



Đề tài

ÁP DỤNG GIẢI THUẬT GREEDY CHO BÀI TOÁN CẢI TÚI

Môn học: TRÍ TUỆ NHÂN TẠO

GVHD: TS. Đặng Ngọc Hoàng Thành

Nhóm sinh viên thực hiện

Họ tên

MSSV

Đỗ Ngọc Phương Anh

31221020325

Nguyễn Ngọc Thúy Anh

31221020005

Nguyễn Thùy Yến Nhi

33231020124

Huỳnh Minh Phương

31221020502

MỤC LỤC

DANH MỤC HÌNH ẢNH.....	1
LỜI CẢM ƠN.....	1
CHƯƠNG 1. TỔNG QUAN VỀ BÀI TOÁN KNAPSACK PROBLEM (BÀI TOÁN CẢI TÚI).....	1
1.1. Giới thiệu về bài toán cái túi (Knapsack problem).....	1
1.2. Phát biểu bài toán.....	1
1.3. Một số hướng tiếp cận giải quyết bài toán.....	2
CHƯƠNG 2: GIẢI THUẬT GREEDY ALGORITHM.....	4
2.1. Giới thiệu về giải thuật Greedy Algorithm.....	4
2.2. Độ phức tạp của giải thuật.....	4
2.3. Ứng dụng của giải thuật cho bài toán Knapsack problem.....	5
CHƯƠNG 3: THIẾT KẾ GIAO DIỆN.....	8
3.1. Giao diện màn hình.....	8
3.2. Chi tiết các chức năng.....	8
CHƯƠNG 4: CÁC KẾT QUẢ THỰC NGHIỆM.....	12
4.1. Các tình huống.....	12
4.2. Phân tích và đánh giá.....	15
CHƯƠNG 5: KẾT LUẬN.....	17
5.1 Các kết quả đạt được.....	17
5.2. Những hạn chế và hướng phát triển.....	17
TÀI LIỆU THAM KHẢO.....	17
PHỤ LỤC.....	17

DANH MỤC HÌNH ẢNH

LỜI CẢM ƠN

Lời đầu tiên, nhóm chúng em xin trân trọng cảm ơn thầy Đặng Ngọc Hoàng Thành - người đã trực tiếp giảng dạy, hỗ trợ nhóm chúng em tiếp cận với kiến thức môn Trí tuệ nhân tạo, để nhóm có thể hoàn thành bài tiểu luận này.

Mặc dù đã rất nỗ lực, song do nhóm còn nhiều hạn chế về kiến thức nên khó tránh khỏi có những thiếu sót trong bài làm. Nhóm kính mong nhận được sự đóng góp ý kiến của thầy để có thể rút ra kinh nghiệm cho tương lai.

Nhóm chúng em xin trân trọng cảm ơn!

CHƯƠNG 1. TỔNG QUAN VỀ BÀI TOÁN KNAPSACK PROBLEM (BÀI TOÁN CẢI TÚI)

1.1. Giới thiệu về bài toán cái túi (Knapsack problem)

Bài toán cái túi, hay Knapsack Problem, là một bài toán tối ưu hóa tổ hợp có nguồn gốc từ đầu thế kỷ 19. Do tính ứng dụng cao trong nhiều lĩnh vực như quản lý dự án, cắt ghép và tối ưu hóa mạng lưới, nó đã thu hút sự chú ý của giới khoa học máy tính và toán học vào thế kỷ 20.

Bài toán cái túi là một bài toán quan trọng và kinh điển, có ứng dụng đa dạng trong thực tế. Nó có thể được áp dụng để tối ưu hóa việc chọn lựa các vật phẩm để đặt vào một cái túi có giới hạn về trọng lượng. Nhờ tính ứng dụng rộng rãi, bài toán được sử dụng để tối ưu hóa nhiều vấn đề thực tế như tối ưu hóa mạng lưới, sử dụng nguyên liệu trong sản xuất, lập kế hoạch dự án và nhiều vấn đề khác.

Tuy nhiên, bài toán cái túi thuộc loại NP-khó, nghĩa là không có thuật toán chính xác nào có thể giải quyết nhanh chóng cho tất cả các trường hợp. Do đó, các nhà nghiên cứu đã đề xuất các phương pháp giải gần đúng như thuật toán tham lam, quy hoạch động và lập trình nhị phân. Mặc dù không đảm bảo tìm ra giải pháp tối ưu, nhưng những phương pháp này cung cấp giải pháp gần đúng đáp ứng được yêu cầu thực tế.

Dù còn nhiều thách thức trong việc tìm giải tối ưu cho mọi trường hợp, bài toán cái túi vẫn đóng vai trò quan trọng trong việc tối ưu hóa quy trình và giảm thiểu chi phí. Giải pháp tối ưu cho bài toán này có thể mang lại hiệu quả, tiết kiệm thời gian và tăng cường sự cạnh tranh trong nhiều ngành công nghiệp và dịch vụ.

1.2. Phát biểu bài toán

● Bài toán:

Cho trước n vật, mỗi vật với cân nặng w_i , giá trị x_i và sức chứa tối đa của cái túi là W , mục tiêu của bài toán là tìm ra một tập hợp các vật để tối đa hoá giá trị của các vật phẩm trong cái túi sao cho các vật phẩm đã chọn không được vượt quá sức chứa tối đa của cái túi.

Ta phát biểu bài toán như sau:

- n là số vật phẩm

- v_i là giá trị của vật phẩm thứ i ($1 \leq i \leq n$)
- w_i là trọng lượng của vật phẩm thứ i ($1 \leq i \leq n$)
- W là sức chứa tối đa của cái túi
- x_i là biến nhị phân cho vật phẩm thứ i :
 - $x_i = 1$ nếu vật phẩm thứ i được chọn.
 - $x_i = 0$ nếu vật phẩm thứ i không được chọn.

Khi đó, bài toán Knapsack có thể được biểu diễn dưới dạng một bài toán quy hoạch tuyến tính như sau:

$$\max \sum_{i=1}^n v_i x_i$$

$$\text{với } \sum_{i=1}^n w_i x_i \leq W \text{ và } x_i \in \{0,1\}, 1 \leq i \leq n$$

- **Tóm tắt yêu cầu**
 - Túi có sức chứa tối đa W kg.
 - Nhiều vật phẩm với trọng lượng w_i kg và giá trị v_i đồng tiền mỗi vật.
- Chọn số lượng tối đa cho mỗi vật phẩm để mang vào cái túi sao cho đảm bảo tổng trọng lượng không vượt quá W kg. Tìm cách để tổng giá trị mang theo là lớn nhất.

1.3. Một số hướng tiếp cận giải quyết bài toán

Có nhiều thuật toán khác nhau để giải quyết bài toán cái túi, bao gồm:

- **Thuật toán trực tiếp (Brute-force):** Thuật toán này duyệt tất cả các khả năng các vật có thể để vào túi, tìm ra những khả năng có tổng giá trị lớn nhất mà không vượt qua sức chứa tối đa của cái túi. Đây là một phương pháp đơn giản có thể đảm bảo tìm được tối ưu toàn cục nhưng chỉ có thể áp dụng cho những bài toán có số lượng đồ vật nhỏ.

Các bước thực hiện của thuật toán trực tiếp:

- Bước 1: Duyệt qua tất cả các tập con của tập các đồ vật
- Bước 2: Tính tổng giá trị của từng tập con

- Bước 3: Chọn tập con có tổng giá trị lớn nhất nhưng không vượt quá trọng lượng của túi.
- **Thuật toán tham lam (Greedy Algorithm):** Thuật toán này lựa chọn vật phẩm có giá trị cao nhất trên mỗi đơn vị trọng lượng cho đến khi cái túi đầy. Thuật toán tham lam là một phương pháp đơn giản và hiệu quả để giải bài toán cái túi, đặc biệt là cho các trường hợp đơn giản. Tuy nhiên, cần lưu ý rằng thuật toán này nhanh nhưng không đảm bảo tìm ra giải pháp tối ưu trong tất cả các trường hợp.

Các bước thực hiện của thuật toán tham lam:

- Bước 1: Sắp xếp các đồ vật theo thứ tự giảm dần về giá trị hoặc tăng dần về trọng lượng.
- Bước 2: Chọn đồ vật có giá trị cao nhất (hoặc trọng lượng thấp nhất) và thêm vào túi.
- Bước 3: Tiếp tục chọn đồ vật thứ hai, thứ ba, cho đến khi đạt giới hạn trọng lượng của túi hoặc hết các đồ vật.
- **Quy hoạch động (Dynamic Programming):** Thuật toán này tận dụng các kết quả trung gian đã được tính toán trước đó để giảm thiểu số lần tính toán lặp lại và cải thiện hiệu suất giải thuật. Quy hoạch động là một phương pháp mạnh mẽ để giải bài toán cái túi, đặc biệt là khi cần đảm bảo giải pháp tối ưu hoặc khi phải giải quyết các bài toán cái túi phức tạp. Tuy nhiên, cần cân nhắc độ phức tạp, thời gian thực hiện và tài nguyên bộ nhớ khi lựa chọn phương pháp giải phù hợp. Các bước thực hiện phương pháp Quy hoạch động (Dynamic Programming):

Các bước thực hiện của thuật toán:

- Bước 1: Sắp xếp các đồ vật theo tỷ lệ giá trị/trọng lượng giảm dần.
- Bước 2: Tạo bảng quy hoạch động (DP table): Sử dụng một mảng 2 chiều để lưu trữ kết quả của các bài toán con. Mảng này có kích thước $(n+1) \times (W+1)$, trong đó n là số lượng đồ vật và W là trọng lượng tối đa của túi.
- Bước 3: Duyệt qua từng đồ vật và tính giá trị tối ưu cho từng trọng lượng từ 0 đến W .
 - Nếu trọng lượng đồ vật thứ i nhỏ hơn hoặc bằng trọng lượng hiện tại, ta cân nhắc chọn đồ vật này vào túi. So sánh giá trị tối ưu khi chọn và không chọn đồ vật thứ i , lưu vào mảng DP.

- Nếu trọng lượng đồ vật thứ i lớn hơn trọng lượng hiện tại, ta không chọn đồ vật này.
- Bước 4: Kết quả của bài toán cái túi là giá trị tối ưu tại ô (n, W) trong DP table.

Đây là ba cách phổ biến để giải quyết bài toán cái túi. Ngoài ra hiện nay có rất nhiều hướng tiếp cận khác nhau như thuật toán chia để trị (Divide and Conquer), thuật toán nhánh và cận (Branch and Bound), thuật toán quay lui (Backtracking)... Trong thực tế, không có thuật toán nào được coi là tối ưu nhất hay được sử dụng nhiều nhất cả. Các cách tiếp cận khác nhau này thường được kết hợp với nhau để tạo ra các giải pháp tốt nhất cho từng trường hợp cụ thể.

CHƯƠNG 2: THUẬT TOÁN THAM LAM (GREEDY ALGORITHM)

2.1. Giới thiệu về thuật toán Greedy

Thuật toán Greedy hay còn được gọi là thuật toán tham lam là một thuật toán giải quyết bài toán theo kiểu metaheuristic để tìm kiếm lựa chọn tối ưu địa phương ở mỗi bước đi với hy vọng tìm được tối ưu toàn cục. Nói một cách dễ hiểu hơn, thuật toán tham lam lựa chọn giải pháp được cho là tốt nhất ở thời điểm hiện tại, và sau đó tiếp tục giải bài toán con nảy sinh từ việc thực hiện lựa chọn lúc này. Lựa chọn của thuật toán tham lam có thể phụ thuộc vào các lựa chọn trước đó, nhưng nó không thể phụ thuộc vào các lựa chọn trong tương lai hay lời giải của các bài toán con. Thuật toán tiến triển theo hướng thực hiện các lựa chọn theo một vòng lặp, cùng lúc đó, thu nhỏ bài toán đã cho thành một bài toán nhỏ hơn.

Thuật toán tham lam có ưu điểm là có thể đưa ra giải pháp tốt trong thời gian ngắn và dễ dàng để triển khai trong thực tế. Tuy nhiên, với một số bài toán, đây có thể là một thuật toán không chính xác vì thuật toán tham lam là một kỹ thuật heuristic, nó có thể bám chặt lấy một số lựa chọn nhất định một cách quá sớm, dẫn đến hậu quả trong giai đoạn sau, vì thế có thể không tìm ra được giải pháp tốt nhất, các kết quả thu được cũng sẽ có sự khác nhau tùy thuộc vào thứ tự của các phần tử trong dữ liệu đầu vào.

2.2. Độ phức tạp của thuật toán

Độ phức tạp của thuật toán Greedy Algorithm phụ thuộc vào từng thuật toán cụ thể và cấu trúc dữ liệu được sử dụng. Nhìn chung, độ phức tạp thời gian của thuật toán Greedy thường là $O(n \log n)$ hoặc $O(n^2)$, với n là số lượng phần tử trong dữ liệu đầu vào.

Bên cạnh đó, một số thuật toán như thuật toán Greedy sorting, sử dụng cây nhị phân để lưu trữ dữ liệu, hay Greedy coloring, sử dụng mảng để lưu trữ màu của mỗi đỉnh, có độ phức tạp không gian là $O(n)$, nghĩa là chúng cần một lượng không gian lưu trữ tỷ lệ thuận với số lượng phần tử trong dữ liệu đầu vào.

Trên thực tế, độ phức tạp về thời gian và không gian của thuật toán tham lam chỉ là ước tính trung bình, tùy thuộc vào cấu trúc dữ liệu và cách thức triển khai thuật toán, độ phức tạp có thể thay đổi, một số có thể có độ phức tạp thời gian và không gian tốt hơn, một số khác sẽ có độ phức tạp thời gian và không gian xấu hơn ước tính trung bình kể trên.

2.3. Ứng dụng của giải thuật cho bài toán Knapsack problem

Áp dụng thuật toán Greedy cho bài toán Fractional Knapsack, bài toán được giải quyết bằng cách xét các vật theo thứ tự đơn giá giảm dần (đơn giá = giá trị / trọng lượng), với mỗi đồ vật được xét sẽ lấy trọng lượng tối đa mà sức chứa còn lại của túi có thể chứa, lặp lại quá trình này cho đến khi không thể chọn bất kỳ đồ vật nào để bỏ vào túi nữa.

- Input: Trọng lượng tối đa mà túi có thể chứa, trọng lượng và giá trị mỗi đồ vật có thể chọn.
- Output: Danh sách các vật được chọn, khối lượng và giá trị của các vật được chọn, tổng giá trị tối đa đạt được, tổng trọng lượng của túi sau khi bỏ các đồ vật vào.
- Ý tưởng: Ưu tiên lựa chọn các đồ vật có đơn giá từ cao nhất đến thấp nhất cho vào túi cho đến khi túi đầy.

Để giải quyết bài toán, tạo class Knapsack bao gồm các hàm sau:

- Hàm `__init__(self, max_capacity)` dùng để khởi tạo các đối tượng của bài toán, tạo danh sách các đồ vật đầu vào và danh sách các đồ vật đầu ra (các vật được lựa chọn).

```
class Knapsack:
    def __init__(self, max_capacity):
        self.max_capacity = max_capacity
        self.items = [] # List of (weight, value) tuples
        self.output_items = [] # List to store selected items
```

Cụ thể, hàm này nhận tham số đầu vào là đối tượng hiện tại của class (self) và trọng lượng tối đa của túi (max_capacity).

Trong hàm này, giá trị của tham số đầu vào max_capacity được gán cho thuộc tính max_capacity của đối tượng trong class, khi tạo bất kỳ đối tượng nào từ class này, nó sẽ mang thuộc tính max_capacity có giá trị bằng với giá trị mà hàm được cung cấp khi khởi tạo đối tượng. Thuộc tính items được khởi tạo dưới dạng danh sách rỗng, dùng để lưu trữ các cặp giá trị (weight, value) đại diện cho từng đồ vật dưới dạng các tuples. Thuộc tính output_items được khởi tạo dưới dạng danh sách rỗng, dùng để lưu trữ các đồ vật được chọn từ danh sách items cho danh sách đầu ra.

- Hàm `knapsack_greedy(self)` là hàm thực hiện thuật toán tham lam (Greedy Algorithm) để giải quyết bài toán Fractional Knapsack.

```

def knapsack_greedy(self):
    # Filter out items with zero weight or zero value
    valid_items = [(weight, value) for weight, value in self.items if weight > 0 and value > 0]

    # Calculate value-to-weight ratio for each item
    items_with_ratio = [(value / weight, weight, value) for weight, value in valid_items]
    items_with_ratio.sort(reverse=True, key=lambda x: x[0]) # Sort by ratio in descending order

    total_value = 0
    total_weight = 0
    remaining_capacity = self.max_capacity
    for _, weight, value in items_with_ratio:
        if weight <= remaining_capacity:
            total_value += value
            total_weight += weight
            remaining_capacity -= weight
            self.output_items.append((weight, value)) # Add selected item
            if remaining_capacity == 0:
                break
        else:
            # Fractional part of the last item
            fraction = remaining_capacity / weight
            fractional_value = fraction * value
            total_weight += remaining_capacity
            total_value += fractional_value
            self.output_items.append((remaining_capacity, fractional_value)) # Add fractional item
            break

    return total_value, total_weight

```

Cụ thể, hàm này nhận tham số đầu vào là các đối tượng hiện tại của class (self), tham chiếu đến đối tượng hiện tại của class.

Trong hàm này, đầu tiên khởi tạo một danh sách mới là valid_items để lưu trữ các cặp giá trị hợp lệ trong kết quả trả về. Vòng lặp for lặp qua từng cặp giá trị (weight, value) trong danh sách items của đối tượng hiện tại (self), mỗi lần lặp, biến weight và value sẽ được gán giá trị tương ứng của cặp giá trị hiện tại, sử dụng hàm if để kiểm tra xem cả hai giá trị weight và value trong cặp giá trị có bằng 0 hay không. Mục đích của danh sách này là để chọn lọc và lưu trữ các cặp giá trị (weight, value) lớn hơn 0, tức là khi hiển thị kết quả, các cặp giá trị (0.0, 0.0) sẽ bị loại bỏ.

Tiếp theo sau đó, thuộc tính items_with_ratio được khởi tạo dưới dạng danh sách chứa các tuples, mỗi tuples bao gồm đơn giá (weight / value), trọng lượng (weight) và giá trị (value) của từng đồ vật trong danh sách items đã khởi tạo trước đó. Sau đó, tiến hành sắp xếp (sort) danh sách items_with_ratio theo thứ tự đơn giá (weight / value) giảm dần. Tham số reverse=True chỉ định danh sách sắp xếp theo thứ tự giảm dần. Tham số key chỉ định hàm được truyền vào để hàm sort sắp xếp các phần tử trong danh sách dựa trên kết quả trả về của hàm đó. Trong trường hợp này, với mong muốn sắp xếp thứ tự các phần tử trong danh sách items_with_ratio,

ta truyền vào key hàm lambda x: x[0], nó nhận phần tử x là x[0] và trả về phần tử đầu tiên của danh sách, tức là đơn giá (weight / value).

Khởi tạo biến `total_value = 0` là biến lưu trữ tổng giá trị các đồ vật được chọn, biến `total_weight = 0` là biến lưu trữ tổng trọng lượng các đồ vật đã được thêm vào túi, biến `remaining_capacity` để lưu trữ trọng lượng còn lại của túi sau khi thêm từng đồ vật vào, biến này được khởi tạo với giá trị ban đầu là `self.max_capacity`, tức là ban đầu sức chứa còn lại bằng sức chứa tối đa của túi.

Vòng lặp `for` duyệt qua từng đồ vật trong danh sách `items_with_ratio`. Các giá trị mà vòng lặp `for` sẽ duyệt qua là trọng lượng (weight) và giá trị của đồ vật (value). Ký hiệu `_` được sử dụng như một biến, đây là phần tử đầu tiên của danh sách, đại diện cho đơn giá (weight / value). Việc sử dụng ký hiệu `_` cho biết rằng ta đang bỏ qua phần tử đầu tiên của danh sách.

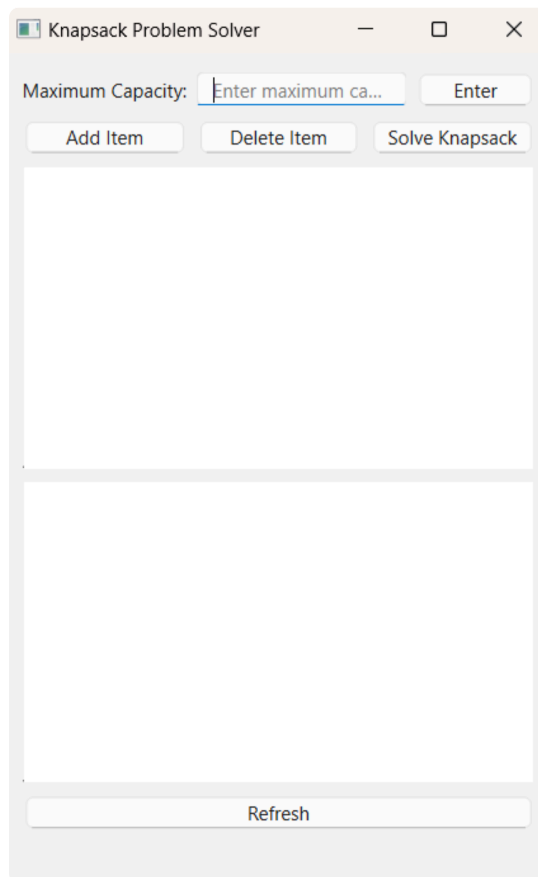
Trong vòng lặp `for`, sử dụng hàm `if` với điều kiện nếu trọng lượng của đồ vật nhỏ hơn hoặc bằng sức chứa còn lại của túi, vật phẩm sẽ được thêm vào túi. Tiếp theo, biến `total_value` sẽ được cộng thêm giá trị bằng với giá trị của vật phẩm vừa thêm vào, sau đó biến `total_weight` sẽ cộng thêm giá trị bằng với trọng lượng của đồ vật vừa thêm vào túi, biến `remaining_capacity` sẽ trừ đi khối lượng của đồ vật vừa thêm vào túi, cuối cùng là thêm đồ vật đó vào danh sách `output_items`. Nếu sức chứa còn lại của túi bằng 0, thoát vòng lặp. Nếu trọng lượng của đồ vật lớn hơn sức chứa còn lại của túi và sức chứa của túi lớn hơn không, chuyển sang thực hiện khối lệnh trong hàm `else`, một phần của đồ vật sẽ được thêm vào túi sao cho có thể lấp đầy sức chứa còn lại của túi. Khởi tạo biến `fraction = remaining_capacity / weight` để tính phần tỷ lệ trọng lượng được thêm vào túi của đồ vật cuối cùng được xét và biến `fractional_value = fraction * value` phần tỷ lệ giá trị của phần trọng lượng này của vật đó. Tiếp theo, cộng giá trị `remaining_capacity` vào biến `total_weight`, cộng tỷ lệ `fractional_value` vào biến `total_value`, thêm biến `remaining_capacity` và biến tỷ lệ `fractional_value` của đồ vật này vào danh sách `output_items` và cuối cùng là thoát vòng lặp vì túi đã đầy.

Sau cùng, hàm `knapsack_greedy` sẽ trả về tổng giá trị (`total_value`) và tổng trọng lượng (`total_weight`) của các đồ vật trong danh sách `output_items`.

CHƯƠNG 3: THIẾT KẾ GIAO DIỆN

3.1. Giao diện màn hình

Giao diện màn hình đơn giản cho phép người dùng nhập thông tin về một bài toán cái túi và chạy thuật toán tham lam để giải quyết vấn đề. Người dùng có thể nhập trọng lượng tối đa của túi, tùy chỉnh thêm và xóa các vật phẩm có các giá trị và trọng lượng tương ứng. Sau khi nhấn nút “Solve Knapsack”, thuật toán tham lam sẽ được thực hiện và kết quả, bao gồm giá trị tối đa, tổng trọng lượng và các vật phẩm được chọn sẽ được hiển thị lên màn hình.

The image shows a web application window titled "Knapsack Problem Solver". It features a text input field labeled "Maximum Capacity:" with a placeholder "Enter maximum ca...". To the right of the input is an "Enter" button. Below the input field are three buttons: "Add Item", "Delete Item", and "Solve Knapsack". The main area of the application is a large, empty rectangular box. At the bottom of the application, there is a "Refresh" button.

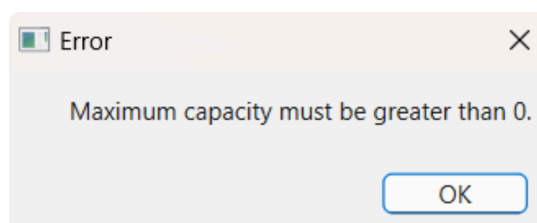
Hình 1: Giao diện màn hình chính

3.2. Chi tiết các chức năng

- *Trường nhập “Maximum Capacity”*: Đây là trường để người dùng nhập trọng lượng tối đa của túi.
- *Nút “Enter”*: Đây là nút xác nhận trọng lượng tối đa nhập vào, sau khi nhấn nút này, các nút như “Add Item”, “Delete Item”, “Solve Knapsack” sẽ được

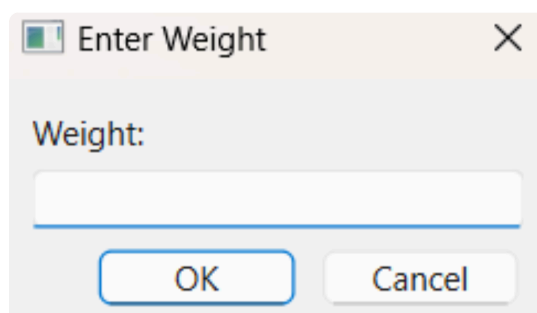
kích hoạt. Đồng thời, nút này còn để xác nhận cập nhật trọng lượng tối đa mới nếu như muốn sửa trọng lượng tối đa của túi nhưng vẫn muốn giữ nguyên các đồ vật, không cần phải nhập lại từ đầu.

- *Cửa sổ báo lỗi*: Đây là cửa sổ hiện ra nếu người dùng nhập trọng lượng tối đa của túi bằng 0.



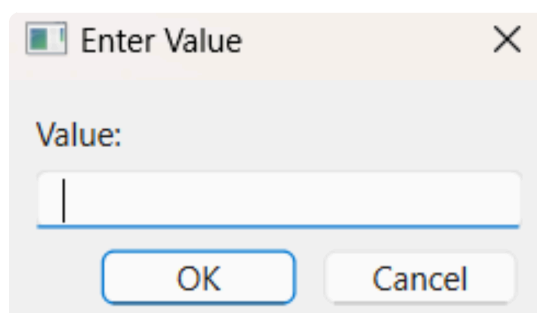
Hình 2: Giao diện báo lỗi

- *Nút “Add Item”*: Đây là nút để thêm các đồ vật với giá trị và khối lượng tương ứng. Sau khi nhấn nút này, sẽ xuất hiện hai cửa sổ nhỏ, lần lượt chứa các trường để nhập khối lượng của vật và giá trị của vật.
- *Cửa sổ chứa trường nhập “Weight”*: Đây là cửa sổ chứa trường để người dùng nhập khối lượng của đồ vật.



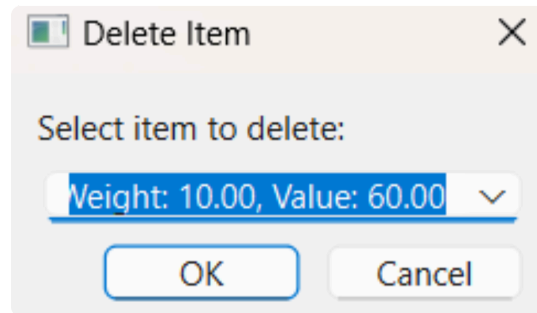
Hình 3: Giao diện trường nhập “Weight”

- *Cửa sổ chứa trường nhập “Value”*: Đây là cửa sổ chứa trường để người dùng nhập giá trị của đồ vật.



Hình 4: Giao diện trường nhập “Value”

- *Nút “Delete Item”*: Đây là nút để xóa đồ vật một cách riêng lẻ mà không cần phải nhập lại từ đầu. Sau khi xóa đồ vật, trường hiển thị kết quả sẽ tự động được xóa và cập nhật mới.
- *Cửa sổ chứa trường nhập “Select item to delete”*: Đây là cửa sổ chứa trường để người dùng nhập khối lượng và giá trị của đồ vật hoặc thao tác với danh sách thả xuống để lựa chọn đồ vật muốn xóa. Đồ vật xuất hiện mặc định trong trường nhập là đồ vật được thêm vào đầu.



Hình 5: Giao diện trường xóa đồ vật

- *Nút “Solve Knapsack”*: Đây là nút để thực hiện giải quyết bài toán sau khi đã nhập thông tin khối lượng và giá trị của các đồ vật. Sau khi cập nhật giá trị mới như thêm đồ vật mới, hoặc nhập trọng lượng mới rồi nhấn nút “Solve Knapsack”, nút này sẽ tự động xóa dữ liệu kết quả cũ và hiện ra dữ liệu kết quả mới trên trường hiển thị kết quả.
- *Trường hiển thị đồ vật*: Đây là trường hiển thị các đồ vật được nhập vào.



Hình 6: Trường hiển thị đồ vật

- *Trường hiển thị kết quả*: Đây là trường hiển thị kết quả của bài toán. Bao gồm các đồ vật được chọn (Selected items), giá trị lớn nhất (Maximum value) và tổng trọng lượng (Total weight).

```
Selected items:  
Weight: 6.00, Value: 75.00  
Weight: 7.00, Value: 60.00  
Weight: 1.00, Value: 5.00  
Weight: 1.00, Value: 5.00  
*****  
Maximum value: 145.00  
Total weight: 15.00
```

Hình 7: Trường hiển thị kết quả bài toán

- *Nút “Refresh”*: Đây là nút cập nhật toàn bộ cửa sổ, sau khi nhấn nút này, toàn bộ các trường sẽ được làm mới, trạng thái các nút cũng sẽ trở lại như cũ.

CHƯƠNG 4: CÁC KẾT QUẢ THỰC NGHIỆM

4.1. Các tình huống

Thực hiện thuật toán tham lam (Greedy Algorithm) cho các bài toán, ta thu được các kết quả là những tình huống khác nhau như sau:

- **Tình huống 1:** Giả sử bài toán cho cái túi có thể chứa trọng lượng tối đa là 15 (kg), trọng lượng và giá trị tương ứng của các vật được thể hiện trong bảng sau:

Items	1	2	3	4	5	6	7
Value	5	10	15	7	8	9	4
Weight	1	3	5	4	1	4	2

Kết quả thu được ở tình huống 1:

```
Selected items:  
Weight: 1.00, Value: 8.00  
Weight: 1.00, Value: 5.00  
Weight: 3.00, Value: 10.00  
Weight: 5.00, Value: 15.00  
Weight: 4.00, Value: 9.00  
Weight: 1.00, Value: 2.00  
*****  
Maximum value: 49.00  
Total weight: 15.00
```

Hình 8: Kết quả tình huống 1

- **Tình huống 2:** Giả sử bài toán cho cái túi có thể chứa trọng lượng tối đa là 15 (kg), trọng lượng và giá trị tương ứng của các vật được thể hiện trong bảng sau:

Items	1	2	3	4	5	6	7
Value	5	10	15	20	30	35	40
Weight	1	2	3	4	6	7	8

Kết quả thu được ở tình huống 2:

```
Selected items:
Weight: 1.00, Value: 5.00
Weight: 2.00, Value: 10.00
Weight: 3.00, Value: 15.00
Weight: 4.00, Value: 20.00
Weight: 5.00, Value: 25.00
*****
Maximum value: 75.00
Total weight: 15.00
```

Hình 9: Kết quả tình huống 2

- **Tình huống 3:** Giả sử bài toán cho cái túi có thể chứa trọng lượng tối đa là 15 (kg), trọng lượng và giá trị tương ứng của các vật được thể hiện trong bảng sau:

Items	1	2	3	4	5	6	7	8
Value	5	10	15	20	75	60	40	50
Weight	1	2	3	4	6	7	8	12

Kết quả thu được ở tình huống 3:

```
Selected items:
Weight: 6.00, Value: 75.00
Weight: 7.00, Value: 60.00
Weight: 1.00, Value: 5.00
Weight: 1.00, Value: 5.00
*****
Maximum value: 145.00
Total weight: 15.00
```

Hình 10: Kết quả tình huống 3

- **Tình huống 4:** Giả sử bài toán cho cái túi có thể chứa trọng lượng tối đa là 15.6 (kg), trọng lượng và giá trị tương ứng của các vật được thể hiện trong bảng sau:

Items	1	2	3	4	5	6	7	8
Value	5	10	15.3	21	75.9	60	40	50.45
Weight	1.2	2	3	4	6	7.4	9	12.75

Kết quả thu được ở tình huống 4:

```

Selected items:
Weight: 6.00, Value: 75.90
Weight: 7.40, Value: 60.00
Weight: 2.20, Value: 11.55
*****
Maximum value: 147.45
Total weight: 15.60

```

Hình 11: Kết quả tình huống 4

- **Tình huống 5:** Giả sử bài toán cho cái túi có thể chứa trọng lượng tối đa là 100 (kg), trọng lượng và giá trị tương ứng của các vật được thể hiện trong bảng sau:

Items	1	2	3	4	5	6	7
Value	3	0	50	60	0	29	70
Weight	1	2	12	18	4	10	15

Kết quả của tình huống 5:

```
Selected items:
Weight: 15.00, Value: 70.00
Weight: 12.00, Value: 50.00
Weight: 18.00, Value: 60.00
Weight: 1.00, Value: 3.00
Weight: 10.00, Value: 29.00
*****
Maximum value: 212.00
Total weight: 56.00
```

Hình 12: Kết quả tình huống 5

- **Tình huống 6:** Giả sử bài toán cho cái túi có thể chứa trọng lượng tối đa là 20 (kg), trọng lượng và giá trị tương ứng của các vật được thể hiện trong bảng sau:

Items	1	2	3	4	5	6	7
Value	3	6	36	54	60	30	45
Weight	1	2	12	18	20	10	15

Kết quả thu được ở tình huống 6:

```
Selected items:
Weight: 1.00, Value: 3.00
Weight: 2.00, Value: 6.00
Weight: 12.00, Value: 36.00
Weight: 5.00, Value: 15.00
*****
Maximum value: 60.00
Total weight: 20.00
```

Hình 13: Kết quả tình huống 6

4.2. Phân tích và đánh giá

- **Tình huống 1:** Trọng lượng, giá trị và các đơn giá của các đồ vật đều khác nhau, thuật toán Greedy tiến hành tính đơn giá từng đồ vật, sắp xếp các đồ vật theo thứ

tự đơn giá giảm dần và ưu tiên chọn các đồ vật có đơn giá từ cao đến thấp cho đến khi đầy túi.

- **Tình huống 2:** Các đồ vật có trọng lượng, giá trị khác nhau nhưng tất cả các vật lại có cùng tỷ lệ đơn giá, thuật toán Greedy tiến hành tính đơn giá của từng đồ vật nhưng không sắp xếp các đồ vật theo thứ tự đơn giá giảm dần, mà chọn các đồ vật theo thứ tự đưa vào danh sách lựa chọn ban đầu cho đến khi đầy túi.
- **Tình huống 3:** Các đồ vật có trọng lượng, giá trị khác nhau nhưng có một số vật lại có cùng tỷ lệ đơn giá (Item1, Item2, Item3, Item4, Item7), thuật toán Greedy tiến hành tính đơn giá từng đồ vật, sắp xếp lại thứ tự các đồ vật theo đơn giá giảm dần và ưu tiên chọn các đồ vật có đơn giá từ cao đến thấp. Đối với các đồ vật có cùng tỷ lệ đơn giá, thuật toán Greedy không tiến hành sắp xếp lại thứ tự các đồ vật theo đơn giá giảm dần mà chọn các đồ vật theo thứ tự đưa vào danh sách lựa chọn ban đầu cho đến khi đầy túi.
- **Tình huống 4:** Ở tình huống này, khối lượng tối đa, trọng lượng và giá trị của các đồ vật là các số thực, thuật toán Greedy tiến hành tính đơn giá từng đồ vật, sắp xếp các đồ vật theo thứ tự đơn giá giảm dần và ưu tiên chọn các đồ vật có đơn giá từ cao đến thấp.
- **Tình huống 5:** Các đồ vật có trọng lượng, giá trị khác nhau nhưng có một số vật lại có giá trị bằng 0 (Item2, Item5) và trọng lượng tối đa của túi lớn hơn rất nhiều so với tổng trọng lượng của tất cả đồ vật, thuật toán Greedy tiến hành tính đơn giá từng đồ vật, sắp xếp các đồ vật theo thứ tự đơn giá giảm dần và ưu tiên chọn các đồ vật có đơn giá từ cao đến thấp. Sau cùng, ngoại trừ những đồ vật có giá trị bằng 0, tất cả đồ vật trong danh sách lựa chọn đều được thêm vào túi.
- **Tình huống 6:** Các đồ vật có trọng lượng, giá trị khác nhau nhưng tất cả các vật lại có cùng tỷ lệ đơn giá, ngoài ra trong số các đồ vật, Item5 có trọng lượng bằng với trọng lượng tối đa của túi, thuật toán Greedy tiến hành tính đơn giá từng đồ vật nhưng không sắp xếp các đồ vật theo thứ tự đơn giá giảm dần, mà chọn các đồ vật theo thứ tự đưa vào danh sách lựa chọn ban đầu cho đến khi đầy túi.

CHƯƠNG 5: KẾT LUẬN

5.1 Các kết quả đạt được

- Ở tình huống 1, giá trị cao nhất sau khi bỏ các đồ vật là 49.00 với tổng khối lượng túi đạt được là 15 (kg) thì thuật toán greedy chọn những đồ vật sau: (1, 8), (1, 5), (3, 10), (5, 15), (4, 9), (1, 2).
- Ở tình huống 2, giá trị cao nhất sau khi bỏ các đồ vật là 75.00 với tổng khối lượng túi đạt được là 15 (kg) thì thuật toán greedy chọn những đồ vật sau: (1, 5), (2, 10), (3, 15), (4, 20), (5, 25).
- Ở tình huống 3, giá trị cao nhất sau khi bỏ các đồ vật là 145.00 với tổng khối lượng túi đạt được là 15 (kg) thì thuật toán greedy chọn những đồ vật sau: (6, 75), (7, 60), (1, 5), (1, 5).
- Ở tình huống 4, giá trị cao nhất sau khi bỏ các đồ vật là 147.45 với tổng khối lượng túi đạt được là 15.6 (kg) thì thuật toán greedy chọn những đồ vật sau: (6, 75.9), (7.4, 60), (2.2, 11.55).
- Ở tình huống 5, giá trị cao nhất sau khi bỏ các đồ vật là 212.00 với tổng khối lượng túi đạt được là 56 (kg) thì thuật toán greedy chọn những đồ vật sau: (15, 70), (12, 50), (18, 60), (1, 3), (10, 29).
- Ở tình huống 6, giá trị cao nhất sau khi bỏ các đồ vật là 60.00 với tổng khối lượng túi đạt được là 20 (kg) thì thuật toán greedy chọn những đồ vật sau: (1, 3), (2, 6), (12, 36), (5, 15).

5.2. Những hạn chế và hướng phát triển

Thuật toán tham lam là một trong những phương pháp được sử dụng rộng rãi để giải quyết các vấn đề tối ưu trong nhiều lĩnh vực khác nhau. Tuy nhiên, thuật toán tham lam cũng có một số hạn chế cần được lưu ý. Những hạn chế của thuật toán tham lam là có thể được kể tới như:

- Khả năng giải pháp được đưa ra rơi vào tối ưu cục bộ, khiến cho thuật toán không đảm bảo tìm được giải pháp tối ưu toàn cục.
- Thuật toán tham lam không đảm bảo hoàn toàn về việc tìm ra được giải pháp tối ưu nhất hay có thể đưa ra được các giải pháp mang tính tối ưu cho một vài trường hợp.

- Các thuật toán tham lam có thể không được áp dụng cho các vấn đề mà các giải pháp tối ưu trong vấn đề ấy phụ thuộc vào thứ tự được xử lý, kích thước hay thành phần của đầu vào.
- Thuật toán tham lam cũng có thể nhạy cảm và có thể không thể xử lý được ràng buộc về không gian hay những thay đổi của đầu vào, điều này có thể ảnh hưởng tới thuật toán khiến cho thuật toán không còn tính ổn định và không thể đoán trước.

Bên cạnh đó, thuật toán tham lam cũng có thể được phát triển thêm bằng những cách sau đây:

- Tối ưu hóa thuật toán cùng với việc kết hợp với thuật toán tối ưu toàn cục như Simulated Annealing hay thuật toán Genetic sẽ giúp cho thuật toán tham lam cải thiện được khả năng tìm kiếm các giải pháp tối ưu toàn cục.
- Việc phát triển công cụ, framework cũng như là thuật toán tham lam song song và phân tán cũng giúp cho việc áp dụng thuật toán này trở nên hữu dụng hơn về khả năng ứng dụng mà còn mở rộng thêm cơ hội nghiên cứu và cải tiến mới, giúp cho việc giải quyết các bài toán phức tạp hiệu quả hơn.

TÀI LIỆU THAM KHẢO

PHỤ LỤC