hiệu quả hơn Dinic trong thực nghiệm.

$\Rightarrow O(q * \max(n^2, O(E^2 \log(V) \ln(|f^*|)))$

4.5.3. Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> node;

int n,q,s,t,level[5005],par[5005],
RC[5005][5005],v[5005],si[5005],m[5005],d[5005],add_flow;
vector<int> adj[5005];
clock_t start, endt;
-----
bool pfs(int s, int t,int n)
{
    int Bmax[n+1]={0};
    for (int i = 0 ; i <= n ; i++)
        par[i] = -1;
    par[s] = 0; Bmax[s] = INT_MAX;
    add_flow = INT_MAX;
    priority_queue <node> pq;
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
    pq.push({Bmax[s], s});
    while (!pq.empty())
    {
        pair<int, int> tmp;
        tmp = pq.top();
        pq.pop();
        int B = tmp.first;
        int u = tmp.second;
        add_flow=min(add_flow,B);

        for(auto v:adj[u])
            if(par[v]==-1 && min(B,RC[u][v]) > Bmax[v] )
            {
                Bmax[v] = min(B,RC[u][v]);
                par[v] = u;
                pq.push({Bmax[v],v});
                if(v==t)
                    return true;
            }
        }
        return false;
    }
    //-----
    int Maxflow(int s, int t)
    {
        int flow = 0;
        while(pfs(s,t,2*n+1))
        {
            flow += add_flow;
            int v = t;
            while(v!=s)
            {
                int u = par[v];
                RC[v][u]+=add_flow;
                RC[u][v]-=add_flow;
                v = u;
            }
        }
        return flow;
    }
    //-----
    void khoiphuc(int k)
    {   //Khôi phục cung (s,i)
        for(int i=1; i<=n; i++)
        {
            RC[s][i] = 1;
            RC[i][s] = 0;
        }
        //Khôi phục cung (n+j,t)
        for(int j=1; j<=n; j++)
        {
            RC[n+j][t] = 1;
            RC[t][n+j] = 0;
        }
        //Bỏ các cung từ cột v[i] --> j
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
        if(abs(j-v[i])<= si[i]/m[k])
        {   RC[i][n+j] = 0;
            RC[n+j][i] = 0;
        }
    }
void nhap()
{
    cin>>n;
    for(int i=1; i<=n; i++)
        cin>>v[i];
    for(int i=1; i<=n; i++)
        cin>>si[i];
    cin>>q;
    for(int i=1; i<=q; i++)
        cin>>m[i];
//Xây dựng mạng G
s=0; t=2*n+1;
//Thêm cung (s,i) - dòng i
for(int i=1; i<=n; i++)
{
    adj[s].push_back(i);
    adj[i].push_back(s);
    RC[s][i] = 1;
}
//Thêm cung (j,t) - cột j
for(int j=1; j<=n; j++)
{
    adj[n+j].push_back(t);
    adj[t].push_back(n+j);
    RC[n+j][t] = 1;
}
//Thêm cung từ cột v[i] --> cột j thỏa điều kiện
for(int k=1; k<=q; k++)
{
    for(int i=1; i<=n; i++)
        for(int j=1; j<=n; j++)
        {
            if(abs(j-v[i])<= si[i]/m[k])
            {
                adj[i].push_back(n+j);
                adj[n+j].push_back(i);
                RC[i][n+j] = 1;
            }
        }
    int tmp = Maxflow(s,t);

    if(tmp == n) cout<<1<<" ";
    else cout<<0<<" ";
    khoiphuc(k);
}
}
//-----
int main()
{   freopen("CODSPORTS.INP","r",stdin);
    freopen("CODSPORTS.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
```

```

    cin.tie(0);
    nhap();
    return 0;
}

```

4.5.4. Link tải bộ test

https://drive.google.com/drive/folders/1q_OFMlOc0iQsg3IbqtapumOURAVnqpKV?usp=sharing

4.6. CEDGEDES

Cho một đồ thị G có N đỉnh và M cạnh nối 2 chiều. Mỗi cạnh có một độ dài và một chi phí phá hủy. Bạn hãy tăng độ dài đường đi ngắn nhất từ đỉnh 1 đến đỉnh N bằng cách phá hủy một số cạnh nối.

Yêu cầu: Hãy tìm cách phá hủy sao cho chi phí phá hủy ít nhất

Input: CEDGEDES.INP

- Dòng 1: Ghi 2 số nguyên N, M
- M dòng tiếp theo: Mỗi dòng ghi 4 số nguyên X, Y, D, C cho biết có cạnh nối giữa đỉnh X và đỉnh Y. Cạnh nối có độ dài D và chi phí phá hủy là C.

Output: CEDGEDES.OUT

- In ra chi phí phá hủy ít nhất tìm được.

Ví dụ:

CEDGEDES.INP	CEDGEDES.OUT	Mình họa
4 6 1 2 4 1 1 3 8 6 1 4 2 8 2 3 8 8 2 4 5 7 3 4 7 5	8	

Giải thích:

Nhìn vào hình ta thấy đường đi ngắn nhất từ 1 → 4 là 2. Khi ta xóa cạnh (1,4) thì đường đi ngắn nhất sẽ tăng lên là 9 (1 – 2 – 4).

➔ Chi phí phá hủy là 8

Ràng buộc:

- $2 \leq N \leq 1000$
- $1 \leq M \leq 4*10^5$
- $1 \leq X, Y \leq N$
- $1 \leq D, C \leq 10^6$

4.6.1. CEDGEDES_SOLUTION

Nhận xét:

- 1) Nếu sau khi xóa các cạnh cần thiết, và khi này không còn tồn tại đường đi ngắn nhất (cùng giá trị) từ $1 \rightarrow n$. Ta sẽ có 2 trường hợp:
 - TH1: Tồn tại 1 đường đi từ $1 \rightarrow n$ dài hơn \rightarrow thỏa yêu cầu đề bài
 - TH2: Không còn tồn tại đường đi từ $1 \rightarrow n$, thì xem như đường đi ngắn nhất (DDNN) từ $1 \rightarrow n$ là vô cùng (∞). Nghĩa là vẫn thỏa yêu cầu của bài toán.
- 2) Không có lý do gì ta lại xóa một cạnh bên ngoài DDNN từ $1 \rightarrow n$. Nghĩa là ta chỉ xét các cạnh thuộc ít nhất một DDNN từ $1 \rightarrow n$ mà thôi.

Từ nhận xét 2 ở trên:

Ta tiến hành xây dựng một đồ thị có dạng mạng G chỉ gồm các cạnh thuộc ít nhất 1 DDNN từ $1 \rightarrow n$. Để làm điều này ta sẽ tính các giá trị sau:

- $s = 1; t = n; \text{số đỉnh} = n;$
 - Gọi $d_1[v]$ là độ dài DDNN từ $1 \rightarrow v$
 - Gọi $d_n[v]$ là độ dài DDNN từ $n \rightarrow v$
 - Một cạnh (u,v) thuộc 1 DDNN khi và chỉ khi thỏa điều kiện:

$$d_1[u] + c(u,v) + d_n[v] = d_1[n].$$
 Trong đó $c(u,v)$ là độ dài cạnh (u,v)
- Ta sẽ dùng BFS từ $1 \rightarrow n$ để duyệt qua các cạnh và lần lượt thêm các cung $e(u,v)$ thỏa điều kiện vào mạng G, với $c(e) = \text{chi phí phá hủy cạnh } (u,v)$

Cuối cùng, ta sẽ tìm độ lớn của lát cắt nhỏ nhất (A,B) (với $s \in A, t \in B$) là $\text{cap}(A,B)$. Đây cũng chính là giá trị luồng cực đại trên G, và là kết quả của bài toán (*).

Giải thích (*):

Giả sử sau khi tìm được lát cắt hẹp nhất ta có 2 cung đi từ $A \rightarrow B$. Điều này nghĩa là ta sẽ có ít nhất 2 cung trọng yếu mà các DDNN từ $1 \rightarrow N$ chắc chắn sẽ đi qua 1 trong 2 cung.

Nếu ta xóa 1 cung, thì vẫn sẽ tồn tại DDNN từ $1 \rightarrow N$ đi qua cung còn lại. Nên không thể tăng độ dài DDNN được.

→ Ta cần xóa hết các cung đi từ $A \rightarrow B$. Nghĩa là tổng độ phá hủy cần tìm = $\text{cap}(A,B)$

4.6.2. Độ phức tạp

- Thuật toán Dijkstra với priority_queue là: $O((m+n)\log n)$
 - Thuật toán BFS duyệt để xây dựng mạng G: $O(m+n)$
 - thuật toán fattest path tìm luồng cực đại là: $O(E^2 \log(V) \ln(|f^*|))$ với E, V là số cạnh, đỉnh trong mạng G (Tùy vào số lượng cạnh trên đường đi ngắn nhất)
- Độ phức tạp bài toán là $O(\max((m+n)\log n, E^2 \log(V) \ln(|f^*|)))$

4.6.3. Code tham khảo

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> node;
const int N = 1e3 + 5;
const int M = 4e5 + 5;
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
const int INF = 1e9 + 100;

node edges[M];
bool vis[N];
int dist[N][2];
int n,m,s,t,par[N], RC[N][N],add_flow;
vector<node> adj[N];
vector<int> a[N];
//-----
bool pfs(int s, int t,int n)
{
    int Bmax[n+1]={0};
    for (int i = 0 ; i <= n ; i++)
        par[i] = -1;
    par[s] = 0; Bmax[s] = INT_MAX;
    add_flow = INT_MAX;
    priority_queue <node> pq;
    pq.push({Bmax[s],s});
    while (!pq.empty())
    {
        pair<int, int> tmp;
        tmp = pq.top();
        pq.pop();
        int B = tmp.first;
        int u = tmp.second;
        add_flow=min(add_flow,B);
        if(u==t) return true;
        for(auto v:a[u])
            if(par[v]==-1 && min(B,RC[u][v]) > Bmax[v])
            {
                Bmax[v] = min(B,RC[u][v]);
                par[v] = u;
                pq.push({Bmax[v],v});
            }
    }
    return false;
}
//-----
int Maxflow(int s, int t, int n)
{
    int flow = 0;
    while(pfs(s,t,n))
    {
        flow += add_flow;
        int v = t;
        while(v!=s)
        {
            int u = par[v];
            RC[v][u]+=add_flow;
            RC[u][v]-=add_flow;
            v = u;
        }
    }
    return flow;
}
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
-----  
void dijkstra(int s, int num) {  
    for (int i = 1; i <= n; i++)  
        dist[i][num] = INF;  
    dist[s][num] = 0;  
    pq.push({dist[s][num], s});  
    while (!pq.empty()) {  
        auto temp = pq.top();  
        pq.pop();  
        int d = temp.first, u = temp.second;  
        if (dist[u][num] != d) continue;  
        for (auto& e : adj[u]) {  
            int v = e.first;  
            int w = edges[e.second].first;  
            if (dist[v][num] > dist[u][num] + w) {  
                dist[v][num] = dist[u][num] + w;  
                pq.push({dist[v][num], v});  
            }  
        }  
    }  
}  
-----  
void themcung(int u, int v, int c)  
{  
    a[u].push_back(v);  
    a[v].push_back(u);  
    RC[u][v] = c;  
}  
-----  
int main() {  
    freopen("CEDGEDES.INP", "r", stdin);  
    freopen("CEDGEDES.OUT", "w", stdout);  
    ios_base::sync_with_stdio(0);  
    cin.tie(0);  
    cout.tie(0);  
  
    cin >> n >> m;  
    s=1; t=n;  
    for (int i = 1; i <= m; i++) {  
        int u, v, w, d;  
        cin >> u >> v >> w >> d;  
        adj[u].push_back({v, i});  
        adj[v].push_back({u, i});  
        edges[i] = {w, d};  
    }  
    dijkstra(1, 0); //đánh dấu 0 - Kc ngắn nhất từ đỉnh 1 đến các đỉnh  
    dijkstra(n, 1); //đánh dấu 1 - Kc ngắn nhất từ đỉnh n đến các đỉnh  
    int distance = dist[n][0]; //Kc ngắn nhất từ 1 --> n  
  
    queue<int> q; //Dùng BFS để duyệt giúp tạo các cung trên G the chiều từ s --> t  
    q.push(1); vis[1] = 1;  
    while (!q.empty()) {  
        int u = q.front();  
        q.pop();  
        for (auto& e : adj[u]) {
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
int v = e.first;
auto& cur = edges[e.second];
//Kc ngắn nhất từ 1 ---> u + w(u,v) + Kc ngắn nhất từ n --> v
if (dist[u][0] + cur.first + dist[v][1] == distance)
    themcung(u, v, cur.second);
//Thêm cung e(u,v) vào mạng G,
//với đk trên thì cung (v,u) sẽ ko được thêm vào G
//c(e) = chi phí phá hủy cạnh (u,v)
if (vis[v]) continue;
vis[v] = 1;
q.push(v);
}
}
cout << Maxflow(s,t,n) << '\n';//Xuất giá trị luồng cực đại = giá trị lát cắt cực
tiêu
return 0;
}
```

4.6.4. Link tải bộ test

https://drive.google.com/drive/folders/1IrrH2g8t1qiYHVeGd5dlJXNY_taVL5Nn?usp=sharing

4.7. DTELETOWER

Tháp truyền tin (Telecom Tower) là một phần quan trọng trong mạng lưới viễn thông của mỗi đất nước. Thực tế chúng là đắt nhất và có giá trị sử dụng rất cao. Một công ty viễn thông đang xây dựng n tháp truyền tin tại đất nước Byte Land. Để định vị các tháp truyền tin, công ty sử dụng hệ trục tọa độ Cartesian quen thuộc (Oxy). Tháp thứ i được đặt tại tọa độ (x_i, y_i) .

Sau khi xây dựng các tháp truyền tin, công ty nhận ra rằng có một số cuộc gọi bị gián đoạn xảy ra với người dùng. Một cuộc điều tra được tiến hành. Và nguyên nhân được xác định thông qua nhiều cuộc gọi có đặc điểm chung là đều được thực hiện giữa hai tháp truyền tin có khoảng cách Euclidean là d. Để giải quyết sự cố này, công ty quyết định sẽ hủy một số tháp truyền tin, sao cho không còn hai tháp nào có khoảng cách là d tồn tại.

Yêu cầu: Bạn hãy giúp công ty xác định số tháp cần hủy ít nhất để xử lý sự cố trên.

Input: DTELETOWER.INP

- Dòng 1: Ghi 2 số nguyên n và d, cho biết số lượng tháp được xây dựng và khoảng cách d là nguyên nhân mà các cuộc gọi bị gián đoạn.
- n dòng tiếp theo: Mỗi dòng ghi 2 số nguyên x_i và y_i cho biết tọa độ của tháp truyền tin thứ i

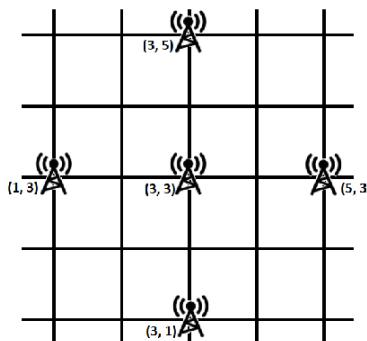
Output: DTELETOWER.OUT

- Ghi một số duy nhất là số lượng tháp truyền tin cần hủy ít nhất sao cho không còn 2 tháp nào có khoảng cách là d

Ví dụ:

DTELETOWER.INP	DTELETOWER.OUT
5 2	1
1 3	
3 1	
3 3	
3 5	
5 3	

Giải thích: Hình dưới mô tả mạng lưới các tháp truyền tin, ta sẽ hủy tháp có tọa độ (3,3)



Ràng buộc:

- $2 \leq n \leq 10^4$
- $1 \leq d \leq 200$
- $1 \leq x_i, y_i \leq 200$
- Không có 2 tháp nào có cùng tọa độ

4.7.1. DTELETOWER_SOLUTION

Ý tưởng

Gọi S là tập các điểm đã cho sao cho không có 2 điểm nào trong S có khoảng cách là d . Ta cần tìm sao cho tập S là lớn nhất có thể. Và kết quả của bài toán sẽ là $n - |S|$

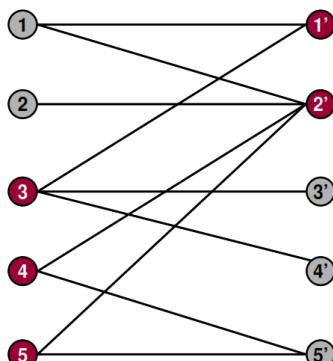
Thuật toán:

- Ta xây dựng một đồ thị G với V đỉnh thuộc tập các điểm trên mặt phẳng tọa độ đã cho. Và giữa 2 đỉnh u, v tồn tại cạnh khi nào khoảng cách $\text{dis}(u, v) = d$
 - Gọi S là tập các đỉnh độc lập trên G . Các đỉnh độc lập trên đồ thị được hiểu là giữa mọi cặp đỉnh trong S không có cạnh nối.
- ⇒ Ta có $|S| = V - \text{Số lượng đỉnh phủ tối thiểu}$ (**minimum vertex cover**)

Vậy tập đỉnh phủ (Vertex cover) là gì?

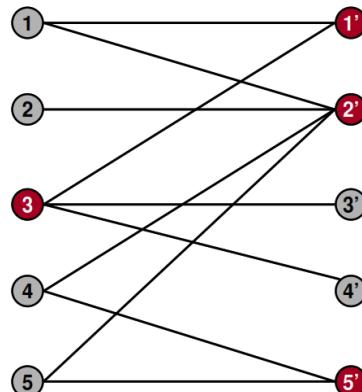
Cho một đồ thị vô hướng $G = (V, E)$, một tập đỉnh phủ trên G là tập các đỉnh $C \subseteq V$ sao cho: Mọi cung $(u, v) \in E$ thì hoặc $u \in C$ hoặc $v \in C$

Ví dụ:



$$C = \{ 3, 4, 5, 1', 2' \}$$

$$|C| = 5$$



$$C = \{ 3, 1', 2', 5' \}$$

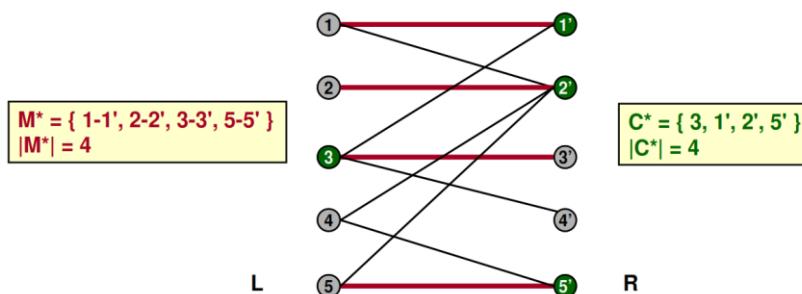
$$|C| = 4$$

Trở lại bài toán. Việc tìm tập S các đỉnh độc lập trong các dạng đồ thị tổng quát không phải dễ. Song, trong trường hợp này, bất kỳ đồ thị G nào có các đỉnh thuộc mặt phẳng tọa độ Oxy, và độ dài các cạnh là d , thì đều là đồ thị hai phía (Bipartite Graph). Ta sẽ chứng minh vấn đề này bên dưới (*).

4.7.2. Định lý Konig

Trong đồ thị hai phía, số lượng cạnh trong **cặp ghép cực đại** bằng với số lượng đỉnh trong **tập phủ tối thiểu**.

Ví dụ:



Theo định lý Konig thì kết quả của bài toán sẽ là:

$$n - |S_{\text{tập đỉnh độc lập}}| = n - (n - \text{Số lượng đỉnh phủ tối thiểu}) = \text{Số lượng đỉnh phủ tối thiểu}$$

→ Đây chính là bài toán cặp ghép cực đại trên đồ thị hai phía ta đã tìm hiểu trong bài CBADGES ở trên.

Do số lượng đỉnh lớn, nên ta có thể dùng thuật toán Dinic với độ phức tạp $O(m\sqrt{n})$

Tiếp theo ta sẽ chứng minh vấn đề (*) ở trên:

Bất kỳ đồ thị G nào có các đỉnh thuộc mặt phẳng tọa độ Oxy, và độ dài các cạnh là d , thì đều là đồ thị hai phía (Bipartite Graph).

Chứng minh:

Xét trường hợp 1: d là số lẻ

Ta nhận thấy sẽ có một vài cặp số (a,b) sao cho $a^2 + b^2 = d^2 \rightarrow (a+b)\%2 = 1$. Nghĩa là một trong 2 số a, b là lẻ hay $(a+b)$ là số lẻ (**).

Với đồ thị G đã xây dựng, ta tiến hành tô màu các đỉnh có tọa độ (x,y) theo màu $(x+y)\%2$. Vì các cạnh đều có độ dài d , nên đỉnh kề với đỉnh (x,y) sẽ là đỉnh $(x+a, y+b)$.

Cụ thể:

- Nếu $x+y$ là chẵn, thì $x+a + y+b = (x + y) + (a + b)$ sẽ là số lẻ \rightarrow đỉnh (x,y) sẽ có màu 0, và đỉnh $(x+a, y+b)$ sẽ có màu 1
- Tương tự cho trường hợp đỉnh (x,y) có $x+y$ là lẻ tương ứng với màu 1, thì đỉnh $(x+a, y+b)$ sẽ là chẵn, tương ứng với màu 0.

→ Ta thấy không có cặp đỉnh nào trùng màu, nên đồ thị G trở thành đồ thị hai phía.

Một cách chứng minh khác:

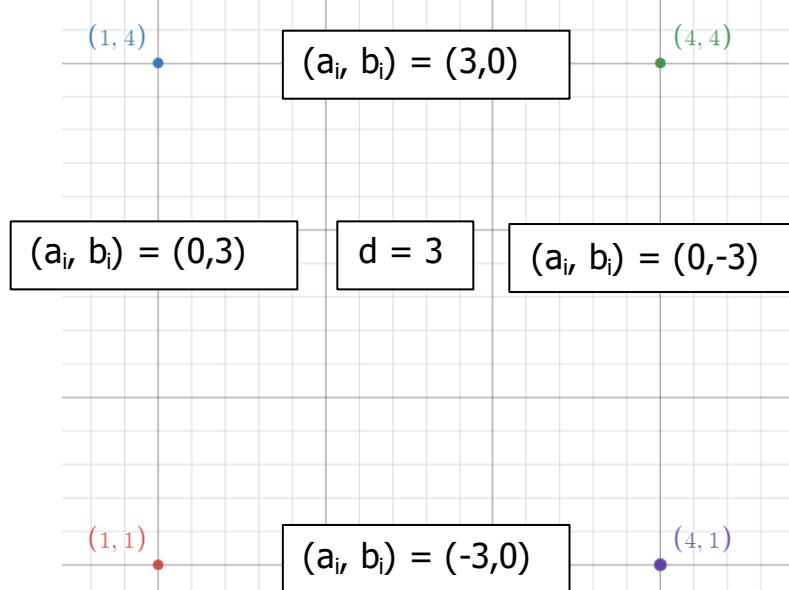
Ta thấy rằng trong đồ thị hai phía thì mọi chu trình đều có độ dài chẵn⁶. Vì vậy chỉ cần chứng minh G không có chu trình độ dài lẻ là xong.

→ Ta xét một chu trình bất kỳ của G . Mỗi cạnh của chu trình sẽ có một giá trị x tịnh tiến, và y tịnh tiến mà ta gọi lần lượt các giá trị tịnh tiến đó là a_i và b_i . Ta sẽ có:

- $a_i^2 + b_i^2 = d^2$
- $\sum a_i = 0$, do quay về đỉnh xuất phát
- $\sum b_i = 0$, do quay về đỉnh xuất phát

Ta thấy nếu a_i lẻ thì phải có chẵn cạnh để $\sum a_i = 0$ xảy ra. Tương tự nếu b_i là lẻ thì cũng phải có chẵn cạnh để $\sum b_i = 0$ xảy ra. Mà theo nhận xét ở (**) trên thì a , hoặc b phải là lẻ. Nghĩa là mọi chu trình phải có số cạnh là chẵn.

⁶ <https://math.stackexchange.com/questions/311665/proof-a-graph-is-bipartite-if-and-only-if-it-contains-no-odd-cycles>



Quan sát hình trên ta thấy chu trình trên xuất phát từ đỉnh (1,1) qua 4 cạnh, và đều có khoảng cách là 3, ta có:

- Tổng $a_i = 0 + 3 + 0 + -3 = 0$
- Tổng $b_i = 3 + 0 + -3 + 0 = 0$

Xét trường hợp 2: d là số chẵn

- Vì d là chẵn và $a^2 + b^2 = d^2$ nên a, b hoặc cùng chẵn, hoặc cùng lẻ
 - ✓ Nếu a, b cùng lẻ thì: $(a^2 + b^2) \% 4 = 2$ mà $d^2 \% 4 = 0 \rightarrow$ Vô lý
 - ✓ Nếu a, b cùng chẵn thì: Ta chuyển trực tọa độ (x, y) thành trực tọa độ $(\frac{x}{2}, \frac{y}{2}) \rightarrow d$ cũng sẽ giảm xuống còn thành $d = \frac{d}{2}$. Ta sẽ có 2 trường hợp:
 - Nếu d lẻ \rightarrow Ta quay trở lại trường hợp 1 ở trên
 - Nếu d vẫn chẵn \rightarrow Ta tiếp tục chuyển đổi trực tọa độ cho đến khi d lẻ

4.7.3. Độ phức tạp

- Xử lý nhập liệu: $O(n.c.logn)$
- Tô màu đồ thị: $O(m + n)$
- Tìm luồng cực đại trên Unit Network với Dinic: $O(E\sqrt{V}) \approx O(m\sqrt{n})$ – riêng bài toán này $\Rightarrow O(m\sqrt{n})$

4.7.4. Code tham khảo

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;
const int N = 10005;
map<pii,int> M;
int col[N];
int n,d,s,t,level[N],par[N],RC[N][N],add_flow;
vector<int> adj[N];
vector<int> a[N];
//-----
bool BFS(int s, int t, int n)
{
    for (int i = 0 ; i <= n ; i++)
        level[i] = -1;
    level[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int v : adj[u])
        {
            if (RC[u][v] > 0 && level[v] == -1)
            {
                level[v] = level[u] + 1;
                par[v] = u;
                q.push(v);
            }
        }
    }
    if (level[t] == -1)
        return false;
    else
        return true;
}
int maxFlow()
{
    int flow = 0;
    while (BFS(s, t, n))
    {
        int u = t;
        int minCapacity = INT_MAX;
        while (u != s)
        {
            int v = par[u];
            minCapacity = min(minCapacity, RC[v][u]);
            u = v;
        }
        u = t;
        while (u != s)
        {
            int v = par[u];
            RC[v][u] -= minCapacity;
            RC[u][v] += minCapacity;
            u = v;
        }
        flow += minCapacity;
    }
    return flow;
}
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
    level[i] = -1, par[i] = -1;
level[s] = 0;
par[s] = 0;
queue< int > q;
q.push(s);
while (!q.empty())
{
    int u = q.front();
    q.pop();
    for(auto v:adj[u])
        if (level[v] < 0 && RC[u][v])
    {
        level[v] = level[u] + 1;
        par[v] = u;
        q.push(v);
    }
}
return level[t] < 0 ? false : true;
}

-----
int sendFlow(int u, int flow)
{
    if (u == t)
        return flow;
    for(auto v:adj[u])
    {
        if (level[v] == level[u]+1 && RC[u][v])
        {
            int curr_flow = min(flow, RC[u][v]);
            int add_flow = sendFlow(v, curr_flow);
            if (add_flow > 0)
            {
                RC[v][u] += add_flow;
                RC[u][v] -= add_flow;
                return add_flow;
            }
        }
    }
    return 0;
}

-----
int DinicMaxflow(int s, int t, int n)
{
    if (s == t)
        return -1;
    int flow = 0;
    while (BFS(s, t, n))
        while (int add_flow = sendFlow(s, INT_MAX))
            flow += add_flow;
    return flow;
}

-----
void dfs(int u)
{
    for(auto v:a[u])
        if(col[v] == -1)
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
        {
            col[v] = 1 - col[u];
            dfs(v);
        }
    }
//-----
void themcung(int u, int v, int c)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
    RC[u][v] = c;
}
//-----
void nhap()
{
    cin>>n>>d;
    vector <pii> ds;//danh sách các cặp giá trị tịnh tiến có khoảng cách trên hệ oxy là d
    for (int i = -d; i <= d; ++i)
        for (int j = -d; j <= d; ++j)
            if(i*i + j*j == d*d)
                { ds.push_back(pii(i,j)); }

    for (int i = 1; i <= n; ++i)//Nhập các đỉnh với độ phức tạp n.c.logn
    {
        int x,y;
        cin>>x>>y;
        M[pii(x, y)] = i;
        for (int j = 0; j < ds.size(); ++j)
        {
            int nx = x + ds[j].first, ny = y + ds[j].second;
            if(M.find(pii(nx,ny)) != M.end())
            {
                int id = M[pii(nx,ny)];
                a[i].push_back(id);
                a[id].push_back(i);
            }
        }
    }
    memset(col, -1, sizeof col);//color
    for (int i = 1; i <= n)//Tô màu đồ thị với 2 màu 0, 1
    {
        if(col[i] == -1)
        {
            col[i] = 0;
            dfs(i);
        }
    }
}
//-----
int main()
{
    freopen("DTELETOWER.INP","r",stdin);
    freopen("DTELETOWER.OUT","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    nhap();
}
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
s = n+1; t = n+2;

for (int i = 1; i <= n; ++i)
    if(col[i] == 0)//Các đỉnh màu 0 thuộc phía s
    {
        themcung(s,i,1);
        for (auto j:a[i])
            themcung(i,j,1);//Các cạnh từ đỉnh màu 0 --> đỉnh màu 1
    }
    else
        themcung(i,t,1);//các đỉnh màu 1 thuộc phía t

cout<<DinicMaxflow(s,t,n+2)<<"\n";
return 0;
}
```

4.7.5. Link tải bộ test

<https://drive.google.com/drive/folders/188rXRoemsQv7uG0fMxL49HUGuyZojHu1?usp=sharing>

4.8. DGRIDS

Các bài toán về đồ thị trên lưới ô vuông được khai thác rất nhiều trong các kỳ thi học sinh giỏi. Bài toán lần này khá thú vị và khó cho các em!

Ta sẽ có một lưới ô vuông D kích thước $n \times m$, gồm n dòng và m cột. Ta được cho một dãy n số nguyên $a_1, a_2, a_3, \dots, a_n$. Với a_i là tổng các phần tử trên dòng thứ i . Và một dãy gồm m số nguyên $b_1, b_2, b_3, \dots, b_m$. Với b_i là tổng các phần tử trên cột thứ i .

Yêu cầu: Ta cần xây dựng một lưới được lắp đầy giá trị các ô bởi các số nguyên thuộc $[1, 5]$. Ta chọn cách sắp sao cho dãy các phần tử $(d_{11}, d_{12}, d_{21}, d_{22})$ có thứ tự từ điển là nhỏ nhất.

Chú thích:

- d_{ij} : là biểu diễn cho giá trị tại dòng i cột j của lưới
- Nếu ta có 2 phương án xây dựng:

a_1	a_2		
a_3	a_4		

b_1	b_2		
b_3	b_4		

Trong đó ta có dãy (a_1, a_2, a_3, a_4) có thứ tự từ điển nhỏ hơn dãy (b_1, b_2, b_3, b_4) nếu tồn tại một chỉ số i sao cho $a_i < b_i$

Input: DGRIDS.INP

- Dòng đầu tiên: Ghi số nguyên T là số lượng test case

Với mỗi test case ta sẽ có:

- Dòng 1: Ghi 2 số nguyên n và m là số dòng và số cột
- Dòng 2: Ghi n số nguyên $a_1, a_2, a_3, \dots, a_n$
- Dòng 3: Ghi m số nguyên $b_1, b_2, b_3, \dots, b_m$

Output: DGRIDS.OUT

- Ghi ra các giá trị trên lưới $n \times m$ tìm được sao cho thỏa yêu cầu đề bài

Ví dụ:

DGRIDS.INP	DGRIDS.OUT
1	1 1 4
3 3	3 5 2
6 10 6	2 3 1
6 9 7	

Ràng buộc:

- $t \leq 50$
- Sub1: $1 \leq m, n \leq 30$, Time: 1s
- Sub2: $30 < m, n \leq 35$, Time: 3s
- Sub3: $35 \leq m, n \leq 40$, Time: 5s

4.8.1. DGRIDS_SOLUTION

Ta sẽ tìm giải pháp để xây dựng lưới đáp án bằng cách mô hình hóa bài toán thành dạng luồng trên mạng.

Điều quan trọng là chúng ta sẽ lắp các ô của lưới bằng một số nguyên thuộc [1,5]. Ta thấy rằng mỗi phần tử trong lưới có giá trị nhỏ nhất là 1. Nên đầu tiên, ta sẽ gán cho mỗi ô giá trị 1. Sau khi gán ta có tổng các dòng, cột còn lại như sau:

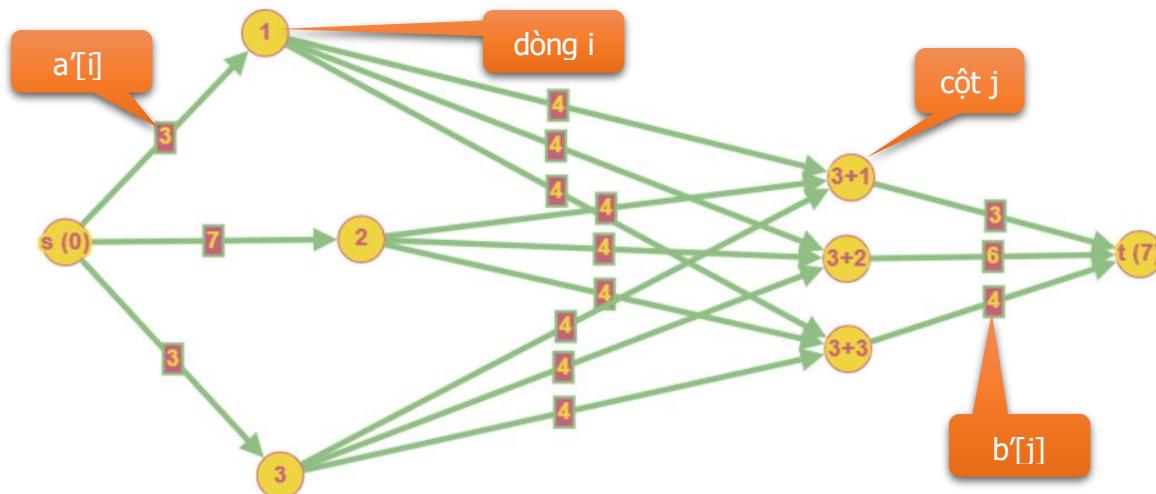
- $a'[i] = a_i - m$, $\forall i \in [1..n]$.
- $b'[j] = b_j - n$, $\forall j \in [1..m]$

Bây giờ, ta chỉ cần lắp các ô của lưới bằng một số nguyên thuộc [0,4] sao cho tổng dòng $i = a'_i$ và tổng cột $j = b'_j$

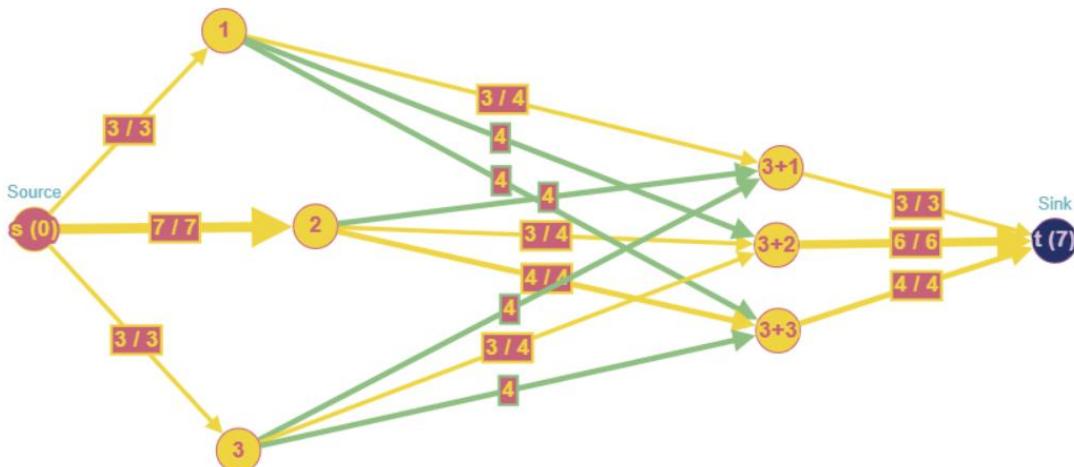
Ta thực hiện xây dựng mạng G như sau:

- Số lượng đỉnh: $n+m+2$
- Đỉnh phát $s = 0$
- Đỉnh thu $t = n+m+1$
- Mỗi dòng i ta tạo thành 1 đỉnh số hiệu i
- Mỗi cột j ta tạo thành 1 đỉnh số hiệu $n+j$
- Thêm các cung $e(s,i)$ với giá trị $c(e) = a'[i]$
- Thêm các cung $e(n+j,t)$ với giá trị $c(e) = b'[j]$
- Thêm các cung $(i,n+j)$ ứng với ô dòng i , cột j . Ta thấy rằng mỗi ô đều có giá trị tối đa là 4. Và trên dòng i ta phải lắp hết các ô từ cột 1 → m. Vậy nên với mỗi đỉnh i ta sẽ thêm các cung $(i, n+1); (i, n+2); (i, n+3); \dots (i, n+m)$; mỗi cung e có giá trị $c(e) = 4$
- ➔ Giá trị $\text{val}(f^*)$ trên G cho ta một phương án lắp đầy lưới đáp án: Giá trị luồng $f(e)$ trên mỗi cung $e(i, n+j)$ chính là giá trị $d[i][j]$ trên lưới. Kết quả: $d[i][j]+1$

Theo như test mẫu thì ta sẽ xây dựng mạng G như hình sau:



Sau khi chạy một thuật toán luồng cực đại thì ta có 1 phương án như sau:



Nhìn vào các luồng $f(e)$ trên mỗi cạnh $(i, n+j)$ ta có thể suy ra phương án cho lưới kết quả D như sau:

3	0	0
0	3	4
0	3	0

➡ $d[i][j] + 1$

6	9	7	b_j
4	1	1	a_i
1	4	5	6
1	4	1	10

Quan sát phương án trên ta thấy chưa thể đạt được yêu cầu đề bài. Ta tiến hành cải tiến thuật toán như sau:

Ý tưởng:

- Xét cung $e_1 (1, n+1)$ ứng với ô $d[1][1]$, và luồng đi qua là $f(e_1)$. Ta sẽ thử dồn giá trị luồng $f(e_1)$ sang các cung $e_2 (1, n+2)$ và $e_3 (1, n+3)$ xem sao?
 - Giả sử ta giảm được luồng $f(e_1)$ một lượng v . Và các luồng $f(e_2)$, $f(e_3)$ sẽ tăng thêm một lượng v_1 , v_2 theo thứ tự, sao cho $v_1+v_2 = v$
- Giá trị ô $d[1][1] = f(e_1) - v$

Vậy câu hỏi là: Làm sao để dồn giá trị luồng $f(e_1)$ sang các luồng còn lại?

Trả lời:

Ta sẽ thực hiện như sau:

- Ta thực hiện xóa cung $e_1(1,n+1)$ khỏi mạng $G \rightarrow$ giảm luồng trên các cung $(s,1)$ và $(n+1,t)$ một lượng là $f(e_1)$
 - Chạy lại thuật toán tìm luồng cực đại \rightarrow sẽ làm tăng các luồng trên các cung $e_2 (1, n+2)$ và $e_3 (1, n+3)$ một lượng có tổng là v
 - Gán giá trị $d[i][j] = f(e_1) - v$
 - Khôi phục luồng trên các cung $f(s,1) = a'[1]$, $f(n+1, t) = b'[1]$
 - Thực hiện các bước từ 1 → 3 lần lượt với các cung $e_2(1,n+2)$, $e_3(1,n+3)$, $e_4(2, n+1)$,...
- Kết quả ta sẽ được phương án có thứ tự từ điển nhỏ nhất thỏa yêu cầu của đề bài.

4.8.2. Độ phức tạp

- Số lượng cung ($i, n+i$) = $n*m$
- Tìm luồng cực đại: $O(E^2 \log(V) \ln(|f^*|))$
- $\Rightarrow O(n*m* E^2 \log(V) \ln(|f^*|))$

4.8.3. Code tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define N 82
typedef pair<int, int> node;
int n,m,s,t,q,level[N],a[N],b[N],par[N],RC[2*N+1][2*N+1],add_flow;
vector<int> adj[N];
-----
bool pfs(int s, int t,int n)
{
    int Bmax[n+1]={0};
    for (int i = 0 ; i <= n ; i++)
        par[i] = -1;
    par[s] = 0; Bmax[s] = INT_MAX;
    add_flow = INT_MAX;
    priority_queue <node> pq;
    pq.push({Bmax[s],s});
    while (!pq.empty())
    {
        pair<int, int> tmp;
        tmp = pq.top();
        pq.pop();
        int B = tmp.first;
        int u = tmp.second;
        add_flow=min(add_flow,B);
        if(u==t) return true;
        for(auto v:adj[u])
            if(par[v]==-1 && min(B,RC[u][v]) > Bmax[v])
            {
                Bmax[v] = min(B,RC[u][v]);
                par[v] = u;
                pq.push({Bmax[v],v});
            }
    }
    return false;
}
-----
int Maxflow(int s, int t, int n)
{
    int flow = 0;
    while(pfs(s,t,n))
    {
        flow += add_flow;
        int v = t;
        while(v!=s)
        {
            int u = par[v];
            RC[v][u]+=add_flow;
            RC[u][v]-=add_flow;
            v = u;
        }
    }
}
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
        }
    }
    return flow;
}
//-----
void nhap()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++) {
        cin>>a[i];
        a[i]-=m;//Trừ đi m đơn vị do mỗi cột đã điền 1
    }
    for(int i=1;i<=m;i++) {
        cin>>b[i];
        b[i]-=n;//Trừ đi n đơn vị do mỗi dòng đã điền 1
    }
    s = 0; t = n+m+1;
}
//-----
void themcung(int u, int v, int c)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
    RC[u][v] = c;
}
//-----
//xoá cung (i,n+j)
void xoacung(int i, int j)
{
    RC[i][n+j] = 0;
    RC[n+j][i] = 0;
}
//-----
void solve(){
    for(int i=1;i<=n;i++)//Thêm các cung e từ i --> j, có c(e) = 4
        for(int j=n+1;j<=n+m;j++)
            themcung(i,j,4);
    for(int i=1;i<=n;i++)//Thêm cung e: s --> i (dòng i), với c(e) = a[i] (tổng dòng còn lại)
        themcung(0,i,a[i]);
    for(int j=1;j<=m;j++)//Thêm cung e: cột j (đánh số là đỉnh n+j) --> t, với c(e) = b[j] (tổng cột còn lại)
        themcung(j+n,n+m+1,b[j]);

    int k=Maxflow(s,t,n+m+2);//k = val(f*)

    int ans[101][101];//Mảng kết quả
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            int tmp=RC[n+j][i];//luồng qua cung (i,n+j)
            //Luồng trên cung (s,i) giảm đi 1 lượng bằng luồng đã có trên cung (i,n+j)
            RC[s][i] += tmp; RC[i][s] -= tmp;
            //Luồng trên cung (n+j,t) giảm đi 1 lượng bằng luồng đã có trên cung (n+j,t)
            RC[n+j][t] += RC[n+j][i]; RC[t][n+j] -= tmp;

            xoacung(i,j);//xoá cung (i,n+j)
        }
    }
    int p1=Maxflow(s,t,n+m+2);//đồn luồng sang các cung khác
}
```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
RC[s][i] = 0; RC[i][s] = a[i]; //Khởi tạo lại luồng trên cung (s,i) là a[i]
RC[n+j][t] = 0; RC[t][n+j] = b[j]; //Khởi tạo lại luồng trên cung (n+j,t) là b[i]
ans[i][j+n]=tmp-p1;//Kết quả tại ô (i,j)
}
}
for(int i=1;i<=n;i++) {
    for(int j=n+1;j<=n+m;j++)
        cout<<ans[i][j]+1<<' ';
    cout<<"\n";
}
//-----
int main() {
    freopen("DGRIDS.INP","r",stdin);
    freopen("DGRIDS.OUT","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(NULL);
    cin>>q;
    while(q--) {
        nhap();
        solve();
        for(int i=0; i<=n; i++)
            RC[i][s] = 0;
        for(int j=n+1;j<=n+m; j++)
            RC[t][j] = 0;
        for(int i=0;i<= n+m+1; i++)
            adj[i].clear();
    }
    return 0;
}
```

4.8.4. Link tải bộ test

<https://drive.google.com/drive/folders/1IXzKtQpcJJVyMpPDmxWhQan6SDImzhwj?usp=sharing>

4.9. ECOVERCHESS

Cho một lưới ô vuông có n dòng và m cột. Mỗi ô vuông trên lưới có một giá trị nhất định. ô vuông có địa chỉ tại dòng i , cột j sẽ có giá trị là $v_{i,j}$.

Bạn có thể chọn một ô vuông có địa chỉ (i,j) trên lưới theo hai cách như sau:

- Cách 1: Ta có thể chọn trực tiếp ô vuông, sẽ tốn một chi phí là $c_{i,j}$
- Cách 2: Ta sẽ chọn gián tiếp ô vuông, sẽ tốn chi phí là $b_{i,j} \leq c_{i,j}$. Ta chỉ có thể chọn gián tiếp ô vuông (i,j) khi mà tất cả các ô vuông kề với ô (i,j) đã được chọn.

Hai ô vuông gọi là kề nhau nếu chúng có chung cạnh.

Giá trị bạn đạt được sẽ bằng tổng giá trị các ô mà bạn chọn trừ đi tổng chi phí cần thiết để chọn các ô đó.

Yêu cầu: Hãy tìm cách chọn các ô sao cho giá trị mà ta đạt được là lớn nhất có thể.

Input: ECOVERCHESS.INP

- Dòng 1: Ghi hai số nguyên n, m
- n dòng tiếp theo: Mỗi dòng ghi m số nguyên. Số thứ i là giá trị $v_{i,j}$
- n dòng tiếp theo: Mỗi dòng ghi m số nguyên. Số thứ i là giá trị $b_{i,j}$
- n dòng tiếp theo: Mỗi dòng ghi m số nguyên. Số thứ i là giá trị $c_{i,j}$

Output: ECOVERCHESS.OUT

- Ghi một số nguyên duy nhất là giá trị đạt được lớn nhất tìm được

Ví dụ:

ECOVERCHESS.INP	ECOVERCHESS.OUT
3 5	13
9 0 3 2 4	
0 2 8 6 2	
5 3 4 1 3	
5 1 7 9 5	
2 4 1 2 4	
9 1 2 2 0	
9 1 8 9 7	
2 7 3 2 9	
9 1 9 8 2	

Giải thích:

Với test mẫu ta sẽ đọc được dữ liệu theo thứ tự như sau:

3 5 Ta sẽ chọn các ô trực tiếp như sau (ô đánh dấu x)

9 0 3 2 4 O X O O O

0 2 8 6 2 X O X X O

5 3 4 1 3 O X O O X

5 1 7 9 5 ➔ Chi phí là: $1+2+3+1+2+2 = 11$

2 4 1 2 4 Ta chỉ chọn 1 ô gián tiếp là ô ở góc trái trên cùng của lưới

9 1 2 2 0 ➔ Chi phí là: 5

9 1 8 9 7 ➔ Tổng chi phí chọn ô là: $11 + 5 = 16$

2 7 3 2 9 ➔ Tổng giá trị các ô đã chọn là: $9+0+0+3+8+6+3 = 29$

9 1 9 8 2 Giá trị đạt được = $29 - 16 = 13$

Ràng buộc:

Với tất cả các sub:

- $0 \leq v_{i,j} \leq 100,000$

Subtask 1 (45%): $n, m \leq 3$

- $0 \leq b_{i,j} \leq c_{i,j} \leq 100,000$

Subtask 2 (35%): $n, m \leq 7$

Subtask 3 (20%): $n, m \leq 25$

- $1 \leq n, m$

4.9.1. ECOVERCHESS_SOLUTION

Với sub1, ta có thể giải bài toán bằng cách duyệt mọi trường hợp. Và nếu như cách duyệt tốt thì có thể ăn một số sub lớn hơn. Tuy nhiên, để có thuật toán tối ưu thì ta nên dùng luồng cực đại.

Nhận xét:

1/ Chúng ta cần tìm giá trị đạt được lớn nhất thông qua việc chọn các ô. Nếu ta chọn 1 ô thì nghĩa là ta sẽ mất đi 1 lượng bằng với chi phí chọn ô đó. Nếu ta không chọn 1 ô thì ta sẽ mất đi 1 lượng bằng với giá trị của ô đó. Nhiệm vụ của ta là tìm cách chọn để **tổng giá trị bị mất là nhỏ nhất**.

2/ Ta sẽ xây dựng một đồ thị dạng mạng G. Trong đó, mỗi cung thể hiện 1 hành động gồm: chọn trực tiếp, chọn gián tiếp, không chọn. Mỗi cung sẽ có một thông lượng bằng với lượng giá trị mà ta sẽ mất đi khi thực hiện hành động đó.

→ Từ 2 nhận xét trên ta thấy bài toán của chúng ta trở thành bài toán tìm "**lát cắt nhỏ nhất**" trên mạng G xây dựng ở trên. Những cung thuộc lát cắt chính là lượng giá trị nhỏ nhất mà ta mất đi tương ứng với các hành động trên mỗi ô.

→ Để tìm lát cắt nhỏ nhất, như đã tìm hiểu ở trên, ta sẽ tìm luồng cực đại của G

Xây dựng mạng G như sau:

- Số đỉnh V = $2*n*m + 2$
- Đỉnh phát s = 1
- Đỉnh thu t = V
- Ta thực hiện "duỗi mảng 2 chiều" thành mảng 1 chiều và khi đó ô (i,j) sẽ có thứ tự là: $(i-1)*m + j$. Ví dụ: trên lưới $n = 3, m = 5$, ô (1,1) có thứ tự là: $(1-1)*5 + 1 = 1$. Ô (2,1) có thứ tự là: $(2-1)*5 + 1 = 6$... Gọi x là thứ tự của ô (i,j) trên lưới. Ta sẽ tách ô (i,j) thành 2 đỉnh lần lượt là $2*x$ và $2*x+1$ (chẵn và lẻ). Bằng cách này ô (i,j) sẽ trở thành 1 cung trên G

Thêm các cung:

- Với mỗi ô (i,j) với x là thứ tự của ô, ta thực hiện:
 - ✓ Thêm cung e(s, $2*x$) với $c(e) = c[i][j]$ tương ứng với chi phí chọn trực tiếp ô (i,j)
 - ✓ Thêm cung e($2*x+1, t$) với $c(e) = b[i][j]$ tương ứng với chi phí chọn gián tiếp ô (i,j)
 - ✓ Thêm cung e($2*x, 2*x+1$) với $c(e) = v[i][j]$ tương ứng với giá trị của ô (i,j)

Từ nhận xét 1 ở trên, khi mỗi cung của ô (i,j) thuộc lát cắt nhỏ nhất, nghĩa là ta sẽ mất đi 1 lượng bằng với thông lượng trên cung đó.

Tuy nhiên, Với cách xây dựng mạng G như trên, ta **không thể hiện được ràng buộc** giữa chọn gián tiếp và trực tiếp như đề mô tả.

Vậy ta cần cải tiến cách xây dựng mạng G như thế nào?

Ta hình dung các ô trên lưới như một bàn cờ vua đen trắng. ô gốc trái trên sẽ là ô đen. Tiếp theo là ô trắng, đen, trắng... Như vậy các ô đen (i,j) sẽ có đặc điểm là $(i+j)\%2 = 0$. Ngược lại các ô trắng sẽ có đặc điểm là $(i+j)\%2 = 1$.

Ta sẽ thực hiện 2 cải tiến sau:

1) Với mỗi ô trăng (i,j) ta sẽ hoán đổi 2 cung $(s, 2*x)$ và $(2*x+1, t)$ cho nhau hay nói cách khác sau khi hoán đổi, cung $(s, 2*x)$ sẽ ứng với chi phí chọn gián tiếp, còn cung $(2*x+1, t)$ ứng với chi phí chọn trực tiếp.

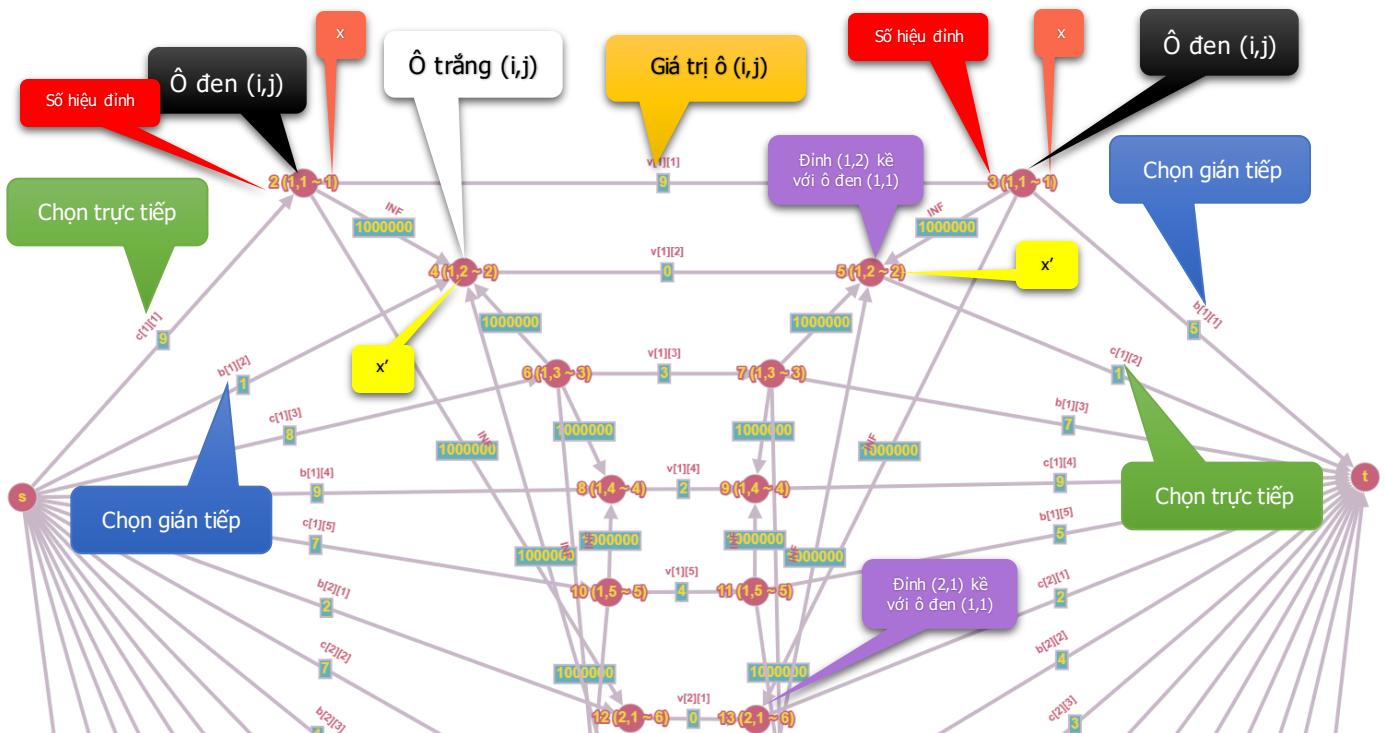
2) Với mỗi ô đen (i,j) , ta sẽ thêm các cung đến các ô (i',j') màu trăng kề với nó. Gọi x là thứ tự của ô (i,j) trong phép duỗi mảng, và x' là thứ tự của ô (i',j') thì ta sẽ thêm các cung: $e(2*x, 2*x')$ và $e(2*x+1, 2*x'+1)$. Mỗi cung sẽ có thông lượng $c(e) = INF$ (vô cùng).

→ Với cách cải tiến trên, ta sẽ thể hiện được ràng buộc của đề bài. Ví dụ:

Xét ô (i,j) màu đen có thứ tự x trong phép duỗi mảng. Các ô trăng kề với (i,j) có thứ tự x' . Ta có:

- Nếu trong lát cắt cực tiểu, ta “cắt” cung “chọn gián tiếp” $(2*x+1, t)$ thì ta phải “cắt” tất cả các cung $(2*x'+1, t)$. Vì nếu không ta vẫn sẽ có đường đi từ $s \rightarrow t$ (trong khi nếu bỏ các cung thuộc lát cắt ta sẽ không thể đi đến t được như đã tìm hiểu ở phần lý thuyết)
- Lập luận tương tự với các ô màu trăng

Hình ảnh minh họa một phần mạng G trong test mẫu đề bài:

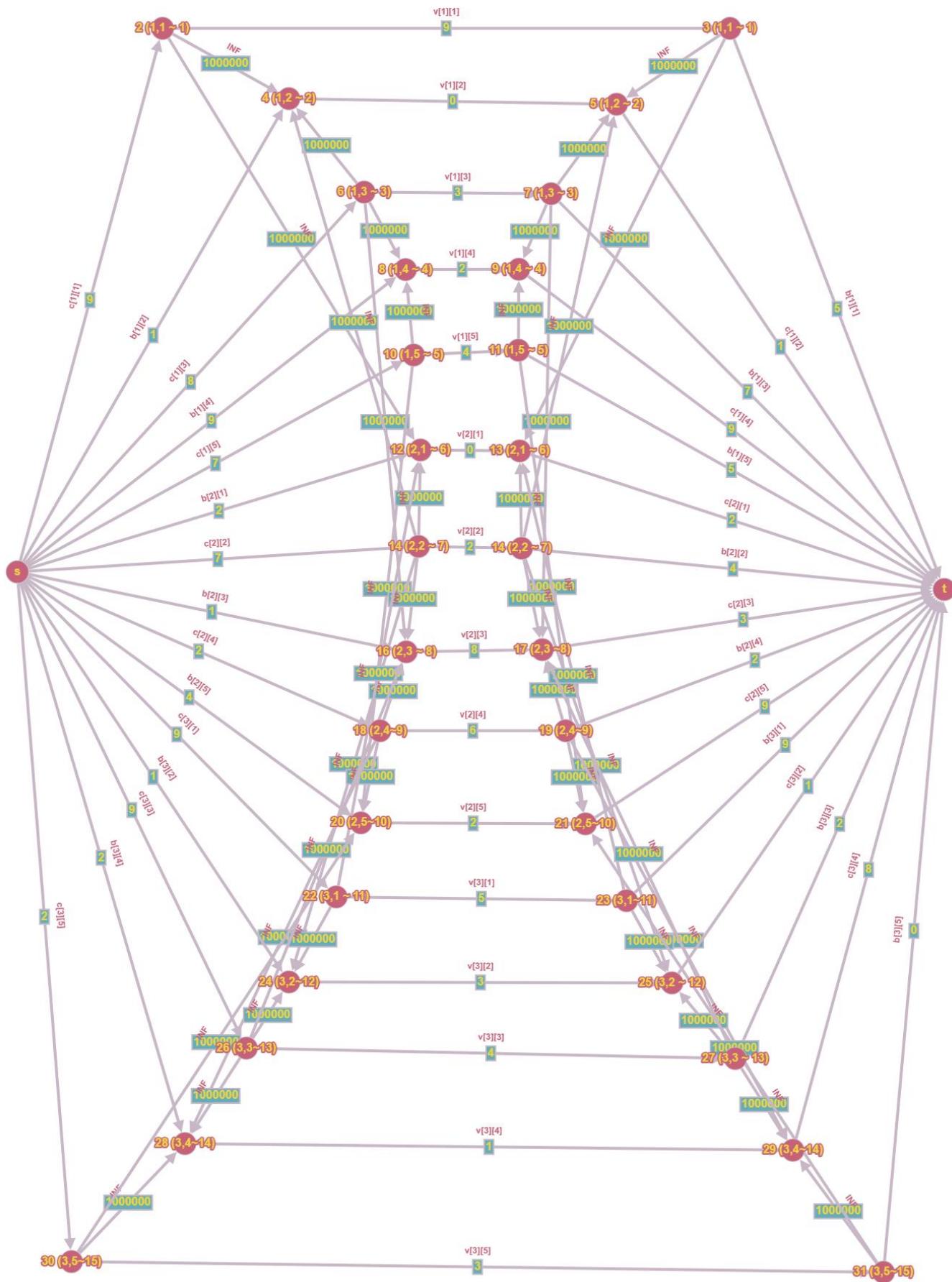


- Nhìn vào hình, xét ô đen $(1,1)$ - đỉnh 3, nếu ta “cắt” cung “chọn gián tiếp” $(3,t)$, thì ta phải “cắt” luôn các cung đến những đỉnh trắng kề $(1,2)$ – đỉnh 5 và $(2,1)$ – đỉnh 13 gồm: cung $(3,5)$ và $(3,13)$. Nếu không ta vẫn sẽ có đường đi từ $s \rightarrow t$ qua đỉnh 3 ($s \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow t$ hoặc $s \rightarrow 2 \rightarrow 3 \rightarrow 13 \rightarrow t$).
- Tương tự, xét ô trăng $(1,2)$ - đỉnh 4, nếu ta “cắt” cung “chọn gián tiếp” $(s,4)$, thì ta phải “cắt” luôn các cung đến những đỉnh đen kề $(1,1)$ – đỉnh 4, $(1,3)$ – đỉnh 6 và $(2,2)$ – đỉnh 14 gồm: cung $(2,4)$, $(6,4)$ và $(14,4)$. Nếu không ta vẫn sẽ có đường đi từ $s \rightarrow t$ qua đỉnh 4 ($s \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow t$, $s \rightarrow 6 \rightarrow 4 \rightarrow 5 \rightarrow t$ hoặc $s \rightarrow 14 \rightarrow 4 \rightarrow 5 \rightarrow t$).

Kết quả bài toán:

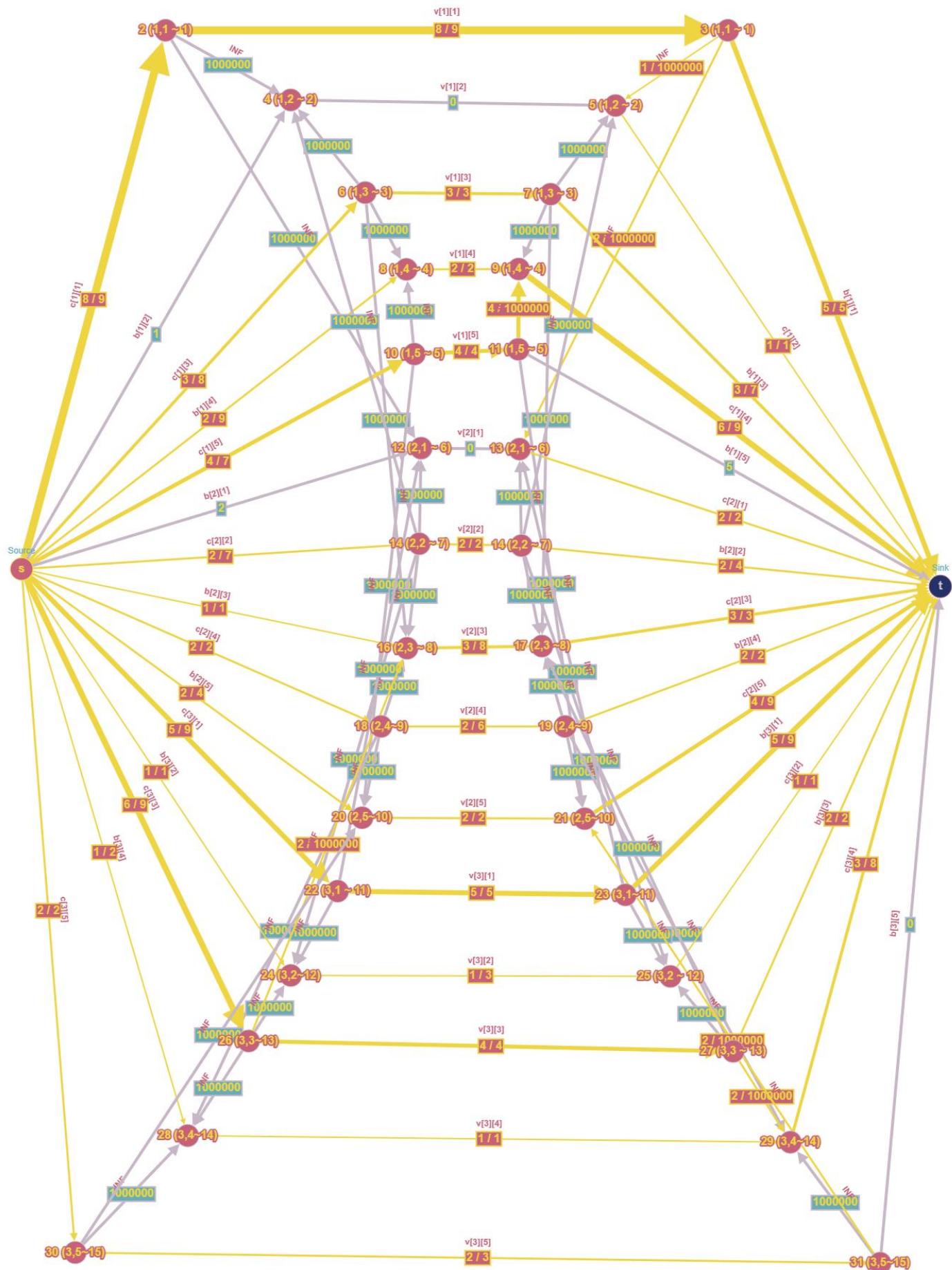
$Val(f^*)$ = Giá trị luồng cực đại trên G = Giá trị của lát cắt cực tiểu trên G = Lượng giá trị ít nhất mất đi của phương án tối ưu.

Output: Tổng giá trị $v(i,j)$ của các ô – $val(f^*)$. Đây chính là lượng giá trị đạt được lớn nhất tìm được.



Hình ảnh minh họa mạng G trên test mẫu của bài toán

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến



Hình ảnh minh họa luồng cực đại trên mạng G trên test mẫu của bài toán

4.9.2. Độ phức tạp

- Nhập dữ liệu: $O(n*m)$
- Tìm luồng cực đại: $O(E^2 \log(V) \ln(|f^*|))$ với E, V là số lượng cung, đỉnh trên G
 $\Rightarrow O(E^2 \log(V) \ln(|f^*|))$

4.9.3. Code tham khảo

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 1e9;
typedef pair<int, int> node;
#define N 2005
int n,m,s,t,par[N],RC[N][N],add_flow;
vector<int> adj[N];
int v[100][100], b[100][100], c[100][100];
//-----
bool pfs(int s, int t,int n)
{
    int Bmax[n+1]={0};
    for (int i = 0 ; i <= n ; i++)
        par[i] = -1;
    par[s] = 0; Bmax[s] = INT_MAX;
    add_flow = INT_MAX;
    priority_queue <node> pq;
    pq.push({Bmax[s],s});
    while (!pq.empty())
    {
        pair<int, int> tmp;
        tmp = pq.top();
        pq.pop();
        int B = tmp.first;
        int u = tmp.second;
        add_flow=min(add_flow,B);
        if(u==t) return true;
        for(auto v:adj[u])
            if(par[v]==-1 && min(B,RC[u][v]) > Bmax[v])
            {
                Bmax[v] = min(B,RC[u][v]);
                par[v] = u;
                pq.push({Bmax[v],v});
            }
    }
    return false;
}
//-----
int Maxflow(int s, int t, int n)
{
    int flow = 0;
    while(pfs(s,t,n))
    {
        flow += add_flow;
        int v = t;
        while(v!=s)
        {

```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```

        int u = par[v];
        RC[v][u] += add_flow;
        RC[u][v] -= add_flow;
        v = u;
    }
}
return flow;
}

//-----
void themcung(int u, int v, int d)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
    RC[u][v] = d;
}

//-----
int getIdx(int i, int j) {
    return (i-1) * m + j;
}

//-----
int main() {
    freopen("ECOVERCHESS.INP", "r", stdin);
    freopen("ECOVERCHESS.OUT", "w", stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> v[i][j];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> b[i][j];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> c[i][j];
        }
    }
    int nodes = 2 * n * m + 2;
    s = 1, t = nodes;
    const int dx[] = {1, -1, 0, 0};
    const int dy[] = {0, 0, 1, -1};
    int total = 0; //tổng giá trị
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            total += v[i][j];
            int curIdx = getIdx(i, j); //Đổi ô (i,j) sang thứ tự 1 chiều
            if ((i + j) % 2 == 0) // (i,j) là ô đen trên bàn cờ
                for (int k = 0; k < 4; k++) {
                    int ci = i + dx[k];
                    int cj = j + dy[k];
                    if (ci < 1 || ci > n) continue;
                    if (cj < 1 || cj > m) continue;
                    if (b[ci][cj] == 1) continue;
                    if (c[ci][cj] == 1) continue;
                    if (RC[ci][cj] <= 0) continue;
                    if (adj[ci].size() <= 1) continue;
                    if (adj[cj].size() <= 1) continue;
                    if (v[ci] == v[cj]) continue;
                    if (v[ci] < 0) continue;
                    if (v[cj] < 0) continue;
                    if (v[ci] > v[cj]) continue;
                    if (v[ci] - v[cj] > 1) continue;
                    if (v[ci] - v[cj] < -1) continue;
                    if (v[ci] - v[cj] == 1) {
                        adj[ci].push_back(cj);
                        adj[cj].push_back(ci);
                        RC[ci][cj] -= 1;
                        RC[cj][ci] += 1;
                    }
                }
            }
        }
    }
}

```

Bài toán Luồng cực đại – Thuật toán Ford Fulkerson và những cải tiến

```
        if (cj < 1 || cj > m) continue;
        int adjIdx = getIdx(ci, cj); //Thứ tự ô trống kề với ô (i,j)
        themcung(2 * curIdx, 2 * adjIdx, INF);
        themcung(2 * curIdx + 1, 2 * adjIdx + 1, INF);
    }
    themcung(s, 2 * curIdx, c[i][j]);
    themcung(2 * curIdx + 1, t, b[i][j]);
} else {
    themcung(s, 2 * curIdx, b[i][j]);
    themcung(2 * curIdx + 1, t, c[i][j]);
}
themcung(2 * curIdx, 2 * curIdx + 1, v[i][j]);
}

}

cout << total - Maxflow(s,t,nodes) << '\n';

return 0;
}
```

4.9.4. Link tải bộ test

https://drive.google.com/drive/folders/1lszJO2z830Ju2PdQIXUdbJXU3I_uxAUQ?usp=sharing

5. Một số bài tập luyện tập:

<https://oj.vnoi.info/problem/nkflow>

<https://oj.vnoi.info/problem/assign1>

<https://oj.vnoi.info/problem/kway>

<https://oj.vnoi.info/problem/stnode>

<https://codeforces.com/problemset?tags=flows>

<https://www.codechef.com/tags/problems/maximum-flow>

6. Tài liệu tham khảo

[1] <https://www.geeksforgeeks.org>

[2] <https://www.giaithuatlaptrinh.com>

[3] <http://people.csail.mit.edu/moitra/docs/6854lec8.pdf>

[4] <https://cp-algorithms.com>

[5] <https://math.stackexchange.com/questions/311665/proof-a-graph-is-bipartite-if-and-only-if-it-contains-no-odd-cycles>

[6] <https://codeforces.com/>

[7] <https://oj.vnoi.info/>

[8] <https://vnoi.info/wiki/Home>

[9] <https://www.codechef.com/>

7. Kết luận:

Bài toán luồng cực đại, ngoài bản chất của bài toán là tìm lưu lượng lớn nhất có thể truyền từ đỉnh phát s đến đỉnh thu t sao cho không bị quá tải tại các cung, còn được ứng dụng trong việc tìm “lát cắt nhỏ nhất”, và “cặp ghép cực đại”. Đây cũng là hai dạng bài khá quan trọng trong bồi dưỡng học sinh giỏi.

Để giải được bài toán luồng cực đại, đòi hỏi học sinh cần có tư duy tưởng tượng phong phú, nhanh chóng xây dựng được một đồ thị mạng G với các đỉnh phát s, đỉnh thu t, và các đỉnh trung gian khác. Học sinh cũng cần xác định được chiều và thông lượng cụ thể trên các cung của G. Bên cạnh đó, học sinh phải làm chủ được mạng thặng dư tương ứng, nhuần nhuyễn các thao tác xóa, khôi phục luồng, “đồn” luồng,... Ngoài ra, đôi khi vẫn cần sử dụng kết hợp với một số thuật toán khác như tìm đường đi ngắn nhất, loang trên mảng 2 chiều kỹ thuật duỗi mảng 2 chiều, ...

Trong suốt quá trình tìm hiểu chuyên đề, mặc dù đã rất cố gắng và tâm huyết rất nhiều, tuy nhiên, vẫn sẽ không thể tránh khỏi những thiếu sót trong việc sưu tầm các dạng bài về luồng cực đại. Bản thân tôi rất mong nhận được những đóng góp, góp ý từ các bạn đồng nghiệp để ngày càng hoàn thiện hơn nữa chuyên đề.

Sau cùng, rất mong chuyên đề này sẽ trở thành một tư liệu quan trọng và bổ ích cho các Thầy Cô tham gia hội thảo lần này nói riêng, và cả nước nói chung có thể sử dụng trong quá trình giảng dạy đội tuyển học sinh giỏi của mình.

Tôi xin chân thành cảm ơn!