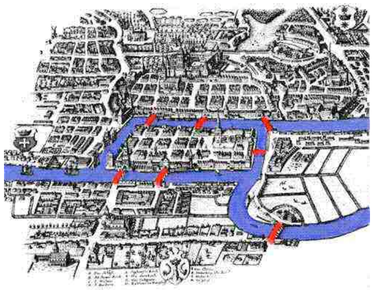
Undirected Graphs



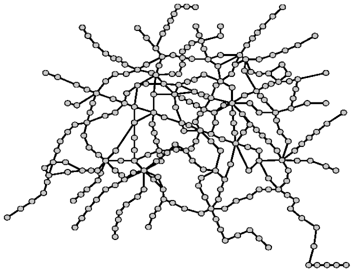
Reference: Chapter 17-18, Algorithms in Java, 3rd Edition, Robert Sedgewick

Undirected Graphs

Graph. Set of objects with pairwise connections.

Why study graph algorithms?

- Interesting and broadly useful abstraction.
- Challenging branch of computer science and discrete math.
- Hundreds of graph algorithms known.
- Thousands of practical applications.



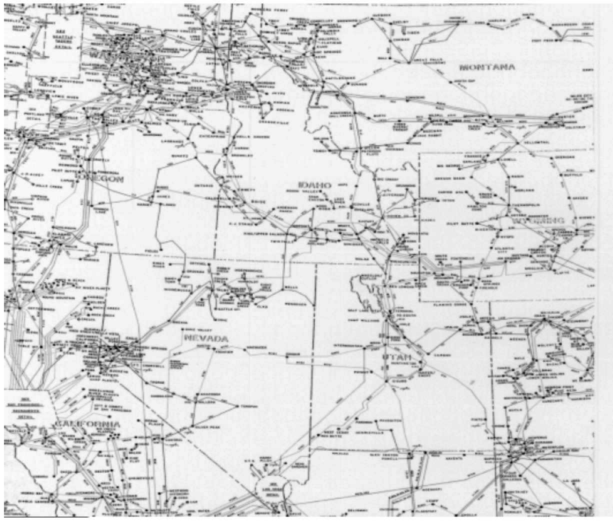
Graph Applications

Graph	Vertices	Edges
communication	telephones, computers	fiber optic cables
circuits	gates, registers, processors	wires
mechanical	joints	rods, beams, springs
hydraulic	reservoirs, pumping stations	pipelines
financial	stocks, currency	transactions
transportation	street intersections, airports	highways, airway routes
scheduling	tasks	precedence constraints
software systems	functions	function calls
internet	web pages	hyperlinks
games	board positions	legal moves
social relationship	people, actors	friendships, movie casts
neural networks	neurons	synapses
protein networks	proteins	protein-protein interactions
chemical compounds	molecules	bonds

September 11 Hijackers and Associates

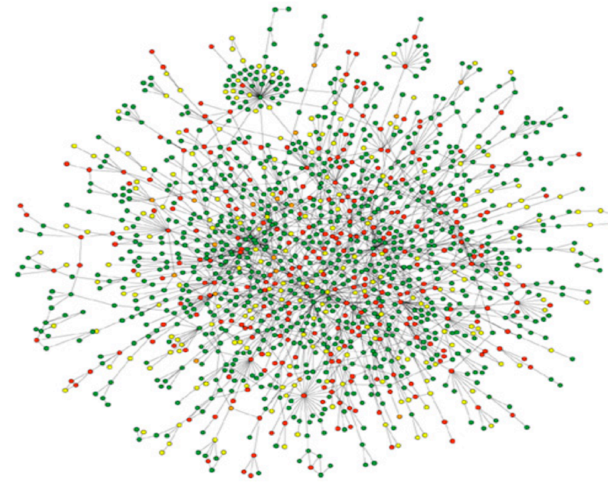
Reference: Valdis Krebs
http://www.firstmonday.org/issues/issue7_4/krebs

Power Transmission Grid of Western US



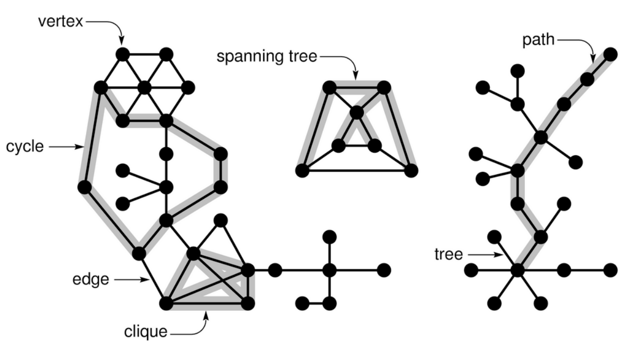
Reference: Duncan Watts

Protein Interaction Network



Reference: Jeong et al, Nature Review | Genetics

Graph Terminology



Some Graph Problems

Path. Is there a path between s to t?
Shortest path. What is the shortest path between s and t?
Longest path. What is the longest simple path between s and t?

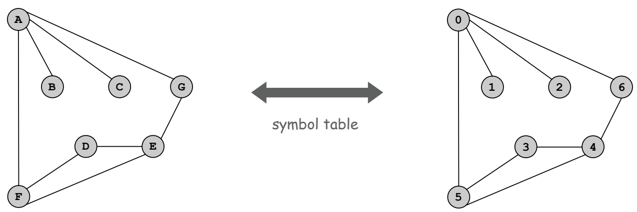
Cycle. Is there a cycle in the graph?
Euler tour. Is there a cycle that uses each edge exactly once?
Hamilton tour. Is there a cycle that uses each vertex exactly once?

Connectivity. Is there a way to connect all of the vertices?
MST. What is the best way to connect all of the vertices?
Biconnectivity. Is there a vertex whose removal disconnects the graph?

Planarity. Can you draw the graph in the plane with no crossing edges?
Isomorphism. Do two adjacency matrices represent the same graph?

Graph Representation

Vertex representation.
- This lecture: use integers between `0` and `V-1`.
- Real world: convert between names and integers with symbol table.



Other issues. Parallel edges, self-loops.

Graph API

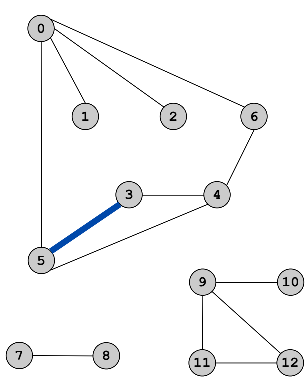
public class Graph	(graph data type)	
	`Graph(int V)`	create an empty graph with V vertices
	`Graph(int V, int E)`	create a random graph with V vertices, E edges
`void`	`insert(int v, int w)`	add an edge v-w
`Iterable<Integer>`	`adj(int v)`	return an iterator over the neighbors of v
`int`	`V()`	return number of vertices
`String`	`toString()`	return a string representation

```
Graph G = new Graph(V, E);
System.out.println(G);
for (int v = 0; v < G.V(); v++)
   for (int w : G.adj(v))
      // edge v-w
```

iterate through all edges (in each direction)

Set of Edge Representation

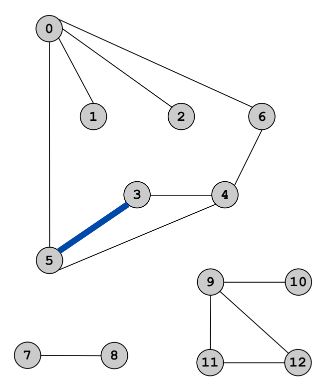
Set of edge representation. Store list of edges.




```
0-1
0-6
0-2
11-12
9-12
9-11
9-10
4-3
5-3
7-8
5-4
0-5
6-4
```


Adjacency Matrix Representation

Adjacency matrix representation.
- Two-dimensional `V` × `V` boolean array.
- Edge `v-w` in graph: `adj[v][w] = adj[w][v] = true`.



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	1	0	1	1	0	0	0	0	0	0	0	0
6	1	1	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Adjacency Matrix Representation: Java Implementation

```java
public class Graph {
   private int V;            // number of vertices
   private boolean[][] adj;  // adjacency matrix

   // empty graph with V vertices
   public Graph(int V) {
      this.V = V;
      this.adj = new boolean[V][V];
   }

   // insert edge v-w, no parallel edges
   public void insert(int v, int w) {
      adj[v][w] = true;
      adj[w][v] = true;
   }

   // return iterator for neighbors of v
   public Iterable<Integer> adj(int v) {
      return new AdjIterator(v);
   }
}
```

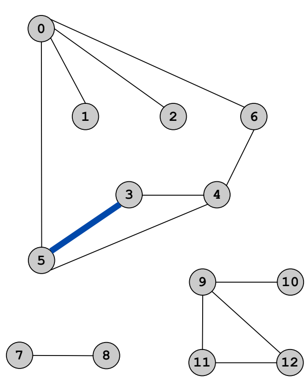

Adjacency Matrix Iterator

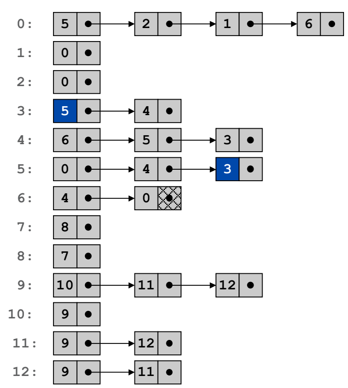
```java
private class AdjIterator implements Iterator<Integer>,
                                     Iterable<Integer> {
   int v, w = 0;
   AdjIterator(int v) { this.v = v; }

   public boolean hasNext() {
      while (w < V) {                    does v have another neighbor w?
         if (adj[v][w]) return true;
         w++;
      }
      return false;
   }

   public int next() {
      if (!hasNext()) throw new NoSuchElementException();
      return w++;
   }

   public Iterator<Integer> iterator() { return this; }

}
```


Adjacency List Representation

Adjacency list.

- Vertex indexed array of lists.
- Two representations of each undirected edge.





Adjacency List Representation: Java Implementation

```java
public class Graph {
   private int V;                          // # vertices
   private Sequence<Integer>[] adj;  // adjacency lists

   public Graph(int V) {
      this.V = V;
      adj = (Sequence<Integer>[]) new Sequence[V];
      for (int v = 0; v < V; v++)
         adj[v] = new Sequence<Integer>();
   }

   // insert v-w, parallel edges allowed
   public void insert(int v, int w) {
      adj[v].add(w);
      adj[w].add(v);
   }

   public Iterable<Integer> adj(int v) {
      return adj[v];
   }
}
```

Graph Representations

Graphs are abstract mathematical objects.

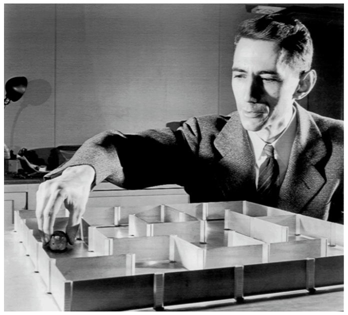
- ADT implementation requires specific representation.
- Efficiency depends on matching algorithms to representations.

Representation	Space	Edge between v and w?	Iterate over edges incident to v?
List of edges	E	E	E
Adjacency matrix	V^2	1	V
Adjacency list	E + V	degree(v)	degree(v)

Graphs in practice. [use adjacency list representation]

- Real world graphs are sparse.
- Bottleneck is iterating over edges incident to v.

Maze Exploration

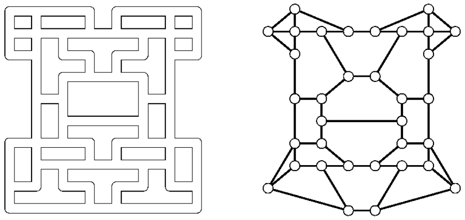


Claude Shannon (with Theseus mouse)

Maze Exploration

Maze graphs.

- Vertex = intersections.
- Edge = passage.



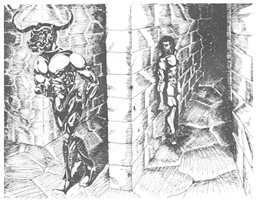
Goal. Explore every passage in the maze.

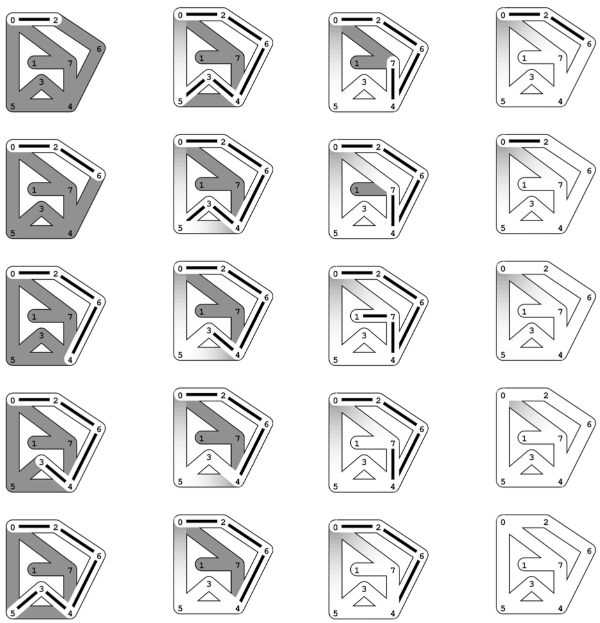
Trémaux Maze Exploration

Trémaux maze exploration.

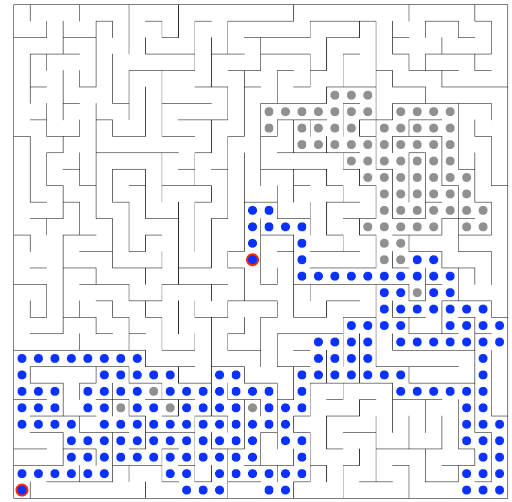
- Unroll a ball of string behind you.
- Mark each visited intersection by turning on a light.
- Mark each visited passage by opening a door.

History. Theseus entered labyrinth to kill the monstrous Minotaur; Ariadne held ball of string.

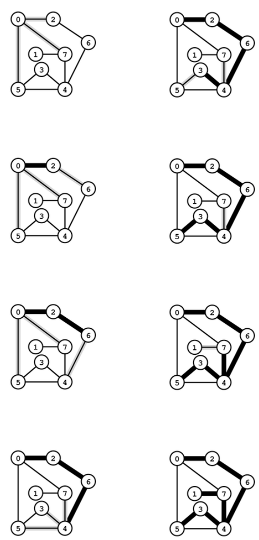




Maze Exploration



Depth First Search



Goal. Find all vertices connected to `s`.

DFS (to visit a vertex `v`)

Mark `v` as visited.
Visit all unmarked vertices `w` adjacent to `v`.

Running time. O(E) since each edge examined at most twice.

Graph Processing Client

Typical client program.

- Create a `Graph`.
- Pass the `Graph` to a graph processing routine, e.g., `DFSearcher`.
- Query the graph processing routine for information.

```
public static void main(String[] args) {
   int V = Integer.parseInt(args[0]);
   int E = Integer.parseInt(args[1]);
   Graph G = new Graph(V, E);
   int s = 0;
   DFSearcher dfs = new DFSearcher(G, s);
   for (int v = 0; v < G.V(); v++)
      if (dfs.isReachable(v))
         System.out.println(v);
}
```

find and print all vertices reachable from s

Design pattern. Decouple graph from graph algorithms.

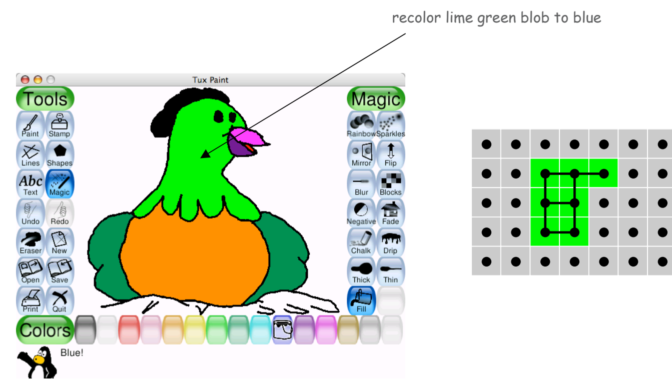
Depth First Search

```
public class DFSearcher {
   private boolean[] marked;

   public DFSearcher(Graph G, int s) {
      marked = new boolean[G.V()];
      dfs(G, s);
   }

   // depth first search from v
   private void dfs(Graph G, int v) {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w]) dfs(G, w);
   }

   public boolean isReachable(int v) {
      return marked[v];
   }
}
```


Reachability Application: Flood Fill

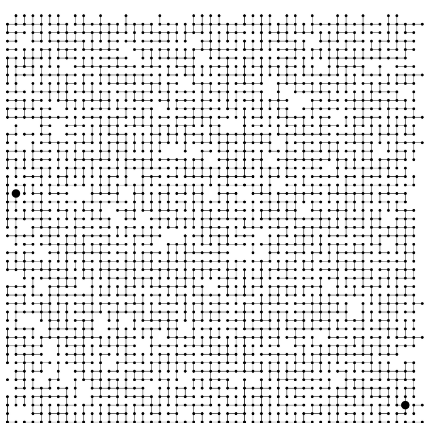
Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Vertex: pixel.
- Edge: between two adjacent lime pixels.
- Blob: all pixels reachable from chosen lime pixel.



Paths

Path. Is there a path from s to t? If so, find one.



Paths

Path. Is there a path from s to t? If so, find one.

Method	Preprocess Time	Query Time	Space
Union Find	E log* V †	log* V †	V
DFS	E + V	1	V + E

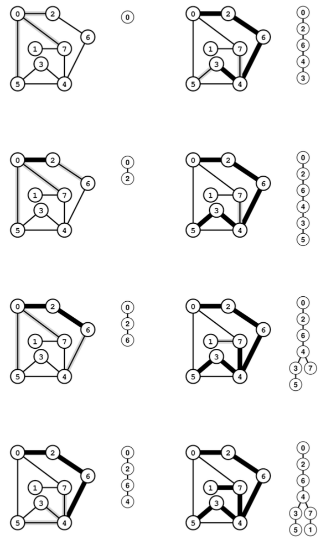
† amortized

UF advantage. Can intermix query and edge insertion.
DFS advantage. Can recover path itself in same running time.

Keep Track of Path

DFS tree. Upon visiting a vertex `v` for the first time, remember from where you came `pred[v]`.

Retrace path. To find path between `s` and `v`, follow `pred[]` values back from `v`.



Find Path

```
public class DFSearcher {
   // initialize pred[v] to -1 for all v

   private void dfs(Graph G, int v) {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w]) {
            pred[w] = v;
            dfs(G, w);
         }
   }

   // return path from s to v
   public Iterable<Integer> path(int v) {
      Stack<Integer> list = new Stack<Integer>();
      while (v != -1 && marked[v]) {
         list.push(v);
         v = pred[v];
      }
      return list;
   }
}
```


DFS Summary

Enables direct solution of simple graph problems.

- Find path between s to t.
- Connected components.
- Euler tour.
- Cycle detection.
- Bipartiteness checking.

Basis for solving more difficulty graph problems.

- Biconnected components.
- Planarity testing.

Shortest Path

Breadth First Search

Depth-first search. Put unvisited vertices on a stack.
Breadth-first search. Put unvisited vertices on a queue.

Shortest path. Find path from s to t that uses fewest number of edges.

BFS (from source vertex `s`)

Put `s` onto a FIFO queue.
Repeat until the queue is empty:
- remove the least recently added vertex `v`
- add each of `v`'s unvisited neighbors to the queue, and mark them as visited.

Property. BFS examines vertices in increasing distance from s.

Breadth First Search

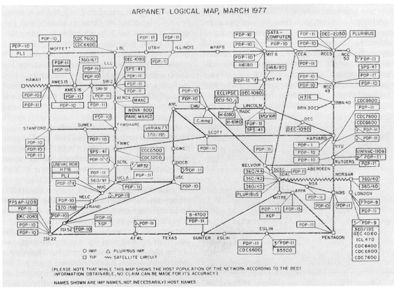
```
public class BFSearcher {
   private static int INFINITY = Integer.MAX_VALUE;

   private int[] dist;

   public BFSearcher(Graph G, int s) {
      dist = new int[G.V()];
      for (int v = 0; v < G.V(); v++) dist[v] = INFINITY;
      dist[s] = 0;
      bfs(G, s);
   }

   public int distance(int v)        { return dist[v]; }
   private void bfs(Graph G, int s) { // NEXT SLIDE    }
}
```


Breadth First Search

```
// breadth-first search from s
private void bfs(Graph G, int s) {
   Queue<Integer> q = new Queue<Integer>();
   q.enqueue(s);
   while (!q.isEmpty()) {
      int v = q.dequeue();
      for (int w : G.adj(v)) {
         if (dist[w] == INFINITY) {
            q.enqueue(w);
            dist[w] = dist[v] + 1;
         }
      }
   }
}
```


BFS Application

BFS applications.
- Facebook.
- Kevin Bacon numbers.
- Fewest number of hops in a communication network.



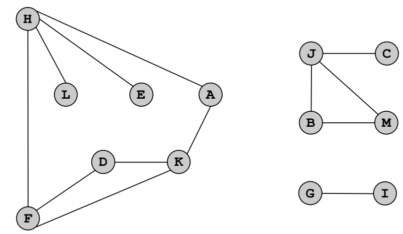
ARPANET

Connected Components

Connectivity Queries

Def. Vertices v and w are connected if there is a path between them.
Property. Symmetric and transitive.

Goal. Preprocess graph to answer queries: is `v` connected to `w`?
Brute force. Run DFS from each vertex `v`: quadratic time and space.

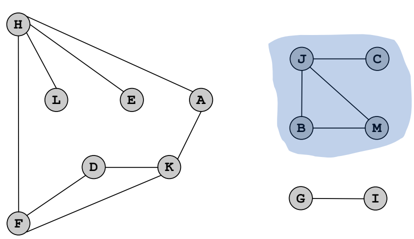


Vertex	Component
A	0
B	1
C	1
D	0
E	0
F	0
G	2
H	0
I	2
J	1
K	0
L	0
M	1

Connectivity Queries

Def. Vertices v and w are connected if there is a path between them.
Property. Symmetric and transitive.

Goal. Preprocess graph to answer queries: is `v` connected to `w`?



Vertex	Component
A	0
B	1
C	1
D	0
E	0
F	0
G	2
H	0
I	2
J	1
K	0
L	0
M	1

Connected component. Maximal set of mutually connected vertices.

Connected Components

Goal. Partition vertices into connected components.

Connected components

Initialize all vertices `v` as unmarked.
For each unmarked vertex `v`, run DFS and identify all vertices discovered as part of the same connected component.

Preprocess Time	Query Time	Space
E + V	1	V

Depth First Search: Connected Components

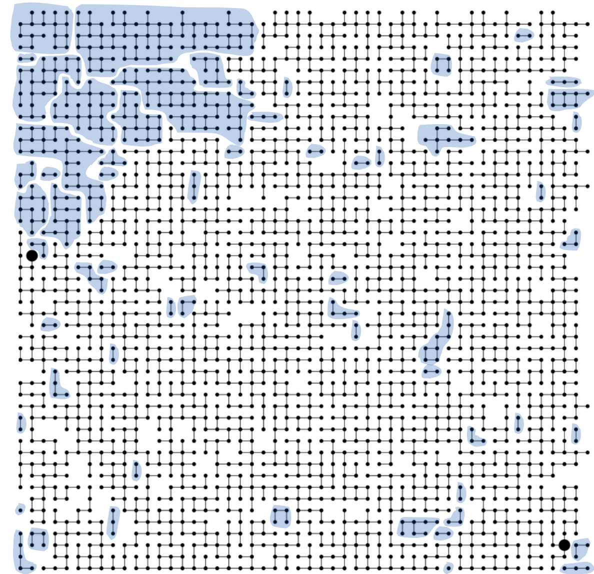
```java
public class CCFinder {
   private int components;
   private int[] cc;

   public CCFinder(Graph G) {
      cc = new int[G.V()];
      for (int v = 0; v < G.V(); v++) cc[v] = -1;
      for (int v = 0; v < G.V(); v++)
         if (cc[v] == -1) { dfs(G, v); components++; }
   }

   // depth first search from v
   private void dfs(Graph G, int v) {
      cc[v] = components;
      for (int w : G.adj(v))
         if (cc[w] == -1) dfs(G, w);
   }

   public int connected(int v, int w) { return cc[v] == cc[w]; }
}
```

unmarked

are v and w in same connected component?

Connected Components

63 components



Connected Components Application: Image Processing

Goal. Read in a 2D color image and find regions of connected pixels that have the same color.

original

labeled

Connected Components Application: Image Processing

Goal. Read in a 2D color image and find regions of connected pixels that have the same color.

Efficient algorithm.

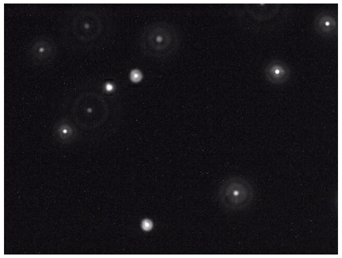
- Connect each pixel to neighboring pixel if same color.
- Find connected components in resulting graph.

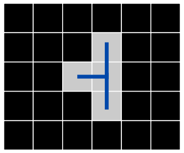
0	1	1	1	1	1	6	6	8	9	9	11
0	0	0	1	6	6	6	8	8	11	9	11
3	0	0	1	6	6	4	8	11	11	11	11
3	0	0	1	1	6	2	11	11	11	11	11
10	10	10	10	1	1	2	11	11	11	11	11
7	7	2	2	2	2	2	11	11	11	11	11
7	7	5	5	5	2	2	11	11	11	11	11

Particle detection. Given grayscale image of particles, identify "blobs."

- Vertex: pixel.
- Edge: between two adjacent pixels with grayscale value ≥ 70.
- Blob: connected component of 20-30 pixels.

black = 0
white = 255





Particle tracking. Track moving particles over time.