Trần Nguyễn Đắc Lãm - CT06N0130

**Bài tập 1: Xây dựng hàm đệ quy**

Bài toán 1: Tìm phần tử lớn nhất trong mảng 1 chiều a gồm 1 n phần tử nguyên

Phân tích bài toán:

B1: Thông số hoá bài toán

N chính là kích thước dữ liệu và thông số tổng quát cho bài toán

B2: Xác định phần cơ bản (neo)

Nếu n = 1 thì a[] có 1 phần tử a[0], cũng chính là phần tử lớn nhất trong mảng

B3: Xác định đệ quy (n >=2)

n = 2: max của a[] chính là max(max(a[0]), a[1])

n = 3: max của a[] chính là max(max(max(a[0]), a[1]), a[2])

…

n = n: max của a[] chính là max(max(max(a[0])… a[n-2]), a[n-1])

int max2so(int x, int y)

{

    return x > y ? x : y

}

int timMax\_DeQuy(int arr[], int n)

{

    if (n == 1)

        return arr[1]

    return max2so(timMax\_DeQuy(arr, n - 1), arr[n - 1])

}

Bài toán 2: Đếm số chữ số của số nguyên dương

Ví dụ: n = 29084952106 => kq = 11

B1: Thông số hoá bài toán

N chính là kích thước dữ liệu và thông số tổng quát cho bài toán

B2: Xác định phần cơ bản (neo)

Nếu n < 10, tức n chỉ có một chữ số, trả về 1

B3: Xác định đệ quy (n >= 10)

Nếu n >= 10 gọi đệ quy với giá trị đầu vào là n / 10 và tang kết quả lên 1 cho mỗi lần gọi đệ quy

int demSLChuSo(int n) {

    // Trường hợp cơ bản: nếu số n chỉ có 1 chữ số, trả về 1

    if (n < 10) {

        return 1;

    }

    // Sử dụng đệ quy để đếm số chữ số của số n

    return 1 + demSLChuSo(n / 10);

Bài toán 3. Liệt kê các hoán vị giữa 2 phần tử có thể có của 1 mảng có n phần tử.

B1: Thông số hoá bài toán

N chính là kích thước dữ liệu và thông số tổng quát cho bài toán

Bước 2:

Nếu dãy a có n = 1 phần tử thì số hoán vị chỉ có 1: a

Nếu dãy a có n = 2 phần tử: a[1] = a, a[2] = b thì số hoán vị là 2 dãy:

a b : b a

Nếu dãy a có n = 3 phần tử : a[1] = a, a[2] = b, a[3] = c thì có 6 hoán vị:

a b c

b a c

a c b

c a b

b c a

c b a

Bước 3:

số hoán vị là n!, dãy số hoán vị là return 1\*n\*soHoanVi(n-1)

int soHoanVi(int arr[], int n) {

    if(n==1) return 1;

    return n\*soHoanVi(n-1)

}

Bài 4: Nhập vào 1 số nguyên dương n hệ thập phân, chuyển nó thành số nhị phân tương ứng và in ra màn hình

Đệ quy:

// Hàm đệ quy để chuyển đổi số thập phân sang nhị phân

void thapPhanToNhiPhan(int n) {

    if (n > 1) {

        // Gọi đệ quy với phần nguyên của n

        thapPhanToNhiPhan(n / 2);

    }

    // In ra dư của n sau khi chia cho 2

    cout << n % 2;

}

Bài toán 5. Nhập vào số nguyên a (cơ số), số nguyên n (số mũ), tính a mũ n

// Hàm tính a mũ n bằng đệ quy

int tinhSoMu(int a, int n) {

    if (n == 0) {

        return 1; // a^0 = 1 cho mọi a (điểm dừng của đệ quy)

    } else if (n < 0) {

        return 1 / tinhSoMu(a, -n); // Xử lý trường hợp số mũ âm

    } else {

        return a \* tinhSoMu(a, n - 1); // Sử dụng đệ quy cho số mũ dương

    }

}

Bài toán 6. Tìm số Fibonacci lớn nhất mà nhỏ hơn số nguyên được cho trước

// Hàm đệ quy để tính số Fibonacci thứ n

int fibonacci(int n) {

    if (n <= 1) {

        return n;

    }

    return fibonacci(n - 1) + fibonacci(n - 2);

}

// Hàm tìm số Fibonacci lớn nhất nhỏ hơn số nguyên cho trước

int findLargestFibonacci(int limit) {

    int n = 0;

    while (fibonacci(n) < limit) {

        n++;

    }

    return fibonacci(n - 1);

}

**Bài tập 2: khử đệ quy cho bài toán sau**  
  
Bài toán 2.  
Đếm số chữ số của số nguyên dương n.   Ví dụ: n=29084952106 và kq = 11

int demSLChuSo(int n) {

    if (n == 0) {

        return 1; // Trường hợp đặc biệt khi số n là 0

    }

    int count = 0;

    stack<int> StackSL;

    // Đẩy từng chữ số của số n vào ngăn xếp

    while (n > 0) {

        StackSL.push(n % 10);

        n /= 10;

    }

    // Đếm số chữ số bằng cách lấy ra từng phần tử từ ngăn xếp

    while (!StackSL.empty()) {

        StackSL.pop();

        count++;

    }

    return count;

}

Bài toán 4. Nhập vào 1 số nguyên dương n hệ thập phân, chuyển nó thành số nhị phân tương ứng và in ra màn hình.

Khử đệ quy:

// Khử đệ quy

void F(x) {

    stack<int> binaryStack;

    if (number == 0) {

        binaryStack.push(0);

    } else {

        while (number > 0) {

            binaryStack.push(number % 2);

            number /= 2;

        }

    }

    while (!binaryStack.empty()) {

        cout << binaryStack.top();

        binaryStack.pop();

    }

}

Bài toán 5. Nhập vào số nguyên a (cơ số), số nguyên n (số mũ), tính a mũ n

// Hàm tính a mũ n không sử dụng đệ quy và sử dụng stack

int tinhSoMu(int a, int n) {

    int result = 1;

    std::stack<int> StackSoMu;

    if (n < 0) {

        std::cout << "Số mũ phải là số nguyên không âm." << std::endl;

        return -1; // Trả về -1 để chỉ ra lỗi

    }

    // Đưa từng chữ số của số mũ vào stack

    while (n > 0) {

        StackSoMu.push(n % 2);

        n /= 2;

    }

    // Sử dụng ngăn xếp để tính a mũ n

    while (!StackSoMu.empty()) {

        result \*= result;

        if (StackSoMu.top() == 1) {

            result \*= a;

        }

        StackSoMu.pop();

    }

    return result;

}

Bài toán 6. Tìm số Fibonacci lớn nhất mà nhỏ hơn số nguyên được cho trước

// Hàm tính số Fibonacci thứ n

int tinhFn(int n) {

    int a = 0, b = 1;

    for (int i = 2; i <= n; i++) {

        int temp = a + b;

        a = b;

        b = temp;

    }

    return b;

}

// Hàm tìm số Fibonacci lớn nhất nhỏ hơn số nguyên cho trước sử dụng stack

int timMaxFn(int limit) {

    stack<int> fibStack;

    int n = 0;

    // Bắt đầu tính và đẩy giá trị Fibonacci vào ngăn xếp cho đến khi vượt quá giới hạn

    while (true) {

        int fib = tinhFn(n);

        if (fib >= limit) {

            break;

        }

        fibStack.push(fib);

        n++;

    }

    // Lấy giá trị Fibonacci lớn nhất từ ngăn xếp

    int largestFibonacci = 0;

    if (!fibStack.empty()) {

        largestFibonacci = fibStack.top();

        fibStack.pop();

    }

    return largestFibonacci;

}