

Final Group

Manuweera Dilhani Samanmali – M20W0034

Fernando Pulle Kaushila Hansini - M20W0067

Ranasinghage Dona Dulani Piyumika Ranasi - M20W0068

Fundamentals of Computer Programming

Minimum Spanning Tree

Group Members

2020

Spring Semester

ASSIGNMENT

Contents

[Minimum Spanning Tree 2](#_Toc48310653)

[Prim’s Algorithm 4](#_Toc48310654)

[Kruskal’ Algorithm 8](#_Toc48310655)

[Comparing the same graph with Prim’s Algorithm and Kruskal’s Algorithm 12](#_Toc48310656)

[Application of Prim’s Algorithm in Python 13](#_Toc48310657)

[Application of Kruskal’s Algorithm in Python 16](#_Toc48310658)

[References 20](#_Toc48310659)

# Minimum Spanning Tree

Definition of spanning tree: A spanning subgraph which contains all vertices with a minimum number of edges without forming a cycle.

**Example:**

Spanning tree 2

Spanning tree 1

c

d

b

a

d

a

c

b

a

b

c

d

The minimum number of edges is always n-1. Therefore, in the above example the original graph has 4 edges and the spanning tree has n-1 number of edges that is 4-1=3.

Minimum Spanning Tree is a tree or a subgraph which has minimum total weight of the edges. For that, there must be a weight for every edge.

**Example 1:**

The weights are given for the edges. Even though the edge has minimum weight, if it forms a cycle, we should not connect them.

e= n-1

e=4-1 =3

a

d

b

c

5

4

3

2

3

1

a

d

b

c

1

2

3

**Example 2:**

a

a

3

1

b

c

d

2

b

d

c

4

2

2

f

e

3

f

e

In the above example the minimum weight has taken into consideration and drawn the tree. This is how a minimum spanning tree is drawn.

Spanning tree is used in different applications. Below are some of the applications.

Spanning tree applications:

* + Computer Network Routing Protocol
  + Cluster Analysis
  + Civil Network Planning

Minimum spanning tree applications:

* + To find paths in a map
  + To design networks like telecommunication networks, water supply networks, and electrical grids.

# Prim’s Algorithm

A

4

B

C

D

J

H

I

F

G

9

7

7

3

4

4

1

2

2

1

5

1

E

3

2

9

9

3

2

E

9

8

Step 1: Remove loops and parallel edges while keeping minimum weight. When removing loops, you do not have to think about the weight.

Here, the loop at E should be removed without doubt. But when removing one of the parallel lines at CD, we have to consider the minimum weight. We need to keep the one with the minimum weight and remove the highest weight. The highlighted lines should be removed.

A

4

B

C

D

J

H

I

F

G

9

7

7

3

4

4

1

2

2

1

5

1

E

3

A

4

B

C

D

J

H

I

F

G

7

7

3

4

4

1

2

1

5

1

E

3

2

2

Step 2:

While adding new edge, select edge with minimum weight out of the edges from already visited vertices. While doing that, a cycle should not be formed.

Step 3:

Stop at n-1 edges.

So if we take the above algorithm we need to keep a note of the visited vertices and the edge that had the minimum weight. The number of edges we have to stop at is 10 - 1 = 9. You can start from anywhere; here we have started from A. When you select an edge, it is cut from the list and added to the minimum spanning tree.

1st Edge:

A

4

B

C

D

J

H

I

F

G

7

7

3

4

4

1

2

1

5

1

E

3

2

9

9

Visited Vertices = { A, B

Edges to choose from = { ~~AB (1)~~, AC(7),

* First edge is AB since it is having the minimum weight.

2nd Edge:

* When going for the 2nd edge, we need to write all the

possible edges that are connected to B as we stopped from B.

* Since BD has the minimum weight we select BD.

Visited Vertices = { A, B, D

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~

3rd Edge:

* When going for the 3rd edge, we have to write down all the edges connected to the last visited vertices which is D.
* By now we have to have all the edges in the list of edges to choose from because we have to choose the minimum weighted edge from all the edges we have written down.

Visited Vertices = { A, B, D, C

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~, ~~DC(1)~~, DE(2), DH(9), DJ(3),

4th Edge:

* When considering C and the edges that are incident from C, we can see all the edges at already written. Therefore, we can select a minimum weighted edge from already written edges as the 4th edge.

Visited Vertices = { A, B, D, C, E

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~, ~~DC(1)~~, ~~DE(2)~~, DH(9), DJ(3),

5th Edge:

A

4

B

C

D

J

H

I

F

G

7

7

3

4

4

1

2

1

5

1

E

3

2

9

9

Visited Vertices = { A, B, D, C, E, F

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~, ~~DC(1)~~,

~~DE(2)~~, DH(9), DJ(3), ~~EF(1)~~, EH(4)

6th Edge:

Visited Vertices = { A, B, D, C, E, F, G

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~, ~~DC(1)~~,

~~DE(2)~~, DH(9), DJ(3), ~~EF(1)~~, EH(4), ~~FG(2)~~

7th Edge:

* Here, even though we have GH edge, we are not going to select it as the weight is higher than the other edges available. Therefore, we select DJ as the 7th edge which has a minimum weight of 3.

Visited Vertices = { A, B, D, C, E, F, G, H, J

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~, ~~DC(1)~~,

~~DE(2)~~, DH(9), DJ(3), ~~EF(1)~~, EH(4), ~~FG(2)~~, GH(9)

8th Edge:

* Here we can see that we have two edges with the same weight; they are JI and EH. We cannot take BE since it makes a cycle. We can select either edge from JI and EH. Here, we will select JI.

Visited Vertices = { A, B, D, C, E, F, G, H, J,I

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~, ~~DC(1)~~,

~~DE(2)~~, DH(9), DJ(3), ~~EF(1)~~, EH(4), ~~FG(2)~~, GH(9), ~~JI(4)~~

9th Edge:

* We have written IH, but we cannot select it as the 9th edge which is because we another edge that has lesser weight than the IH; that is EH with a weight of 4.

Visited Vertices = { A, B, D, C, E, F, G, H, J,I,H

Edges to choose from = { ~~AB (1)~~, AC(7), BE (4), BC (5), ~~BD(3)~~, ~~DC(1)~~,

~~DE(2)~~, DH(9), DJ(3), ~~EF(1)~~, EH(4), ~~FG(2)~~, GH(9), ~~JI(4)~~, IH(7)

We have drawn a minimum spanning tree as per Prim’s Algorithm. The below is the final output.

A

B

C

D

J

H

I

F

G

3

4

4

1

2

1

1

E

3

2

A

4

B

C

D

J

H

I

F

G

7

7

3

4

4

1

2

1

5

1

E

3

2

9

9

# Kruskal’ Algorithm

A

B

C

D

E

F

2

1

5

3

3

6

8

1

4

2

9

Step 1: Remove loops and parallel edges while keeping minimum weight. When removing loops, you do not have to think about the weight.

A

B

C

D

E

F

2

1

5

3

3

6

8

1

4

2

9

* Here we will be removing the loop at F directly.
* Also we will be removing the curve or the

parallel line at CE considering the minimum

weight.

Step 2: List all the edges and sort them according to the weights. (Ascending order)

A

B

C

D

E

F

2

1

5

3

3

6

8

1

4

|  |  |
| --- | --- |
| Edge | Weight |
| AB | 2 |
| AC | 1 |
| BC | 3 |
| BD | 5 |
| DC | 3 |
| DE | 8 |
| DF | 4 |
| FE | 1 |
| EC | 6 |

|  |  |
| --- | --- |
| Ascending Order | |
| Edge | Weight |
| AC | 1 |
| FE | 1 |
| AB | 2 |
| BC | 3 |
| DC | 3 |
| DF | 4 |
| BD | 5 |
| EC | 6 |
| DE | 8 |

Step 3: Take n-1 edges from the sorted list while skipping cycle making edges.

* In this graph, minimum number of edges is 6-1 = 5.
* When drawing the graph according to the kruskal’s algorithm, the edges are disconnected during the process, but will be connected once complete.
* The order of drawing the graph are numbered in red circles.

A

B

C

D

E

F

2

1

5

3

3

6

8

1

4

|  |  |
| --- | --- |
| Ascending Order | |
| Edge | Weight |
| AC | 1 |
| FE | 1 |
| AB | 2 |
| BC | 3 |
| DC | 3 |
| DF | 4 |
| BD | 5 |
| EC | 6 |
| DE | 8 |

This is the final minimum spanning tree that we get with Kruskal’s Algorithm

A

B

C

D

E

F

Skipped as it makes a cycle

Example 2:

A

4

B

C

D

J

H

I

F

G

9

7

7

3

4

4

1

2

2

1

5

1

E

3

2

9

9

* Let’s remove parallel lines and loops and do the listing

A

4

B

C

D

J

H

I

F

G

7

7

3

4

4

1

2

1

5

1

E

3

2

9

9

|  |  |
| --- | --- |
| Edge | Weight |
| AB | 1 |
| AC | 7 |
| BC | 5 |
| BE | 4 |
| BD | 3 |
| CD | 1 |
| DJ | 3 |
| DH | 9 |
| DE | 2 |
| EH | 4 |
| EF | 1 |
| FG | 2 |
| GH | 9 |
| HI | 7 |
| IJ | 4 |

|  |  |
| --- | --- |
| Ascending Order | |
| Edge | Weight |
| AB | 1 |
| CD | 1 |
| EF | 1 |
| DE | 2 |
| FG | 2 |
| BD | 3 |
| DJ | 3 |
| BE | 4 |
| EH | 4 |
| IJ | 4 |
| BC | 5 |
| AC | 7 |
| HI | 7 |
| DH | 9 |
| GH | 9 |

* Let’s draw the spanning tree (minimum edges = 10-1=9)

A

4

B

C

D

J

H

I

F

G

7

7

3

4

4

1

2

1

5

1

E

3

2

9

9

|  |  |
| --- | --- |
| Ascending Order | |
| Edge | Weight |
| AB | 1 |
| CD | 1 |
| EF | 1 |
| DE | 2 |
| FG | 2 |
| BD | 3 |
| DJ | 3 |
| BE | 4 |
| EH | 4 |
| IJ | 4 |
| BC | 5 |
| AC | 7 |
| HI | 7 |
| DH | 9 |
| GH | 9 |

This is the final minimum spanning tree that we get with Kruskal’s Algorithm

A

B

H

E

D

C

J

I

F

G

Skipped as it makes a cycle

# Comparing the same graph with Prim’s Algorithm and Kruskal’s Algorithm

Kruskal’s Algorithm

Prim’s Algorithm

A

B

H

E

D

C

J

I

F

G

A

B

C

D

J

H

I

F

G

E

As visible above, both the graphs are same. Compared to the Kruskal’s Agorithm, it takes much time to draw the minimum spanning according to the Prim’s Algorithm.

The total weight for this minimum spanning tree is 21 which is the same in any algorithm.

# Application of Prim’s Algorithm in Python

Example:

0

1

8

7

2

6

5

3

4

4

8

8

11

6

7

2

1

2

4

7

14

10

9

* A Python program for Prim's

Minimum Spanning Tree (MST) algorithm.

* The program is for adjacency

matrix representation of the graph.

import sys # Library for INT\_MAX

class Graph():

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = [[0 for column in range(vertices)]

for row in range(vertices)]

# A utility function to print the constructed MST stored in parent[]

def printMST(self, parent):

print "Edge \tWeight"

for i in range(1, self.V):

print parent[i], "-", i, "\t", self.graph[i][ parent[i] ]

# A utility function to find the vertex with

# minimum distance value, from the set of vertices

# not yet included in shortest path tree

def minKey(self, key, mstSet):

# Initilaize min value

min = sys.maxint

for v in range(self.V):

if key[v] < min and mstSet[v] == False:

min = key[v]

min\_index = v

return min\_index

# Function to construct and print MST for a graph

# represented using adjacency matrix representation

def primMST(self):

# Key values used to pick minimum weight edge in cut

key = [sys.maxint] \* self.V

parent = [None] \* self.V # Array to store constructed MST

# Make key 0 so that this vertex is picked as first vertex

key[0] = 0

mstSet = [False] \* self.V

parent[0] = -1 # First node is always the root of

for cout in range(self.V):

# Pick the minimum distance vertex from

# the set of vertices not yet processed.

# u is always equal to src in first iteration

u = self.minKey(key, mstSet)

# Put the minimum distance vertex in

# the shortest path tree

mstSet[u] = True

# Update dist value of the adjacent vertices

# of the picked vertex only if the current

# distance is greater than new distance and

# the vertex in not in the shotest path tree

for v in range(self.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 self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:

key[v] = self.graph[u][v]

parent[v] = u

self.printMST(parent)

g = Graph(5)

g.graph = [ [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]]

g.primMST();

# Application of Kruskal’s Algorithm in Python

Example:

0

1

8

7

2

6

5

3

4

4

8

8

11

6

7

2

1

2

4

7

14

10

9

* Python program for Kruskal's algorithm to find
* Minimum Spanning Tree of a given connected,
* Undirected and weighted graph

from collections import defaultdict

#Class to represent a graph

class Graph:

def \_\_init\_\_(self,vertices):

self.V= vertices #No. of vertices

self.graph = [] # default dictionary

# to store graph

# function to add an edge to graph

def addEdge(self,u,v,w):

self.graph.append([u,v,w])

# A utility function to find set of an element i

# (uses path compression technique)

def find(self, parent, i):

if parent[i] == i:

return i

return self.find(parent, parent[i])

# A function that does union of two sets of x and y

# (uses union by rank)

def union(self, parent, rank, x, y):

xroot = self.find(parent, x)

yroot = self.find(parent, y)

# Attach smaller rank tree under root of

# high rank tree (Union by Rank)

if rank[xroot] < rank[yroot]:

parent[xroot] = yroot

elif rank[xroot] > rank[yroot]:

parent[yroot] = xroot

# If ranks are same, then make one as root

# and increment its rank by one

else :

parent[yroot] = xroot

rank[xroot] += 1

# The main function to construct MST using Kruskal's

# algorithm

def KruskalMST(self):

result =[] #This will store the resultant MST

i = 0 # An index variable, used for sorted edges

e = 0 # An index variable, used for result[]

# 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 graph

self.graph = sorted(self.graph,key=lambda item: item[2])

parent = [] ; rank = []

# Create V subsets with single elements

for node in range(self.V):

parent.append(node)

rank.append(0)

# Number of edges to be taken is equal to V-1

while e < self.V -1 :

# Step 2: Pick the smallest edge and increment

# the index for next iteration

u,v,w = self.graph[i]

i = i + 1

x = self.find(parent, u)

y = self.find(parent ,v)

# If including this edge does't cause cycle,

# include it in result and increment the index

# of result for next edge

if x != y:

e = e + 1

result.append([u,v,w])

self.union(parent, rank, x, y)

# Else discard the edge

# print the contents of result[] to display the built MST

print "Following are the edges in the constructed MST"

for u,v,weight in result:

#print str(u) + " -- " + str(v) + " == " + str(weight)

print ("%d -- %d == %d" % (u,v,weight))

# Driver code

g = Graph(4)

g.addEdge(0, 1, 10)

g.addEdge(0, 2, 6)

g.addEdge(0, 3, 5)

g.addEdge(1, 3, 15)

g.addEdge(2, 3, 4)

g.KruskalMST()

# References

[1] Kruskal’s Minimum Spanning Tree Algorithm | Greedy Algo-2 - GeeksforGeeks. 2020. *GeeksforGeeks*. https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/.

[2] Prim’s Minimum Spanning Tree (MST) | Greedy Algo-5 - GeeksforGeeks. 2020. *GeeksforGeeks*. https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/.

[3] Spanning Tree and Minimum Spanning Tree. 2020. *Programiz.com*. https://www.programiz.com/dsa/spanning-tree-and-minimum-spanning-tree?fbclid=IwAR0zZR\_r1-hmlmS0zqmMqE0\_bx8JHp--FZvm1vpDpuHhi6RTvDMTVuKlw3Y.