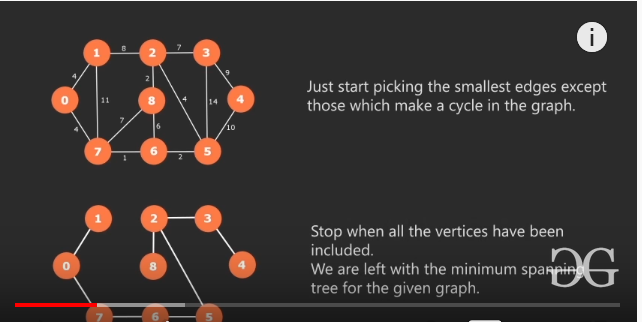
Minimum Spanning Tree

What is Minimum Spanning Tree?  
Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

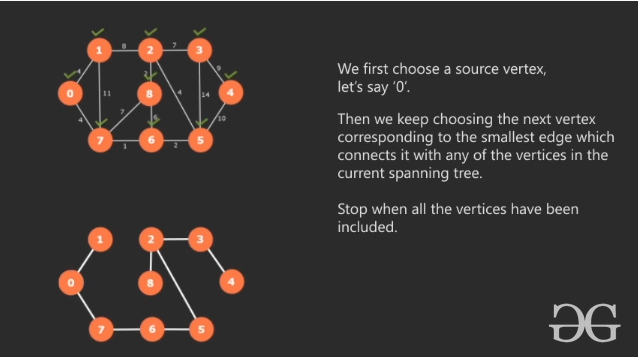
How many edges does a minimum spanning tree has?  
A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

Below are the steps for finding MST using Kruskal’s algorithm

**1.** Sort all the edges in non-decreasing order of their weight.  
**2.** Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.  
**3.** Repeat step#2 until there are (V-1) edges in the spanning tree.



Prims Algorithm:



// A Java program for Prim's Minimum Spanning Tree (MST) algorithm.

// The program is for adjacency matrix representation of the graph

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class MST {

// Number of vertices in the graph

private static final int V = 5;

// A utility function to find the vertex with minimum key

// value, from the set of vertices not yet included in MST

int minKey(int key[], Boolean mstSet[])

{

// Initialize min value

int min = Integer.MAX\_VALUE, min\_index = -1;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min) {

min = key[v];

min\_index = v;

}

return min\_index;

}

// A utility function to print the constructed MST stored in

// parent[]

void printMST(int parent[], int graph[][])

{

System.out.println("Edge \tWeight");

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

System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);

}

// Function to construct and print MST for a graph represented

// using adjacency matrix representation

void primMST(int graph[][])

{

// Array to store constructed MST

int parent[] = new int[V];

// Key values used to pick minimum weight edge in cut

int key[] = new int[V];

// To represent set of vertices not yet included in MST

Boolean mstSet[] = new Boolean[V];

// Initialize all keys as INFINITE

for (int i = 0; i < V; i++) {

key[i] = Integer.MAX\_VALUE;

mstSet[i] = false;

}

// Always include first 1st vertex in MST.

key[0] = 0; // Make key 0 so that this vertex is

// picked as first vertex

parent[0] = -1; // First node is always root of MST

// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

// Pick thd minimum key vertex from the set of vertices

// not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = true;

// Update key value and parent index of the adjacent

// vertices of the picked vertex. Consider only those

// vertices which are not yet included in MST

for (int v = 0; v < V; v++)

// graph[u][v] is non zero only for adjacent vertices of m

// mstSet[v] is false for vertices not yet included in MST

// Update the key only if graph[u][v] is smaller than key[v]

if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

// print the constructed MST

printMST(parent, graph);

}

public static void main(String[] args)

{

/\* Let us create the following graph

2 3

(0)--(1)--(2)

| / \ |

6| 8/ \5 |7

| / \ |

(3)-------(4)

9 \*/

MST t = new MST();

int graph[][] = new int[][] { { 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 } };

// Print the solution

t.primMST(graph);

}

}

// Java program to count islands in boolean 2D matrix

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Islands {

// No of rows and columns

static final int ROW = 5, COL = 5;

// A function to check if a given cell (row, col) can

// be included in DFS

boolean isSafe(int M[][], int row, int col, boolean visited[][])

{

// row number is in range, column number is in range

// and value is 1 and not yet visited

return (row >= 0) && (row < ROW) && (col >= 0) && (col < COL) && (M[row][col] == 1 && !visited[row][col]);

}

// A utility function to do DFS for a 2D boolean matrix.

// It only considers the 8 neighbors as adjacent vertices

void DFS(int M[][], int row, int col, boolean visited[][])

{

// These arrays are used to get row and column numbers of 8 neighbors of a given cell

int rowNbr[] = new int[] { -1, -1, -1, 0, 0, 1, 1, 1 };

int colNbr[] = new int[] { -1, 0, 1, -1, 1, -1, 0, 1 };

// Mark this cell as visited

visited[row][col] = true;

// Recur for all connected neighbours

for (int k = 0; k < 8; ++k)

if (isSafe(M, row + rowNbr[k], col + colNbr[k], visited))

DFS(M, row + rowNbr[k], col + colNbr[k], visited);

}

// The main function that returns count of islands in a given

// boolean 2D matrix

int countIslands(int M[][])

{

// Make a bool array to mark visited cells.

// Initially all cells are unvisited

boolean visited[][] = new boolean[ROW][COL];

// Initialize count as 0 and travese through the all cells

// of given matrix

int count = 0;

for (int i = 0; i < ROW; ++i)

for (int j = 0; j < COL; ++j)

if (M[i][j] == 1 && !visited[i][j]) // If a cell with

{ // value 1 is not visited yet, then new island found, Visit all cells in this island and increment island count

DFS(M, i, j, visited);

++count;

}

return count;

}

// Driver method

public static void main(String[] args) throws java.lang.Exception

{

int M[][] = new int[][] { { 1, 1, 0, 0, 0 },

{ 0, 1, 0, 0, 1 },

{ 1, 0, 0, 1, 1 },

{ 0, 0, 0, 0, 0 },

{ 1, 0, 1, 0, 1 } };

Islands I = new Islands();

System.out.println("Number of islands is: " + I.countIslands(M));

}

}

**DFS**

**BFS**

**Detect Cycle in a Directed Graph**

# Strongly Connected Components

# Dijkstra’s shortest path algorithm

# Bellman–Ford Algorithm

# Topological Sorting

# Floyd Warshall Algorithm