TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

=======*=======**

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN THỰC TẬP CƠ SỞ NGÀNH

ĐỀ TÀI:

NGHIÊN CỨU CƠ SỞ LÝ THUYẾT, ỨNG DỤNG VÀ CÀI ĐẶT ÍT NHẤT 2 THUẬT TOÁN ĐỂ GIẢI BÀI TOÁN PHÂN CÔNG CÔNG VIỆC (JOB ASSIGNMENT PROBLEM)

GVHD:	**TS. Nguyễn Thị Mỹ Bình**	
Nhóm:	**11**	
Lớp:	**20241IT6040002**	
Thành viên:	**Vũ Văn Dũng**	**2022605767**
	Lê Mạnh Duy	**2022606639**
	Nguyễn Duy Hưng	**2022605277**
	Nguyễn Trung Kiên	**2022602731**
	Lê Quang Thắng	**2022602065**

Hà nội, 12/2024

MỤC LỤC

DANH SÁCH HÌNH ẢNH

LỜI MỞ ĐẦU

Bài toán Phân công công việc (Job Assignment Problem) là một trong những bài toán quan trọng trong lĩnh vực tối ưu hóa, với nhiều ứng dụng đồng thời trong khoa học máy tính, vận hành sản xuất, và khoa học dữ liệu. Trong bài toán này, các tài nguyên (như nhân lực, thiết bị) được gán cho các công việc để tối đa hóa một hàm mục tiêu, thường là giảm chi phí hoặc tăng hiệu suất.

Bài toán này không chỉ thu hút sự quan tâm từ góc độ lý thuyết do tính phức tạp của nó (thuộc lớp NP-hard), mà còn có nhiều ứng dụng thực tế quan trọng như: tối ưu hóa trong phân bố lao động, và quản lý dự án.

Bố cục bài báo cáo gồm 3 chương:

Chương 1: Cơ sở lý thuyết

Chương này cung cấp cái nhìn tổng quan về các kiến thức liên quan đến ma trận chi phí, bài toán phân công công việc, và các thuật toán để giải quyết bài toán này như: Hungary, Quy hoạch động.

Chương 2: Thiết kế thuật toán cho bài toán phân công công việc

Chương này đi sâu vào phân tích và thiết kế các thuật toán Hungary và Quy hoạch động cho bài toán phân công công việc.

Chương 3: Cài đặt thuật toán và kiểm thử

Chương này trình bày chi tiết quy trình chuẩn bị dữ liệu, môi trường và thư viện cài đặt. Ngoài ra, kết quả kiểm thử hai thuật toán Hungary và Quy hoạch động được trình bày, từ đó đưa ra đánh giá tổng quát về hai thuật toán.

Chúng em xin chân thành cảm ơn sự hướng dẫn tận tình của cô, cùng những đóng góp ý kiến quý báu của các bạn đồng nghiệp. Mặc dù đã rất cố gắng, nhưng báo cáo không tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự góp ý để hoàn thiện hơn.

Chúng em xin trân trọng cảm ơn!

GIỚI THIỆU

1. Lý do chọn đề tài

Trong bất kỳ tổ chức hay doanh nghiệp nào, việc phân công công việc sao cho hiệu quả luôn là một bài toán quan trọng. Làm thế nào để phân bổ các công việc cho từng cá nhân trong nhóm nhằm tối ưu hóa nguồn lực, giảm thiểu chi phí và thời gian, đồng thời tăng cường hiệu suất làm việc là một câu hỏi mà nhiều nhà quản lý cần giải quyết. Bài toán Phân công công việc (Job Assignment Problem) chính là một trong những bài toán nổi bật trong lĩnh vực tối ưu hóa, được nghiên cứu nhiều trong toán học và khoa học máy tính.

Bài toán này không chỉ xuất hiện trong các doanh nghiệp mà còn được ứng dụng rộng rãi trong các lĩnh vực như: phân công máy móc sản xuất, phân công nhân lực trong dự án, hoặc thậm chí trong lập lịch thi đấu thể thao. Mục tiêu chính của bài toán là tìm ra cách phân công các công việc cho các nguồn lực (nhân viên, máy móc) sao cho tổng chi phí là nhỏ nhất hoặc hiệu suất đạt được là cao nhất.

Bài toán Phân công công việc là việc phân chia nhiệm vụ giữa các người làm việc sao cho tổng chi phí (hoặc thời gian) là nhỏ nhất.Trong bài toán này, có một tập hợp công việc và một tập hợp người làm việc. Mỗi công việc phải được giao cho một người làm việc, và mỗi người làm việc chỉ thực hiện một công việc.

2. Mục tiêu nghiên cứu

Mục tiêu chính của nghiên cứu này là giải quyết bài toán Phân công công việc bằng cách áp dụng các phương pháp tối ưu hóa khác nhau. Cụ thể, nghiên cứu này hướng tới:

- Xây dựng mô hình toán học cho bài toán Phân công công việc.
- Tìm hiểu và áp dụng các thuật toán tối ưu như Quy hoạch động (Dynamic Programming), Thuật toán Hungary, Thuật toán nhánh cận (Branch and Bound), hoặc các phương pháp heuristic, meta-heuristic (như Thuật toán di truyền, Thuật toán Simulated Annealing).
- So sánh, đánh giá hiệu quả của các phương pháp về thời gian thực hiện và độ chính xác của kết quả.
- Ứng dụng thực tế của bài toán trong một số lĩnh vực cụ thể như phân bổ nguồn lực, lập kế hoạch dự án, quản lý công việc nhóm.

3. Phạm vi nghiên cứu

- Các mô hình toán học của bài toán phân công công việc.
- Các thuật toán giải quyết bài toán, bao gồm cả phương pháp cổ điển và hiện đại như phương pháp quy hoạch động, thuật toán nhánh cận, thuật toán di truyền, thuật toán tìm kiếm địa phương.
- Ứng dụng của bài toán trong thực tế, đặc biệt trong lĩnh vực sản xuất và quản lý dự án.

4. Phương pháp nghiên cứu

Nghiên cứu sẽ áp dụng các phương pháp lý thuyết và thực nghiệm. Đầu tiên, nghiên cứu các tài liệu liên quan đến bài toán Phân công công việc và các phương pháp giải quyết hiện có. Sau đó, tiến hành mô phỏng và thử nghiệm các phương pháp tối ưu hóa được lựa chọn trên các bộ dữ liệu thực tế. So sánh kết quả thu được về hiệu quả, thời gian xử lý và khả năng mở rộng của các phương pháp. Nghiên cứu cũng sẽ đánh giá khả năng ứng dụng thực tiễn của các phương pháp này trong môi trường doanh nghiệp.

CHƯƠNG 1 : CƠ SỞ LÝ THUYẾT

1.1. Cơ sở lý thuyết bài toán Phân công công việc

1.1.1. Định nghĩa

Bài toán phân công công việc (Job Assignment Problem) là một bài toán tối ưu hóa trong lĩnh vực nghiên cứu hoạt động và quản lý. Mục tiêu của bài toán này là phân công một tập hợp các công việc cho một tập hợp các nhân viên sao cho chi phí hoàn thành công việc là nhỏ nhất, hoặc hiệu quả công việc là cao nhất.

Bài toán này thường được mô hình hóa dưới dạng một ma trận chi phí, trong đó mỗi ô của ma trận biểu thị thời gian mà một nhân viên cần để hoàn thành một công việc cụ thể. Nhiệm vụ là tìm ra một cách phân công sao cho tổng thời gian là tối ưu.

1.1.2. Một số khái niệm cơ bản

1.1.2.1. Ma trận chi phí

- Định nghĩa: Ma trận chi phí là một bảng được sử dụng trong các bài toán tối ưu hóa, biểu diễn các chi phí liên quan đến việc phân bổ tài nguyên cho các tác vụ hoặc công việc. Mỗi ô trong ma trận biểu thị chi phí phân bổ một tài nguyên cụ thể cho một tác vụ cụ thể, tạo điều kiện thuận lợi cho việc phân tích và giải quyết các thách thức phân bổ. Công cụ này rất cần thiết để tìm ra cách phân bổ tối ưu các tác vụ cho các tài nguyên trong khi giảm thiểu tổng chi phí.
- Bài toán gán có thể phát biểu dưới dạng n x n ma trận chi phí C phần tử thực như bảng sau:

		Jobs				
		1	*2*	*3*	*j*	*n*
Persons	1	C_{11}	C_{12}	C_{13}	C_{ij}	C_{1n}
	2	C_{21}	C_{22}	C_{23}	C_{2j}	C_{2n}
	3	C_{31}	C_{32}	C_{33}	C_{3j}	C_{3n}
	i	C_{11}	C_{12}	C_{13}	C_{ij}	C_{in}
	n	C_{n1}	C_{n2}	C_{n3}	C_{nj}	C_{nn}

Hình 1. Ma trận chi phí

  + n người và n công việc.
  + Một ma trận chi phí C=[cij], trong đó [cij] đại diện cho chi phí khi phân công i cho công việc j.
- Công thức

Tối thiểu hóa của tổng chi phí

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}.$$

Trong đó:

$$x_{ij} = \begin{cases} 1, & \text{nếu người } i \text{ được phân công cho công việc } j \\ 0, & \text{ngược lại.} \end{cases}$$

Điều kiện:

+ Mỗi người được phân công cho đúng một công việc

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, 2, \ldots, n.$$

+ Mỗi công việc chỉ được phân công cho đúng một người

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, 2, \ldots, n.$$

+ Biến quyết định nhị phân

$$x_{ij} \in \{0, 1\} \quad \forall i, j.$$

1.1.2.2. Giải thuật tối ưu

- Giải thuật tối ưu là thuật toán được thiết kế để tìm ra giải pháp tốt nhất cho một bài toán tối ưu hóa, tức là bài toán có mục tiêu tối đa hóa hoặc tối thiểu hóa một hàm mục tiêu nào đó, trong khi thỏa mãn các điều kiện ràng buộc nhất định.
- Trong bài toán tối ưu, nhiệm vụ của giải thuật tối ưu là tìm ra giải pháp tối ưu trong tập hợp tất cả các giải pháp khả thi.
- Các thành phần chính của giải thuật tối ưu:

+ Hàm mục tiêu (Objective Function): là đại lượng mà giải thuật cần tối ưu hóa (có thể là tối thiểu hóa hoặc tối đa hóa). Ví dụ: tối thiểu hóa chi phí, tối đa hóa lợi nhuận.
+ Biến quyết định (Decision Variables): là các biến mà giải thuật có thể điều chỉnh để tối ưu hóa hàm mục tiêu.
+ Ràng buộc (Constraints): là các điều kiện mà các biến quyết định phải thỏa mãn. Ví dụ, một số điều kiện về giới hạn tài nguyên hoặc thời gian.

- Phân loại giải thuật tối ưu:
  + Giải thuật chính xác (Exact Algorithms): là các giải thuật tìm ra giải pháp tối ưu tuyệt đối cho bài toán. Các giải thuật này đảm bảo sẽ tìm ra giải pháp tốt nhất nếu có đủ thời gian và tài nguyên.

  · Ví dụ:

    - Thuật toán Hungary (cho bài toán phân công công việc).
    - Quy hoạch tuyến tính (Linear Programming).
    - Thuật toán nhánh và cận (Branch and Bound).

  + Giải thuật xấp xỉ (Approximation Algorithms):là các giải thuật có thể không tìm được giải pháp tối ưu tuyệt đối nhưng tìm được giải pháp gần tối ưu trong thời gian ngắn hơn. Những giải pháp này thường chấp nhận được trong các bài toán có kích thước lớn và yêu cầu khắt khe về thời gian.

    Ví dụ:

    - Giải thuật tham lam (Greedy Algorithm).
    - Giải thuật di truyền (Genetic Algorithm).

  + Giải thuật heuristic: là các giải thuật không đảm bảo tìm ra giải pháp tối ưu nhưng có thể tìm được giải pháp "đủ tốt" một cách nhanh chóng. Các thuật toán heuristic thường được sử dụng cho các bài toán lớn và phức tạp, khi giải pháp tối ưu không thể tìm được trong thời gian ngắn.

    Ví dụ: Simulated Annealing, Tabu Search, Ant Colony Optimization.

  - Ví dụ: Giải thuật tối ưu cho bài toán phân công công việc

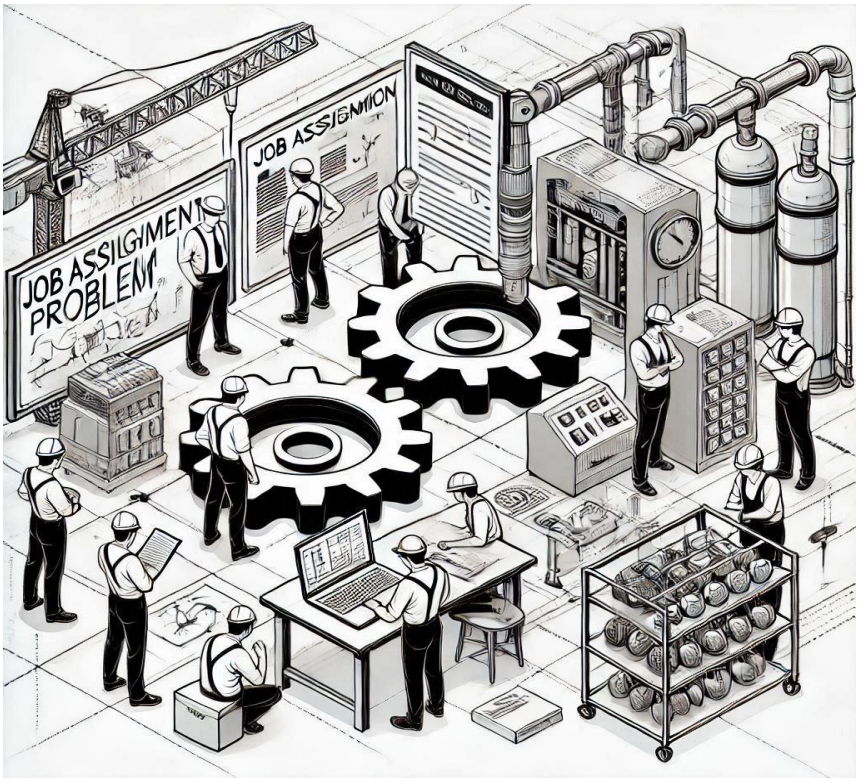
Bài toán phân công công việc có thể sử dụng giải thuật Hungarian để tìm cách phân công sao cho tổng chi phí là nhỏ nhất. Thuật toán Hungary là một giải thuật chính xác, tìm ra lời giải tối ưu bằng cách biến đổi ma trận

chi phí và áp dụng các bước tìm số 0 tối thiểu trong ma trận để phân công người thực hiện công việc.

1.1.3. Ứng dụng của bài toán

Bài toán phân công công việc có nhiều ứng dụng thực tiễn quan trọng trong nhiều lĩnh vực khác nhau. Dưới đây là một số ví dụ tiêu biểu:

- Ứng dụng trong sản xuất: Giúp gán các công việc sản xuất cho máy móc và nhân công một cách tối ưu, từ đó giảm chi phí vận hành, tiết kiệm thời gian sản xuất và tăng hiệu quả sử dụng nguồn lực. Ví dụ, trong dây chuyền sản xuất, bài toán này phân công các nhiệm vụ như lắp ráp, gia công sao cho các công đoạn phối hợp nhịp nhàng, giảm thời gian chờ.



Hình 2. Phân công công việc trong sản xuất

- Ứng dụng trong quản lý dự án: Hỗ trợ quản lý dự án phân công nhiệm vụ cho các thành viên dự án dựa trên kỹ năng và khối lượng công việc, đảm bảo dự án hoàn thành đúng tiến độ và ngân sách. Nó giúp tránh tình trạng quá tải cho một số thành viên, đồng thời tối ưu hóa hiệu quả làm việc của toàn đội ngũ.

Hình 3. Phân công công việc trong quản lý dự án

- Ứng dụng trong tối ưu hóa tài nguyên: Giúp phân bổ hợp lý tài nguyên nhân lực, thiết bị và tài chính. Ví dụ, trong quản lý nhân sự, bài toán này phân công công việc dựa trên năng lực của từng người, tránh lãng phí thời gian và nguồn lực. Trong việc sử dụng máy móc, nó giúp tối đa hóa thời gian hoạt động và bảo trì thiết bị hợp lý.

Hình 4. Phân công công việc trong tối ưu hóa tài nguyên

- Các ứng dụng khác trong thực tiễn: Bài toán này còn được áp dụng trong các lĩnh vực như logistics (tối ưu hóa lộ trình giao hàng, phân công tài xế), y tế (phân công bác sĩ trực ca, xử lý bệnh nhân), giáo dục (gán giảng viên cho các môn học), và công nghệ thông tin (phân công lập trình viên cho các nhiệm vụ phát triển phần mềm). Trong tất cả các lĩnh vực này, nó giúp nâng cao hiệu quả và giảm thiểu lãng phí tài nguyên.

1.1.4. Các biến thể của bài toán

1.1.4.1. Bài toán phân công tổng quát(GAP)

Mỗi công việc có thể được thực hiện bởi một số máy hoặc nhân viên khác nhau với chi phí và thời gian khác nhau. Tuy nhiên, mỗi máy hoặc nhân viên có một nguồn lực giới hạn (ví dụ: thời gian hoặc năng lượng).

Ví dụ:

Một công ty có 3 nhân viên (A, B, C) và 4 nhiệm vụ (1, 2, 3, 4). Mỗi nhân viên có giới hạn thời gian làm việc tối đa là:

- A: 8 giờ
- B: 10 giờ
- C: 7 giờ

Thời gian cần thiết để hoàn thành mỗi nhiệm vụ

	Nhiệm vụ 1	Nhiệm vụ 2	Nhiệm vụ 3	Nhiệm vụ 4
A	3	4	5	2
B	2	6	4	3
C	4	5	3	2

Cách giải:

Giả sử ta thử nghiệm các phương án phân công nhiệm vụ khác nhau và tính tổng thời gian làm việc.

Phương án 1:

- Nhiệm vụ 1: A (3 giờ)
- Nhiệm vụ 2: B (6 giờ)
- Nhiệm vụ 3: C (3 giờ)
- Nhiệm vụ 4: A (2 giờ)

Tổng thời gian:

- A: 3+2=5
- B: 6
- C: 3

Đáp ứng giới hạn:

- A: 5≤ 8
- B: 6≤ 10
- C: 3 ≤ 7

Tổng chi phí thời gian: 3+6+3+2=14

Phương án 2:

- Nhiệm vụ 1: B (2 giờ)
- Nhiệm vụ 2: A (4 giờ)
- Nhiệm vụ 3: C (3 giờ)
- Nhiệm vụ 4: B (3 giờ)

Tổng thời gian:

- A: 4
- B: 2+3=5
- C: 3

Đáp ứng giới hạn:

- A: 4 ≤ 8
- B: 5 ≤ 10
- C: 3 ≤ 7

Tổng chi phí thời gian: 2+4+3+3=12

Kết luận: Phương án 2 tối ưu hơn với tổng chi phí thời gian :12.

1.1.4.2. Bài toán phân công bậc hai

Bài toán phân công bậc hai (QAP) tìm cách tối thiểu hóa tổng chi phí khi phân công một tập hợp các cơ sở (facilities) vào một tập hợp các địa điểm (locations) với chi phí không chỉ phụ thuộc vào từng cơ sở riêng lẻ mà còn phụ thuộc vào sự tương tác giữa các cặp cơ sở.

Ví dụ:

Bài toán: Công ty cần đặt 3 phòng ban A,B,C vào 3 địa điểm 1,2,3.

Chi phí di chuyển giữa các phòng ban (ma trận khoảng cách):

	1	2	3
A	0	10	20
B	10	0	30
C	20	30	0

Mức độ tương tác giữa các phòng ban (ma trận tương tác):

	A	B	C
A	0	5	2
B	5	0	4
C	2	4	0

Cách giải:

Phương án thử nghiệm:

Đặt A tại 1, B tại 2, C tại 3:
Tổng chi phí = $0.0+5\cdot 10+2\cdot 20+5\cdot 10+4\cdot 30+2\cdot 30=300$

Đặt A tại 2, B tại 1, C tại 3:
Tổng chi phí = $10\cdot 5+0\cdot 0+2\cdot 30+5\cdot 10+4\cdot 10+2\cdot 20=240$

Kết quả tối ưu: Đặt A tại 2, B tại 1, C tại 3 với tổng chi phí 240.

1.2. Cơ sở lý thuyết thuật toán của bài toán Phân công công việc

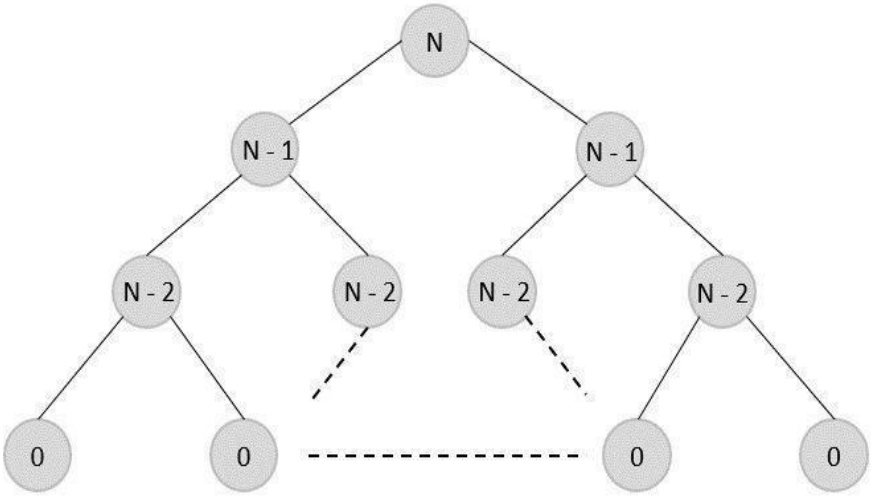
1.2.1. Thuật toán Quy hoạch động

1.2.1.1. Khái niệm và nguyên lý hoạt động

- Khái niệm: Quy hoạch động (dynamic programming) là một phương pháp được sử dụng trong toán học và khoa học máy tính để giải quyết các vấn đề phức tạp bằng cách chia chúng thành các bài toán con đơn giản hơn. Bằng

cách giải mỗi bài toán con chỉ một lần và lưu trữ kết quả, nó tránh được các phép tính dư thừa, dẫn đến giải pháp hiệu quả hơn cho nhiều bài toán.

- Nguyên lý hoạt động:



Hình 5 Nguyên lý hoạt động của quy hoạch động

+ Chia bài toán thành các bài toán con nhỏ hơn: Bài toán ban đầu được chia thành các bài toán con mà mỗi bài toán con là một phần của bài toán lớn hơn. Các bài toán con này thường có tính chất lặp lại.
+ Lưu trữ kết quả của các bài toán con: Để tránh việc tính toán lặp lại các bài toán con nhiều lần, kết quả của chúng được lưu trữ trong một bảng (thường là mảng hoặc ma trận).
+ Sử dụng lại kết quả đã lưu: Khi cần kết quả của một bài toán con nào đó, chương trình sẽ kiểm tra bảng lưu trữ để xem kết quả đã được tính toán trước đó chưa. Nếu đã có, nó sẽ sử dụng lại kết quả đó thay vì tính toán lại từ đầu.

1.2.1.2. Các bước của thuật toán

- Bước 1: Xác định cấu trúc của bài toán con tối ưu: Hiểu rõ cách mà bài toán lớn được chia thành các bài toán con và cách mà kết quả của các bài toán con này kết hợp lại để giải quyết bài toán lớn
- Bước 2: Định nghĩa hàm hồi quy: Xác định công thức hoặc hàm hồi quy để giải quyết bài toán dựa trên các bài toán con. Đây là bước quan trọng để hiểu cách các bài toán con liên kết với nhau.
- Bước 3: Tính giá trị của bài toán con theo thứ tự từ nhỏ đến lớn: Bắt đầu từ những bài toán con nhỏ nhất và sử dụng công thức hồi quy để tính giá trị của các bài toán con lớn hơn dựa trên các bài toán con nhỏ hơn đã được tính toán trước đó.

- Bước 4: Lưu trữ kết quả của các bài toán con: Kết quả của mỗi bài toán con được lưu trữ trong bảng để sử dụng lại sau này.
- Bước 5: Giải quyết bài toán lớn nhất: Sau khi đã tính toán và lưu trữ kết quả của tất cả các bài toán con, sử dụng chúng để giải quyết bài toán ban đầu.

1.2.1.3. Độ phức tạp của thuật toán

Quy hoạch động (dynamic programming - DP) là một kỹ thuật lập trình được sử dụng để giải quyết các bài toán tối ưu bằng cách chia nhỏ vấn đề thành các bài toán con và lưu trữ kết quả của các bài toán con đã giải quyết để tránh tính toán lại nhiều lần. Độ phức tạp của thuật toán quy hoạch động phụ thuộc vào cách tiếp cận và cấu trúc của bài toán cụ thể.

- Độ Phức Tạp Thời Gian: Độ phức tạp thời gian của thuật toán quy hoạch động thường được xác định bởi hai yếu tố chính:
    + Số lượng bài toán con: Đây là số lượng trạng thái (hoặc quyết định) mà thuật toán cần giải quyết. Ví dụ, trong bài toán Fibonacci, có hai bài toán con cho mỗi trạng thái n, dẫn đến độ phức tạp O(n).
    + Chi phí để giải quyết mỗi bài toán con: Đối với mỗi trạng thái, thời gian cần thiết để tính toán giá trị có thể thay đổi. Nếu mỗi bài toán con có chi phí O(1), tổng độ phức tạp thời gian sẽ là O(n). Ngược lại, nếu chi phí lớn hơn, độ phức tạp sẽ tăng tương ứng.

- Độ Phức Tạp Không Gian: Độ phức tạp không gian thường liên quan đến việc lưu trữ các giá trị của các bài toán con. Quy hoạch động có thể sử dụng các bảng (table) để lưu trữ kết quả và giảm số lượng phép toán cần thiết trong tương lai.
    + Bảng 2 chiều: Nhiều bài toán sử dụng bảng 2 chiều để lưu trữ giá trị cho từng trạng thái, dẫn đến độ phức tạp không gian là O(n^2).
    + Tối ưu hóa không gian: Trong một số trường hợp, có thể tối ưu hóa không gian bằng cách chỉ lưu trữ các giá trị cần thiết cho bước tiếp theo, giảm độ phức tạp không gian xuống O(n) hoặc thậm chí O(1).

1.2.1.4. Ưu và nhược điểm

- Ưu điểm
    + Giảm thời gian tính toán: Quy hoạch động lưu trữ kết quả của các bài toán con đã được tính toán trước đó, giúp giảm thiểu đáng kể số lượng phép tính cần thực hiện. Điều này đặc biệt hiệu quả trong các bài toán có tính chất lặp lại, như dãy Fibonacci hoặc bài toán tối ưu hóa.

+ Giải quyết được các bài toán phức tạp: DP cho phép giải quyết nhiều bài toán tối ưu hóa và lập lịch phức tạp mà các phương pháp khác không thể làm được hiệu quả, chẳng hạn như bài toán balo, chuỗi con chung dài nhất (LCS), và nhiều bài toán đồ thị.
+ Tránh được các tính toán lặp lại không cần thiết: Bằng cách lưu trữ kết quả của các bài toán con, DP tránh được việc tính toán lặp lại những phần đã giải quyết, giúp tiết kiệm tài nguyên và tăng hiệu suất.
+ Độ chính xác cao: Kỹ thuật DP đảm bảo tìm ra giải pháp tối ưu cho bài toán bằng cách xây dựng các giải pháp từ các bài toán con tối ưu.

- Nhược điểm
  + Tốn bộ nhớ: DP yêu cầu lưu trữ kết quả của tất cả các bài toán con, điều này có thể tốn rất nhiều bộ nhớ, đặc biệt đối với các bài toán có không gian trạng thái lớn. Điều này có thể gây ra vấn đề khi áp dụng DP cho các bài toán có quy mô lớn.
  + Phức tạp trong việc triển khai: Hiểu và triển khai một giải pháp DP đòi hỏi người lập trình phải hiểu rõ cấu trúc của bài toán và cách chia nhỏ nó thành các bài toán con. Điều này có thể phức tạp và tốn nhiều thời gian để thiết kế và debug.
  + Không phải lúc nào cũng khả thi: Không phải bài toán nào cũng có thể áp dụng được DP. Các bài toán cần có tính chất con tối ưu (optimal substructure) và tính chất lặp lại của bài toán con (overlapping subproblems). Nếu bài toán không có các tính chất này, DP sẽ không hiệu quả.
  + iêu tốn thời gian cho việc lưu trữ và truy xuất: Mặc dù giảm thời gian tính toán, việc lưu trữ và truy xuất kết quả từ bộ nhớ cũng tiêu tốn thời gian, đặc biệt khi kích thước của bảng lưu trữ lớn.

1.2.2. Thuật toán Hungary

1.2.1.1. Khái niệm

Thuật toán Hungarian, còn gọi là thuật toán Kuhn-Munkres, là một phương pháp để giải quyết bài toán phân công công việc với mục tiêu tìm ra cách phân công tối ưu sao cho tổng chi phí là nhỏ nhất.

1.2.1.2. Các bước của thuật toán

- Bước 1: Lập bảng phân việc và máy theo dữ liệu thực tế;
- Bước 2: Tìm số nhỏ nhất trong từng hàng của bảng phân việc và trừ các số trong hàng cho số đó;
- Bước 3: Tìm số nhỏ nhất trong từng cột và trừ các số trong từng cột cho số đó;

- Bước 4: Tìm cách kẻ các đường thẳng đi qua hàng hoặc cột có các số 0 sao cho số đường thẳng kẻ được ít nhất (tức là hàng, cột nào có 1 số 0 thì kẻ đường thẳng, hàng thì kẻ cột, cột thì kẻ hàng, số 0 nào bị gạch rồi thì không tính).

  Thực hiện theo cách sau:

  + Bắt đầu từ những hàng có 1 số 0, khoanh tròn số đó lại và kẻ một đường thẳng xuyên suốt cột;
  + Tìm các cột có 1 số 0, khoanh tròn số đó lại rồi kẻ một đường xuyên suốt hàng.
- Bước 5: Lặp lại bước 4 cho đến khi không còn có thể khoanh được nữa. Nếu số đường thẳng kẻ được ít nhất bằng số hàng (số cột) thì bài toán đã có lời giải tối ưu. Nếu số đường kẻ được nhỏ hơn số hàng (số cột) thì cần làm tiếp: Tìm số chưa bị gạch nhỏ nhất và lấy tất cả các số chưa bị gạch trừ đi số đó; các số bị gạch bởi 2 đường thẳng cộng với số đó; còn các số khác giữ nguyên.
- Bước 6: Quay trở lại bước 4 và 5 cho đến khi tìm được lời giải tối ưu.

  Sau đó, ma trận cuối cùng khoanh ở cột nào thì chọn người tương ứng làm việc đó

1.2.1.3. Độ phức tạp thuật toán

- Độ phức tạp thời gian: O(n^3)
- Không gian phụ trợ: O(n^2)

  Với n là số cột hoặc số hàng trận của ma trận chi phí

1.2.1.4. Ưu và nhược điểm

- Ưu điểm
  + Tối ưu và chính xác:Thuật toán Hungarian đảm bảo tìm ra ghép cặp có chi phí tối thiểu hoặc lợi ích tối đa, mang lại lời giải tối ưu cho bài toán phân công trong nhiều trường hợp.
  + Tính toán đa năng: Thuật toán có thể áp dụng cho cả hai bài toán cực tiểu và cực đại, chỉ cần điều chỉnh các tham số. Điều này làm cho nó trở nên linh hoạt trong các bài toán cần tối ưu hóa chi phí hoặc tối đa hóa lợi ích.
  + Độ phức tạp thời gian hợp lý: O(n^3)Với độ phức tạp thời gian , thuật toán Hungary có thể xử lý các bài toán lớn một cách tương đối nhanh so với các phương pháp khác.

+ Thích hợp cho các ứng dụng thực tế: Thuật toán Hungarian đã được ứng dụng rộng rãi trong các lĩnh vực như phân công công việc, lập lịch, ghép cặp, và nhận dạng đối tượng trong thị giác máy tính.

- Nhược điểm
  + Chỉ áp dụng được cho bài toán phân công công việc với kích thước ma trận vuông (số nhân viên bằng số công việc). Tuy nhiên, có thể điều chỉnh bằng cách thêm hàng hoặc cột giả với chi phí bằng 0.
  + Có thể trở nên phức tạp và khó triển khai trong thực tế đối với các bài toán có nhiều điều kiện và ràng buộc phức tạp hơn.

CHƯƠNG 2: THIẾT KẾ THUẬT TOÁN

2.1. Mô hình hóa bài toán

Như cái tên của nó, bài toán phân công công việc này liên quan đến vấn đề phân bổ nhân lực, hỗ trợ người quản lý trong việc điều chỉnh nhân sự của họ với các công việc phù hợp để doanh thu đạt tối đa hay chi phí đạt tối thiểu. Một ví dụ điển hình như sau:

Tại một khách sạn nọ, vào dịp lễ sau covid, khách hàng đặt phòng rất đông, cần phục vụ rất nhiều, vì không đủ nhân lực nên chúng tôi có tuyển thêm nhân sự tạm thời và có 4 bạn tên Anh, Nhi, Xinh, Nhất thỏa điều kiện vào cả 4 vị trí khác nhau trong bộ phận tiền sảnh là:

Reservation (Đặt phòng: Chịu trách nhiệm tiếp nhận thông tin đặt phòng từ khách hàng sau đó kiểm tra, xử lý trên hệ thống rồi xác nhận lại với khách).

Reception (Tiếp tân: Chịu trách nhiệm chào đón, làm thủ tục check in, check out, giải quyết các yêu cầu (bao gồm cả phàn nàn) của khách trong thời gian lưu trú).

Cashier (Thu ngân: Chịu trách nhiệm làm thủ tục thanh toán cho khách hàng, cụ thể kiểm tra, nhập dịch vụ khách đã sử dụng vào hệ thống
sau đó in hóa đơn, thu tiền của khách).

Operator (Tổng đài: Chịu trách nhiệm tiếp nhận, xử lý các cuộc gọi đến khách sạn, xử lý các yêu cầu của khách hàng. Tiếp nhận và thực hiện cuộc gọi báo thức cho khách lưu trú).

Mỗi vị trí, tương ứng với từng người, họ có yêu cầu mức lương khác nhau, số liệu cụ thể ở bảng bên dưới:

Nhân viên/số giờ có thể làm trong tuần	Đặt phòng	Lễ tân	Thu ngân	Tổng đài
Anh	32	48	58	56
Nhi	48	50	33	32
Xinh	40	34	27	25
Nhất	24	56	49	40

Câu hỏi đặt ra là sự phân công công việc nào sẽ giúp cho khách sạn đạt mức chi phí tối thiểu nhất với điều kiện là mỗi người chỉ được thực hiện đúng một công việc ?

2.2. Thiết kế thuật toán

2.2.1. Thuật toán Quy hoạch động

2.2.1.1. Thuật toán Quy hoạch động trong bài toán Phân công công việc

- **Mục tiêu:** Gán n công việc cho n nhân viên sao cho tổng chi phí là nhỏ nhất. Mỗi công việc chỉ được thực hiện bởi một nhân viên và mỗi nhân viên chỉ nhận một công việc
- **Phương pháp:** Sử dụng Quy hoạch động để thử tất cả các cách gán công việc một cách hiệu quả bằng cách:
    + Lưu trữ các trạng thái công việc đã được gán cho nhân viên.
    + Sử dụng cấu trúc mask (nhị phân) để biểu diễn các công việc đã thực hiện.

2.2.1.2. Bài toán và phân tích yêu cầu

- Input:
    + n: Số nhân viên và số công việc.
    + Ma trận chi phí costMatrix: Một ma trận n×n trong đó costMatrix[i][j] biểu diễn chi phí mà người thứ i thực hiện công việc j.
- Output:
    + Tổng chi phí tối thiểu để gán tất cả các công việc cho các nhân viên.
    + Mảng phân công assignment: Cho biết công việc nào được giao cho từng người.
- Ràng buộc:
    + Mỗi nhân viên chỉ được gán một công việc
    + n là số nguyên dương, giới hạn khả thi $n \leq 20$ vì thuật toán có độ phức tạp $O(n \cdot 2n)$
    + Giá trị trong ma trận chi phí costMatrix[i][j] là các số nguyên không âm
    + Mỗi công việc chỉ được thực hiện bởi một nhân viên.

2.2.1.3. Các bước giải thuật

Bước 1: Định nghĩa trạng thái

- Sử dụng một mảng dp[mask] để lưu tổng chi phí tối thiểu của trạng thái mask, trong đó: **mask** là một số nhị phân kích thước nnn, biểu diễn trạng thái các công việc:
    + Bit i bằng 1: Công việc i đã được giao.
    + Bit i bằng 0: Công việc i chưa được giao.

Bước 2: Công thức chuyển trạng thái

- dp[mask]:Tổng chi phí tối thiểu cho trạng thái mask.

- Trường hợp cơ sở:dp[0]=0 (nếu không có công việc nào được giao, tổng chi phí là 0).
- Công thức quy hoạch động:

$$dp[mask] = \min_{job \notin mask} (dp[mask'] + costMatrix[person][job])$$

$mask' = mask \vee (1 << job)$: Trạng thái khi giao công việc job cho người tiếp theo.

$person = \text{count\_set\_bits}(mask)$: Số người đã được giao công việc.

Bước 3: Khởi tạo

Khởi tạo mảng dp với kích thước 2^n, tất cả giá trị là ∞, ngoại trừ dp[0]=0

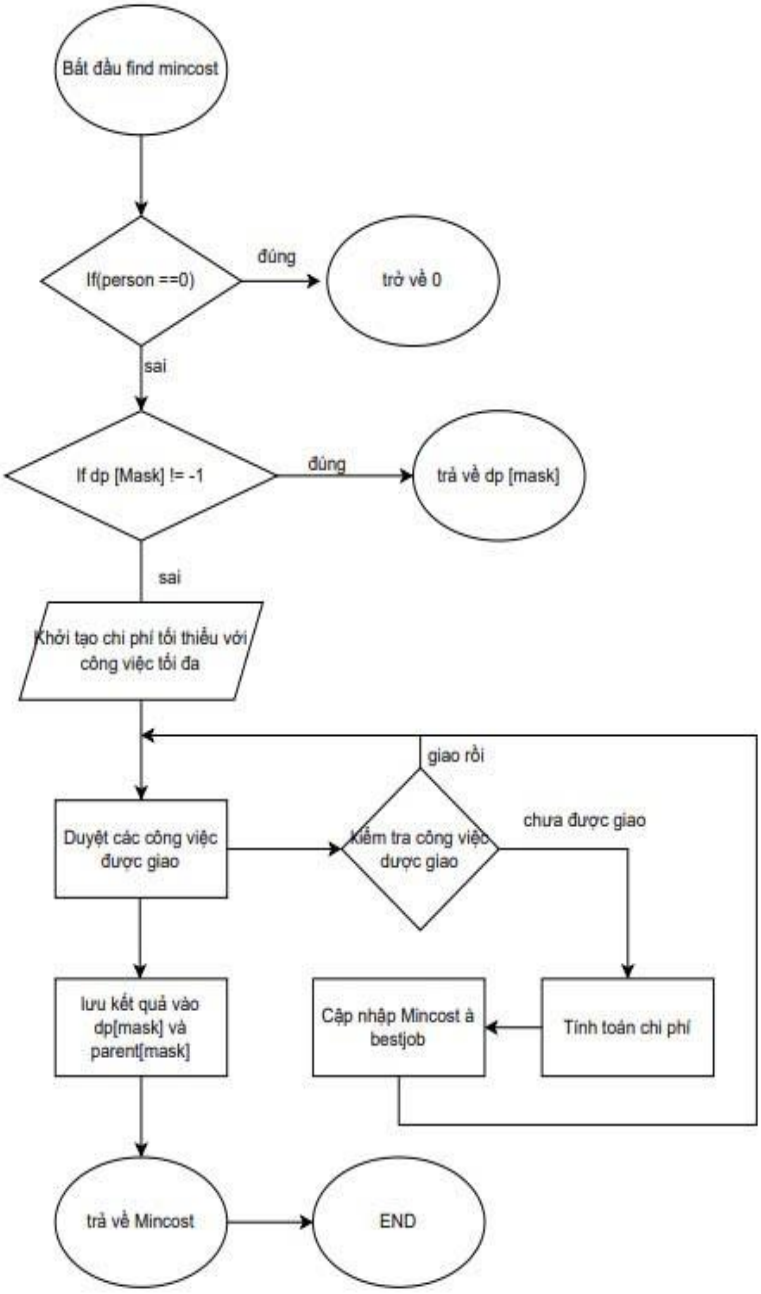
Bước 4: Tính toán Quy hoạch động

- Với mỗi trạng thái mask, xác định số người person đã được giao công việc.
- Duyệt qua các công việc chưa được giao job (những công việc có bit chưa bật mask).
- Cập nhật dp[mask] sử dụng công thức chuyển trạng thái.

Bước 5: Kết xuất kết quả

- Chi phí tối thiểu: Nằm ở dp[0].
- Ghi nhận phân công công việc:Từ dp[mask] truy vết lại các công việc và gán vào mảng assignment.

2.2.1.4. Lưu đồ thuật toán



Hình 6. Sơ đồ thuật toán Quy hoạch động

2.2.2. Thuật toán Hungary

2.2.2.1. Thuật toán Hungary trong bài toán phân công công việc

- **Mục tiêu:** Tìm cách phân công n công việc cho n người sao cho tổng chi phí thực hiện là nhỏ nhất.Cụ thể Cho một ma trận costMatrix, trong đó costMatrix[i][j] biểu diễn chi phí người i thực hiện công việc j. Bài toán yêu cầu ghép nối từng người với một công việc sao cho tổng chi phí là tối ưu.
- **Phương pháp:** Sử dụng Hungarian Algorithm:

+ Đây là một thuật toán tuyến tính (combinatorial optimization) để giải bài toán ghép nối nhị phân trong đồ thị cân bằng với chi phí tối ưu.
+ Thuật toán vận hành dựa trên việc giảm dần giá trị của slack trong các vòng lặp và duy trì nhãn (label) để tìm được ghép nối tối ưu.

2.2.2.2. Bài toán và phân tích yêu cầu

- Input:
  + n: Số nhân viên và số công việc.
  + Ma trận chi phí costMatrix: Một ma trận n×n trong đó costMatrix[i][j] biểu diễn chi phí mà người thứ i thực hiện công việc j.
- Output:
  + Tổng chi phí tối thiểu để gán tất cả các công việc cho các nhân viên.
  + Mảng phân công assignment: Cho biết công việc nào được giao cho từng người.
- Ràng buộc:
  + Mỗi nhân viên chỉ được gán một công việc
  + n là số nguyên dương, giới hạn khả thi n≤20 vì thuật toán có độ phức tạp O(n·2n)
  + Giá trị trong ma trận chi phí costMatrix[i][j] là các số nguyên không âm
  + Mỗi công việc chỉ được thực hiện bởi một nhân viên

2.2.2.3. Các bước giải thuật

Bước 1: Khởi tạo nhãn (labeling)

- Xác định nhãn ban đầu cho người và công việc

$$labelByWorker[i] = \min(costMatrix[i][j]) \quad \forall i,$$

$$labelByJob[j] = 0 \quad \forall j.$$

- Nhãn này đảm bảo giá trị slack của tất cả các ghép nối là không âm

$$slack[j] = costMatrix[i][j] - labelByWorker[i] - labelByJob[j] \geq 0.$$

Bước 2: Tìm ghép nối tối ưu

- Lặp qua từng người (worker):Với mỗi người i, tìm công việc chưa được ghép nối sao cho tổng slack là nhỏ nhất:
  + Duyệt qua toàn bộ công việc, kiểm tra nếu slack[j]=0
  + Nếu có ghép nối khả thi, gán nhãn phù hợp.
- Nếu không tồn tại ghép nối trực tiếp:Tăng nhãn (Update Labels):

+ Tăng nhãn labelByWorker hoặc giảm labelByJob để tối thiểu hóa giá trị slack, đảm bảo trạng thái mới không làm hỏng các ghép nối hiện có.

Bước 3: Lặp ghép nối

- Tiếp tục lặp qua các người và công việc, cập nhật trạng thái ghép nối.
- Nếu một công việc được gán, cập nhật mảng ghép nối:

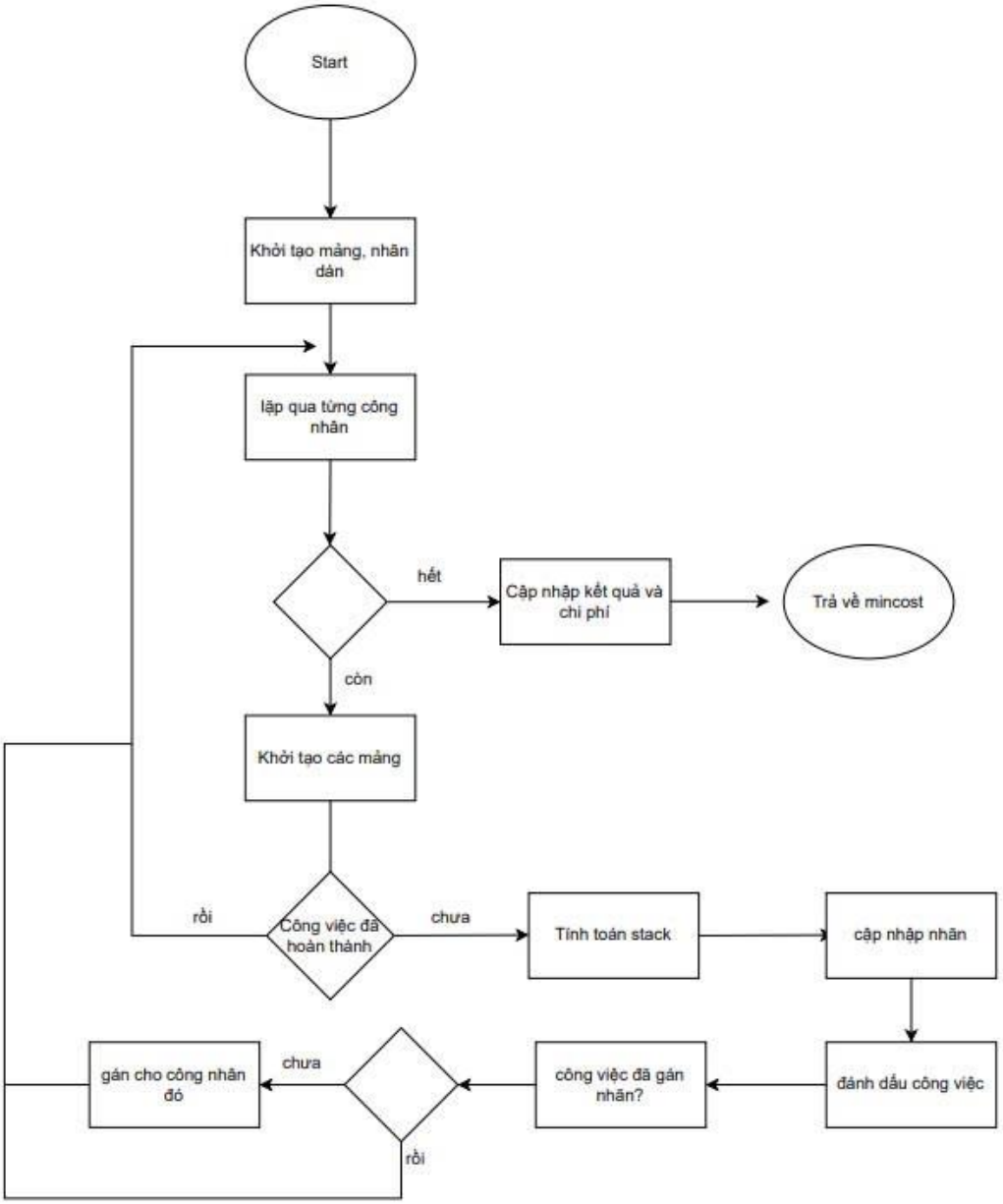
$$matchWorkerByJob[j] = i, \quad matchJobByWorker[i] = j$$

Bước 4: Kết thúc thuật toán

- Khi không còn người nào chưa được ghép nối:
  + Duyệt qua matchWorkerByJob[j] , xây dựng mảng assignment.
  + Tính toán tổng chi phí tối thiểu

$$minCost = \sum_{i=0}^{n-1} costMatrix[i][assignment[i]]$$

2.2.2.4. Lưu đồ thuật toán



Hình 7. Lưu đồ thuật toán Hungary

CHƯƠNG 3 : CÀI ĐẶT VÀ KIỂM THỬ THUẬT TOÁN

3.1. Môi trường, ngôn ngữ và công cụ triển khai

3.1.1. Môi trường

Môi trường IDE (Integrated Development Environment): Là các phần mềm tích hợp đầy đủ các công cụ phát triển phần mềm, hỗ trợ viết, biên dịch, debug và quản lý mã nguồn rất tốt giúp tăng năng suất lập trình, phát hiện lỗi nhanh và quản lý hiệu quả dự án.

Ví dụ: Apache NetBeans, Visual Studio Code, Eclipse, ...

Môi trường trực tuyến: được lập trình chạy trực tiếp trên trình duyệt mà không cần cài đặt phần mềm. Dễ dàng chia sẻ và truy cập nhưng hạn chế của nó là phụ thuộc vào Internet, hiệu năng kém và bảo mật thấp.

Môi trường điện toán đám mây: là môi trường máy tính được cung cấp qua Internet cho phép lưu trữ, quản lý và xử lý dữ liệu từ xa. Hơn nữa còn cung cấp tài nguyên máy tính một cách linh hoạt. Có khả năng mở rộng và bảo mật cao giúp phân tích các dữ liệu lớn của bài toán. Điểm hạn chế của nó là phụ thuộc nhiều vào Internet kèm theo đó chi phí có thể tăng cao nếu sử dụng nhiều.

3.1.2. Ngôn ngữ lập trình

Trong Cuốn báo cáo này, ngôn ngữ lập trình Java được sử dụng để giải quyết các vấn đề trong bài toán. Java (phiên âm Tiếng Việt: "Gia-va") là một ngôn ngữ lập trình hướng đối tượng, dựa trên lớp được thiết kế để có càng ít phụ thuộc thực thi càng tốt. Nó là ngôn ngữ lập trình có mục đích chung cho phép các nhà phát triển ứng dụng viết một lần, chạy ở mọi nơi (WORA), nghĩa là mã Java đã biên dịch có thể chạy trên tất cả các nền tảng hỗ trợ Java mà không cần biên dịch lại.

Lịch sử:

Java được khởi xướng vào tháng 6 năm 1991 bởi James Gosling, Mike Sheridan và Patrick Naughton tại Sun Microsystems, ban đầu nhằm phục vụ cho truyền hình tương tác nhưng bị xem là quá tiên tiến vào thời điểm đó. Ban đầu có tên là Oak, ngôn ngữ này sau đó được đổi thành Java, lấy cảm hứng từ cà phê Java của Indonesia. Java được thiết kế với cú pháp tương tự C/C++ để dễ dàng tiếp cận lập trình viên.

Sun Microsystems phát hành phiên bản đầu tiên, Java 1.0, vào năm 1996 với triết lý "Viết một lần, chạy mọi nơi" (WORA), cung cấp môi trường chạy miễn phí trên các nền tảng phổ biến và bảo mật cao. Java nhanh chóng phổ biến

nhờ sự tích hợp trong trình duyệt web, hỗ trợ các applet. Với Java 2 (J2SE 1.2) ra mắt năm 1998-1999, Sun chia Java thành ba cấu hình: J2SE cho máy tính để bàn, J2EE cho ứng dụng doanh nghiệp, và J2ME cho thiết bị di động, sau này được đổi tên thành Java SE, Java EE, và Java ME vào năm 2006.

Năm 2006, Sun phát hành phần lớn mã nguồn Java dưới dạng mã nguồn mở theo Giấy phép Công cộng GNU (GPL). Đến năm 2009, Oracle mua lại Sun Microsystems và trở thành nhà quản lý Java, nhưng sau đó đã kiện Google vì sử dụng Java trong Android SDK. Vào năm 2016, Oracle thông báo ngừng hỗ trợ plugin Java trên trình duyệt từ JDK 9.

Ngày nay, Java chạy trên mọi nền tảng, từ máy tính cá nhân đến trung tâm dữ liệu, và đóng vai trò quan trọng trong phát triển ứng dụng doanh nghiệp, di động, và hệ thống nhúng.

Ưu điểm của java:

+ Chạy được trên nhiều nền tảng
+ Có cơ chế quản lý bộ nhớ tự động (Garbage Collection) nên rất an toàn
+ Hướng đối tượng mạnh mẽ
+ Cộng đồng lập trình lớn và hỗ trợ tốt
+ Hiệu năng ổn định
+ Bảo mật cao
+ Hỗ trợ đa luồng
+ Nhiều thư viện và framework mạnh mẽ

Nhược điểm của java:

+ Tốc độ thực thi chậm hơn so với C/C++
+ Yêu cầu nhiều bộ nhớ
+ Hiệu năng kém so với các ngôn ngữ biên dịch trực tiếp
+ Mã nguồn dài dòng

3.1.3. Công cụ triển khai

3.1.3.1. Thư viện Java

Có rất nhiều thư viện và công cụ hữu ích trong Java giúp nghiên cứu, kiểm thử và trực quan hóa các thuật toán. Dưới đây là một số thư viện phổ biến:

- **Junit:** là một khung kiểm thử đơn vị cho ngôn ngữ lập trình Java, giúp bạn kiểm tra các phương thức và lớp của mã nguồn một cách tự động.

  Tính năng nổi bật:

  + Viết các bài kiểm thử tự động.
  + Dễ dàng tích hợp với các công cụ CI/CD như Jenkins, Travis CI.
  + Hỗ trợ kiểm tra các ngoại lệ, thời gian chạy và hơn thế nữa.

- **TestNG:** là một khung kiểm thử được lấy cảm hứng từ JUnit và NUnit, với nhiều tính năng hơn như kiểm thử nhóm, phụ thuộc kiểm thử, và cấu hình linh hoạt.

  Tính năng nổi bật:

  + Hỗ trợ kiểm thử nhóm và phụ thuộc giữa các bài kiểm thử.
  + Tích hợp dễ dàng với các công cụ báo cáo và CI/CD.
  + Quản lý và chạy các bài kiểm thử từ tệp XML.
- **Apache Commons Math**

  Thư viện này cung cấp các công cụ toán học và thống kê cần thiết cho việc phân tích và tối ưu hóa thuật toán.

  Tính năng nổi bật:

  + Các thuật toán tối ưu hóa và phân tích dữ liệu.
  + Hỗ trợ các phép toán ma trận, giải hệ phương trình, và thống kê.
- **JFreeChart**

  JFreeChart là một thư viện Java mạnh mẽ để tạo các biểu đồ chất lượng cao.

  Tính năng nổi bật:

  + Hỗ trợ nhiều loại biểu đồ như biểu đồ cột, đường, bánh, v.v.
  + Tích hợp dễ dàng với các ứng dụng Java.
  + Hỗ trợ tùy chỉnh cao và tương tác với các biểu đồ.
- **GraphStream**

  GraphStream là một thư viện Java cho phép bạn tạo, xử lý và trực quan hóa đồ thị.

  Tính năng nổi bật:

  + Hỗ trợ các thao tác cơ bản và nâng cao với đồ thị.
  + Các công cụ trực quan hóa đồ thị mạnh mẽ.
  + Hỗ trợ thời gian thực và tích hợp dễ dàng với các dự án Java

3.1.3.2. IDE và công cụ phát triển

- **NetBeans:** Là một IDE mạnh mẽ với gioa diện trực quan, phù hợp với cả người mới học và lập trình viên chuyên nghiệp cũng như việc nghiên cứu các thuật toán liên quan đến ngôn ngữ java. Trong báo cáo này chúng tôi sử dụng phần lớn là NetBeans IDE.
- **VSCode**: Nếu cần một IDE mạnh mẽ, hỗ trợ quản lý dự án lớn, debugging, profiling tốt.

- **Eclipse/IntelliJ**: IDE được tích hợp rất nhiều công cụ hỗ trợ mạnh mẽ cho ngôn ngữ Java

3.1.3.3. Công cụ hỗ trợ tính toán phân tán

- **Apache Hadoop:** là một framework mã nguồn mở cho phép xử lý và lưu trữ lượng lớn dữ liệu trên một cụm máy phân tán. Các thành phần chính bao gồm Hadoop Distributed File System (HDFS) và MapReduce.
- **Apache Spark:** là một hệ thống xử lý dữ liệu phân tán hiệu năng cao, cung cấp giao diện lập trình dễ sử dụng cho xử lý dữ liệu phân tán.
- **Hazelcast:** là một nền tảng tính toán phân tán trong bộ nhớ mạnh mẽ, giúp dễ dàng xây dựng các ứng dụng có độ trễ thấp.
- **Akka:** là một công cụ mạnh mẽ cho việc xây dựng các ứng dụng phân tán, hướng sự kiện bằng cách sử dụng mô hình diễn viên (Actor Model).

3.1.3.4. Công cụ trực quan hóa

- **Matplotlib** : Dùng để vẽ biểu đồ biểu diễn quá trình hội tụ hoặc vẽ sơ đồ tour tốt nhất mà thuật toán tìm ra.
- **Tableau** hoặc **Google Data Studio**: Dùng để báo cáo kết quả dưới dạng dashboard.

3.2. Dữ liệu thực nghiệm

- Mô tả: Bộ dữ liệu bao gồm một số công nhân và các mức lương mà công nhân đó yêu cầu tương ứng với từng công việc.
- Mục tiêu: Tìm ra tổng chi phí tối thiểu, và các công việc mà từng công nhân phải làm sao cho mỗi công nhân chỉ làm một công việc duy nhất.
- Dữ liệu đầu vào: Số công nhân, số công việc và ma trận chi phí, ma trận này tương ứng với từng chi phí mà công nhân yêu cầu nhận được tương ứng với từng công việc.

Bộ dữ liệu 1: Gồm 4 công nhân và 4 công việc:

4
4

	Cv1	Cv2	Cv3	Cv3
Cn1	9	2	7	8
Cn2	6	4	3	7
Cn3	5	8	1	8
Cn4	7	6	9	4

Bộ dữ liệu 2: Gồm 6 công nhân và 6 công việc:

6
6
10 15 20 5 10 6

6 14 15 8 12 10
12 10 15 5 8 14
6 12 10 7 5 9
9 7 10 5 15 6
8 6 12 10 9 15

Bộ dữ liệu 3: Gồm 5 công nhân và 5 công việc:

5
5
10 1 5 8 3
4 6 2 9 7
8 4 1 2 5
7 3 6 4 8
9 5 7 3 2

Bộ dữ liệu 4: Gồm 15 công nhân và 15 công việc:

15
15
12 7 9 7 9 8 6 7 8 9 10 11 12 13 14
8 7 12 8 7 6 5 8 6 7 8 9 10 11 12
9 10 6 7 8 5 4 5 6 5 4 3 2 1 8
10 5 7 10 6 7 9 8 7 6 5 4 3 2 1
7 8 9 6 5 4 7 6 5 4 3 2 1 9 8
8 7 6 5 4 3 2 1 9 8 7 6 5 4 3
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
6 5 4 3 2 1 7 8 9 10 11 12 13 14 15
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
15 13 11 9 7 5 3 1 2 4 6 8 10 12 14
14 12 10 8 6 4 2 1 3 5 7 9 11 13 15

Dữ liệu đầu ra: Hiện ra được tổng chi phí tối thiểu, và công nhân sẽ làm công việc gì

Ví dụ về kết quả mong đợi:

Chi phi toi thieu: 38
Cong nhan 1 lam cong viec 6
Cong nhan 2 lam cong viec 1
Cong nhan 3 lam cong viec 4
Cong nhan 4 lam cong viec 5
Cong nhan 5 lam cong viec 3

Cong nhan 6 lam cong viec 2

3.3. Khởi tạo thuật toán

3.3.1. Thuật toán quy hoạch động

```
public class DynamicProgrammingSolver implements Solver {

    @Override
    public int solve(int[][] costMatrix, int[] assignment) {
        int n = costMatrix.length; // Số lượng công việc và công nhân
        int[] dp = new int[1 << n]; // Khởi tạo mảng dp với 2^n trạng thái
        int[] parent = new int[1 << n]; // Lưu trạng thái công việc tối ưu
        Arrays.fill(dp, -1);
        Arrays.fill(parent, -1);
        Arrays.fill(assignment, -1);

        // Gọi hàm tìm chi phí tối thiểu
        int minCost = findMinCost(costMatrix, dp, parent, 0, 0, n);

        // Truy vết ngược để tìm mảng assignment
        int mask = 0;
        for (int person = 0; person < n; person++) {
            int job = parent[mask]; // Lấy công việc gán từ trạng thái hiện tại
            assignment[person] = job; // Gán công việc cho nhân viên
            mask |= (1 << job); // Cập nhật trạng thái
        }

        return minCost; // Trả về chi phí tối thiểu
    }

    private int findMinCost(int[][] costMatrix, int[] dp, int[] parent, int mask,
int person, int n) {
        if (person == n) { // Nếu tất cả các công việc đã được giao
```

```
        return 0; // Không còn chi phí nào thêm
    }
    if (dp[mask] != -1) { // Nếu trạng thái này đã được tính toán
        return dp[mask]; // Trả về giá trị đã lưu
    }
    int minCost = Integer.MAX_VALUE; // Khởi tạo chi phí tối thiểu
    int bestJob = -1; // Khởi tạo công việc tối ưu
    for (int job = 0; job < n; job++) {
        if ((mask & (1 << job)) == 0) { // Nếu công việc chưa được giao
            int cost = costMatrix[person][job]
                    + findMinCost(costMatrix, dp, parent, mask | (1 << job),
person + 1, n); // Tính toán chi phí
            if (cost < minCost) { // Nếu chi phí tính toán ít hơn chi phí tối
thiểu hiện tại
                minCost = cost; // Cập nhật chi phí tối thiểu
                bestJob = job; // Cập nhật công việc tối ưu
            }
        }
    }
    dp[mask] = minCost; // Lưu kết quả vào mảng dp
    parent[mask] = bestJob; // Lưu công việc tối ưu vào mảng parent
    return minCost; // Trả về chi phí tối thiểu
}
```

Giải thích thuật toán quy hoạch động:

Hàm solve

1. **Ý nghĩa:**
   - Giải bài toán giao việc, trả về chi phí tối thiểu và mảng assignment chứa cách gán công việc.
2. **Các bước:**
   - Khởi tạo:
     - dp: Lưu chi phí tối thiểu của mỗi trạng thái.
     - parent: Lưu công việc tối ưu tại mỗi trạng thái.

        - assignment: Lưu kết quả gán công việc.
    - Gọi hàm findMinCost để tính chi phí tối thiểu.
    - Truy vết ngược qua parent để tìm cách gán công việc cho từng công nhân.
    - Trả về tổng chi phí tối thiểu.

Hàm findMinCost

1. **Ý nghĩa:**
    - Tính tổng chi phí tối thiểu khi:
        - mask: Các công việc đã được giao.
        - person: Công nhân đang xét.
    - Lưu kết quả vào dp[mask] để tái sử dụng.
2. **Các bước:**
    - **Điều kiện dừng:** Nếu tất cả công việc đã được giao (person == n), chi phí là 0.
    - **Trường hợp đã tính trước:** Nếu trạng thái đã được tính (dp[mask] != -1), trả về giá trị lưu sẵn.
    - **Duyệt qua các công việc chưa được giao:**
        - Tính chi phí khi gán công việc đó cho person.
        - Gọi đệ quy cho trạng thái mới (mask | (1 << job)).
        - Cập nhật chi phí tối thiểu và công việc tối ưu.
    - **Lưu kết quả:**
        - dp[mask]: Chi phí tối thiểu.
        - parent[mask]: Công việc tối ưu tại trạng thái hiện tại.

Ý tưởng chính

- **Trạng thái mask:** Số nhị phân biểu diễn công việc đã giao.
- **Quy hoạch động:** Lưu chi phí tối thiểu của từng trạng thái để tối ưu hóa.
- **Truy vết ngược:** Sử dụng mảng parent để xây dựng cách gán công việc từ trạng thái ban đầu.

Độ phức tạp

- **Thời gian:** O(n×2^n) (Duyệt 2^n trạng thái, mỗi trạng thái xét tối đa nnn công việc).
- **Không gian:** O(2^n) cho mảng dp và parent.

3.3.2. Thuật toán Hungary

package algorithms;

```
import java.util.Arrays;

public class HungarianSolver implements Solver {

    @Override
    public int solve(int[][] costMatrix, int[] assignment) {
        int n = costMatrix.length;
        // Các nhãn của công nhân và công việc
        int[] labelByWorker = new int[n];
        int[] labelByJob = new int[n];
        int[] minSlackWorker = new int[n];
        int[] minSlackValue = new int[n];

        // Các mảng lưu kết quả
        int[] matchWorkerByJob = new int[n];
        int[] matchJobByWorker = new int[n];
        Arrays.fill(matchWorkerByJob, -1);
        Arrays.fill(matchJobByWorker, -1);

        // Khởi tạo nhãn
        for (int i = 0; i < n; i++) {
            labelByWorker[i]                                    =
Arrays.stream(costMatrix[i]).min().orElse(0);
        }

        // Xử lý từng công nhân
        for (int worker = 0; worker < n; worker++) {
            Arrays.fill(minSlackValue, Integer.MAX_VALUE);
            boolean[] visitedWorkers = new boolean[n];
            boolean[] visitedJobs = new boolean[n];
            int[] parentJob = new int[n];
            Arrays.fill(parentJob, -1);
```

```
            int currentWorker = worker;

            while (true) {
                visitedWorkers[currentWorker] = true;

                int committedJob = -1;
                int delta = Integer.MAX_VALUE;

                // Tính toán slack
                for (int job = 0; job < n; job++) {
                    if (!visitedJobs[job]) {
                        int slack = costMatrix[currentWorker][job] -
labelByWorker[currentWorker] - labelByJob[job];
                        if (slack < minSlackValue[job]) {
                            minSlackValue[job] = slack;
                            minSlackWorker[job] = currentWorker;
                        }
                        if (minSlackValue[job] < delta) {
                            delta = minSlackValue[job];
                            committedJob = job;
                        }
                    }
                }

                // Cập nhật nhãn
                for (int j = 0; j < n; j++) {
                    if (visitedJobs[j]) {
                        labelByWorker[matchWorkerByJob[j]] += delta;
                        labelByJob[j] -= delta;
                    } else {
                        minSlackValue[j] -= delta;
```

```
            }
        }

        // Đánh dấu công việc
        visitedJobs[committedJob] = true;
        int matchedWorker = matchWorkerByJob[committedJob];

        if (matchedWorker == -1) {
            // Truy vết ngược để cập nhật kết quả gán
            while (committedJob != -1) {
                int parentWorker = minSlackWorker[committedJob];
                int temp = matchJobByWorker[parentWorker];
                matchWorkerByJob[committedJob] = parentWorker;
                matchJobByWorker[parentWorker] = committedJob;
                committedJob = temp;
            }
            break;
        } else {
            currentWorker = matchedWorker;
        }
    }
}
// Tính tổng chi phí tối thiểu và cập nhật assignment
int minCost = 0;
for (int job = 0; job < n; job++) {
    if (matchWorkerByJob[job] != -1) {
        assignment[matchWorkerByJob[job]] = job;
        minCost += costMatrix[matchWorkerByJob[job]][job];
    }
}
return minCost;
```

```
    }
}
```

Giải thích thuật toán:

1. **Khởi tạo các mảng hỗ trợ:**
    - labelByWorker và labelByJob: Mảng lưu nhãn (label) cho công nhân và công việc. Các nhãn này được sử dụng để điều chỉnh chi phí gán công việc sao cho tối ưu.
    - minSlackWorker và minSlackValue: Dùng để lưu công nhân có slack nhỏ nhất cho mỗi công việc, cùng với giá trị slack tương ứng.
    - matchWorkerByJob và matchJobByWorker: Mảng lưu thông tin về việc gán công nhân cho công việc, ngược lại từ công nhân và công việc.
2. Khởi tạo nhãn ban đầu cho công nhân:
    - Đối với mỗi công nhân, nhãn được khởi tạo bằng giá trị nhỏ nhất trong dòng của ma trận chi phí tương ứng với công nhân đó. Điều này giúp giảm chi phí khởi điểm khi gán công việc.
3. Thuật toán Hungary:
    - Với mỗi công nhân, thuật toán tìm công việc tối ưu bằng cách tính toán slack cho từng công việc chưa được gán.
    - Slack là sự chênh lệch giữa chi phí thực tế và chi phí đã được điều chỉnh bởi nhãn của công nhân và công việc.
    - Dùng giá trị slack để tìm công việc phù hợp với công nhân hiện tại. Sau đó, cập nhật các nhãn (labels) và slack để đảm bảo sự tối ưu trong các vòng tiếp theo.
4. Cập nhật nhãn và slack:
    - Mỗi khi công nhân được gán một công việc, thuật toán cập nhật nhãn của công nhân và công việc dựa trên giá trị slack tối thiểu tìm được.
    - Các công việc đã được gán sẽ có nhãn điều chỉnh sao cho chi phí tổng thể giảm thiểu, trong khi các công việc chưa được gán sẽ tiếp tục được điều chỉnh slack.
5. Truy vết và gán công việc:
    - Nếu công việc không có công nhân được gán ngay lập tức, thuật toán sẽ truy vết ngược qua các mảng minSlackWorker và matchJobByWorker để tìm ra công nhân phù hợp cho công việc đó.
    - Quá trình truy vết này đảm bảo rằng các công việc được gán đúng công nhân với chi phí thấp nhất.
6. Tính toán chi phí tối thiểu:

- Sau khi hoàn thành quá trình gán công việc cho công nhân, thuật toán tính tổng chi phí tối thiểu bằng cách cộng dồn chi phí từ các công việc đã được gán cho công nhân.

7. Kết quả:
   - Tổng chi phí tối thiểu và mảng assignment chứa thông tin công nhân được gán cho công việc tương ứng.

Mục tiêu chính của thuật toán Hungary:

- Tìm gán tối ưu: Thuật toán giúp tìm ra cách phân công công nhân vào các công việc sao cho tổng chi phí là tối thiểu.
- Điều chỉnh nhãn: Thực hiện điều chỉnh nhãn để giảm thiểu chi phí mỗi khi công nhân và công việc được gán.
- Slack: Slack giúp đánh giá và tối ưu chi phí trong mỗi vòng lặp của thuật toán.

Độ phức tạp:

- Thời gian: O(n^3), do mỗi công nhân phải duyệt qua tất cả các công việc (tối đa nnn) và điều chỉnh nhãn và slack cho từng công việc.
- Không gian: O(n), do lưu trữ nhãn và thông tin gán cho công nhân và công việc.

3.3.3. Hàm main

```
package main;

import algorithms.*;

import java.util.Scanner;

public class AssignmentProblem {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        //System.out.print("Nhap so cong nhan: ");

        int workers = scanner.nextInt(); // Nhap so luong cong nhan

        //System.out.print("Nhap so cong viec: ");

        int jobs = scanner.nextInt(); // Nhap so luong cong viec
```

```
        if (workers != jobs) {
            System.out.println("So luong cong nhan va cong viec phai bang nhau.");
            return;
        }
        int[][] costMatrix = new int[workers][jobs];
       // System.out.println("Nhap cac phan tu ma tran chi phi (Theo tung hang):");
        for (int i = 0; i < workers; i++) {
            for (int j = 0; j < jobs; j++) {
                costMatrix[i][j] = scanner.nextInt(); // Nhap cac gia tri cua ma tran chi phi
            }
        }
        Solver[] solvers = {
            new DynamicProgrammingSolver(), // Khoi tao solver Dynamic Programming
            new HungarianSolver() // Khoi tao solver Hungarian
        };
        String[] solverNames = {"Dynamic Programming", "Hungarian"}; // Ten cac solver
        for (int i = 0; i < solvers.length; i++) {
            int[] assignment = new int[workers]; // Mang assignment de luu ket qua phan cong
            int result = solvers[i].solve(costMatrix, assignment); // Giai bai toan phan cong
```

```
            System.out.println("\n" + solverNames[i] + ":");

            System.out.println("Chi phi toi thieu: " + result); // In ra chi
        phi toi thieu

            for (int j = 0; j < assignment.length; j++) {

                System.out.println("Cong nhan " + (j + 1) + " lam cong
        viec " + (assignment[j] + 1)); // In ra ket qua phan cong

            }

        }

        scanner.close(); // Dong Scanner de tranh ro ri tai nguyen

    }

}
```

3.4. Kiểm thử

Input:

- Dòng đầu là số lượng công việc
- Dòng 2 cũng là số lượng công nhân.
- Ma trận tiếp theo là chi phí để hoàn thành từng loại công việc của mỗi công nhân.

Output

- Chi phí tối tiểu
- Các công việc mà từng công nhân sẽ làm

3.4.1. Thuật toán quy hoạch động

❖ Test 1:

Input:

6

6

10 15 20 5 10 6

6 14 15 8 12 10

12 10 15 5 8 14

6 12 10 7 5 9

9 7 10 5 15 6

8 6 12 10 9 15

Kết quả:

```
Dynamic Programming:
Chi phi toi thieu: 38
Cong nhan 1 lam cong viec 6
Cong nhan 2 lam cong viec 1
Cong nhan 3 lam cong viec 4
Cong nhan 4 lam cong viec 5
Cong nhan 5 lam cong viec 3
Cong nhan 6 lam cong viec 2
```

Hình 8. Kết quả thuật toán quy hoạch động với ma trận chi phí nhỏ

❖ Test 2:

Input:

15

15

12 7 9 7 9 8 6 7 8 9 10 11 12 13 14

8 7 12 8 7 6 5 8 6 7 8 9 10 11 12

9 10 6 7 8 5 4 5 6 5 4 3 2 1 8

10 5 7 10 6 7 9 8 7 6 5 4 3 2 1

7 8 9 6 5 4 7 6 5 4 3 2 1 9 8

8 7 6 5 4 3 2 1 9 8 7 6 5 4 3

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

24 23 22 21 20 19 18 17 16 15 14 13 12 11 10

5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

19 18 17 16 15 14 13 12 11 10 9 8 7 6 5

6 5 4 3 2 1 7 8 9 10 11 12 13 14 15

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

15 13 11 9 7 5 3 1 2 4 6 8 10 12 14

14 12 10 8 6 4 2 1 3 5 7 9 11 13 15

Kết quả:

```
Dynamic Programming:
Chi phi toi thieu: 70
Cong nhan 1 lam cong viec 4
Cong nhan 2 lam cong viec 5
Cong nhan 3 lam cong viec 10
Cong nhan 4 lam cong viec 11
Cong nhan 5 lam cong viec 12
Cong nhan 6 lam cong viec 7
Cong nhan 7 lam cong viec 1
Cong nhan 8 lam cong viec 13
Cong nhan 9 lam cong viec 2
Cong nhan 10 lam cong viec 14
Cong nhan 11 lam cong viec 6
Cong nhan 12 lam cong viec 15
Cong nhan 13 lam cong viec 3
Cong nhan 14 lam cong viec 9
Cong nhan 15 lam cong viec 8
```

Hình 9. Kết quả thuật toán quy hoạch động với ma trận chi phí lớn

3.4.2. Thuật toán Hungary

- Test 1:

Input:

6

6

10 15 20 5 10 6

6 14 15 8 12 10

12 10 15 5 8 14

6 12 10 7 5 9

9 7 10 5 15 6

8 6 12 10 9 15

Kết quả:

```
Hungarian:
Chi phi toi thieu: 38
Cong nhan 1 lam cong viec 6
Cong nhan 2 lam cong viec 1
Cong nhan 3 lam cong viec 4
Cong nhan 4 lam cong viec 5
Cong nhan 5 lam cong viec 3
Cong nhan 6 lam cong viec 2
BUILD SUCCESSFUL (total time: 1 second)
```

Hình 10. Kết quả thuật toán Hungarian với ma trận chi phí nhỏ

❖ Test 2:

Input:

15

15

12 7 9 7 9 8 6 7 8 9 10 11 12 13 14

8 7 12 8 7 6 5 8 6 7 8 9 10 11 12

9 10 6 7 8 5 4 5 6 5 4 3 2 1 8

10 5 7 10 6 7 9 8 7 6 5 4 3 2 1

7 8 9 6 5 4 7 6 5 4 3 2 1 9 8

8 7 6 5 4 3 2 1 9 8 7 6 5 4 3

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

24 23 22 21 20 19 18 17 16 15 14 13 12 11 10

5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

19 18 17 16 15 14 13 12 11 10 9 8 7 6 5

6 5 4 3 2 1 7 8 9 10 11 12 13 14 15

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

15 13 11 9 7 5 3 1 2 4 6 8 10 12 14

14 12 10 8 6 4 2 1 3 5 7 9 11 13 15

Kết quả:

```
Hungarian:
Chi phi toi thieu: 70
Cong nhan 1 lam cong viec 4
Cong nhan 2 lam cong viec 7
Cong nhan 3 lam cong viec 14
Cong nhan 4 lam cong viec 10
Cong nhan 5 lam cong viec 13
Cong nhan 6 lam cong viec 5
Cong nhan 7 lam cong viec 1
Cong nhan 8 lam cong viec 12
Cong nhan 9 lam cong viec 2
Cong nhan 10 lam cong viec 11
Cong nhan 11 lam cong viec 6
Cong nhan 12 lam cong viec 15
Cong nhan 13 lam cong viec 3
Cong nhan 14 lam cong viec 9
Cong nhan 15 lam cong viec 8
```

Hình 11. Kết quả thuật toán Hungarian với ma trận chi phí lớn

3.4.3. So sánh

1. Quy hoạch động (Dynamic Programming)

Độ chính xác:

- Quy hoạch động luôn đảm bảo tìm ra giải pháp tối ưu cho bài toán phân chia công việc. Độ chính xác của thuật toán là tuyệt đối, miễn là không có lỗi trong quá trình triển khai.

Tốc độ:

- Input nhỏ (N <= 10): Quy hoạch động hoạt động khá tốt và nhanh chóng. Với các bài toán nhỏ, thời gian thực thi thường là chấp nhận được.

- Input lớn (N > 10): Quy hoạch động trở nên chậm hơn đáng kể do độ phức tạp thời gian và không gian là O(n * 2^n). Điều này làm cho thuật toán không khả thi với các bài toán có kích thước lớn.

2. Thuật toán Hungary (Hungarian Algorithm)

Độ chính xác:

- Thuật toán Hungary cũng đảm bảo tìm ra giải pháp tối ưu cho bài toán phân chia công việc. Độ chính xác của thuật toán là tuyệt đối, miễn là không có lỗi trong quá trình triển khai.

Tốc độ:

- Input nhỏ (N <= 10): Thuật toán Hungary hoạt động nhanh chóng và hiệu quả. Với các bài toán nhỏ, thời gian thực thi thường là rất nhanh.
- Input lớn (N > 10): Thuật toán Hungary vẫn hoạt động hiệu quả với độ phức tạp thời gian là O(n^3). Điều này làm cho thuật toán phù hợp hơn với các bài toán có kích thước lớn.

3. Kết luận

Độ chính xác: Cả hai thuật toán đều đảm bảo độ chính xác tuyệt đối.

Tốc độ:

- Với các bài toán nhỏ (N <= 10), cả hai thuật toán đều hoạt động nhanh chóng và hiệu quả.
- Với các bài toán lớn (N > 10), thuật toán Hungary vượt trội hơn về tốc độ và khả năng xử lý, trong khi quy hoạch động trở nên không khả thi do độ phức tạp thời gian và không gian.

KẾT LUẬN

Báo cáo này đã tập trung nghiên cứu cơ sở lý thuyết, ứng dụng thực tiễn và cài đặt hai thuật toán giải bài toán Phân công công việc (Job Assignment Problem): thuật toán Hungary và thuật toán Quy Hoạch Động (Dynamic Programming). Đây là những phương pháp tối ưu hóa nổi bật trong khoa học máy tính, giúp giải quyết hiệu quả các bài toán tối ưu liên quan đến phân phối nguồn lực, lập lịch trình và quản lý sản xuất.

Thuật toán Hungarian, dựa trên lý thuyết đồ thị, là một phương pháp hiệu quả để giải quyết bài toán Phân công công việc trong các trường hợp có ma trận chi phí vuông. Thuật toán này tìm kiếm lời giải tối ưu bằng cách xây dựng các nhãn trên đồ thị, định nghĩa các cặp ghép và điều chỉnh nhãn để giảm chi phí tổng thể. Với độ phức tạp tính toán O(n^3), Hungarian là một lựa chọn lý tưởng cho các bài toán có quy mô vừa và lớn, mang lại sự cân bằng giữa tính chính xác và hiệu quả thời gian xử lý. Tuy nhiên, trong trường hợp ma trận chi phí không vuông hoặc các bài toán với ràng buộc phức tạp hơn, cần thực hiện các bước tiền xử lý để đảm bảo tính khả thi.

Ngược lại, thuật toán Quy Hoạch Động sử dụng phương pháp chia để trị và tận dụng kết quả của các bài toán con chồng chéo. Với cách tiếp cận từ dưới lên (bottom-up), thuật toán này đảm bảo tính chính xác và tối ưu hóa tốc độ tính toán bằng cách lưu trữ kết quả trung gian. Đây là lựa chọn lý tưởng khi bài toán có cấu trúc rõ ràng và yêu cầu xử lý nhiều bài toán con lặp lại. Tuy nhiên, điểm hạn chế của Quy Hoạch Động là tiêu tốn bộ nhớ trong các bài toán có không gian trạng thái lớn.

Qua thực nghiệm, cả hai thuật toán đều cho thấy hiệu quả trong việc tối ưu hóa thời gian hoàn thành công việc hoặc chi phí phân công. Thuật toán Nhánh Cận phù hợp cho các trường hợp yêu cầu tối ưu toàn cục, trong khi Quy Hoạch Động là lựa chọn tốt khi cần giải quyết nhanh và chính xác các bài toán có quy mô lớn.

Bên cạnh giá trị học thuật, nghiên cứu này còn minh chứng khả năng ứng dụng của các thuật toán trong thực tiễn. Các lĩnh vực như quản lý sản xuất, lập lịch trình lao động, hoặc phân phối tài nguyên đều có thể áp dụng các phương pháp đã nghiên cứu để nâng cao hiệu quả vận hành. Trong tương lai, việc kết hợp các thuật toán này với các kỹ thuật hiện đại như heuristic hoặc học máy có thể mở ra nhiều tiềm năng giải quyết các bài toán phức tạp hơn trong thời gian ngắn hơn.

TÀI LIỆU THAM KHẢO

[1] Engineering Notes. (n.d.). *Assignment problem: Meaning, methods and variations*. Retrieved from https://www.engineeringenotes.com/project-management-2/operations-research/assignment-problem-meaning-methods-and-variations-operations-research/15652#google_vignette

[2]GeeksforGeeks. (n.d.). *Hungarian algorithm for assignment problem - Set 2 (implementation)*. Retrieved from https://www.geeksforgeeks.org/hungarian-algorithm-for-assignment-problem-set-2-implementation/

[3] HR Consulting Vietnam. (2016, February). *Sử dụng thuật toán Hungary để phân công*. Retrieved from https://hrconsultingvn.blogspot.com/2016/02/su-dung-thuat-toan-hungary-e-phan-cong.html

[4] TopDev. (n.d.). *Thuật toán quy hoạch động*. Retrieved from https://topdev.vn/blog/thuat-toan-quy-hoach-dong/

[5] Warbleton Council. (n.d.). *Método húngaro*. Retrieved from https://vi1.warbletoncouncil.org/metodo-hungaro-5228