1.4 Arrays

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	doug
2	rs
3	dgabai
4	mona
5	cbienia
6	wkj
7	mkc

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;
a9 = 0.0;
a9 = 0.0;

double x = a4 + a8;
```


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]`.
- Indices start at 0.

```
int N = 10;
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]`.
- Indices start at 0.

```
int N = 10;
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.

- Combine declare, create, and initialize in one statement.
- Default initialization: all values automatically set to 0.

```
int N = 10;
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 sum = 0.0;
for (int i = 0; i < N; i++)
   sum += x[i]*y[i];
```

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

Compile-Time Initialize

Compile-time initialize. Can initialize array by listing values.

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 = (int) (Math.random() * 13);   <- between 0 and 12
int j = (int) (Math.random() * 4);    <- between 0 and 3

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


Run-Time Initialize

Run-time initialize. Initialize variables while program is running.

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]);
```

Q. What does it output?

Shuffling

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

Shuffling algorithm.

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

```
int N = 52;
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;
}
```

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;

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

      // shuffle the deck
      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;
      }

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

avoid hardwired constants

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 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 if we've already collected a card of type `i`.

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 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 i = (int) (Math.random() * N);
         cardcnt++;
         if (!found[i]) valcnt++;
         found[i] = true;
      }
      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	valcnt	cardcnt
	0 1 2 3 4 5		
	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).

see ORF 245 or COS 341

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

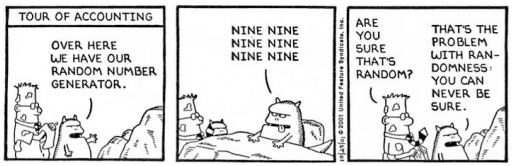
under idealized assumptions

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.



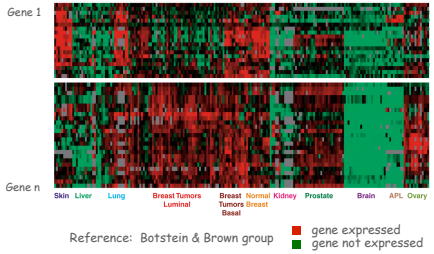
Multidimensional Arrays

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.



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 = 6, 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

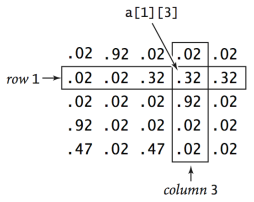
a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]
a[3][0]	a[3][1]	a[3][2]
a[4][0]	a[4][1]	a[4][2]
a[5][0]	a[5][1]	a[5][2]

A 6-by-3 array

Compile-Time Initialization

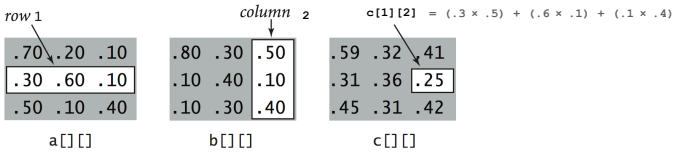
Compile-time initialization. Initialize 2D array by listing values.

```
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 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} row of `b`.



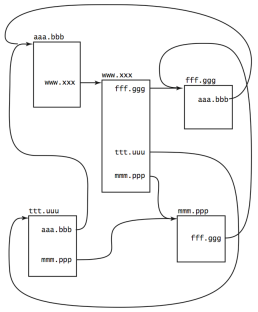
run-time initialize (all values set to 0)

```
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];
```

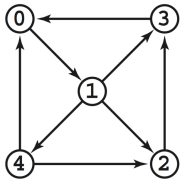

Hyperlink Structure of Web

Relevance. Use web page content to determine its relevance to query.

Importance. Use hyperlink structure of Web to determine importance of web pages, independent of query.



Pages and links



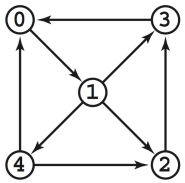
Random Surfer Model

90-10 rule. Web surfer chooses next page:

- 90% of the time surfer clicks random hyperlink.
- 10% of the time surfer types a random page.

Caveat. Very crude, but useful, model of reality.

Transition matrix. `p[i][j]` = prob that surfer moves from page i to j.



surfer on page 1 will go to page 2 next 32% of the time

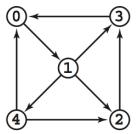
random page	random link	combination
.02 .02 .02 .02 .02	0 .90 0 0 0	.02 .92 .02 .02 .02
.02 .02 .02 .02 .02	0 0 .30 .30 .30	.02 .02 .32 .32 .32
.02 .02 .02 .02 .02	0 0 0 .90 0	.02 .02 .02 .92 .02
.02 .02 .02 .02 .02	.90 0 0 0 0	.92 .02 .02 .02 .02
.02 .02 .02 .02 .02	.45 0 .45 0 0	.47 .02 .47 .02 .02

1-step transition matrix

Random Surfer and Matrix Multiplication

Q. What is prob that surfer moves from page i to page j in two steps?

A. $P^2 = P \times P$. [Matrix multiplication!]



probability of surfing from i to 2 in one step

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

probability of surfing from 1 to i in one step

p^2

.05	.04	.31	.31	.30
.45	.04	.17	.31	.03
.86	.04	.04	.04	.03
.05	.85	.04	.04	.03
.05	.44	.04	.45	.03

probability of surfing from 1 to 2 in two steps (dot product)

Squaring a Markov chain

Random Surfer: Mathematical Context

Q. What is prob that surfer moves from page i to page j in the limit?

A. $\lim_{k \to \infty} P^k = P \times P \times \ldots \times P$.

Mixing theorem. P^k converges as k approaches infinity.
Moreover, all rows are equal.

for our random surfer model

fraction of time surfer spends on page j
is independent of starting point!

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

P^2

.05	.04	.31	.31	.30
.45	.04	.17	.31	.03
.86	.04	.04	.04	.03
.05	.85	.04	.04	.03
.05	.44	.04	.45	.03

P^4

.31	.41	.05	.19	.04
.20	.30	.16	.19	.15
.09	.08	.27	.30	.26
.42	.08	.16	.30	.04
.26	.41	.11	.19	.04

P^8

.27	.29	.13	.21	.10
.25	.27	.15	.23	.11
.26	.22	.17	.25	.11
.30	.25	.13	.24	.08
.30	.27	.14	.22	.11

P^{16}

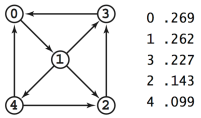
.27	.26	.14	.23	.10
.27	.26	.14	.23	.10
.27	.26	.14	.23	.10
.27	.26	.14	.23	.10
.27	.26	.14	.23	.10

surfing from 1 to 2
in 8 steps

Random Surfer: Scientific Context

Google's PageRank™ algorithm. [Sergey Brin and Larry Page, 1998]

- Rank importance of pages based on hyperlink structure of Web, using 90-10 rule.
- Revolutionized access to world's information.



0 .269
1 .262
3 .227
2 .143
4 .099

Page ranks

Scientific challenges. Cope with 4 billion-by-4 billion matrix!

- Need data structures to enable computation.
- Need linear algebra to fully understand computation.

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.

Caveats:

- Need to fix size of array ahead of time.
- Don't forget to allocate memory with `new`.
- Indices start at 0 not 1.
- Out-of-bounds to access `a[-1]` or `a[N]` of N element array.
  - in Java: `ArrayIndexOutOfBoundsException`
  - in C: "ghastly error"

"You're always off by 1 in this business." - *J. Morris*

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