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

Phần 1: Giải thuật bằng đệ quy

Phân tích bài toán:

B1: Thông số hóa 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 mảng a chỉ có duy nhất 1 phần tử a[0], đồng thời cũng là phần tử lớn nhất

B3: Xác định phần đệ quy (n >= 2)

N = 2: max của mảng a là max(a[0] a[1])

N = 3: max của mảng a là max(max(a[0],a[1]), a[2])

N = 4: max của mảng a là max(max(max(a[0], a[1]), a[2]), a[3])

…

…

…

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

int maxNumber(int a, int b) {

return a > b ? a : b;

}

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

if(n == 1)

return arr[0]

return maxNumber(maxArray(arr, n - 1), arr[n – 1])

}

Phần 2: Giải thuật khử đệ quy

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

    int max = arr[0];

    for(int i = 1; i < n; i++) {

        if(max < arr[i])

            max = arr[i];

    }

    return max;

}

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

Ví dụ: n = 29084952106 => kq = 11

Phần 1: Giải thuật đệ quy

Phân tích bài toán:

B1: Thông số hóa 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 thì n chỉ có một chữ số, trả về 1

B3: Xác định phần đệ quy (n >= 10)

Nếu n >= 10 thì gọi lại đệ quy với giá trị đâu vào là n / 10 và tăng kết quả lên 1 cho mỗi lần đệ quy

Giải thuật:

int numberOfN(int n) {

if(n < 10) return 1;

return 1 + numberOfN(n / 10);

}

Phần 2: Giải thuật khử đệ quy

#include <stack>

int numberOfN(int n) {

std::stack <int> Stack;

int count = 1;

while(n >= 10) {

Stack.push(n % 10)

n = n / 10;

}

while(!Stack.empty()) {

Stack.pop()

count ++;

}

return count;

}

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

Phân tích bài toán:

Sử dụng phương pháp đệ quy quay lui

Để xác nhận một phần tử chỉ được dùng một lần ta sẽ dùng mảng flag để đánh dấu. Nếu phần tử chưa sử dụng thì sẽ có giá trị là 0, ngược lại là 1. Ban đầu ta khởi tạo tất cả các phần tử của mảng flag đều có giá trị là 0.

Ý tưởng của phương pháp quay lui là ta sẽ chọn ra một phần tử chưa sử dụng. Lưu phần tử đó vào cấu hình tổ hợp, sau đó đánh dấu nó đã sử dụng. Ta sẽ lặp lại công việc như trên đến khi đủ cấu hình cho một tổ hợp thì sẽ xuất ra màn hình. Sau khi xuất ra ta lại quay trở lại bước trước đó để đánh dấu là nó chưa được chọn

B1: Thông số hóa bài toán

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

B2: Xác định phần cơ bản (phần neo):

Nến số lần thử quay lui k bằng n (số lượng phần tử của mảng) thì xuất ra một cấu hình hoàn chỉnh

B3: Nếu cấu hình chưa hoàn chỉnh, tiếp tục gọi đệ quy quay lui cho đến khi tìm được cấu hình hoàn chỉnh

Giải thuật:

#include <iostream>

int arr[] = {1, 2, 3};

int size = sizeof(arr) / sizeof(arr[0]);

int \*flag = new int [size];

int \*brr = new int [size];

void printArray(int crr[]) {

    for(int i = 0; i < size; i++) {

        std::cout << crr[i] << " ";

    }

    std::cout << std::endl;

}

void swapArray(int k) {

    for(int i = 0; i < size; i++) {

        if(!flag[i]) {

            brr[k] = arr[i];

            flag[i] = 1;

            if(k == size - 1)

                printArray(brr);

            else

                swapArray(k + 1);

            flag[i] = 0;

        }

    }

}

int main() {

    // Reset flag

    for(int i = 0; i < size; i++) {

        flag[i] = 0;

    }

    swapArray(0);

}

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

Phần 1: Giải thuật đệ quy

Phân tích bài toán

B1: Thông số hóa bài toán

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

B2: Xác định phần cơ bản (phần neo):

Nếu n <= 1 in ra n % 2

B3: Xác định phần đệ quy:

Nếu n > 1, gọi đệ quy với đối số truyền vào là n / 2

#include <iostream>

void numToBinary(int n) {

    if(n > 1) {

        numToBinary(n / 2);

    }

    std::cout << n % 2;

}

int main() {

    int n = 123;

    numToBinary(n);

}

Phần 2: Khử đệ quy

#include <stack>

void numToBinary(int n) {

    std::stack <int> Stack;

    while(n > 1) {

        Stack.push(n % 2);

        n = n /2;

    }

    while(!Stack.empty()) {

        std::cout << Stack.top();

        Stack.pop();

    }

}

// Khu de quy bang vong lap

function f(n) {

    if(n === n0) return C

    return (g(n, f(n - 1)))

}

function f(n) {

    k = n0

    F = C

    while(k < n) {

        k ++

        F = g(k, F);

    }

    return F

}

// Khu de quy duoi

function P(X) {

    if (B(X)) D(X)

    else {

        A(X)

        P(f(X))

    }

}

function P(X) {

    while(!B(X)) {

        A(X)

        X = f(X)

    }

    D(X)

}

// Khu de quy dung stack, 1 lan goi de quy

function P(X) {

    if (C(X)) D(X)

    else {

        A(X)

        P(f(X))

        B(X)

    }

}

function P(X) {

    createStack(Stack)

    while(!C(X)) {

        A(X)

        Stack.push(X)

        X => F(X)

    }

    D(X)

    while(!Stack.empty()) {

        Stack.pop()

        B(X)

    }

}

// Khu de quy dung stack, 2 lan goi de quy

function P(X) {

    if(C(X)) D(X)

    else {

        A(X); P(f(X));

        B(X); P(g(X));

    }

}

function P(X) {

}