**Trần Hoàng Triển - MSSV:46.01.104.196**

**Cài đặt bài toán Tìm đường đi ngắn nhất (Sử dụng thuật toán Floyd)**

**Dữ liệu vào:**

- Dòng 1: Nhập 3 số N u v

‘N’: số đỉnh; u,v là đường đi ngắn nhất đi từ đỉnh u đến v

- N dòng tiếp theo: nhập ma trận trọng số của đồ thị

**A screenshot of a phone

Description automatically generated with medium confidence**

**Dữ liêu ra :**

- Nếu có đường đi thì xuất:

+ Tổng độ dài đường đi

+ Đường đi từ u đến v theo mẫu: u->...->v

A picture containing text, font, white, screenshot

Description automatically generated

- Nếu không có đường đi xuất “NO”

**A screenshot of a phone

Description automatically generated with low confidence A close-up of a white square with black text

Description automatically generated with low confidence**

**Miêu tả cách xử lý dữ liệu như sau:**

Cách xử lý dữ liệu trong đoạn mã trên như sau:

1. Sau khi nhập vào số đỉnh `n` và hai đỉnh `u` và `v` (trong trường hợp này là n= 4, u= 2 và v= 4).

2. Sử dụng hàm `nhap()` để nhập ma trận trọng số `a`. Dựa vào ma trận đã cho, ta có ma trận trọng số như sau:

0 9 0 3

0 0 8 0

5 0 0 0

0 4 1 0

Giá trị 0 trong ma trận thể hiện hai đỉnh không kề nhau.

3. Tiếp theo, chạy thuật toán **Floyd** thông qua hàm `Floyd()` để tìm đường đi ngắn nhất giữa các cặp đỉnh trong đồ thị.

Thuật toán Floyd hoạt động trong đoạn mã trên:

* Khởi tạo ma trận trọng số `d` ban đầu giống với ma trận trọng số `a` đã nhập. Ma trận `d` được dùng để lưu trữ độ dài đường đi ngắn nhất từ mỗi đỉnh đến mọi đỉnh khác.
* Khởi tạo ma trận `next` để lưu trữ đỉnh kế tiếp trên đường đi ngắn nhất từ mỗi đỉnh đến đỉnh khác. Ban đầu, đỉnh kế tiếp của mỗi cặp đỉnh là đỉnh đến chính nó.
* Bắt đầu quá trình cập nhật ma trận `d` và `next` bằng cách duyệt qua tất cả các đỉnh trung gian có thể là điểm dừng trên đường đi từ đỉnh `i` đến đỉnh `j`.
* Duyệt qua tất cả các đỉnh `k` từ 1 đến `n` (số đỉnh của đồ thị).
* Duyệt qua tất cả các cặp đỉnh `i` và `j` (từ 1 đến `n`).
* Kiểm tra nếu đường đi từ `i` đến `j` thông qua đỉnh `k` ngắn hơn đường đi trực tiếp từ `i` đến `j` thì cập nhật giá trị `d[i][j]` thành độ dài đường đi mới và cập nhật đỉnh kế tiếp `next[i][j]` thành `k`.
* Sau khi kết thúc vòng lặp, ma trận `d` sẽ chứa độ dài đường đi ngắn nhất giữa tất cả các cặp đỉnh và ma trận `next` sẽ cho biết đỉnh kế tiếp trên đường đi ngắn nhất từ mỗi đỉnh đến đỉnh khác.
* Hàm `Floyd()` kết thúc và kết quả được sử dụng để tìm đường đi ngắn nhất từ đỉnh `u` đến đỉnh `v`.

4. Sau khi thực hiện thuật toán Floyd, gọi hàm `xuat(u, v)` để in ra đường đi ngắn nhất từ đỉnh `u` đến đỉnh `v`.

5. Kết quả in ra sẽ bao gồm độ dài đường đi ngắn nhất từ đỉnh `u` đến đỉnh `v`, sau đó là các đỉnh trên đường đi ngắn nhất theo thứ tự từ đỉnh `u` đến đỉnh `v`. Nếu không có đường đi, sẽ in ra "NO".

Với dữ liệu nhập vào n=4, u=2, v=4 và ma trận trọng số như trên, kết quả in ra sẽ là:

16

2->3->1->4

Đường đi ngắn nhất từ đỉnh 2 đến đỉnh 4 có độ dài là 16, và các đỉnh trên đường đi ngắn nhất lần lượt là 2, 3, 1, 4.

**Để thực thi chương trình trên, bạn có thể làm theo các bước sau:**

1. Mở trình biên dịch C++ (như Dev-C++, Visual Studio,).

2. Tạo một tệp tin mới và sao chép đoạn mã vào tệp tin đó.

3. Lưu tệp tin với phần mở rộng `.cpp` (ví dụ: `**Floyd.cpp**`).

4. Tiến hành chạy chương trình và nhập dữ liệu từ bàn phím ‘N’: số đỉnh;

u,v là đường đi ngắn nhất đi từ đỉnh u đến v

- N dòng tiếp theo: nhập ma trận trọng số của đồ thị

5. Trình biên dịch sẽ thực thi chương trình và hiển thị kết quả.

**Source code:**

#include<iostream>

#define MAX 1000

using namespace std;

int a[50][50], n, d[50][50], next[50][50];

void nhap() {

for (int i = 1; i <= n; i++)

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

cin >> a[i][j];

if (i != j && a[i][j] == 0)

a[i][j] = MAX;

}

}

void Floyd() {

for (int i = 1; i <= n; i++) /\* Khoi tao \*/

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

d[i][j] = a[i][j];

next[i][j] = j;

}

for (int k = 1; k <= n; k++)

for (int i = 1; i <= n; i++)

for (int j = 1; j <= n; j++)

if (d[i][j] > d[i][k] + d[k][j]) {

d[i][j] = d[i][k] + d[k][j];

next[i][j] = next[i][k];

}

}

void xuat(int x, int y) {

int arr[100];

int n = 0;

if (d[x][y] == MAX) {

cout << "NO" << endl;

}

else {

cout << d[x][y] << endl; // xuat so dinh;

cout << x << "->";

arr[n++] = x;

while (x != y) {

x = next[x][y];

if (x == y)

{

cout << y << endl;

arr[n++] = y;

}

else

{

cout << x << "->";

arr[n++] = x;

}

}

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

{

if (i == 0) cout << arr[i] << endl;

else cout << arr[i] << "<-";

}

}

}

int main() {

int x, y;

cin >> n >> x >> y;

nhap();

Floyd();

xuat(x, y);

return 0;

}