GRAPH ALGORITHM

GROUP 8

MỤC LỤC

- Giới thiệu sơ lược
- Các dạng đồ thị
- Cách biểu diễn đồ thị
- Duyệt đồ thị
- Một số bài toán thông thường
- Thuật toán Dijkstra
- Thuật toán Prim

BRIEF INTRODUCTION

GRAPHS EVERYWHERE

Lý thuyết đồ thị (tiếng Anh: graph theory) nghiên cứu các tính chất của đồ thị

Đồ thị có thể là bất cứ thứ gì trong cuộc sống và có thể áp dụng lý thuyết đồ thị vào cuộc sống

GRAPHS EVERYWHERE

Có thể là bản đồ

GRAPHS EVERYWHERE

Mạng xã hội

NGUỒN GỐC

Lý Thuyết đồ thị có lịch sử bắt nguồn từ bài toán Seven Bridges of Königsberg

SEVEN BRIDGES OF KÖNIGSBERG

Thành phố Königsberg, Phổ (nay là Kaliningrad, Nga) nằm trên sông Pregel, bao gồm hai hòn đảo lớn nối với nhau và với đất liền bởi bảy cây cầu. Bài toán đặt ra là tìm một tuyến đường mà đi qua mỗi cây cầu một lần và chỉ đúng một lần (bất kể điểm xuất phát hay điểm tới)

SEVEN BRIDGES OF KÖNIGSBERG

Và dĩ nhiên không có cách nào mà có đi hết 7 cây cầu mà chỉ đi qua chúng 1 lần

KÖNIGSBERG HIỆN GIỜ

SAU THẾ CHIẾN THỨ 2 THÀNH PHỐ BỊ PHÁ HỦY HOÀN TOÀN. BÂY GIỜ PHONG CÁCH KIẾN TRÚC MANG ĐẬM THỜI KỲ XÔ VIẾT

MÔ TẢ SƠ LƯỢC VỀ ĐỒ THỊ

MÔ TẢ SƠ LƯỢC VỀ ĐỒ THỊ

Đồ thị là một tập các đối tượng các đỉnh (hay nút) nối với nhau bởi các cạnh

Các nút có thể đại diện cho:

- Con người
- Điểm

Các cạnh có thể đại diện cho:

- Khoảng cách giữa các thành phố
- Mối quan hệ con người

CÁC DẠNG ĐỒ THỊ

UNDIRECTED GRAPH

DIRECTED GRAPH

WEIGHTED GRAPH

MỘT SỐ DẠNG ĐỒ THỊ KHÁC

TREE

ROOTED TREE

DIRECTED ACYCLIC GRAPH

BIPARTITE GRAPH

COMPLETE GRAPHS

K_1

K_2

K_3

K_4

K_5

REPRESENTING GRAPHS

- Adjacency Matrix
- Adjacency List
- Edge List

ADJACENCY MATRIX

Ý tưởng: với mỗi phần tử m[i][j] sẽ đại diện cho tổn phí đi từ điểm I đến điểm j

	A	B	C	D
A	0	4	1	9
B	3	0	6	11
C	4	1	0	2
D	6	5	-4	0

PROS AND CONS

- Ưu điểm:

Tiết kiệm không gian bộ nhớ nếu biểu diễn đồ thị dày

Để kiểm tra 2 đỉnh u, v liền kề hay không, ta chỉ cần kiểm tra độ phức tạp O(1)

- Khuyết điểm:

Yêu cầu O(V^2) bộ nhớ

Tốn O(V^2) thời gian để duyệt tất cả các cạnh

VÍ DỤ:

Hãy lập 1 adjacency matrix đối với hình bên cạnh

ADJACENCY LIST

Với mỗi đỉnh của đồ thị, ta lưu một danh sách các đỉnh kề với đỉnh đó.

```
A -> [(B,4),(C,1)]
B -> [(C,6)]
C -> [(A,4),(B,1),(D,2)]
D -> []
```

PROS AND CONS

- Ưu điểm:

Với đồ thị thưa, sử dụng ô nhớ hiệu quả để lưu đồ thị

Duyệt tất cả các cạnh khá là hiệu quả

- Khuyết điểm:

Kém hiệu quả đối với các đồ thị dày

Tốn O(E) thời gian để tìm kiếm cạnh

Edge List

```
[(C,A,4),  (A,C,1),
 (B,C,6),  (A,B,4),
 (C,B,1),  (C,D,2)]
```

PROS AND CONS

- Ưu điểm:

Với đồ thị thưa, ta chỉ cần mất m (số lượng cạnh) ô nhớ để lưu đồ thị
Duyệt tất cả các cạnh khá là hiệu quả
Cấu trúc đơn giản

- Khuyết điểm:

Kém hiệu quả đối với các đồ thị dày
Tốn O(E) thời gian để tìm kiếm cạnh

VÍ DỤ:

Hãy lập 1 edge list đối với hình bên cạnh

MỘT SỐ BÀI TOÁN THÔNG THƯỜNG

SHORTEST PATH FINDING

Tìm quảng đường giữa 2 điểm (hay 2 cạnh) trong đồ thị mà tổng tiêu thao của các thành phần là nhỏ nhất

Algorithm: BFS(Unweighted Graph), Dijkstra's, A*,etc..

SHORTEST PATH FINDING

Tìm quảng đường giữa 2 điểm (hay 2 cạnh) trong đồ thị mà tổng tiêu thao của các thành phần là nhỏ nhất

Algorithm: BFS(Unweighted Graph), Dijkstra's, A*,etc..

STRONGLY CONNECTED COMPONENT

Một đồ thị có hướng là liên thông mạnh nếu như có đường từ bất kì đỉnh nào tới bất kì đỉnh nào khác.

Một **thành phần liên thông mạnh** của một đồ thị có hướng là một đồ thị con tối đại liên thông mạnh.

Algorithm: Tarjan’s và Kosaraju’s

STRONGLY CONNECTED COMPONENTS

STRONGLY CONNECTED COMPONENTS

BRIDGES

Cạnh cầu là cạnh khi loại bỏ đi sẽ làm tang số lượng đồ thị liên thông (connected components)

BRIDGES

Cạnh cầu là 1 yếu tố quan trọng của lý thuyết đồ thị nó chỉ ra những điểm yếu hay nút thoắt cổ trai trong đồ thị

ARTICULATION POINT

Đỉnh chóp là đỉnh khi loại bỏ đi sẽ làm tang số lượng đồ thị liên thông (connected components)

ARTICULATION POINT

Đỉnh chóp là yếu tố quan trọng của lý thuyết đồ thị nó chỉ ra những điểm yếu hay nút thoắt cổ trai trong đồ thị

NEGATIVE CYCLE

Tìm xem coi đồ thị có Negative Cycle

Negative Cycle là 1 vòng lặp mà giá trị tổng của các cạnh thành phần là âm

Algorithm: Bellman Ford

Những node trong negative cycle

Những node bình thường

NEGATIVE CYCLE

Tìm xem coi đồ thị có Negative Cycle

Negative Cycle là 1 vòng lặp mà giá trị tổng của các cạnh thành phần là âm

Algorithm: Bellman Ford

Những node trong negative cycle

Những node bình thường

YOUR TITLE GOES HERE

SUBTITLE GOES HERE

TRAVELING SALESMAN PROBLEM

Cho trước một danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất thăm mỗi thành phố đúng một lần.

Algorithm: Branch and bound ,etc

MINIMUM SPANNING TREE

Algorithm : Kruskal's , Prim's , Borůvka's algorithm

MINIMUM SPANNING TREE

Algorithm : Kruskal's , Prim's , Borůvka's algorithm

MINIMUM SPANNING TREE

Với một đồ thị liên thông, vô hướng cho trước, cây bao trùm của nó là một đồ thị con có dạng cây và có tất cả các đỉnh liên thông với nhau. Một đồ thị có thể có nhiều cây bao phủ khác nhau. Chúng ta cũng có thể gán một *trọng số* cho mỗi cạnh, là con số biểu thị sự "không ưa thích" và dùng nó để tính toán trọng số của một cây bao trùm bằng cách cộng tất cả trọng số của cạnh trong cây bao trùm đó. Khi đó, một **cây bao trùm nhỏ nhất** là một cây bao trùm có trọng số bé hơn bằng trọng số của tất cả các cây bao trùm khác. Tổng quát hơn, bất kỳ một đồ thị vô hướng (không nhất thiết liên thông) đều có một **rừng bao phủ nhỏ nhất**, là hội của các cây bao trùm nhỏ nhất của các thành phần liên thông của nó

NETWORK FLOW: MAXIMUM FLOW

Algorithm : Ford-Fulkerson , Elmonds-Karp & Dinic ‘s algorithm

NETWORK FLOW : MAXIMUM FLOW

Hãy tưởng tượng đồ thị trên như những con đường đầy xe hay những dãy hành lang đông người đang di chuyển. Những trọng số đại diện cho số lượng xe tối đa giao thông ở con đường đó có thể chịu được , số lượng người tối đa có thể băng qua dãy thành lang.

- Câu hỏi đặt ra là với lượng tài nguyên không giới hạn thì dòng chảy tối đa chảy qua đồ thị là bao nhiêu

GRAPH TRAVERSAL

- Depth-first search
- Breadth-first search

DEPTH-FIRST SEARCH

Thuật toán:

1. Thăm đỉnh xuất phát, đỉnh u, đánh dấu đỉnh u.
2. Xét các đỉnh v kề với đỉnh hiện đang thăm:
   - Nếu đỉnh v chưa được đánh dấu (chưa thăm), thăm đỉnh v.
   - Nếu đỉnh v đã được đánh đấu, bỏ qua.

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

Backtrack lại khi đã đến điểm cuối

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

APPLICATION

- Detecting cycle
- Path finding
- Finding strongly connected components
- Etc..

DEPTH-FIRST SEARCH SOURCE CODE

```
n = số node trong graph
g = graph được biểu diễn bằng danh sách kề
visited = [False,...,False] # size n

function dfs(at):
    if visited[at]: return
    visited[at] = true

    neighbours = g[at]
    for next in  neighbour:
        dfs(next)
```

count = 0

```
function findComponents():
    for (i = 0; i < n; i++):
        if !visited[i]:
            count++
            dfs(i)
    return count
```

BREADTH-FIRST SEARCH

Thuật toán:

1.Thêm đỉnh gốc vào queue và đánh dấu đỉnh gốc.

2.Nếu queue chưa rỗng, lấy ra đỉnh u đầu tiên khỏi queue. Xét các đỉnh v kề với đỉnh u

Nếu đỉnh v đã được đánh dấu thì bỏ qua.

Nếu v chưa được đánh dấu thì thêm đỉnh v vào queue và đánh dấu đỉnh v.

3. Nếu queue rỗng, dừng quá dình tìm kiếm.

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

BREADTH-FIRST SEARCH

4
2
5
12
1
3
6
8
10
11
7
9
0

BREADTH-FIRST SEARCH

4
2
5
12
1
3
6
8
10
11
7
9
0

APPLICATION

- Shortest Path and Minimum Spanning Tree for unweighted graph
- Peer to peer network. Ex: Bit Torent
- Finding all nodes within one connected component
- Etc…

BREADTH-FIRST SEARCH SOURCE CODE

n = số node của graph
g = danh sách kề biểu diễn graph

```
# s = start node, e = end node
function bfs(s,e):
    # thực hiện BFS
    prev = sovle(s)

    # return quãng đường s->e
    return reconstructPath(s, e, prev)
```

```
function solve(s):
    q = queue()
    q .enqueue(s)

    visited = [False,...,False] # size n
    visited[s] = True

    prev = [null,...,null] # size n
    while !q.isEmty():
        node = q.dequeue()
        neighbours = g[node]

        for(next : neighbours):
            if !visited[next]:
                q.enqueus(next)
                visited[next] = True
                prev[next] = node
    return prev
```

```
function reconstructPath(s, e, prev):
    path = []

    for(at = e; at != null; at = prev[at]):
        path.add(at)

    path.reverse()

    if path[0] == s:
            return path
    return []
```

Depth-first search

- **Time complexity:** O(V + E), trong đó V là số đỉnh và E là số đỉnh trong đồ thị
- **Space Complexity:** O(V).

Thường được sử dụng để tìm connected component , xác định connectivity, tìm điểm bridges và articulation

Breadth-first search

- **Time complexity:** O(V + E), trong đó V là số đỉnh và E là số đỉnh trong đồ thị
- **Space Complexity:** O(V).

Thường được sử dụng để tìm quãng đường ngắn nhất

DIJKSTRA'S ALGORITHM FOR SHORTEST PATH PROBLEM

GIỚI THIỆU

Dijkstra’s Algorithm do nhà khoa học máy tính người Hà Lan ‘ Edsger Wybe Dijkstra ‘ đưa ra vào năm 1956 và xuất bản năm 1959.

HOW IT WORKS ?

Thuật toán tìm ra đường đi với chi phí thấp nhất (đường đi ngắn nhất) giữa đỉnh nguồn với các đỉnh còn lại

Ý TƯỞNG

1. Tạo ra 2 danh sách. Một danh sách visited [ ] chứa các đỉnh đã được viếng thăm, và một danh sách unvisited [ ] chứa các đỉnh chưa được viếng thăm. Ban đầu danh sách visited rỗng.
2. Gán giá trị cho tất cả các đỉnh trong đồ thị. Khởi tạo các giá trị khoảng cách dưới dạng infinity(∞). Đối với đỉnh nguồn gán giá trị là 0.
3. Chọn một đỉnh u trong unvisited có giá trị khoảng cách ngắn nhất, đưa nó vào visited. Cập nhật lại giá trị khoảng cách của tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu tổng giá trị khoảng cách của u (nguồn) và trọng số của cạnh uv nhỏ hơn giá trị khoảng cách của v, thì cập nhật lại giá trị khoảng cách của v.

DIJSKTRA SHORTEST PATH ALGORITHM

Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ] Unvisited = [A, B, C, D, E]

Consider the start vertex, A

Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ] Unvisited = [A, B, C, D, E]

Consider the start vertex, A
Distance from A to A = 0
Distances to all other vertices from A are unknow, therefor infinity

Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ] Unvisited = [A, B, C, D, E]

For the current vertex, examine unvisited neighbours
We are currently visiting A and its unvisited neighbours are B and D

Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ] Unvisited = [A, B, C, D, E]

For the current vertex, calculate the distance of each neighbour from start vertex

Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ] Unvisited = [A, B, C, D, E]

If the calculated distance of a vertex is less than the know distance, update the shortest distance

Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	
C	∞	
D	1	
E	∞	

Visited = [ ] Unvisited = [A, B, C, D, E]

Update the previous vertex for each of the updated distance
In this case we visited B and D via A

Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [ ] Unvisited = [A, B, C, D, E]

Add the current vertex to the list of visited vertices

Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A] Unvisited = [B, C, D, E]

Visit the unvisited vertex with the smallest known distance from the start vertex
This time around, it is vertex D

Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A] Unvisited = [B, C, D, E]

For the current vertex, examine its unvisited neighbour
We are currently visiting D and its unvisited neighbour are B and E

Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A] Unvisited = [B, C, D, E]

For the current vertex, calculate the distance of each neighbour from start vertex

Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A] Unvisited = [B, C, D, E]

If the calculated distance of a vertex is less than the know distance, update the shortest distance

Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A] Unvisited = [B, C, D, E]

If the calculated distance of a vertex is less than the know distance, update the shortest distance

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	A
C	∞	
D	1	A
E	2	

Visited = [A] Unvisited = [B, C, D, E]

Update the previous vertex for each of the updated distance
In this case we visited B and E via D

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A] Unvisited = [B, C, D, E]

Add the current vertex to the list of visited vertices

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D] Unvisited = [B, C, E]

Visit the unvisited vertex with the smallest known distance from the start vertex
This time around, it is vertex D

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D] Unvisited = [B, C, E]

For the current vertex, examine its unvisited neighbour
We are currently visiting E and its unvisited neighbour are B and C

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D] Unvisited = [B, C, E]

For the current vertex, calculate the distance of each neighbour from start vertex

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D] Unvisited = [B, C, E]

If the calculated distance of a vertex is less than the know distance, update the shortest distance
We do not need update the distance to B

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	
D	1	A
E	2	D

Visited = [A, D] Unvisited = [B, C, E]

Update the previous vertex for each of the updated distance
In this case we visited C via E

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D] Unvisited = [B, C, E]

Add the current vertex to the list of visited vertices

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

Visit the unvisited vertex with the smallest known distance from the start vertex
This time around, it is vertex B

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

For the current vertex, examine its unvisited neighbour
We are currently visiting E and its only unvisited neighbour is C

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

For the current vertex, calculate the distance of each neighbour from start vertex

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

If the calculated distance of a vertex is less than the know distance, update the shortest distance
We do not need update the distance to C

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

Update the previous vertex for each of the updated distances
No distance were updated, so we don't need to do this either

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

Add the current vertex to the list of visited vertices

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

Visit the unvisited vertex with the smallest known distance from the start vertex
This time around, it is vertex C

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

For the current vertex, examine its unvisited neighbour
We are currently visiting C and it has no unvisited neighbour

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

Add the current vertex to the list of visited vertices

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B, C] Unvisited = [ ]

n = số node trong graph
g = graph được biểu diễn bằng danh sách kề

```
function dijkstra(g, n, s):
    visited = [False,…,False] # size n
    dist = [∞, … , ∞] # size n
    dist[s] = 0

    for (i = 0 ;i < n; i++):
        u = minDist(dist, visited)
        visited[u] = True
            for (edge : g[u]):
                if visited[edge]: continue
                        newDist = dist[u] + edge.cost
                  if newDist < dist[edge.to]:
                      dist[edge.to] = newDist
    return dist
```

```
function dijkstra(g, n, s):
  visited = [False,…,False] # size n
  dist = [∞, … , ∞] # size n
  dist[s] = 0
  prev = [null,…,null] # size n
  for (i = 0 ;i < n; i++):
    u = minDist(dist, visited)
    visited[u] = True
      for (edge : g[u]):
        if visited[edge]: continue
          newDist = dist[u] + edge.cost
          if newDist < dist[edge.to]:
            dist[edge.to] = newDist
            prev[edge.to] = u
  return dist, prev
```

```
function findShortestPath(g, n, s, e):
    dist, prev = dijkstra(g, n, s)
    path = [ ]
    if dist[e] = ∞ : return path
    for (at = e; at != null; at = prev[at])
        path.add[at]
    path.reverse()
return path
```

APPLICATION:

- Sử dụng trong hệ thống thông tin giao thông
- Map (Map Quest)
- Routing Systems

DIJKSTRA ALGORITHMS

- Dijkstra là thuật toán tìm đường đi ngắn nhất cho một nguồn trong lý thuyết đồ thị.
- Hoạt động trên cả đồ thị có hướng và vô hướng
- Cách tiếp cận: Greedy
- Input: Đồ thị có trọng số G = { E, V } và đỉnh nguồn, sao cho các trọng số của các cạnh không âm
- Output: Đường đi ngắn nhất từ đỉnh nguồn đã cho.

PRIM'S ALGORITHM FOR MINIMUM SPAN TREE

ĐỊNH NGHĨA

1. Spaning Tree: cây bao trùm hay được gọi là cây khung của đồ thị G là cây con của đồ thị G, chứa tất cả các đỉnh của G, liên thông và không có chu trình

2. Minimum Spaning Tree(MST): là cây khung có tổng trọng số của các cạnh là nhỏ nhất

. Một đồ thị có thể có nhiều hơn một minimum spaning tree

VÍ DỤ

Spaning Tree

Cost = 26

Minimum Spaning Tree

Cost = 12

CÁC BƯỚC TÌM MST BẰNG PRIM'S ALGORITHM

1. Tạo ra danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [ ]

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [ ]
Source: 0

Vertex	0	1	2	3	4	5	6	7	8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0]
Source: 0

Vertex	0	1	2	3	4	5	6	7	8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0]
Source: 0

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	∞	∞	∞	∞	∞	∞	∞	∞

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0]

If w(u,v) < v.key
v.key = w(u, v)

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	∞	∞	∞	∞	∞	8	∞

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	∞	∞	∞	∞	∞	8	∞

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	8	∞	∞	∞	∞	8	∞

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	8	∞	∞	∞	∞	8	∞

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	8	∞	∞	∞	1	8	7

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	8	∞	∞	∞	1	8	7

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	8	∞	∞	2	1	8	6

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	8	∞	∞	2	1	8	6

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	14	10	2	1	8	6

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	14	10	2	1	8	6

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	7	10	2	1	8	2

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2, 8]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	7	10	2	1	8	2

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2, 8, 3]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	7	10	2	1	8	2

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2, 8, 3]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	7	9	2	1	8	2

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2, 8, 3, 4]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	7	10	2	1	8	2

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2, 8, 3, 4]

If w(u,v) < v.key
v.key = w(u, v)

u

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	7	9	2	1	8	2

1. Tạo ra hai danh sách, một danh sách MST chứa các đỉnh của Graph, danh sách mstSet để theo dõi các đỉnh trong MST
2. Gán các giá trị khoá cho tất cả các đỉnh trong đồ thị đầu vào. Khởi tạo tất cả các giá trị dưới dạng infinity (∞). Gán giá trị khoá là 0 cho đỉnh đầu tiên cho nó được chọn trước.
3. Trong mstSet:
   - Chọn một đỉnh u không có trong mstSet có giá trị khoá nhỏ nhất.
   - Đưa u vào mstSet
   - Cập nhật giá trị khoá của tất cả các đỉnh liền kề u. Lặp lại qua tất cả các đỉnh liền kề của u. Đối với mỗi đỉnh liền kề v, nếu trọng số của cạnh uv nhỏ hơn giá trị khoá trước đó của v, thì cập nhật giá trị khoá dưới dạng trọng số của uv.

EXAMPLE

mstSet = [0, 1, 7, 6, 5, 2, 8, 3, 4]

Vertex	0	1	2	3	4	5	6	7	8
Key	0	4	4	7	9	2	1	8	2

Time Complexity

- Thuật toán Dijkstra có độ phức tạp nếu đồ thị biểu diễn bằng ma trận kề: $O(V^2)$
- Nếu biểu diễn bằng danh sách kề: $O(E \log V)$

APPLICATION

- Design of network
- Computer network
- Transportation networks
- Water supply networks

PRIM'S ALGORITHMS

- Prim là thuật toán tìm cây bao trùm nhỏ nhất của một đồ thị vô hướng có trọng số liên thông
- Cách tiếp cận: Greedy
- Input: Đồ thị có trọng số G = { E, V } và đỉnh nguồn với E là cạnh và V là đỉnh
- Output: Khung cây bao trùm nhỏ nhất.

CODE ANALYSIS PRIM'S ALGORITHM FOR MINIMUM SPAN TREE

```
int minKey(int key[], bool mstSet[])
{
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
      if (mstSet[v] == false && key[v] < min)
         min = key[v], min_index = v;
   return min_index;
}
void printMST(int parent[], int graph[V][V])
{
   cout<<"Edge \tWeight\n";
   for (int i = 1; i < V; i++)
      cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";
}
int main()
{
   int graph[V][V] = { { 0, 2, 0, 6, 0 },
                       { 2, 0, 3, 8, 5 },
                       { 0, 3, 0, 0, 7 },
                       { 6, 8, 0, 0, 9 },
                       { 0, 5, 7, 9, 0 } };
   primMST(graph);
   return 0;
}
```

```
void primMST(int graph[V][V])
{
   int parent[V];
   int key[V];
   bool mstSet[V];
   for (int i = 0; i < V; i++)
      key[i] = INT_MAX, mstSet[i] = false;
   key[0] = 0;
   parent[0] = -1;
   for (int count = 0; count < V - 1; count++)
   {
      int u = minKey(key, mstSet);
      mstSet[u] = true;
      for (int v = 0; v < V; v++)
         if (graph[u][v] && mstSet[v] == false &&
graph[u][v] < key[v])
            parent[v] = u, key[v] = graph[u][v];
   }
   printMST(parent, graph);
}
```

BTVN:

Bạn đang mắc kẹt trong 1 mê cung và cần tìm ra lối thoát nhanh nhất. Mê cung có thể chứa đầy chướng ngại vật hoặc không. di chuyển về trái, phải, lên, xuống. Bạn không thể di chuyển theo đường chéo và mê cung được bao quanh bởi chướng ngại vật ở mọi phía.

Có thể trốn thoát được không? Nếu có thì mất bao lâu?

Input: R C (R,C <100) là số hàng và số cột tạo nên mê cung

Chướng ngại vật sẽ được biểu thị bằng dấu # và các ô trống được biểu thị bằng dấu '.'

Ví trí xuất phát sẽ được biểu thị bằng 's' và đích biểu thị bằng 'e'

Đầu vào kết thúc bằng 2 số 0 cho R và C

Output: Nếu có đường ra thì in ra bước ít nhất

Nếu không in ra -1

Input:
3 3
s.#
.##
#.e
0 0

Output:
-1

s	.	#
.	#	#
#	.	e

Input:
4 5
S .###
#.#.#
#..e.
.#.##
0 0

Output:
5

s →	.	#	#	#
#	↓ .	#	.	#
#	↓ . →	. →	e	.
.	#	.	#	#

Những thắc mắc và câu hỏi

GỬI BÀI TẬP QUA MAIL SAU:
19521512@gm.uit.edu.vn

THAM KHẢO

[1] https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

[2] https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[3] https://www.youtube.com/watch?v=09_LlHjoEiY&t=2029s