# Undirected Graphs

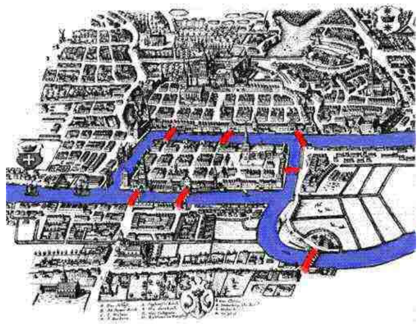
Undirected graphs

Adjacency lists

BFS

DFS

Euler tour



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



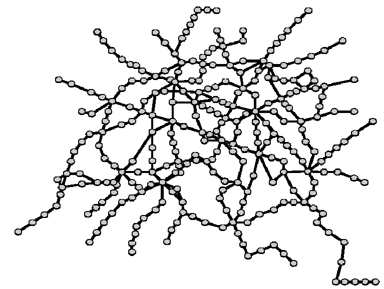
## Undirected Graphs

GRAPH. Set of OBJECTS with pairwise CONNECTIONS.

- Interesting and broadly useful abstraction.

Why study graph algorithms?

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





## Graph Applications

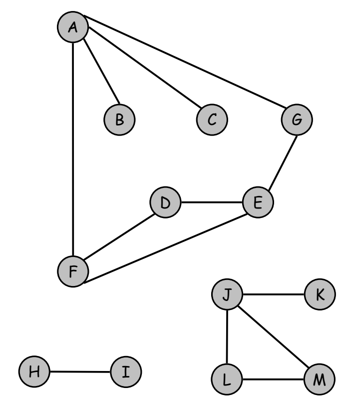
| Graph | Vertices | Edges |
|---|---|---|
| communication | telephones, computers | fiber optic cables |
| circuits | gates, registers, processors | wires |
| mechanical | joints | rods, beams, springs |
| hydraulic | reservoirs, pumping stations | pipelines |
| financial | stocks, currency | transactions |
| 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 |



## Graph Jargon

Terminology.

- Vertex: `v`.
- Edge: `e = v-w`.
- Graph: `G`.
- `V` vertices, `E` edges.
- Parallel edge, self loop.
- Directed, undirected.
- Sparse, dense.
- Path, cycle.
- Cyclic path, tour.
- Tree, forest.
- Connected, connected component.



## A Few Graph Problems

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

Cycle. Is there a cycle in the graph?
Euler tour. Is there a cyclic path 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?
Bi-connectivity. Is there a vertex whose removal disconnects graph?

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



## Graph ADT in Java

Typical client program.

- Create a `Graph`.
- Pass the `Graph` to a graph processing routine, e.g., `DFSearcher`.
- Query the graph processing routine for information.
- Design pattern: separate graph from graph algorithm.

```
public static void main(String args[]) {
   int V = Integer.parseInt(args[0]);
   int E = Integer.parseInt(args[1]);
   Graph G = new Graph(V, E);
   System.out.println(G);
   DFSearcher dfs = new DFSearcher(G);
   int comp = dfs.components();
   System.out.println("Components = " + comp);
}
```

calculate number of connected components

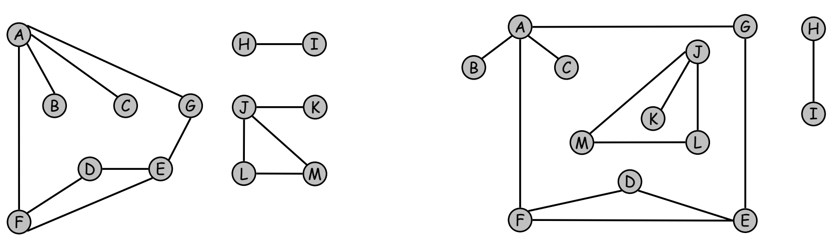


## Graph Representation

Vertex names. A B C D E F G H I J K L M

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

Two drawing represent same graph.




Set of edges representation.

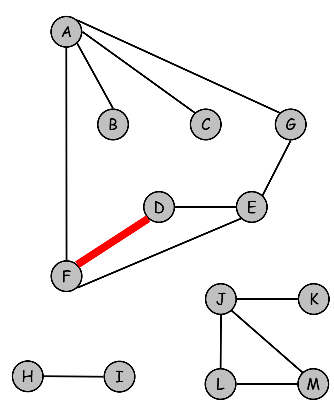
- A-B A-G A-C L-M J-M J-L J-K E-D F-D H-I F-E A-F G-E



## Adjacency Matrix Representation

Adjacency matrix representation.

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




| | | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | A | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | B | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | C | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | D | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | E | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | F | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | G | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 8 | I | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | J | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 10 | K | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 11 | L | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 12 | M | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

adjacency matrix

## Adjacency Matrix: Java Implementation

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

   // empty graph with V vertices
   public Graph(int V) {
      this.V = V;
      this.E = 0;
      this.adj = new boolean[V][V];
   }

   // insert edge v-w if it doesn't already exist
   public void insert(int v, int w) {
      if (!adj[v][w]) E++;
      adj[v][w] = true;
      adj[w][v] = true;
   }
}
```



## Iterator

Iterator.

- Client needs way to iterate through elements of adjacency list.
- Graph implementation doesn't want to reveal details of list.
- Design pattern: give client just enough to iterate.

```
interface IntIterator {
   int next();
   boolean hasNext();
}
```

contract for implementing an integer-valued iterator

```
IntIterator i = G.neighbors(v);
while (i.hasNext()) {
   int w = i.next();
   // do something with edge v-w
}
```

idiom for traversing with an iterator



## Adjacency Matrix Iterator: Java Implementation

```
public IntIterator neighbors(int v) {
   return new AdjMatrixIterator(v);
}

private class AdjMatrixIterator implements IntIterator {
   int v, w = 0;
   AdjMatrixIterator(int v) { this.v = v; }

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

   public int next() {              return next neighbor w of v
      if (hasNext()) return w++;
      return -1;
   }
}
```

Graph.java



## Iterator Diversion: Java Collections

Iterator.

- Java uses interface `Iterator` with all of its collection data types.
- Its method `next` returns an `Object` instead of an `int`.
- Need to `import java.util.Iterator` and `java.util.ArrayList`.

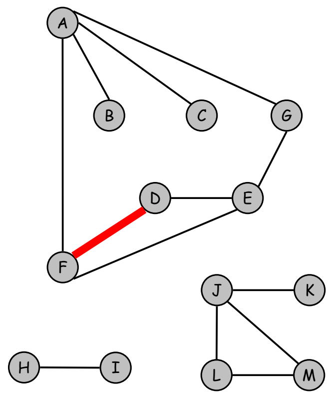
```
ArrayList list = new ArrayList();
...
list.add(value);
...
Iterator i = list.iterator();
while(i.hasNext()) {
   System.out.println(i.next());
}
```

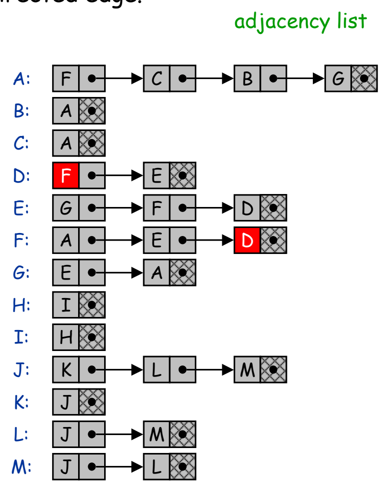
- You can now write the `ArrayList` or `LinkedList` libraries and use an `Iterator` to traverse them.

## Adjacency List Representation

Vertex indexed array of lists.

- Space proportional to number of edges.
- Two representations of each undirected edge.






## Adjacency List: Java Implementation

```java
public class Graph {
   private int V;            // # vertices
   private int E;            // # edges
   private AdjList[] adj;    // adjacency lists

   private static class AdjList {
      int w;
      AdjList next;
      AdjList(int w, AdjList next) { this.w = w; this.next = next; }
   }

   public Graph(int V) {
      this.V = V;
      this.E = 0;
      adj = new AdjList[V];
   }

   public void insert(int v, int w) {
      adj[v] = new AdjList(w, adj[v]);
      adj[w] = new AdjList(v, adj[w]);
      E++;
   }
```

add w to adjacency list

empty graph with V vertices

insert edge v-w,
parallel edges allowed



## Adjacency List Iterator: Java Implementation

```java
public IntIterator neighbors(int v) {
   return new AdjListIterator(adj[v]);
}

private class AdjListIterator implements IntIterator {
   AdjList x;
   AdjListIterator(AdjList x) { this.x = x; }

   public boolean hasNext() {
      return x != null;
   }

   public int next() {
      int w = x.w;
      x = x.next;
      return w;
   }
}
```

does v have another neighbor?

return next neighbor w of v

Graph.java



## 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? | Edge from v to anywhere? | Enumerate all edges |
|---|---|---|---|---|
| Adjacency matrix | $\Theta(V^2)$ | $\Theta(1)$ | $O(V)$ | $\Theta(V^2)$ |
| Adjacency list | $\Theta(E + V)$ | $O(E)$ | $\Theta(1)$ | $\Theta(E + V)$ |

Graphs in practice.

- Typically sparse.
- Typically bottleneck is iterating through all edges.
- Use adjacency list representation.

## Graph Search

Goal. Visit every node and edge in Graph.

A solution. Depth-first search.

- To visit a node `v`:
  - mark it as visited
  - recursively visit all unmarked nodes `w` adjacent to `v`
- To traverse a Graph `G`:
  - initialize all nodes as unmarked
  - visit each unmarked node

Enables direct solution of simple graph problems.

- Connected components.
- Cycles.

Basis for solving more difficult graph problems.

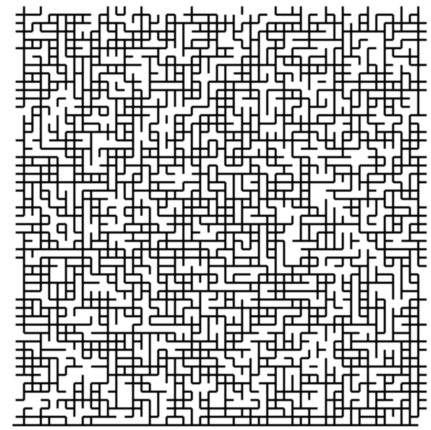
- Biconnectivity.
- Planarity.



## Connected Components

Define problem.

- Disconnected pieces may be hard to spot, especially for computer!



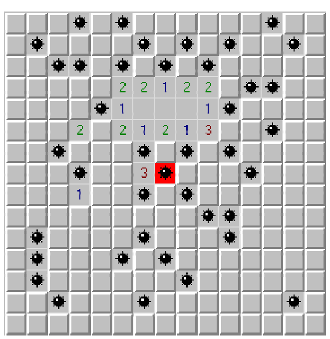


## Connected Components Application: Minesweeper

Challenge: implement the game of Minesweeper.

Critical subroutine.

- User selects a cell and program reveals how many adjacent mines.
- If zero, reveal all adjacent cells.
- If any newly revealed cells have zero adjacent mines, repeat.

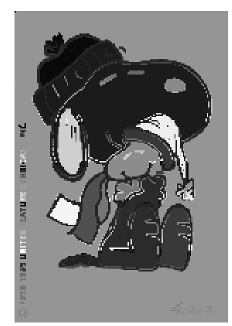




## Connected Components Application: Image Processing

Challenge: read in a 2D color image and find regions of connected pixels that have the same color.

Original



Labeled

## Connected Components Application: Image Processing

Challenge: 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 |



## Depth First Search: Connected Components

```
public class DFSearcher {
   private final static int UNMARKED = -1;
   private Graph G;
   private int[] cc;
   private int components = 0;

   public DFSearcher(Graph G) {
      this.G = G;
      this.cc = new int[G.V()];
      for (int v = 0; v < G.V(); v++)
         cc[v] = UNMARKED;
      dfs();
   }

   private void dfs()           { // NEXT SLIDE      }
   public int component(int v)  { return cc[v];      }
   public int components()      { return components; }

}
```



## Depth First Search: Connected Components

```
// run dfs from each unmarked vertex
private void dfs() {
   for (int v = 0; v < G.V(); v++) {
      if (cc[v] == UNMARKED) {
         dfs(v);
         components++;
      }
   }
}

// depth first search
private void dfs(int v) {
   cc[v] = components;
   IntIterator i = G.neighbors(v);
   while (i.hasNext()) {
      int w = i.next();
      if (cc[w] == UNMARKED) dfs(w);
   }
}
```

loop idiom



## Connected Components

Path. Is there a path from s to t?

| Method | Preprocess Time | Query Time | Space |
|---|---|---|---|
| Union Find | O(E log* V) | O(log* V) | Θ(V) |
| DFS | Θ(E + V) | Θ(1) | Θ(V) |

UF advantage. Dynamic: can intermix query and edge insertion.

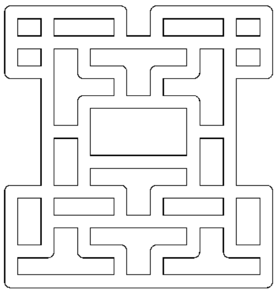
DFS advantage.

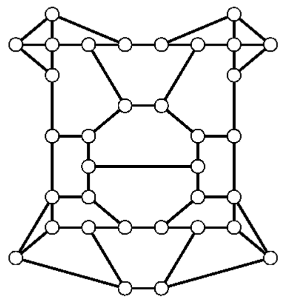
- Can get path itself in same running time.
- Extends to more general problems.

## Graphs and Mazes

Maze graphs.

- Vertices = intersections
- Edges = corridors.





DFS.

- Mark ENTRY and EXIT halls at each vertex.
- Leave by ENTRY when no unmarked halls.



## Breadth First Search

Graph search. Visit all nodes and edges of graph.
Depth-first search. Put unvisited nodes on a STACK.
Breadth-first search. Put unvisited nodes on a QUEUE.

Shortest path: what is fewest number of edges to get from s to t?

Solution = BFS.

- Initialize `dist[v] = ∞`, `dist[s] = 0`.
- When considering edge `v-w`:
  - if `w` is marked, then ignore
  - otherwise else set `dist[w] = dist[v] + 1`, `pred[w] = v`, and add `w` to the queue

if you want to find shortest path itself



## Breadth First Search

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

   private Graph G;
   private int[] dist;

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

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

max integer



## Breadth First Search

```
// breadth-first search from s
private void bfs(int s) {
   IntQueue q = new IntQueue();
   q.enqueue(s);
   while (!q.isEmpty()) {
      int v = q.dequeue();
      IntIterator i = G.neighbors(v);
      while (i.hasNext()) {
         int w = i.next();
         if (dist[w] == INFINITY) {
            q.enqueue(w);
            dist[w] = dist[v] + 1;
         }
      }
   }
}
```

## Related Graph Search Problems

Path. Is there a path from s to t?
- Solution: DFS, BFS, or PFS.

Shortest path. Find shortest path (fewest edges) from s to t.
- Solution: BFS.

Bi-connected components. Which nodes participate in cycles?
- Solution: DFS (see textbook).

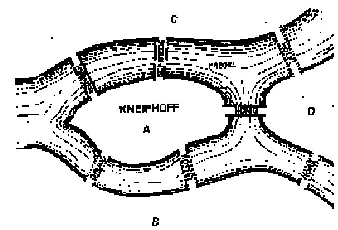
Euler tour. Is there a cyclic path that uses each edge exactly once?
- Solution: DFS.

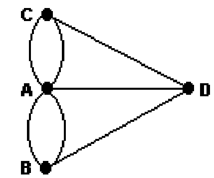
Hamilton tour. Is there a cycle that uses each vertex exactly once?
- Solution: ??? (NP-complete).



## Bridges of Königsberg

Leonhard Euler, *The Seven Bridges of Königsberg*, 1736.



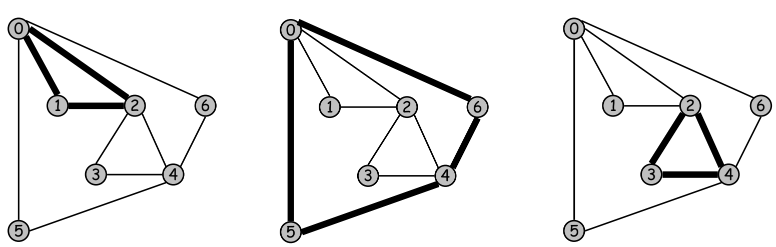

"..... in Königsberg in Prussia, there is an island A, called the Kneiphof; the river which surrounds it is divided into two branches ... and these branches are crossed by seven bridges. Concerning these bridges, it was asked whether anyone could arrange a route in such a way that he could cross each bridge once and only once....."

Euler tour. Is there a cyclic path that uses each edge exactly once?
- Yes if connected and degrees of all vertices are even.



## Euler Tour

How to find an Euler tour (assuming graph is Eulerian).
- Start at some vertex v and repeatedly follow unused edge until you return to v.
  - always possible since all vertices have even degree
- Find additional cyclic paths using remaining edges and splice back into original cyclic path.






## Euler Tour

How to find an Euler tour (assuming graph is Eulerian).
- Start at some vertex v and repeatedly follow unused edge until you return to v.
  - always possible since all vertices have even degree
- Find additional cyclic paths using remaining edges and splice back into original cyclic path.

How to efficiently keep track of unused edges?
- Delete edges after you use them.

How to efficiently find and splice additional cyclic paths?
- Push each visited vertex onto a stack.

# Euler Tour: Implementation

```
private int euler(int v) {
   while (true) {
      IntIterator i = G.neighbors(v);
      if (!i.hasNext()) break;
      stack.push(v);
      G.remove(v, w);        destroys graph
      v = w;
   }
   return v;
}

public void show() {
   stack = new intStack();
   stack.push(0);          found cyclic path from v to itself
   while (euler(v) == v && !stack.isEmpty()) {
      v = stack.pop();
      System.out.println(v);
   }
   if (!stack.isEmpty())
      System.out.println("Not Eulerian");
}
```

assumes graph is connected