What do gambling, magic, and human evolution have in common?

Limsoon Wong

Fun With Invariants

Limsoon Wong

Plan

- **What is an invariant?**
  - Bet on color of the bean
  - 21 cards
- **Origin of Polynesians**
- **Make a list sorted**
- **Design a good database**
- **Diagnose leukemia's**
- **Apples vs oranges**
- **Make exponentiation faster**
- **Draw a straight line**

- **What will we learn?**
  - Problem solving by logical reasoning on invariants
  - Problem solving by rectifying violation of invariants
  - Guilt by association of invariants
  - Solution optimization by preserving invariants

What is an invariant?

- **Suppose you have a bag of x red beans and y green beans**
- **Repeat the following:**
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- **If one bean is left behind, can you predict its colour?**

Shall we bet on the color of the bean that is left behind?

Bet on the last green bean

- **Suppose you have a bag of x red beans and y green beans**
- **Repeat the following:**
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- **If one bean is left behind, can you predict its colour?**

- **When the parity of green beans is odd, it remains odd…**
- **Start with y=2n+1**
- **y=2n+1 → y=2n-1**
- **y=2n+1 → y=2n+1**
- **y=2n+1 → y=2n+1**
- **It must be green!**

7

Bet on the last red bean

- Suppose you have a bag of x red beans and y green beans
- Repeat the following:
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- If one bean is left behind, can you predict its colour?

- When the parity of green beans is even, it remains even…
- Start with y=2n
- y=2n → y=2n-2
- y=2n → y=2n
- y=2n → y=2n
- It must be red!

CS3108, Guest lecture on 2/9/2008

8

Bet on color of the last bean … and win!

- Suppose you have a bag of x red beans and y green beans
- Repeat the following:
  - Remove 2 beans
  - If both green, discard both
  - If both red, discard one, put back one
  - If one green and one red, discard red, put back green
- If one bean is left behind, can you predict its colour?

- If you start with odd # (even #) of green beans, there will always be an odd # (even #) of green beans in the bag

⇒ Parity of green beans is invariant

⇒ Bean left behind is green iff you start with odd # of green beans

CS3108, Guest lecture on 2/9/2008

- **What have we just seen?**
- **Problem solving by logical reasoning on invariants**

Welcome to the Magical World…

This section of the ppt is courtesy of Toh Xiu Ping

The 21 Card Trick

1. Magician asks you to remember any one card from a deck of 21 cards as your card. Do not tell him what the card is
2. He deals the 21 cards face down, from top to bottom and left to right, into 3 equal piles
3. Next, he fans the piles to you and asks you to look for the pile of cards which contains your card and pass the pile back to him
4. Again, he stacks up the 3 piles on top of each other and redistribute, from top to bottom and left to right, into 3 equal piles
5. He repeats step (3) and (4) 2 more times
6. Finally, he deals your card right out from the rest of the 21 cards!

How does he manage that?!

This section of the ppt is courtesy of Toh Xiu Ping

The Trick

- **The pile containing the card is being placed in the middle of the other 2 piles**

- **Imposing constraints on where the card can move to…**

This section of the ppt is courtesy of Toh Xiu Ping

The Invariant Underlying the Trick

Assuming the chosen card is in the first pile.

11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37

After the first distribution, …

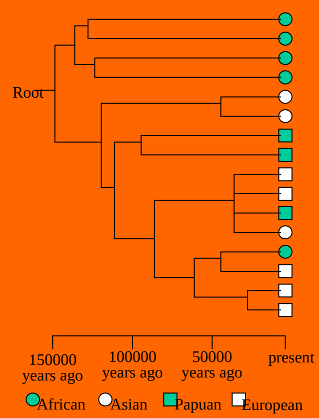
21	24	27	13	16	32	35
22	25	11	14	17	33	36
23	26	12	15	31	34	37

After the second distribution, …

After the third distribution, …

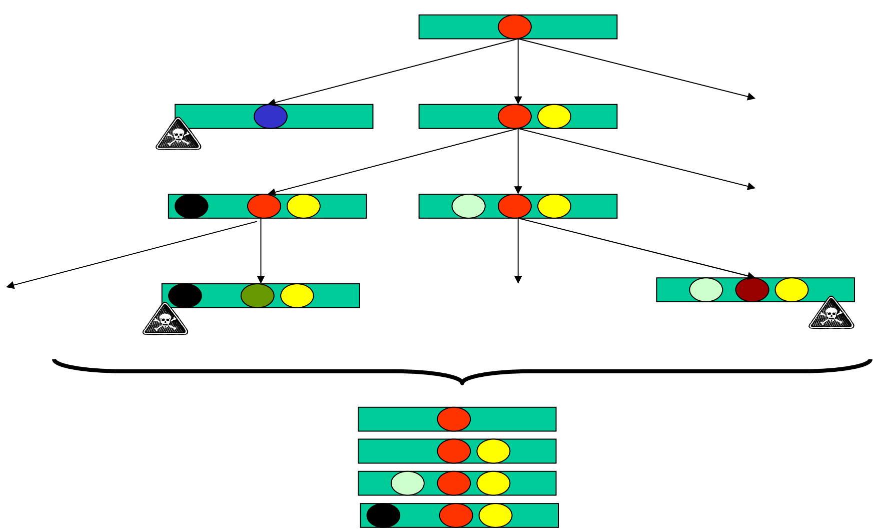
This section of the ppt is courtesy of Toh Xiu Ping

- **What have we just seen?**
- **Problem solving by logical reasoning on invariants**



Where do Polynesians come from?

In the course of evolution…

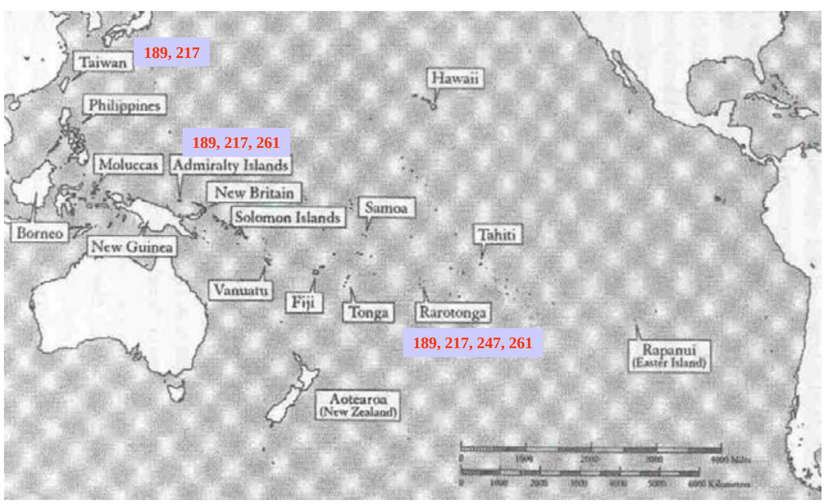


What is the invariant?

- **Mitochrondrial DNA accumulates 1 mutation about every 10,000 years**
- **Human history is not so long relative to this**

⇒ When a nucleotide in mitochrondrial DNA is mutated it stays mutated through future generations

Do Polynesians come from Asia or America?



Origin of Polynesians

- Common mitochondrial control seq from Rarotonga have variants at positions 189, 217, 247, 261. Less common ones have 189, 217, 261
- Seq from Taiwan natives have variants 189, 217
- Seq from regions in betw have variants 189, 217, 261.

- More 189, 217 closer to Taiwan. More 189, 217, 261 closer to Rarotonga
- 247 not found in America

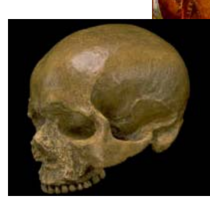
⇒ Polynesians came from Taiwan!

- Taiwan seq sometimes have extra mutations not found in other parts

⇒ These are mutations that happened since Polynesians left Taiwan!

Are Europeans descended purely from Cro Magnons? Purely from Neanderthals? Or mixed?


Neanderthal




Cro Magnon

Neanderthal vs Cro Magnon

- Based on palaeontology, Neanderthal & Cro Magnon last shared an ancestor 250,000 yrs ago
- Mitochondrial DNA accumulates 1 mutation per 10,000 yrs

⇒ If Europeans have mixed ancestry, the mitochondrial DNA betw 2 Europeans should have ~25 diff w/ high probability

- The number of diff betw Welsh is ~3, & at most 8.
- When compared w/ other Europeans, 14 diff at most

⇒ Ancestor either 100% Neanderthal or 100% Cro Magnon

- Mitochondrial DNA from Neanderthal have 26 diff from Europeans

⇒ Ancestor must be 100% Cro Magnon

The "Invariant" Perspective

- The invariant:

  When a nucleotide in mitochrondrial DNA is mutated it stays mutated through future generations

- The lesson learned:

Figure out origins of Polynesians and Europeans by logical reasoning on invariant

How to get a list sorted?

CS3108, Guest lecture on 2/9/2008

- What is a sorted list?

  A list L is sorted iff L[i] ≤ L[j] for all adjacent positions i < j

- So how do you make a list M become sorted?

  While M[i] > M[j] for some adjacent positions i < j {

  swap M[i], M[j]

  }

What makes a list a sorted list?

- Invariant of sorted lists

  A list L is sorted iff L[i] ≤ L[j] for all adjacent positions i < j

Sorting a list

- Making a list M become sorted:

```
While M[i] > M[j] for some adjacent positions i < j {
    swap M[i], M[j]
}
```

- Find violation of the invariant
- Fix it
- When no more violation, the list must be sorted!

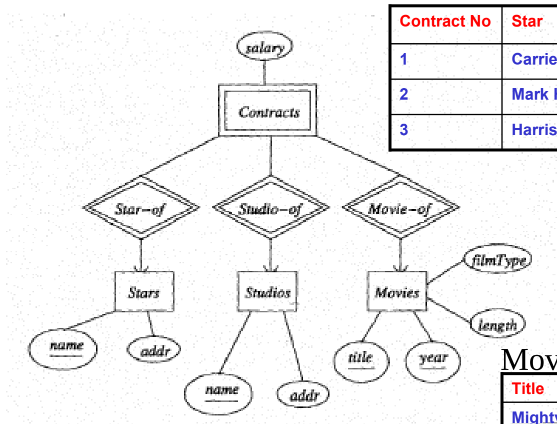
- What have we just seen?
- Problem solving by rectifying violation of invariants

What is a good database design?

Relational Data Model

- **Data are represented as a two-dimensional table**
- **It is a logical representation, not a physical representation**
  - Ordering of the rows is irrelevant
  - Ordering of the columns is irrelevant
  - How the rows and columns of a table are stored is irrelevant
  - …

Example



Contracts

Contract No	Star	Studio	Title	Salary
1	Carrie Fisher	Fox	Star Wars	$$$
2	Mark Hamill	Fox	Star Wars	$$$
3	Harrison Ford	Fox	Star Wars	$$$

Stars

Name	Address
Carrie Fisher	Hollywood
Mark Hamill	Brentwood
Harrison Ford	Beverly Hills

Movies

Title	Year	Length	Film Type
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color
Star Wars	1977	124	Color

Design Issues

- **How many possible alternate ways to represent movies using tables?**
- **Why this particular set of tables to represent movies?**
- **Indeed, why not use this alternative single table below to represent movies?**

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

Anomalies

- What's wrong with the "Wrong Movies" table?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Redundancy: Unnecessary repetition of info**
- **Update anomalies: If Star Wars is 125 min, we might carelessly update row 1 but not rows 2 & 3**
- **Deletion anomalies: If Emilio Estevez is deleted from stars of Mighty Ducks, we lose all info on that movie**

Functional Dependency

- **Functional dependency (A_1, …, $A_n \rightarrow B_1$, …, B_m)**
  - If two tuples of a table R agree on attributes A_1, …, A_n, then they must also agree on attributes B_1, …, B_m
  - $\Rightarrow$ Values of B's depend on values of A's
- **Example: Title, Year → Length, Film Type, Studio**
- **FD (A_1, …, $A_n \rightarrow B_1$, …, B_m) is trivial if a B_i is an A_j**

Can you identify the FD's here?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Some FD's:**
  - Title, Year → Length
  - Title, Year → Film Type
  - Title, Year → Studio

Keys

- **Key**
  - A minimal set of attributes $\{A_1, \ldots, A_n\}$ that functionally determine all other attributes of a table
  - A key is trivial if it comprises the entire set of attributes of a table
- **Superkey**
  - A set of attributes that contains a key

Can you identify the superkeys here?

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **Superkeys :**
  - Any set of attributes that contains {Title, Year, Star} as a subset

Boyce-Codd Normal Form

- **A relation R is in Boyce-Codd Normal Form iff whenever there is a nontrivial FD ($A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$) for R, it is the case that {$A_1, \ldots, A_n$} is a superkey for R**

- Theorem A1 (Codd, 1972)
  A database design has no anomalies due to FD iff all its relations are in Boyce-Codd Normal Form

How is BCNF violated here?

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

- **A nontrivial FD:**
  - Title, Year → Length, Film Type, Studio
  - The LHS not superset of the key {Title,Year, Star}
  - ⇒ Violate BCNF!
- **Anomalies are due to FD's whose LHS is not superkey**

Towards a Better Design

- **Use an offending FD ($A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$) to decompose $R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_h)$ into 2 tables**
  - $R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$
  - $R_2(A_1, \ldots, A_n, C_1, \ldots, C_h)$

No update anomaly

No redundant info

Title	Year	Length	Film Type	Studio
Star Wars	1997	124	Color	Fox
Mighty Ducks	1991	104	Color	Disney

No deletion anomaly

Wrong Movies

Title	Year	Length	Film Type	Studio	Star
Star Wars	1997	124	Color	Fox	Carrie Fisher
Star Wars	1997	124	Color	Fox	Mark Hamill
Star Wars	1997	124	Color	Fox	Harrison Ford
Mighty Ducks	1991	104	Color	Disney	Emilio Estevez

	Year	Star
Star Wars	1997	Carrie Fisher
Star Wars	1997	Mark Hamill
Star Wars	1997	Harrison Ford
Mighty Ducks	1991	Emilio Estevez

The "Invariant" Perspective

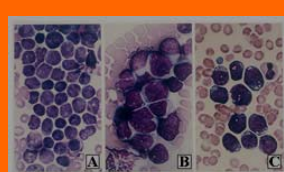
- **The invariants:**

  BCNF is an invariant of a good database design

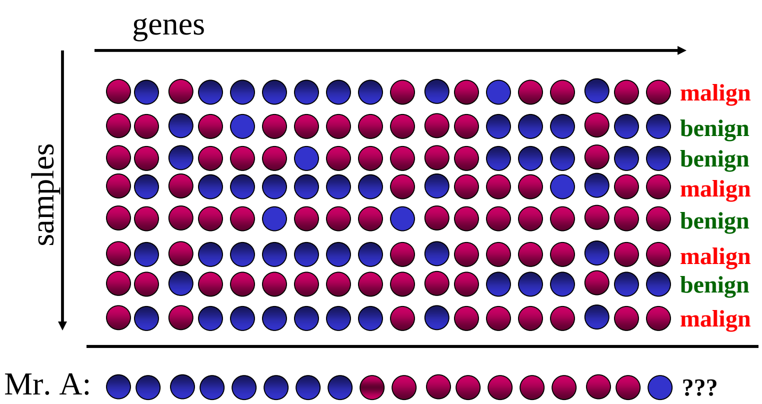
- **The lesson learned:**

Deliver a better database design by fixing violated invariants

Diagnosing Leukemias

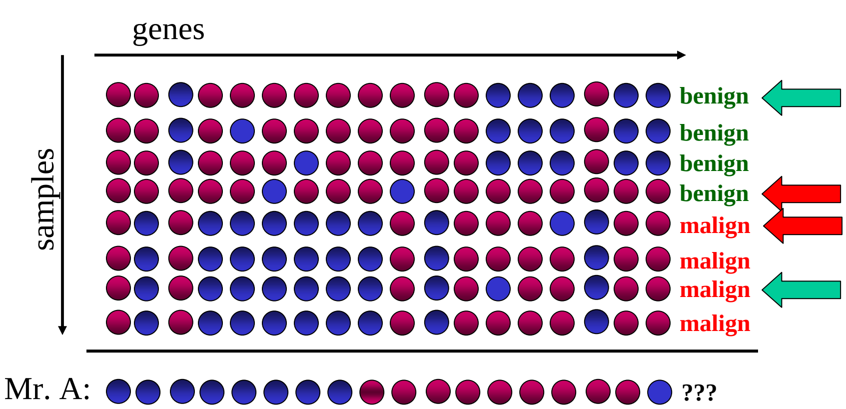


Some Patient Samples



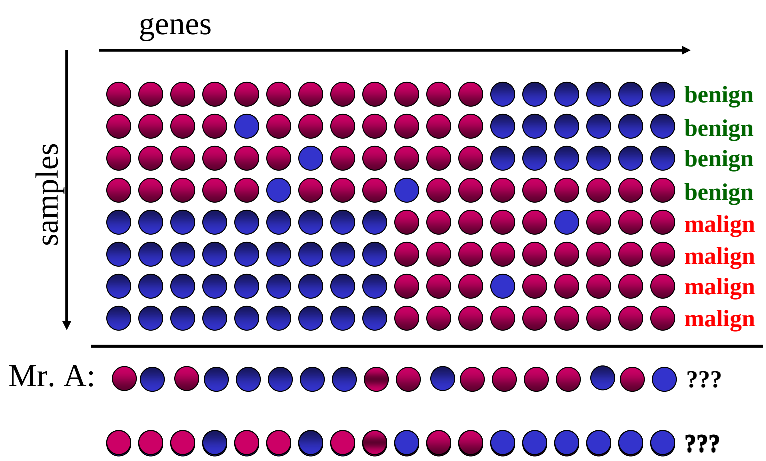
- **Does Mr. A have cancer?**

Let’s rearrange the rows…

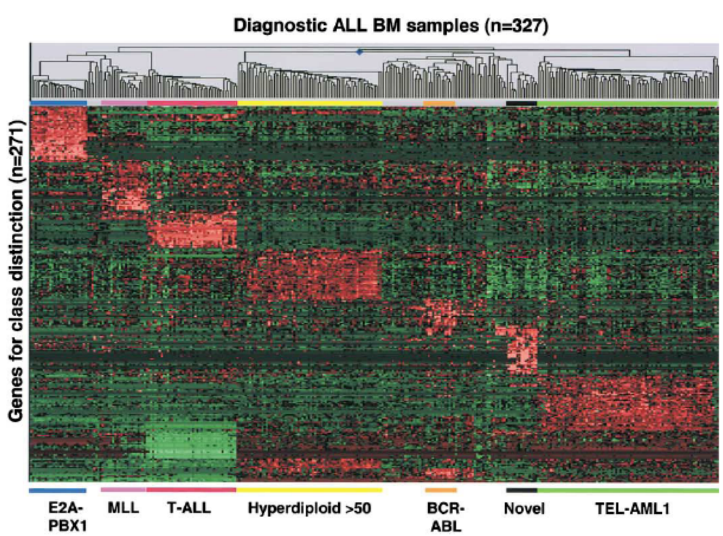


- **Does Mr. A have cancer?**

and the columns too…



Invariant Profile of Leukemia Subtypes



- **What have we just seen?**
- **Guilt by association of invariants**

Comparing Apples to Oranges

Image credit: www.comstock.com

Dissimilarity as Invariant

Differences with other fruits are identical

Differences between members of 2 different groups are constant

	$Orange_1$	$Banana_1$	...
$Apple_1$	Color = red vs orange Skin = smooth vs rough **Size = small vs small** **Shape = round vs round**	Color = red vs yellow **Skin = smooth vs smooth** **Size = small vs small** Shape = round vs oblong	...
$Orange_2$	**Color = orange vs orange** **Skin = rough vs rough** **Size = small vs small** **Shape = round vs round**	Color = orange vs yellow **Skin = rough vs smooth** **Size = small vs small** Shape = round vs oblong	...
$Unknown_1$	**Color = red vs orange** **Skin = smooth vs rough** Size = small vs small Shape = round vs round	**Color = red vs yellow** Skin = smooth vs smooth Size = small vs small **Shape = round vs oblong**	...
...	...	...	...

The unknown is an APPLE !!!

- **What have we just seen?**
- **Guilt by association of invariants &**
- **Even differences can be invariant!**

CS3108, Guest lecture on 2/9/2008
Copyright 2008 © Limsoon Wong

How to take exponentiation faster?

- **What does this program do?**

  F(a, 0) = 1
  F(a, n+1) = a * F(a, n)

- **We see that**

  F(a, n) = a^n

Exponentiation

$$a^n = \underbrace{a \times \cdots \times a}_{n},$$

Playing the invariant…

- What does this program do?

  F(a, 0) = 1
  F(a, n+1) = a * F(a, n)

- We see that

  defn

  $F(a, n) = a^n$

  invariant

- Then
- $F(a, 2*n) = a^{2*n}$ by inv
  $= a^n * a^n$ by inv
  = y * y where y = F(a, n)
- $F(a, 2*n+1) = a^{2*n+1}$ by inv
  $= a * a^n * a^n$
  = a * y * y where F(a, n)
- So we get … by inv

What's the difference?

- Original program:

  F(a, 0) = 1
  F(a, n+1) = a * F(a, n)

- Cost of F(a, n) = n

exponentially Faster!

- New program:

  F(a, 0) = 1
  F(a, 1) = a
  F(a, n) = if n is odd
  then a * y * y
  else y * y
  where y = F(a, n div 2)

  Parity can be tested by checking least significant bit

  Div2 can be implemented by bit shifting

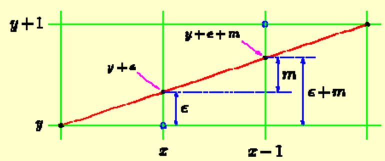
- Cost of $F(a, n) = \log_2 n$

n	log n	call sequence		
8	3	4	2	1
9	3	4	2	1
10	3	5	2	1
11	3	5	2	1

- **What have we just seen?**
- **Optimizing a solution by preserving invariant**

Drawing a Straight Line

Raster Line Drawing



- Want to plot f(X) = m X + c, where $0 \le m \le 1$
- At point x+1, should we plot at y or at y+1?
- Choose whichever is closer to f(x+1)

- Basic Bresenham's algo

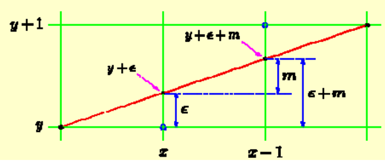
$\epsilon \leftarrow 0, \quad y \leftarrow y_1$
For $x \leftarrow x_1$ **to** x_2 **do**
  Plot point at (x, y).
  If ($\epsilon + m < 0.5$)
    $\epsilon \leftarrow \epsilon + m$
  Else
    $y \leftarrow y + 1, \quad \epsilon \leftarrow \epsilon + m - 1$
  EndIf
EndFor

m is a floating pt #.
floating pt ops are expensive

Let m = Δy / Δx and ε' = ε Δx, …

- Basic Bresenham's algo

$\epsilon \leftarrow 0, \quad y \leftarrow y_1$
For $x \leftarrow x_1$ **to** x_2 **do**
  Plot point at (x, y).
  If ($\epsilon + m < 0.5$)
    $\epsilon \leftarrow \epsilon + m$
  Else
    $y \leftarrow y + 1, \quad \epsilon \leftarrow \epsilon + m - 1$
  EndIf
EndFor

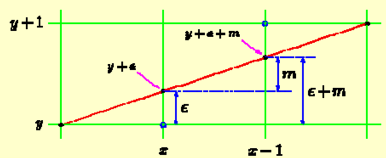


- Then
  - $\varepsilon + m < 0.5 \rightarrow$
    $2(\varepsilon' + \Delta y) < \Delta x$
  - $\varepsilon_{new} = \varepsilon + m \rightarrow$
    $\varepsilon'_{new} = \varepsilon_{new}\, \Delta x \rightarrow$
    $\varepsilon'_{new} = \varepsilon\, \Delta x + m\, \Delta x \rightarrow$
    $\varepsilon'_{new} = \varepsilon' + \Delta y$
  - $\varepsilon_{new} = \varepsilon + m - 1 \rightarrow$
    $\varepsilon'_{new} = \varepsilon_{new}\, \Delta x \rightarrow$
    $\varepsilon'_{new} = \varepsilon\, \Delta x + m\, \Delta x - \Delta x \rightarrow$
    $\varepsilon'_{new} = \varepsilon' + \Delta y - \Delta x$

A much faster algo

- **Basic Bresenham's algo**

$\epsilon \leftarrow 0, \quad y \leftarrow y_1$
For $x \leftarrow x_1$ **to** x_2 **do**
  Plot point at (x, y).
  If ($\epsilon + m < 0.5$)
    $\epsilon \leftarrow \epsilon + m$
  Else
    $y \leftarrow y + 1, \quad \epsilon \leftarrow \epsilon + m - 1$
  EndIf
EndFor



- **Integer Bresenham's algo**

$\epsilon' \leftarrow 0, \quad y \leftarrow y_1$
For $x \leftarrow x_1$ **to** x_2 **do**
  Plot point at (x, y).
  If ($2(\epsilon' + \Delta y) < \Delta x$)
    $\epsilon' \leftarrow \epsilon' + \Delta y$
  Else
    $y \leftarrow y + 1, \quad \epsilon' \leftarrow \epsilon' + \Delta y - \Delta x$
  EndIf
EndFor

Can do shift instead

All integer ops.
Very efficient!

- **Basic Bresenham's algo**

$\epsilon \leftarrow 0, \quad y \leftarrow y_1$
For $x \leftarrow x_1$ **to** x_2 **do**
  Plot point at (x, y).
  If ($\epsilon + m < 0.5$)
    $\epsilon \leftarrow \epsilon + m$
  Else
    $y \leftarrow y + 1, \quad \epsilon \leftarrow \epsilon + m - 1$
  EndIf
EndFor

- **Integer Bresenham's algo**

$\epsilon' \leftarrow 0, \quad y \leftarrow y_1$
For $x \leftarrow x_1$ **to** x_2 **do**
  Plot point at (x, y).
  If ($2(\epsilon' + \Delta y) < \Delta x$)
    $\epsilon' \leftarrow \epsilon' + \Delta y$
  Else
    $y \leftarrow y + 1, \quad \epsilon' \leftarrow \epsilon' + \Delta y - \Delta x$
  EndIf
EndFor

Invariant A:
  If $\varepsilon + m < 0.5$,
  Do not increment y

Invariant B:
  If $2(\varepsilon' + \Delta y) < \Delta x$,
  Do not increment y

This works because A is preserved by B,
as $\varepsilon + m < 0.5$ iff $\mathbf{2(\varepsilon' + \Delta y) < \Delta x}$

- **What have we just seen?**
- **Optimizing a solution by preserving invariant**

What have we learned?

What have we learned?

- **Invariant is a fundamental property of many problems**
- **Paradigms of problem solving**
  - Problem solving by logical reasoning on invariants
  - Problem solving by rectifying violation of invariants
  - Guilt by association of invariants
  - Solution optimization by preserving invariants

I didn’t get to telling you yet, but …

- **Every time you write a loop in a program, it involves an invariant**
- **Every time you do a recursive function call, it involves an invariant**
- **Every time you do an induction proof, it involves an invariant**
- **… Computing is about discovering, understanding, exploiting, and having fun with invariants!**