TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT

BÁO CÁO TỔNG KẾT
ĐỀ TÀI NCKH SINH VIÊN

MỘT SỐ BÀI TOÁN THỰC TẾ CỦA MA TRẬN, ĐỊNH THỨC VÀ HỆ PHƯƠNG TRÌNH TUYẾN TÍNH

Hà Nội, 5/2025

TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT

BÁO CÁO TỔNG KẾT
ĐỀ TÀI NCKH SINH VIÊN

MỘT SỐ BÀI TOÁN THỰC TẾ CỦA MA TRẬN, ĐỊNH THỨC VÀ HỆ PHƯƠNG TRÌNH TUYẾN TÍNH

Trưởng nhóm nghiên cứu: Nguyễn Hoàng Long — Lớp: DCCDTD68B

Thành viên tham gia thực hiện: Nguyễn Đại Lực — Lớp: DCCTCT68A

Vũ Ngọc Linh — Lớp: DCCBHD69

Nguyễn Cao Thanh Huyền — Lớp: DCCBHD69

Người hướng dẫn: GVC.TS Lê Bích Phượng

Hà Nội, 5/2025

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn quý thầy cô Bộ môn Toán Trường Đại học Mỏ - Địa chất đã tận tình giảng dạy và truyền đạt cho em những kiến thức quý báu về đại số tuyến tính cũng như các ứng dụng của môn học trong thực tiễn.

Đặc biệt, chúng em xin bày tỏ lòng biết ơn sâu sắc đến cô Lê Bích Phượng người đã tận tâm hỗ trợ, định hướng và góp ý trong suốt quá trình em thực hiện đề tài "**Một số bài toán thực tế của ma trận, định thức và hệ phương trình tuyến tính**".

Nhóm nghiên cứu cũng xin trân trọng cảm ơn các bạn trong nhóm nghiên cứu khoa học và những người đã chia sẻ tài liệu, thảo luận học thuật trong suốt thời gian học tập và thực hiện đề tài.

Mặc dù đã cố gắng tìm hiểu và trình bày đề tài một cách nghiêm túc và có hệ thống, song do hạn chế về thời gian và năng lực bản thân, báo cáo chắc chắn không tránh khỏi những thiếu sót. Chúng em rất mong nhận được những nhận xét và đóng góp từ quý thầy cô để chúng em có thể tiếp tục hoàn thiện hơn trong những nghiên cứu sau này.

LỜI CAM ĐOAN

Chúng em là nhóm sinh viên thực hiện đề tài **"Một số bài toán thực tế của ma trận, định thức và hệ phương trình tuyến tính"**, xin cam đoan toàn bộ nội dung của báo cáo là kết quả của quá trình làm việc nghiêm túc, chủ động nghiên cứu, thảo luận và phân công thực hiện giữa các thành viên trong nhóm. Báo cáo được xây dựng dựa trên kiến thức đã học trong môn Đại số tuyến tính, kết hợp với việc tìm hiểu thêm từ các tài liệu học thuật và nguồn thông tin chính thống, nhằm mở rộng hiểu biết và liên hệ với các ứng dụng thực tiễn trong kỹ thuật, công nghệ và đời sống.

Chúng em khẳng định rằng báo cáo này không sao chép từ bất kỳ sản phẩm có sẵn nào, không sử dụng trí tuệ nhân tạo hay công cụ hỗ trợ một cách vi phạm quy định học thuật, cũng như không thuê người làm thay dưới bất kỳ hình thức nào. Tất cả các trích dẫn, số liệu, hình ảnh và nội dung tham khảo đều được ghi rõ nguồn, thể hiện tinh thần trung thực, nghiêm túc trong nghiên cứu.

Nếu phát hiện có bất kỳ vi phạm nào liên quan đến đạo đức học thuật, chúng em xin chịu hoàn toàn trách nhiệm trước các thầy cô và nhà trường.

MỞ ĐẦU

1. Tính cấp thiết của đề tài

Trong thời đại công nghiệp 4.0, các công cụ toán học giữ vai trò then chốt trong việc phân tích và giải quyết các bài toán kỹ thuật và công nghệ. Ma trận, định thức và hệ phương trình tuyến tính là những khái niệm cơ bản của đại số tuyến tính, không chỉ xuất hiện trong lý thuyết mà còn gắn liền với nhiều ứng dụng thực tiễn trong các lĩnh vực như kỹ thuật, kinh tế và trí tuệ nhân tạo.

Chẳng hạn, trong tối ưu hóa, các bài toán phân bổ nguồn lực hay tối ưu đều có thể mô hình hóa bằng hệ phương trình tuyến tính và giải bằng phương pháp ma trận. Trong mã hóa thông tin, ma trận được dùng làm khóa để mã hóa và giải mã văn bản (như mật mã Hill). Thuật toán PageRank nổi tiếng của Google cũng dựa trên lý thuyết ma trận để xếp hạng độ quan trọng của các trang web trong mạng liên kết. Ngoài ra, chuỗi Markov – một mô hình xác suất quan trọng có thể biểu diễn bằng ma trận chuyển trạng thái, giúp dự báo các hiện tượng ngẫu nhiên như thời tiết hay hành vi khách hàng.

Trong trí tuệ nhân tạo và học máy, các thuật toán như hồi quy tuyến tính, phân tích thành phần chính (PCA), mạng nơ-ron,… đều dựa trên thao tác ma trận và yêu cầu người học nắm vững kiến thức đại số tuyến tính. Trong khoa học máy tính và công nghệ phần mềm, việc xử lý ảnh, mô phỏng vật lý, thiết kế đồ họa hay hệ thống điều khiển cũng phụ thuộc mạnh mẽ vào các phép biến đổi ma trận và giải hệ phương trình tuyến tính.

Tuy nhiên, trong quá trình giảng dạy và học tập hiện nay, nội dung về ma trận, định thức và hệ phương trình tuyến tính thường được trình bày theo kiểu học thuật, khiến người học ít nhận thấy được giá trị ứng dụng sâu rộng của chúng. Vì vậy, việc nghiên cứu và trình bày một số bài toán thực tế liên quan đến ma trận, định thức và hệ phương trình tuyến tính không chỉ giúp người học hiểu sâu kiến thức toán học nền tảng mà còn phát triển tư duy logic, năng lực mô hình hóa và giải quyết vấn đề – những kỹ năng thiết yếu trong thời đại số.

Chính vì những lý do đó, đề tài này mang tính cấp thiết cao, góp phần rút ngắn khoảng cách giữa lý thuyết và thực tiễn, đồng thời nâng cao hiệu quả học tập, giúp sinh viên – đặc biệt là sinh viên các ngành kỹ thuật, kinh tế và công

nghệ – có thể vận dụng toán học vào giải quyết hiệu quả các vấn đề trong học tập và đời sống nghề nghiệp.

2. Đối tượng và phạm vi nghiên cứu

2.1. Đối tượng nghiên cứu

Các bài toán thực tiễn nổi bật có sử dụng ma trận, định thức và hệ phương trình tuyến tính để mô hình hóa và giải quyết. Cụ thể, đề tài tập trung vào bốn chủ đề: Bài toán tối ưu hóa sử dụng hệ phương trình tuyến tính; Ứng dụng ma trận trong mã hóa văn bản; Thuật toán PageRank trong xếp hạng mạng liên kết; Mô hình chuỗi Markov trong hệ thống xác suất.

2.2. Phạm vi nghiên cứu

Đề tài giới hạn phạm vi ở mức độ cơ bản và mô phỏng các ứng dụng kể trên. Mỗi ứng dụng sẽ được minh họa bằng ví dụ cụ thể và mã lập trình Python để mô phỏng hoặc tính toán kết quả. Phạm vi không đi sâu vào chứng minh lý thuyết phức tạp, thay vào đó nhấn mạnh sự liên hệ giữa lý thuyết và thực tiễn thông qua các bài toán mẫu.

3. Mục đích và mục tiêu nghiên cứu

3.1. Mục đích nghiên cứu

Xây dựng một báo cáo tổng quan và minh họa sinh động về một số ứng dụng thực tế tiêu biểu của ma trận, định thức và hệ phương trình tuyến tính. Thông qua đó, giúp sinh viên hiểu rõ hơn ý nghĩa của các khái niệm toán học này, thấy được mối liên hệ giữa kiến thức toán học và các vấn đề kỹ thuật đời sống, cũng như rèn luyện kỹ năng lập mô hình và giải quyết vấn đề bằng công cụ toán học.

3.2. Mục tiêu nghiên cứu

Trình bày rõ ràng cơ sở lý thuyết về ma trận, định thức, hệ phương trình tuyến tính và các khái niệm liên quan (vector riêng, chuỗi Markov...) làm nền tảng cho các ứng dụng. Mô tả chi tiết 4 ứng dụng thực tế đã nêu, bao gồm: tối ưu hóa vận tải và phân bổ nguồn lực; mã hóa văn bản bằng ma trận (Hill cipher); thuật toán PageRank xếp hạng trang web; chuỗi Markov dự báo hệ thống ngẫu nhiên. Với mỗi ứng dụng, đưa ra từ 2–3 ví dụ minh họa cụ thể, kèm theo phân

tích và giải bằng Python, nhằm kiểm chứng và trực quan hóa lý thuyết. Đánh giá kết quả thu được và rút ra kết luận.

4. Ý nghĩa khoa học và giá trị thực tiễn

Ý nghĩa khoa học: Báo cáo này góp phần hệ thống hóa và làm rõ những kiến thức cốt lõi của đại số tuyến tính như ma trận, định thức và hệ phương trình tuyến tính, đồng thời mở rộng sang việc liên hệ với các thuật toán và mô hình toán học hiện đại như PageRank hay chuỗi Markov. Thông qua việc tiếp cận các chủ đề mang tính liên ngành, sinh viên không chỉ củng cố kiến thức lý thuyết mà còn phát triển tư duy mô hình hóa toán học, khả năng phân tích và lập trình mô phỏng – những năng lực thiết yếu trong thời đại công nghệ số.

Giá trị thực tiễn: Các ứng dụng được lựa chọn và phân tích trong báo cáo đều có tính thực tiễn cao và gắn bó mật thiết với các hoạt động trong đời sống, sản xuất và công nghiệp hiện đại. Bài toán vận tải và phân phối giúp doanh nghiệp xây dựng chiến lược sử dụng nguồn lực hợp lý. Mật mã Hill là ví dụ điển hình cho việc mã hóa thông tin, tạo nền tảng cho các hệ thống bảo mật trong thời đại số. Thuật toán PageRank là trụ cột của các công cụ tìm kiếm và các hệ thống gợi ý và xếp hạng phổ biến hiện nay. Mô hình chuỗi Markov hiệu quả trong các bài toán dự báo, từ dự đoán hành vi người dùng, biến động thị trường đến mô phỏng các quy trình kỹ thuật.

Việc hiểu và ứng dụng các mô hình này giúp sinh viên nâng cao khả năng vận dụng toán học vào thực tiễn, từ đó hiểu sâu hơn về các vấn đề trong chuyên ngành và phát triển tư duy liên ngành – yếu tố ngày càng được đánh giá cao trong môi trường làm việc hiện đại và hội nhập.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1. Ma trận, định thức và ma trận nghịch đảo

1.1. Ma trận

Định nghĩa: Ma trận là một bảng hình chữ nhật gồm các phần tử sắp xếp theo hàng và cột. Kích thước của ma trận được xác định bởi số hàng m và số cột n [5] . Ví dụ, ma trận A cấp m $\times$ n có dạng:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Trong đó $A = [a_{ij}]_{m\times n}$ là phần tử nằm ở hàng i, cột j của ma trận. Các phép toán cơ bản trên ma trận gồm: cộng hai ma trận cùng kích thước (cộng từng phần tử tương ứng), nhân vô hướng (nhân mọi phần tử với một số vô hướng), và nhân hai ma trận (nếu ma trận thứ nhất kích thước m $\times$ k và ma trận thứ hai k $\times$ n thì tích là ma trận m $\times$ n với phần tử (i, j) bằng tích vô hướng của hàng i của ma trận thứ nhất với cột j của ma trận thứ hai).

Ma trận đơn vị I_n là ma trận vuông n $\times$ n có các phần tử trên đường chéo chính bằng 1 và các phần tử khác bằng 0. Một số ma trận đơn và tổng quát ma trận đơn:

$$I_1 = [1],\ I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ \ldots,\ I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Hình 1: Một số ma trận đơn và tổng quát ma trận đơn

1.2. Định thức

Số nghịch thế của một hoán vị của n số tự nhiên đầu tiên:

Xét Ω = {1, 2, ..., n} là tập n số tự nhiên đầu tiên. Ta đã biết có tất cả n! hoán vị của n số tự nhiên đầu tiên.

Giả sử $(\alpha_1, \alpha_2, ..., \alpha_n)$ là một hoán vị của tập Ω, khi đó hai số α_i và α_j được gọi là tạo thành một nghịch thế nếu $\alpha_i > \alpha_j$ (với $i < j$). Để tính số nghịch thế của một hoán vị $(\alpha_1, \alpha_2, ..., \alpha_n)$ ta thực hiện như sau:

Đi từ trái qua phải trong hoán vị, gọi k_1 là số phần tử $\alpha_i < \alpha_1$ trong dãy $\alpha_1, \alpha_2, ..., \alpha_n$; gọi k_2 là số phần tử $\alpha_i < \alpha_2$ trong dãy $\alpha_1, \alpha_2, ..., \alpha_n$; ...; gọi k_{n-1} là số phần tử $\alpha_i < \alpha_{n-1}$ trong dãy $\alpha_1, \alpha_2, ..., \alpha_n$. Khi đó $k_1 + k_2 + ... + k_{n-1}$ chính là tổng số nghịch thế của hoán vị $(\alpha_1, \alpha_2, ..., \alpha_n)$.

Thành phần của định thức:

Cho $A = (a_{ij})_{n\times n}$ là ma trận vuông cấp n và $(\alpha_1, \alpha_2, ..., \alpha_n)$ là một hoán vị của n số tự nhiên đầu tiên.

Gọi α là số nghịch thế của hoán vị đó. Khi ấy $(-1)^{\alpha} \prod_i^n a_i\alpha_i$ được gọi là một thành phần của định thức (còn gọi là phần tử định thức) của ma trận A tương ứng với hoán vị $(\alpha_1, \alpha_2, ..., \alpha_n)$.

Định nghĩa định thức:

Với $A = (a_{ij})_{n\times n}$ là ma trận vuông cấp n, khi đó định thức của ma trận A là một số được ký hiệu là det(A) hoặc |A|, và được xác định bởi công thức:

$$\det(A) = \sum_{(\alpha_1,\alpha_2,...,\alpha_n) \in S}(-1)^{\alpha} \times \prod_i^n a_i\alpha_i$$

Trong đó: α là số nghịch thế của hoán vị $(\alpha_1, \alpha_2, ..., \alpha_n)$ và S là tập hợp tất cả các hoán vị của n số tự nhiên đầu tiên.

Các tính chất của định thức:

Tính chất 1. Cho A là ma trận vuông, khi đó:

$$|A| = |A^T|,$$

Tức định thức của ma trận bằng định thức của ma trận chuyển vị.

Tính chất 2. Nếu tất cả các phần tử của một dòng hoặc một cột trong định thức đều bằng 0 thì định thức đó bằng 0.

Tính chất 3. Trong một định thức, nếu đổi chỗ hai dòng (hoặc hai cột) với nhau và giữ nguyên các dòng (hoặc cột) còn lại, thì định thức đổi dấu.

Hệ quả 1: Nếu định thức có hai dòng hoặc hai cột giống nhau, thì định thức đó bằng 0.

Tính chất 4. Nếu ta nhân một dòng (hoặc một cột) của định thức với một số thực α, thì định thức mới bằng định thức ban đầu nhân với α.

***Hệ quả 2**:* $\det(\alpha A) = \alpha^n \det(A)$ (với A là ma trận vuông cấp n, và α là một số thực*)*.

***Hệ quả 3**:* Nếu định thức có hai dòng hoặc hai cột tỉ lệ nhau, thì định thức đó bằng 0.

Tính chất 5. Nếu ta cộng vào một dòng (hoặc cột) một bội số của dòng (hoặc cột) khác, thì giá trị của định thức không thay đổi.

Tính chất 6. Định thức bằng 0 nếu và chỉ nếu các vectơ dòng hoặc vectơ cột của ma trận phụ thuộc tuyến tính.

Hệ quả 4. Định thức khác 0 nếu và chỉ nếu các vectơ dòng hoặc vectơ cột của ma trận độc lập tuyến tính.

Tính chất 7. Khi các phần tử của một dòng (hoặc một cột) là tổng của hai biểu thức, thì định thức có thể tách thành tổng của hai định thức tương ứng.

1.3. Phép nhân ma trận

Định nghĩa:

Cho hai ma trận:

$A = (a_{ij})_{m\times p}$ là ma trận cấp m × p,

$B = (b_{ij})_{p\times n}$ là ma trận cấp p × n.

Trong đó, số cột của ma trận A bằng số dòng của ma trận B.

Khi đó, tích của ma trận A và ma trận B là một ma trận mới cấp m×n, được ký hiệu là AB và được xác định như sau:

$$AB = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & & c_{2n} \\ \vdots & & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}, \text{ trong đó } c_{ij} = A_i^d \times B_j^c$$

$$= (a_{i1}, a_{i2}, \ldots, a_{in}) \times \begin{pmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{nj} \end{pmatrix} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \ldots + a_{in} \times b_{nj}$$

Tính chất:

Phép nhân ma trận không tuân theo quy tắc giống phép nhân số thực, mà có các tính chất riêng biệt quan trọng trong đại số tuyến tính. Các tính chất cơ bản của phép nhân ma trận bao gồm:

Tính chất 1. Tính chất kết hợp:

$$(A \times B) \times C = A \times (B \times C)$$

Tính chất 2. Phân phối đối với phép cộng:

$$A \times (B + C) = \mathrm{A} \times \mathrm{B} + \mathrm{A} \times \mathrm{C};$$

$$(\mathrm{A} + \mathrm{B}) \times \mathrm{C} = \mathrm{A} \times \mathrm{C} + \mathrm{B} \times \mathrm{C}.$$

Tính chất 3. Nhân với một số thực α:

$$\alpha \times (A \times B) = (\alpha \times A) \times B = A \times (\alpha \times B)$$

Tính chất 4. Nhân với ma trận đơn vị:

$$A \times I = I \times A = A \text{ (với I là ma trận đơn vị)}$$

Hệ quả:

$$A \times (\alpha \times I) = (\alpha \times I) \times A = \alpha \times A$$

Tính chất 5. Chuyển vị tích hai ma trận:

$$(A \times B)T = BT \times AT$$

Tính chất 6. Định thức tích ma trận:

$$det\,(A \times B) = det(A) \times det(B)$$

$$det\,(A_1 \cdot A_2 \cdot \ldots \cdot A_\mathrm{n}) = \prod_{i=1}^{n} det(A_i)$$

Luỹ thừa của ma trận vuông:

$$A^\mathrm{n} = A \times \ldots \times A\ (n\ lần)$$

Hệ quả: $det(A^\mathrm{n}) = (det\ A)^\mathrm{n}$

Tính chất 7. Đồng nhất thức khi A và B giao hoán ($A \times B = B \times A$)

Tính chất 8. Giao hoán khi tích là bội của ma trận đơn vị:

$$A \times B = \alpha \times I\ (\alpha \neq 0) \rightarrow A \times B = B \times A$$

Tính chất 9. Ma trận phụ hợp:

$$A \cdot A^* = A^* \cdot A = det(A) \cdot I$$

Tính chất 10. Tổng bình phương các phần tử:

Tổng các phần tử trên đường chéo chính của $\mathrm{A} \cdot \mathrm{A}^\mathrm{T}$ là:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2$$

Tính chất 11. Ma trận luỹ thừa triệt tiêu:

Nếu tồn tại k ≥ n sao cho:

$$A^\mathrm{k} = O \rightarrow A^\mathrm{n} = O$$

Tính chất 12. Vết của AB và BA bằng nhau:

$$Tr(AB) = Tr(BA)$$

1.4. Ma trận nghịch đảo

1.4.1. Ma trận phụ hợp

***Định nghĩa**:*

Cho ma trận $A = (a_{ij})_{n\times n}$ và A_{ij} là phần bù đại số của phần tử a_{ij} của ma trận A. Khi đó:

$A^* = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}$ được gọi là ma trận phụ hợp của A.

Vậy

$$\begin{cases} A^* = \left(a_{ij}^*\right)_{n\times n} = \left(A_{ji}\right)_{n\times n} \rightarrow (kA)^* = k^{n-1}A^*. \\ (kA)_{ij} = k^{n-1}A_{ij} \end{cases}$$

Suy ra:

Phần tử thuộc trên dòng i, cột j của A^* là $a_{ij}^* = A_{ji} = (-1)^{j+i}M_{ji}$

Phần tử thuộc trên dòng i, cột j của $(kA)^*$ là $k^{n-1}A_{ji} = k^{n-1}(-1)^{j+i}M_{ji}$.

Các tính chất:

***Bổ đề 1**:* Cho ma trận vuông $A = (a_{ij})_{n\times n}$ và A_{ij} là phần bù đại số của phần tử a_{ij}. Khi đó*:*

$$i) \quad a_{i1}A_{k1} + a_{i2}A_{k2} + \ldots + a_{in}A_{kn} = \begin{cases} det(A), nếu\ i = k \\ 0 \quad , \ nếu\ i \neq k \end{cases}$$

$$ii) \quad a_{1j}A_{1q} + a_{2j}A_{2q} + \ldots + a_{nj}A_{nq} = \begin{cases} det(A), nếu\ j = q \\ 0 \quad , nếu\ j \neq q \end{cases}$$

***Bổ đề 2**: Cho A là ma trận vuông cấp n và A× là ma trận phụ hợp của A:*

$$AA^* = A^*A = det(A)I$$

***Bổ đề 3**: Ta có:*

$$det(A^*) = (det\ A)^{n-1}$$

***Bổ đề 4**:*

Nếu det(A) = 0 thì các cột của A^* là nghiệm của hệ thuần nhất $AX = 0$.

1.4.2. Định nghĩa ma trận nghịch đảo

Xét ma trận vuông A. Ma trận vuông X cùng cấp với A được gọi là ma trận nghịch đảo của A nếu thỏa mãn:

$A \times X = X \times A = I$, được ký hiệu là A^{-1}.

Ma trận nghịch đảo nếu tồn tại thì là duy nhất.

Ma trận vuông A có ma trận nghịch đảo khi và chỉ khi det(A) ≠ 0. Khi đó:

$$A^{-1} = \frac{1}{\det(A)} \times A^{*}$$

Nên ta gọi A là ma trận khả nghịch hoặc ma trận không suy biến.

Ngược lại, nếu det(A) = 0 thì A là ma trận suy biến (không khả nghịch).

1.5. Phép toán modulo cho ma trận

Trong một số ứng dụng như Mật mã Hill, các phép toán trên ma trận được thực hiện trong một vành số nguyên modulo n (thường là n=26 đối với bảng chữ cái tiếng Anh). Điều này có nghĩa là tất cả các kết quả của phép cộng, trừ, nhân các phần tử ma trận, cũng như các phần tử của ma trận nghịch đảo, đều được lấy phần dư khi chia cho n.

Đặc biệt, một ma trận K vuông có ma trận nghịch đảo K^{-1} theo modulo n nếu và chỉ nếu định thức của K, det(K), có nghịch đảo theo modulo n. Tức là, phải tồn tại một số nguyên d sao cho $d \times \det(K) \equiv 1 \pmod n$. Điều này xảy ra khi và chỉ khi gcd(det(K), n) = 1. Nếu điều kiện này được thỏa mãn, ma trận nghịch đảo K^{-1} có thể được tính bằng công thức $K^{-1} \equiv (\det(K))^{-1} \times K^{*} \pmod n$, trong đó K^{*} là ma trận phụ hợp của K và $(\det(K))^{-1}$ là nghịch đảo modulo n của det(K). Các phần tử của K^{*} cũng được tính theo modulo n.

2. Hạng của ma trận

2.1. Định nghĩa.

Xét ma trận $A = (a_{ij})_{m \times n}$.

Đặt:

$$A_i^d = (a_{i1}, a_{i2}, \ldots, a_{in});$$

$$A_j^d = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}.$$

Hạng của ma trận A là hạng của hệ vectơ dòng $\{A_1^d, A_2^d, \ldots, A_m^d\}$ và cũng chính là hạng của hệ vectơ cột $\{A_1^c, A_2^c, \ldots, A_n^c\}$. Được ký hiệu là r(A).

2.2. Định thức con của ma trận và mối quan hệ với hạng của ma trận

Xét ma trận $A = (a_{ij})_{m\times n}$. Chọn ra s dòng $(i_1, i_2, ..., i_s)$ với $i_1 < i_2 < ... < i_s$ và s cột $(j_1, j_2, ..., j_s)$ với $j_1 < j_2 < ... < j_s$, rồi xóa tất cả các dòng và cột còn lại trong A (nếu có), ta thu được một ma trận vuông cấp s.Định thức của ma trận này được gọi là định thức con cấp s của A, được ký hiệu:

$$D\begin{matrix} i_1 i_2 \dots i_s \\ j_1 j_2 \dots j_s \end{matrix}.$$

Một ma trận A cấp m×n có tất cả $C_m^s \times C_n^s$ định thức con cấp s.

Nếu tồn tại một định thức con cấp $s \neq 0$ và tất cả các định thức con cấp lớn hơn s đều bằng 0, thì hạng của ma trận A bằng s, tức r(A) = s.

2.3. Tìm hạng của ma trận

Tìm hạng của ma trận bằng phương pháp biến đổi sơ cấp:

Biến đổi ma trận về dạng bậc thang $\begin{pmatrix} b_{11} & b_{12} & \dots & b_{1r} & \cdots & b_{1n} \\ 0 & b_{22} & \dots & b_{2r} & & b_{2n} \\ & \vdots & & & \ddots & \vdots \\ 0 & 0 & \cdots & b_{rr} & & b_{rn} \\ 0 & 0 & \cdots & 0 & \cdots & 0 \end{pmatrix}$

Trong đó $b_{ii} \neq 0$ với i = 1, 2, ..., r. Khi đó, hạng của ma trận r(A) = r.

Tìm hạng của ma trận bằng phương pháp định thức bao quanh:

Chỉ ra một định thức con cấp s của ma trận khác 0, tính tất cả các định thức con cấp s + 1 bao quanh định thức con cấp s, nếu tất cả các định thức con cấp s + 1 này bằng 0 thì hạng của ma trận bằng s, ngược lại nếu có một định thức con cấp s + 1 khác 0, tính tất cả các định thức con cấp s + 2 bao quanh định thức con cấp s + 1…

2.4. Các tính chất về hạng của ma trận

Hạng của ma trận bằng hạng của ma trận chuyển vị:

$r(A) = r(A^T)$

Nếu A là ma trận vuông cấp n thì:

$r(A) = n \leftrightarrow \det(A) \neq 0$, dựa vào tính chất này, ta có thể dùng định thức để tìm hoặc biện luận hạng của một ma trận vuông.

Nếu A là ma trận vuông cấp n, thì hệ vectơ dòng (hoặc hệ vectơ cột) của A độc lập tuyến tính khi và chỉ khi r(A) = n.

2.5. Hạng của ma trận phụ hợp

Định lí: Cho ma trận A = $(a_{ij})_{n\times n}$, n ≥ 2 và A^* là ma trận phụ hợp của A, khi đó ta có:

r(A) = n ↔ r(A^*) = n;

r(A) = n – 1 ↔ r(A^*) = 1;

r(A) ≤ n – 2 ↔ r(A^*) = 0.

3. Hệ phương trình tuyến tính

3.1. Hệ phương trình tuyến tính tổng quát

Hệ phương trình tuyến tính tổng quát có dạng:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_{1n} = b2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n = b\text{m} \end{cases}$$

Với A = $\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$, X = $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, B = $\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$,

$$\bar{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & & a_{2n} & b_2 \\ \vdots & & \ddots & \vdots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{bmatrix}$$

Ta gọi là hệ phương trình tuyến tính gồm m phương trình và n ẩn

Hệ phương trình đã cho có thể viết dưới dạng ma trận AX = B.

Đặt $A_j^c = \begin{pmatrix} a1\text{j} \\ a2\text{j} \\ \vdots \\ a\text{mj} \end{pmatrix}$, j = 1, 2, ..., n là vectơ cột thứ j của ma trận hệ số A.

Khi đó hệ phương trình đã cho có thể viết dưới dạng vector:

$$x_2 A_1^c + x_2 A_2^c + \ldots + x_n A_n^c = B.$$

Vậy hệ có nghiệm khi và chỉ khi vectơ B biểu diễn tuyến tính qua hệ vectơ cột $\{A_1^c, A_2^c, \ldots, A_n^c\}$ của ma trận A. Hệ có bao nhiêu nghiệm thì có bấy nhiêu cách biểu diễn tuyến tính vectơ B qua hệ vectơ cột của ma trận A.

Do mọi định thức con của A đều là định thức con của Ā do đó:

$$0 \le r(A) \le r(\bar{A}) \le \min\{m, n+1\}.$$

Điều kiện cần và đủ để hệ phương trình tuyến tính tổng quát có nghiệm:

Định lý Kronecker – Capelli:

Cho hệ phương trình tuyến tính n ẩn AX = B . Điều kiện cần và đủ để hệ phương trình tuyến tính có nghiệm là:

$$r(A) = r(\bar{A}).$$

Chứng minh:

Ta có r(A) = r$\{A_1^c, A_2^c, \dots, A_n^c\}$, r($\bar{A}$) = $\{A_1^c, A_2^c, \dots, A_n^c, B\}$

Điều kiện cần:

Nếu hệ có nghiệm thì vectơ B được biểu diễn tuyến tính qua hệ vectơ:

$$\{A_1^c, A_2^c, \dots, A_n^c\}.$$

Do đó $r\{A_1^c, A_2^c, \dots, A_n^c, B\} = r\{A_1^c, A_2^c, \dots, A_n^c\} \rightarrow r(A) = r(\bar{A})$.

Điều kiện đủ:

Nếu $r(A) = r(\bar{A}) \rightarrow r\{A_1^c, A_2^c, \dots, A_n^c, B\} = r\{A_1^c, A_2^c, \dots, A_n^c\}$.

Ta có điều phải chứng minh.

Khảo sát tổng quát hệ phương trình tuyến tính:

Nếu r (A) = r ($\bar{A}$) = n thì hệ có nghiệm duy nhất;

Nếu r (A) = r ($\bar{A}$) = r < n thì hệ có vô số nghiệm phụ thuộc n − r tham số;

Nếu r (A) < r ($\bar{A}$) thì hệ vô nghiệm.

3.2. Hệ phương trình tuyến tính thuần nhất

Hệ phương trình tuyến tính tổng quát có dạng:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = 0 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_{1n} = 0 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n = 0 \end{cases}$$

Với $A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, O = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

Ta gọi là hệ phương trình tuyến tính gồm m phương trình và n ẩn.

Hệ phương trình đã cho có thể viết dưới dạng ma trận AX = O.

Đặt $A_j^c = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}$, j = 1, 2, ..., n là vectơ cột thứ j của ma trận hệ số A.

Hệ phương trình đã cho có thể viết dưới dạng vector:

$$x_1 A_1^c + x_2 A_2^c + \ldots + x_n A_n^c = O.$$

Hạng của ma trận hệ số và hạng của ma trận hệ số mở rộng của hệ thuần nhất bằng nhau do đó nó luôn có nghiệm. Hệ phương trình tuyến tính thuần nhất luôn có nghiệm $x_1 = x_2 = \ldots = x_n = 0$, nghiệm này được gọi là nghiệm tầm thường của hệ phương trình tuyến tính thuần nhất.

Điều kiện cần và đủ để hệ phương trình thuần nhất có nghiệm không tầm thường (vô số nghiệm):

Hệ phương trình thuần nhất n ẩn số có nghiệm không tầm thường khi và chỉ khi hạng của ma trận hệ số nhỏ hơn số ẩn.

Hệ quả 1: Hệ phương trình thuần nhất có số phương trình nhỏ hơn số ẩn luôn có nghiệm không tầm thường (vô số nghiệm).

Hệ quả 2: Hệ phương trình thuần nhất có số phương trình bằng số ẩn có nghiệm không tầm thường khi và chỉ khi định thức của ma trận hệ số bằng 0.

Hệ quả 3: Hệ phương trình thuần nhất có số phương trình bằng số ẩn chỉ có nghiệm tầm thường (nghiệm duy nhất) khi và chỉ khi định thức của ma trận hệ số khác 0.

Cấu trúc tập hợp nghiệm của hệ phương trình tuyến tính thuần nhất:

Tập $\ker(A) = \left\{X = \begin{pmatrix} x1 \\ x2 \\ \vdots \\ xn \end{pmatrix} \in \mathbb{R}^n | AX = O\right\}$ là một không gian con của không gian vectơ $\mathbb{R}^n$ và được gọi là tập hợp tất cả các nghiệm của hệ thuần nhất AX = O hay không gian nghiệm của hệ thuần nhất.

Mỗi cơ sở của ker(A) được gọi là một hệ nghiệm của hệ thuần nhất.

Số chiều không gian nghiệm của hệ thuần nhất dim(ker(A)) = n − r (A).

Vậy r (A) = r < n thì hệ thuần nhất có vô số nghiệm phụ thuộc n − r.

Mối quan hệ giữa hệ phương trình tuyến tính tổng quát với hệ phương trình tuyến tính thuần nhất liên kết:

Xét hệ phương trình tuyến tính tổng quát AX = B có n ẩn số. Khi đó hệ phương trình AX = O được gọi là hệ thuần nhất liên kết với hệ phương trình tổng quát đã cho.

Gọi X_0 là một nghiệm riêng của hệ phương trình tuyến tính tổng quát;

Gọi $\{P_1, P_2, \ldots, P_{n-r(A)}\}$ là một hệ nghiệm cơ bản của hệ thuần nhất liên kết;

Khi đó nghiệm tổng quát của hệ phương trình tuyến tính tổng quát là:

$$X = X_0 + t_1P_1 + t_2P_2 + \ldots + t_{n-r(A)}P_{n-r(A)}.$$

4. Giá trị riêng và vectơ riêng

4.1. Định nghĩa

Cho ma trận vuông A cấp n, một vectơ khác vectơ không được gọi là vector riêng của A nếu thỏa mãn:

$$Av = \lambda v$$

Trong đó λ là một vô hướng, được gọi là giá trị riêng tương ứng với v.

Để phương trình này có nghiệm khác không, ta cần:

$$\det(A - \lambda I) = 0$$

Phương trình này được gọi là phương trình đặc trưng của A

Mỗi giá trị riêng sẽ có ít nhất 1 vectơ riêng tương ứng với nó.

Trong các ứng dụng như thuật toán PageRank và Chuỗi Markov, các khái niệm liên quan đến giá trị riêng và vector riêng đóng vai trò trung tâm. Cụ thể, ma trận chuyển trong các mô hình này thường là ma trận ngẫu nhiên (stochastic matrix), tức là ma trận vuông không âm mà tổng các phần tử trên mỗi hàng (hoặc mỗi cột, tùy theo quy ước) bằng 1.

Theo Định lý Perron-Frobenius:

Một ma trận ngẫu nhiên (hoặc một lớp rộng hơn của các ma trận không âm) luôn có giá trị riêng lớn nhất bằng 1.

Vector riêng tương ứng với giá trị riêng bằng 1 này (thường được chuẩn hóa để tổng các thành phần bằng 1) chính là phân phối dừng của chuỗi Markov, hoặc là vector PageRank trong thuật toán của Google. Vector này thể hiện xác suất ở mỗi trạng thái trong dài hạn hoặc tầm quan trọng tương đối của mỗi trang web.

CHƯƠNG 2. ỨNG DỤNG ĐẠI SỐ TUYẾN TÍNH VÀO CÁC BÀI TOÁN THỰC TẾ

1. Tối ưu hóa các bài toán thực tế

Các bài toán ứng dụng trong kinh doanh, kinh tế học, khoa học xã hội và khoa học đời sống thường yêu cầu chúng ta đưa ra quyết định dựa trên một số điều kiện nhất định. Những điều kiện hoặc ràng buộc này thường có dạng các bất phương trình. Trong phần này, chúng ta sẽ bắt đầu xây dựng mô hình, phân tích và giải các bài toán như vậy ở mức độ đơn giản, nhằm hiểu rõ các thành phần cấu thành nên một bài toán như thế.

Một bài toán quy hoạch tuyến tính điển hình bao gồm việc tìm giá trị lớn nhất hoặc nhỏ nhất của một hàm tuyến tính dưới một số ràng buộc nhất định. Chúng ta hoặc là cố gắng tối đa hóa hoặc tối thiểu hóa giá trị của hàm tuyến tính đó, chẳng hạn như tối đa hóa lợi nhuận hoặc doanh thu, hoặc tối thiểu hóa chi phí. Chính vì vậy, các bài toán quy hoạch tuyến tính được phân loại thành bài toán cực đại hoặc cực tiểu, hay nói chung là bài toán tối ưu. Hàm mà ta cần tối ưu gọi là *hàm mục tiêu* (objective function), và các điều kiện ràng buộc phải thỏa mãn gọi là các *ràng buộc* (constraints).

1.1. Quy hoạch tuyến tính (Dạng cực đại)

Chúng ta bắt đầu bằng cách giải một bài toán tối đa hóa.

Bài toán 2.1: Nhi có 2 công việc part-time là dạy gia sư và đi phụ bán quán cà phê. Vì Nhi còn đang là sinh viên và phải tập trung học nên Nhi không muốn đi làm bên ngoài quá 12 tiếng trong một tuần. Sau một thời gian làm thử, Nhi thấy rằng trước khi đi dạy gia sư, cô cần tổng thời gian cả chuẩn bị và di chuyển là 2 tiếng, và tương tự, đối với việc phụ bán quán cà phê thì Nhi cần tổng cộng 24 phút để tới nơi. Ngoài ra, Nhi không thể dành quá 16 giờ trong một tuần cho việc chuẩn bị và di chuyển. Với mức thu nhập hiện tại từ việc dạy gia sư là 75.000 đồng/giờ và từ việc phụ quán cà phê là 27.000 đồng/giờ, câu hỏi đặt ra là: Nhi nên làm bao nhiêu giờ mỗi tuần ở mỗi công việc để kiếm được nhiều tiền nhất?

Tóm tắt bài toán:

Nhi có 2 công việc gọi là A và B.

Nhi chỉ muốn làm không quá 12 tiếng/tuần.

Nhi cũng muốn tổng thời gian mất thêm không quá 16 tiếng/tuần.

Công việc A: thu nhập 75.000 đồng/giờ, mất thêm 2 giờ.

Công việc B: thu nhập 27.000 đồng/giờ, mất thêm 0,4 giờ (tức 24 phút).
Yêu cầu: Tìm số giờ làm ở mỗi công việc sao cho tổng thu nhập lớn nhất.

Lời giải cho bài toán:

Đầu tiên, ta sẽ gọi các biến:

Gọi x là số giờ mỗi tuần Nhi làm ở Công việc A.

Gọi y là số giờ mỗi tuần Nhi làm ở Công việc B.

Bây giờ, ta viết hàm mục tiêu. Vì Nhi được trả 100k/h công việc A và 27k/giờ công việc B, nên tổng thu nhập (ký hiệu là I) được cho bởi phương trình sau:

$$I = 75x + 27y \text{ (nghìn đồng)}$$

Tiếp theo, chúng ta xác định các ràng buộc.

Trong đề bài nói rằng: "Nhi không muốn làm quá 12 giờ mỗi tuần."

Ta dịch điều đó thành ràng buộc sau:

$$x + y \leq 12$$

Tiếp theo đề bài lại nói: "Mỗi giờ làm việc ở Công việc A cần 2 giờ chuẩn bị, mỗi giờ làm ở Công việc B cần 0.4 giờ chuẩn bị, và tổng thời gian chuẩn bị không được vượt quá 16 giờ."

Điều này được biểu diễn bằng bất phương trình:

$$2x + 0.4y \leq 16$$

Vì x và y là số giờ làm việc nên chúng luôn không âm, tức là:

$$x \geq 0 \text{ và } y \geq 0$$

Chúng ta trình bày lại như sau:

Hàm ràng buộc: $I = 75x + 27y$ (nghìn đồng)

Các ràng buộc: $\begin{cases} x + y \leq 12, \\ 2x + 0.4y \leq 16, \\ x \geq 0, \\ y \geq 0. \end{cases}$

Hoặc có thể viết dưới dạng ma trận:

$$\begin{bmatrix} 1 & 1 \\ 2 & 0.4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} 12 \\ 16 \end{bmatrix}, \begin{bmatrix} x \\ y \end{bmatrix} \geq 0.$$

Để giải bài toán, chúng ta vẽ các ràng buộc lên đồ thị và tô đậm vùng thỏa mãn tất cả các bất phương trình.

Mỗi đường thẳng ràng buộc sẽ chia mặt phẳng thành hai vùng, trong đó chỉ một vùng là thỏa mãn điều kiện bất đẳng thức.

Ta dùng một điểm thử (test point) để xác định vùng nào cần tô. Bất kỳ điểm nào không nằm trên đường thẳng đều có thể làm điểm thử.

Nếu điểm thử thỏa mãn bất đẳng thức, thì vùng chứa điểm đó chính là vùng thỏa mãn.

Nếu điểm thử không thỏa mãn, thì vùng thỏa mãn sẽ nằm phía đối diện với đường thẳng so với điểm thử.

Trong đồ thị bên dưới, sau khi vẽ các đường thẳng biểu diễn ràng buộc bằng phương pháp thích hợp, điểm (0,0) được chọn làm điểm thử để kiểm tra:

(0,0) thỏa mãn ràng buộc $x + y \leq 12$, vì $0 + 0 < 12$

(0,0) thỏa mãn ràng buộc $2x + \frac{4y}{10} \leq 16$, vì $2 \cdot 0 + \frac{0}{10} < 16$

Vì vậy, trong ví dụ này, ta tô vùng nằm bên dưới và bên trái cả hai đường thẳng, đồng thời cũng nằm phía trên trục hoành ($x \geq 0$) và bên phải trục tung ($y \geq 0$), để đảm bảo tất cả các ràng buộc $x \geq 0$ và $y \geq 0$ cũng được thỏa mãn.

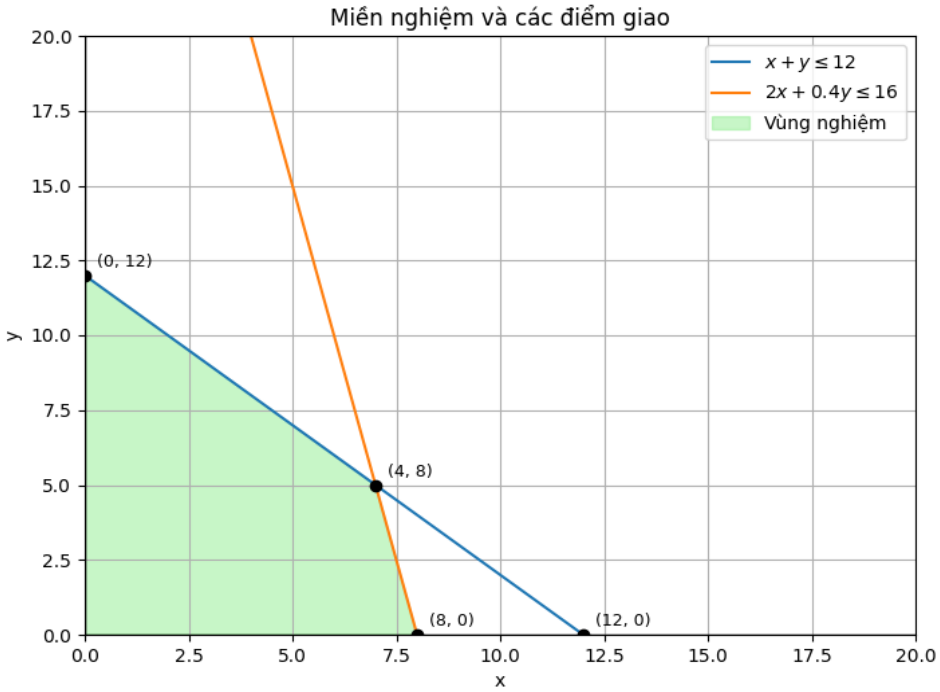
Vùng được tô đậm thỏa mãn tất cả các điều kiện, được gọi là vùng khả thi.

Định lý cơ bản của Quy hoạch Tuyến tính phát biểu rằng:

Giá trị lớn nhất (hoặc nhỏ nhất) của hàm mục tiêu luôn đạt được tại một trong các đỉnh của vùng khả thi.

Do đó, chúng ta sẽ xác định tất cả các đỉnh (điểm góc) của vùng khả thi.

Các điểm lần lượt là: (0, 0); (7, 5); (8, 0).



Hình 2: Đồ thị ràng buộc và vùng nghiệm tối ưu

Để tối đa hóa thu nhập của Nhi, chúng ta sẽ thay từng điểm vào hàm mục tiêu để xem điểm nào cho thu nhập cao nhất mỗi tuần.

Điểm	**Thu nhập**
(0, 0)	$75 \times 0 + 27 \times 0 = 0$
(7, 5)	$75 \times 7 + 27 \times 5 = 660$
(8, 0)	$75 \times 8 + 27 \times 0 = 600$

Từ bảng trên ta có thể thấy ở điểm (7, 5) mang lại thu nhập lớn nhất: 660 (nghìn đồng).

Do đó Nhi nên dạy gia sư 7 giờ và 5 giờ phụ bán quán cà phê.

1.2. Quy hoạch tuyến tính (Dạng cực tiểu)

Bài toán 2.2: Tại một trường đại học ở Việt Nam, Thầy Phương cần tuyển hai người trợ giảng để hỗ trợ chấm bài kiểm tra cho các lớp học của mình thì có 2 người: bạn Nam và bạn Mai.

Nam hiện tại là sinh viên năm nhất, có thể chấm 20 bài/giờ, và mức thù lao là 150.000 đồng/giờ. Còn Mai hiện tại là sinh viên năm thứ ba, có thể chấm 30

bài/giờ, và mức thù lao là 250.000 đồng/giờ. Để đảm bảo tính hợp lý trong việc tuyển dụng, mỗi người phải làm ít nhất 1 giờ mỗi tuần. Câu hỏi đặt ra: Nếu mỗi tuần thầy Phương có ít nhất 110 bài cần được chấm, vậy nên thuê Nam và Mai mỗi người bao nhiêu giờ mỗi tuần để tổng chi phí chấm bài là thấp nhất?

Tóm tắt bài toán:

Thầy Phương cần thuê hai người để chấm bài kiểm tra.

Nam có thể chấm 20 bài/giờ, lương là 150.000 đồng/giờ.

Mai có thể chấm 30 bài/giờ, lương là 250.000đồng/giờ.

Mỗi người phải làm ít nhất 1 giờ mỗi tuần.

Mỗi tuần có ít nhất 110 bài cần được chấm.

Yêu cầu: Tìm số giờ thuê Nam và Mai mỗi tuần sao cho chi phí là thấp nhất.

Lời giải cho bài toán:

Đầu tiên, ta gọi các biến:

Gọi x là số giờ mỗi tuần Nam làm việc.

Gọi y là số giờ mỗi tuần Mai làm việc.

Vì Nam được trả 150.000 đồng/giờ và Mai được trả 250.000 đồng/giờ, tổng chi phí (ký hiệu là C) được cho bởi: $C = 150x + 250y$ (nghìn đồng)

Mục tiêu của bài toán là tối thiểu hóa C.

Mỗi tuần phải chấm ít nhất 110 bài:

$$20x + 30y \geq 110$$

Mỗi người phải làm ít nhất 1 giờ:

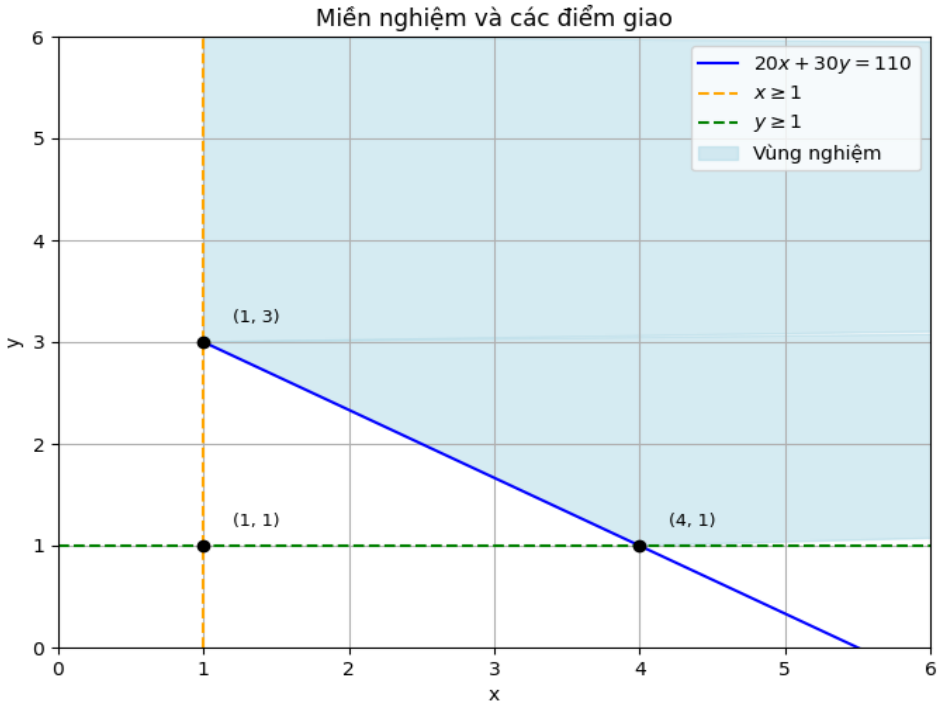
$$x \geq 1, y \geq 1$$

Chúng ta trình bày lại như sau:

Hàm ràng buộc: $C = 20x + 30y$ (nghìn đồng)

Các ràng buộc: $\begin{cases} x \geq 1, \\ y \geq 1, \\ 20x + 30y \geq 110. \end{cases}$

Để giải quyết vấn đề này, chúng ta biểu diễn các ràng buộc theo đồ thị:



Hình 3: Đồ thị ràng buộc và vùng nghiệm tối ưu

Nếu ta dùng điểm thử (0, 0) (là điểm không nằm trên bất kỳ đường ràng buộc nào), thì không thỏa mãn bất kỳ ràng buộc nào: $20x + 30y \geq 110$.

Do đó, vùng được tô đậm (vùng khả thi) sẽ nằm ở phía đối diện với các đường ràng buộc so với điểm (0, 0).

Nếu ta dùng điểm thử (4, 6) (cũng không nằm trên bất kỳ đường ràng buộc nào), ta sẽ thấy rằng (4, 6) thỏa mãn tất cả các bất phương trình ràng buộc. Vậy, vùng được tô đậm sẽ nằm cùng phía với các đường ràng buộc như điểm (4, 6).

Vì giá trị cực trị (lớn nhất hoặc nhỏ nhất) của hàm mục tiêu luôn xảy ra tại các đỉnh của vùng khả thi, chúng ta xác định hai điểm tới hạn là (1, 3) và (4, 1). Để tối thiểu hóa chi phí, ta sẽ thay các điểm này vào hàm mục tiêu để xem điểm nào cho chi phí thấp nhất mỗi tuần.

Điểm	**Chi phí**
(1, 3)	150 × 1 + 250× 3 = 900
(4, 1)	150 × 4 + 250× 1 = 850

Điểm (4, 1) đưa ra chi phí thấp nhất và chi phí đó là 850.000 đồng. Vậy để giảm thiểu chi phí chấm điểm, Thầy Phương nên thuê Nam 4 giờ một tuần và Mai 1 giờ một tuần với chi phí là 850.000 đồng một tuần.

2. Ứng dụng trong mã hóa văn bản

Một ứng dụng thú vị của ma trận là trong mã hóa thông tin. Mật mã Hill (Hill cipher) là một loại mật mã khóa đối xứng được đề xuất bởi Lester Sanders Hil năm 1929, trong đó ma trận khóa được sử dụng để biến đổi tuyến tính các khối ký tự của bản rõ thành bản mã. Đây là một trong những hệ mã hóa cổ điển dựa trên đại số tuyến tính.

2.1. Mật mã Hill và ma trận khóa

Định nghĩa mật mã Hill:

Trong hệ mật mã này, văn bản gốc được chia thành các khối ký tự có độ dài cố định (thường là 2 hoặc 3), sau đó mỗi ký tự được ánh xạ thành một số nguyên từ 0 đến 25 (A = 0, B = 1, ..., Z = 25). Mỗi khối được coi như một vectơ cột và được nhân với một *ma trận khóa* vuông *khả nghịch* theo mô-đun 26. Kết quả phép nhân (mod 26) sẽ tạo thành bản mã. Quá trình giải mã sử dụng ma trận nghịch đảo của khóa để thu lại bản rõ ban đầu. Phương pháp này minh họa ứng dụng của đại số tuyến tính vào mã hóa cổ điển[11].

Để mã hóa, ta chia bản rõ thành các khối gồm n ký tự (xem như các vectơ cột có n phần tử) rồi nhân mỗi vectơ với ma trận khóa K kích thước n×n, và lấy kết quả modulo 26. Kết quả được gọi là vectơ bản mã; sau cùng ta chuyển các số 0–25 về ký tự tương ứng.

Để giải mã, ta nhân các block bản mã với ma trận nghịch đảo K^{-1} mod 26 để thu lại bản rõ ban đầu.

Ma trận khóa và điều kiện khả nghịch mod 26:

Ma trận khóa K là một ma trận vuông n×n có các phần tử thuộc $\mathbb{Z}$ (26). Ma trận này phải khả nghịch modulo 26, nghĩa là tồn tại ma trận K^{-1} sao cho $K \times K^{-1} \equiv 1 \bmod 26$. Điều này tương đương với điều kiện gcd(det(K), 26) = 1, tức định thức det(K) không chia hết cho 2 hay 13. Ví dụ, với ma trận $K = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, công thức chung của ma trận nghịch đảo mod 26 là

$$K^{-1} = det(K)^{-1} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} mod\ 26,$$

Với $\det(K)^{-1}$ là nghịch đảo modulo của det(K) = ad – bc, nếu gcd(det(K), 26) ≠ 1, tức det(K) không có nghịch đảo mod 26, thì ma trận K không giải mã duy nhất được.

Quy trình mã hóa:

Để mã hóa bản rõ, các bước cơ bản như sau:

Chuyển đổi ký tự thành số: Dùng quy ước A = 0, B = 1, …, Z = 25 để chuyển mỗi ký tự bản rõ thành số (theo modulo 26)

Chia khối: Chia bản rõ thành các *khối* liên tiếp, mỗi khối gồm n ký tự. Mỗi khối được biểu diễn thành vectơ cột n×1. Nếu chiều dài bản rõ không chia hết, ta thường thêm các ký tự đệm (ví dụ 'X') để đủ khối.

Nhân ma trận: với mỗi vectơ P, tính vectơ C theo công thức:

$$C = K \times P \bmod 26$$

Tức nhân ma trận khóa K với vectơ P rồi lấy modulo 26. (Theo định nghĩa K phải khả nghịch modulo 26 nên phép nhân này có thể đảo ngược được).

Chuyển kết quả: Các phần tử kết quả của vectơ C là các số trong 0–25. Chúng ta chuyển chúng sang chữ cái tương ứng (0→A, …, 25→Z) để tạo thành bản mã.

Quy trình giải mã:

Để giải mã bản mã đã cho, ta cần tính ma trận nghịch đảo K^{-1} của khóa K theo modulo 26 (nếu tồn tại) và thực hiện các bước ngược lại:

Tính ma trận nghịch đảo: Tìm K^{-1} mod 26 thỏa mãn $K \times K^{-1} \equiv 1 \bmod 26$. Đối với ma trận 2×2, sử dụng công thức đã nêu ở trên. Đối với ma trận n×n tổng quát, có thể tính bằng cách tính ma trận liên hợp nhân với nghịch đảo của định thức modulo 26. Lưu ý K^{-1} tồn tại khi và chỉ khi gcd(det(K), 26) = 1.

Chuẩn bị vectơ: Chia bản mã thành các block kích thước n và chuyển thành vectơ cột (sử dụng quy ước A = 0,…).

Nhân nghịch đảo: Với mỗi vectơ bản mã C, tính vectơ bản rõ P theo công thức $P = K^{-1} \times C \bmod 26$.

Chuyển sang ký tự: Chuyển kết quả P (các số 0–25) về chữ cái tương ứng để thu lại bản rõ.

Như vậy, quá trình giải mã về cơ bản dùng ma trận nghịch đảo để đảo ngược phép biến đổi tuyến tính của mã hóa.

Ví dụ minh họa:

Giả sử m = 2 và chọn khóa K = $\begin{pmatrix}3 & 3\\ 2 & 5\end{pmatrix}$. Ta kiểm tra $det(K) = 3 \times 5 - 3 \times 2 = 15 - 6 = 9$. Modulo 26, ta có det(K) = 9. Vì gcd(9, 26) = 1 nên K khả nghịch.

Văn bản "HELP" được chia thành hai cặp: "HE" và "LP". Biểu diễn dưới dạng vectơ số: H=7, E=4 tạo cột $\begin{pmatrix}7\\4\end{pmatrix}$; L=11, P=15 tạo cột $\begin{pmatrix}11\\15\end{pmatrix}$.

Mã hóa từng cột:

Với "HE": $y_1 = K \times \begin{pmatrix}7\\4\end{pmatrix}; = \begin{pmatrix}3 \times 7 + 3 \times 4\\ 2 \times 7 + 5 \times 4\end{pmatrix} = \begin{pmatrix}33\\34\end{pmatrix} \equiv \begin{pmatrix}7\\8\end{pmatrix} \bmod 26.$

Vectơ $\begin{pmatrix}7\\8\end{pmatrix}$ tương ứng chữ cái (7→H, 8→I) nên thu được "HI".

Với "LP": $y_2 = K \times \begin{pmatrix}11\\15\end{pmatrix}; = \begin{pmatrix}78\\97\end{pmatrix} \equiv \begin{pmatrix}0\\19\end{pmatrix} \bmod 26.$

Vectơ $\begin{pmatrix}0\\19\end{pmatrix}$ ứng với (0→A, 19→T) nên thu được "AT".

Ghép hai khối lại, bản mã thu được là "HIAT".

Để giải mã, ta cần tính K^{-1} mod 26. Với Det(K) = 9, tìm nghịch đảo của 9 mod 26 là d sao cho 9×d ≡ 1 mod 26. Thử d = 3, ta có 9×3 = 27 ≡ 1 mod 26 nên $9^{-1} \equiv 3 \bmod 26$.

Từ công thức nghịch đảo ma trận 2x2, ta có:

$$K^{-1} \equiv 3 \times \begin{pmatrix}5 & -3\\ -2 & 3\end{pmatrix} \equiv \begin{pmatrix}15 & 17\\ 20 & 9\end{pmatrix} \bmod 26.$$

Thật vậy, nhân K^{-1} với K ta được ma trận đơn vị:

$$\begin{pmatrix}15 & 17\\ 20 & 9\end{pmatrix} \times \begin{pmatrix}5 & -3\\ -2 & 3\end{pmatrix} \equiv \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \bmod 26.$$

Giải mã "HIAT": biến "HI" thành vectơ $\begin{pmatrix}7\\8\end{pmatrix}$, nhân với K^{-1}:

$\begin{pmatrix}7\\8\end{pmatrix} \rightarrow K^{-1} \times \begin{pmatrix}7\\8\end{pmatrix} = \begin{pmatrix}15 \times 7 + 17 \times 8\\ 20 \times 7 + 9 \times 8\end{pmatrix} = \begin{pmatrix}361\\212\end{pmatrix} \equiv \begin{pmatrix}7\\4\end{pmatrix} \bmod 26 \rightarrow$ "HE"

Tương tự, "AT" → $\begin{pmatrix}0\\19\end{pmatrix}$, nhân với K^{-1} → thu lại $\begin{pmatrix}11\\15\end{pmatrix}$ = "LP"

Vậy văn bản khôi phục đúng là "HELP".

2.2. Một số bài toán ứng dụng

Bài toán 2.3: Mã hóa dòng chữ "LOVEHUMGMATHCLUB" thành một thông điệp bí mật bằng ma trận.

Tóm tắt bài toán:

Mã hóa dòng chữ “LOVEHUMGMATHCLUB”.

Lời giải cho bài toán:

Ta chọn m = 2 và khóa K = $\begin{pmatrix}2 & 3\\1 & 4\end{pmatrix}$. Ta kiểm tra $det(K) = 2 \times 4 - 3 \times 1 = 8 - 3 = 5$. Modulo 26, ta có det(K) = 5. Vì gcd(5, 26) = 1 nên K khả nghịch.

Mã hóa:

Dòng chữ “LOVEHUMGMATHCLUB” được chuyển thành các số:

$L = 11, O = 14, V = 21, E = 4, H = 7, U = 20, M = 13, G = 6,$

$M = 12, A = 0, T = 19, H = 7, C = 2, L = 11, U = 20, B = 1.$

Vectơ biểu diễn:

$$v = \begin{pmatrix}11\\14\\21\\4\\7\\20\\13\\6\\12\\0\\19\\7\\2\\11\\20\\1\end{pmatrix}$$

Vì ta đã chọn m = 2 nên chia v thành các khối 2×1 :

$$v1 = \begin{pmatrix}11\\14\end{pmatrix}, v2 = \begin{pmatrix}21\\4\end{pmatrix}, \ldots, v8 = \begin{pmatrix}20\\1\end{pmatrix}$$

Mã hóa từng khối:

Nhân từng khối với ma trận K: $v_i' = K \times v_i$.

Tính toán:

$$v1' = \begin{pmatrix}2 & 3\\1 & 4\end{pmatrix} \times v1 = \begin{pmatrix}64\\67\end{pmatrix} mod\ 26 = \begin{pmatrix}12\\15\end{pmatrix}$$

$$v2' = \begin{pmatrix}2 & 3\\1 & 4\end{pmatrix} \times v2 = \begin{pmatrix}54\\37\end{pmatrix} mod\ 26 = \begin{pmatrix}2\\11\end{pmatrix}$$

$$v3' = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \times v2 = \begin{pmatrix} 74 \\ 87 \end{pmatrix} mod\ 26 = \begin{pmatrix} 22 \\ 9 \end{pmatrix}$$
$$v4' = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \times v2 = \begin{pmatrix} 44 \\ 37 \end{pmatrix} mod\ 26 = \begin{pmatrix} 18 \\ 11 \end{pmatrix}$$
$$v5' = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \times v2 = \begin{pmatrix} 24 \\ 12 \end{pmatrix} mod\ 26 = \begin{pmatrix} 24 \\ 12 \end{pmatrix}$$
$$v6' = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \times v2 = \begin{pmatrix} 59 \\ 47 \end{pmatrix} mod\ 26 = \begin{pmatrix} 7 \\ 21 \end{pmatrix}$$
$$v7' = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \times v2 = \begin{pmatrix} 37 \\ 46 \end{pmatrix} mod\ 26 = \begin{pmatrix} 11 \\ 20 \end{pmatrix}$$
$$v8' = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \times v8 = \begin{pmatrix} 43 \\ 24 \end{pmatrix} mod\ 26 = \begin{pmatrix} 17 \\ 24 \end{pmatrix}$$

Thông điệp mã hóa:

Kết quả sau khi mã hóa là:

$$v' = \begin{pmatrix} 12 \\ 15 \\ 2 \\ 11 \\ 22 \\ 9 \\ 18 \\ 11 \\ 24 \\ 12 \\ 7 \\ 21 \\ 11 \\ 20 \\ 17 \\ 24 \end{pmatrix}$$

Văn bản sau khi mã hóa: MPCLWJSLYMHVLURY

Giải mã:

Tính ma trận nghịch đảo của K:

$$K = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix},\ det(K) = 5.$$

Nghịch đảo:

$$K^{-1} = \frac{1}{5} \begin{pmatrix} 4 & -3 \\ -1 & 2 \end{pmatrix}$$

Giải mã từng khối:

Nhân các khối giải mã v_i' với K^{-1} để khôi phục v_i

Kết quả sau khi giải mã:

$$v = \begin{pmatrix} 11 \\ 14 \\ 21 \\ 4 \\ 7 \\ 20 \\ 13 \\ 6 \\ 12 \\ 0 \\ 19 \\ 7 \\ 2 \\ 11 \\ 20 \\ 1 \end{pmatrix}$$

Chuyển các số về ký tự sẽ thu được dòng chữ ban đầu:

"LOVEHUMGMATHCLUB".

Nhận xét: Mật mã Hill là một ví dụ tiêu biểu cho việc ứng dụng kiến thức đại số tuyến tính vào lĩnh vực mật mã học. Trong đó, khóa được biểu diễn dưới dạng một ma trận vuông, quá trình mã hóa là phép nhân ma trận giữa khóa và vectơ bản rõ, sau đó lấy modulo 26. Giải mã được thực hiện thông qua việc sử dụng ma trận nghịch đảo của khóa modulo 26, đảm bảo tính song ánh – tức là có thể khôi phục bản rõ từ bản mã.

Trong thực tế, các hệ mật mã tiên tiến như AES, RSA và các thuật toán mã hóa công khai khác tuy không trực tiếp sử dụng ma trận như Hill cipher, nhưng vẫn dựa trên các khái niệm cốt lõi của đại số tuyến tính và lý thuyết số. Ví dụ, AES thực hiện các phép toán trên trường hữu hạn GF (28) với cấu trúc bảng mã , ma trận hoán vị và các bước nhân ma trận trong quá trình biến đổi trạng thái.

Các ứng dụng của ma trận trong bảo mật thông tin còn thể hiện rõ trong lĩnh vực mã sửa lỗi, thông qua các khái niệm như ma trận sinh và ma trận kiểm tra.

Thuật toán RSA trong mật mã công khai cũng liên quan đến lý thuyết số và đại số tuyến tính (không dùng ma trận trực tiếp nhưng dùng modulo và tính toán

tương tự). Nhìn chung, hiểu biết về ma trận và nghịch đảo modulo giúp ta tiếp cận lĩnh vực bảo mật thông tin một cách thuận lợi.

2.3. Các điểm yếu của mật mã Hill

Mặc dù Mật mã Hill là một cải tiến so với các mật mã thay thế đơn bảng chữ cái và đa bảng chữ cái đơn giản hơn, nó vẫn có những điểm yếu cố hữu, chủ yếu do bản chất tuyến tính của nó.

Điểm yếu chính của Mật mã Hill là tính nhạy cảm với tấn công biết trước bản rõ (known-plaintext attack). Nếu kẻ tấn công có được một số cặp bản rõ và bản mã tương ứng, họ có thể thiết lập một hệ phương trình tuyến tính để giải tìm ma trận khóa K. Cụ thể, nếu kẻ tấn công có n khối P_1, P_2, …, P_n (mỗi khối là vectơ $n \times 1$) và các khối bản mã tương ứng C_1, C_2, …, C_n thì họ có thể tạo thành các ma trận $P = [P_1P_2…P_n]$ và $C = [C_1C_2…C_n]$ (Kích thước $n \times n$). Khi đó mối quan hệ $C \equiv KP \pmod m$ (với m thường là kích thước bảng chữ cái, thường là 26) có thể giải để tìm được K bằng cách tính $K \equiv CP^{-1} \pmod m$ miễn là P khả nghịch theo modulo m. Như vậy chỉ cần n cặp khối bản rõ-mã (tương đương n^2 kí tự) là đủ để phá vỡ mật mã.

Do tính tuyến tính này, Mật mã Hill không cung cấp sự khuếch tán và hỗn độn tốt, vốn là các nguyên tắc thiết kế quan trọng cho các mật mã hiện đại. Các mật mã hiện đại như AES (Advanced Encryption Standard) sử dụng các phép toán phi tuyến tính phức tạp hơn nhiều, các vòng lặp và các phép biến đổi khác nhau để chống lại các cuộc tấn công phân tích tuyến tính và vi phân

3. Ứng dụng ma trận trong thuật toán PageRank

Vào năm 1998, Larry Page và Sergey Brin đã giới thiệu khái niệm PageRank một thuật toán cốt lõi trong công cụ tìm kiếm Google. Khác với các phương pháp truy xuất thông tin truyền thống vốn chủ yếu dựa trên việc khớp từ khóa hoặc mẫu văn bản, PageRank đề xuất một cách tiếp cận hoàn toàn mới: đánh giá "tầm quan trọng" của một trang web dựa trên cấu trúc liên kết của toàn bộ mạng Web. Cụ thể, một trang được coi là quan trọng nếu nó được nhiều trang khác trỏ liên kết tới, và giá trị của mỗi liên kết lại phụ thuộc vào độ quan trọng của chính các trang liên kết đó[1]. Ý tưởng cốt lõi của PageRank là mô hình hóa

hành vi người dùng web như một chuỗi Markov ngẫu nhiên "lướt web", và sử dụng vectơ riêng trội của một ma trận để xác định độ quan trọng của mỗi trang.

3.1. Nguyên lý thuật toán PageRank

Ý tưởng cốt lõi: "Một trang web được coi là quan trọng nếu các trang web quan trọng khác liên kết đến nó."

Hãy tưởng tượng một người dùng "lướt web ngẫu nhiên" (random surfer): Người này bắt đầu ở một trang bất kỳ.

Với xác suất d (damping factor, thường là 0.85), người này sẽ chọn ngẫu nhiên một liên kết trên trang hiện tại để đi đến trang tiếp theo.

Với xác suất (1-d), người này sẽ "chán" và nhảy ngẫu nhiên đến bất kỳ trang nào trên Internet (bao gồm cả trang hiện tại). Hành vi này giúp thuật toán hội tụ và tránh các "lỗ đen" (black holes) nơi các trang chỉ liên kết với nhau mà không liên kết ra ngoài.

Điểm PageRank của một trang web là xác suất mà người lướt web ngẫu nhiên sẽ dừng lại ở trang đó sau một số lượng lớn bước di chuyển

Mô hình mạng web: Hãy tưởng tượng web như một đồ thị có hướng, trong đó mỗi trang web là một đỉnh (node) và mỗi liên kết (hyperlink) từ trang này sang trang khác là một cung có hướng. Một trang có nhiều trang khác liên kết tới được coi là quan trọng, và nếu các trang trỏ tới nó cũng quan trọng thì nó càng được đánh giá cao. PageRank đưa ra công thức định lượng cho ý tưởng này.

Công thức PageRank: Gọi PR(i) là điểm PageRank của trang i. Giả sử trang j có các liên kết trỏ đến trang i. Ký hiệu L(j) là số liên kết ra ngoài từ trang j. Khi đó, trang j "đóng góp" cho trang i một lượng $\frac{PR(j)}{L(j)}$. Tổng hợp ảnh hưởng từ tất cả trang trỏ tới i, ta có công thức:

$$PR(i) = \sum_{j \in M(i)} \frac{PR(j)}{L(j)},$$

Trong đó M(i) là tập tất cả trang có liên kết đến trang i. Công thức này mang tính chất đệ quy – điểm của một trang phụ thuộc điểm của các trang trỏ đến nó.

Tuy nhiên, trong thực tế cần hiệu chỉnh để mô hình hành vi người dùng. Người dùng có một xác suất 1 - d để ngẫu nhiên truy cập một trang bất kỳ (nhập

URL trực tiếp chẳng hạn) thay vì chỉ theo liên kết. Tham số d (damping factor) khoảng 0.85 thường được dùng. Khi đó công thức PageRank được điều chỉnh:

$$PR(i) = \frac{1-d}{N} + d \sum_{j \in M(i)} \frac{PR(j)}{L(j)},$$

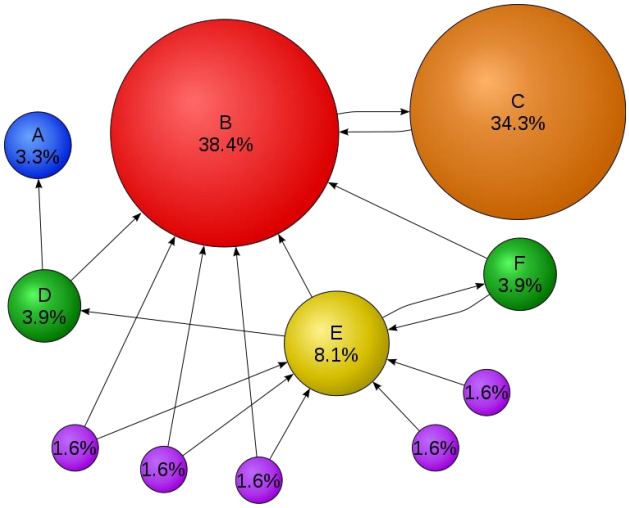
với N là tổng số trang trong mạng. Thành phần $\frac{1-d}{N}$ đảm bảo ngay cả trang không có backlink nào cũng nhận được một giá trị cơ bản từ "random surfer" (người lướt web ngẫu nhiên). Viết dưới dạng ma trận, nếu đánh số các trang 1 đến N, ta xây dựng ma trận chuyển P kích thước N × N với phần tử $pij = \frac{1}{L(j)}$ nếu có liên kết từ trang j đến trang i, và $p_{ij} = 0$ nếu không. Khi đó mỗi cột j của P biểu diễn phân phối xác suất nhảy từ trang j sang các trang khác. Ma trận này sau khi điều chỉnh damping trở thành ma trận G gọi là Google matrix:

$$G = dP + \frac{1-d}{N}E$$

Trong đó E là ma trận N × N với mọi phần tử đều bằng $\frac{1}{N}$. Ma trận G là ma trận stochastic (mỗi cột cộng lại 1) và không có cột nào toàn 0 (damping xử lý trường hợp trang không có link ra). Theo lý thuyết Perron–Frobenius, ma trận stochastic nguyên dương như G có trị riêng lớn nhất bằng 1 và tồn tại vectơ riêng trội ứng với trị riêng 1 mà mọi thành phần đều dương. Vectơ riêng đó chính là vectơ PageRank $R = \begin{pmatrix} PR(1) \\ PR(2) \\ \vdots \\ PR(N) \end{pmatrix}$, thỏa mãn: $GR = R$.

Nói cách khác, R là vectơ trạng thái dừng của ma trận chuyển G. Ta có thể tìm R bằng cách giải phương trình eigen trên, hoặc dùng thuật toán lặp: xuất phát từ một vectơ xấp xỉ (ví dụ phân phối đều $\frac{1}{N}$), liên tục tính vector mới $R^{(k+1)} = GR^k$ cho đến khi hội tụ (sai khác đủ nhỏ). Thuật toán này thường rất nhanh do ma trận G có khoảng cách trị riêng (eigengap) đủ lớn. Ý nghĩa: Giá trị PageRank của mỗi trang có thể hiểu như xác suất người dùng truy cập vào trang đó nếu họ lướt web một cách ngẫu nhiên lâu vô hạn (với xác suất nhảy liên kết d và ngẫu nhiên chọn trang 1−d). Điểm PageRank càng cao nghĩa là trang đó càng dễ được ghé thăm, hay nói cách khác, có tầm quan trọng lớn trong mạng. Thuật toán PageRank đánh giá trang một cách khách quan dựa trên cấu trúc liên kết,

không phụ thuộc nội dung, do đó khắc phục được việc spam từ khóa. Googc để xếp hạng kết quả tìm kiếm hiệu quả.để khắc phục được việc spam từ khóa. Google đã kết hợp PageRank với các yếu tố khác để xếp hạng kết quả tìm kiếm hiệu quả.



Hình 4: Ví dụ minh hoạ PageRank [11]

Trong ví dụ minh họa trên, ta có 11 trang web. Trang web B có nhiều trang web liên kết với nó nhất, do đó rank của trang web B đương nhiên là sẽ cao nhất. Trang web C mặc dù số trang web liên kết với nó ít hơn trang E nhưng mà nó được trang B liên kết tới, do đó nó sẽ có thứ hạng cao hơn trang E. Như vậy kết quả thứ hạng cuối cùng của một trang web dựa trên cấu trúc liên kết của toàn bộ các trang web.

3.2. Mô phỏng thuật toán PageRank

Bài toán 2.4: Trong một nhóm nhỏ trên facebook có 5 người chơi chung với nhau là An, Bình, Châu, Dũng và Huyền. Sau một quá trình quan sát thử nghiệm thì họ thu thập được dữ liệu tương tác sau:

An hay thả tim bình luận bài của Bình, Châu

Bình hay tương tác với bài viết của An và Dũng

Châu thì hay tương tác chủ yếu với An

Còn Dũng thì năng nổ tương tác với cả nhóm

Nhưng Huyền thì chỉ tương tác qua lại với Dũng

Và ta thấy mỗi hành vi trên có thể xem như một liên kết có hướng, người tương tác đến người nhận tương tác

Câu hỏi đặt ra: Ai là người có độ ảnh hưởng cao nhất trong nhóm?

Tóm tắt bài toán*:* Bài toán này mô tả một kịch bản tương tác xã hội trong một nhóm nhỏ trên Facebook, nơi có 5 thành viên là An, Bình, Châu, Dũng và Huyền. Mục tiêu là đánh giá mức độ ảnh hưởng hay tầm quan trọng của mỗi người dựa trên hành vi tương tác của họ.

Các hành vi tương tác như "thả tim", "bình luận", hay "tương tác với bài viết" được xem là các liên kết có hướng:

Người tương tác là nguồn (người tạo liên kết).

Người nhận tương tác là đích (người được liên kết đến).

Cụ thể về các liên kết:

An → Bình, Châu

Bình → An, Dũng

Châu → An

Dũng → An, Bình, Châu, Huyền (tương tác với cả nhóm trừ chính Dũng)

Huyền ↔ Dũng (tức Huyền → Dũng và Dũng → Huyền)

Lời giải cho bài toán: Ta sẽ áp dụng thuật toán pagerank để xác định độ ảnh hưởng của mỗi người trong nhóm đó.

Xây dựng Ma trận kề (Adjacency Matrix) A:

Ma trận A có kích thước 5×5. Phần tử A_{ij} =1 nếu có liên kết từ người j đến người i, và = 0 nếu không

Khi đó ta sẽ tạo được ma trận $A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$

Xây dựng Ma trận xác suất chuyển đổi (Transition Matrix) M:

Ta sẽ gán thứ tự từng người dùng:

An = 0, Bình = 1, Châu = 2, Dũng = 3, Huyền = 4

Khi đó ta sẽ tạo được ma trận $P = \begin{pmatrix} 0 & 0.5 & 1 & 0.25 & 0 \\ 0.5 & 0 & 0 & 0.25 & 0 \\ 0.5 & 0 & 0 & 0.25 & 0 \\ 0 & 0.5 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.25 & 0 \end{pmatrix}$

Thiết lập Vector PageRank ban đầu PR^0 :

Với 5 thành viên, mỗi người được gán PageRank ban đầu bằng nhau:

$$1 / N = 1 / 5 = 0.2.$$

Vậy

$$PR^0 = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}$$

Thực hiện phương pháp lặp:

Vector dịch chuyển ngẫu nhiên là một vector cột mà tất cả các phần tử đều là $\frac{1}{N}$:

$$v = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}$$

Với d = 0.85 (hệ số tắt dần), thì 1 − d = 0.15.

Chúng ta sẽ sử dụng công thức lặp PageRank:

$$PR_{k+1} = (1 - d) \times v + d \times P \times PR_k$$

Với k = 30 khi đó ta được mức độ quan trọng của mỗi người trong nhóm là:

An: 0.33592, Bình: 0.21093, Châu: 0.21135, Dũng: 0.17718, Huyền: 0.06862.

Khi đó ta thấy An có PageRank cao nhất.

Huyền có ảnh hưởng thấp nhất.

Nhận xét: Có thể mở rộng để đánh giá độ lan truyền thông tin,…

Bài toán 2.5. Tính PageRank cho một mạng web đơn giản gồm 4 trang A, B, C, D với cấu trúc liên kết như sau (X -> Y nghĩa là trang X liên kết đến trang Y), 4 trang khởi tạo đồng đều:

Trang A liên kết đến B và C. (Số liên kết ra của A, L(A) = 2)

Trang B liên kết đến A và C. (Số liên kết ra của B, L(B) = 2)

Trang C liên kết đến A. (Số liên kết ra của C, L(C) = 1)

Trang D liên kết đến A, B, C. (Số liên kết ra của D, L(D) = 3)

Hệ phương trình PageRank (với hệ số tắt d=0.85) sẽ là:

$PR(A) = \frac{1-d}{4} + d(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)})$

(vì các trang B, C, D liên kết đến A)

$PR(B) = \frac{1-d}{4} + d(\frac{PR(D)}{L(D)})$ (vì chỉ có trang D liên kết đến B)

$PR(C) = \frac{1-d}{4} + d(\frac{PR(A)}{L(A)} + \frac{PR(B)}{L(B)} + \frac{PR(D)}{L(D)})$ (vì các trang A, B, D liên kết đến C)

$PR(D) = \frac{1-d}{4} + d(0)$ (vì không có trang nào liên kết đến D)

Thay d = 0.85 và các giá trị $L()$ tương ứng vào, giải hệ phương trình trên ta sẽ tìm được giá trị PageRank của từng trang.

Ta xây dựng ma trận $P = \begin{pmatrix} 0 & 0.5 & 1 & 1/3 \\ 0.5 & 0 & 0 & 1/3 \\ 0.5 & 0.5 & 0 & 1/3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Và vectơ rank ban đầu $PR(0) = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$

Ta tính P $\times PR(0) = \begin{bmatrix} 0.4583 \\ 0.2083 \\ 0.3333 \\ 0.0001 \end{bmatrix}$

Cứ tiếp tục như vậy.

Kết quả tính PageRank cho mạng web đơn giản 4 trang là: $PR(A) \approx 0.4174$, $PR(B) \approx 0.2252, PR(C) \approx 0.3210$, $PR(D) \approx 0.0384$. Tổng các giá trị này xấp xỉ 1 (đúng như tính chất xác suất). Xếp hạng các trang từ cao đến thấp là: A > C > B > D. Trang A có điểm cao nhất (~ 45.74%), bởi vì A nhận liên kết từ B, C, D (3 trang khác), trong đó liên kết từ D và C đóng góp đáng kể vào PageRank của A. Trang C có điểm ~ 32.10%, nhận link từ A, B, D. Trang B có điểm ~ 22.52%, chỉ được D trỏ tới. Trang D có điểm thấp nhất ~03.84% do không có trang nào liên kết đến D. Kết quả này phù hợp với nhận định *trực quan*: trang nào được nhiều trang khác có thứ hạng cao trỏ đến thì sẽ có thứ hạng cao.

Ta có thể kiểm tra thủ công tính hội tụ của thuật toán: bắt đầu với phân phối PageRank đều cho các trang (mỗi trang 0.25). Lặp lại phép tính $Rnew = \frac{1-d}{N} \times 1 + d \times PT \times Rold$ (với d = 0.85, N = 4, 1 là vectơ cột chứa toàn số 1, P^T là ma trận chuyển vị của ma trận chuyển) qua vài bước lặp

Hội tụ khoảng sau ~20 bước lặp, vectơ PageRank xấp xỉ là [0.417, 0.225, 0.321, 0.038] như đã trình bày ở trên.

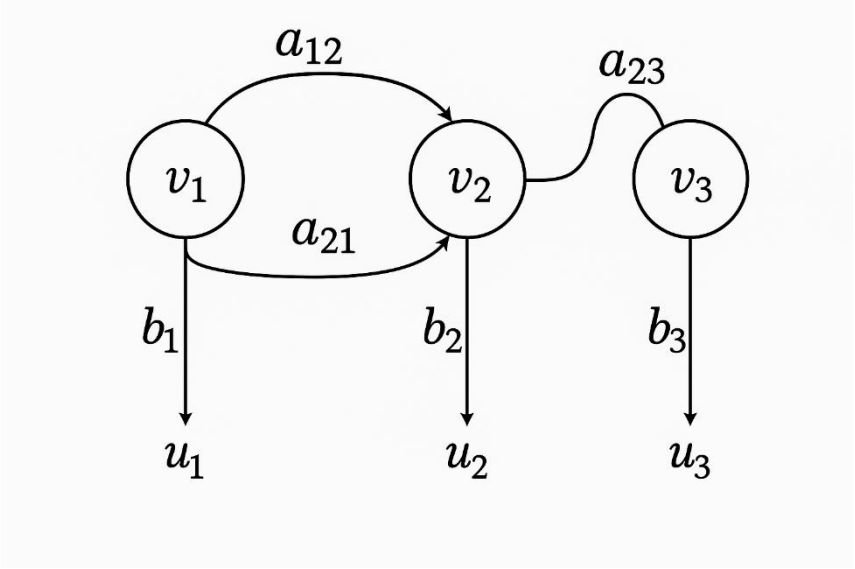
3.3. Hạn chế và sự phát triển của PageRank

Mặc dù PageRank là một thuật toán nền tảng và mang tính cách mạng, nó cũng có những hạn chế và đã trải qua nhiều sự phát triển kể từ khi ra đời. Một trong những hạn chế chính là tính dễ bị thao túng thông qua các kỹ thuật như "link farms" (trang trại liên kết) hoặc "Google bombing" (tạo ra một lượng lớn liên kết với cùng một anchor text để đẩy thứ hạng một trang cho một từ khóa cụ thể). Google và các công cụ tìm kiếm khác đã liên tục cập nhật thuật toán của mình để chống lại các hành vi thao túng này.

Ngày nay, các công cụ tìm kiếm hiện đại không chỉ dựa vào PageRank mà sử dụng hàng trăm yếu tố khác nhau để xếp hạng trang web. Các yếu tố này bao gồm mức độ liên quan của nội dung (content relevance), chất lượng nội dung, trải nghiệm người dùng, tốc độ tải trang, tính thân thiện với thiết bị di động, và các yếu tố E-A-T (Expertise, Authoritativeness, Trustworthiness - Chuyên môn, Thẩm quyền, Độ tin cậy). Sự phát triển của trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên cũng đã cho phép các công cụ tìm kiếm hiểu rõ hơn về ngữ nghĩa của truy vấn tìm kiếm và nội dung trang web, thay vì chỉ dựa vào từ khóa. Do đó, PageRank hiện chỉ là một trong nhiều tín hiệu được sử dụng trong một hệ thống xếp hạng phức tạp hơn nhiều.

4. Chuỗi Markov và mô hình hóa hệ thống xác suất

Chuỗi Markov là một công cụ mạnh mẽ để mô hình hóa các quá trình ngẫu nhiên. Nhiều hệ thống thực tế có thể được xấp xỉ bằng chuỗi Markov – từ dự báo thời tiết, hành vi khách hàng, đến quản lý hàng tồn kho. Trong chương này, ta sẽ trình bày khái niệm chuỗi Markov, khái niệm trạng thái dừng và minh họa bằng hai ứng dụng: dự báo thời tiết và phân tích hành vi khách hàng.



Hình 5: Ví dụ về chuỗi Markov

4.1. Chuỗi Markov và mô hình xác suất.

***Định nghĩa chuỗi Markov**:* Chuỗi Markov (Markov Chain) là một mô hình xác suất mô tả sự chuyển đổi giữa các trạng thái trong một hệ thống theo thời gian, trong đó xác suất chuyển đổi sang trạng thái kế tiếp chỉ phụ thuộc vào trạng thái hiện tại và không phụ thuộc vào lịch sử các trạng thái trước đó.

Tính chất Markov:

Tính chất đặc trưng nhất của chuỗi Markov là:

$$P(X_{n+1} = s_j \mid X_n = s_i, X_{n-1} = s_k, ...) = P(X_{n+1} = s_j \mid X_n = s_i) \; \forall \text{ trạng thái } s_i, s_j.$$

Tính chất này giúp mô hình Markov trở nên đơn giản và khả thi khi mô hình hóa các quá trình phức tạp.

Thành phần chính của chuỗi Markov:

Tập trạng thái: Tập hợp tất cả các trạng thái mà hệ thống có thể tồn tại. Tập này có thể hữu hạn hoặc vô hạn.

Xác suất chuyển tiếp: Xác suất từ một trạng thái hiện tại chuyển sang trạng thái khác trong bước tiếp theo.

Ma trận chuyển tiếp: Biểu diễn xác suất chuyển tiếp dưới dạng ma trận vuông, mỗi phần tử P_{ij} là xác suất chuyển từ trạng thái i sang trạng thái j.

Ma trận chuyển trạng thái: Chuỗi Markov thường được đặc trưng bởi ma trận xác suất chuyển P. Giả sử có N trạng thái có thể có là S_1, S_2, ..., S_N. Ma trận P kích thước N×N, trong đó phần tử $p_{ij} = P(X_{t+1} = s_j \mid X_t = s_i)$ là xác suất chuyển

từ trạng thái i sang trạng thái j trong một bước. Mỗi hàng i của ma trận là một phân phối xác suất trên các trạng thái kế tiếp, do đó $\sum_{j=1}^{N} p_{ij} = 1$ với mọi i. (Một số tài liệu dùng ma trận chuyển P^T dạng cột, khi đó tổng các phần tử trong mỗi cột bằng 1, nhưng bản chất mô hình không thay đổi).

Phân phối trạng thái: Giả sử ta có phân phối xác suất trạng thái ban đầu $\pi^{(0)}$ (là một vectơ hàng độ dài N với $\pi_i^{(0)} = P(X_0 = S_i)$ và $\sum \pi_i^{(0)}=1$). Tổng quát, phân phối trạng thái sau n bước là $\pi^{(n)} = \pi^{(0)} P^n$.

Trạng thái dừng: Trong nhiều chuỗi Markov, tồn tại một phân phối xác suất π sao cho nếu hệ đang ở phân phối π và sau một bước chuyển cũng vẫn ở phân phối π. Tức là π thỏa mãn phương trình cân bằng π = πP. Thì được giá trị là phân phối dừng hay phân phối ổn định. Pân phối dừng này thể hiện tỷ lệ thời gian dài hạn mà quá trình ở mỗi trạng thái, hoặc là phân phối xác suất mà hệ thống sẽ tiến tới sau một thời gian dài hoạt động, bất kể trạng thái ban đầu (đối với các chuỗi Markov ergodic).

4.2. Ứng dụng chuỗi markov

Bài toán 2.6: Mô hình thời thời tiết chỉ có hai trạng thái: Nắng (S1) và Mưa (S2). Giả sử dựa trên số liệu thống kê, ta có ma trận chuyển sau: $P = \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix}$, nghĩa là nếu hôm nay nắng, xác suất mai tiếp tục nắng là 90%, mưa 10%. Nếu hôm nay mưa, xác suất mai nắng là 50%, mưa tiếp 50%. Nếu hôm nay nắng, hãy dự báo thời tiết sau 3 ngày.

Dự báo ngắn hạn: Nếu hôm nay nắng, phân phối trạng thái ngày mai là [0.9, 0.1] (90% nắng, 10% mưa). Ngày mốt (2 ngày sau) sẽ có phân phối = [0.9,0.1] × P = [0.90.9 + 0.10.5, 0.90.1 + 0.10.5] = [0.86, 0.14] (86% nắng, 14% mưa). Tiếp tục tính vài bước:

Ở ngày thứ 3: $v_3 = v_2 \times P$ = [0.844, 0.156]

Ta thấy xác suất dự báo ngày càng "ổn định" quanh ~83.3% nắng, 16.7% mưa. Giá trị này chính là phân phối dừng của chuỗi Markov. Ta có thể giải phương trình dừng π = πP để xác nhận:

Giả sử phân phối dừng là π = (x, 1−x), với x là xác suất ở trạng thái Nắng. Điều kiện dừng π = πP tương đương với:

$$(x, 1 - x)\begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix} = (x, 1 - x)$$

Tức là hệ phương trình:

$$\begin{cases} 0.9x + 0.5(1 - x) = x \\ 0.1x + 0.5(1 - x) = 1 - x \end{cases}$$

Giải hệ phương trình này ta được $x = \frac{5}{6} \approx 0.8333$. Vậy phân phối dừng là $\pi = (\frac{5}{6}, \frac{1}{6}) \approx (0.8333, 0.1667)$. Kết quả này khớp với kết quả mô phỏng sự hội tụ. Điều này nghĩa là về dài hạn, trung bình khoảng 83.33% số ngày là nắng và 16.67% ngày mưa (theo mô hình đã chọn).

Bài toán 2.7: Hành vi khách hàng trong siêu thị

Trong một siêu thị, hành vi khách hàng được mô hình hóa qua 3 trạng thái:

S1: Nhìn quanh (Browse)

S2: Mua hàng (Buy)

S3: Rời khỏi siêu thị (Exit)

Ma trận chuyển trạng thái: $P = \begin{pmatrix} 0.6 & 0.3 & 0.1 \\ 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$

"Lưu ý rằng trong mô hình này, trạng thái S3 ('Rời khỏi siêu thị') là một trạng thái hấp thụ. Điều này có nghĩa là một khi khách hàng đã vào trạng thái S3 (rời khỏi siêu thị), họ sẽ không chuyển sang bất kỳ trạng thái nào khác ($P_{33} = 1$).

Đây là lý do tại sao phân phối dừng của chuỗi Markov này cho thấy toàn bộ xác suất cuối cùng sẽ tập trung ở trạng thái S3"

Yêu cầu: Nếu khách hàng bắt đầu từ trạng thái Nhìn quanh (Browse), xác suất họ rời siêu thị sau 3 bước là bao nhiêu?

Trạng thái ban đầu $v_0 = [1, 0, 0]$

Bước 1: $v_1 = v_0 P = [0.6, 0.3, 0.1]$

Bước 2: $v_2 = v_1 P = [0.36, 0.33, 0.31]$

Bước 3: $v_3 = v_2 P = [0.216, 0.273, 0.511]$

Vậy sau 3 bước khách hàng đã rời siêu thị là 51.1%

Ứng dụng thực tế: Mô hình thời tiết 2 trạng thái này tuy đơn giản, nhưng nguyên tắc có thể mở rộng cho nhiều trạng thái phức tạp hơn. Ví dụ này đã minh họa việc dùng ma trận chuyển để tính toán phân phối xác suất trạng thái.

CHƯƠNG 3: ỨNG DỤNG LẬP TRÌNH ĐỂ GIẢI TOÁN

1. Tại sao lại sử dụng Python để giải toán và mô phỏng?

Việc giải quyết các bài toán đại số tuyến tính ứng dụng trong Chương 2 bằng phương pháp thủ công gặp nhiều hạn chế về độ phức tạp và quy mô. Do đó, nhóm đã lựa chọn ngôn ngữ lập trình Python làm công cụ để giải quyết và mô phỏng hiệu quả các bài toán này.

Lý do chọn Python:

Ngôn ngữ mạnh mẽ và phổ biến: Python được ưa chuộng trong khoa học dữ liệu và tính toán khoa học nhờ cú pháp rõ ràng, dễ tiếp cận.

Thư viện hỗ trợ đa dạng: Các thư viện như NumPy (tính toán ma trận), SciPy (thuật toán tối ưu, giải hệ phương trình), SymPy (tính toán ký hiệu) và Matplotlib (trực quan hóa) cung cấp nền tảng vững chắc cho việc triển khai các thuật toán.

Kết nối lý thuyết và thực hành: Python giúp trực quan hóa và hiện thực hóa các khái niệm lý thuyết, làm sâu sắc thêm hiểu biết về thuật toán.

Ý nghĩa và ưu thế so với giải toán thông thường:

Hiệu quả với bài toán lớn: Python xử lý nhanh chóng và chính xác các bài toán phức tạp, quy mô lớn mà giải tay không khả thi.

Tự động hóa và giảm sai sót: Tự động hóa các quy trình tính toán lặp đi lặp lại, giảm thiểu sai sót do con người.

Linh hoạt thử nghiệm: Dễ dàng thay đổi tham số, thử nghiệm các kịch bản khác nhau để khám phá bài toán.

Phát triển kỹ năng: Rèn luyện tư duy thuật toán, kỹ năng giải quyết vấn đề và chuẩn bị cho công việc thực tế.

2. Minh họa một số bài toán bằng ngôn ngữ lập trình Python

Phần này trình bày việc áp dụng Python để giải quyết các bài toán cụ thể từ Chương 2, nêu bật những cải tiến và định hướng của nhóm.

2.1. Tối ưu hóa các bài toán thực tế

Định hướng của nhóm: Nhận thấy hạn chế của phương pháp hình học với các bài toán nhiều biến, nhóm hướng đến việc triển khai thuật toán **đơn hình hai pha** (Two-Phase Simplex Method)[10] bằng Python để giải quyết các bài toán quy hoạch tuyến tính tổng quát (cả cực đại và cực tiểu).

Phương pháp và Kết quả: Nhóm đã phát triển một lớp *Linear Programming* trong Python, tận dụng hàm linprog từ thư viện *SciPy* (với phương thức *highs*) để giải các bài toán quy hoạch tuyến tính. Chương trình cho phép người dùng nhập các hệ số của hàm mục tiêu, ma trận ràng buộc và các giới hạn một cách linh hoạt. Chi tiết về cách thức triển khai và mã nguồn đầy đủ có thể tham khảo tại **Phụ lục A.1**. Chương trình đã được áp dụng để giải thành công Bài toán 2.1 (Tối đa hóa thu nhập) và Bài toán 2.2 (Tối thiểu hóa chi phí). Kết quả thu được từ Python không chỉ nhanh chóng, chính xác mà còn giúp kiểm chứng lại các giải pháp tìm được bằng phương pháp hình học trước đó.

```
Nhập dạng bài toán ('max' hoặc 'min'): max
Nhập số lượng biến: 2
Nhập các hệ số của hàm mục tiêu:
  Hệ số của x1: 75
  Hệ số của x2: 27
Nhập số lượng ràng buộc: 4
Nhập chi tiết cho từng ràng buộc:

Ràng buộc 1:
  Hệ số của x1 trong ràng buộc 1: 1
  Hệ số của x2 trong ràng buộc 1: 1
  Nhập dấu của ràng buộc ('<=', '>=', '='): <=
  Nhập giá trị vế phải của ràng buộc 1: 12

Ràng buộc 2:
  Hệ số của x1 trong ràng buộc 2: 2
  Hệ số của x2 trong ràng buộc 2: 0.4
  Nhập dấu của ràng buộc ('<=', '>=', '='): <=
  Nhập giá trị vế phải của ràng buộc 2: 16

Ràng buộc 3:
  Hệ số của x1 trong ràng buộc 3: 1
  Hệ số của x2 trong ràng buộc 3: 0
  Nhập dấu của ràng buộc ('<=', '>=', '='): >=
  Nhập giá trị vế phải của ràng buộc 3: 0

Ràng buộc 4:
  Hệ số của x1 trong ràng buộc 4: 0
  Hệ số của x2 trong ràng buộc 4: 1
  Nhập dấu của ràng buộc ('<=', '>=', '='): >=
  Nhập giá trị vế phải của ràng buộc 4: 0

Trạng thái: Optimal solution found
Nghiệm tối ưu: [7. 5.]
Giá trị mục tiêu tối ưu: 659.9999999999999
```

Hình 6: Kết quả chương trình Python cho Bài toán 2.1 về tối đa hóa thu nhập

```
Nhập dạng bài toán ('max' hoặc 'min'): min
Nhập số lượng biến: 2
Nhập các hệ số của hàm mục tiêu:
  Hệ số của x1: 150
  Hệ số của x2: 250
Nhập số lượng ràng buộc: 3
Nhập chi tiết cho từng ràng buộc:

Ràng buộc 1:
  Hệ số của x1 trong ràng buộc 1: 20
  Hệ số của x2 trong ràng buộc 1: 30
  Nhập dấu của ràng buộc ('<=', '>=', '='): >=
  Nhập giá trị vế phải của ràng buộc 1: 110

Ràng buộc 2:
  Hệ số của x1 trong ràng buộc 2: 1
  Hệ số của x2 trong ràng buộc 2: 0
  Nhập dấu của ràng buộc ('<=', '>=', '='): >=
  Nhập giá trị vế phải của ràng buộc 2: 1

Ràng buộc 3:
  Hệ số của x1 trong ràng buộc 3: 0
  Hệ số của x2 trong ràng buộc 3: 1
  Nhập dấu của ràng buộc ('<=', '>=', '='): >=
  Nhập giá trị vế phải của ràng buộc 3: 1

Trạng thái: Optimal solution found
Nghiệm tối ưu: [4. 1.]
Giá trị mục tiêu tối ưu: 850.0
```

Hình 7: Kết quả chương trình Python cho Bài toán 2.2 về tối thiểu hóa chi phí

2.2. Mã hóa văn bản

Định hướng của nhóm: Mật mã Hill là một ví dụ kinh điển về ứng dụng ma trận trong mật mã học. Tuy nhiên, phiên bản gốc thường bị giới hạn bởi bảng chữ cái cố định. Nhóm đã đặt mục tiêu mở rộng thuật toán để có thể mã hóa văn bản đa ngôn ngữ (sử dụng Unicode), tự động sinh ma trận khóa khả nghịch, và tăng cường thêm một chút tính an toàn cho bản mã thông qua padding ngẫu nhiên và mã hóa Base64, đồng thời đảm bảo khả năng giải mã chính xác.

Phương pháp và kết quả: Một lớp *HillCipher* cùng các hàm hỗ trợ như *get_dynamic_alphabet_from_text* (để tạo bảng chữ cái từ chính thông điệp) và *generate_invertible_key_matrix* (để sinh khóa khả nghịch sử dụng *SymPy* và *numpy.gcd*) đã được phát triển. Mã nguồn chi tiết được trình bày tại **Phụ lục A.2**. Chương trình đã được thử nghiệm với bài toán 2.3, cho thấy khả năng mã hóa và

giải mã thành công. Những cải tiến này minh họa sự linh hoạt của lập trình trong việc tùy biến và mở rộng các thuật toán mật mã cổ điển.

```
--- MẬT MÃ HILL MỞ RỘNG ---
Nhập thông điệp cần mã hóa (hỗ trợ Unicode): LOVEHUMGMATHCLUB
Bảng chữ cái tự động tạo (12 ký tự): ABCEGHLMOTUV
Ma trận khóa khả nghịch ngẫu nhiên (mod 12): [[7, 9], [4, 11]]

--- KẾT QUẢ ---
Thông điệp gốc: LOVEHUMGMATHCLUB
Bản mã hóa (Base64): MDE2TEdPSEhVQkFCR0FNT0NNRQ==
Bản giải mã: LOVEHUMGMATHCLUB
```

Hình 8: Kết quả chương trình Python cho Bài toán 2.3 Mật mã Hill

2.3. Thuật toán PageRank

Định hướng của nhóm: Việc tính toán PageRank thủ công cho các mạng lưới lớn là không khả thi. Do đó, nhóm đã xây dựng một công cụ Python cho phép người dùng định nghĩa một mạng lưới (các nút và liên kết có hướng) một cách linh hoạt. Chương trình sẽ tự động xây dựng ma trận kề, xử lý các nút không có liên kết ra (dangling nodes), chuẩn hóa thành ma trận chuyển xác suất, và triển khai phương pháp lặp lũy thừa (Power Iteration) để tìm vector PageRank hội tụ.

Phương pháp và kết quả: Hàm *calculate_pagerank* là trung tâm của việc tính toán. Ngoài ra, chương trình còn tích hợp thư viện *networkx* và *matplotlib* để trực quan hóa đồ thị mạng lưới cùng với giá trị PageRank của từng nút. Mã nguồn hoàn chỉnh có thể được tìm thấy tại **Phụ lục A.3**. Ứng dụng thuật toán này, chương trình đã tính toán và xếp hạng PageRank cho các đối tượng trong Bài toán 2.4 (mô phỏng tương tác nhóm Facebook) và Bài toán 2.5 (mạng web 4 trang). Kết quả không chỉ cho thấy tính đúng đắn của thuật toán mà còn minh họa khả năng ứng dụng cho các loại mạng khác nhau.

```
Nhập số lượng đỉnh (node): 5
Nhập tên các đỉnh:
  Đỉnh 1: AN
  Đỉnh 2: BÌNH
  Đỉnh 3: CHÂU
  Đỉnh 4: DŨNG
  Đỉnh 5: HUYỀN

Nhập các liên kết có hướng (ví dụ: An -> Bình). Gõ 'done' để kết thúc.
Liên kết: AN -> BÌNH
Liên kết: AN -> CHÂU
Liên kết: BÌNH -> AN
Liên kết: BÌNH -> DŨNG
Liên kết: CHÂU -> AN
Liên kết: DŨNG -> AN
Liên kết: DŨNG -> BÌNH
Liên kết: DŨNG -> CHÂU
Liên kết: DŨNG -> HUYỀN
Liên kết: HUYỀN -> DŨNG
Liên kết: done

--- PageRank Cuối Cùng ---
AN      : 0.3355
BÌNH    : 0.2101
CHÂU    : 0.2101
DŨNG    : 0.1767
HUYỀN   : 0.0676
```

Hình 9: Kết quả PageRank và đồ thị mạng cho Bài toán 2.4

```
Nhập số lượng đỉnh (node): 4
Nhập tên các đỉnh:
  Đỉnh 1: A
  Đỉnh 2: B
  Đỉnh 3: C
  Đỉnh 4: D

Nhập các liên kết có hướng (ví dụ: An -> Bình). Gõ 'done' để kết thúc.
Liên kết: A -> B
Liên kết: A -> C
Liên kết: B -> A
Liên kết: B -> C
Liên kết: C -> A
Liên kết: D -> A
Liên kết: D -> B
Liên kết: D -> C
Liên kết: done

--- PageRank Cuối Cùng ---
A       : 0.4165
B       : 0.2251
C       : 0.3208
D       : 0.0375
```

Hình 10: Kết quả PageRank và đồ thị mạng cho Bài toán 2.5

2.4. Chuỗi Markov

Định hướng của nhóm: Để hiểu rõ hơn về hành vi của các hệ thống ngẫu nhiên thay đổi theo thời gian và có tính chất "không nhớ" (Markov property), nhóm đã sử dụng Python để mô phỏng chuỗi Markov. Mục tiêu là dễ dàng định nghĩa ma trận chuyển trạng thái, tính toán phân phối xác suất của các trạng thái sau một số bước n bất kỳ, và quan trọng hơn là tìm ra phân phối dừng (steady-state distribution), thể hiện trạng thái cân bằng dài hạn của hệ thống.

Phương pháp và Kết quả: Chương trình Python sử dụng thư viện NumPy cho các phép toán ma trận. Phân phối trạng thái sau n bước được tính bằng cách nhân vector phân phối trạng thái ban đầu với lũy thừa bậc n của ma trận chuyển P^n. Phân phối dừng được tìm bằng cách giải hệ phương trình $\pi P = \pi$ (kết hợp với $\sum \pi_i = 1$), hoặc thông qua việc tìm vector riêng của P^T ứng với giá trị riêng bằng 1, hoặc quan sát sự hội tụ của P^n khi n lớn. Mã nguồn minh họa cho các tính toán này được cung cấp trong. Các mô phỏng cho Bài toán 2.6 (Mô hình thời tiết) và Bài toán 2.7 (Hành vi khách hàng) đã được thực hiện, cho thấy xác suất của các trạng thái thay đổi qua từng bước và hội tụ về phân phối dừng, giúp trực quan hóa các khái niệm quan trọng của chuỗi Markov.

```
--- Mô phỏng Bài toán 2.6 (Mô hình thời tiết) ---
Ma trận chuyển thời tiết P:
[[0.9 0.1]
 [0.5 0.5]]

Dự báo ngắn hạn:
Phân phối ban đầu: [1 0]
Sau bước 1: [0.9 0.1]
Sau bước 2: [0.86 0.14]
Sau bước 3: [0.844 0.156]
Phân phối xác suất sau 3 ngày (Nắng, Mưa): [0.844 0.156]
```

Hình 11: Kết quả mô phỏng Python cho Bài toán 2.6 về mô hình thời tiết

```
--- Mô phỏng Bài toán 2.7 (Hành vi khách hàng) ---
Ma trận chuyển hành vi khách hàng P:
[[0.6 0.3 0.1]
 [0.5 0.  0.5]
 [0.  0.  1. ]]

Dự báo ngắn hạn:
Phân phối ban đầu: [1 0 0]
Sau bước 1: [0.6 0.3 0.1]
Sau bước 2: [0.51 0.18 0.31]
Sau bước 3: [0.396 0.153 0.451]
Phân phối xác suất sau 3 bước (Nhìn, Mua, Rời): [0.396 0.153 0.451]
Xác suất khách hàng rời siêu thị sau 3 bước: 0.4510

Tìm phân phối dừng (hành vi khách hàng):
Phân phối dừng (Nhìn, Mua, Rời): [0. 0. 1.]
```

Hình 12: Kết quả mô phỏng Python cho Bài toán 2.7 về hành vi khách hàng

KẾT LUẬN VÀ KIẾN NGHỊ

1. Kết luận

Bài nghiên cứu đã trình bày các ứng dụng thực tế quan trọng và đa dạng của ma trận, định thức và hệ phương trình tuyến tính trong bốn lĩnh vực cụ thể: (1) bài toán tối ưu hóa; (2) mã hóa văn bản (Hill Cipher); (3) thuật toán PageRank; và (4) mô hình chuỗi Markov. Mỗi ứng dụng đều được minh họa bằng ví dụ cụ thể, có đoạn mã Python minh hoạ, qua đó người đọc có thể nắm rõ bản chất toán học cũng như tính thực tiễn cao của các khái niệm đại số tuyến tính.

Thông qua đề tài này, chúng em nhận thấy rõ tầm quan trọng và sức mạnh của toán học đại cương nói chung và đại số tuyến tính nói riêng trong giải quyết các bài toán thực tế. Việc hiểu sâu và vận dụng các công cụ toán học như ma trận, định thức, hệ phương trình tuyến tính không chỉ giúp sinh viên nâng cao năng lực phân tích, mà còn là nền tảng để tiếp cận các lĩnh vực công nghệ như học máy, trí tuệ nhân tạo, xử lý dữ liệu lớn, an toàn thông tin.

2. Kiến nghị

Đối với nhà trường, khuyến khích tổ chức các buổi seminar, hội thảo hoặc khóa học chuyên đề về ứng dụng đại số tuyến tính trong các lĩnh vực cụ thể như kỹ thuật, an ninh mạng, phân tích dữ liệu, trí tuệ nhân tạo nhằm tăng tính thực tiễn và liên ngành cho sinh viên.

TÀI LIỆU THAM KHẢO

[1]. Brin and L. Page. (1998). The anatomy of a large-scale hypertextual Web search engine. doi:https://doi.org/10.1016/S0169-7552(98)00110-X

[2]. Kolman, B. & Hill, D. (2008). Elementary Linear Algebra with Applications. (9th, Ed., & G. t. dụng, Trans.) Pearson Prentice Hall.

[3]. LibreTexts. (n.d.). Systems of Inequalities and Linear Programming. From Finite Mathematics: Finite Mathematics.

[4]. Nguyễn Thị Lan Hương, Lê Bích Phượng, Nguyễn Văn Ngọc, Đào Xuân Hưng, Lê Thị Hương Giang, Nguyễn Thị Hiền, Hà Hữu Cao Trình. (2021). Giáo trình Đại số tuyến tính. Nhà xuất bản Giao Thông Vận Tải.

[5]. Ross, S. M. (2014). Introduction to Probability Models. (11th, Ed., & c. v. Chương 4 giới thiệu quá trình Markov và các ứng dụng, Trans.) Academic Press.

[6]. Taha, H. A. (2011). Operations Research: An Introduction. (9th, Ed., & b. g. Sách về phương pháp nghiên cứu vận hành, Trans.) Pearson.

[7]. Thompson, R. (2013). Graph Theory and Social Networks – Math Horizons. (B. b. hội, Trans.) MAA.

[8]. Trappe, W., & Washington, L. C. (2006). Introduction to Cryptography with Coding Theory (2nd ed.). (t. b. Tài liệu về mật mã, Trans.) Pearson.

[9]. Vted.vn. (2018). Phép nhân ma trận và các tính chất. From Vted: https://vted.vn/tin-tuc/phep-nhan-ma-tran-va-cac-tinh-chat-4795.html

[10]. Two-Phase Simplex Algorithm. (n.d.). Math 340 – Universidad Nacional Autónoma de México. From: https://www.matem.unam.mx/~omar/math340/2-phase.html

[11]. Wikipedia. (n.d.). PageRank. From Wikipedia tiếng Việt: https://vi.wikipedia.org/wiki/PageRank

PHỤ LỤC

PHỤ LỤC A: MÃ NGUỒN PYTHON MINH HỌA

Trong phụ lục này, nhóm trình bày chi tiết mã nguồn Python đã được sử dụng để giải và mô phỏng các bài toán được thảo luận trong Chương 3. Mã nguồn được tổ chức theo từng bài toán ứng dụng cụ thể.

A.1. Mã nguồn bài toán Tối ưu hóa

Mã nguồn dưới đây triển khai một lớp LinearProgramming để giải quyết các bài toán quy hoạch tuyến tính (cực đại hóa hoặc cực tiểu hóa) sử dụng hàm linprog từ thư viện *SciPy*. Chương trình cho phép người dùng nhập các hệ số của hàm mục tiêu, ma trận hệ số của các ràng buộc, vế phải của các ràng buộc và loại của từng ràng buộc (<=, >=, =)

```
import numpy as np
from sympy import Matrix
from unicodedata import normalize
import random
import base64

class HillCipher:
    def __init__(self, key_matrix, alphabet):
        self.alphabet = alphabet
        self.n = len(key_matrix)
        self.m = len(alphabet)
        self.key_matrix = Matrix(key_matrix)
        self.modulus = self.m

        det = int(self.key_matrix.det()) % self.modulus
        if np.gcd(det, self.modulus) != 1:
            raise ValueError("The key matrix is not invertible under modulo alphabet length.")
        self.inv_key = self.key_matrix.inv_mod(self.modulus)

    def _text_to_vector_blocks(self, text):
        nums = [self.alphabet.index(ch) for ch in text]
        while len(nums) % self.n != 0:
            nums.append(random.randint(0, self.m - 1))  # padding random
        blocks = [nums[i:i+self.n] for i in range(0, len(nums), self.n)]
        return blocks

    def _vector_blocks_to_text(self, blocks):
        chars = [self.alphabet[num % self.modulus] for block in blocks for num in block]
        return ''.join(chars)

    def encrypt(self, plaintext):
        original_len = len(plaintext)
        blocks = self._text_to_vector_blocks(plaintext)
        encrypted_blocks = [(self.key_matrix @ Matrix(block)) % self.modulus for block in blocks]
        encrypted_text = self._vector_blocks_to_text(encrypted_blocks)
        full_ciphertext = f"{original_len:03d}" + encrypted_text
        encoded_cipher = base64.b64encode(full_ciphertext.encode('utf-8')).decode('utf-8')
        return encoded_cipher
```

```
    def decrypt(self, ciphertext):
        decoded_cipher = base64.b64decode(ciphertext.encode('utf-8')).decode('utf-8')
        original_len = int(decoded_cipher[:3])
        actual_ciphertext = decoded_cipher[3:]
        blocks = self._text_to_vector_blocks(actual_ciphertext)
        decrypted_blocks = [(self.inv_key @ Matrix(block)) % self.modulus for block in blocks]
        raw_text = self._vector_blocks_to_text(decrypted_blocks)
        return raw_text[:original_len]

def get_dynamic_alphabet(text):
    cleaned = normalize('NFC', text.upper())
    unique_chars = sorted(set(cleaned))

    extra_chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .,!?'
    idx = 0
    while len(unique_chars) < 3:
        char_to_add = extra_chars[idx]
        if char_to_add not in unique_chars:
            unique_chars.append(char_to_add)
        idx += 1

    return ''.join(unique_chars)

def generate_invertible_matrix(modulus, n=2):
    attempt = 0
    while True:
        attempt += 1
        matrix = [[random.randint(0, modulus-1) for _ in range(n)] for _ in range(n)]
        det = int(Matrix(matrix).det()) % modulus
        if np.gcd(det, modulus) == 1:
            return matrix
        if attempt > 1000:
            raise ValueError("Cannot find invertible matrix. Try expanding alphabet or changing n.")

def get_user_input():
    print("\n--- EXTENDED HILL CIPHER ---")
    message = input("Enter message to encrypt (can include Vietnamese, Russian, Chinese, etc.): ").strip().upper()
    alphabet = get_dynamic_alphabet(message)
    print(f"Auto-generated alphabet with {len(alphabet)} characters: {alphabet}")
    return alphabet, message

if __name__ == "__main__":
    try:
        alphabet, message = get_user_input()
        key = generate_invertible_matrix(len(alphabet), n=2)
        print(f"Generated invertible key matrix mod {len(alphabet)}: {key}")

        cipher = HillCipher(key_matrix=key, alphabet=alphabet)

        encrypted = cipher.encrypt(message)
        decrypted = cipher.decrypt(encrypted)

        print("\n--- RESULT ---")
        print("Original message:", message)
        print("Encrypted:", encrypted)
        print("Decrypted:", decrypted)

    except ValueError as ve:
        print("Error:", ve)
    except Exception as e:
        print("An unexpected error occurred:", e)
```

A.2. Mã nguồn Mật mã Hill Mở rộng

Mã nguồn dưới đây hiện thực hóa Mật mã Hill với các cải tiến: sử dụng bảng chữ cái động (hỗ trợ Unicode), tự động sinh ma trận khóa 2x2 khả nghịch theo modulo độ dài bảng chữ cái (sử dụng *SymPy*), thêm padding ngẫu nhiên, mã

hóa bản mã cuối cùng bằng Base64 và đảm bảo giải mã chính xác bằng cách lưu độ dài gốc của bản rõ.

```
import numpy as np
from sympy import Matrix # Sử dụng SymPy cho các phép toán ma trận trên trường hữu hạn
from unicodedata import normalize # Chuẩn hóa ký tự Unicode
import random
import base64

class HillCipher:
    def __init__(self, key_matrix_list, alphabet_str):
        """
        Khởi tạo Mật mã Hill.
        Tham số:
            key_matrix_list (list of lists): Ma trận khóa dạng list.
            alphabet_str (str): Chuỗi bảng chữ cái sử dụng.
        """
        self.alphabet = alphabet_str
        self.n = len(key_matrix_list) # Kích thước khối (bằng kích thước ma trận khóa)
        self.m = len(self.alphabet) # Độ dài bảng chữ cái (modulus)

        # Chuyển ma trận khóa sang đối tượng Matrix của SymPy để dễ dàng tính toán det và inv_mod
        self.key_matrix = Matrix(key_matrix_list)
        self.modulus = self.m

        # Kiểm tra tính khả nghịch của ma trận khóa
        determinant = int(self.key_matrix.det()) % self.modulus
        if np.gcd(determinant, self.modulus) != 1:
            raise ValueError(f"Ma trận khóa không khả nghịch với modulus {self.modulus}. GCD(det, modulus) != 1.")

        # Tính ma trận nghịch đảo theo modulus
        self.inv_key_matrix = self.key_matrix.inv_mod(self.modulus)

    def _text_to_vector_blocks(self, text_input):
        """
        Chuyển văn bản thành các khối vector số, có padding nếu cần.
        """
        # Ánh xạ ký tự sang số dựa trên vị trí trong bảng chữ cái
        numerical_representation = [self.alphabet.index(char) for char in text_input]

        # Thêm padding ngẫu nhiên nếu độ dài văn bản không chia hết cho kích thước khối n
        while len(numerical_representation) % self.n != 0:
            numerical_representation.append(random.randint(0, self.m - 1))

        # Chia thành các khối vector
        blocks = [numerical_representation[i:i+self.n] for i in range(0, len(numerical_representation), self.n)]
        return blocks

    def _vector_blocks_to_text(self, vector_blocks):
        """
        Chuyển các khối vector số trở lại thành văn bản.
        """
        text_chars = []
        for block in vector_blocks:
            for num in block:
                # Đảm bảo num nằm trong khoảng của alphabet sau phép toán modulo
                text_chars.append(self.alphabet[int(num) % self.modulus])
        return ''.join(text_chars)

    def encrypt(self, plaintext):
        """
        Mã hóa bản rõ.
        Lưu độ dài gốc, thực hiện padding, mã hóa, sau đó mã hóa Base64.
        """
        original_len = len(plaintext)
        vector_blocks = self._text_to_vector_blocks(plaintext)

        encrypted_blocks = []
        for block in vector_blocks:
            # Nhân ma trận khóa với khối vector (dưới dạng Matrix của SymPy)
            encrypted_vector = (self.key_matrix * Matrix(block)) % self.modulus
            encrypted_blocks.append([val for val in encrypted_vector]) # Chuyển lại thành list

        encrypted_text_padded = self._vector_blocks_to_text(encrypted_blocks)
```

```
        # Thêm độ dài gốc vào đầu bản mã để giải mã chính xác
        # Định dạng độ dài gốc thành 3 chữ số (ví dụ: 005, 012, 123)
        # Điều này giới hạn độ dài bản rõ tối đa là 999 ký tự nếu dùng 3 chữ số.
        # Có thể tăng số chữ số nếu cần xử lý văn bản dài hơn.
        full_ciphertext_payload = f"{original_len:03d}" + encrypted_text_padded

        # Mã hóa Base64 để tăng tính phức tạp và dễ truyền tải
        encoded_cipher_base64 = base64.b64encode(full_ciphertext_payload.encode('utf-8')).decode('utf-8')
        return encoded_cipher_base64

    def decrypt(self, base64_ciphertext):
        """
        Giải mã bản mã đã được mã hóa Base64.
        Tách độ dài gốc, giải mã Hill, sau đó cắt bỏ padding.
        """
        # Giải mã Base64
        decoded_payload = base64.b64decode(base64_ciphertext.encode('utf-8')).decode('utf-8')

        # Tách độ dài gốc và bản mã thực sự
        original_len = int(decoded_payload[:3])
        actual_ciphertext_padded = decoded_payload[3:]

        vector_blocks_cipher = self._text_to_vector_blocks(actual_ciphertext_padded)

        decrypted_blocks = []
        for block in vector_blocks_cipher:
            # Nhân ma trận nghịch đảo với khối vector
            decrypted_vector = (self.inv_key_matrix * Matrix(block)) % self.modulus
            decrypted_blocks.append([val for val in decrypted_vector])

        raw_decrypted_text_padded = self._vector_blocks_to_text(decrypted_blocks)

        # Cắt bỏ padding để lấy lại bản rõ gốc
        return raw_decrypted_text_padded[:original_len]
def get_dynamic_alphabet_from_text(text_input):
    """
    Tạo bảng chữ cái động từ văn bản đầu vào, chuẩn hóa Unicode và đảm bảo có ít nhất 3 ký tự.
    """
    # Chuẩn hóa Unicode (NFC form) và chuyển sang chữ hoa
    cleaned_text = normalize('NFC', text_input.upper())
    unique_chars_set = sorted(list(set(cleaned_text))) # Lấy các ký tự duy nhất và sắp xếp

    # Đảm bảo bảng chữ cái có ít nhất 3 ký tự để tránh lỗi khi sinh ma trận khóa
    # (ví dụ: ma trận 2x2 cần modulus >= 2, nhưng một số phép toán det có thể = 0 nếu modulus quá nhỏ)
    fallback_chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .,!?'
    char_idx_fallback = 0
    while len(unique_chars_set) < 3 and char_idx_fallback < len(fallback_chars):
        char_to_add = fallback_chars[char_idx_fallback]
        if char_to_add not in unique_chars_set:
            unique_chars_set.append(char_to_add)
        char_idx_fallback += 1
        unique_chars_set = sorted(list(set(unique_chars_set))) # Sắp xếp lại sau khi thêm

    return "".join(unique_chars_set)

def generate_invertible_key_matrix(modulus, size=2):
    """
    Sinh ngẫu nhiên một ma trận khóa khả nghịch kích thước 'size x size' theo modulus.
    """
    attempts = 0
    max_attempts = 1000 # Giới hạn số lần thử để tránh vòng lặp vô hạn
    while attempts < max_attempts:
        attempts += 1
        # Tạo ma trận ngẫu nhiên với các phần tử từ 0 đến modulus-1
        matrix_list = [[random.randint(0, modulus - 1) for _ in range(size)] for _ in range(size)]
        sympy_matrix = Matrix(matrix_list)

        determinant = int(sympy_matrix.det()) % modulus

        # Kiểm tra tính khả nghịch: GCD(det, modulus) phải bằng 1
        if np.gcd(determinant, modulus) == 1:
            return matrix_list # Trả về dạng list of lists
```

```
    raise ValueError(f"Không thể tìm thấy ma trận khóa khả nghịch sau {max_attempts} lần thử. "
                     "Hãy thử mở rộng bảng chữ cái hoặc thay đổi kích thước ma trận (n).")

def get_user_input():
    print("\n--- MẬT MÃ HILL MỞ RỘNG ---")
    message_to_encrypt = input("Nhập thông điệp cần mã hóa (hỗ trợ Unicode): ").strip()

    if not message_to_encrypt:
        print("Thông điệp không được để trống.")
        return
    dynamic_alphabet = get_dynamic_alphabet_from_text(message_to_encrypt)
    print(f"Bảng chữ cái tự động tạo ({len(dynamic_alphabet)} ký tự): {dynamic_alphabet}")
    return message_to_encrypt, dynamic_alphabet

if __name__ == "__main__":
    print("\n--- MẬT MÃ HILL MỞ RỘNG ---")
    message_to_encrypt, dynamic_alphabet = get_user_input()
    try:
        # Sử dụng ma trận khóa 2x2 cho ví dụ này
        key_matrix_values = generate_invertible_key_matrix(len(dynamic_alphabet), size=2)
        print(f"Ma trận khóa khả nghịch ngẫu nhiên (mod {len(dynamic_alphabet)}): {key_matrix_values}")

        cipher_instance = HillCipher(key_matrix_list=key_matrix_values, alphabet_str=dynamic_alphabet)

        encrypted_message = cipher_instance.encrypt(message_to_encrypt.upper()) # Mã hóa bản chữ hoa
        decrypted_message = cipher_instance.decrypt(encrypted_message)

        print("\n--- KẾT QUẢ ---")
        print("Thông điệp gốc:", message_to_encrypt.upper())
        print("Bản mã hóa (Base64):", encrypted_message)
        print("Bản giải mã:", decrypted_message)

    except ValueError as ve:
        print("Lỗi:", ve)
    except Exception as e:
        print("Đã xảy ra lỗi không mong muốn:", e)
```

A.3. Mã nguồn tính toán PageRank

Mã nguồn này cho phép người dùng nhập một mạng lưới các nút và liên kết có hướng, sau đó tính toán PageRank cho từng nút bằng phương pháp lặp lũy thừa. Chương trình cũng sử dụng thư viện *networkx* và *matplotlib* để vẽ đồ thị mạng lưới và hiển thị PageRank.

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

def calculate_pagerank(adjacency_matrix, damping_factor=0.85, max_iterations=100, tolerance=1e-6):
    """
    Tính toán PageRank cho một đồ thị dựa trên ma trận kề của nó.
    Tham số:
        adjacency_matrix (np.array): Ma trận kề (A_ij = 1 nếu có link từ j đến i).
        damping_factor (float): Hệ số tắt dần (thường là 0.85).
        max_iterations (int): Số lần lặp tối đa.
        tolerance (float): Ngưỡng hội tụ.
    Trả về:
        np.array: Vector PageRank của các nút.
    """
    num_nodes = adjacency_matrix.shape[0]

    # Chuyển ma trận kề sang kiểu float để tính toán
    adj_matrix_float = adjacency_matrix.astype(float)

    # Tính tổng số liên kết ra từ mỗi nút (out-degree)
    # Trong ma trận kề A_ij (link từ j đến i), out-degree của nút j là tổng cột j
    out_degrees = adj_matrix_float.sum(axis=0) # Tổng theo cột

    # Xử lý các nút không có liên kết ra (dangling nodes)
    # Nếu một nút không có liên kết ra, coi như nó liên kết đến tất cả các nút khác với xác suất bằng nhau.
    for j_col in range(num_nodes):
        if out_degrees[j_col] == 0:
            adj_matrix_float[:, j_col] = 1.0 / num_nodes # Tất cả các phần tử trong cột j được gán 1/n
            out_degrees[j_col] = 1.0 # Cập nhật out_degree để tránh chia cho 0 ở bước sau
                                     # (thực tế là n * (1/n) = 1 nếu có n hàng)

    # Xây dựng ma trận chuyển xác suất M (hay P trong một số tài liệu)
    # M_ij = xác suất chuyển từ nút j sang nút i
    # M_ij = A_ij / out_degree(j)
    M_transition_matrix = adj_matrix_float / out_degrees # Chia từng cột cho out-degree tương ứng

    # Khởi tạo vector PageRank ban đầu (phân phối đều)
    pagerank_vector_r = np.full(num_nodes, 1.0 / num_nodes)
```

```
    # Vector dịch chuyển ngẫu nhiên (teleportation vector)
    teleport_vector = np.full(num_nodes, 1.0 / num_nodes)

    # Thực hiện phương pháp lặp lũy thừa (Power Iteration)
    for iteration_count in range(max_iterations):
        previous_pagerank_vector_r = pagerank_vector_r.copy()

        # Công thức PageRank: r_next = (1-d)*teleport + d * (M @ r)
        # M @ r là tích ma trận M với vector r (M nhân với r)
        pagerank_vector_r = (1 - damping_factor) * teleport_vector + \
                            damping_factor * (M_transition_matrix @ previous_pagerank_vector_r)

        # Kiểm tra điều kiện hội tụ: nếu chuẩn L1 của hiệu giữa r_next và r nhỏ hơn ngưỡng
        if np.linalg.norm(pagerank_vector_r - previous_pagerank_vector_r, 1) < tolerance:
            # print(f"Hội tụ sau {iteration_count + 1} lần lặp.")
            break

    return pagerank_vector_r

def get_user_input():
    """
    Hàm nhận thông tin về đồ thị từ người dùng.
    """
    num_nodes_input = int(input("Nhập số lượng nút (ví dụ: 4 cho A, B, C, D): "))
    node_names_list = []
    print("Nhập tên các nút (ví dụ: A, B, C, D):")
    for i in range(num_nodes_input):
        node_names_list.append(input(f"  Tên nút {i+1}: ").strip())

    # Khởi tạo ma trận kề với các giá trị 0
    adj_matrix_input = np.zeros((num_nodes_input, num_nodes_input), dtype=int)

    print("\nNhập các liên kết có hướng (ví dụ: A -> B). Nhập 'done' để kết thúc.")
    while True:
        link_input_line = input("Liên kết: ").strip()
        if link_input_line.lower() == 'done':
            break
        try:
            source_node_str, destination_node_str = map(str.strip, link_input_line.split("->"))

            # Lấy chỉ số của nút nguồn và nút đích từ danh sách tên nút
            source_idx = node_names_list.index(source_node_str)
            destination_idx = node_names_list.index(destination_node_str)

            # Đặt A_ij = 1 nếu có link từ j (source) đến i (destination)
            # adj_matrix[row_idx_destination, col_idx_source]
            adj_matrix_input[destination_idx, source_idx] = 1
        except ValueError:
            print("Lỗi: Tên nút không hợp lệ hoặc không tìm thấy. Vui lòng nhập lại.")
        except Exception as e:
            print(f"Lỗi nhập liệu: {e}. Vui lòng sử dụng định dạng 'Nguồn -> Đích'.")

    return adj_matrix_input, node_names_list

def draw_graph_with_pagerank(adj_matrix_graph, node_names_graph, pagerank_scores_graph, title="Đồ thị Mạng và PageRank"):
    """
    Vẽ đồ thị mạng lưới và hiển thị PageRank của các nút.
    """
    G = nx.DiGraph() # Tạo đồ thị có hướng
    num_nodes_graph = len(node_names_graph)

    # Thêm các cạnh vào đồ thị từ ma trận kề
    # adj_matrix[j,i] == 1 nghĩa là có link từ nút i (nguồn) đến nút j (đích)
    for i_source_idx in range(num_nodes_graph): # Duyệt qua các cột (nút nguồn)
        for j_dest_idx in range(num_nodes_graph): # Duyệt qua các hàng (nút đích)
            if adj_matrix_graph[j_dest_idx, i_source_idx] == 1:
                G.add_edge(node_names_graph[i_source_idx], node_names_graph[j_dest_idx])

    # Thiết lập vị trí các nút cho việc vẽ đồ thị
    pos = nx.spring_layout(G, seed=42, k=1.0 / np.sqrt(G.number_of_nodes()) if G.number_of_nodes() > 0 else 1.0)
    plt.figure(figsize=(10, 8))

    # Vẽ các nút
    nx.draw_networkx_nodes(G, pos, node_size=3000, node_color='skyblue', edgecolors='black', linewidths=1.5)
    # Vẽ các cạnh
    nx.draw_networkx_edges(G, pos, edge_color='gray', arrows=True, arrowsize=25, width=2, connectionstyle="arc3,rad=0.1")
```

```
    # Vẽ tên các nút
    nx.draw_networkx_labels(G, pos, font_size=12, font_weight='bold')

    # Tạo nhãn PageRank để hiển thị dưới mỗi nút
    pagerank_labels_for_nodes = {}
    for idx, node_name_val in enumerate(node_names_graph):
        pagerank_labels_for_nodes[node_name_val] = f'PR={pagerank_scores_graph[idx]:.3f}'

    # Điều chỉnh vị trí của nhãn PageRank để không chồng lên tên nút
    offset_positions_for_pr = {node_key: (x_val, y_val - 0.12) for node_key, (x_val, y_val) in pos.items()}
    nx.draw_networkx_labels(G, offset_positions_for_pr, labels=pagerank_labels_for_nodes, font_size=10, font_color='darkgreen')

    plt.title(title, fontsize=16)
    plt.axis('off') # Tắt trục tọa độ
    plt.tight_layout() # Tự động điều chỉnh để vừa vặn
    plt.show()

if __name__ == "__main__":
    # Nhập số lượng đỉnh
    n = int(input("Nhập số lượng đỉnh (node): "))
    nodes = []
    print("Nhập tên các đỉnh:")
    for i in range(n):
        nodes.append(input(f"  Đỉnh {i+1}: "))

    # Khởi tạo ma trận kề
    adj_matrix = np.zeros((n, n), dtype=int)

    # Nhập các liên kết có hướng
    print("\nNhập các liên kết có hướng (ví dụ: An -> Bình). Gõ 'done' để kết thúc.")
    while True:
        line = input("Liên kết: ")
        if line.strip().lower() == 'done':
            break
        try:
            src, dst = map(str.strip, line.split("->"))
            i, j = nodes.index(src), nodes.index(dst)
            adj_matrix[j][i] = 1  # Cột là nguồn, dòng là đích
        except:
            print("Đầu vào không hợp lệ. Vui lòng sử dụng định dạng 'Nguồn -> Đích'.")

    # Tính PageRank
    pagerank_vector = calculate_pagerank(adj_matrix)
    print("\n--- PageRank Cuối Cùng ---")
    for name, score in zip(nodes, pagerank_vector):
        print(f"{name:8s}: {score:.4f}")
    # print("Tổng PageRank:", np.sum(pagerank_vector)) # Nên xấp xỉ 1
    draw_graph_with_pagerank(adj_matrix, nodes, pagerank_vector, title="PageRank Mạng Tương Tác Nhóm")
```

A.4. Mã nguồn mô phỏng Chuỗi Markov

Mã nguồn này minh họa cách tính toán phân phối trạng thái sau n bước và tìm phân phối dừng (nếu có) cho một chuỗi Markov với ma trận chuyển trạng thái cho trước.

```python
import numpy as np

def simulate_markov_chain(initial_distribution, transition_matrix, num_steps):
    """
    Tính toán phân phối trạng thái sau một số bước nhất định.
    Tham số:
        initial_distribution (np.array): Vector hàng phân phối xác suất trạng thái ban đầu.
        transition_matrix (np.array): Ma trận chuyển trạng thái P (P_ij là P(X_t+1=j | X_t=i)).
        num_steps (int): Số bước mô phỏng.
    Trả về:
        np.array: Vector hàng phân phối xác suất trạng thái sau num_steps.
    """
    current_distribution = np.array(initial_distribution)
    P = np.array(transition_matrix)

    print(f"Phân phối ban đầu: {current_distribution}")
    for step in range(num_steps):
        current_distribution = current_distribution @ P # Phép nhân vector hàng với ma trận
        print(f"Sau bước {step + 1}: {current_distribution}")

    return current_distribution

def find_stationary_distribution(transition_matrix, max_iter=1000, tol=1e-7):
    """
    Tìm phân phối dừng của chuỗi Markov bằng phương pháp lặp lũy thừa trên ma trận chuyển.
    Hoặc giải hệ phương trình pi * P = pi và sum(pi) = 1.
    Lưu ý: Phương pháp này hội tụ nếu chuỗi Markov là ergodic.
    Tham số:
        transition_matrix (np.array): Ma trận chuyển trạng thái P.
    Trả về:
        np.array: Vector phân phối dừng (vector hàng).
    """
    P = np.array(transition_matrix)
    n_states = P.shape[0]

    # Phương pháp 1: Lặp lũy thừa P^k cho đến khi hội tụ
    # Lấy một hàng bất kỳ của P^k khi k lớn (các hàng sẽ hội tụ về phân phối dừng)
    # P_n = np.linalg.matrix_power(P, max_iter)
    # stationary_dist_power_iteration = P_n[0, :] # Lấy hàng đầu tiên làm ví dụ

    # Phương pháp 2: Giải hệ phương trình (P^T - I) * pi^T = 0 và sum(pi) = 1
    # (P^T - I)pi_transpose = 0  =>  A * pi_transpose = 0
    # Thêm ràng buộc sum(pi_i) = 1
    # Điều này tương đương với việc tìm vector riêng của P^T ứng với giá trị riêng 1.

    evals, evecs = np.linalg.eig(P.T) # Tìm trị riêng, vector riêng của P chuyển vị
    # Vector riêng ứng với trị riêng 1 là phân phối dừng (dưới dạng vector cột)
    stationary_evec = evecs[:, np.isclose(evals, 1)] # Lấy vector riêng có trị riêng gần bằng 1

    if stationary_evec.shape[1] > 0: # Nếu tìm thấy vector riêng
        # Chọn vector riêng đầu tiên (trường hợp có nhiều vector riêng cho trị riêng 1, thường không xảy ra với chuỗi ergodic)
        stationary_dist_column = stationary_evec[:,0].real # Lấy phần thực
        # Chuẩn hóa để tổng các phần tử bằng 1 (để là phân phối xác suất)
        stationary_dist_normalized = stationary_dist_column / np.sum(stationary_dist_column)
        return stationary_dist_normalized # Trả về vector cột, có thể chuyển thành hàng nếu muốn
    else:
        print("Không tìm thấy phân phối dừng bằng phương pháp vector riêng.")
        # Fallback to power iteration if eigenvector method fails or for checking
        # Khởi tạo một phân phối đều
        pi_iter = np.ones(n_states) / n_states
        for _ in range(max_iter):
            pi_next = pi_iter @ P
            if np.allclose(pi_next, pi_iter, atol=tol):
                return pi_next
            pi_iter = pi_next
        print("Phân phối dừng (lặp): Không hội tụ sau số lần lặp tối đa.")
        return None


if __name__ == "__main__":
    # Ví dụ Bài toán 2.6 (Mô hình thời tiết)
    # S1: Nắng, S2: Mưa
    # P = [[0.9, 0.1], [0.5, 0.5]]
    # Nếu hôm nay nắng (trạng thái ban đầu [1, 0]), xác suất nắng sau 3 ngày.
    print("\n--- Mô phỏng Bài toán 2.6 (Mô hình thời tiết) ---")
    initial_dist_weather = [1, 0] # Hôm nay nắng
    transition_matrix_weather = [[0.9, 0.1],
```

```
                              [0.5, 0.5]]
print("Ma trận chuyển thời tiết P:")
print(np.array(transition_matrix_weather))

print("\nDự báo ngắn hạn:")
dist_after_3_days = simulate_markov_chain(initial_dist_weather, transition_matrix_weather, 3)
print(f"Phân phối xác suất sau 3 ngày (Nắng, Mưa): {dist_after_3_days}")

print("\nTìm phân phối dừng (thời tiết):")
stationary_dist_weather_col = find_stationary_distribution(transition_matrix_weather)
if stationary_dist_weather_col is not None:
     # Chuyển vector cột thành vector hàng để hiển thị
    stationary_dist_weather_row = stationary_dist_weather_col.T
    print(f"Phân phối dừng (Nắng, Mưa): {stationary_dist_weather_row}")


# Ví dụ Bài toán 2.7 (Hành vi khách hàng)
# S1: Nhìn quanh, S2: Mua hàng, S3: Rời khỏi
# P = [[0.6, 0.3, 0.1], [0.5, 0, 0.5], [0, 0, 1]] (Lưu ý: S3 là trạng thái hấp thụ)
# Nếu khách hàng bắt đầu từ S1 ([1,0,0]), xác suất rời siêu thị sau 3 bước.
print("\n--- Mô phỏng Bài toán 2.7 (Hành vi khách hàng) ---")
initial_dist_customer = [1, 0, 0] # Bắt đầu ở trạng thái Nhìn quanh
transition_matrix_customer = [[0.6, 0.3, 0.1],
                              [0.5, 0.0, 0.5], # Sửa lại P[1,1] = 0 theo ví dụ
                              [0.0, 0.0, 1.0]] # S3 (Rời đi) là trạng thái hấp thụ
print("Ma trận chuyển hành vi khách hàng P:")
print(np.array(transition_matrix_customer))

print("\nDự báo ngắn hạn:")
dist_after_3_steps_customer = simulate_markov_chain(initial_dist_customer, transition_matrix_customer, 3)
print(f"Phân phối xác suất sau 3 bước (Nhìn, Mua, Rời): {dist_after_3_steps_customer}")
print(f"Xác suất khách hàng rời siêu thị sau 3 bước: {dist_after_3_steps_customer[2]:.4f}")

print("\nTìm phân phối dừng (hành vi khách hàng):")
# Với trạng thái hấp thụ, phân phối dừng sẽ là tất cả xác suất dồn vào trạng thái hấp thụ.
# Phương pháp vector riêng có thể cần điều chỉnh cho trường hợp có nhiều trạng thái hấp thụ
# hoặc khi không phải tất cả các trạng thái đều giao tiếp với nhau (reducible chain).
# Đối với chuỗi có trạng thái hấp thụ, về lâu dài, hệ thống sẽ bị "hút" vào trạng thái hấp thụ.
# Ví dụ, nếu S3 là trạng thái hấp thụ duy nhất và có thể đến được từ mọi trạng thái khác,
# thì phân phối dừng sẽ là [0, 0, 1].
# Hàm find_stationary_distribution hiện tại có thể cho kết quả [0,0,1] hoặc cần kiểm tra kỹ hơn.
stationary_dist_customer = find_stationary_distribution(transition_matrix_customer)
if stationary_dist_customer is not None:
    print(f"Phân phối dừng (Nhìn, Mua, Rời): {stationary_dist_customer.T}")
```