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

Many Variables of the Same Type

Goal. 10 variables of the same type.

```java
// 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;
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 \( a[i] \).
- Array indices start at 0.

```java
int N = 10;
double[] a;
// declare the array
da = 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.

```java
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.

```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];
}
```

Array Processing Code

```java
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-1-i] = temp;
        }
```
**Shuffling a Deck**

**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]);
```

**Setting Array Values at Run Time**

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

```java
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.** 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]`.

```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;
}
```
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;
        System.out.println(deck[i]);
    }
}

Coupon Collector

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

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

```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</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>F F T</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>T F F</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>T F F</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>T F F</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>T T F</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>T T F</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>T T T</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>T T T</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>T T T</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>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 $N (1 + 1/2 + 1/3 + ... + 1/N)$.

Ex. $N = 30$ baseball teams. Expect to wait $\approx 120$ 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.
Two Dimensional Arrays

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.

Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

```java
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 },
};
```

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.

```java
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;
    }
}
```
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 \(a[i][j] + b[i][j]\).

```java
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 \(b\).

```java
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];
```

Array Challenge 2

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

A.  \(N\)  
B.  \(N^2\)  
C.  \(N^3\)  
D.  \(N^4\)

```java
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];
```
Self-Avoiding Walk

Model.
- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previous intersections.

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: 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 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++; } // up
                else if (r < 0.50) { if (!a[x-1][y]) x--; } // down
                else if (r < 0.75) { if (!a[x][y+1]) y++; } // right
                else if (r < 1.00) { if (!a[x][y-1]) y--; } // left
            }
        }
        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.

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