TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

======***======

BÁO CÁO BÀI TẬP LỚN THUỘC HỌC PHẦN
THỰC TẬP CƠ SỞ NGÀNH

NGHIÊN CỨU CƠ SỞ LÝ THUYẾT,ỨNG DỤNG VÀ CÀI ĐẶT THUẬT TOÁN ĐỂ GIẢI BÀI TOÁN CÂY KHUNG NHỎ NHẤT (MINIMALSPANNING TREE PROBLEM)

GVHD:	TS. Nguyễn Xuân Hoàng
Nhóm - Lớp:	3_20242IT6055002
Thành viên:	Nguyễn Minh Trí – 2023604797
	Vũ Huy Đạt - 2023604757
	Nguyễn Trúc Linh - 2023604989
	Nghiêm Thị Ngọc Anh - 2023604770
	Nguyễn Thị Bích Ngọc - 2023605756

Hà Nội,năm 2025

Mục Lục

MỞ ĐẦU

Trong những năm gần đây, sự phát triển nhanh chóng của khoa học máy tính đã tác động mạnh mẽ đến mọi lĩnh vực trong đời sống xã hội, từ giáo dục đến sản xuất. Công nghệ hiện đại ngày càng xâm nhập vào từng khía cạnh của cuộc sống, giúp cải thiện chất lượng nghiên cứu, tối ưu hóa quy trình sản xuất và quản lý. Các thiết bị và công nghệ tiên tiến như hệ thống tự động hóa, trí tuệ nhân tạo, cùng các phần mềm quản lý, đã trở thành công cụ quan trọng, hỗ trợ con người nâng cao hiệu quả công việc và giảm bớt sức lao động thủ công.

Sự phát triển mạnh mẽ của ngành công nghệ thông tin đã dẫn đến sự ra đời của nhiều công ty công nghệ, kèm theo hàng loạt ứng dụng đáp ứng nhu cầu ngày càng cao của xã hội. Công nghệ thông tin không chỉ mang lại những tiến bộ trong sản xuất mà còn thay đổi cách thức quản lý, giao tiếp và học tập qua các hệ thống tự động, phần mềm, và thiết bị thông minh.

Để các hệ thống phần mềm hoạt động hiệu quả trong thực tế, thuật toán là yếu tố quyết định. Thuật toán là nền tảng cốt lõi của bất kỳ phần mềm hay ứng dụng nào, giúp chúng vận hành một cách chính xác và ổn định. Trong bài viết này, chúng ta sẽ cùng khám phá một bài toán quan trọng với nhiều ứng dụng thực tiễn: Bài toán cây khung nhỏ nhất (Minimum Spanning Tree - MST).

Nhóm sinh viên thực hiện!

I. CƠ SỞ LÝ THUYẾT

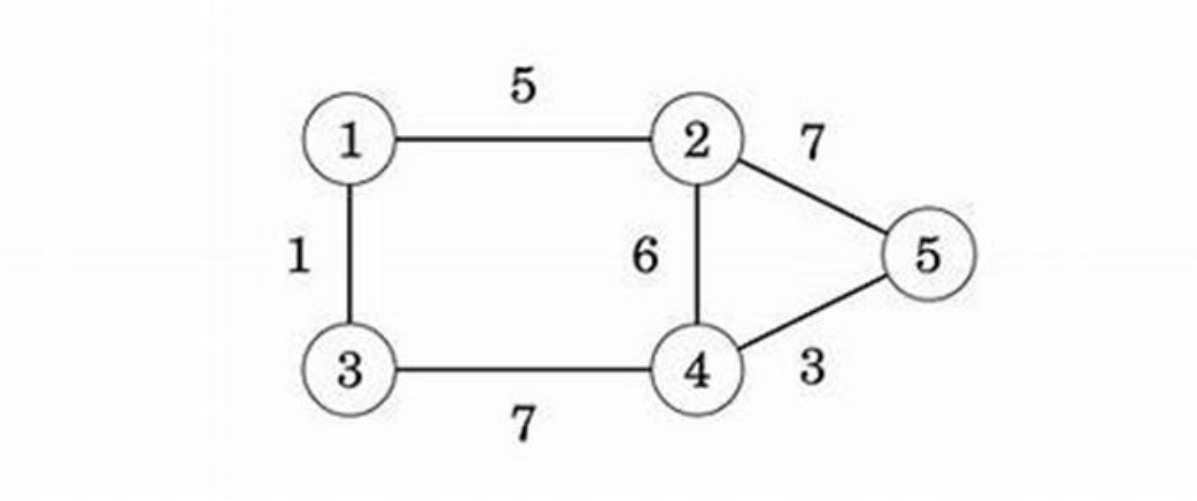
1.1. Cơ sở lý thuyết của bài toán cây khung nhỏ nhất

1.1.1. Bài toán cây khung nhỏ nhất

❖ Định nghĩa bài toán cây khung nhỏ nhất

Bài toán tìm cây khung nhỏ nhất là một trong số những bài toán tối ưu, cơ bản và quan trọng trong lý thuyết đồ thị. Bài toán có nhiều ứng dụng trong thực tế nhằm tìm ra phương án tối ưu nhất để giải quyết vấn đề. Bài toán được phát biểu như sau:

Cho G = (V,E) là đồ thị vô hướng liên thông, với tập đỉnh V và tập cạnh E. Mỗi cạnh của đồ thị được gán với một số không âm biểu thị độ dài của cạnh. Bài toán được ta đặt ra ở đây là trong tất cả các cây khung của đồ thị G hãy tìm ra cây khung với tổng độ dài các cạnh là 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 cây khung nhỏ nhất.



Hình 1.1 Minh họa đồ thị vô hướng có trọng số

Ví dụ: Trong hình 1.1 ta có đồ thị vô hướng có trọng số G với tập đỉnh V = {A, B, C, D, E} và tập cạnh E = {AB, AE, BC, BD, BE, CD, DE}. Và cây khung nhỏ nhất của đồ thị trên là A-B-C-D-E với tổng trọng số là 25.

Bài toán cây khung nhỏ nhất có nhiều ứng dụng trong đời sống, giúp tìm được cách giải quyết, làm việc tối ưu nhất. Ta xem qua hai bài toán ví dụ dưới đây:

- Bài toán kết nối mạng máy tính: Cần nối mạng một hệ thống gồm n máy tính. Biết chi phí nối giữa hai máy là m (chi phí này thay đổi phụ thuộc vào độ dài cáp nối cần sử dụng giữa hai máy). Giải quyết bài toán ta có thể tìm được cách nối mạng sao cho tổng chi phí là nhỏ nhất.

- Bài toán giao thông vận tải: Cần xây dựng hệ thống đường sắt đi qua tất cả n thành phố. Biết chi phí nguyên vật liệu là m. Qua bài toán ta có thể 5 tìm được cách sao cho hệ thống đường sắt này có thể đi qua tất cả các thành phố với tổng quãng đường là ngắn nhất và chi phí nguyên vật liệu để thi công là nhỏ nhất.

1.1.2. Các khái niệm cơ bản

1.1.2.1 Đồ thị

Đồ thị là một cấu trúc dữ liệu dùng để mô hình hóa các mối quan hệ giữa các đối tượng. Ta có thể hình dung đơn giản đồ thị như một bản đồ, nơi các thành phố (đỉnh) được kết nối với nhau bởi các con đường (cạnh). Đồ thị được phát biểu như sau:

Cho G = (V,E) trong đó đồ thị G gồm có, V là tập các đỉnh của đồ thị, E tập các cặp không thứ tự chứa các đỉnh phân biệt, được gọi là cạnh. Hai đỉnh thuộc một cạnh được gọi là các đỉnh đầu cuối của cạnh đó.

Tùy vào đặc điểm và ứng dụng mà đồ thị được chia ra làm nhiều loại. Ta sẽ xét một số loại đồ thị tiêu biểu dưới đây:

- Đồ thị vô hướng: Là đồ thị mà đường đi từ đỉnh này tới đỉnh còn lại tương tự với đường đi chiều ngược lại.

- Đồ thị có hướng: Là đồ thị mà đường đi từ đỉnh này tới đỉnh còn lại sẽ hoàn toàn khác biệt với đường đi chiều ngược lại.

- Đơn đồ thị: Là đồ thị không có khuyên và cạnh song song.

- Đa đồ thị: Là đồ thị không thỏa mãn đơn đồ thị.

- Đồ thị không trọng số: Là đồ thị mà các cạnh không có đơn vị biểu diễn độ dài.

- Đồ thị có trọng số: Là đồ thị mà các cạnh có đơn vị biểu diễn độ dài.

❖ Biểu diễn đồ thị

Để lưu trữ đồ thị và thực hiện các thuật toán khác nhau với đồ thị trên máy tính cần phải tìm những cấu trúc dữ liệu thích hợp để mô tả đồ thị. Việc chọn cấu trúc dữ liệu nào để biểu diễn đồ thị có tác động rất lớn đến hiệu quả của thuật toán. Vì vậy, việc chọn lựa cấu trúc dữ liệu để biểu diễn đồ thị phụ thuộc vào từng tình huống cụ thể (bài toán và thuật toán cụ thể). Chúng ta sẽ xét một số phương pháp cơ bản được sử dụng để biểu diễn đồ thị trên máy tính dưới đây.

● Ma trận kề, ma trận trọng số

Xét đơn đồ thị vô hướng G=(V,E). Ký hiệu tập đỉnh V={1,2,.., n} và tập cạnh E={e1 ,e2 ,.. em}. Ta gọi ma trận kề của đồ thị G là ma trận.

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

Trong đó các phần tử ai,j được xác định như sau

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

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

Trong trường hợp đồ thị có trọng số thay vì ma trận kề, để biểu diễn đồ thị ta sử dụng ma trận trọng số.

$$C = \{c[i, j] \mid i, j = 1, 2,..., n\}$$

Trong đó:

$$c[i, j] = c(i, j) \text{ nếu } (i, j) \in E$$

$$c[i, j] = \infty \text{ nếu } (i, j) \notin E$$

● Danh sách cạnh (cung)

Trong trường hợp đồ thị thưa ta thường dùng cách biểu diễn đồ thị dưới dạng danh sách cạnh. Trong cách biểu diễn đồ thị bởi danh sách cạnh chúng ta sẽ lưu trữ danh sách tất cả các cạnh của đồ thị vô hướng (có hướng). Một cạnh e=(x, y) của đồ thị sẽ tương ứng với hai biến Dau[e], Cuoi[e]. như vậy, để lưu trữ đồ thị ta cần sử dụng 2*m đơn vị bộ nhớ. Nhược điểm của cách biểu diễn này là để xác định những đỉnh nào của đồ thị là kề với một đỉnh cho trước chúng ta phải làm cỡ m phép so sánh (khi duyệt qua danh sách tất cả các cạnh của đồ thị).

*Chú ý: Trong trường hợp đồ thị có trọng số ta cần thêm m đơn vị bộ nhớ để lưu trữ trọng số của các cạnh

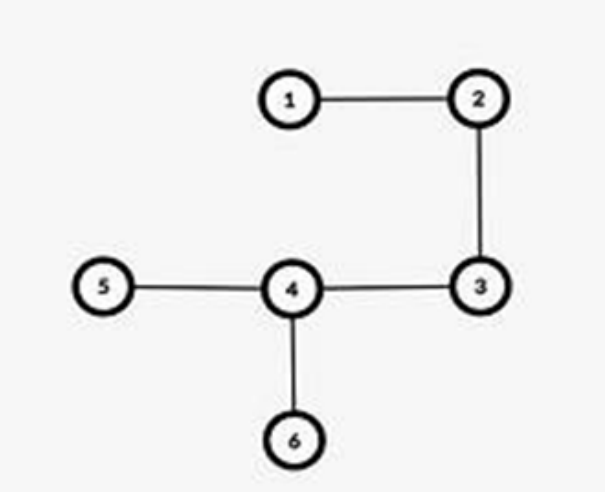
- Danh sách đỉnh kề

Trong cách biểu diễn này, với mỗi đỉnh v của đồ thị chúng ta lưu trữ danh sách các đỉnh kề với nó, mà ta sẽ ký hiệu là:

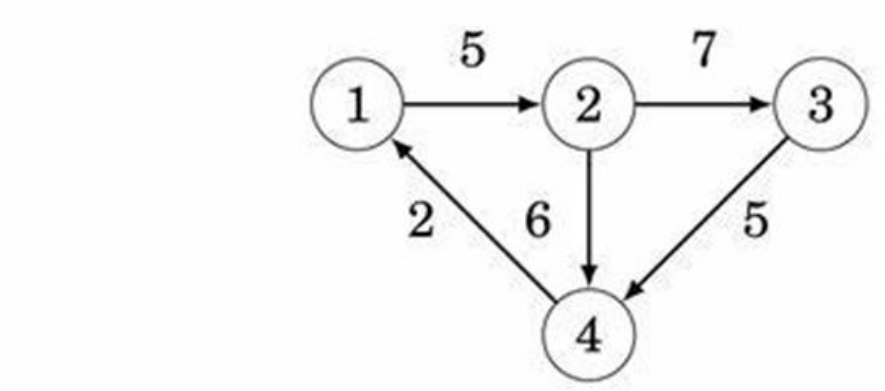
$$Ke(v) = \{u \in V \mid (v, u) \in E\}$$

❖ Đồ thị vô hướng Định nghĩa 1.1 (Đơn đồ thị vô hướng): Đơn đồ thị vô hướng G = (V,E) bao gồm V là tập các đỉnh, E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh.

Ví dụ mạng lưới giao thông thể hiện giữa hai địa điểm khác nhau có đường đi trực tiếp đến nhau hay không là một đơn đồ thị vô hướng.



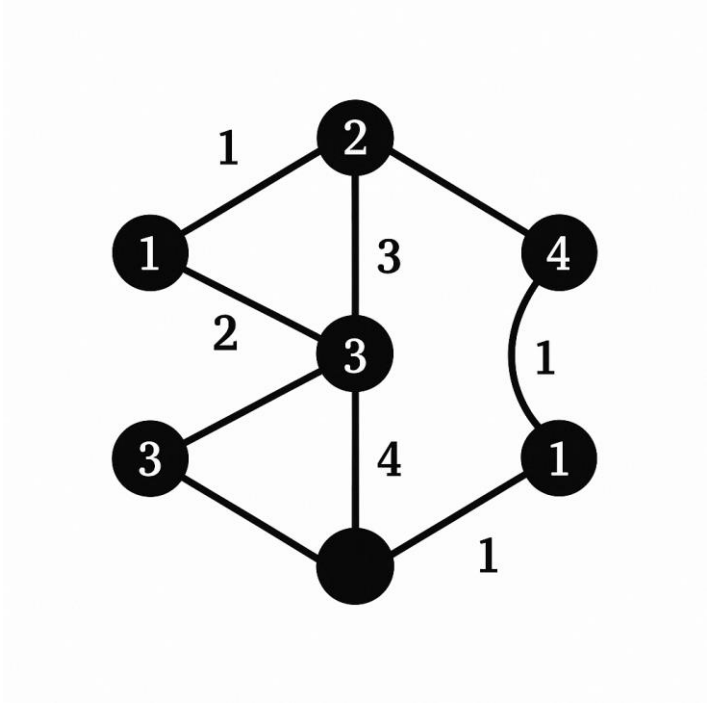
Hình 1.2 Đơn đồ thị vô hướng không trọng số



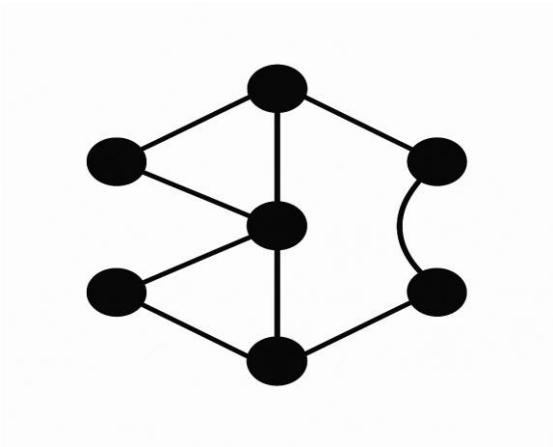
Hình 1.3 Đơn đồ thị vô hướng có trọng số

Định nghĩa 1.2 (Đa đồ thị vô hướng): Đa đồ thị vô hướng G = (V,E) bao gồm V là tập các đỉnh, E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Hai cạch E1 và E2 được gọi là cạnh lặp nếu chúng cùng tương ứng với một cặp đỉnh.

Ví dụ mạng lưới giao thông thể hiện giữa hai địa điểm khác nhau có bao nhiêu đường đi trực tiếp đến nhau là một đa đồ thị vô hướng.



Hình 1.4 Đa đồ thị vô hướng không trọng số

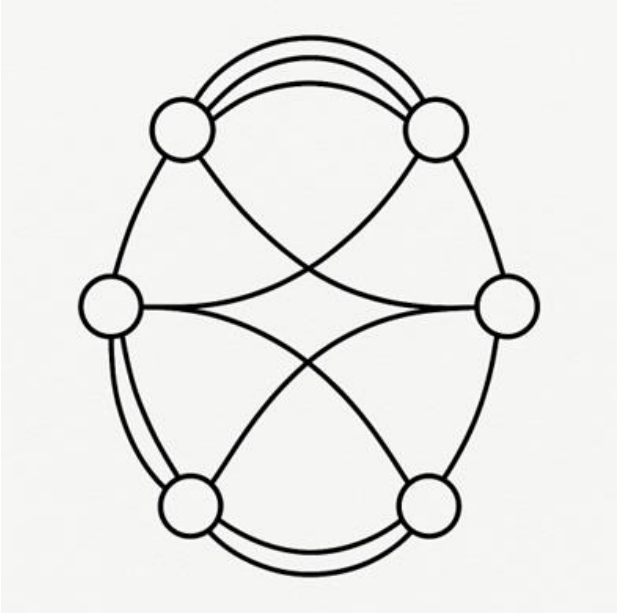


Hình 1.5 Đa đồ thị vô hướng có trọng số

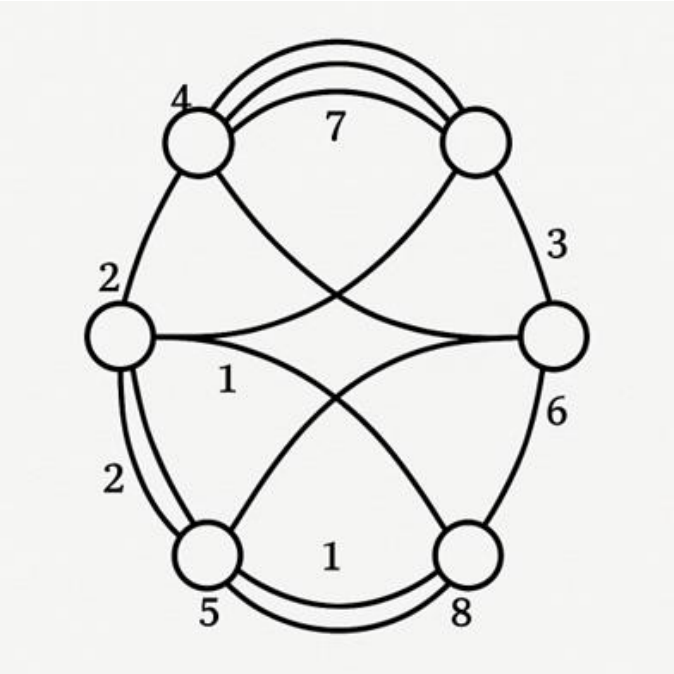
Định nghĩa 1.3 (Giả đồ thị vô hướng): Giả đồ thị vô hướng G = (V, E) bao gồm V là tập các đỉnh và E là tập các cặp không có thứ tự gồm hai phần tử

(không nhất thiết phải khác nhau) của V gọi là cạnh. Một cạnh mà 2 đỉnh của nó trùng nhau gọi là khuyên.

Ví dụ mạng lưới giao thông thể hiện giữa 2 địa điểm bất kỳ (có thể 2 địa điểm đó trùng nhau) có đường đi trực tiếp đến nhau hay không, là một Giả đồ thị vô hướng.



Hình 1.6 Giả đồ thị vô hướng không trọng số



Hình 1.7 Giả đồ thị vô hướng có trọng số

Một số khái niệm trong đồ thị vô hướng:

- Một cạnh được gọi là khuyên nếu có dạng (v,v)

- Cạnh song song là hai cạnh khác nhau nhưng có cùng đỉnh đầu đỉnh cuối.

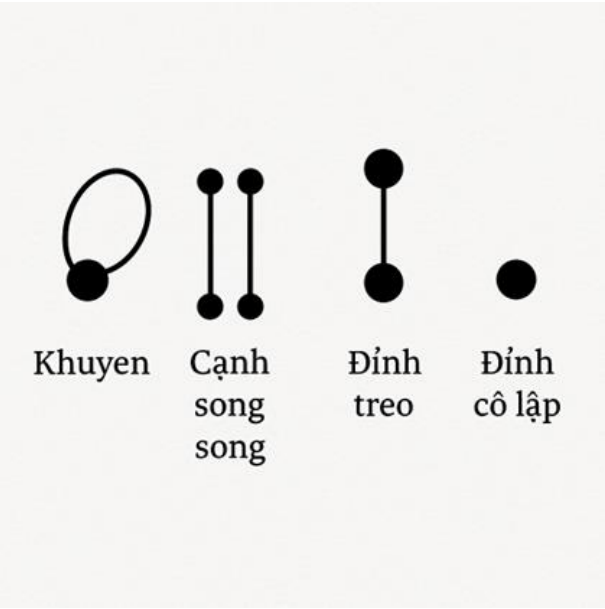
- Bậc của đỉnh là số cạnh kề với đỉnh đó. Ký hiệu là deg(v) là bậc của đỉnh v.

- Đỉnh treo là đỉnh có bậc 1

- Đỉnh cô lập là đỉnh có bậc 0

- Hai đỉnh của 1 cạnh gọi là hai đỉnh kề nhau.

- Hai cạnh chung một đỉnh gọi là hai cạnh kề nhau.

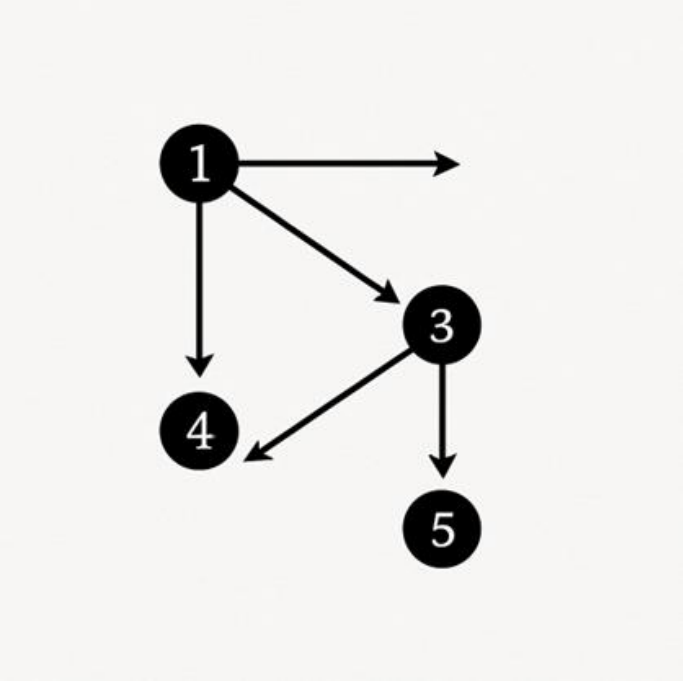


Hình 1.8 Khuyên, cạnh song song, đỉnh treo, đỉnh cô lập

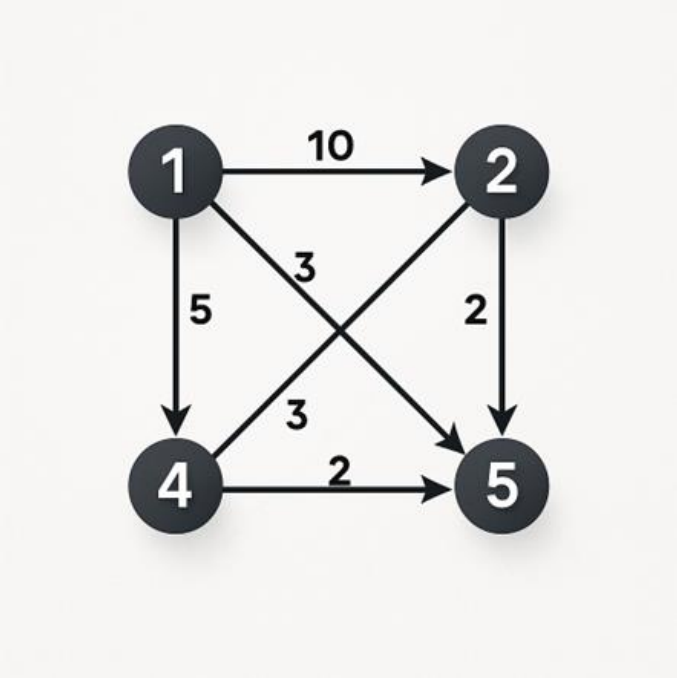
❖ Đồ thị có hướng

Hoàn toàn tương tự với đồ thị vô hướng, có các loại đồ thị có hướng: Đơn đồ thị có hướng, đa đồ thị có hướng, giả đồ thị có hướng.

Định nghĩa 1.4 (Đơn đồ thị có hướng): Đơn đồ thị có hướng G = (V,E) bao gồm V là tập các đỉnh, và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Một cạnh e=(u,v) thì u gọi là đỉnh đầu, v gọi là đỉnh cuối, đồng thời e gọi là cạnh ra của đỉnh u và gọi là cành vào của đỉnh v. Hai cạnh e1 và e2 được gọi là cạnh lặp nếu chúng có cùng đỉnh đầu và đỉnh cuối. Đơn đồ thị có hướng khi tập E không chứa cạnh lặp.

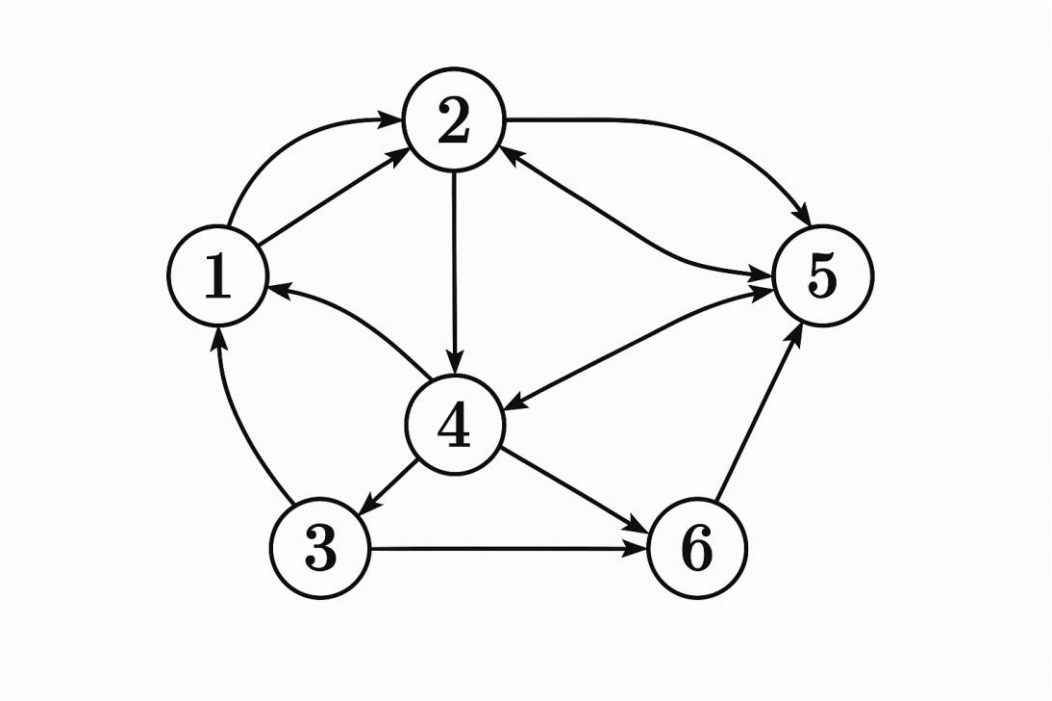


Hình 1.9 Đơn đồ thị có hướng không trọng số

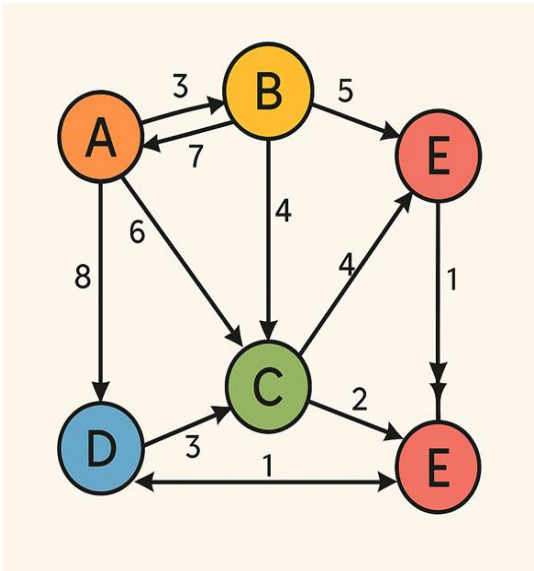


Hình 1.10 Đơn đồ thị có hướng có trọng số

Định nghĩa 1.5 (Đa đồ thị có hướng): Đa đồ thị có hướng G= (V, E) bao gồm V là tập các đỉnh, và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Đồ thị có hướng là Đa đồ thị có hướng khi tập E có chứa cạnh lặp.



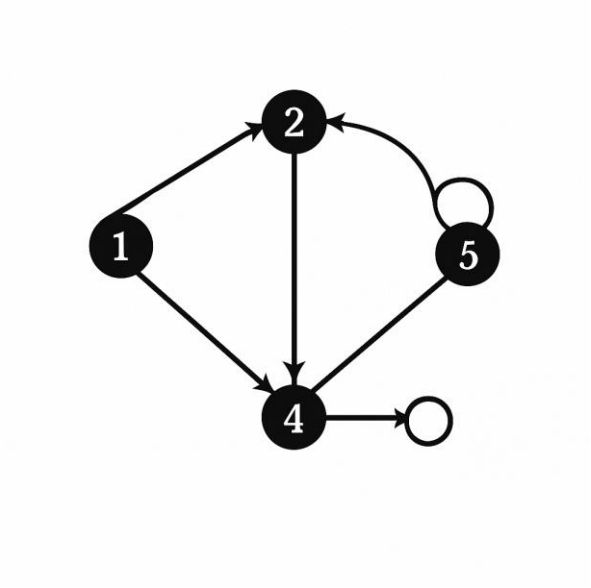
Hình 1.11 Đa đồ thị có hướng không trọng số



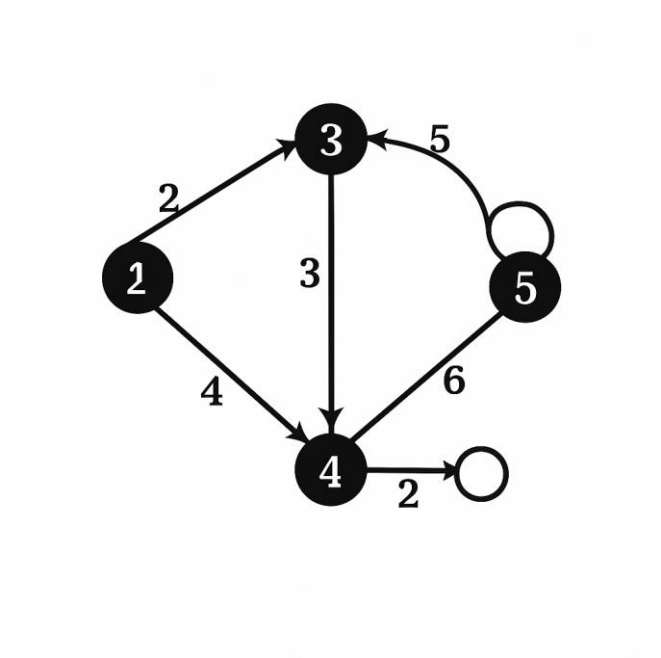
Hình 1.12 Đa đồ thị có hướng có trọng số

Định nghĩa 1.6 (Giả đồ thị có hướng): Giả đồ thị có hướng G = (V, E) bao gồm V là tập các đỉnh và E là tập các cặp có thứ tự gồm hai phần tử (không nhất thiết phải khác nhau) của V gọi là cạnh. Một cạnh mà đỉnh đầu và đỉnh

cuối của nó trùng nhau gọi là khuyên. Đồ thị có hướng là Giả đồ thị có hướng khi tập E chứa khuyên.



Hình 1.13 Giả đồ thị có hướng không trọng số



Hình 1.14 Giả đồ thị có hướng có trọng số

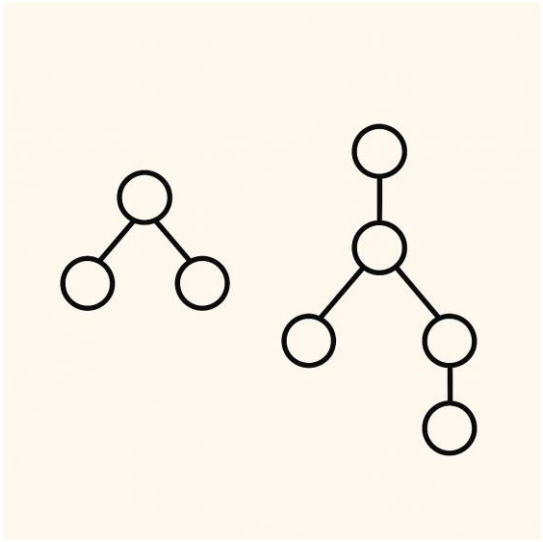
Một số khái niệm trong đồ thị có hướng:

- Các khái niệm tương tự như trong đồ thị vô hướng: Khuyên, cạnh song song, đỉnh cô lập, đỉnh kề.

- Bậc vào của một đỉnh là số cung đi vào đỉnh đó, bậc ra là số cung đi ra khỏi đỉnh đó.

1.1.2.2 Cây, cây khung nhỏ nhất

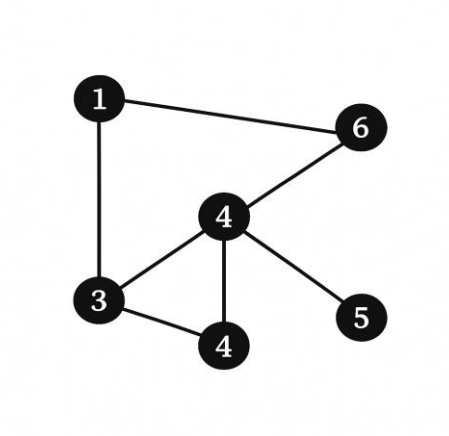
Cây là một đồ thị vô hướng liên thông không có chu trình. Đồ thị không liên thông và không có chu trình được gọi là rừng. Như vậy rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.



Hình 1.15 Rừng gồm 2 cây

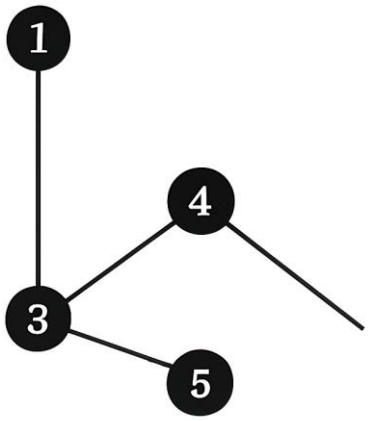
Cây khung của đồ thị bao gồm tất cả các đỉnh của đồ thị và một số cạnh của đồ thị sao cho có một cạnh giữa hai đỉnh bất kỳ. Giống như cây tổng quát cây khung là liên thông và không có chu trình. Cây khung có tổng trọng số các cạnh nhỏ nhất là cây khung nhỏ nhất.

Ví dụ: Xét đồ thị sau:



Hình 1.16 Đồ thị minh họa

Cây khung nhỏ nhất:



Hình 1.17 Cây khung nhỏ nhất

Tính chất của cây khung nhỏ nhất::

- Tính tối ưu: Trong tất cả các cây khung của một đồ thị, cây khung nhỏ nhất luôn có tổng trọng số của các cạnh là nhỏ nhất.

- Tính duy nhất: Nếu tất cả các cạnh trong đồ thị đều có trọng số khác nhau, thì cây khung nhỏ nhất sẽ là duy nhất.

1.1.2.3 Mối quan hệ giữa cây và đồ thị

Trong bài toán tìm cây khung nhỏ nhất, chúng ta thường bắt đầu với một đồ thị và mục tiêu là tìm một cây đặc biệt từ đồ thị đó. Để hiểu rõ hơn về mối quan hệ giữa cây và đồ thị trong bài toán này, hãy cùng phân tích:

Đồ thị: Điểm bắt đầu

- Đồ thị: đại diện cho toàn bộ hệ thống mà chúng ta đang nghiên cứu. Mỗi đỉnh trong đồ thị có thể là một thành phố, một máy tính, một gen, v.v. Các cạnh kết nối các đỉnh thể hiện mối quan hệ hoặc liên kết giữa chúng.

- Trọng số: Mỗi cạnh trong đồ thị thường được gán một trọng số, đại diện cho chi phí, khoảng cách hoặc một thuộc tính nào đó liên quan đến cạnh đó. Trọng số này rất quan trọng trong việc tìm cây khung nhỏ nhất.

Cây: Mục tiêu cần tìm

- Cây là một đồ thị con đặc biệt của đồ thị ban đầu.

- Cây khung: Là một cây con bao gồm tất cả các đỉnh của đồ thị ban đầu.

- Cây khung nhỏ nhất: Trong số tất cả các cây khung của đồ thị, cây khung có tổng trọng số các cạnh nhỏ nhất được gọi là cây khung nhỏ nhất.

Quan hệ giữa đồ thị và cây

- Cây là một phần của đồ thị: Cây khung là một tập hợp con của các cạnh trong đồ thị ban đầu.

- Cây đơn giản hóa đồ thị: Cây khung loại bỏ đi các cạnh dư thừa, chỉ giữ lại các cạnh cần thiết để kết nối tất cả các đỉnh.

- Cây khung nhỏ nhất tối ưu hóa: Cây khung nhỏ nhất tìm ra cách kết nối tất cả các đỉnh với chi phí thấp nhất (tổng trọng số nhỏ nhất). Như vậy nói một cách đơn giản, ta có thể coi đồ thị đại diện cho toàn bộ vấn đề mà bài toán đặt ra, cây là một giải pháp đơn giản hóa và tối ưu hóa cho vấn đề, và cây khung nhỏ nhất là giải pháp tốt nhất, với chi phí thấp nhất.

1.2. Cơ sở lý thuyết của thuật toán

1.2.1. Thuật toán Prim

1.2.1.1 Ý tưởng của thuật toán

· Ý tưởng:

1. Bắt đầu từ một đỉnh tùy ý: Chọn một đỉnh bất kỳ trong đồ thị làm điểm bắt đầu.

2. Xây dựng cây khung từng bước:

i. Mỗi bước, từ các đỉnh đã có trong cây khung, tìm cạnh có trọng số nhỏ nhất mà kết nối với một đỉnh mới chưa thuộc cây khung.

ii. Thêm cạnh đó vào cây khung.

3. Lặp lại: Tiếp tục thêm các đỉnh vào cây khung bằng cách chọn cạnh có trọng số nhỏ nhất nối giữa các đỉnh đã thuộc cây khung và đỉnh chưa thuộc cây.

4. Dừng lại: Khi tất cả các đỉnh trong đồ thị đã được bao gồm trong cây khung, thuật toán dừng lại.

1.2.1.2 Mô tả chi tiết thuật toán

· Các bước thực hiện:

Cho đồ thị G = . Tìm cây khung T nhỏ nhất.

• Bước 1: Chọn một cạnh e = (a, b) có trọng số nhỏ nhất.

• Bước 2: V1 = {a, b}, E1 = {e}, V = V - {a, b}. Cây T = .

• Bước 3: Nếu |E1| = n - 1 thì dừng. Cây Khung nhỏ nhất là T.

• Bước 4: Tìm một cạnh e = (a, b) thoả mãn: a ∈ V1 và b ∈ V sao cho e có trọng số nhỏ nhất.

• Bước 5: E1 = E1 ∪ {e} và V1 = V1 ∪ {b} và V = V - {b}.

• Quay lại bước 3

Hình 1.18. Lưu đồ thuật toán Prim

1.2.1.3 Độ phức tạp của thuật toán

Độ phức tạp về thời gian

Độ phức tạp thời gian của thuật toán Prim phụ thuộc vào cấu trúc dữ liệu được sử dụng để lưu trữ đồ thị:

1. Sử dụng danh sách kề với hàng đợi ưu tiên nhị phân (binary heap):

○ Khi sử dụng danh sách kề và hàng đợi ưu tiên nhị phân, thuật toán Prim có độ phức tạp thời gian là O(ElogV), với:

+V là số lượng đỉnh của đồ thị.

+E là số lượng cạnh của đồ thị.

○ Mỗi cạnh sẽ được duyệt một lần, và thời gian cho mỗi lần duyệt là O(logV) khi sử dụng hàng đợi ưu tiên nhị phân để tìm phần tử có độ ưu tiên cao nhất.

2. Sử dụng danh sách kề với Fibonacci heap:

○ Khi sử dụng Fibonacci heap, độ phức tạp thời gian giảm xuống còn O(E+VlogV).

○ Điều này giúp thuật toán nhanh hơn trong trường hợp đồ thị có số đỉnh rất lớn so với số cạnh.

Độ phức tạp về không gian

Độ phức tạp về không gian của thuật toán Prim là O(V+E), vì cần lưu trữ:

1. Danh sách kề hoặc ma trận kề của đồ thị để đại diện các cạnh giữa các đỉnh.

2. Một mảng hoặc hàng đợi ưu tiên để lưu trữ các cạnh đang được xét.

3. Một tập hợp để lưu các đỉnh đã được chọn vào cây khung nhỏ nhất.

1.2.1.4 Ưu điểm, nhược điểm của thuật toán.

· Ưu điểm của thuật toán Prim:

+Thuật toán Prim hoạt động tốt trên đồ thị có số cạnh lớn (dense graph), vì nó dựa vào việc duyệt qua các đỉnh thay vì các cạnh.

+Với cấu trúc dữ liệu phù hợp như hàng đợi ưu tiên (priority queue), độ phức tạp thời gian có thể giảm xuống O(E+VlogV), trong đó E là số cạnh và V là số đỉnh.

+Dễ triển khai với cấu trúc đơn giản và dễ hiểu, phù hợp cho việc triển khai với các cấu trúc dữ liệu như ma trận kề hoặc danh sách kề.

+Khi chọn đỉnh mới để mở rộng cây khung, Prim chỉ cần cập nhật danh sách cạnh liên quan, giúp tiết kiệm tài nguyên so với việc xét toàn bộ đồ thị.

· Nhược điểm của thuật toán Prim:

+Trên đồ thị có số cạnh ít , thuật toán có thể lãng phí tài nguyên, đặc biệt khi sử dụng ma trận kề .

+Để đạt hiệu quả cao, thuật toán yêu cầu sử dụng cấu trúc dữ liệu phức tạp như hàng đợi ưu tiên hoặc cây nhị phân heap. Điều này có thể khó khăn nếu không có thư viện hỗ trợ.

+Prim là thuật toán tuần tự, vì vậy khó áp dụng vào các hệ thống tính toán song song hoặc phân tán.

1.2.2. Thuật toán Kruskal

1.2.2.1 Ý tưởng của thuật toán

· Ý tưởng:

Thuật toán Kruskal sử dụng thuật toán tham lam (greedy algorithm). Ý tưởng chính là liên tục chọn cạnh có trọng số nhỏ nhất trong số các cạnh chưa được chọn, miễn là cạnh đó không tạo thành chu trình trong cây khung đang xây dựng.

· Hướng tiếp cận:

Thuật toán bắt đầu với một rừng các cây, mỗi cây chỉ chứa một đỉnh. Sau đó, nó lần lượt xét các cạnh theo thứ tự tăng dần của trọng số. Nếu hai đỉnh của một cạnh thuộc hai cây khác nhau trong rừng, thì cạnh đó được thêm vào cây khung và hai cây được hợp nhất lại. Quá trình này tiếp tục cho đến khi tất cả các đỉnh đều thuộc cùng một cây.

1.2.2.2 Mô tả chi tiết thuật toán

· Các bước thực hiện:

1.Khởi tạo dữ liệu và các biến:

- Nhập đồ thị G với n đỉnh và m cạnh.

- Sắp xếp các cạnh của đồ thị theo thứ tự trọng số tăng dần.

-Tạo n tập rời nhau, mỗi tập chứa 1 đỉnh.

- Khởi tạo cây khung rỗng để lưu trữ các cạnh của cây khung nhỏ nhất.

2.Vòng lặp:

- Lặp qua các cạnh theo thứ tự trọng số tang dần.

- Với mỗi cạnh e, kiểm tra xem hai đầu e có tạo thành chi trình hay không.

- Nếu không tạo chu trình :

+Thêm cạnh e vào MST

+Hợp nhất tập hợp chứa hai đầu của e.

3.Kết thúc:

- Nếu cây khung đã đủ n-1 cạnh (với n là số đỉnh), dừng và trả về cây khung nhỏ nhất.

Các trường hợp đặc biệt hoặc lỗi

· Đồ thị không liên thông: Nếu đồ thị không liên thông, thuật toán sẽ không thể tạo ra cây khung nhỏ nhất bao gồm tất cả các đỉnh. Trong trường hợp này, cần kiểm tra và thông báo lỗi hoặc xử lý riêng các thành phần liên thông.

· Cạnh có trọng số âm: Thuật toán Kruskal vẫn hoạt động bình thường với các cạnh có trọng số âm, nhưng cần đảm bảo rằng các trọng số được sắp xếp đúng thứ tự.

· Đồ thị có nhiều cạnh có trọng số bằng nhau: Khi có nhiều cạnh có trọng số bằng nhau, thuật toán vẫn hoạt động đúng, nhưng có thể có nhiều cây khung nhỏ nhất khác nhau

1.19. Lưu đồ thuật toán Kruskal

1.2.2.3 Độ phức tạp của thuật toán

· Độ phức tạp thời gian :

- Sắp xếp cạnh: Bước đầu tiên là sắp xếp các cạnh theo trọng số tăng dần. Độ phức tạp của bước này phụ thuộc vào thuật toán sắp xếp sử dụng. Thường ta sử dụng thuật toán sắp xếp nhanh (quicksort) hoặc sắp xếp hợp nhất (merge sort) với độ phức tạp trung bình là O(m log m), trong đó m là số cạnh.

- Duyệt các cạnh và thực hiện union-find: Trong vòng lặp, ta duyệt qua tất cả các cạnh và thực hiện các phép toán find_set và union. Độ phức tạp của phần này là O(mα(m, n)).

+ mα(m, n) là hàm Ackermann nghịch đảo, tăng rất chậm và gần như có thể coi là hằng số trong thực tế.

- Độ phức tạp thời gian của thuật toán Kruskal là O(m log m + mα(m, n)), trong đó m là số cạnh, n là số đỉnh. Tuy nhiên, do hàm Ackermann nghịch đảo tăng trưởng rất chậm nên ta thường coi độ phức tạp của thuật toán là O(m log m).

· Độ phức tạp không gian :

- Mảng để lưu các cạnh: Cần một mảng để lưu trữ các cạnh của đồ thị, có độ phức tạp là O(m).

- Cấu trúc dữ liệu disjoint-set: Để quản lý các cây trong rừng, ta cần sử dụng cấu trúc dữ liệu disjoint-set. Độ phức tạp không gian của disjoint-set là O(n).

- Độ phức tạp không gian của thuật toán Kruskal là O(m + n).

1.2.2.4 Ưu điểm, nhược điểm của thuật toán.

· Ưu điểm của thuật toán Kruskal:

- Thuật toán Kruskal hoạt động hiệu quả trên đồ thị thưa, nơi số lượng cạnh ít hơn nhiều so với số đỉnh

- Kruskal dễ dàng triển khai nhờ việc sử dụng cấu trúc dữ liệu Union-Find để kiểm tra chu trình và hợp nhất các tập hợp đỉnh. Các thao tác này có thể được thực hiện với độ phức tạp gần như hằng số, giúp cải thiện đáng kể hiệu suất của thuật toán.

- Khác với các thuật toán khác như Prim, Kruskal không yêu cầu phải lựa chọn một đỉnh bắt đầu cụ thể, mà chỉ cần duyệt qua các cạnh đã sắp xếp. Điều này giúp thuật toán đơn giản hơn và tránh được các vấn đề liên quan đến lựa chọn đỉnh ban đầu.

· Nhược điểm của thuật toán Kruskal:

- Cần sắp xếp cạnh bước sắp xếp các cạnh theo trọng số có thể tiêu tốn nhiều thời gian, đặc biệt với các đồ thị lớn.

- Việc sử dụng và quản lý cấu trúc dữ liệu disjoint-set có thể phức tạp hơn so với các cấu trúc dữ liệu khác.

1.2.3. Cơ sở lý thuyết của giải thuật di truyền

1.2.3.1 Ý tưởng của thuật toán

Giải thuật di truyền (GAs) mô phỏng quá trình tiến hóa tự nhiên dựa trên lý thuyết chọn lọc tự nhiên của Darwin. Trong tự nhiên, các cá thể mạnh mẽ hơn, thích nghi tốt hơn có khả năng sống sót cao hơn và truyền lại gen tốt của mình cho thế hệ sau. GAs vận dụng quy trình này để giải quyết các bài toán tối ưu hóa và tìm kiếm trong không gian lớn.

*Ý tưởng

- Bài toán được biểu diễn dưới dạng một cá thể (individual), thường mã hóa dưới dạng chuỗi (mã nhị phân, số thực, hoặc các cấu trúc phức tạp khác).

- Một tập hợp các cá thể (population) cùng tồn tại và được đánh giá dựa trên hàm thích nghi (fitness function).

- Các cá thể "tốt" sẽ được chọn lọc để tham gia vào quá trình tái tổ hợp, lai ghép và đột biến, tạo ra thế hệ mới với chất lượng cao hơn.

Ví dụ ý tưởng:

Nếu bạn cần tối ưu hóa việc bố trí một nhà kho (tối thiểu hóa thời gian vận chuyển), giải thuật di truyền có thể thử nhiều phương án khác nhau và cải thiện dần thông qua việc kết hợp các phương án tốt nhất.

1.2.3.2 Mô tả chi tiết thuật toán

Các bước thực hiện thuật toán GA:

- Khởi tạo quần thể:

- Tạo một quần thể ban đầu gồm N cá thể (population size).

- Mỗi cá thể được biểu diễn dưới dạng mã hóa nhị phân (binary encoding) hoặc biểu diễn khác như số thực, cây, hoặc ma trận.

- Đánh giá hàm thích nghi:

- Tính toán giá trị fitness của mỗi cá thể dựa trên hàm mục tiêu (fitness function). Hàm fitness đánh giá mức độ "tốt" của mỗi lời giải.

● Chọn lọc:

- Chọn các cá thể tốt nhất dựa trên giá trị fitness để đưa vào quá trình lai ghép và đột biến.

- Phương pháp chọn lọc phổ biến:

+ Chọn lọc tỷ lệ bánh xe quay (Roulette Wheel Selection): Xác suất chọn cá thể tỷ lệ thuận với fitness của nó..

+ Chọn lọc giải đấu (Tournament Selection): Chọn ngẫu nhiên k cá thể và giữ lại cá thể tốt nhất..

● Lai ghép :

- Lai ghép hai cá thể cha mẹ (parents) để tạo ra cá thể con (offspring).

Các phương pháp lai ghép phổ biến:

+ Lai ghép một điểm (one-point crossover): Chọn một điểm cắt trong chuỗi gene, hoán đổi phần sau điểm cắt.

+ Lai ghép hai điểm (two-point crossover): Chọn hai điểm cắt, hoán đổi đoạn giữa hai điểm.. 23 + Lai ghép đồng nhất (uniform crossover): Hoán đổi từng gen dựa trên xác suất..

● Đột biến:

- Ngẫu nhiên thay đổi một số gen trong cá thể để tăng tính đa dạng di truyền.

- Tỷ lệ đột biến thường nhỏ (mutation rate)

Ví dụ: Đột biến gen từ 0 thành 1

● Lặp lại:

- Lặp lại các bước trên qua nhiều thế hệ, quần thể sẽ dần hội tụ đến giải pháp tốt nhất.

● Điều kiện dừng:

- Thuật toán dừng khi đạt một trong các điều kiện:

+ Đã thực hiện đủ số thế hệ.

+ Đạt được giá trị fitness mong muốn.

+ Không còn cải thiện sau một số vòng lặp.

Lưu đồ thuật toán:

1.2.3.3 Độ phức tạp của thuật toán

- Độ phức tạp thời gian:

Với N là số cá thể trong quần thể,G là số thế hệ, và E là thời gian đánh giá fitness cho mỗi cá thể:

$$T=O(N \cdot G \cdot E)$$

Trong đó, thời gian đánh giá fitness(E) phụ thuộc vào bài toán cụ thể.

- Độ phức tạp không gian:

Để lưu trữ thông tin quần thể, ta cần:

N: Số lượng cá thể.

M: Kích thước bộ gen của mỗi cá thể (số gene/biến số).

Không gian lưu trữ cho quần thể là:

$S=O(N \cdot M)$

1.2.3.4 Ưu và nhược điểm của thuật toán.

- Ưu điểm của thuật toán GA:

+ Đặc biệt phù hợp cho các vấn đề khó khăn, nơi ít được biết về cơ bản không gian tìm kiếm + Được sử dụng rộng rãi trong kinh doanh, khoa học và kỹ thuật

+ Có thể được sử dụng để giải quyết nhiều vấn đề khác nhau không dễ giải quyết bằng cách sử dụng thuật toán khác.

+ GA có khả năng tìm ra nhiều lời giải tối ưu cục bộ và đôi khi cả tối ưu toàn cục trong không gian giải pháp phức tạp.

+ Do các cá thể trong quần thể được đánh giá độc lập, GA dễ dàng được triển khai song song để tăng tốc độ tính toán.

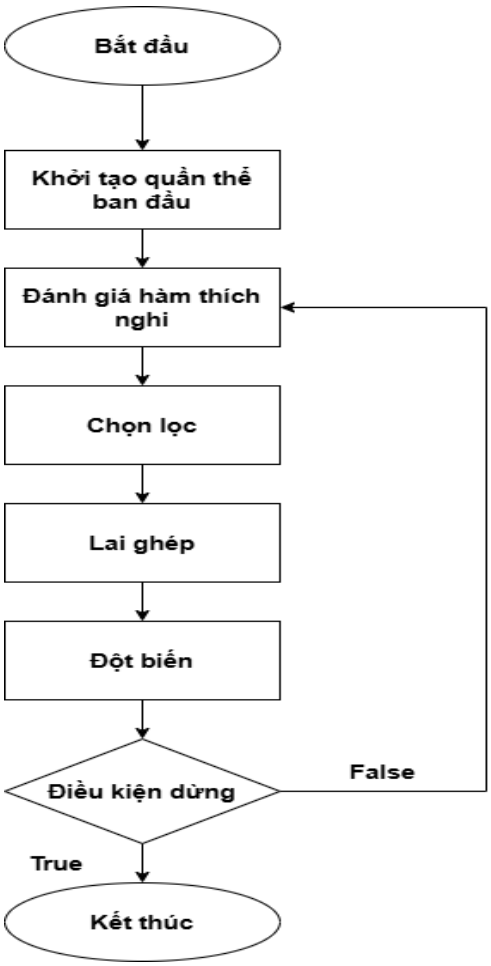
- Nhược điểm của thuật toán GA:

+ GA thường yêu cầu nhiều lần tính toán hàm thích nghi qua nhiều thế hệ, đặc biệt với các bài toán lớn hoặc phức tạp. Điều này dẫn đến chi phí tính toán cao.

+ GA có thể mất nhiều thế hệ để đạt được giải pháp tốt, đặc biệt khi quần thể khởi đầu không đủ đa dạng.

+ Chất lượng và hiệu quả của GA phụ thuộc nhiều vào việc thiết kế hàm thích nghi. Hàm không phù hợp có thể dẫn đến kết quả không tối ưu hoặc không thực tế.

+ Đối với các bài toán có cấu trúc rõ ràng hoặc có thuật toán chuyên dụng, GA thường chậm và kém hiệu quả hơn.



1.20. Lưu đồ thuật toán di truyền

II. THIẾT KẾ THUẬT TOÁN

2.1 Thiết kế thuật toán Prim

2.1.1 Thuật toán Prim trong bài toán cây khung nhỏ nhất

- Mục tiêu: Với một đồ thị vô hướng có trọng số, hãy tìm cây khung nhỏ nhất, tức là cây con có tất cả các đỉnh và có tổng trọng số các cạnh là nhỏ nhất.

- Phương pháp tham lam: Tại mỗi bước, chọn đỉnh gần nhất để thêm vào cây khung hiện tại, giúp mở rộng cây mà không tạo chu trình.

2.1.2 Các bước giải thuật

2.1.2.1 Bài toán và phân tích yêu cầu

- Input:

+ Số đỉnh V và số cạnh E của đồ thị.

+ Ma trận kề hoặc danh sách các cạnh của đồ thị với trọng số.

- Output:

+ Một cây khung nhỏ nhất với tổng trọng số các cạnh là nhỏ nhất.

+ Tổng trọng số của cây khung này.

- Ràng buộc: Cây khung không được tạo ra chu trình và chỉ chứa V−1 cạnh để kết nối tất cả các đỉnh của đồ thị.

2.1.2.2 Giải thuật

Bước 1: Khởi tạo bài toán

- Đầu vào: Ma trận kề hoặc danh sách các cạnh với trọng số của đồ thị.

- Biến khởi tạo:

+ Một vector dinh gồm các pair bên trong để lưu tập các đỉnh kề của một đỉnh và trọng số của đỉnh đó với đỉnh kề.
+ Mảng used có kiểu dữ liệu là bool để gán cho một đỉnh là thuộc tập đỉnh V ban đầu hay thuộc tập đỉnh V(MST).
+ Một vector MST để lưu các cạnh của cây khung cực tiểu.

+ Một struct canh để lưu các cạnh.
+ Biến d để ghi nhận tổng trọng số của cây khung.

Bước 2: Xác định lựa chọn cục bộ

- Lựa chọn cục bộ tối ưu: Tại mỗi bước, chọn đỉnh có trọng số nhỏ nhất mà chưa nằm trong cây khung nhỏ nhất.

- Điều kiện tham lam: Chọn đỉnh có cạnh nối đến cây khung hiện tại với trọng số nhỏ nhất.

Bước 3: Cập nhật trạng thái bài toán

- Sau khi thêm một đỉnh vào cây khung, nếu cạnh nối của đỉnh đó với một đỉnh trong cây khung có trọng số là nhỏ nhất thì gán cho biến min_w bằng giá trị nhỏ nhất đó.

Bước 4: Lặp lại quá trình

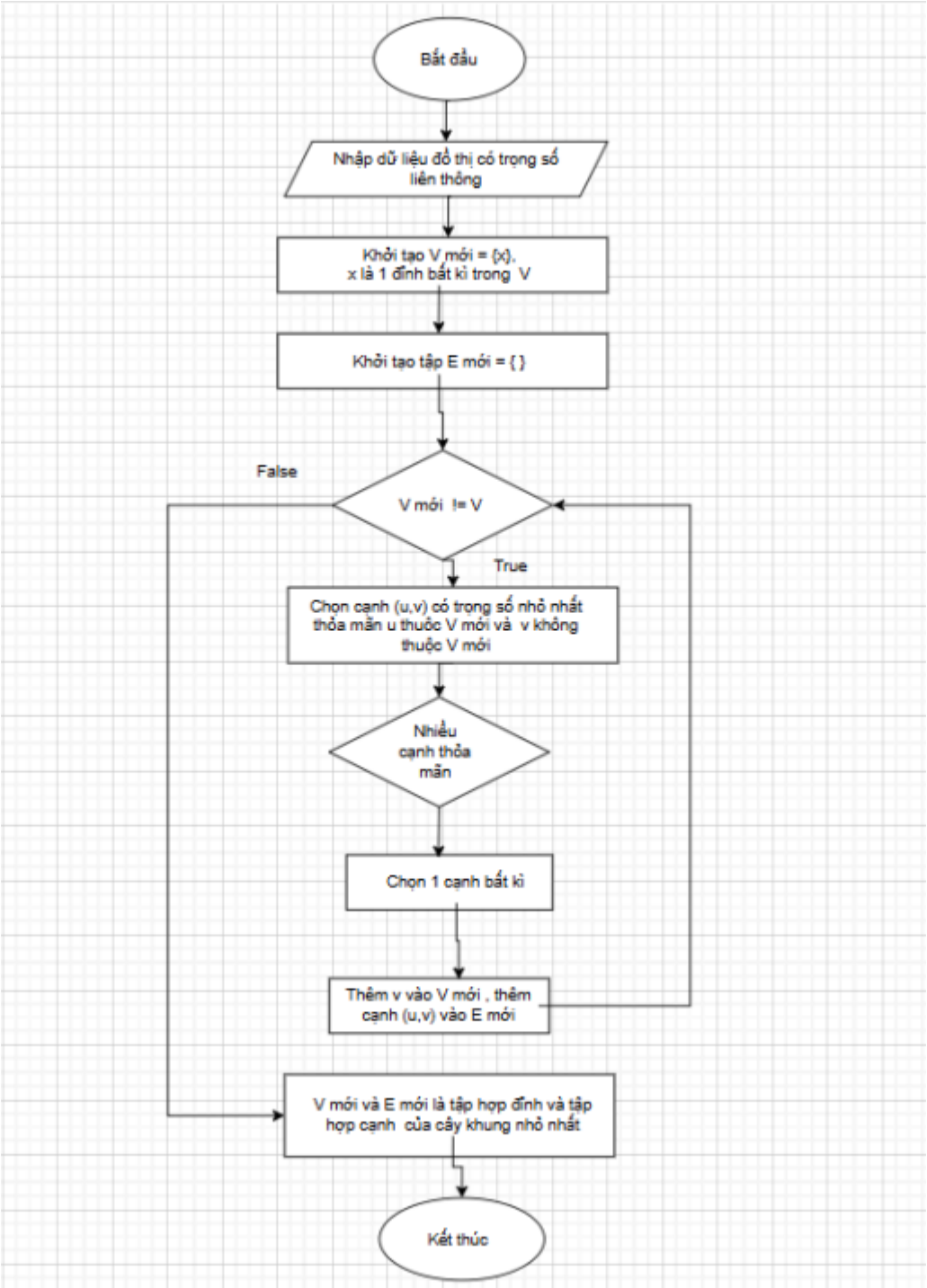
- Tiếp tục chọn đỉnh cục bộ tối ưu cho đến khi cây khung chứa đủ $V-1$ cạnh.

Bước 5: Kết thúc và xuất kết quả

- Sau khi cây khung có đủ $V-1$ cạnh, kết thúc quá trình lặp.

- Kết quả: Xuất ra các cạnh trong cây khung nhỏ nhất và tổng trọng số của nó.

2.1.3 Lưu đồ thuật toán



2.2. Thiết kế thuật toán Kruskal

2.2.1 Thuật toán Kruskal trong bài toán cây khung nhỏ nhất.

- Mục tiêu: Với một đồ thị vô hướng có trọng số, hãy tìm cây khung nhỏ nhất, tức là cây con có tất cả các đỉnh và có tổng trọng số các cạnh là nhỏ nhất.

- Phương pháp tham lam đưa ra lựa chọn tốt nhất ở mỗi bước (chọn cạnh có trọng số nhỏ nhất), mà không xét đến tác động lâu dài của những lựa chọn đó. Trong thuật toán Kruskal, điều này có nghĩa là tại mỗi bước, chọn cạnh có trọng số nhỏ nhất mà không tạo ra chu trình trong cây khung hiện tại.

2.2.2 Các bước giải thuật

2.2.2.1 Bài toán và phân tích yêu cầu

- Input:

+ Số đỉnh V và số cạnh E của đồ thị.

+ Một danh sách các cạnh với trọng số, mỗi cạnh được biểu diễn dưới dạng (u,v,w) trong đó:

- u và v là hai đỉnh của cạnh,

- w là trọng số của cạnh.

- Output:

+ Một cây khung nhỏ nhất với tổng trọng số của các cạnh là nhỏ nhất.

+ Tổng trọng số của cây khung này.

Hạn chế: Cây khung không được tạo ra chu trình, và chỉ chứa V-1 cạnh để kết nối tất cả các đỉnh của đồ thị.

2.2.2.2 Giải thuật

Bước 1: Khởi tạo bài toán

- Đầu vào: Một tập hợp các cạnh với trọng số của đồ thị.

- Biến khởi tạo:

+ Sắp xếp các cạnh theo thứ tự trọng số tăng dần để chuẩn bị cho việc lựa chọn cạnh nhỏ nhất ở mỗi bước.

+ Sử dụng cấu trúc Disjoint Set (tập hợp rời rạc) để quản lý các tập

hợp đỉnh, giúp kiểm tra chu trình.

+ Một mảng MST để lưu trữ các cạnh của cây khung nhỏ nhất. + Một biến total_weight để ghi nhận tổng trọng số của cây khung.

Bước 2: Xác định lựa chọn cục bộ

- Lựa chọn cục bộ tối ưu: Tại mỗi bước, chọn cạnh có trọng số nhỏ nhất

mà hai đỉnh của nó nằm trong các tập hợp rời rạc khác nhau (tức là việc

thêm cạnh này không tạo chu trình).

- Điều kiện tham lam: Chỉ cần kiểm tra trọng số của các cạnh và tính hợp

lệ của các cạnh với điều kiện không tạo chu trình.

Bước 3: Cập nhật trạng thái bài toán

- Sau khi chọn được cạnh phù hợp, thực hiện:

+ Thêm cạnh vào mảng MST.

+ Gộp các tập hợp của hai đỉnh trong Disjoint Set để cập nhật trạng

thái kết nối giữa chúng.

+ Cập nhật total_weight với trọng số của cạnh vừa được chọn.

Bước 4: Lặp lại quá trình

- Tiếp tục chọn cạnh cục bộ tối ưu cho đến khi cây khung chứa đủ V-1

cạnh (tức là kết nối tất cả các đỉnh mà không tạo chu trình).

Bước 5: Kết thúc và xuất kết quả

- Sau khi cây khung có đủ V-1 cạnh, kết thúc quá trình lặp.

- Kết quả: Xuất ra các cạnh trong cây khung nhỏ nhất và tổng trọng số

của nó.

2.2.3 Lưu đồ thuật toán

2.3. Thiết kế giải thuật di truyền

2.3.1. Tiếp cận giải thuật di truyền trong bài toán cây khung nhỏ nhất

- Mục tiêu: Tìm cây khung nhỏ nhất của một đồ thị vô hướng có trọng số bằng cách

sử dụng giải thuật di truyền.

2.3.2 Các bước giải thuật

2.3.2.1 Bài toán và phân tích yêu cầu

Input:

● Số đỉnh V và số cạnh E của đồ thị.

● Một danh sách các cạnh với trọng số, mỗi cạnh có dạng (u,v,w) trong đó:

○ u và v là hai đỉnh của cạnh.

○ w là trọng số của cạnh.

Output:

● Một cây khung nhỏ nhất (tập hợp V-1 cạnh) sao cho tổng trọng số của các cạnh là nhỏ nhất.

Hạn chế:

● Cây khung không được tạo ra chu trình.

● Phải chứa đủ V-1 cạnh để kết nối tất cả các đỉnh.

2.3.2.2 Giải thuật

Bước 1: Khởi tạo bài toán

Đầu vào:

● Một tập hợp các cạnh của đồ thị vô hướng, mỗi cạnh được biểu diễn dưới dạng (u,v,w) trong đó:

○ u và v là hai đỉnh của cạnh.

○ w là trọng số của cạnh.

● Số đỉnh V và số cạnh E của đồ thị.

Biến khởi tạo:

● Quần thể: Một tập hợp các cá thể, mỗi cá thể là một cây khung khả thi. Cá thể được biểu diễn dưới dạng danh sách V-1 cạnh.

● Hàm fitness: Được sử dụng để đánh giá chất lượng của các cá thể trong quần thể. Giá trị fitness được tính dựa trên tổng trọng số của cây khung nhỏ nhất.

● Tham số di truyền:

○ Kích thước quần thể Psize

○ Số thế hệ tối đa Gmax

○ Xác suất lai ghép Cr

○ Xác suất đột biến Mr

Bước 2: Xác định lựa chọn cục bộ

Lựa chọn cục bộ tối ưu:

● Mỗi cá thể được đánh giá bằng hàm fitness:

$$\text{fitness}(T) = -\sum_{(u,v,w)\in} w$$

(Dùng dấu âm vì đây là bài toán tối thiểu hóa tổng trọng số.)

Điều kiện:

● Chỉ giữ lại các cá thể thỏa mãn:

○ Là cây khung hợp lệ (liên thông và không chứa chu trình).

○ Gồm đủ V-1 cạnh.

Bước 3: Cập nhật trạng thái bài toán

Quá trình tiến hóa bao gồm:

1. Chọn lọc (Selection):

○ Chọn các cá thể tốt nhất dựa trên fitness bằng Roulette Wheel Selection hoặc Tournament Selection.

2. Lai ghép (Crossover):

○ Chọn ngẫu nhiên hai cá thể bố mẹ.

○ Lai ghép bằng cách trộn các cạnh của hai cá thể bố mẹ, sau đó sử dụng thuật toán sửa chữa (repair) để đảm bảo tính hợp lệ của cây khung.

3. Đột biến (Mutation):

○ Chọn ngẫu nhiên một cá thể.

○ Thay thế một cạnh trong cá thể bằng một cạnh khác từ tập cạnh EEE, sau đó sửa chữa nếu cần.

4. Sửa chữa (Repair):

○ Đảm bảo cá thể sau quá trình lai ghép hoặc đột biến vẫn là cây khung hợp lệ:

■ Không chứa chu trình.

■ Liên thông.

Cập nhật: Quần thể mới được tạo ra sau các bước trên sẽ thay thế quần thể cũ.

Bước 4: Lặp lại quá trình

● Tiếp tục thực hiện chọn lọc, lai ghép, đột biến và sửa chữa qua các thế hệ.

● Dừng lại khi đạt đến số thế hệ tối đa Gmax hoặc khi không còn cải thiện đáng kể trong fitness sau Gtable thế hệ.

Bước 5: Kết thúc và xuất kết quả

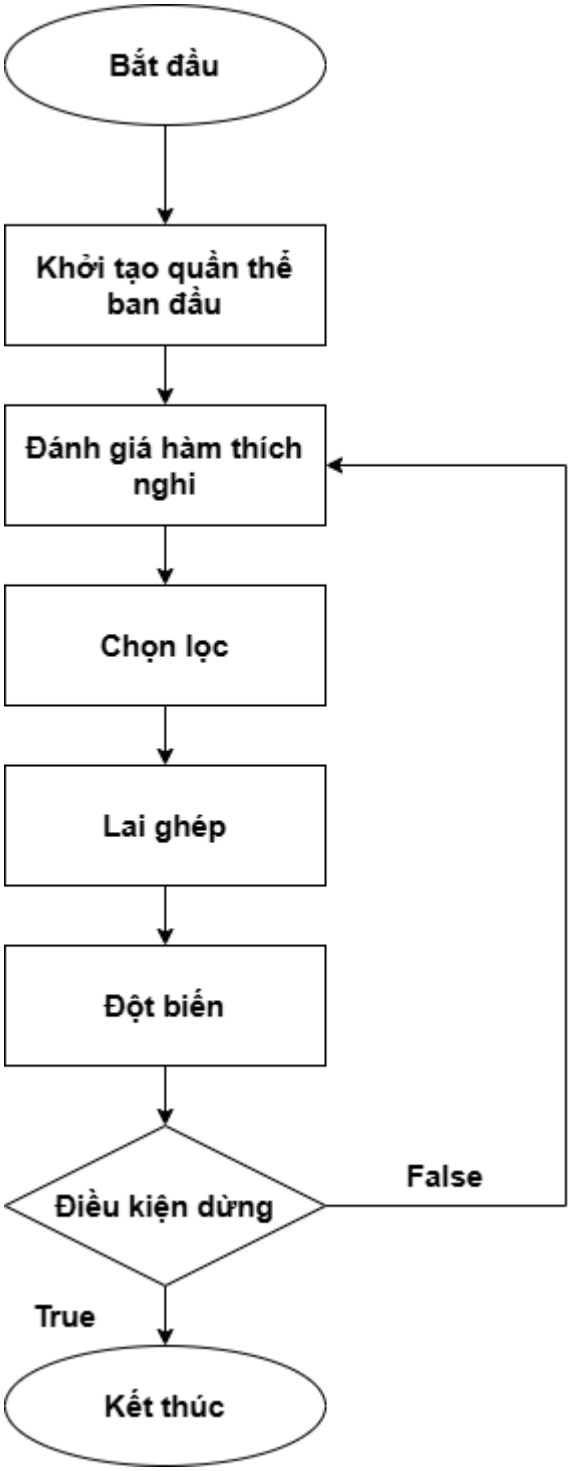
● Sau khi thuật toán kết thúc, cá thể tốt nhất (tức cây khung nhỏ nhất) trong quần thể sẽ được trả về.

● Kết quả bao gồm:

○ Tổng trọng số nhỏ nhất của cây khung.

○ Danh sách các cạnh trong cây khung nhỏ nhất.

2.3.3 Lưu đồ thuật toán



III. CÀI ĐẶT VÀ KIỂM THỬ

3.1. Cài đặt và kiểm thử thuật toán Prim.

3.1.1. Mục tiêu.

Xây dựng và kiểm thử thuật toán Prim để tìm cây khung nhỏ nhất (Minimum Spanning Tree - MST) của một đồ thị vô hướng, liên thông, có trọng số.

3.1.2. Giới thiệu thuật toán Prim.

Thuật toán Prim là một giải thuật tham lam (Greedy) dùng để tìm cây khung nhỏ nhất trong đồ thị.
Nó luôn chọn cạnh có trọng số nhỏ nhất nối từ tập đỉnh đã chọn đến một đỉnh chưa chọn.

Ý tưởng chính:

- Bắt đầu từ một đỉnh bất kỳ.
- Tại mỗi bước, chọn cạnh nhỏ nhất nối từ đỉnh đã chọn đến đỉnh chưa chọn.
- Lặp lại cho đến khi bao phủ tất cả đỉnh.

3.1.3. Cài đặt thuật toán prim.

Ngôn ngữ sử dụng C++

```cpp
#include<bits/stdc++.h>
using namespace std;

// V(MST) la tap cac dinh trong cay khung, V la tap cac dinh ban dau
const int maxn = 1001; // so dinh toi da cua do thi
int n; // so dinh cua do thi
int m; // so canh cua do thi
vector<pair <int, int> > dinh[maxn]; // tap cac dinh ke cua 1 dinh va trong so cua dinh do voi dinh ke
bool used[maxn]; // used[i] = true : i thuoc tap V(MST), used[i] = false : i thuoc tap V

// luu lai canh cua cay khung
struct canh{
    int x; // dinh dau cua 1 canh
    int y; // dinh cuoi cua 1 canh
    int w; // trong so cua 1 canh
};

// Nhap cac canh cua do thi
void nhap(){
    cout <<"Nhap so dinh :";
    cin >> n;
    cout <<"Nhap so canh :";
```

```
    cin >> m;
    for(int i = 0; i < m; i++){
        int x, y, w;
        cout <<"Nhap dinh dau cua canh :";
        cin >> x;
        cout <<"Nhap dinh cuoi cua canh :";
        cin >> y;
        cout <<"Nhap trong so cua canh :";
        cin >> w;
        dinh[x].push_back({y, w}); // y la dinh ke cua x voi trong so la w
        dinh[y].push_back({x, w}); // x la dinh ke cua y voi trong so la w
    }
    memset(used, false, sizeof(used)); // khoi tao cho tat ca cac dinh ban dau thuoc tap V
}

void prim(int u){
    vector<canh> MST; // cay khung MST
    int d = 0; // do dai ban dau cua cay khung
    used[u] = true; // dua dinh u vao tap V(MST)
    while(MST.size() < n - 1){
        // tim canh (x, y) ngan nhat co x thuoc V va y thuoc V(MST)
        int min_w = INT_MAX; // khoi tao do dai nho nhat cua cay khung
```

```
        int X, Y; // luu 2 dinh cua canh ngan nhat
        // duyet qua tat ca cac dinh cua tap V ban dau
        for(int i = 1; i <= n; i++){
            // neu dinh i thuoc tap V(MST)
            if(used[i] == true){
                // duyet danh sach ke cua dinh i
                for(pair<int, int> it : dinh[i]){
                    int j = it.first, trongso = it.second;
                    // neu dinh ke voi i thuoc tap V ban dau va trong so nho hon trong so min thi luu 2 dinh
                    if(used[j] == false && trongso < min_w){
                        min_w = trongso;
                        X = i; Y = j;
                    }
                }
            }
        }
        MST.push_back({X, Y, min_w});// them canh ngan nhat cua tim duoc vao cay khung MST
        d += min_w; // do dai cay khung khi them 1 canh moi
        used[Y] = true; // cho dinh X vao V(MST), loai x ra khoi V
    }
    cout <<"Do dai cuc tieu cua cay khung la :"<< d << endl;
    for(int i = 0; i < MST.size(); i++){
            cout << MST[i].x << " " << MST[i].y << " " << MST[i].w << endl;
        }
}
```

```
int main(){
    nhap();
    int k;
    do{
        cout <<"Nhap dinh xuat phat :";
        cin >> k;
    }while(k <= 0 || k > n);
    prim(k);
}
```

3.1.4. Kiểm thử thuật toán Prim.

+Trường hợp 1: Đồ thị nhỏ 3 đỉnh, liên thông.

```
Nhap so dinh :3
Nhap so canh :3
Nhap dinh dau cua canh :1
Nhap dinh cuoi cua canh :2
Nhap trong so cua canh :5
Nhap dinh dau cua canh :2
Nhap dinh cuoi cua canh :3
Nhap trong so cua canh :6
Nhap dinh dau cua canh :3
Nhap dinh cuoi cua canh :1
Nhap trong so cua canh :8
Nhap dinh xuat phat :1
Do dai cuc tieu cua cay khung la :11
1 2 5
2 3 6

Process returned 0 (0x0)   execution time : 49.252 s
Press any key to continue.
```

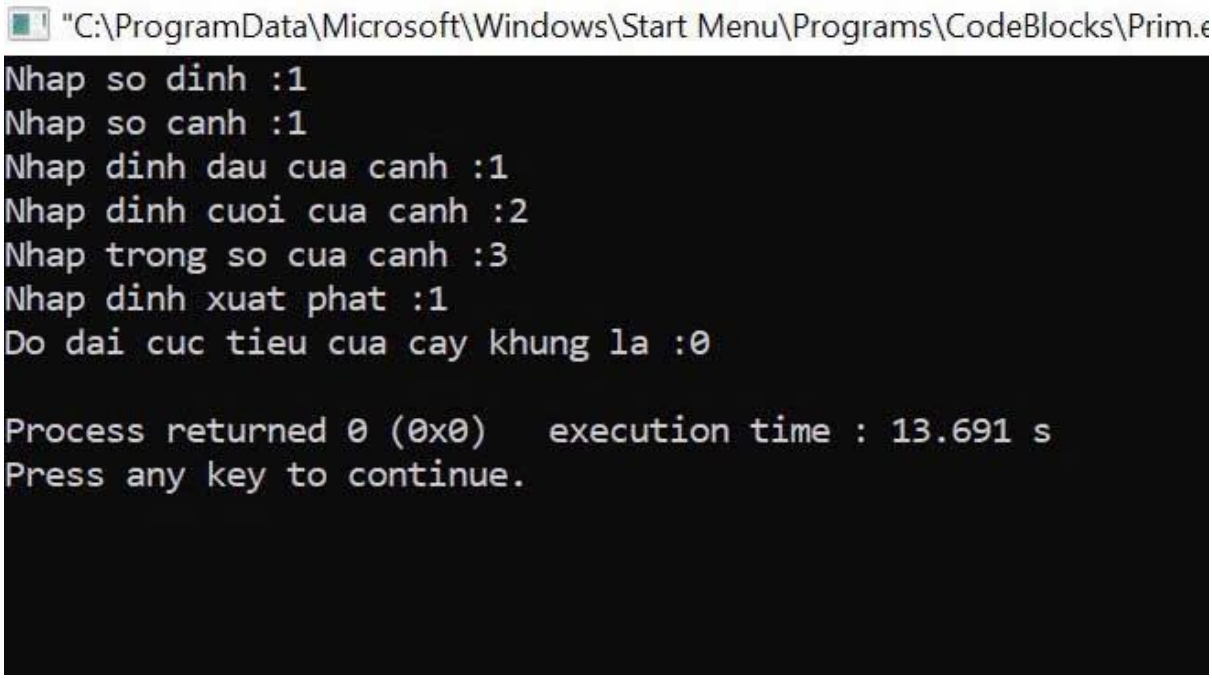
+Trường hợp 2: Trọng số bằng nhau.

```
Nhap so dinh :4
Nhap so canh :4
Nhap dinh dau cua canh :1
Nhap dinh cuoi cua canh :2
Nhap trong so cua canh :9
Nhap dinh dau cua canh :2
Nhap dinh cuoi cua canh :3
Nhap trong so cua canh :9
Nhap dinh dau cua canh :3
Nhap dinh cuoi cua canh :4
Nhap trong so cua canh :9
Nhap dinh dau cua canh :4
Nhap dinh cuoi cua canh :1
Nhap trong so cua canh :9
Nhap dinh xuat phat :1
Do dai cuc tieu cua cay khung la :27
1 2 9
1 4 9
2 3 9

Process returned 0 (0x0)   execution time : 29.381 s
Press any key to continue.
```

+Trường hợp 3:Có trọng số bằng 0.

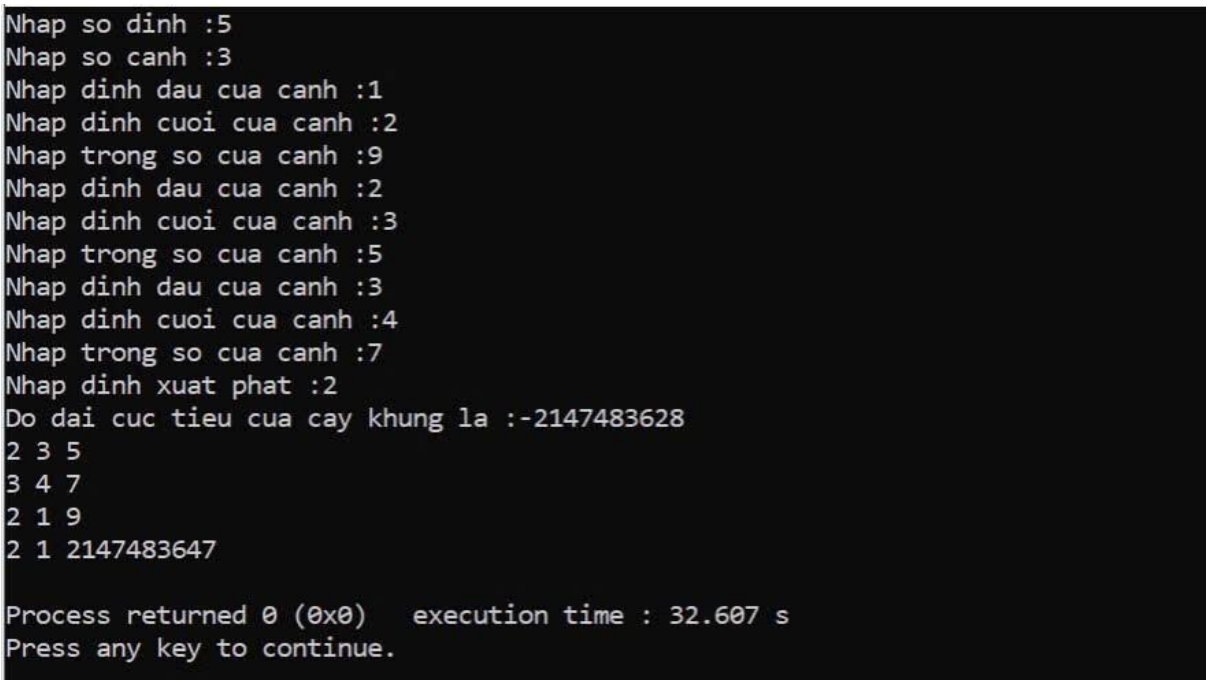
```
Nhap so dinh :5
Nhap so canh :5
Nhap dinh dau cua canh :1
Nhap dinh cuoi cua canh :2
Nhap trong so cua canh :0
Nhap dinh dau cua canh :2
Nhap dinh cuoi cua canh :3
Nhap trong so cua canh :0
Nhap dinh dau cua canh :3
Nhap dinh cuoi cua canh :4
Nhap trong so cua canh :0
Nhap dinh dau cua canh :4
Nhap dinh cuoi cua canh :5
Nhap trong so cua canh :0
Nhap dinh dau cua canh :5
Nhap dinh cuoi cua canh :1
Nhap trong so cua canh :0
Nhap dinh xuat phat :1
Do dai cuc tieu cua cay khung la :0
1 2 0
1 5 0
2 3 0
3 4 0

Process returned 0 (0x0)   execution time : 33.989 s
Press any key to continue.
```

+Trường hợp 4: Đồ thị 1 đỉnh.



=>Cây khung rỗng.

+Trường hợp 5: Đồ thị không liên thông.



=> Không tồn tại cây khung nhỏ nhất do đồ thị không liên thông.

+Trường hợp 6: Đồ thị lớn 1000 đỉnh.

Dữ liệu đầu vào:

- Số đỉnh: 1000
- Số cạnh: ít nhất 999, đảm bảo liên thông
- Trọng số ngẫu nhiên

Kết quả mong đợi:

- Cây khung có 999 cạnh
- Tổng trọng số phụ thuộc dữ liệu đầu vào
- Đánh giá hiệu năng thuật toán với đầu vào lớn (thời gian thực thi)

3.1.5. Nhận xét-Đánh giá.

- Thuật toán Prim hoạt động tốt trên đồ thị lớn nếu dùng hàng đợi ưu tiên.
- Mức độ phức tạp phù hợp bài toán thực tế.
- Có thể áp dụng trong mạng máy tính, thiết kế sơ đồ điện, v.v.

3.1.6 Kết luận.

- Đã xây dựng thành công giải thuật Prim.
- Kiểm thử đầy đủ với đồ thị nhỏ và đồ thị lớn.

3.2. Cài đặt và Kiểm thử thuật toán Kruskal.

3.2.1. Mục tiêu.

Xây dựng và kiểm thử thuật toán Kruskal để tìm cây khung nhỏ nhất (Minimum Spanning Tree - MST) của một đồ thị vô hướng, liên thông, có trọng số.

3.2.2. Giới thiệu thuật toán Kruskal.

Thuật toán Kruskal là một giải thuật tham lam (Greedy) dùng để tìm cây khung nhỏ nhất trong đồ thị.

Nó luôn chọn cạnh có trọng số nhỏ nhất trong số các cạnh còn lại mà không tạo chu trình.

Ý tưởng chính:

- Sắp xếp tất cả các cạnh theo trọng số tăng dần.
- Duyệt lần lượt từng cạnh, nếu thêm không tạo chu trình → thêm vào cây khung.
- Lặp lại cho đến khi có đủ n − 1 cạnh.

3.2.3. Cài đặt thuật toán Kruskal.

Ngôn ngữ lập trình C++

```cpp
#include<bits/stdc++.h>
using namespace std;

// luu canh cua do thi
struct canhdothi{
    int u; // u : dinh dau cua 1 canh
    int v; // v : dinh cuoi cua 1 canh
    int w; // w : trong so cua 1 canh
};
vector<canhdothi> canh; // luu cac canh cua do thi
const int maxn = 1001; // so dinh toi da cua do thi
int n; // so dinh cua do thi
int m; // so canh cua do thi
int cha[maxn]; // mang luu cha cua 1 tap dinh
int size[maxn]; // mang luu so phan tu cua 1 tap con thuoc 1 cha

// nhap cac canh cua do thi
void nhap(){
    cout <<"Nhap so dinh :";
    cin >> n; // nhap so dinh cua do thi
    cout <<"Nhap so canh :";
    cin >> m; // nhap so canh cua do thi
    // duyet qua tung canh cua do thi, nhap cac gia tri : dinh dau, dinh cuoi, trong so
    for(int i = 0; i < m; i++){
        int x, y, z;
```

```
        cout <<"Nhap dinh dau :";
        cin >> x; // nhap dinh dau
        cout <<"Nhap dinh cuoi :";
        cin >> y; // nhap dinh cuoi
        cout <<"Nhap trong so :";
        cin >> z; // nhap trong so
        canhdothi e; // tao 1 canh roi gan cac gia tri cho canh do
        e.u = x;
        e.v = y;
        e.w = z;
        canh.push_back(e); // them canh do vao do thi
    }
}

// duyet qua tung dinh cua do thi, gan cha la chinh no va size cua moi tap ban dau la 1
void khoitao(){
    for(int i = 1; i <= n; i++){
        cha[i] = i;
        size[i] = 1;
    }
}

// tim cha cua 1 tap con
int timcha(int v){
    if(v == cha[v]){
        return v;
    }// neu cha cua dinh do la chinh no thi return chinh no
    else{
        return cha[v] = timcha(cha[v]);
    }// nguoc lai, dung de quy de tim tiep cha cua dinh do
}

// noi 2 dinh voi nhau, neu noi duoc return true, nguoc lai return false
bool noidinh(int a, int b){
    a = timcha(a); // tim cha cua dinh a
    b = timcha(b); // tim cha cua dinh b
    if(a == b){
        return false;
    }// khong the noi 2 dinh a, b vi no co chung dinh cha
    else{
        if(size[a] < size[b]){
            cha[a] = b;
            size[b] += size[a];
            return true;
        }// neu size[a] < size[b] thi tat ca dinh con cua a co cha moi la b
        else{
            cha[b] = a;
            size[a] += size[b];
```

```
            return true;
        }// nguoc lai neu size[a] >= size[b] thi tat ca dinh con cua b co cha moi la a
    }// noi thanh cong 2 dinh a, b voi nhau
}

// ham so sanh trong so cac canh
bool cmp(canhdothi a, canhdothi b){
    return a.w < b.w;
}


void kruskal(){
    vector<canhdothi> mst; // luu cac canh cua cay khung
    int d = 0; // do dai ban dau cua cay khung
    sort(canh.begin(), canh.end(), cmp); // sap xep cac canh theo chieu tang dan cua trong so
    // duyet qua cac canh cua do thi, canh nao thoa man thi them vao cay khung
    for(int i = 0; i < m; i++){
        if(mst.size() == n - 1){
            break;
        }
        else{
            canhdothi e = canh[i]; // chon canh nho nhat
            if(noidinh(e.u, e.v) == true){
                mst.push_back(e);
                d += e.w; // do dai moi cua cay khung
            }
        }
    }
    // tra ve ket qua
    if(mst.size() != n - 1){
        cout <<"Do thi khong lien thong !"<< endl;
    }
    else{
        cout <<"Do dai cay khung la : "<< d << endl;
        cout <<"Cac canh trong cay khung la : "<< endl;
        for(int i = 0; i < mst.size(); i++){
            cout << mst[i].u << " " << mst[i].v << " " << mst[i].w << endl;
        }
    }
}


int main(){
    nhap();
    khoitao();
    kruskal();
    return 0;
}
```

3.2.4. Kiểm thử thuật toán kruskal.

+Trường hợp 1: Đồ thị nhỏ 3 đỉnh, liên thông.

```
Nhap so dinh :3
Nhap so canh :3
Nhap dinh dau :1
Nhap dinh cuoi :2
Nhap trong so :5
Nhap dinh dau :2
Nhap dinh cuoi :3
Nhap trong so :3
Nhap dinh dau :1
Nhap dinh cuoi :3
Nhap trong so :6
Do dai cay khung la : 8
Cac canh trong cay khung la :
2 3 3
1 2 5

--------------------------------
Process exited after 44.21 seconds with return value 0
Press any key to continue . . .
```

+Trường hợp 2: Trọng số bằng nhau.

```
Nhap so dinh :4
Nhap so canh :4
Nhap dinh dau :1
Nhap dinh cuoi :2
Nhap trong so :1
Nhap dinh dau :2
Nhap dinh cuoi :3
Nhap trong so :1
Nhap dinh dau :3
Nhap dinh cuoi :4
Nhap trong so :1
Nhap dinh dau :4
Nhap dinh cuoi :1
Nhap trong so :1
Do dai cay khung la : 3
Cac canh trong cay khung la :
1 2 1
2 3 1
3 4 1

--------------------------------
Process exited after 30.21 seconds with return value 0
Press any key to continue . . .
```

+Trường hợp 3: Có trọng số bằng 0.

```
Nhap so dinh :3
Nhap so canh :3
Nhap dinh dau :1
Nhap dinh cuoi :2
Nhap trong so :0
Nhap dinh dau :2
Nhap dinh cuoi :3
Nhap trong so :4
Nhap dinh dau :1
Nhap dinh cuoi :3
Nhap trong so :2
Do dai cay khung la : 2
Cac canh trong cay khung la :
1 2 0
1 3 2

--------------------------------
Process exited after 13 seconds with return value 0
Press any key to continue . . .
```

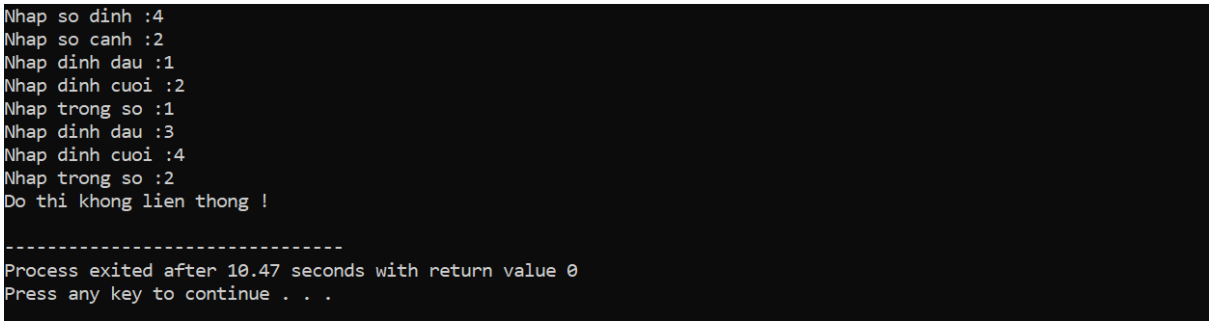
+Trường hợp 4: Đồ thị 1 đỉnh.

```
Nhap so dinh :1
Nhap so canh :0
Do dai cay khung la : 0
Cac canh trong cay khung la :

--------------------------------
Process exited after 11.13 seconds with return value 0
Press any key to continue . . .
```

=> Cây khung rỗng.

Tổng trọng số: 0

+Trường hợp 5: Đồ thị không liên thông.



+Trường hợp 6: Đồ thị lớn 1000 đỉnh.

Dữ liệu đầu vào:

- Số đỉnh: 1000
- Số cạnh: ít nhất 999, đảm bảo liên thông
- Trọng số ngẫu nhiên

Kết quả mong đợi:

- Cây khung có 999 cạnh
- Tổng trọng số phụ thuộc dữ liệu đầu vào
- Đánh giá hiệu năng thuật toán với đầu vào lớn (thời gian thực thi)

3.2.5. Nhận xét - Đánh giá.

- Thuật toán Kruskal đơn giản, dễ cài đặt và trực quan.
- Phù hợp với đồ thị thưa vì không cần duy trì danh sách kề hoặc hàng đợi ưu tiên.

- Hiệu quả cao nếu sử dụng cấu trúc Union-Find có tối ưu đường đi và hợp nhất theo rank.

- Có thể áp dụng trong thiết kế mạng, xây dựng đường giao thông, hệ thống điện, v.v.

3.2.6. Kết luận.

Thuật toán Kruskal là một trong những thuật toán hiệu quả để tìm cây khung nhỏ nhất. Với cách tiếp cận tham lam, thuật toán đảm bảo tìm được lời giải tối ưu trong thời gian hợp lý, đặc biệt là đối với đồ thị thưa. Đây là một giải pháp phù hợp trong nhiều bài toán thực tế liên quan đến tối ưu hóa chi phí kết nối các điểm trong mạng lưới.

3.3. Giải thuật di truyền.

3.3.1. Mục đích kiểm thử

Mục đích của kiểm thử là đánh giá:

- Tính đúng đắn của giải pháp (cây khung có hợp lệ không).

- Chất lượng lời giải (tổng trọng số).

- Tính ổn định của giải thuật qua nhiều lần chạy.

- Hiệu suất thời gian của thuật toán.

3.3.2. Môi trường kiểm thử

- Ngôn ngữ: C++

- Trình biên dịch: g++ 11.4.0

- Máy tính: Windows 10 / Ubuntu 22.04

- Cấu hình: RAM 8GB, Intel Core i5

3.3.3. Cài đặt giải thuật di truyền.

```cpp
#include <iostream>

#include <vector>

#include <algorithm>
#include <random>
#include <numeric>
#include <limits>
#include <unordered_set>
using namespace std;
struct Edge {
    int u, v, w;
    Edge(int u, int v, int w) : u(u), v(v), w(w) {}
    bool operator==(const Edge& other) const {
        return (u == other.u && v == other.v) || (u == other.v && v == other.u);
}
};
struct Individual {
    vector<Edge> edges;
    int fitness;
    Individual(const vector<Edge>& edges, int fitness) : edges(edges), fitness(fitness) {}
    bool operator<(const Individual& other) const {

        return fitness < other.fitness;
        }
};
class UnionFind {
private:
    vector<int> parent;
public:
    UnionFind(int size) {
    parent.resize(size);
        iota(parent.begin(), parent.end(), 0);
```

```cpp
    }
    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);
            }
            return parent[x];
            }
            bool unite(int x, int y) {
            int rootX = find(x);
            int rootY = find(y);
            if (rootX != rootY) {
            parent[rootX] = rootY;
            return true;
            }
        return false;
        }
};
vector<Individual> initializePopulation(const vector<Edge>& edges, int numVertices, int populationSize) {
    vector<Individual> population;
    random_device rd;
    mt19937 g(rd());
    while (population.size() < populationSize) {
        vector<Edge> shuffledEdges = edges;
        shuffle(shuffledEdges.begin(), shuffledEdges.end(), g);
        vector<Edge> candidate;
        UnionFind uf(numVertices);
        for (const Edge& edge : shuffledEdges) {
            if (uf.unite(edge.u, edge.v)) {
                candidate.push_back(edge);
                if (candidate.size() == numVertices - 1) break;
                }
        }
        if (candidate.size() == numVertices - 1) {
```

```
        int fitness = 0;
        for (const Edge& e : candidate) {
        fitness += e.w;
        }
            population.emplace_back(candidate, fitness);
        }
    }
    return population;
}
Individual selectIndividual(const vector<Individual>& population) {
    if (population.empty()) {
        throw invalid_argument("Qu?n th? r?ng, không th? ch?n cá th?!");
    }
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, population.size() - 1);
    return population[dis(gen)];
}
Individual repair(vector<Edge> edges, int numVertices) {
random_device rd;
mt19937 g(rd());
shuffle(edges.begin(), edges.end(), g);
        vector<Edge> repairedEdges;
        UnionFind uf(numVertices);
    for (const Edge& edge : edges) {
        if (uf.unite(edge.u, edge.v)) {
            repairedEdges.push_back(edge);
            if (repairedEdges.size() == numVertices - 1) break;
        }
    }
    int fitness = 0;
    for (const Edge& e : repairedEdges) {
        fitness += e.w;
```

```
}
    return Individual(repairedEdges, fitness);
}
Individual crossover(const Individual& parent1, const Individual& parent2, int numVertices) {
    vector<Edge> childEdges = parent1.edges;
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(0.0, 1.0);
    for (const Edge& edge : parent2.edges) {
    if (dis(gen) < 0.5 && find(childEdges.begin(), childEdges.end(), edge) == childEdges.end()) {
        childEdges.push_back(edge);
    }
}
  return repair(childEdges, numVertices);
}
void mutate(Individual& individual, const vector<Edge>& allEdges, int numVertices) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, individual.edges.size() - 1);
    uniform_int_distribution<> edgeDis(0, allEdges.size() - 1);
    int index = dis(gen);
    individual.edges.erase(individual.edges.begin() + index);
   for (int i = 0; i < allEdges.size(); i++) {
        const Edge& newEdge = allEdges[edgeDis(gen)];
        if (find(individual.edges.begin(), individual.edges.end(), newEdge) == individual.edges.end()) {
            individual.edges.push_back(newEdge);
            break;
        }
    }
    Individual repaired = repair(individual.edges, numVertices);
    individual = repaired;
}
Individual geneticAlgorithm(const vector<Edge>& edges, int numVertices, int populationSize,
```

```
            int maxGenerations, double crossoverRate, double mutationRate) {
            vector<Individual> population = initializePopulation(edges, numVertices, populationSize);
    if (population.empty()) {
            throw runtime_error("Không th? t?o qu?n th? h?p l?. ?? th? có th? không liên thông.");
        }
        random_device rd;
        mt19937 gen(rd());
        uniform_real_distribution<> dis(0.0, 1.0);
    for (int generation = 0; generation < maxGenerations; generation++) {
        vector<Individual> newPopulation;
        for (int i = 0; i < populationSize; i++) {
        Individual parent1 = selectIndividual(population);
        Individual parent2 = selectIndividual(population);
        Individual child = parent1;
            if (dis(gen) < crossoverRate) {
                child = crossover(parent1, parent2, numVertices);
            }
            if (dis(gen) < mutationRate) {
                mutate(child, edges, numVertices);
            }
newPopulation.push_back(child);
        }
        population = newPopulation;
    }
    return *min_element(population.begin(), population.end());
}
int main() {
    vector<Edge> edges = {
        {0, 1, 3},
        {0, 2, 3},
        {1, 2, 4},
        {1, 3, 6},
        {2, 3, 5},
        {3, 4, 7},
        {4, 0, 8}
```

```
    };
    int numVertices = 5;
    int populationSize = 20;
    int maxGenerations = 100;
    double crossoverRate = 0.8;
    double mutationRate = 0.3;
    try {
        Individual result = geneticAlgorithm(edges, numVertices, populationSize,
                maxGenerations, crossoverRate, mutationRate);
    cout << "Cay khung nho nhat:\n";
        for (const Edge& edge : result.edges) {
            cout << edge.u << " - " << edge.v << " : " << edge.w << endl;
        }
        cout << "Tong trong so: " << result.fitness << endl;
    }
    catch (const exception& e) {
    cerr << "Loi: " << e.what() << endl;
        return 1;
        }
 return 0;
}
```

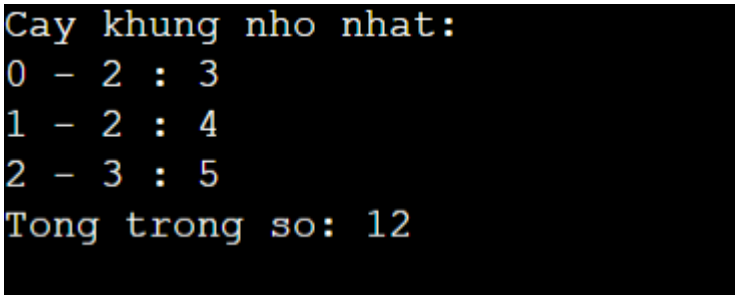
3.3.4. Kiểm thử giải thuật di truyền.

- Đồ thị nhỏ liên thông.

Thêm các cạnh:

```
vector<Edge> edges = {
    {0, 1, 3},
    {0, 2, 3},
    {1, 2, 4},
    {1, 3, 6},
    {2, 3, 5},
    {3, 4, 7},
    {4, 0, 8}
};
```

Kết quả:



3.3.5. Đánh giá thuật toán Di truyền trong bài toán này

Ưu điểm:

- Tìm được nghiệm hợp lệ (cây khung đầy đủ) mỗi lần chạy.
- Đa dạng hóa quần thể, giúp tiếp cận lời giải tốt hơn.
- Hiệu năng tốt trên đồ thị lớn (1000 đỉnh).
- Tính linh hoạt cao: dễ thay đổi thông số như mutation/crossover, kích thước quần thể.

Nhược điểm:

- Không đảm bảo tìm được nghiệm tối ưu tuyệt đối (so với Prim/Kruskal).
- Kết quả dao động: có độ lệch chuẩn tương đối, tức có thể tìm nghiệm kém hơn tối ưu 1–2%.
- Chi phí chạy nhiều vòng (generations) có thể lớn hơn thuật toán tham lam.

3.3.6. So sánh với thuật toán truyền thống (Prim/Kruskal)

Tiêu chí	Di truyền	Prim / Kruskal
Độ chính xác	Gần tối ưu (dao động nhẹ)	Luôn tối ưu
Tốc độ	Nhanh, nhưng chậm hơn Prim	Rất nhanh (đã tối ưu hóa)
Tính linh hoạt	Cao (thay đổi thông số dễ)	Thấp hơn
Ứng dụng mở rộng	Dễ tích hợp bài toán phức tạp	Khó mở rộng với ràng buộc mới

3.3.7. Kết luận

- Thuật toán di truyền hoạt động tốt trong việc tìm kiếm nghiệm gần tối ưu cho bài toán cây khung nhỏ nhất trên đồ thị lớn.

- Phù hợp cho các ứng dụng mà bài toán mở rộng thêm ràng buộc (điều kiện riêng), ví dụ: độ ưu tiên cạnh, hoặc chi phí thay đổi theo thời gian.

- Không nên thay thế hoàn toàn Prim/Kruskal nếu mục tiêu là tìm nghiệm tối ưu tuyệt đối đơn thuần.

Tài liệu tham khảo

- [1] Giáo trình Cấu trúc dữ liệu và giải thuật - Trường đại học Công nghiệp Hà Nội.

- [2] Cấu trúc dữ liệu và giải thuật - Tác giả Đỗ Xuân Lôi.

- [3] Minimum Spanning Tree - MST

https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

https://wiki.vnoi.info/algo/graph-theory/minimum-spanning-tree.md

- [4] Prim's Algorithm for Minium Spanning Tree - MST
https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo5/

- [5] Kruskal's Algorithm for Minium Spanning Tree - MST

https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/ -

- [6] Genetic Algorithm

https://www.geeksforgeeks.org/genetic-algorithms/

https://www.mathworks.com/discovery/genetic-algorithm.html

Đường Link liên kết youtube buổi họp online

Tuần 1:https://youtu.be/_wpqaRSIsqw

Tuần 2:https://youtu.be/99Wm_iCRCV0

Tuần 3:https://youtu.be/I6SwGDDzkGk

https://youtu.be/V4J4O317mok

Tuần 4:https://youtu.be/_uJwbDl95L4

Tuần 5:https://youtu.be/5KfGnXPsamg

Tuần 6:https://youtu.be/TTrQv-xCQGA

Tuần 7:https://youtu.be/Wxwy6aTr87Q

Tuần 8: https://youtu.be/VG4RkKT5hck

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 1

Thời gian: 14h00, ngày 09/03/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

1. **Mục đích cuộc họp**

- Thảo luận, thống nhất chủ đề nghiên cứu.
- Bầu Nhóm Trưởng.
- Phân công công việc giữa các thành viên.
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

- **Giới thiệu chủ đề**
  Nhóm lựa chọn nghiên cứu bài toán cây khung nhỏ nhất, một bài toán cơ bản trong lý thuyết đồ thị và có nhiều ứng dụng trong mạng máy tính, thiết kế mạch, giao thông...

- **Phân công công việc ban đầu:**

Họ và tên	**Nhiệm vụ được phân công**
Nguyễn Thị Bích Ngọc:	Nghiên cứu và cài đặt **thuật toán di truyền** áp dụng cho bài toán cây khung nhỏ nhất
Nghiêm Thị Ngọc Anh:	Phụ trách kiểm thử chương trình, xây dựng kịch bản kiểm thử, kiểm tra kết quả
Vũ Huy Đạt:	Nghiên cứu lý thuyết và cài đặt thuật toán Prim bằng C++
Nguyễn Trúc Linh:	Nghiên cứu lý thuyết và cài đặt thuật toán Kruskal bằng C++
Nguyễn Minh Trí:	Thiết kế báo cáo, tổng hợp tài liệu, Viết biên bản báo cáo cuộc họp (trưởng nhóm)

- **Thống nhất định dạng báo cáo**:
  Gồm 4 chương chính:

  1. Mô tả bài toán và mục tiêu nghiên cứu
  2. Cơ sở lý thuyết (thuật toán, ứng dụng thực tiễn)
  3. Phân tích thiết kế hệ thống (mô hình, biểu đồ)

4. Cài đặt và kiểm thử

---

3. Kết luận và kế hoạch tiếp theo

- Thời hạn hoàn thành phần lý thuyết và mã nguồn cơ bản: **27/04/2025**
- Buổi họp tiếp theo: **14h00 ngày 16/03/2025**
- Mỗi thành viên có trách nhiệm hoàn thành phần được giao và báo cáo tiến độ vào buổi họp sau.

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 2

Thời gian: 14h00, ngày 16/03/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

2. Mục đích cuộc họp

- Thảo luận, thống nhất lại về phần công việc
- Phân công công việc giữa các thành viên.
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

Phân công công việc ban đầu:

Họ và tên	**Nhiệm vụ được phân công**
Nguyễn Thị Bích Ngọc:	Nghiên cứu và cài đặt **thuật toán di truyền** áp dụng cho bài toán cây khung nhỏ nhất
Nghiêm Thị Ngọc Anh:	Phụ trách kiểm thử chương trình, xây dựng kịch bản kiểm thử, kiểm tra kết quả
Vũ Huy Đạt:	Nghiên cứu lý thuyết và cài đặt thuật toán Prim bằng C++
Nguyễn Trúc Linh:	Nghiên cứu lý thuyết và cài đặt thuật toán Kruskal bằng C++
Nguyễn Minh Trí:	Thiết kế báo cáo, , Viết biên bản báo cáo cuộc họp,tìm hiểu về I. Cơ Sở lí thuyết (trưởng nhóm)

---

3. Kết luận và kế hoạch tiếp theo

- Thời hạn hoàn thành phần lý thuyết và mã nguồn cơ bản: **27/04/2025**
- Buổi họp tiếp theo: **14h00 ngày 23/03/2025**

Mỗi thành viên có trách nhiệm hoàn thành phần được giao và báo cáo tiến độ vào buổi họp sau.

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 3

Thời gian: 14h00, ngày 23/03/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

3. Mục đích cuộc họp

- Báo cáo về tiến độ công việc của mỗi thành viên
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

Họ và tên	**Phần việc hoàn thành**
Nguyễn Thị Bích Ngọc:	Trình bày về Cơ sở lý thuyết của bài di truyền
Nghiêm Thị Ngọc Anh:	Xây dựng **kịch bản kiểm thử**, kiểm tra kết quả
Vũ Huy Đạt:	Trình bày về Cơ sở lý thuyết của thuật toán **Prim**
Nguyễn Trúc Linh:	Trình bày về Cơ sở lý thuyết của thuật toán **Kruskal**
Nguyễn Minh Trí:	Trình bày về Cơ sở lý thuyết của bài toán cây khung nhỏ nhất,viết biên bản báo cáo cuộc họp

---

3. Kết luận và kế hoạch tiếp theo

- Thời hạn hoàn thành phần lý thuyết và mã nguồn cơ bản: **30/04/2025**
- Buổi họp tiếp theo: **14h00 ngày 30/03/2025**
- Mỗi thành viên có trách nhiệm hoàn thành phần được giao và báo cáo tiến độ vào buổi họp sau.

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 4

Thời gian: 14h00, ngày 30/03/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

4. **Mục đích cuộc họp**

- Báo cáo về tiến độ công việc của mỗi thành viên
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

Họ và tên	**Phần việc hoàn thành**
Nguyễn Thị Bích Ngọc:	Trình bày về độ phức tạp, ưu nhược điểm di truyền
Nghiêm Thị Ngọc Anh:	Xây dựng **kịch bản kiểm thử**, kiểm tra kết quả
Vũ Huy Đạt:	Trình bày về độ phức tạp, ưu nhược điểm Prim
Nguyễn Trúc Linh:	Trình bày về độ phức tạp, ưu nhược điểm Kruskal
Nguyễn Minh Trí:	Trình bày về Cơ sở lý thuyết của bài toán cây khung nhỏ nhất,viết biên bản báo cáo cuộc họp

---

3. Kết luận và kế hoạch tiếp theo

- Thời hạn hoàn thành phần lý thuyết và mã nguồn cơ bản: **27/04/2025**
- Buổi họp tiếp theo: **14h00 ngày 06/04/2025**
- Mỗi thành viên có trách nhiệm hoàn thành phần được giao và báo cáo tiến độ vào buổi họp sau.

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 5

Thời gian: 14h00, ngày 06/04/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

5. Mục đích cuộc họp

- Báo cáo về tiến độ công việc của mỗi thành viên
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

Họ và tên	**Phần việc hoàn thành**
Nguyễn Thị Bích Ngọc:	Thiết kế thuật toán giải thuật di truyền : Bài toán, giải thuật
Nghiêm Thị Ngọc Anh:	Xây dựng **kịch bản kiểm thử**, kiểm tra kết quả
Vũ Huy Đạt:	Thiết kế thuật toán Prim: Bài toán, giải thuật
Nguyễn Trúc Linh:	Thiết kế thuật toán Kruskal: Bài toán, giải thuật
Nguyễn Minh Trí:	Viết biên bản báo cáo cuộc họp,làm báo cáo bài tập lớn

---

3. Kết luận và kế hoạch tiếp theo

- Thời hạn hoàn thành phần lý thuyết và mã nguồn cơ bản: **27/04/2025**
- Buổi họp tiếp theo: **14h00 ngày 13/04/2025**
- Mỗi thành viên có trách nhiệm hoàn thành phần được giao và báo cáo tiến độ vào buổi họp sau.

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 6

Thời gian: 14h00, ngày 13/04/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

6. Mục đích cuộc họp

- Báo cáo về tiến độ công việc của mỗi thành viên
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

Họ và tên	**Phần việc hoàn thành**
Nguyễn Thị Bích Ngọc:	Thiết kế thuật toán giải thuật di truyền : Bài toán, giải thuật
Nghiêm Thị Ngọc Anh:	Cài đặt và kiểm thử thuật toán Prim,Kruskal
Vũ Huy Đạt:	Thiết kế thuật toán Prim: Bài toán, giải thuật
Nguyễn Trúc Linh:	Thiết kế thuật toán Kruskal: Bài toán, giải thuật
Nguyễn Minh Trí:	Viết biên bản báo cáo cuộc họp,làm báo cáo bài tập lớn

---

3. Kết luận và kế hoạch tiếp theo

- Thời hạn hoàn thành phần lý thuyết và mã nguồn cơ bản: **27/04/2025**
- Buổi họp tiếp theo: **14h00 ngày 20/04/2025**

Mỗi thành viên có trách nhiệm hoàn thành phần được giao và báo cáo tiến độ vào buổi họp sau.

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 7

Thời gian: 14h00, ngày 20/04/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

7. Mục đích cuộc họp

- Báo cáo về tiến độ công việc của mỗi thành viên
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

Họ và tên	**Phần việc hoàn thành**
Nguyễn Thị Bích Ngọc:	Trình bày về phần code thuật toán giải thuật di truyền
Nghiêm Thị Ngọc Anh:	Cài đặt và kiểm thử thuật toán Prim,Kruskal,giải thuật di truyền
Vũ Huy Đạt:	Trình bày về phần code thuật toán Prim
Nguyễn Trúc Linh:	Trình bày về phần code thuật toán Kruskal
Nguyễn Minh Trí:	Viết biên bản báo cáo cuộc họp,làm báo cáo bài tập lớn

---

3. Kết luận và kế hoạch tiếp theo

- Thời hạn hoàn thành phần lý thuyết và mã nguồn cơ bản: **27/04/2025**
- Buổi họp tiếp theo: **14h00 ngày 27/04/2025**
- Mỗi thành viên có trách nhiệm hoàn thành phần được giao và báo cáo tiến độ vào buổi họp sau.

BIÊN BẢN CUỘC HỌP NHÓM 3 TUẦN 8

Thời gian: 14h00, ngày 03/04/2025

Địa điểm: Họp online qua Zoom

Chủ đề: *Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt thuật toán để giải bài toán Cây khung nhỏ nhất (Minimal Spanning Tree Problem)*

Thành phần tham dự: Toàn bộ các thành viên trong nhóm (5 người)

---

8. Mục đích cuộc họp

- Báo cáo về tiến độ công việc của mỗi thành viên
- Đặt ra tiến độ, kế hoạch thực hiện các giai đoạn tiếp theo.

---

2. Nội dung trao đổi chính

Họ và tên	**Phần việc hoàn thành**
Nguyễn Thị Bích Ngọc	Sửa báo cáo và làm slide
Nghiêm Thị Ngọc Anh	Sửa báo cáo và làm slide
Vũ Huy Đạt	Tóm tắt nội dung để tổng hợp thông tin làm slide
Nguyễn Trúc Linh	Sửa các định dạng báo cáo và làm slide
Nguyễn Minh Trí	Sửa báo cáo,viết biên bản báo cáo cuộc họp hoàn thiện báo cáo

---

3. Kết luận và kế hoạch tiếp theo

- Mỗi thành viên Hoàn thành tốt phần nhiệm vụ được giao