# 1 Graph Primitives

## 1.1 Graph

A Graph object contains a list of vertices, a list of edges, and an adjacent list. Therefore, each vertex is implicitly assigned an id from 0 to n-1, each edge is implicitly assigned an id from 0 to m-1. For each vertex, the adjacency list contains the id of its neighbors.

### 1.1.1 Read Graph From File: $O(m+n)$

```
String filename = "../../data/testgraph.txt";
Graph graph = GraphReader.Read(filename);
```

### 1.1.2 Insert a Vertex or an Edge: $O(1)$

```
Vertex vertex = new Vertex(vertexId);
graph.InsertVertex(vertex);
Edge edge = new Edge(node1Id, node2Id, edgeId);
graph.InsertEdge(edge);
```

### 1.1.3 Deletion of a Vertex $v$: $O(d(v))$

```
graph.DeleteVertex(vertex);
```

### 1.1.4 Deletion of an Edge: $O(1)$

```
graph.DeleteEdge(vertex1, vertex2);
```

### 1.1.5 Iterate Over Vertices: $O(n)$

```
foreach (Vertex w in G.Vertices.Values) w.Color = 0;
```

### 1.1.6 Iterate Over the Adjacency List: $O(m+n)$

```
foreach (int VertexId in graph.AdjList.Keys){
 foreach (int neighborId in graph.AdjList[VertexId]){
 }
}
```

## 1.2 Tree

Supports the operations as in a graph, with the added support for the parent child relationship.

### 1.2.1 Insert a Directed Edge: $O(1)$

```
Tree tree = new Tree(NumOfNodes,NumOfEdges);
//initialize nodes - each node with an unique Id...
//insertion of an edge ...
Edge e = new Edge(node1Id, node2Id, edgeId);
tree.InsertDirectedEdge(e, parentId, kidId, kidPosition)
```

### 1.2.2 Access Children: $O(1)$

For left and rightmost children, kidposition is set to 0 and Kids.Count - 1. One can access the left and right kids by invoking node.leftKid(), and node.rightKid(), respectively.

```
Node root = tree.RootNode;
Node leftKid = root.leftKid();
```

#### 1.2.3 Iterate Over the Adjacency List: $O(m+n)$

```
foreach (int VertexId in graph.AdjList.Keys){
 foreach (int neighborId in graph.AdjList[VertexId]){
 }
}
```

#### 1.2.4 Check whether two Vertices are Neighbors: $O(1)$

```
graph.ContainsEdge(edge);
graph.AreAdjacent(vertex1, vertex2);
```

# 2 Graph Algorithms

## 2.1 BFS Ordering: $O(m+n)$

```
int [] bfsOrder = VertrexOrdering.BfsOrdering(graph, startVertex);
```

## 2.2 Connected Components: $O(m+n)$

```
List<Graph> components = ConnectedComponents.getAllComponents(G);
```

## 2.3 Minimum Spanning Tree (MST): $O((m+n)\log m)$

# 3 Geometric Algorithms

## 3.1 L1 MST of a Point Set: $O(n \log n)$

# 4 Utilities

## 4.1 Nearest Neighbor

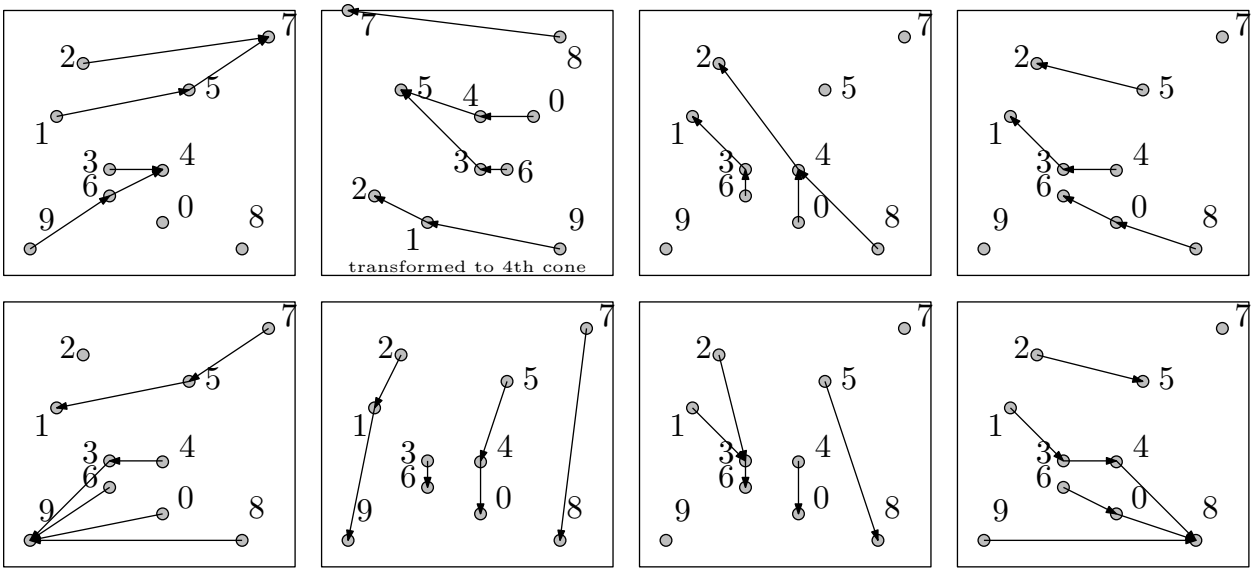


Figure 1: L1 Nearest Neighbor in a Cone

### 4.1.1 L1 Nearest Neighbor in a Cone: $O(n \log n)$

```
Dictionary<int, int> N =
  NearestNeighbor.L1NearestNeighborInCone(P, ConeID);
```

Figure 2 visually illustrates the output of the algorithm, when ConeID $= 1, 2, \ldots, 8$. Uses algorithm [**?**].

## 4.2 Array Search

### 4.2.1 Binary Search: $O(\log n)$

```
double[] A = {.251, .5468, .84245, .84245};
int indexFound;
ArraySearch.BinarySearch(.251, A, out indexFound);
```

### 4.2.2 Binary Search Closest Neighbor: $O(\log n)$

```
int l,r;
ArraySearch.BinarySearchClosest(1.2, A, out l, out r);
```

# 5 Data Structures

## 5.1 Segment Tree

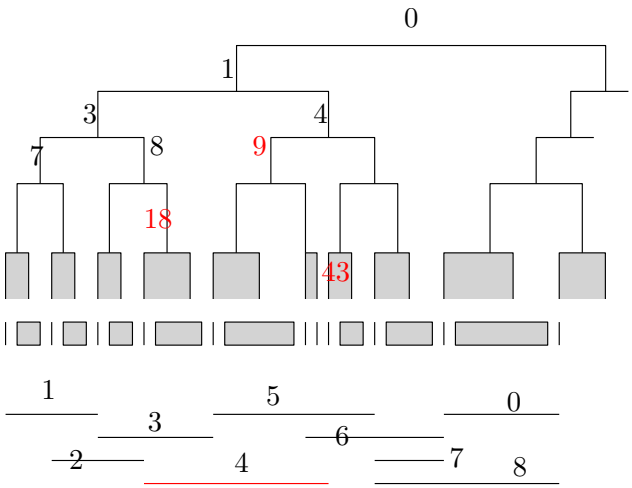


Figure 2: Segment Tree, Segment 4 is inserted in nodes 9, 18, 43

### 5.1.1 Construction: $O(n \log n)$

```
HInterval[] h = new HInterval[100];
//initialize h with unique ids
SegmentTree.Build(h);
```

### 5.1.2 Get all Intersecting Segments $O(\log n)$

```
Point q = new Point(x,y);
List<HInterval> intervals =
 SegmentTree.GetAllIntersectingIntervals(q);
foreach (var interval in intervals)
 Console.Write(interval.Id+",");
```

### 5.1.3 Get the Segment Immediately Above $O(\log^2 n)$

```
Point q = new Point(x,y);
HInterval intv =
 SegmentTree.GetImmediatelyAbove(q);
if (intv.Id > 0) Console.Write(intv.Id);
```