**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO TOÁN RỜI RẠC**

**CHỦ ĐỀ: TỔNG QUAN VỀ CÁC KỸ THUẬT QUY HOẠCH ĐỘNG, NHÁNH CẬN**

**Học phần: Toán rời rạc**

**Nhóm HP: 63.CNTT-2**

**Họ và tên: Nguyễn Chí Hiếu**

**MSSV: 63134016**

**Giảng viên: GV. Nguyễn Hải Triều**

2024

# MỤC LỤC

[1 MỤC LỤC 1](#_Toc165151898)

[2 DANH MỤC HÌNH 1](#_Toc165151899)

[3 LÝ THUYẾT 3](#_Toc165151900)

[3.1 Kỹ Thuật Quy Hoạch Động 3](#_Toc165151901)

[3.1.1 Các kỹ thuật chung cho phương pháp quy hoạch động: 3](#_Toc165151902)

[3.2 Kỹ Thuật Nhánh và Cận: 4](#_Toc165151903)

[3.2.1 Các kỹ thuật chung cho phương pháp Nhánh và Cận: 4](#_Toc165151904)

[3.3 So sánh ưu, nhược điểm: 5](#_Toc165151905)

[4 LẬP TRÌNH (Ngôn ngữ Python) 6](#_Toc165151906)

[4.1 Bài toán xếp ba lô 1, 2: 6](#_Toc165151907)

[4.1.1 Mô Tả Bài Toán: 7](#_Toc165151908)

[4.1.2 Ý tưởng giải quyết bài toán: 8](#_Toc165151909)

[4.1.3 Cài đặt: 10](#_Toc165151910)

[4.2 Bài toán người du lịch - TSP (Travelling Salesman Problem) 12](#_Toc165151911)

[4.2.1 Mô Tả Bài Toán: 12](#_Toc165151912)

[4.2.2 Ý tưởng giải quyết bài toán: 13](#_Toc165151913)

[4.2.3 Cài đặt 14](#_Toc165151914)

[4.3 Bài toán rút tiền ATM 18](#_Toc165151915)

[4.3.1 Mô Tả Bài Toán: 18](#_Toc165151916)

[4.3.2 Ý tưởng giải quyết bài toán: 18](#_Toc165151917)

[4.3.3 Cài Đặt 19](#_Toc165151918)

[4.4 Bài toán Tháp Hà Nội 23](#_Toc165151919)

[4.4.1 Mô Tả Bài Toán: 23](#_Toc165151920)

[4.4.2 Ý tưởng giải quyết bài toán: 23](#_Toc165151921)

[4.4.3 Cài đặt 25](#_Toc165151922)

[4.5 IV. Link Github: 27](#_Toc165151923)

# 

# 

# DANH MỤC HÌNH

[Hình 3‑1 Quy Hoạch Động 3](#_Toc165153019)

[Hình 3‑2 Nhánh và Cận 4](#_Toc165153020)

[Hình 4‑1 Bài Toán Ba Lô 7](#_Toc165153021)

[Hình 4‑2 Phần chương trình sử lý bài toán ba lô 10](#_Toc165153022)

[Hình 4‑3 Ví dụ của Bài toán ba lô 10](#_Toc165153023)

[Hình 4‑4 Kết quả của bài toán ba lô 11](#_Toc165153024)

[Hình 4‑5 hàm hoán vị bài toán người du lịch 14](#_Toc165153025)

[Hình 4‑6 hàm tính chi phí bài toán người du lịch 15](#_Toc165153026)

[Hình 4‑7 hàm nhánh và cận bài toán người du lịch 15](#_Toc165153027)

[Hình 4‑8 chương trình chính bài toán người du lịch 17](#_Toc165153028)

[Hình 4‑9 Kết quả bài toán người du lịch 18](#_Toc165153029)

[Hình 4‑10 hàm init bài toán ATM 20](#_Toc165153030)

[Hình 4‑11 hàm optimize bài toán ATM 21](#_Toc165153031)

[Hình 4‑12 hàm result bài toán ATM 22](#_Toc165153032)

[Hình 4‑13 Kết quả bài toán ATM 23](#_Toc165153033)

[Hình 4‑14 hàm hanoi bài toán tháp hanoi 25](#_Toc165153034)

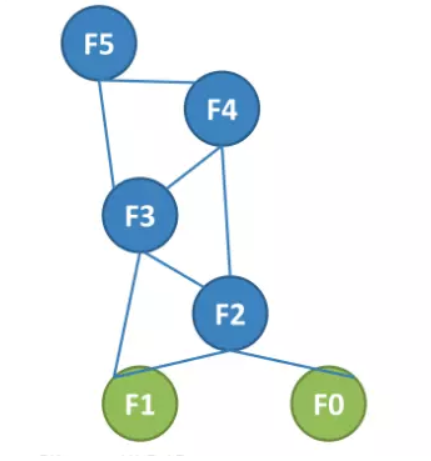
[Hình 4‑15 hàm hanoi\_dp bài toán tháp hanoi 26](#_Toc165153035)

[Hình 4‑16 Kết quả bài toán tháp hanoi 27](#_Toc165153036)

# LÝ THUYẾT

## **Kỹ Thuật Quy Hoạch Động**

Phương pháp quy hoạch động là một phương pháp giải quyết các bài toán tối ưu bằng cách chia các bài toán lớn thành các bài toán con nhỏ hơn có cùng cấu trúc để từ đó giải quyết và lưu trữ kết quả của các bài toán con đó, sau đó sử dụng lại kết quả của bài toán con đó để giải quyết bài toán lớn.



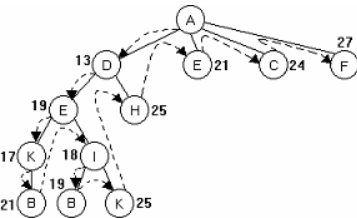
Hình 3‑1 Quy Hoạch Động

### Các kỹ thuật chung cho phương pháp quy hoạch động:

* Để giải một bài toán bằng phương pháp quy hoạch động ta có thể sử dụng các kỹ thuật sau:
* Chia nhỏ bài toán thành các bài toán con:
  + Chia các bài toán lớn phức tạp thành các bài toán con nhỏ hơn và có thể giải được dễ dàng hơn.
  + Xác định các mối quan hệ giữa bài toán con và bài toán lớn ban đầu.
* Lưu trữ kết quả:
  + Để lưu trữ kết quả của các bài toán con đã được giải ta có thể sử dụng một cấu trúc dữ liệu (có thể là: mảng, bảng băm).
  + Lưu trữ kết quả giúp tình trạng phải làm lại các bài toán con đã giải được và giúp tiết kiệm thời gian và bộ nhớ
* Xác định hàm truy hồi:
  + Sử dụng các hàm truy hồi để giải các bài toán con.
  + Các hàm truy hồi nên sử dụng để lưu trữ kết quả của bài toán con để sử dụng cho bài toán lớn hơn.
* Tối ưu hóa thuật toán:
  + Để giảm thiểu thời gian bộ nhớ cách tốt nhất là tối ưu hóa thuật toán.
  + Các kỹ thuật tối ưu hóa phổ biến:
    - Kỹ thuật đổi biến.
    - Kỹ thuật chia để trị.
    - Kỹ thuật lập bảng.
* Phân tích độ phức tạp:
  + Phân tích độ phức tạp Thời gian và bộ nhớ của thuật toán.
  + Đánh giá hiệu quả của thuật toán với các thuật toán khác.
* Một số tính chất nổi bật của bài toán quy hoạch động :
* Bài toán con gối nhau.
* Bài toán có cấu trúc con tối ưu.

## **Kỹ Thuật Nhánh và Cận**:

Phương pháp nhánh cận là một sự cải tiến của thuật toán quay lui dựa trên cơ sở xác định cận trên/dưới cho từng phương án bộ phận. Bằng cách chia nhỏ bài toán thành các nhánh con và loại bỏ các nhánh con không thể dẫn đến nghiệm tối ưu với thuật toán quay lui.



Hình 3‑2 Nhánh và Cận

### Các kỹ thuật chung cho phương pháp Nhánh và Cận:

* Để giải một bài toán bằng phương pháp Nhánh và Cận ta có thể sử dụng các kỹ thuật sau:
* Kỹ thuật đánh giá:
  + Sử dụng một hàm đánh giá để ước tính giá trị tối ưu của các nhánh con.
  + Sử dụng các giá trị từ hàm đánh giá để so sánh các nhánh con và loại bỏ các nhánh con có giá trị đánh giá thấp.
* Kỹ thuật lựa chọn nhánh:
  + Sử dụng các chiến lược để lựa chịn các nhánh con sẽ được khám phá tiếp theo.
  + Các chiến lược lựa chọn nhánh phổ biến:
    - Chiến lược tốt nhất đàu tiên.
    - Chiến lược độ sâu đàu tiên.
    - Chiến lược độ rộng đầu tiên.
* Kỹ thuật cắt nhánh:
  + Sử dụng các quy tắc để loại bỏ các nhánh con không thể dẫn đến nghiệm tối ưu.
  + Các quy tắc có thể dựa trên giá trị đánh giá, ràng buộc của bài toán hoặc thông tin khác.
* Kỹ thuật lưu trữ:
  + Để lưu trữ kết quả của các nhánh con cần khám phá ta có thể sử dụng một cấu trúc dữ liệu (có thể là: cây, danh sách).
  + Phải đảm bảo hiệu quả truy cập và cập nhật.
* Kỹ thuật kết thúc:
  + Tìm các điều kiện để khi nào kết thúc thuật toán.
  + Điều kiện kết thúc có thể dựa trên số lượng nhánh con đã khám phá,…các tiêu chí khác.

## So sánh ưu, nhược điểm:

|  |  |  |
| --- | --- | --- |
| Kỹ Thuật | **Kỹ Thuật Quy Hoạch Động** | **Kỹ Thuật Nhánh và Cận** |
| **Ưu Điểm** | **Hiệu quả**: Có thể giải quyết chính xác nhiều bài toán phức tạp trong thời gian hợp lý.  **Dễ lập trình**: Việc triển khai thuật toán quy hoạch động tương đối đơn giản.  **Áp dụng cho nhiều loại bài toán**: Quy hoạch động có thể được áp dụng cho nhiều loại bài toán tối ưu hóa khác nhau, bao gồm bài toán dãy con tăng dài nhất, bài toán balô, bài toán đường đi ngắn nhất, v.v. | **Giải quyết các bài toán có nhiều biến**: Nhánh cận có thể giải quyết các bài toán có nhiều biến phức tạp mà khó có thể giải quyết bằng các phương pháp khác.  **Tìm ra nghiệm tối ưu**: Nhánh cận có thể đảm bảo tìm ra nghiệm tối ưu cho bài toán.  **Kết hợp với các kỹ thuật khác**: Nhánh cận có thể được kết hợp với các kỹ thuật tối ưu hóa khác để cải thiện hiệu quả. |
| **Nhược Điểm** | **Tốn nhiều bộ nhớ**: Quy hoạch động có thể tốn nhiều bộ nhớ để lưu trữ kết quả của các bài toán con.  **Không hiệu quả cho các bài toán có nhiều trạng thái**: Quy hoạch động có thể không hiệu quả cho các bài toán có nhiều trạng thái, vì cần lưu trữ kết quả cho tất cả các trạng thái.  **Khó áp dụng cho một số loại bài toán**: Việc áp dụng quy hoạch động cho một số loại bài toán có thể khó khăn, đòi hỏi sự hiểu biết sâu sắc về bài toán và các kỹ thuật quy hoạch | **Tốn nhiều thời gian**: Nhánh cận có thể tốn nhiều thời gian để khám phá tất cả các nhánh con.  **Khó cắt nhánh hiệu quả**: Việc phát triển các quy tắc cắt nhánh hiệu quả có thể khó khăn, đặc biệt là cho các bài toán phức tạp.  **Gặp vấn đề với các bài toán có nhiều nghiệm tối ưu**: Nhánh cận có thể gặp vấn đề với các bài toán có nhiều nghiệm tối ưu, vì có thể không tìm thấy tất cả các nghiệm tối ưu. |

Để sử dụng các kỹ thuật nào cho phù hợp với bài toán phụ thuộc vào nhiều yếu tố, bao gồm:

Loại bài toán: Quy hoạch động phù hợp hơn cho các bài toán có cấu trúc rõ ràng và có thể chia nhỏ thành các bài toán con. Nhánh cận có thể phù hợp hơn cho các bài toán có nhiều biến phức tạp và khó có thể chia nhỏ bài toán thành các bài toán con.

Kích thước bài toán: Quy hoạch động có thể hiệu quả hơn cho các bài toán có kích thước nhỏ. Nhánh cận có thể hiệu quả hơn cho các bài toán có kích thước lớn.

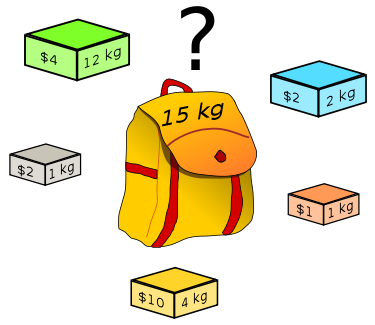
Độ phức tạp bài toán: Quy hoạch động có thể dễ áp dụng hơn cho các bài toán có độ phức tạp thấp. Nhánh cận có thể dễ áp dụng hơn cho các bài toán có độ phức tạp cao.

Yêu cầu về độ chính xác: Nếu độ chính xác cao, nhánh cận có thể làm tốt hơn.

# LẬP TRÌNH (Ngôn ngữ Python)

## Bài toán xếp ba lô 1, 2:

Bài toán ba lô là bài toán tối ưu hóa tổ hợp nhất, trong đó ta cần mang một số vật thể cho vào ba lô, sao cho tổng giá trị là lớn nhất. bài toán xếp ba lô được nguyên cứu từ (Martello and Toth, 1990; Martello et al., 2000; Pisinger et al., 2004) và tới nay nó là một trong những bài toán có ứng dụng thực tế vô cùng lớn.



Hình 4‑1 Bài Toán Ba Lô

Ngoài bài toán ba lô ta còn có các bài toán tương tự có cùng dạng như là:

* Bài toán Knapscak với các giá trị là số thực.
* Bài toán Knapsack cắt nhỏ đồ vật.
* Bài toán ba lô 1 -2.

Trong bài toán này chúng ta sẽ tappj trung nghiên cứu bài toán ba lo 1 -2 và một số ứng dụng của nó trong việc giải quyết các bài toán quy hoạch động.

### Mô Tả Bài Toán:

Bài toán ba lô: Tối ưu hóa giá trị mang theo trong giới hạn trọng lượng.

Mô tả:

Giả sử ta có n đồ vật khác nhau, mỗi đồ vật có trọng lượng w\_i và giá trị v\_i. Ta có một chiếc ba lô có sức chứa tối đa là W. Mục tiêu của ta là chọn ra tập con đồ vật sao cho tổng giá trị của các đồ vật mang theo là lớn nhất mà không vượt quá trọng lượng ba lô cho phép.

Đầu vào:

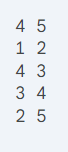
* Dòng đầu tiên: Hai số nguyên dương n và W lần lượt biểu thị số lượng đồ vật và sức chứa tối đa của ba lô.
* n dòng tiếp theo: Mỗi dòng chứa hai số nguyên dương w\_i và v\_i lần lượt biểu thị trọng lượng và giá trị của đồ vật thứ i.

Đầu ra:

* Dòng đầu tiên: In ra giá trị tối đa có thể đạt được.
* Dòng thứ hai: In ra danh sách chỉ số của các đồ vật được chọn theo thứ tự tăng dần.

Ví dụ:

Input:



Output



Giải thích:

Trong ví dụ trên, ta có thể chọn ba lô 3 và 4 với tổng trọng lượng là 3 + 4 = 7 và giá trị là 3 + 4 = 7, đạt được giá trị tối đa 10.

### **Ý tưởng giải quyết bài toán:**

Phân tích bài toán ba lô bằng quy hoạch động 3. Ý tưởng giải quyết Bài toán ba lô có thể được giải quyết hiệu quả bằng thuật toán quy hoạch động. Ý tưởng chính là lưu trữ giá trị tối ưu cho từng trường hợp con của bài toán trong một bảng hai chiều.

* Gọi dp[i][j] là giá trị tối đa có thể đạt được khi chọn các đồ vật từ 1 đến i với giới hạn trọng lượng j. Giá trị cuối cùng dp[n][W] chính là giá trị tối ưu cho toàn bộ bài toán, với n là số lượng đồ vật và W là sức chứa tối đa của ba lô.
* Quy trình giải quyết:
  + Khởi tạo bảng dp:
    - dp[0][j] = 0 cho tất cả j từ 0 đến W. Lý do là vì không có đồ vật nào dẫn đến giá trị khác 0.
  + Duyệt qua các đồ vật từ 1 đến n:
    - Với mỗi đồ vật i, ta xét hai trường hợp:
      * Trường hợp 1: Không chọn đồ vật thứ i: Giá trị tối ưu trong trường hợp này sẽ bằng giá trị tối ưu khi chỉ chọn các đồ vật từ 1 đến i - 1 với cùng giới hạn trọng lượng j: dp[i][j] = dp[i - 1][j].
      * Trường hợp 2: Chọn đồ vật thứ i: Giá trị tối ưu trong trường hợp này sẽ bằng giá trị tối đa giữa hai giá trị: - Giá trị tối ưu khi chỉ chọn các đồ vật từ 1 đến i - 1 với giới hạn trọng lượng còn lại sau khi trừ đi trọng lượng của đồ vật i (j - w\_i): dp[i - 1][j - w\_i]. - Giá trị v\_i của đồ vật i cộng với giá trị tối ưu khi chỉ chọn các đồ vật từ 1 đến i - 1 với giới hạn trọng lượng j: dp[i - 1][j - w\_i] + v\_i. Tuy nhiên, trường hợp 2 chỉ khả thi khi trọng lượng w\_i của đồ vật i nhỏ hơn hoặc bằng giới hạn trọng lượng j. Do đó, ta có công thức quy hoạch động:

dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w\_i] + v\_i) if w\_i <= j

dp[i - 1][j]

* + Sau khi hoàn thành bước 2, ta sẽ có bảng dp chứa giá trị tối ưu cho tất cả các trường hợp con. Giá trị tối ưu cho toàn bộ bài toán là dp[n][W].

**1.Truy vết:**

Để xác định các đồ vật nào được chọn trong phương án tối ưu, ta thực hiện truy vết ngược từ vị trí dp[n][W] trong bảng dp.

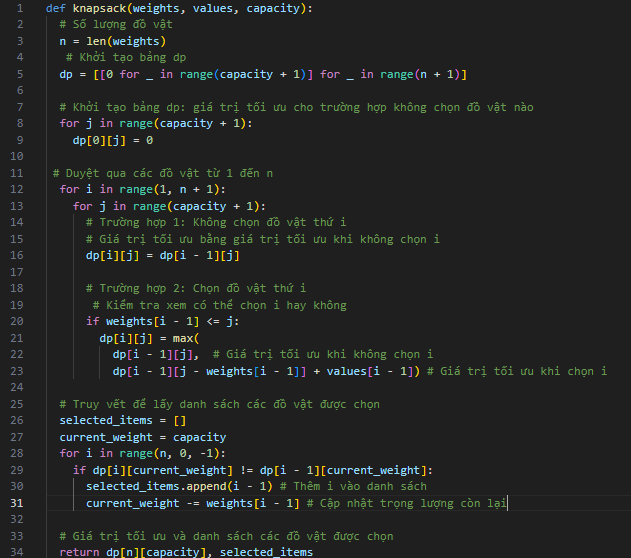
* Bắt đầu từ dp[n][W], ta so sánh giá trị với dp[n - 1][W].
  + Nếu dp[n][W] == dp[n - 1][W], nghĩa là đồ vật thứ n không được chọn. Ta di chuyển đến dp[n - 1][W] và tiếp tục truy vết.
  + Nếu dp[n][W] > dp[n - 1][W], nghĩa là đồ vật thứ n được chọn. Ta di chuyển đến dp[n - 1][W - w\_n] và tiếp tục truy vết.
* Tiếp tục truy vết theo cách này cho đến khi ta đến bảng dp ở hàng 0. Các đồ vật được chọn sẽ là các chỉ số i mà ta di chuyển qua trong quá trình truy vết.

**2.Độ phức tạp tính toán:**

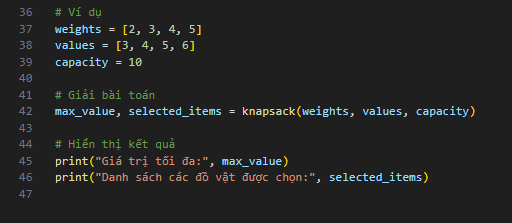
* Thuật toán quy hoạch động cho bài toán ba lô có độ phức tạp thời gian là O(nW), với n là số lượng đồ vật và W là sức chứa tối đa của ba lô.
* Độ phức tạp không gian là O(nW) để lưu trữ bảng dp.

### Cài đặt:

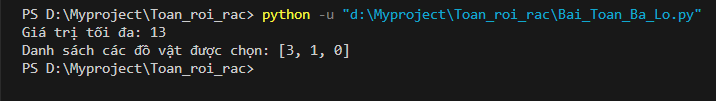
Phần hàm chương trình:

****

Hình 4‑2 Phần chương trình sử lý bài toán ba lô

Phần Ví dụ chương trình:  


Hình 4‑3 Ví dụ của Bài toán ba lô

Kết quả nhận được:  


Hình 4‑4 Kết quả của bài toán ba lô

Hàm knapsack

* **Tham số đầu vào:**
  + weights: Danh sách trọng lượng của các đồ vật (n phần tử).
  + values: Danh sách giá trị của các đồ vật (n phần tử).
  + capacity: Sức chứa tối đa của ba lô.
* **Giá trị trả về:**
  + Tuple:
    - Giá trị tối đa có thể đạt được.
    - Danh sách các đồ vật được chọn (chỉ số).
* **Cấu trúc:**

**1.Khởi tạo biến:**

* n: Số lượng đồ vật (từ danh sách weights).
* dp: Bảng hai chiều (n + 1) x (capacity + 1) để lưu trữ giá trị tối ưu.

**2.Khởi tạo bảng dp:**

* Sử dụng vòng lặp for để khởi tạo giá trị 0 cho tất cả các phần tử trong hàng đầu tiên (dp[0][j] = 0). Lý do: không có đồ vật nào dẫn đến giá trị khác 0.

3. **Duyệt qua các đồ vật**:

* Vòng lặp for i in range(1, n + 1): duyệt qua các đồ vật từ 1 đến n.
* Vòng lặp for j in range(capacity + 1): duyệt qua các khả năng sức chứa còn lại của ba lô từ 0 đến capacity.
* Xét hai trường hợp cho mỗi đồ vật i:

**Trường hợp 1: Không chọn đồ vật thứ i:**

- Giá trị tối ưu trong trường hợp này sẽ bằng giá trị tối ưu khi chỉ chọn các đồ vật từ 1 đến `i - 1` với cùng giới hạn trọng lượng `j`: `dp[i][j] = dp[i - 1][j]`.

- Cập nhật giá trị `dp[i][j]` bằng giá trị đã tính toán.

**Trường hợp 2: Chọn đồ vật thứ i:**

- Giá trị tối ưu trong trường hợp này sẽ bằng giá trị tối đa giữa hai giá trị:

- Giá trị tối ưu khi chỉ chọn các đồ vật từ 1 đến `i - 1` với giới hạn trọng lượng còn lại sau khi trừ đi trọng lượng của đồ vật `i` (`j - weights[i - 1]`).

- Giá trị `values[i - 1]` của đồ vật `i` cộng với giá trị tối ưu khi chỉ chọn các đồ vật từ 1 đến `i - 1` với giới hạn trọng lượng `j`: `dp[i - 1][j - weights[i - 1]] + values[i - 1]`.

- Tuy nhiên, trường hợp 2 chỉ khả thi khi trọng lượng `weights[i - 1]` của đồ vật `i` nhỏ hơn hoặc bằng giới hạn trọng lượng `j`.

- Cập nhật giá trị `dp[i][j]` bằng giá trị tối đa đã tính toán.

**4. Truy vết để lấy danh sách các đồ vật được chọn:**

* Khởi tạo danh sách selected\_items để lưu trữ các chỉ số của các đồ vật được chọn.
* Biến current\_weight lưu trữ trọng lượng còn lại của ba lô sau khi chọn các đồ vật.
* Duyệt ngược từ hàng n đến hàng 0 trong bảng dp:
  + So sánh giá trị dp[i][current\_weight] với dp[i - 1][current\_weight]:
    - Nếu khác nhau, nghĩa là đồ vật thứ i được chọn.
      * Thêm chỉ số i - 1 (vị trí thực tế của đồ vật) vào danh sách selected\_items.
      * Cập nhật current\_weight bằng cách trừ đi trọng lượng weights[i - 1] của đồ vật i.
* Danh sách selected\_items sau khi duyệt xong sẽ chứa các chỉ số của các đồ vật được chọn để đạt được giá trị tối ưu.

**5. Kết quả:**

* Giá trị tối ưu cho toàn bộ bài toán là dp[n][capacity].
* Danh sách các đồ vật được chọn là selected\_items.

## **Bài toán người du lịch - TSP (Travelling Salesman Problem**)

### Mô Tả Bài Toán:

Bài toán người du lịch là một bài toán quen thuộc trong lĩnh vực tối ưu hóa và khoa học máy tính. Bài toán đặt ra câu hỏi: "Một người du lịch cần phải đi qua một số thành phố, mỗi thành phố đúng một lần và quay lại thành phố xuất phát, sao cho tổng khoảng cách đi là ngắn nhất có thể?".

Trong giải quyết bài toán này, thuật toán nhánh và cận là một trong những phương pháp hiệu quả nhất. Ý tưởng cơ bản của thuật toán là liệt kê tất cả các lộ trình có thể có, sau đó loại bỏ các lộ trình không tối ưu bằng cách sử dụng một hàm cận để cắt tỉa những lời giải không triển khai hứa hẹn.

### Ý tưởng giải quyết bài toán:

**Bước 1: Khởi tạo:**

* Khởi tạo cap\_do với một thứ tự ban đầu (ví dụ: cap\_do = [0, 1, 2, 3, 4]).
* Khởi tạo chi\_phi\_nho\_nhat với giá trị rất lớn (ví dụ: chi\_phi\_nho\_nhat = sys.maxsize).
* Khởi tạo duong\_di\_tot\_nhat là mảng rỗng.

**Bước 2:**  **Duyệt theo nhánh:**

* Hàm nhanh\_va\_can được gọi đệ quy với các tham số khởi tạo.
* Hàm này thực hiện duyệt theo nhánh như sau:
  + **Kiểm tra trường hợp cơ sở:**
    - Nếu muc bằng N (số lượng thành phố), nghĩa là một đường đi hoàn chỉnh đã được xây dựng.
    - Tính toán chi\_phi\_hien\_tai cho đường đi này.
    - Nếu chi\_phi\_hien\_tai nhỏ hơn chi\_phi\_nho\_nhat, cập nhật chi\_phi\_nho\_nhat và duong\_di\_tot\_nhat với đường đi mới này.
  + **Duyệt nhánh:**
    - Sử dụng vòng lặp for i in range(muc, N) để duyệt qua tất cả các thành phố chưa được thăm viếng.
    - Đối với mỗi thành phố chưa thăm viếng:
      * Sử dụng hoan\_vi để đổi vị trí thành phố hiện tại (muc) với thành phố chưa thăm viếng (i).
      * Tính toán gioi\_han\_hien\_tai, là ước tính chi phí tối thiểu có thể đạt được cho đường đi dở dang.
      * Nếu gioi\_han\_hien\_tai nhỏ hơn chi\_phi\_nho\_nhat, nghĩa là nhánh này có tiềm năng dẫn đến đường đi tối ưu hơn.
        + Gọi đệ quy nhanh\_va\_can với muc tăng lên 1 để tiếp tục duyệt nhánh con.
        + Sau khi duyệt nhánh con, sử dụng hoan\_vi để đổi vị trí các thành phố trở lại trạng thái ban đầu.
  + Quay lại:
    - Sau khi duyệt qua tất cả các nhánh con, quay lại trạng thái trước đó bằng cách tăng muc lên 1.

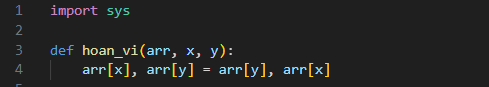
**Bước 3: Kết quả:**

* Sau khi duyệt qua tất cả các nhánh khả thi, chi\_phi\_nho\_nhat sẽ lưu trữ chi phí của đường đi ngắn nhất được tìm thấy, và duong\_di\_tot\_nhat sẽ lưu trữ thứ tự các thành phố trong đường đi tối ưu.

**Bước 4: Phân tích độ phức tạp:**

* Độ phức tạp thời gian của thuật toán Branch and Bound phụ thuộc vào nhiều yếu tố, bao gồm kích thước của ma trận chi phí, hiệu quả của hàm ước tính giới hạn và cách triển khai thuật toán.
* Trong trường hợp xấu nhất, độ phức tạp thời gian có thể là **O(N!)**, với N là số lượng thành phố. Tuy nhiên, trong thực tế, thuật toán thường hoạt động tốt hơn nhiều so với trường hợp xấu nhất.
* Hiệu quả của thuật toán có thể được cải thiện bằng cách sử dụng các kỹ thuật tối ưu hóa khác nhau, chẳng hạn như:
  + Sử dụng các hàm ước tính giới hạn hiệu quả hơn.
  + Loại bỏ các nhánh không hứa hẹn sớm hơn.
  + Áp dụng các kỹ thuật cắt nhánh (branching) khác nhau.

### Cài đặt

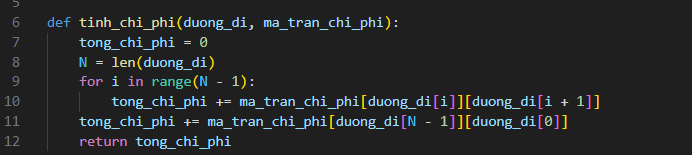


Hình 4‑5 hàm hoán vị bài toán người du lịch

**1. Hàm hoan\_vi:**

Hàm này hoán vị hai phần tử trong một mảng.

* Tham số:
  + arr: Mảng cần hoán vị.
  + x: Chỉ số phần tử thứ nhất.
  + y: Chỉ số phần tử thứ hai.
* Hoạt động:
  + Sử dụng phép gán tạm thời để hoán vị hai phần tử arr[x] và arr[y].

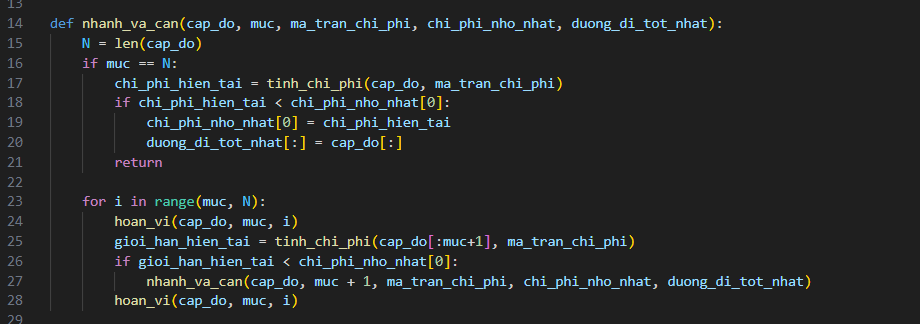


Hình 4‑6 hàm tính chi phí bài toán người du lịch

**2. Hàm tinh\_chi\_phi:**

Hàm này tính toán tổng chi phí cho một đường đi trong ma trận chi phí.

* Tham số:
  + duong\_di: Mảng lưu trữ các đỉnh theo thứ tự đi qua.
  + ma\_tran\_chi\_phi: Ma trận chi phí.
* Hoạt động:
  + Khởi tạo biến tong\_chi\_phi bằng 0.
  + Lặp qua các đỉnh liền kề trong duong\_di, cộng dồn chi phí trên mỗi cạnh vào tong\_chi\_phi.
  + Cộng thêm chi phí của cạnh từ đỉnh cuối cùng về đỉnh đầu tiên vào tong\_chi\_phi.
  + Trả về tong\_chi\_phi.



Hình 4‑7 hàm nhánh và cận bài toán người du lịch

**3. Hàm nhanh\_va\_can:**

Hàm này sử dụng thuật toán quay lui để tìm đường đi ngắn nhất.

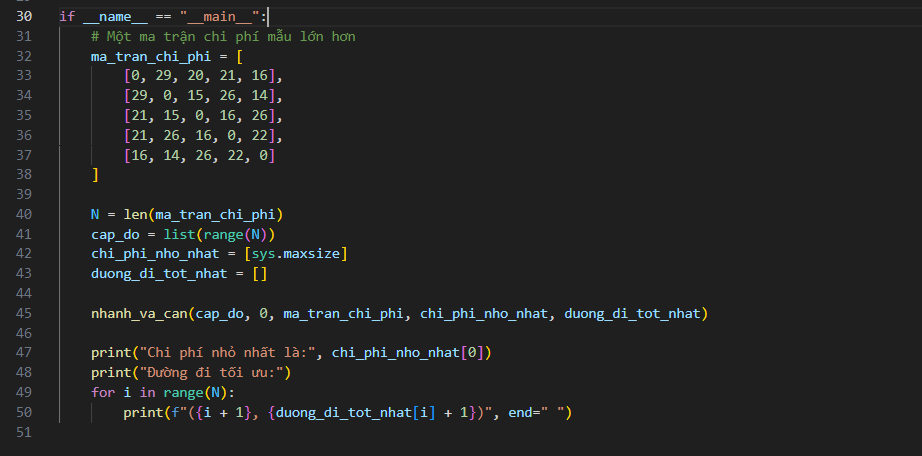
* Tham số:
  + cap\_do: Mảng lưu trữ thứ tự các đỉnh được duyệt.
  + muc: Chỉ số đỉnh hiện tại đang xét.
  + ma\_tran\_chi\_phi: Ma trận chi phí.
  + chi\_phi\_nho\_nhat: Mảng lưu trữ chi phí nhỏ nhất được tìm thấy.
  + duong\_di\_tot\_nhat: Mảng lưu trữ đường đi tối ưu.
* Hoạt động:

**Điều kiện dừng:**

* + - Nếu muc bằng độ dài của cap\_do, nghĩa là đã duyệt qua tất cả các đỉnh.
      * Tính toán chi phí hiện tại cho đường đi.
      * Cập nhật chi\_phi\_nho\_nhat và duong\_di\_tot\_nhat nếu chi phí hiện tại nhỏ hơn chi phí nhỏ nhất được tìm thấy trước đó.

**Quay lui:**

* + - Lặp qua các đỉnh còn lại chưa được duyệt.
      * Hoán vị đỉnh hiện tại (muc) với đỉnh đang xét.
      * Cập nhật muc để xét đỉnh tiếp theo.
      * Tính toán giới hạn chi phí hiện tại cho đường đi.
      * Nếu giới hạn chi phí hiện tại nhỏ hơn chi phí nhỏ nhất được tìm thấy, gọi đệ quy nhanh\_va\_can để tiếp tục duyệt.
      * Hoán vị lại hai đỉnh đã hoán vị trước đó.
* Ý tưởng chính:
  + Thuật toán duyệt qua tất cả các hoán vị khả thi của các đỉnh, tính toán chi phí cho mỗi đường đi và lưu lại đường đi có chi phí nhỏ nhất.

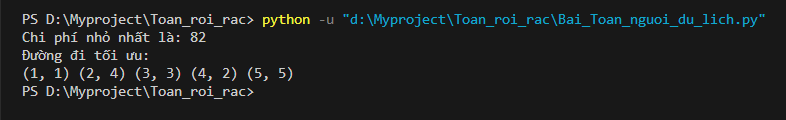


Hình 4‑8 chương trình chính bài toán người du lịch

4. Khối lệnh if \_\_name\_\_ == "\_\_main\_\_"::

Khối lệnh này chứa phần mã chính thực thi khi chương trình được chạy trực tiếp.

* **Khởi tạo ma trận chi phí:**
  + Định nghĩa một ma trận ma\_tran\_chi\_phi mô tả chi phí cho mỗi cặp đỉnh.
* **Thiết lập các biến:**
  + N: Độ dài của ma trận chi phí (số lượng đỉnh).
  + cap\_do: Mảng lưu trữ thứ tự các đỉnh, ban đầu được khởi tạo theo thứ tự từ 0 đến N-1.
  + chi\_phi\_nho\_nhat: Mảng lưu trữ chi phí nhỏ nhất được tìm thấy, khởi tạo bằng giá trị tối đa.
  + duong\_di\_tot\_nhat: Mảng lưu trữ đường đi tối ưu, khởi tạo bằng mảng rỗng.
* **Gọi hàm nhanh\_va\_can:**
  + Gọi hàm nhanh\_va\_can với các tham số:
    - cap\_do: Mảng lưu trữ thứ tự các đỉnh.
    - 0: Chỉ số đỉnh bắt đầu (đỉnh 0).
    - ma\_tran\_chi\_phi: Ma trận chi phí.
    - chi\_phi\_nho\_nhat: Mảng lưu trữ chi phí nhỏ nhất.
* duong\_di\_tot\_nhat: Mảng lưu trữ đường đi tối ưu, khởi tạo bằng mảng rỗng.
* **Hiển thị kết quả:**
* In ra chi phí nhỏ nhất được tìm thấy.
* In ra đường đi tối ưu theo định dạng (đỉnh xuất phát, đỉnh đến tiếp theo).



Hình 4‑9 Kết quả bài toán người du lịch

## Bài toán rút tiền ATM

### Mô Tả Bài Toán:

Giả sử có N loại tiền giấy với các mệnh giá khác nhau, lần lượt là v1, v2, ..., vn (v1 < v2 < v3 < ... < vn). Mục tiêu là xác định cách thức rút tiền tối ưu nhất (sử dụng ít tờ tiền nhất) từ máy ATM cho số tiền cần thanh toán là M.

### Ý tưởng giải quyết bài toán:

Sử dụng thuật toán Quy hoạch động để giải bài toán này. Định nghĩa mảng F[i, j] biểu thị số lượng tờ tiền tối thiểu cần thiết để thanh toán số tiền j sử dụng tối đa i loại tiền giấy đầu tiên (v1, v2, ..., vi).

**Công thức truy hồi:**

Công thức truy hồi cho F[i, j] được thể hiện như sau:

1. **Trường hợp không sử dụng loại tiền thứ i:**

F[i, j] = F[i - 1, j] (1 <= j <= M)

1. **Trường hợp sử dụng loại tiền thứ i:**

F[i, j] = 1 + F[i, j - vi] (1 <= j <= M và j >= vi)

**Giá trị cuối cùng:**

Giá trị F[N, M] đại diện cho số lượng tờ tiền tối thiểu cần thiết để thanh toán số tiền M sử dụng tất cả N loại tiền giấy.

**Cơ sở quy hoạch động:**

* Khởi tạo:
  + F[0, j] = ∞ (1 <= j <= M): Thanh toán số tiền j bằng tiền mệnh giá 0 (v0 = 0) cần vô số tờ tiền.
  + F[i, 0] = 0 (1 <= i <= N): Không có cách thanh toán cho số tiền 0.

**Ví dụ:**

Giả sử có 3 loại tiền giấy với mệnh giá 2, 5 và 7. Cần rút số tiền 11.

Bảng F được khởi tạo:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 |
| 3 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 2 |

* Áp dụng công thức truy hồi để tính toán các giá trị trong bảng F.
* Giá trị cuối cùng F[3, 11] = 2, thể hiện cách thanh toán tối ưu là sử dụng 1 tờ 7 và 1 tờ 4.

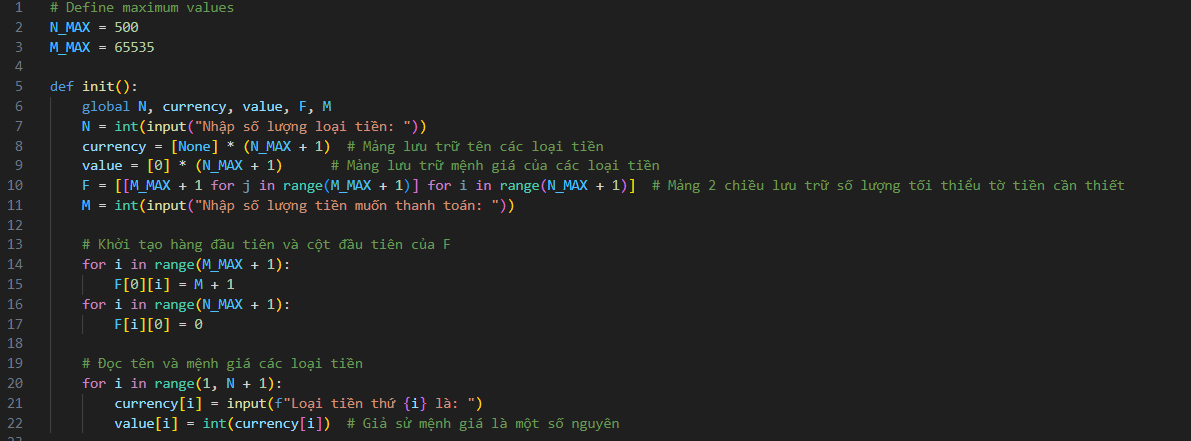
**Kết luận:**

Thuật toán Quy hoạch động cung cấp cách thức hiệu quả để giải bài toán rút tiền ATM tối ưu, giúp xác định số lượng tờ tiền tối thiểu cần thiết cho số tiền thanh toán cụ thể.

### Cài Đặt

**1. Khởi tạo (init function):**

* **Mục đích:** Khởi tạo các biến toàn cục và đọc thông tin đầu vào từ người dùng.
* **Giải thích:**
  + N: Số lượng loại tiền giấy.
  + currency: Mảng lưu trữ tên các loại tiền giấy.
  + value: Mảng lưu trữ mệnh giá của các loại tiền giấy.
  + F: Mảng 2 chiều lưu trữ số lượng tối thiểu tờ tiền cần thiết cho mỗi số tiền (hàng) và mỗi loại tiền giấy (cột).
  + M: Số tiền cần thanh toán.
* **Mã code:**

****

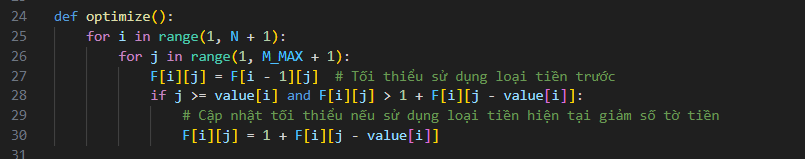
Hình 4‑10 hàm init bài toán ATM

**Giải thích chi tiết:**

* Hàm init() được sử dụng để khởi tạo các biến toàn cục cần thiết cho chương trình.
* Đầu tiên, hàm lấy thông tin đầu vào từ người dùng cho số lượng loại tiền (N) và số tiền cần thanh toán (M).
* Sau đó, hàm khởi tạo các mảng currency, value và F.
  + Mảng currency lưu trữ tên của các loại tiền giấy.
  + Mảng value lưu trữ mệnh giá của các loại tiền giấy.
  + Mảng F là một mảng 2 chiều có kích thước (N + 1) x (M + 1), trong đó:
    - Hàng i (1 <= i <= N) đại diện cho việc sử dụng tối đa i loại tiền giấy đầu tiên.
    - Cột j (0 <= j <= M) đại diện cho việc thanh toán số tiền j.
    - Giá trị F[i][j] đại diện cho số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền j sử dụng tối đa i loại tiền giấy đầu tiên.
* Cuối cùng, hàm khởi tạo các giá trị ban đầu cho các mảng F.
  + Hàng đầu tiên (i = 0) của F được đặt thành M + 1, nghĩa là không thể thanh toán bất kỳ số tiền nào bằng 0 loại tiền giấy.
  + Cột đầu tiên (j = 0) của F được đặt thành 0, nghĩa là thanh toán số tiền 0 bằng bất kỳ số lượng tờ tiền nào.
* Sau khi khởi tạo, các biến toàn cục đã được thiết lập và chương trình sẵn sàng để thực hiện thuật toán Quy hoạch động.

**2. Tối ưu hóa (optimize function):**

* **Mục đích:** Thực hiện thuật toán Quy hoạch động để tìm số lượng tối thiểu tờ tiền cần thiết.
* **Giải thích:**
  + Thuật toán Quy hoạch động được áp dụng để tính toán giá trị của F[i][j] cho tất cả các giá trị i (1 <= i <= N) và j (1 <= j <= M).
  + Quy trình hoạt động như sau:
    1. **Khởi tạo:**
       - Khởi tạo giá trị của F[i][j] cho tất cả i và j bằng M\_MAX + 1, nghĩa là không thể thanh toán bất kỳ số tiền nào bằng bất kỳ loại tiền giấy nào.
    2. **Tính toán:**
       - Áp dụng công thức Quy hoạch động cho từng giá trị i và j:
         * F[i][j] = F[i - 1][j]: Số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền j sử dụng tối đa i - 1 loại tiền giấy đầu tiên.
         * F[i][j] = 1 + F[i][j - value[i]]: Số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền j sử dụng tối đa i loại tiền giấy đầu tiên, bao gồm 1 tờ tiền loại thứ i và số lượng tối thiểu tờ tiền cần thiết cho số tiền còn lại (j - value[i]).
         * Cập nhật F[i][j] bằng giá trị nhỏ nhất trong hai trường hợp trên.
       - Lặp lại bước 2 cho đến khi tính toán được giá trị F[N][M].
* **Mã code:**

****

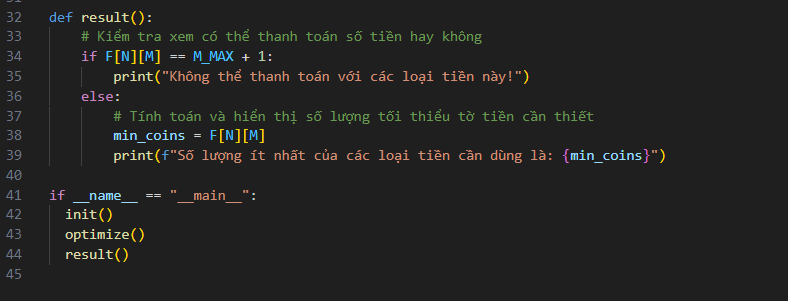
Hình 4‑11 hàm optimize bài toán ATM

**Giải thích chi tiết:**

* Hàm optimize() thực hiện thuật toán Quy hoạch động để tính toán số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền M sử dụng tối đa N loại tiền giấy.
* Hàm sử dụng hai vòng lặp để duyệt qua tất cả các giá trị i (1 <= i <= N) và j (1 <= j <= M).
* Trong mỗi vòng lặp, hàm tính toán giá trị F[i][j] bằng cách so sánh hai trường hợp:
  + **Trường hợp 1:** Không sử dụng loại tiền thứ i.
    - Trong trường hợp này, F[i][j] được đặt bằng F[i - 1][j], nghĩa là số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền j sử dụng tối đa i - 1 loại tiền giấy đầu tiên.
  + **Trường hợp 2:** Sử dụng loại tiền thứ i.
    - Trong trường hợp này, F[i][j] được đặt bằng 1 + F[i][j - value[i]].
    - Giải thích:
      * 1 đại diện cho 1 tờ tiền loại thứ i được sử dụng.
      * F[i][j - value[i]] đại diện cho số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền còn lại (j - value[i]) sử dụng tối đa i loại tiền giấy đầu tiên.
    - Hàm so sánh giá trị F[i][j] tính toán được trong hai trường hợp và chọn giá trị nhỏ nhất để cập nhật cho F[i][j].
* Sau khi thực hiện hai vòng lặp, hàm optimize() đã tính toán được giá trị F[N][M], đại diện cho số lượng tối thiểu tờ tiền cần thiết để thanh toán.

**3. Hiển thị kết quả (result function):**

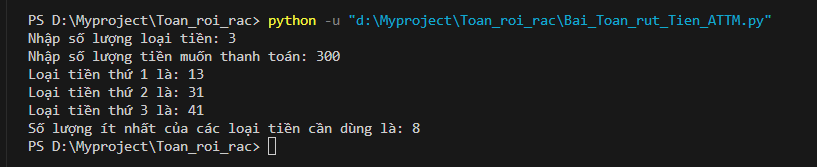
* **Mục đích:** Hiển thị số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền M.
* **Giải thích:**
  + Hàm result() kiểm tra xem liệu có thể thanh toán số tiền M bằng các loại tiền giấy có sẵn hay không.
  + Nếu không thể, hàm thông báo lỗi.
  + Nếu có thể, hàm tính toán và hiển thị số lượng tối thiểu tờ tiền cần thiết.
* **Mã code:**

****

Hình 4‑12 hàm result bài toán ATM

**Giải thích chi tiết:**

* Hàm result() thực hiện các bước sau:
  1. **Kiểm tra khả năng thanh toán:**
     + Hàm kiểm tra giá trị F[N][M].
     + Nếu F[N][M] == M\_MAX + 1, nghĩa là không thể thanh toán số tiền M bằng bất kỳ tổ hợp nào của các loại tiền giấy có sẵn.
     + Hàm thông báo lỗi và kết thúc.
  2. **Tính toán số lượng tối thiểu tờ tiền:**
     + Nếu có thể thanh toán, hàm lấy giá trị F[N][M].
     + Giá trị này đại diện cho số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền M sử dụng tối đa N loại tiền giấy.
  3. **Hiển thị kết quả:**
     + Hàm hiển thị thông báo cho người dùng biết số lượng tối thiểu tờ tiền cần thiết để thanh toán số tiền M.



Hình 4‑13 Kết quả bài toán ATM

## Bài toán Tháp Hà Nội

### Mô Tả Bài Toán:

Bài toán Tháp Hà Nội là một bài toán kinh điển trong khoa học máy tính, được sử dụng để minh họa cho các khái niệm như đệ quy, lập trình, và giải quyết vấn đề. Bài toán có thể được mô tả như sau:

* Cho n đĩa có kích thước khác nhau.
* Có ba cọc: A, B, và C.
* Ban đầu, tất cả các đĩa được xếp chồng lên nhau trên cọc A, theo thứ tự kích thước giảm dần, đĩa nhỏ nhất ở trên cùng.
* Mục tiêu là di chuyển tất cả các đĩa sang cọc C, cũng theo thứ tự kích thước giảm dần, vẫn giữ nguyên thứ tự tương đối giữa các đĩa.
* Điều kiện:
  + Chỉ được di chuyển một đĩa tại một thời điểm.
  + Một đĩa không thể được đặt lên trên một đĩa có kích thước nhỏ hơn.
  + Có thể sử dụng cọc B làm cọc trung gian.

### Ý tưởng giải quyết bài toán:

Bài toán Tháp Hà Nội có thể được giải quyết hiệu quả bằng kỹ thuật quy hoạch động. Kỹ thuật này dựa trên ý tưởng lưu trữ kết quả của các bài toán con để tránh tính toán trùng lặp.

**Giải pháp:**

Để giải bài toán Tháp Hà Nội bằng quy hoạch động, ta có thể sử dụng bảng hai chiều h(n, c), nơi:

* n là số lượng đĩa.
* c là cọc đích (A, B hoặc C).

Mỗi phần tử h(n, c) trong bảng biểu thị số lượng bước tối thiểu cần thiết để di chuyển n đĩa từ cọc A sang cọc c.

**Công thức:**

* h(1, c) = 1 (di chuyển 1 đĩa cần 1 bước).
* h(n, A) = h(n - 1, C) + 1 (di chuyển n đĩa sang A bằng cách di chuyển n - 1 đĩa sang C, sau đó di chuyển 1 đĩa sang A).
* h(n, B) = h(n - 1, A) + 1 (di chuyển n đĩa sang B bằng cách di chuyển n - 1 đĩa sang A, sau đó di chuyển 1 đĩa sang B).
* h(n, C) = h(n - 1, B) + 1 (di chuyển n đĩa sang C bằng cách di chuyển n - 1 đĩa sang B, sau đó di chuyển 1 đĩa sang C).

**Ví dụ:**

Giả sử ta muốn di chuyển 3 đĩa từ cọc A sang cọc C. Bảng h(n, c) sẽ được cập nhật như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| n | A | B | C |
| 1 | 1 | 2 | 3 |
| 2 | 3 | 4 | 5 |
| 3 | 5 | 7 | 9 |

Theo bảng, số lượng bước tối thiểu cần thiết để di chuyển 3 đĩa từ cọc A sang cọc C là 9.

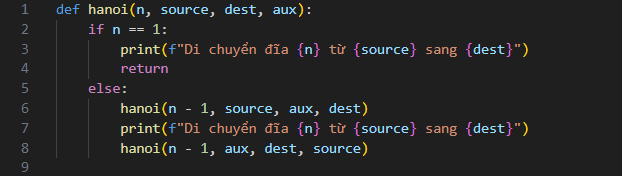
**Phân tích độ phức tạp:**

Độ phức tạp thời gian của giải pháp quy hoạch động cho bài toán Tháp Hà Nội là O(2^n), tương đương với độ phức tạp của giải pháp đệ quy. Tuy nhiên, giải pháp quy hoạch động sử dụng ít bộ nhớ hơn so với giải pháp đệ quy.

**Ứng dụng:**

Bài toán Tháp Hà Nội có thể được áp dụng để giải quyết các bài toán khác có cấu trúc tương tự, chẳng hạn như bài toán sắp xếp các phần tử trong một danh sách theo thứ tự nhất định.

### Cài đặt

****

Hình 4‑14 hàm hanoi bài toán tháp hanoi

**1. Hàm hanoi(n, source, dest, aux)**

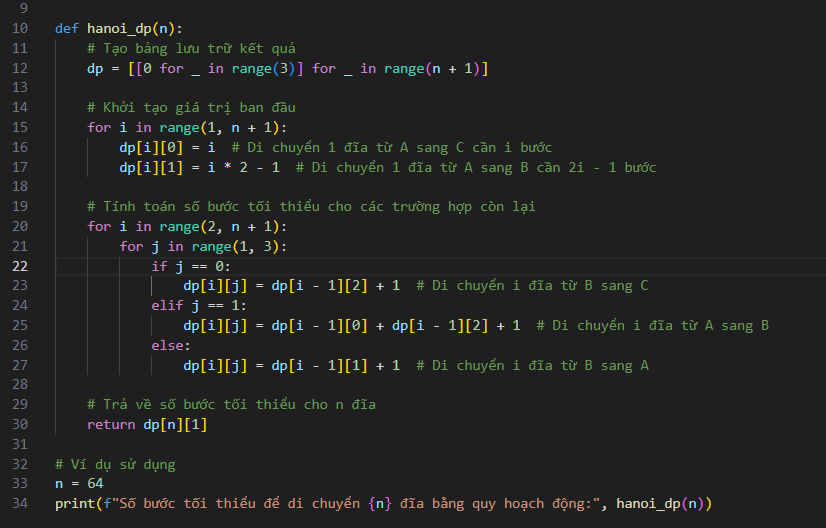
Hàm này sử dụng đệ quy để giải bài toán Tháp Hà Nội.

**Tham số:**

* n: Số lượng đĩa cần di chuyển.
* source: Cọc nguồn (A, B hoặc C).
* dest: Cọc đích (A, B hoặc C).
* aux: Cọc phụ trợ (A, B hoặc C).

**Hoạt động:**

1. **Kiểm tra trường hợp cơ bản:**
   * Nếu n == 1, di chuyển đĩa duy nhất từ cọc nguồn sang cọc đích và in ra thông báo.
   * Kết thúc hàm.
2. **Gọi đệ quy:**
   * Gọi hàm hanoi để di chuyển n - 1 đĩa từ cọc nguồn sang cọc phụ trợ, sử dụng cọc đích làm cọc tạm thời.
3. **Di chuyển đĩa lớn nhất:**
   * Di chuyển đĩa lớn nhất (n) từ cọc nguồn sang cọc đích và in ra thông báo.
4. **Gọi đệ quy:**
   * Gọi hàm hanoi để di chuyển n - 1 đĩa từ cọc phụ trợ sang cọc đích, sử dụng cọc nguồn làm cọc tạm thời.

****

Hình 4‑15 hàm hanoi\_dp bài toán tháp hanoi

**2.Hàm hanoi\_dp(n)**

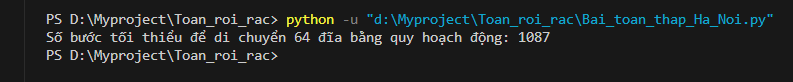
Hàm này sử dụng quy hoạch động để tính toán số bước tối thiểu cần thiết để di chuyển n đĩa.

**Tham số:**

* n: Số lượng đĩa cần di chuyển.

**Hoạt động:**

1. **Tạo bảng dp:**
   * Tạo bảng hai chiều dp có kích thước (n + 1) x 3.
   * Mỗi phần tử dp[i][j] lưu trữ số bước tối thiểu cần thiết để di chuyển i đĩa từ cọc A sang cọc j (j = 0: C, j = 1: B, j = 2: A).
2. **Khởi tạo giá trị ban đầu:**
   * dp[i][0] = i: Di chuyển i đĩa từ A sang C cần i bước.
   * dp[i][1] = i \* 2 - 1: Di chuyển i đĩa từ A sang B cần 2i - 1 bước.
3. **Tính toán số bước tối thiểu cho các trường hợp còn lại:**
   * Duyệt qua các trường hợp i từ 2 đến n.
   * Duyệt qua các cọc đích j từ 1 đến 2.
   * Tính toán số bước tối thiểu cho trường hợp hiện tại dp[i][j] bằng cách sử dụng các giá trị đã được tính toán trước đó trong bảng dp.
4. **Trả về kết quả:**
   * Trả về giá trị dp[n][1], số bước tối thiểu cần thiết để di chuyển n đĩa từ A sang B.

****

Hình 4‑16 Kết quả bài toán tháp hanoi

## IV. Link Github:

Xem toàn bộ code tại Github: zinxus/toan-roi\_rac