BỘ KHOA HỌC VÀ CÔNG NGHỆ
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO MÔN HỌC
LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

ĐỀ TÀI: XÂY DỰNG CÂY KHUNG TỐI THIỂU (MST)
ÁP DỤNG THUẬT TOÁN PRIM

Giảng viên: NGUYỄN THỊ TUYẾT HẢI

Nhóm sinh viên thực hiện:

Huỳnh Thị Hồng Thắm	N23DCCN055	Trưởng nhóm
Nguyễn Văn Hậu	N23DCCN018	Thành viên
Võ Hà Như Thủy	N23DCCN061	Thành viên

TP.HCM, tháng 11/2025

MỤC LỤC

DANH SÁCH HÌNH, BẢNG

TÓM TẮT

Cây khung tối thiểu (MST - Minimum Spanning Tree) là một cấu trúc đồ thị vô hướng có trọng số, giúp tối ưu hóa hệ thống mạng với chi phí thấp nhất. Trong bối cảnh thiết kế mạng lưới, thuật toán Prim cho phép tìm MST bằng cách chọn từng cạnh nhỏ nhất từ tập hợp đỉnh đã chọn, đảm bảo hệ thống được kết nối với tổng chi phí tối thiểu.

Báo cáo này phân tích cách triển khai thuật toán Prim:

Sử dụng danh sách kề + hàng đợi ưu tiên Heap, tối ưu với đồ thị lớn hoặc thưa, giúp lựa chọn cạnh nhẹ nhất nhanh nhờ cấu trúc dữ liệu ưu tiên.

Bảng phân công nhiệm vụ:

TÊN	MSSV	CHỨC VỤ	Nhiệm vụ
Võ Hà Như Thủy	N23DCCN061	Thành viên	Chọn đề tài, hoàn thiện chương I, II
Huỳnh Thị Hồng Thắm	N23DCCN055	Nhóm trưởng	Hỗ trợ chỉnh sửa, hoàn thiện chương III
Nguyễn Văn Hậu	N23DCCN018	Thành viên	Tạo giao diện, hoàn thiện chương IV

Link tới bản demo: https://github.com/ngvanhau1604/OOP_FinalProject.git

CHƯƠNG I. TỔNG QUAN VỀ ĐỀ TÀI

1.1 Giới thiệu đề tài.

1.1.1 Cây khung tối thiểu

Trong thực tế, nhiều bài toán trong lĩnh vực kỹ thuật, giao thông, viễn thông và tin học đều có thể được mô hình hóa dưới dạng đồ thị. Một trong những bài toán cơ bản và quan trọng nhất là tìm cách kết nối tất cả các điểm trong hệ thống với chi phí thấp nhất mà vẫn đảm bảo được tính liên thông của đồ thị. Bài toán này được mô hình hóa thành bài toán Cây khung tối thiểu trong đồ thị.

Cây khung tối thiểu (Minimum Spanning Tree – MST) là một tập hợp các cạnh của đồ thị kết nối tất cả các đỉnh (nút) mà không tạo thành chu trình và tổng trọng số của các cạnh là nhỏ nhất. Trong đó:

- Đỉnh: các điểm (nút) trong đồ thị cần kết nối với nhau. Ví dụ: trạm điện, nút giao thông, …
- Cạnh: các kết nối giữa các đỉnh (dây điện, tuyến đường, …).
- Trọng số: giá trị (hoặc chi phí) gắn với mỗi cạnh.

1.1.2 Ứng dụng của cây khung tối thiểu về thiết kế mạng lưới trong thực tế

- Viễn thông: MST được sử dụng để thiết kế mạng lưới truyền thông hiệu quả, giảm thiểu chi phí lắp đặt cáp.
- Mạng máy tính: Trong mạng LAN, MST giúp thiết kế cấu trúc mạng để đảm bảo độ trễ và chi phí ở mức tối thiểu.
- Mạng giao thông: Quy hoạch đường đi kết nối các thành phố với chi phí xây dựng thấp nhất.
- Mạng lưới điện: Thiết kế hệ thống truyền tải điện, giảm chi phí dây dẫn.
- Hệ thống cấp nước: Xây dựng mạng lưới đường ống hiệu quả để kết nối nguồn nước với khu dân cư.
- Mạng IoT: Tối ưu hóa kết nối giữa các thiết bị IoT để truyền dữ liệu hiệu quả.
- Mạng xã hội: Xác định kết nối quan trọng nhất trong mạng xã hội.
- Phân cụm dữ liệu: Áp dụng vào thuật toán phân cụm để nhóm dữ liệu tương đồng.
- Robot đa tác vụ: Điều phối di chuyển và nhiệm vụ trong hệ thống nhiều robot.
- An ninh mạng: Thiết kế mạng an toàn hơn bằng cách giảm thiểu các điểm có thể bị tấn công.

Thuật toán được sử dụng trong đề tài: PRIM

Ngôn ngữ lập trình: Java

1.2 Cơ sở lý thuyết.

1.2.1 Đồ thị

Đồ thị là một cấu trúc rời rạc gồm tập hợp các đỉnh và các cạnh nối giữa các đỉnh đó G = (V, E), tức là đồ thị G có tập các đỉnh là V, tập các cạnh là E. Có thể hiểu E là tập hợp các cặp (u, v) với u và v là hai đỉnh thuộc V.

- Đồ thị vô hướng (hoặc đồ thị G) là một cặp không có thứ tự G := (V,E), trong đó:
  + V (Vertices): tập các đỉnh hoặc nút.
  + E (Edges): 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 đỉnh đó.
- Đồ thị có hướng G là một cặp có thứ tự G := (V,E) , trong đó:
  + V: tập các đỉnh hoặc nút.
  + E: tập các cặp có thứ tự chứa các đỉnh, được gọi là các cạnh có hướng hoặc cung. Một cạnh e = (x, y) được coi là cạnh có hướng từ x tới y; x được gọi là điểm đầu (gốc) và y được gọi là điểm cuối (ngọn) của cạnh.

1.2.2 Các khái niệm trên đồ thị

a) Cạnh liên thuộc, đỉnh kề, bậc và khuyên

- Đối với đồ thị vô hướng G = (V, E), xét cạnh e = (u, v) ∈ E. Ta nói rằng hai đỉnh u và v kề nhau (adjacent) và cạnh e này liên thuộc (incident) với hai đỉnh u và v.
- Bậc (degree) của đỉnh u thuộc đồ thị được kí hiệu là deg(u), là số cạnh liên thuộc với u. Trên đơn đồ thị, số cạnh liên thuộc với u cũng chính là số đỉnh kề với u.
- Cạnh khuyên: Trên đồ thị có hướng hoặc vô hướng, trong một số trường hợp có thể có những cạnh nối một đỉnh với chính nó, cạnh này được gọi là khuyên của đồ thị.

b) Đường đi và chu trình

- Đường đi: Một đường đi P độ dài từ đỉnh v_0 tới đỉnh v_k là tập đỉnh $\{v_0, v_1, v_2, ..., v_k\}$ sao cho (v_{i-1}, v_i) □ E, □i: $1 \leq i$ □ k. Khi đó ta nói đường đi này gồm các đỉnh $\{v_0, v_1, v_2, \ldots, v_k\}$ và các cạnh $\{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\}$; và v_0 đến được v_k thông qua đường đi P.
- Đường đi được gọi là đường đi đơn giản (simple path) nếu tất cả các đỉnh trên đường đi đó đều phân biệt.
- Đường đi đơn: là đường đi mà không có cạnh nào trên đường đi đó qua hơn một lần.
- Chu trình (circuit) là đương đi P mà điểm đầu và điểm cuối trùng nhau.

- Chu trình đơn giản (simple circuit) là chu trình P có $\{v_1, v_2, \ldots, v_k\}$ đôi một khác nhau.
- Chu trình đơn: chu trình mà trong đó không có cạnh nào đi qua hơn 1 lần.

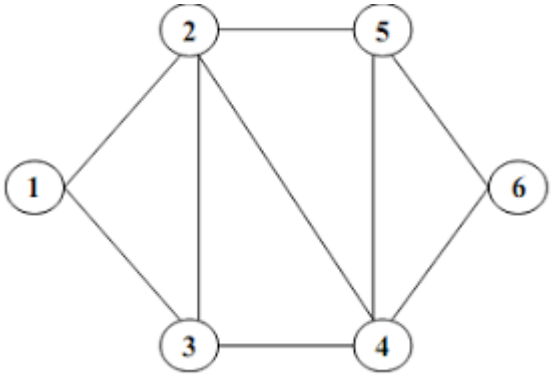
c) Tính liên thông của đồ thị

- Đối với đồ thị vô hướng G = (V, E) thì G được gọi là liên thông nếu như với mọi cặp đỉnh phân biệt (u, v), ta đều có u đến được v (đồng nghĩa với v cũng đến được u).
- Đối với đồ thị có hướng G = (V, E):
  + G được gọi là liên thông mạnh (strongly connected) nếu mọi cặp đỉnh phân biệt (u, v), ta có u đến được v và v đến được u.
  + G được gọi là liên thông yếu (weakly connected) nếu như đồ thị vô hướng nền của nó là liên thông (tức là hủy bỏ chiều của các cạnh trên đồ thị thì đồ thị đó liên thông).

1.2.3 Danh sách kề

Danh sách kề được xem là cách thích hợp nhất trong rất nhiều vấn đề ứng dụng của lý thuyết đồ thị.

Trong cách biểu diễn này, với mỗi đỉnh v của đồ thị chúng ta lưu trữ danh sách các đỉnh kề với nó, kí hiệu là Ke(v)= {u ∈ V: (v, u) ∈ E}



Ke(1) = { 2, 3}.

Ke(2) = { 1, 3, 4, 5}.

Ke(3) = { 1, 2, 4}.

Ke(4) = { 2, 3, 5, 6}.

Ke(5) = { 2, 4, 6}.

Ke(6) = { 4, 5}.

Hình 1.2.3.1 Đồ thị vô hướng G và danh sách kề của G

- Biểu diễn danh sách kề: Tạo mảng NMAX giá trị, mảng giá trị thứ u lưu danh sách các đỉnh kề với u

```
import java.util.*;

public class AdjacencyList {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Nhap so dinh (n) va so canh (m): ");
        int n = sc.nextInt();
        int m = sc.nextInt();

        List<List<Integer>> adj = new ArrayList<>();
        for (int i = 0; i <= n; i++) {
            adj.add(new ArrayList<>());
        }
        System.out.println("Nhap cac canh (dinh dau, dinh cuoi):");
        for (int i = 0; i < m; i++) {
            int x = sc.nextInt();
            int y = sc.nextInt();
            adj.get(x).add(y);
            adj.get(y).add(x);
        }
        System.out.println("\nDanh sach ke cua tung dinh:");
        for (int i = 1; i <= n; i++) {
            System.out.print(i + ": ");
            for (int v : adj.get(i)) {
                System.out.print(v + " ");
            }
            System.out.println();
        }
    }
}
}
```

Ta được kết quả:

```
Ke(1): 2 3
Ke(2): 1 3 4 5
Ke(3): 1 2 4
Ke(4): 2 3 5 6
Ke(5): 2 4 6
Ke(6): 4 5
```

Hình 1.2.3.2 Kết quả biểu diễn danh sách kề

Ưu điểm: Tiết kiệm bộ nhớ cho đồ thị thưa, truy xuất nhanh các đỉnh kề.

1.2.4 Hàng đợi ưu tiên

Hàng đợi ưu tiên là một kiểu dữ liệu trừu tượng tương tự như kiểu dữ liệu trừu tượng hàng đợi thông thường hoặc ngăn xếp .

Trong hàng đợi ưu tiên, mỗi phần tử có một *mức độ ưu tiên* liên quan, xác định thứ tự dịch vụ của phần tử đó. Hàng đợi ưu tiên phục vụ các mục có mức độ ưu tiên cao nhất trước. Các giá trị ưu tiên phải là các thể hiện của một kiểu dữ liệu được sắp xếp và mức độ ưu tiên cao hơn có thể được đưa ra cho các giá trị nhỏ hơn hoặc lớn hơn đối với mối quan hệ thứ tự đã cho.

Hàng đợi ưu tiên thường được triển khai bằng cách sử dụng heap.

Cấu trúc dữ liệu Heap:

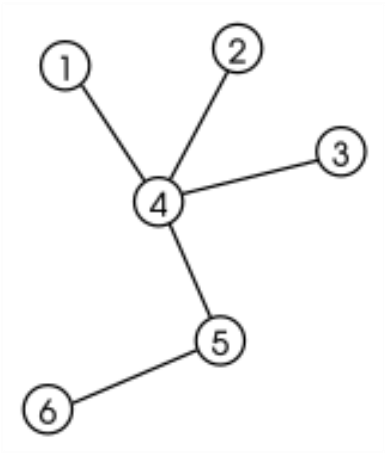
- Cấu trúc dữ liệu heap là một trường hợp đặc biệt của câú trúc dữ liệu cây nhị phân cân bằng, trong đó khóa nút gốc được so sánh với các con của nó và được sắp xếp một cách cân bằng.
- Có hai loại Heap chính:
  + Max – Heap: Nút cha có giá trị lớn hơn hoặc bằng giá trị của các nút con. Phần tử lớn nhất luôn nằm ở gốc.
  + Min – Heap: Nút cha có giá trị nhỏ hơn hoặc bằng giá trị của các nút con. Phần tử nhỏ nhất luôn nằm ở gốc.

Hàng đợi có ưu tiên được chia làm 2 loại:

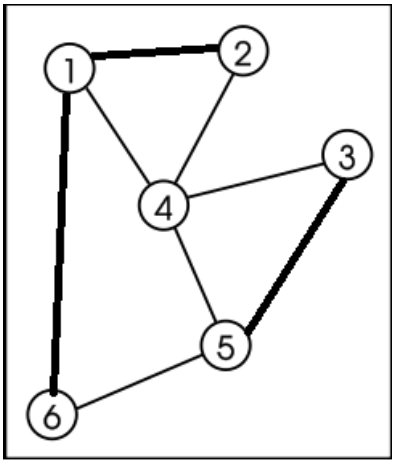
- Hàng đợi ưu tiên theo giá trị lớn nhất (max – priority queue): node có giá trị cao nhất sẽ được lấy ra trước.
- Hàng đợi ưu tiên theo giá trị thấp nhất (min – priority queue): node có giá trị thấp nhất sẽ được lấy ra trước.

1.2.5 Cây khung đồ thị (Spanning Tree)

- Cây là một đồ thị vô hướng liên thông và không chứa chu trình và có ít nhất hai đỉnh.



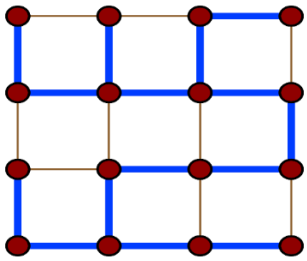
Hình 1.2.5.1 Cây



Hình 1.2.5.2 Không phải là cây

- Cây khung: Cho G = (V, E) là đồ thị vô hướng liên thông. Ta gọi đồ thị con T của G là một cây khung (cây bao trùm) của G nếu T thỏa mãn hai điều kiện:
  + T là một cây.

+ Tập đỉnh của T bằng tập đỉnh của G



Hình 1.2.5.3 Cây khung

– Cấu trúc cây tổng quát:
  + Cây là một cấu trúc gồm 1 tập hữu hạn các node có cùng kiểu dữ liệu.
  + Các node được nối với nhau bởi 1 cạnh.
  + Có duy nhất 1 node gốc.
  + Có duy nhất 1 đường đi từ node gốc đến 1 node bất kỳ.
– Cây m phân:
  + Bậc của một nút pi (degree of node) là số nút con của nút pi.
  + Bậc của cây (degree of tree): là bậc lớn nhất của tất cả các node trên cây.
  + Cây m phân: một cây có gốc T với mỗi đỉnh có nhiều nhất là m con.
  + Cây nhị phân: cây m phân có m=2, mỗi con được chỉ rõ là con bên trái hay con bên phải; con bên trái (hoặc bên phải) được vẽ phía dưới về bên trái (hoặc bên phải) của cha.

1.2.6 Cây khung tối thiểu

Cho G = (V, E) là đồ thị vô hướng liên thông có trọng số, mỗi cạnh e ∈ E có trọng số m(e)≥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.

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 của đồ thị.

1.2.7 Thuật toán PRIM

Thuật toán Prim là một thuật toán tham lam để tìm cây bao trùm nhỏ nhất của một đồ thị vô hướng có trọng số liên thông. Nghĩa là nó tìm một tập hợp các cạnh của đồ thị tạo thành một cây chứa tất cả các đỉnh, sao cho tổng trọng số các cạnh của cây là nhỏ nhất.

Thuật toán được tìm ra năm 1930 bởi nhà toán học người Séc Vojtěch Jarník và sau đó bởi nhà nghiên cứu khoa học máy tính Robert C. Prim năm 1957 và một lần nữa độc lập bởi Edsger Dijkstra năm 1959. Do đó nó còn được gọi là thuật toán DJP, thuật toán Jarník, hay thuật toán Prim–Jarník.

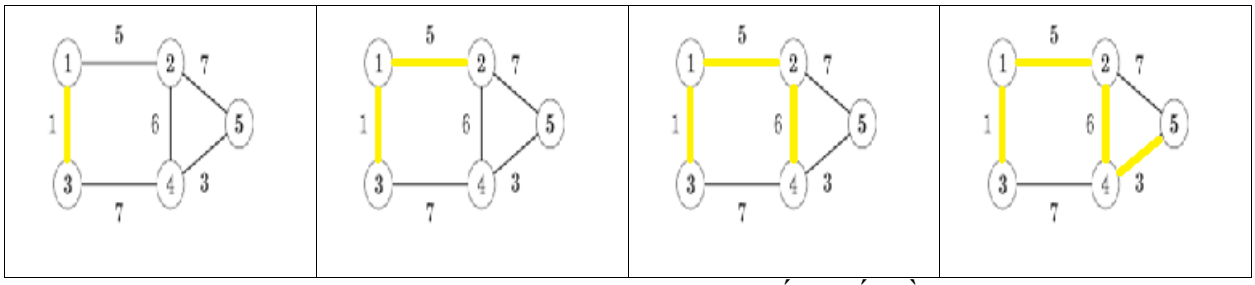
Input: đồ thị liên thông G = (V, E); với V gồm n đỉnh, E gồm m cạnh và mỗi cạnh e = (u, v) đều có trọng số w(u, v).

Output: Một cây khung tối thiểu MST bao gồm tập cạnh T của cây khung tối thiểu và tổng trọng số d(H) của cây khung nhỏ nhất.

Các bước thực hiện thuật toán Prim:

- Bước 1: Chọn tùy ý đỉnh s ∈ V và khởi tạo V_H={s}; T=∅, d(H) = 0.
- Bước 2: Trong các cạnh e = (u, v) nối từ những đỉnh có trong V_H (v ∈ V_H) và những đỉnh có trong V-V_H (u ∈ V-V_H), chọn cạnh có trọng số nhỏ nhất e.
- Bước 3: V_H = V_H ∪ u và T = T ∪ {e}, d(H) = d(H) + d(e).
- Bước 4: Nếu T đủ n-1 cạnh ⇒ dừng.

  Ngược lại, quay lại bước 2.



Hình 1.2.7.1 Thực hiện tìm đường đi ngắn nhất bằng Prim

Cách cài đặt thuật toán PRIM:

- Hàng đợi có ưu tiên Heap trong danh sách kề: sử dụng Min – Heap priority queue.

Đặc điểm thuật toán:

Tiêu chí	Hàng đợi ưu tiên Heap + Danh sách kề
Thứ tự duyệt	Duyệt qua danh sách kề của từng đỉnh O(E)
Cấu trúc dữ liệu	Danh sách kề lưu các cạnh liên kết với từng đỉnh; Heap giúp truy xuất cạnh nhỏ nhất nhanh.
Thời gian tính toán	O(ElogV)
Không gian tính toán	O(V+E): tiết kiệm bộ nhớ, phù hợp với đồ thị thưa
Ứng dụng phổ biến	Dùng trong đồ thị lớn, thuật toán Prim tối ưu, Dijkstra với Heap

CHƯƠNG II. THIẾT KẾ THUẬT TOÁN

2.1 Thuật toán PRIM

2.1.1 Nguyên lý hoạt động:

Trong một đồ thị liên thông có trọng số:

- Bắt đầu từ một đỉnh bất kỳ trong đồ thị.
- Ở mỗi bước, chọn cạnh có trọng số nhỏ nhất nối từ một đỉnh đã được chọn đến một đỉnh chưa được chọn.
- Thêm đỉnh mới vào MST và tiếp tục chọn cạnh nhỏ nhất tiếp theo.
- Lặp lại quá trình trên cho tới khi mọi đỉnh đều được duyệt qua và đưa vào cây khung.

  Thuật toán đảm bảo tạo ra một cây khung có tổng trọng số nhỏ nhất, không chứa chu trình.

2.1.2 Giải thích bài toán thiết kế mạng lưới với chi phí tối thiểu

Mô hình hóa bài toán:

- Đỉnh (node): biểu diễn các điểm cần kết nối như thành phố, máy chủ, trạm truyền dẫn, điểm phân phối, …
- Cạnh (edge): biểu diễn đường dây cáp, kênh truyền, hay kết nối giữa hai nút.
- Trọng số cạnh: là chi phí triển khai kết nối hai điểm đó.

Mục tiêu: Thiết kế một mạng lưới kết nối tất cả các điểm mà tổng chi phí là nhỏ nhất và không có chu trình.

=> Đây chính là bài toán cây khung tối thiểu (MST) và thuật toán Prim là giải pháp hiệu quả.

Thuật toán PRIM (s):
Begin:
Bước 1 (Khởi tạo):
$V_H = \{s\}$; //Tập đỉnh cây khung thiết lập ban đầu là s
V = V\{s}; //Tập đỉnh V được bớt đi s
T = ∅; //Tập cạnh cây khung thiết lập ban đầu là ∅
d(H) = 0; //Độ dài cây khung được thiết lập là 0
Bước 2 (Lặp):
while (V≠ ∅) do {
e = <u, v>: cạnh có độ dài nhỏ nhất thỏa mãn u∈V, v∈V_H;
d(H) = d(H) + d(e); // Thiết lập đồ dài cây khung nhỏ nhất
T = T ∪ {e}; //Kết nạp e vào cây khung
V = V \{u}; // Tập đỉnh V bớt đi đỉnh u
$V_H = V_H \cup \{u\}$; // Tập đỉnh V_H thêm vào đỉnh u
endwhile;
Bước 3 (Trả lại kết quả):
if (|T|<n-1) then <Đồ thị không liên thông>;
else Return(T, d(H));
End.

Hình 2.1.2.1 Mã giả thuật toán Prim

2.2 Thuật toán PRIM sử dụng hàng đợi có ưu tiên Heap trong danh sách kề

Ý tưởng triển khai: thuật toán được cài đặt dựa trên cấu trúc

- Danh sách kề adj[u] = list<(v, w)> để lưu các cạnh xuất phát từ đỉnh u.
- Hàng đợi ưu tiên Heap (Min – Priority Heap) để luôn chọn đỉnh có chi phí nhỏ nhất đưa vào MST.

Các bước:

1. Khởi tạo đồ thị bằng danh sách kề.
2. Đưa đỉnh bắt đầu vào Heap với chi phí 0.
3. Mỗi vòng lặp:
    - Lấy đỉnh có chi phí nhỏ nhất ra khỏi Heap.
    - Nếu đỉnh chưa thuộc MST, thì chọn vào MST.
    - Cập nhật các đỉnh kề và đẩy vào Heap nếu tìm được chi phí nhỏ hơn.
4. Dừng khi tất cả n đỉnh đã được chọn.

Ưu điểm: Tiết kiệm bộ nhớ O(V + E), hiệu suất cao, phù hợp với đồ thị thưa.

Khuyết điểm: Phức tạp khi cài đặt, cần thao tác với Heap (đẩy, cập nhật trọng số, …)

Thiết kế thuật toán:

- Thư viện:
    + java.util.*: cung cấp các lớp hỗ trợ nhập xuất và cấu trúc dữ liệu cần thiết.
    + Scanner: đọc dữ liệu từ bàn phím.
    + ArrayList: tạo danh sách kề động.
    + PriorityQueue: cài đặt hàng đợi ưu tiên (Heap) để chọn cạnh có trọng số nhỏ nhất.
    + Comparator: giúp so sánh phần tử trong hàng đợi.
- Cấu trúc dữ liệu chính:
    + List<Canh>[] dsKe: danh sách kề, trong đó mỗi phần tử lưu danh sách cạnh dạng (u, v, w).
    + int[] dis: mảng khoảng cách, trong đó dis[v]: trọng số cạnh nhỏ nhất đang dùng để kết nối đỉnh v vào MST.
    + int[] parent: mảng cha, lưu đỉnh cha v trong cây khung tối thiểu, dùng để truy vết các cạnh của MST.
    + boolean[] used: mảng đánh dấu; used[v] = true nếu đỉnh v đã được đưa vào MST.
    + PriorityQueue<int[]> pq: Min-Heap chứa các phần tử dạng {dis[u], u} để chọn đỉnh gần MST nhất.
    + List<Canh> mst: lưu danh sách các cạnh thuộc cây khung tối thiểu.
- Hàm prim – Tìm cây khung nhỏ nhất (MST)

Thuật toán Prim bắt đầu chạy từ đỉnh nguồn s.

+ Khởi tạo:
  - Đặt mọi giá trị dis[i] = INF, parent[i] = -1.
  - Đỉnh bắt đầu s có dis[s] = 0.
  - Đưa (0, s) vào Min-Heap.
+ Vòng lặp chính: lặp cho đến khi hàng đợi rỗng
  - Lấy ra đỉnh u có dis nhỏ nhất từ Min-Heap.
  - Nếu u đã thuộc MST thì bỏ qua.
  - Đánh dấu used[u] = true.
  - Nếu parent[u] != -1, bổ sung cạnh (parent[u], u) vào MST.
  - Duyệt các đỉnh kề v của u:
    - Nếu v chưa thuộc MST và w < dis[v] thì cập nhật:
      - dis[v] = w
      - parent[v] = u
      - Đưa (dis[v], v) vào Heap.
+ Kết thúc:
  - Danh sách mst chứa toàn bộ các cạnh của cây khung tối thiểu.
  - Có thể tính tổng chi phí bằng cách cộng tất cả trọng số.

- Kết quả đầu ra: thuật toán trả về
  + Danh sách cạnh có trong cây khung tối thiểu (MST).
  + Tổng chi phí.
  + MST được dùng để trực quan hóa trong giao diện bằng lớp GraphPanel.

CHƯƠNG III. CƠ SỞ DỮ LIỆU

3.1. Công cụ và cấu trúc

3.1.1. Công cụ:

SQLITE + DB Browser(SQLITE)

3.1.2. Cấu trúc

```
CREATE TABLE IF NOT EXISTS DoThi(

ID INTEGER PRIMARY KEY AUTOINCREMENT,

Ten TEXT NOT NULL,

SoDinh INTEGER NOT NULL,

SoCanh INTEGER NOT NULL,

DinhBatDau INTEGER NOT NULL,
```

DanhSachCanh TEXT NOT NULL

);

3.2. Mô tả

CREATE TABLE IF NOT EXISTS DoThi: tạo bảng DoThi, không tạo lại nếu bảng đã có, tránh trùng tên

INTEGER PRIMARY KEY: khóa chính, mỗi giá trị phải duy nhất

AUTOINCREMENT: khi thêm đồ thị mới, giá trị id sẽ tự tăng lên

TEXT: kiểu dữ liệu chuỗi ký tự

NOT NULL: bắt buộc phải có giá trị, không được để trống

ID(PK) : khoá chính, tự tăng

Ten: tên đồ th, bắt buộc nhập

SoDinh: số đỉnh, bắt buộc nhập

SoCanh: số cạnh, bắt buộc nhập

DinhBatDau: đỉnh bắt đầu, bắt buộc nhập

DanhSachCanh: danh sách các cạnh, bắt buộc nhập.

Thao tác cơ bản

- XEM DỮ LIỆU

SELECT * FROM DoThi; (toàn bộ bảng)

SELECT * FROM DoThi WHERE ID = 1; (theo ID)

- CHÈN ĐỒ THỊ + CẠNH:

INSERT INTO DoThi (Ten, SoDinh, SoCanh, DinhBatDau, DanhSachCanh)

VALUES ('DT1', 3, 3, 1, '1 2 4\n1 3 2\n2 3 5');

- XÓA DỮ LIỆU

DELETE FROM DoThi WHERE ID = 1; (XÓA 1 ĐỒ THỊ)

DELETE FROM DoThi WHERE ID IN (1,2,….); (XÓA NHIỀU ĐỒ THỊ)

3.3. Kết nối(SQLite)

private static final String DB_URL = "jdbc:sqlite:finalG.db";

MỞ KẾT NỐI:

```
public Connection getConnection() throws SQLException {
   return DriverManager.getConnection(DB_URL);
}
```

KHỞI TẠO DATABASE:

```
public void khoiTaoDatabase() {
   String sql = """
      CREATE TABLE IF NOT EXISTS DoThi(
         ID INTEGER PRIMARY KEY AUTOINCREMENT,
         Ten TEXT NOT NULL,
         SoDinh INTEGER NOT NULL,
         SoCanh INTEGER NOT NULL,
         DinhBatDau INTEGER NOT NULL,
         DanhSachCanh TEXT NOT NULL
      );
      """;
   try (Connection conn = getConnection(); Statement st = conn.createStatement()) {
      st.execute(sql);
   } catch (SQLException e) {
      JOptionPane.showMessageDialog(null, "Lỗi khởi tạo DB: " + e.getMessage());
   }
}
```

LƯU ĐỒ THỊ VÀO DB : SQLite tự thêm một dòng mới trong bảng với ID+1

```
public void saveDoThiVaoDB(String ten, int n, int m, int s, String dsCanh) throws
SQLException {
   String sql = "INSERT INTO DoThi(Ten, SoDinh, SoCanh, DinhBatDau,
DanhSachCanh) VALUES (?, ?, ?, ?, ?)";
```

```
    try (Connection conn = getConnection();
        PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, ten);
        ps.setInt(2, n);
        ps.setInt(3, m);
        ps.setInt(4, s);
        ps.setString(5, dsCanh);
        ps.executeUpdate();
    }
}
```

TẢI ĐỒ THỊ THEO ID: kết nối mới được mở để load dữ liệu, trả về ResultSet cho ui xử lý(kết nối sẽ tự đóng khi ui đọc xong)

```
public ResultSet loadDoThiTuDB(int id) throws SQLException {
    String sql = "SELECT * FROM DoThi WHERE ID = ?";
    Connection conn = getConnection();
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setInt(1, id);
    return ps.executeQuery();
}
```

TRONG HÀM MAIN: java chạy hàm main, tạo object db, khởi tạo (nếu chưa có bảng thì tạo bảng, nếu có rồi thì bỏ qua), mở ui(nhập dữ liệu, chạy prim, lưu vào DB, load từ DB)

```
public static void main(String[] args) {
    db database = new db();
    database.khoiTaoDatabase();
    new ui().setVisible(true);
}
```

4.1 Giới thiệu tổng quan về Giao diện Người dùng

Giao diện người dùng (GUI) của ứng dụng được xây dựng trên nền tảng Java Swing và thư viện đồ họa AWT, nhằm cung cấp một công cụ trực quan hóa và tính toán cho thuật toán Prim.

Về mặt kiến trúc, dự án hiện đang triển khai theo mô hình thiết kế tích hợp (monolithic), nơi toàn bộ logic xử lý (giao diện, thuật toán Prim, và tương tác CSDL) đều được đóng gói và quản lý tập trung bên trong lớp chính prim (kế thừa từ javax.swing.JFrame). Lớp Database (trong database.java) hoạt động như một lớp tiện ích (utility class) cung cấp phương thức tĩnh để thiết lập kết nối JDBC. Sự tích hợp này giúp đơn giản hóa việc truy cập và cập nhật trạng thái dữ liệu giữa các thành phần giao diện.

4.2 Cấu trúc bố cục và Các thành phần chi tiết

Giao diện chính sử dụng BorderLayout làm layout manager cơ bản, phân chia màn hình thành hai khu vực chính: khu vực điều khiển (Tây) và khu vực hiển thị đồ họa (Trung tâm).

4.2.1 Khu vực Điều khiển và Nhập liệu (Panel Tây - WEST)

Khu vực này, được đặt trong một JPanel container, chứa toàn bộ các thành phần nhập liệu và điều khiển.

- Nhập tham số và Dữ liệu:
    - txtSoDinh (JTextField): Nhập số lượng đỉnh (N) của đồ thị.
    - txtDinhBatDau (JTextField): Xác định đỉnh gốc (S) để khởi chạy thuật toán.
    - txtDanhSachCanh (JTextArea): Khu vực nhập liệu chính cho danh sách các cạnh, theo định dạng văn bản u v w (đỉnh nguồn, đỉnh đích, trọng số) trên mỗi dòng. Thành phần này được lồng trong JScrollPane để hỗ trợ nhập liệu khối lượng lớn.
- Khu vực Log và Kết quả (txtKetQua):
    - Khác với bố cục ở vùng SOUTH, thành phần txtKetQua (JTextArea ở chế độ chỉ đọc) được đặt trong khu vực bên trái này.
    - Đây là nơi hiển thị danh sách các cạnh thuộc Cây khung tối thiểu (MST) và tổng chi phí cuối cùng sau khi thuật toán hoàn tất.
- Hệ thống nút chức năng:
    - btnChayPrim: Kích hoạt việc thực thi thuật toán.
    - btnLuu: Kích hoạt chức năng lưu trữ trạng thái đồ thị hiện tại vào CSDL.
    - btnTai: Kích hoạt chức năng tải dữ liệu đồ thị từ CSDL theo ID bản ghi do người dùng nhập qua hộp thoại (JOptionPane).

4.2.2 Khu vực Trực quan hóa Đồ họa (Panel Trung tâm - CENTER)

Khu vực hiển thị đồ họa được quản lý bởi lớp nội bộ GraphPanel (kế thừa từ JPanel), đây là nơi xử lý toàn bộ các tác vụ vẽ hình học 2D.

- Cơ chế đồ họa: Sử dụng đối tượng Graphics2D với RenderingHints để kích hoạt khử răng cưa (anti-aliasing), đảm bảo chất lượng hình ảnh sắc nét.
- Thuật toán bố trí đỉnh (Circular Layout):
    - Vị trí của các đỉnh được tính toán trong phương thức setGraph(...) để sắp xếp đều trên một vòng tròn (Circular Layout).
    - Bán kính vòng tròn (Radius) được tính toán dựa trên kích thước tối thiểu của panel, với ràng buộc cụ thể: $\text{radius} = \max\left(\frac{\min(\text{width},\text{height})}{2} - 60{,}150\right)$, nhằm tránh đỉnh bị che khuất.
- Biểu diễn kết quả MST:
    - Màu sắc: Sau khi thuật toán hoàn tất, GraphPanel sẽ vẽ lại đồ thị. Các cạnh thuộc Cây khung tối thiểu được tô bằng màu đỏ (Color.RED) và nét vẽ dày hơn để làm nổi bật.
    - Minh họa: Mã nguồn hiện tại chỉ thực hiện vẽ kết quả cuối cùng của MST; không có cơ chế animation hoặc cập nhật tiến độ (step-by-step) theo thời gian thực để minh họa từng bước chọn cạnh.

4.3 Logic Xử lý Sự kiện và Vận hành Thuật toán

Do kiến trúc tích hợp, toàn bộ logic nghiệp vụ được gắn trực tiếp vào các sự kiện hành động của giao diện.

4.3.1 Luồng thực thi Thuật toán (Action Listener)

Khi người dùng nhấn nút btnChayPrim, chuỗi xử lý sau được thực hiện trên Luồng Xử lý Sự kiện (Event Dispatch Thread - EDT):

1. Phân tích Dữ liệu: Lớp prim gọi các hàm nội bộ để đọc dữ liệu từ txtDanhSachCanh bằng Scanner. Dữ liệu u v w được parse thành số nguyên.
2. Kiểm tra Ràng buộc: Hệ thống kiểm tra các ràng buộc về số đỉnh (dương) và ID đỉnh (nằm trong phạm vi 1 đến N). Các lỗi định dạng số (NumberFormatException) được bắt và thông báo qua JOptionPane. (Lưu ý: Mã nguồn hiện tại không có kiểm tra tường minh về trọng số âm).
3. Vận hành Prim: Phương thức chứa logic thuật toán Prim (tập trung tại lớp prim) được gọi. Quá trình tính toán diễn ra tuần tự và có thể block (làm đóng băng) giao diện người dùng nếu đồ thị có kích thước lớn.
4. Hiển thị Kết quả:
    - Kết quả văn bản được cập nhật ngay lập tức vào txtKetQua.

- Dữ liệu đồ thị mới được chuyển tới GraphPanel và repaint() được gọi để vẽ lại đồ thị với các cạnh MST được tô màu đỏ.

4.3.2 Cơ chế Tương tác Cơ sở Dữ liệu (SQLite)

Tương tác với CSDL SQLite được xử lý trực tiếp trong lớp prim thông qua JDBC.

- Kết nối: Lớp prim chứa một helper method nội bộ (getConnection()) sử dụng URL tĩnh để kết nối đến file finalG.db. Phương thức tĩnh Database.getConnection() trong database.java cũng có thể được sử dụng như một tiện ích kết nối.
- Lưu Dữ liệu: Phương thức saveDoThiVaoDB() thu thập trạng thái các trường nhập liệu và thực hiện truy vấn INSERT vào bảng CSDL.
- Tải Dữ liệu: Phương thức loadDoThiTuDB() kích hoạt hộp thoại JOptionPane để yêu cầu người dùng nhập ID của bản ghi cần khôi phục. Sau đó, nó thực hiện truy vấn SELECT để lấy dữ liệu, khôi phục lại các trường nhập liệu và vẽ lại đồ thị.

4.4 Khởi chạy Ứng dụng

Điểm khởi chạy chính của giao diện được đặt tại phương thức main của lớp prim. Lớp main.java hiện tại đóng vai trò kiểm tra tính hợp lệ của kết nối CSDL và các tác vụ khởi động môi trường đơn giản. Khi ứng dụng chạy, lớp prim sẽ thiết lập cửa sổ JFrame và hiển thị giao diện để bắt đầu tương tác.

CHƯƠNG V. KẾT LUẬN

Trong quá trình giải quyết bài toán thiết kế mạng lưới tối ưu, việc xây dựng cây khung tối thiểu là bước quan trọng và cốt lõi. Báo cáo đã triển khai thành công thuật toán Prim qua phương pháp kết hợp danh sách kề và hang đợi ưu tiên bằng ngôn ngữ lập trình java.

Các kết quả chính có thể tổng kết như sau:

- Thuật toán Prim chứng minh được hiệu quả khi xử lý đồ thị liên thông, đặc biệt là các mạng có phân bố trọng số thực tế.
- Danh sách kề kết hợp heap (min-priority queue) vượt trội về thời gian thực thi và khả năng mở rộng, giúp xử lý mạng lưới lớn với hàng nghìn đến hàng chục nghìn đỉnh và cạnh.

Dưới góc nhìn kỹ thuật, Prim hoạt động ổn định, dễ điều chỉnh để tích hợp thêm ràng buộc thực tế như:

- Giới hạn chi phí kết nối.
- Loại trừ một số liên kết nhất định.
- Ưu tiên những tuyến kết nối đặc biệt.

TÀI LIỆU THAM KHẢO

1. *Applications of Minimum Spanning Trees (MST) in Network Design*, heycoach, [Truy cập ngày 6/7/2025]:

https://blog.heycoach.in/applications-of-mst-in-network-design/

2. Vũ Quế Lâm, *Giới thiệu về lý thuyết đồ thị*, VIBLO, [đã đăng vào ngày 24/1/2022], [truy cập ngày 7/6/2025]:

https://viblo.asia/p/gioi-thieu-ve-ly-thuyet-do-thi-07LKXQ1eZV4

3. *Cây bao trùm*, wikipedia, [cập nhật ngày 21/7/2024], [truy cập ngày 7/6/2025]

https://vi.wikipedia.org/wiki/C%C3%A2y_bao_tr%C3%B9m

4. *Cây bao trùm nhỏ nhất*, wikipedia, [cập nhật ngày 17/5/2024], [truy cập ngày 7/6/2025]

https://vi.wikipedia.org/wiki/C%C3%A2y_bao_tr%C3%B9m_nh%E1%BB%8F_nh%E1%BA%A5t

5. *Minimum spanning tree – Prim's algorithm*, Algorithms for Competitive Programming, [cập nhật ngày 22/4/2025], [truy cập ngày 7/6/225]

https://cp-algorithms.com/graph/mst_prim.html

6. *Priority queue*, wikipedia, [cập nhật ngày 25/4/2025], [truy cập ngày 7/6/2025]

https://en.wikipedia.org/wiki/Priority_queue