

# Minimum Spanning Tree

---

## Introduction

---

### Application

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Reducing data storage in sequencing amino acids in a protein.
- Model locality of particle interactions in turbulent fluid flows.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, computer, road).

## Greedy Algorithm

---

### Cut Property

### Step

- start with all edges colored gray
- find cut with no black crossing edges; colors its min-weight edge black
- repeat until  $V-1$  edges are colored black

## Edge-weighted graph API

---

### Weighted Edge

#### API

```
public class Edge implements Comparable<Edge>
    Edge(int v, int w, double weight)
    double weight()
    int either()
    int other(int v)
    int compareTo(Edge that)
    String toString()
```

---

*initializing constructor*  
*weight of this edge*  
*either of this edge's vertices*  
*the other vertex*  
*compare this edge to e*  
*string representation*

## Implementation

```
1  import java.util.NoSuchElementException;
2
3  public class Edge implements Comparable<Edge> {
4
5      private final int v;
6      private final int w;
7      private final double weight;
8
9      public Edge(int v, int w, double weight) {
10         if (v < 0) throw new IllegalArgumentException("vertex index must be a
non-negative integer");
11         if (w < 0) throw new IllegalArgumentException("vertex index must be a
non-negative integer");
12         if (Double.isNaN(weight)) throw new IllegalArgumentException("weight is
NaN");
13         this.v = v;
14         this.w = w;
15         this.weight = weight;
16     }
17
18     public double weight() {
19         return weight;
20     }
21
22     public int either() {
23         return v;
24     }
25
26     public int other(int vertex) {
27         if (vertex == v) return w;
28         else if (vertex == w) return v;
29         else throw new IllegalArgumentException("Illegal endpoint");
30     }
31
32     @Override
33     public int compareTo(Edge that) {
34         return Double.compare(this.weight, that.weight);
35     }
36
37     public String toString() {
38         return String.format("%d-%d %.5f", v, w, weight);
39     }
40
41     public static void main(String[] args) {
42         Edge e = new Edge(12, 34, 5.67);
43         StdOut.println(e);
44     }
45 }
```

# Edge Weighted Graph

## API

<code>public class</code>	<code>EdgeWeightedGraph</code>	
	<code>EdgeWeightedGraph(int V)</code>	<i>create an empty V-vertex graph</i>
	<code>EdgeWeightedGraph(In in)</code>	<i>read graph from input stream</i>
<code>int</code>	<code>V()</code>	<i>number of vertices</i>
<code>int</code>	<code>E()</code>	<i>number of edges</i>
<code>void</code>	<code>addEdge(Edge e)</code>	<i>add edge e to this graph</i>
<code>Iterable&lt;Edge&gt;</code>	<code>adj(int v)</code>	<i>edges incident to v</i>
<code>Iterable&lt;Edge&gt;</code>	<code>edges()</code>	<i>all of this graph's edges</i>
<code>String</code>	<code>toString()</code>	<i>string representation</i>

## Adjacency-lists representation

```
1 import java.util.NoSuchElementException;
2
3 public class EdgeWeightedGraph {
4     private static final String NEWLINE =
5         System.getProperty("line.separator");
6
7     private final int V;
8     private int E;
9     private Bag<Edge>[] adj;
10
11     public EdgeWeightedGraph(int V) {
12         if (V < 0) throw new IllegalArgumentException("Number of vertices must
13             be non-negative");
14         this.V = V;
15         this.E = 0;
16         adj = (Bag<Edge>[]) new Bag[V];
17         for (int v = 0; v < V; v++) {
18             adj[v] = new Bag<Edge>();
19         }
20     }
21
22     public EdgeWeightedGraph(int V, int E) {
23         this(V);
24         if (E < 0) throw new IllegalArgumentException("Number of edges must be
25             non-negative");
26         for (int i = 0; i < E; i++) {
27             int v = StdRandom.uniformInt(V);
28             int w = StdRandom.uniformInt(V);
29             double weight = 0.01 * StdRandom.uniformInt(0, 100);
30             Edge e = new Edge(v, w, weight);
31             addEdge(e);
32         }
33     }
34 }
```

```

31
32     public EdgeweightedGraph(In in) {
33         if (in == null) throw new IllegalArgumentException("argument is
null");
34
35         try {
36             v = in.readInt();
37             adj = (Bag<Edge>[]) new Bag[V];
38             for (int v = 0; v < V; v++) {
39                 adj[v] = new Bag<Edge>();
40             }
41
42             int E = in.readInt();
43             if (E < 0) throw new IllegalArgumentException("Number of edges
must be non-negative");
44             for (int i = 0; i < E; i++) {
45                 int v = in.readInt();
46                 int w = in.readInt();
47                 validateVertex(v);
48                 validateVertex(w);
49                 double weight = in.readDouble();
50                 Edge e = new Edge(v, w, weight);
51                 addEdge(e);
52             }
53         }
54         catch (NoSuchElementException e) {
55             throw new IllegalArgumentException("invalid input format in
EdgeweightedGraph constructor", e);
56         }
57
58     }
59
60     public EdgeweightedGraph(EdgeweightedGraph G) {
61         this(G.V());
62         this.E = G.E();
63         for (int v = 0; v < G.V(); v++) {
64             // reverse so that adjacency list is in same order as original
65             Stack<Edge> reverse = new Stack<Edge>();
66             for (Edge e : G.adj[v]) {
67                 reverse.push(e);
68             }
69             for (Edge e : reverse) {
70                 adj[v].add(e);
71             }
72         }
73     }
74
75     public int V() {
76         return V;
77     }
78
79     public int E() {

```

```

80         return E;
81     }
82
83     // throw an IllegalArgumentException unless {@code 0 <= v < V}
84     private void validateVertex(int v) {
85         if (v < 0 || v >= V)
86             throw new IllegalArgumentException("vertex " + v + " is not
between 0 and " + (V-1));
87     }
88
89     public void addEdge(Edge e) {
90         int v = e.either();
91         int w = e.other(v);
92         validateVertex(v);
93         validateVertex(w);
94         adj[v].add(e);
95         adj[w].add(e);
96         E++;
97     }
98
99     public Iterable<Edge> adj(int v) {
100         validateVertex(v);
101         return adj[v];
102     }
103
104     public int degree(int v) {
105         validateVertex(v);
106         return adj[v].size();
107     }
108
109     public Iterable<Edge> edges() {
110         Bag<Edge> list = new Bag<Edge>();
111         for (int v = 0; v < V; v++) {
112             int selfLoops = 0;
113             for (Edge e : adj(v)) {
114                 if (e.other(v) > v) {
115                     list.add(e);
116                 }
117                 // add only one copy of each self loop (self loops will be
consecutive)
118                 else if (e.other(v) == v) {
119                     if (selfLoops % 2 == 0) list.add(e);
120                     selfLoops++;
121                 }
122             }
123         }
124         return list;
125     }
126
127     public String toString() {
128         StringBuilder s = new StringBuilder();
129         s.append(V + " " + E + NEWLINE);

```

```

130         for (int v = 0; v < V; v++) {
131             s.append(v + ": ");
132             for (Edge e : adj[v]) {
133                 s.append(e + " ");
134             }
135             s.append(NEWLINE);
136         }
137         return s.toString();
138     }
139
140     public static void main(String[] args) {
141         In in = new In(args[0]);
142         EdgeweightedGraph G = new EdgeweightedGraph(in);
143         Stdout.println(G);
144     }
145 }

```

## Kruskal's Algorithm

*ElogV*

### Basic

- consider edges in ascending order of weight
- add next edge to tree  $T$  unless doing so would create a cycle

### Implementation

```

1  import java.util.Arrays;
2
3  public class KruskalMST {
4      private static final double FLOATING_POINT_EPSILON = 1.0E-12;
5
6      private double weight; // weight of MST
7      private Queue<Edge> mst = new Queue<Edge>(); // edges in MST
8
9      public KruskalMST(EdgeweightedGraph G) {
10
11         // create array of edges, sorted by weight
12         Edge[] edges = new Edge[G.E()];
13         int t = 0;
14         for (Edge e: G.edges()) {
15             edges[t++] = e;
16         }
17         Arrays.sort(edges);
18
19         // run greedy algorithm
20         UF uf = new UF(G.V());
21         for (int i = 0; i < G.E() && mst.size() < G.V() - 1; i++) {
22             Edge e = edges[i];
23             int v = e.either();

```

```

24         int w = e.other(v);
25
26         // v-w does not create a cycle
27         if (uf.find(v) != uf.find(w)) {
28             uf.union(v, w);    // merge v and w components
29             mst.enqueue(e);    // add edge e to mst
30             weight += e.weight();
31         }
32     }
33
34     // check optimality conditions
35     assert check(G);
36 }
37
38 public Iterable<Edge> edges() {
39     return mst;
40 }
41
42 public double weight() {
43     return weight;
44 }
45
46 // check optimality conditions (takes time proportional to E V lg* V)
47 private boolean check(EdgeweightedGraph G) {
48
49     // check total weight
50     double total = 0.0;
51     for (Edge e : edges()) {
52         total += e.weight();
53     }
54     if (Math.abs(total - weight()) > FLOATING_POINT_EPSILON) {
55         System.err.printf("Weight of edges does not equal weight(): %f vs.
56 %f\n", total, weight());
57         return false;
58     }
59
60     // check that it is acyclic
61     UF uf = new UF(G.V());
62     for (Edge e : edges()) {
63         int v = e.either(), w = e.other(v);
64         if (uf.find(v) == uf.find(w)) {
65             System.err.println("Not a forest");
66             return false;
67         }
68         uf.union(v, w);
69     }
70
71     // check that it is a spanning forest
72     for (Edge e : G.edges()) {
73         int v = e.either(), w = e.other(v);
74         if (uf.find(v) != uf.find(w)) {
75             System.err.println("Not a spanning forest");

```

```

75         return false;
76     }
77 }
78
79 // check that it is a minimal spanning forest (cut optimality
conditions)
80 for (Edge e : edges()) {
81
82     // all edges in MST except e
83     uf = new UF(G.V());
84     for (Edge f : mst) {
85         int x = f.either(), y = f.other(x);
86         if (f != e) uf.union(x, y);
87     }
88
89     // check that e is min weight edge in crossing cut
90     for (Edge f : G.edges()) {
91         int x = f.either(), y = f.other(x);
92         if (uf.find(x) != uf.find(y)) {
93             if (f.weight() < e.weight()) {
94                 System.err.println("Edge " + f + " violates cut
optimality conditions");
95                 return false;
96             }
97         }
98     }
99
100 }
101
102 return true;
103 }
104
105 public static void main(String[] args) {
106     In in = new In(args[0]);
107     EdgeweightedGraph G = new EdgeweightedGraph(in);
108     KruskalMST mst = new KruskalMST(G);
109     for (Edge e : mst.edges()) {
110         StdOut.println(e);
111     }
112     StdOut.printf("%.5f\n", mst.weight());
113 }
114
115 }

```

## Prim's Algorithm



## Basic

- start with vertex 0 and greedily grow tree  $T$
- add to  $T$  the min weight edge with exactly one endpoint in  $T$
- repeat until  $V - 1$  edges

## Lazy Implementation

- Challenge---find the min weight edge with exactly one endpoint in  $T$
- Lazy solution---Maintain a PQ of edges with at least one endpoint in  $T$ 
  - Key = edge; priority = weight of edge
  - Delete-min to determine next edge  $e = v - w$  to add to  $T$
  - Disregard if both endpoints  $v$  and  $w$  are in  $T$
  - Otherwise, let  $w$  be the vertex not in  $T$ 
    - add to PQ any edge incident to  $w$  (assuming other endpoint not in  $T$ )
    - add  $w$  to  $T$
- complexity

operation	frequency	binary heap
delete min	$E$	$\log E$
insert	$E$	$\log E$

```
1 public class LazyPrimMST {
2     private static final double FLOATING_POINT_EPSILON = 1.0E-12;
3
4     private double weight;          // total weight of MST
5     private Queue<Edge> mst;        // edges in the MST
6     private boolean[] marked;      // marked[v] = true iff v on tree
7     private MinPQ<Edge> pq;        // edges with one endpoint in tree
8
9     public LazyPrimMST(EdgeWeightedGraph G) {
10         mst = new Queue<Edge>();
11         pq = new MinPQ<Edge>();
12         marked = new boolean[G.V()];
13         for (int v = 0; v < G.V(); v++) // run Prim from all vertices to
14             if (!marked[v]) prim(G, v); // get a minimum spanning forest
15
16         // check optimality conditions
17         assert check(G);
18     }
19
20     // run Prim's algorithm
21     private void prim(EdgeWeightedGraph G, int s) {
22         scan(G, s);
```

```

23         while (!pq.isEmpty()) {                                // better to stop when
mst has v-1 edges
24             Edge e = pq.delMin();                                // smallest edge on pq
25             int v = e.either(), w = e.other(v);                // two endpoints
26             assert marked[v] || marked[w];
27             if (marked[v] && marked[w]) continue;                // lazy, both v and w
already scanned
28             mst.enqueue(e);                                        // add e to MST
29             weight += e.weight();
30             if (!marked[v]) scan(G, v);                            // v becomes part of
tree
31             if (!marked[w]) scan(G, w);                            // w becomes part of
tree
32         }
33     }
34
35     // add all edges e incident to v onto pq if the other endpoint has not yet
been scanned
36     private void scan(EdgeWeightedGraph G, int v) {
37         assert !marked[v];
38         marked[v] = true;
39         for (Edge e : G.adj(v))
40             if (!marked[e.other(v)]) pq.insert(e);
41     }
42
43     public Iterable<Edge> edges() {
44         return mst;
45     }
46
47     public double weight() {
48         return weight;
49     }
50
51     // check optimality conditions (takes time proportional to E V lg* V)
52     private boolean check(EdgeWeightedGraph G) {
53
54         // check weight
55         double totalWeight = 0.0;
56         for (Edge e : edges()) {
57             totalWeight += e.weight();
58         }
59         if (Math.abs(totalWeight - weight()) > FLOATING_POINT_EPSILON) {
60             System.err.printf("weight of edges does not equal weight(): %f vs.
%f\n", totalWeight, weight());
61             return false;
62         }
63
64         // check that it is acyclic
65         UF uf = new UF(G.V());
66         for (Edge e : edges()) {
67             int v = e.either(), w = e.other(v);
68             if (uf.find(v) == uf.find(w)) {

```

```

69         System.err.println("Not a forest");
70         return false;
71     }
72     uf.union(v, w);
73 }
74
75 // check that it is a spanning forest
76 for (Edge e : G.edges()) {
77     int v = e.either(), w = e.other(v);
78     if (uf.find(v) != uf.find(w)) {
79         System.err.println("Not a spanning forest");
80         return false;
81     }
82 }
83
84 // check that it is a minimal spanning forest (cut optimality
conditions)
85 for (Edge e : edges()) {
86
87     // all edges in MST except e
88     uf = new UF(G.V());
89     for (Edge f : mst) {
90         int x = f.either(), y = f.other(x);
91         if (f != e) uf.union(x, y);
92     }
93
94     // check that e is min weight edge in crossing cut
95     for (Edge f : G.edges()) {
96         int x = f.either(), y = f.other(x);
97         if (uf.find(x) != uf.find(y)) {
98             if (f.weight() < e.weight()) {
99                 System.err.println("Edge " + f + " violates cut
optimality conditions");
100                 return false;
101             }
102         }
103     }
104 }
105
106
107     return true;
108 }
109
110 public static void main(String[] args) {
111     In in = new In(args[0]);
112     EdgeweightedGraph G = new EdgeweightedGraph(in);
113     LazyPrimMST mst = new LazyPrimMST(G);
114     for (Edge e : mst.edges()) {
115         StdOut.println(e);
116     }
117     StdOut.printf("%.5f\n", mst.weight());
118 }

```

```
119
120 }
```

## Eager Implementation

- Challenge---Find min weight edge with exactly one endpoint in  $T$
- Eager Solution---Maintain a PQ of vertices connected by an edge to  $T$ , where priority of vertex  $v$  = weight of shortest edge connecting  $v$  to  $T$ 
  - Delete-min vertex  $v$  and add its associated edge  $e = v - w$  to  $T$
  - Update PQ by considering all edges  $e = v - x$  incident to  $v$ 
    - ignore if  $x$  is already in  $T$
    - add  $x$  to PQ if not already on it
    - decrease priority of  $x$  if  $v - x$  becomes shortest edge connecting to  $T$

```
1 public class PrimMST {
2     private static final double FLOATING_POINT_EPSILON = 1.0E-12;
3
4     private Edge[] edgeTo;          // edgeTo[v] = shortest edge from tree vertex
    to non-tree vertex
5     private double[] distTo;        // distTo[v] = weight of shortest such edge
6     private boolean[] marked;      // marked[v] = true if v on tree, false
    otherwise
7     private IndexMinPQ<Double> pq;
8
9     public PrimMST(EdgeweightedGraph G) {
10         edgeTo = new Edge[G.V()];
11         distTo = new double[G.V()];
12         marked = new boolean[G.V()];
13         pq = new IndexMinPQ<Double>(G.V());
14         for (int v = 0; v < G.V(); v++)
15             distTo[v] = Double.POSITIVE_INFINITY;
16
17         for (int v = 0; v < G.V(); v++)    // run from each vertex to find
18             if (!marked[v]) prim(G, v);    // minimum spanning forest
19
20         // check optimality conditions
21         assert check(G);
22     }
23
24     // run Prim's algorithm in graph G, starting from vertex s
25     private void prim(EdgeweightedGraph G, int s) {
26         distTo[s] = 0.0;
27         pq.insert(s, distTo[s]);
28         while (!pq.isEmpty()) {
29             int v = pq.delMin();
30             scan(G, v);
31         }
32     }
```

```

33
34 // scan vertex v
35 private void scan(EdgeWeightedGraph G, int v) {
36     marked[v] = true;
37     for (Edge e : G.adj(v)) {
38         int w = e.other(v);
39         if (marked[w]) continue; // v-w is obsolete edge
40         if (e.weight() < distTo[w]) {
41             distTo[w] = e.weight();
42             edgeTo[w] = e;
43             if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
44             else pq.insert(w, distTo[w]);
45         }
46     }
47 }
48
49 public Iterable<Edge> edges() {
50     Queue<Edge> mst = new Queue<Edge>();
51     for (int v = 0; v < edgeTo.length; v++) {
52         Edge e = edgeTo[v];
53         if (e != null) {
54             mst.enqueue(e);
55         }
56     }
57     return mst;
58 }
59 }

```

## MST Context

- Does a linear-time MST exists?

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	optimal	Pettie-Ramachandran
20xx	$E$	???

- Single-link clustering algorithm