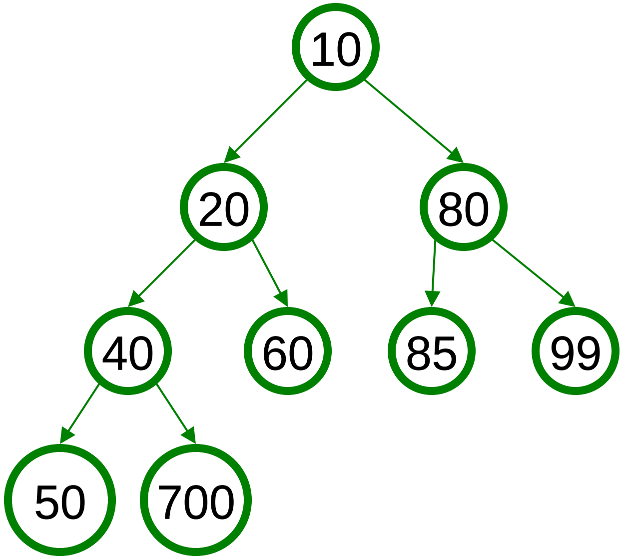
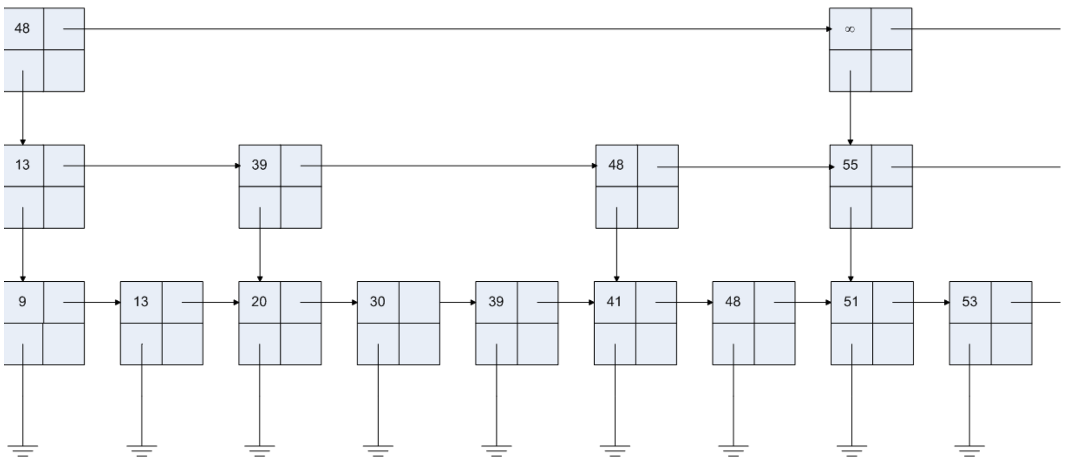
A Few more applications

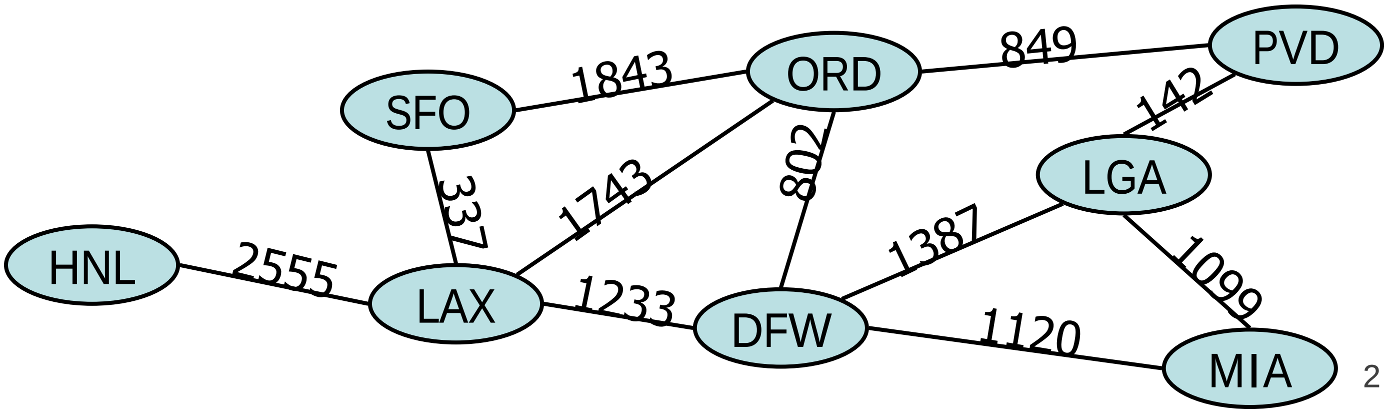
Why Learn Computer Science?

Data structures

- graphs, heaps, skip lists
- balanced trees (AVL, splay, red-black)

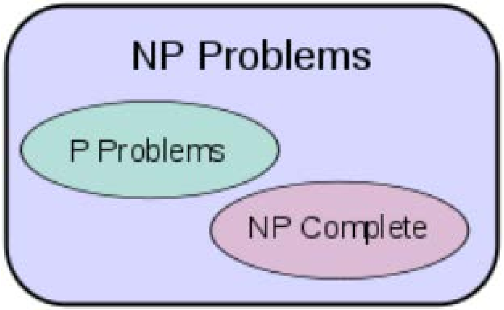






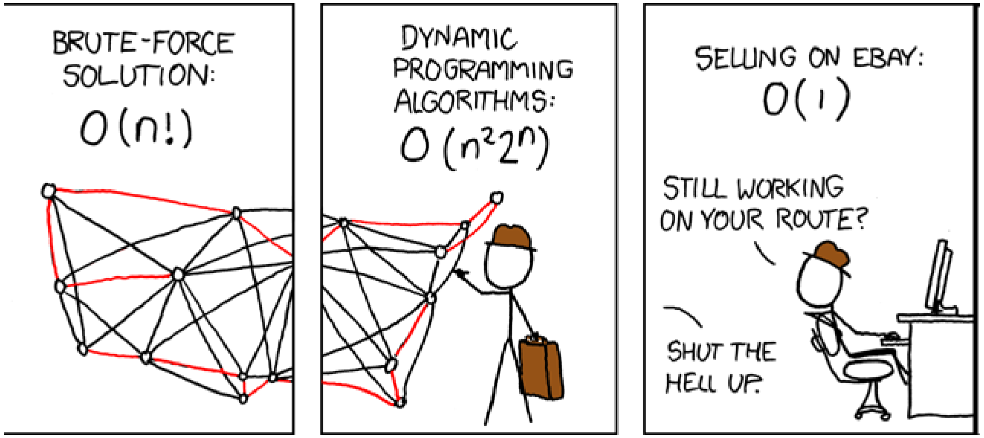
Theory of computation

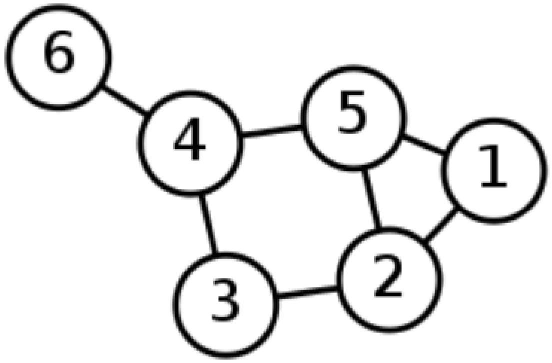
- languages, grammars, and automata
- computational complexity and intractability
  - Big-Oh
  - polynomial vs. exponential time
  - P = NP?



?

- graph theory

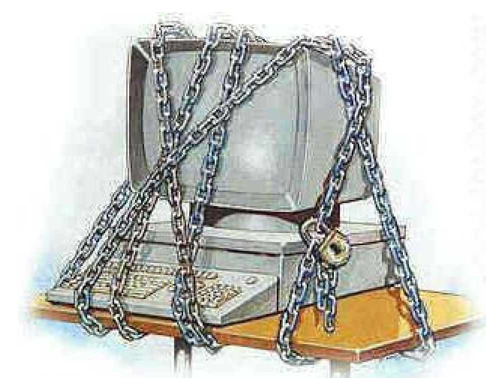




Security

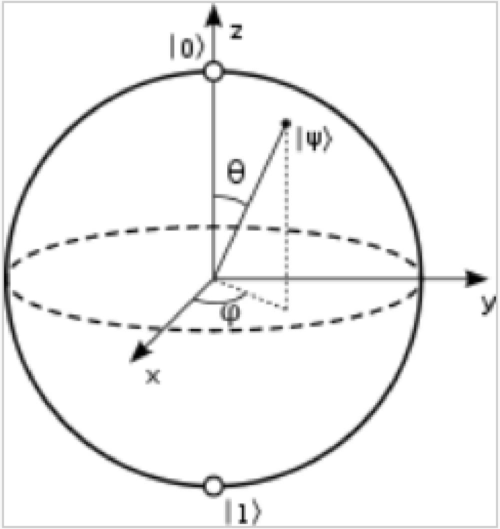
- **cryptography**: study of hiding information
  - enigma machine
  - RSA encryption
  - steganography

- security problems and attacks
  - social engineering
  - viruses, worms, trojans
  - rootkits, key loggers



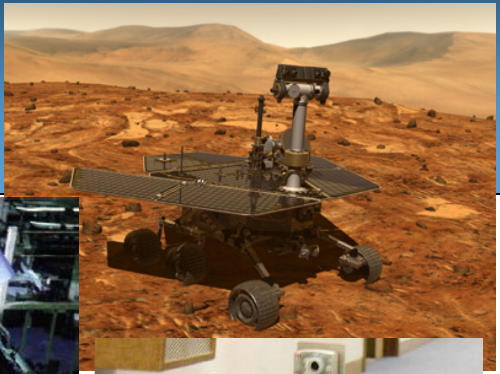
- CSE security course
  - hacking assignment: hack into grades, change from 0 to 100%

Quantum computing

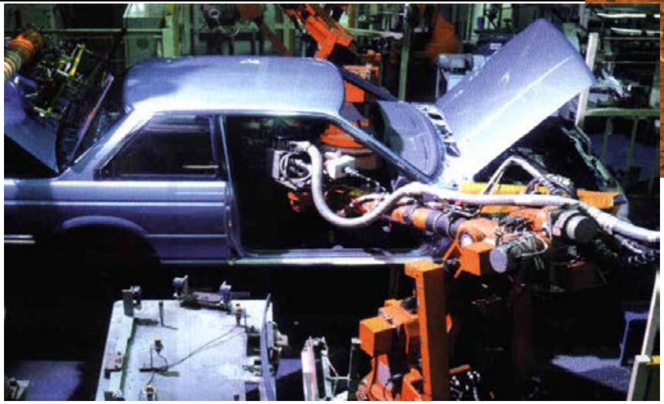


- **qubit**: A particle that can store 0, 1, or any "superposition" between
  - a bit that can sort of be 0 and 1 at once
  - **quantum computer**: uses qubits, not bits
  - theoretically makes it possible to perform certain computations very quickly
    - Example: factoring integers (why is that useful?)
  - actual implementation still in its infancy
    - can add single-digit numbers; can factor 15

Robots

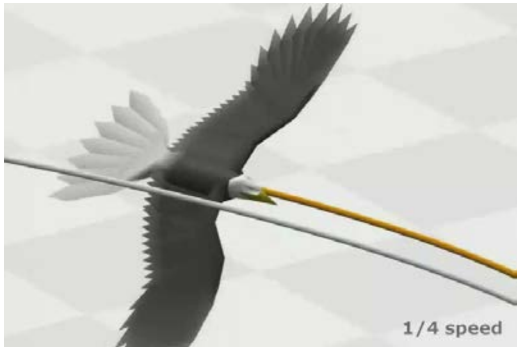


- toys, building cars, vacuums, surgery, search and rescue, elder care, exploration

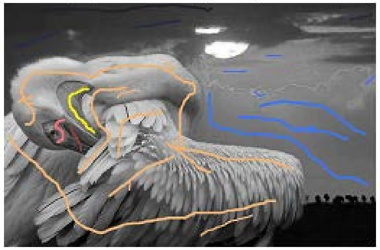


Graphics and vision

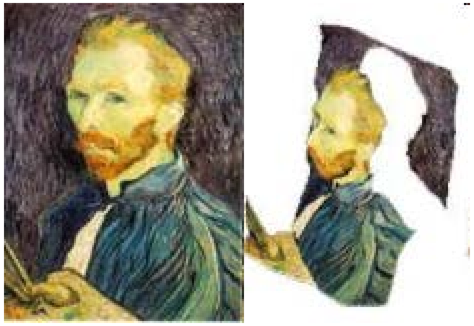
- GRAIL (Graphics and AI Lab)
- computer vision
- AI and the Turing Test

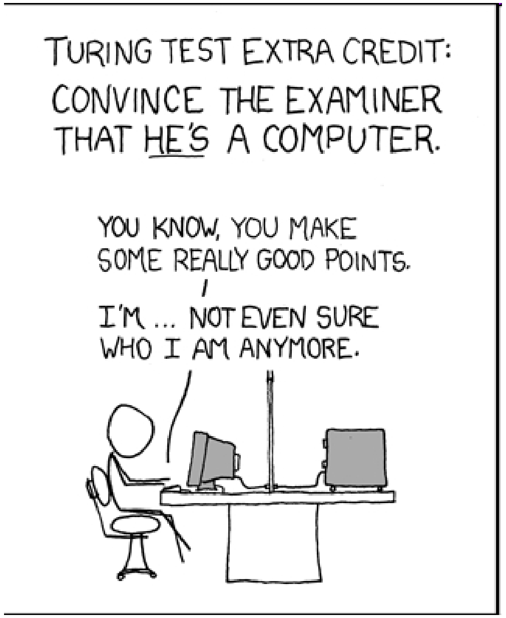


(c) Psuedo relighting filter

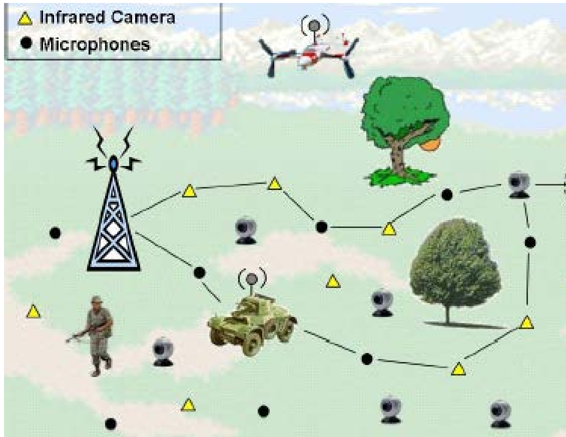


High dynamic range | Enhanced exposure | Object removal

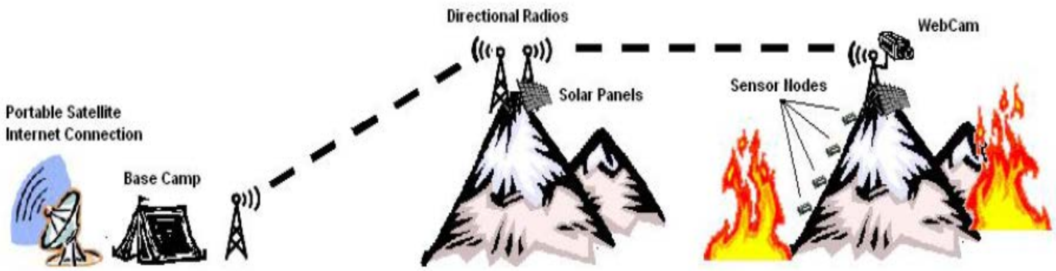




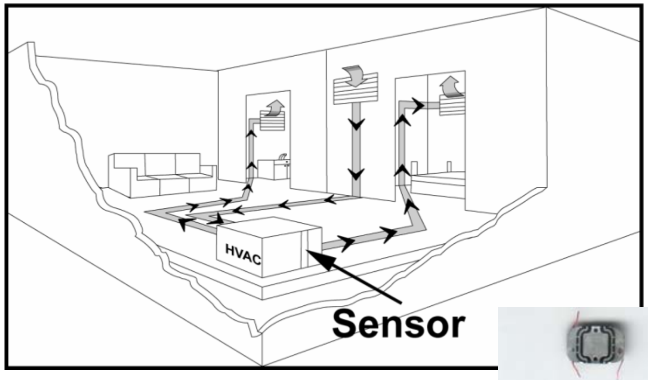
Sensor networks



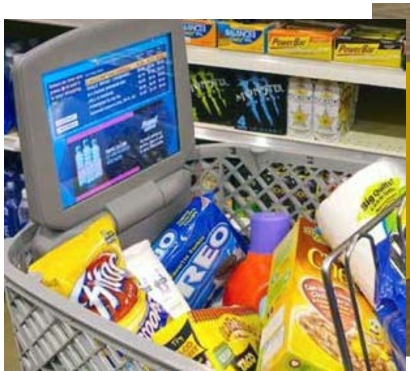
- Environment monitoring
- Military Intelligence

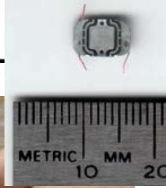


- Intelligent homes
  - detecting human activity through device usage / voltage



- radio freq. identification (RFID)
  - shopping, inventory
  - credit cards, toll roads, badges



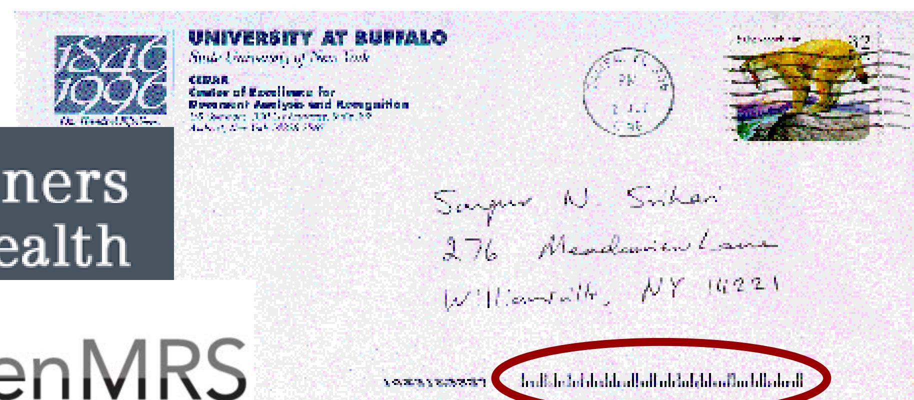


Data mining

- **data mining**: extracting patterns from large data sets
  - What do these two lists have in common?
    - coughing, rash, high fever, sore throat, headache, heartburn
    - V14GR4, cheap meds, home loans, Nigeria, lower interest rate
  - And what does it have to do with sorting your mail?
    (90% of mail is sorted automatically)
    - http://www.usps.com/strategicplanning/cs05/chp2_009.html (2005)

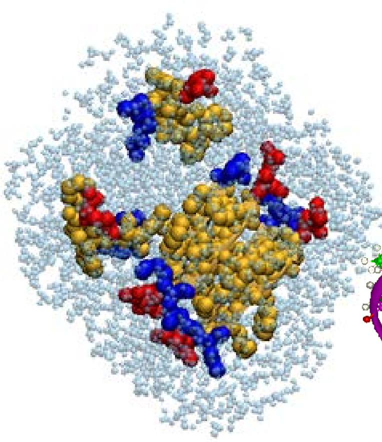
Partners In Health

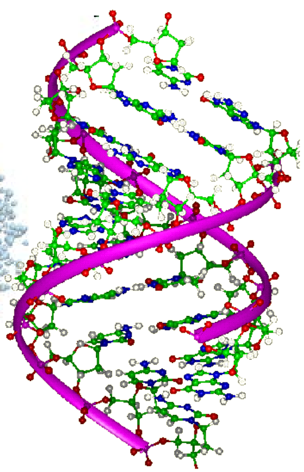
OpenMRS MEDICAL RECORD SYSTEM

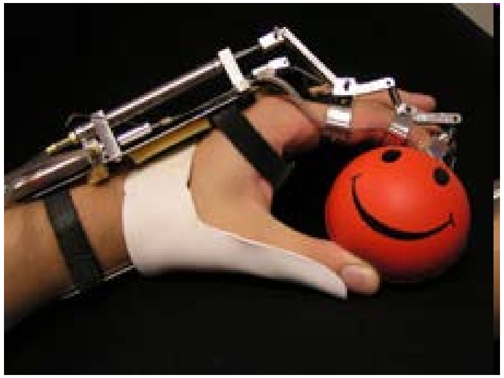


Science and medicine

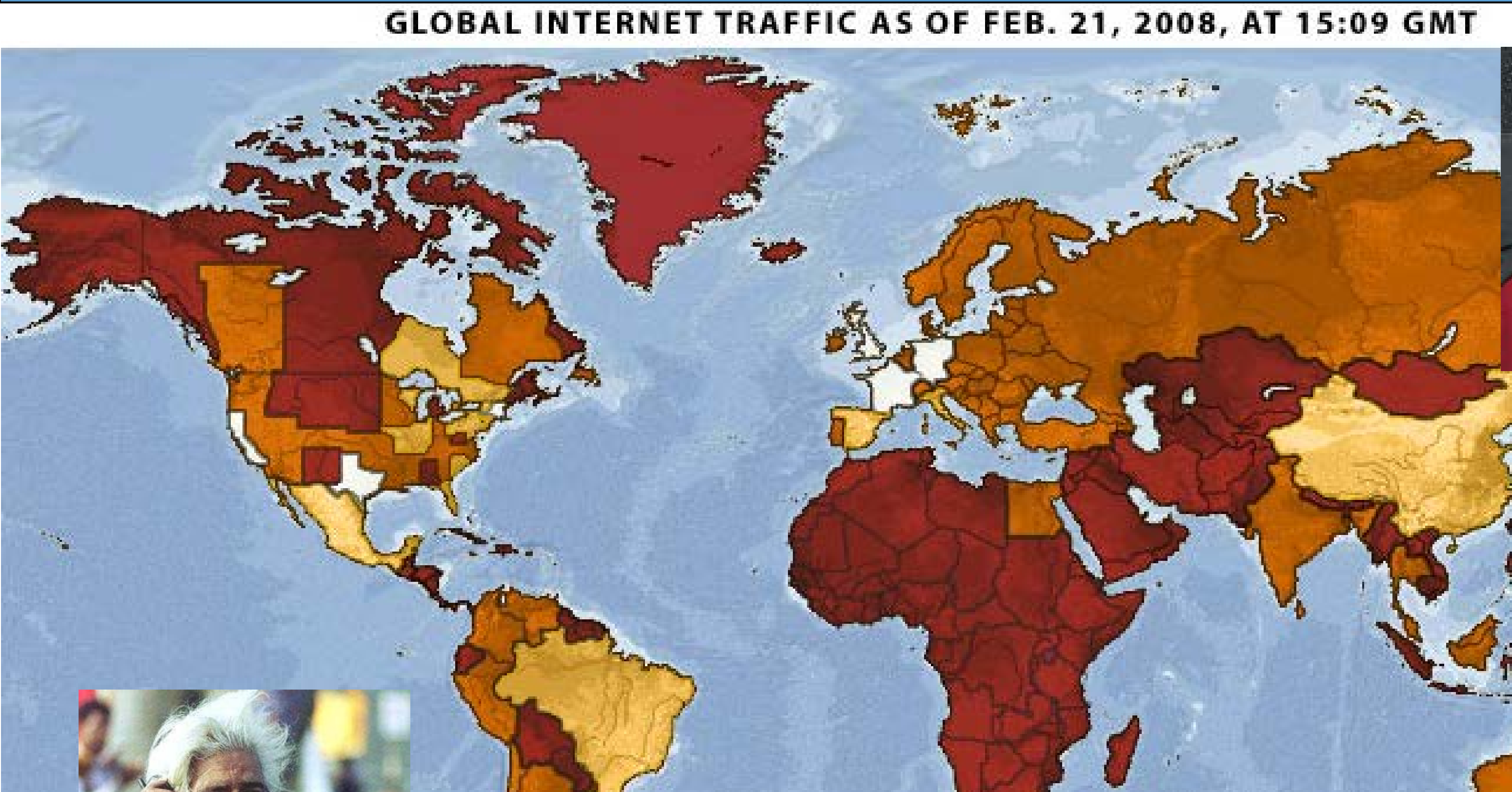
- computer <u>science</u>
  - **bioinformatics**: applying algorithms/stats to biological datasets
  - **computational genomics**: study genomes of cells/organisms
  - **neurobotics**: robotic brain-operated devices to assist human motor control
  - assistive technologies

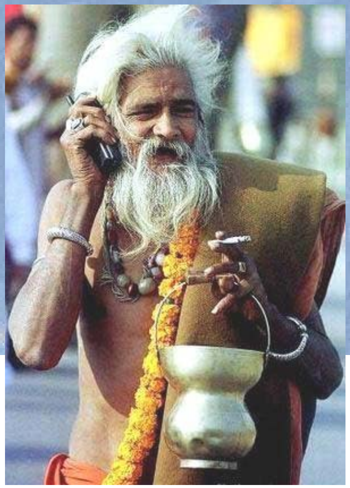






The developing world

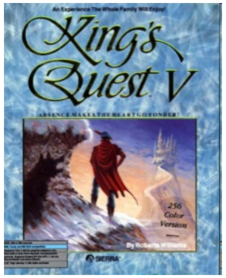




One Laptop Per Child (OLPC)
Mary Lou Jepsen, CTO

Experience optional

- Mark Zuckerberg, Facebook
  - side project while soph. CS major at Harvard
    - in 2 weeks, 2/3 of Harvard students joined
- Bill Gates started "Micro-Soft" at age 20
- Larry Page / Sergei Brin, Google
  - made "BackRub" search at age 23
- Roberta Williams, Sierra
  - pioneer of adventure gaming



Trees

But first a few python basics

Tuples Revision

- Ordered collection
- Accessed by offset
- Immutable
- Heterogeneous, Nestable
- Arrays of object references
- To get help use:
  - help(())
  - dir(())
- Example:

```
>>> T = ("VZ",110,26.75)
```

Tuples

Operation	Interpretation
`()`	An empty tuple
`T = (0,)`	A one-item tuple (not an expression)
`T = (0, 'Ni', 1.2, 3)`	A four-item tuple
`T = 0, 'Ni', 1.2, 3`	Another four-item tuple (same as prior line)
`T = ('abc', ('def', 'ghi'))`	Nested tuples
`T = tuple('spam')`	Tuple of items in an iterable
`T[i]`	Index, index of index, slice, length
`T[i][j]`	
`T[i:j]`	
`len(T)`	
`T1 + T2`	Concatenate, repeat
`T * 3`	
`for x in T: print(x)`	Iteration, membership

Tuples

```
('red','green')
(‘x',)       # 1-item tuple
(1,) != (1)
()           # empty tuple
```

```
help(())
 dir(())
```

```
X = (1,2,3,4)
X[2]               # -> 3
(1,2,3,4)[1:3]     # -> (2,3)
(1,2)[2] = 5       # Error!

(a,b,c) = (1,2,3)
(a,b,c) =  1,2,3
 a,b,c  = (1,2,3)
 a,b,c  = [1,2,3]

 a,b = b,a  # swap
```

```
for i,c in [(1,'I'), (2,'II), (3,'III')]:
    print(i,c)

# vector addition
def add(v1, v2):
    x,y = v1[0]+v2[0], v1[1]+v2[1]
    return (x,y)
```

Tuples

- Why Tuples when list exists?

- Efficiency
- Lists – optimized for appends()
  - Uses more memory

- Integrity – tuples can't change.
- Tuples can be used as dictionary keys, Lists can't.

Type Classification

Object type	Category	Mutable?
Numbers (all)	Numeric	No
Strings	Sequence	No
Lists	Sequence	Yes
Dictionaries	Mapping	Yes
Tuples	Sequence	No
Files	Extension	N/A
Sets	Set	Yes
frozenset	Set	No
bytearray (3.0)	Sequence	Yes

Sets

- Mutable
- Can only contain immutable types
- frozenset = Immutable version of sets

- Construction

```
>>> set('orange')
set(['a', 'e', 'g', 'o', 'n', 'r'])
>>> s = set(['vz', 110, 26.75])
>>> s
set([26.75, 'vz', 110])
>>> s = {1,2,3,4}
>>> s
set([1, 2, 3, 4])
```

19

Set Operations

```
>>> x = set('bat')
>>> y = set('ball')
>>> 'b' in x             # Membership
True
>>> x - y                # Difference
set(['t'])
>>> x | y                # Union
set(['a', 'b', 't', 'l'])
>>> x & y                # Intersection
set(['a', 'b'])
>>> x ^ y                # Symmetric Difference (XOR)
set(['l', 't'])
>>> x > y, x < y         # Superset, subset
(False, False)
```

20

Set Operations

```
>>> z = x.intersection(y)        # x & y
>>> z
set(['a', 'b'])
>>> z.add('call')                # Insert an item
>>> z
set(['a', 'b', 'call'])
>>> z.update( {'X', 'Y'} )       # Merge: In-place union
>>> z
set(['a', 'Y', 'b', 'call', 'X'])
>>> z.remove('X')                # Delete an item
>>> z
set(['a', 'Y', 'b', 'call'])
>>> sorted(z)
['Y', 'a', 'b', 'call']
>>> sorted(z, key=str.lower)
['a', 'b', 'call', 'Y']
>>> L = [1,1,2,3,4,5,4,6,6,5]
>>> list(set(L))
[1, 2, 3, 4, 5, 6]
```

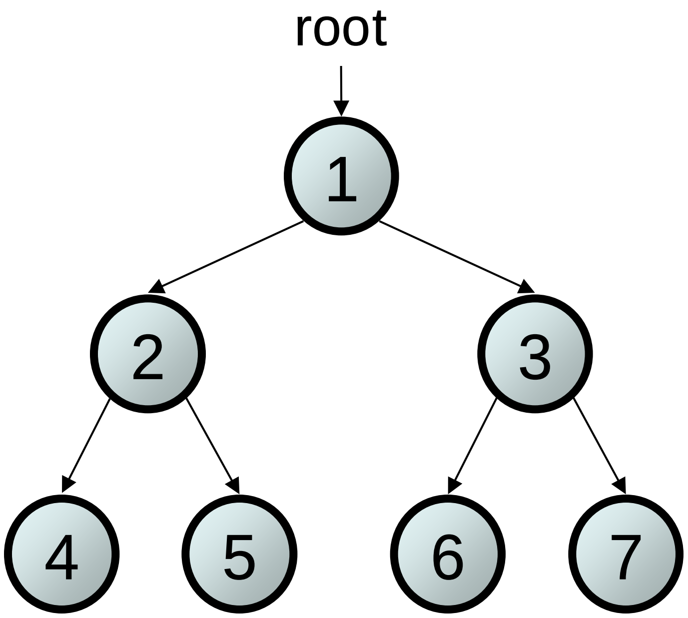
Usual operations still work: max(), min(), len(), sum(), help(set), help(set.add)

for x in S: print x

Trees

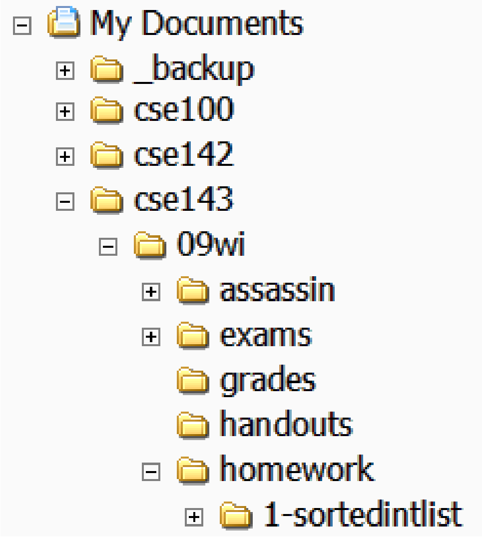
- **tree**: A directed, acyclic structure of linked nodes.
  - *directed* : Has one-way links between nodes.
  - *acyclic* : No path wraps back around to the same node twice.
  - **binary tree**: One where each node has at most two children.

- A tree can be defined as either:
  - empty (`null`), or
  - a **root** node that contains:
    - **data**,
    - a **left** subtree, and
    - a **right** subtree.
      - (The left and/or right subtree could be empty.)

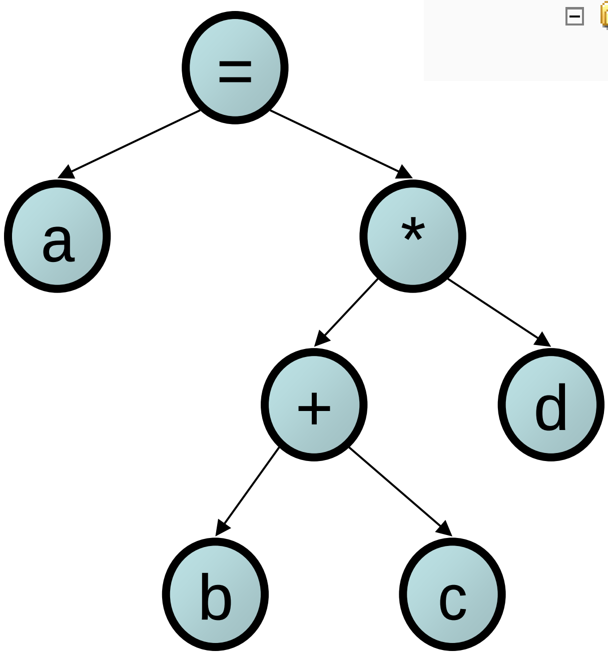


Trees in computer science

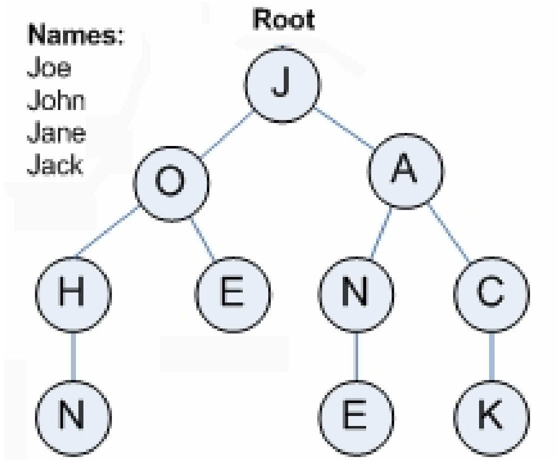
- folders/files on a computer



- family genealogy; organizational charts
- AI: decision trees
- compilers: parse tree
  - a = (b + c) * d;



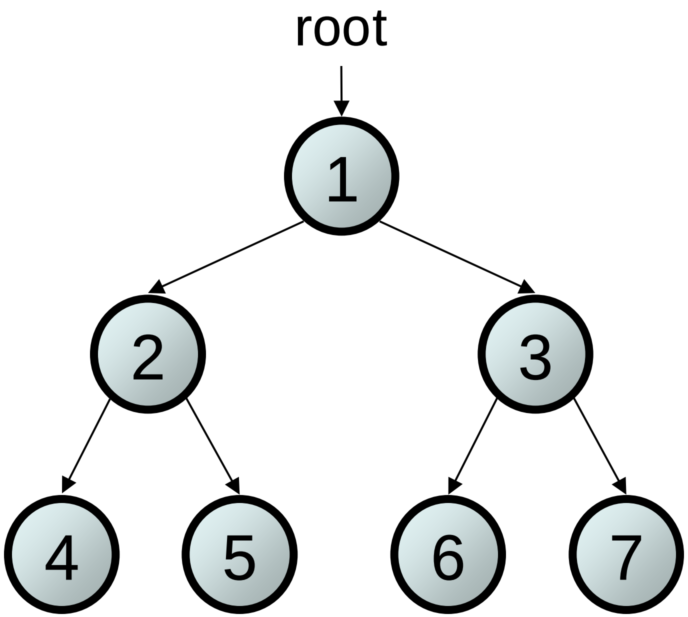
- cell phone T9



Terminology

- **node**: an object containing a data value and left/right children
- **root**: topmost node of a tree
- **leaf**: a node that has no children
- **branch**: any internal node; neither the root nor a leaf

- **parent**: a node that refers to this one
- **child**: a node that this node refers to
- **sibling**: a node with a common



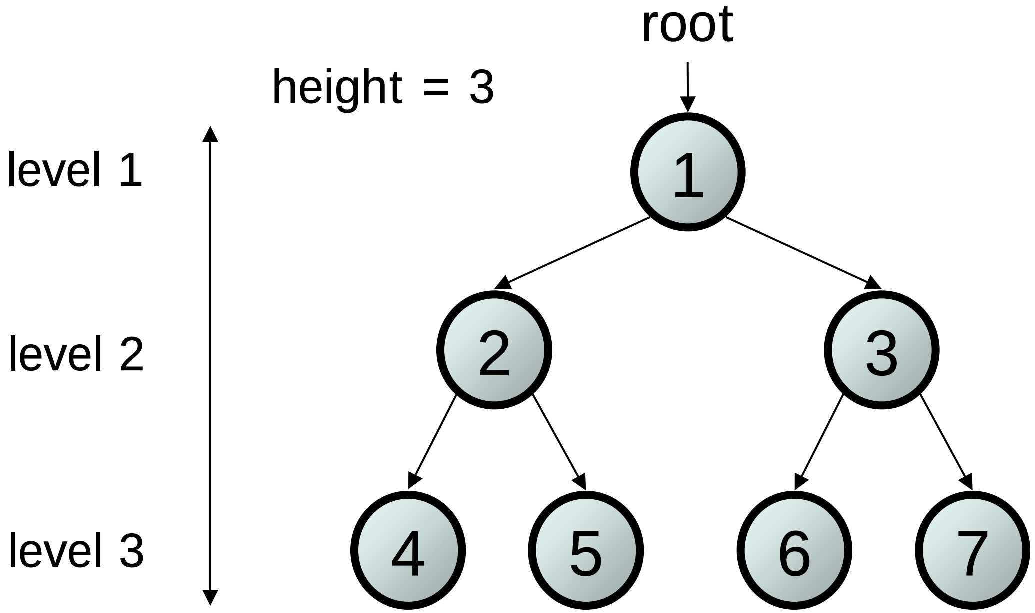
Terminology 2

- **subtree**: the tree of nodes reachable to the left/right from the current node

- **height**: length of the longest path from the root to any node

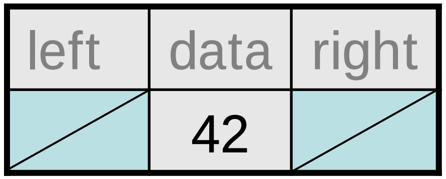
- **level** or **depth**: length of the path from a root to a given node

- **full tree**: one where every branch has 2 children

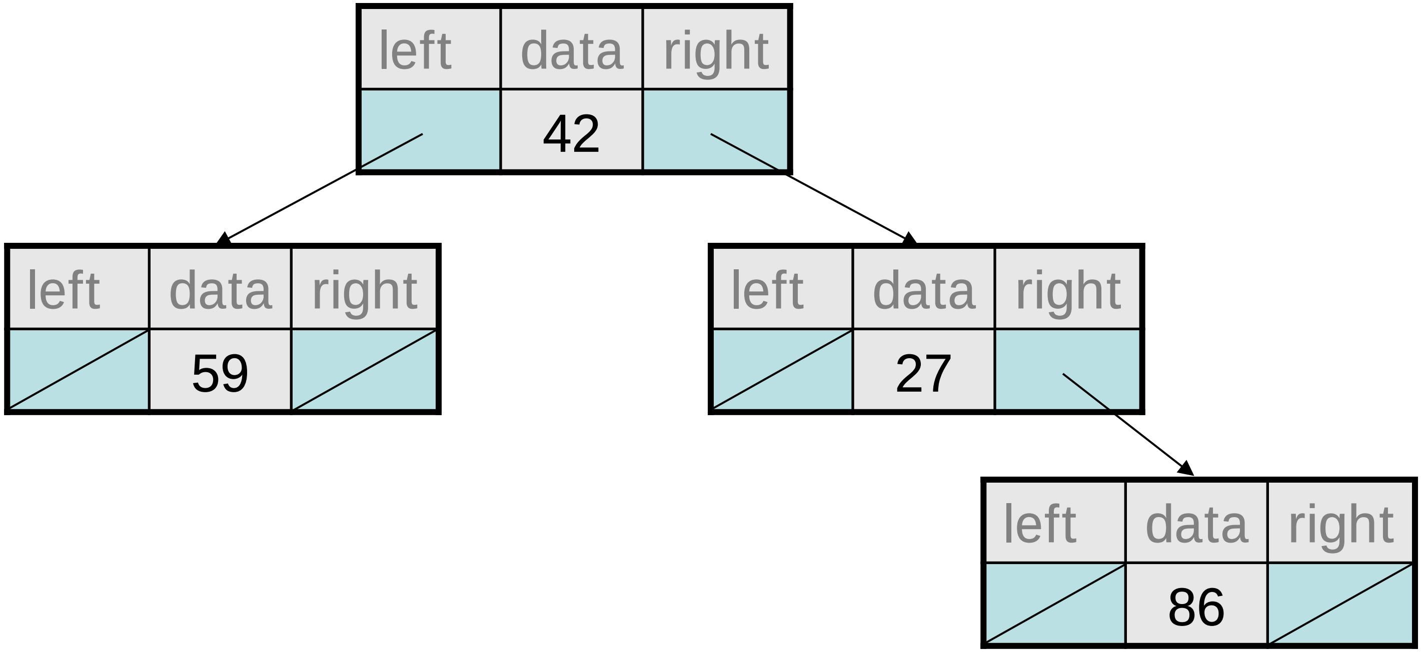


A tree node for integers

- A basic **tree node object** stores data and refers to left/right



- Multiple nodes can be linked together into a larger tree



API

BINARY TREE METHOD	WHAT IT DOES
`T = BinaryTree(item)`	Creates a new binary tree with **`item`** as the root and empty left and right subtrees. This is essentially a leaf node.
`T.__str__()`	Same as **`str(T)`**. Returns a string representation of the tree that shows its structure.
`T.isEmpty()`	Returns **`True`** if **`T`** is empty, or **`False`** otherwise.
`T.preorder(aList)`	Performs a preorder traversal of **`T`**. *Postcondition*: the items visited are added to **`aList`**.
`T.inorder(aList)`	Performs an inorder traversal of **`T`**. *Postcondition*: the items visited are added to **`aList`**.
`T.postorder(aList)`	Performs a postorder traversal of **`T`**. *Postcondition*: the items visited are added to **`aList`**.

continued

API

`T.levelorder(aList)`	Performs a level order traversal of **`T`**. *Postcondition*: the items visited are added to **`aList`**.
`T.getRoot()`	Returns the item at the root. *Precondition*: **`T`** is not an empty tree.
`T.getLeft()`	Returns the left subtree. *Precondition*: **`T`** is not an empty tree.
`T.getRight()`	Returns the right subtree. *Precondition*: **`T`** is not an empty tree.
`T.setRoot(item)`	Sets the root to **`item`**. *Precondition*: **`T`** is not an empty tree.
`T.setLeft(tree)`	Sets the left subtree to **`tree`**. *Precondition*: **`T`** is not an empty tree.
`T.setRight(tree)`	Sets the right subtree to **`tree`**. *Precondition*: **`T`** is not an empty tree.
`T.removeLeft()`	Removes and returns the left subtree. *Precondition*: **`T`** is not an empty tree. *Postcondition*: the left subtree is empty.
`T.removeRight()`	Removes and returns the right subtree. *Precondition*: **`T`** is not an empty tree. *Postcondition*: the left subtree is empty.

[TABLE 18.3] The operations on a binary tree ADT