

## Hanout Buổi 12: Thuật toán Floyd và bài toán luồng cực đại

### PHẦN 1: LÝ THUYẾT

#### 1. Thuật toán Floyd (Warshall)

##### 1.1. Định lý cơ sở

Bài toán đặt ra là: **Làm thế nào để tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong một đồ thị có trọng số (không âm hoặc âm nhưng không có chu trình âm)?**

Đây là một bài toán kinh điển trong lý thuyết đồ thị và ứng dụng rộng rãi trong mạng máy tính, định tuyến, phân tích mạng xã hội,...

**Định lý động cơ sở** cho phép ta sử dụng kỹ thuật lập trình động (dynamic programming) để giải bài toán trên bằng cách từng bước cải thiện ma trận khoảng cách giữa các đỉnh.

##### Nguyên lý hoạt động

Giả sử ta có một đồ thị có  $n$  đỉnh, đánh số từ 1 đến  $n$ . Ta định nghĩa:

- $D^{(k)}[i][j]$ : độ dài đường đi ngắn nhất từ đỉnh  $i$  đến đỉnh  $j$  chỉ đi qua các đỉnh trung gian thuộc tập  $\{1, 2, \dots, k\}$ .

Khi đó, nguyên lý chính là **xét dần từng đỉnh trung gian**:

$$D^{(k)}[i][j] = \min(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

Nói cách khác, khi xét thêm đỉnh trung gian  $k$ , ta tự hỏi: "*Liệu đường đi ngắn nhất từ  $i$  đến  $j$  có thể được cải thiện nếu ta đi qua  $k$  không?*"

##### Tư duy minh họa bằng ví dụ nhỏ

Giả sử đồ thị ban đầu (ma trận trọng số) như sau:

Ma trận trọng số ban đầu (nếu không có cạnh, gán giá trị  $\infty$ ):

	A	B	C	D
A	0	3	8	$\infty$
B	$\infty$	0	2	5
C	$\infty$	$\infty$	0	1
D	4	$\infty$	$\infty$	0

Sau mỗi vòng xét đỉnh trung gian  $k=1,2,3,4$ , ta cập nhật ma trận khoảng cách.

Ví dụ: Khi xét  $k=B$ , để cập nhật đường đi từ  $A \rightarrow D$ , ta kiểm tra:

$$D^{(B)}[A][D] = \min(D^{(A)}[A][D], D^{(A)}[A][B] + D^{(A)}[B][D]) = \min(\infty, 3+5) = 8$$

## 1.2. Thuật toán Floyd-Warshall

### Nguyên lý tổng quát

Thuật toán Floyd-Warshall dựa trên việc dần dần cải thiện ma trận khoảng cách  $D$  giữa các cặp đỉnh, thông qua việc xét từng đỉnh trung gian. Với mỗi cặp đỉnh  $(i,j)$ , ta tự hỏi: liệu có thể đi từ  $i$  đến  $j$  qua một đỉnh trung gian  $k$  nào đó để rút ngắn quãng đường không?

### Giả mã thuật toán

Giả sử ta có đồ thị có  $n$  đỉnh, trọng số lưu trong ma trận  $D$  ban đầu, với quy ước:

- $D[i][j]$  = trọng số của cạnh từ  $i \rightarrow j$  nếu có cạnh,
- $D[i][j] = \infty$  nếu không có cạnh,
- $D[i][i] = 0$  cho mọi  $i$ .

Giả mã của thuật toán Floyd-Warshall như sau:

```
for k from 1 to n do
  for i from 1 to n do
    for j from 1 to n do
      if  $D[i][j] > D[i][k] + D[k][j]$  then
         $D[i][j] = D[i][k] + D[k][j]$ 
```

### Giải thích cấu trúc 3 vòng lặp lồng nhau

- **Vòng ngoài cùng (k):** duyệt từng đỉnh trung gian  $k$  từ 1 đến  $n$ . Đây là đỉnh được đưa vào tập các đỉnh cho phép làm trung gian tính đường đi.
- **Vòng giữa (i):** xét từng đỉnh bắt đầu  $i$ .
- **Vòng trong cùng (j):** xét từng đỉnh kết thúc  $j$ .

Ở mỗi bước, ta kiểm tra xem đường đi từ  $i \rightarrow j$  có thể được cải thiện bằng cách đi qua  $k$  hay không:

- Nếu  $D[i][k] + D[k][j] < D[i][j]$ , thì cập nhật  $D[i][j]$ .

### Ma trận khoảng cách được cập nhật từng bước

Sau mỗi lần chạy vòng ngoài với một giá trị  $k$  nhất định, toàn bộ ma trận  $D$  được cập nhật với khả năng có thêm những đường đi ngắn hơn. Do đó, **ma trận khoảng cách tiến hóa theo từng bước**, hướng dẫn đến kết quả cuối cùng – chính là đường đi ngắn nhất giữa mọi cặp đỉnh.

### Độ phức tạp

- **Thời gian:**  $O(n^3)$  – vì 3 vòng lặp lồng nhau.
- **Không gian:**  $O(n^2)$  – để lưu ma trận khoảng cách.

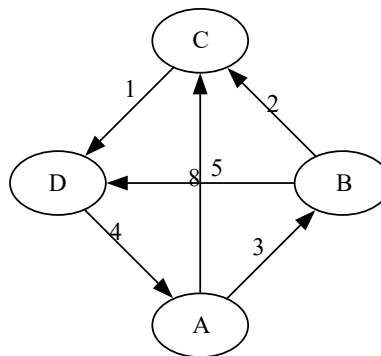
### Lưu ý mở rộng

- Có thể mở rộng thuật toán để lưu lại đường đi cụ thể bằng cách sử dụng một ma trận truy vết (path reconstruction).
- Thuật toán hoạt động được cả với trọng số âm, nhưng **không được có chu trình âm** (vì sẽ gây ra vòng lặp không dừng trong việc giảm chi phí).

## 1.3. Ví dụ minh họa

### Đề bài

Cho đồ thị có 4 đỉnh với các cạnh và trọng số như sau:



Gán tên đỉnh theo số để tiện mô tả:

- $A = 1$

- $B = 2$
- $C = 3$
- $D = 4$

**Bước 1: Khởi tạo ma trận trọng số  $D^{(0)}$**

	1	2	3	4
1	0	3	8	-4
2	$\infty$	0	1	7
3	$\infty$	4	0	$\infty$
4	2	$\infty$	6	0

( $\infty$  biểu thị không có đường đi trực tiếp)

**Bước 2: Lặp qua từng đỉnh trung gian  $k \in \{1, 2, 3, 4\}$**

**Vòng lặp  $k=1$**

Ta xét liệu có đường đi tốt hơn từ  $i \rightarrow j$  thông qua đỉnh 1.

Chẳng hạn:

- $D[4][2] = \infty$ , nhưng  $D[4][1] + D[1][2] = 2 + 3 = 5 \rightarrow$  Cập nhật:  $D[4][2] = 5$
- $D[4][3] = 6$ ,  $D[4][1] + D[1][3] = 2 + 8 = 10 \rightarrow$  Không cập nhật

Sau khi cập nhật, ma trận:

	1	2	3	4
1	0	3	8	-4
2	$\infty$	0	1	7
3	$\infty$	4	0	$\infty$
4	2	5	6	0

**Vòng lặp  $k=2$**

Xét cập nhật qua đỉnh 2.

Ví dụ:

- $D[1][3]=8$ ,  $D[1][2]+D[2][3]=3+1=4 \rightarrow$  Cập nhật:  $D[1][3]=4$
- $D[3][4]=\infty$ ,  $D[3][2]+D[2][4]=4+7=11 \rightarrow$  Cập nhật:  $D[3][4]=11$

Ma trận sau khi cập nhật:

	1	2	3	4
1	0	3	4	-4
2	$\infty$	0	1	7
3	$\infty$	4	0	11
4	2	5	6	0

### Vòng lặp k=3

Cập nhật thông qua đỉnh 3.

Không có cập nhật đáng kể vì không có nhiều đường đi mới tốt hơn khi qua đỉnh 3.

### Vòng lặp k=4

Cập nhật qua đỉnh 4.

Ví dụ:

- $D[2][1]=\infty$ ,  $D[2][4]+D[4][1]=7+2=9 \rightarrow$  Cập nhật:  $D[2][1]=9$
- $D[2][3]=1$ ,  $D[2][4]+D[4][3]=7+6=13 \rightarrow$  Không cập nhật

Ma trận cuối cùng:

	1	2	3	4
1	0	3	4	-4
2	9	0	1	7
3	13	4	0	11
4	2	5	6	0

- Ma trận cuối cùng chứa **khoảng cách ngắn nhất giữa mọi cặp đỉnh**.
- Sinh viên có thể thực hiện lại bằng tay trên giấy hoặc mô phỏng trên máy để rèn kỹ năng bước cập nhật.

## 1.4. Ứng dụng trong Công nghệ Thông tin

Thuật toán Floyd-Warshall được sử dụng rộng rãi trong nhiều lĩnh vực của công nghệ thông tin nhờ khả năng tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị. Dưới đây là một số ứng dụng điển hình:

**a) Mạng máy tính và định tuyến**

- **Xây dựng bảng định tuyến tĩnh:** Floyd-Warshall hỗ trợ xây dựng bảng định tuyến trong các mạng IP, nơi cần xác định đường đi ngắn nhất giữa các nút mạng.
- **Tối ưu hóa truyền tin:** Giúp giảm thiểu độ trễ truyền dữ liệu trong các mạng có kiến trúc phức tạp bằng cách xác định tuyến đường tối ưu.

**b) Hệ thống quản lý giao thông và bản đồ số**

- **Tìm đường ngắn nhất giữa mọi địa điểm:** Các ứng dụng như Google Maps hoặc các hệ thống định vị GPS có thể sử dụng Floyd-Warshall để tiền xử lý dữ liệu bản đồ, đặc biệt hữu ích khi cần truy vấn nhiều lần giữa các cặp địa điểm khác nhau.
- **Phân tích tắc nghẽn:** Giúp đánh giá và so sánh các tuyến đường thay thế trong điều kiện thay đổi.

**c) Lý thuyết đồ thị và phân tích mạng xã hội**

- **Tính toán khoảng cách giữa các nút** trong mạng xã hội để xác định các chỉ số như độ trung gian (betweenness centrality) hoặc độ gần (closeness centrality).
- **Phát hiện cộng đồng:** Qua việc đánh giá độ liên kết giữa các nhóm nút.

**d) Quản lý cơ sở dữ liệu và truy vấn**

- **Tối ưu hóa truy vấn trong các cơ sở dữ liệu đồ thị:** Floyd-Warshall được dùng để tiền xử lý khoảng cách giữa các thực thể (entity) trong hệ thống quản lý dữ liệu có cấu trúc mạng (như Neo4j).
- **Phát hiện mối quan hệ gián tiếp** giữa các thực thể thông qua các đường đi ngắn nhất.

**e) Các bài toán lập lịch và phân tích hệ thống**

- **Phân tích phụ thuộc trong trình biên dịch:** Dùng để xác định thứ tự thực thi các tác vụ phụ thuộc lẫn nhau.

- **Tính toán đường dẫn tới hạn (critical path)** trong các hệ thống lập lịch hoặc quản lý dự án phức tạp.

#### f) An ninh mạng

- **Phân tích độ lan truyền tấn công:** Xác định các con đường ngắn nhất mà tin tặc có thể khai thác từ một điểm yếu cụ thể tới hệ thống quan trọng.
- **Đánh giá rủi ro hệ thống:** Phân tích khoảng cách giữa các thành phần bảo mật quan trọng.

## 2. Bài toán luồng cực đại

### 2.1. Các định nghĩa cơ bản

#### Đồ thị luồng

Một **đồ thị luồng** là một đồ thị có hướng  $G=(V,E)$ , trong đó:

- Mỗi cạnh  $(u,v) \in E$  có **dung lượng (capacity)**  $c(u,v) \geq 0$
- Có một **đỉnh nguồn**  $s \in V$  và một **đỉnh đích (bồn chứa)**  $t \in V$ , với  $s \neq t$

#### Dung lượng cạnh

- $c(u,v)$ : là **giới hạn tối đa** lượng luồng có thể đi qua cạnh  $(u,v)$ .
- Nếu  $(u,v) \notin E$  thì ta coi  $c(u,v)=0$ .

#### Luồng (Flow)

Một **hàm luồng**  $f: V \times V \rightarrow \mathbb{R}$  thỏa mãn ba điều kiện:

##### 1. Giới hạn luồng (capacity constraint):

$$0 \leq f(u,v) \leq c(u,v), \forall u,v \in V$$

##### 2. Luồng bảo toàn (flow conservation):

$$\sum_{v \in V} f(u,v) = \sum_{v \in V} f(v,u)$$

Tức là: tổng luồng đi vào một đỉnh (trừ s và t) bằng tổng luồng đi ra.

### 3. Luồng đối nghịch:

$$f(u,v) = -f(v,u)$$

Điều này đảm bảo rằng nếu có luồng đi từ  $u \rightarrow v$ , thì tương ứng có luồng ngược từ  $v \rightarrow u$  với giá trị âm.

### Luồng khả thi

Một hàm luồng  $f$  là **khả thi** nếu nó thỏa mãn ba điều kiện trên.

### Giá trị của luồng

Giá trị của luồng  $f$  là tổng luồng chảy ra từ nguồn  $s$ , hoặc tương đương là tổng luồng chảy vào đích  $t$ :

$$|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$$

### Luồng cực đại

Luồng cực đại là luồng khả thi có **giá trị lớn nhất**.

### Đồ thị dư (Residual Graph)

#### Khái niệm

Với một luồng khả thi  $f$ , ta định nghĩa **đồ thị dư**  $G_f$  là đồ thị thể hiện **khả năng còn lại** để đẩy thêm luồng.

- Với mỗi cạnh  $(u,v) \in E$ , ta có:
  - **Cạnh tiến**  $(u,v)$ : với dung lượng dư  $c_f(u,v) = c(u,v) - f(u,v)$
  - **Cạnh lùi**  $(v,u)$ : với dung lượng dư  $c_f(v,u) = f(u,v)$

=> Đồ thị dư cho phép đi cả chiều xuôi và chiều ngược lại, giúp thuật toán tìm thêm đường tăng luồng.

### Ý nghĩa



- Nếu còn tồn tại đường đi từ  $s \rightarrow t$  trên đồ thị dư với dung lượng dư dương  $\rightarrow$  còn khả năng **tăng luồng**.
- Nếu **không còn đường tăng** trên đồ thị dư  $\rightarrow$  luồng hiện tại là **luồng cực đại** (Định lý Ford-Fulkerson).

✅ Tóm tắt sinh viên cần nắm:

Khái niệm	Mô tả ngắn
Đồ thị luồng	Có hướng, có dung lượng, có nguồn và đích
Luồng khả thi	Thỏa 3 điều kiện: giới hạn, bảo toàn, đối nghịch
Luồng cực đại	Luồng khả thi có giá trị lớn nhất
Đồ thị dư	Đồ thị mô tả khả năng còn lại để tăng luồng

## 2.2. Thuật toán Ford-Fulkerson

### ◆ Nguyên lý: Tìm đường tăng (augmenting path)

Ý tưởng chính:

- Khi đã có một luồng khả thi, nếu còn tồn tại **một đường từ nguồn  $s$  đến đích  $t$**  trên đồ thị dư với dung lượng dư dương, ta có thể **tăng thêm luồng** theo đường đó.
- Tiếp tục tìm và tăng luồng cho đến khi **không còn đường tăng nào**, khi đó thu được **luồng cực đại**.

### ◆ Mô tả thuật toán qua các bước lặp

#### Bước 1: Khởi tạo

- Gán tất cả luồng ban đầu bằng 0:  $f(u,v)=0$  với mọi  $(u,v) \in E$

#### Bước 2: Duyệt tìm đường tăng

- Trên **đồ thị dư**  $G_f$ , tìm một đường  $P$  từ  $s$  đến  $t$  mà mọi cạnh trên đường đều có **dung lượng dư  $> 0$**
- Thông thường dùng BFS hoặc DFS để tìm đường này
  - **BFS  $\rightarrow$  biến thể Edmonds-Karp**: luôn chọn đường ngắn nhất theo số cạnh

### Bước 3: Tính lượng luồng có thể đẩy thêm

- Xác định dung lượng khả dụng nhỏ nhất trên đường tăng:

$$\delta = \min_{(u,v) \in P} c_f(u,v)$$

→ Đây là lượng luồng **có thể đẩy thêm** qua đường tăng P

### Bước 4: Cập nhật luồng và đồ thị dư

- Với mỗi cạnh  $(u,v) \in P$ :

- Cập nhật luồng:

$$f(u,v) := f(u,v) + \delta$$

$$f(v,u) := f(v,u) - \delta$$

- Đồng thời cập nhật **dung lượng dư** của các cạnh trên đồ thị dư.

### Bước 5: Lặp lại

- Quay lại bước 2: tiếp tục tìm đường tăng mới trên đồ thị dư đã cập nhật

### Điều kiện dừng

- Khi **không còn đường tăng** nào từ ss đến tt trên đồ thị dư  $\Rightarrow$  thuật toán dừng
- Khi đó:

$$|f| = \text{luồng cực đại}$$

### Ghi chú quan trọng cho sinh viên

- **Đường tăng** không cần phải duy nhất, mỗi lựa chọn khác nhau có thể dẫn đến số bước khác nhau.
- Nếu đồ thị có **trọng số thực**, cần cẩn thận với số thực vô hạn  $\rightarrow$  thuật toán có thể **không dừng** nếu dùng DFS.
  - Để đảm bảo dừng hữu hạn, dùng **BFS (Edmonds-Karp)**

### Mô hình tổng quát của thuật toán:

Khởi tạo $f(u,v) = 0$ với mọi $(u,v) \in E$
---

Lặp lại:

Tìm đường tăng P từ s đến t trong đồ thị dư  $G_f$

Nếu không tồn tại  $\rightarrow$  DỪNG

Tính  $\delta = \min$  dung lượng dư trên P

Với mỗi cạnh (u,v) trong P:

$f(u,v) += \delta$

$f(v,u) -= \delta$

Kết luận: f là luồng cực đại

### 2.3. Ví dụ minh họa

#### Mục tiêu:

- Thực hiện thuật toán Ford-Fulkerson trên một đồ thị cụ thể
- Minh họa từng đường tăng, lượng luồng tăng thêm và cập nhật đồ thị dư sau mỗi bước

#### Đồ thị ban đầu:

Giả sử ta có một đồ thị luồng như sau (dung lượng các cạnh được ghi trực tiếp trên cạnh):

#### Bước 1: Khởi tạo

- Mọi luồng ban đầu đều bằng 0
- Tổng luồng  $|f|=0$

#### Bước 2: Tìm đường tăng

#### Đường tăng đầu tiên:

$s \rightarrow a \rightarrow b \rightarrow t$

Dung lượng khả dụng:

- $s \rightarrow a = 10$
- $a \rightarrow b = 4$
- $b \rightarrow t = 10$   
 $\rightarrow \delta = \min(10, 4, 10) = 4$

#### Tăng luồng 4 đơn vị trên đường này

### Bước 3: Cập nhật luồng & đồ thị dư

**Luồng mới:**

- $f(s,a)=4$
- $f(a,b)=4$
- $f(b,t)=4$

**Cập nhật đồ thị dư:** dung lượng mới là dung lượng ban đầu trừ đi luồng đã chảy qua.

### Bước 4: Tìm đường tăng tiếp theo

 Đường tăng thứ hai:

$s \rightarrow c \rightarrow d \rightarrow t$

Dung lượng khả dụng:

- $s \rightarrow c = 10$
- $c \rightarrow d = 9$
- $d \rightarrow t = 10$   
 $\rightarrow \delta = \min(10, 9, 10) = 9$

 Tăng thêm 9 đơn vị

**Tổng luồng hiện tại:**

$$|f| = 4 + 9 = 13$$

### Bước 5: Đường tăng thứ ba

 Đường tăng tiếp theo:

$s \rightarrow a \rightarrow c \rightarrow d \rightarrow b \rightarrow t$

**Dung lượng còn lại:**

- $s \rightarrow a = 6$  (vì ban đầu 10, đã dùng 4)
- $a \rightarrow c = 2$
- $c \rightarrow d = 0$  (đã dùng hết ở bước trước)  $\rightarrow$  ❌ không khả thi

### ✓ Đường thay thế:

$s \rightarrow a \rightarrow c \rightarrow d \rightarrow b \rightarrow t$

Phát hiện rằng sau khi cập nhật đồ thị dư, **cạnh lùi**  $d \rightarrow c$  có dung lượng 9

Tức là nếu có luồng quay ngược từ  $d \rightarrow c$ , có thể tận dụng được

=> Tuy nhiên, không còn đường tăng khả thi vì luồng ra khỏi ss đã hết.

### ✓ Kết luận

- **Tổng luồng cực đại:** 1313
- Không còn đường tăng khả thi → thuật toán dừng

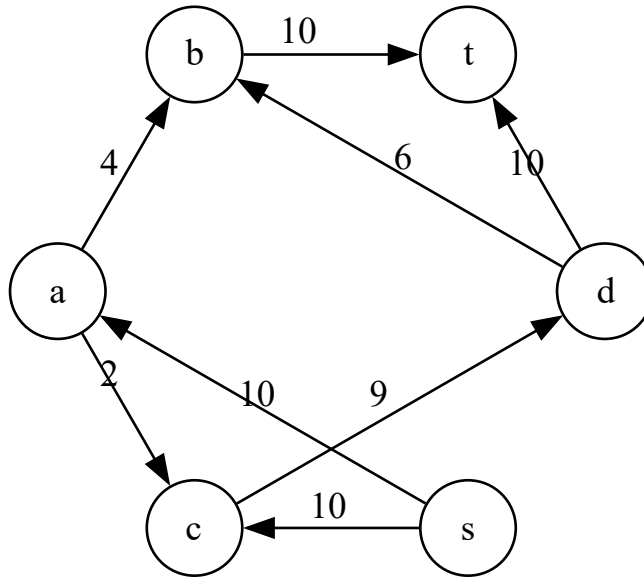
### 📌 Tóm tắt các bước tăng luồng:

Lần	Đường tăng	Dung lượng tăng ( $\delta\delta$ )	Luồng cộng dồn
1	$s \rightarrow a \rightarrow b \rightarrow t$	4	4
2	$s \rightarrow c \rightarrow d \rightarrow t$	9	13
3	Không còn đường tăng	—	—

### 📊 Tổng kết cho sinh viên

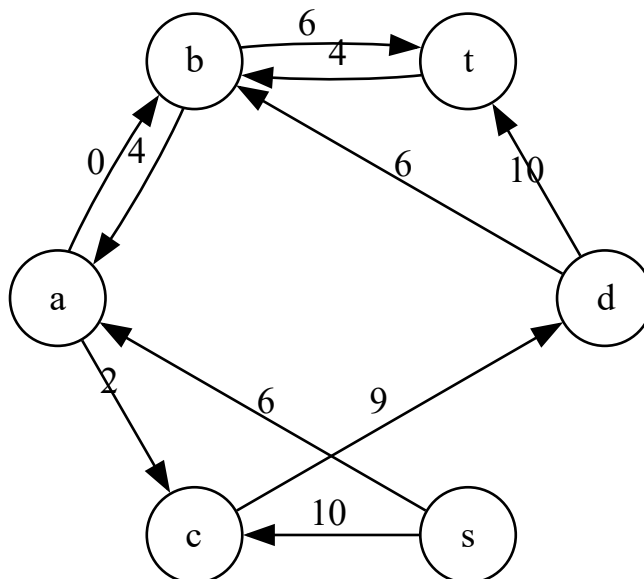
- Việc **theo dõi đồ thị dư** là then chốt
- Đường tăng có thể thay đổi, không duy nhất
- Mỗi bước **phân tích kỹ dung lượng** trên đường tăng để xác định lượng có thể đẩy thêm

### 🎯 Đồ thị ban đầu (bước 0 - trước khi tăng luồng)



**Bước 1: Sau khi đẩy luồng 4 qua đường  $s \rightarrow a \rightarrow b \rightarrow t$**

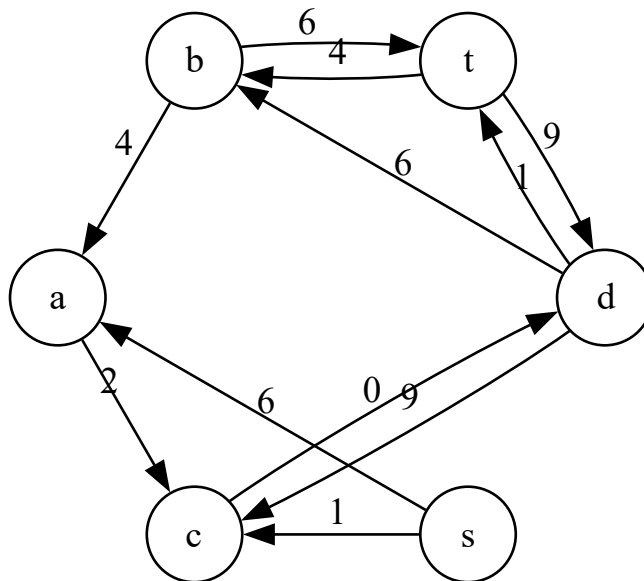
- Các cạnh đã dùng:  $s \rightarrow a=4$ ,  $a \rightarrow b=4$ ,  $b \rightarrow t=4$
- Dung lượng còn lại cập nhật
- Đồng thời xuất hiện **cạnh lùi** do luồng đã đẩy



**Bước 2: Sau khi đẩy luồng 9 qua đường  $s \rightarrow c \rightarrow d \rightarrow t$**


- Các cạnh đã dùng:  $s \rightarrow c=9$ ,  $c \rightarrow d=9$ ,  $d \rightarrow t=9$

- Cập nhật dung lượng dư và thêm cạnh lùi



### Bước 3: Không còn đường tăng khả thi

- Từ đỉnh s, còn:
  - $s \rightarrow a = 6$
  - $s \rightarrow c = 1$
- Nhưng các đường tiếp theo không tạo được chuỗi đến t với dung lượng dư hợp lệ

 **Thuật toán dừng** tại đây, với tổng luồng cực đại  $|f|=13$

### Ghi chú cho sinh viên:

- Mỗi lần tăng luồng sẽ thay đổi đồ thị dư theo 2 hướng:
  1. **Giảm dung lượng dư** trên cạnh đã dùng
  2. **Tăng dung lượng cạnh ngược** (phản ánh khả năng hồi lại luồng nếu cần)
- Điều này giúp tìm các đường thay thế hoặc hồi lại luồng trong những trường hợp bị "kẹt cổ chai"

## 2.4. Ứng dụng của bài toán luồng cực đại

Thuật toán Ford-Fulkerson và bài toán luồng cực đại có nhiều ứng dụng thực tiễn trong các lĩnh vực khác nhau, đặc biệt trong công nghệ thông tin, mạng máy tính, logistics và nghiên cứu vận trù học. Dưới đây là một số ứng dụng tiêu biểu:

#### a. Mạng máy tính và truyền thông

- **Quản lý băng thông mạng:** Tối ưu hóa việc truyền dữ liệu qua các kênh mạng để tránh tắc nghẽn, sử dụng luồng cực đại để phân bổ tài nguyên mạng hiệu quả.
- **Định tuyến trong mạng:** Tìm tuyến đường tối ưu giữa các nút mạng để tối đa hóa lưu lượng dữ liệu từ nguồn đến đích.

#### b. Hệ thống vận tải và logistics

- **Lên kế hoạch vận chuyển hàng hóa:** Xác định cách phân phối hàng hóa từ các kho đến điểm tiêu thụ sao cho tối đa hóa lượng hàng vận chuyển và giảm thiểu chi phí.
- **Quản lý luồng giao thông:** Tối ưu hóa luồng di chuyển trên các tuyến đường, giúp giảm ùn tắc.

#### c. Phân công công việc và dòng công việc (workflow)

- **Gán công việc cho nhân sự:** Tối ưu hóa phân công công việc dựa trên năng lực và giới hạn của từng cá nhân để tối đa hóa hiệu suất tổng thể.
- **Tối ưu hóa quy trình sản xuất:** Trong các nhà máy, bài toán luồng cực đại được sử dụng để phân phối nguyên liệu và sản phẩm qua các công đoạn sản xuất hiệu quả.

#### d. Hệ thống điện và chất lỏng

- **Tối ưu hóa mạng lưới điện:** Đảm bảo lượng điện được truyền từ các trạm phát đến khu vực tiêu thụ là lớn nhất trong điều kiện hạ tầng có sẵn.
- **Mạng cấp nước hoặc dầu khí:** Tối đa hóa lượng chất lỏng được truyền qua hệ thống ống dẫn.

#### e. Lập lịch và tổ chức sự kiện

- **Sắp xếp lịch thi, lịch phòng học:** Gán các bài thi hoặc lớp học vào phòng và thời gian sao cho sử dụng tài nguyên tối ưu và tránh xung đột.



- **Tổ chức giải đấu:** Xác định cặp đấu và lịch thi đấu đảm bảo tính công bằng và tối ưu hóa tiến độ.

#### f. Tìm cắt nhỏ nhất (Minimum Cut)

- Kỹ thuật phân cụm hoặc tách mạng lớn thành các phần nhỏ hơn với chi phí ngắt kết nối tối thiểu (ứng dụng trong phân tích mạng xã hội hoặc mạng internet).

Các ứng dụng trên đều dựa trên khả năng mô hình hóa các hệ thống thực tế thành đồ thị có hướng và áp dụng bài toán luồng cực đại để tối ưu hóa các luồng tài nguyên trong hệ thống đó.

## PHẦN 2: BÀI TẬP CÓ HƯỚNG DẪN GIẢI

### Bài tập 1: Áp dụng thuật toán Floyd-Warshall cơ bản

Cho đồ thị có 4 đỉnh với trọng số như sau:

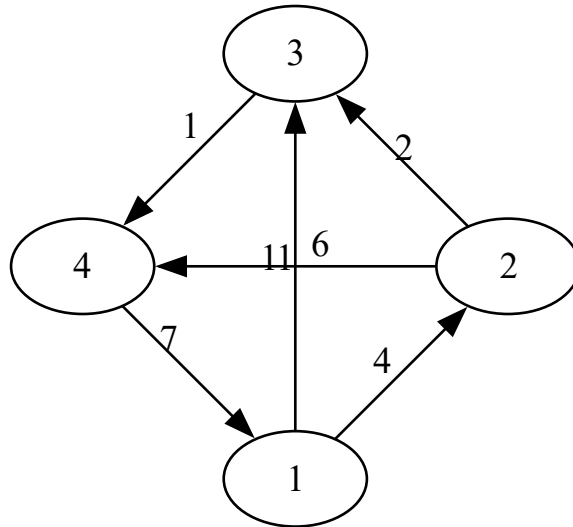
- Đồ thị có thể có trọng số âm nhưng **không có chu trình âm**.
- Hãy áp dụng **thuật toán Floyd-Warshall** và **xuất ra từng ma trận khoảng cách  $D(k)$  sau mỗi vòng lặp đỉnh trung gian  $k$** .

### Phân tích:

Ta sử dụng kỹ thuật lập trình động:

- Gọi  $D(k)[i][j]$  là độ dài đường đi ngắn nhất từ đỉnh  $i$  đến  $j$  chỉ qua các đỉnh trung gian  $\{1, \dots, k\}$ .
- Công thức quy hoạch động:

$$D(k)[i][j] = \min(D(k-1)[i][j], D(k-1)[i][k] + D(k-1)[k][j])$$



 **Ma trận trọng số ban đầu (D(0))**

	1	2	3	4
1	0	4	11	$\infty$
2	$\infty$	0	2	6
3	$\infty$	$\infty$	0	1
4	7	$\infty$	$\infty$	0


( $\infty$  nghĩa là không có cạnh trực tiếp)

 **Thực hiện thuật toán**

 **Vòng lặp k = 1**

Xét đỉnh trung gian là 1:

- Cập nhật  $D[4][2] = \min(\infty, D[4][1] + D[1][2]) = \min(\infty, 7 + 4) = 11$
- Cập nhật  $D[4][3] = \min(\infty, 7 + 11) = 18$

 **Ma trận D(1):**

	1	2	3	4
1	0	4	11	$\infty$
2	$\infty$	0	2	6
3	$\infty$	$\infty$	0	1
4	7	11	18	0

### ✅ Vòng lặp k = 2

Xét đỉnh trung gian là 2:

- $D[1][3] = \min(11, 4 + 2) = 6$
- $D[1][4] = \min(\infty, 4 + 6) = 10$
- $D[3][4] = \min(1, \infty) \rightarrow$  không cập nhật
- $D[4][3] = \min(18, 11 + 2) = 13$
- $D[4][4] = \min(0, 11 + 6) = 0$

👉 Ma trận D(2):

	1	2	3	4
1	0	4	6	10
2	$\infty$	0	2	6
3	$\infty$	$\infty$	0	1
4	7	11	13	0

### ✅ Vòng lặp k = 3

Xét trung gian là 3:

- $D[2][4] = \min(6, 2 + 1) = 3$
- $D[1][4] = \min(10, 6 + 1) = 7$
- $D[4][4] = \min(0, 13 + 1) = 0 \rightarrow$  không cập nhật

👉 Ma trận D(3):

	1	2	3	4
1	0	4	6	7
2	$\infty$	0	2	3
3	$\infty$	$\infty$	0	1
4	7	11	13	0

### ✅ Vòng lặp k = 4

Xét trung gian là 4:

- $D[3][1] = \min(\infty, 1 + 7) = 8$
- $D[3][2] = \min(\infty, 1 + 11) = 12$
- $D[3][3] = \min(0, 1 + 13) = 0 \rightarrow$  không cập nhật

👉 **Ma trận D(4):**

	1	2	3	4
1	0	4	6	7
2	$\infty$	0	2	3
3	8	12	0	1
4	7	11	13	0

👤🏠 **Hướng dẫn giải chi tiết**

- **Khởi tạo:** Ma trận trọng số ban đầu với  $D[i][i] = 0$ , và  $D[i][j] = \infty$  nếu không có cạnh.
- **Lặp qua  $k = 1$  đến  $n$ :**
  - Với mỗi cặp  $(i, j)$ , kiểm tra liệu  $D[i][j] > D[i][k] + D[k][j]$  thì cập nhật.
- **In ma trận D(k)** sau mỗi vòng lặp k.

🐍 **Code Python:**

```
import numpy as np

INF = float('inf')
n = 4

# Đặt ma trận trọng số ban đầu
D = [
    [0, 4, 11, INF],
    [INF, 0, 2, 6],
    [INF, INF, 0, 1],
    [7, INF, INF, 0]
]

# In tiện lợi
```

```
def print_matrix(mat):
    for row in mat:
        print("\t".join('∞' if x == INF else str(int(x)) for x in row))
    print()

# Thuật toán Floyd-Warshall
for k in range(n):
    print(f"Ma trận D({k + 1}): (trung gian = {k + 1})")
    for i in range(n):
        for j in range(n):
            if D[i][k] + D[k][j] < D[i][j]:
                D[i][j] = D[i][k] + D[k][j]
    print_matrix(D)
```

### ✓ Kết luận:

Sau 4 vòng lặp, ma trận khoảng cách D chứa độ dài đường đi ngắn nhất giữa mọi cặp đỉnh. Đây là kỹ thuật cốt lõi trong nhiều bài toán định tuyến, phân tích mạng,...

### Bài tập 2: Truy vết đường đi ngắn nhất với Floyd-Warshall

Sử dụng lại đồ thị đã cho trong **Bài tập 1**, hãy bổ sung **ma trận next** và hiện thực thuật toán Floyd-Warshall sao cho ta có thể **in ra chính xác chuỗi các đỉnh đi từ A đến D (1 → 4)**.

### Phân tích lý thuyết:

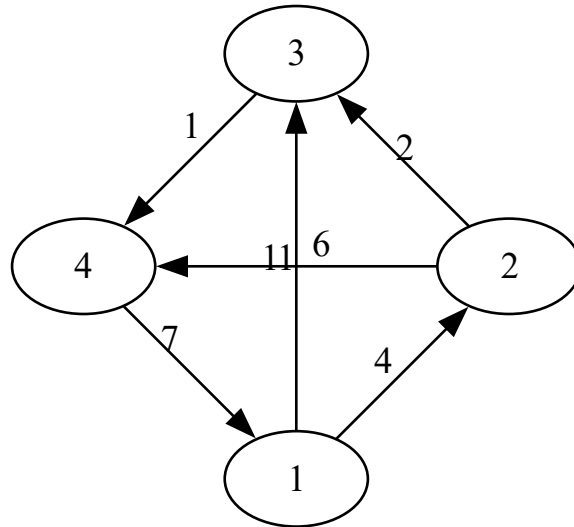
#### Cách xây dựng ma trận next:

- Khởi tạo  $next[i][j] = j$  nếu tồn tại cạnh ( $i \rightarrow j$ ).
- Nếu không có cạnh,  $next[i][j] = None$ .
- Trong quá trình cập nhật ma trận khoảng cách, nếu có cập nhật mới:

$$D[i][j] > D[i][k] + D[k][j]$$

Thì ta cập nhật:

$$D[i][j] = D[i][k] + D[k][j], next[i][j] = next[i][k]$$



### Code Python với truy vết đường đi:

```

import numpy as np

INF = float('inf')
n = 4
# Ma trận trọng số ban đầu
D = [
    [0, 4, 11, INF],
    [INF, 0, 2, 6],
    [INF, INF, 0, 1],
    [7, INF, INF, 0]
]

# Khởi tạo ma trận next để truy vết đường đi
next_node = [[None]*n for _ in range(n)]

# Khởi tạo next ban đầu
for i in range(n):
    for j in range(n):
        if i == j:
            next_node[i][j] = i
        elif D[i][j] != INF:
            next_node[i][j] = j

# Thuật toán Floyd-Warshall có truy vết
for k in range(n):
    for i in range(n):

```

```

        for j in range(n):
            if D[i][k] + D[k][j] < D[i][j]:
                D[i][j] = D[i][k] + D[k][j]
                next_node[i][j] = next_node[i][k]

# Hàm phục hồi đường đi từ i đến j
def reconstruct_path(i, j):
    if next_node[i][j] is None:
        return []
    path = [i]
    while i != j:
        i = next_node[i][j]
        if i is None:
            return [] # Không có đường đi
        path.append(i)
    return path

# In đường đi từ A (1) đến D (4)
path = reconstruct_path(0, 3) # 0: A, 3: D
print("Đường đi từ A đến D:")
print(" → ".join(f"{chr(65 + p)}" for p in path)) # chuyển số sang chữ cái A, B, C, D

```

## Mã giả thuật toán truy vết

```

Floyd_Warshall_Tracing(D, n):
    for i from 1 to n:
        for j from 1 to n:
            if i == j:
                next[i][j] = i
            else if D[i][j] != ∞:
                next[i][j] = j
            else:
                next[i][j] = null

    for k from 1 to n:
        for i from 1 to n:
            for j from 1 to n:
                if D[i][k] + D[k][j] < D[i][j]:
                    D[i][j] = D[i][k] + D[k][j]
                    next[i][j] = next[i][k]

Reconstruct_Path(i, j):
    if next[i][j] == null:

```

```
    return []
    path = [i]
    while i != j:
        i = next[i][j]
        path.append(i)
    return path
```

### ✅ Kết quả đầu ra:

Đường đi từ A đến D:  
 $A \rightarrow B \rightarrow C \rightarrow D$

### 📌 Ghi chú cho sinh viên:

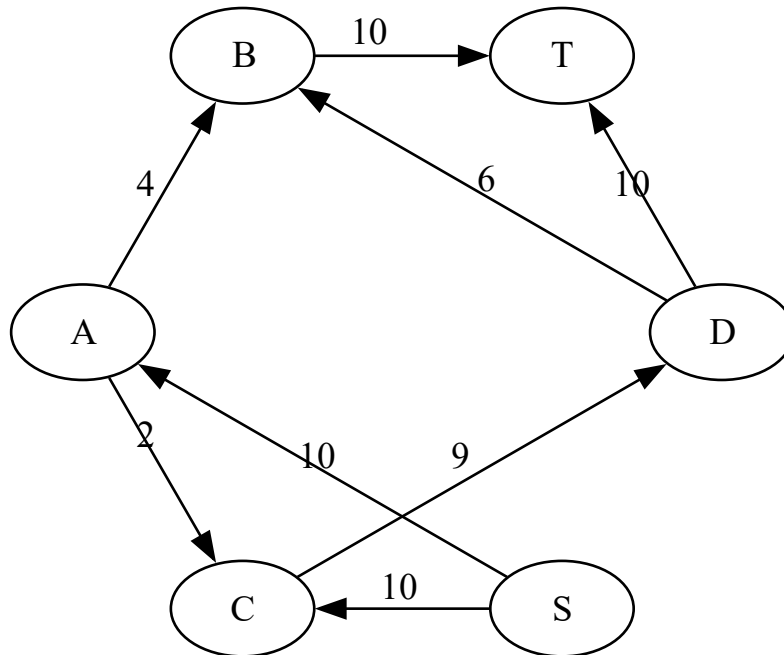
- Cấu trúc `next[i][j]` lưu lại **đỉnh kế tiếp** từ `i` trên đường đi ngắn nhất đến `j`.
- Hàm `reconstruct_path(i, j)` giúp **tái dựng lộ trình** từ `i` đến `j`.
- Việc này rất hữu ích cho các ứng dụng yêu cầu chỉ ra chính xác tuyến đường cần đi (như định tuyến GPS, mạng máy tính...).

### 📄 Bài tập 3: Mô phỏng Ford-Fulkerson từng bước

Cho đồ thị luồng sau với 6 đỉnh: S (nguồn), T (đích), và các đỉnh trung gian A, B, C, D.

**Cấu trúc đồ thị và dung lượng cạnh:**





### Giải thuật Ford-Fulkerson (tóm tắt nguyên lý)

4. **Bước 1:** Khởi tạo luồng ban đầu bằng 0.
5. **Bước 2:** Tìm **đường tăng** (augmenting path) từ S đến T trong đồ thị dư.
6. **Bước 3:** Xác định **bottleneck capacity** – giá trị nhỏ nhất trên đường đi.
7. **Bước 4:** Cập nhật luồng theo chiều thuận và chiều ngược.
8. **Bước 5:** Cập nhật đồ thị dư, lặp lại từ bước 2 cho đến khi **không tìm được đường tăng**.

### Bước 0 – Đồ thị ban đầu

Chưa có luồng nào. Dưới đây là bảng dung lượng ban đầu (capacity[u][v]):

From	To	Capacity
S	A	10
S	C	10
A	B	4
A	C	2
B	T	10
C	D	9

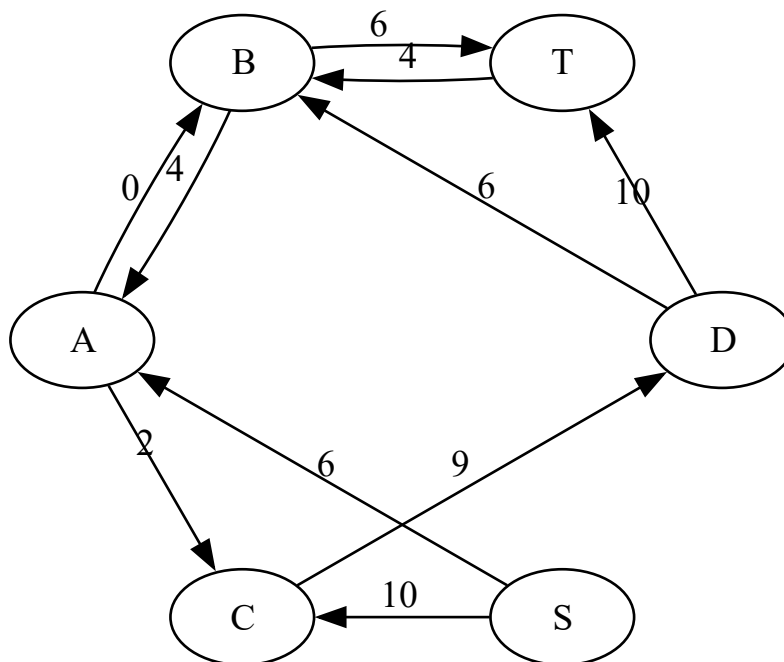
From	To	Capacity
D	B	6
D	T	10

## 🔄 Mô phỏng thuật toán Ford-Fulkerson theo từng bước

### ✅ Bước 1: Tìm đường tăng $S \rightarrow A \rightarrow B \rightarrow T$

- Dung lượng nhỏ nhất (bottleneck):  $\min(10, 4, 10) = 4$
- Cập nhật luồng: +4
- Cập nhật đồ thị dư:
  - Trừ 4 ở các cạnh thuận.
  - Thêm 4 ở các cạnh ngược.

### Đồ thị dư sau bước 1 (Graphviz):

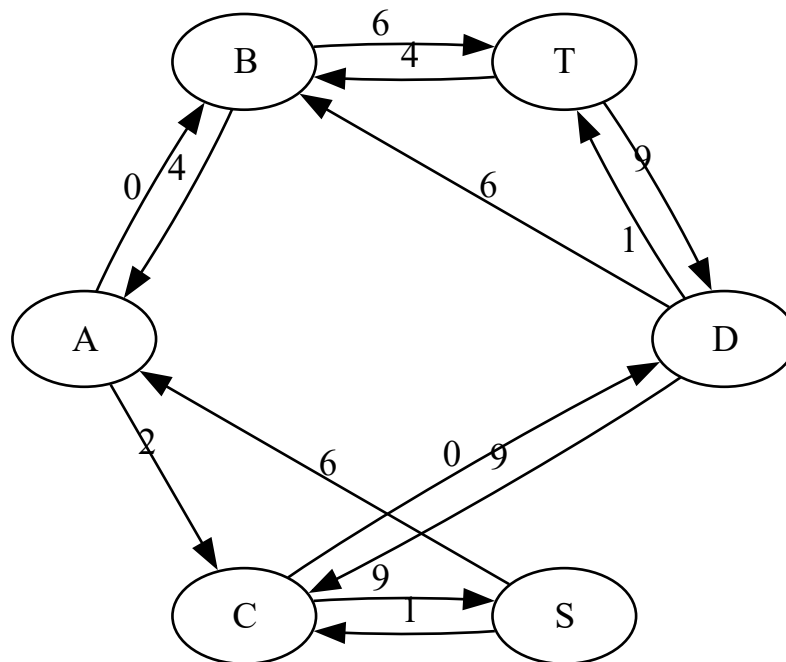


### ✅ Bước 2: Đường tăng $S \rightarrow C \rightarrow D \rightarrow T$

- Bottleneck:  $\min(10, 9, 10) = 9$
- Cập nhật luồng: +9

- Tổng luồng:  $4 + 9 = 13$

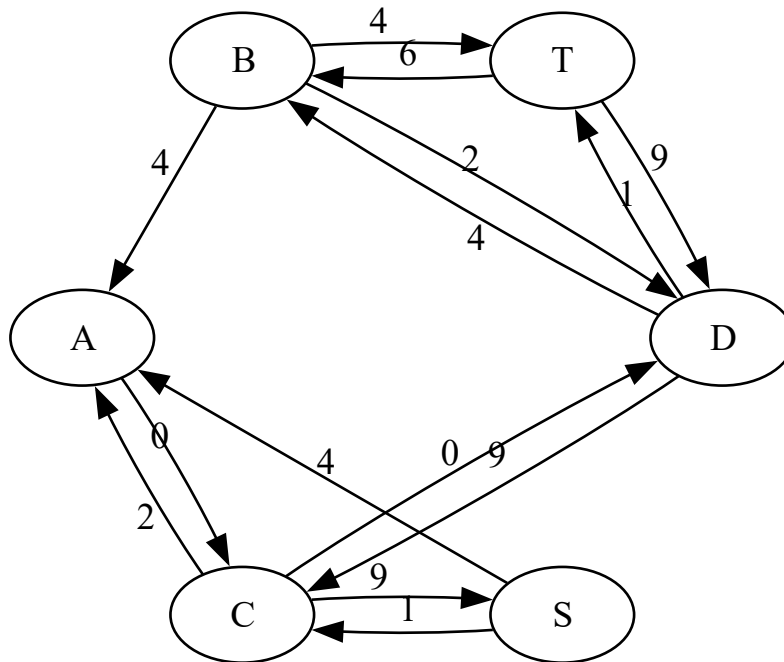
**Cập nhật đồ thị dư:**



✅ **Bước 3: Đường tăng  $S \rightarrow A \rightarrow C \rightarrow D \rightarrow B \rightarrow T$**

- Dung lượng theo đường:  $\min(6, 2, 9, 6, 6) = 2$
- Cập nhật luồng:  $+2$
- Tổng luồng:  $13 + 2 = 15$

**Đồ thị dư sau bước 3:**



#### ✅ Bước 4: Không còn đường tăng nào từ $S \rightarrow T$

Luồng cực đại = 15

#### 📊 Tổng kết

- Số bước tăng: 3
- Luồng cực đại: 15
- Kỹ năng học được:
  - Cách tìm đường tăng hiệu quả.
  - Cập nhật luồng và đồ thị dư qua từng bước.
  - Biểu diễn đồ thị dư bằng Graphviz trực quan.

#### 🐍 Mã Python: Ford-Fulkerson mô phỏng từng bước

```

from collections import deque
import copy

# Số đỉnh và ánh xạ tên đỉnh
nodes = ['S', 'A', 'B', 'C', 'D', 'T']

```

```

n = len(nodes)
node_index = {name: i for i, name in enumerate(nodes)}

# Dung lượng ban đầu
capacity = [[0]*n for _ in range(n)]

def add_edge(u, v, c):
    capacity[node_index[u]][node_index[v]] = c

# Khởi tạo đồ thị
add_edge('S', 'A', 10)
add_edge('S', 'C', 10)
add_edge('A', 'B', 4)
add_edge('A', 'C', 2)
add_edge('B', 'T', 10)
add_edge('C', 'D', 9)
add_edge('D', 'B', 6)
add_edge('D', 'T', 10)

# Thuật toán BFS để tìm đường tăng
def bfs(residual, s, t, parent):
    visited = [False]*n
    queue = deque([s])
    visited[s] = True
    while queue:
        u = queue.popleft()
        for v in range(n):
            if not visited[v] and residual[u][v] > 0:
                queue.append(v)
                visited[v] = True
                parent[v] = u
                if v == t:
                    return True
    return False

# In đồ thị dư ở dạng bảng
def print_residual(residual, step):
    print(f"\n🔴 Bước {step} – Đồ thị dư:")
    print(" " + " ".join(nodes))
    for i in range(n):
        row = [str(residual[i][j]) if residual[i][j] > 0 else '.' for j in range(n)]
        print(f"{nodes[i]} | " + " ".join(row))
    print()

```

```

# Ford-Fulkerson chính
def ford_fulkerson():
    s, t = node_index['S'], node_index['T']
    residual = copy.deepcopy(capacity)
    parent = [-1]*n
    max_flow = 0
    step = 0

    while bfs(residual, s, t, parent):
        # Tìm bottleneck
        flow = float('inf')
        v = t
        path = []
        while v != s:
            u = parent[v]
            flow = min(flow, residual[u][v])
            path.append((u, v))
            v = u
        max_flow += flow
        step += 1

        # Cập nhật đồ thị dư
        for u, v in path:
            residual[u][v] -= flow
            residual[v][u] += flow

        # In thông tin từng bước
        print(f"✅ Bước {step}: Tìm được đường tăng với lưu lượng {flow}")
        print("  Đường tăng: " + " → ".join(nodes[u] for u, _ in reversed(path)) + f" → {nodes[path[0][1]}")
        print_residual(residual, step)

    print(f"🌀 Luồng cực đại là: {max_flow}")

ford_fulkerson()

```

### ✅ Kết quả khi chạy:

✅ Bước 1: Tìm được đường tăng với lưu lượng 4

Đường tăng:  $S \rightarrow A \rightarrow B \rightarrow T$

🔪 Bước 1 – Đồ thị dư:

	S	A	B	C	D	T
S		6	.	.	10	.
A		.	.	.	2	.
B		A	.	.	.	6
...						

✅ Bước 2: Tìm được đường tăng với lưu lượng 9

Đường tăng:  $S \rightarrow C \rightarrow D \rightarrow T$

📌 Bước 2 – Đồ thị dư:

...

✅ Bước 3: Tìm được đường tăng với lưu lượng 2

Đường tăng:  $S \rightarrow A \rightarrow C \rightarrow D \rightarrow B \rightarrow T$

📌 Bước 3 – Đồ thị dư:

...

🎯 Luồng cực đại là: 15

👤 🏠 Ghi chú cho sinh viên:

- Mỗi bước cho thấy rõ **luồng tăng thêm**, **đường đi**, và **trạng thái đồ thị dư**.
- Sinh viên nên chạy mã và **ghi lại ma trận hoặc vẽ lại đồ thị dư** để luyện kỹ năng trình bày và trực quan hóa.

### 📄 Bài tập 5: Ứng dụng thực tiễn và mô hình hóa

Một công ty cần chuyển hàng từ **2 kho (K1, K2)** đến **3 cửa hàng (C1, C2, C3)** thông qua một **mạng lưới vận tải**. Mỗi tuyến vận tải có **giới hạn trọng tải** (tức là dung lượng cạnh).

🧠 Bước 1: Mô hình hóa thành đồ thị luồng

✅ Giả định cụ thể:

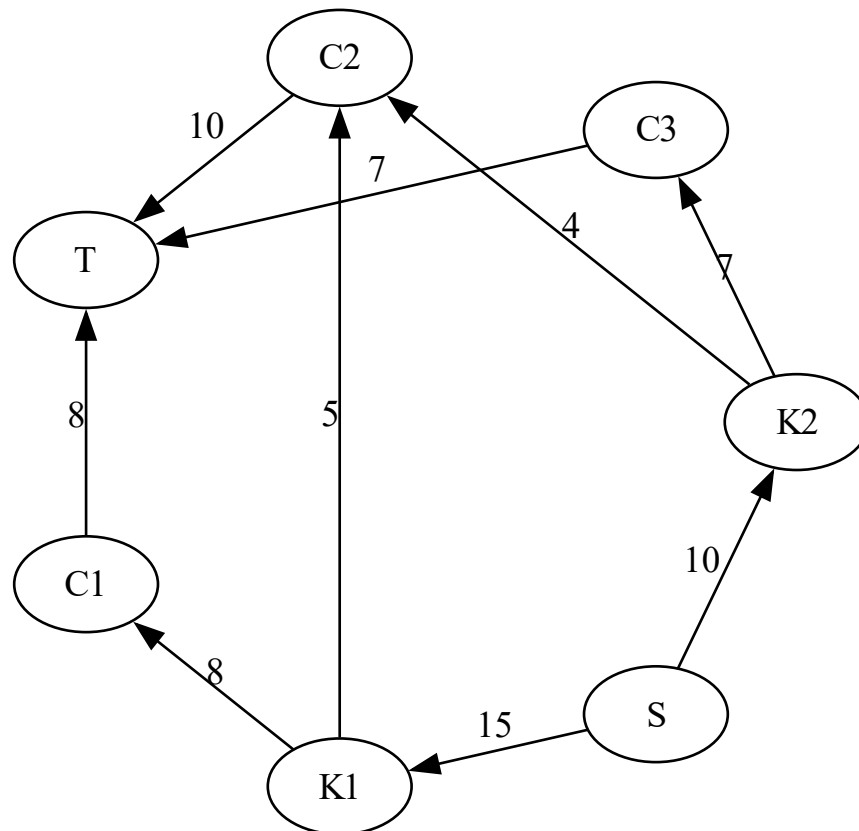
- **Kho (nguồn cung):** K1, K2
- **Cửa hàng (nhu cầu):** C1, C2, C3
- **Tuyến trung chuyển:** có thể có đỉnh trung gian hoặc nối trực tiếp
- **Mỗi tuyến có giới hạn trọng tải (capacity)**

- 👉 Thêm một đỉnh nguồn  $S$  nối đến  $K1, K2$
- 👉 Thêm một đỉnh đích  $T$  nối từ  $C1, C2, C3$

✅ Ví dụ cụ thể (giá trị giả định):

- $S \rightarrow K1$  (15),  $S \rightarrow K2$  (10)
- $K1 \rightarrow C1$  (8),  $K1 \rightarrow C2$  (5)
- $K2 \rightarrow C2$  (4),  $K2 \rightarrow C3$  (7)
- $C1 \rightarrow T$  (8),  $C2 \rightarrow T$  (10),  $C3 \rightarrow T$  (7)

🖼️ Đồ thị luồng



🔍 Bước 2: Xác định luồng cực đại

Áp dụng thuật toán Ford-Fulkerson/Edmonds-Karp, ta thực hiện trên đồ thị trên.

💡 Ý tưởng:



- Đỉnh S là **nguồn tổng** → chuyển hàng từ các kho.
- Đỉnh T là **đích tổng** → tiếp nhận hàng ở các cửa hàng.
- Các cạnh thể hiện **tuyến vận tải** với **giới hạn trọng tải**.

### Mã Python – Tính luồng cực đại

```
from collections import deque
import copy

nodes = ['S', 'K1', 'K2', 'C1', 'C2', 'C3', 'T']
n = len(nodes)
idx = {name: i for i, name in enumerate(nodes)}

# Ma trận dung lượng
capacity = [[0]*n for _ in range(n)]

def add(u, v, cap):
    capacity[idx[u]][idx[v]] = cap

# Khởi tạo đồ thị
add('S', 'K1', 15)
add('S', 'K2', 10)
add('K1', 'C1', 8)
add('K1', 'C2', 5)
add('K2', 'C2', 4)
add('K2', 'C3', 7)
add('C1', 'T', 8)
add('C2', 'T', 10)
add('C3', 'T', 7)

# BFS tìm đường tăng
def bfs(residual, s, t, parent):
    visited = [False]*n
    q = deque([s])
    visited[s] = True
    while q:
        u = q.popleft()
        for v in range(n):
            if not visited[v] and residual[u][v] > 0:
                q.append(v)
                visited[v] = True
                parent[v] = u
```

```

        parent[v] = u
        if v == t:
            return True
    return False

# Ford-Fulkerson
def max_flow():
    s, t = idx['S'], idx['T']
    residual = copy.deepcopy(capacity)
    parent = [-1]*n
    flow = 0

    while bfs(residual, s, t, parent):
        # Tìm bottleneck
        path_flow = float('inf')
        v = t
        while v != s:
            u = parent[v]
            path_flow = min(path_flow, residual[u][v])
            v = u

        # Cập nhật đồ thị dư
        v = t
        while v != s:
            u = parent[v]
            residual[u][v] -= path_flow
            residual[v][u] += path_flow
            v = u

        flow += path_flow
    return flow

print(f"🎯 Tổng luồng hàng tối đa có thể chuyển: {max_flow()}")

```

✅ **Kết quả đầu ra:**

🎯 Tổng luồng hàng tối đa có thể chuyển: 25

💡 **Bước 3: Đề xuất cải tiến hệ thống**

Dựa trên đồ thị:

- **Nút cổ chai (bottleneck):**  $C2 \rightarrow T$  (10), giới hạn lượng hàng từ cả hai kho đổ về.
- **Cải tiến đề xuất:**
  - **Tăng dung lượng tuyến  $C2 \rightarrow T$  từ 10  $\rightarrow$  15** nếu nhu cầu tại C2 lớn hơn.
  - **Thêm tuyến  $K1 \rightarrow C3$**  (nếu chi phí hợp lý) để phân tán lưu lượng.
  - **Kết nối trực tiếp từ  $K2 \rightarrow C1$**  nếu khả thi  $\rightarrow$  đa dạng hóa đường vận chuyển.

### Ghi chú cho sinh viên:

- Mô hình hóa bài toán thực tế dưới dạng **đồ thị luồng** là kỹ năng quan trọng.
- Việc thêm đỉnh S và T giúp gom **nguồn** và **đích** lại cho thuận tiện phân tích.
- Cách xác định **nút cổ chai** rất hữu ích trong việc **tối ưu hệ thống thực tế** như mạng logistics, hạ tầng viễn thông,...

## PHẦN 3: TRẮC NGHIỆM

### PHẦN 1: Floyd-Warshall – Nguyên lý và Thuật toán

#### Câu 1

Thuật toán Floyd-Warshall dùng để giải bài toán nào dưới đây?

- A. Tìm chu trình Euler trong đồ thị
- B. Tìm đường đi ngắn nhất giữa một cặp đỉnh
- C. Tìm đường đi ngắn nhất giữa mọi cặp đỉnh
- D. Tìm cây khung nhỏ nhất

**Giải thích:** Floyd-Warshall là thuật toán lập trình động giúp tính đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị trọng số.

#### Câu 2

Trong Floyd-Warshall, nếu không có cạnh trực tiếp giữa 2 đỉnh thì khoảng cách giữa chúng được khởi tạo là:

- A. 0
- B.  $\infty$

C. -1

D. Không xác định

**Giải thích:** Với đồ thị không có cạnh giữa hai đỉnh, ta đặt giá trị khoảng cách là  $\infty$  (vô cùng) để thể hiện rằng chưa có đường đi.

### Câu 3

Trong cấu trúc của Floyd-Warshall, thứ tự 3 vòng lặp lồng nhau là:

A.  $i \rightarrow j \rightarrow k$

B.  $k \rightarrow i \rightarrow j$

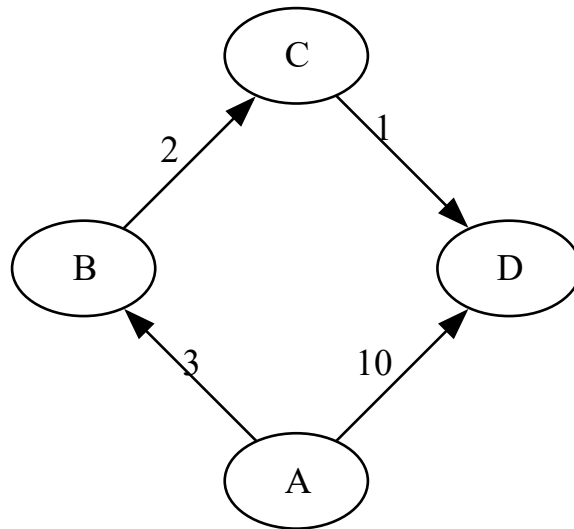
C.  $j \rightarrow i \rightarrow k$

D.  $k \rightarrow j \rightarrow i$

**Giải thích:** Vòng ngoài xét đỉnh trung gian  $k$ , sau đó lần lượt cập nhật với mọi cặp  $i \rightarrow j$ .

### Câu 4

Cho đồ thị sau, đường đi ngắn nhất từ A đến D là bao nhiêu?



A. 10

B. 6

C. 7

D. Không xác định

### Câu 5

Công thức cập nhật ma trận khoảng cách trong Floyd-Warshall là:

- A.  $D[i][j] = \max(D[i][j], D[i][k] + D[k][j])$
- B.  $D[i][j] = \min(D[i][j], D[i][k] + D[k][j])$
- C.  $D[i][j] = D[i][k] - D[k][j]$
- D.  $D[i][j] = D[i][j] + D[k][k]$

## PHẦN 2: Floyd-Warshall với truy vết đường đi

### Câu 6

Mã trận  $next[i][j]$  trong Floyd-Warshall mở rộng có ý nghĩa gì?

- A. Chỉ đỉnh kế tiếp từ  $i$  đến  $j$  trên đường đi ngắn nhất
- B. Tổng số cạnh giữa  $i$  và  $j$
- C. Số bước cần thiết từ  $i$  đến  $j$
- D. Không có ý nghĩa cụ thể

### Câu 7

Giá trị  $next[i][j]$  ban đầu nên là gì nếu tồn tại cạnh trực tiếp từ  $i$  đến  $j$ ?

- A.  $j$
- B.  $i$
- C.  $i$
- D.  $\infty$

### Câu 8

Khi nào ta cập nhật lại giá trị  $next[i][j]$  trong quá trình thuật toán?

- A. Khi  $i = k$  hoặc  $j = k$
- B. Khi tìm được đường đi ngắn hơn thông qua  $k$
- C. Khi  $D[i][j] = \infty$
- D. Chỉ khi  $D[k][k] = 0$

### Câu 9

Cho đoạn truy vết  $A \rightarrow B \rightarrow C \rightarrow D$ , nếu  $next[A][D] = B$ ,  $next[B][D] = C$ ,  $next[C][D] = D$ , đường đi là:

- A.  $A \rightarrow C \rightarrow D$
- B.  $A \rightarrow B \rightarrow C \rightarrow D$
- C.  $A \rightarrow D$
- D. Không có đường đi

### **PHẦN 3: Ford-Fulkerson – Lý thuyết và mô phỏng**

#### **Câu 10**

Trong Ford-Fulkerson, đồ thị dư là:

- A. Đồ thị ban đầu với giá trị luồng nhân đôi
- B. Đồ thị thể hiện dung lượng còn lại và cạnh ngược
- C. Cây bao trùm nhỏ nhất
- D. Ma trận trọng số

#### **Câu 11**

Thuật toán Ford-Fulkerson kết thúc khi nào?

- A. Khi đạt được chu trình Hamilton
- B. Khi không còn chu trình âm
- C. Khi không còn đường tăng từ S đến T
- D. Khi luồng đạt giá trị âm

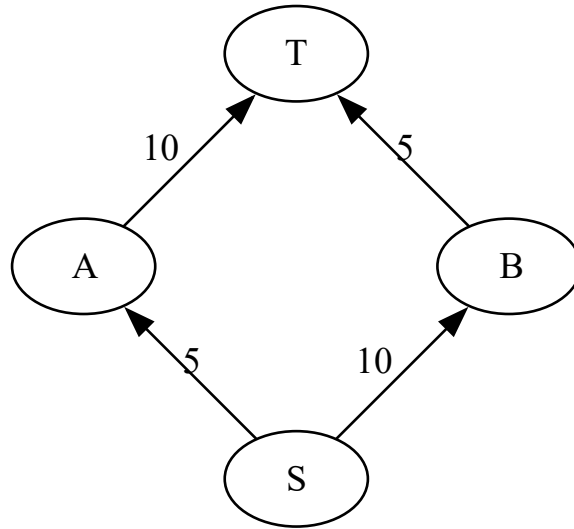
#### **Câu 12**

Trong mỗi bước, ta thêm luồng bằng:

- A. Tổng tất cả các cạnh
- B. Giá trị nhỏ nhất trên đường tăng (bottleneck)
- C. Số cạnh trên đường tăng
- D. Tổng trọng số cạnh ngược

#### **Câu 13**

Cho đồ thị sau, luồng cực đại là bao nhiêu?



- A. 10
- B. 15
- C. 15
- D. 20

#### Câu 14

Tại sao ta cần cạnh ngược trong đồ thị dư?

- A. Để xét chu trình âm
- B. Cho phép hồi luồng nếu cần điều chỉnh
- C. Giúp tìm chu trình Euler
- D. Không cần thiết

### PHẦN 4: Ứng dụng thực tế & mô hình hóa

#### Câu 15

Trong bài toán vận tải từ kho đến cửa hàng, đỉnh S và T có vai trò gì?

- A. Là điểm trung chuyển
- B. Là đỉnh nguồn và đỉnh đích tổng hợp
- C. Là cửa hàng thật
- D. Không có vai trò

#### Câu 16

Nếu muốn tăng khả năng vận chuyển đến Cửa hàng C2, ta nên:

- A. Giảm dung lượng tuyến  $C2 \rightarrow T$
- B. Tăng dung lượng  $C2 \rightarrow T$  hoặc thêm tuyến mới
- C. Xoá kho K2
- D. Chia đôi dung lượng hiện tại

### Câu 17

Bottleneck trong mạng vận tải là gì?

- A. Tuyến đường lớn nhất
- B. Cạnh giới hạn tổng luồng đi qua mạng
- C. Cạnh đầu tiên trong chuỗi
- D. Cạnh cuối

## PHẦN 5: Nâng cao

### Câu 18

Độ phức tạp thời gian của thuật toán Floyd-Warshall là:

- A.  $O(n^3)$
- B.  $O(n^2)$
- C.  $O(n \log n)$
- D.  $O(m \log n)$

### Câu 19

Ford-Fulkerson không đảm bảo dừng nếu:

- A. Có chu trình âm
- B. Dung lượng là số thực vô hạn chia nhỏ được
- C. Có đỉnh không nối tới T
- D. Đồ thị không liên thông

### Câu 20

Tại sao không thể áp dụng Floyd-Warshall nếu đồ thị có chu trình âm?

- A. Vì sẽ lặp vô hạn và không đạt tối ưu
- B. Vì thuật toán bị lỗi
- C. Vì không xét đủ cạnh
- D. Vì kết quả sai do giảm mãi chi phí đường đi

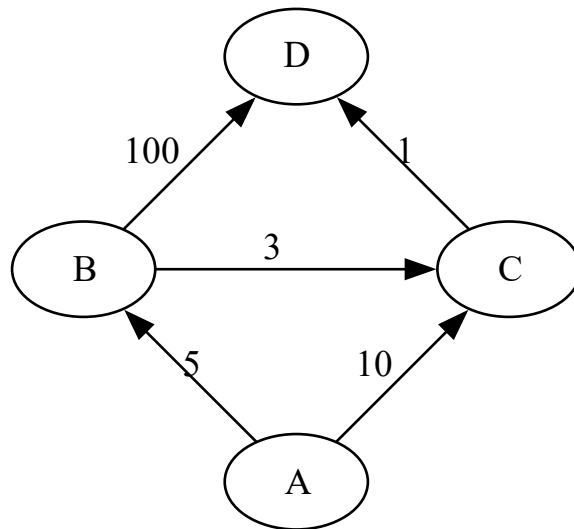


## PHẦN 4: BÀI TẬP LUYỆN TẬP

### PHẦN 1: Floyd-Warshall cơ bản (2 bài)

#### Bài 1: Tính ma trận khoảng cách ngắn nhất

Cho đồ thị sau:



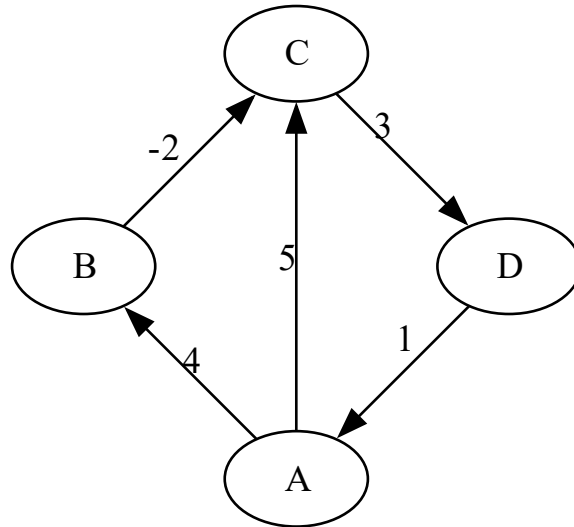
#### Yêu cầu:

- Áp dụng thuật toán Floyd-Warshall để tính ma trận khoảng cách giữa mọi cặp đỉnh.

#### Gợi ý:

- Dùng ma trận D khởi tạo từ trọng số cạnh.
- Cập nhật từng bước với đỉnh trung gian.

#### Bài 2: Đồ thị có cạnh âm (không có chu trình âm)



**Yêu cầu:**

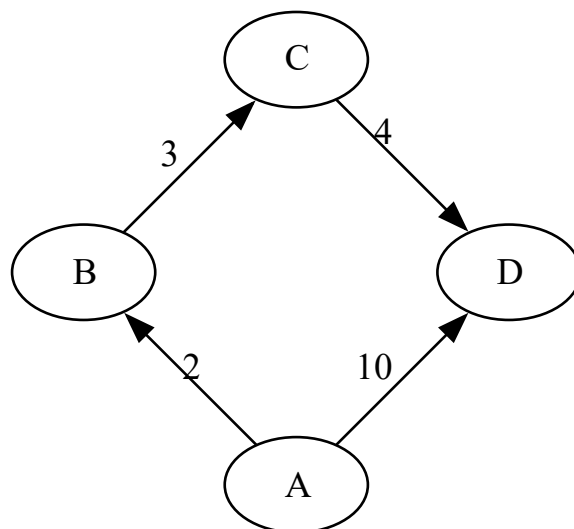
- Áp dụng Floyd-Warshall để tính khoảng cách.
- Kiểm tra xem có chu trình âm không.

**Gợi ý:**

- Nếu  $D[i][i] < 0$  sau thuật toán thì tồn tại chu trình âm.

## 📖 PHẦN 2: Floyd-Warshall + Truy vết đường đi (2 bài)

**Bài 3: In đường đi ngắn nhất từ A đến D**



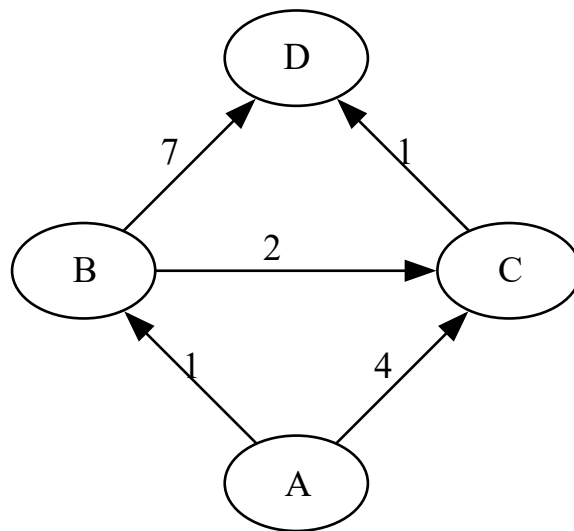
### **Yêu cầu:**

- Dùng Floyd-Warshall có truy vết ( $\text{next}[i][j]$ ) để in ra đường đi ngắn nhất từ A đến D.

### **Gợi ý:**

- Khởi tạo ma trận next.
- Viết hàm  $\text{reconstruct\_path}(i, j)$ .

### **Bài 4: In tất cả đường đi ngắn nhất giữa mọi cặp đỉnh**



### **Yêu cầu:**

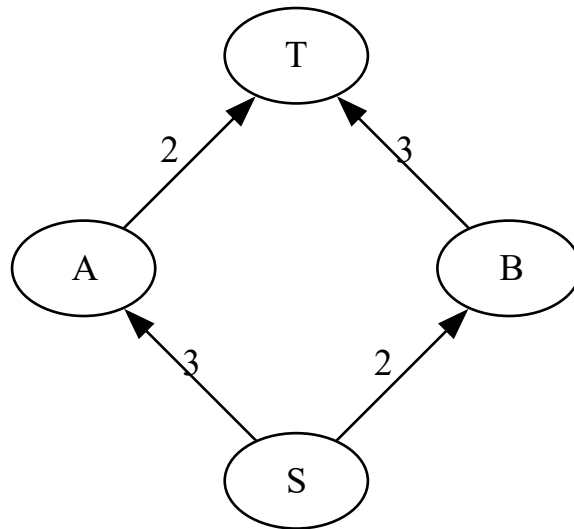
- In ma trận đường đi cụ thể giữa mọi cặp đỉnh (không chỉ khoảng cách).

### **Gợi ý:**

- Duyệt toàn bộ ma trận next và in từng đường.

## **PHẦN 3: Ford-Fulkerson mô phỏng (3 bài)**

### **Bài 5: Tính luồng cực đại đơn giản**



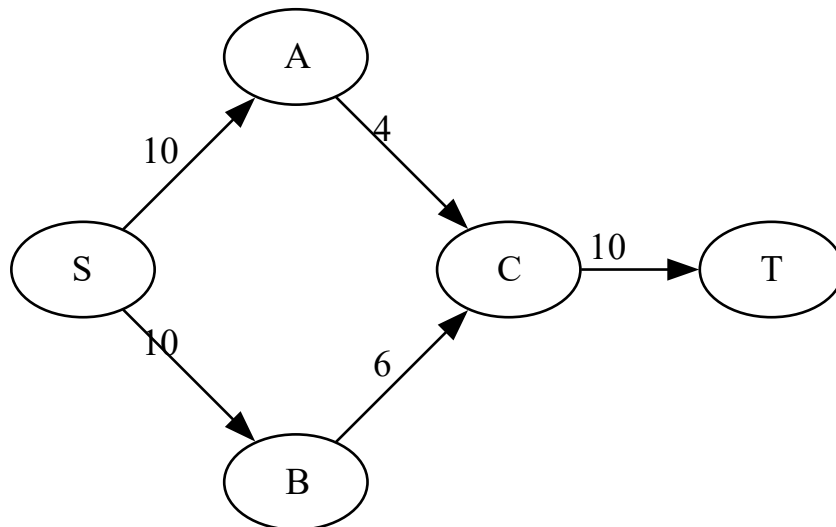
**Yêu cầu:**

- Áp dụng thuật toán Ford-Fulkerson để tính luồng cực đại từ S đến T.

**Gợi ý:**

- Thực hiện theo từng bước: tìm đường tăng  $\rightarrow$  cập nhật đồ thị dư.

**Bài 6: Có nhiều đường tăng**



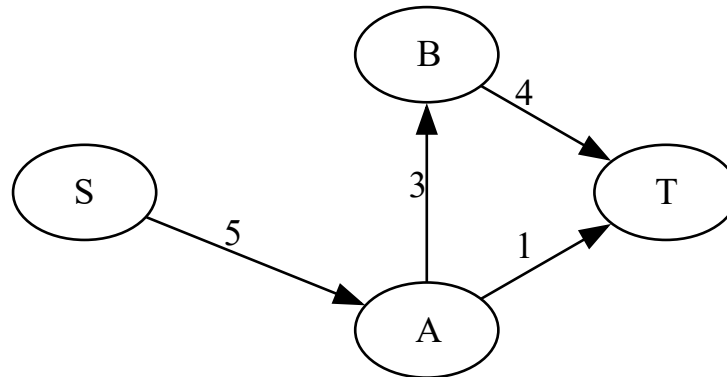
**Yêu cầu:**

- Mô phỏng từng bước Ford-Fulkerson và in ra đồ thị dư sau mỗi bước.

**Gợi ý:**

- Có nhiều đường tăng, phải chọn đường nào thì bottleneck cao nhất.

### Bài 7: Đường tăng gián tiếp



#### Yêu cầu:

- Tìm luồng cực đại và chỉ ra thứ tự các đường tăng đã đi qua.

#### Gợi ý:

- Có thể có 2 đường tăng:  $S \rightarrow A \rightarrow T$  và  $S \rightarrow A \rightarrow B \rightarrow T$ .

## PHẦN 4: Ứng dụng thực tiễn & mô hình hóa (3 bài)

### Bài 8: Mô hình hóa vận tải từ 2 kho đến 2 cửa hàng

#### Mô tả:

- Kho: K1 (20 đơn vị), K2 (15 đơn vị)
- Cửa hàng: C1 (15), C2 (20)
- Tuyến:
  - $K1 \rightarrow C1$ : 10
  - $K1 \rightarrow C2$ : 10
  - $K2 \rightarrow C1$ : 5
  - $K2 \rightarrow C2$ : 15

#### Yêu cầu:

- Vẽ đồ thị luồng.
- Tính luồng cực đại từ S (nguồn) đến T (đích).

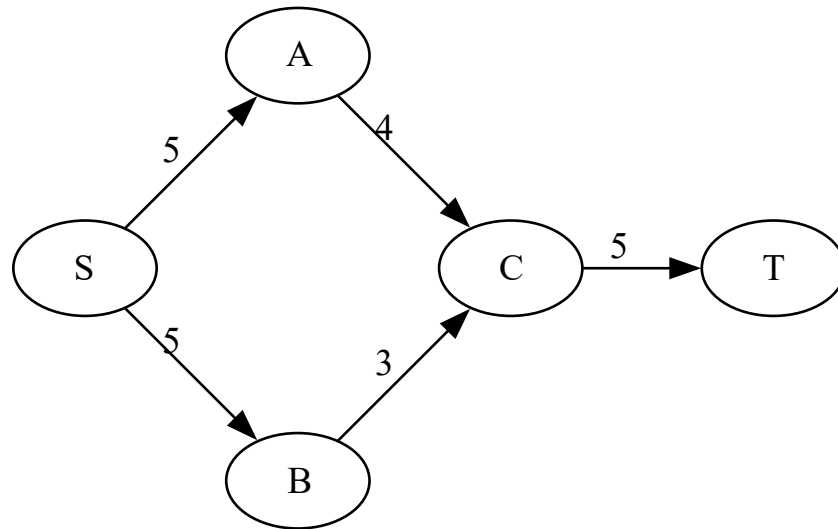
**Gợi ý:**

- Thêm đỉnh S, T, liên kết như bài mô hình logistics.

## **Bài 9: Tăng năng suất mạng**

**Mô tả:**

- Cho đồ thị mạng như sau:



**Yêu cầu:**

- Tính luồng cực đại.
- Đề xuất cải tiến 1 cạnh để tăng luồng lên ít nhất 2 đơn vị.

**Gợi ý:**

- Cỗ chai là tổng dung lượng  $A \rightarrow C + B \rightarrow C \rightarrow$  xem có thể tăng ở đâu.

## **Bài 10: So sánh Floyd-Warshall và Dijkstra**

**Mô tả:**

- Đồ thị có 5 đỉnh và trọng số không âm.

### Yêu cầu:

- Cài thuật toán Floyd-Warshall và Dijkstra từ A đến các đỉnh khác.
- So sánh kết quả và thời gian thực thi.

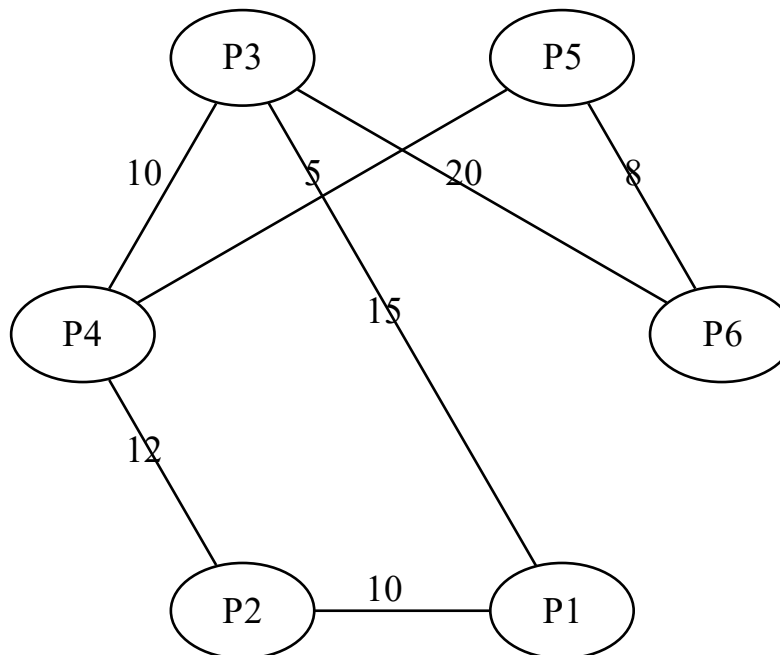
### Gợi ý:

- Dijkstra nhanh hơn nếu chỉ cần 1 nguồn → các câu hỏi tư duy thuật toán.

## PHẦN 5: BÀI TẬP DẠNG DỰ ÁN

### ✂ Dự án 1: Tối ưu hóa mạng giao thông trong khu công nghiệp

Một khu công nghiệp có 6 nhà máy: P1, P2, P3, P4, P5, P6. Các tuyến đường nối giữa chúng có **thời gian di chuyển (phút)** như sau:



Giả định rằng đây là **đồ thị vô hướng** (đường 2 chiều), không có chu trình âm.

### 📌 Nhiệm vụ:

- Biểu diễn đồ thị bằng ma trận trọng số.

- Áp dụng **Floyd-Warshall** để tính ma trận khoảng cách ngắn nhất giữa mọi nhà máy.
- Xác định:
  - Tuyến đường dài nhất trong tập các đường ngắn nhất.
  - Trung bình thời gian di chuyển giữa tất cả các cặp nhà máy.
- **Đề xuất cải tiến:** Nếu được phép xây thêm 1 tuyến mới, nên nối giữa 2 nhà máy nào để tối ưu?

### **Hướng dẫn thực hiện:**

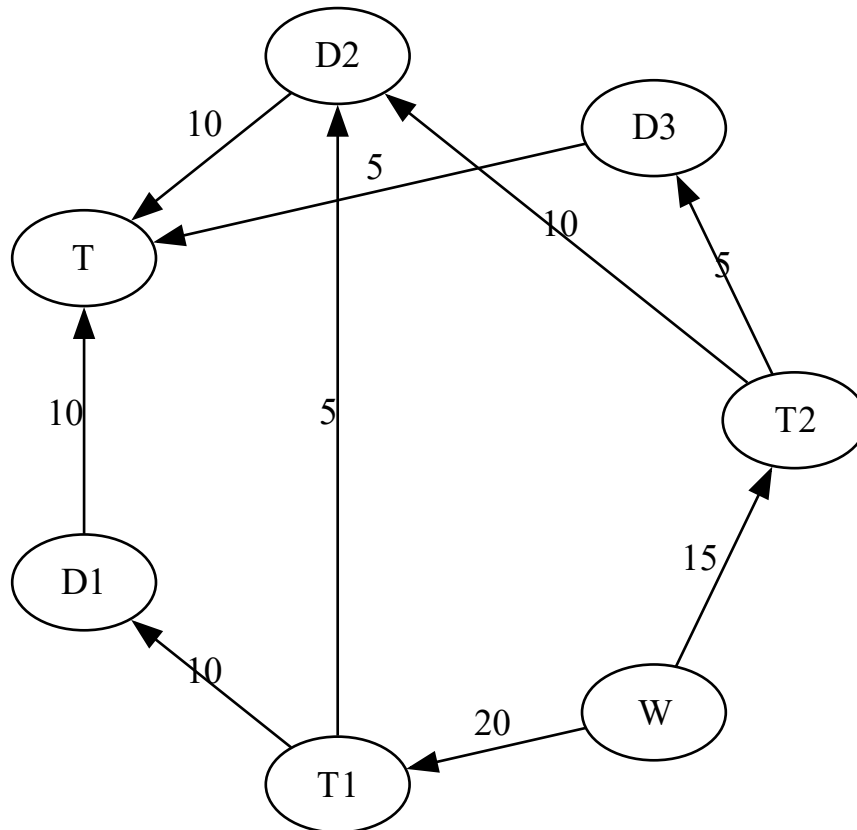
- **Bước 1:** Chuyển đồ thị thành ma trận  $D[i][j]$ .
- **Bước 2:** Triển khai Floyd-Warshall (nên lưu  $next[i][j]$ ).
- **Bước 3:** Duyệt toàn bộ  $D[i][j]$  để tính trung bình và xác định điểm tắc nghẽn.
- **Bước 4:** Mô phỏng thêm tuyến mới (ví dụ:  $P3 \leftrightarrow P5$ ), chạy lại thuật toán và so sánh.

### **Dự án 2: Quản lý phân phối nước sạch trong đô thị**

Một đô thị có 1 nhà máy nước W, 2 trạm trung chuyển T1, T2, và 3 khu dân cư D1, D2, D3.

Hệ thống ống dẫn có dung lượng như sau:





T là đỉnh tổng hợp "tiêu thụ" cuối cùng.

### **Nhiệm vụ:**

- **Vẽ đồ thị luồng** từ W đến T.
- Áp dụng **Ford-Fulkerson** để tính luồng nước cực đại có thể phân phối.
- Kiểm tra: Nếu nhu cầu nước ở các khu dân cư tăng lên 10%, có đáp ứng được không?
- **Đề xuất nâng cấp** tuyến cụ thể nếu cần.

### **Hướng dẫn thực hiện:**

- **Bước 1:** Thêm đỉnh nguồn  $S = W$ , đỉnh đích T, chuyển đồ thị sang dạng phù hợp.
- **Bước 2:** Áp dụng thuật toán Ford-Fulkerson hoặc Edmonds-Karp.
- **Bước 3:** So sánh tổng luồng cực đại với tổng nhu cầu:

- D1: 10, D2: 10, D3: 5 → tổng nhu cầu hiện tại: 25
- Nhu cầu tăng 10% → cần 27.5
- **Bước 4:** Xác định cạnh cổ chai (VD: T1 → D2) → đề xuất tăng dung lượng.