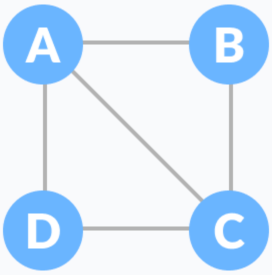
Spanning Tree and Minimum Spanning Tree

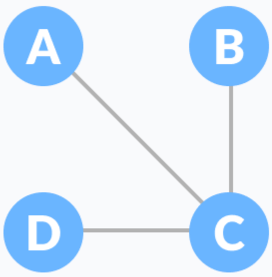
Before we learn about spanning trees, we need to understand two graphs: undirected graphs and connected graphs.

An **undirected graph** is a graph in which the edges do not point in any direction (ie. the edges are bidirectional).



Undirected Graph

A **connected graph** is a graph in which there is always a path from a vertex to any other vertex.



Connected Graph

Spanning tree

A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.

The edges may or may not have weights assigned to them.

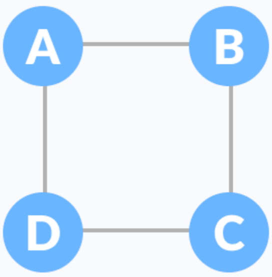
The total number of spanning trees with n vertices that can be created from a complete graph is equal to $n^{(n-2)}$.

If we have $n = 4$, the maximum number of possible spanning trees is equal to $4^{4-2} = 16$. Thus, 16 spanning trees can be formed from a complete graph with 4 vertices.

Example of a Spanning Tree

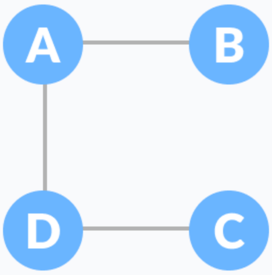
Let's understand the spanning tree with examples below:

Let the original graph be:

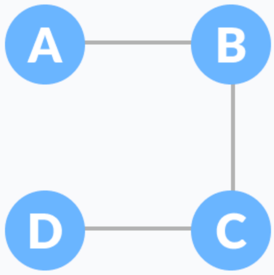


Normal graph

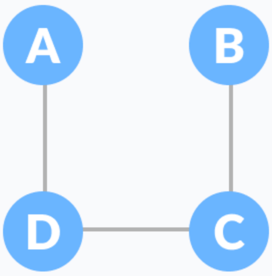
Some of the possible spanning trees that can be created from the above graph are:



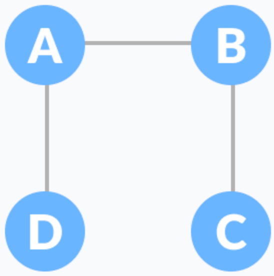
A spanning tree



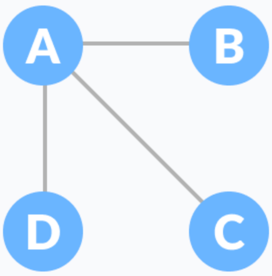
A spanning tree



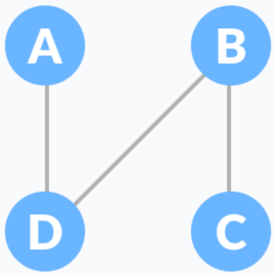
A spanning tree



A spanning tree



A spanning tree



A spanning tree

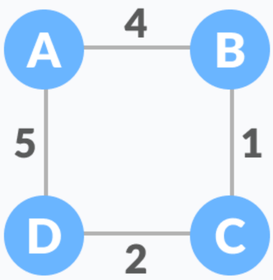
Minimum Spanning Tree

A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.

Example of a Spanning Tree

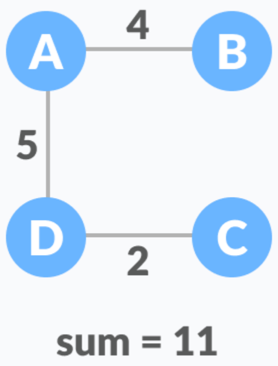
Let's understand the above definition with the help of the example below.

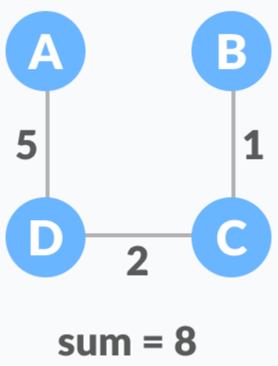
The initial graph is:



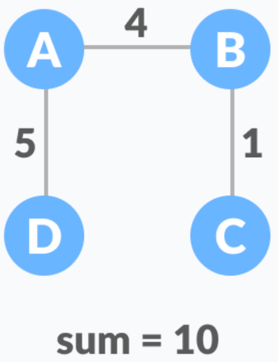
Weighted graph

The possible spanning trees from the above graph are:



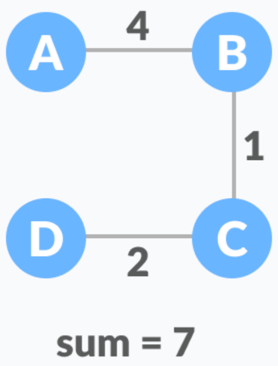


Minimum spanning tree - 1



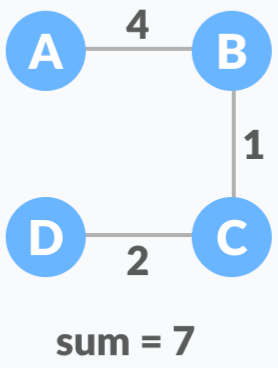
Minimum spanning tree - 2

Minimum spanning tree - 3



Minimum spanning tree - 4

The minimum spanning tree from the above spanning trees is:



Minimum spanning tree

The minimum spanning tree from a graph is found using the following algorithms:

1. Prim's Algorithm
2. Kruskal's Algorithm

Spanning Tree Applications

- Computer Network Routing Protocol
- Cluster Analysis
- Civil Network Planning

Minimum Spanning tree Applications

- To find paths in the map
- To design networks like telecommunication networks, water supply networks, and electrical grids.

Prim’s algorithmPrim's Algorithm

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph

---

How Prim's algorithm works

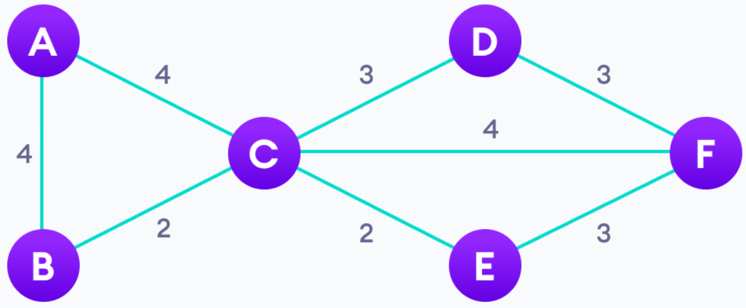
It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.
We start from one vertex and keep adding edges with the lowest weight until we reach our goal.

The steps for implementing Prim's algorithm are as follows:

1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
3. Keep repeating step 2 until we get a minimum spanning tree

---

Example of Prim's algorithm

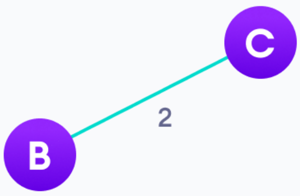


Step: 1

Start with a weighted graph

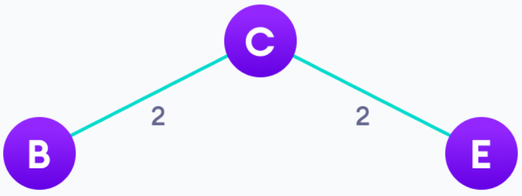
Step: 2

Choose a vertex



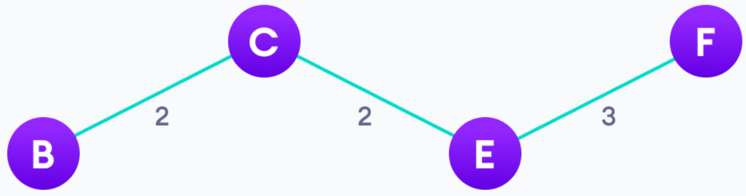
Step: 3

Choose the shortest edge from this vertex and add it



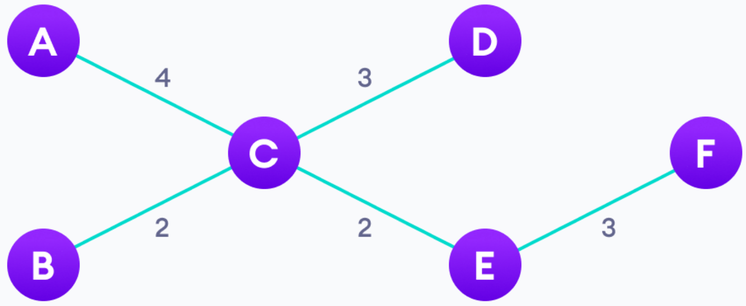
Step: 4

Choose the nearest vertex not yet in the solution



Step: 5

Choose the nearest edge not yet in the solution, if there are multiple choices, choose one at random



Step: 6

Repeat until you have a spanning tree

Prim's Algorithm pseudocode

The pseudocode for prim's algorithm shows how we create two sets of vertices U and V-U. U contains the list of vertices that have been visited and V-U the list of vertices that haven't. One by one, we move vertices from set V-U to set U by connecting the least weight edge.

```
T = ∅;
U = { 1 };
while (U ≠ V)
    let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
    T = T ∪ {(u, v)}
```

```
U = U ∪ {v}
```

Python, Java and C/C++ Examples

Although adjacency matrix representation of graphs is used, this algorithm can also be implemented using Adjacency List to improve its efficiency.

Python
Java
C
C++

```
// Prim's Algorithm in Java

import java.util.Arrays;

class PGraph {

  public void Prim(int G[][], int V) {

    int INF = 9999999;

    int no_edge; // number of edge

    // create a array to track selected vertex
    // selected will become true otherwise false
    boolean[] selected = new boolean[V];

    // set selected false initially
    Arrays.fill(selected, false);

    // set number of edge to 0
    no_edge = 0;

    // the number of egde in minimum spanning tree will be
    // always less than (V -1), where V is number of vertices in
    // graph

    // choose 0th vertex and make it true
    selected[0] = true;

    // print for edge and weight
```

```
    System.out.println("Edge : Weight");

    while (no_edge < V - 1) {
      // For every vertex in the set S, find the all adjacent vertices
      // , calculate the distance from the vertex selected at step 1.
      // if the vertex is already in the set S, discard it otherwise
      // choose another vertex nearest to selected vertex at step 1.

      int min = INF;
      int x = 0; // row number
      int y = 0; // col number

      for (int i = 0; i < V; i++) {
        if (selected[i] == true) {
          for (int j = 0; j < V; j++) {
            // not in selected and there is an edge
            if (!selected[j] && G[i][j] != 0) {
              if (min > G[i][j]) {
                min = G[i][j];
                x = i;
                y = j;
              }
            }
          }
        }
      }
      System.out.println(x + " - " + y + " :  " + G[x][y]);
      selected[y] = true;
      no_edge++;
    }
  }

  public static void main(String[] args) {
    PGraph g = new PGraph();

    // number of vertices in grapj
    int V = 5;

    // create a 2d array of size 5x5
    // for adjacency matrix to represent graph
    int[][] G = { { 0, 9, 75, 0, 0 }, { 9, 0, 95, 19, 42 }, { 75, 95, 0, 51, 66 }, {
0, 19, 51, 0, 31 },
        { 0, 42, 66, 31, 0 } };

    g.Prim(G, V);
  }
```

```
}
```

Prim's vs Kruskal's Algorithm

Kruskal's algorithm is another popular minimum spanning tree algorithm that uses a different logic to find the MST of a graph. Instead of starting from a vertex, Kruskal's algorithm sorts all the edges from low weight to high and keeps adding the lowest edges, ignoring those edges that create a cycle.

Prim's Algorithm Complexity

The time complexity of Prim's algorithm is `O(E log V)`.

Prim's Algorithm Application

- Laying cables of electrical wiring
- In network designed
- To make protocols in network cycles

Kruskal's Algorithm

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph

---

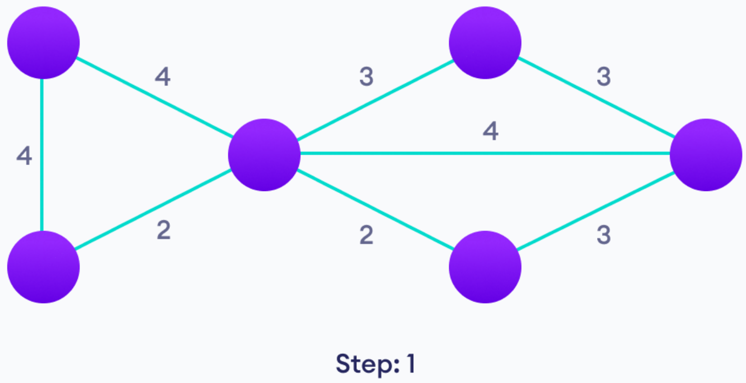
How Kruskal's algorithm works

It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.
We start from the edges with the lowest weight and keep adding edges until we reach our goal.

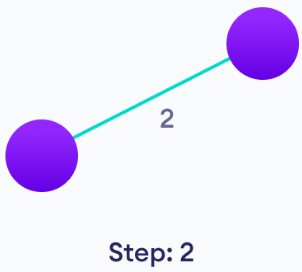
The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

Example of Kruskal's algorithm

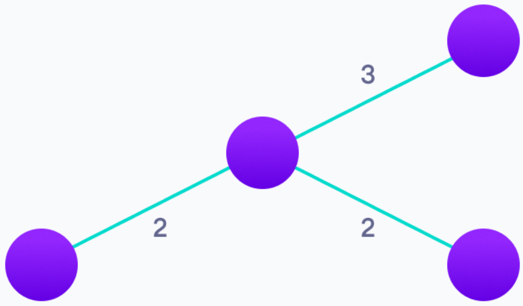


Start with a weighted graph



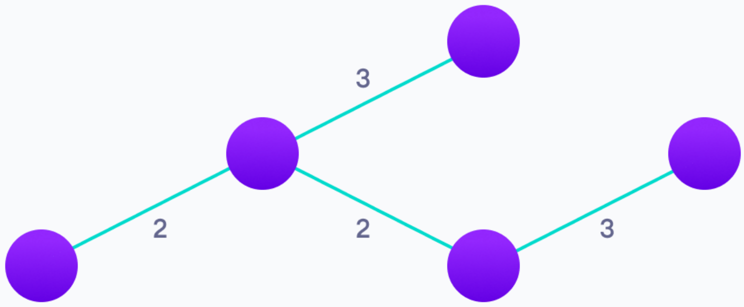
Choose the edge with the least weight, if there are more than 1, choose anyone

Choose the next shortest edge and add it



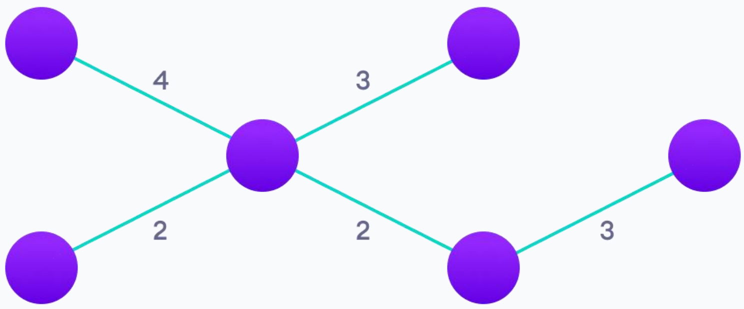
Step: 4

Choose the next shortest edge that doesn't create a cycle and add it



Step: 5

Choose the next shortest edge that doesn't create a cycle and add it



Step: 6

Repeat until you have a spanning tree

Kruskal Algorithm Pseudocode

Any minimum spanning tree algorithm revolves around checking if adding an edge creates a loop or not.

The most common way to find this out is an algorithm called Union FInd. The Union-Find algorithm divides the vertices into clusters and allows us to check if two vertices belong to the same cluster or not and hence decide whether adding an edge creates a cycle.

```
KRUSKAL(G):
A = ∅
For each vertex v ∈ G.V:
    MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
    if FIND-SET(u) ≠ FIND-SET(v):
    A = A ∪ {(u, v)}
    UNION(u, v)
return A
```

Java Examples

```java
// Kruskal's algorithm in Java

import java.util.*;

class Graph {
  class Edge implements Comparable<Edge> {
    int src, dest, weight;

    public int compareTo(Edge compareEdge) {
      return this.weight - compareEdge.weight;
    }
  };

  // Union
  class subset {
    int parent, rank;
```

```
};

int vertices, edges;
Edge edge[];

// Graph creation
Graph(int v, int e) {
  vertices = v;
  edges = e;
  edge = new Edge[edges];
  for (int i = 0; i < e; ++i)
    edge[i] = new Edge();
}

int find(subset subsets[], int i) {
  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
  return subsets[i].parent;
}

void Union(subset subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);

  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot;
  else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;
  }
}

// Applying Krushkal Algorithm
void KruskalAlgo() {
  Edge result[] = new Edge[vertices];
  int e = 0;
  int i = 0;
  for (i = 0; i < vertices; ++i)
    result[i] = new Edge();

  // Sorting the edges
  Arrays.sort(edge);
  subset subsets[] = new subset[vertices];
  for (i = 0; i < vertices; ++i)
```

```
      subsets[i] = new subset();

    for (int v = 0; v < vertices; ++v) {
      subsets[v].parent = v;
      subsets[v].rank = 0;
    }
    i = 0;
    while (e < vertices - 1) {
      Edge next_edge = new Edge();
      next_edge = edge[i++];
      int x = find(subsets, next_edge.src);
      int y = find(subsets, next_edge.dest);
      if (x != y) {
        result[e++] = next_edge;
        Union(subsets, x, y);
      }
    }
    for (i = 0; i < e; ++i)
      System.out.println(result[i].src + " - " + result[i].dest + ": " +
result[i].weight);
  }

  public static void main(String[] args) {
    int vertices = 6; // Number of vertices
    int edges = 8; // Number of edges
    Graph G = new Graph(vertices, edges);

    G.edge[0].src = 0;
    G.edge[0].dest = 1;
    G.edge[0].weight = 4;

    G.edge[1].src = 0;
    G.edge[1].dest = 2;
    G.edge[1].weight = 4;

    G.edge[2].src = 1;
    G.edge[2].dest = 2;
    G.edge[2].weight = 2;

    G.edge[3].src = 2;
    G.edge[3].dest = 3;
    G.edge[3].weight = 3;

    G.edge[4].src = 2;
    G.edge[4].dest = 5;
    G.edge[4].weight = 2;
```

```
    G.edge[5].src = 2;
    G.edge[5].dest = 4;
    G.edge[5].weight = 4;

    G.edge[6].src = 3;
    G.edge[6].dest = 4;
    G.edge[6].weight = 3;

    G.edge[7].src = 5;
    G.edge[7].dest = 4;
    G.edge[7].weight = 3;
    G.KruskalAlgo();
  }
}
```

Kruskal's vs Prim's Algorithm

Prim's algorithm is another popular minimum spanning tree algorithm that uses a different logic to find the MST of a graph. Instead of starting from an edge, Prim's algorithm starts from a vertex and keeps adding lowest-weight edges which aren't in the tree, until all vertices have been covered.

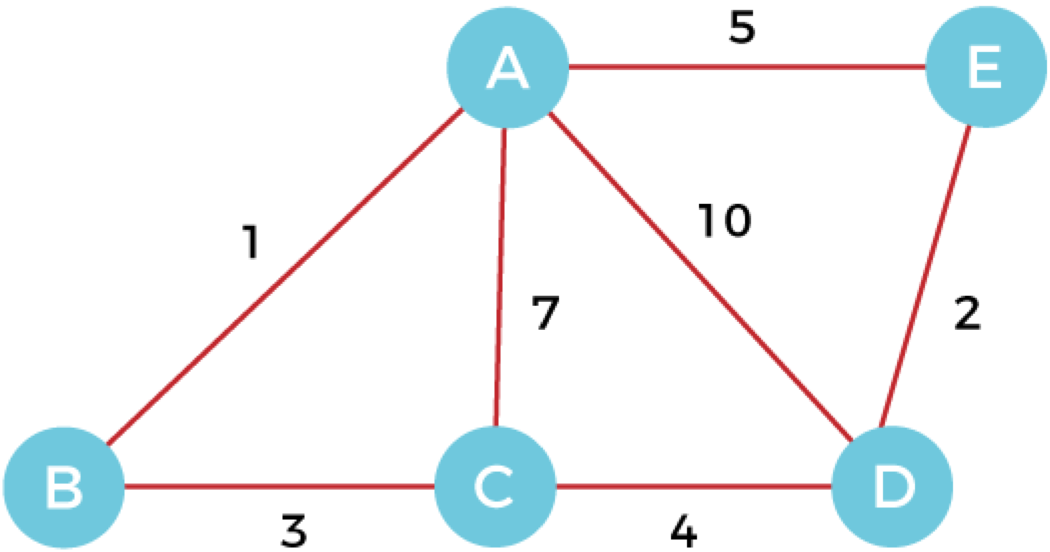
Kruskal's Algorithm Complexity

The time complexity Of Kruskal's Algorithm is: `O(E log E)`.

Kruskal's Algorithm Applications

- In order to layout electrical wiring

- In computer network (LAN connection)



Answer