TRƯỜNG ĐẠI HỌC SÀI GÒN

**KHOA CÔNG NGHỆ THÔNG TIN**

Báo cáo học phần Cấu trúc dữ liệu và giải thuật

**BTTH 4: NGĂN XẾP VÀ HÀNG ĐỢI**

|  |  |
| --- | --- |
| **Sinh viên thực hiện:** | Nguyễn Hoài Quân MSSV: 3124411249 |

Thành phố Hồ Chí Minh, tháng 03 năm 2025

# 1. Mục tiêu

* Hiểu và áp dụng các thao tác cơ bản trong ngăn xếp và hàng đợi
* Áp dụng ngăn xếp và hàng đợi trong bài toán thực tế

# 2. Bài tập thực hành

**Bài 1: Ngăn xếp số nguyên**

(a) Cài đặt CTDL StackInt và LinkedStackInt dùng để chứa các số nguyên, trong đó:

* Sử dụng mảng (StackInt) và danh sách liên kết (LinkedStackInt)
* Cài đặt các thao tác: InitStack, IsEmpty, IsFull, PopStack, PushStack, PeekStack, Clear

(b) Ứng dụng ngăn xếp để đảo số,

(c) Ứng dụng ngăn xếp để kiểm tra xâu đối xứng (mở rộng sang StackString),

(d) Ứng dụng ngăn xếp để đổi từ số thập phân sang nhị phân,

(e) Nhập một biểu thức từ bàn phím, hãy chuyển sang dạng hậu tố và tính giá trị biểu thức.

#include <iostream>

#include <stack>

#include <string>

using namespace std;

// ===================== Nguyên mẫu hàm =====================

class StackInt {

private:

    static const int MAX\_SIZE = 100; // Kích thước tối đa

    int arr[MAX\_SIZE];               // Mảng lưu dữ liệu

    int top;                         // Vị trí phần tử đầu

public:

    StackInt();                      // Khởi tạo stack

    bool IsEmpty();                   // Kiểm tra rỗng

    bool IsFull();                    // Kiểm tra đầy

    void PushStack(int value);        // Thêm phần tử

    int PopStack();                   // Lấy phần tử ra

    int PeekStack();                  // Xem phần tử đầu

    void Clear();                     // Xóa toàn bộ stack

};

// ===================== Ngăn xếp danh sách liên kết =====================

struct Node {

    int data;

    Node\* next;

};

class LinkedStackInt {

private:

    Node\* top; // Con trỏ đầu danh sách

public:

    LinkedStackInt();       // Khởi tạo

    bool IsEmpty();         // Kiểm tra rỗng

    void PushStack(int val); // Thêm phần tử

    int PopStack();         // Lấy phần tử ra

    int PeekStack();        // Xem phần tử đầu

    void Clear();           // Xóa toàn bộ stack

};

// ===================== Ứng dụng ngăn xếp =====================

int ReverseNumber(int num);      // Đảo số nguyên

bool IsPalindrome(string str);   // Kiểm tra xâu đối xứng

string DecimalToBinary(int num); // Chuyển thập phân → nhị phân

string InfixToPostfix(string expr); // Chuyển trung tố → hậu tố

int EvaluatePostfix(string expr);   // Tính giá trị hậu tố

// ===================== Hàm main =====================

int main() {

    int num = 12345;

    cout << "So dao nguoc: " << ReverseNumber(num) << endl;

    string str = "madam";

    cout << "Xau \"" << str << "\" doi xung? " << (IsPalindrome(str) ? "Co" : "Khong") << endl;

    int decimal = 42;

    cout << "So nhi phan cua " << decimal << " la: " << DecimalToBinary(decimal) << endl;

    string infix = "3+5\*2";

    string postfix = InfixToPostfix(infix);

    cout << "Bieu thuc hau to: " << postfix << endl;

    cout << "Gia tri bieu thuc: " << EvaluatePostfix(postfix) << endl;

    return 0;

}

// ===================== Định nghĩa hàm StackInt =====================

StackInt::StackInt() {

    top = -1; // Khởi tạo rỗng

}

bool StackInt::IsEmpty() {

    return top == -1;

}

bool StackInt::IsFull() {

    return top == MAX\_SIZE - 1;

}

void StackInt::PushStack(int value) {

    if (IsFull()) {

        cout << "Stack day!" << endl;

        return;

    }

    arr[++top] = value;

}

int StackInt::PopStack() {

    if (IsEmpty()) {

        cout << "Stack rong!" << endl;

        return -1;

    }

    return arr[top--];

}

int StackInt::PeekStack() {

    if (IsEmpty()) {

        cout << "Stack rong!" << endl;

        return -1;

    }

    return arr[top];

}

void StackInt::Clear() {

    top = -1; // Đặt lại trạng thái rỗng

}

// ===================== Định nghĩa LinkedStackInt =====================

LinkedStackInt::LinkedStackInt() {

    top = nullptr;

}

bool LinkedStackInt::IsEmpty() {

    return top == nullptr;

}

void LinkedStackInt::PushStack(int val) {

    Node\* newNode = new Node{val, top};

    top = newNode;

}

int LinkedStackInt::PopStack() {

    if (IsEmpty()) {

        cout << "Stack rong!" << endl;

        return -1;

    }

    int value = top->data;

    Node\* temp = top;

    top = top->next;

    delete temp;

    return value;

}

int LinkedStackInt::PeekStack() {

    if (IsEmpty()) {

        cout << "Stack rong!" << endl;

        return -1;

    }

    return top->data;

}

void LinkedStackInt::Clear() {

    while (!IsEmpty()) {

        PopStack();

    }

}

// ===================== Định nghĩa ứng dụng =====================

int ReverseNumber(int num) {

    StackInt stack;

    while (num > 0) {

        stack.PushStack(num % 10);

        num /= 10;

    }

    int reversed = 0;

    while (!stack.IsEmpty()) {

        reversed = reversed \* 10 + stack.PopStack();

    }

    return reversed;

}

bool IsPalindrome(string str) {

    stack<char> stack;

    for (char ch : str) {

        stack.push(ch);

    }

    for (char ch : str) {

        if (stack.top() != ch) return false;

        stack.pop();

    }

    return true;

}

string DecimalToBinary(int num) {

    stack<int> stack;

    while (num > 0) {

        stack.push(num % 2);

        num /= 2;

    }

    string binary = "";

    while (!stack.empty()) {

        binary += to\_string(stack.top());

        stack.pop();

    }

    return binary;

}

int GetPriority(char op) {

    if (op == '+' || op == '-') return 1;

    if (op == '\*' || op == '/') return 2;

    return 0;

}

string InfixToPostfix(string expr) {

    stack<char> stack;

    string postfix = "";

    for (char ch : expr) {

        if (isdigit(ch)) {

            postfix += ch;

        } else {

            while (!stack.empty() && GetPriority(stack.top()) >= GetPriority(ch)) {

                postfix += stack.top();

                stack.pop();

            }

            stack.push(ch);

        }

    }

    while (!stack.empty()) {

        postfix += stack.top();

        stack.pop();

    }

    return postfix;

}

int EvaluatePostfix(string expr) {

    stack<int> stack;

    for (char ch : expr) {

        if (isdigit(ch)) {

            stack.push(ch - '0');

        } else {

            int val2 = stack.top(); stack.pop();

            int val1 = stack.top(); stack.pop();

            switch (ch) {

                case '+': stack.push(val1 + val2); break;

                case '-': stack.push(val1 - val2); break;

                case '\*': stack.push(val1 \* val2); break;

                case '/': stack.push(val1 / val2); break;

            }

        }

    }

    return stack.top();

}

**Bài 2: Hàng đợi số nguyên**

(a) Cài đặt CTDL QueueInt và LinkedQueueInt dùng để chứa các số nguyên, trong đó:

• Sử dụng mảng (StackInt) và danh sách liên kết (LinkedStackInt)

• Cài đặt các thao tác: InitStack, IsEmpty, IsFull, PopStack, PushStack, PeekStack, Clear

(b) Ứng dụng hàng đợi để làm bài toán xếp lịch cặp múa nam/nữ (như trong bài giảng: Sàn nhảy chỉ có 4 cặp nam – nữ nhảy đồng thời; Có hai hàng: một hàng nam và một hàng nữ; Từng cặp nam – nữ của đầu mỗi hàng sẽ đi vào sàn nhảy cho đến khi đủ số lượng cặp nam – nữ trên sàn nhảy; Nếu không đủ một cặp nam – nữ thì một diễn viên phải chờ).

(c) Ứng dụng hàng đợi để cài thuật toán RadixSort

#include <iostream>

#include <queue>

using namespace std;

// ===================== Nguyên mẫu hàm =====================

class QueueInt {

private:

    static const int MAX\_SIZE = 100;

    int arr[MAX\_SIZE];

    int front, rear, count;

public:

    QueueInt();

    bool IsEmpty();

    bool IsFull();

    void PushQueue(int value);

    int PopQueue();

    int PeekQueue();

    void Clear();

};

// ===================== Hàng đợi danh sách liên kết =====================

struct Node {

    int data;

    Node\* next;

};

class LinkedQueueInt {

private:

    Node\* front;

    Node\* rear;

public:

    LinkedQueueInt();

    bool IsEmpty();

    void PushQueue(int value);

    int PopQueue();

    int PeekQueue();

    void Clear();

};

// ===================== Ứng dụng hàng đợi =====================

void ScheduleDanceQueue(queue<string>& queueNam, queue<string>& queueNu);

void RadixSort(int arr[], int n);

// ===================== Hàm main =====================

int main() {

    queue<string> queueNam, queueNu;

    queueNam.push("Anh"); queueNam.push("Bình"); queueNam.push("Châu");

    queueNu.push("Dương"); queueNu.push("Hà");

    ScheduleDanceQueue(queueNam, queueNu);

    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};

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

    cout << "Mang truoc khi sap xep: ";

    for (int i = 0; i < n; i++) cout << arr[i] << " ";

    cout << endl;

    RadixSort(arr, n);

    cout << "Mang sau khi sap xep: ";

    for (int i = 0; i < n; i++) cout << arr[i] << " ";

    cout << endl;

    return 0;

}

// ===================== Định nghĩa QueueInt =====================

QueueInt::QueueInt() {

    front = 0;

    rear = -1;

    count = 0;

}

bool QueueInt::IsEmpty() {

    return count == 0;

}

bool QueueInt::IsFull() {

    return count == MAX\_SIZE;

}

void QueueInt::PushQueue(int value) {

    if (IsFull()) {

        cout << "Queue day!" << endl;

        return;

    }

    rear = (rear + 1) % MAX\_SIZE;

    arr[rear] = value;

    count++;

}

int QueueInt::PopQueue() {

    if (IsEmpty()) {

        cout << "Queue rong!" << endl;

        return -1;

    }

    int value = arr[front];

    front = (front + 1) % MAX\_SIZE;

    count--;

    return value;

}

int QueueInt::PeekQueue() {

    if (IsEmpty()) {

        cout << "Queue rong!" << endl;

        return -1;

    }

    return arr[front];

}

void QueueInt::Clear() {

    front = 0;

    rear = -1;

    count = 0;

}

// ===================== Định nghĩa LinkedQueueInt =====================

LinkedQueueInt::LinkedQueueInt() {

    front = rear = nullptr;

}

bool LinkedQueueInt::IsEmpty() {

    return front == nullptr;

}

void LinkedQueueInt::PushQueue(int value) {

    Node\* newNode = new Node{value, nullptr};

    if (IsEmpty()) {

        front = rear = newNode;

    } else {

        rear->next = newNode;

        rear = newNode;

    }

}

int LinkedQueueInt::PopQueue() {

    if (IsEmpty()) {

        cout << "Queue rong!" << endl;

        return -1;

    }

    int value = front->data;

    Node\* temp = front;

    front = front->next;

    delete temp;

    if (front == nullptr) rear = nullptr;

    return value;

}

int LinkedQueueInt::PeekQueue() {

    if (IsEmpty()) {

        cout << "Queue rong!" << endl;

        return -1;

    }

    return front->data;

}

void LinkedQueueInt::Clear() {

    while (!IsEmpty()) {

        PopQueue();

    }

}

// ===================== Ứng dụng: Xếp cặp nhảy =====================

void ScheduleDanceQueue(queue<string>& queueNam, queue<string>& queueNu) {

    const int MAX\_COUPLES = 4;

    int couples = 0;

    cout << "💃🕺 Danh sach ghep cap:" << endl;

    while (!queueNam.empty() && !queueNu.empty() && couples < MAX\_COUPLES) {

        cout << "- " << queueNam.front() << " nhay voi " << queueNu.front() << endl;

        queueNam.pop();

        queueNu.pop();

        couples++;

    }

    if (queueNam.empty() && queueNu.empty()) {

        cout << "✅ Tat ca da co doi!" << endl;

    } else if (!queueNam.empty()) {

        cout << "🕺 " << queueNam.front() << " dang cho!" << endl;

    } else {

        cout << "💃 " << queueNu.front() << " dang cho!" << endl;

    }

}

// ===================== Ứng dụng: Radix Sort =====================

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

    int maxVal = arr[0];

    for (int i = 1; i < n; i++) {

        if (arr[i] > maxVal) maxVal = arr[i];

    }

    return maxVal;

}

void CountingSort(int arr[], int n, int exp) {

    queue<int> buckets[10]; // Mảng 10 hàng đợi

    // Đưa số vào hàng đợi theo chữ số hàng đơn vị, chục, trăm...

    for (int i = 0; i < n; i++) {

        int digit = (arr[i] / exp) % 10;

        buckets[digit].push(arr[i]);

    }

    // Ghép lại mảng sau khi gom vào bucket

    int index = 0;

    for (int i = 0; i < 10; i++) {

        while (!buckets[i].empty()) {

            arr[index++] = buckets[i].front();

            buckets[i].pop();

        }

    }

}

void RadixSort(int arr[], int n) {

    int maxVal = GetMax(arr, n);

    for (int exp = 1; maxVal / exp > 0; exp \*= 10) {

        CountingSort(arr, n, exp);

    }

}

# 3. Bài tập mở rộng

**Bài 1: Dùng stack khử đệ qui**

(a) Viết chương trình đệ qui / khử đệ qui (dùng stack) cho bài toán tính số Fibonancy, đảo ngược số

(b) Viết chương trình đệ qui / khử đệ qui (dùng stack) cho bài toán đảo ngược số

(c) HanoiTower là một tháp có nhiều tầng, tầng nhỏ nằm trên tầng lớn. Sau đó viết phương thức di chuyển tháp này từ vị trí 1 đến vị trí 3 thông qua vị trí trung gian 2; mỗi lần chỉ được di chuyển tầng trên cùng của tháp và tại mỗi vị trí 1, 2, và 3 đều là tháp. Hãy viết chương trình đệ qui / khử đệ qui cho bài toán tháp Hà Nội.

#include <iostream>

#include <stack>

using namespace std;

// --- Khung gọi hàm cho Fibonacci ---

struct FibFrame {

    int n;      // Số cần tính fib(n)

    int state;  // 0: cần tính fib(n-1), 1: đã tính fib(n-1), 2: sẵn sàng gộp kết quả

    int ret;    // Lưu fib(n-1)

};

int FibNonRecursive(int n);

// --- Hàm đảo ngược số bằng stack ---

int ReverseNumberStack(int num);

// --- Khung gọi hàm cho Tháp Hà Nội ---

struct HanoiFrame {

    int n;      // Số đĩa cần di chuyển

    char from;  // Cột xuất phát

    char to;    // Cột đích

    char aux;   // Cột trung gian

    int state;  // 0: chưa giải n-1 từ from->aux; 1: đã di chuyển đĩa lớn; 2: chưa giải n-1 từ aux->to

};

void HanoiNonRecursive(int n, char from, char to, char aux);

int main() {

    // Tính Fibonacci không đệ qui

    int nFib = 10;

    cout << "Fibonacci(" << nFib << ") = " << FibNonRecursive(nFib) << endl;

    // Đảo ngược số

    int num = 12345;

    cout << "So ban dau: " << num << "\nSo dao nguoc: " << ReverseNumberStack(num) << endl;

    // Tháp Hà Nội không đệ qui

    int nHanoi = 3;

    cout << "\nCac buoc di chuyen Thap Ha Noi (voi " << nHanoi << " dia):" << endl;

    HanoiNonRecursive(nHanoi, 'A', 'C', 'B');

    return 0;

}

// --- Fibonacci không đệ qui (mô phỏng đệ qui bằng stack) ---

int FibNonRecursive(int n) {

    stack<FibFrame> s;

    s.push({n, 0, 0});

    int result = 0;

    while (!s.empty()) {

        FibFrame top = s.top(); s.pop();

        if (top.n == 0) result = 0;

        else if (top.n == 1) result = 1;

        else if (top.state == 0) {  // Tính fib(n-1)

            top.state = 1; s.push(top);

            s.push({top.n - 1, 0, 0});

            continue;

        } else if (top.state == 1) {  // Đã có fib(n-1), tính fib(n-2)

            top.ret = result;

            top.state = 2; s.push(top);

            s.push({top.n - 2, 0, 0});

            continue;

        } else { // state == 2, kết hợp fib(n-1) và fib(n-2)

            result = top.ret + result;

        }

    }

    return result;

}

// --- Đảo ngược số bằng stack ---

int ReverseNumberStack(int num) {

    stack<int> s;

    while (num > 0) {

        s.push(num % 10);

        num /= 10;

    }

    int rev = 0;

    while (!s.empty()) {

        rev = rev \* 10 + s.top();

        s.pop();

    }

    return rev;

}

// --- Tháp Hà Nội không đệ qui (mô phỏng đệ qui bằng stack) ---

void HanoiNonRecursive(int n, char from, char to, char aux) {

    stack<HanoiFrame> s;

    s.push({n, from, to, aux, 0});

    while (!s.empty()) {

        HanoiFrame frame = s.top(); s.pop();

        if (frame.n == 1) {

            cout << "Di chuyen dia 1 tu " << frame.from << " den " << frame.to << endl;

            continue;

        }

        if (frame.state == 0) {  // Chưa giải bài toán con đầu tiên

            frame.state = 1; s.push(frame);

            s.push({frame.n - 1, frame.from, frame.aux, frame.to, 0});

        } else if (frame.state == 1) { // Đã giải xong n-1 từ from->aux

            cout << "Di chuyen dia " << frame.n << " tu " << frame.from << " den " << frame.to << endl;

            frame.state = 2; s.push(frame);

            s.push({frame.n - 1, frame.aux, frame.to, frame.from, 0});

        }

    }

}

**Bài 2: Mê Cung (MECUNG.\*)**

Cho mê cung kích thước N dòng và M cột. Ô (i, j) có giá trị 0 (không có chướng ngại), 1 (có chướng ngại). Một người xuất phát từ ô (x0, y0) và di chuyển qua trái, qua phải, trên và dưới qua các ô không có chướng ngại. Hỏi người đó có đi tới được ô (x1, y1)? Nếu có, chỉ đường đi từ (x0,y0) đến (x1,y1). (Áp dụng BFS – hàng đợi và DFS – ngăn xếp).

Dữ liệu:

* Dòng đầu chứa N M x0 y0 x1 y1, ứng với số dòng, số cột mê cung, điểm xuất phát, điểm cần đi tới.
* N dòng tiếp, mỗi dòng có M số chứa 0 (không có chướng ngại) hoặc 1 (có chướng ngại).

Kết quả:

* Dòng đầu tiên ghi số 0 hoặc K với 0 ứng với không có lộ trình, K ứng với có lộ trình và đi qua K ô.
* Nếu có lộ trình, K dòng, mỗi dòng ghi u v, trong đó (u, v) là 1 ô trong lộ trình từ (x0, y0) đến (x1, y1).

Ví dụ:

|  |  |
| --- | --- |
| **Dữ liệu** | **Kết quả** |
| 4 5 2 2 1 6  0 1 0 0 1 0  0 0 0 0 0 0  0 0 1 0 0 0  0 1 0 1 0 0 | 6  2 2  2 3  2 4  2 5  2 6  1 6 |

#include <iostream>

#include <queue>

#include <stack>

using namespace std;

const int MAX = 105; // Giới hạn tối đa của số dòng và cột

// Cấu trúc lưu tọa độ của một ô trong mê cung

struct Cell {

    int r, c;

};

// --- Hàm BFS: tìm đường đi ngắn nhất bằng hàng đợi ---

// parent lưu ô cha để truy vết đường đi

bool bfsMaze(int N, int M, int startX, int startY, int endX, int endY,

             int maze[][MAX], Cell parent[][MAX]) {

    bool visited[MAX][MAX] = {false};

    queue<Cell> q;

    int dr[4] = {-1, 1, 0, 0}; // Di chuyển: lên, xuống, trái, phải

    int dc[4] = {0, 0, -1, 1};

    q.push({startX, startY});

    visited[startX][startY] = true;

    parent[startX][startY] = {-1, -1}; // Điểm gốc không có cha

    while (!q.empty()) {

        Cell cur = q.front();

        q.pop();

        if (cur.r == endX && cur.c == endY)

            return true;

        for (int i = 0; i < 4; i++) {

            int nr = cur.r + dr[i], nc = cur.c + dc[i];

            // Kiểm tra biên, chướng ngại và ô đã thăm

            if (nr >= 1 && nr <= N && nc >= 1 && nc <= M &&

                maze[nr][nc] == 0 && !visited[nr][nc]) {

                visited[nr][nc] = true;

                parent[nr][nc] = cur;

                q.push({nr, nc});

            }

        }

    }

    return false;

}

// --- Hàm DFS: tìm đường đi (không đảm bảo ngắn nhất) dùng ngăn xếp ---

bool dfsMaze(int N, int M, int startX, int startY, int endX, int endY,

             int maze[][MAX], Cell parent[][MAX]) {

    bool visited[MAX][MAX] = {false};

    stack<Cell> st;

    int dr[4] = {-1, 1, 0, 0};

    int dc[4] = {0, 0, -1, 1};

    st.push({startX, startY});

    visited[startX][startY] = true;

    parent[startX][startY] = {-1, -1};

    while (!st.empty()) {

        Cell cur = st.top();

        st.pop();

        if (cur.r == endX && cur.c == endY)

            return true;

        for (int i = 0; i < 4; i++) {

            int nr = cur.r + dr[i], nc = cur.c + dc[i];

            if (nr >= 1 && nr <= N && nc >= 1 && nc <= M &&

                maze[nr][nc] == 0 && !visited[nr][nc]) {

                visited[nr][nc] = true;

                parent[nr][nc] = cur;

                st.push({nr, nc});

            }

        }

    }

    return false;

}

int main() {

    int N, M, startX, startY, endX, endY;

    // Đọc dữ liệu: N dòng, M cột, tọa độ xuất phát và đích

    cin >> N >> M >> startX >> startY >> endX >> endY;

    int maze[MAX][MAX];

    for (int i = 1; i <= N; i++) {

        for (int j = 1; j <= M; j++) {

            cin >> maze[i][j];

        }

    }

    // Chọn thuật toán: true -> BFS, false -> DFS

    bool useBFS = true;

    Cell parent[MAX][MAX];

    bool found = false;

    if (useBFS)

        found = bfsMaze(N, M, startX, startY, endX, endY, maze, parent);

    else

        found = dfsMaze(N, M, startX, startY, endX, endY, maze, parent);

    if (!found) {

        cout << "0" << endl; // Không tìm thấy lộ trình

    } else {

        // Truy vết đường đi từ đích về gốc

        Cell path[MAX \* MAX];

        int count = 0;

        int r = endX, c = endY;

        while (!(r == -1 && c == -1)) {

            path[count++] = {r, c};

            Cell par = parent[r][c];

            r = par.r; c = par.c;

        }

        // In kết quả theo thứ tự từ điểm xuất phát đến đích

        cout << count << endl;

        for (int i = count - 1; i >= 0; i--) {

            cout << path[i].r << " " << path[i].c << endl;

        }

    }

    return 0;

}