Elementary Set Theory

Peeking into Computer Science

- Mandatory: Chapter 2 – Sections 2.3 and 2.4

Reading Assignment

Relations

Relating sets

At the end of this section, you will be able to:

1. Define relations and represent them in two ways (sets and graphical)
2. Understand the property of symmetric relations
3. Identify which relation is a function

Objectives

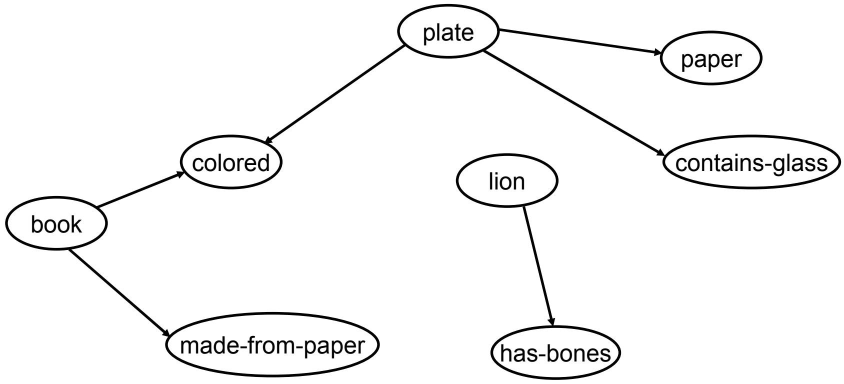
- A relation from A to B is a subset of A x B
- Example
  - Let A = {book, lion, plate}
  - Let B = {colored, made-from-paper, has-bones, contains-glass}
  - The association of objects A with properties B is a relation from A to B
  - R = {(book, colored), (book, made-from-paper), (lion, has-bones), (plate, colored), (plate, made-from-paper), (plate, contains-glass)}
- In general, A relation on A_1 , A_2 , … , A_n is a subset of A_1 x A_2 x … x A_n

Relations

- Recall from the InfoViz slides (JT's extra info on when to use images, what type etc.)
- Graphical representations work well for showing how things are inter-related.
- Consequently it may be easier to show relations between sets in the form of graph (image).

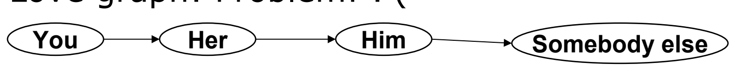
JT's Extra: Relations And Pictures

JT's Extra: Relation Graph (Previous Example)

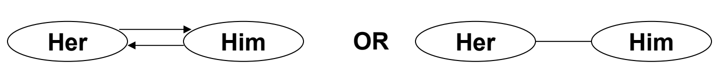


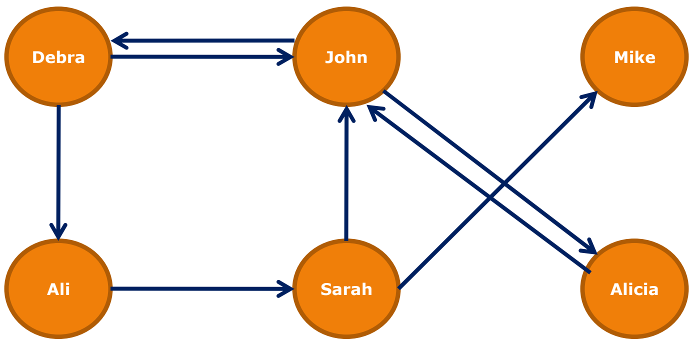
Set Relations: Types

- Relations can be directed (one way) as the previous example and the example below.
  - Love graph: Problem! :'(



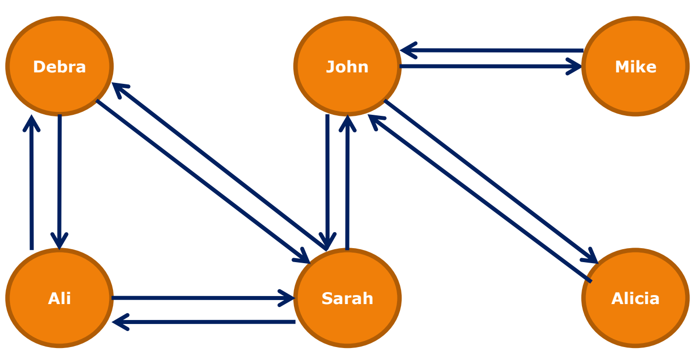
- Relations can also be symmetric (two way the same e.g., example below).
  - Love graph: Life is good! :D





“Likes” Relation Graph

- Whenever (a,b) □ R, implies that (b,a) A □ R, the relation is called *symmetric*



Symmetric Relations

- Has-the-same age
- Has-the-same name
- Children of a common parent

Symmetric Relations

- A function f from A to B
- Written *f:* A → B
- Is a relation from A to B, such that:
- There is exactly one pair (a,b) ∈ *f* for each a ∈ A
- We write *f*(a) = b

Functions

Functions

- A = {1,2,3}, B={yes,no}
- {(1,yes), (2,yes), (3,no)} is a function
- {(1,yes) (2,no), (2,yes), (3,no)} is not a function
- {(2,yes), (3,no)} is not a function

Functions

- Functions "return values"
- *f*(x) = 2x
- *f*(1) returns 2
- *f*(20) returns 40

Algorithms

Computer recipes

At the end of this section, you will be able to:

1. Understand what algorithms are and how to specify them
2. Understand the importance of algorithms to computers
3. Understand what pseudo-code is
4. Develop and specify an example algorithm using pseudo-code
5. Differentiate between the correctness and efficiency of algorithms
6. Work out an example to assess the efficiency of algorithms

Objectives

- A **problem** is a specification of the relationship between input and output
  - WHAT needs to be done, not HOW
- Cooking problem
  - Input: ingredients
  - Output: cooked food
- **Computational problems** are the ones to be carried out by computers

Problems

- The algorithm specifies how to solve the problem
  - How to convert the input to output
- Cooking problem
- Cooking recipe (“algorithm?”)
- Algorithms are specific to Computational problems

Algorithms

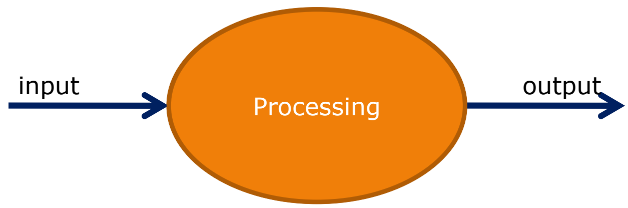
- 780-850
- Left behind the most important Math book
  - Kitab Al Jaber Wal Mokaballa
  - The book of Restoration and Comparison
  - Al Jaber (restoration) became Algebra
- When his book was translated to Greek, it was called: Thus said Algorismi
- With time, it became algorithmi

Mohamed Bin Musa Al Khawarizmi

- Was given to mathematical operations used by merchant's to simplify their accounting

- Today, it is used to mean: *computational* procedure that accepts **input**, processes it, and produces **output**

'Algorithm'



Modern 'Algorithm'

- '***algorithm***' for washing your hair

1. Wet your hair
2. Open the shampoo bottle
3. Pour 5ml of shampoo to your palm
4. Apply to hair and rub for 1 minute
5. Rinse hair with water

- Not really an algorithm, unless performed by a computer

Computational?

Specifying Algorithms

- Code (Computer program)
  - Python and Jython
  - Java
  - C++
- Finite State Machine
- Pseudo-code
  - English like code
- Flow charts
  - Visual representation

Min Algorithm

- Given a set of objects (numbers)
- How do we find the smallest member of the set?
- Example input : {10, 46, 3, 100}
- Output: 3

Given input S, a set

1. Let the first element in S be *min-so-far*
2. Check if the next element is less than *min-so-far*
   - If so, make it the *min-so-far*
   - If not ignore it
3. Repeat step 2 until all elements in S have been examined
4. The minimum is *min-so-far*

Attempt

- What if the input set is empty?
- What if the input set is a *singleton* (contains one element only)?

Problems with the attempt

The Min Algorithm

Input: set S
Output: minimum element in S

1. If S is empty, stop, there is no min
2. Otherwise, let the first element be MSF
3. Repeat the following until all elements in S have been examined:
   check if the next element is less than MSF
   - If so, make it the MSF
   - If not ignore it
4. The minimum is MSF

Correctness vs. Efficiency

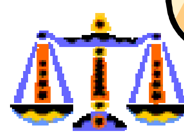
- Algorithms must be correct
  - Solve a problem, taking care of all special cases
- Algorithms should be efficient
  - Take less time to compute solution (time-efficiency)
  - Use less space (space-efficiency)
  - Use less power (power-efficiency)
  - Use fewer message (message-efficiency)
- Here, we only focus on time-efficiency

Example

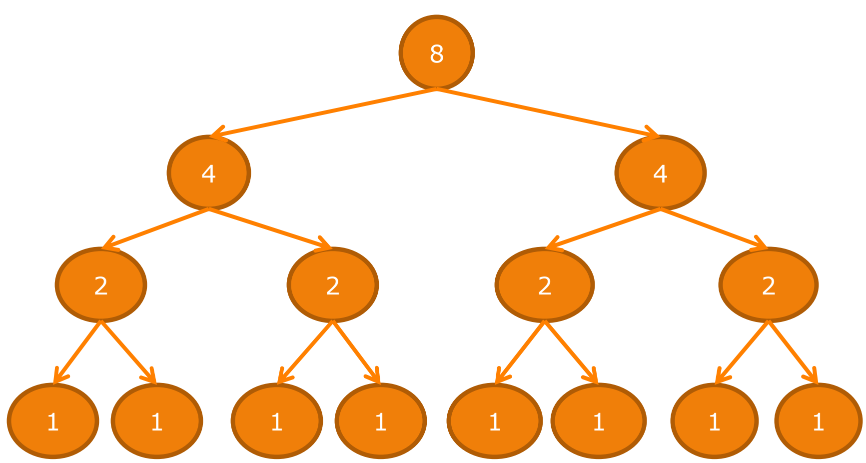
- A merchant learnt that one of his 4096 golden coins is fake
- All coins look exactly the same (including the fake one)
- The only difference between a fake and a real coin is in the weight
- A fake coin is lighter

- Find an efficient algorithm to locate the fake coin

- Hint: Can be done in 12 comparisons



A partial tree



- If you compare each pair of coins, you may need 4096/2 = 2048 "operations"

1. Weigh 2048 against 2048, pick the lighter pile
2. Weigh 1024 against 1024, pick the lighter pile
3. Weigh 512 against 512, pick the lighter pile
4. Weigh 256 against 256, pick the lighter pile
5. Weigh 128 against 128, pick the lighter pile
6. Weigh 64 against 64, pick the lighter pile
7. Weigh 32 against 32, pick the lighter pile
8. Weigh 16 against 16, pick the lighter pile
9. Weigh 8 against 8, pick the lighter pile
10. Weigh 4 against 4, pick the lighter pile
11. Weigh 2 against 2, pick the lighter pile
12. Weigh 1 against 1, pick the lighter **coin**

Efficient Algorithms

- It can be shown that the height of this binary tree is $\mathbf{log_2 n}$, where n is the number of coins

- $\mathbf{log_2 4096 = 12}$
- **@#$$&^&)(&*^$##$%^$#**

Logarithmic Time