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 Yale.
- 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. 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];

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];

Arrays in Java

Java has special language support for arrays.
• To make an array: declare, create, and initialize it.
• To access entry 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 entry 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.

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

<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>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

**Shuffling a Deck**

**Ex.** Print a random card.

\[
\text{String}[\text{rank}] = \{\text{"2", } \text{"3", } \text{"4", } \text{"5", } \text{"6", } \text{"7", } \text{"8", } \text{"9",}
\text{"10", } \text{"Jack", } \text{"Queen", } \text{"King", } \text{"Ace"}\};
\text{String}[\text{suit}] = \{\text{"Clubs", } \text{"Diamonds", } \text{"Hearts", } \text{"Spades"}\};
\text{int } i = (\text{int}) (\text{Math.random()} \times 13); \text{ // between 0 and 12}
\text{int } j = (\text{int}) (\text{Math.random()} \times 4); \text{ // between 0 and 3}
\text{System.out.println(rank[i] + " of " + suit[j]);}
\]
Creating a Deck of Playing Cards and Print Them Out.

```
public class Deck {
    public static void main(String[] args) {
        String[] suits = {"Clubs", "Diamonds", "Hearts", "Spades"};
        String[] ranks = {"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];
        build the deck
        for (int i = 0; i < RANKS; i++)
            for (int j = 0; j < SUITS; j++)
                deck[j*4 + i] = rank[i] + " of " + suit[j];

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

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

### 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].
War Story (PlanetPoker.com)

Texas hold ‘em poker. Software must shuffle electronic deck of cards.

How we learned to cheat at online poker: a study in software security
http://itmanagement.earthweb.com/entdev/article.php/616221

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?

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 \( \text{found}[i] \) 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</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>T</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>T</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>T</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>T</td>
<td>5</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 \( N \left(1 + 1/2 + 1/3 + \ldots + 1/N\right) \approx N \ln N.\)

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

See CPSC 223 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

---

 TOUR OF ACCOUNTING
 OVER HERE, WE HAVE OUR RANDOM NUMBER GENERATOR.

 NONE NONE NONE NONE NONE

 ARE YOU QUACKING RAMPS?

 THAT'S THE PROBLEM WITH RAMPS, YOU CAN NEVER BE SURE.
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.

Array access. Use `a[i][j]` to access entry 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] = 0.5;
    }
}
```

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[M][N];
for (int i = 0; i < M; 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 $b[][]$.

$$\text{double}[][] c = \text{new} \text{double}[N][N];$$

$$\text{for} (\text{int} i = 0; i < N; i++)$$

$$\text{for} (\text{int} j = 0; j < N; j++)$$

$$c[i][j] += a[i][k] * b[k][j];$$

Array Challenge

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

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

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.
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

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 a 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.
    }
}
```
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]) {
                    break; // dead end
                }
                a[x][y] = true; // mark as visited
                double r = Math.random(); // take random unvisited step
                if (r < 0.25) { if (!a[x+1][y]) x++; } // only take step if site is unoccupied
                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");
        }
    }
}
```

Visualization of Self-Avoiding Walks

![Visualization of Self-Avoiding Walks](image)

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.** \( \pi(N) = \# \text{primes} \leq N. \)
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 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.** \( \pi(N) = \# \text{primes} \leq N \).
Ex. \( \pi(25) = 9 \).

| 1 | 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 | T | T |
| 2 | T | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 3 | T | T | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 5 | T | T | T | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 7 | T | T | F | T | F | F | F | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| 11 | T | T | F | T | F | F | F | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |

Trace of java PrimeSieve 25

**Sieve of Eratosthenes: Implementation**

```java
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; i * j <= N; j += i)
                    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