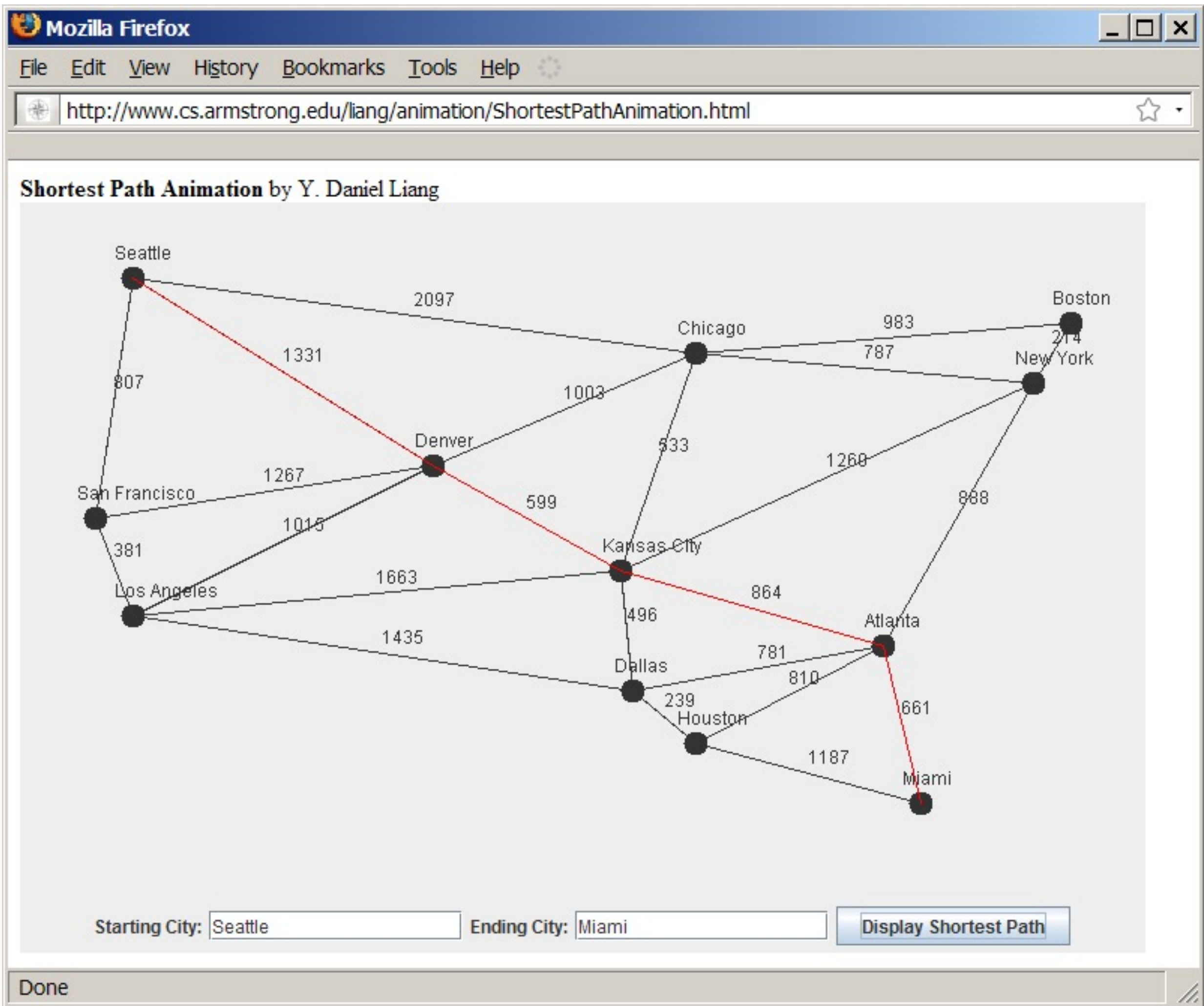
Chapter 23 Weighted Graphs and Applications

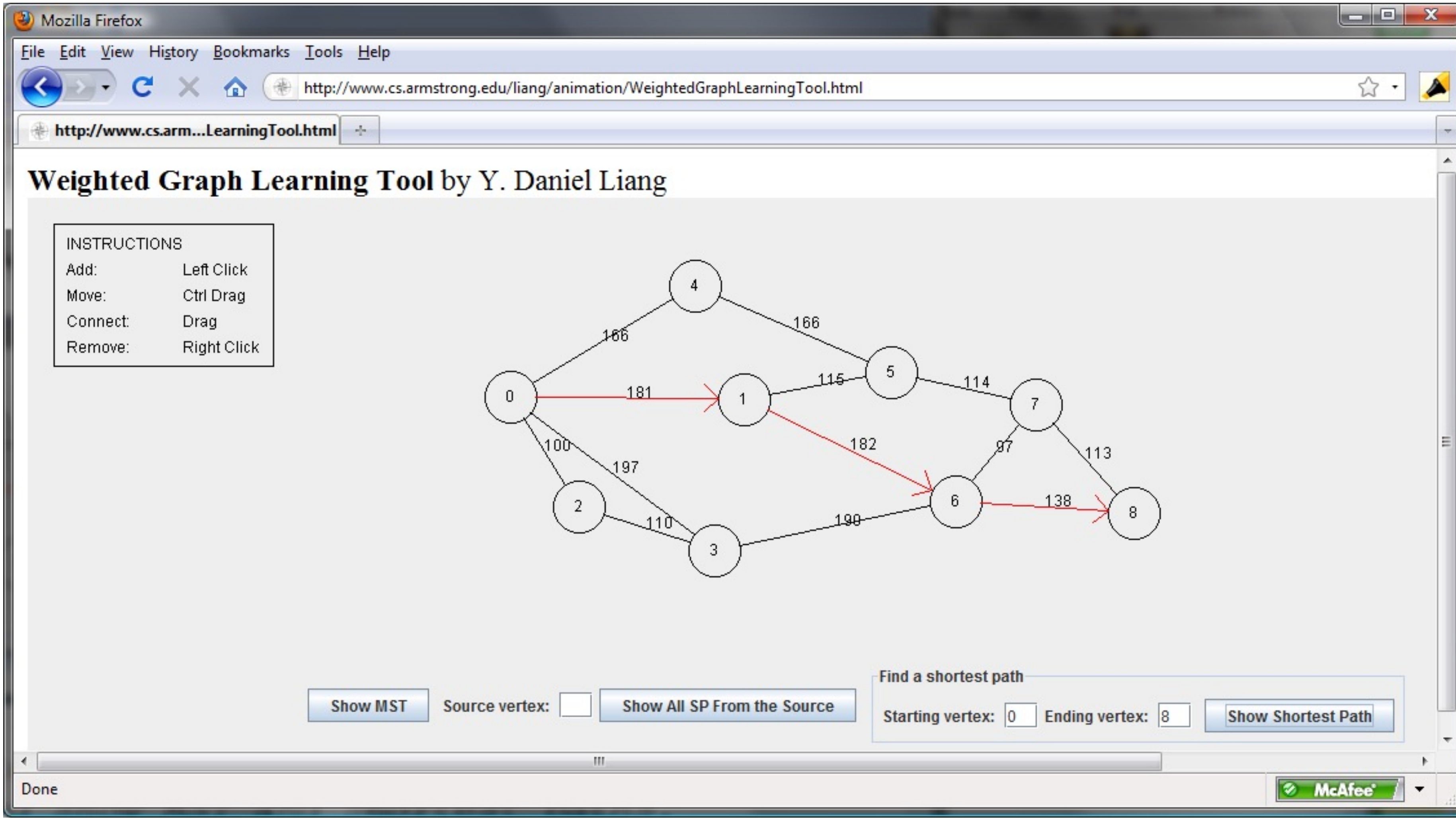
Weighted Graph Animation

www.cs.armstrong.edu/liang/animation/ShortestPathAnimation.html



Weighted Graph Animation

www.cs.armstrong.edu/liang/animation/WeightedGraphLearningTool.html



Objectives

- To represent weighted edges using adjacency matrices and priority queues (§23.2).
- To model weighted graphs using the WeightedGraph class that extends the AbstractGraph class (§23.3).
- To design and implement the algorithm for finding a minimum spanning tree (§23.4).
- To define the MST class that extends the Tree class (§23.4).
- To design and implement the algorithm for finding single-source shortest paths (§23.5).
- To define the ShortestPathTree class that extends the Tree class (§23.5).
- To solve the weighted nine tail problem using the shortest-path algorithm (§23.6).

Representing Weighted Graphs

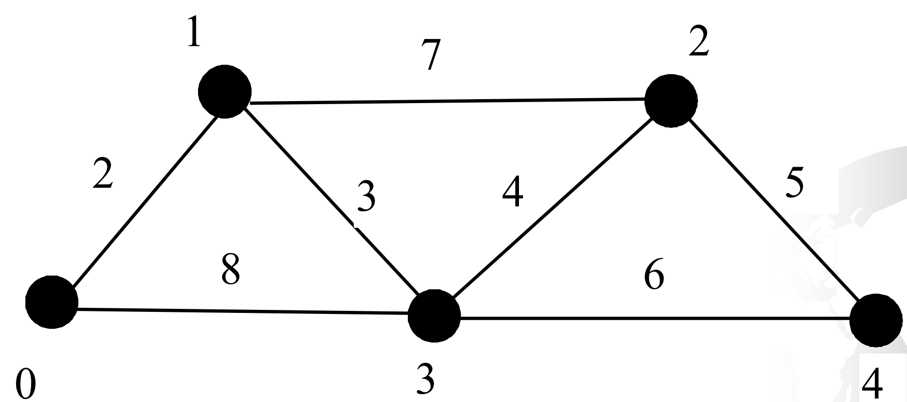
Representing Weighted Edges: Edge List

Weighted Adjacency Matrices

Priority Adjacency Lists

Representing Weighted Edges: Edge Array

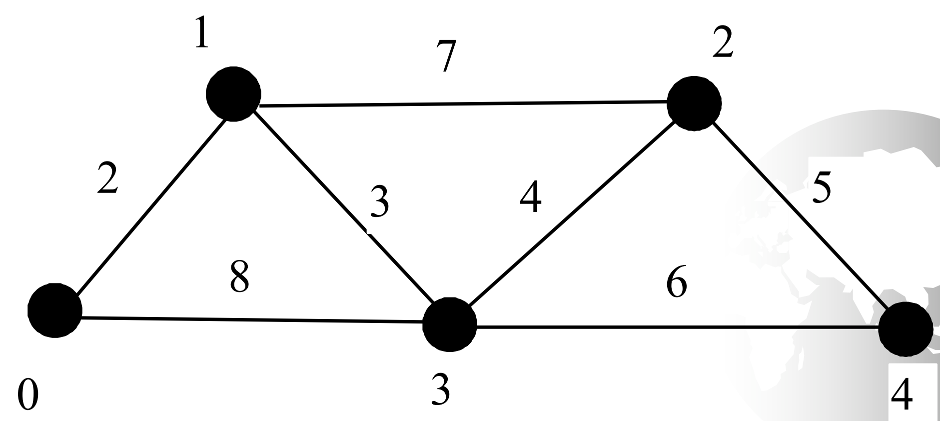
edges = [[0, 1, 2], [0, 3, 8],
  [1, 0, 2], [1, 2, 7], [1, 3, 3],
  [2, 1, 7], [2, 3, 4], [2, 4, 5],
  [3, 0, 8], [3, 1, 3], [3, 2, 4], [3, 4, 6],
  [4, 2, 5], [4, 3, 6]
]



Representing Weighted Edges: Edge Array

```
adjacencyMatrix = [
    [None, 2, None, 8, None],
    [2, None, 7, 3, None],
    [None, 7, None, 4, 5],
    [8, 3, 4, None, 6],
    [None, None, 5, 6, None]
]
```

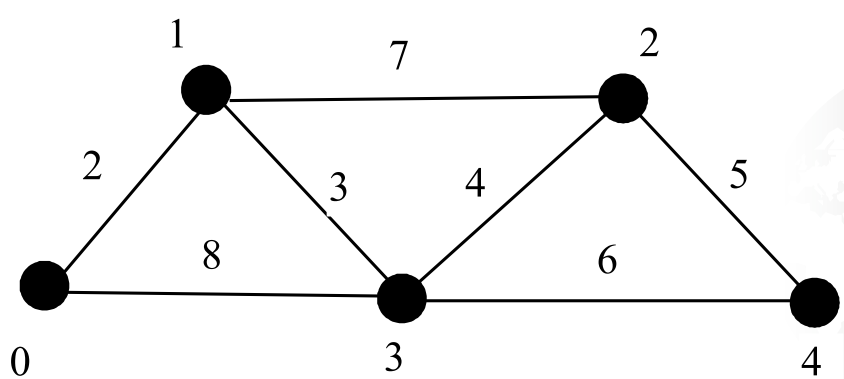
	0	1	2	3	4
0	null	2	null	8	null
1	2	null	7	3	null
2	null	7	null	4	5
3	8	3	4	null	6
4	null	null	5	6	null



Priority Adjacency Lists

queues[0]	WeightedEdge(0, 1, 2)	WeightedEdge(0, 3, 8)		
queues[1]	WeightedEdge(1, 0, 2)	WeightedEdge(1, 3, 3)	WeightedEdge(1, 2, 7)	
queues[2]	WeightedEdge(2, 3, 4)	WeightedEdge(2, 4, 5)	WeightedEdge(2, 1, 7)	
queues[3]	WeightedEdge(3, 1, 3)	WeightedEdge(3, 2, 4)	WeightedEdge(3, 4, 6)	WeightedEdge(3, 0, 8)
queues[4]	WeightedEdge(4, 2, 5)	WeightedEdge(4, 3, 6)		

WeightedEdge



Graph

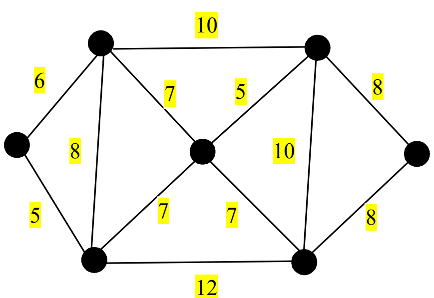
↑

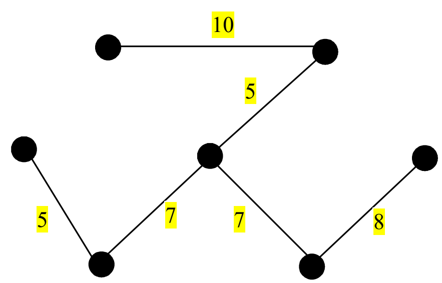
WeightedGraph	
queues: list	queues[i] is a heap that contains all the weighted edges adjacent to vertex i.
WeightedGraph(vertices: list, edges: list)	Constructs a weighted graph with the specified vertices and edges.
getQueueForWeightedEdges(edges): list	Creates a priority queue and returns it.
printWeightedEdges(): void	Displays all edges and weights.
getWeightedEdges(): list	Returns all weighted edges for each vertex in a priority queue.
clear(): void	Removes all vertices and edges from the graph.
addVertex(v: V): void	Adds a vertex to the graph.
addEdge(u: int, v: int, weight: double): void	Adds a weighted edge to the graph.
getMinimumSpanningTree(): MST	Returns a minimum spanning tree starting from vertex 0.
getMinimumSpanningTreeAt(index: int): MST	Returns a minimum spanning tree starting from vertex v.
getShortestPath(index: int): ShortestPathTree	Returns all single-source shortest paths.

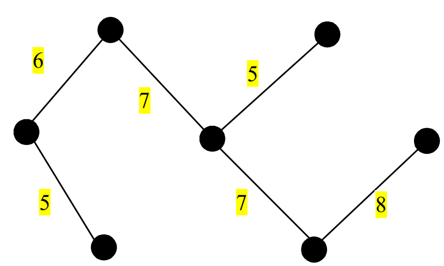
WeightedGraph TestWeightedGraph

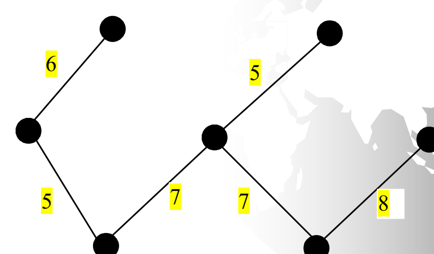
Minimum Spanning Trees

A graph may have many spanning trees. Suppose that the edges are weighted. A minimum spanning tree is a spanning tree with the minimum total weights. For example, the trees in Figures 23.3(b), 23.3(c), 23.3(d) are spanning trees for the graph in Figure 23.3(a). The trees in Figures 23.3(c) and 23.3(d) are minimum spanning trees.



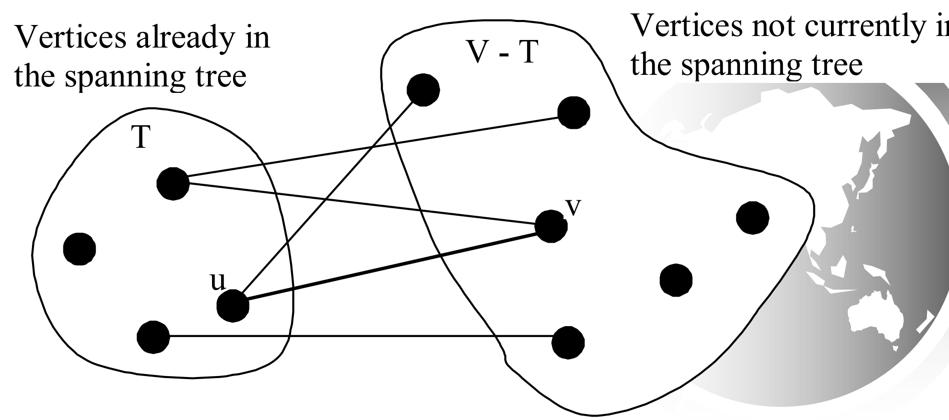




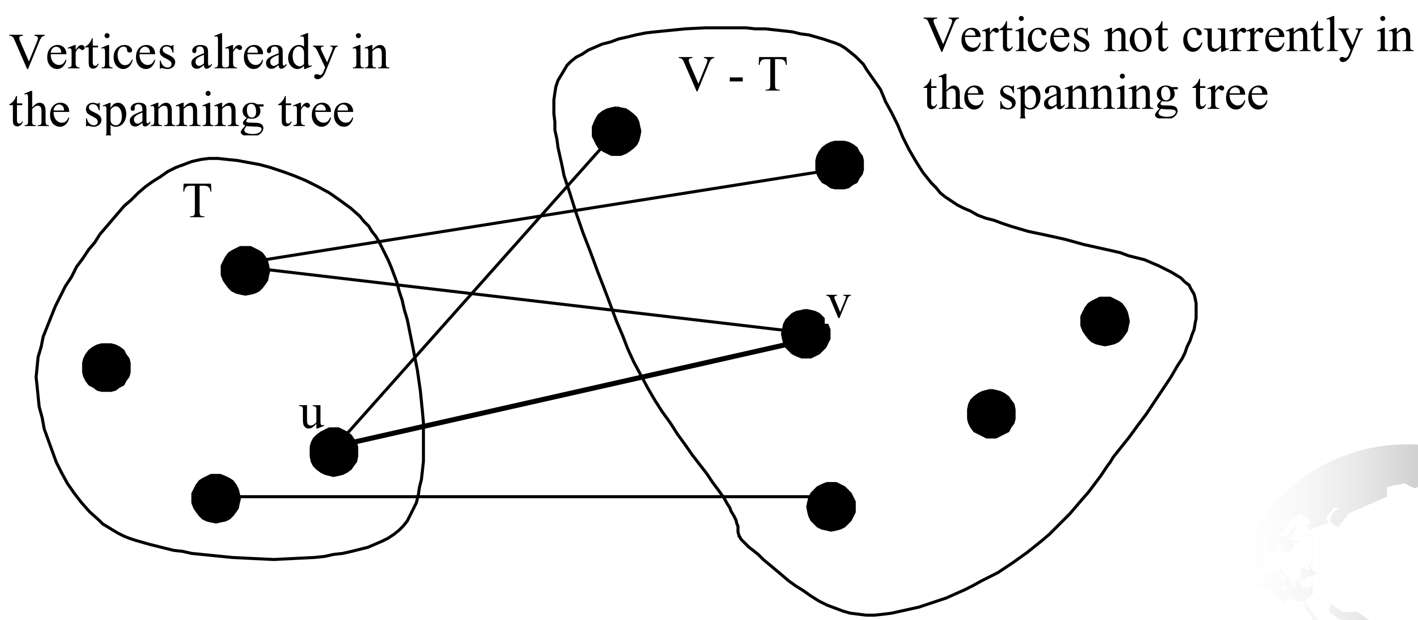


Minimum Spanning Tree Algorithm

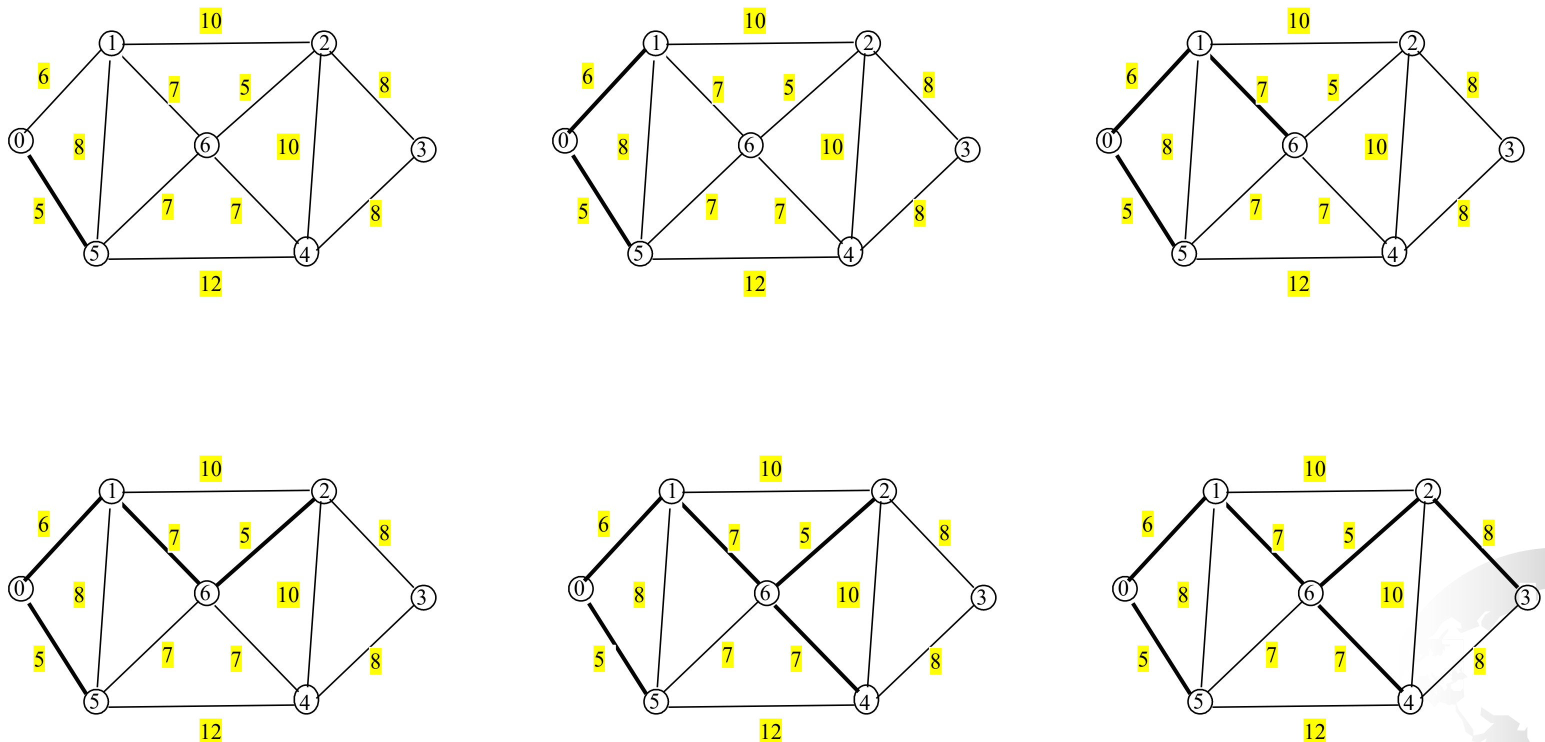
```
def minimumSpanningTree():
    Let V denote the set of vertices in the graph;
    Let T be a set for the vertices in the spanning tree;
    Initially, add the starting vertex to T;
    while size of T < n:
        find u in T and v in V – T with the smallest weight
          on the edge (u, v), as shown in Figure 23.6;
        add v to T;
```



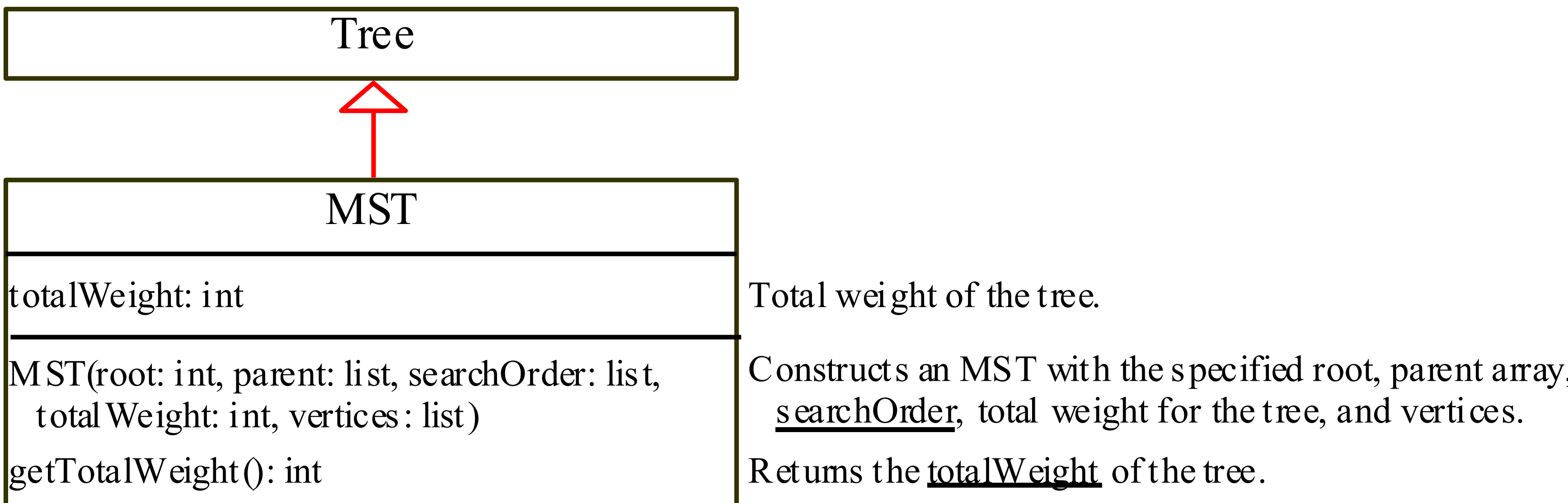
Minimum Spanning Tree Algorithm



Minimum Spanning Tree Algorithm Example



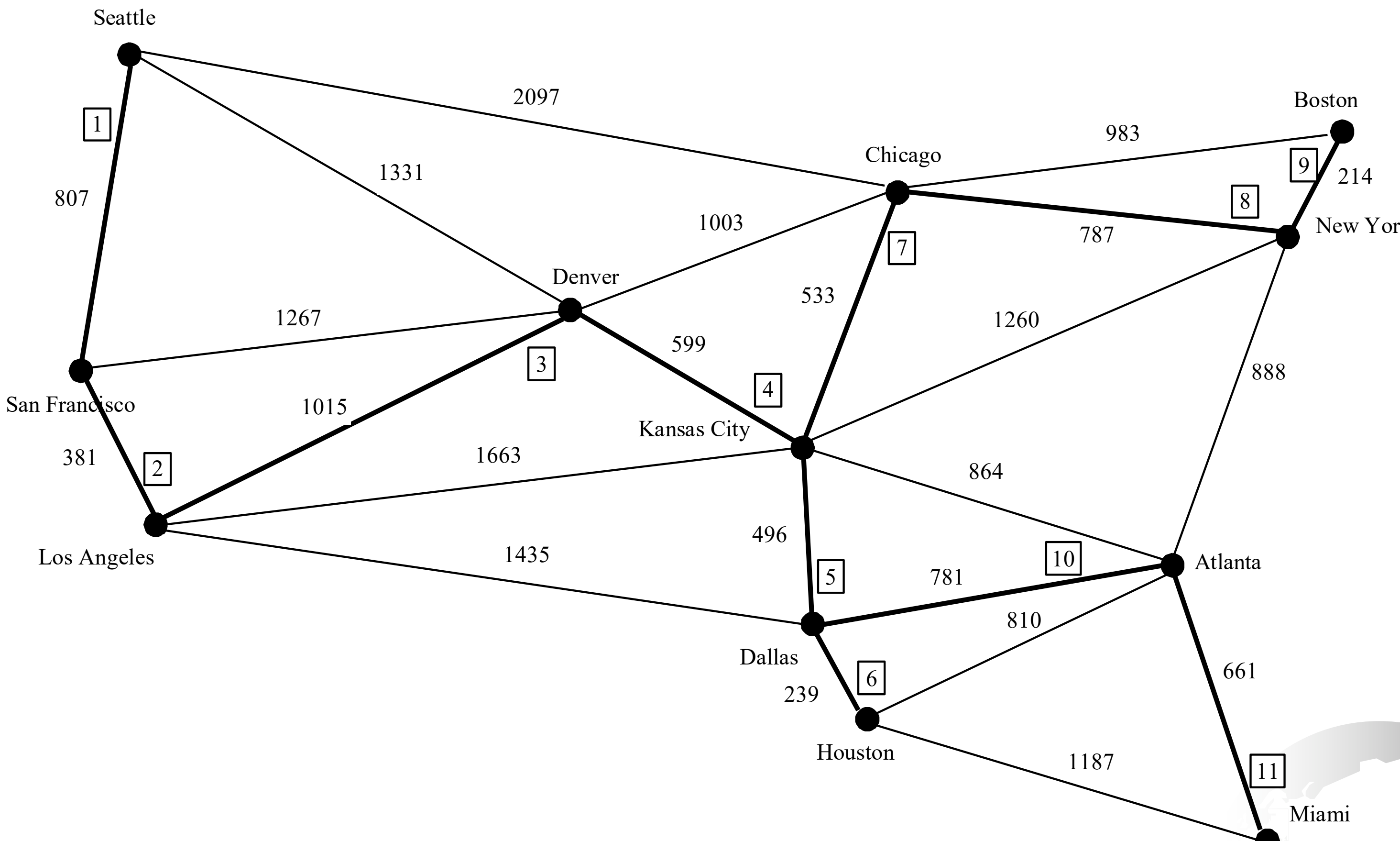
Implementing MST Algorithm



Time Complexity

For each vertex, the program constructs a priority queue for its adjacent edges. It takes O(log|V|) time to insert an edge to a priority queue and the same time to remove an edge from the priority queue. So the overall time complexity for the program is O(|E|log|V|) , where |E| denotes the number of edges and |V| denotes the number of vertices.

Test MST



TestMinimumSpanningTree

Shortest Path

§23.1 introduced the problem of finding the shortest distance between two cities for the graph in Figure 23.1. The answer to this problem is to find a shortest path between two vertices in the graph.

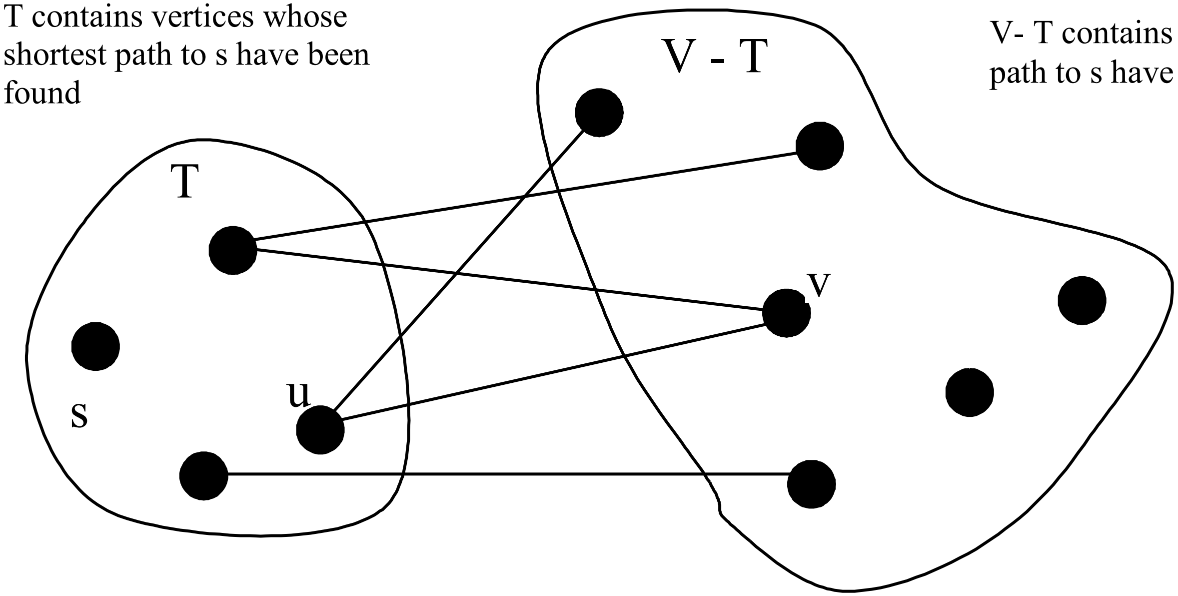
Single Source Shortest Path Algorithm

```
def shortestPath(s):
    Let V denote the set of vertices in the graph;
    Let T be a set that contains the vertices whose
      paths to s have been found
    Initially T contains source vertex s with costs[s] = 0
    while size of T < n:
        find v in V – T with the smallest costs[u] + w(u, v) value
          among all u in T
        add v to T and costs[v] = costs[u] + w(u, v)
```

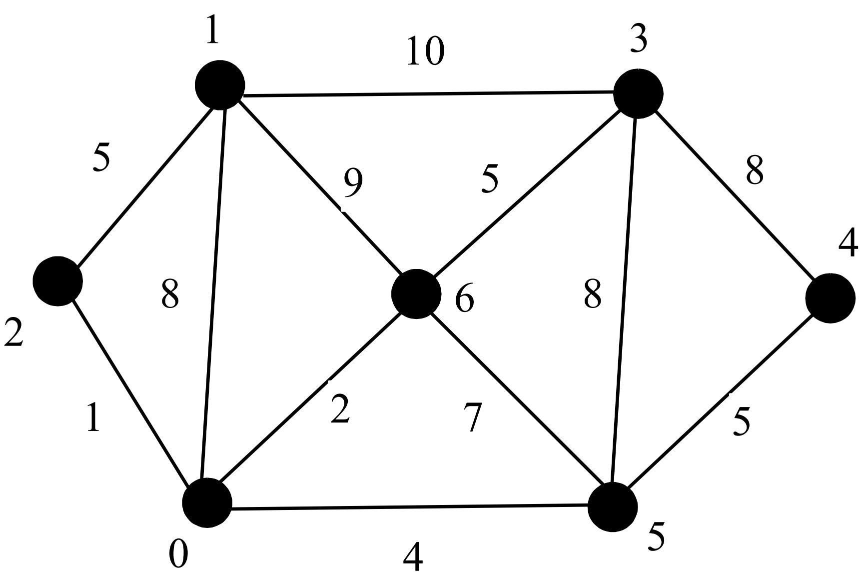
Single Source Shortest Path Algorithm

T contains vertices whose shortest path to s have been found

V- T contains vertices whose shortest path to s have not been found



SP Algorithm Example



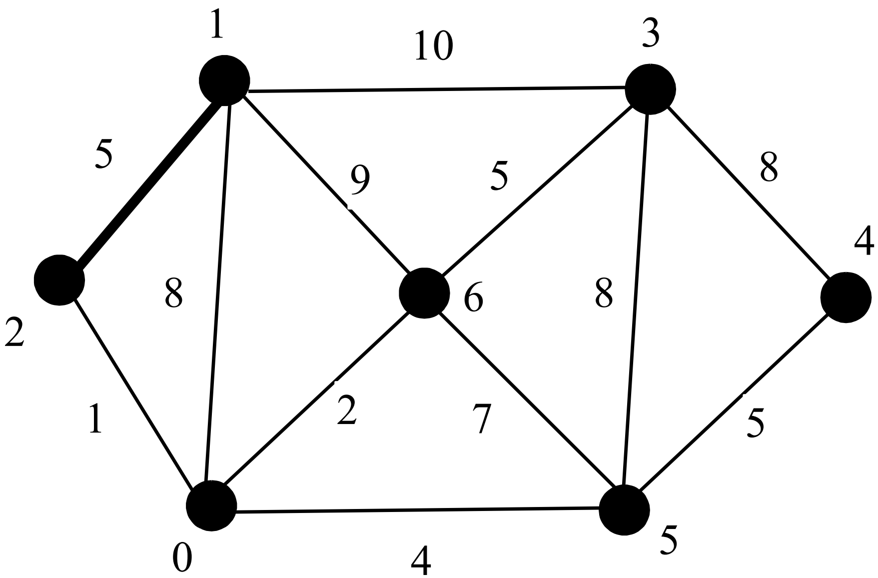
costs

0	1	2	3	4	5	6
∞	0	∞	∞	∞	∞	∞

parent

0	1	2	3	4	5	6
	-1					

SP Algorithm Example



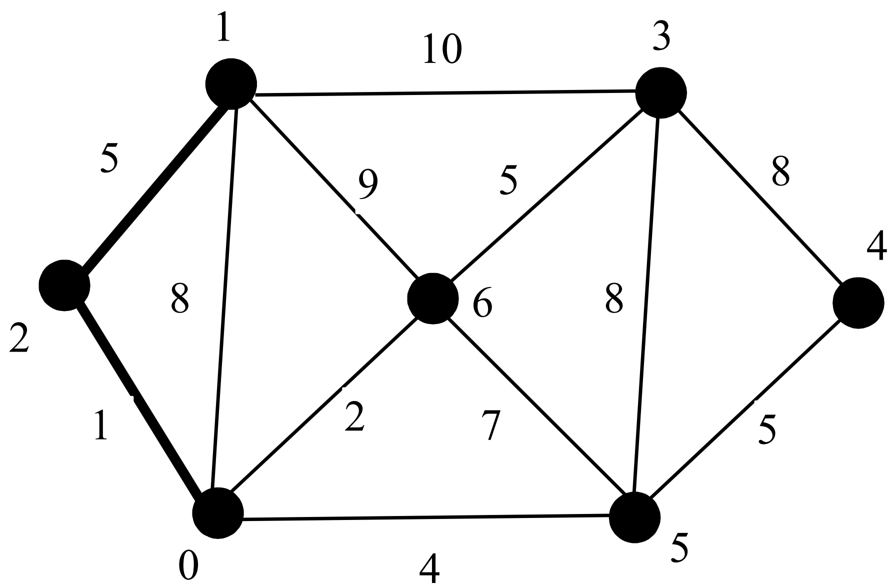
costs

0	1	2	3	4	5	6
∞	0	5	∞	∞	∞	∞

parent

0	1	2	3	4	5	6
	-1	1				

SP Algorithm Example



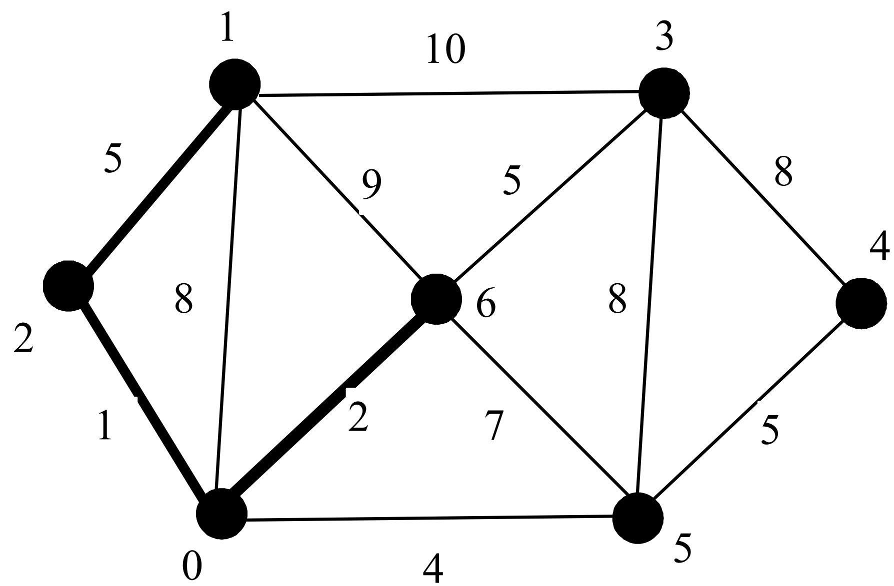
costs

6	0	5	∞	∞	∞	∞
0	1	2	3	4	5	6

parent

2	-1	1				
0	1	2	3	4	5	6

SP Algorithm Example



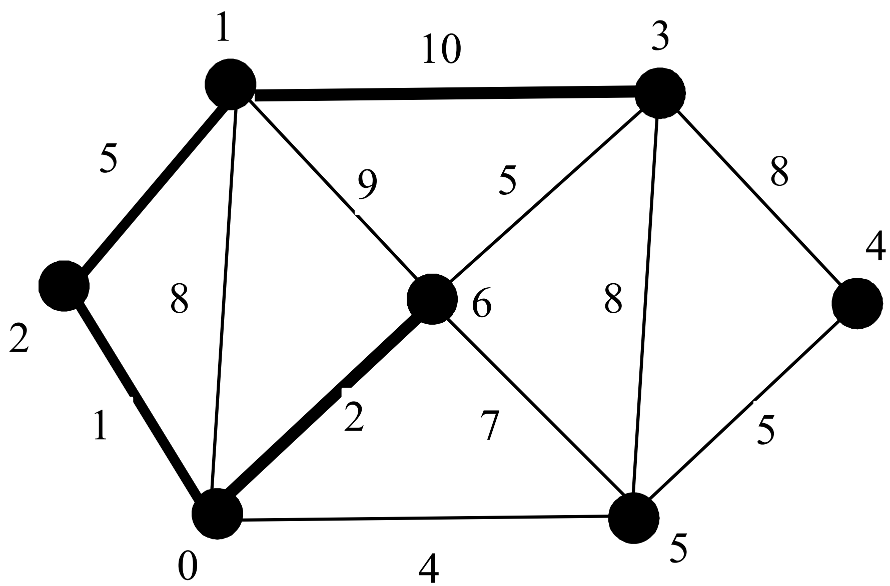
costs

0	1	2	3	4	5	6
6	0	5	∞	∞	∞	8

parent

0	1	2	3	4	5	6
2	-1	1				0

SP Algorithm Example



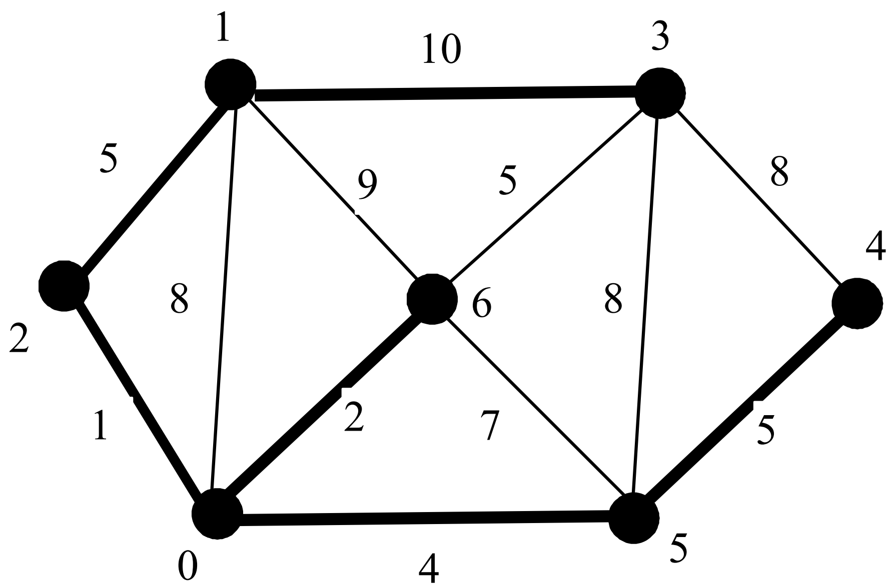
costs

0	1	2	3	4	5	6
6	0	5	10	∞	∞	8

parent

0	1	2	3	4	5	6
2	-1	1	1			0

SP Algorithm Example



costs

6	0	5	10	15	10	8
0	1	2	3	4	5	6

parent

2	-1	1	1	5	0	0
0	1	2	3	4	5	6

SP Algorithm Implementation

Tree

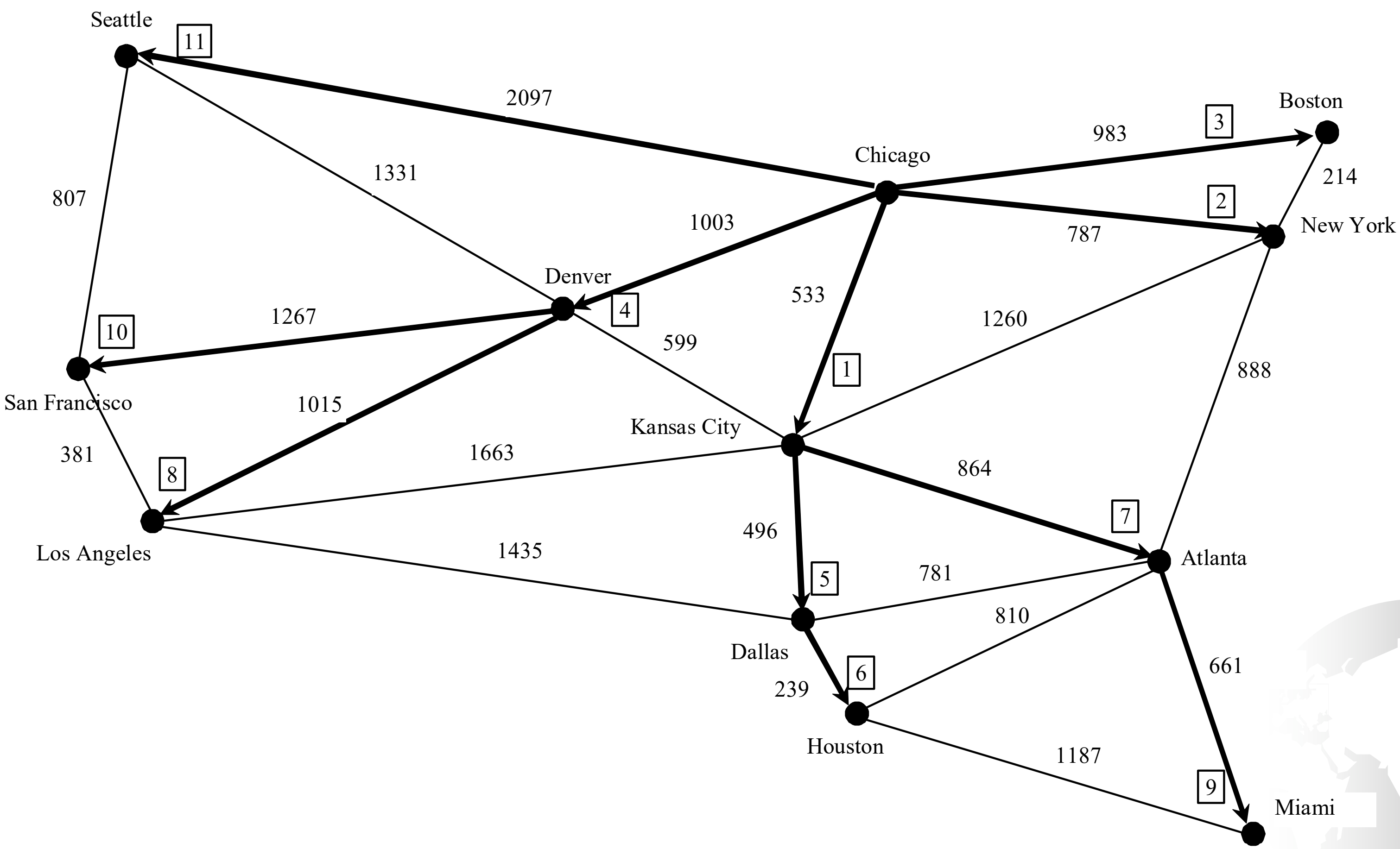
↑

ShortestPathTree

ShortestPathTree	
costs: list	costs[v] stores the cost for the path from the source to v.
ShortestPathTree(source: int, parent: list, searchOrder: list, costs: list, vertices)	Constructs a shortest path tree with the specified source, parent array, and costs array.
getCost(vertexIndex: int): int	Returns the cost for the path from the source to the vertex.
printAllPaths(): void	Displays all paths from the source.

TestShortestPath

SP Algorithm Example



The Weighted Nine Tail Problem

The nine tail problem is to find the minimum number of the moves that lead to all coins face down. Each move flips a head coin and its neighbors. The weighted nine tail problem assigns the number of the flips as a weight on each move. For example, you can move from the coins in Figure (a) to Figure (b) by flipping the three coins. So the weight for this move is <u>3</u>.

H	H	H
T	T	T
H	H	H

T	T	H
H	T	T
H	H	H

WeightedNineTailModel

NineTailModel	
tree: Tree	A tree rooted at node 511.
NineTailModel()	Constructs a model for the nine tail problem and obtains the tree.
getShortestPath(nodeIndex: int): list	Returns a path from the specified node to the root. The path returned consists of the node labels in a list.
getEdges(): list	Returns an edge list for the graph.
getNode(index: int): list	Returns a node consisting of nine characters of H's and T's.
getIndex(node: list): int	Returns the index of the specified node.
getFlippedNode(node: list, position: int): int	Flips the node at the specified position and returns the index of the flipped node.
flipACell(node: list, row: int, column: int): void	Flips the node at the specified row and column.
printNode(node: list): void	Displays the node to the console.

WeightedNineTailModel	
WeightedNineTailModel()	Constructs a model for the weighted nine tail problem and obtains a ShortestPathTree rooted from the target node.
getNumberOfFlipsFrom(u: int): int	Returns the number of flips from node u to the target node 511.
getNumberOfFlips(u: int, v: int): int	Returns the number of different cells between the two nodes.
getWeightedEdges(): list	Creates and return all edges for the graph.

WeightedNineTailModel NineTailModel