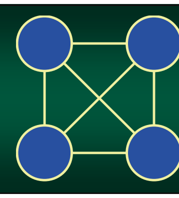
Graphs

Part 14

1

Introduction to Graphs

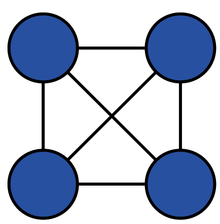


Nope, not the same as charts

2

Graphs

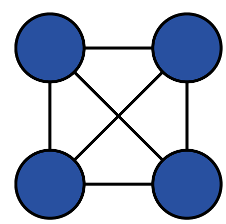
- Lists and trees are just a special case of another structure - the *graph*
- Graphs are the basis for <u>all</u> of computer science



3

Graphs

- Computer science is not about chips, processors, etc...
- ... this is just implementation technology



4

Where are Graphs Used?

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

5

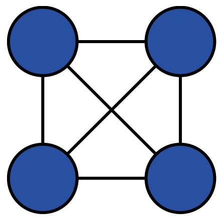
Motivation

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

6

Terminology

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



7

Formal Definition

A graph $G = (V, E)$ is defined by a pair of two sets:

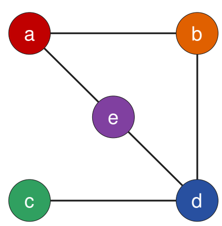
a finite set V of items called *vertices*

a finite set E of vertex ordered-pairs called *edges*

8

Example Graph

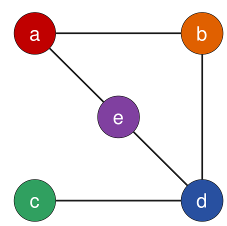
- Set of vertices V
  - a
  - b
  - c
  - d
  - e



9

Example Graph

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



10

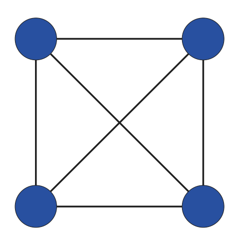
Adjacent and Incident

- When two vertices share an edge (x, y)
  - they are said to be *adjacent*
  - in other words, they are connected
- The edge (x, y)
  - is called *incident* on vertices x and y
  - in other words, it is the connection

11

Complete Graphs

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



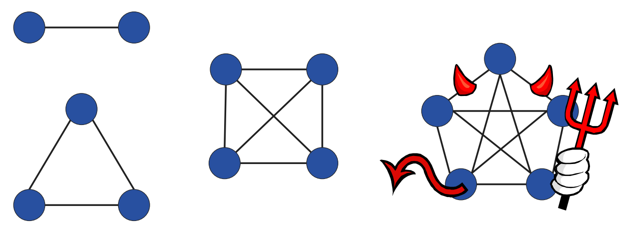
12

Complete Graphs

- The # of edges in a complete graph...
  - if *n* is the total number of vertices, then 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 = $n \times (n - 1) / 2$
- For a noncomplete graph...
  - number of edges < $n \times (n - 1) / 2$

13

Example Complete Graphs



14

Paths

Some edges are one-way streets!

15

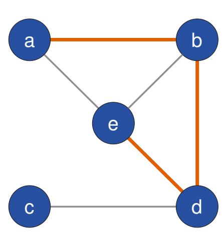
Paths

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

16

Paths

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



17

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

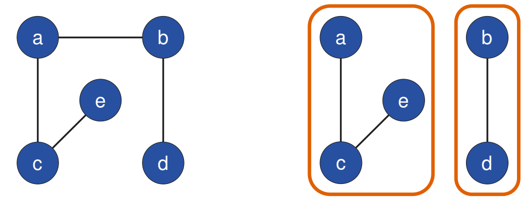
18

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

19

Connected and Unconnected

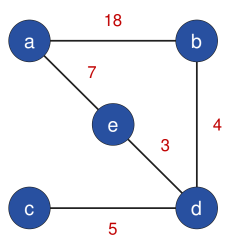


Connected **Unconnected**

20

Weighted Graphs

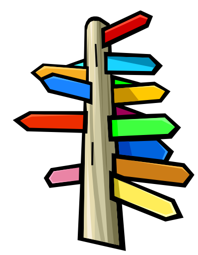
- *Weighted graph* has values on each edge
- These values can be anything – and are defined by the ADT using the graph



21

Weighted Graphs

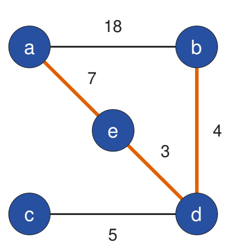
- These weights are abstract and can represent *anything*
- Examples
  - distances – driving, flight paths
  - costs
  - server latency times
  - etc...



22

Minimum Path

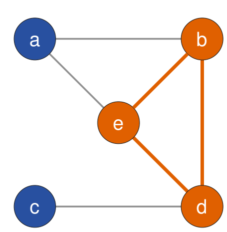
- Often it is useful to find the minimum path – e.g. the smallest sum of edges
- Example: minimum path from a to b is: a → e → d → b
- We'll cover that soon!



23

Cycles

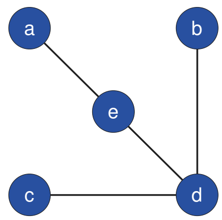
- A *cycle* is sequence of vertices v_1, v_2, …v_k where…
  - v_n and v_{n+1} are adjacent
  - $v_1 = v_k$
- So, it's a loop of vertex



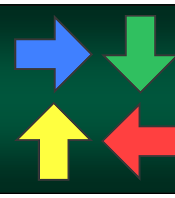
24

Cycles

- An *acyclic graph* does not contain any possible cycles
- This definition will come up later as an essential attribute for some solutions



25



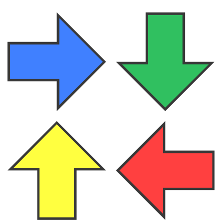
Directed & Undirected

Some edges are one-way streets!

26

Directed & Undirected

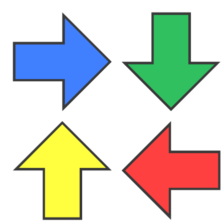
- Sometimes, an edge can be travelled both directions
- However, sometimes, they have a distinct source and destination
- Think of it as streets – some are two-way, and some are one-way



27

Directed Graph

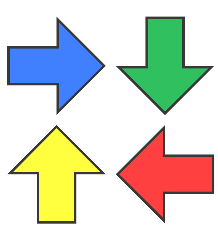
- A *directed graph (digraph)* is a graph where each edge has a source and target vertex
- This is the basis most of the data structures used today:
  - trees
  - linked lists



28

Undirected Graphs

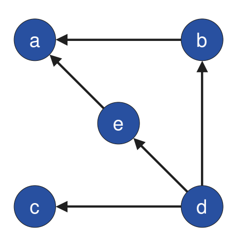
- Undirected graphs have edges that have the same meaning for both directions
- So, if there is an edge (a, b) then there is also an edge (b, a)



29

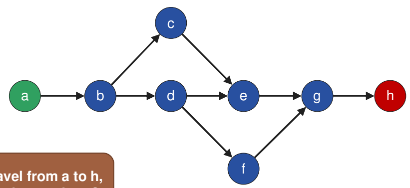
Directed Acyclic Graphs

- When a directed graph lacks cycles, it is called a *Directed Acyclic Graph* (DAG)
- Since edges only travel one way, this leads to some interesting graphs



30

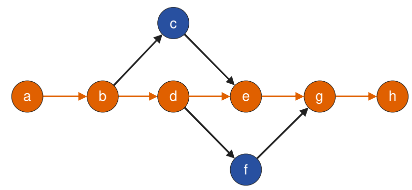
Examine this DAG: Path from a to h



If we want travel from a to h, how many paths are there?

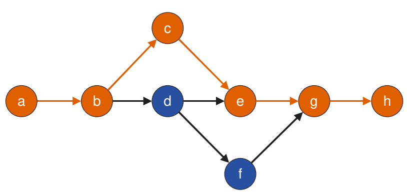
31

Examine this DAG: Path 1



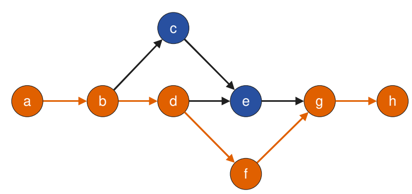
32

Examine this DAG: Path 2



33

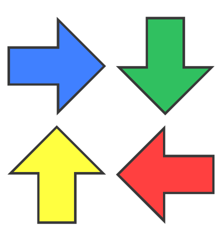
Examine this DAG: Path 3



34

Directed Acyclic Graphs

- In the last example, there are 3 paths from a to h
- Yet, there are **no** cycles
- So, while a DAG contains no cycles, there may be multiple paths from one node to another



35

Trees & Arborescence

A graph is a tree… if…

36

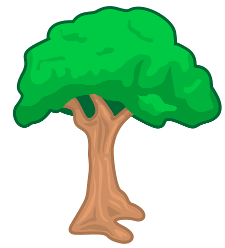
Arboresence

- An *arborescence* is a directed graph where, if there is a path from one node to another, it is **unique**
- So, there is at most one path from a vertex A to B

37

Alternative terms for Arborescence

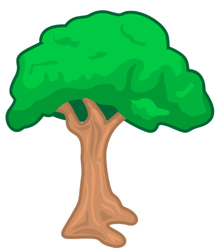
- There are alternative terms (all of which are valid)
- Such as….
  - directed rooted tree
  - out-arborescence,
  - out-tree
  - branching



38

Trees

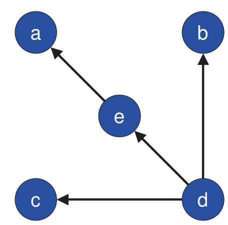
- All trees are acyclic graphs (which may be directed)
- But, not all DAGs are trees
  - DAGS allow multiple paths to the same node
  - since they directed, this won't create cycles



39

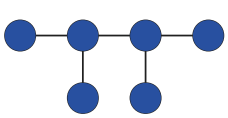
Trees

- *Rooted Tree* selects an arbitrary vertex as the root
- *Forest* is a collection of trees

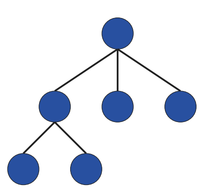


40

Trees



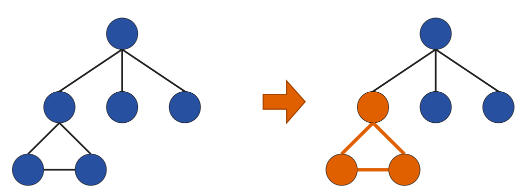
Tree



Same Tree

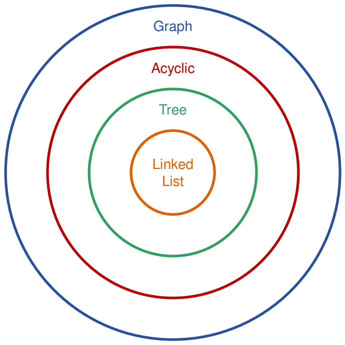
41

Trees



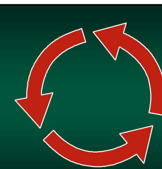
NOT a Tree

42



43

Graph Traversal

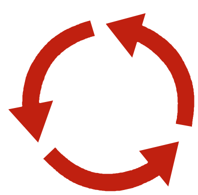


Not as easy as trees

44

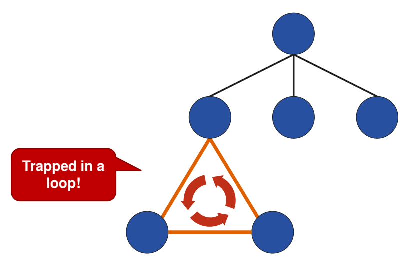
Graph Traversal

- Typically when you search a tree, you can use a simple depth-first search
- However, graphs can contain cycles
- Your program can get stuck in an graph loop and never escape
- This has to be taken into account



45

Graph Traversal



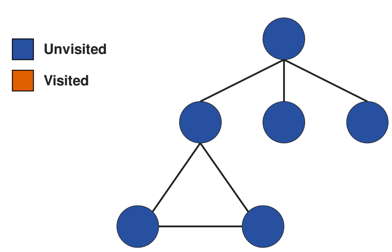
46

Graph Traversal

- So, to before you search…
  - the vertices need to visited only ONCE
  - otherwise, we have a loop
- *Solution:* vertex has a "visited" property
  - before the search, each vertex is set to false
  - when the search visits them, it is set true
  - search never follows an edge to a visited vertex

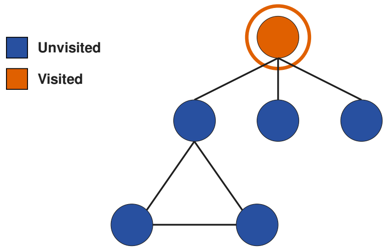
47

Graph Traversal



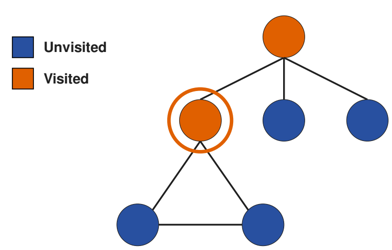
48

Graph Traversal



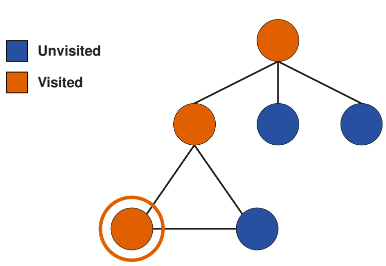
49

Graph Traversal



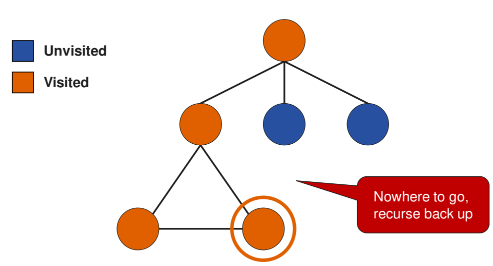
50

Graph Traversal

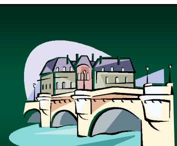


51

Graph Traversal



52



Birth of Graph Theory

... and, later, computer science!

53

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

54

Birth of Graph Theory

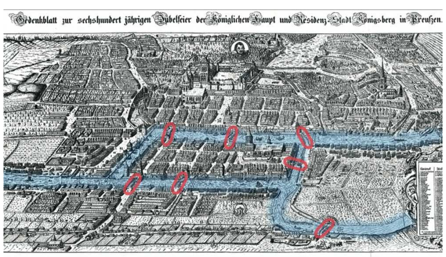
- People wondered they could:
  - leave home…
  - cross every bridge <u>once</u>
  - and return to their starting point
- This is known as the *Konigsberg Bridge Problem* & was unsolved in the 1700's

55

Konigsberg Bridge Problem

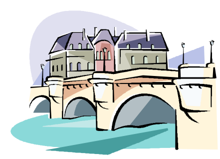
56

Konigsberg Bridge Problem



57

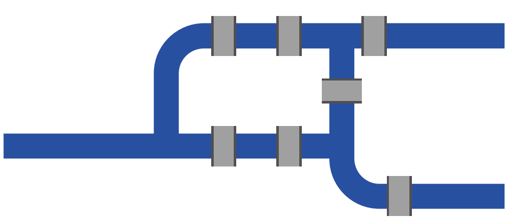
Konigsberg Bridge Problem



- The problem reduces to 4 points and several links to between the points
- From this, Euler created the first graph and began the study of their properties

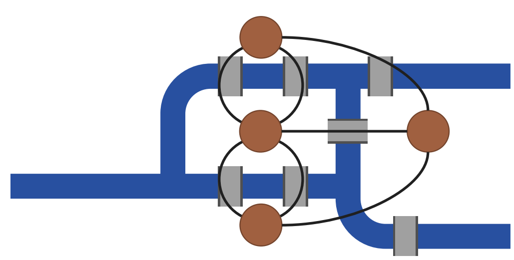
58

Konigsberg Bridge Problem



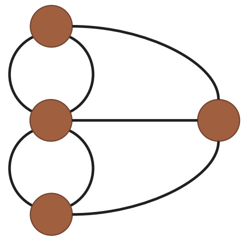
59

Konigsberg Bridge Problem



60

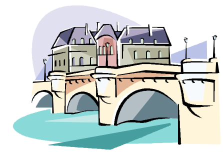
Konigsberg Bridge Problem



61

The Solution to Konigsberg

- In 1736, Euler proved that no such traversal exists
- *Eulerian circuit,* in a graph...
  - is cycle containing all the edges in the graph
  - and only traversing each edge once



62

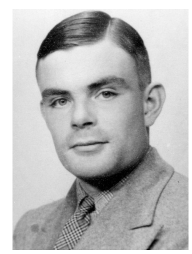
The Solution to Konigsberg

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

63

Alan Turing

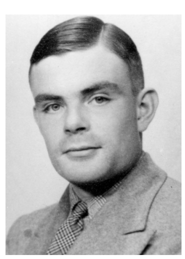
- Mathematician, logician & cryptographer
- *Father of Computer Science*
  - Highest award in Computer Science is the *Turing Award*
  - Developed Turing Machines



64

Major Work: Turing Machines

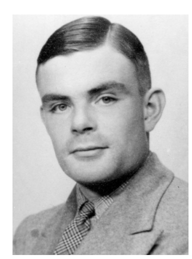
- Invented in 1937
- Logical model – not an actual computer or machine
- Based on 2 graphs (and sets on each of the edge)



65

Major Work: Turing Machines

- One graph is simple array, but the other could be anything
- From this, he proved programming



66

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

67

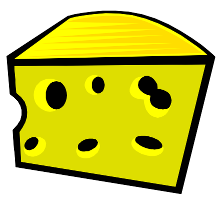
Real World Examples

The origin and the usage

68

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?



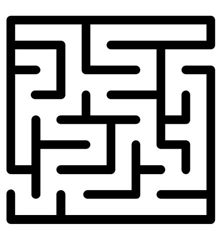
69

The London Underground Subway

70

Maze Traversal

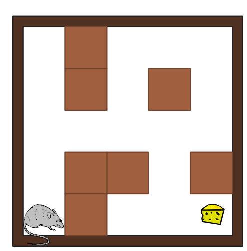
- One example of where a graph is useful is a maze traversal
- Basically, any maze can be represented with a graph
- ... and this is not so much different to how networks actually work
- ... a source must find a destination through various vertices



71

Maze Traversal

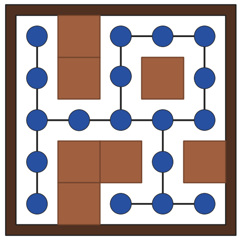
- This is a simple maze – though not to the mouse!
- We can help him find the cheese if we convert this to a graph



72

Maze Traversal

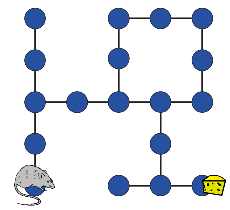
- The empty spaces are vertices
- The bordering ones are connected with an edge
- Is this a directed graph?



73

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



74