

Applications of Minimum Spanning Tree Problem

Prim's Minimum Spanning Tree (MST) | Greedy Algo-5

Print all the cycles in an undirected graph

Detect cycle in an undirected graph using BFS

Samsung Semiconductor Institute of Research(SSIR Software) intern/FTE | Set-3

Check if a given graph is Bipartite using DFS

Largest connected component on a grid

Nutanix Interview Experience (On-Campus)

Kruskal's Algorithm (Simple Implementation for Adjacency Matrix)

Detect Cycle in a Directed Graph using BFS

Find the number of distinct islands in a 2D matrix

Sum of the minimum elements in all connected components of an undirected graph

Tree, Back, Edge and Cross Edges in DFS of Graph

Level Ancestor Problem

Maximum number of edges among all connected components of an undirected graph

Minimum cost path from source node to destination node via an intermediate node

Johnson's algorithm for All-pairs shortest paths | Implementation

Dijkstra's shortest path with minimum edges

Applications of Graph Data Structure

Coloring a Cycle Graph

Number of Isosceles triangles in a binary tree

Edge Coloring of a Graph

Prim's Algorithm (Simple Implementation for Adjacency Matrix Representation)

Minimum time to return array to its original state after given modifications

Find the probability of a state at a given time in a Markov chain | Set 1

Number of Walks from source to destination

Find whether it is possible to finish all tasks or not from given dependencies

Find the ordering of tasks from given dependencies

Subtree of all nodes in a tree using DFS

Graph Types and Applications

Kruskal's Minimum Spanning Tree Algorithm | Greedy Algo-2

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.

What are the applications of Minimum Spanning Tree?

See [this](#) for applications of MST.

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

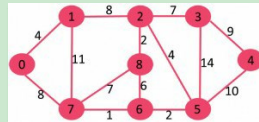
1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are $(V-1)$ edges in the spanning tree.

The step#2 uses [Union-Find algorithm](#) to detect cycle. So we recommend to read following post as a prerequisite.

[Union-Find Algorithm | Set 1 \(Detect Cycle in a Graph\)](#)

[Union-Find Algorithm | Set 2 \(Union By Rank and Path Compression\)](#)

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Consider the below input graph.



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having $(9 - 1) = 8$ edges.

After sorting:

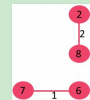
Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from sorted list of edges

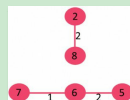
1. Pick edge 7-6: No cycle is formed, include it.



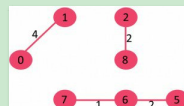
2. Pick edge 8-2: No cycle is formed, include it.



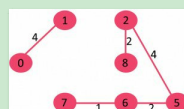
3. Pick edge 6-5: No cycle is formed, include it.



4. Pick edge 0-1: No cycle is formed, include it.



5. Pick edge 2-5: No cycle is formed, include it.



Most popular in Graph

Jump Pointer Algorithm

Find maximum path length in a binary matrix

De Bruijn sequence | Set 1

Degree of a Cycle Graph

Maximum and minimum isolated vertices in a graph

Most visited in Greedy

Maximum sum by picking elements from two arrays in order

Coin game of two corners (Greedy Approach)

Maximum sum of all elements of array after performing given operations

Check if an array is Wave Array

Water drop problem



Geeks Classes

Classroom program on Algorithms in Noida
Mentored by Mr. Sandeep Jain

Batch starts from **22nd Jan, 2019**
Classes on **Tuesday and Friday**

GeeksforGeeks

A computer science portal for geeks



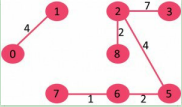
TECHNICAL
SCRIPTER

2 OCT to 10 JAN

A technical content writing
event by GeeksforGeeks

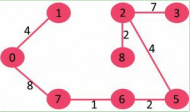
6. Pick edge 8-6: Since including this edge results in cycle, discard it.

7. Pick edge 2-3: No cycle is formed, include it.



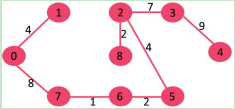
8. Pick edge 7-8: Since including this edge results in cycle, discard it.

9. Pick edge 0-7: No cycle is formed, include it.



10. Pick edge 1-2: Since including this edge results in cycle, discard it.

11. Pick edge 3-4: No cycle is formed, include it.



Since the number of edges included equals $(V - 1)$, the algorithm stops here.

Recommended: Please try your approach on [IDE](#) first, before moving on to the solution.

C/C++ Java Python



```
// C++ program for Kruskal's algorithm to find Minimum Spanning Tree
// of a given connected, undirected and weighted graph
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// a structure to represent a weighted edge in graph
struct Edge
{
    int src, dest, weight;
};

// a structure to represent a connected, undirected
// and weighted graph
struct Graph
{
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges.
    // Since the graph is undirected, the edge
    // from src to dest is also edge from dest
    // to src. Both are counted as 1 edge here.
    struct Edge* edge;
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;

    graph->edge = new Edge[E];

    return graph;
}

// A structure to represent a subset for union-find
struct subset
{
    int parent;
    int rank;
};

// A utility function to find set of an element i
// (uses path compression technique)
int find(struct subset subsets[], int i)
{
    // find root and make root as parent of i
    // (path compression)
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high
    // rank tree (Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // If ranks are same, then make one as root and
    // increment its rank by one
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// Compare two edges according to their weights.
// Used in qsort() for sorting an array of edges
int myComp(const void* a, const void* b)
{

```

Maximum Possible Edge Disjoint Spanning Tree From a Complete Graph

Find if an undirected graph contains an independent set of a given size

All vertex pairs connected with exactly k edges in a graph

Finding the probability of a state at a given time in a Markov chain | Set 2

Program to find total number of edges in a Complete Graph

Right sibling of each node in a tree given as array of edges

Program to find Circuit Rank of an Undirected Graph

Undirected graph splitting and its application for number pairs

Dominant Set of a Graph

Check if a given tree graph is linear or not

Product of lengths of all cycles in an undirected graph

Maximize number of nodes which are not part of any edge in a Graph

Program to Calculate the Edge Cover of a Graph

Program to find the diameter, cycles and edges of a Wheel Graph

Computer Networks | Cuts and Network Flow

```

    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}

// The main function to construct MST using Kruskal's algorithm
void KruskalMST(struct Graph* graph)
{
    int V = graph->V;
    struct Edge result[V]; // This will store the resultant MST
    int e = 0; // An index variable, used for result[]
    int i = 0; // An index variable, used for sorted edges

    // Step 1: Sort all the edges in non-decreasing
    // order of their weight. If we are not allowed to
    // change the given graph, we can create a copy of
    // array of edges
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

    // Allocate memory for creating V subsets
    struct subset *subsets =
        (struct subset*) malloc( V * sizeof(struct subset) );

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Number of edges to be taken is equal to V-1
    while (e < V - 1)
    {
        // Step 2: Pick the smallest edge. And increment
        // the index for next iteration
        struct Edge next_edge = graph->edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        // If including this edge does't cause cycle,
        // include it in result and increment the index
        // of result for next edge
        if (x != y)
        {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
        // Else discard the next_edge
    }

    // print the contents of result[] to display the
    // built MST
    printf("Following are the edges in the constructed MST\n");
    for (i = 0; i < e; ++i)
        printf("%d -- %d == %d\n", result[i].src, result[i].dest,
            result[i].weight);

    return;
}

// Driver program to test above functions
int main()
{
    /* Let us create following weighted graph
        10
        0-----1
        |      \
        6   5\   15
        |      \
        2-----3
            4
        */
    int V = 4; // Number of vertices in graph
    int E = 5; // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    // add edge 0-2
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;

    // add edge 0-3
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;

    // add edge 1-3
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;

    // add edge 2-3
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;

    KruskalMST(graph);

    return 0;
}

```

```

Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10

```

Time Complexity: $O(E \log E)$ or $O(E \log V)$. Sorting of edges takes $O(E \log E)$ time. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take atmost $O(\log V)$ time. So overall complexity is $O(E \log E + E \log V)$ time. The value of E can be atmost $O(V^2)$, so $O(\log V)$ are $O(\log E)$ same. Therefore, overall time complexity is $O(E \log E)$ or $O(E \log V)$



References:

http://www.ics.uci.edu/~epstein/161/960206.html
http://en.wikipedia.org/wiki/Minimum_spanning_tree

This article is compiled by [Aashish Barnwal](#) and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

- Prim's Minimum Spanning Tree (MST) | Greedy Algo-5
- Boruvka's algorithm for Minimum Spanning Tree
- Reverse Delete Algorithm for Minimum Spanning Tree
- Kruskal's Minimum Spanning Tree using STL in C++
- Minimum Product Spanning Tree
- Applications of Minimum Spanning Tree Problem
- Greedy Algorithm to find Minimum number of Coins
- Boruvka's algorithm | Greedy Algo-9
- Graph Coloring | Set 2 (Greedy Algorithm)
- Greedy Algorithm for Egyptian Fraction
- Job Sequencing Problem | Set 1 (Greedy Algorithm)
- K Centers Problem | Set 1 (Greedy Approximate Algorithm)
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Dijkstra's shortest path algorithm | Greedy Algo-7
- Dijkstra's Algorithm for Adjacency List Representation | Greedy Algo-8

Article Tags : [Graph](#) [Greedy](#) [Kruskal](#) [Kruskal'sAlgorithm](#) [MST](#)

Practice Tags : [Greedy](#) [Graph](#)



3.5

☐ To-do ☐ Done

Based on 215 vote(s)

- Feedback
- Add Notes
- Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous

[K-Union-Find Algorithm | Set 2 \(Union By Rank and Path Compression\)](#)

Next

[Huffman Coding | Greedy Algo-3](#)

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

GeeksforGeeks
A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-147, Noida-201305

COMPANY

- About Us
- Careers
- Privacy Policy
- Contact Us

LEARN

- Algorithms
- Data Structures
- Languages
- CS Subjects

PRACTICE

- Company-wise
- Topic-wise
- Contests
- Subjective Questions

CONTRIBUTE

- Write an Article
- Write Interview Experience
- Internships
- Videos

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#)

Got it!