**Mô tả bài toán**

* **Bài toán 8 quân hậu**

Mô tả bài toán:

Bài toán được mô tả trong mã nguồn trên là bài toán N-queens, nơi chúng ta cần đặt N quân hậu trên một bàn cờ kích thước 8x8 sao cho không có hai quân hậu nào đang tấn công nhau. Quân hậu có thể tấn công theo các đường dọc, ngang và đường chéo.

**\*Đầu vào:**

Không có đầu vào từ người dùng. Các thông số bài toán, như kích thước bàn cờ và số lượng quân hậu (8 quân hậu trong trường hợp này), được cố định trực tiếp trong mã nguồn.

**\*Đầu ra:**

- Mỗi lần tìm thấy một cách đặt quân hậu hợp lệ, mã nguồn sẽ in ra màn hình các bước đặt quân hậu và ma trận 8x8 biểu diễn vị trí của các quân hậu.

- Cuối cùng, mã nguồn sẽ in ra số lượng cách đặt quân hậu hợp lệ trên bàn cờ.

**\*Cách xử lý:**

1. Hàm `thu` được sử dụng để đệ quy và thử tất cả các cách đặt quân hậu trên bàn cờ.

2. Trong hàm `thu`, vòng lặp bên trong sẽ thử từng cột trên dòng hiện tại để đặt quân hậu. Nếu vị trí đó không bị tấn công bởi các quân hậu đã đặt trước đó, thì quân hậu sẽ được đặt và đệ quy được gọi tiếp theo để tiếp tục đặt các quân hậu trên dòng tiếp theo.

3. Khi tất cả các quân hậu đã được đặt thành công, hàm `print` sẽ được gọi để in ra màn hình ma trận biểu diễn cách đặt các quân hậu.

4. Cuối cùng, số lượng cách đặt quân hậu hợp lệ sẽ được đếm bởi biến `dem` và được in ra màn hình.

Trong mã nguồn trên, hàm `tim()` được sử dụng để tìm tất cả các cách đặt 8 quân Hậu trên bàn cờ 8x8 và in ra màn hình. Thuật toán sử dụng là "quay lui" (backtracking), trong đó từng vị trí trên bàn cờ được thử một cách tuần tự. Nếu vị trí đó hợp lệ (không bị tấn công bởi các quân Hậu đã đặt trước đó), quân Hậu được đặt vào vị trí đó và tiếp tục đặt quân Hậu tiếp theo. Nếu không hợp lệ, quay lại vị trí trước đó và thử một vị trí khác. Quá trình này được lặp lại cho đến khi tìm được tất cả các cách đặt quân Hậu hoặc không còn cách nào khả dĩ.

Kết quả của mã nguồn trên là hiển thị tất cả các cách đặt quân Hậu trên bàn cờ 8x8 và số lượng tổng cộng các cách đó.

* **Bài toán tìm xau con chung lớn nhất**

Mô tả bài toán:

Bài toán được mô tả trong mã nguồn trên là tìm xâu con chung dài nhất giữa hai xâu `s1` và `s2` sử dụng thuật toán Quy hoạch động.

**\*Đầu vào:**

- Người dùng sẽ chọn một số từ menu để thực hiện các thao tác.

- Nếu người dùng chọn 1, họ sẽ nhập vào hai xâu `s1` và `s2` từ bàn phím.

**Đầu ra:**

- Mã nguồn sẽ in ra màn hình bảng quy hoạch động và xâu con chung dài nhất của `s1` và `s2`.

- Bảng quy hoạch động hiển thị số lượng ký tự chung của các xâu con từ `s1` và `s2`.

- Đầu ra cuối cùng là xâu con chung dài nhất của `s1` và `s2`.

**\*Cách xử lý:**

1. Lớp `Xau\_Con` chứa các phương thức để thực hiện việc tìm xâu con chung dài nhất.

2. Phương thức `QHXau\_Con` thực hiện tạo bảng quy hoạch động và tìm xâu con chung dài nhất.

3. Trước tiên, độ dài của `s1` và `s2` được tính toán.

4. Một mảng hai chiều `B` và `T` được tạo để lưu trữ thông tin về bảng quy hoạch động.

5. Ban đầu, các giá trị trong mảng `B` và `T` được khởi tạo là 0.

6. Quá trình tính toán bảng quy hoạch động được thực hiện bằng cách so sánh ký tự từng vị trí của `s1` và `s2`.

7. Nếu ký tự tại vị trí `i` của `s1` và vị trí `j` của `s2` giống nhau, ta tăng giá trị `B[i][j]` lên 1 và gán `T[i][j]` bằng 0.

8. Ngược lại, nếu `B[i - 1][j]` lớn hơn hoặc bằng `B[i][j - 1]`, ta gán `B[i][j]` bằng `B[i - 1][j]` và `T[i][j]` bằng -1.

9. Nếu không, ta gán `B[i][j]` bằng `B[i][j - 1]` và `T[i][j]` bằng 1.

10. Kết quả cuối cùng là giá trị lưu trữ trong ô `B[m][n]`, tức là độ dài xâu con chung dài

nhất.

11. Xâu con chung dài nhất được tái tạo bằng cách theo dõi lại các giá trị trong mảng `T`.

12. Xâu con chung dài nhất được in ra màn hình.

13. Các vùng nhớ được giải phóng sau khi hoàn thành quá trình tính toán.

14. Trong phương thức `Xu\_Ly`, người dùng chọn các tùy chọn và thực hiện các thao tác tương ứng.

* **Bài toán QuickSort**

**Mô tả bài toán:**

Bài toán được mô tả trong mã nguồn trên là sắp xếp một mảng các số thực sử dụng thuật toán QuickSort.

**\*Đầu vào:**

- Người dùng sẽ nhập vào số lượng phần tử `n` của mảng và các giá trị của mảng `a` từ bàn phím.

**\*Đầu ra:**

- Mã nguồn sẽ in ra màn hình các phần tử của mảng `a` sau khi được sắp xếp theo thứ tự tăng dần bằng thuật toán QuickSort.

**\*Cách xử lý:**

1. Hàm `nhap` được sử dụng để đọc các giá trị đầu vào từ bàn phím. Người dùng sẽ nhập số lượng phần tử `n` và sau đó nhập các giá trị của mảng `a`.

2. Hàm `Phanchia` thực hiện việc phân chia mảng thành các phần tử nhỏ hơn và lớn hơn một giá trị chốt (`x`).

3. Trong hàm `Phanchia`, ta lựa chọn phần tử ở giữa mảng làm giá trị chốt (`x`).

4. Sử dụng hai biến `i` và `j` để duyệt mảng từ đầu đến cuối.

5. Trong quá trình duyệt, ta tìm vị trí `i` đầu tiên mà `a[i]` lớn hơn hoặc bằng `x`, và tìm vị trí `j` đầu tiên mà `a[j]` nhỏ hơn hoặc bằng `x`.

6. Nếu `i <= j`, ta hoán đổi giá trị `a[i]` và `a[j]`, tăng `i` và giảm `j`.

7. Sau khi hoàn thành vòng lặp, ta gọi đệ quy hàm `Phanchia` để phân chia các phần tử nhỏ hơn `x` và lớn hơn `x`.

8. Hàm `QuickSort` gọi hàm `Phanchia` để sắp xếp toàn bộ mảng.

9. Hàm `Ghi` được sử dụng để in ra màn hình các phần tử của mảng `a` sau khi sắp xếp.

10. Mã nguồn chính sử dụng các hàm trên để đọc dữ liệu từ người dùng, thực hiện sắp xếp mảng và in kết quả ra màn hình.