VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING (CO1007)

Assignment

STOCHASTIC PROGRAMMING AND APPLICATIONS

Advisor: Mai Xuân Toàn
Students: Nguyễn Tấn Tài - 2212990.
Nguyễn Anh Khoa - 2211614.
Nguyễn Đăng Khoa - 2211618.
Hồ Tấn Phát - 2212507.
Trần Thế Đại Phát - 2212537.

HO CHI MINH CITY, DECEMBER 2023

Contents

1 Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Nguyễn Tấn Tài	2212990	- Code Problem 2 - Soạn thảo latex Problem 2	20%
2	Nguyễn Anh Khoa	2211614	- Code Problem 1	20%
3	Nguyễn Đăng Khoa	2211618	- Nội dung giai đoạn 2 Problem 2 - Soạn thảo latex Problem 1	20%
4	Hồ Tấn Phát	2212507	- Nội dung Problem 1	20%
5	Trần Thế Đại Phát	2212537	- Nội dung giai đoạn 1 Problem 2	20%

2 Report

2.1 Problem 1

Một công ty F sản xuất 8 sản phẩm, sử dụng 5 nguyên liệu mà công ty phải đặt trước từ bên thứ ba. Mỗi sản phẩm i cần một lượng a_{ij} nguyên liệu; với $i = 1, 2, \ldots, 8$, $j = 1, 2, \ldots, 5$ và $a_{ij} \geq 0$. Nhu cầu cho mỗi sản phẩm được biểu diễn bằng vector ngẫu nhiên $\mathbf{D} = (D_1, D_2, \ldots, D_8)$.

Trước khi biết được nhu cầu, công ty F phải đặt trước nguyên liệu từ bên thứ ba với chi phí là b_j cho mỗi nguyên liệu j. Sau khi nhu cầu được biết, công ty phải quyết định phương thức sản xuất sao cho không vượt quá nhu cầu của sản phẩm i. Giá để mua một sản phẩm bù vào phần thiếu so với nhu cầu là l_i, trong khi giá khi bán một sản phẩm là q_i. Nguyên liệu còn thừa trong kho sẽ được bán với giá $s_j < b_j$. Những sản phẩm vượt quá nhu cầu sẽ bị loại bỏ.

Bài toán được tóm tắt qua bảng sau:

Loại sản phẩm	P1	P2	P3	P4	P5	P6	P7	P8	Giá mua ($)	Giá bán($)
Nguyên liệu 1	5	2	9	6	7	7	2	6	7	5
Nguyên liệu 2	6	0	5	2	9	6	0	7	7	4
Nguyên liệu 3	7	5	2	2	2	3	1	3	7	5
Nguyên liệu 4	6	6	1	1	5	4	2	4	7	5
Nguyên liệu 5	6	4	1	1	5	2	8	4	6	5
Giá mua($)	50	20	40	20	12	15	25	50		
Giá bán($)	140	150	200	100	250	120	180	140		

2.2 Problem 2

Trong trường hợp thảm họa, việc sơ tán an toàn và kịp thời người dân khỏi khu vực bị ảnh hưởng là nhiệm vụ cấp bách. Mô hình lập trình tỉnh số hai giai đoạn (SLP) nổi bật với khả năng thiết kế kế hoạch sơ tán linh hoạt và hiệu quả, giúp đối phó với những biến động không thể dự đoán trước. Mục tiêu chính là cung cấp nơi trú ẩn và hỗ trợ cần thiết cho người dân, đồng thời giảm thiểu rủi ro và tổn thất.

Mô hình SLP bao gồm hai giai đoạn chính: Giai đoạn đầu tiên phát triển kế hoạch sơ tán tổng quát, trong khi giai đoạn thứ hai điều chỉnh kế hoạch dựa trên điều kiện cụ thể và thực tế của thảm họa. Sự kết hợp của hai giai đoạn này tạo nên một chiến lược sơ tán toàn diện, đảm bảo an toàn cho người dân và hiệu quả trong việc quản lý tình hình khẩn cấp.

Qua việc áp dụng mô hình này, chúng ta có thể nâng cao khả năng phản ứng và hiệu quả trong việc sơ tán, từ đó góp phần vào việc phục hồi nhanh chóng và bền vững sau thảm họa.

3 Probability

3.1 Problem 1

3.1.1 Giai đoạn 1

Trước khi biết được chính xác nhu cầu, chúng ta cần xác định số lượng nguyên liệu cần đặt trước $x_j = (x_1, x_2, \ldots, x_5)$ qua vấn đề tối ưu sau:

$$\begin{aligned} \min G(x) &= b^\top x + Q(x) \\ x &\geqslant 0 \end{aligned}$$

Với:

- $Q(x) = \mathbb{E}[Z]$ là giá trị tối ưu của vấn đề giai đoạn 2.
- b là ma trận cột giá để đặt 1 đơn vị sản phẩm.

3.1.2 Giai đoạn 2

Sau khi biết được chính xác nhu cầu trong các trường hợp có thể xảy ra, chúng ta có thể tìm được phương án sản xuất tối ưu cho công ty F bằng cách giải quyết hai vấn đề con sau:

- Vấn đề 1:

$$\begin{aligned} \min Z_1(z_1, y_1) &= c^T z_1 - s^T y_1 \\ \text{s.t.} \quad y_1 &= x - A^T z_1 \\ 0 \leqslant z_1 &\leqslant d_1,\ y_1 \geqslant 0 \end{aligned}$$

- Vấn đề 2:

$$\begin{aligned} \min Z_2(z_2, y_2) &= c^T z_2 - s^T y_2 \\ \text{s.t.} \quad y_2 &= x - A^T z_2 \\ 0 \leqslant z_2 &\leqslant d_2,\ y_2 \geqslant 0 \end{aligned}$$

Với:

- $c = (c_i := l_i - q_i)$ là hệ số chi phí.
- $A = [a_{ij}]$ là ma trận 8×5.
- s là ma trận cột giá bán của các nguyên liệu còn lại trong kho.
- x là ma trận cột biến quyết định ở giai đoạn 1.
- z_1 là ma trận cột số sản phẩm sản xuất ở trường hợp 1.
- y_1 là ma trận cột số nguyên liệu còn lại ở trường hợp 1.
- z_2 là ma trận cột số sản phẩm sản xuất ở trường hợp 2.

- y_2 là ma trận cột số nguyên liệu còn lại ở trường hợp 2.

Gọi Q_1, Q_2 lần lượt là giá trị tối ưu của Z_1, Z_2. Dễ thấy rằng Z_1, Z_2 phụ thuộc vào giá trị x của vấn đề giai đoạn 1, từ đó ta có $Q(x) = \mathbb{E}[Z] = \frac{1}{2}Q_1 + \frac{1}{2}Q_2$. $\mathbb{E}[Z]$ được gọi là giá trị tối ưu của vấn đề giai đoạn 2 này.

3.2 Problem 2

3.2.1 Ký hiệu:

- V: tập đỉnh
- A: tập cạnh
- i, j: chỉ số đỉnh, $i, j \in V$
- s: chỉ số ngữ cảnh (biến cố)
- S: số lượng ngữ cảnh (biến cố)
- v: giá trị lượng đi ra từ đỉnh phát
- T': ngưỡng thời gian trước và sau khi thảm họa xảy ra
- T: số khoảng thời gian được xét từ trước đến sau khi thảm họa xảy ra
- u_{ij}: khả năng thông qua của cạnh (i, j)
- $u_{ij}^s(t)$: khả năng thông qua của cạnh (i, j) trong ngữ cảnh s tại thời điểm t
- $c_{ij}^s(t)$: thời gian di chuyển trên cạnh (i, j) trong ngữ cảnh s tại thời điểm t
- μ_s: xác suất xảy ra ngữ cảnh s

3.2.2 Biến quyết định:

- x_{ij}: luồng bên (i, j)
- $y_{ij}^s(t)$: luồng trên cạnh (i, j) trong ngữ cảnh s tại thời điểm t

3.2.3 Giai đoạn 1: Quyết định những kế hoạch giải cứu trước khi thảm họa xảy ra

Điều kiện cân bằng lượng:

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = \begin{cases} u, & \text{if } i = s \\ -u, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases}$$

Điều kiện giới hạn lượng thông qua:

$$0 \le x_{ij} \le u_{ij}, \quad \forall (i, j) \in A$$

Lưu ý khác, rằng buộc cân bằng lượng có thể tạo ra một đường đi với các vòng lặp và hành trình con nếu có các vòng lặp tiềm ẩn trong mạng lưới sơ tán. Để loại bỏ các vòng lặp trên đường đi so tán vật lý được tạo ra, hàm phát kết nối p_{ij}, $(i,j) \in A$ được giới thiệu đặc biệt. Với xem xét này, hàm phát có thể được định nghĩa như sau:

$$f(X) = p^T X = \sum_{(x,j)\in A} p_{ij} x_{ij}$$

Với:

- p_{ij}, $(i,j) \in A$ là giá trị phạt trên cạnh (i,j) nghĩa là khi lượng đi qua cạnh (i,j) sẽ bị phạt giá trị là p_{ij}.
- $X = \{x_{ij} | (i,j) \in A\}$ là vector lượng.

3.2.4 Giai đoạn 2: Điều chỉnh các quyết định được đưa ra ở giai đoạn 1 sau khi nhận được thông tin theo thời gian thực

Những người bị ảnh hưởng sẽ sơ tán theo kế hoạch đã được đề ra, trước khoảng thời gian T. Sau khoảng thời gian T người bị ảnh hưởng sẽ nhận được thông tin theo thời gian thực để điều chỉnh đường đi hợp lý. Dựa theo điều kiện trên ta có các ràng buộc với mỗi kịch bản xảy ra như sau:

$$\sum_{t\le T} y_{ij}^s(t) = x_{ij}, \quad (i,j) \in A, s = 1,2,\ldots,S$$

Ràng buộc trên mô tả mối liên hệ giữa những liên kết và khung thời gian sơ tán. Nếu có một lưu lượng giao thông trên liên kết (i, j) (ở đây là x_{ij}) thì lưu lượng giao thông trên liên kết (i,j) trước khoảng thời gian T cũng là x_{ij} với mỗi kịch bản mà ta biết.

Kế tiếp, ta đi xây dựng hàm mục tiêu:

$$Q(Y,s) = \min \sum_{(i,j)\in A_s} C_{ij}^s(t) \cdot y_{ij}^s(t)$$

Với:

- $Y_{ij}^s(t)$: là luồng giao thông tại thời điểm t trong kịch bản s.
- $C_{ij}^s(t)$: là thời gian di chuyển giữa (i, j) tại thời điểm t trong kịch bản s.

Hàm mục tiêu phụ thuộc vào nhữn điều kiện sau:

$$\sum_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum_{(j_t',i_t)\in A_s} y_{ji}^s(t') = d_i, \forall i \in V \quad \text{(Điều kiện cân bằng lượng)}$$

$$t \in \{1,2,\ldots,T\}; \quad s = 1,2,\ldots,S$$

$$0 \le y_{ij} \le u_{ij} \quad \forall (i,j) \in A \quad \text{(Điều kiện về khả năng thông qua)}$$

$$\sum_{t\le T} y_{ij}^s(t) = x_{ij}, \quad (i,j) \in A, s = 1,2,\ldots,S$$

Mô hình hai giai đoạn của kế hoạch sơ tán là:

$$
\begin{cases}
\min \sum\limits_{(i,j)\in A} p_{ij}x_{ij} + \sum\limits_{s=1}^{S}(\mu_s \cdot Q(Y,s)) \\
\text{s.t.} \\
\sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i \in V \\
0 \le x_{ij} \le u_{ij}, \quad \forall (i,j) \in A \\
\text{in which,} \\
Q(Y,s) = \min \sum\limits_{(i,j)\in A_s} c_{ij}^s(t) \cdot y_{ij}^s(t) \\
\text{s.t.} \\
\sum\limits_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum\limits_{(j'_t,i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \quad \forall i \in V, \\
t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\
0 \le y_{ij}^s(t) \le u_{ij}, \quad \forall (i,j) \in A, \quad t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\
\sum\limits_{t\le T} y_{ij}^s(t) = x_{ij}, \quad (i,j) \in A, \quad s = 1,2,\ldots,S
\end{cases} \tag{1}
$$

Với μ_s là xác suất khả năng xảy ra kịch bản s

Mô hình trên là một mô hình phụ thuộc vào thời gian và cũng là mô hình quy hoạch ngẫu nhiên hai giai đoạn. Mục tiêu của nó là tìm ra một kế hoạch tối ưu nhất để đưa ra những hướng dẫn cho người bị ảnh hưởng trong những sự kiện khẩn cấp. Mô hình trên tương đương với mô hình sau:

$$
\begin{cases}
\min \sum\limits_{(i,j)\in A} p_{ij}x_{ij} + \sum\limits_{s=1}^{S}\left(\mu_s \cdot \sum\limits_{(i,j)\in A_s} c_{ij}^s(t) \cdot y_{ij}^s(t)\right) \\
\text{s.t.} \\
\sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i \in V \\
0 \le x_{ij} \le u_{ij}, \quad \forall (i,j) \in A \\
\sum\limits_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum\limits_{(j'_t,i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \quad \forall i \in V, \\
t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\
0 \le y_{ij}^s(t) \le u_{ij}, \quad \forall (i,j) \in A, \quad t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\
\sum\limits_{t\le T} y_{ij}^s(t) = x_{ij}, \quad (i,j) \in A, \quad s = 1,2,\ldots,S
\end{cases}
$$

Thuật toán giải quyết: Ở mô hình trên, ta có thể thấy rằng có những ràng buộc (*). Những ràng buộc này mô tả mối quan hệ giữa sự lựa chọn liên kết (i, j) và kịch bản tương ứng phụ thuộc vào khung thời gian gây khó khăn khi giải quyết. Vì vậy ta sẽ sử dụng phương pháp Lagrange

Relaxation để giản lượt những ràng buộc này đưa vào hàm mục tiêu.

$$\sum_{s=1}^{S} \sum_{(i,j)\in A} \alpha_{ij}^{s}(t) \left(\sum_{t\leq T} y_{ij}^{s}(t) - x_{ij} \right)$$

Với $\alpha_{ij}^{s}(t)$ là nhân tử Lagrange tại thời điểm t trong thời gian s của liên kết (i, j).
Sau khi ta giản lượt biểu thức trên đưa vào hàm mục tiêu ta được mô hình giản lượt sau:

$$\begin{cases} \min \sum\limits_{(i,j)\in A} p_{ij}x_{ij} + \sum\limits_{s=1}^{S} \left(\mu_s \cdot \sum\limits_{(i,j)\in A_s} c_{ij}^{s}(t) \cdot y_{ij}^{s}(t) \right) + \sum\limits_{s=1}^{S} \sum\limits_{t\leq T} \sum\limits_{(i,j)\in A} \alpha_{ij}^{s}(t) \left(y_{ij}^{s}(t) - x_{ij} \right) \\ \text{s.t.} \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \quad \forall (i,j) \in A \\ \sum\limits_{(i_t,j_t)\in A_s} y_{ij}^{s}(t) - \sum\limits_{(j'_t,i_t)\in A_s} y_{ji}^{s}(t') = d_i^{s}(t), \quad \forall i \in V, \\ t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\ 0 \leq y_{ij}^{s}(t) \leq u_{ij}^{s}(t), \quad \forall (i,j) \in A, \quad t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\ \sum\limits_{t\leq T} y_{ij}^{s}(t) = x_{ij}, \quad (i,j) \in A, \quad s = 1,2,\ldots,S \end{cases}$$

Mô hình trên có nghiệm tối ưu là cận trên của mô hình ban đầu. Từ mô hình giản lượt trên ta có thể tách riêng những biến X và Y . Từ đó ta có thể chia mô hình trên thành hai vấn đề con:

3.2.4.1 Vấn đề con thứ nhất: Min − cost Flow Problem

Nó có dạng là:

$$\begin{cases} \min SP1(\alpha) = \sum\limits_{(i,j)\in A} (p_{ij} - \sum\limits_{s=1}^{S} \sum\limits_{t\leq T} \alpha_{ij}^{s}(t)) x_{ij} \\ \text{s.t.} \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \quad \forall (i,j) \in A \end{cases}$$

Nơi mà ta có thể gọi $C_{ij} = \sum_{(i,j)\in A}(p_{ij} - \sum_{s=1}^{S} \sum_{t\leq T} \alpha_{ij}^{s}(t))$ như là hàm biểu diễn chi phí trên mỗi liên kết. Vấn đề trên liên quan tới những biến quyết định X, có thể giải quyết bằng thuật toán successive path algorithm. Ta ký hiệu giá trị tối ưu của vấn đề con thứ nhất này là $Z_{SP1}(\alpha)$.

3.2.4.2 Vấn đề con thứ hai: Time-Dependent Min – cost Flow Problem Nó có dạng là:

$$\begin{cases} \min SP2(\alpha) = \sum_{s=1}^{S} \sum_{(i,j)\in A} \left(\sum_{t\in\{0,1,\ldots,T\}} \mu_s \cdot c_{ij}^s(t) + \sum_{t\leq T} \tilde{\alpha}_{ij}^s(t) \right) y_{ij}^s(t) \\ \text{s.t.} \\ \sum_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum_{(j'_t,i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \quad \forall i \in V, \\ t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \quad \forall (i,j) \in A, \quad t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \end{cases}$$

Vấn đề con thứ hai liên quan tới biến quyết định Y. Với mỗi kịch bản $s \in \{1,2,\ldots,S\}$ vấn đề con này có thể đưa về vấn đề con thứ nhất với chi phí phụ thuộc vào khung thời gian $c_{ij}^s(t)$ và khả năng thông qua phụ thuộc vào khung thời gian $u_{ij}^s(t)$ từ đó ta có thể gọi:

$$c_{ij}^s(t) = \begin{cases} \mu_s \cdot c_{ij}^s(t) + \alpha_{ij}^s(t), & \text{if } t \leq T' \\ \mu_s \cdot c_{ij}^s(t), & \text{if } T' \leq t \leq T \end{cases}$$

Là hàm chi phí trên mỗi kiên kết. Vấn đề có thể giải quyết bằng thuật toán successive path algorithm sửa đổi ở bước thứ 2. Và ta gọi giá trị tối ưu của vấn đề con thứ hai này là $Z_{SP2}(\alpha)$.

Sau khi giải quyết hai vấn đề con ta sẽ giá trị tối ưu mô hình giải lượt với tập hợp các vector α là:

$$Z_{LR}^*(\alpha) = Z_{SP1}^*(\alpha) + Z_{SP2}^*(\alpha)$$

Và ta biết rằng giá trị tối ưu của vấn đề ban đầu là cận dưới của giá trị tối ưu của mô hình giản lượt trên cho nên để có được nghiệm tối ưu nhất chúng ta cần đưa biểu thức trên đạt được gần với giá trị cận dưới nhất có thể. Cho nên, ta có thể biểu diễn cận dưới tốt nhất theo biểu thức sau:

$$Z_{LD}(\alpha^*) = \max_{\alpha\geq 0} Z_{LR}(\alpha)$$

4 Graph

4.1 Problem 1

4.1.1 Mô hình quy hoạch ngẫu nhiên hai giai đoạn

Để tìm ra được phương án sản xuất tối ưu cho công ty F, chúng ta sẽ tiếp cận vấn đề theo hướng phân tích các trường hợp xảy ra:

Chúng ta sẽ chia vấn đề thành hai giai đoạn:

- Giai đoạn 1: trước khi biết được nhu cầu d.
- Giai đoạn 2: sau khi biết được nhu cầu d.

Giả sử: Có hai trường hợp có thể xảy ra với xác suất xảy ra cho mỗi trường hợp là 50%:

- Trường hợp 1: vector nhu cầu $d_1 = (120, 140, 100, 150, 150, 150, 120, 90)$.
- Trường hợp 2: vector nhu cầu $d_2 = (18, 28, 35, 17, 18, 16, 29, 20)$.

4.1.1.1 Mô hình quy hoạch tuyến tính quy mô lớn Từ mô hình quy hoạch ngẫu nhiên hai giai đoạn trên, ta cũng có thể chuyển nó thành một mô hình quy hoạch tuyến tính quy mô lớn sau:

$$\begin{aligned} \min P(x, y_1, y_2, z_1, z_2) &= b^T x + \frac{1}{2}(c^T z_1 - s^T y_1) + \frac{1}{2}(c^T z_2 - s^T y_2) \\ \text{s.t.} \quad y_1 &= x - A^T z_1 \\ y_2 &= x - A^T z_2 \\ 0 \leqslant z_1 &\leqslant d_1,\ y_1 \geqslant 0 \\ 0 \leqslant z_2 &\leqslant d_2,\ y_2 \geqslant 0 \\ x &\geqslant 0 \end{aligned}$$

4.1.1.2 Kết quả tối ưu được tính bởi phần mềm Sau khi tính toán sử dụng thuật toán MIP trong GamsPy, ta có được kết quả sau:

$$\begin{aligned} Z &= 7348\$ \\ x &= (929, 379, 310, 368, 1071) \\ z_1 &= (0, 8, 74, 0, 1, 0, 120, 0) \\ z_2 &= (0, 28, 35, 17, 18, 0, 29, 0) \\ y_1 &= (0, 0, 0, 1, 0) \\ y_2 &= (272, 8, 1, 0, 585) \end{aligned}$$

Với:

- x_j $(j = 1, 2, \ldots, 5)$ là số lượng nguyên liệu cần đặt trước để tối ưu hóa lợi nhuận trong các trường hợp có thể xảy ra.
- Z là lợi nhuận ròng trung bình của công ty trong cả hai trường hợp.

- y_1 là số lượng nguyên liệu còn lại trong tình huống 1.
- y_2 là số lượng nguyên liệu còn lại trong tình huống 2.
- z_1 là sản phẩm nên sản xuất của mỗi loại trong tình huống 1.
- z_2 là sản phẩm nên sản xuất của mỗi loại trong tình huống 2.

4.1.2 Source code và kết quả chạy

Listing 1: Python script for optimization

```
import numpy as np
from gamspy import Container
from gamspy import Equation
from gamspy import Model
from gamspy import Variable
from gamspy import Set
from gamspy import Parameter
from gamspy import Sum
from gamspy import Sense
from gamspy import Problem
from gamspy import *

# so luong san pham
n = 8
# so luowng nguyen lieu can dat truoc khi san xuat
m = 5
# so kich ban S = 2

# [Rest of the code...]

SLP.solve()
print("so nguyen lieu moi loai x:\n",x.records)
print("\n")
print("so nguyen lieu con lai  y1 (kich ban 1):\n",y1.records
    )
print("\n")
print("so nguyen lieu con lai y2 (kich ban 2):\n",y2.records)
print("\n")
print("so luong moi loai san pham z1 (kich ban 1):\n",z1.
    records)
print("\n")
print("so luong moi loai san pham z2 (kich ban 2):\n",z2.
    records)
print("\n")
print("Gia tri ham muc tieu: ",SLP.objective_value)
```

số lượng nguyên liệu mỗi loại x:

	j	level	marginal	lower	upper	scale
0	0	929.0	7.0	0.0	inf	1.0
1	1	379.0	7.0	0.0	inf	1.0
2	2	310.0	7.0	0.0	inf	1.0
3	3	368.0	7.0	0.0	inf	1.0
4	4	1071.0	6.0	0.0	inf	1.0

Hình 1: Giá trị của x

số đơn vị nguyên liệu còn lại y1 (kịch bản 1):

	j	level	marginal	lower	upper	scale
0	0	0.0	-2.5	0.0	inf	1.0
1	1	0.0	-2.0	0.0	inf	1.0
2	2	0.0	-2.5	0.0	inf	1.0
3	3	1.0	-2.5	0.0	inf	1.0
4	4	0.0	-2.5	0.0	inf	1.0

số đơn vị nguyên liệu còn lại y2 (kịch bản 2):

	j	level	marginal	lower	upper	scale
0	0	272.0	-2.5	0.0	inf	1.0
1	1	8.0	-2.0	0.0	inf	1.0
2	2	1.0	-2.5	0.0	inf	1.0
3	3	0.0	-2.5	0.0	inf	1.0
4	4	585.0	-2.5	0.0	inf	1.0

Hình 2: Giá trị của y

số lượng mỗi loại sản phẩm z1 (kịch bản 1):

	i	level	marginal	lower	upper	scale
0	0	0.0	-45.0	0.0	inf	1.0
1	1	8.0	-65.0	0.0	inf	1.0
2	2	74.0	-80.0	0.0	inf	1.0
3	3	0.0	-40.0	0.0	inf	1.0
4	4	1.0	-119.0	0.0	inf	1.0
5	5	0.0	-52.5	0.0	inf	1.0
6	6	120.0	-77.5	0.0	inf	1.0
7	7	0.0	-45.0	0.0	inf	1.0

số lượng mỗi loại sản phẩm z2 (kịch bản 2):

	i	level	marginal	lower	upper	scale
0	0	0.0	-45.0	0.0	inf	1.0
1	1	28.0	-65.0	0.0	inf	1.0
2	2	35.0	-80.0	0.0	inf	1.0
3	3	17.0	-40.0	0.0	inf	1.0
4	4	18.0	-119.0	0.0	inf	1.0
5	5	0.0	-52.5	0.0	inf	1.0
6	6	29.0	-77.5	0.0	inf	1.0
7	7	0.0	-45.0	0.0	inf	1.0

Hình 3: Giá trị của z

```
giá trị của hàm mục tiêu:  -7384.0
```

Hình 4: Giá trị hàm mục tiêu

4.2 Problem 2: Áp dụng thuật toán Minimum-cost flow

Thuật toán Minimum-cost flow

Cho mạng lưới G gồm n đỉnh và m cạnh. Trên mỗi cạnh thì có the capacity (khả năng chứa) và the cost per unit of flow (chi phí). Bên cạnh đó, sourse s và sink t đã được cho. Với một giá trị K cho trước, ta phải tìm ra dòng dòng chảy (flow) có chi phí THẤP NHẤT $\Rightarrow$ minimum-cost flow problem.

Cách áp dụng thuật toán

Ta gọi U_{ij} là khả năng chứa của (i, j), nếu cạnh đó tồn tại. C_{ij} là chí phí cho mỗi đơn vị dòng chảy dọc theo cái cạnh (i, j). Và F_{ij} là flow dọc theo cạnh $(i, j) \Rightarrow$ Ban đầu tất cả flow đều bằng 0.

Ta bắt đầu modify lại mạng lưới đã có sẵn:

- Đối với mỗi cạnh $(i, j) \Rightarrow$ ta thêm một cạnh (j, i) vào đồ thị với $U_{ji} = 0$ và $C_{ji} = -C_{ij}$. Ngoài ra, chúng ta luôn giữ điều kiện $F_{ji} = -F_{ij}$ đúng trong các bước của thuật toán.
- Chúng ta định nghĩa mạng dư cho một dòng chảy cố định F như sau (giống như trong thuật toán Ford-Fulkerson): mạng dư chỉ chứa các cạnh không bão hòa (tức là các cạnh mà $F_{ij} < U_{ij}$), và khả năng dư của mỗi cạnh như vậy là $R_{ij} = U_{ij} - F_{ij}$.
- Bây giờ chúng ta có thể nói về các thuật toán để tính toán dòng chảy chi phí tối thiểu. Tại mỗi lần lặp của thuật toán, chúng ta tìm đường đi ngắn nhất trong mạng dư từ s đến t.
- Không khó để thấy, nếu chúng ta đặt K thành vô hạn, thì thuật toán sẽ tìm dòng chảy tối đa chi phí tối thiểu. Vì vậy, cả hai biến thể của vấn đề có thể được giải quyết bằng cùng một thuật toán.

Độ phức tạp của thuật toán

Thuật toán ở đây nói chung là hàm mũ theo kích thước đầu vào. Cụ thể hơn, trong trường hợp xấu nhất, nó có thể chỉ đẩy được 1 đơn vị lưu lượng ở mỗi lần lặp, mất $O(F)$ lần lặp để tìm kiếm dòng chảy có chi phí tối thiểu với kích thước F, tạo ra tổng thời gian chạy là $O(F \cdot T)$, trong đó T là thời gian cần thiết để tìm đường đi ngắn nhất từ nguồn đến đích.

Nếu sử dụng thuật toán Dijkstra cho việc này, thời gian chạy sẽ là $O(F \cdot mn)$. Cũng có thể chỉnh sửa thuật toán Dijkstra, sao cho nó cần $O(nm)$ tiền xử lý làm bước khởi đầu và sau đó hoạt động trong $O(m \log n)$ mỗi lần lặp, tạo ra tổng thời gian chạy là $O(mn + F \cdot m \log n)$. Dưới đây là một trình tạo đồ thị, mà thuật toán như vậy sẽ yêu cầu thời gian $O(2^{n/2} \cdot n^2 \log n)$.

Giải thích thuật toán

Khởi tạo và Kiểm tra Điều Kiện Lặp: Hàm bắt đầu bằng việc khởi tạo *flow* (dòng chảy hiện tại) và *flow_ cost* (chi phí dòng chảy) là 0. Một vòng lặp `while` được thiết lập để chạy miễn là *flow* nhỏ hơn K (dòng chảy mong muốn).

Tìm Đường Đi Ngắn Nhất: Trong mỗi lần lặp của vòng lặp `while`, hàm `shortest_paths` được gọi. Hàm này cập nhật **d** (vector lưu trữ khoảng cách ngắn nhất từ nguồn đến mọi đỉnh) và **p** (vector lưu trữ đỉnh trước của mỗi đỉnh trên đường đi ngắn nhất). Nếu $d[t]$ (khoảng cách từ nguồn đến đích) là `INF` (vô cực), nghĩa là không có đường đi từ nguồn đến đích, vòng lặp sẽ dừng và trả về -1, biểu thị không thể tìm thấy dòng chảy mong muốn.

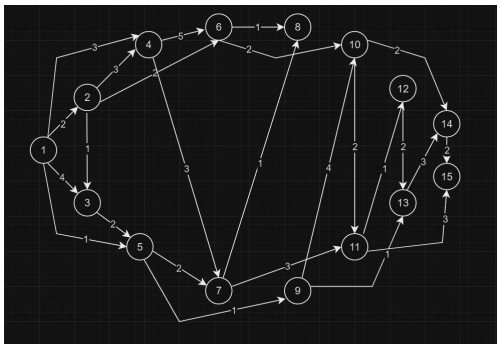
Xác Định Lượng Dòng Chảy Trên Đường Đi Được Chọn: Hàm sau đó tìm kiếm lượng dòng chảy tối đa f có thể chảy qua đường đi ngắn nhất này. f bắt đầu bằng $K - flow$ và được cập

nhật thông qua việc đi ngược lại từ đích t đến nguồn s, tìm giá trị nhỏ nhất của sức chứa trên đường đi.

Cập Nhật Dòng Chảy và Chi Phí: Khi f được xác định, dòng chảy *flow* được cập nhật bằng cách cộng thêm f và *flow_ cost* cũng được cập nhật bằng cách cộng thêm $f \cdot d[t]$ (chi phí của dòng chảy qua đường đi ngắn nhất). Hàm sau đó cập nhật sức chứa của đồ thị bằng cách trừ đi lượng dòng chảy f khỏi sức chứa của mỗi cạnh trên đường đi từ đích t đến nguồn s, và cộng lượng dòng chảy tương ứng vào sức chứa ngược chiều.

Ví dụ

Ta có một đồ thị có hướng có trọng số như sau:



Ta mặc định cho sức chứa tối đa của các cạnh bằng const = 10, và đỉnh có thảm hỏa là 1 và đỉnh an toàn là đỉnh 15.

4.2.1 Source code và kết quả chạy

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

const int INF = 1e9;

struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;
int N; // Number of nodes
int K; // Desired flow
int source, sink;

void add_edge(int from, int to, int cap, int cst)
{
    adj[from].push_back(to);
```

```
    adj[to].push_back(from);
    capacity[from][to] = cap;
    capacity[to][from] = 0;
    cost[from][to] = cst;
    cost[to][from] = -cst;
}

void init_code()
{
    N = 15;
    K = 10;
    source = 1;
    sink = 15;

    adj.assign(N + 1, vector<int>());
    cost.assign(N + 1, vector<int>(N + 1));
    capacity.assign(N + 1, vector<int>(N + 1));

    vector<Edge> edges = {
        {1, 2, 10, 2},
        {1, 3, 10, 4},
        {2, 3, 10, 1},
        {2, 4, 10, 3},
        {3, 5, 10, 2},
        {4, 6, 10, 5},
        {5, 7, 10, 2},
        {6, 8, 10, 1},
        {7, 8, 10, 1},
        {8, 9, 10, 3},
        {9, 10, 10, 4},
        {10, 11, 10, 2},
        {11, 12, 10, 1},
        {12, 13, 10, 2},
        {13, 14, 10, 3},
        {14, 15, 10, 4},
        {2, 6, 10, 2},
        {6, 10, 10, 2},
        {10, 14, 10, 2},
        {1, 5, 10, 1},
        {5, 9, 10, 1},
        {9, 13, 10, 1},
        {1, 4, 10, 3},
        {4, 7, 10, 3},
        {7, 11, 10, 3},
        {11, 15, 10, 3}};
    for (const Edge &e : edges)
        add_edge(e.from, e.to, e.capacity, e.cost);
}
```

```
void shortest_paths(int n, int v0, vector<int> &d, vector<int> &p)
{
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u])
        {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v])
            {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v])
                {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, int K, int s, int t)
{
    int flow = 0;
    int flow_cost = 0;
    vector<int> d, p;
    while (flow < K)
    {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s)
        {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }
```

```
        // apply flow
        flow += f;
        flow_cost += f * d[t];
        cur = t;
        while (cur != s)
        {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return flow_cost;
}

int main()
{
    init_code();

    int min_cost = min_cost_flow(N + 1, K, source, sink);
    if (min_cost == -1)
        cout << "It was not possible to find the desired flow." << endl;
    else
        cout << "The minimum cost to achieve the flow " << K << " is " <<
            min_cost << "." << endl;

    return 0;
}
```

Đáp án của bài toán này khi kết thúc là:

“The minimum cost to achieve the flow 10 is 90".

References

[1] MM Assignment Stochastic Programming and Applications 2023

[2] https://gamspy.readthedocs.io/en/latest/user/index.html