

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
KHOA: CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN
HỌC PHẦN: TOÁN RỜI RẠC**

TÊN BÀI TẬP LỚN: LÝ THUYẾT ĐỒ THỊ

Sinh viên thực hiện	Lớp	Khóa
Hoàng Việt Dũng	DCCNTT12.10.10	K12
Bùi Đức Duy Anh	DCCNTT12.10.10	K12
Nguyễn Quý Ngọc	DCCNTT12.10.10	K12
Bùi Văn Đạt	DCCNTT12.10.10	K12
Nguyễn Trí Tường	DCCNTT12.10.10	K12

Hà Nội, năm 20.....

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
KHOA: CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: TOÁN RỜI RẠC

Nhóm:.....

TÊN (BÀI TẬP LỚN): LÝ THUYẾT ĐỒ THỊ

STT	Sinh viên thực hiện	Mã sinh viên	Điểm bằng số	Điểm bằng chữ
1	Hoàng Việt Dũng	20212818		
2	Bùi Đức Duy Anh	20212823		
3	Nguyễn Quý Ngọc	20212867		
4	Bùi Văn Đạt	20212784		
5	Nguyễn Trí Tường	20212796		
...				

CÁN BỘ CHẤM 1

(Ký và ghi rõ họ tên)

CÁN BỘ CHẤM 2

(Ký và ghi rõ họ tên)

Table of Contents

CHƯƠNG I: GIỚI THIỆU VỀ TOÁN RỜI RẠC	4
CHƯƠNG II: LÝ THUYẾT ĐỒ THỊ	4
2.1: KHÁI NIỆM ĐỒ THỊ	4
2.2: BIỂU DIỄN ĐỒ THỊ TRÊN MÁY	7
1. MA TRẬN KỀ. MA TRẬN TRỌNG SỐ	7
2. CÁC TÍNH CHẤT CỦA MA TRẬN KỀ	7
2.3: DUYỆT ĐỒ THỊ (DFS, BFS)	8
1. ĐỊNH NGHĨA CỦA BFS	8
2. ĐỊNH NGHĨA CỦA DFS	10
2.4: THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT	11
1. THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT LÀ GÌ ?	11
2. Ý TƯỞNG MÔ TẢ THUẬT TOÁN DIJKSTRA	11
2.5: THUẬT TOÁN TÌM CÂY KHUNG NHỎ NHẤT	15
1. Định nghĩa Cây khung	15
2. Bài toán tìm cây khung nhỏ nhất	15
3. Thuật toán Kruskal	16
4. Thuật toán Prim	18
CHƯƠNG III: CODE THUẬT TOÁN VÀ DEMO CÀI ĐẶT THUẬT TOÁN	
3.1 : Code Thuật Toán	21
3.2 : Các Thuật Toán Sử Dụng Trong Code Cài Thuật Toán Và Chạy Demo	
Thuật Toán Trong Menu	32
Tổng Kết Bài Tập Lớn	37

CHƯƠNG I: GIỚI THIỆU VỀ TOÁN RỜI RẠC

- **Toán rời rạc** là một lĩnh vực của toán học nghiên cứu các đối tượng rời rạc. Chúng ta sẽ sử dụng công cụ của toán rời rạc khi phải đếm các đối tượng, khi nghiên cứu quan sát giữa các tập rời rạc, khi phân tích các quá trình hữu hạn. Một trong những nguyên nhân chủ yếu làm nâng tầm quan trọng của toán rời rạc là việc cất giữ và xử lý thông tin trên máy tính bản chất là quá trình rời rạc.

- Toán rời rạc bao gồm 3 phần sau đây:

- **Phần I** : Trình bày các vấn đề của lý thuyết tổ hợp xoay quanh 4 bài toán cơ bản : Bài toán đếm, Bài toán tồn tại, Bài toán liệt kê và Bài toán tối ưu tổ hợp. Nội dung của phần I không những giúp nâng cao tư duy toán, mà còn làm quen với tư duy thuật toán trong việc giải quyết các vấn đề thực tế, đồng thời cũng rèn luyện kỹ thuật lập trình các bài toán tổ hợp

- **Phần II**: Đề cập đến lý thuyết đồ thị - một cấu trúc rời rạc tìm được những ứng dụng rộng rãi trong nhiều lĩnh vực của khoa học kỹ thuật và đời sống. Trong phần này sau phần 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, Bài toán đường đi ngắn nhất, Bài toán luồng cực đại trong mạng...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.

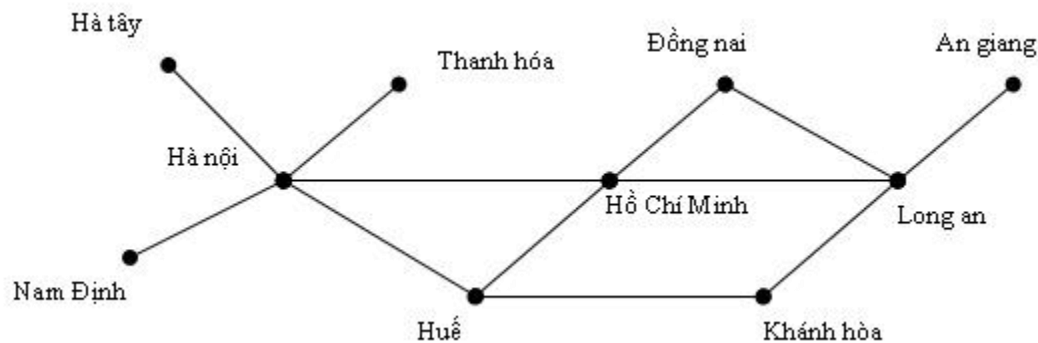
- **Phần III**: Liên quan đến lý thuyết hàm đại số logic là cơ sở để nắm bắt những vấn đề phức tạp của kỹ thuật máy tính. Sau phần trình bày các khái niệm cơ bản, phần này đi sâu vào vấn đề tối thiểu hóa các hàm đại số logic và mô tả một số thuật toán quan trọng để giải quyết vấn đề đặt ra như thuật toán Quine - McCluskey, Black - Poreski.

CHƯƠNG II: LÝ THUYẾT ĐỒ THỊ

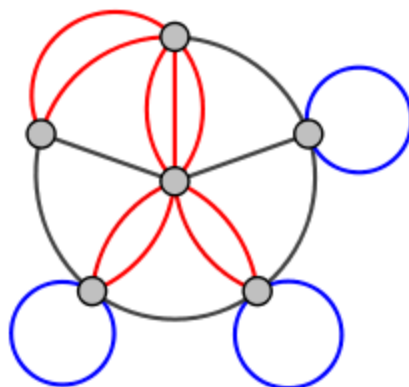
2.1: KHÁI NIỆM ĐỒ THỊ

- Đồ thị là một cấu trúc rời rạc bao gồm các đỉnh và các cạnh nối các đỉnh này. • Phân biệt các loại đồ thị khác nhau bởi kiểu và số lượng cạnh nối hai đỉnh nào đó của đồ thị.

Định nghĩa 1 (Đơn đồ thị). θ Đơn đồ thị vô hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh

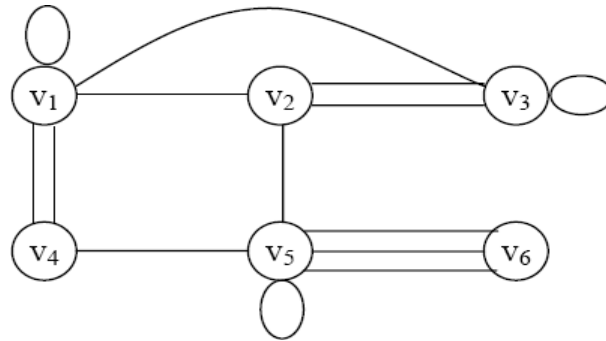


Định nghĩa 2 (Đa đồ thị). θ Đa đồ thị vô hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Hai cạnh e_1 và e_2 được gọi là cạnh lặp (bội hay song song) nếu chúng cùng tương ứng với một cặp đỉnh. θ Mỗi đơn đồ thị là đa đồ thị, nhưng không phải đa đồ thị nào cũng là đơn đồ thị, vì trong đa đồ thị có thể có hai (hoặc nhiều hơn) cạnh nối một cặp đỉnh nào đó.

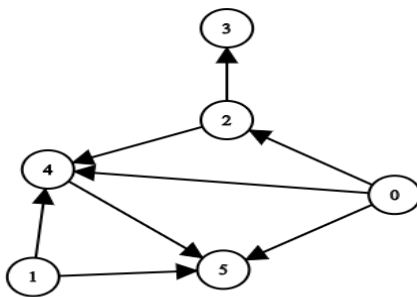


Định nghĩa 3 (Giả đồ thị). θ Giả đồ thị vô hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng và E là tập các cặp không có thứ tự gồm hai phần tử (không

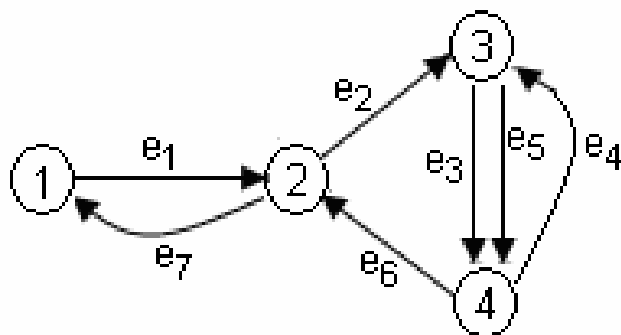
nhất thiết phải khác nhau) của V gọi là cạnh. Với $v \in V$, nếu $(v,v) \in E$ thì ta nói có một khuyên tại đỉnh v



Định nghĩa 4 (Đơn đồ thị có hướng). θ Đơn đồ thị có hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung



Định nghĩa 5 (Đa đồ thị có hướng). θ Đa đồ thị có hướng $G = (V, E)$ bao gồm V là tập các đỉnh khác rỗng và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung. Hai cung e_1, e_2 tương ứng với cùng một cặp đỉnh được gọi là cung lặp.



2.2: BIỂU DIỄN ĐỒ THỊ TRÊN MÁY

1. MA TRẬN KỀ. MA TRẬN TRONG SỐ

Xét đơn đồ thị vô hướng $G=(V,E)$, với tập đỉnh $V=\{ 1, 2, \dots, n\}$, tập cạnh $E=\{ e_1, e_2, \dots, e_m\}$. Ta gọi ma trận kề của đồ thị G là ma trận.

$$A=\{ a_{i,j} : i,j=1, 2, \dots, n\}$$

Với các phần tử được xác định theo qui tắc sau đây:

$$a_{i,j} = 0, \text{ nếu } (i,j) \notin E \text{ và}$$

$$a_{i,j} = 1, \text{ nếu } (i,j) \in E, i, j=1, 2, \dots, n.$$

2. CÁC TÍNH CHẤT CỦA MA TRẬN KỀ

1) Rõ ràng ma trận kề của đồ thị vô hướng là ma trận đối xứng, tức là

$$a[i,j]=a[j,i], i,j=1,2,\dots,n.$$

Ngược lại, mỗi $(0,1)$ -ma trận đối xứng cấp n sẽ tương ứng, chính xác đến cách đánh số đỉnh (còn nói là: chính xác đến đẳng cấu), với một đơn đồ thị vô hướng n đỉnh.

2) Tổng các phần tử trên dòng i (cột j) của ma trận kề chính bằng bậc của đỉnh i (đỉnh j).

3) nếu ký hiệu

$$a_{ij}^p, i, j=1, 2, \dots, n$$

là phần tử của ma trận

$$A^p = A.A. \dots A$$

p thừa số

Khi đó

$$a_{ij}^p, i, j=1, 2, \dots, n$$

cho ta số đường đi khác nhau từ đỉnh i đến đỉnh j qua $p-1$ đỉnh trung gian.

Ma trận kề của đồ thị có hướng được định nghĩa một cách hoàn toàn tương tự.

2.3: DUYỆT ĐỒ THỊ (DFS, BFS)

1. ĐỊNH NGHĨA CỦA BFS

Breadth First Search (BFS) là phương pháp di chuyển ngang được sử dụng trong biểu đồ. Nó sử dụng một hàng đợi để lưu trữ các đỉnh đã truy cập. Trong phương pháp này, phần nhân mạnh nằm trên các đỉnh của đồ thị, một đỉnh được chọn lúc đầu sau đó được truy cập và đánh dấu. Các đỉnh liền kề với đỉnh được truy cập sau đó được truy cập và lưu trữ trong hàng đợi một cách tuần tự.

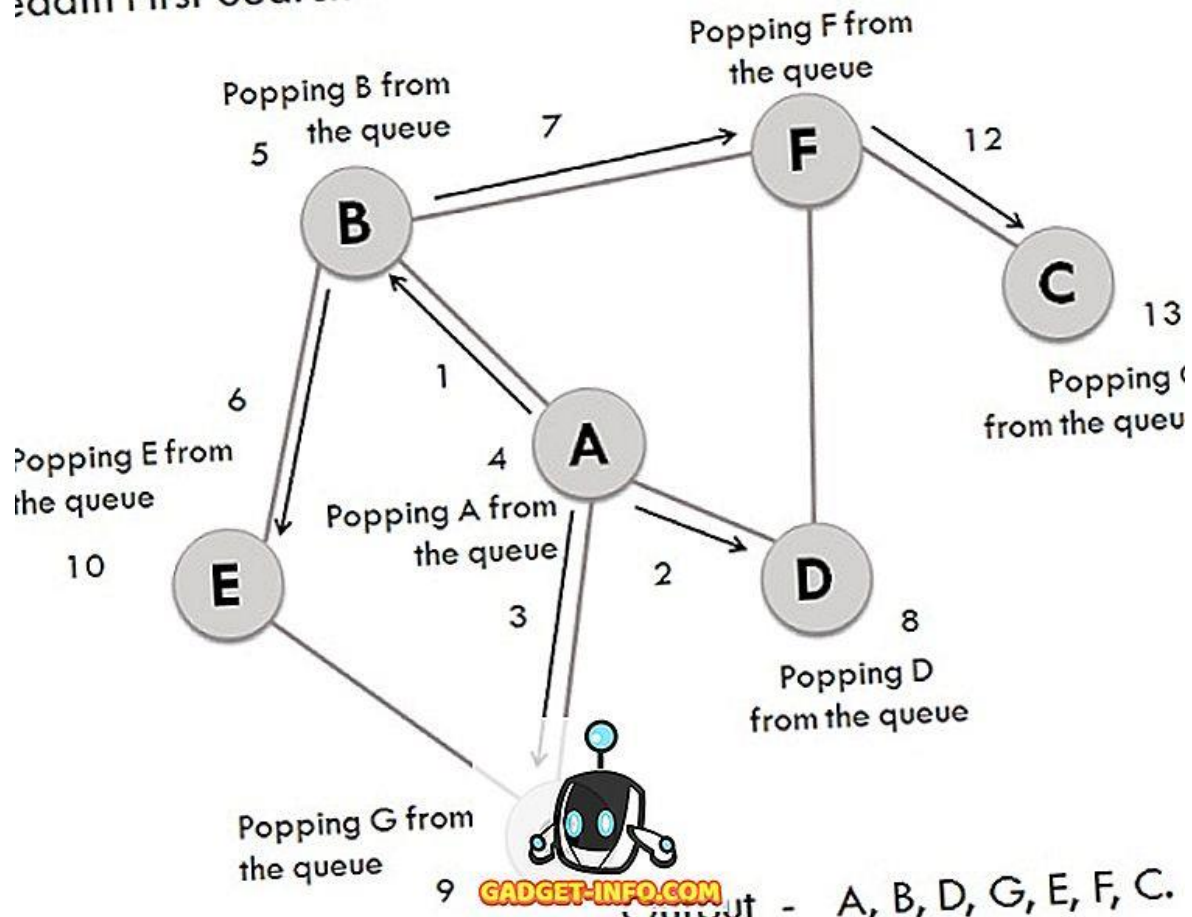
Tương tự, các đỉnh được lưu trữ sau đó được xử lý từng cái một và các đỉnh liền kề của chúng được truy cập. Một nút được khám phá đầy đủ trước khi truy cập bất kỳ nút nào khác trong biểu đồ, nói cách khác, nó đi qua các nút chưa được khám phá nông nhất trước tiên.

- Ví dụ:

- Chúng ta có một đồ thị có các đỉnh là A, B, C, D, E, F, G. Coi A là điểm bắt đầu. Các bước liên quan đến quy trình là:
- Vertex A được mở rộng và lưu trữ trong hàng đợi.

- Các kế tiếp B, D và G của A, được mở rộng và lưu trữ trong hàng đợi trong khi Vertex A bị xóa.
- Bây giờ B ở đầu trước của hàng đợi được loại bỏ cùng với việc lưu trữ các đỉnh kế tiếp E và F.
- Vertex D ở đầu trước của hàng đợi được loại bỏ và nút F được kết nối của nó đã được truy cập.
- Vertex G bị xóa khỏi hàng đợi và nó có E kế tiếp đã được truy cập.
- Bây giờ E và F được xóa khỏi hàng đợi và đỉnh C kế tiếp của nó được duyệt và lưu trong hàng đợi.
- Cuối cùng C cũng bị loại bỏ và hàng đợi trống có nghĩa là chúng ta đã hoàn thành.
- Đầu ra được tạo là - A, B, D, G, E, F, C.

Depth First Search -



Các ứng dụng

BFS có thể hữu ích trong việc tìm kiếm xem biểu đồ có kết nối các thành phần hay không. Và nó cũng có thể được sử dụng trong việc phát hiện đồ thị lưỡng cực .

Một đồ thị là lưỡng cực khi các đỉnh đồ thị được chia thành hai bộ khác nhau; không có hai đỉnh liên kề sẽ nằm trong cùng một tập hợp. Một phương pháp khác để kiểm tra đồ thị lưỡng cực là kiểm tra sự xuất hiện của một chu kỳ lẻ trong biểu đồ. Một đồ thị lưỡng cực không được chứa chu kỳ lẻ.

BFS cũng tốt hơn trong việc tìm ra con đường ngắn nhất trong biểu đồ có thể được xem như một mạng.

2. ĐỊNH NGHĨA CỦA DFS

Depth First Search (DFS) là phương pháp di chuyển ngang tìm kiếm sâu sử dụng ngăn xếp để lưu trữ các đỉnh đã truy cập. DFS là phương pháp dựa trên cạnh và hoạt động theo kiểu đệ quy trong đó các đỉnh được khám phá dọc theo một đường dẫn (cạnh). Việc thăm dò một nút bị đình chỉ ngay khi tìm thấy một nút chưa được khám phá khác và các nút chưa được khám phá sâu nhất được duyệt qua trước hết. DFS di chuyển / truy cập mỗi đỉnh chính xác một lần và mỗi cạnh được kiểm tra chính xác hai lần.

Ví dụ

- Tương tự như BFS, hãy lấy cùng một biểu đồ để thực hiện các hoạt động DFS và các bước liên quan là:
- Coi A là đỉnh bắt đầu được khám phá và lưu trữ trong ngăn xếp.
- B đỉnh kế tiếp của A được lưu trữ trong ngăn xếp.
- Vertex B có hai người kế vị E và F, trong số đó, bản chữ cái E được khám phá đầu tiên và được lưu trữ trong ngăn xếp.
- Sự kế thừa của đỉnh E, tức là G được lưu trữ trong ngăn xếp.
- Vertex G có hai đỉnh được kết nối và cả hai đều đã được truy cập, do đó G được bật ra khỏi ngăn xếp.
- Tương tự, E s cũng bị loại bỏ.
- Bây giờ, đỉnh B nằm ở đầu ngăn xếp, một nút khác (đỉnh) F được khám phá và lưu trữ trong ngăn xếp.
- Vertex F có hai người kế nhiệm C và D, giữa họ C được duyệt trước và được lưu trữ trong ngăn xếp.

- Vertex C chỉ có một người tiền nhiệm đã được truy cập, do đó, nó bị xóa khỏi ngăn xếp.
- Bây giờ đỉnh D được kết nối với F được truy cập và lưu trữ trong ngăn xếp.
- Vì đỉnh D không có bất kỳ nút không mong muốn nào, do đó D bị loại bỏ.
- Tương tự, F, B và A cũng được bật lên.
- Đầu ra được tạo là - A, B, E, G, F, C, D.

2.4: THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT

1. THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT LÀ GÌ ?

Nhắc đến giải thuật duyệt đồ thị, chắc ai học công nghệ thông tin cũng biết đến 2 thuật toán cơ bản: Depth-First Search (DFS) , Breadth-First Search (BFS). Về mặt ý nghĩa, cả 2 thuật toán đều thực hiện trên đồ thị không có trọng số. Thuật toán DFS cho ta tìm đường đi đến đỉnh ta muốn (có thể chưa ngắn nhất).

Còn thuật toán BFS cũng tìm đường đi đến đỉnh ta muốn nhưng là đường ngắn nhất có thể. Trong đồ thị có trọng số, vấn đề sẽ khó hơn, nó nảy sinh ra 1 bài toán, đó là bài toán tìm đường đi ngắn nhất giữa 2 đỉnh.

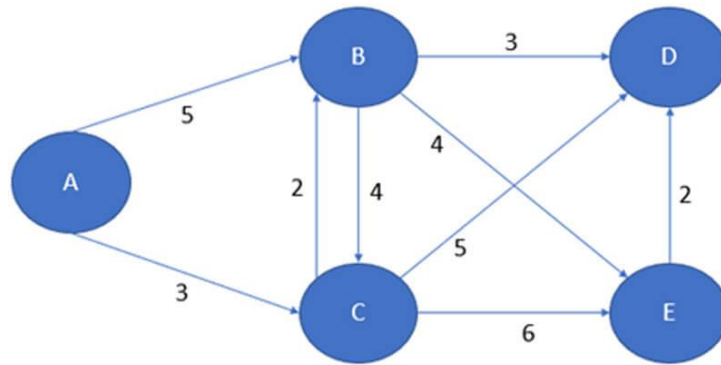
Nó là bài toán rất quen thuộc với những người bắt đầu học Graph. Để giải quyết bài toán này, đến giờ có nhiều thuật toán và các biến thể. Trong bài viết này mình sẽ chắc lọc và lựa chọn sử dụng Moore – Dijkstra.

Thuật toán Dijkstra cho phép tìm đường đi ngắn nhất từ một đỉnh **s** đến các đỉnh còn lại của đồ thị và chiều dài (trọng số) tương ứng. Phương pháp của thuật toán là xác định tuần tự đỉnh có chiều dài đến **s** theo thứ tự tăng dần. Thuật toán được xây dựng trên cơ sở gán cho mỗi đỉnh các nhãn tạm thời. Nhãn tạm thời của các đỉnh cho biết cận trên của chiều dài đường đi ngắn nhất từ **s** đến đỉnh đó. Nhãn của các đỉnh sẽ biến đổi trong các bước lặp, mà ở mỗi bước lặp sẽ có một nhãn tạm thời trở thành chính thức.

Nếu nhãn của một đỉnh nào đó trở thành chính thức thì đó cũng chính là chiều dài ngắn nhất của đường đi từ **s** đến đỉnh đó.

2. Ý TƯỞNG MÔ TẢ THUẬT TOÁN DIJKSTRA

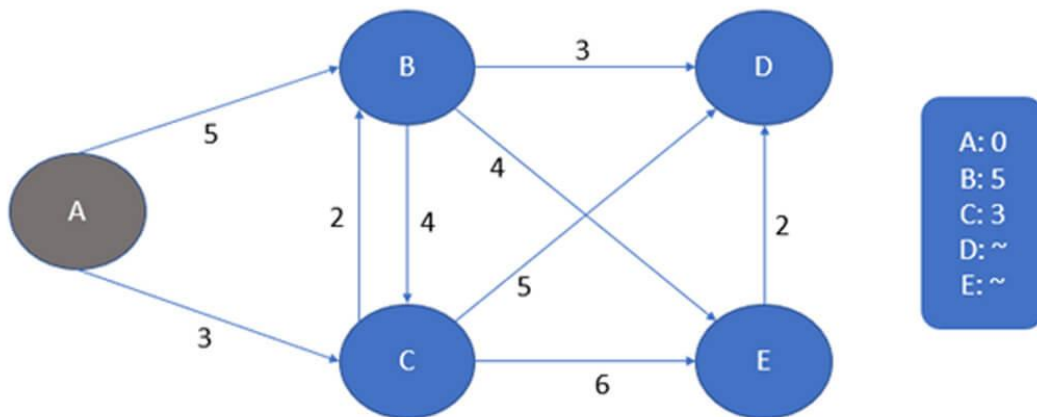
Trong phần này, chúng ta sẽ phân tích từng bước Thuật toán Dijkstra. Ở đây ta sẽ sử dụng đồ thị bên dưới để làm ví dụ để giúp bạn hiểu rõ hơn về thuật toán này.



Như bạn đã biết, thuật toán của Dijkstra là một thuật toán Greedy (Thuật toán tham lam) Điều này có nghĩa là chúng ta sẽ đi một con đường ngắn hơn từ đỉnh này đến đỉnh kia. Thuật toán hoàn tất khi chúng ta truy cập tất cả các đỉnh của đồ thị.

Tuy nhiên, hãy cẩn thận - đôi khi khi chúng ta tìm thấy một đỉnh mới, có thể có các đường đi ngắn hơn qua nó từ một đỉnh đã truy cập đến một đỉnh đã được truy cập khác.

Chúng ta có thể xem bên dưới các bước để hoàn thành thuật toán Dijkstra.

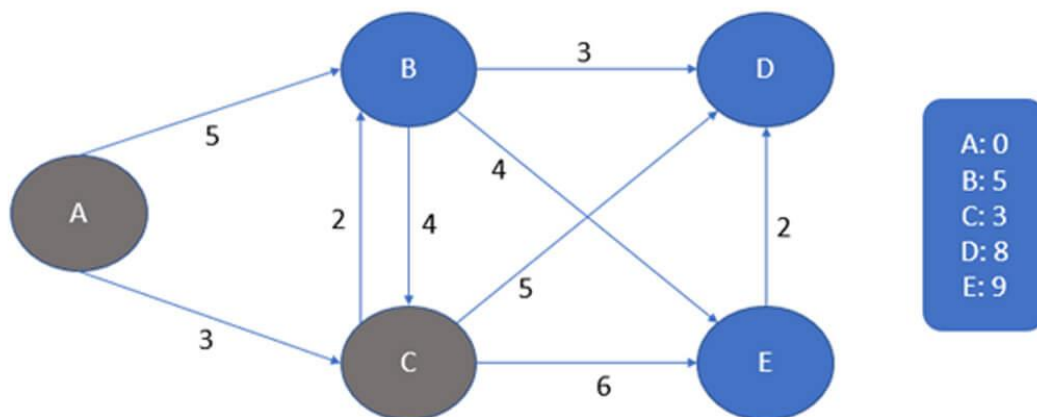


Chúng ta có thể bắt đầu với nút **A** và chúng ta có 2 con đường.

- Đầu tiên là từ **A** đến **B** với độ dài **5**
- Và từ **A** đến **C** với độ dài **3**.

Vì vậy, chúng ta có thể viết trong danh sách kế bên với các đỉnh đã truy cập là 2 đỉnh mới (**B**, **C**) và trọng số đề đến đó.

Sau đó, như đã nói trước đó - chúng ta sẽ chọn con đường từ **A** -> **C**.

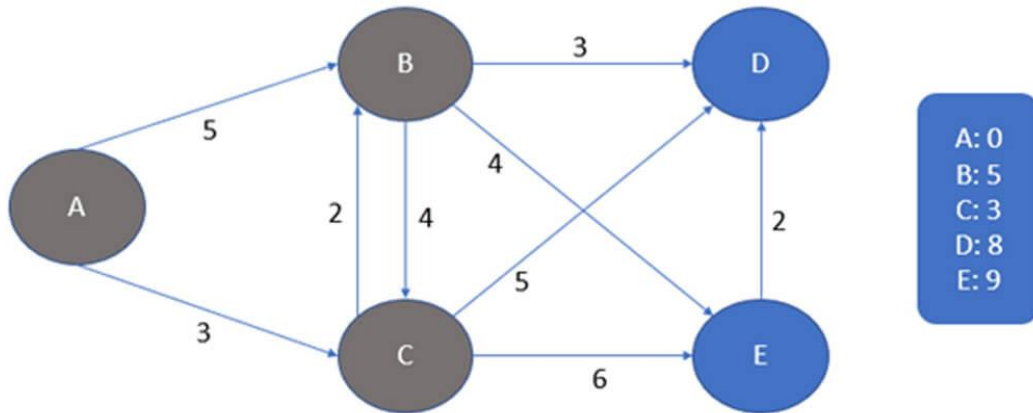


Khi truy cập vào đỉnh **C**, chúng ta có thể thấy rằng có 3 đường đi khác nhau:

- Con đường đầu tiên là **C** đến **B**

- Con đường thứ hai là **C** đến **D**
- Con đường thứ ba là **C** đến **E**

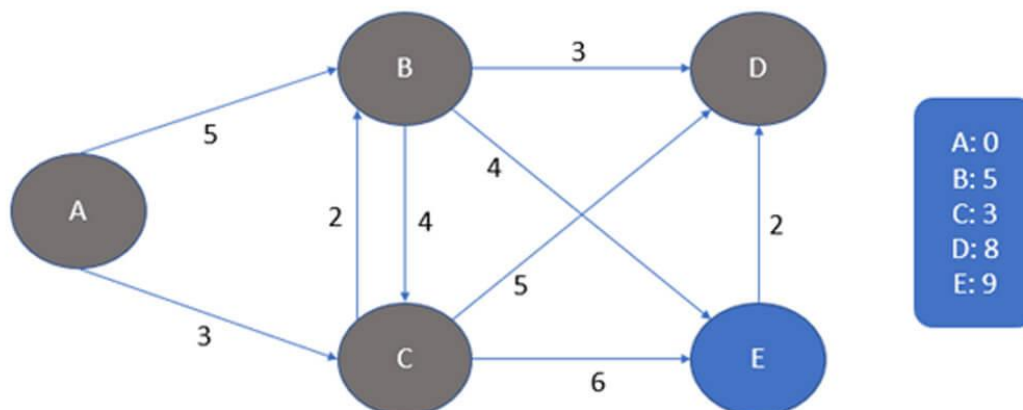
Vì vậy, hãy ghi vào danh sách hai đỉnh mới và chọn con đường ngắn nhất là **C** đến **B**.



Bây giờ tại **B**, chúng ta có 3 đường:

- **B** đến **D**
- **B** đến **E**
- Và **B** quay lại **C**

Chúng ta chọn con đường ngắn nhất là **B** đến **D** và chúng ta cập nhật vào danh sách trọng số mới của các đường đi từ **A** đến các đỉnh khác.



Bây giờ như chúng ta có thể thấy không có đường đi mới nào từ **D** đến **E**. Trong trường hợp đó, chúng ta quay lại đỉnh trước đó để kiểm tra đường đi ngắn nhất. Bây giờ có một đường với độ dài **4** đi đến **E** và một đường đi đến **C**.

Trong trường hợp này, chúng ta chọn bất kỳ đường nào chúng ta thích. Cuối cùng, chúng ta có thể thấy rằng bất kỳ phương án nào chúng ta đi trên đường từ **A** đến **E** đều có trọng số như nhau vì các đường đi ngắn nhất được ghi trong danh sách. Cuối cùng, chúng ta có thể thấy tất cả các đường đi mà chúng ta đã sử dụng.

2.5: THUẬT TOÁN TÌM CÂY KHUNG NHỎ NHẤT

1. Định nghĩa Cây khung

Trong đồ thị liên thông G , nếu ta loại bỏ cạnh nằm trên chu trình nào đó thì ta sẽ được đồ thị vẫn là liên thông. Nếu cứ loại bỏ các cạnh ở các chu trình khác cho đến khi nào đồ thị không còn chu trình (vẫn liên thông) thì ta thu được một cây nối các đỉnh của G . Cây đó gọi là cây khung hay cây bao trùm của đồ thị G .

Tổng quát, nếu G là đồ thị có n đỉnh, m cạnh và k thành phần liên thông thì áp dụng thủ tục vừa mô tả đối với mỗi thành phần liên thông của G , ta thu được đồ thị gọi là rừng khung của G . Số cạnh bị loại bỏ trong thủ tục này bằng $m-n+k$, số này ký hiệu là $n(G)$ và gọi là chu số của đồ thị G .

2. Bài toán tìm cây khung nhỏ nhất

Bài toán tìm cây khung nhỏ nhất của đồ thị là một trong số những bài toán tối ưu trên đồ thị tìm được ứng dụng trong nhiều lĩnh vực khác nhau của đời sống. Trong phần này ta sẽ có hai thuật toán cơ bản để giải bài toán này. Trước hết, nội dung của bài toán được phát biểu như sau.

Cho $G=(V,E)$ là đồ thị vô hướng liên thông có trọng số, mỗi cạnh $e \in E$ có trọng số $m(e) \geq 0$. Giả sử $T=(V_T, E_T)$ là cây khung của đồ thị G ($V_T=V$). Ta gọi độ dài $m(T)$ của cây khung T là tổng trọng số của các cạnh của nó:

$$m(T) = \sum_{e \in E_T} m(e).$$

Bài toán đặt ra là trong số tất cả các cây khung của đồ thị G , hãy tìm cây khung có độ dài nhỏ nhất. Cây khung như vậy được gọi là cây khung nhỏ nhất của đồ thị và bài toán đặt ra được gọi là bài toán tìm cây khung nhỏ nhất.

Để minh họa cho những ứng dụng của bài toán cây khung nhỏ nhất, dưới đây là hai mô hình thực tế tiêu biểu cho nó.

Bài toán xây dựng hệ thống đường sắt: Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất cứ một thành phố nào đến bất kỳ một trong số các thành phố còn lại. Mặt khác, trên quan điểm kinh tế đòi hỏi là chi phí về xây dựng hệ thống đường phải là nhỏ nhất. Rõ ràng là đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng, với phương án xây dựng tối ưu phải là cây. Vì vậy, bài toán đặt ra dẫn về bài toán tìm cây khung nhỏ nhất trên đồ thị đầy đủ n đỉnh, mỗi đỉnh tương ứng với một thành phố với độ dài trên các cạnh chính là chi phí xây dựng hệ thống đường sắt nối hai thành phố.

Bài toán nối mạng máy tính: Cần nối mạng một hệ thống gồm n máy tính đánh số từ 1 đến n . Biết chi phí nối máy i với máy j là $m(i,j)$ (thông thường chi phí này phụ thuộc vào độ dài cáp nối cần sử dụng). Hãy tìm cách nối mạng sao cho tổng chi phí là nhỏ nhất. Bài toán này cũng dẫn về bài toán tìm cây khung nhỏ nhất.

Bài toán tìm cây khung nhỏ nhất đã có những thuật toán rất hiệu quả để giải chúng. Ta sẽ xét hai trong số những thuật toán như vậy: thuật toán Kruskal và thuật toán Prim.

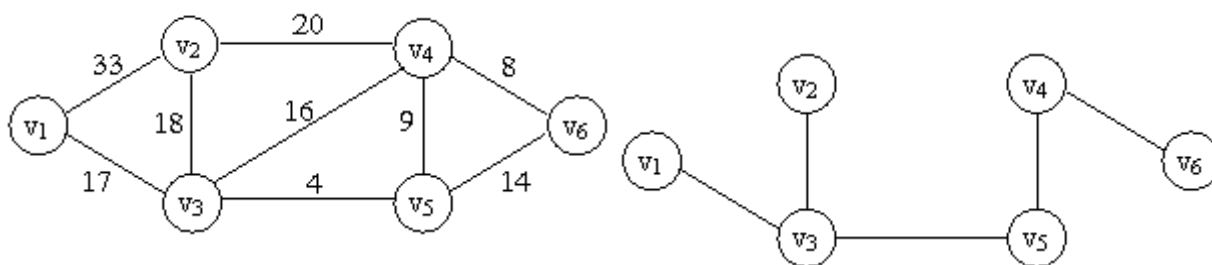
3. Thuật toán Kruskal

Thuật toán sẽ xây dựng tập cạnh E_T của cây khung nhỏ nhất $T=(V_T, E_T)$ theo từng bước. Trước hết sắp xếp các cạnh của đồ thị G theo thứ tự không giảm của

trọng số. Bắt đầu từ $E_T = \emptyset$, ở mỗi bước ta sẽ lần lượt duyệt trong danh sách cạnh đã sắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bổ sung nó vào tập E_T không tạo thành chu trình trong tập này. Thuật toán sẽ kết thúc khi ta thu được tập E_T gồm $n-1$ cạnh. Cụ thể có thể mô tả như sau:

- 1. Bắt đầu từ đồ thị rỗng T có n đỉnh.
- 2. Sắp xếp các cạnh của G theo thứ tự không giảm của trọng số.
- 3. Bắt đầu từ cạnh đầu tiên của dãy này, ta cứ thêm dần các cạnh của dãy đã được xếp vào T theo nguyên tắc cạnh thêm vào không được tạo thành chu trình trong T .
- 4. Lặp lại Bước 3 cho đến khi nào số cạnh trong T bằng $n-1$, ta thu được cây khung nhỏ nhất cần tìm.

Ví dụ 1: Tìm cây khung nhỏ nhất của đồ thị cho trong hình dưới đây:



Bắt đầu từ đồ thị rỗng T có 6 đỉnh.

Sắp xếp các cạnh của đồ thị theo thứ tự không giảm của trọng số:

$\{(v_3, v_5), (v_4, v_6), (v_4, v_5), (v_5, v_6), (v_3, v_4), (v_1, v_3), (v_2, v_3), (v_2, v_4), (v_1, v_2)\}$.

Thêm vào đồ thị T cạnh (v_3, v_5) .

Do số cạnh của T là $1 < 6-1$ nên tiếp tục thêm cạnh (v_4, v_6) vào T . Bây giờ số cạnh của T đã là 2 vẫn còn nhỏ hơn 6, ta tiếp tục thêm cạnh tiếp theo trong dãy đã sắp xếp vào T . Sau khi thêm cạnh (v_4, v_5) vào T , nếu thêm cạnh (v_5, v_6) thì nó sẽ tạo thành với 2 cạnh $(v_4, v_5), (v_4, v_6)$ đã có trong T một chu trình. Tình huống tương tự cũng xảy ra đối với cạnh (v_3, v_4) là cạnh tiếp theo trong dãy. Tiếp theo ta bổ sung cạnh $(v_1, v_3), (v_2, v_3)$ vào T và thu được tập E_T gồm 5 cạnh:

$\{(v_3, v_5), (v_4, v_6), (v_4, v_5), (v_1, v_3), (v_2, v_3)\}$.

Tính đúng đắn của thuật toán: Rõ ràng đồ thị thu được theo thuật toán có $n-1$ cạnh và không có chu trình. Vì vậy theo Định lý 6.1.3, nó là cây khung của đồ thị G . Như vậy chỉ còn phải chỉ ra rằng T có độ dài nhỏ nhất. Giả sử tồn tại cây khung S của đồ thị mà $m(S) < m(T)$. Ký hiệu e_k là cạnh đầu tiên trong dãy các cạnh

của T xây dựng theo thuật toán vừa mô tả không thuộc S . Khi đó đồ thị con của G sinh bởi cây S được bổ sung cạnh e_k sẽ chứa một chu trình duy nhất C đi qua e_k . Do chu trình C phải chứa cạnh e thuộc S nhưng không thuộc T nên đồ thị con thu được từ S bằng cách thay cạnh e của nó bởi e_k , ký hiệu đồ thị này là S' , sẽ là cây khung. Theo cách xây dựng, $m(e_k) \leq m(e)$, do đó $m(S') \leq m(S)$, đồng thời số cạnh chung của S' và T đã tăng thêm một so với số cạnh chung của S và T . Lặp lại quá trình trên từng bước một, ta có thể biến đổi S thành T và trong mỗi bước tổng độ dài không tăng, tức là $m(T) \leq m(S)$. Mâu thuẫn này chứng tỏ T là cây khung nhỏ nhất của G .

Độ phức tạp của thuật toán Kruskal được đánh giá như sau. Trước tiên, ta sắp xếp các cạnh của G theo thứ tự có chiều dài tăng dần; việc sắp xếp này có độ phức tạp $O(p^2)$, với p là số cạnh của G . Người ta chứng minh được rằng việc chọn e_{i+1} không tạo nên chu trình với i cạnh đã chọn trước đó có độ phức tạp là $O(n^2)$. Do $p \leq n(n-1)/2$, thuật toán Kruskal có độ phức tạp là $O(p^2)$.

4. Thuật toán Prim

Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị dày (đồ thị có số cạnh $m \gg n(n-1)/2$). Trong trường hợp đó, thuật toán Prim tỏ ra hiệu quả hơn. Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất.

- **1.** $V_T := \{v_*\}$, trong đó v_* là đỉnh tùy ý của đồ thị G .

$E_T := \emptyset$.

- **2.** Với mỗi đỉnh $v_j \in V_T$, tìm đỉnh $w_j \in V_T$ sao cho

$$m(w_j, v_j) = \min_{x_i \in V_T} m(x_i, v_j) =: b_j$$

và gán cho đỉnh v_j nhãn $[w_j, b_j]$. Nếu không tìm được w_j như vậy (tức là khi v_j không kề với bất cứ đỉnh nào trong V_T) thì gán cho v_j nhãn $[0, \infty]$.

- **3.** Chọn đỉnh v_{j^*} sao cho

$$b_{j^*} = \min_{v_j \in V_T} b_j$$

$$V_T := V_T \cup \{v_{j^*}\},$$

$$E_T := E_T \cup \{(w_{j^*}, v_{j^*})\}.$$

Nếu $|V_T| = n$ thì thuật toán dừng và (V_T, E_T) là cây khung nhỏ nhất.

Nếu $|V_T| < n$ thì chuyển sang Bước 4.

- **4.** Đối với tất cả các đỉnh $v_j \in V_T$ mà kề với v_{j^*} , ta thay đổi nhãn của chúng như sau:

Nếu $b_j > m(v_{j*}, v_j)$ thì đặt $b_j := m(v_{j*}, v_j)$ và nhãn của v_j là $[v_{j*}, b_j]$. Ngược lại, ta giữ nguyên nhãn của v_j . Sau đó quay lại Bước 3.

Ví dụ 2: Tìm cây khung nhỏ nhất bằng thuật toán Prim của đồ thị gồm các đỉnh A, B, C, D, E, F, H, I được cho bởi ma trận trọng số sau.

		B								
		A	C	D	E	F	H	I		
		∞	15	16	19	23	20	32	18	
A		15	∞	33	13	34	19	20	12	
B		16	33	∞	13	29	21	20	19	
C		19	13	13	∞	22	30	21	11	D
		23	34	29	22	∞	34	23	21	E
		20	19	21	30	34	∞	17	18	F
		32	20	20	21	23	17	∞	14	H
I		18	12	19	11	21	18	14	∞	

Yêu cầu viết các kết quả trung gian trong từng bước lặp, kết quả cuối cùng cần đưa ra tập cạnh và độ dài của cây khung nhỏ nhất.

V.lặp	A	B	C	D	E	F	H	I	V_T	E_T
K.tạo	-	[A,15]	[A,16]	[A,19]	[A,23]	[A,20]	[A,32]	[A,18]	A	ặ
1	-	-	[A,16]	[B,13]	[A,23]	[B,19]	[B,20]	[B,12]	A, B	(A,B)
2	-	-	[A,16]	[I,11]	[I,21]	[I,18]	[I,14]	-	A, B, I	(A,B), (B,I)
3	-	-	[D,13]	-	[I,21]	[I,18]	[I,14]	-	A, B, I, D	(A,B), (B,I), (I,D)

4	-	-	-	-	[I,21]	[I,18]	[I,14]	-	A, B, I, D, C	(A,B), (B,I), (I,D), (D,C)
5	-	-	-	-	[I,21]	[H,17]	-	-	A, B, I, D, C, H	(A,B), (B,I), (I,D), (D,C), (I,H)
6	-	-	-	-	[I,21]	-	-	-	A, B, I, D, C, H, F	(A,B), (B,I), (I,D), (D,C), (I,H), (H,F)
7	-	-	-	-	-	-	-	-	A, B, I, D, C, H, F, E	(A,B), (B,I), (I,D), (D,C), (I,H), (H,F), (I,E)

Vậy độ dài cây khung nhỏ nhất là:

$$15 + 12 + 11 + 13 + 14 + 17 + 21 = 103.$$

Tính đúng đắn của thuật toán: Để chứng minh thuật toán Prim là đúng, ta chứng minh bằng quy nạp rằng $T(k)$ ($k=1, 2, \dots, n$), đồ thị nhận được trong vòng lặp thứ k , là một đồ thị con của cây khung nhỏ nhất của G , do đó $T(n)$ chính là một cây khung nhỏ nhất của G .

$T(1)$ chỉ gồm đỉnh v_* của G , do đó $T(1)$ là đồ thị con của mọi cây khung của G . Giả sử $T(i)$ ($1 \leq i < n$) là một đồ thị con của một cây khung nhỏ nhất của G . Ta chứng minh rằng $T(i+1)$ cũng là đồ thị con của một cây khung nhỏ nhất.

Thật vậy, theo thuật toán Prim $E_{T(i+1)} = E_{T(i)} \cup \{e_{i+1}\}$, với e_{i+1} là cạnh ngắn nhất trong tất cả các cạnh có một đầu mút thuộc $V_{T(i)}$, đầu mút kia không thuộc $V_{T(i)}$.

Nếu e_{i+1} là một cạnh của T thì T_{i+1} là đồ thị con của T .

Nếu e_{i+1} không phải là một cạnh của T thì T_{i+1} là đồ thị con $T'=(V_T, E_T \setminus \{e_{i+1}\})$. Đồ thị T' chứa một chu trình sơ cấp duy nhất C (theo tính chất 6 của định lý về cây). Ta chọn trong C một cạnh e_j có một đỉnh thuộc $T(i)$ và đỉnh kia không thuộc $T(i)$ và $e_j \perp e_{i+1}$. Ta bỏ e_j trong C . Khi đó

$$T''=(V_T, E_T \setminus \{e_j\})$$

là một cây khung của G và $T(i+1)$ là đồ thị con của T' nên cũng là đồ thị con của T'' . Theo cách chọn e_{i+1} của thuật toán Prim, ta có

$$m(e_{i+1}) \leq m(e_j) \text{ do đó } m(T'') \leq m(T).$$

Nhưng T'' là một cây khung của G , còn T là cây khung nhỏ nhất, vì vậy phải có $m(T'')=m(T)$, tức là T'' cũng là cây khung nhỏ nhất của G .

Độ phức tạp của thuật toán Prim là $O(n^3)$. Thật vậy, nếu $T(k)$ có k đỉnh thì có $n-k$ đỉnh không thuộc $T(k)$, do đó ta phải chọn chiều dài nhỏ nhất của nhiều nhất là $k(n-k)$ cạnh. Do $k(n-k) < (n-1)^2$, nên độ phức tạp của bước chọn e_{k+1} là $O(n^2)$. Vì phải chọn $n-1$ cạnh, nên độ phức tạp của thuật toán Prim là $O(n^3)$.

CHƯƠNG III: CODE THUẬT TOÁN VÀ DEMO CÀI ĐẶT THUẬT TOÁN

3.1 : Code Thuật Toán

```
#include<iostream>
#include<fstream>
#include<stack>
#include<queue>
#include<stdio.h>
#include<conio.h>
#include<vector>
#include <string>
#define infinity 9999
#define MAX 20
#define MAX 100
#define TRUE 1
#define FALSE 0
using namespace std;
```

```
int n,m , minl , connect ; // so dinh cua do thi.
```

```
int dau[1001], cuoi[1001], w[1001]; //Mang chua dinh dau va cuoi cua do thi
```

```

int dau[100], cuoi[100], father[100];
int matrix[1001][1001]; //ma tran ke luu tru
bool visited[1001]; //xac nhan da duyet
// tim duong di
int forward_point[1001];
//doc ma tra
void Init()
{
    ifstream file("fileinput.txt", ios::in); //doc file text
    file >> n; //doc dinh , de luu vao dem
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            file >> matrix[i][j];
        }
        visited[i] = false; // CHUA THANG
        forward_point[i] = 0;
    }
}
// Doc ma File Cac cah
void intFile(){
    freopen("input.txt", "r", stdin);
    cin >> n >> m;
    cout << "So dinh do thi: " << n << endl;
    cout << "So canh do thi: " << m << endl;
    //Nhap danh sach ke
    for (int i = 1; i <= m; i++){

        cin >> dau[i] >> cuoi[i] >> w[i];
    }
}
// DFS
void DFS(int u)
{

```

```

cout << "DFS(" << u << "): "; // In ra lo trinh
stack<int> st; // chuyen doi tu chux sang so
st.push(u); // dau dinh vao stack
visited[u] = true; // dc tham roi
cout << u << " "; // in dinh ra
//Lap cho den khi stack rong
while(!st.empty())
{
    int v = st.top(); //lay dinh dau stack
    //tim duong di
    int s = st.top();
    // chung tim duong di va DFS
    st.pop();
    for(int t = 1; t <= n; t++)
    {
        if(visited[t] == false && matrix[v][t] == 1); //kiem tra dinh thang hay
chua va ke voi no
        if(visited[t] == false && matrix[s][t] == 1)
        {
            cout << t << " "; // di tham
            forward_point[t] = s;
            visited[t] = true; //da tham thanh true
            st.push(v); //dua vao ngan xep
            //tim duong di
            st.push(s);
            //
            st.push(t); //dua vao ngan xep
            //khi push ra phai dua lai vao
            break;
            //Chu y : Khi thang dinh xong thi se break ra ngoai va lap lai xet
thang dinh khach
        }
    }
}
}
}

```

```

// BFS
void BFS(int u)
{
    cout << "BFS(" << u << "): "; //IN RA LO TRINH
    queue<int> Q; //CHYEN DOI DINH
    Q.push(u); //DAU U VAO QUEUE
    visited[u] = true; //THANG DINH U
    while(!Q.empty())
    {
        int s = Q.front(); //LAY DINH DAU TIEN
        cout << s << " "; // DAY DINH RA MAN HINH
        Q.pop(); // LAY DINH DAU TIEN RA KHOI QUEUE
        for(int t = 1; t <= n; t++)
        {
            if(visited[t] == false && matrix[s][t] == 1) //kiem tra dinh thang hay
chua va ke voi no
            {
                Q.push(t); //CHO T VAO OUEUE
                visited[t] = true; //DA THANG
            }
        }
    }
}

// Cai dat thuat toan Dijkstra
void PrintPath(int s, int e)
{
    if(forward_point[e] == 0)
        cout << "Khong Co Con Duong Nao Giua " << s << " Va " << e << endl;
    else
    {
        cout << "Duong Dan: ";
        cout << e ;
        int u = forward_point[e];
    }
}

```



```

while(u != s)
{
    cout << " <- " << u;
    u = forward_point[u];
}
cout << " <- " << s;
}
}

```

// Cài đặt thuật toán Krusal

```
void Heap(int First, int Last){
```

```
    int a, k, t1, t2, t3;
```

```
    a = First;
```

```
    while (a <= (Last / 2)){
```

```
        if ((2 * a) < Last && w[2 * a + 1] < w[2 * a])
```

```
            k = 2 * a + 1;
```

```
        else
```

```
            k = 2 * a;
```

```
        if (w[k] < w[a]){
```

```
            t1 = dau[a];
```

```
            t2 = cuoi[a];
```

```
            t3 = w[a];
```

```
            dau[a] = dau[k];
```

```

    cuoi[a] = cuoi[k];

    w[a] = w[k];

    dau[k] = t1;

    cuoi[k] = t2;

    w[k] = t3;

    a = k;

}

else a = Last;

}

}

int Find(int i){

    int tro = i;

    while (father[tro] > 0)

        tro = father[tro];

    return(tro);

}

void Union(int i, int a){

```

```
int x = father[i] + father[a];
```

```
if (father[i] > father[a]) {
```

```
    father[i] = a;
```

```
    father[a] = x;
```

```
}
```

```
else {
```

```
    father[a] = i;
```

```
    father[i] = x;
```

```
}
```

```
}
```

```
void Krusal(void){
```

```
    int last, u, v, r1, r2, ncanh, ndinh;
```

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

```
        father[i] = -1;
```

// Su dung thuat toan vun dong de sap xep cac canh cua do thi theo thu tu
khong giam cua do canh

```
    for (int i = m / 2; i > 0; i--)
```

```
        Heap(i, m);
```

```

last = m; ncanh = 0; ndinh = 0; minl = 0; connect = TRUE;

//Lua chon canh bo xung vao cay khung

while (ndinh < n - 1 && ncanh < m){

    ncanh = ncanh + 1;

    u = dau[1];

    v = cuoi[1];

    //tim goc cua phan hach 1

    r1 = Find(u);

    //tim goc cua phan hach 2

    r2 = Find(v);

    if (r1 != r2) { //neu hai goc khac nhau thi canh co the them vao dc do thi

        ndinh = ndinh + 1;

        Union(r1, r2);

        daut[ndinh] = u;

        cuoit[ndinh] = v;

        minl = minl + w[1];

    }

```

```

    dau[1] = dau[last];

    cuoi[1] = cuoi[last];

    w[1] = w[last];

    last = last - 1;

    Heap(1, last);

}

if (ndinh != n - 1) connect = FALSE;

}

void Result(void){

    cout<<"Do dai cay khung nho nhat:"<< minl<<endl;

    cout<<"Cac canh cua cay khung nho nhat:"<<endl;

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

        cout<< dau[i]<<" "<<cuoi[i]<<endl;

}

//end
int menu(){

    int control;
    do{
        cout << "Cai Dat Thuat Toan Trong Do Thi" << endl;

```

```

        cout << "Hay chon menu sau:" << endl;
        cout << "Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)" <<
endl;
        cout << "Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong
(BFS)" << endl;
        cout << "Nhap 3: Cai Dat Thuat Toan Dijkstra" << endl;
        cout << "Nhap 4: Cai Dat Thuat Toan Krusal" << endl;
        cout << "Nhap 0: De Thoat Khoi Ung Dung" << endl;
        cout << "-----" << endl;
        cout << "Ban Chon" << endl;

    cin >> control;

    switch (control){
    case 1:
        cout << "Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)" << endl;
        // Code Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
        //DFS
        int s; // dinh bat dau.
        Init();
        cout << "Chon dinh bat dau DFS: ";
        cin >> s;//nhap s vao
        DFS(s);//goi vaf nhap s vao
        cout << "\n";
        break;
    case 2:
        cout << "Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)" <<
endl;
        // Code Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)
        //BFS
        int u; // DINH BAT DAU
        Init();
        cout << "Chon dinh bat dau BFS : ";
        cin >> u;//NHAP U VAO
        BFS(u);// GAI VA NHAP U VAO

```

```

        cout << "\n";
break;
case 3:
    cout << "Cai Dat Thuat Toan Dijkstra" << endl;
    // Code Cai Dat Thuat Toan Dijkstra
    int e;
    Init();
    cout << "Nhap Diem Dau Tien: ";
    cin >> s;
    cout << "Nhap Diem Cuoi Cung: ";
    cin >> e;
    DFS(s);
    PrintPath(s, e);
    cout << "\n";
break;
case 4:
    cout << "Cai Dat Thuat Toan Krusal" << endl;
    // Code Cai Dat Thuat Toan Krusal

    intFile();

    Krusal();
    Result();
    getch();
    return 2;
break;
case 0 :
    cout << "Ban da thoat chuong trinh" << endl;
    break;

default :
    cout << "Ban chon sai chuong trinh" << endl;
    break;

};

```

```

    }while (control);
}

```

```

int main() {
    menu();
    return 0;
}

```

2: Các Thuật Toán Sử Dụng Trong Code Cài Thuật Toán Và Chạy Demo Thuật Toán Trong Menu :

+ 1 : Cài thuật toán DFS , sử dụng Stack để khởi tạo

Thuật toán DFS(u):

Begin

Bước 1 (Khởi tạo):

```

stack =  $\emptyset$ ; //Khởi tạo stack là  $\emptyset$ 
Push(stack, u); //Đưa đỉnh u vào ngăn xếp
<Thăm đỉnh u>; //Duyệt đỉnh u
chuaxet[u] = False; //Xác nhận đỉnh u đã duyệt

```

Bước 2 (Lặp) :

```

while ( stack  $\neq \emptyset$  ) do
    s = Pop(stack); //Loại đỉnh ở đầu ngăn xếp
    for each t $\in$  Ke(s) do //Lấy mỗi đỉnh t $\in$ Ke(s)
        if ( chuaxet[t] ) then //Nếu t đúng là chưa duyệt
            <Thăm đỉnh t>; // Duyệt đỉnh t
            chuaxet[t] = False; // Xác nhận đỉnh t đã duyệt
            Push(stack, s); //Đưa s vào stack
            Push(stack, t); //Đưa t vào stack
            break; //Chỉ lấy một đỉnh t
        EndIf;
    EndFor;
EndWhile;

```

Bước 3 (Trả lại kết quả):

```

Return(<Tập đỉnh đã duyệt>);

```

End.


```
C:\Users\Admin\Downloads\BaiTapLopToanRoiRac11234567899991234567899.exe
Cai Dat Thuat Toan Trong Do Thi
Hay chon menu sau:
Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)
Nhap 3: Cai Dat Thuat Toan Dijkstra
Nhap 4: Cai Dat Thuat Toan Krusal
Nhap 0: De Thoat Khoi Ung Dung
-----
Ban Chon
1
Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
Chon dinh bat dau DFS: 2
DFS(2): 2 1 4 3 5 6 7
Cai Dat Thuat Toan Trong Do Thi
Hay chon menu sau:
Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)
Nhap 3: Cai Dat Thuat Toan Dijkstra
Nhap 4: Cai Dat Thuat Toan Krusal
Nhap 0: De Thoat Khoi Ung Dung
-----
Ban Chon
-
```

* Thuật Toán DFS chạy bắt đầu tại đỉnh s=2

+ 2: Cài thuật Toán BFS :

Thuật toán BFS(s):

Bước 1(Khởi tạo):

Queue = \emptyset ; Push(Queue,s); chuaxet[s] = False;

Bước 2 (Lặp):

while (Queue $\neq \emptyset$) do

u = Pop(Queue);

for each $v \in Ke(u)$ do

if (chuaxet[v]) then

Push(Queue, v);chuaxet[v]=False;truoc[v]=u;

EndIf;

EndFor ;

EndWhile ;

Bước 3 (Trả lại kết quả) :

Return(<Tập đỉnh được duyệt>) ;

End.

--

```
C:\Users\Admin\Downloads\BaiTapLopToanRoiRac11234567899991234567899.exe
Nhap 0: De Thoat Khoi Ung Dung
-----
Ban Chon
1
Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
Chon dinh bat dau DFS: 2
DFS(2): 2 1 4 3 5 6 7
Cai Dat Thuat Toan Trong Do Thi
Hay chon menu sau:
Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)
Nhap 3: Cai Dat Thuat Toan Dijkstra
Nhap 4: Cai Dat Thuat Toan Krusal
Nhap 0: De Thoat Khoi Ung Dung
-----
Ban Chon
2
Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)
Chon dinh bat dau BFS : 2
BFS(2): 2 1 3 6 4 5 7
Cai Dat Thuat Toan Trong Do Thi
Hay chon menu sau:
Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)
Nhap 3: Cai Dat Thuat Toan Dijkstra
Nhap 4: Cai Dat Thuat Toan Krusal
Nhap 0: De Thoat Khoi Ung Dung
-----
Ban Chon
```

* Thuật Toán BFS chạy bắt đầu tại đỉnh $u = 2$

- 3 : Cài thuật toán Dijkstra : Dùng DFS

Thuật toán DFS(s):

Begin

Bước 1 (Khởi tạo):

```
stack =  $\emptyset$ ; // Khởi tạo stack là  $\emptyset$   
Push(stack, s); // Đưa đỉnh s vào ngăn xếp  
chuaxet[s] = False; // Xác nhận đỉnh u đã duyệt
```

Bước 2 (Lặp) :

```
while ( stack  $\neq \emptyset$  ) do  
    u = Pop(stack); // Loại đỉnh ở đầu ngăn xếp  
    for each v  $\in$  Ke(u) do // Lấy mỗi đỉnh u  $\in$  Ke(v)  
        if ( chuaxet[v] ) then // Nếu v đúng là chưa duyệt  
            chuaxet[v] = False; // Xác nhận đỉnh v đã duyệt  
            Push(stack, u); // Đưa u vào stack  
            Push(stack, v); // Đưa v vào stack  
            truoc[v] = u; // Ghi nhận truoc[v] là u  
            break; // Chỉ lấy một đỉnh t
```

EndIf;

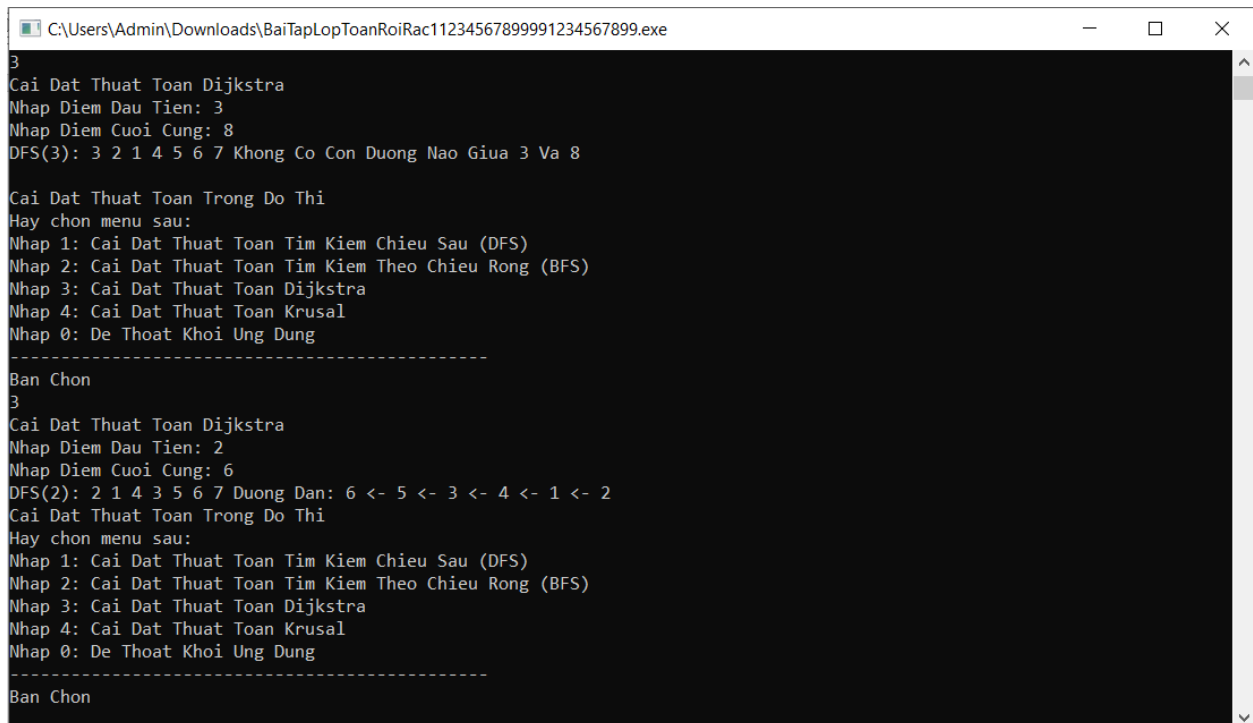
EndFor;

EndWhile;

Bước 3 (Trả lại kết quả):

```
Return(<Tập đỉnh đã duyệt>);
```

End.



```
C:\Users\Admin\Downloads\BaiTapLopToanRoiRac11234567899991234567899.exe  
3  
Cai Dat Thuat Toan Dijkstra  
Nhap Diem Dau Tien: 3  
Nhap Diem Cui Cung: 8  
DFS(3): 3 2 1 4 5 6 7 Khong Co Con Duong Nao Giua 3 Va 8  
  
Cai Dat Thuat Toan Trong Do Thi  
Hay chon menu sau:  
Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)  
Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)  
Nhap 3: Cai Dat Thuat Toan Dijkstra  
Nhap 4: Cai Dat Thuat Toan Krusal  
Nhap 0: De Thoat Khoi Ung Dung  
-----  
Ban Chon  
3  
Cai Dat Thuat Toan Dijkstra  
Nhap Diem Dau Tien: 2  
Nhap Diem Cui Cung: 6  
DFS(2): 2 1 4 3 5 6 7 Duong Dan: 6 <- 5 <- 3 <- 4 <- 1 <- 2  
Cai Dat Thuat Toan Trong Do Thi  
Hay chon menu sau:  
Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)  
Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)  
Nhap 3: Cai Dat Thuat Toan Dijkstra  
Nhap 4: Cai Dat Thuat Toan Krusal  
Nhap 0: De Thoat Khoi Ung Dung  
-----  
Ban Chon
```

* Thuật toán Dijkstra dùng DFS : Đi từ đỉnh 2 đến đỉnh 6

+ 4 : Cài Thuật Toán Kruskal:

Thuật toán sẽ xây dựng tập cạnh **T** của cây khung nhỏ nhất **H=<V, T>** theo từng bước như sau:

1. Sắp xếp các cạnh của đồ thị **G** theo thứ tự tăng dần của trọng số cạnh;
2. Xuất phát từ tập cạnh **T=φ**, ở mỗi bước, ta sẽ lần lượt duyệt trong danh sách các cạnh đã được sắp xếp, từ cạnh có trọng số nhỏ đến cạnh có trọng số lớn để tìm ra cạnh mà khi bổ sung nó vào **T** không tạo thành chu trình trong tập các cạnh đã được bổ sung vào **T** trước đó;
3. Thuật toán sẽ kết thúc khi ta thu được tập **T** gồm **n-1** cạnh.

Thuật toán được mô tả thông qua **thủ tục Kruskal** như sau:

```
Void Kruskal () {  
    T= φ;  
    While ( | T | < ( n-1) and ( E ≠ φ )) {  
        Chọn cạnh e ∈ E là cạnh có độ dài nhỏ nhất ;  
        E: = E \ { e };  
        If ( T ∪ { e } : Không tạo nên chu trình )  
            T= T ∪ { e };  
    }  
  
    IF ( | T | < n - 1 )  
        Đồ thị không liên thông ;  
}
```

```
C:\Users\Admin\Downloads\ BaiTapLopToanRoiRac11234567899991234567899.exe
Nhap 0: De Thoat Khoi Ung Dung
-----
Ban Chon
3
Cai Dat Thuat Toan Dijkstra
Nhap Diem Dau Tien: 2
Nhap Diem Cuoi Cung: 6
DFS(2): 2 1 4 3 5 6 7 Duong Dan: 6 <- 5 <- 3 <- 4 <- 1 <- 2
Cai Dat Thuat Toan Trong Do Thi
Hay chon menu sau:
Nhap 1: Cai Dat Thuat Toan Tim Kiem Chieu Sau (DFS)
Nhap 2: Cai Dat Thuat Toan Tim Kiem Theo Chieu Rong (BFS)
Nhap 3: Cai Dat Thuat Toan Dijkstra
Nhap 4: Cai Dat Thuat Toan Krusal
Nhap 0: De Thoat Khoi Ung Dung
-----
Ban Chon
4
Cai Dat Thuat Toan Krusal
So dinh do thi: 7
So canh do thi:12
Do dai cay khung nho nhat:11
Cac canh cua cay khung nho nhat:
2 6
4 5
5 6
1 4
3 4
3 7
```

* Thuật Toán Kruskal

Tổng Kết :

- Bài làm của nhóm em vẫn còn sai sót một số chỗ và đã khắc phục tối ưu nhất , mong thầy xem xét và đưa ra lời khuyên để tối ưu hơn nữa ạ .
- Các tài liệu tham khảo:
 - + <https://howkteam.vn/course/cau-truc-du-lieu-va-giai-thuat/bfs-va-dfs-4320>
 - + <https://vi.gadget-info.com/difference-between-bfs>