1.4 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.
- $6.02 \times 10^{23}$ particles in a mole.

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>doug</td>
</tr>
<tr>
<td>1</td>
<td>cmoretti</td>
</tr>
<tr>
<td>2</td>
<td>dgabai</td>
</tr>
<tr>
<td>3</td>
<td>maia</td>
</tr>
<tr>
<td>4</td>
<td>jcmelson</td>
</tr>
<tr>
<td>5</td>
<td>cmiller</td>
</tr>
<tr>
<td>6</td>
<td>tspereira</td>
</tr>
<tr>
<td>7</td>
<td>dkthree</td>
</tr>
<tr>
<td>8</td>
<td>adror</td>
</tr>
<tr>
<td>9</td>
<td>soumyade</td>
</tr>
<tr>
<td>10</td>
<td>idavey</td>
</tr>
<tr>
<td>11</td>
<td>drutakoy</td>
</tr>
<tr>
<td>12</td>
<td>mrbrowni</td>
</tr>
</tbody>
</table>

Many Variables of the Same Type

Goal. 10 variables of the same type.

```c
// Tedious and error-prone code.
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;
...
a8 = 8.0;
...
double x = a4 + a8;
```
Many Variables of the Same Type

**Goal.** 10 variables of the same type.

```java
// Easy alternative.
double[] a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```

Many Variables of the Same Type

**Goal.** 1 million variables of the same type.

```java
// Scales to handle large arrays.
double[] a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```

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.

```java
int N = 1000;
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 alternatives: Declare, create, and initialize in one statement.

- Default: all entries automatically set to 0.
- Initialize to literal values

```java
double[] a = new double[1000];
double[] x = { 0.3, 0.6, 0.1 };
```
Sample Array Code: 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.

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

<table>
<thead>
<tr>
<th>i</th>
<th>x[i]</th>
<th>y[i]</th>
<th>x[i]*y[i]</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>0.3</td>
<td>0.5</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td>0.1</td>
<td>0.06</td>
<td>0.21</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.4</td>
<td>0.04</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Array Processing Examples

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

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

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

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

double sum = 0.0;
for (int i = 0; i < N; i++)
    System.out.println(a[i]);
print the array values, one per line

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

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

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

Mumbo-Jumbo Demystification, Part 1

```java
public class Gambler {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);
        . . .
        . . .
    }
}
```

Shuffling a Deck
Setting Array Values at Compile Time

Ex. Print a random card.

```java
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]);
```

Array Challenge 2

Swap the for statements: rank index in inner loop, suit index in outer loop.

Now, in which order are they printed?

```java
String[] rank = { "2", "3" ..., "King", "Ace" }; String[] suit = { "clubs", "diamonds", "hearts", "spades" }; 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]);
```

Array Challenge 3

The following code sets array values to the 52 card values and prints them. What change to the code will produce the "B" order?

```java
String[] rank = { "2", "3" ..., "King", "Ace" }; String[] suit = { "clubs", "diamonds", "hearts", "spades" }; 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]);
```

A. 2 of clubs  2 of clubs  2 of diamonds  2 of hearts  2 of spades  3 of clubs...
B. 2 of clubs  3 of clubs  4 of diamonds  4 of hearts  5 of spades  6 of clubs...

Processing code changes values at runtime
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].

```java
int N = deck.length;
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;
}
```

Shuffle a deck of cards.

- In $i$th iteration, put a random element from remainder of deck at index $i$.
  - choose random integer $r$ between $i$ and N-1
  - swap values in positions $r$ and $i$

```java
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 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]);
    }
}
```
Coupon Collector

Coupon Collector:  Java Implementation

```java
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 = (int) (Math.random() * N);
            cardcnt++;
            if (!found[val])
            {
                valcnt++;
                found[val] = true;
            }
        }

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

Coupon Collector:  Debugging

**Debugging.** Add code to print contents of all variables.

<table>
<thead>
<tr>
<th>val</th>
<th>found</th>
<th>valcnt</th>
<th>cardcnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F F F F F F</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>F F T F F F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>T F T F F F</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>T F T F T F</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>T F T F T F</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>T T T F T F</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>T T T F T F</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>T T T F T T</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>T T T F T T</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>T T T F T T</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>T T T T T T</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

**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 $N \left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N}\right) \sim N \ln N$

See ORF 245 or COS 341

**Ex.** N = 30 baseball teams. Expect to wait $\sim 120$ years before all teams win a World Series.

---

**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**

- 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**

**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
Two Dimensional Arrays in Java

Declare, create, initialize. Like 1D, but add another pair of brackets.

```
int M = 10;
int N = 3;
double[][] a = new double[M][N];
```

Array access.
Use `a[i][j]` to access entry in row `i` and column `j`.
Indices start at 0.

Initialize.
This code is implicit (sets all entries to 0).

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
        a[i][j] = 0.0;
```

Warning. This implicit code might slow down your program for big arrays.

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 row of `b`.

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

Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

```
double[][] p = {
    {0.02, 0.92, 0.02, 0.02},
    {0.02, 0.92, 0.02, 0.02},
    {0.02, 0.92, 0.02, 0.02},
    {0.02, 0.92, 0.02, 0.02},
    {0.47, 0.02, 0.47, 0.02},
};
```
Array Challenge 4

How many multiplications to 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];
```

Application: 2D Random Walks

Application: Self-Avoiding Walks

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

Applications. Polymers, statistical mechanics, etc.

Q. What fraction of time will you escape in a 5-by-5 lattice?
Q. In an N-by-N lattice?
Q. In an N-by-N-by-N lattice?
Self-Avoiding Walk: Implementation

```java
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 ending at dead end
        for (each trial )
        {
            boolean[][] a = new boolean[N][N]; // intersections visited
            int x = N/2, y = N/2; // current position
            while (you're still inside the lattice)
            {
                if (you're at a dead end)
                {
                    deadEnds++;
                    break;
                }
                a[x][y] = true; // mark as visited
                double r = Math.random();
                if (r < 0.25) { if (!a[x+1][y]) x++; }
                else if (r < 0.50) { if (!a[x-1][y]) x--; }
                else if (r < 0.75) { if (!a[x][y+1]) y++; }
                else if (r < 1.00) { if (!a[x][y-1]) y--; }
            }
            System.out.println(100*deadEnds/T + "% dead ends");
        }
    }
}
```

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"

Ahead. Reading in large quantities of data from a file into an array.
"You're always off by 1 in this business." - J. Morris