Definition 2.11 – Countable and uncountable sets  
A set M is called countably infinite, if it has the same cardinality as the set N of natural numbers, i.e. if there is a bijective function (1-to-1-mapping) f:M -> N where every element m ∈ M of the set is assigned to an enumeration position f(m) and for every position there is a value m ∈ M too

A set M is called uncountable, if it is infinite and not countable, i.e. there is no 1-to-1 mapping to the natural numbers

We can basically map the numbers from the set to natural numbers are called countable numbers

A set is uncountable if no such bijection exists  
Show that the set of all real numbers between 0 and 1 is countable

**Countable Set Example:**  
Set of Natural Numbers between 0 to 1 – Because there are specific amount of entries that can be considered  
  
**Uncountable Set Example:**  
Set of Real Numbers between 0 to 1 --- Because we can always construct a new value  
  
Due to the Cantor’s diagonal argument, which shows that you can always construct a **new** real number that differs from each number in the list by changing the nth digit in the nth place (like creating a new number based on the diagonal)

Number

Computing the degree of a vertex from adjacency matrix representation:

**Undirected Graph**Count ones in the column (or row) of the vertex, count diagonal elements twice (loops count twice!)  
Count diagonal element twice, does this mean the same thing like that vertex having a link to themselves will be considered with a value two

A screenshot of a graph

Description automatically generated

**Directed graph**number of ones in row of vertex + number of ones in line of vertex.

How can you compute the degree of a vertex from the adjacency lists representation?  
 **Undirected Graph**length of adjacency list  
add 1, if the vertex is contained in its adjacency (for loop)

A white and red text with black text

Description automatically generated with medium confidence

**Directed Graph**Length of the adjacency list of the vertex (outdegree) + number of occurrences of the vertex in all adjacency lists (indegree)

Let G = (V, E) be a directed graph, where V = {1, 2, 3, 4} E = { (1,2), (1,4), (2,2), (2,3), (2,4), (3,1), (4,3) }   
  
a) Draw G as a diagram.

First question: any advice of drawing the diagram to follow the planer concepts, like can i connect with a curve.

b) Determine the indegree and the outdegree of vertex 2.

c) Is G planar?

This means that the graph can be drawn in the plane without crossing edges 🡪 plane What does plane mean, does it mean that the arrows need to be straight or can be connected with curve based arrows, similar question in part a

d) Is G strongly connected?

This means that each vertex can be reached from each other vertex. All vertices should be able reach each other

e) Is G acyclic?

So, if a cycle exists, this case as there exist cycles such as: (1,2,3,1) or (1,2,4,1), so because of the cycles this will be called not acyclic.

Exercise 1.4

**Steps to Solve the "Knows" Problem (Exercise 1.4):**

1. **List the People**:
   * Write down the names or use abbreviations (Anna = A, Bruno = B, Cindy = C, Dave = D, Emma = E).
   * This helps keep the problem organized and makes it easier to visualize relationships.
2. **Identify the Person with the Most Connections**:
   * In the problem, Dave knows the most people (four). Start with him since this will lay a strong foundation for the graph.
   * **Explanation**: Starting with the person who has the most connections reduces the number of possible placements for other connections.
3. **Add Connections for Dave**:
   * Draw Dave (D) and connect him with the other four people. This forms the basic structure of the graph.
   * **Explanation**: Since Dave knows everyone, this step gives you the maximum number of connections, reducing uncertainty for others.
4. **Move to the Next Person with Fewer Connections**:
   * After Dave, move to Cindy (C), who knows three people.
   * Based on Dave's existing connections, add Cindy's connections to the graph.
   * **Explanation**: Handling the second most-connected person next helps refine the graph further and provides more structure to guide the remaining steps.
5. **Continue with Bruno**:
   * Bruno (B) knows two people. Add his connections, making sure that the relationships are symmetric (if Bruno knows Cindy, Cindy must know Bruno).
   * **Explanation**: Since fewer people are left to connect, this step should start becoming easier. You can only place Bruno's connections where they haven't been connected yet.
6. **Handle Anna's Connection**:
   * Anna (A) knows just one person. Based on the existing structure, determine Anna's connection, ensuring the "knows" relationship is symmetric.
   * **Explanation**: By this stage, most of the graph is already formed, and Anna's connection should be the final piece to add without ambiguity.
7. **Determine Emma’s Connections**:
   * Since everyone else’s connections are already set, the graph will reveal that Emma (E) knows two people. Ensure this fits within the remaining available connections.
   * **Explanation**: After placing all the other people, Emma's connections should naturally follow as the remaining unconnected relationships.
8. **Double-Check Symmetry**:
   * Ensure that all connections are symmetric, meaning if one person knows another, the reverse relationship is also true.
   * **Explanation**: Symmetry is a key rule in this problem, so verifying this ensures your solution is accurate.

**Key Explanation of the Strategy:**

* **Start with the most complex person**: Dave, having the most connections, simplifies the graph as most relationships become fixed early on.
* **Work in decreasing order of connections**: Handling people with fewer connections later reduces the complexity of choices.
* **Iterative correction**: If something seems wrong, adjusting along the way helps prevent errors from propagating throughout the graph.
* **Symmetry ensures accuracy**: The relation "knows" is symmetric, so checking for symmetry throughout avoids mistakes.

Exercise 1.5 – Obligatory

**A Eulerian Cycle (also known as Eulerian Tour):**  
**A Eulerian Cycle is a path in a graph that**

1. Starts and ends at the same vertex
2. Traverses every edge exactly once

**For a graph to have a Eulerian cycle, the following conditions must be met:**

1. All vertices must have an even degree (i.e., an even number of edges connected to each vertex)
   1. Edges are the connections, basically the lines, whereas the vertices are the nodes, like the points themselves
2. The graph must be connected (there must be a path between any two vertices).

If a graph satisfies these conditions, it is said to have a Eulerian Cycle

**A Eulerian Path is a path in a graph that:**

1. Traverses every edge exactly once, but does not necessarily start and end the same vertex

For a graph to have a Eulerian Path, the following conditions must be met

1. Exactly two vertices must have an odd degree
2. All other vertices must have an even degree
3. The graph must be connected

The path will start at one of the vertices with an odd degree and end at the other

Important points

* Yes, vertices can be visited multiple times as long as you use different edges each time.
* For an Eulerian Path, you must connect the two odd-degree vertices, and all edges must be traversed exactly once.
* **DFA**
  1. DFA - > Language
  2. Strings -> DFA
* **e-NFA**
  1. e-NFA -> DFA (Subset Construction)
     1. Improved
     2. Normal
* **Regular Expression**
  1. Regular Expression -> Tree
  2. Regular Expression -> e-NFA
* **Language**
  1. Languages -> Context Free Grammer
* **EBNF grammar**
  1. EBNF grammar -> context-free grammar
* **Context-Free Grammar**
  1. Context-Free Grammar -> Tree
  2. Context-Free Grammar -> Language