Trường Đại học Sài Gòn

**Khoa công nghệ thông tin**

A blue circle with purple letters

Description automatically generated

**CTDLGT25 - 04 - Bai tap ngan xep va hang doi**

Sinh viên thực hiện: Bùi Viết Quang Vinh

MSSV:3124411347

Thành phố Hồ Chí Minh, tháng 3 năm 2025

**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>

#define MAX 100

using namespace std;

// Cấu trúc Stack dùng mảng

int top;

int arr[MAX];

// Khởi tạo Stack

void Init() {

    top = -1;

}

// Kiểm tra Stack rỗng

bool IsEmpty() {

    return (top == -1);

}

// Kiểm tra Stack đầy

bool IsFull() {

    return (top == MAX - 1);

}

// Thêm phần tử vào Stack

void Push(int *value*) {

    if (IsFull()) {

        cout << "Stack đầy!" << endl;

        return;

    }

    arr[++top] = value;

}

// Xóa phần tử khỏi Stack

int Pop() {

    if (IsEmpty()) {

        cout << "Stack rỗng!" << endl;

        return -1;

    }

    return arr[top--];

}

// Lấy phần tử đầu Stack mà không xóa

int Peek() {

    if (IsEmpty()) {

        cout << "Stack rỗng!" << endl;

        return -1;

    }

    return arr[top];

}

// Xóa toàn bộ Stack

void Clear() {

    top = -1;

}

// (b) Ứng dụng đảo số

int Reverse(int *num*) {

    Init();

    while (num != 0) {

        Push(num % 10);

        num /= 10;

    }

    int reversed = 0;

    int factor = 1;

    while (!IsEmpty()) {

        reversed += Pop() \* factor;

        factor \*= 10;

    }

    return reversed;

}

// (c) Kiểm tra chuỗi đối xứng

bool IsPalindrome(const *string* &*str*) {

    stack<char> s;

    for (char ch : str) {

        s.push(ch);

    }

    for (char ch : str) {

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

        s.pop();

    }

    return true;

}

// (d) Chuyển số thập phân sang nhị phân

void DecimalToBinary(int *num*) {

    Init();

    while (num > 0) {

        Push(num % 2);

        num /= 2;

    }

    while (!IsEmpty()) {

        cout << Pop();

    }

    cout << endl;

}

// (e) Chuyển trung tố sang hậu tố và tính toán biểu thức hậu tố

float Evaluate(char \**Postfix*) {

    stack<float> S;

    const char \*p = Postfix;

    while (\*p != '\0') {

        while (\*p == ' ' || \*p == '\t') p++;

        if (isdigit(\*p)) {

            float num = 0;

            while (isdigit(\*p)) {

                num = num \* 10 + (\*p - '0');

                p++;

            }

            S.push(num);

        } else {

            if (S.size() < 2) {

                cout << "Không đủ toán hạng!" << endl;

                return 0;

            }

            float op1 = S.top(); S.pop();

            float op2 = S.top(); S.pop();

            float result = 0;

            switch (\*p) {

                case '+': result = op2 + op1; break;

                case '-': result = op2 - op1; break;

                case '\*': result = op2 \* op1; break;

                case '/':

                    if (op1 == 0) {

                        cout << "Lỗi chia cho 0!" << endl;

                        return 0;

                    }

                    result = op2 / op1; break;

                default:

                    cout << "Toán tử không hợp lệ!" << endl;

                    return 0;

            }

            S.push(result);

        }

        p++;

    }

    return S.top();

}

int main() {

    Init();

    cout << " Câu b : Đảo số của 12345: " << Reverse(12345) << endl;

    cout << " Câu c : Kiểm tra 'madam' có đối xứng không: " << (IsPalindrome("madam") ? "Có" : "Không") << endl;

    cout << " Câu d : Số 10 đổi sang nhị phân: "; DecimalToBinary(10);

    char postfix[] = "23 34 + 2 \*";

    cout << " Câu e : Giá trị của biểu thức hậu tố: " << Evaluate(postfix) << endl;

    return 0;

}

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),

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

**\*\*\*Dùng LinkedQueueInt**

#include<iostream>

using namespace std;

// Khoi tao danh sach lien ket

struct *Node* {

    int data; // du lieu cua node

*Node*\* next; // con tro tro den node tiep theo

};

// Khoi tao 2 nut dau va cuoi

*Node*\* front = NULL;

*Node*\* rear = NULL;

// Khoi tao hang doi rong

void Init() {

    front = NULL;

    rear = NULL;

}

// Kiem tra danh sach rong

bool IsEmpty() {

    return (front == NULL && rear == NULL);

}

// Them phan tu vao hang doi

void PushStack(int *value*) {

*Node*\* temp = **new** *Node*; // khoi tao node moi

    temp->data = *value*; // gan gia tri cho node

    temp->next = NULL; // Nut moi khong tro den nut nao

    // Neu hang doi rong

    if (IsEmpty()) {

        front = rear = temp; // front va rear cung tro node moi

    } else {

        rear->next = temp; // rear tro den node moi

        rear = temp; // rear tro den node moi

    }

}

// Lay phan tu khoi hang doi

void PopStack() {

    if (IsEmpty()) {

        cout << "Hang doi rong\n";

        return;

    }

    // Luu tru node dau de xoa

*Node*\* temp = front;

    // Chuyen front den node tiep theo

    front = front->next;

    // xoa node dau

**delete** temp;

    if (front == nullptr) {

        rear = nullptr; // Neu hang doi rong, dat rear ve nullptr

    }

}

// Lay phan tu dau hang doi ma khong xoa

int PeekStack() {

    if (!IsEmpty()) {

        return front->data; // Tra ve du lieu cua nut dau

    }

    cout << "Queue is empty\n";

    return -1;

}

// Xoa hang doi

void Clear() {

    while (!IsEmpty()) {

        PopStack(); // xoa phan tu cho den khi hang doi rong

    }

}

// Test case

int main() {

    Init();

    cout << "Them cac phan tu vao hang doi: 10, 20, 30\n";

    PushStack(10);

    PushStack(20);

    PushStack(30);

    cout << "Phan tu o dau hang doi: " << PeekStack() << "\n";

    cout << "Xoa mot phan tu khoi hang doi\n";

    PopStack();

    cout << "Phan tu o dau hang doi sau khi xoa: " << PeekStack() << "\n";

    cout << "Xoa toan bo hang doi\n";

    Clear();

    cout << "Kiem tra hang doi co rong khong: " << (IsEmpty() ? "Co" : "Khong") << "\n";

    return 0;

}

**\*\*\*Dùng StackInt**

//Cài bằng stackinit

#include<iostream>

#include<queue>

#define MAX 100

using namespace std;

int front =-1;

int rear =-1;

int arr[MAX];

//Tao queue

void Init()

{

    front=rear=-1; //Khơi tạo hàng đợi rỗng

}

//Kiem tra hang doi co bi day khong

bool IsFull()

{

    return rear==MAX-1;

}

//Kiem tra hang doi co bi rong khong

bool IsEmpty()

{

    return (front==-1 && rear==-1);

}

//Them phan tu vao hang doi

void Push(int *value*)

{

    if(IsFull())

    {

        cout<<" Hang doi day";

        return ;

    }

    //Neu hang doi rong thi front va rear se cung 0

    if(IsEmpty())

    {

        front=rear=0;

    }

    else

    {

        rear++;

    }

    arr[rear]=*value*;

}

//lay phan tu khoi hang doi

int Pop()

{

    if(IsEmpty())

    {

        cout<<"Hang doi rong";

        return -1;

    }

    if (front == rear) {

        front = rear = -1; // Nếu chỉ có một phần tử, đặt hàng đợi về rỗng

    } else {

        front++; // Tăng chỉ số đầu của hàng đợi

    }

}

//Lay phan tu dau hang doi ma khong xoa

int PeekStack() {

    if (!IsEmpty()) {

        return arr[front]; // Trả về phần tử đầu của hàng đợi

    }

    cout << "Queue is empty\n";

    return -1;

}

//Xoa toan bo hang doi

void Clear() {

    front = rear = -1; // Xóa toàn bộ hàng đợi

}

int main()

{

    return 0;

}

**\*\*\*Bài 2 b:**

#include<iostream>

#include<queue>

#include<string.h>

using namespace std;

struct *Node*

{

    char data[50];

*Node*\* next;

};

*Node*\* front =NULL;

struct *LinkedQueueInt* {

*Node*\* front; // Con trỏ đầu của hàng đợi

*Node*\* rear; // Con trỏ cuối của hàng đợi

};

// Khởi tạo hàng đợi rỗng

void Init(*LinkedQueueInt* &*queue*){

*queue*.front =NULL;

*queue*.rear =NULL;

}

//Kiem tra hang doi rong

bool IsEmpty(*LinkedQueueInt* &*queue*)

{

    return (*queue*.front ==NULL && *queue*.rear ==NULL);

}

//Them  vao hang doi

void Enqueue(*LinkedQueueInt* &*queue* ,const char\* *name*)

{

*Node*\*temp=**new** *Node*;//Khoi tao node moi

    strcpy(temp->data,*name*);//Gan gia tri cho node

    temp->next=NULL;//Nut moi khong tro den nut nao

    if(IsEmpty(*queue*))

    {

*queue*.front=*queue*.rear=temp;//front va rear cung tro den node moi

    }

    else

    {

*queue*.rear->next=temp;//rear tro den node moi

*queue*.rear=temp;//dat rear ve node moi

    }

}

//Loai bo phan tu khoi hang doi

void Dequeue(*LinkedQueueInt* &*queue*)

{

    if(IsEmpty(*queue*))

    {

        cout<<"hang doi rong";

        return;

    }

*Node*\*temp=*queue*.front;//Luu tru node dau de xoa

*queue*.front=*queue*.front->next;//dat front lam node tiep theo

**delete** temp;//xoa node dau

    // neu hang doi rong

    if(*queue*.front==NULL)

    {

*queue*.rear=NULL;//dat rear ve NULL

    }

}

const char\* PeekQueue(*LinkedQueueInt*& *queue*) {

    if (!IsEmpty(*queue*)) {

        return *queue*.front->data; //Tra ve du lieu ban dau

    }

    return nullptr;

}

/\*  Tích hợp Ai để kiếm testkey\*/

int main() {

*LinkedQueueInt* maleQueue, femaleQueue;

    Init(maleQueue);

    Init(femaleQueue);

    // Thêm các bạn nam và nữ vào hàng đợi

    Enqueue(maleQueue, "Nam 1");

    Enqueue(maleQueue, "Nam 2");

    Enqueue(femaleQueue, "Nữ 1");

    Enqueue(femaleQueue, "Nữ 2");

    // Xếp cặp

    while (!IsEmpty(maleQueue) && !IsEmpty(femaleQueue)) {

        const char\* male = PeekQueue(maleQueue); // Lấy tên nam từ hàng đợi

        Dequeue(maleQueue); // Xóa nam khỏi hàng đợi

        const char\* female = PeekQueue(femaleQueue); // Lấy tên nữ từ hàng đợi

        Dequeue(femaleQueue); // Xóa nữ khỏi hàng đợi

        std::cout << "Cặp: " << male << " và " << female << std::endl; // In ra cặp nam nữ

    }

    return 0;

}

**\*\*\*Bài 2 c:**

#include <iostream>

#include <cmath>

#include <algorithm>

struct *Node* {

    int data; // Dữ liệu của nút

*Node*\* next; // Con trỏ đến nút tiếp theo

};

struct *LinkedQueueInt* {

*Node*\* front; // Con trỏ đầu của hàng đợi

*Node*\* rear; // Con trỏ cuối của hàng đợi

};

void InitQueue(*LinkedQueueInt*& *queue*) {

*queue*.front = *queue*.rear = nullptr; // Khởi tạo hàng đợi rỗng

}

bool IsEmpty(*LinkedQueueInt*& *queue*) {

    return (*queue*.front == nullptr); // Kiểm tra hàng đợi có rỗng không

}

void Enqueue(*LinkedQueueInt*& *queue*, int *x*) {

*Node*\* temp = **new** *Node*(); // Tạo nút mới

    temp->data = *x*; // Gán dữ liệu cho nút

    temp->next = nullptr; // Nút mới không trỏ đến nút nào

    if (IsEmpty(*queue*)) {

*queue*.front = *queue*.rear = temp; // Nếu hàng đợi rỗng, đặt cả front và rear về nút mới

    } else {

*queue*.rear->next = temp; // Nút cuối trỏ đến nút mới

*queue*.rear = temp; // Đặt rear về nút mới

    }

}

int Dequeue(*LinkedQueueInt*& *queue*) {

    if (IsEmpty(*queue*)) {

        std::cout << "Queue is empty\n";

        return -1;

    }

*Node*\* temp = *queue*.front; // Lưu trữ nút đầu để xóa

    int data = temp->data; // Lấy dữ liệu của nút đầu

*queue*.front = *queue*.front->next; // Đặt front về nút tiếp theo

**delete** temp; // Xóa nút đầu

    if (*queue*.front == nullptr) {

*queue*.rear = nullptr; // Nếu hàng đợi rỗng, đặt rear về nullptr

    }

    return data;

}

// Hàm tìm giá trị lớn nhất trong mảng

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;

}

// Hàm RadixSort sử dụng LinkedQueueInt

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

*LinkedQueueInt* buckets[10]; // Mảng các hàng đợi cho các chữ số từ 0 đến 9

    for (int i = 0; i < 10; i++) {

        InitQueue(buckets[i]); // Khởi tạo các hàng đợi

    }

    int maxVal = getMax(*arr*, *n*); // Tìm giá trị lớn nhất trong mảng

    int maxDigits = log10(maxVal) + 1; // Tính số chữ số của giá trị lớn nhất

    for (int digit = 0; digit < maxDigits; digit++) {

        for (int i = 0; i < *n*; i++) {

            int bucketIndex = (*arr*[i] / static\_cast<int>(pow(10, digit))) % 10; // Tính chỉ số hàng đợi

            Enqueue(buckets[bucketIndex], *arr*[i]); // Thêm phần tử vào hàng đợi tương ứng

        }

        int index = 0;

        for (int i = 0; i < 10; i++) {

            while (!IsEmpty(buckets[i])) {

*arr*[index++] = Dequeue(buckets[i]); // Lấy phần tử từ hàng đợi và đặt lại vào mảng

            }

        }

    }

}

int main() {

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

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

    RadixSort(arr, n);

    for (int i = 0; i < n; i++) {

        std::cout << arr[i] << " "; // In ra mảng đã được sắp xếp

    }

    return 0;

}

**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.

**\*\*\*Bài 1 a:**

#include<iostream>

#include<stack>

using namespace std;

//Phien ban dung de quy

int Fibonacci(int *n*)

{

    if(*n*==0) return 0;

    if(*n*==1) return 1;

    return Fibonacci(*n*-1) + Fibonacci(*n*-2);

}

//Phien ban su dung stack

int Fibonacci\_Stack(int *n*)

{

    //truong hop co so

    if(*n*<=1) return *n*;

    stack<int> st;//luu gia tri can tinh

    st.push(*n*);//gia tri ban dau vao stack

    int result=0;//gia tri ket qua

    //duyet cac gia tri trong stack

    while(!st.empty())

    {

        int x=st.top();//lay gia tri tren cung cua stack

        st.pop();//xoa gia tri da lay

        if(x==1)

            result +=1;//Neu x = 1 thi cong them 1 vao ket qua

        else if(x == 0) //Neu x = 0 thi khong lam gi ca

            continue;

        else{

            //DayF(n-1) vao stack

            st.push(x-1);

            st.push(x-2);

        }

    }

    return result;

}

int main()

{

    int n;

    cout<<"Nhap n: ";cin>>n;

    cout<<"Fibonacci(n) de quy = "<<Fibonacci(n)<<endl;

    cout<<"Fibonacci(n) khong de quy = "<<Fibonacci\_Stack(n)<<endl;

    return 0;

}

**\*\*\*Bài 1 b:**

#include<iostream>

#include<stack>

using namespace std;

//Phien ban dung de quy

int Reverse\_recursive(int *num*,int *rev*=0)

{

    //Neu n=0 thi tra ve

    if(*num*==0) return *rev*;

    //Neu n khac 0 thi in ra so cuoi cung cua n

    else  {

*rev* =*rev*\*10 + *num*%10;

        return Reverse\_recursive(*num*/10,*rev*);

    }

}

//Phien ban su dung stack

int Reverse\_stack(int *num*)

{

    stack<int> s;

    //Tach tung so

    while(*num* !=0)

    {

        s.push(*num*%10);//lay chu so cuoi cung

*num* /=10;//loai bo chu so cuoi cung

    }

    int rev=0;//so dao nguoc

    int place=1;//vi tri hang

    while (!s.empty()) {

        rev += s.top() \* place; // Lấy chữ số từ stack

        s.pop();               // Xóa chữ số đã lấy

        place \*= 10;           // Tăng vị trí hàng

    }

    return rev;

}

int main()

{

    int num;

    cout<<"Nhap so can dao nguoc: ";cin>>num;

    cout<<"So dao nguoc de quy: "<<Reverse\_recursive(num)<<endl;

    cout<<"So dao nguoc su dung stack: "<<Reverse\_stack(num)<<endl;

    return 0;

}

**\*\*\*Bài 1 c đệ quy :**

#include<iostream>

using namespace std;

//Su dung de quy

void ThapHN(int *n*,char *a*,char *b*,char *c*)

{

    //Truong hop 1 dia

    if(*n*==1)

        {

            cout<<*a*<<"-->"<<*b*<<endl;

            return ;

        }

//  //Truong hop co 2 dia

//  if(n==2)

//      {

//          cout<<"t"<<a<<"-->"<<b<<endl;

//          cout<<"t"<<a<<"-->"<<c<<endl;

//          cout<<"t"<<b<<"-->"<<c<<endl;

//      }

    //truong hop n>=3

    ThapHN(*n*-1,*a*,*c*,*b*);

    ThapHN(1,*a*,*b*,*c*);

    ThapHN(*n*-1,*b*,*a*,*c*);

}

int main()

{

    int n;char a='A',b='B',c='C';

    cout<<"Nhap n :";cin>>n;

    ThapHN(n,a,b,c);

    return 0;

}

**\*\*\*Bài 1 c không đệ quy :**

#include<iostream>

#include<stack>

using namespace std;

//Khoi tao cau truc lu thong tin di chuyen

struct *Move*{

    int num;//so dia

    char from;//coc nguon

    char to;//coc dich

    char aux;//coc trung gian

};

//Ham thuc hien thap Ha Noi khong dung de quy

void ThapHaNoi(int *num*,char *from*,char *to*,char *aux*)

{

    //Khoi tao stack luu cac buoc di chuyen

    stack<*Move*> s;

    //Dua bai toan vao tong the stack

    s.push({*num*,*from*,*to*,*aux*});

    while(!s.empty())

    {

*Move* current =s.top();//Lay phan tu dau tiene cua stack

        s.pop();//Xoa phan tu dau tien cua stack

        //TH1: Neu chi co 1 dia

        if(current.num==1)

        {

            cout<< current.from << " --> " << current.to << endl;

        }

        //TH2:Neu co nhieu hon 1 dia

        else

        {

            //Dua n-1 dia tu from sang aux

            s.push({current.num - 1, current.aux, current.to, current.from});

            //chuyen dia lon sang to

            s.push({1, current.from, current.to, current.aux});

            //chuyen n-1 dia tu aux sang to

            s.push({current.num - 1, current.from, current.aux, current.to});

        }

    }

}

int main()

{

    int num;cout<<"Nhap so dia :";cin>>num;

    ThapHaNoi(num,'A','B','C');

    return 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).

#include <iostream>

#include <queue>

#include <stack>

using namespace std;

const int MAX\_N = 100; // Giới hạn kích thước mê cung

int maze[MAX\_N][MAX\_N]; // Mê cung

bool visited[MAX\_N][MAX\_N]; // Đánh dấu ô đã đi qua

pair<int, int> parent[MAX\_N][MAX\_N]; // Lưu đường đi

// Hướng di chuyển: trái, phải, lên, xuống

int dx[] = {0, 0, -1, 1}; // Dịch chuyển hàng

int dy[] = {-1, 1, 0, 0}; // Dịch chuyển cột

struct *Cell* {

    int x, y; // Tọa độ ô

};

// Hàm kiểm tra một ô có hợp lệ để di chuyển không

bool isValid(int *x*, int *y*, int *N*, int *M*) {

    return (*x* >= 0 && *x* < *N* && *y* >= 0 && *y* < *M* && maze[*x*][*y*] == 0 && !visited[*x*][*y*]);

}

void bfs(int *N*, int *M*, int *x0*, int *y0*, int *x1*, int *y1*) {

    queue<*Cell*> q; // Hàng đợi BFS

    q.push({*x0*, *y0*}); // Đưa điểm xuất phát vào hàng đợi

    visited[*x0*][*y0*] = true;

    parent[*x0*][*y0*] = {-1, -1};

    while (!q.empty()) {

*Cell* curr = q.front(); q.pop();

        // Nếu đến điểm đích, thoát vòng lặp

        if (curr.x == *x1* && curr.y == *y1*) break;

        // Duyệt 4 hướng di chuyển

        for (int i = 0; i < 4; i++) {

            int nx = curr.x + dx[i];

            int ny = curr.y + dy[i];

            if (isValid(nx, ny, *N*, *M*)) {

                visited[nx][ny] = true;

                parent[nx][ny] = {curr.x, curr.y}; // Lưu ô trước đó để truy vết đường đi

                q.push({nx, ny});

            }

        }

    }

    // Kiểm tra có đến đích được không

    if (!visited[*x1*][*y1*]) {

        cout << "0" << endl; // Không có đường đi

        return;

    }

    // Truy vết ngược để tìm đường đi từ (x1, y1) về (x0, y0)

    stack<*Cell*> path;

    for (*Cell* at = {*x1*, *y1*}; at.x != -1; at = {parent[at.x][at.y].first, parent[at.x][at.y].second}) {

        path.push(at);

    }

    // In số ô trong đường đi

    cout << path.size() << endl;

    // In đường đi theo thứ tự từ (x0, y0) đến (x1, y1)

    while (!path.empty()) {

*Cell* p = path.top(); path.pop();

        cout << p.x + 1 << " " << p.y + 1 << endl; // Đổi về chỉ số 1-based

    }

}

int main() {

    int N, M, x0, y0, x1, y1;

    cin >> N >> M >> x0 >> y0 >> x1 >> y1;

    x0--; y0--; x1--; y1--; // Chuyển về chỉ số 0-based

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < M; j++) {

            cin >> maze[i][j];

            visited[i][j] = false;

        }

    }

    bfs(N, M, x0, y0, x1, y1);

    return 0;

}