Minimum Cost Spanning Trees

Advanced Data Structures & Algorithms

1

Outline

1. **Disjoint Data Structure**
2. Minimum Cost Spanning Tree
    - Kruskal Algorithm
    - Prim's Algorithm
3. Shortest Path Algorithm
    - Floyd Warshal Algorithm

2

Disjoint Set Data Structures

- Two sets are called disjoint sets if they don't have any element in common, the intersection of sets is a null set.

$$\boldsymbol{S_1 = \{a, b, c\}}$$
$$\boldsymbol{S_2 = \{d, e\}}$$
$$\boldsymbol{S_1 \cap S_2 = \phi}$$

- The disjoint set data structure supports following operations:
  - Adding new sets to the disjoint set
  - Merging disjoint sets to a single disjoint set using **union** operation.
  - Finding representative of a disjoint set using **find** operation.
  - Check if two sets are disjoint or not.

3

Disjoint Set Data Structures

- $MAKE - SET(x)$ creates a new set whose only member (and thus representative) is x.
- $UNION(x, y)$ unites the dynamic sets that contain x and y, say S_x and S_y into a new set that is the union of these two sets. It is represented as $S_x \cup S_y$.
- $FIND - SET(x)$ returns a pointer to the representative of the (unique) set containing x.

4

$MAKE - SET(x)$

```
MAKE-SET(x)
1  $x.p = x$
2  $x.rank = 0$
```

5

$UNION(x)$

```
UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if $x.rank > y.rank$
2      $y.p = x$
3  else $x.p = y$
4      if $x.rank == y.rank$
5          $y.rank = y.rank + 1$
```

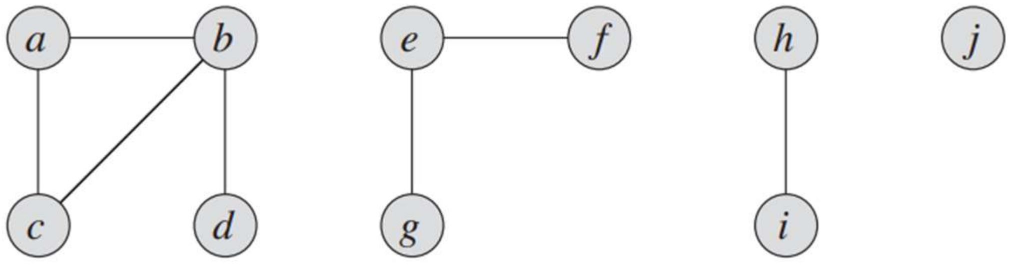
6

$FIND-SET(x)$

```
FIND-SET($x$)
1  if $x \neq x.p$
2      $x.p$ = FIND-SET($x.p$)
3  return $x.p$
```

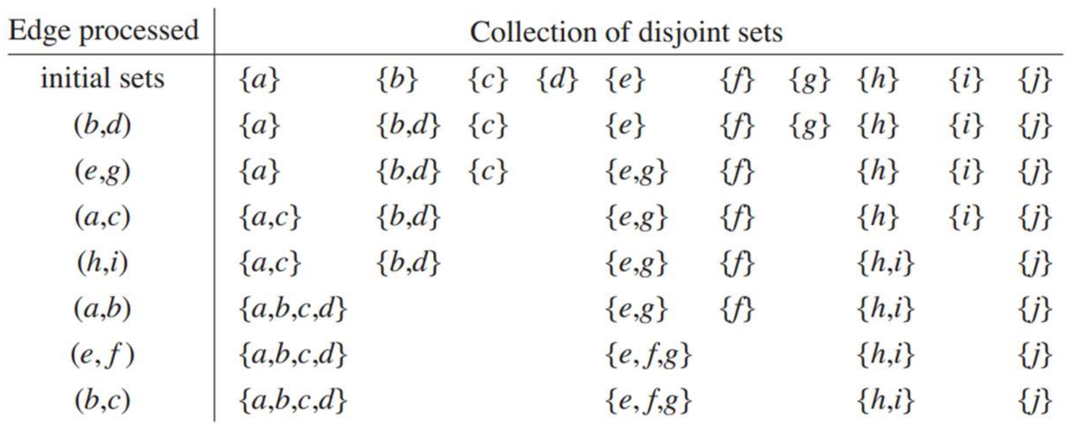
7

Disjoint Set



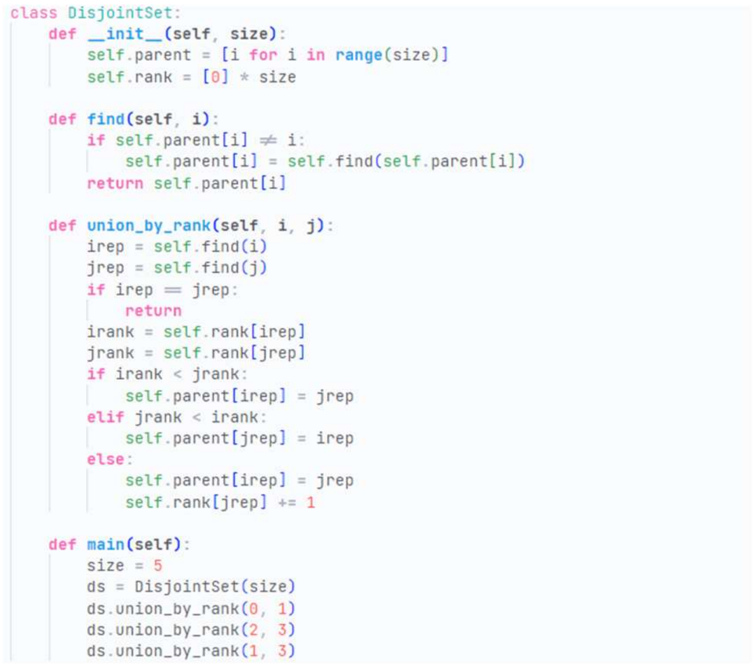
8

9



Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

10



```python
class DisjointSet:
    def __init__(self, size):
        self.parent = [i for i in range(size)]
        self.rank = [0] * size

    def find(self, i):
        if self.parent[i] != i:
            self.parent[i] = self.find(self.parent[i])
        return self.parent[i]

    def union_by_rank(self, i, j):
        irep = self.find(i)
        jrep = self.find(j)
        if irep == jrep:
            return
        irank = self.rank[irep]
        jrank = self.rank[jrep]
        if irank < jrank:
            self.parent[irep] = jrep
        elif jrank < irank:
            self.parent[jrep] = irep
        else:
            self.parent[irep] = jrep
            self.rank[jrep] += 1

    def main(self):
        size = 5
        ds = DisjointSet(size)
        ds.union_by_rank(0, 1)
        ds.union_by_rank(2, 3)
        ds.union_by_rank(1, 3)
```

11

Time Complexity

Make Set	Find Set	Union
$O(1)$	• $O(\log n)$ without path compression • $O(\alpha(n))$ with path compression; $\alpha(n)$ is the inverse Ackermann function	$O(\log n)$

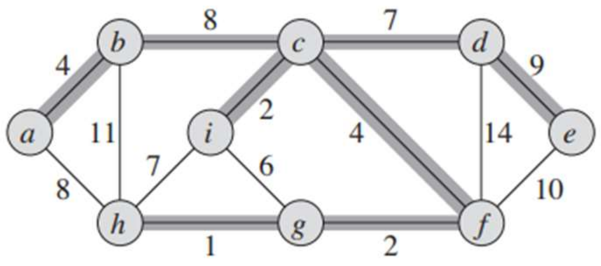
12

Outline

1. Disjoint Data Structure
2. **Minimum Cost Spanning Tree**
   - **Kruskal Algorithm**
   - Prim's Algorithm
3. Shortest Path Algorithm
   - Floyd Warshal Algorithm

13

Minimum Spanning Trees



14

Minimum Spanning Tree

- A minimum spanning tree (MST) is a subset of the edges of a connected, edge-weighted graph that connects all the vertices together without any cycles and with the minimum possible total edge weight.
- It is a way of finding the most economical way to connect a set of vertices.

15

Minimum Spanning Tree

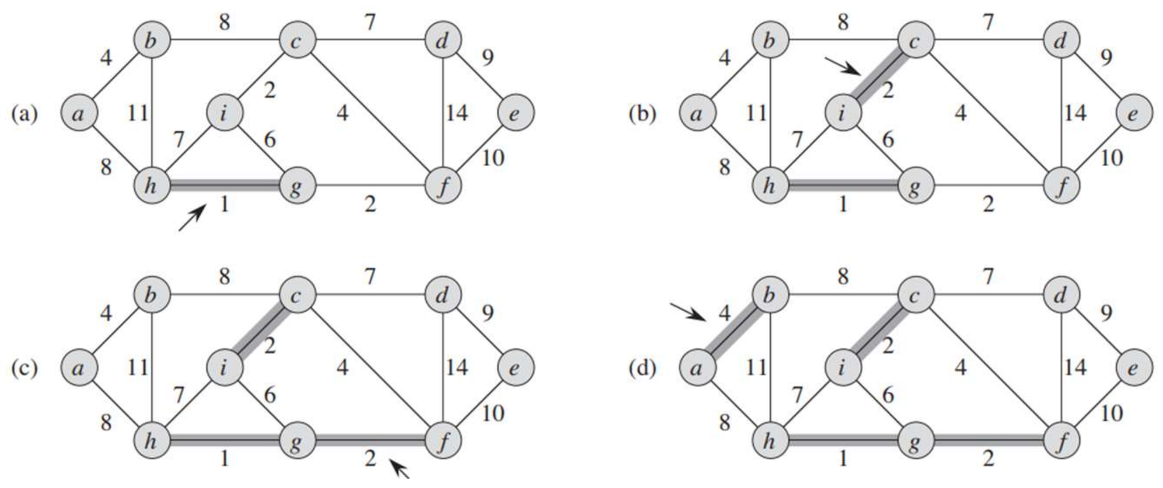
- The total weight connected across the vertices should be minimum.
- Definition
  - To find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight $w(T) = \sum w(u, v)$ is minimized. Since T is acyclic and connects all of the vertices, it must form a tree, which we call a spanning tree since it spans the graph G.
  - We call the problem of determining the tree T the minimum spanning tree problem.

16

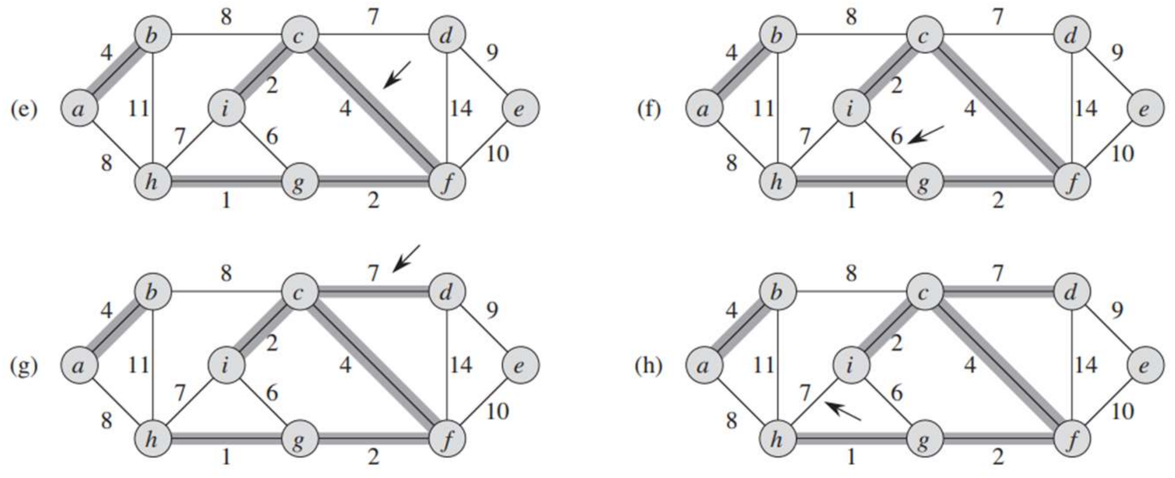
Kruskal Algorithm

```
MST-KRUSKAL($G, w$)
1  $A = \emptyset$
2  for each vertex $v \in G.V$
3      MAKE-SET($v$)
4  sort the edges of $G.E$ into nondecreasing order by weight $w$
5  for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6      if FIND-SET($u$) $\neq$ FIND-SET($v$)
7          $A = A \cup \{(u, v)\}$
8          UNION($u, v$)
9  return $A$
```

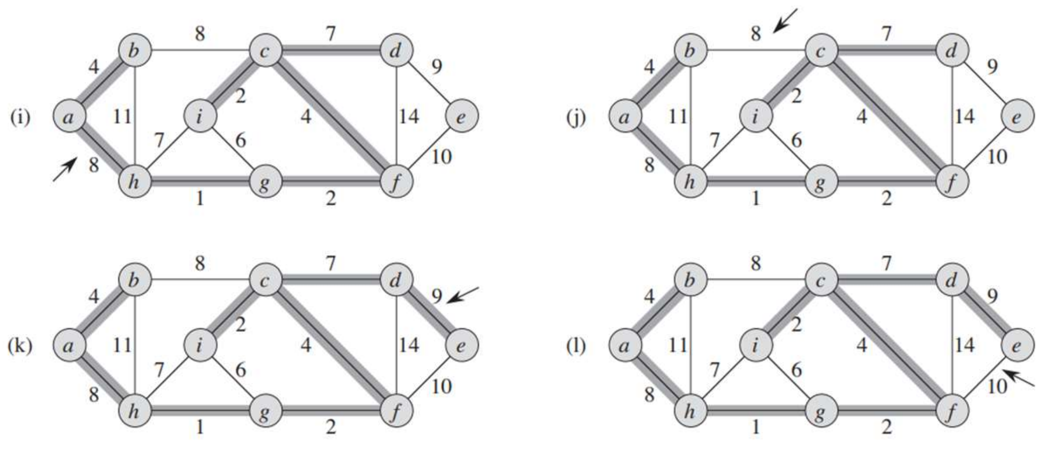
17



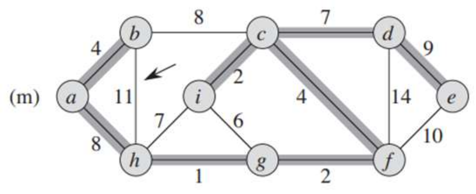
18

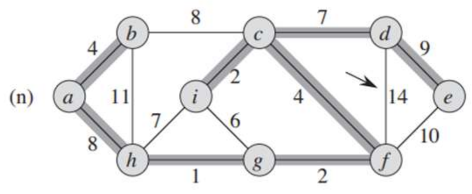

(e)
(f)
(g)
(h)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

19


(i)
(j)
(k)
(l)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

20





21

Time Complexity

- $O(E. \log V)$

22

Outline

1. Disjoint Data Structure
2. **Minimum Cost Spanning Tree**
   - Kruskal Algorithm
   - **Prim's Algorithm**
3. Shortest Path Algorithm
   - Floyd Warshal Algorithm

23

Prim's Algorithm

- Greedy algorithm that is used to find the minimum spanning tree from a graph.
- Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.
- Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph get selected.

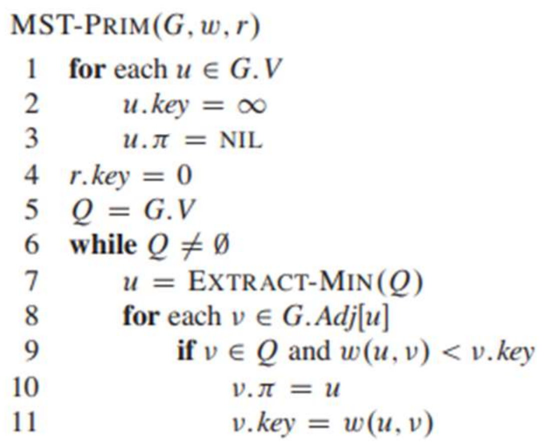
24

Steps

- Maintain two disjoint sets of vertices. One containing vertices that are in the growing spanning tree and other that are not in the growing spanning tree.
- Select the cheapest vertex that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using Priority Queues. Insert the vertices, that are connected to growing spanning tree, into the Priority Queue.
- Check for cycles. To do that, mark the nodes which have been already selected and insert only those nodes in the Priority Queue that are not marked.

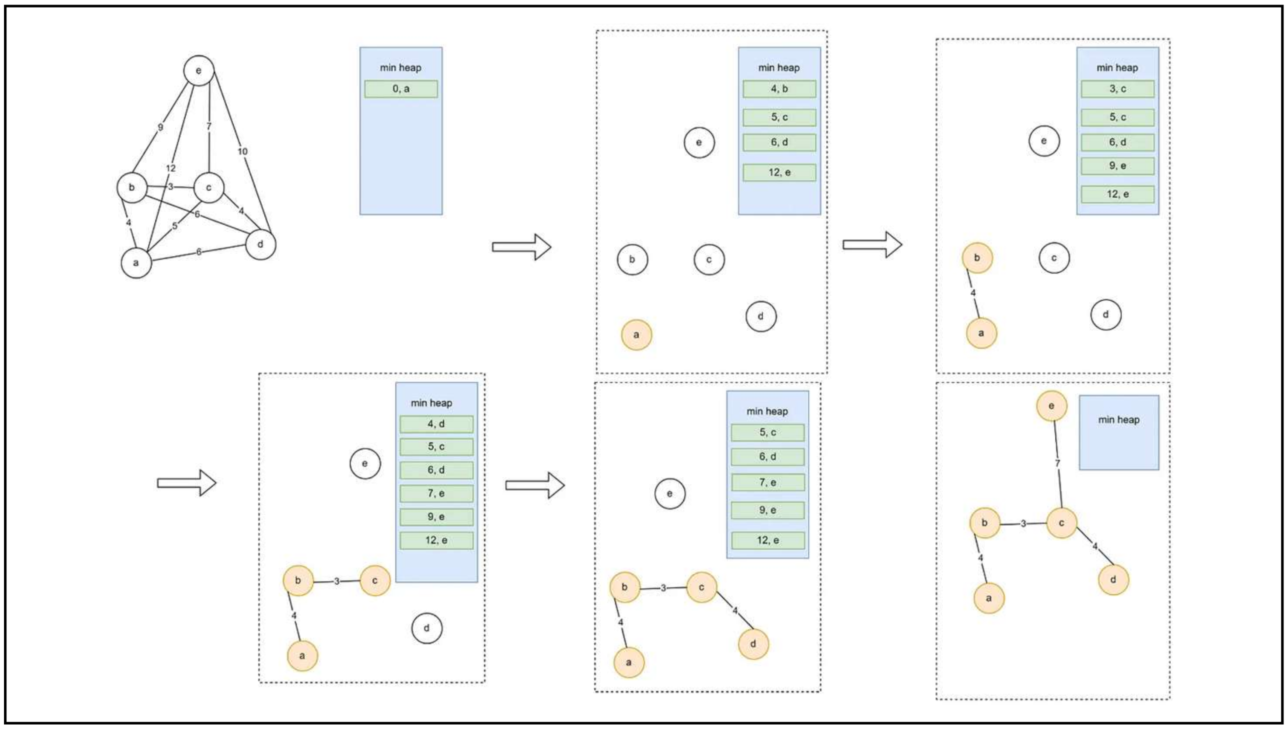
25

Prim's Algorithm



```
MST-PRIM($G, w, r$)
for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = $ NIL
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = $ EXTRACT-MIN($Q$)
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$
```

26



27

Time Complexity

- $O(E + \log V)$

28

Outline

1. Disjoint Data Structure
2. Minimum Cost Spanning Tree
   - Kruskal Algorithm
   - Prim's Algorithm
3. Shortest Path Algorithm
   - **Floyd Warshal Algorithm**

29

Floyd Warshall Algorithm

- Floyd Warshall Algorithm is used to find the shortest paths between all pairs of vertices in a graph, where each edge in the graph has a weight which is positive or negative.
- The biggest advantage of using this algorithm is that all the shortest distances between any vertices could be calculated in where is the number of vertices in a graph.

30

Floyd Warshall Algorithm

Steps:

For a graph G(V,E) with N vertices:

1. Initialize the shortest paths between any 2 vertices with infinity
2. Find all pair shortest paths that use 0 intermediate vertices, then find the shortest paths that use 1 intermediate vertex. Repeat until all N vertices as intermediate nodes.
3. Minimize the shortest paths between any 2 pair in the previous operation.
4. For any 2 vertices (i, j), one should actually minimize the distances between this pair using the first K nodes, so the shortest path will be $\min(dist[i][k] + dist[k][j], dist[i][j])$.

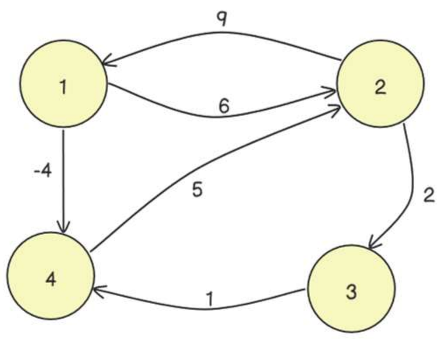
31

Floyd Warshall Algorithm

- The time complexity for Floyd Warshall Algorithm is $O(V^3)$, where V is the number of vertices in a graph.

32

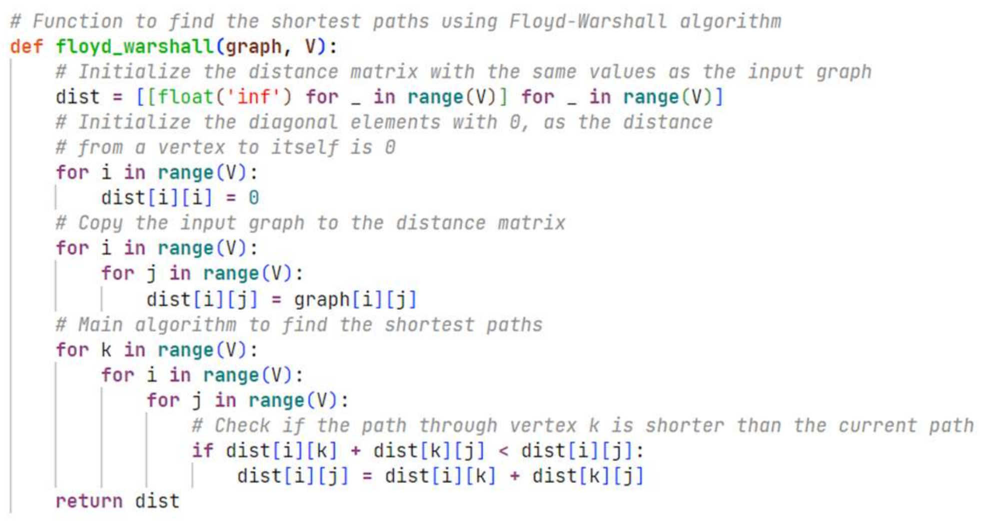
Example



33

34

Algorithm: Floyd Warshall



```python
# Function to find the shortest paths using Floyd-Warshall algorithm
def floyd_warshall(graph, V):
    # Initialize the distance matrix with the same values as the input graph
    dist = [[float('inf') for _ in range(V)] for _ in range(V)]
    # Initialize the diagonal elements with 0, as the distance
    # from a vertex to itself is 0
    for i in range(V):
        dist[i][i] = 0
    # Copy the input graph to the distance matrix
    for i in range(V):
        for j in range(V):
            dist[i][j] = graph[i][j]
    # Main algorithm to find the shortest paths
    for k in range(V):
        for i in range(V):
            for j in range(V):
                # Check if the path through vertex k is shorter than the current path
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
    return dist
```

35

Applications of Floyd Warshall Algorithm

- Finding shortest paths in the weight graph
- Network Routing Protocols
- Traffic Engineering

36