**2.1. Thiết kế thuật toán tham lam.**

**2.1.1. Thuật toán tham lam trong bài toán phân công công việc**

* Mục tiêu của bài toán là phân công n công việc cho n công nhân sao cho mỗi công nhân chỉ làm một công việc và mỗi công việc chỉ do một công nhân thực hiện. Tổng thời gian thực hiện các công việc là nhỏ nhất.
* Phương pháp tham lam đưa ra các lựa chọn tốt nhất ở mỗi bước (theo thời gian nhỏ nhất), mà không xét đến tác động lâu dài của những lựa chọn đó.
* Tư tưởng tham lam trong bài toán này là:

Tại mỗi bước, chọn công việc có thời gian ngắn nhất cho một công nhân mà công việc đó chưa được thực hiện bởi bất kỳ công nhân nào khác.

**2.1.2. Các bước giải thuật**

**2.1.2.1. Bài toán và phân tích yêu cầu**

* **Input**:
  + Một ma trận cost[n][n], trong đó:
    - cost[i][j] đại diện cho thời gian mà công nhân thứ i cần để thực hiện công việc thứ j.
  + Số lượng công nhân là n.
* **Output**:
  + Một phương án phân công công việc cho các công nhân sao cho mỗi công nhân được gán một công việc, mỗi công việc chỉ được thực hiện bởi một công nhân, và tổng thời gian là nhỏ nhất.
  + In ra tổng thời gian tối thiểu.
* **Hạn chế**: Mỗi công nhân chỉ làm một công việc và mỗi công việc chỉ do một công nhân thực hiện.

**2.1.2.2. Thuật toán**

1. **Khởi tạo**:
   * Một mảng assigned[] kích thước n, dùng để đánh dấu những công việc đã được gán (ban đầu chưa công việc nào được gán).
   * Một mảng assignment[] kích thước n, lưu công việc được gán cho từng công nhân.
   * Một biến totalCost để tính tổng thời gian.
2. **Quá trình phân công**:
   * Với mỗi công nhân i:
     + Tìm công việc j sao cho thời gian cost[i][j] là nhỏ nhất trong các công việc chưa được gán.
     + Gán công việc đó cho công nhân i.
     + Cộng thời gian hoàn thành công việc đó vào totalCost.
     + Đánh dấu công việc j đã được gán.
3. **Đầu ra**:
   * In ra phương án phân công công việc cho từng công nhân.
   * In ra tổng thời gian hoàn thành các công việc.

**2.1.2.3. Thiết kế chi tiết của thuật toán**

**Input:**

* cost: Ma trận chi phí thời gian n x n (số công nhân và công việc là như nhau).

**Output:**

* Mảng assignment[] chứa công việc được gán cho từng công nhân.
* Tổng thời gian hoàn thành công việc (totalCost).

**Lưu đồ thuật toán**



**2.1.2.4. Độ phức tạp thời gian**

* **Vòng lặp chính**: Duyệt qua tất cả các công nhân (tức là n công nhân).
* **Tìm công việc tốt nhất cho mỗi công nhân**: Với mỗi công nhân, phải duyệt qua tất cả các công việc (tức là n công việc).

=> Do đó, độ phức tạp thời gian của thuật toán là **O(n²)**, với n là số công nhân (và số công việc).

**2.1.2.5. Ví dụ minh họa**

Giả sử ta có ma trận chi phí thời gian như sau:

| **Công nhân/Công việc** | **Công việc 1** | **Công việc 2** | **Công việc 3** | **Công việc 4** |
| --- | --- | --- | --- | --- |
| Công nhân 1 | 10 | 19 | 8 | 15 |
| Công nhân 2 | 10 | 18 | 7 | 17 |
| Công nhân 3 | 13 | 16 | 9 | 14 |
| Công nhân 4 | 12 | 19 | 13 | 19 |

**Bước 1**: Công nhân 1 chọn công việc 3 vì có thời gian ngắn nhất (8).

**Bước 2**: Công nhân 2 chọn công việc 1 vì thời gian là 10 (sau khi công việc 3 đã được gán).

**Bước 3**: Công nhân 3 chọn công việc 4 vì có thời gian ngắn nhất (14) trong các công việc chưa gán.

**Bước 4**: Công nhân 4 chỉ còn lại công việc 2 để chọn.

Kết quả:

* Công nhân 1 làm công việc 3 với thời gian 8.
* Công nhân 2 làm công việc 1 với thời gian 10.
* Công nhân 3 làm công việc 4 với thời gian 14.
* Công nhân 4 làm công việc 2 với thời gian 19.

**Tổng thời gian**: 8 + 10 + 14 + 19 = 51.

**2.1.2.6. Đánh giá**

* **Ưu điểm**:
  + Thuật toán đơn giản, dễ hiểu và dễ cài đặt.
  + Độ phức tạp thời gian là O(n²), khả thi cho các bài toán có kích thước vừa phải.
* **Nhược điểm**:
  + Thuật toán tham lam không luôn tìm ra phương án tối ưu toàn cục. Nó chỉ đưa ra quyết định dựa trên những lựa chọn tốt nhất tại từng bước mà không xét đến tác động sau này.