BÀI 3. CHIẾN LƯỢC THAM LAM

(Greedy Strategy)

1. Khái niệm

Chiến lược tham lam (greedy strategy) là một phương pháp trong thiết kế thuật toán mà tại mỗi bước của quá trình giải quyết vấn đề, người ta chọn hành động tốt nhất hiện có mà không quan tâm đến toàn bộ bài toán. Nói cách khác, chiến lược tham lam liên tục thực hiện những lựa chọn cục bộ tốt nhất với hy vọng rằng những lựa chọn này sẽ dẫn đến giải pháp tối ưu toàn cục.

Trong chiến lược tham lam, "hành động tốt nhất" tại mỗi bước được hiểu là lựa chọn cục bộ (local choice) mà có vẻ hứa hẹn nhất hoặc mang lại lợi ích ngay lập tức lớn nhất, mà không xét đến hậu quả lâu dài hay các bước tiếp theo. Việc xác định hành động tốt nhất thường dựa trên một tiêu chí cụ thể phù hợp với mục tiêu của bài toán.

Các bước để xác định hành động tốt nhất thường bao gồm:

[1]. Xác định tiêu chí lựa chọn: Đặt ra các tiêu chí rõ ràng để đánh giá hành động nào là tốt nhất. Tiêu chí này thường liên quan trực tiếp đến mục tiêu tối ưu hóa của bài toán.

[2]. Đánh giá các lựa chọn: Tại mỗi bước, đánh giá tất cả các lựa chọn có thể dựa trên tiêu chí đã đặt ra.

[3]. Chọn hành động tối ưu cục bộ: Chọn lựa hành động thỏa mãn tiêu chí tốt nhất tại thời điểm hiện tại.

Ví dụ về hành động tốt nhất trong một số bài toán cụ thể:

Bài toán cây khung nhỏ nhất (Minimum Spanning Tree):

- Tiêu chí: Chọn cạnh có trọng số nhỏ nhất mà không tạo thành chu trình.
- Hành động tốt nhất: Trong thuật toán Kruskal, tại mỗi bước chọn cạnh nhỏ nhất từ các cạnh còn lại chưa được chọn.

Bài toán chọn hoạt động (Activity Selection Problem):

- Tiêu chí: Chọn hoạt động có thời gian kết thúc sớm nhất mà không xung đột với các hoạt động đã chọn trước đó.

- Hành động tốt nhất: Tại mỗi bước, chọn hoạt động có thời gian kết thúc sớm nhất từ các hoạt động còn lại chưa được chọn.

Bài toán bộ tiền xu (Coin Changing):

- Tiêu chí: Chọn đồng tiền có giá trị lớn nhất mà không vượt quá số tiền cần đổi.
- Hành động tốt nhất: Tại mỗi bước, chọn đồng tiền lớn nhất từ các đồng tiền còn lại chưa được sử dụng.

2. Nguyên lý hoạt động

Nguyên lý hoạt động của chiến lược tham lam dựa trên việc thực hiện các lựa chọn tốt nhất tại mỗi bước của quá trình giải quyết vấn đề, nhằm đạt được giải pháp tối ưu toàn cục. Dưới đây là các bước cơ bản để áp dụng chiến lược tham lam:

[1]. Khởi tạo: Bắt đầu với một tập hợp rỗng hoặc một trạng thái ban đầu phù hợp.

[2]. Lựa chọn tham lam: Ở mỗi bước, chọn hành động tốt nhất theo tiêu chí đã xác định. Hành động tốt nhất là hành động mà tại thời điểm đó, có vẻ là lựa chọn tối ưu nhất dựa trên các thông tin hiện có.

[3]. Cập nhật trạng thái: Cập nhật trạng thái của bài toán dựa trên lựa chọn vừa thực hiện. Điều này có thể bao gồm việc loại bỏ các lựa chọn không còn hợp lệ hoặc cập nhật các giá trị liên quan.

[4]. Kiểm tra điều kiện dừng: Kiểm tra xem bài toán đã được giải quyết hoàn toàn chưa. Nếu có, kết thúc quá trình. Nếu chưa, quay lại bước 2.

[5]. Thu kết quả: Khi điều kiện dừng được thỏa mãn, giải pháp cuối cùng được tạo ra từ các lựa chọn cục bộ đã thực hiện trong suốt quá trình.

3. Đặc điểm của chiến lược tham lam

Chiến lược tham lam có một số đặc điểm chính giúp nó phân biệt với các chiến lược khác trong thiết kế thuật toán. Các đặc điểm này bao gồm:

[1]. Lựa chọn cục bộ tối ưu (Greedy-choice property):

Tại mỗi bước, chiến lược tham lam chọn hành động tốt nhất mà không cần phải xem xét các lựa chọn trong tương lai hay giải pháp tổng thể. Quyết định được đưa ra dựa trên tình hình hiện tại với hy vọng rằng các lựa chọn cục bộ này sẽ dẫn đến giải pháp tối ưu toàn cục.

[2]. Cấu trúc con tối ưu (Optimal substructure):

Một bài toán có thể được chia thành các bài toán con, và giải pháp tối ưu cho bài toán lớn có thể được xây dựng từ các giải pháp tối ưu của các bài toán con. Điều này có nghĩa là giải quyết các phần nhỏ của bài toán sẽ giúp đạt được giải pháp tối ưu cho toàn bộ bài toán.

[3]. Đơn giản và hiệu quả:

Chiến lược tham lam thường đơn giản và dễ hiểu, với thời gian tính toán thấp. Thuật toán tham lam thường có độ phức tạp thời gian thấp hơn so với các chiến lược khác như quy hoạch động.

[4]. Không thể đảo ngược (Non-reversible):

Một khi một quyết định được thực hiện, nó không thể bị đảo ngược. Chiến lược tham lam không quay lại để xem xét lại các lựa chọn trước đó. Điều này khác với các chiến lược như quay lui (backtracking) hay quy hoạch động.

[5]. Không đảm bảo tối ưu toàn cục:

Chiến lược tham lam không phải lúc nào cũng đảm bảo tìm ra giải pháp tối ưu toàn cục. Hiệu quả của nó phụ thuộc vào cấu trúc của bài toán và tiêu chí lựa chọn cục bộ. Có nhiều bài toán mà chiến lược tham lam không thể tìm ra giải pháp tối ưu.

4. Các điều kiện để chiến lược tham lam hiệu quả

Chiến lược tham lam thường hiệu quả khi bài toán có hai đặc điểm chính:

- **Optimal Substructure (Cấu trúc con tối ưu):** Lời giải tối ưu cho bài toán lớn được xây dựng từ lời giải tối ưu của các bài toán con.
- **Greedy Choice Property (Thuộc tính lựa chọn tham lam):** Lựa chọn tốt nhất tại mỗi bước dẫn đến lời giải tối ưu toàn cục. Lựa chọn tại mỗi bước phải là lựa chọn tối ưu cục bộ dẫn đến giải pháp tốt nhất cho bài toán tổng thể.

- **No backward dependencies (Không có sự phụ thuộc ngược)**: Các quyết định được đưa ra tại mỗi bước không làm ảnh hưởng đến các quyết định đã được đưa ra trước đó. Điều này đảm bảo rằng: Khi một hành động được chọn, nó không cần phải thay đổi hoặc xem xét lại dựa trên các hành động tiếp theo.

5. Ví dụ minh họa

Dưới đây là một số bài toán điển hình sử dụng chiến lược tham lam:

5.1. Coin changing

Mô tả bài toán:

Bài toán "Coin Changing" (đổi tiền) là một bài toán kinh điển trong lý thuyết thuật toán, nơi chúng ta cần tìm cách đổi một số tiền cụ thể bằng số lượng ít nhất các đồng xu có mệnh giá cho trước.

Ví dụ: giả sử bạn có các mệnh giá đồng xu là 1, 5, 10 và 25 (các đồng xu phổ biến ở Mỹ) và bạn cần đổi số tiền n = 63 đơn vị. Giả sử số đồng xu của mỗi mệnh giá là vô hạn. Mục tiêu của bài toán là lựa chọn số lượng ít nhất các đồng xu để tổng mệnh giá của chúng bằng 63.

25 10 5 1

Hình 3.1. Danh sách các mệnh giá có thể lựa chọn

Chiến lược tham lam:

- Tại mỗi bước, chọn đồng xu có mệnh giá lớn nhất nếu có thể (tức không vượt quá số tiền còn lại cần đổi).

Rõ ràng là tại mỗi bước, việc lựa chọn đồng xu có mệnh giá cao nhất thể hiện sự tối ưu hóa cục bộ, tức là sự lựa chọn tốt nhất tại thời điểm đó để có số lượng đồng xu được chọn là ít nhất mà không quan tâm tới các lựa chọn trước đó hay sau đó.

Các bước thực hiện:

Bước 1. Khởi tạo số tiền cần đổi là 63.

Bước 2. Chọn đồng xu có mệnh giá lớn nhất không vượt quá 63:

- Chọn đồng xu 25.
- Số tiền còn lại cần đổi: 63 - 25 = 38.

Bước 3. Chọn đồng xu có mệnh giá lớn nhất không vượt quá 38:

- Chọn đồng xu 25.
- Số tiền còn lại cần đổi: 38 - 25 = 13.

Bước 4. Chọn đồng xu có mệnh giá lớn nhất không vượt quá 13:

- Chọn đồng xu 10.
- Số tiền còn lại cần đổi: 13 - 10 = 3.

Bước 5. Chọn đồng xu có mệnh giá lớn nhất không vượt quá 3:

- Chọn đồng xu 1.
- Số tiền còn lại cần đổi: 3 - 1 = 2.

Bước 6. Chọn đồng xu có mệnh giá lớn nhất không vượt quá 2:

- Chọn đồng xu 1.
- Số tiền còn lại cần đổi: 2 - 1 = 1.

Bước 7. Chọn đồng xu có mệnh giá lớn nhất không vượt quá 1:

- Chọn đồng xu 1.
- Số tiền còn lại cần đổi: 1 - 1 = 0.

Kết quả: Để đổi số tiền n = 63 đơn vị, chiến lược tham lam chọn các đồng xu như sau:

- 2 đồng xu 25
- 1 đồng xu 10
- 3 đồng xu 1.

Tổng cộng là 6 đồng xu (2x25 + 1x10 + 3x1).

Phân tích:

Chiến lược tham lam hoạt động tốt với tập hợp các mệnh giá đồng xu như 1, 5, 10, 25 vì chúng cho phép luôn chọn đồng xu lớn nhất mà không gây ra trường hợp "bỏ lỡ" giải pháp tối ưu.

Tuy nhiên, chiến lược tham lam không phải lúc nào cũng đảm bảo tìm ra giải pháp tối ưu cho mọi tập hợp mệnh giá đồng xu. Ví dụ, nếu chúng ta có các mệnh giá đồng xu là 1, 3 và 4, và cần đổi số tiền 6 đơn vị, chiến lược tham lam sẽ chọn 4 + 1 + 1 (3 đồng xu), trong khi giải pháp tối ưu là 3 + 3 (2 đồng xu). Việc lựa chọn tham lam ngay từ đầu (chọn mệnh giá 4) khiến cho sau đó không thể chọn

đồng xu có mệnh giá 3 mà phải lựa chọn nhiều đồng xu có mệnh giá nhỏ hơn. Trong trường hợp này, chiến lược tham lam không hoạt động tốt và ta cần sử dụng các phương pháp khác như quy hoạch động để tìm giải pháp tối ưu.

Chiến lược tham lam đôi khi không tìm được nghiệm tối ưu của bài toán, mặc dù nghiệm này vẫn tồn tại. Hãy xem xét ví dụ sau: giả sử chúng ta có các mệnh giá đồng xu là 3 và 4, và cần đổi số tiền 6 đơn vị. Có thể thấy một lựa chọn tối ưu là 2 đồng có mệnh giá 3 (2x3 = 6). Tuy nhiên, chiến lược tham lam chọn đồng xu mệnh giá lớn nhất (4), sau đó không thể tìm được đồng xu nào để hoàn thành số tiền còn lại (2). Mặc dù có giải pháp hợp lệ tồn tại (2 đồng xu mệnh giá 3), chiến lược tham lam không tìm thấy do việc lựa chọn không phù hợp ban đầu.

Giải thuật tham lam cho bài toán đổi tiền:

Bước 1. Sắp xếp các đồng xu theo thứ tự giảm dần của mệnh giá.

Bước 2. Lặp lại các bước sau cho đến khi số tiền còn lại là 0:

a. Chọn đồng xu có mệnh giá lớn nhất trong danh sách đồng xu và kiểm tra xem nó có nhỏ hơn hoặc bằng số tiền còn lại không.

b. Trừ đi mệnh giá của đồng xu đã chọn khỏi số tiền còn lại.

c. Tăng số lượng đồng xu được sử dụng lên 1:

Bước 3. Trả về kết quả là số lượng đồng xu đã sử dụng.

Một giải thuật cụ thể hơn có thể được mô tả như sau:

```
bool CASHIERS_ALGORITHM(int *C, int m, long n, int * S)
{
   Khởi tạo S[]: S[i]=0 ∀i = 0..m;
   i=0;
   while (n>0 && i<m)
   {
                  S[i] = Số xu có mệnh giá C[i] nhiều nhất có thể lấy;
                  n = Số tiền còn lại ;
                  i++;
   }
   if(n>0)     return false;
   else        return true;
}
```

Đầu vào:

- `int *C`: Mảng chứa các mệnh giá của đồng xu.
- `int m`: Số loại đồng xu hay số mệnh giá khác nhau.
- `long n`: Số tiền cần đổi.

Đầu ra:

- Hàm trả về `true` nếu số tiền `n` đã được đổi thành công, ngược lại `false`.
- `int *S`: Đối ra của hàm. Hàm trả về mảng S chứa số lượng đồng xu từ mỗi loại mà thuật toán đã chọn để đổi số tiền `n` thông qua đối ra.

Hàm viết bằng C++:

```
bool CASHIERS_ALGORITHM(int *C, int m, long n, int *S)
{
	for (int i = 0; i < m; ++i) S[i] = 0;
	int i = 0;
	while (n > 0 && i < m)
	{
		S[i] = n / C[i];
		n = n % C[i];
		++i;
	}
	if (n > 0) return false;
	return true;
}
```

5.2. Bottling (chiết nước vào chai)

Một bình chứa chứa đầy nước với một lượng nước hữu hạn n. Cho m chiếc chai rỗng (dung tích các chai có thể khác nhau) để chiết nước từ bình chứa vào đầy các chai. Hãy cho biết số lượng chai tối đa có thể được đổ đầy nước.

Ví dụ: Bình chứa nước có dung tích n=20 lít, và có m = 5 chai rỗng với dung tích chai lần lượt là 5, 7, 10, 4, và 8 lít. Dễ thấy số chai nhiều nhất mà 20 lít nước có thể đổ đầy là 3.

Chiến lược tham lam:

- Lần lượt đổ nước từ bình chứa vào các chai theo thứ tự: chai nhỏ trước, chai lớn sau cho tới khi chai được đổ đầy hoặc bình chứa hết nước.

Rõ ràng là tại mỗi bước, việc lựa chọn chai có dung tích nhỏ nhất để đổ đầy nước thể hiện sự tối ưu hóa cục bộ, tức là sự lựa chọn tốt nhất tại thời điểm đó để có số

lượng chai được đổ đầy là ít nhất mà không quan tâm tới các lựa chọn trước đó hay sau đó.

Hàm viết bằng C++:

```
int maxBottlesFilled(int *C, int m, long n)
{
      int filledBottles = 0;
      for (int i = 0; i < m; ++i)
      {
            if (n >= C[i])
            {
                  n -= C[i];
                        filledBottles++;
            }
            else break;
         }
     }
     return filledBottles;
}
```

5.3. Knapsack (xếp ba lô)

Mô tả: Bạn có một chiếc túi với tải trọng tối đa được cố định và một danh sách các đồ vật có giá trị và trọng lượng khác nhau. Mục tiêu là chọn một tập hợp các đồ vật sao cho tổng trọng lượng không vượt quá tải trọng tối đa của túi, và tổng giá trị của các đồ vật được chọn là lớn nhất có thể.

Chiến lược tham lam:

- Sắp xếp các đồ vật theo tỷ lệ **giá trị/trọng lượng** giảm dần.
- Lần lượt thêm các đồ vật vào túi, bắt đầu từ đồ vật có tỷ lệ **giá trị/trọng lượng** cao nhất cho đến khi túi đầy.

Ví dụ:

Giả sử bạn có một túi có trọng lượng tối đa là 15 và các đồ vật sau:

Đồ vật	Giá trị	Trọng lượng	Giá trị/ Trọng lượng
A	10	2	5.0
B	5	3	1.7
C	15	5	3.0
D	7	7	1.0
E	6	1	6.0

Bước 1: Sắp xếp các đồ vật theo tỷ lệ **giá trị/trọng lượng** giảm dần:

Đồ vật	Giá trị	Trọng lượng	Giá trị/ Trọng lượng
E	6	1	**6.0**
A	10	2	**5.0**
C	15	5	**3.0**
B	5	3	**1.7**
D	7	7	**1.0**

Bước 2: Bắt đầu thêm các đồ vật vào túi:

- Chọn đồ vật E: tổng giá trị = 6, tổng trọng lượng = 1, còn lại trọng lượng trong túi là 14.
- Chọn đồ vật A: tổng giá trị = 6 + 10 = 16, tổng trọng lượng = 1 + 2 = 3, còn lại trọng lượng trong túi là 12.
- Chọn đồ vật C: tổng giá trị = 31, tổng trọng lượng = 3 + 5 = 8, còn lại trọng lượng trong túi là 7.
- Chọn đồ vật B: tổng giá trị = 36, tổng trọng lượng = 11, còn lại trọng lượng trong túi là 4.
- Không thể chọn thêm bất kỳ đồ vật nào khác vì tổng trọng lượng sẽ vượt quá trọng lượng tối đa của túi.

Kết quả: Các đồ vật được chọn là E, A, C và B, với tổng giá trị là 36 và tổng trọng lượng là 11.

Trong ví dụ này, chiến lược tham lam hoạt động bằng cách chọn đồ vật có tỷ lệ **giá trị/trọng lượng** cao nhất mỗi lần, mà không xem xét lại các lựa chọn trước đó.

Chương trình viết bằng C++:

- Hàm `knapsackGreedy()` nhận các đối vào như sau:
  - double v[]: mảng chứa các giá trị của đồ vật.
  - double w[]: mảng chứa trọng lượng của đồ vật.
  - int n: số đồ vật hiện có (kích thước của v và w).
  - double capacity: tải trọng tối đa của túi.
- Hàm `knapsackGreedy()` trả về các kết quả sau, thông qua đối ra:
  - double& totalv: tổng giá trị đồ vật chất vào túi.
  - double& totalw: tổng trọng lượng túi sau khi chất đồ.
  - bool r[]: một mảng có kích thước n, trong đó r[i] = 1 đồng nghĩa với việc đồ vật thứ i được chọn, r[i] = 0 là đồ vật thứ I không được chọn.

```
void knapsackGreedy(double v[], double w[], int n, double
capacity, double& totalv, double& totalw, bool r[])
{
    double c[n];    //mảng chứa tỷ lệ "giá trị/ trọng lượng"
    int pos[n];     //mảng lưu vị trí ban đầu của các đồ vật
    for(int i=0; i<n; i++)
    {
        pos[i] = i;
        r[i] = false;
    }
    for (int i = 0; i < n; i++)
        c[i] = v[i] / w[i];
    //Sắp xếp mảng c giảm dần, đồng thời sắp mảng pos
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (c[i] < c[j])
            {
                swap(c[i], c[j]);
                swap(pos[i], pos[j]);
            }
        }
    }
    totalv = 0.0;
    totalw = 0;
    for (int i = 0; i < n; i++)
    {
        if (totalw + w[pos[i]] <= capacity)
        {
            totalv += v[pos[i]];
            totalw += w[pos[i]];
            r[pos[i]] = true;
        }
        else   break;
    }
}
int main()
{
    double v[] = {10, 5, 15, 7, 6};
    double w[] = {2, 3, 5, 7, 1};
    int n = 5;
    bool r[n];
    double c = 15;
    double totalv, totalw;
    knapsackGreedy(v, w, n, c, totalv, totalw, r);
    cout<<"Tong trong luong: "<<totalw<<endl;
    cout<<"Tong gia tri    : "<<totalv<<endl;
    cout<<"Cac do vat duoc lay (1 = lay):"<<endl;
    for(int i = 0; i<n; i++)
        cout<<r[i]<<" ";
}
```

6. Ứng dụng

Chiến lược tham lam (greedy algorithm) thường được sử dụng trong nhiều lĩnh vực khác nhau do tính đơn giản và hiệu quả của nó. Dưới đây là một số ứng dụng phổ biến của chiến lược tham lam:

[1]. Bài toán lập lịch: Trong lập lịch, chiến lược tham lam thường được sử dụng để chọn ra lịch trình tốt nhất dựa trên các tiêu chí cụ thể, chẳng hạn như thời gian hoàn thành nhanh nhất, lợi nhuận cao nhất, hoặc chi phí thấp nhất.

[2]. Bài toán tối ưu hóa vận tải: Trong bài toán vận tải, chiến lược tham lam có thể được sử dụng để chọn lộ trình vận chuyển sao cho chi phí vận chuyển là thấp nhất, bằng cách lựa chọn lộ trình ngắn nhất hoặc lộ trình có chi phí vận chuyển mỗi đơn vị hàng hóa thấp nhất.

[3]. Bài toán tìm kiếm đường đi ngắn nhất (Shortest Path Problem): Trong một số trường hợp, chiến lược tham lam có thể được sử dụng để tìm đường đi ngắn nhất từ một điểm đến một điểm khác trong đồ thị.

[4]. Bài toán phân tích văn bản và ngôn ngữ tự nhiên: Trong xử lý ngôn ngữ tự nhiên, chiến lược tham lam có thể được sử dụng để tìm từ ngữ hoặc cụm từ phổ biến nhất, dựa trên số lần xuất hiện, để phục vụ cho việc tóm tắt văn bản hoặc phân loại văn bản.

[5]. Bài toán chia tiền (Change-Making Problem): Trong bài toán chia tiền, chiến lược tham lam được sử dụng để chọn loại tiền ít nhất cần phải trả lại khi trả tiền thối cho một số tiền nhất định.

[6]. Bài toán lập lịch hoạt động (Activity Scheduling Problem): Trong bài toán này, chiến lược tham lam có thể được sử dụng để chọn ra tập hợp các hoạt động sao cho số lượng hoạt động là lớn nhất mà không có xung đột thời gian.

[7]. Bài toán phân tích ảnh và xử lý video: Trong xử lý ảnh và video, chiến lược tham lam có thể được sử dụng để chọn ra các phép biến đổi hoặc phân tích đơn giản dựa trên các thuộc tính đơn lẻ của ảnh hoặc video, chẳng hạn như độ sáng, màu sắc, hoặc cạnh.

[8]. Bài toán quản lý tài nguyên (Resource Allocation Problem): Trong quản lý tài nguyên, chiến lược tham lam có thể được sử dụng để phân chia tài nguyên như máy chủ, băng thông, hoặc thiết bị, sao cho tối ưu về hiệu suất hoặc lợi nhuận.

[9]. Bài toán tìm kiếm cây khung nhỏ nhất (Minimum Spanning Tree Problem): Trong bài toán này, chiến lược tham lam có thể được sử dụng để chọn cạnh với trọng lượng nhỏ nhất mà vẫn đảm bảo rằng cây khung được tạo ra là nhỏ nhất có thể.

[10]. Bài toán quy hoạch động (Dynamic Programming): Trong một số trường hợp, chiến lược tham lam được sử dụng để giải các bài toán quy hoạch động đơn giản bằng cách chọn lựa giải pháp tốt nhất tại mỗi bước, thay vì sử dụng phương pháp lập trình động đầy đủ.

[11]. Bài toán xếp hạng (Ranking Problem): Trong một số trường hợp, chiến lược tham lam có thể được sử dụng để xếp hạng các phần tử theo một tiêu chí nhất định, chẳng hạn như xếp hạng các trang web dựa trên số lượt truy cập hoặc xếp hạng các sản phẩm trên trang web mua sắm trực tuyến dựa trên số lượng bán được.

[12]. Bài toán tối ưu hóa lượng tử (Quantum Optimization): Trong lĩnh vực lượng tử tính toán, chiến lược tham lam có thể được sử dụng để giải quyết một số bài toán tối ưu hóa đơn giản bằng cách chọn lựa phương án tốt nhất dựa trên tính toán lượng tử cơ bản.

[13]. Bài toán vận chuyển hàng hóa (Cargo Transportation Problem): Trong bài toán vận chuyển hàng hóa, chiến lược tham lam có thể được sử dụng để quyết định lộ trình vận chuyển sao cho tổng chi phí vận chuyển là thấp nhất, bằng cách lựa chọn lộ trình ngắn nhất hoặc lộ trình có chi phí vận chuyển mỗi đơn vị hàng hóa thấp nhất.

[14]. Bài toán tạo lập lịch trình hành trình (Traveling Salesman Problem): Trong một số trường hợp, chiến lược tham lam có thể được sử dụng để giải quyết bài toán hành trình người đi bán hàng bằng cách chọn lựa điểm kế tiếp gần nhất mà chưa được thăm.

[15]. Bài toán sắp xếp dữ liệu (Sorting Problem): Trong một số trường hợp, chiến lược tham lam có thể được sử dụng để sắp xếp dữ liệu bằng cách chọn lựa phương pháp sắp xếp đơn giản như sắp xếp chọn (selection sort) hoặc sắp xếp chèn (insertion sort).

[16]. Bài toán phân loại (Classification Problem): Trong học máy, chiến lược tham lam có thể được sử dụng để xây dựng các mô hình phân loại đơn giản dựa trên một số thuộc tính đơn lẻ của dữ liệu.

Các ứng dụng này chỉ là một số ví dụ và chiến lược tham lam có thể được áp dụng trong nhiều lĩnh vực khác nhau, tuỳ thuộc vào tính chất của bài toán cụ thể.