**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP CƠ SỞ**

**Cài đặt thuật toán tìm đường đi ngắn nhất sử dụng ma trận kề**

**GVHD:Nguyễn Thủy Đoan Trang**

**Họ và Tên: Lê Vũ Thanh Hải**

**MSSV: 60139139**

**Lớp : 60.CNTT-2**

**MỤC LỤC**

[PHẦN MỞ ĐẦU 3](#_Toc61818870)

[LỜI CẢM ƠN 4](#_Toc61818871)

[NỘI DUNG 5](#_Toc61818872)

[I. Mục tiêu cần đạt được: 5](#_Toc61818873)

[II. Lý do chọn đề tài: 5](#_Toc61818874)

[III. Tài liệu đã tìm kiếm: 6](#_Toc61818875)

[CHƯƠNG I: LÝ THUYẾT 8](#_Toc61818876)

[I. Định nghĩa đồ thị (GRAPH): 8](#_Toc61818877)

[II. Các khái niệm: 9](#_Toc61818878)

[III. MA TRẬN LIỀN KỀ (MA TRẬN KỀ): 11](#_Toc61818879)

[IV. THUẬT TOÁN DIJKSTRA: 13](#_Toc61818880)

[V. THUẬT TOÁN DIJKSTRA VÀ CẤU TRÚC HEAP: 14](#_Toc61818881)

[CHƯƠNG II: PHÂN TÍCH, THIẾT KẾ 15](#_Toc61818882)

[1. Phân tích dữ liệu, thuật toán: 15](#_Toc61818883)

[2. Mô tả thuật toán và ý tưởng: 16](#_Toc61818884)

[3. Xây dựng chương trình: 19](#_Toc61818885)

[4. Code: 25](#_Toc61818886)

[CHƯƠNG III: MÔ PHỎNG BẰNG ĐỒ HỌA 29](#_Toc61818887)

**LỜI NHẬN XÉT**

**( Của giảng viên hướng dẫn )**

# PHẦN MỞ ĐẦU

Lý thuyết đồ thị là một lĩnh vực nghiên cứu đã có từ lâu đờivà có nhiều

ứng dụng hiện đại.Những tư tưởng cơ bản của lý thuyết đồ thị đươc đề xuất từ

những năm đầu của thế kỷ 18 bởi nhà toán học lỗi lạc người Thụy Sĩ Leonhard

Euler.Chính ông là người đã sử dụng đồ thị để giải bài toán nổi tiếng về các cái

cầu ở thàng phố Konigsberg.

Đồ thị được sử dụng để giải quyết các bài toán trong nhiều lĩnh vực khác

nhau .Chẳng hạn , đồ thị có thể sử dụng để xác định các mạch vòng trong vấn đề

giải tích mạch điện.Chúng ta có thể phân biệt các hợp chất hoá học hữu cơ khác

nhau với cùng công thức phân tử nhưng khác nhau về cấu trúc phân tử nhờ đồ

thị.Chúng ta có thể xác định xem hai máy tính trong mạng có thể trao đổi thông tin

được với nhau hay không nhờ mô hình đồ thị của mạng máy tính. Đồ thị có trọng

số trên các cạnh có thể sử dụng để giải các bài toán như : tìm đường đi ngắn nhất

giữa hai thành phố trong cùng một mạng giao thông . Chúng ta còn sử dụng đồ thị

để giải các bài toán về lập lịch,thời khoá biểu,và phân bố tần số cho các trạm phát

thanh và truyền hình....

Mục đích ta tìm hiểu là nhằm giới thiệu các khái niệm cơ bản,các bài toán

ứng dụng quan trọng của lý thuyết đồ thị như bài toán cây khung nhỏ nhất , bài

toán tìm đường đi ngắn nhất... và những thuật toán để giải quyết chúng đã được

trình bày chi tiết cùng với việc phân tích và hướng dẫn cài đặt chương trình trên

máy tính.

# LỜI CẢM ƠN

Nhờ sự chỉ dạy và giúp đỡ nhiệt tình của cô Nguyễn Thủy Đoan Trang mà em đã hoàn thành bài báo cáo này. Mỗi tuần đều được cô hướng dẫn và góp ý để hoàn thiện bài báo cáo, được cô củng cố thêm nhiều kiến thức bổ ích. Em đã làm hết sức khả năng của mình nhưng khó có thể không có chỗ sai sót, mong cô sẽ xem xét.

Chúc cô sức khỏe và thành công.

# NỘI DUNG

## I. Mục tiêu cần đạt được:

* Mức 1:

1. Sử dụng ngôn ngữ C/C++ cài đặt các thuật toán tìm đường đi ngắn nhất đồ thị Disktra với bằng ma trận kề.
2. Dữ liệu vào: File văn bản DIJKSTRA.INP
   * Dòng đầu tiên chứa số đỉnh 𝑛, số cung 𝑚 của đồ thị và 2 đỉnh 𝑥, 𝑦.
   * 𝑚 dòng tiếp theo, mỗi dòng chứa 3 số 𝑢, 𝑣, 𝑤 mô tả cung (𝑢, 𝑣) có trọng số 𝑤.

Dữ liệu ra: File văn bản DIJKSTRA.OUT

* + Dòng đầu tiên ghi số đỉnh đường đi ngắn nhất đi qua (tính luôn hai đỉnh 𝑥 và 𝑦) và độ dài đường đi.
  + Dòng thứ hai ghi danh sách các đỉnh của đường đi ngắn nhất tìm được từ 𝑠 đến 𝑡.
* Mức 2: Mô phỏng bằng đồ họa.

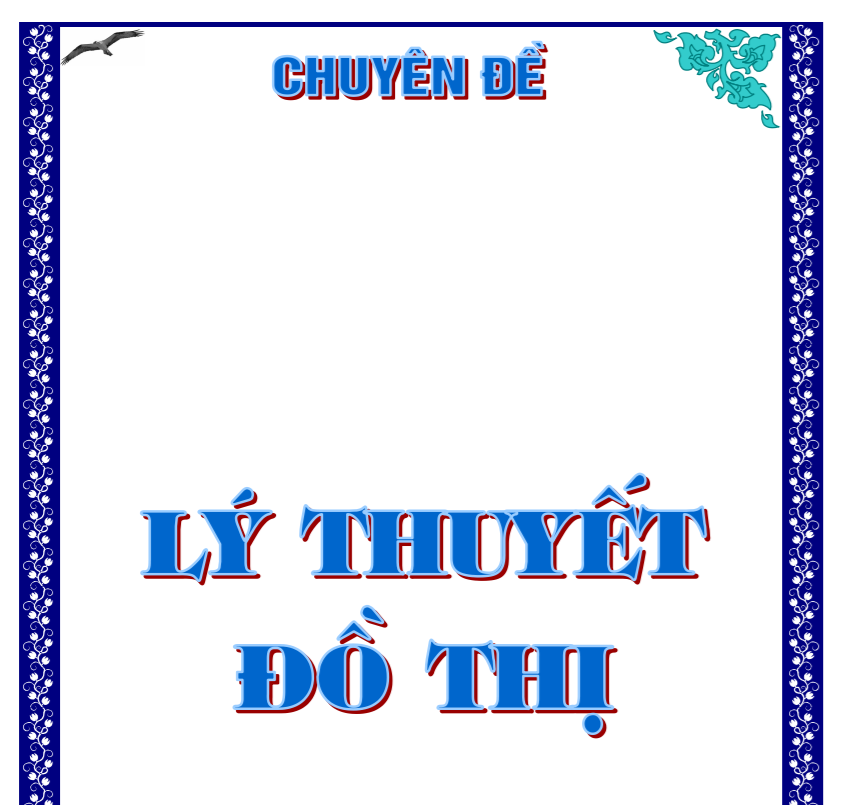
## II. Lý do chọn đề tài:

-Lý do em chọn đề tài này là vì em cảm thấy hứng thú với tìm đường đi ngắn nhất và thuật toán Dijkstra

## III. Tài liệu đã tìm kiếm:

+ Lý thuyết đồ thị: thầy Lê Minh Hoàng

https://www.hnue.edu.vn/Portals/0/TeachingSubject/hongntcntt/07b6e3d3-6727-489d-a0c5-c81f5f24daa1ly-thuyet-do-thi---le-minh-hoang.pdf



**+** Lý thuyết đồ thị: tiến sỹ Ngô Hữu Phúc và thạc sỹ Vi Bảo Ngọc

<https://fit.lqdtu.edu.vn/files/FileMonHoc/Ly%20Thuyet%20do%20thi.pdf>



**Tìm hiểu GITHUB:**

B1: GITHUB.com => Login

B2: Tạo kho lưu trữ

B3: Tải github

B4: Mở github vừa tải về => Clone a repository => Kho lưu trữ

# CHƯƠNG I: LÝ THUYẾT

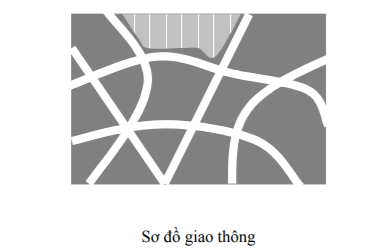
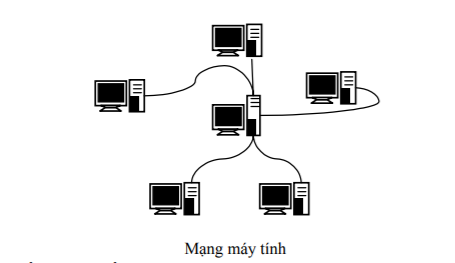
## I. Định nghĩa đồ thị (GRAPH):

Là một cấu trúc rời rạc gồm các đỉnh và các cạnh nối các đỉnh đó. Được mô tả hình thức:

G = (V, E)

V gọi là tập các đỉnh (Vertices) và E gọi là tập các cạnh (Edges). Có thể coi E là tập các cặp (u, v) với u và v là hai đỉnh của V.

Một số hình ảnh của đồ thị:

****

Có thể phân loại đồ thị theo đặc tính và số lượng của tập các cạnh E:

Cho đồ thị G = (V, E). Định nghĩa một cách hình thức

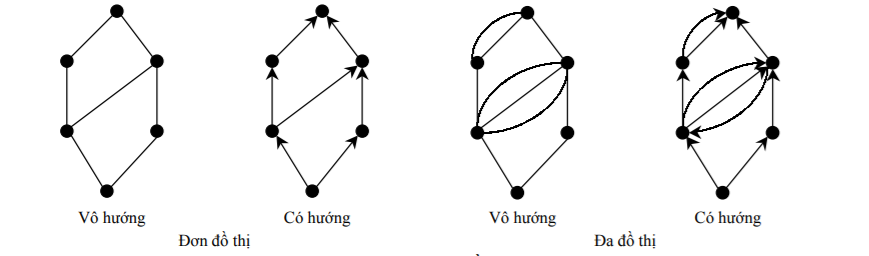
1. G được gọi là đơn đồ thị nếu giữa hai đỉnh u, v của V có nhiều nhất là 1 cạnh trong E nối từ u tới v.

2. G được gọi là đa đồ thị nếu giữa hai đỉnh u, v của V có thể có nhiều hơn 1 cạnh trong E nối từ u tới v (Hiển nhiên đơn đồ thị cũng là đa đồ thị).

3. G được gọi là đồ thị vô hướng nếu các cạnh trong E là không định hướng, tức là cạnh nối hai đỉnh u, v bất kỳ cũng là cạnh nối hai đỉnh v, u. Hay nói cách khác, tập E gồm các cặp (u, v) không tính thứ tự. (u, v)≡(v, u)

4. G được gọi là đồ thị có hướng nếu các cạnh trong E là có định hướng, có thể có cạnh nối từ đỉnh u tới đỉnh v nhưng chưa chắc đã có cạnh nối từ đỉnh v tới đỉnh u. Hay nói cách khác, tập E gồm các cặp (u, v) có tính thứ tự: (u, v) ≠ (v, u). Trong đồ thị có hướng, các cạnh được gọi là các cung. Đồ thị vô hướng cũng có thể coi là đồ thị có hướng nếu như ta coi cạnh nối hai đỉnh u, v bất kỳ tương đương với hai cung (u, v) và (v, u).

Ví dụ:

****

## II. Các khái niệm:

Như trên định nghĩa đồ thị G = (V, E) là một cấu trúc rời rạc, tức là các tập V và E hoặc là tập hữu hạn, hoặc là tập đếm được, có nghĩa là ta có thể đánh số thứ tự 1, 2, 3... cho các phần tử của tập V và E. Hơn nữa, đứng trên phương diện người lập trình cho máy tính thì ta chỉ quan tâm đến các đồ thị hữu hạn (V và E là tập hữu hạn) mà thôi, chính vì vậy từ đây về sau, nếu không chú thích gì thêm thì khi nói tới đồ thị, ta hiểu rằng đó là đồ thị hữu hạn.

Cạnh liên thuộc, đỉnh kề, bậc

• Đối với đồ thị vô hướng G = (V, E). Xét một cạnh e ∈ E, nếu e = (u, v) thì ta nói hai đỉnh u và v là kề nhau (adjacent) và cạnh e này liên thuộc (incident) với đỉnh u và đỉnh v.

• Với một đỉnh v trong đồ thị, ta định nghĩa bậc (degree) của v, ký hiệu deg(v) là số cạnh liên thuộc với v. Dễ thấy rằng trên đơn đồ thị thì số cạnh liên thuộc với v cũng là số đỉnh kề với v.

Định lý: Giả sử G = (V, E) là đồ thị vô hướng với m cạnh, khi đó tổng tất cả các bậc đỉnh trong V sẽ bằng 2m:



Chứng minh: Khi lấy tổng tất cả các bán bậc ra hay bán bậc vào, mỗi cung (u, v) bất kỳ sẽ được tính đúng 1 lần trong deg+ (u) và cũng được tính đúng 1 lần trong deg- (v). Từ đó suy ra kết quả

Một số tính chất của đồ thị có hướng không phụ thuộc vào hướng của các cung. Do đó để tiện trình bày, trong một số trường hợp ta có thể không quan tâm đến hướng của các cung và coi các cung đó là các cạnh của đồ thị vô hướng. Và đồ thị vô hướng đó được gọi là đồ thị vô hướng nền của đồ thị có hướng ban đầu

Nhận xét:

Trên đây là nêu các cách biểu diễn đồ thị trong bộ nhớ của máy tính, còn nhập dữ liệu cho đồ thị thì có nhiều cách khác nhau, dùng cách nào thì tuỳ. Chẳng hạn nếu biểu diễn bằng ma trận kề mà cho nhập dữ liệu cả ma trận cấp n x n (n là số đỉnh) thì khi nhập từ bàn phím sẽ rất mất thời gian, ta cho nhập kiểu danh sách cạnh cho nhanh. Chẳng hạn mảng A (nxn) là ma trận kề của một đồ thị vô hướng thì ta có thể khởi tạo ban đầu mảng A gồm toàn số 0, sau đó cho người sử dụng nhập các cạnh bằng cách nhập các cặp (i, j); chương trình sẽ tăng A[i, j] và A[j, i] lên 1. Việc nhập có thể cho kết thúc khi người sử dụng nhập giá trị i = 0.

Ví dụ:

program Nhap\_Do\_Thi;

var A: array[1..100, 1..100] of Integer; {Ma trận kề của đồ thị}

n, i, j: Integer;

begin

Write('Number of vertices'); ReadLn(n);

FillChar(A, SizeOf(A), 0);

Repeat

Write('Enter edge (i, j) (i = 0 to exit) ');

ReadLn(i, j); {Nhập một cặp (i, j) tưởng như là nhập danh sách cạnh}

if i <> 0 then

begin {nhưng lưu trữ trong bộ nhớ lại theo kiểu ma trận kề}

Inc(A[i, j]);

Inc(A[j, i]);

end;

until i = 0; {Nếu người sử dụng nhập giá trị i = 0 thì dừng quá trình nhập, nếu không thì tiếp tục}

end.

Trong nhiều trường hợp đủ không gian lưu trữ, việc chuyển đổi từ cách biểu diễn nào đó sang cách biểu diễn khác không có gì khó khăn. Nhưng đối với thuật toán này thì làm trên ma trận kề ngắn gọn hơn, đối với thuật toán kia có thể làm trên danh sách cạnh dễ dàng hơn v.v... Do đó, với mục đích dễ hiểu, các chương trình sau này sẽ lựa chọn phương pháp biểu diễn sao cho việc cài đặt đơn giản nhất nhằm nêu bật được bản chất thuật toán. Còn trong trường hợp cụ thể bắt buộc phải dùng một cách biểu diễn nào đó khác, thì việc sửa đổi chương trình cũng không tốn quá nhiều thời gian.

## III. MA TRẬN LIỀN KỀ (MA TRẬN KỀ):

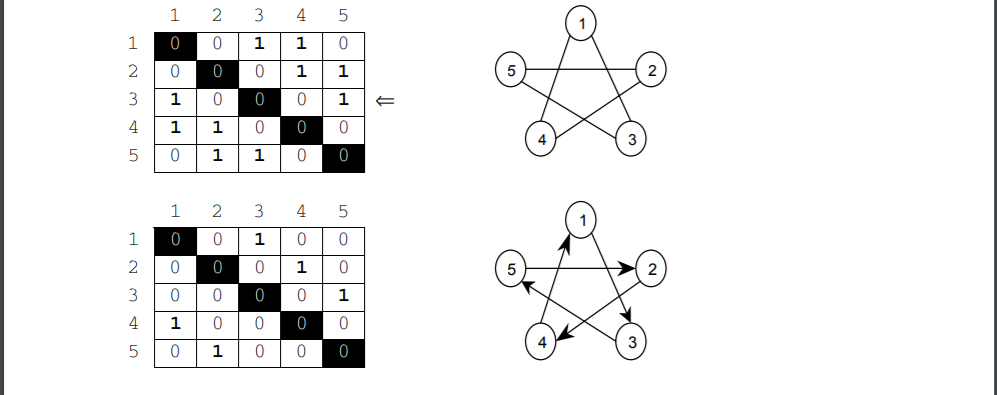
Giả sử G = (V, E) là một đơn đồ thị có số đỉnh (ký hiệu |V|) là n, Không mất tính tổng quát có thể coi các đỉnh được đánh số 1, 2, ..., n. Khi đó ta có thể biểu diễn đồ thị bằng một ma trận vuông A = [aij] cấp n. Trong đó:

• aij = 1 nếu (i, j) ∈ E

• aij = 0 nếu (i, j) ∉ E

• Quy ước aii = 0 với ∀i; Đối với đa đồ thị thì việc biểu diễn cũng tương tự trên, chỉ có điều nếu như (i, j) là cạnh thì không phải ta ghi số 1 vào vị trí aij mà là ghi số cạnh nối giữa đỉnh i và đỉnh j

Ví dụ:



Các tính chất của ma trận liền kề:

1. Đối với đồ thị vô hướng G, thì ma trận liền kề tương ứng là ma trận đối xứng (aij = aji), điều này không đúng với đồ thị có hướng.

2. Nếu G là đồ thị vô hướng và A là ma trận liền kề tương ứng thì trên ma trận A: Tổng các số trên hàng i = Tổng các số trên cột i = Bậc của đỉnh i = deg(i)

3. Nếu G là đồ thị có hướng và A là ma trận liền kề tương ứng thì trên ma trận A:

• Tổng các số trên hàng i = Bán bậc ra của đỉnh i = deg+ (i)

• Tổng các số trên cột i = Bán bậc vào của đỉnh i = deg- (i)

Trong trường hợp G là đơn đồ thị, ta có thể biểu diễn ma trận liền kề A tương ứng là các phần tử logic. aij = TRUE nếu (i, j) ∈ E và aij = FALSE nếu (i, j) ∉ E

Ưu điểm của ma trận liền kề:

• Đơn giản, trực quan, dễ cài đặt trên máy tính

• Để kiểm tra xem hai đỉnh (u, v) của đồ thị có kề nhau hay không, ta chỉ việc kiểm tra bằng một phép so sánh: auv ≠ 0.

Nhược điểm của ma trận liền kề:

• Bất kể số cạnh của đồ thị là nhiều hay ít, ma trận liền kề luôn luôn đòi hỏi n2 ô nhớ để lưu các phần tử ma trận, điều đó gây lãng phí bộ nhớ dẫn tới việc không thể biểu diễn được đồ thị với số đỉnh lớn.

Với một đỉnh u bất kỳ của đồ thị, nhiều khi ta phải xét tất cả các đỉnh v khác kề với nó, hoặc xét tất cả các cạnh liên thuộc với nó. Trên ma trận liền kề việc đó được thực hiện bằng cách xét tất cả các đỉnh v và kiểm tra điều kiện auv ≠ 0. Như vậy, ngay cả khi đỉnh u là **đỉnh cô lập** (không kề với đỉnh nào) hoặc **đỉnh treo** (chỉ kề với 1 đỉnh) ta cũng buộc phải xét tất cả các đỉnh và kiểm tra điều kiện trên dẫn tới lãng phí thời gian.

## IV. THUẬT TOÁN DIJKSTRA:

Trong trường hợp trọng số trên các cung không âm, thuật toán do Dijkstra đề xuất dưới đây hoạt động hiệu quả hơn nhiều so với thuật toán Ford-Bellman. Ta hãy xem trong trường hợp này, thuật toán Ford-Bellman thiếu hiệu quả ở chỗ nào: Với đỉnh v ∈ V, Gọi d[v] là độ dài đường đi ngắn nhất từ S tới v. Thuật toán Ford-Bellman khởi tạo d[v] = c[S, v]. Sau đó tối ưu hoá dần các nhãn d[v] bằng cách sửa nhãn theo công thức: d[v] := min(d[v], d[u] + c[u, v]) với ∀u, v ∈ V. Như vậy nếu như ta dùng đỉnh u sửa nhãn đỉnh v, sau đó nếu ta lại tối ưu được d[u] thêm nữa thì ta cũng phải sửa lại nhãn d[v] dẫn tới việc d[v] có thể phải chỉnh đi chỉnh lại rất nhiều lần. Vậy nên chăng, tại mỗi bước không phải ta xét mọi cặp đỉnh (u, v) để dùng đỉnh u sửa nhãn đỉnh v mà sẽ chọn đỉnh u là đỉnh mà không thể tối ưu nhãn d[u] thêm được nữa. Thuật toán Dijkstra (E.Dijkstra - 1959) có thể mô tả như sau:

**Bước 1:** Khởi tạo Với đỉnh v ∈ V, gọi nhãn d[v] là độ dài đường đi ngắn nhất từ S tới v. Ta sẽ tính các d[v]. Ban đầu d[v] được khởi gán bằng c[S, v]. Nhãn của mỗi đỉnh có hai trạng thái tự do hay cố định, nhãn tự do có nghĩa là có thể còn tối ưu hơn được nữa và nhãn cố định tức là d[v] đã bằng độ dài đường đi ngắn nhất từ S tới v nên không thể tối ưu thêm. Để làm điều này ta có thể sử dụng kỹ thuật đánh dấu: Free[v] = TRUE hay FALSE tuỳ theo d[v] tự do hay cố định. Ban đầu các nhãn đều tự do.

**Bước 2:** Lặp Bước lặp gồm có hai thao tác:

1. Cố định nhãn: Chọn trong các đỉnh có nhãn tự do, lấy ra đỉnh u là đỉnh có d[u] nhỏ nhất, và cố định nhãn đỉnh u.

2. Sửa nhãn: Dùng đỉnh u, xét tất cả những đỉnh v và sửa lại các d[v] theo công thức:

d[v] := min(d[v], d[u] + c[u, v])

Bước lặp sẽ kết thúc khi mà đỉnh đích F được cố định nhãn (tìm được đường đi ngắn nhất từ S tới F); hoặc tại thao tác cố định nhãn, tất cả các đỉnh tự do đều có nhãn là +∞ (không tồn tại đường đi).

Có thể đặt câu hỏi, ở thao tác 1, tại sao đỉnh u như vậy được cố định nhãn, giả sử d[u] còn có thể tối ưu thêm được nữa thì tất phải có một đỉnh t mang nhãn tự do sao cho d[u] > d[t] + c[t, u]. Do trọng số c[t, u] không âm nên d[u] > d[t], trái với cách chọn d[u] là nhỏ nhất. Tất nhiên trong lần lặp đầu tiên thì S là đỉnh được cố định nhãn do d[S] = 0. Bước 3: Kết hợp với việc lưu vết đường đi trên từng bước sửa nhãn, thông báo đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi (d[F] = +∞).

## V. THUẬT TOÁN DIJKSTRA VÀ CẤU TRÚC HEAP:

Nếu đồ thị có nhiều đỉnh, ít cạnh, ta có thể sử dụng danh sách kề kèm trọng số để biểu diễn đồ thị, tuy nhiên tốc độ của thuật toán DIJKSTRA vẫn khá chậm vì trong trường hợp xấu nhất, nó cần n lần cố định nhãn và mỗi lần tìm đỉnh để cố định nhãn sẽ mất một đoạn chương trình với độ phức tạp O(n). Để tăng tốc độ, người ta thường sử dụng cấu trúc dữ liệu Heap để lưu các đỉnh chưa cố định nhãn. Heap ở đây là một cây nhị phân hoàn chỉnh thoả mãn: Nếu u là đỉnh lưu ở nút cha và v là đỉnh lưu ở nút con thì d[u] ≤ d[v].

(Đỉnh r lưu ở gốc Heap là đỉnh có d[r] nhỏ nhất).

Tại mỗi bước lặp của thuật toán Dijkstra có hai thao tác: Tìm đỉnh cố định nhãn và Sửa nhãn.

• Thao tác tìm đỉnh cố định nhãn sẽ lấy đỉnh lưu ở gốc Heap, cố định nhãn, đưa phần tử cuối Heap vào thế chỗ và thực hiện việc vun đống (Adjust)

• Thao tác sửa nhãn, sẽ duyệt danh sách kề của đỉnh vừa cố định nhãn và sửa nhãn những đỉnh tự do kề với đỉnh này, mỗi lần sửa nhãn một đỉnh nào đó, ta xác định đỉnh này nằm ở đâu trong Heap và thực hiện việc chuyển đỉnh đó lên (UpHeap) phía gốc Heap nếu cần để bảo toàn cấu trúc Heap.

Cài đặt dưới đây có Input/Output giống như trên nhưng có thể thực hiện trên đồ thị 5000 đỉnh, 10000 cạnh, trọng số mỗi cạnh ≤ 10000.

# CHƯƠNG II: PHÂN TÍCH, THIẾT KẾ

## 1. Phân tích dữ liệu, thuật toán:

Dữ liệu vào: File văn bản DIJKSTRA.INP

+ Dòng đầu tiên chứa số đỉnh 𝑛, số cung 𝑚 của đồ thị và 2 đỉnh 𝑥, 𝑦.

𝑚 dòng tiếp theo, mỗi dòng chứa 3 số 𝑢, 𝑣, 𝑤 mô tả cung (𝑢, 𝑣) có trọng số 𝑤.

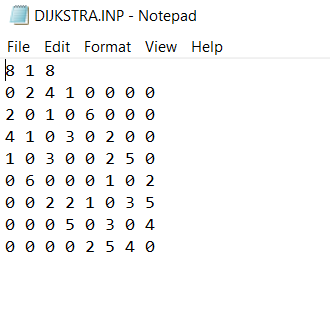
Dữ liệu ra: File văn bản DIJKSTRA.OUT

+ Dòng đầu tiên ghi số đỉnh đường đi ngắn nhất đi qua (tính luôn hai đỉnh 𝑥 và 𝑦) và độ dài đường đi.

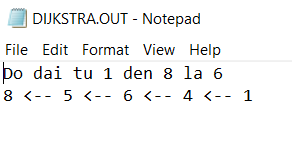
+ Dòng thứ hai ghi danh sách các đỉnh của đường đi ngắn nhất tìm được từ 𝑠 đến 𝑡.

Ví dụ:

Đầu vào:



Đầu ra:



## 2. Mô tả thuật toán và ý tưởng:

+ **Mô tả thuật toán:** Cho đồ thị vô hướng G = (V, E), tìm đường đi ngắn nhất từ điểm 1 đến điểm 8.

+ **Ý tưởng thuật toán:** sử dụng thuật toán Dijkstra

- Nhập dữ liệu đầu vào và đầu ra theo đề tài giảng viên giao.

- Đồ thị cho trong tập tin DIJKSTRA.INP.

- Dòng đầu tiên chứa số đỉnh n, và 2 đỉnh x,y

- n dòng tiếp theo, mỗi dòng chứa n số. Trong đó

Tìm đường đi ngắn nhất sử dụng ma trận kề, ma trận kề có n đỉnh là ma trận vuông G có số hàng số cột là n. G[i][j] là độ dài đường đi từ đỉnh i tới đỉnh j. Ở đây ta xét đồ thị vô hướng thì G[i][j] = G[j][i]. Độ dài từ một đỉnh tới chính nó luôn là 0 (G[i][i] = 0). Nếu giữa 2 cạnh i và j của đồ thị không tồn tại đường đi thì G[i][j] = vô cùng (∞). Tuy nhiên khi biểu diễn trong máy tính thì giá trị ∞ được đặt là 1 hằng số rất lớn hoặc là tổng các giá trị trong ma trận (tổng độ dài các cạnh).

- Các dòng được hiển thị như bảng bên dưới:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 0 | 2 | 4 | 1 | 0 | 0 | 0 | 0 |
| 2 | 2 | 0 | 1 | 0 | 6 | 0 | 0 | 0 |
| 3 | 4 | 1 | 0 | 3 | 0 | 2 | 0 | 0 |
| 4 | 1 | 0 | 3 | 0 | 0 | 2 | 5 | 0 |
| 5 | 0 | 6 | 0 | 0 | 0 | 1 | 0 | 2 |
| 6 | 0 | 0 | 2 | 2 | 1 | 0 | 3 | 5 |
| 7 | 0 | 0 | 0 | 5 | 0 | 3 | 0 | 4 |
| 8 | 0 | 0 | 0 | 0 | 2 | 5 | 4 | 0 |

4

1

3

1

6

2

2

2

3

5

5

4

1

2

Đầu tiên:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

+ Ta sẽ xuất phát từ đỉnh 0, nên giá trị tại 1 là 0 và các đỉnh còn lại là vô cùng

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | . | 2 (1,2) | 4 (1,3) | 1 (1,4) | ∞ | ∞ | ∞ | ∞ |

+ Bắt đầu từ đỉnh 1 ta thấy từ 1 có đường đi tới 3 đỉnh là 2, 3, 4. Xét trọng số (1,2) =2; (1,3) = 4; (1,4) = 3. Ta điền vào bảng, những đỉnh còn lại không có đường đi thì giữ Nguyên vô cùng. Từ bước này ta sẽ chọn đỉnh tiếp theo để đi, chọn đỉnh có giá trị trọng số nhỏ nhất ở từng bước. Ta xét đỉnh 4 vì có trọng số nhỏ nhất từ đỉnh 1 tới đỉnh 4 là 1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | . | 2 (1,2) | 4 (4,3) | . | ∞ | 3 (4,6) | 6 (4,7) | ∞ |

+ Tiếp theo đến đỉnh 4, ta thấy từ đỉnh 4 có đường đi tới 3 đỉnh là 3, 6, 7 có các trọng số lần lượt là 3, 2, 5 cộng với 1 (1,4) thành 4, 3, 6. Ta thấy giá trị trọng số nhỏ nhất ở đây là 3 (4,6), ta xét đỉnh tiếp theo là 6.

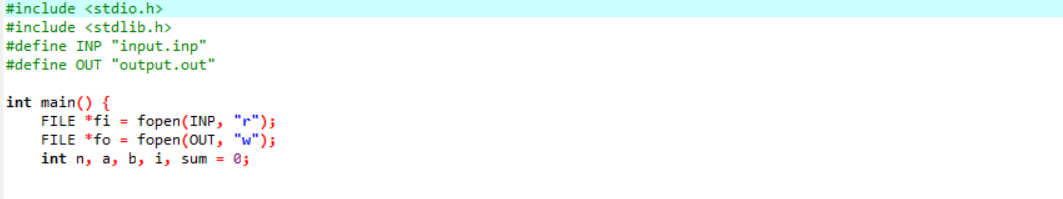
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 3 | . | 2 (1,2) | 5 (6,3) | . | 4 (6,5) | . | 6 (6,7) | 8 (6,8) |

+ Tiếp là đỉnh 6, ta thấy từ đỉnh 6 có đường đi tới 4 đỉnh là 3, 5, 7, 8 có các trọng số lần lượt là 5, 4, 6, 8.Ta thấy giá trị trọng số nhỏ nhất ở đây là 4 (6,5), ta xét đỉnh tiếp theo là 5.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 4 | . | 10(5,2) | 5 (6,3) | . | . | . | 6 (6,7) | 6 (5,8) |

+ Cuối cùng là đỉnh 5, ta thấy từ đỉnh 5 có đường đi tới 3 đỉnh là 2, 8 có các trọng số lần lượt là 10, 6. Ta thấy giá trị trọng số nhỏ nhất ở đây là 6 (5,8). Ta kết thúc thuật toán tìm đường đi ngắn nhất từ đỉnh 1 tới đỉnh 8.

## 3. Xây dựng chương trình:

****

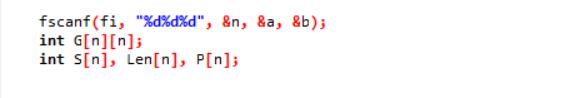
**+** Khai báo các thư viện cần thiết.

**+** Define INP “input.inp” định nghĩa file có tên truy cập là INP.

+ Define INP “output.out” định nghĩa file có tên truy cập là OUT.

+ Tiếp đó khai báo file nhập FILE \*fi và khai báo file xuất FILE \*fo.

+ Khai báo biến n, a, b, i, sum = 0.

****

**+** Tiếp theo là nhập dữ liệu từ file input.

+ Khai báo mảng 2 chiều.

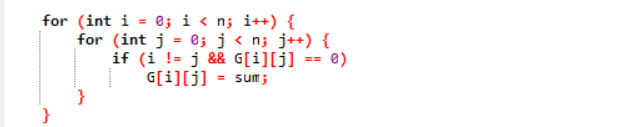
+ Khai báo mảng 1 chiều.

+ Khai báo mảng 1 chiều.



+ Nhập ma trận và tính giá trị vô cùng.

+ Dùng vòng lặp và nhập giá trị trong file.



**+** Các cặp cạnh không nối với nhau thì đặt giá trị vô cùng.

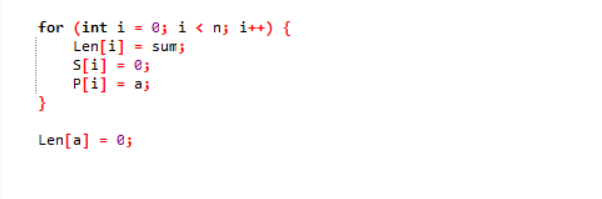
+ Dùng vòng lặp để tính tổng vào mảng 2 chiều.

****

+ Do mảng tính từ G[0][0] nên cần lùi đi vị trí đi 1 đơn vị để tính toán.

+ Giảm biến a đi 1 đơn vị.

+ Giảm biến b đi 1 đơn vị.

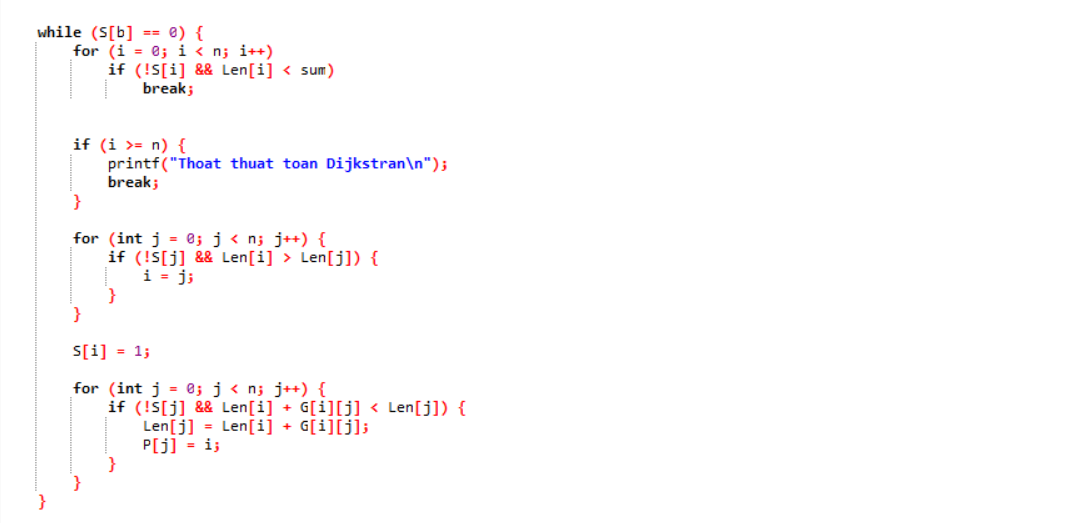


**+** Điểm khởi đầu đặt độ dài từ a tới mỗi đỉnh là vô cùng (∞).

+ Lọc ra danh sách các điểm đã xét.

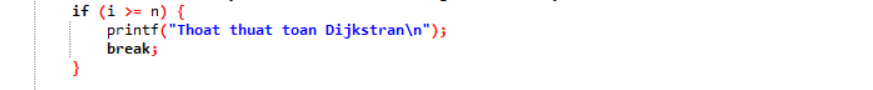
+ Điểm khởi đầu tại mỗi điểm là a.

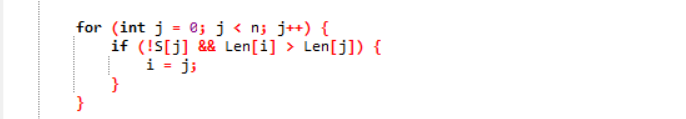
+ Độ dài từ a tới a = 0.

****

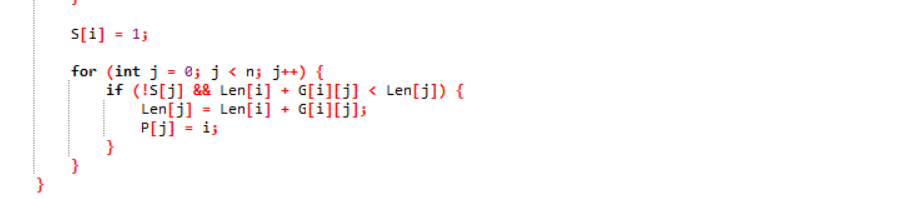
**+** Nếu điểm cuối chưa được xét thì ta dung while.

+ Tìm vị trí nào đó có giá trị không phải là vô cùng.

**  
  
+** Tìm đỉnh b nếu không tìm thấy thì thoát thuật toán.

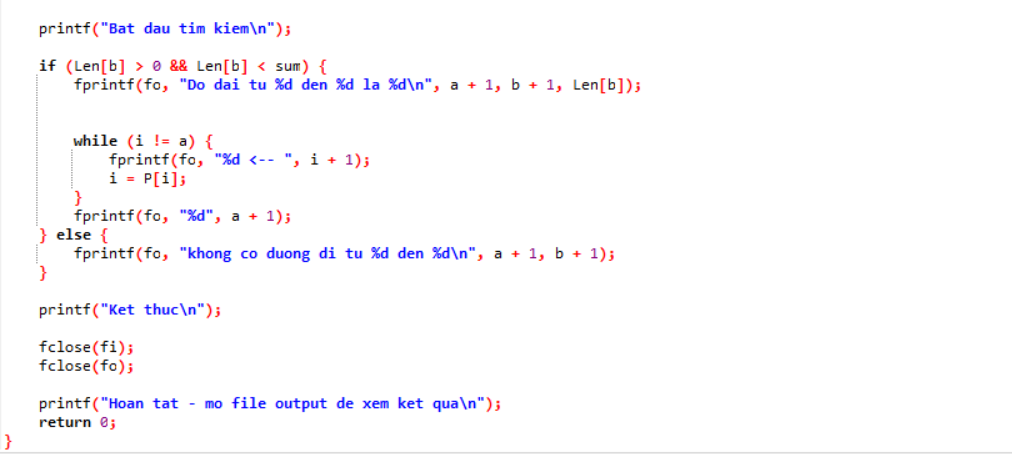
****

**+** Tìm vị trí có độ dài = min.

****

**+** Danh sách đã xét rồi cho điểm I vào.

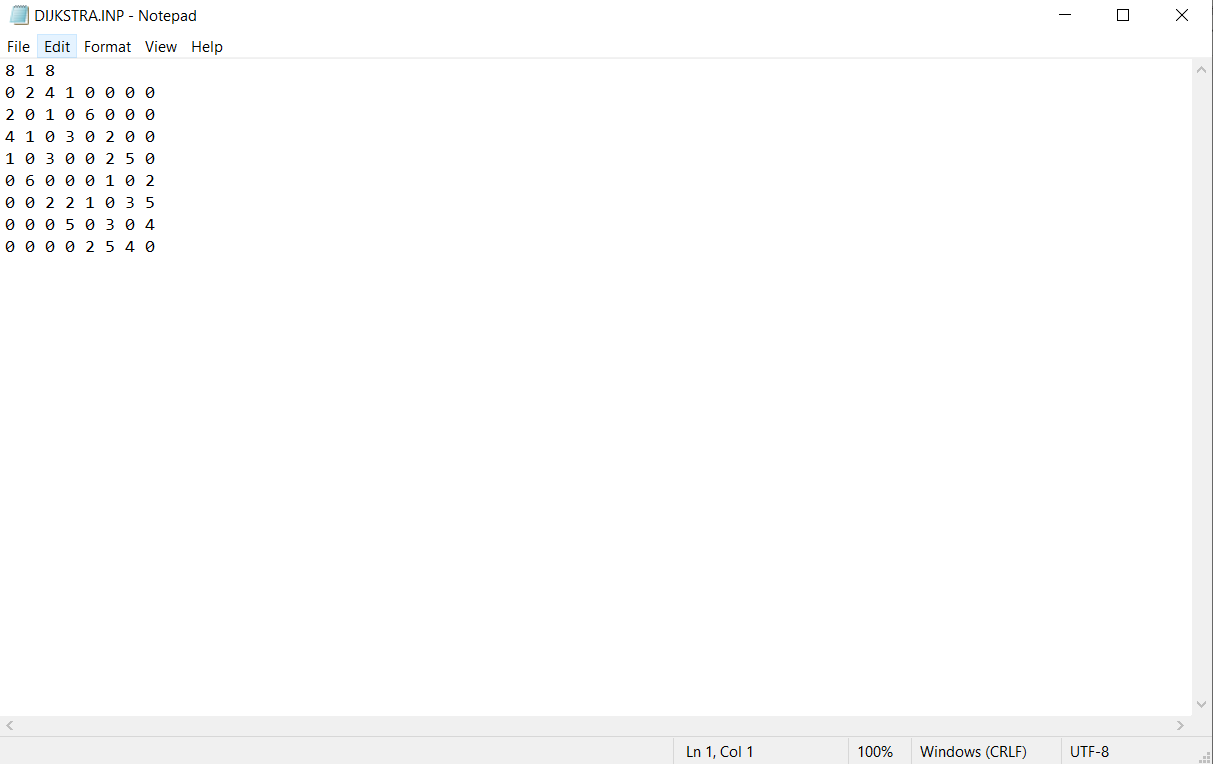
+ Tính lại độ dài tất cả các cạnh chưa được xét.

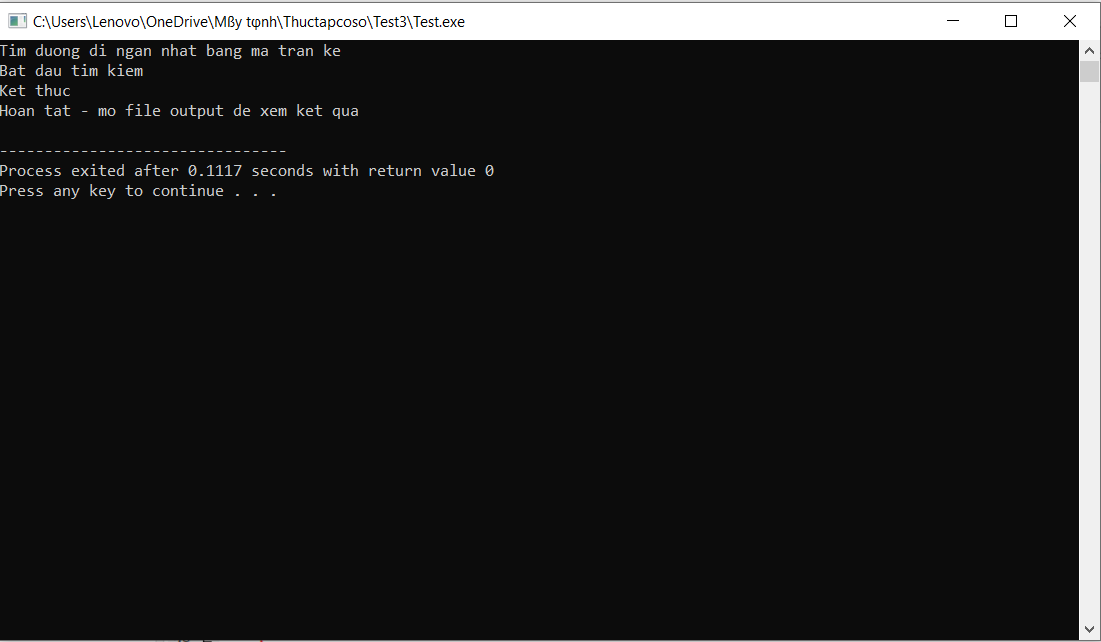
**  
  
+** Do bắt đầu tính toán từ đỉnh 0 mà tính toán bắt đầu từ đỉnh 1 nên dùng I + 1 để chính xác.

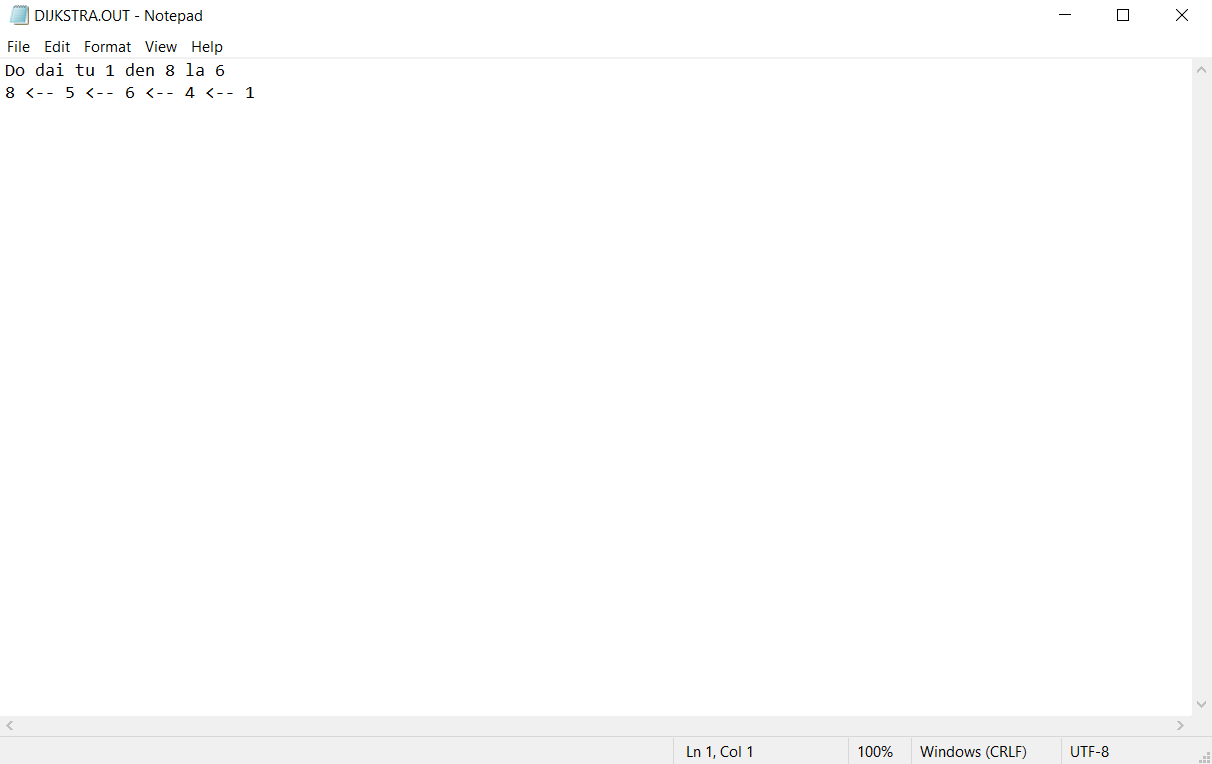
+ Sử dụng truy vết và tính toán độ dài.

+ Đóng file nhập và file xuất.

**Kết quả:**



****

****

## 4. Code:

**#include <stdio.h>**

**#include <stdlib.h>**

**#define INP "DIJKSTRA.INP"**

**#define OUT "DIJKSTRA.OUT"**

**int main() {**

**FILE \*fi = fopen(INP, "r");**

**FILE \*fo = fopen(OUT, "w");**

**int n, a, b, i, sum = 0;**

**fscanf(fi, "%d%d%d", &n, &a, &b);**

**int G[n][n];**

**int S[n], Len[n], P[n];**

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

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

**fscanf(fi, "%d", &G[i][j]);**

**sum += G[i][j];**

**}**

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

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

**if (i != j && G[i][j] == 0)**

**G[i][j] = sum;**

**}**

**}**

**a--;**

**b--;**

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

**Len[i] = sum;**

**S[i] = 0;**

**P[i] = a;**

**}**

**Len[a] = 0;**

**while (S[b] == 0) {**

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

**if (!S[i] && Len[i] < sum)**

**break;**

**if (i >= n) {**

**printf("Thoat thuat toan Dijkstran\n");**

**break;**

**}**

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

**if (!S[j] && Len[i] > Len[j]) {**

**i = j;**

**}**

**}**

**S[i] = 1;**

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

**if (!S[j] && Len[i] + G[i][j] < Len[j]) {**

**Len[j] = Len[i] + G[i][j];**

**P[j] = i;**

**}**

**}**

**}**

**printf("Tim duong di ngan nhat bang ma tran ke\n");**

**printf("Bat dau tim kiem\n");**

**if (Len[b] > 0 && Len[b] < sum) {**

**fprintf(fo, "Do dai tu %d den %d la %d\n", a + 1, b + 1, Len[b]);**

**while (i != a) {**

**fprintf(fo, "%d <-- ", i + 1);**

**i = P[i];**

**}**

**fprintf(fo, "%d", a + 1);**

**} else {**

**fprintf(fo, "khong co duong di tu %d den %d\n", a + 1, b + 1);**

**}**

**printf("Ket thuc\n");**

**fclose(fi);**

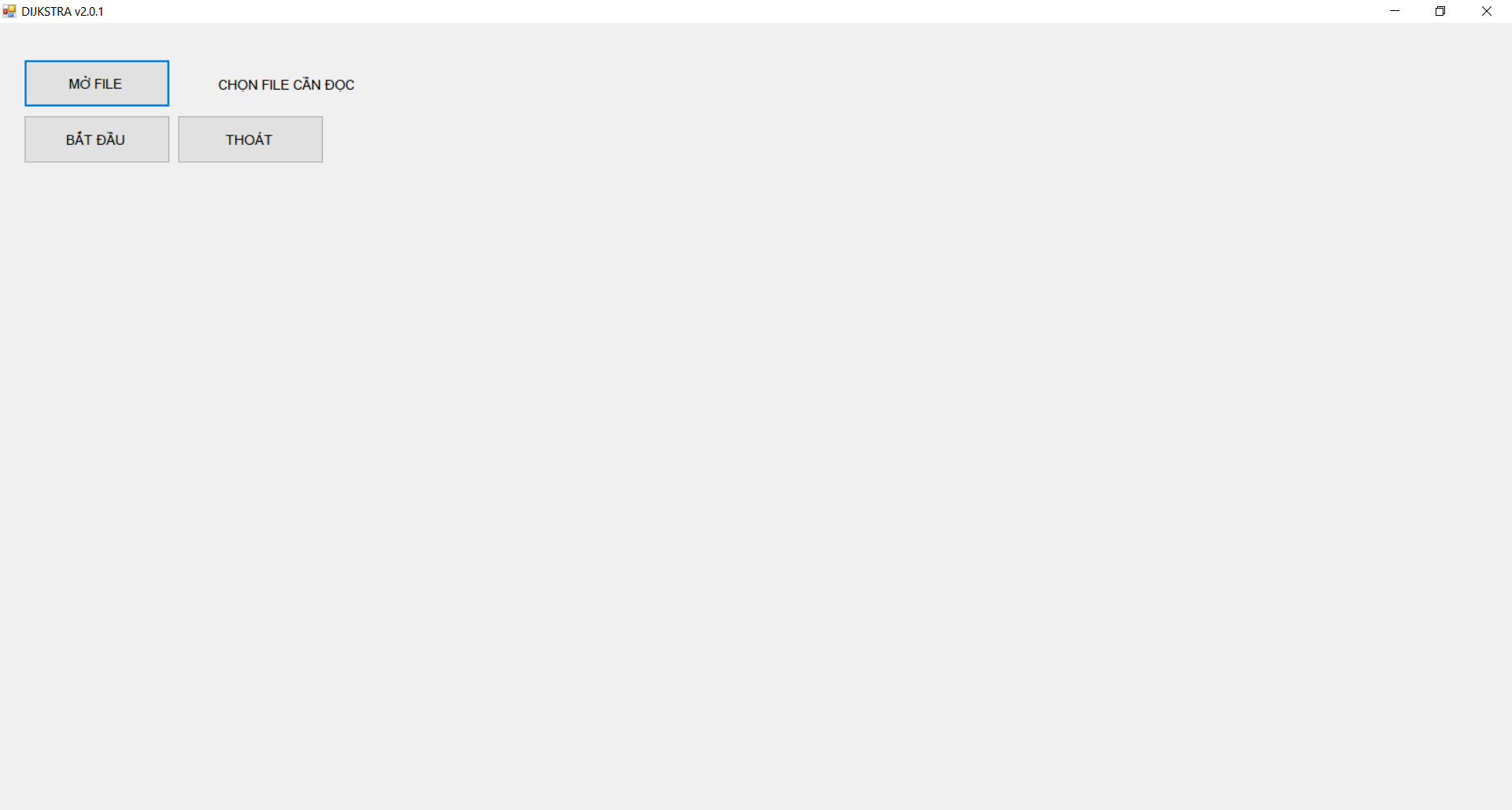
**fclose(fo);**

**printf("Hoan tat - mo file output de xem ket qua\n");**

**return 0;}**

# CHƯƠNG III: MÔ PHỎNG BẰNG ĐỒ HỌA

+ Giao diện khi chạy chương trình:

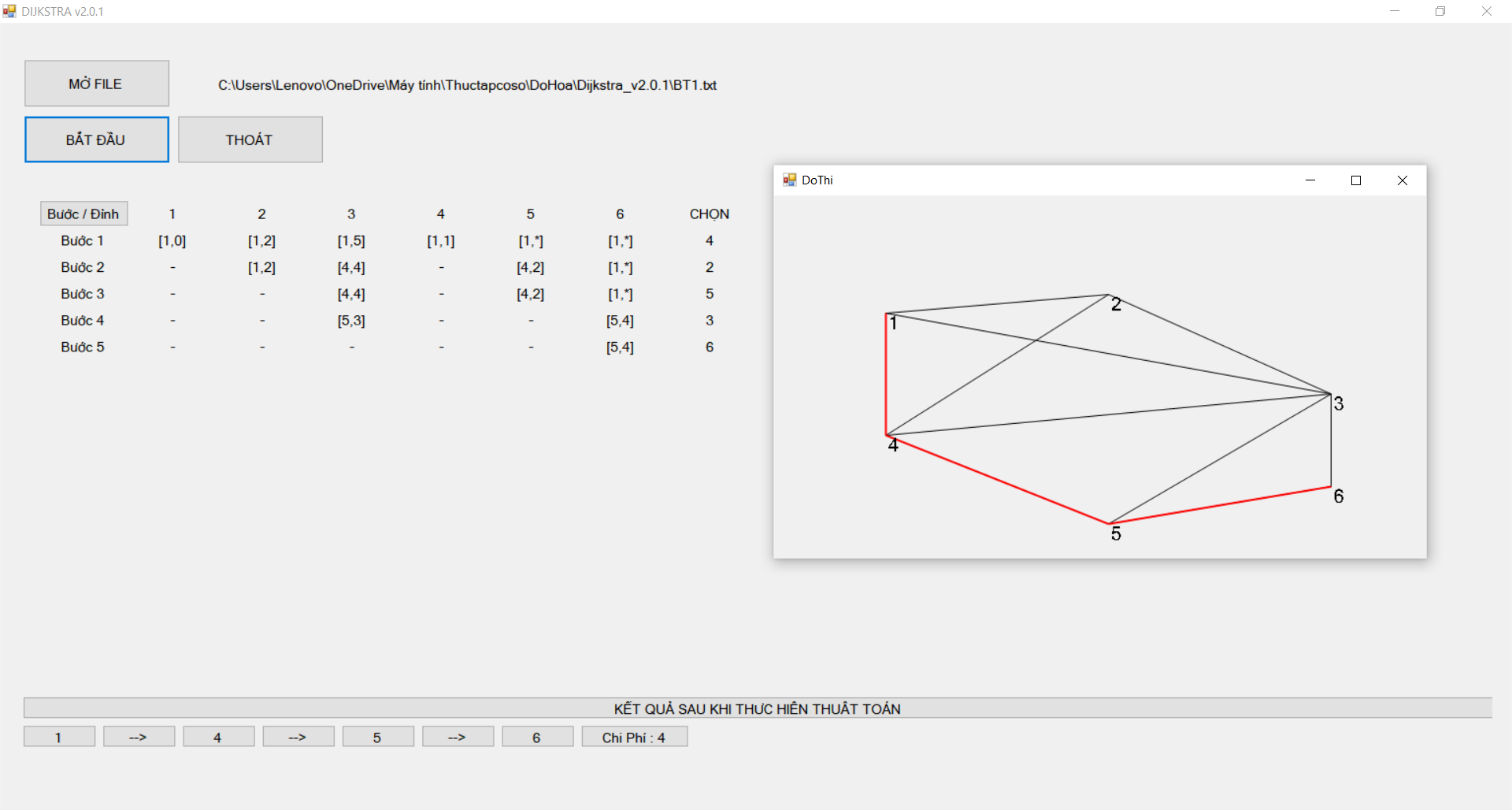


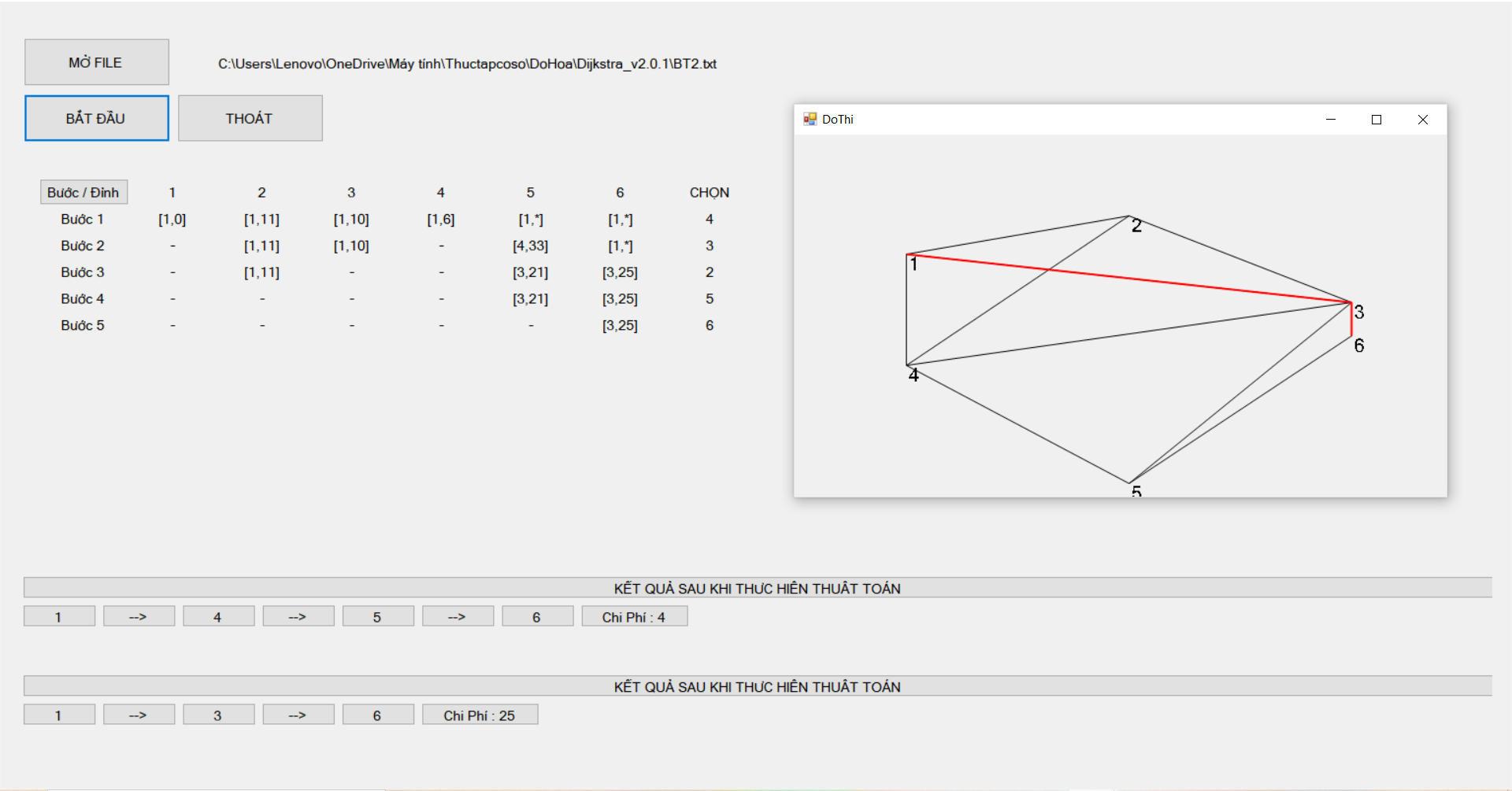
+ Mở file:





+ Bấm bắt đầu để chạy chương trình nhận dữ liệu từ file:





**KẾT LUẬN:**

Em xin phép được kết thúc bài báo cáo. Cảm ơn cô đã dành thời gian để theo dõi. Em cảm ơn.