TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KỲ

PHÂN TÍCH THIẾT KẾ VÀ GIẢI THUẬT

ĐỀ 1

Người hướng dẫn: **TS. Trần Lương Quốc Đại**
Người thực hiện: **Ngô Phan Minh Trí – 52200171**
Châu Nguyễn Khánh Trình – 52200005
Lê Như Đạt - 52200160
Lớp**: 22050301**
Khoá**: 26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KỲ

PHÂN TÍCH THIẾT KẾ VÀ GIẢI THUẬT

ĐỀ 1

Người hướng dẫn: **TS. Trần Lương Quốc Đại**
Người thực hiện: **Ngô Phan Minh Trí – 52200171**
Châu Nguyễn Khánh Trình – 52200005
Lê Như Đạt - 52200160
Lớp**: 22050301**
Khoá**: 26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến thầy Đại vì đã tận tình hướng dẫn, truyền đạt kiến thức và dành thời gian quý báu hỗ trợ chúng em trong suốt quá trình thực hiện bài báo cáo này. Những góp ý và chỉ dẫn của thầy đã giúp chúng em hiểu sâu hơn về đề tài, hoàn thiện kỹ năng nghiên cứu và trình bày. Chúng em xin trân trọng cảm ơn thầy và kính chúc thầy luôn mạnh khỏe, thành công trong sự nghiệp trồng người.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của TS. Trần Lương Quốc Đại. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong luận văn còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung luận văn của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

Châu Nguyễn Khánh Trình

Ngô Phan Minh Trí

Lê Như Đạt

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày 16 tháng 11 năm 2024

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày 16 tháng 11 năm 2024

TÓM TẮT

Bài báo cáo được hoàn tất 100% với các những vấn đề cần được giải quyết là Coinrow Problem và The Maximum-Flow Problem. Giải quyết các vấn đề trong bài báo cáo bằng những tư duy logic và những gì đã học trong môn phân tích thiết kế và giải thuật.

MỤC LỤC

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

CÁC KÝ HIỆU

f *Tần số của dòng điện và điện áp (Hz)*

p *Mật độ điện tích khối (C/m3)*

CÁC CHỮ VIẾT TẮT

CSTD	Công suất tác dụng
MF	Máy phát điện
BER	Tỷ lệ bít lỗi

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

CHƯƠNG 1 - COIN-ROW PROBLEM

1.1 Cơ sở lý thuyết

1.1.1 Mô tả bài toán

Bài toán yêu cầu như sau:

- Đầu vào:
  - Một tập gồm **n** đồng xu, với giá trị lần lượt là $c_1, c_2, \ldots, c_n$.
  - Các giá trị này là số nguyên dương và không nhất thiết phải giống nhau.
- Đầu ra:
  - Tổng giá trị lớn nhất có thể thu được khi chọn các đồng xu, **đảm bảo rằng không chọn hai đồng xu liền kề** trong hàng (Nếu chọn đồng xu thứ i, bạn không được chọn đồng xu thứ i – 1 hoặc i + 1).
  - Ví dụ:

    Giả sử dãy đồng xu có giá trị như sau: {5, 1, 2, 10, 6, 2}

    Bạn không được chọn 5 và 1 cùng lúc vì chúng nằm cạnh nhau. Tương tự, bạn không được chọn 10 và 6 cùng lúc.

    Mục tiêu: Chọn các đồng xu sao cho tổng giá trị thu được là lớn nhất.

1.2 Phương pháp giải

- Bài toán có thể giải quyết bằng **kỹ thuật quy hoạch động (Dynamic Programming).** Đây là một kỹ thuật giải bài toán tối ưu bằng cách chia nhỏ bài toán lớn thành các bài toán con nhỏ hơn, giải quyết chúng một lần, và lưu trữ kết quả để tránh tính toán lại. Ví dụ: Bài toán Fibonacci, Knapsack Problem,…).

- Phương pháp giải:

Gọi **F(n)** là tổng giá trị lớn nhất có thể thu được khi chọn các đồng xu từ dãy gồm n đồng xu đầu tiên, với điều kiện không chọn 2 đồng xu liền kề.

Khi xét đồng xu thứ **n** (đồng xu cuối cùng), ta có hai lựa chọn:

- **Không chọn đồng xu thứ n**: Trong trường hợp này, tổng giá trị lớn nhất khi xét n đồng xu bằng tổng giá trị lớn nhất khi xét n − 1 đồng xu.

  F(n) = F (n - 1)

- **Chọn đồng xu thứ n**: Khi chọn đồng xu thứ n, không chọn đồng xu thứ n - 1. Khi đó, giá trị tối ưu là giá trị của đồng xu n (c_n) cộng với giá trị tối ưu của dãy n − 2 đồng xu trước đó.

  F(n) = c_n + F (n - 2)

- Kết hợp hai trường hợp:

  F(n) = max{F(n − 1), c_n +F(n − 2)}, n > 1

  F(0) = 0: Nếu không có đồng xu nào, giá trị lớn nhất là 0.

  F(1) = c_1: Nếu chỉ có một đồng xu, giá trị lớn nhất là giá trị của đồng xu đó.

1.3 Example

1.3.1 Example 1

- Dãy đồng xu có giá trị {5, 1, 2, 10, 6, 2}

- F(0) = 0
- F(1) = c_1 = 5
- F(2) = max{F(1), c_2 +F(0)} = max{5, 1 + 0} = 5
- F(3) = max{F(2), c_3 +F(1)} = max{5, 2 + 5} = 7
- F(4) = max{F(3), c_4 +F(2)} = max{7, 10 + 5} = 15
- F(5) = max{F(4), c_5 +F(3)} = max{15, 6 + 7} = 15
- F(6) = max{F(5), c_6 +F(4)} = max{15, 2 + 15} = 17

  Vậy tổng giá trị tối đa có thể thu được là 17

n	F(n)	Diễn giải
0	0	Không có đồng xu
1	5	Chọn đồng xu $c_1 = 5$
2	5	Không chọn $c_2 = 1$, giữ F(1) = 5
3	7	Chọn $c_3 = 2$, cộng với F(1) = 5
4	15	Chọn $c_4 = 10$, cộng với F(2) = 5
5	15	Không chọn $c_5 = 6$, giữ F(4) = 15
6	17	Chọn $c_6 = 2$, cộng với F(4) = 15

1.3.2 Example 2

- Dãy đồng xu có giá trị {4, 12, 7}

  - F(0) = 0
  - F(1) = $c_1 = 4$
  - F(2) = max{F(1), c_2 +F(0)} = max{4, 12 + 0} = 12
  - F(3) = max{F(2), c_3 +F(1)} = max{12, 7 + 4} = 12

    Vậy tổng giá trị tối đa có thể thu được là 12

n	F(n)	Diễn giải
0	0	Không có đồng xu
1	4	Chọn đồng xu $c_1 = 4$
2	12	Chọn $c_2 = 12$, bỏ qua F(1) = 4
3	12	Không chọn $c_3 = 7$, giữ F(2) = 12

1.3.3 Example 3

- Dãy đồng xu có giá trị {3, 3, 3, 3, 3}

  - F(0) = 0

- $F(1) = c_1 = 3$
- $F(2) = \max\{F(1), c_2 + F(0)\} = \max\{3, 3 + 0\} = 3$
- $F(3) = \max\{F(2), c_3 + F(1)\} = \max\{3, 3 + 3\} = 6$
- $F(4) = \max\{F(3), c_4 + F(2)\} = \max\{6, 3 + 3\} = 6$
- $F(5) = \max\{F(4), c_5 + F(3)\} = \max\{6, 3 + 6\} = 9$

Vậy tổng giá trị tối đa có thể thu được là 9

n	**F(n)**	**Diễn giải**
0	0	Không có đồng xu
1	3	Chọn đồng xu $c_1 = 3$
2	3	Không chọn $c_2 = 3$, giữ $F(1) = 3$
3	6	Chọn $c_3 = 3$, cộng với $F(1) = 3$
4	6	Không chọn $c_4 = 10$, giữ $F(3) = 6$
5	9	Chọn $c_5 = 3$, cộng với $F(3) = 6$

1.4 Trình bày, so sánh và phân tích kết quả

1.4.1 Trình bày

- Function **coin_row()**:
  - Input: Tập hợp các đồng xu
  - Output: Giá trị tối ưu F(n)

  Thuật toán chạy như trình bày ở phần cơ sở lý thuyết

- Function **TruyXuat()**:
  - Input: Tập hợp các đồng xu
  - Out put: Tập hợp các đồng xu được chọn
    - Đầu tiên, hàm sẽ chạy lại thuật toán CoinRow để có được tập F
    - Hàm sẽ duyệt qua tập hợp bằng cách chạy từ cuối tập hợp chạy về đầu

- Nếu giá trị đang xét F[i] = giá trị F[i-1] thì hàm sẽ duyệt phần tử tiếp theo
- Khi nào hàm thấy giá trị F bị thay đổi thì hàm sẽ lấy đồng xu ở vị trí đó và index – 2(Vì chọn đồng xu không kề nhau).
- Khi index = 0 dừng vòng lặp.

1.4.2 So sánh và phân tích kết quả

- Kết quả:

```
Thực nghiệm thuật toán:

Trường hợp 1: []
  Tổng giá trị tối đa: 0
  Các đồng xu được chọn: []

Trường hợp 2: [1]
  Tổng giá trị tối đa: 1
  Các đồng xu được chọn: [1]

Trường hợp 3: [5, 5, 10, 100, 10, 5]
  Tổng giá trị tối đa: 110
  Các đồng xu được chọn: [5, 100, 5]

Trường hợp 4: [3, 2, 5, 10, 7]
  Tổng giá trị tối đa: 15
  Các đồng xu được chọn: [3, 5, 7]

Trường hợp 5: [10, 20, 30, 40, 50]
  Tổng giá trị tối đa: 90
  Các đồng xu được chọn: [10, 30, 50]
```

Hình 1. Kết quả chạy thực nghiệm 5 trường hợp

- Trường hợp 1: Dãy không có đồng xu nào, giá trị tối ưu của dãy là 0
- Trường hợp 2: Dãy có 1 đồng xu, giá trị của đồng xu là giá trị tối ưu của dãy.
- Trường hợp 3: Có giá trị lớn xen kẽ, thuật toán chọn được 100 là lớn nhất.

- Trường hợp 4: Thuật toán chạy từng bước để có thể chạy ra đáp án cuối cùng.
- Trường hợp 5: Giá trị tăng dần, thuật toán chọn các giá trị cách đều 10, 30, 50.

1.5 Đánh giá độ phức tạp

1.5.1 Độ phức tạp thời gian

- Với n là số lượng đồng xu:
  - Công thức truy hồi tính f(i) trong O(1).
  - Tính toàn bộ f(1), f(2), ..., f(n) mất O(n) thời gian.

=> Độ phức tạp thời gian của thuật toán là: O(n)

1.5.2 Độ phức tạp không gian

- Nếu sử dụng một mảng f để lưu trữ tất cả các giá trị f(0), f(1), ..., f(n) độ phức tạp không gian là O(n).
- Tuy nhiên, vì f(i) chỉ phụ thuộc vào f(i−1) và f(i−2), ta có thể tối ưu hóa bằng cách chỉ lưu trữ hai giá trị gần nhất, giảm không gian xuống O(1).

Kết luận:

- **Độ phức tạp thời gian:** O(n).
- **Độ phức tạp không gian:**
  - O(n) nếu lưu toàn bộ mảng.
  - O(1) nếu tối ưu hóa không gian.

CHƯƠNG 2 - THE MAXIMUM-FLOW PROBLEM

2.1 Cơ sở lý thuyết

2.1.1 Định nghĩa

- Giả sử ta có một mạng lưới để vận chuyển một loại vật chất (như nước trong hệ thống ống dẫn, hoặc dòng điện trong một hệ thống điện). Mục tiêu của ta là đẩy càng nhiều vật chất qua mạng lưới này từ một điểm xuất phát (gọi là *source*) đến điểm đích cuối cùng (gọi là *sink*).
  - **Source**: Đây là nơi vật chất bắt đầu di chuyển vào mạng lưới, không có gì "chảy vào" nguồn.
  - **Sink**: Đây là nơi vật chất thoát ra khỏi mạng lưới, không có gì "chảy ra" từ đích.
  - **Edge capacity**: Mỗi cạnh có hướng với trọng số u_{ij} là một số nguyên dương, được gọi là dung lượng cạnh.

2.1.2 Tính chất

- **Dung lượng cạnh**: Mỗi đoạn đường (cạnh) nối giữa hai điểm trong mạng lưới đều có một giới hạn dung lượng – tức là giới hạn về lượng vật chất tối đa có thể di chuyển qua đó.
- **Yêu cầu bảo toàn dòng chảy**: Tại mỗi điểm trung gian, tổng lượng vật chất đi vào phải bằng tổng lượng đi ra.

$$\sum_{j:(j,i)\in E} x_{ji} = \sum_{j:(i,j)\in E} x_{ij} \quad \text{for } i = 2, 3, \ldots, n-1,$$

- **Value(v)**: Tổng lượng vật liệu đi ra khỏi nguồn phải đi đến bồn chứa.

$$\sum_{j:(1,j)\in E} x_{1j} = \sum_{j:(j,n)\in E} x_{jn}.$$

- Luồng (khả thi): là phép gán các số thực x_{ij} cho các cạnh (i, j) của một mạng nhất định thỏa mãn các ràng buộc **bảo toàn luồng** và các ràng buộc về **dung lượng.**

$$0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i, j) \in E.$$

2.1.3 Phát biểu

- Bài toán maximum-flow problem có thể được phát biểu như sau:

$$\text{maximize} \quad v = \sum_{j:(1,j)\in E} x_{1j}$$

$$\text{subject to} \quad \sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = 0 \quad \text{for } i = 2, 3, \ldots, n-1$$

$$0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i, j) \in E.$$

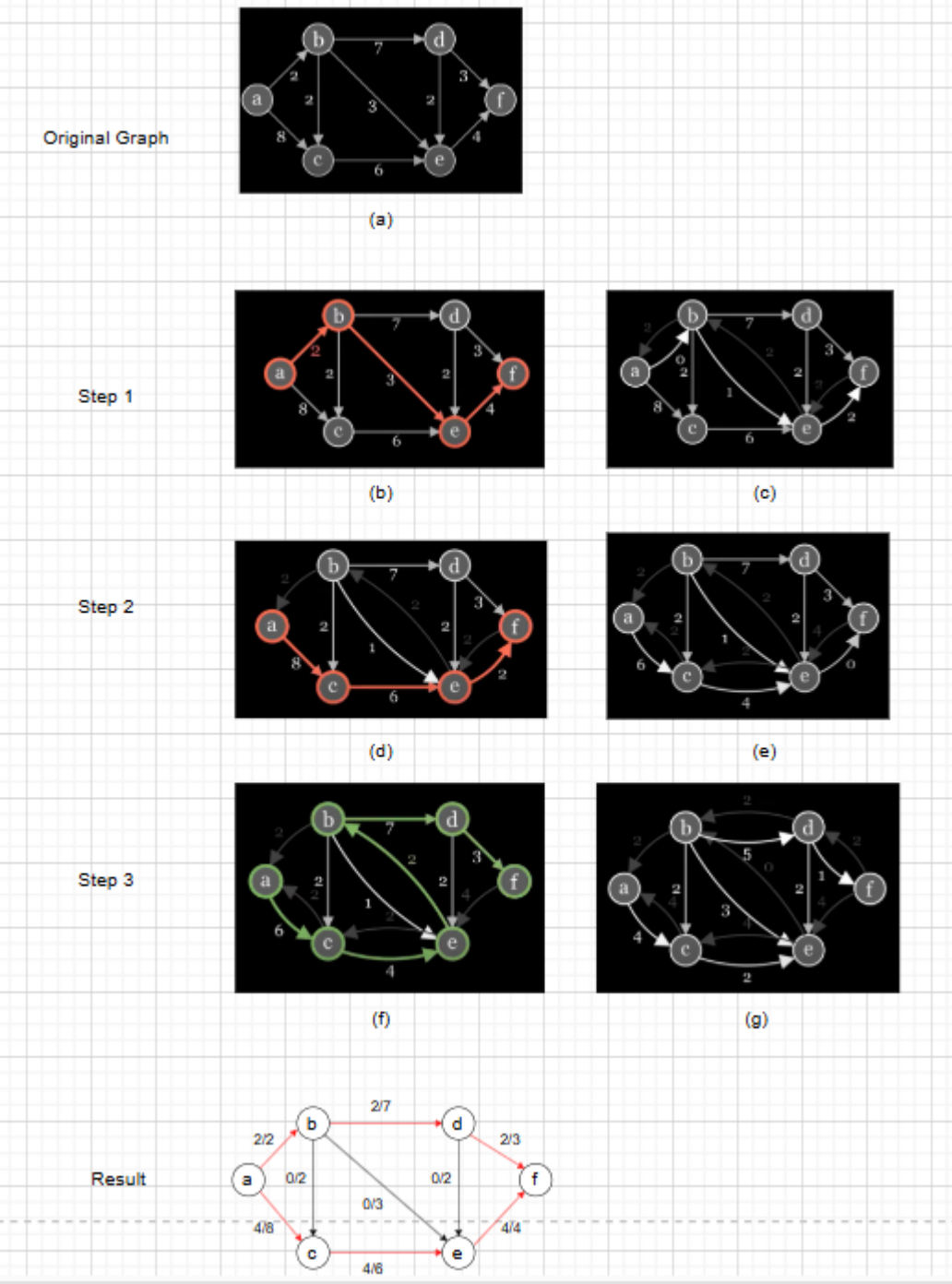
2.2 Phương pháp giải

2.2.1 Ford-Fulkerson

- Bài toán luồng cực đại có thể được giải bằng phương pháp **Ford-Fulkerson**. **Ford-Fulkerson** là một thuật toán để tìm luồng cực đại trong một đồ thị luồng. Thuật toán này thường được sử dụng để giải các bài toán tối ưu hóa mạng, đặc biệt trong các trường hợp liên quan đến luồng (flow) như dòng chảy trong hệ thống ống nước hoặc mạng giao thông.

  Cách hoạt động

1. **Khởi đầu với dòng chảy bằng 0**: Ban đầu, chúng ta đặt lượng dòng chảy qua tất cả các đoạn (cạnh) trong mạng lưới là 0.
2. **Tìm đường dẫn tăng dòng chảy**:
    - Chúng ta tìm một **đường dẫn từ nguồn đến đích** sao cho mỗi đoạn của đường này còn dư sức chứa. Đây là **đường dẫn tăng dòng chảy**.
    - Ví dụ, nếu cạnh từ đỉnh A đến B có sức chứa là 5 và hiện đang có 2 đơn vị dòng chảy qua, thì ta có thể tăng thêm tối đa 3 đơn vị dòng chảy qua cạnh đó.
3. **Tăng dòng chảy dọc theo đường dẫn này**:
    - Khi tìm được một đường dẫn tăng dòng chảy, ta sẽ tăng lượng dòng chảy trên các đoạn của đường này.
    - Lượng tăng thêm được chọn là **dung lượng nhỏ nhất còn trống trên các cạnh** của đường dẫn (ví dụ như đoạn có dư sức chứa ít nhất sẽ giới hạn lượng tăng thêm của toàn bộ đường dẫn).
4. **Lặp lại cho đến khi không thể tìm được đường dẫn tăng dòng chảy**:
    - Ta lặp lại quá trình này nhiều lần, mỗi lần tìm và tăng dòng chảy trên một đường dẫn mới.
    - Khi không còn đường dẫn tăng dòng chảy nào nữa từ nguồn đến đích, lúc này dòng chảy hiện tại chính là dòng chảy tối đa mà mạng có thể đạt được.



Hình ảnh minh họa của phương pháp đường dẫn tăng cường (Ford-Fulkerson method).

Giải thích

Ở step 1, ta có thể chọn ngẫu nhiên một đường đi từ đỉnh nguồn tới đích (Trong trường hợp này ta chọn đường $a \rightarrow b \rightarrow e \rightarrow f$). Ta dễ dàng thấy lượng vật chất mà luồng này có thể đi được là 2 (yêu cầu bảo toàn dòng chảy).

Cạnh ngược sẽ là dung lượng khi đã có một lượng vật chất đi qua. Cạnh ngược được tính như sau: Ta lấy cạnh $a \rightarrow b$ làm ví dụ.

- Khi lượng vật chất truyền qua cạnh $a \rightarrow b$ với 2 đơn vị vật chất. Thì cạnh ngược sẽ được tạo ra và sẽ có giá trị là 2, và cạnh $a \rightarrow b$ sẽ còn lại 0 đơn vị (vì đã sử dụng hết để truyền vật chất)
- Tương tự với cạnh $b \rightarrow e$ ta có: cạnh ngược = 2, cạnh $b \rightarrow e$ lúc này còn lại 1 đơn vị (3 – 2 = 1).

Ở step 2 này thì cũng chọn ra một đường từ nguồn đến đích ($a \rightarrow c \rightarrow e \rightarrow f$) và làm tương tự như bước 1.

Ở step 3 này ta chọn ra một đường từ nguồn đến đích ($a \rightarrow c \rightarrow e \rightarrow b \rightarrow d \rightarrow f$). Để rõ được vì sao ta lại có thể đi được đường này thì chúng ta sẽ làm rõ 2 câu hỏi như sau:

- Tại sao ta có thể đi từ $e \rightarrow b$? Tại vì trước đó ở step 2 ta đã tạo ra một cạnh ngược từ $e \rightarrow b$ và dung lượng trên cạnh này vẫn còn nên ta có thể đi được.
- Tại sao ta không thể đi từ $e \rightarrow f$? Tại vì dung lượng trên cạnh $e \rightarrow f$ đã bằng 0 rồi nên ta không thể đi qua được. Các bước cập nhật cạnh cũng giống như các step trên.

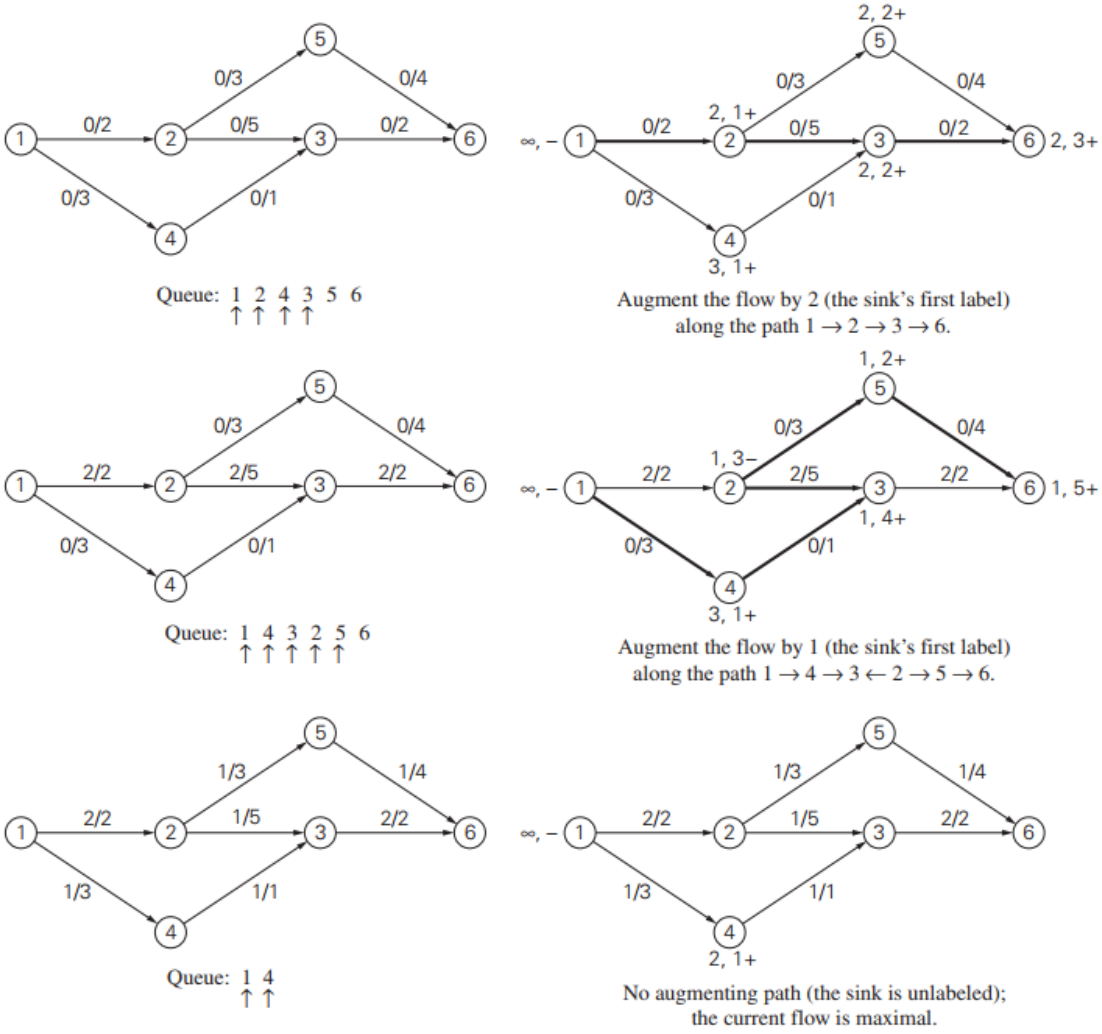
- Sau khi hoàn thành xong step 3 thì thuật toán dừng lại, ta thu được luồng cực đại như hình trên. Lưu ý đường màu đỏ là đường ta chọn để di chuyển lượng vật chất đi qua, còn đường màu đen không cho vật chất đi qua.

2.2.2 Thuật toán Edmonds-Karp

- Edmonds-Karp là một biến thể của Ford-Fulkerson, trong đó **tìm kiếm theo chiều rộng (BFS)** được sử dụng để tìm đường đi tăng cường ngắn nhất (theo số cạnh). Thuật toán này khắc phục được vòng lặp vô hạn trong các đồ thị có cạnh với dung lượng không nguyên. Thuật toán này được gọi là **shortest-augmenting-path** hoặc **first-labeled-firstscanned algorithm.**
- Thuật toán này sử dụng việc đánh nhãn giúp tìm đường đi từ đỉnh nguồn (*source*) đến đỉnh đích (*sink*) trong đồ thị khả dụng (*residual graph*) bằng cách theo dõi khả năng luồng còn có thể truyền qua từng đỉnh. Quy trình cụ thể như sau:
    - **Ý nghĩa của nhãn (label)**: Khi một đỉnh mới (*unlabeled vertex*) được đánh nhãn, nó sẽ có hai thông tin:
        - **Nhãn đầu tiên** cho biết lượng luồng tối đa có thể truyền từ nguồn đến đỉnh này.
        - **Nhãn thứ hai** là tên của đỉnh liền trước đã dẫn đến đỉnh này (có thể bỏ qua với đỉnh nguồn). Để dễ nhận biết hướng của cạnh, ký hiệu "+" sẽ được thêm vào nếu đỉnh được đến qua cạnh thuận (forward edge) và "–" nếu đi qua cạnh ngược (backward edge).
- **Cách đánh nhãn đỉnh nguồn**: Đỉnh nguồn luôn được đánh nhãn là: ∞, - (vô cực, không có đỉnh trước).
- **Cách đánh nhãn các đỉnh khác**:
    - Nếu một đỉnh chưa có nhãn (*vertex j*) được kết nối với đỉnh đầu của hàng đợi (*vertex i*) qua một cạnh thuận từ i đến j có dung lượng khả dụng (*residual capacity*) dương $r_{ij} \leftarrow u_{ij} - x_{ij}$, thì nhãn của j sẽ là:
        - Nhãn đầu tiên: $l_j \leftarrow \min\{l_j, r_{ij}\}$, với l_j là luồng tối đa có thể đi qua đỉnh i.
        - Nhãn thứ hai: i^+, cho biết rằng j được đến từ i qua cạnh thuận.

- Nếu một đỉnh chưa có nhãn (*vertex j*) được kết nối với đỉnh đầu của hàng đợi (*vertex i*) qua một cạnh ngược từ *j* đến *i* có luồng dương x_{ij}, thì nhãn của *j* sẽ là:
  - Nhãn đầu tiên: $l_j \leftarrow \min\{l_j, x_{ij}\}$
  - Nhãn thứ hai: i^-, cho biết rằng *j* được đến từ *i* qua cạnh ngược.

- Nếu quy trình đánh nhãn này dẫn đến việc đỉnh đích (*sink*) cũng được đánh nhãn, có nghĩa là có một đường đi tăng cường (*augmenting path*) từ nguồn đến đích. Luồng có thể được tăng thêm một lượng bằng với giá trị của nhãn đầu tiên của đỉnh đích.
- Việc tăng cường luồng này thực hiện bằng cách đi ngược lại từ đỉnh đích đến đỉnh nguồn theo các nhãn thứ hai, để xác định đường đi tăng cường. Luồng được tăng lên trên các cạnh thuận và giảm đi trên các cạnh ngược.
- Nếu hàng đợi hết mà đỉnh đích không có nhãn, nghĩa là không còn đường đi nào có thể truyền thêm luồng từ nguồn đến đích. Lúc này, thuật toán kết thúc và trả về luồng hiện tại là luồng cực đại.



Hình 2: Hình ảnh minh họa kết quả thuật toán

2.2.3 Mã giả của thuật toán ShortestAugmentingPath

ALGORITHM *ShortestAugmentingPath(G)*

```
//Implements the shortest-augmenting-path algorithm
//Input: A network with single source 1, single sink n, and
//positive integer capacities u_ij on its edges (i, j)
//Output: A maximum flow x
assign x_ij = 0 to every edge (i, j) in the network
label the source with ∞, - and add the source to the empty queue Q
while not Empty(Q) do
    i ← Front(Q); Dequeue(Q)
    for every edge from i to j do //forward edges
        if j is unlabeled
            r_ij ← u_ij - x_ij
            if r_ij > 0
                l_j ← min{l_i , r_ij}; label j with l_j , i+
                Enqueue(Q, j)
    for every edge from j to i do //backward edges
        if j is unlabeled
            if x_ij > 0
                l_j ← min{l_j, x_ij}; label j with l_j, i−
                Enqueue(Q, j)
    if the sink has been labeled
        //augment along the augmenting path found
        j ← n //start at the sink and move backwards using second //labels
        while j ≠ 1 //the source hasn't been reached
```

if the second label of vertex j is i^+

$x_{ij} \leftarrow x_{ij} + l_n$

else //the second label of vertex j is i^-

$x_{ij} \leftarrow x_{ij} - l_n$

j ← i; i ← the vertex indicated by i's second label erase all vertex labels except the ones of the source reinitialize Q with the source

return x //the current flow is maximum

2.2.4 Network cuts

- **Cut**: Một phép cắt được tạo ra bằng cách phân vùng các đỉnh của mạng thành một tập con X chứa nguồn và X, X^- là phần bù của X. Chứa phần chìm là tập hợp tất cả các cạnh có đuôi trong X và đầu trong X^-, tập hợp này được gọi là C(X, X^-), sau khi bỏ đi các cạnh của đồ thị trong tập C thì sẽ không còn đường đi nào từ nguồn đến đích trong đồ thị.
- **Minimum cut**: do tồn tại rất nhiều **cut** nên luôn tồn tại một vết cắt tối thiểu, tức là một vết cắt có khả năng nhỏ nhất.
- **Max-Flow Min-Cut Theorem**: The value of a maximum flow in a network is equal to the capacity of its minimum cut (giá trị của luồng cực đại trong mạng bằng với khả năng cắt cực tiểu của nó).

Thuật toán Augmenting-Path và Cắt Cực Tiểu:

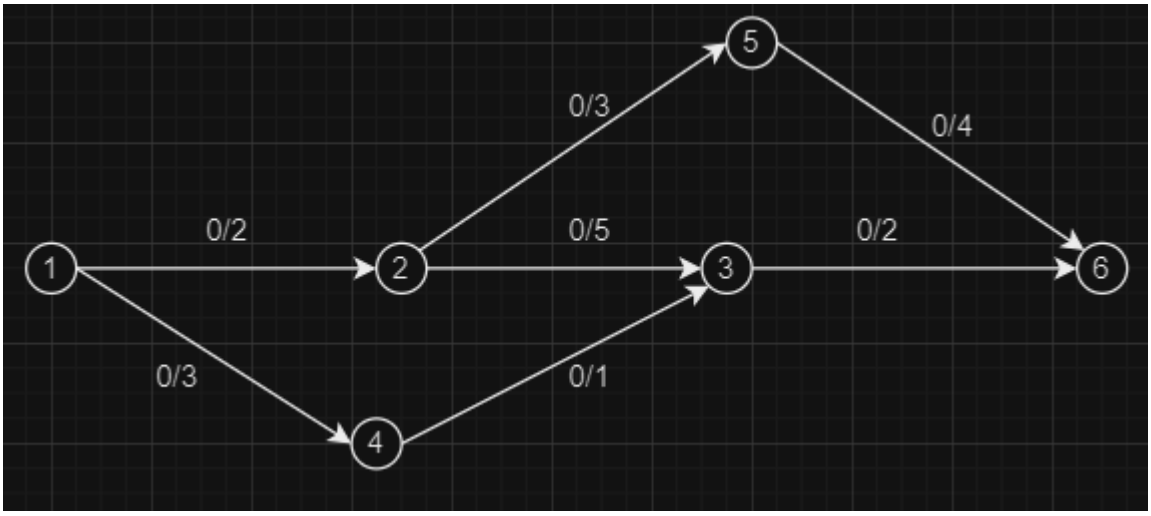
- Thuật toán Shortest-Augmenting-Path tìm một đường đi tăng cường ngắn nhất trong mạng và tiếp tục cho đến khi không thể tăng cường dòng chảy nữa.
- Cắt cực tiểu trong quá trình này được xác định bởi các cạnh từ các đỉnh được gán nhãn (labeled) đến các đỉnh không được gán nhãn (unlabeled)

trong vòng lặp cuối cùng. Các cạnh này tạo thành một cắt cực tiểu và có công suất bằng với giá trị dòng chảy cực đại.

- Khi thuật toán kết thúc, các cạnh từ các đỉnh labeled đến unlabeled phải là đầy (tức là dòng chảy trên các cạnh này phải bằng công suất của chúng). Ngược lại, các cạnh từ các đỉnh unlabeled đến labeled nếu có thì phải có dòng chảy bằng 0 (các cạnh này không có dòng chảy).

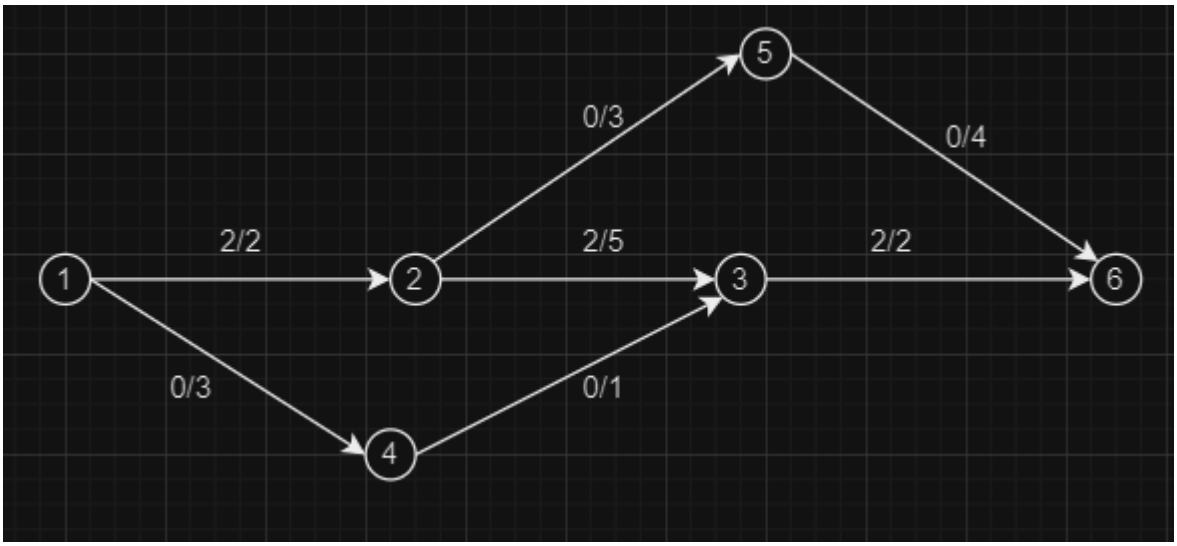
2.3 Example

2.3.1 Example 1



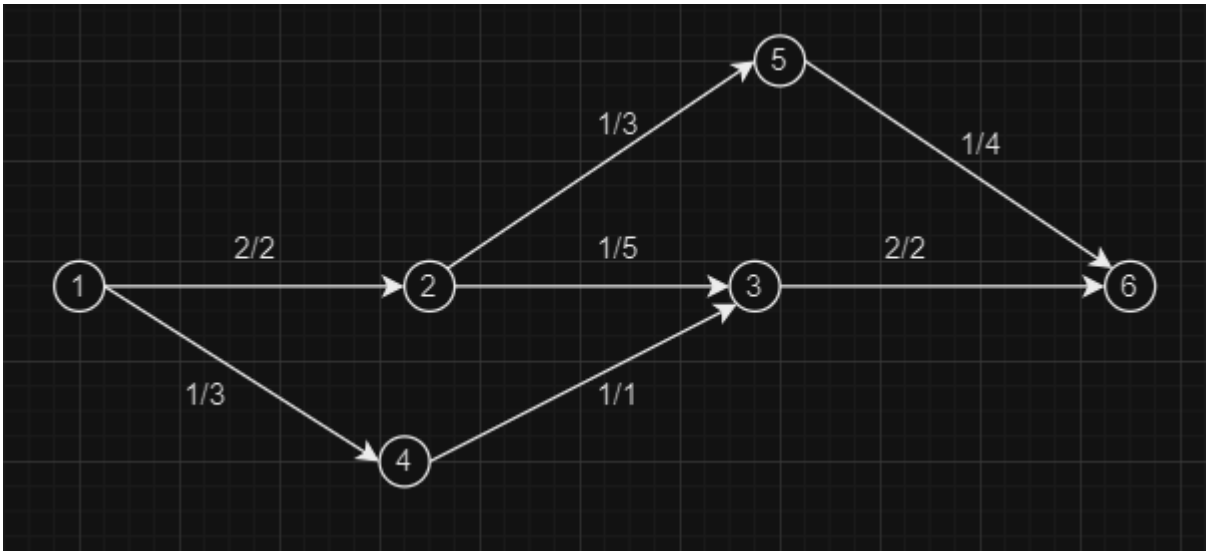
Hình 3. Đồ thị của ví dụ 1

- Gán mọi cạnh ij = 0, tạo label cho source = inf, ‘-’, dùng bfs để lan ra và tạo label cho những đỉnh thỏa điều kiện **giới hạn dung lượng**
- Label: 1: [inf, '-'], 2: [2, '1+'], 3: [2, '2+'], 4: [3, '1+'], 5: [2, '2+'], 6: [2, '3+']
- Khi label của sink đã được gán nhãn thì ta sẽ gán giá trị ngược về từ sink, phải đảm bảo được điều kiện **bảo toàn luồng**



Hình 4. Lần chạy đầu tiên trong ví dụ 1

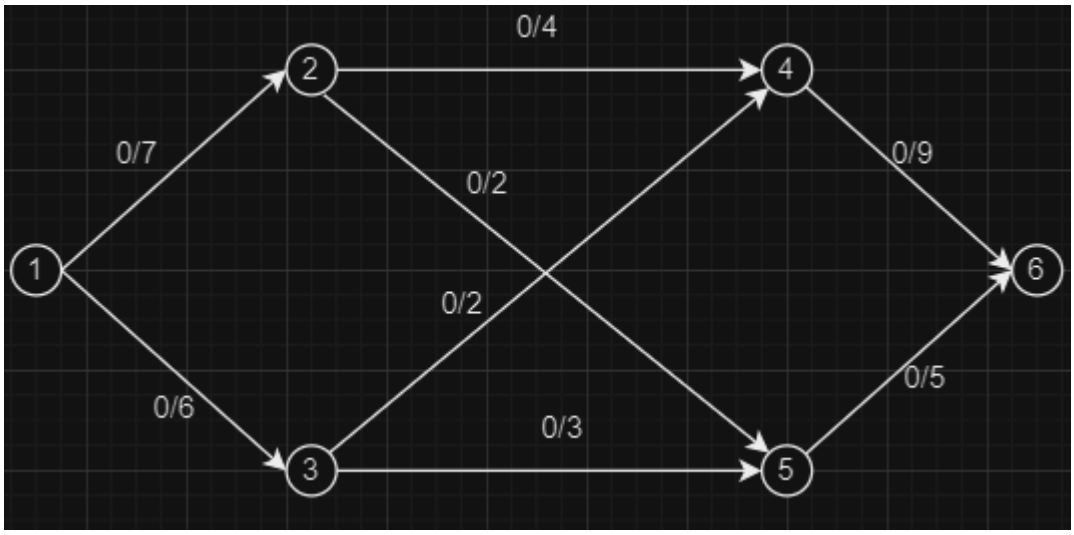
- Xóa tất cả nhãn trừ nhãn source và tiếp tục quá trình gán nhãn
- Label: 1: [inf, '-'], 2: [1, '3-'], 3: [1, '4+'], 4: [3, '1+'], 5: [1, '2+'], 6: [1, '5+']
- Gán giá trị ngược về.



Hình 5. Kết quả của ví dụ 1

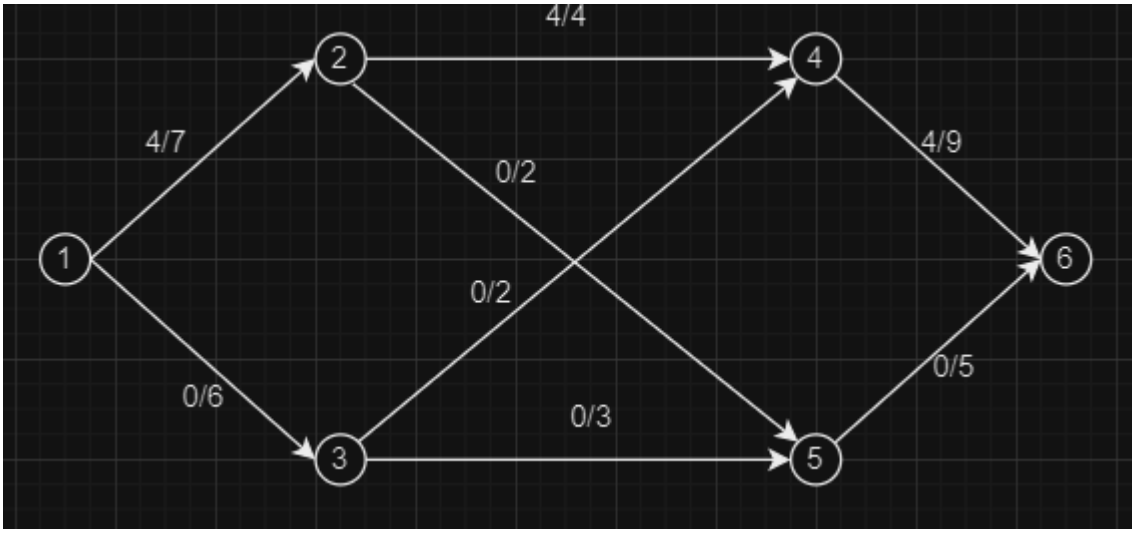
- Kết thúc quá trình do đã đảm bảo được tất cả **giới hạn dung lượng**

2.3.1 Example 2



Hình 6. Đồ thị của ví dụ 2

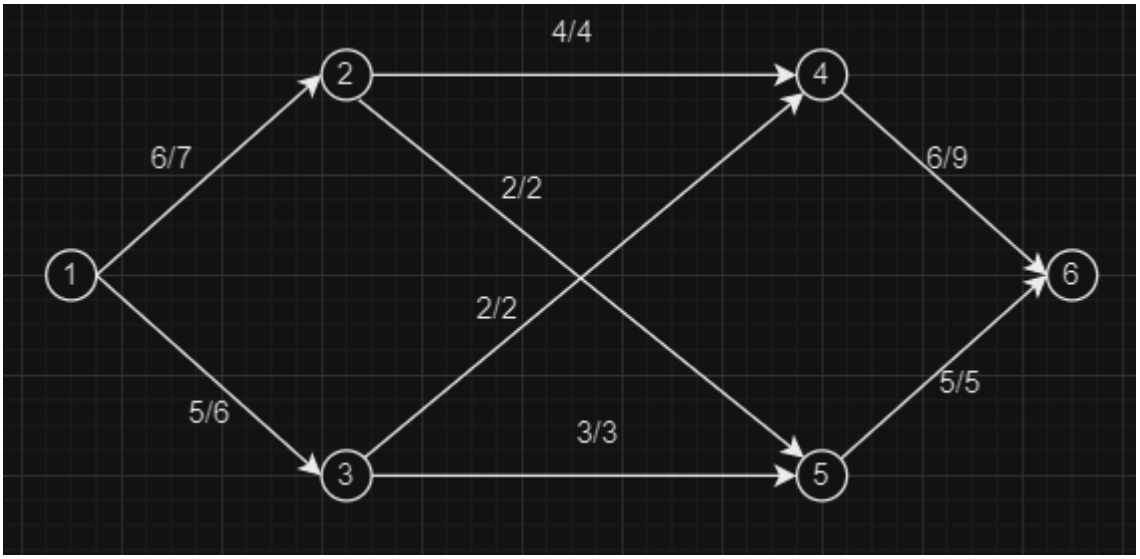
- Gán mọi cạnh ij = 0, tạo label cho source = inf, ‘-’, dùng bfs để lan ra và tạo label cho những đỉnh thỏa điều kiện **giới hạn dung lượng**
- Label: 1: [inf, '-'], 2: [7, '1+'], 3: [6, '1+'], 4: [4, '2+'], 5: [2, '2+'], 6: [4, '4+']
- Khi label của sink đã được gán nhãn thì ta sẽ gán giá trị ngược về từ
- sink, phải đảm bảo được điều kiện **bảo toàn luồng**



Hình 7. Lần chạy đầu tiên của ví dụ 2

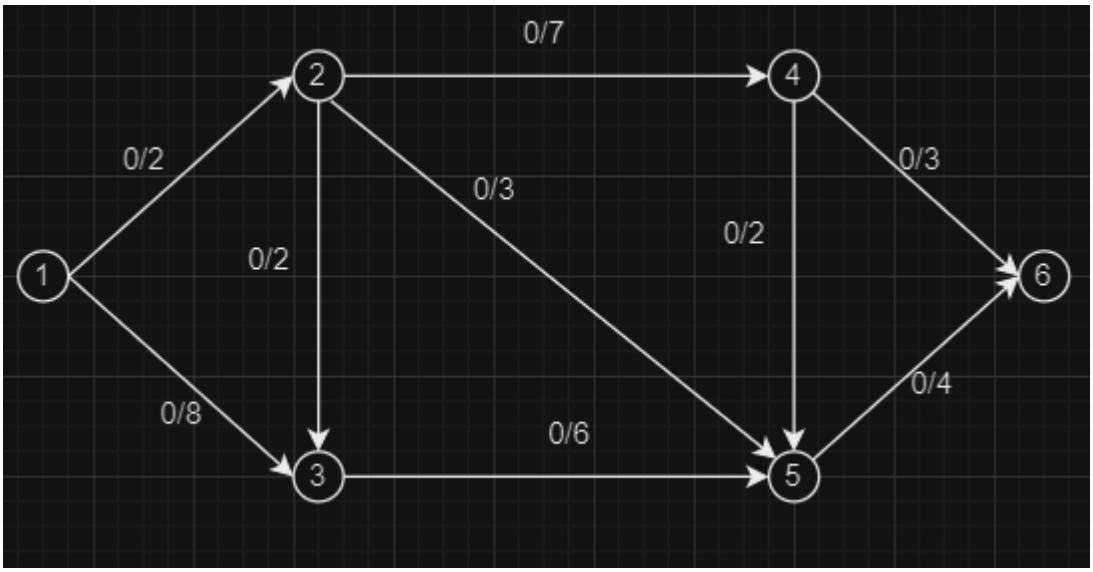
- Lặp lại quá trình này, các label tiếp theo

- 1: [inf, '-'], 2: [3, '1+'], 3: [6, '1+'], 4: [2, '3+'], 5: [2, '2+'], 6: [2, '5+']
- 1: [inf, '-'], 2: [1, '1+'], 3: [6, '1+'], 4: [2, '3+'], 5: [3, '3+'], 6: [2, '4+']
- 1: [inf, '-'], 2: [1, '1+'], 3: [4, '1+'], 4: 0, 5: [3, '3+'], 6: [3, '5+']
- Kết quả cuối



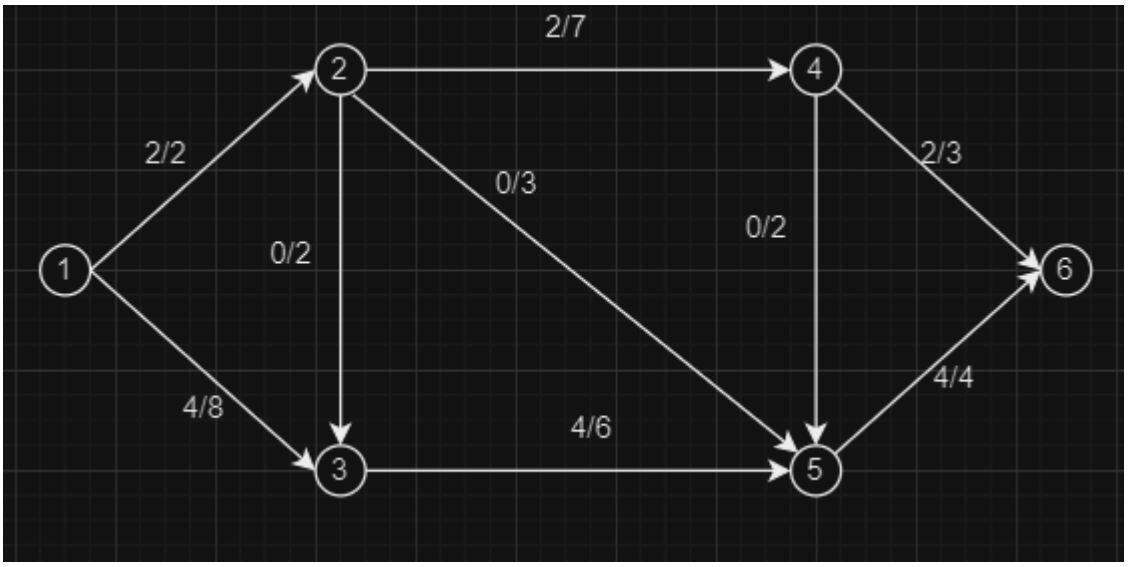
Hình 8. Kết quả của ví dụ 2

2.3.1 Example 3



Hình 9. Đồ thị của ví dụ 3

- Gán mọi cạnh ij = 0, tạo label cho source = inf, ‘-’, dùng bfs để lan ra và tạo label cho những đỉnh thỏa điều kiện **giới hạn dung lượng**
- Label: 1: [inf, '-'], 2: [2, '1+'], 3: [8, '1+'], 4: [2, '2+'], 5: [2, '2+'], 6: [2, '4+']
- Khi label của sink đã được gán nhãn thì ta sẽ gán giá trị ngược về từ sink, phải đảm bảo được điều kiện **bảo toàn luồng**
- Lặp lại quá trình và đây là quá trình cuối cùng



Hình 10. Kết quả của ví dụ 3

2.4 Trình bày, so sánh và phân tích kết quả

2.4.1 Trình bày

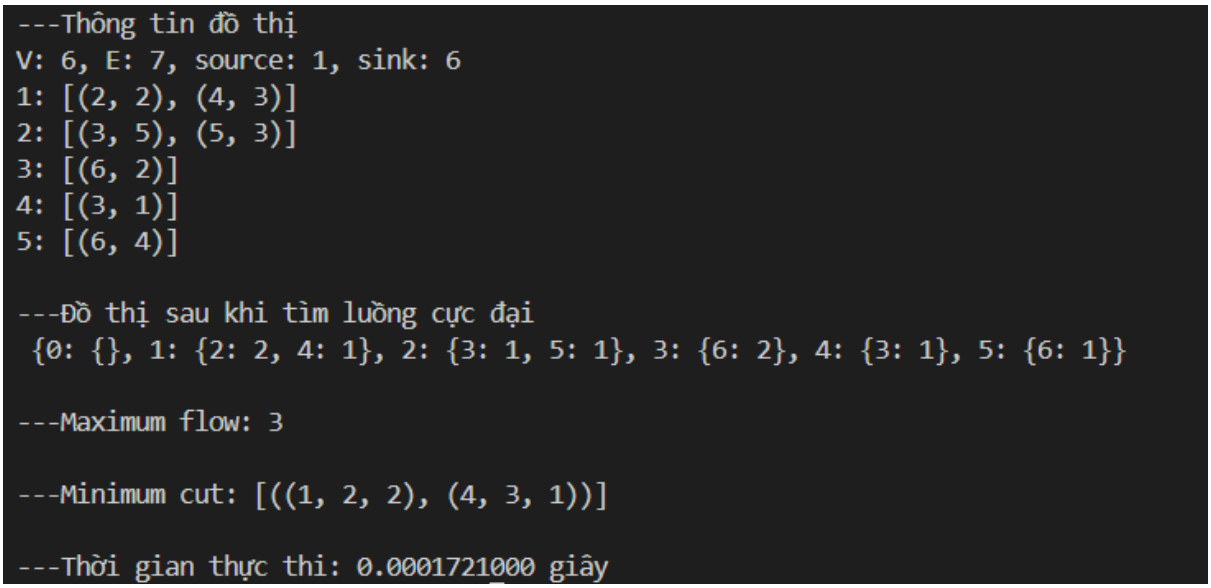
- Function **shortest_augmenting_path()**:
  - Input: thông tin đồ thị
  - Output: x
    - x: đồ thị dưới dạng danh sách kề chứa maximum-flow
  - Mục đích: tìm ra đồ thị maximum-flow
- Function **find_maximum_flow()**:
  - Input: x
  - Output: maximum-flow

- Mục đích: tìm maximum-flow của đồ thị

- Function **find_minimum_cut()**:
  - Input: maximum, x
  - Output: cut_edges
  - Mục đích: tìm minimum-cut
- Function **remove_cut_edges()**:
  - Input: graph, cut_edges
    - graph: đồ thị gốc
    - cut_edges: các trường hợp trong cut_edges
  - Output: new_graph
    - new_graph: đồ thị mới đã loại bỏ các cut_edges
  - Mục đích: trả về đồ thị đã xóa đi cut_edges
- Function **bfs()**:
  - Input: graph, source, sink
    - graph: đồ thị từ kết quả trả về của remove_cut_edges()
    - source: nguồn
    - sink: đích
  - Output: True or False
    - True: có đường đi từ source đến sink
    - False: không có đường đi từ source đến sink
  - Mục đích: kiểm tra xem đồ thị mới có đáp ứng định nghĩa minimum-cut không

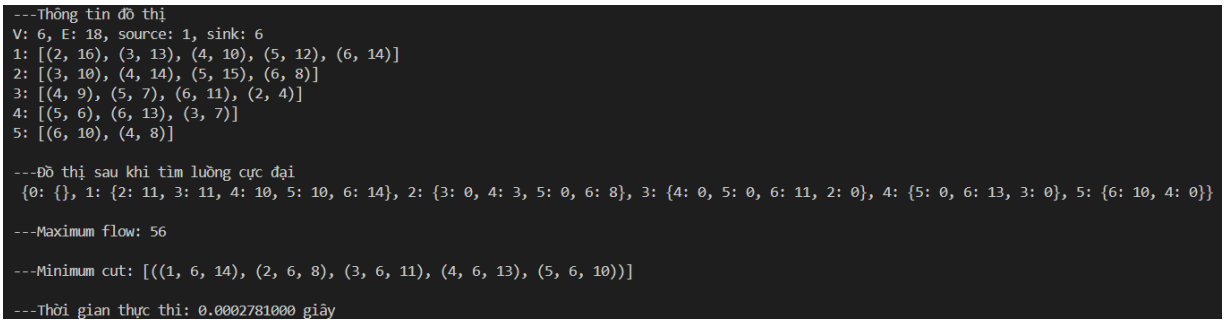
2.4.2 So sánh kết quả

- Trường hợp 1: Đồ thị có số đỉnh nhỏ và cạnh thưa:



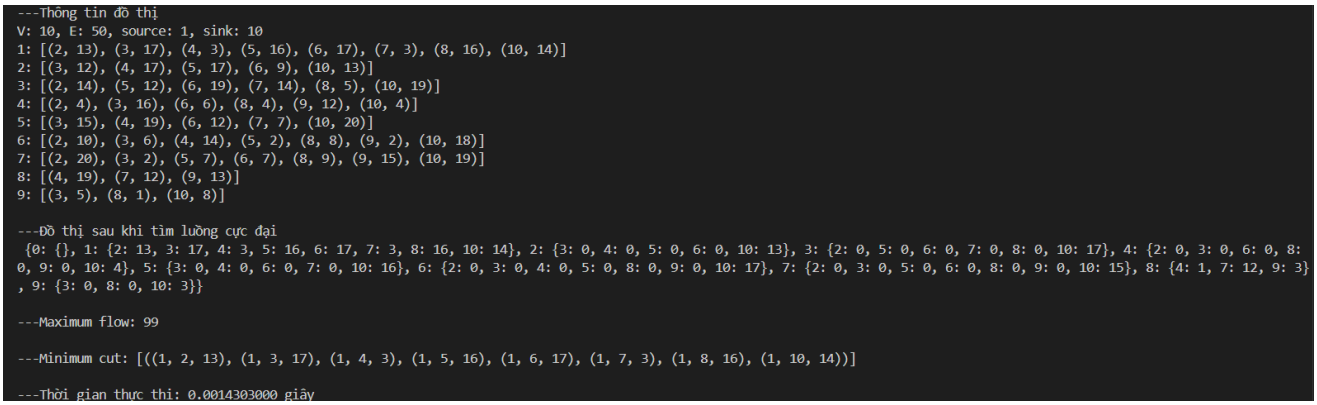
Hình 11. Kết quả khi chạy trên máy của ví dụ 1

- **Trường hợp 2**: Đồ thị có số đỉnh nhỏ và cạnh dày:



Hình 12. Kết quả khi chạy trên máy của ví dụ 2

- **Trường hợp 3**: Đồ thị có số đỉnh lớn hơn và cạnh dày:



Hình 13. Kết quả khi chạy trên máy của ví dụ 3

- **Đánh giá chạy thực nghiệm các trường hợp**:
  - Trường hợp 1: thời gian chạy 0.0001721000 giây
  - Trường hợp 2: thời gian chạy 0.0002781000 giây
  - Trường hợp 3: thời gian chạy 0.0014303000 giây
- **Kết luận:**
  - Dựa vào trường hợp 1 và 2 ta có thể thấy thời gian thực thi thuật toán phụ thuộc vào độ dày của đồ thị.
  - Trường hợp 3 là để chứng minh rõ hơn cho việc thời gian thực thi của thuật toán là dựa vào độ dày của đồ thị, số đỉnh thay đổi từ 6 lên 10 (rất nhỏ) và số cạnh thay đổi từ 7 lên 50 (lớn) và khi đó thời gian thực thi của trường hợp 3 lớn hơn gần như là 10 lần trường hợp 1.

2.4.3 Phân tích kết quả

- **Kết luận**:
  - Dựa trên kết quả chạy tay, ta thấy:
    - Example 1: có label chứa dấu '-', chứng tỏ trong example 1 có backward edges.
    - Example 2, 3: tất cả nhãn đều dấu '+'.

- Dựa vào kết quả code, ta thấy: khi ta dùng shortest-augmenting-path để tìm minimum-cut thì:
  - Example 1,3: chỉ có 1 cut phù hợp là minimum-cut.
  - Example 2: có 2 cut phù hợp là minimum-cut.

2.5 Độ phức tạp bài toán

- Thành phần chính ảnh hưởng đến độ phức tạp là vòng lặp while:

- Bài toán thực hiện gán nhãn bằng BFS, và BFS có độ phức tạp:

O(V+E)

  - O(V): cho việc xử lý các đỉnh.
  - O(E): cho việc xử lý các cạnh.

- Bài toán thực hiện lặp quá các cạnh ngược để kiểm tra luồng ngược, việc này tốn O(E).
- Bài toán cập nhật luồng tăng cường sau khi tìm được đường tăng cường từ nguồn đến đích có độ phức tạp tỉ lệ thuận với đỉnh, tối đa là O(V).
- Vòng lặp while (vòng lặp chính) phụ thuộc vào giá trị maximum-flow của bài toán, mỗi vòng lặp sẽ đẩy ít nhất 1 đơn vị luồng, nên số lần lặp tối đa là O(F).
- Do đó, độ phức tạp tổng quát là:

O(F×(V+E))

CHƯƠNG 3 - EXERCISE

3.1 Câu 1 :

Biểu diễn mạng bằng ma trận kề và tìm source và sink

3.1.1 Vấn đề bài toán :

- Ban đầu bài toán được biểu diễn dưới dạng danh sách kề, và vấn đề chúng ta là thực hiện chuyển cách biểu diễn sang ma trận kề.

- Một vấn đề sau khi chúng ta tìm được ma trận kề đó là chúng ta phải xác định được **source và sink**.

3.1.2 Hướng giải quyết bài toán s :

- Để chuyển từ danh sách kề sang ma trận kề thì với một cạnh có hướng từ đỉnh i đến đỉnh j có dung lượng u_{ij}, thì phần tử trong hàng thứ i và cột thứ j được đặt thành uij, và phần tử trong hàng thứ j và cột thứ i được đặt thành $-u_{ij}$; nếu không có cạnh nào giữa các đỉnh i và j, thì cả hai phần tử này đều được đặt thành 0.

- Để xác định được source và sink thì với định nghĩa của source và sink ta có :

  - **Source** : Đây là nơi vật chất bắt đầu di chuyển vào mạng lưới, **không có gì "chảy vào" nguồn**.
  - **Sink** : Đây là nơi vật chất thoát ra khỏi mạng lưới, **không có gì "chảy ra" từ đích.**

- Do đó ý tưởng tìm souce và sink đó là tính tổng dòng chảy vào và dòng chảy ra của từng đỉnh trong mạng, trong danh sách dòng chảy vào ta sẽ tìm được source và trong danh sách dòng chảy ra ta sẽ tìm được sink.

3.1.3 Thuật toán để giải quyết bài toán :

- Mã giả:

Function adjacency_matrix() :

Input: danh sách kề của mạng.

Output : ma trận kề, source và sink.

```
for each i ∈ V(G)
        for j, uij ∈ V(G)[i]
                do am[i][j] ← uij
                   am[j][i] ← -uij
for i ← 0 to length(row(am))
        for j ← 0 to length(column(am))
```

do out_flow[i] ← out_flow[i] + max(0, am[i][j])

in_flow[i] ← in_flow[i] + max(0, am[j][i])

sink ← (index(0) in out_flow) + 1

source ← (index(0) in in_flow) + 1

return am, source, sink

- Ví dụ:

- Danh sách kề ban đầu:

```
---Thông tin đồ thị
V: 6, E: 7, source: 1, sink: 6
1: [(2, 2), (4, 3)]
2: [(3, 5), (5, 3)]
3: [(6, 2)]
4: [(3, 1)]
5: [(6, 4)]
```

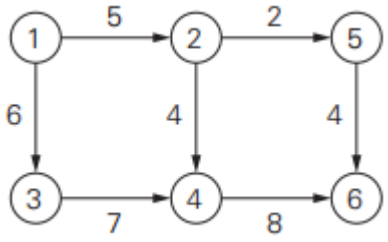
- Ma trận kề, source và sink :

```
---Ma trận kề
 [[ 0.  2.  0.  3.  0.  0.]
 [-2.  0.  5.  0.  3.  0.]
 [ 0. -5.  0. -1.  0.  2.]
 [-3.  0.  1.  0.  0.  0.]
 [ 0. -3.  0.  0.  0.  4.]
 [ 0.  0. -2.  0. -4.  0.]]

---Source = 1, Sink = 6
```

3.2 Câu 2:

Dùng thuật toán shortest-augmenting path để tìm maximum-flow và minimum-cut

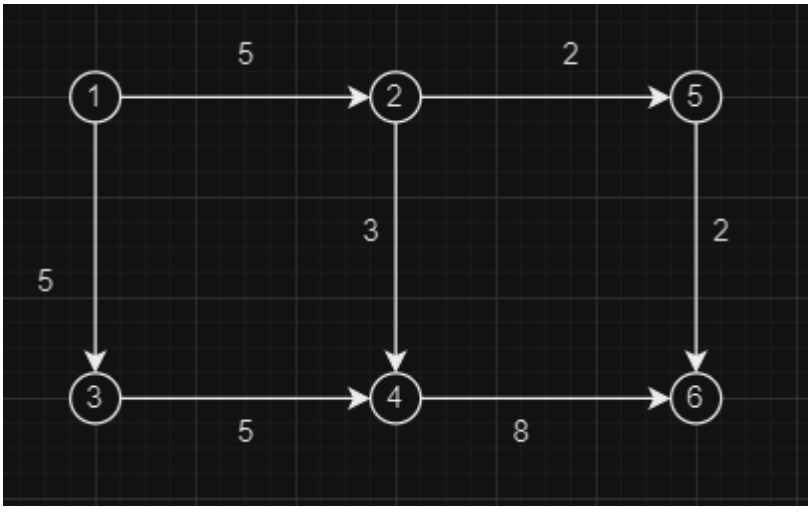
3.2.1 Question a:



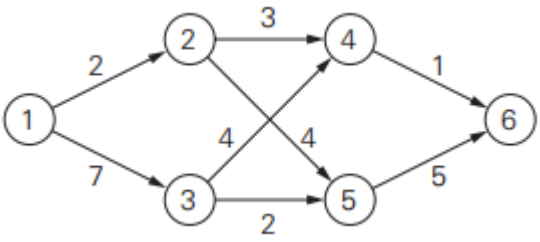
- Kết quả sau khi chạy thuật toán:

```
---Thông tin đồ thị
V: 6, E: 7, source: 1, sink: 6
1: [(2, 5), (3, 6)]
2: [(4, 4), (5, 2)]
3: [(4, 7)]
4: [(6, 8)]
5: [(6, 4)]

---Đồ thị sau khi tìm luồng cực đại
 {0: {}, 1: {2: 5, 3: 5}, 2: {4: 3, 5: 2}, 3: {4: 5}, 4: {6: 8}, 5: {6: 2}}

---Maximum flow: 10

---Minimum cut: [((2, 5, 2), (4, 6, 8))]
```

- Đồ thị sau khi tìm được luồng cực đại:



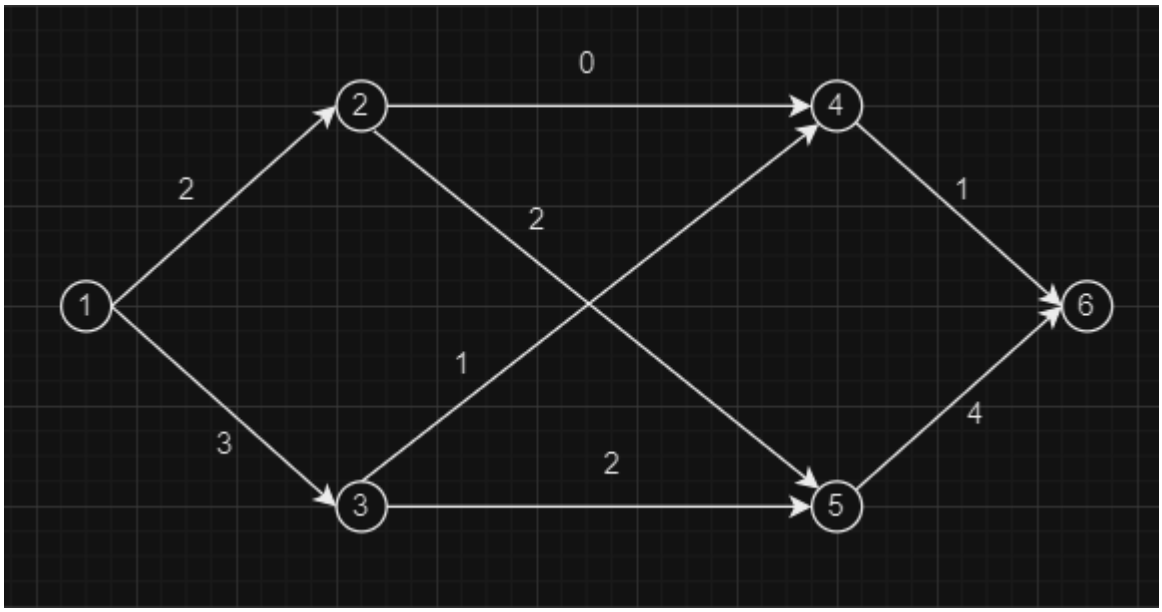
3.2.2 Question b:



- Kết quả sau khi chạy thuật toán:

```
---Thông tin đồ thị
V: 6, E: 8, source: 1, sink: 6
1: [(2, 2), (3, 7)]
2: [(4, 3), (5, 4)]
3: [(4, 4), (5, 2)]
4: [(6, 1)]
5: [(6, 5)]

---Đồ thị sau khi tìm luồng cực đại
 {0: {}, 1: {2: 2, 3: 3}, 2: {4: 0, 5: 2}, 3: {4: 1, 5: 2}, 4: {6: 1}, 5: {6: 4}}

---Maximum flow: 5

---Minimum cut: [(1, 2, 2), (3, 5, 2), (4, 6, 1)]
```

- Đồ thị sau khi tìm được luồng cực đại:



3.3 Câu 4:

3.3.1 Question a:

- Bài toán yêu cầu ta tìm cách chuyển đổi bài toán luồng cực đại (maximum flow) trên một mạng có nhiều nguồn (sources) và nhiều đích (sinks) thành bài toán tương tự nhưng chỉ có một nguồn duy nhất (single source) và một đích duy nhất (single sink).

- Giải pháp để giải bài toán này là ta thêm vào mạng một nguồn ảo (S*) và một đích ảo (T*) để bài toán trở thành tìm luồng cực đại từ S* đến T*. Sau đó áp dụng thuật toán **Edmonds-Karp** để giải bài toán như bình thường.

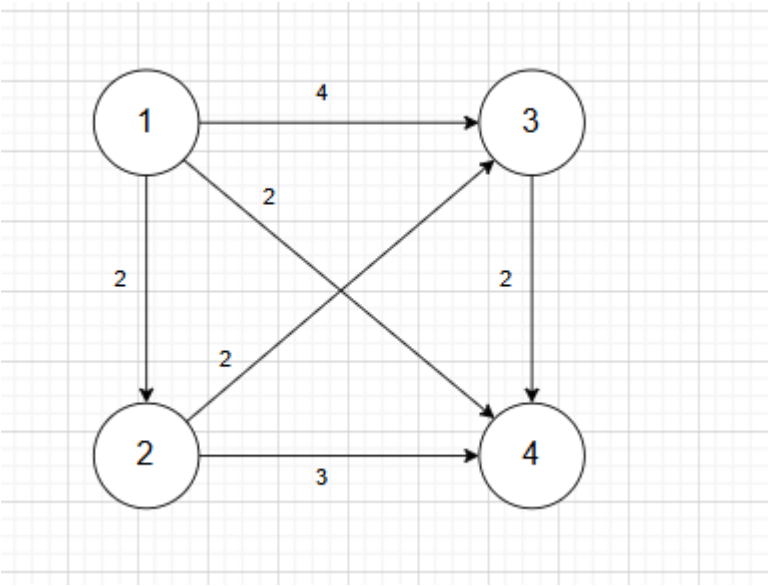
- Các bước thực hiện như sau:

- Thêm một nguồn ảo (S*):
    - Gọi tập các nguồn ban đầu là S = {s_1, s_2, …, s_k}
    - Thêm một đỉnh nguồn ảo S*

- Nối S* với từng đỉnh nguồn s_i bằng các cạnh có sức chứa $c(S^*, s_i)$. Sức chứa của các cạnh này thường được đặt là vô hạn hoặc bằng tổng lưu lượng tối đa đi qua các nguồn.

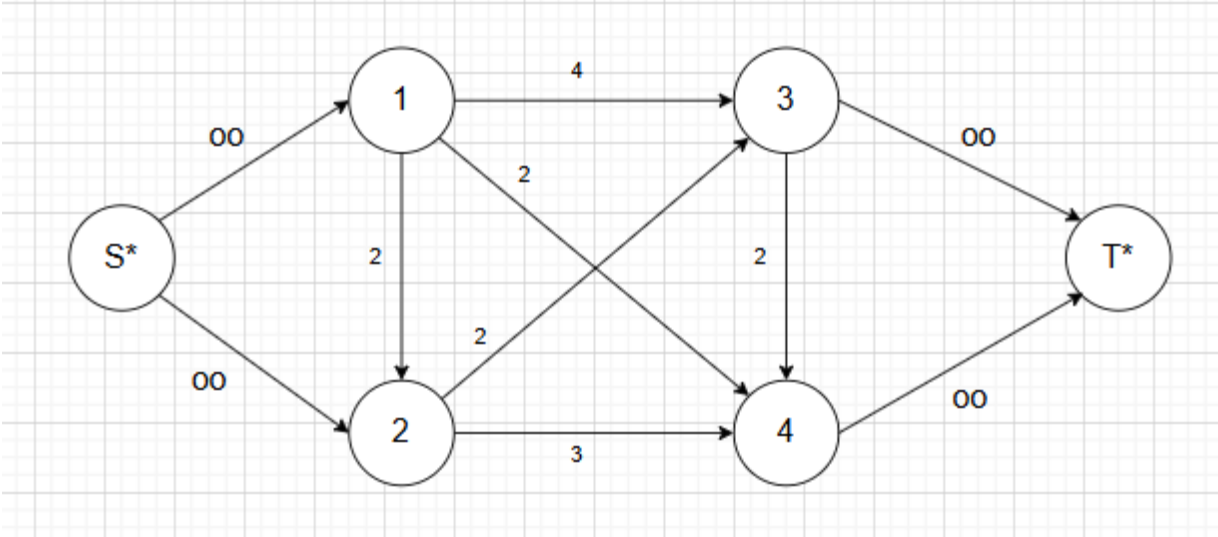
- Thêm một đích ảo (T*):
  - Gọi tập các nguồn ban đầu là $T = \{t_1, t_2, \ldots, t_m\}$.
  - Thêm một đỉnh nguồn ảo T*
  - Nối từng đỉnh đích t_j tới T* bằng các cạnh có sức chứa $c(t_j, T^*)$. Sức chứa của các cạnh này thường được đặt là vô hạn hoặc bằng tổng lưu lượng tối đa đi qua các đích.
- Giải bài toán Maximum Flow trên mạng mới :
  - Với mạng lưới đã chuyển đổi, bài toán trở thành tìm luồng cực đại từ S* đến T*.
  - Áp dụng thuật toán **Edmonds-Karp**

- Ví dụ:



Hình 7. Đồ thị ban đầu của bài toán

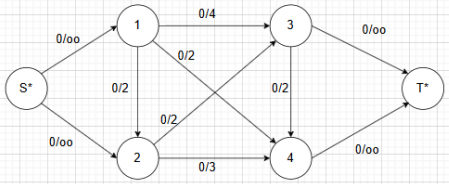
Đồ thị ban đầu có 2 đỉnh nguồn (1, 2) và 2 đỉnh đích (3, 4). Có các cạnh và trọng số tương ứng.



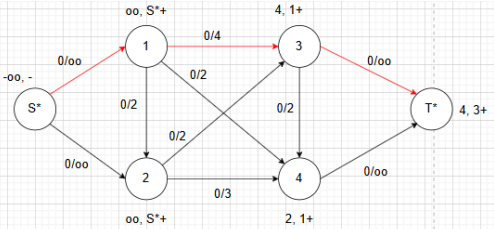
Hình 8. Bài toán sau khi được mô hình hóa

- Các bước giải của thuật toán như sau:

step 1

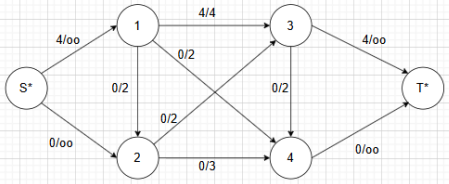


Queue: S*, 1, 2, 3, 4, T*

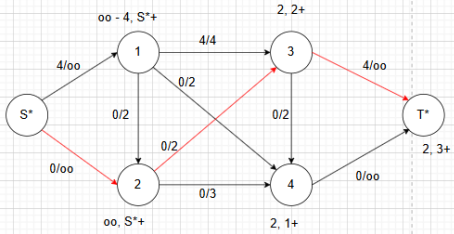


Augment the flow by 4 (the sink's first label) along the path S* -> 1 -> 3 -> T*.

step 2

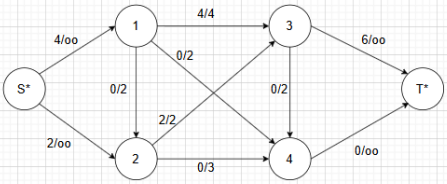


Queue: S*, 1, 2, 3, 4, T*

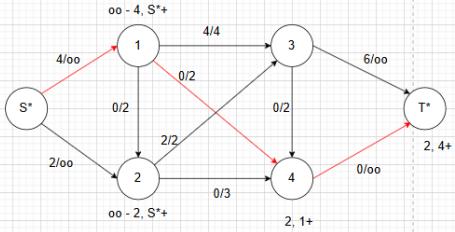


Augment the flow by 2 (the sink's first label) along the path S* -> 2 -> 3 -> T*.

step 3

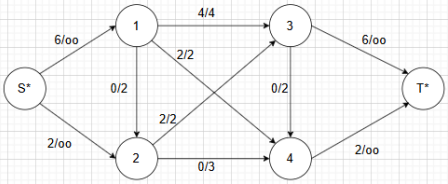


Queue: S*, 1, 2, 4, T*

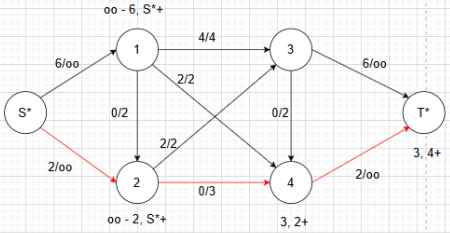


Augment the flow by 2 (the sink's first label) along the path S* -> 1 -> 4 -> T*.

step 4

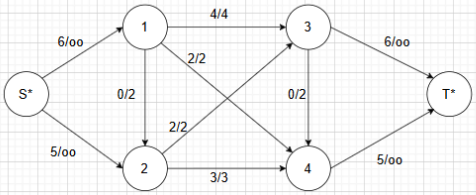


Queue: S*, 1, 2, 4, T*

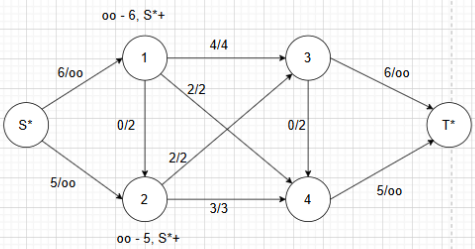


Augment the flow by 3 (the sink's first label) along the path S* -> 2 -> 4 -> T*.

step 5



Queue: S*, 1, 2



No augmenting path (the sink is unlabeled);
the current flow is maximal

3.3.2 Question b:

- Bài toán yêu cầu ta giải thích cách biến đổi bài toán luồng cực đại (maximum-flow problem) trong mạng lưới có ràng buộc về dung lượng trên các đỉnh trung gian (vertex capacity constraints) thành bài toán luồng cực đại chỉ có ràng buộc trên các cạnh (edge capacity constraints).

- Việc này có thể được giải quyết bằng cách thay thế mỗi đỉnh có ràng buộc dung lượng bằng một "đỉnh phân tách" gồm hai nút được kết nối bởi một cạnh.

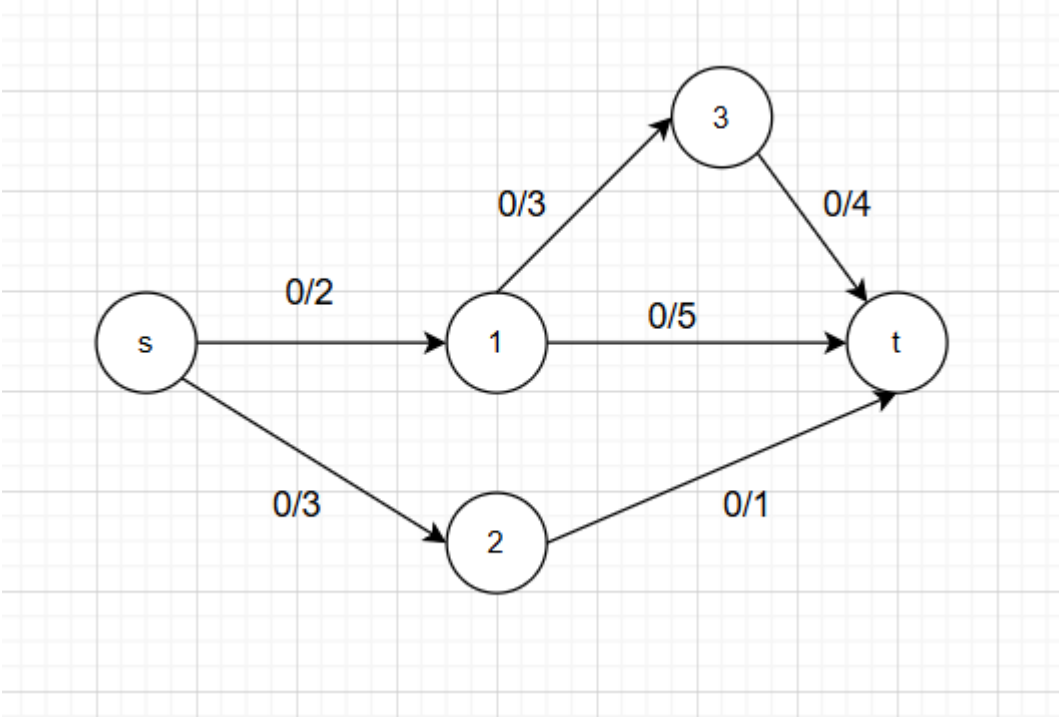
- Các bước làm như sau:

- Phân tách mỗi đỉnh trung gian thành 2 nút:
    - Với mỗi đỉnh v trong mạng, ta thay nó bằng hai nút mới v_{in} và v_{out}.
    - Kết nối v_{in} với v_{out}. bằng một cạnh có dung lượng bằng dung lượng của đỉnh v ban đầu.
- Điều chỉnh các cạnh đầu vào và đầu ra:
    - Với mỗi cạnh $u \to v$ ban đầu: Thay bằng cạnh $u \to v_{in}$ cùng dung lượng ban đầu.
    - Với mỗi cạnh $v \to w$ ban đầu: Thay bằng cạnh $v_{out} \to w$ cùng với dung lượng ban đầu.

- Giữ nguyên các đỉnh nguồn và đích:

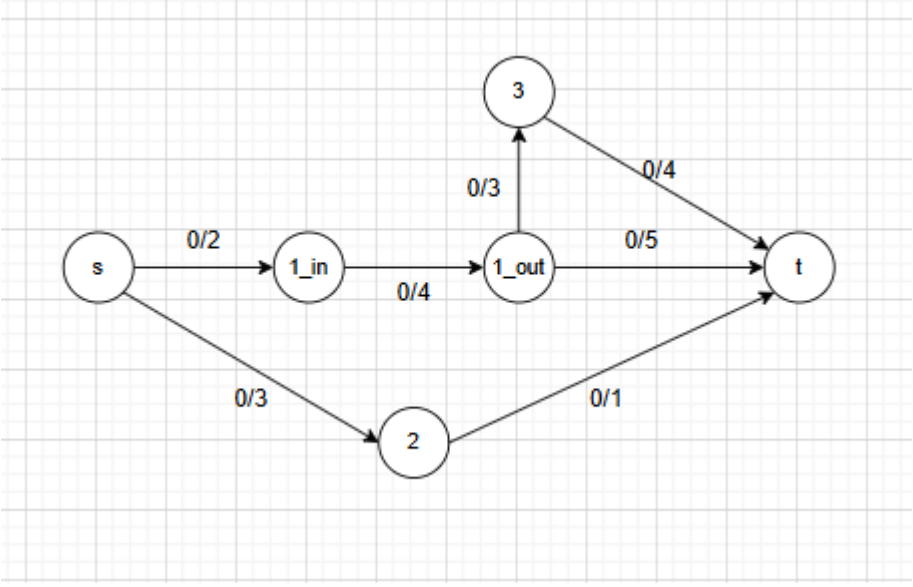
Nếu v là đỉnh nguồn (source) hoặc đỉnh đích (sink), thì không cần thực hiện phân tách. Các ràng buộc dung lượng chỉ áp dụng cho các đỉnh trung gian.

- Ví dụ:



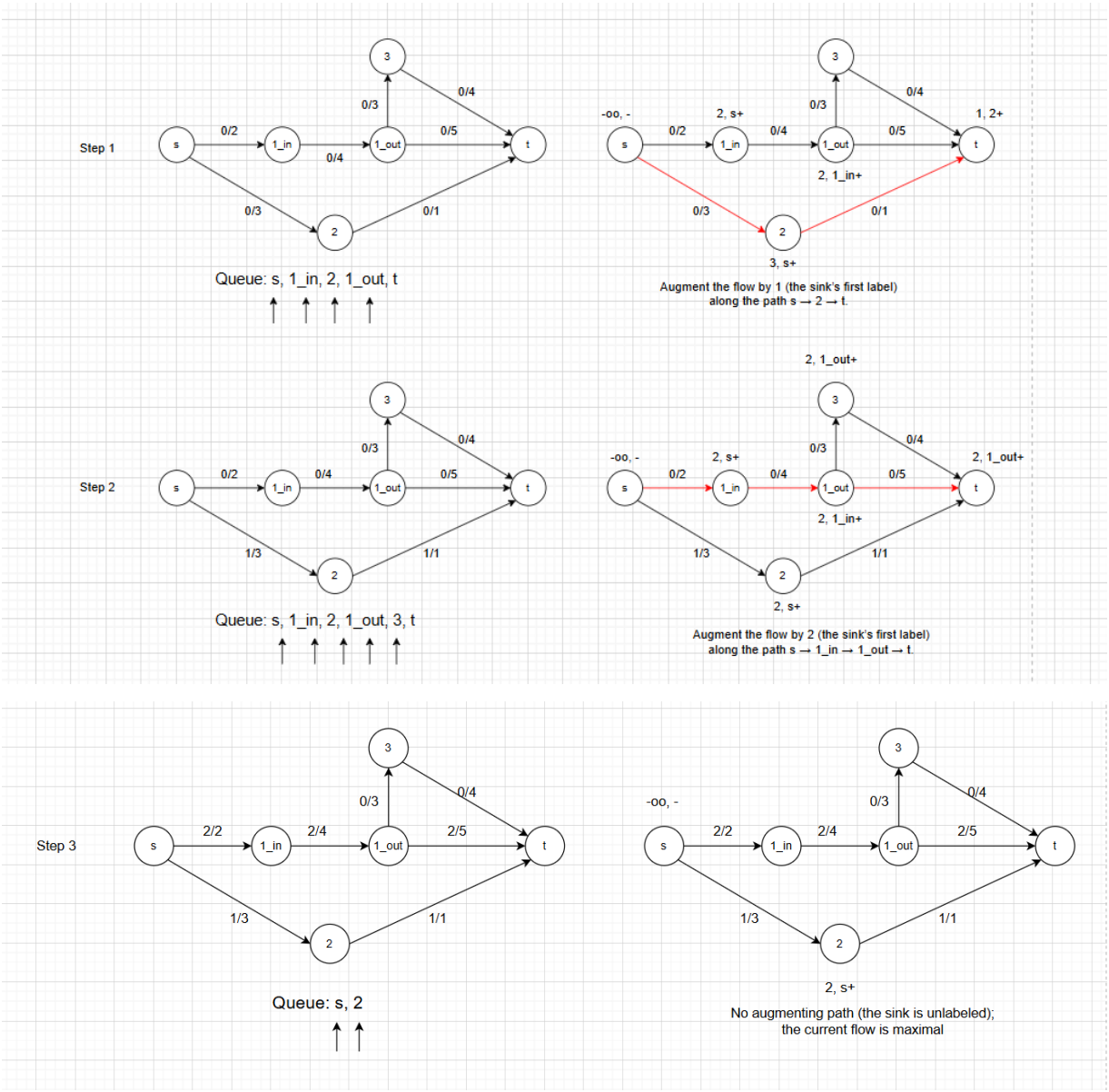
Hình 9. Mạng ban đầu

Giả sử rằng, ta có ràng buộc trên đỉnh sau: đỉnh 1 có ràng buộc là 4



Hình 10. Mạng sau khi sử dụng phương pháp chia đỉnh

Sau khi dùng kỹ thuật tách đỉnh, ta được mạng như Hình 10. Lúc này bài toán trở thành bài toán luồng cực đại thông thường. Ta áp dụng thuật toán Edmonds-Karp để giải bài toán.



3.4 Câu 7:

3.4.1 Question a:

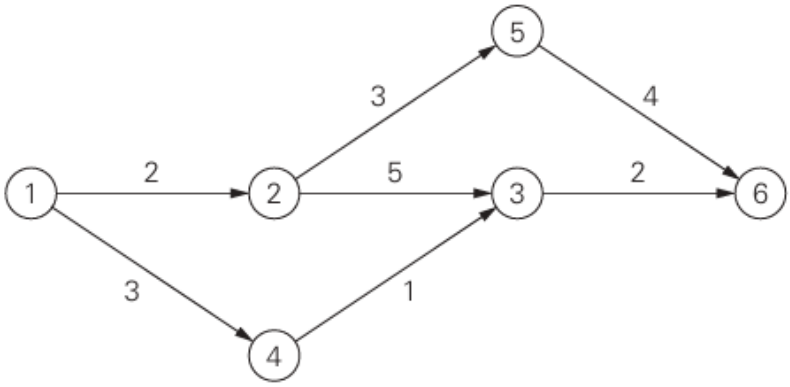


FIGURE 10.4 Example of a network graph. The vertex numbers are vertex "names"; the edge numbers are edge capacities.

Hình 11. Mạng bài toán yêu cầu

- Để biểu diễn bài toán Maximum-Flow cho mạng trong hình 11 thành dạng bài toán quy hoạch tuyến tính. Ta thực hiện như sau:

1. ***Định nghĩa biến****:* x_{ij} *là dòng chảy trên cạnh từ đỉnh* i *đến đỉnh* j *(Ví dụ* x_{12} *là dòng chảy từ đỉnh 1 đến đỉnh 2).*
2. ***Hàm mục tiêu****: Maximize* $v = \sum_{j:(1,j)\in E} x_{1j}$
3. ***Ràng buộc chung****:*
    a) *Ràng buộc về bảo toàn dòng chảy tại mỗi đỉnh:*
        - Với mỗi đỉnh k (trừ đỉnh nguồn và đích):

$$\sum_{j:(j,i)\in E} x_{ji} = \sum_{j:(i,j)\in E} x_{ij} \text{ for } i = 2,3,\dots,n-1 \ (1)$$

        - Biểu thức (1) tương đương với:
            - Tại đỉnh 2: $x_{12} = x_{23} + x_{25}$
            - Tại đỉnh 3: $x_{23} + x_{43} = x_{36}$
            - Tại đỉnh 4: $x_{14} = x_{43}$

- Tại đỉnh 5: $x_{25} = x_{56}$

b) Ràng buộc về giới hạn dung lượng trên cạnh:

- Với mỗi cạnh từ $i \rightarrow j$:

$0 \leq x_{ij} \leq u_{ij}$, trong đó u_{ij} là dung lượng của cạnh

➢ Từ những điều kiện trên ta có dạng tổng quát của bài toán quy hoạch tuyến tính:

$$Maximize \ v = x_{12} + x_{14}$$

$$\begin{cases} x_{12} \leq 2 \\ x_{14} \leq 1 \\ x_{23} \leq 2 \\ x_{25} \leq 2 \\ x_{43} \leq 1 \\ x_{56} \leq 2 \\ x_{36} \leq 2 \\ x_{23} + x_{25} \leq 2 \\ x_{23} + x_{43} \leq 2 \end{cases}$$

3.4.2 Question b:

- Dạng chuẩn tắc của bài toán:

$$Maximize \ v = x_{12} + x_{14}$$

$$\begin{cases} x_{12} + s_1 = 2 \\ x_{14} + s_2 = 1 \\ x_{23} + s_3 = 2 \\ x_{25} + s_4 = 2 \\ x_{43} + s_5 = 1 \\ x_{56} + s_6 = 2 \\ x_{36} + s_7 = 2 \\ x_{23} + x_{25} + s_8 = 2 \\ x_{23} + x_{43} + s_9 = 2 \end{cases}$$

Quy trình giải quyết bài toán

- Công thức $z_j = \sum_{i=1}^{m} c_i . a_{ij}$
  - z_j: giá trị tổng chi phí thực tế cho hàm mục tiêu.

- c_i: hệ số của biến cơ sở.
- a_{ij}: hệ số của biến x_j trong ràng buộc của hàng thứ i.
- m: số ráng buộc.

- $c_j - z_j$: hệ số cải thiện:
  - c_j: hệ số của biến x_j trong hàm mục tiêu.
  - z_j: giá trị chi phí thực tế
- Ý nghĩa của $c_j - z_j$:
  - Đánh giá tiềm năng cải thiện của biến x_j:
    - $c_j - z_j > 0$: đưa x_j vào cơ sở sẽ làm tăng giá trị của hàm mục tiêu.
    - $c_j - z_j = 0$: đưa x_j vào cơ sở sẽ không làm thay đổi giá trị của hàm mục tiêu.
    - $c_j - z_j < 0$: đưa x_j vào cơ sở sẽ làm giảm giá trị của hàm mục tiêu.
  - Điều kiện dừng:
    - Trong bài toán tối ưu hóa thì thuật toán dừng khi $c_j - z_j \leq 0$.
    - Trong bài toán tối thiểu hóa thì thuật toán dừng khi $c_j - z_j > 0$.
- Pivot: quá trình thay thế một biến cơ sở bằng một biến không cơ sở trong bảng đơn hình, sao cho hàm mục tiêu được cải thiện hoặc không thay đổi.
  - Chọn biến vào cơ sở (cột pivot):
    - Chọn biến có hệ số $c_j - z_j$ lớn hoặc nhỏ phụ thuộc vào yêu cầu hàm mục tiêu.
  - Chọn biến khỏi hàng cơ sở (hàng pivot):
    - Dựa vào các giá trị của cột hệ số tự do b, chọn hàng sao cho giá trị $\frac{b_i}{a_{ij}}$ là nhỏ nhất.
  - Cập nhật lại bảng đơn hình:

- Hàng pivot được cập nhật bằng công thức: $R_{pnew} = R_{pold}/x_i$
- Các hàng khác được cập nhật bằng: $R_{new} = R_{old} - x_i . R_{pnew}$

Giải quyết bài toán:

- Bài toán yêu cầu ta giải bằng phương pháp đơn hình (simplex)

  $c_j = (1,1,0,0,0,0,0, 0,0,0,0,0,0,0,0)$ (hệ số của hàm mục tiêu)

  $c_b = (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9) = (0,0,0,0,0,0,0,0,0)$ (hệ số của hàm mục tiêu nhưng với các biến cơ sở)

  Điều kiện dừng: $c_j - z_j \leq 0$

- Lập bảng đơn hình đầu tiên:
  - Tìm z_j và $c_j - z_j$, so sánh với điều kiện dừng.
  - Tìm pivot:
    - Cột pivot: chọn cột x_{12} *với giá trị* $c_j - z_j$ là lớn nhất.
    - Hàng pivot: chọn hàng đầu tiên, do $\frac{b_i}{a_{ij}} = \frac{2}{1} = 2$, những hàng khác có giá trị ∞.

		1	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	
Basis	c_b	x_{12}	x_{14}	x_{23}	x_{25}	x_{43}	x_{53}	x_{36}	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	*b*
s_1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2
s_2	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
s_3	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	2
s_4	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	2
s_5	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1
s_6	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	2
s_7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	2
s_8	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	2
s_9	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	2

	z_j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	$c_j - z_j$	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	> 0

- Cập nhật bảng đơn hình dựa trên pivot:
- Sau đó tiếp tục quá trình cho đến khi thỏa điều kiện dừng.

		1	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	
Basis	c_j	x_{12}	x_{14}	x_{23}	x_{25}	x_{43}	x_{53}	x_{36}	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	b
x_{12}	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2
s_2	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
s_3	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	2
s_4	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	2
s_5	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1
s_6	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	2
s_7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	2
s_8	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	2
s_9	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	2
	z_j	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2
	$c_j - z_j$	0	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	> 0

- Bảng đơn hình tiếp theo:

		1	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	
Basis	c_b	x_{12}	x_{14}	x_{23}	x_{25}	x_{43}	x_{53}	x_{36}	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	b
x_{12}	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2
x_{14}	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
s_3	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	2
s_4	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	2
s_5	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1

s_6	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	2
s_7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	2
s_8	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	2
s_9	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	2
	z_j	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	3
	$c_j - z_j$	0	0	0	0	0	0	0	-1	-1	0	0	0	0	0	0	0	≤ 0

- $c_j - z_j \leq 0$: thỏa điều kiện dừng:

Kết luận

- Nghiệm tối ưu và các biến:
  - $x_{12} = 2$, (vì x_{12} là biến cơ sở với $b = 2$).
  - $x_{14} = 1$.
- Giá trị tối ưu của hàm mục tiêu $v = \sum c_i . b_i = 2.1 + 1.1 = 3$

TÀI LIỆU THAM KHẢO

Tiếng Anh

1. Algorithm Design and Applications[A4]

2. A. Levitin. Introduction to the Design and Analysis of Algorithms, Addison Wesley, 3rd edition, 2011

BẢNG PHÂN CÔNG CÔNG VIỆC

Họ và tên	**Nhiệm vụ**	**Đánh giá**
Châu Nguyễn Khánh Trình – 52200005	Câu 1	100%
Ngô Phan Minh Trí - 52200171	Câu 2 – a, b, c d – 1, 2, 7b	100 %
Lê Như Đạt - 52200160	d – 4, 7a	100 %