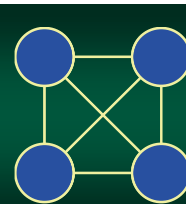
Graph Theory

Part 6

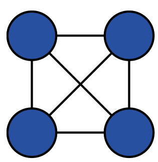
Graphs



Nope, not the same as charts

Graphs

- Lists are just a special case of another structure - the *graph*
- Graphs are the basis for **all** of computer science
- Computer science is not about chips, processors, etc...
- ... this is just implementation technology



Motivation

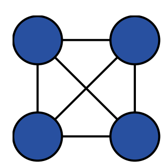
- Several real-life problems can be converted to problems on graphs
- They are one of the pervasive data structures used in computer science
- They are useful tool for modeling real-world problems
- Allows us to abstract details and focus on the problem

Where are Graphs Used?

- The easy answer is: *everywhere*
- In computer science
  - state machines
  - mazes and networks
- Other fields
  - chemistry
  - physics
  - government

Terminology

- The terminology for graphs is a bit different from trees and linked lists
- Rather, it is more generalized
  - "nodes" are called *"vertices"*
  - "branches" or "links" are called *"edges"*

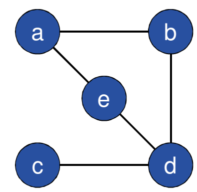


Formal Definition

- A graph $G = (V, E)$ is defined by a pair of two sets
  - a <u>finite</u> set V of items called vertices
  - a <u>finite</u> set E of vertex pairs called edges

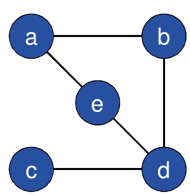
Example Graph

- Set of vertices V
  - a
  - b
  - c
  - d
  - e



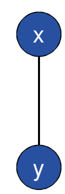
Example Graph

- Set of edges E
  - (a, b)
  - (a, e)
  - (b, d)
  - (c, d)
  - (d, e)



Adjacent and Incident

- If two vertices x and y share an edge (x, y), they are said to be *adjacent*
- The edge (x, y) is called *incident* on vertices x and y



Directed Graph

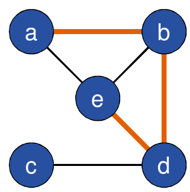
- A *directed graph (digraph)* is a graph where each edge has a source and target vertex
- This is the basis most of the data structures used today:
  - trees
  - linked lists

Undirected Graphs

- Undirected graphs have edges that link both vertices together
- So, the edge has the <u>same</u> meaning for both directions (set rather than tuple)
- Examples:
  - mathematical equality: if a = b then b = a
  - marriage: if Jane is married to Joe, Joe is married to Jane

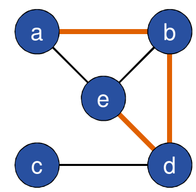
Paths

- A *path* is a sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_n and v_{n+1} are adjacent
- This can represent
  - a physical path
  - logical connection
  - etc...



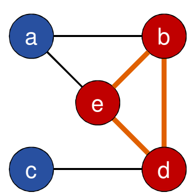
Paths

- In a *simple path*, all edges of a path are <u>distinct</u>
- The *length* of a path is measured by either the total number of edges or vertices



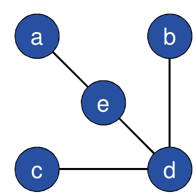
Cycles

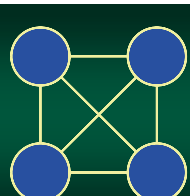
- Unlike linked lists, graphs can have loops
- A *cycle* is a sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_n and v_{n+1} are adjacent and $v_1 = v_k$



Cycles

- An *acyclic graph* contains no cycles
- An acyclic directed graph is often called a *DAG* (Directed Acyclic Graph)



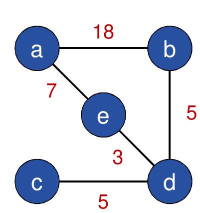


Types of Graphs

Broad categories for a broad topic

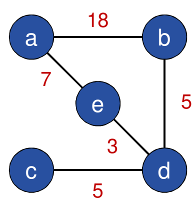
Weighted Graphs

- *Weighted graph* has values on each edge
- These values can be anything – and are defined by the ADT using the graph



Weighted Graph

- Typical uses
  - networks – for finding the fastest speed
  - driving – fastest route
  - etc...
- Example:
  - minimum path from a to b is: a $\rightarrow$ e $\rightarrow$ d $\rightarrow$ b



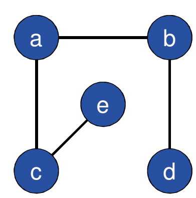
Connected and Unconnected

- A *connected* graph
  - has a path from every vertex to all other vertex
  - so, everything is connected somehow
- An *unconnected* graph
  - at least one vertex exists in which no path exists to another vertex
  - so, there are 2+ sub-graphs that are unlinked

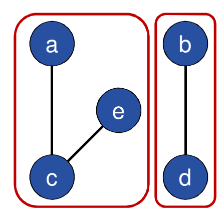
Connected and Unconnected

- The *connected component* is the maximum connected subgraph of a given graph
- If the graph is connected, then the whole graph is one single connected component

Connected and Unconnected



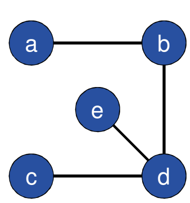
Connected



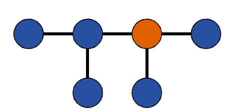
Unconnected

Trees

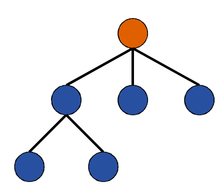
- *Tree* is defined as a connected acyclic graph
- *Forest* is a collection of trees
- *Rooted tree* is a tree graph that selects an arbitrary vertex as the "root"



Trees

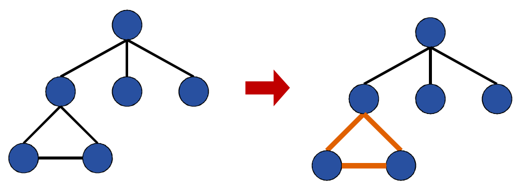


Tree



Same Tree

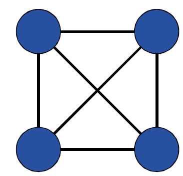
Trees



<u>NOT</u> a Tree

Complete Graphs

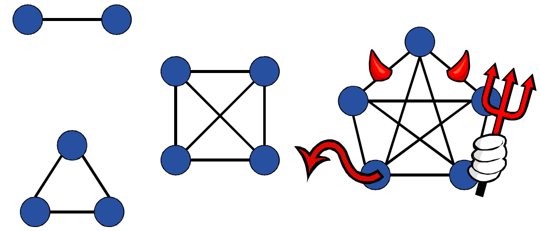
- A *complete graph* is one in which <u>all</u> pairs of vertices are adjacent
- In other words, every vertex is connected to every other vertex

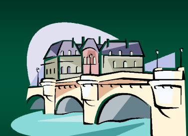


Complete Graphs

- The number of edges in a complete graph...
  - if n is the total number of vertices, each vertex is incident to $n - 1$ edges
  - we can compute $n \times (n - 1)$ edges, but this would count each edge twice!
  - so, the number of edges = $\boldsymbol{n \times (n - 1) / 2}$
- So, for a non-complete graph...
  - number of edges < $\boldsymbol{n \times (n - 1) / 2}$

Example Complete Graphs





Birth of Graph Theory

... and, later, computer science!

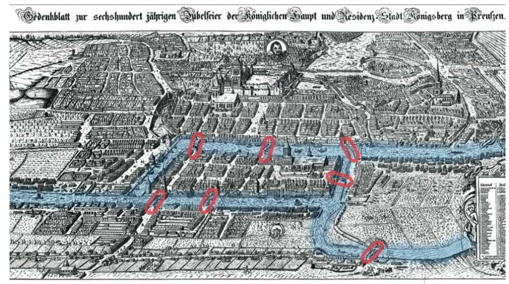
Birth of Graph Theory

- Graph theory was created due to a perplexing question troubling mathematician Leonhard Euler
- Lived in Konigsberg (now "Kaliningrad" in Russia)
  - city occupied 2 islands plus areas on both banks
  - 7 bridges over the Pregel River

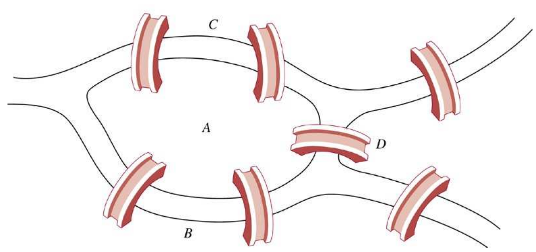
Birth of Graph Theory

- People wondered if they could leave home…
  - cross every bridge exactly <u>once</u>
  - and return to their starting point
  - This is known as the *Konigsberg Bridge Problem*

Konigsberg Bridge Problem



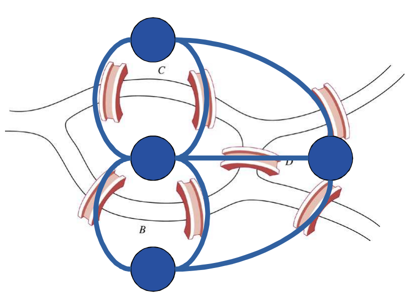
Konigsberg Bridge Problem



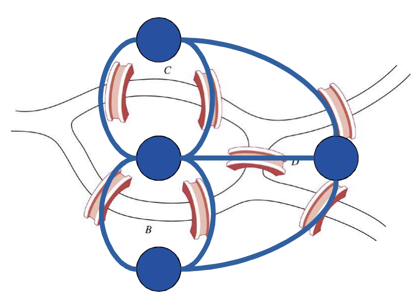
Konigsberg Bridge Problem

- The problem reduces down to a number of points and links to each point
- From this, Euler created the first graph and began the study of their properties

Konigsberg Bridge Problem



Konigsberg Bridge Problem



The Solution to Konigsberg

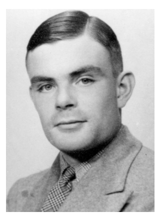
- In 1736, Euler proved that no such traversal exists
- From his work on Konigsberg, a path is named after him
- An *Eulerian circuit,* in a graph...
  - is cycle containing all the edges in the graph
  - and only traversing each edge once

The Solution to Konigsberg

- Euler proved:
  - a graph may have an Eulerian circuit if and only if there are no vertices with an odd number of edges touching them
- Konigsberg Bridge Problem
  - 4 vertices, all with an odd number of edges
  - Sorry people of Konigsberg, there is no solution!

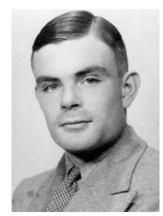
Alan Turing

- Mathematician, logician & cryptographer
- *Father of Computer Science*
  - Highest award in Computer Science is the Turing Award
  - Developed Turing Machines



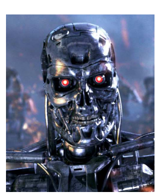
Major Work: Turing Machines

- Invented in 1937
- Logical model – not an actual computer or machine
- Based on 2 graphs (and sets on each of the edge)
- One graph is simple array, but the other could be anything
- From this, he proved programming



Major Work: Turing Test

- Used in artificial intelligence
- Consists of a human operator *texting* a human or computer
- If the operator can't ascertain if it is a computer or human, the computer is "intelligent"
- No computer has passed it

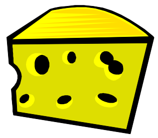


Real World Examples

The origin and the usage

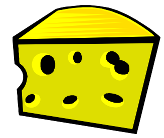
Real World Examples

- How many layers does a computer chip need so that wires in the same layer don't cross?
- How can the season of a sports league be scheduled into the minimum number of weeks?



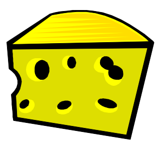
Real World Examples

- In what order should a traveling salesman visit cities to minimize travel time?
- Can we color the regions of every map using four colors so that neighboring regions receive different colors?



Real World Examples

- How can we lay cable at minimum cost to make every network reachable from every other?
- What is the fastest route from the national capital to each state capital?
- How can n jobs be filled by n people with maximum total utility?



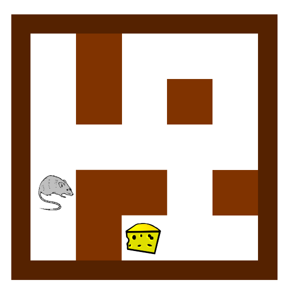
The London Underground Subway

Maze Traversal

- One example of where a graph is useful is a maze traversal
- Basically, any maze can be represented with a graph
- ... and this is not so much different to how networks actually work
- ... a source must find a destination through various vertices

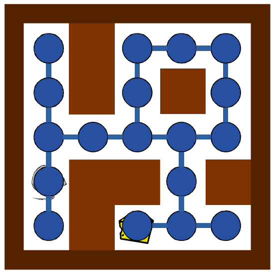
Maze Traversal

- This is a simple maze – though not to the mouse!
- We can help him find the cheese if we convert this to a graph



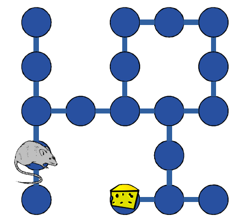
Maze Traversal

- The empty spaces are vertices
- The bordering ones are connected with an edge
- Is this a directed graph?



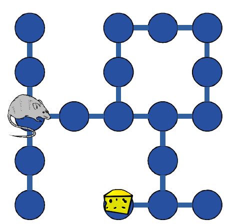
Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order



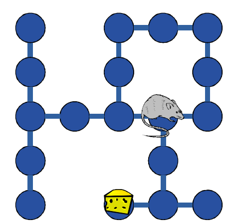
Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order



Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order



Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order