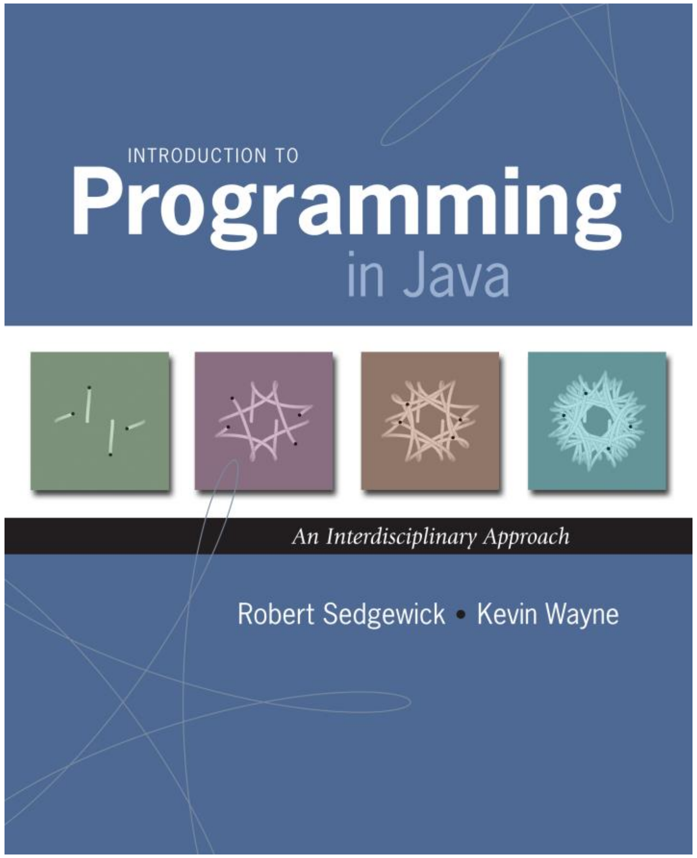
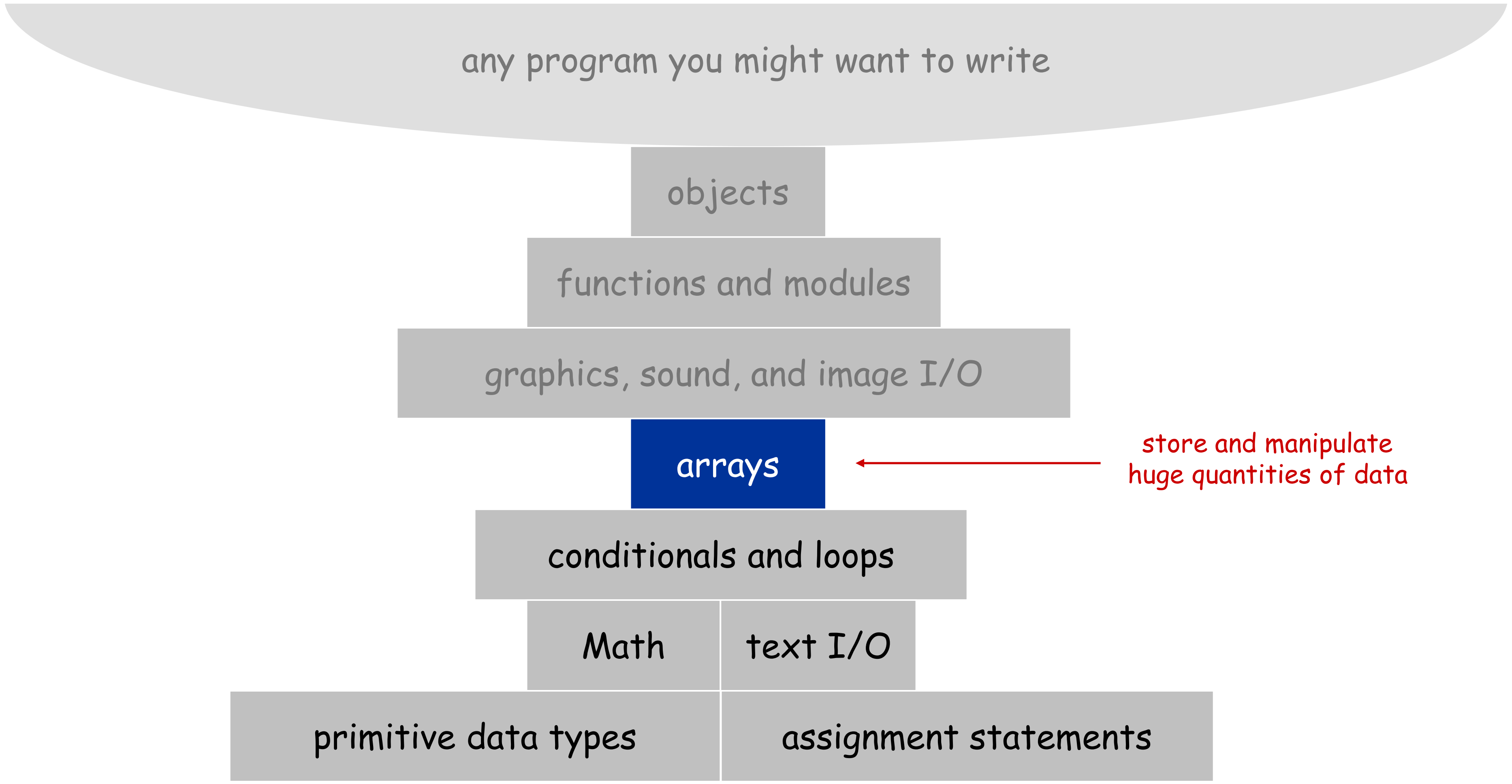
1.4 Arrays


INTRODUCTION TO
Programming
in Java
An Interdisciplinary Approach
Robert Sedgewick • Kevin Wayne

A Foundation for Programming


any program you might want to write
objects
functions and modules
graphics, sound, and image I/O
arrays
store and manipulate huge quantities of data
conditionals and loops
Math
text I/O
primitive data types
assignment statements

Arrays

This lecture. Store and manipulate huge quantities of data.

Array. Indexed sequence of values of the same type.

Examples.

- 52 playing cards in a deck.
- 5 thousand undergrads at Princeton.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 50 trillion cells in the human body.
- 6.02×10^{23} particles in a mole.

index	value
0	**`wayne`**
1	**`rs`**
2	**`doug`**
3	**`dgabai`**
4	**`maia`**
5	**`llp`**
6	**`funk`**
7	**`blei`**

Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// tedious and error-prone
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
…
a4 = 3.0;
…
a4 = 8.0;
…
double x = a4 + a8;
```

Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// easy alternative
double[] a = new double[10];
…
a[4] = 3.0;
…
a[8] = 8.0;
…
double x = a[4] + a[8];
```

declares, creates, and initializes
[stay tuned for details]

Many Variables of the Same Type

Goal. 1 million variables of the same type.

```
// scales to handle large arrays
double[] a = new double[1000000];
…
a[123456] = 3.0;
…
a[987654] = 8.0;
…
double x = a[123456] + a[987654];
```

declares, creates, and initializes
[stay tuned for details]

Arrays in Java

Java has special language support for arrays.

- To make an array: declare, create, and initialize it.
- To access element i of array named a, use a[i].
- Array indices start at 0.

```
int N = 10;                      // size of array
double[] a;                      // declare the array
a = new double[N];               // create the array
for (int i = 0; i < N; i++)      // initialize the array
   a[i] = 0.0;                   // all to 0.0
```

Arrays in Java

Java has special language support for arrays.

- To make an array: declare, create, and initialize it.
- To access element `i` of array named `a`, use `a[i]`.
- Array indices start at `0`.

```
int N = 10;                          // size of array
double[] a;                          // declare the array
a = new double[N];                   // create the array
for (int i = 0; i < N; i++)          // initialize the array
   a[i] = 0.0;                       // all to 0.0
```

Compact alternative.

- Declare, create, and initialize in one statement.
- Default initialization: all numbers automatically set to zero.

```
int N = 10;                          // size of array
double[] a = new double[N];          // declare, create, init
```

Vector Dot Product

Dot product. Given two vectors `x[]` and `y[]` of length `N`, their dot product is the sum of the products of their corresponding components.

```
double[] x = { 0.3, 0.6, 0.1 };
double[] y = { 0.5, 0.1, 0.4 };
int N = x.length;
double sum = 0.0;
for (int i = 0; i < N; i++) {
   sum = sum + x[i]*y[i];
}
```

i	x[i]	y[i]	x[i]*y[i]	sum
				0
0	.30	.50	.15	.15
1	.60	.10	.06	.21
2	.10	.40	.04	.25
				.25

Array-Processing Examples

create an array with random values	`double[] a = new double[N];`
	`for (int i = 0; i < N; i++)`
	`   a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < N; i++)`
	`   System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < N; i++)`
	`   if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < N; i++)`
	`   sum += a[i];`
	`double average = sum / N;`

copy to another array	`double[] b = new double[N];`
	`for (int i = 0; i < N; i++)`
	`   b[i] = a[i];`

reverse the elements within an array	`for (int i = 0; i < N/2; i++)`
	`{`
	`   double temp = b[i];`
	`   b[i] = b[N-1-i];`
	`   b[N-i-1] = temp;`
	`}`

Shuffling a Deck

Setting Array Values at Compile Time

Ex. Print a random card.

```
String[] rank = {
   "2", "3", "4", "5", "6", "7", "8", "9",
   "10", "Jack", "Queen", "King", "Ace"
};

String[] suit = {
   "Clubs", "Diamonds", "Hearts", "Spades"
};

int i = RNG.nextInt(13);   // between 0 and 12
int j = RNG.nextInt(4);    // between 0 and 3

System.out.println(rank[i] + " of " + suit[j]);
```

Setting Array Values at Run Time

Ex. Create a deck of playing cards and print them out.

```
String[] deck = new String[52];
for (int i = 0; i < 13; i++)
   for (int j = 0; j < 4; j++)
      deck[4*i + j] = rank[i] + " of " + suit[j];

for (int i = 0; i < 52; i++)
   System.out.println(deck[i]);
```

typical array-processing code changes values at runtime

Q. In what order does it output them?

A.

```
two of clubs
two of diamonds
two of hearts
two of spades
three of clubs
...
```

B.

```
two of clubs
three of clubs
four of clubs
five of clubs
six of clubs
...
```

Shuffling

Goal. Given an array, rearrange its elements in random order.

Shuffling algorithm.

- In iteration `i`, pick random card from `deck[i]` through `deck[N-1]`, with each card equally likely.
- Exchange it with `deck[i]`.

```
int N = deck.length;
for (int i = 0; i < N; i++) {
   int r = i + RNG.nextInt(N-i);
   String t = deck[r];
   deck[r] = deck[i];
   deck[i] = t;
}
```

swap idiom

between i and N-1

Shuffling a Deck of Cards: Putting Everything Together

```
public class Deck {
   public static void main(String[] args) {
      String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };
      String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9",
                        "10", "Jack", "Queen", "King", "Ace"    };
      int SUITS = suit.length;
      int RANKS = rank.length;
      int N = SUITS * RANKS;

      String[] deck = new String[N];
      for (int i = 0; i < RANKS; i++)
         for (int j = 0; j < SUITS; j++)
            deck[SUITS*i + j] = rank[i] + " of " + suit[j];

      for (int i = 0; i < N; i++) {
         int r = i + (int) (Math.random() * (N-i));
         String t = deck[r];
         deck[r] = deck[i];
         deck[i] = t;
      }

      for (int i = 0; i < N; i++)
         System.out.println(deck[i]);
   }
}
```

avoid "hardwired" constants

build the deck

shuffle

print shuffled deck

Shuffling a Deck of Cards

```
% java Deck
5 of Clubs
Jack of Hearts
9 of Spades
10 of Spades
9 of Clubs
7 of Spades
6 of Diamonds
7 of Hearts
7 of Clubs
4 of Spades
Queen of Diamonds
10 of Hearts
5 of Diamonds
Jack of Clubs
Ace of Hearts
...
5 of Spades
```

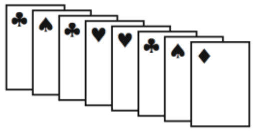
```
% java Deck
10 of Diamonds
King of Spades
2 of Spades
3 of Clubs
4 of Spades
Queen of Clubs
2 of Hearts
7 of Diamonds
6 of Spades
Queen of Spades
3 of Spades
Jack of Diamonds
6 of Diamonds
8 of Spades
9 of Diamonds
...
10 of Spades
```

Coupon Collector

Coupon Collector Problem

Coupon collector problem. Given N different card types, how many do you have to collect before you have (at least) one of each type?

assuming each possibility is equally likely for each card that you collect



Simulation algorithm. Repeatedly choose an integer `i` between `0` and `N-1`. Stop when we have at least one card of every type.

Q. How to check if we've seen a card of type `i`?

A. Maintain a boolean array so that `found[i]` is true if we've already collected a card of type `i`.

Coupon Collector: Java Implementation

```
public class CouponCollector {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int cardcnt = 0;    // number of cards collected
      int valcnt = 0;     // number of distinct cards

      // do simulation
      boolean[] found = new boolean[N];
      while (valcnt < N) {
         int val = RNG.nextInt(N);
         cardcnt++;
         if (!found[val]) {
            valcnt++;
            found[val] = true;
         }
      }

      // all N distinct cards found
      System.out.println(cardcnt);
   }
}
```

type of next card
(between 0 and N-1)

Coupon Collector: Debugging

Debugging. Add code to print contents of all variables.

val	found 0	found 1	found 2	found 3	found 4	found 5	valcnt	cardcnt
	F	F	F	F	F	F	0	0
2	F	F	T	F	F	F	1	1
0	T	F	T	F	F	F	2	2
4	T	F	T	F	T	F	3	3
0	T	F	T	F	T	F	3	4
1	T	T	T	F	T	F	4	5
2	T	T	T	F	T	F	4	6
5	T	T	T	F	T	T	5	7
0	T	T	T	F	T	T	5	8
1	T	T	T	F	T	T	5	9
3	T	T	T	T	T	T	6	10

Challenge. Debugging with arrays requires tracing many variables.

Coupon Collector: Mathematical Context

Coupon collector problem. Given N different possible cards, how many do you have to collect before you have (at least) one of each type?

Fact. About N (1 + 1/2 + 1/3 + … + 1/N) ~ N ln N.

see ORF 245 or COS 340

Ex. N = 30 baseball teams. Expect to wait ≈ 120 years before all teams win a World Series.

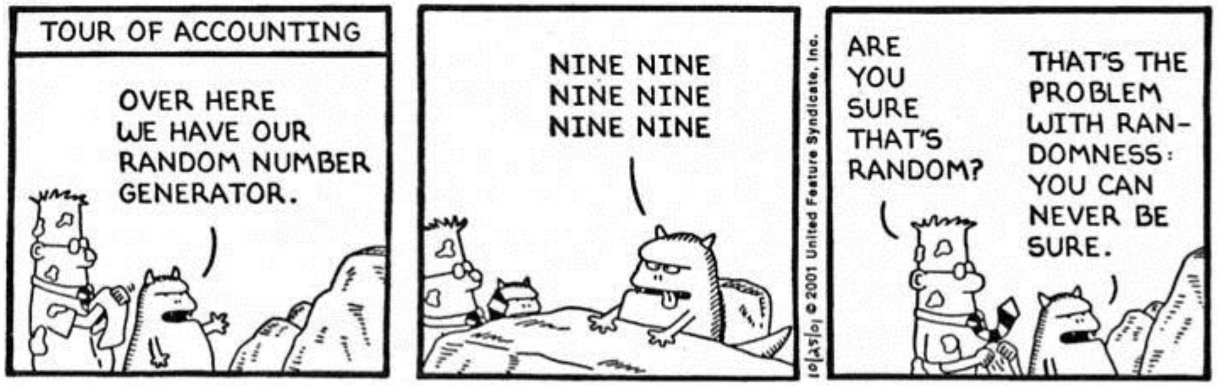
under idealized assumptions

Coupon Collector: Scientific Context

Q. Given a sequence from nature, does it have same characteristics as a random sequence?

A. No easy answer - many tests have been developed.

Coupon collector test. Compare number of elements that need to be examined before all values are found against the corresponding answer for a random sequence.


TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
10/25/01 © 2001 United Feature Syndicate, Inc.

Multidimensional Arrays

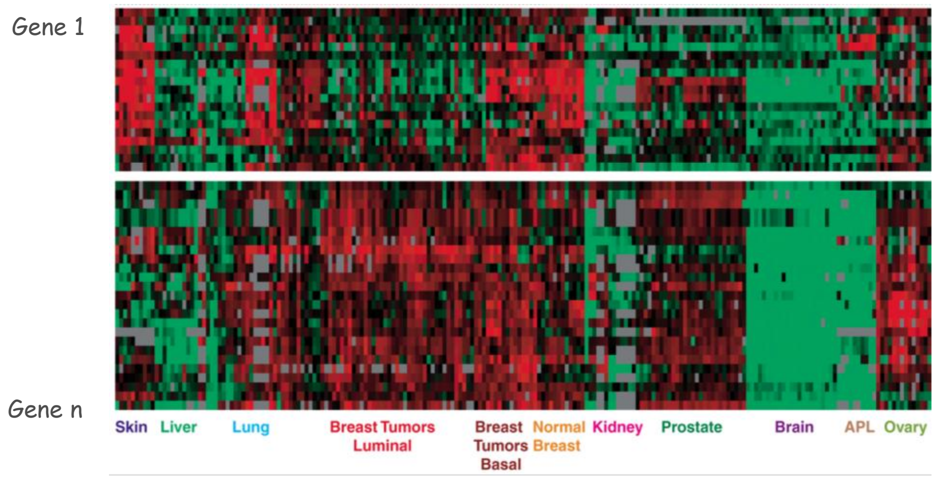
Two-Dimensional Arrays

Two-dimensional arrays.

- Table of data for each experiment and outcome.
- Table of grades for each student and assignments.
- Table of grayscale values for each pixel in a 2D image.

Mathematical abstraction. Matrix.

Java abstraction. 2D array.



Reference: Botstein & Brown group

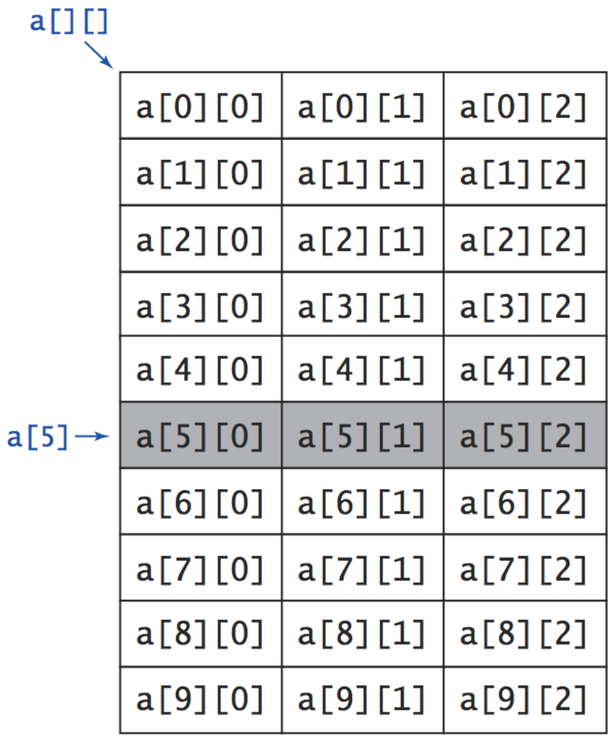
🟥 gene expressed
🟩 gene not expressed

Two-Dimensional Arrays in Java

Array access. Use `a[i][j]` to access element in row `i` and column `j`.

Zero-based indexing. Row and column indices start at `0`.

```
int M = 10;
int N = 3;
double[][] a = new double[M][N];
for (int i = 0; i < M; i++) {
   for (int j = 0; j < N; j++) {
      a[i][j] = 0.0;
   }
}
```



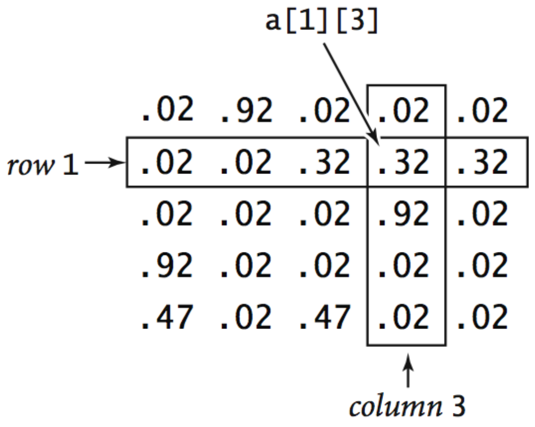
a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]
a[3][0]	a[3][1]	a[3][2]
a[4][0]	a[4][1]	a[4][2]
a[5][0]	a[5][1]	a[5][2]
a[6][0]	a[6][1]	a[6][2]
a[7][0]	a[7][1]	a[7][2]
a[8][0]	a[8][1]	a[8][2]
a[9][0]	a[9][1]	a[9][2]

A 10-by-3 array

Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

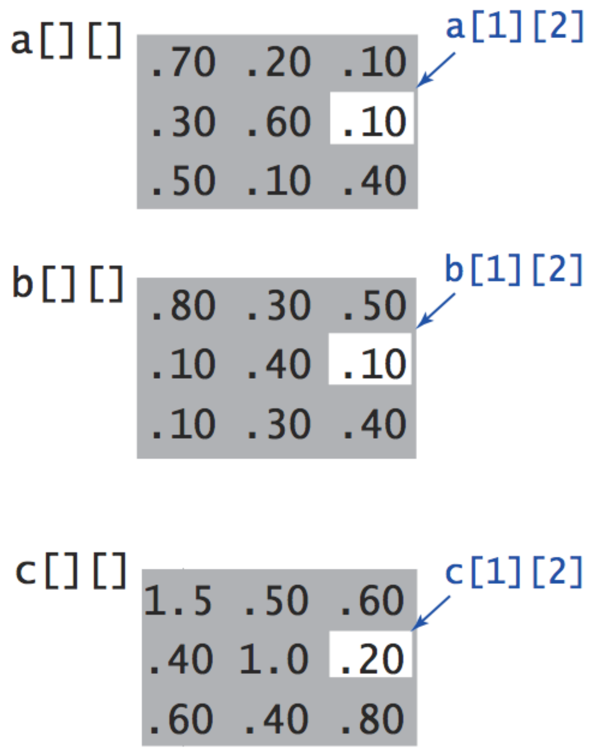
```
double[][] p = {
    { .02, .92, .02, .02, .02 },
    { .02, .02, .32, .32, .32 },
    { .02, .02, .02, .92, .02 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 },
};
```



Matrix Addition

Matrix addition. Given two N-by-N matrices `a` and `b`, define `c` to be the N-by-N matrix where `c[i][j]` is the sum `a[i][j] + b[i][j]`.

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      c[i][j] = a[i][j] + b[i][j];
```



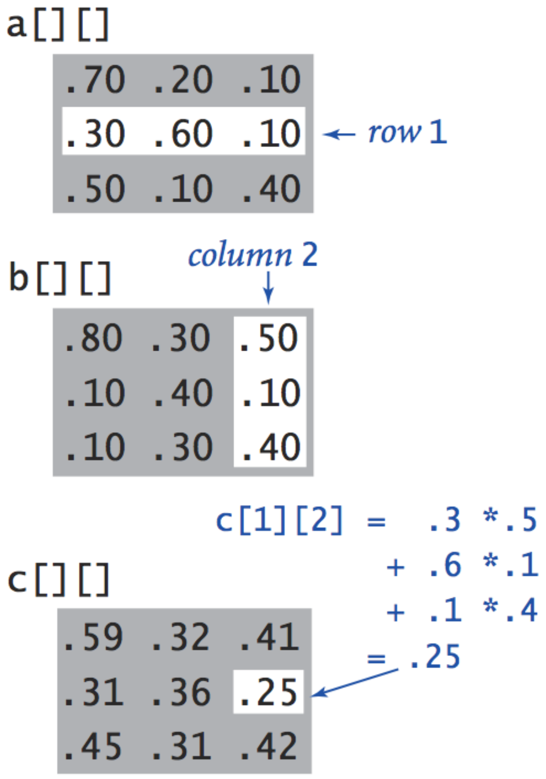
Matrix Multiplication

Matrix multiplication. Given two N-by-N matrices `a` and `b`, define `c` to be the N-by-N matrix where `c[i][j]` is the dot product of the `i`th row of `a[][]` and the `j`th column of `b[][]`.

all values initialized to 0

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

dot product of row `i` of `a[][]` and column `j` of `b[][]`



Array Challenge 2

Q. How many scalar multiplications multiply two N-by-N matrices?

A. N B. N^2 C. N^3 D. N^4

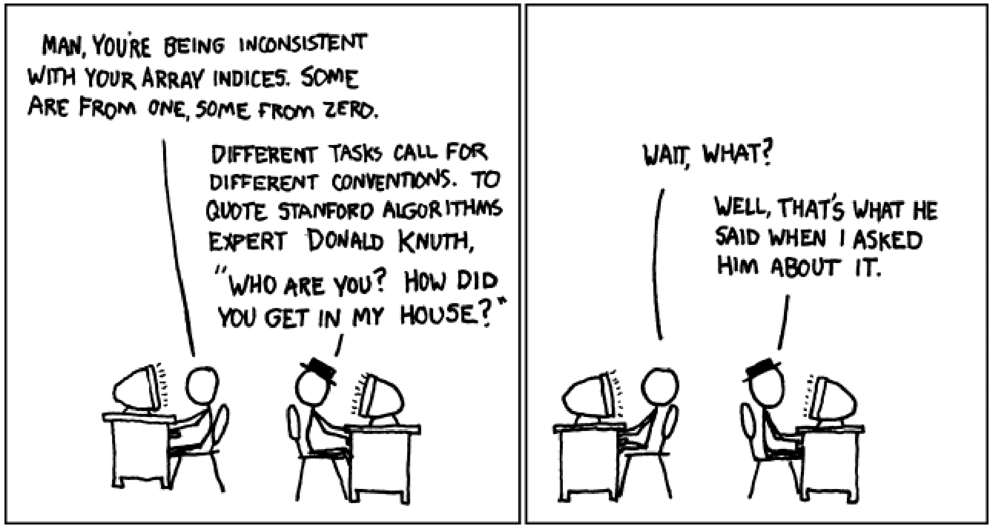
```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

Summary

Arrays.

- Organized way to store huge quantities of data.
- Almost as easy to use as primitive types.
- Can directly access an element given its index.

Ahead. Reading in large quantities of data from a file into an array.


MAN, YOU'RE BEING INCONSISTENT WITH YOUR ARRAY INDICES. SOME ARE FROM ONE, SOME FROM ZERO.
DIFFERENT TASKS CALL FOR DIFFERENT CONVENTIONS. TO QUOTE STANFORD ALGORITHMS EXPERT DONALD KNUTH, "WHO ARE YOU? HOW DID YOU GET IN MY HOUSE?"
WAIT, WHAT?
WELL, THAT'S WHAT HE SAID WHEN I ASKED HIM ABOUT IT.

http://imgs.xkcd.com/comics/donald_knuth.png

1.4 Arrays: Extra Slides

Memory Representation

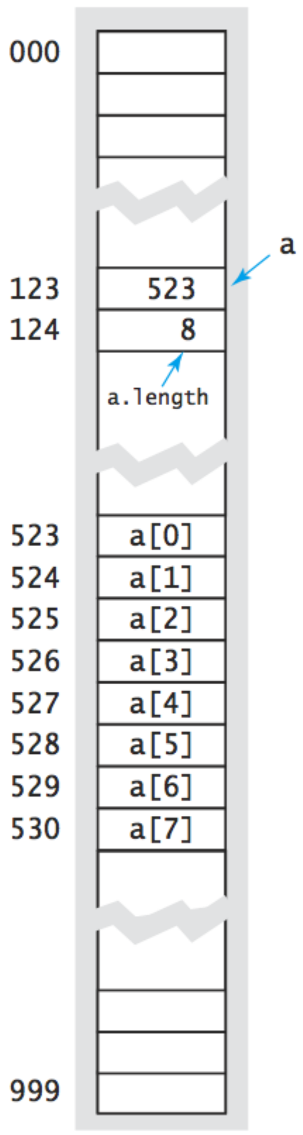
Memory representation. Maps directly to physical hardware.

Consequences.

- Arrays have fixed size.
- Accessing an element by its index is fast.
- Arrays are pointers.

2D array. Array of arrays.

Consequences. Arrays can be ragged.

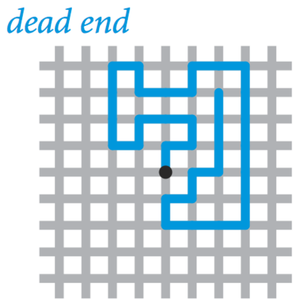


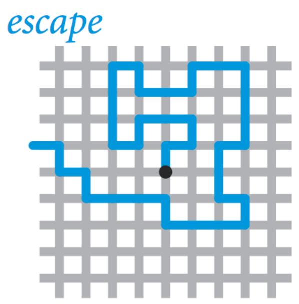
Self-Avoiding Walk

Self-Avoiding Walk

Model.

- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previous intersections.
- Two possible outcomes: dead end and escape.



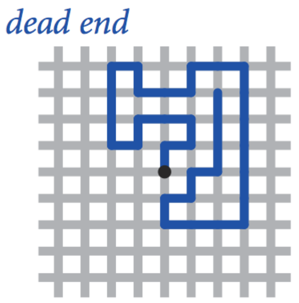


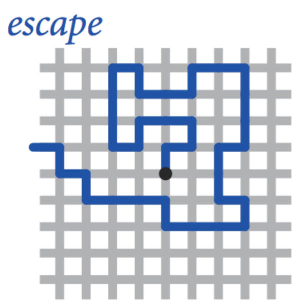
Applications. Polymers, statistical mechanics, etc.

Q. What fraction of time will you escape in an 5-by-5 lattice?
Q. In an N-by-N lattice?
Q. In an N-by-N-by-N lattice?

Self-Avoiding Walk

Skeleton. Before writing any code, write comments to describe what you want your program to do.





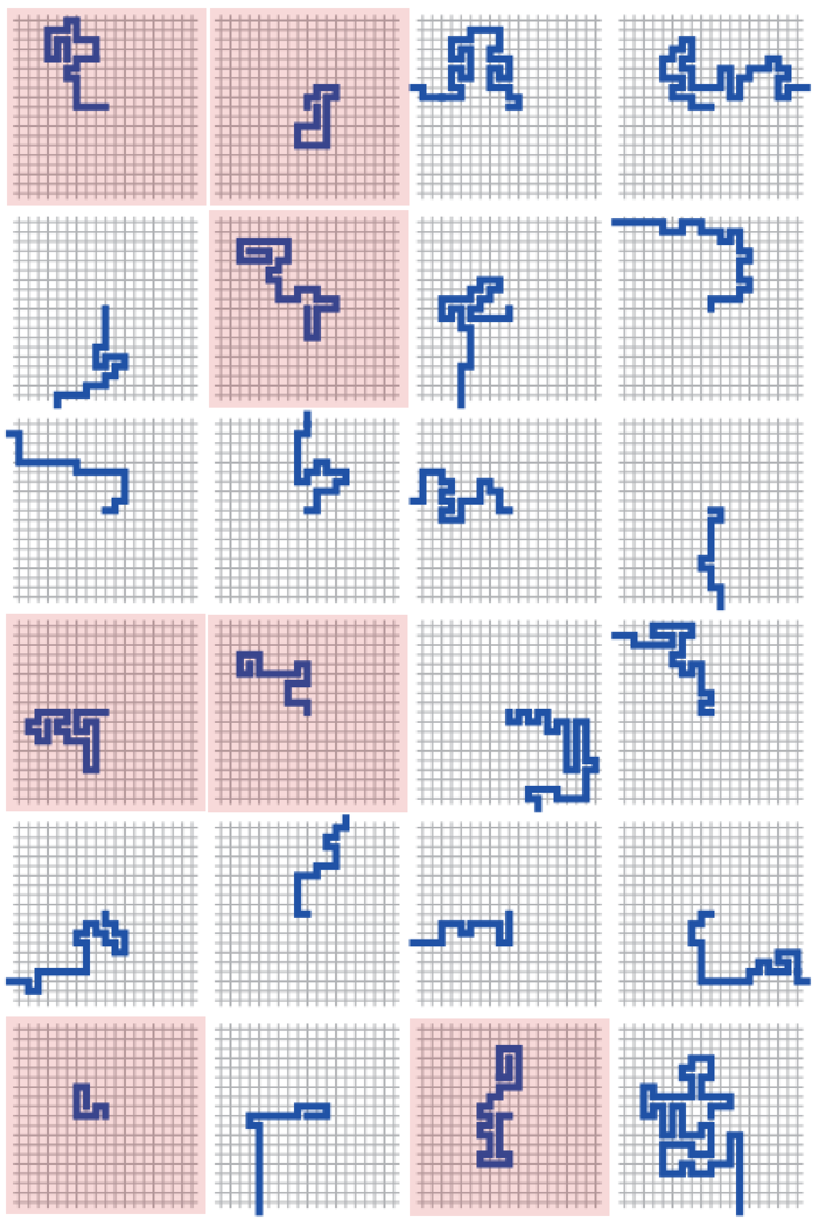
```java
public class SelfAvoidingWalk {
   public static void main(String[] args) {
      // Read in lattice size N as command-line argument.
      // Read in number of trials T as command-line argument.

      // Repeat T times:

         // Initialize (x, y) to center of N-by-N grid.

         // Repeat as long as (x, y) stays inside N-by-N grid:
            // Check for dead end and update count.
            // Mark (x, y) as visited.
            // Take a random step, updating (x, y).


         // Print fraction of dead ends.
}
```

how to implement?

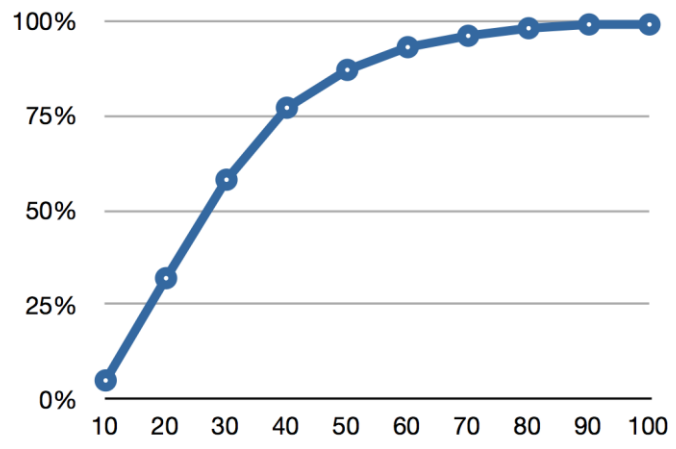
Self-Avoiding Walk: Implementation

```
public class SelfAvoidingWalk {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);     // lattice size
      int T = Integer.parseInt(args[1]);     // number of trials
      int deadEnds = 0;                      // trials resulting in dead end
      for (int t = 0; t < T; t++) {
         boolean[][] a = new boolean[N][N];  // intersections visited
         int x = N/2, y = N/2;               // current position

         while (x > 0 && x < N-1 && y > 0 && y < N-1)  {

            if (a[x-1][y] && a[x+1][y] && a[x][y-1] && a[x][y+1]) {
               deadEnds++;
               break;
            }                                                  dead end

            a[x][y] = true;                  // mark as visited

            double r = Math.random();                   take a random unvisited step
            if        (r < 0.25) { if (!a[x+1][y]) x++; }
            else if (r < 0.50) { if (!a[x-1][y]) x--; }    ←  only take step if
            else if (r < 0.75) { if (!a[x][y+1]) y++; }       site is unoccupied
            else if (r < 1.00) { if (!a[x][y-1]) y--; }
         }
      }
      System.out.println(100*deadEnds/T + "% dead ends");
   }
}
```

Visualization of Self-Avoiding Walks



```
% java SelfAvoidingWalks 10 100000
5% dead ends

% java SelfAvoidingWalks 20 100000
32% dead ends

% java SelfAvoidingWalks 30 100000
58% dead ends
…

% java SelfAvoidingWalks 100
100000
99% dead ends
```



Sieve of Eratosthenes

Sieve of Eratosthenes

Prime. An integer > 1 whose only positive factors are 1 and itself.
Ex. 2, 3, 5, 7, 11, 13, 17, 23, …

Prime counting function. $\pi(N) = \#$ primes $\leq N$.
Ex. $\pi(17) = 7$.

Sieve of Eratosthenes.

- Maintain an array `isPrime[]` to record which integers are prime.
- Repeat for `i=2` to $\sqrt{N}$
  - if `i` is not still marked as prime
    `i` is is not prime since we previously found a factor
  - if `i` is marked as prime
    `i` is prime since it has no smaller factors
    mark all multiples of `i` to be non-prime

Sieve of Eratosthenes

Prime. An integer > 1 whose only positive factors are 1 and itself.
Ex. 2, 3, 5, 7, 11, 13, 17, 23, …

Prime counting function. $\pi(N) = \#$ primes $\leq N$.
Ex. $\pi(25) = 9$.

i	isPrime																							
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
5	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

Trace of `java PrimeSieve 25`

Sieve of Eratosthenes: Implementation

```
public class PrimeSieve {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);

      // initially assume all integers are prime
      boolean[] isPrime = new boolean[N+1];
      for (int i = 2; i <= N; i++)
         isPrime[i] = true;

      // mark non-primes <= N using Sieve of Eratosthenes
      for (int i = 2; i*i <= N; i++) {
         if (isPrime[i]) {
            for (int j = i; i*j <= N; j++)
               isPrime[i*j] = false;
         }
      }

      // count primes
      int primes = 0;
      for (int i = 2; i <= N; i++)
         if (isPrime[i]) primes++;
      StdOut.println("The number of primes <= " + N + " is " + primes);
   }
}
```

if i is prime, mark multiples of i as nonprime