Thuật toán Tham lam là gì?

Thuật toán tham lam (Greedy Algorithm) là một thuật toán được sử dụng để giải quyết các bài toán tối ưu hóa, trong đó mục tiêu là tìm cách đạt được kết quả tốt nhất trong một số lượng lớn các lựa chọn có sẵn . Thuật toán này hoạt động theo cách chọn lựa giải pháp tại mỗi bước theo một cách tối ưu nhất định, mà không quan tâm đến các lựa chọn tiếp theo sẽ dẫn đến giải pháp tối ưu nhất hay không.

Các bước thực hiện của thuật toán tham lam bao gồm:

- Chọn một lựa chọn tối ưu nhất tại mỗi bước.
- Kiểm tra xem giải pháp tại mỗi bước có thể kết hợp với giải pháp tại các bước trước đó để tạo ra một giải pháp tối ưu toàn cục hay không.

Một số ví dụ của thuật toán tham lam bao gồm tìm đường đi ngắn nhất giữa hai điểm trên đồ thị, chia tiền mặt ít tờ tiền nhất hoặc sắp xếp công việc để hoàn thành trong thời gian ngắn nhất.

Tuy nhiên, thuật toán tham lam cũng có thể không đưa ra được giải pháp tối ưu toàn cục trong một số trường hợp, và thường cần phải được kết hợp với các thuật toán khác để đạt được kết quả tối ưu nhất.

Cho ví dụ áp dụng?

Một ví dụ điển hình của thuật toán tham lam là bài toán chia tiền. Giả sử bạn muốn chia số tiền n $ thành các tờ tiền có mệnh giá 1$, 2$, 3$ …. k$ sao cho số tờ tiền cần sử dụng là ít nhất. Giả sử có các mệnh giá tiền tệ là 1$, 5$, 10$, 25$, và số tiền cần chia là 36$. Ta muốn chia số tiền này thành các tờ tiền sao cho số tờ tiền sử dụng là ít nhất.

Để giải quyết bài toán này bằng thuật toán tham lam, ta chọn tờ tiền có mệnh giá lớn nhất mà vẫn nhỏ hơn hoặc bằng số tiền cần chia, sau đó lấy số tờ tiền đó ra khỏi số tiền cần chia. Tiếp theo, ta lặp lại quá trình này với số tiền còn lại cho đến khi số tiền cần chia là 0.

Trong trường hợp này, ta sẽ chọn tờ tiền mệnh giá 25$ và lấy một tờ tiền đó ra. Sau đó, ta sẽ còn lại số tiền cần chia là 36$ – 25$ = 11$. Tiếp theo, ta chọn tờ tiền mệnh giá 10$ và lấy 1$ tờ tiền đó ra, để lại 11-10 = 1$. Cuối cùng, ta chọn tờ tiền mệnh giá 1$ và lấy 1$ tờ tiền đó ra. Vậy, tổng số tờ tiền cần sử dụng là 3, và giá trị của các tờ tiền đó là 25$ + 10$ + 1$ = 36$.

Trong ví dụ này, thuật toán tham lam đã chọn tờ tiền có mệnh giá lớn nhất mà vẫn nhỏ hơn hoặc bằng số tiền cần chia ở mỗi bước để đạt được kết quả tối ưu. Tuy nhiên, nếu các mệnh giá tiền tệ không đủ tốt để sử dụng thuật toán tham lam để tìm giải pháp tối ưu, ta sẽ cần sử dụng các phương pháp giải quyết khác.

Dưới đây là một đoạn mã C++ áp dụng thuật toán tham lam để giải bài toán chia tiền như trong ví dụ trên:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int chia_tien(vector<int>& menh_gia, int tien) {
    // Sắp xếp các mệnh giá tiền theo thứ tự giảm dần
    sort(menh_gia.rbegin(), menh_gia.rend());
    int so_to_tien = 0;
    for (int i = 0; i < menh_gia.size(); i++) {
        int m = menh_gia[i];
        // Lấy số tờ tiền có mệnh giá m và cập nhật số tiền cần chia
        int so_to = tien / m;
        so_to_tien += so_to;
        tien -= so_to * m;
        // Kết thúc nếu đã chia hết số tiền cần chia
        if (tien == 0) {
            break;
        }
    }
    return so_to_tien;
}

int main() {
    vector<int> menh_gia = {1, 5, 10, 25};
    int tien = 36;
    int so_to_tien = chia_tien(menh_gia, tien);
    cout << "So to tien can su dung la: " << so_to_tien << endl;
    return 0;
}
```

Trong đoạn mã trên, hàm **chia_tien** nhận vào một *vector* **menh_gia** chứa các mệnh giá tiền và số tiền cần chia tiền. Đầu tiên, chúng ta sắp xếp các mệnh giá tiền theo thứ tự giảm dần bằng cách sử dụng hàm **sort** kết hợp với hàm **rbegin** và **rend**. Sau đó, với mỗi mệnh giá tiền, chúng ta lấy số tờ tiền có mệnh giá đó và cập nhật số tiền cần chia. Nếu số tiền cần chia đã bằng 0, ta kết thúc thuật toán và trả về số tờ tiền sử dụng được tính bởi biến **so_to_tien**. (hàm rbegin, rend sắp xếp giảm dần)

Cho thêm ví dụ khác và viết nó bằng C++

Ví dụ khác của thuật toán tham lam là bài toán cái túi (Knapsack problem). Giả sử bạn có một cái túi có khả năng chứa trọng lượng tối đa là W và một danh sách các vật phẩm với giá trị và trọng lượng tương ứng. Nhiệm vụ của bạn là chọn một số lượng vật phẩm sao cho giá trị tổng cộng là lớn nhất, nhưng trọng lượng tổng cộng không vượt quá trọng lượng tối đa *W* của túi.

Ví dụ, bạn có các vật phẩm sau:

Vật phẩm	*Giá trị*	*Trọng lượng*
1	*6*	*2*
2	*10*	*3*
3	*12*	*4*
4	*7*	*2*

Nếu túi có khả năng chứa tối đa W = 7 đơn vị trọng lượng, thì chọn các vật phẩm 1, 2 và 4 sẽ cho tổng giá trị là 6 + 10 + 7 = 23, trọng lượng tổng cộng là 2+3+2=7 và không có cách nào chọn các vật phẩm

khác để có tổng giá trị cao hơn. Dưới đây là đoạn mã C++ áp dụng thuật toán tham lam để giải bài toán cái túi như trong ví dụ trên:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct vat_pham {
    int gia_tri;
    int trong_luong;
};

bool ss_theo_ti_le_gia_tri_trong_luong(const vat_pham& a, const
vat_pham& b) {
    double t1 = (double) a.gia_tri / a.trong_luong;
    double t2 = (double) b.gia_tri / b.trong_luong;
    return t1 > t2;
}

double tui_cai(vector<vat_pham>& ds_vat_pham, int khoi_luong_toi_da)
{
    sort(ds_vat_pham.begin(), ds_vat_pham.end(),
ss_theo_ti_le_gia_tri_trong_luong);

    double tong_gia_tri = 0.0;
    int tong_trong_luong = 0;
    for (int i = 0; i < ds_vat_pham.size(); i++) {
        if (tong_trong_luong + ds_vat_pham[i].trong_luong <=
khoi_luong_toi_da) {
            tong_gia_tri += ds_vat_pham[i].gia_tri;
            tong_trong_luong += ds_vat_pham[i].trong_luong;
        } else {
            int can_bo = khoi_luong_toi_da - tong_trong_luong;
            tong_gia_tri += ds_vat_pham[i].gia_tri * ((double) can_bo
/ ds_vat_pham[i].trong_luong);
            break;
        }
    }

    return tong_gia_tri;
}

int main() {
    vector<vat_pham> ds_vat_pham = {{6, 2}, {10, 3}, {12, 4}, {7,
2}};
    int khoiLuongToiDa = 7;
    cout << tui_cai(ds_vat_pham, khoiLuongToiDa);

    return 0;
}
```