BÁO CÁO

- **Giải bài toán Josephus theo danh sách liên kết vòng:**

```
struct Node                         // Cấu trúc Node của DSLK
{
        int data;
        Node *next;
};

Node *newNode(int data)             // Tạo Node mới
{
        Node *temp = new Node;
        temp->next = temp;
        temp->data = data;
        return temp;
}

void Josephus(int n, int m)         // Giải bài toán Josephus
{
        // Tạo DSLK vòng với n Node, mỗi Node có data lần lượt từ 1 đến n
        Node *head = newNode(1);
        Node *prev = head;
        for (int i = 2; i <= n; i++)
        {
                prev->next = newNode(i);
                prev = prev->next;
        }
        prev->next = head;          // Nối  Node cuối với Node đầu tạo DSLK
vòng

        Node *ptr1 = head, *ptr2 = prev;
        while (ptr1->next != ptr1)  // Lặp cho đến khi DSLK chỉ còn 1 nút
        {
                // Tìm Node thứ m để xóa
                int count = 1;
                while (count != m)
                {
                        ptr2 = ptr1;
                        ptr1 = ptr1->next;
                        count++;
                }
```

```
            // Xóa Node thứ m
            ptr2->next = ptr1->next;
            Node *p = ptr1;
            ptr1 = ptr2->next;
            delete p;
        }

        cout << "The win is " << ptr1->data << endl; // In kết quả Node còn lại cuối cùng
}
```

Để xóa (n – 1) Node, phải thực hiện vòng lặp while (n – 1) lần, do mỗi lần lặp sẽ chỉ xóa được một Node. Thêm vào đó, với mỗi lần lặp, phải duyệt m Node để tìm ra Node thứ m để xóa. Do đó độ phức tạp của thuật toán là: (n – 1) * m. Mà với n rất lớn ta có thể xấp xỉ độ phức tạp của thuật toán trên là **O(n * m)**.

- **<u>Giải bài toán Josephus theo phương pháp đệ qui:</u>**

```
int josephus(int n, int k)
{
        if (n == 1)
                return 1;
        else
                return (josephus(n - 1, k) + k - 1) % n + 1;
}
```

⇨ Từ công thức đệ quy ở trên ta có thuật toán quy nạp sau để tiếp cận với n lớn:

```
int j(int n, int k)
{
        int t = 1;
        for (int i = 2; i <= n; i++)
                t = (t + k - 1) % i + 1;
        return t;
}
```

Sau khi người đầu tiên (người thứ k đầu tiên) bị giết, còn lại (n – 1) người. Gọi tiếp josephus(n - 1, k) ta sẽ tìm được vị trí người thứ k tiếp theo với (n – 1) người. Nhưng vị trí được trả về bởi josephus (n - 1, k) sẽ xem xét vị trí bắt đầu (k % n + 1) là người số 1. Vì vậy, chúng ta thêm (k – 1) vào để điều chỉnh lại vị trí trả về, và lấy mô đun của nó với n vì có n phần tử. Và cuối cùng thêm 1(vì bắt đầu đếm thứ tự từ 1). Vì vậy với mỗi lần đệ qui ta sẽ xóa bỏ được 1 nút. Do đó để xóa được (n –

1) người thì cần đệ qui (n – 1) lần. Và với n rất lớn thì độ phức tạp của thuật toán trên xấp xỉ **O(n)**.

- **<u>Giải bài toán Josephus theo phương pháp quy hoạch động:</u>**

```
int J(int m, int n)
{
    // Nếu n nhỏ hơn m thì thực hiện đệ qui theo công thức quy nạp được chứng minh ở trên
    if (n == 1)
        return 1;
    if (n <= m)
        return (J(m, n - 1) + m - 1) % n + 1;

    // Thực hiện quy hoạch động khi n lớn hơn nhiều so với m
    int K = J(m, n - n / m);
    if (K <= n%m)
        return K + n - n%m;
    int L = K - n%m;
    return L + (L - 1) / (m - 1);
}
```

Nếu m nhỏ hơn n nhiều, thì chúng ta có thể giết nhiều người (n / m) người trong một lần chạy mà không cần lặp lại nhiều lần. Sau đó sẽ còn lại (n – n / m) người, và sẽ bắt đầu với vị trí thứ (n / m *n). Chúng ta có thể nhận thấy rằng (n / m * n) đơn giản là n mod m. Và vì đã xóa mọi người thứ m, nên phải thêm số người đã xóa trước khi trả về kết quả.
Ngoài ra, chúng ta cần xử lý trường hợp khi n trở thành nhỏ hơn m - trong trường hợp này, việc tối ưu hóa theo đệ qui sẽ cho hiệu quả tốt.

- Trường hợp n < m, nó sẽ hoạt động theo độ phức tạp của phương pháp đệ qui, trong trường hợp này sẽ là O(m).
- Trong thực tế, trên mỗi lần lặp, thay vì số n, chúng ta sẽ còn lại n * (1−1/m) người, vì vậy tổng số x lần lặp của thuật toán có thể được tìm thấy gần như từ phương trình sau:

$$n * \left(1 - \frac{1}{m}\right)^{x} = 1$$

$$\Rightarrow ln(n) + xln\left(1 - \frac{1}{m}\right) = 0$$

$$\Rightarrow x = -\frac{ln(n)}{ln\left(1-\frac{1}{m}\right)}$$

Bằng cách sử dụng sự phân hủy logarit thành chuỗi Taylor, chúng ta có được ước tính gần đúng: x≈mln(n)
Do đó, độ phức tạp của thuật toán là **O(mlogn)**.

+ Đo thời gian và nhận xét đánh giá:

Với giá trị n lớn thì phương pháp thì thời gian chạy các phương pháp khác nhau lần lượt là: DSLK vòng > Đệ qui > Quy hoạch động