# Graph Algoritms

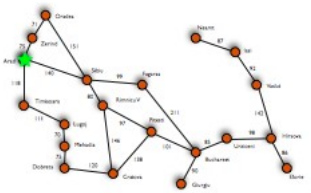
## Graph Traversal

- two algorithms for graph traversal are depth-first search and breath-first search
- depth-first search starts at a node and traverses the graph by pursuing a path as far as it can, then backtracking
- breath-first search starts at a node and visits all immediate children first before proceeding to deeper levels

## Depth-First Search of Romania Graph

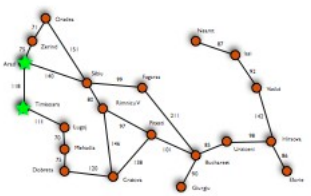
## Depth-First Search of Romania Graph

Arad



## Depth-First Search of Romania Graph

Arad
Timisoara

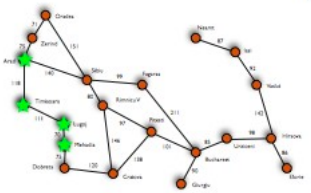


## Depth-First Search of Romania Graph

Arad
Timisoara
Lugoj

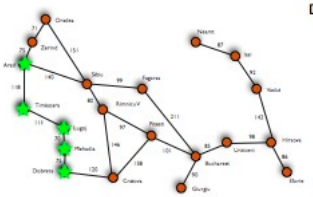
## Depth-First Search of Romania Graph

Arad
Timisoara
Lugoj
Mehadia



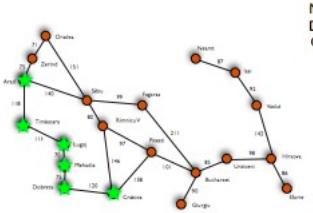
## Depth-First Search of Romania Graph

Arad
Timisoara
Lugoj
Mehadia
Dobreta

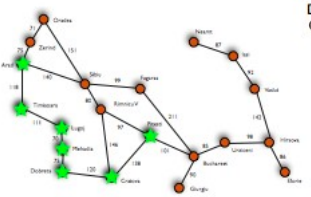


## Depth-First Search of Romania Graph

Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova

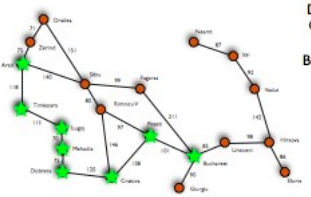


## Depth-First Search of Romania Graph



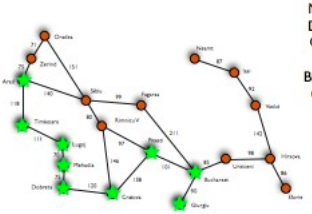
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti

## Depth-First Search of Romania Graph



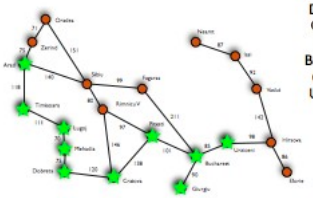
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest

## Depth-First Search of Romania Graph



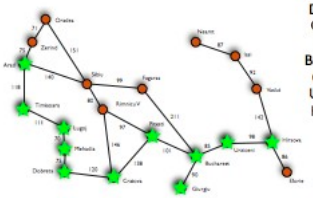
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu

## Depth-First Search of Romania Graph



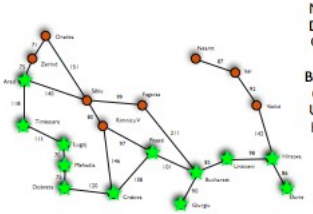
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni

## Depth-First Search of Romania Graph



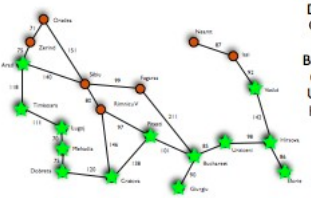
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova

## Depth-First Search of Romania Graph



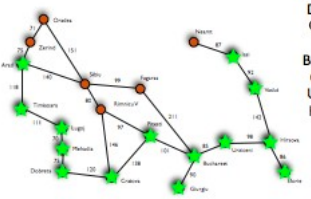
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie

## Depth-First Search of Romania Graph



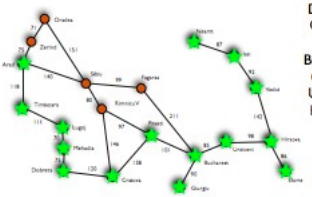
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui

## Depth-First Search of Romania Graph



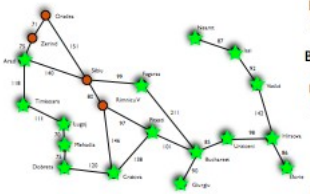
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi

## Depth-First Search of Romania Graph



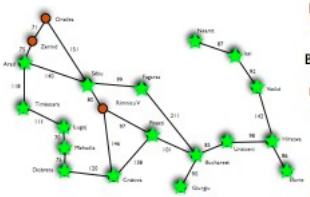
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt

## Depth-First Search of Romania Graph



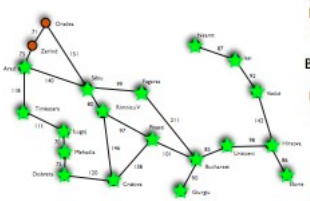
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt
Fagaras

## Depth-First Search of Romania Graph



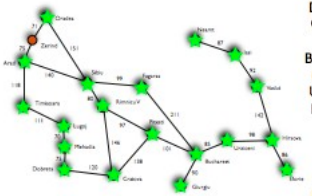
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt
Fagaras
Sibiu

## Depth-First Search of Romania Graph



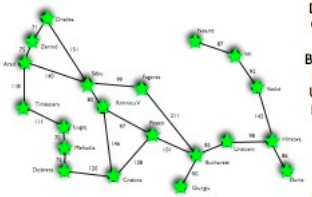
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt
Fagaras
Sibiu
Rimnicu V

## Depth-First Search of Romania Graph



Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt
Fagaras
Sibiu
Rimnicu V
Oradea

## Depth-First Search of Romania Graph



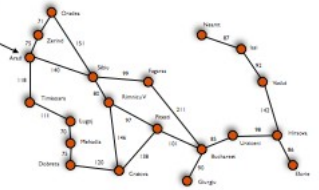
Arad
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Pitesti
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt
Fagaras
Sibiu
Rimnicu V
Oradea
Zerind

```
public abstract class AbstractGraph implements Graph {
.
.
.
    public void dfs () {
        for (int i = 0; i < this.size; i++) this.clear(i);
        for (int i = 0; i < this.size; i++)
            if (!this.marked(i)) this.dfs(i);
    }

    private void dfs (int root) {
        int child;
        this.mark(root);
        this.visit(root);
        Iterator i = this.iterator(root);
        while (i.hasNext()) {
            child = (Integer) i.next();
            if (!this.marked(child)) this.dfs(child);
        }
    }
.
.
.
}
```

## Breadth-First Search of Romania Graph

start here



## Breadth-First Search of Romania Graph

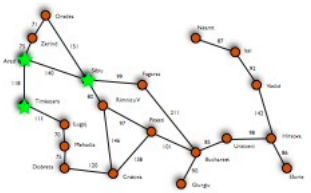
Arad

## Breadth-First Search of Romania Graph

Arad
Timisoara

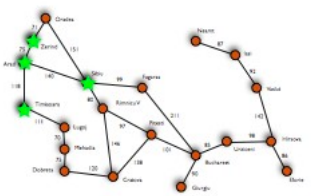
## Breadth-First Search of Romania Graph

Arad
Timisoara
Sibiu



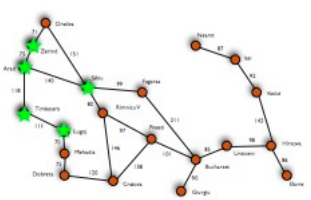
## Breadth-First Search of Romania Graph

Arad
Timisoara
Sibiu
Zerind

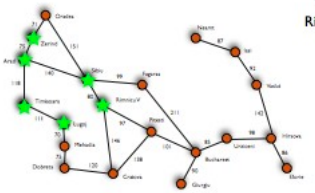


## Breadth-First Search of Romania Graph

Arad
Timisoara
Sibiu
Zerind
Lugoj

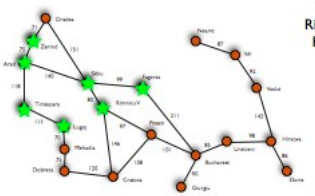


## Breadth-First Search of Romania Graph



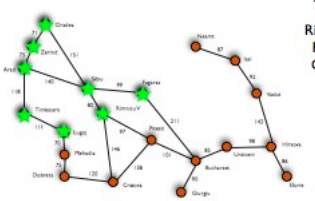
Arad
Timisoara
Sibiu
Zerind
Lugoj
Rimnicu V

## Breadth-First Search of Romania Graph



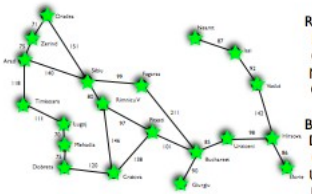
Arad
Timisoara
Sibiu
Zerind
Lugoj
Rimnicu V
Fagaras

## Breadth-First Search of Romania Graph



Arad
Timisoara
Sibiu
Zerind
Lugoj
Rimnicu V
Fagaras
Oradea

# Breadth-First Search of Romania Graph



Arad
Timisoara
Sibiu
Zerind
Lugoj
Rimnicu V
Fagaras
Oradea
Mehadia
Craiova
Pitesti
Bucharest
Dobreta
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Iasi
Neamt

```
public abstract class AbstractGraph implements Graph {
.
.
.
    public void bfs () {
        for (int i = 0; i < this.size; i++) this.clear(i);
        for (int i = 0; i < this.size; i++)
            if (!this.marked(i)) this.bfs(i);
    }
.
.
.
}
```

```
public abstract class AbstractGraph implements Graph {
.
.
.
    private void bfs (int root) {
        Queue fringe = new ListQueue();
        fringe.enter((Integer) root);
        while (!fringe.isEmpty()) {
            int src = (Integer) fringe.leave();
            if (!this.marked(src)) {
                this.mark(src);
                this.visit(src);
                Iterator i = this.iterator(src);
                while (i.hasNext()) {
                    int dest = (Integer) i.next();
                    if (!this.marked(dest))
                        fringe.enter((Integer) dest);
                }
            }
        }
    }
.
.
.
}
```

## Social Graph Simulation

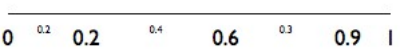
- conservative assumptions: closed population, time boundaries, number of partners — specify probabilities for 0, 1, 2, 3, 4, ... partners
- nodes represent people
- edges represent "connections" — e.g. nodes with no edges are virgins

```
% java Mates 20000 0.2 0.2 0.4 0.1
```

- the above will simulate a population of 20,000 with 20% virgins, 20% one partner, 40% two partners, 10% three partners, and the remainder (10%) four partners

## Implementing Probabalistic Values

- given an array of probabilities, f returns a random integer with a probability of being returned dictated by the array
- e.g. f({ 0.2, 0.4, 0.3 }) should 0 20% of calls, 1 40% of calls, 2 30% of calls, and 3 the remaining 10% of calls
  - generate a random value r between 0 and 1
  - initialize v to be probability array arg[0]
  - for i between 0 and arg.length, if r < v, return i




```
public class MateGraph extends LGraph {
    private int[] nMates;
    private static Random rn = new Random();

    public MateGraph (int size, float[] prob) {
        super(size);
        this.size = size;
        this.nMates = new int[size];
        for (int i = 0; i < size; i++) {
            this.nMates[i] = MateGraph.numMates(prob);
            this.values[i] = String.valueOf(i) + "-" +
                             String.valueOf(this.nMates[i]);
        }
        for (int i = 0; i < size; i++)
            for (int mate = this.pickMate(i);
                 !this.full(i) && mate != Graph.FAIL;
                 mate = this.pickMate(i))
                this.addEdge(i, mate);
    }
.
.
.
}
```

```
public class MateGraph extends LGraph {
.
.
.
    private void addEdge (int src, int dest) {
        this.addEdge(src,dest,0);
        this.addEdge(dest,src,0);
    }

    private boolean full (int i) {
        return this.degree(i) == this.nMates[i];
    }
.
.
.
}
```

```
public class MateGraph extends LGraph {
.
.
.
    private static int numMates (float[] probabilities) {
        float r = MateGraph.rn.nextFloat();
        float f = (float) 0.0;
        int i;
        for (i = 0; i < probabilities.length; i++) {
            f += probabilities[i];
            if (r < f) return i;
        }
        return i;
    }
.
.
.
}
```

```
public class MateGraph extends LGraph {
.
.
.
    private int pickMate (int i) {
        int mate;
        int attempt = 0;
        for (mate = MateGraph.rn.nextInt(this.size);
             (i == mate) || this.full(mate) ||
             this.connected(mate,i);
             mate = MateGraph.rn.nextInt(this.size))
            if (attempt++ > 2500) return Graph.FAIL;
        return mate;
    }
.
.
.
}
```

```
public class MateGraph extends LGraph {
.
.
.
    public int countConnected (int i) {
        int count = 1;
        this.mark(i);
        Iterator iter = this.iterator(i);
        while (iter.hasNext()) {
            int child = (Integer) iter.next();
            if (!this.marked(child))
                count += this.countConnected(child);
        }
        return count;
    }
.
.
.
}
```

```
public class Mates {
    public static void main (String[] arg) {
        int population = Integer.parseInt(arg[1]);
        float[] probability = new float[arg.length - 2];

        for (int i = 0; i < probability.length; i++)
            probability[i] = Float.parseFloat(arg[i+2]);

        int group[] = new int[population+1];
        MateGraph m = new MateGraph(population, probability);

        for (int i = 0; i < population; i++)
            if (!m.marked(i))
                group[m.countConnected(i)]++;

        for (int i = 0; i <= population; i++) {
            String s = group[i] == 1 ? "" : "s";
            if (group[i] != 0)
            IO.stdout.println(group[i]+" group"+s+" of "+i);
        }
    }

}
```