Lecture 5: Arrays

Arrays

Last lecture: read in huge quantities of data.
This lecture: store and manipulate huge quantities of data.

Arrays.

- Organized way to store huge quantities of data.
  - 52 playing cards in a deck
  - 5 thousand Princeton undergrads
  - 1 million characters in a book
  - 4 billion nucleotides in a strand of DNA
  - 73 billion Google queries per year
  - 6.02 x 10^{23} particles in a mole

Today's applications.

- Data analysis. (histogram)
- Data processing. (shuffling, sorting)
- Scientific applications. (DLA simulation)

Arrays in Java

Arrays are built into Java.

- Essential property: can directly access an element given its index.
- Declare and initialize using `[]` and `{}`.

```
int a0 = 3, a1 = 1, a2 = 4, a3 = 1, a4 = 5;
int a5 = 9, a6 = 2, a7 = 6, a8 = 5, a9 = 3;
```

vs.

```
int[] a = { 3, 1, 4, 1, 5, 9, 2, 6, 5, 3 };
```

- To access element i of array named `a`, use `a[i]`

Choosing a Random Student

Simple application: store related data as a group, and select random item.

```
public class RandomStudent {
   public static void main(String[] args) {
      String[] names = { "Clelia Zacharias", "Hannah Xu",
                         "Virginia Wylly",   "Wendy Wu",
                         "Ashely Wolf",      "Eric Whitman",
                         "Will Weidman",     "Sharon Weeks",
                         "Mary Wathall",     "Sarah Wang",
                         "Michael Wang",     "Madeleine Walsh"
             12        };
              ↓
      int N = names.length;
      int r = (int) (Math.random() * N);  ← integer between 0 and 11
      System.out.println(names[r]);
   }
}
```

California Runoff Election '04

135 candidates on ballot for governor of California.

- Alphabetical order prejudiced against Jon Zellhoefer.
- One solution: in each district, randomize the order in which the candidates appear.

name
Iris Adam
Brooke Adams
Cruz Bustamante
Gary Coleman
Larry Flynt
Georgy Russell
Arnold Schwarzenegger
Peter Ueberroth
Jon Zellhoefer

⇨

name
Peter Ueberroth
Gary Coleman
Arnold Schwarzenegger
Brooke Adams
Jon Zellhoefer
Georgy Russell
Cruz Bustamante
Iris Adam
Larry Flynt

Creating Arrays in Java

How to declare an "empty" array.

- Declare using `[]`.

```
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
```

vs.

```
double[] a = new double[10];
```

- Allocate memory using `new`.
- All array elements are auto-initialized to:
  - zero for numeric types
  - `null` for String

Shuffling

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

      for (int i = 0; i < N; i++)
         a[i] = StdIn.readString();

      SHUFFLE

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

← read in and store data

← print data

Shuffling

Shuffle an N-element array.

- In ith iteration:
  - choose random integer r between 0 and i
  - swap values in positions r and i
- Need random access to arbitrary element ⇒ use arrays.

Property: after ith iteration, array positions 0 through i contain random permutation of elements 0 through i.

```
for (int i = 0; i < N; i++) {
   int r = (int) (Math.random() * (i + 1));
   String swap = a[r];
   a[r] = a[i];
   a[i] = swap;
}
```

↑ between 0 and i

shuffle

Gambler's Problem Revisited

Flip a fair coin N times and plot distribution of number of heads.

- Use `freq[i]` to record number of times you get exactly `i` heads.

```
public class Flip {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int[] freq = new int[N + 1];

      for (int i = 0; i < 10000; i++) {
         int heads = 0;                      flip N coins
         for (int j = 0; j < N; j++)
            if (Math.random() < 0.5)
               heads++;                      flip one coin
         freq[heads]++;                      increment counter
      }
   }
}
```

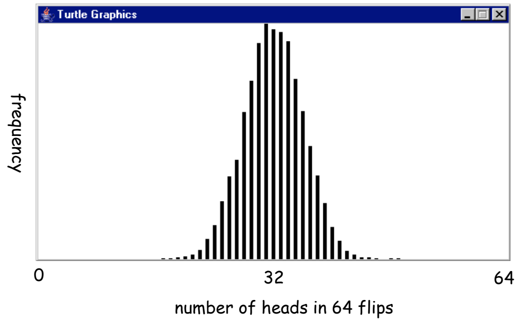
- Add graphic commands to plot.

Gambler's Problem Revisited

What is distribution of number of heads?

- "Bell curve."
- Approximately Gaussian (stay tuned).
- Mean = N / 2, variance = N / 4. ← 95% confidence interval: N/2 ± √N

```
% java Flip 64
```



Benford's Law

Examine listing of statistical data.

- Compute frequency count of leading digit.
  - Ex: leading digit of 456789 is 4.
- Print fraction of occurrences of each digit 1 - 9.
- What is distribution? 11.11% each? Something else?

Use 10-element array `count`.

- `count[i]` counts number of times `i` is leading digit. ← `count[0]` is always 0
- `N` counts total number of items processed.
- Print ratio for each `i`.

Benford's Law

```
public class Benford {
   public static void main(String[] args) {
      int[] count = new int[10];
      int N = 0;
                   auto-initialized to 0
      while (!StdIn.isEmpty()) {
         int x = StdIn.readInt();

         while (x >= 10) x = x / 10;         ← compute leading digit

         count[x]++;   ← increment bin x
         N++;
      }

      for (int i = 1; i < 10; i++)
         System.out.println(i + ": " + 1.0 * count[i] / N);
   }
                                        avoid integer division
}
```

Benford's Law

Newcomb (1881). Tables of logarithms.

Benford (1938).

- River area.
  Newspaper.
  Pressure.
  Drainage.
  Baseball.
  Death rates.
  Population.
  Specific heat.
  Atomic weight.
  Reader's Digest.
  Black body.
  Addresses.
- Scale invariant!

Hill (1996).

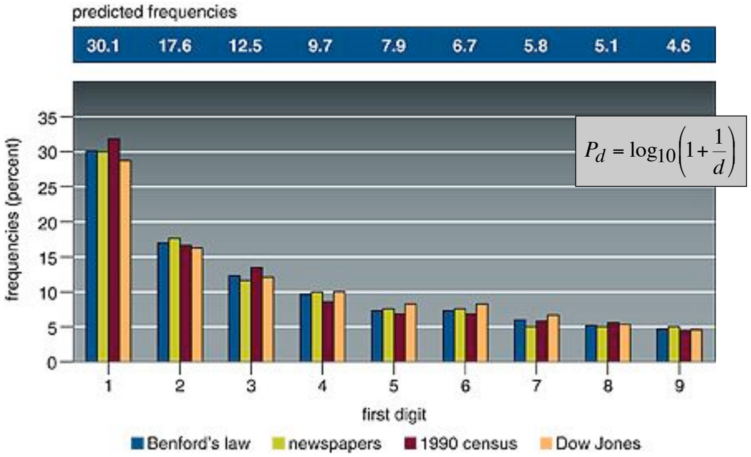
- Distribution of distributions.

```
% more pu-files.txt
96796
4171208
5830
34343656
...

% java Benford < pu-files.txt
1: 0.30788221725654147
2: 0.19250222254258872
3: 0.1302139647757034
4: 0.09865986688771955
5: 0.07445217328623946
6: 0.05945601768423076
7: 0.05162606021288354
8: 0.04417153223287441
9: 0.04103594512121867
```

332,952 file sizes

The First-Digit Phenomenon



$$P_d = \log_{10}\left(1+\frac{1}{d}\right)$$

Reference: *The First-Digit Phenomenon* by T. P. Hill, in *American Scientist*, July-August 1998.

Sorting

Goal: given N items, rearrange them in increasing order.

Applications.

- Sort a list of names.
- Find duplicates in a mailing list.
- Find the median.
- Identify statistical outliers.
- Data compression.
- Computer graphics.
- Computational biology.

name
Hauser
Hong
Hsu
Hayes
Haskell
Hanley
Hornet

name
Hanley
Haskell
Hauser
Hayes
Hong
Hornet
Hsu

Insertion Sort

Insertion sort an N-element array.

- In i^{th} iteration:
  - read i^{th} value
  - repeatedly swap i^{th} value with the one to its left if smaller

Property: after i^{th} iteration, array positions 0 through i contain original elements 0 through i in increasing order.

```
for (int i = 0; i < N; i++) {
   for (int j = i; j > 0; j--) {
      if (x[j-1] > x[j]) {
         double swap = x[j];
         x[j] = x[j-1];
         x[j-1] = swap;
      }
   }
}
```

swap `x[j]` and `x[j-1]`

sort array of real numbers

Linear System of Equations

Linear system of equations.

- N linear equations in N unknowns.
- Matrix notation: find x such that Ax = b.

$$\begin{array}{rcrcrcr} 0\,x_0 & + & 1\,x_1 & + & 1\,x_2 & = & 4 \\ 2\,x_0 & + & 4\,x_1 & - & 2\,x_2 & = & 2 \\ 0\,x_0 & + & 3\,x_1 & - & 15\,x_2 & = & 36 \end{array}$$

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & -15 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$$

Among most fundamental problems in science and engineering.

- Linear regression.
- Kirchoff's current law.
- Polynomial and spline interpolation.
- Linear and nonlinear optimization.
- Numerical solution to differential equations.
- Fluid flow,
- Leontief model of economic equilibrium.

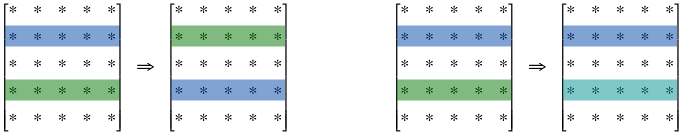
Gaussian Elimination

Gaussian elimination.

- Among oldest and most widely used solutions.
- Repeatedly apply row operations until system is *upper triangular*.
- Solve "trivial" upper triangular system.

Row operations.

- Exchange any two rows.
- Add a multiple of one row to another.



Key invariant. Row operations preserve solutions.

Gaussian Elimination: Forward Elimination

$$\begin{array}{rcrcrcr} 0\,x_0 & + & 1\,x_1 & + & 1\,x_2 & = & 4 \\ 2\,x_0 & + & 4\,x_1 & - & 2\,x_2 & = & 2 \\ 0\,x_0 & + & 3\,x_1 & + & 15\,x_2 & = & 36 \end{array}$$

Interchange row 0 and 1.

$$\begin{array}{rcrcrcr} 2\,x_0 & + & 4\,x_1 & - & 2\,x_2 & = & 2 \\ 0\,x_0 & + & 1\,x_1 & + & 1\,x_2 & = & 4 \\ 0\,x_0 & + & 3\,x_1 & + & 15\,x_2 & = & 36 \end{array}$$

Subtract 3x row 1 from row 2.

$$\begin{array}{rcrcrcr} 2\,x_0 & + & 4\,x_1 & - & 2\,x_2 & = & 2 \\ 0\,x_0 & + & 1\,x_1 & + & 1\,x_2 & = & 4 \\ 0\,x_0 & + & 0\,x_1 & + & 12\,x_2 & = & 24 \end{array}$$

Gaussian Elimination: Back Substitution

Back substitution. *Upper triangular* systems are easy to solve.

$$\begin{array}{rcrcrcr} 2\,x_0 & + & 4\,x_1 & - & 2\,x_2 & = & 2 \\ 0\,x_0 & + & 1\,x_1 & + & 1\,x_2 & = & 4 \\ 0\,x_0 & + & 0\,x_1 & + & 12\,x_2 & = & 24 \end{array}$$

- Equation 2: $x_2 = 24/12 = 2$.
- Equation 1: $x_1 = 4 - x_2 = 2$.
- Equation 0: $x_0 = 2 - 4x_1 + 2x_2 = -1$.

```
for (int i = N-1; i >= 0; i--) {
   double sum = 0.0;
   for (int k = i+1; k < N; k++)
      sum += A[i][k] * x[k];
   x[i] = (b[i] - sum) / A[i][i];
}
```

$$x_i = \frac{1}{A_{ii}} \left[b_i - \sum_{i+1}^{N-1} A_{ik}\, x_k \right]$$

Gaussian Elimination: Forward Elimination

Forward elimination. Apply row operations to make upper triangular.

Pivot. Zero out entries below pivot A_{ii}.

$$A_{jk} = A_{jk} - \frac{A_{ji}}{A_{ii}} A_{ik}$$

$$b_j = b_j - \frac{A_{ji}}{A_{ii}} b_i$$

$$\begin{array}{c|cccccc|c} & & & i & & & & \\ & * & * & * & * & * & * & \\ & 0 & * & * & * & * & * & \\ & 0 & 0 & * & * & * & * & \\ i & 0 & 0 & * & * & * & * & \\ & 0 & 0 & * & * & * & * & \\ & 0 & 0 & * & * & * & * & \end{array} \Rightarrow \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \end{bmatrix}$$

```
for (int i = 0; i < N; i++) {
   for (int j = i + 1; j < N; j++)
      b[j] -= (A[j][i] / A[i][i]) * b[i];
   for (int j = i + 1; j < N; j++)
      for (int k = N - 1; k >= i; k--)
         A[j][k] -= (A[j][i] / A[i][i]) * A[i][k];
}
```

Gaussian Elimination: Partial Pivoting

Observation. Previous code fails spectacularly if pivot $A_{ii} = 0$.

Partial pivoting. Swap row i with the row that has biggest entry in column i among unreduced rows j > i.

$$\begin{array}{c|cccccc|} & & & i & & & \\ & * & * & * & * & * & * \\ & 0 & * & * & * & * & * \\ i & 0 & 0 & 0 & * & * & * \\ & 0 & 0 & 3 & * & * & * \\ \text{max} & 0 & 0 & 9 & * & * & * \\ & 0 & 0 & 2 & * & * & * \end{array}$$

```
// find pivot row
int max = i;
for (int j = i + 1; j < N; j++)
   if (Math.abs(A[j][i]) > Math.abs(A[max][i]))
      max = j;

// swap rows i and max of A and b
double[] T = A[i]; A[i] = A[max]; A[max] = T;
double   t = b[i]; b[i] = b[max]; b[max] = t;
```

Gaussian Elimination: Pathologies

Degeneracy. Partial pivot on a value close to zero.

- System is overdetermined: no solutions.
- System is underdetermined: many solutions.

Numerical stability. Floating point roundoff error swamps computation.

- Partial pivoting helps control roundoff error.
- Pathological instances exist that blow up partial pivoting.

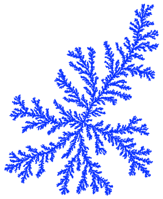
Ill-conditioning. Some problems are inherently unsuitable for floating point solution techniques.

Scientific computing. Much of hard work in designing numerical algorithms is addressing such pathologies.

Diffusion Limited Aggregation

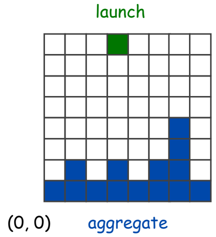
Diffusion limited aggregation (DLA).

- Models formation of an aggregate on a surface.
  - growth of lichen on rocks
  - growth of coral reef
  - generation of polymers out of solutions
  - path of electrical discharge
  - urban settlement
  - carbon deposits on walls of a cylinder of Diesel engine



Monte Carlo simulation.

- Launch particle from *launch site.*
- Particle randomly wanders through 2-D grid until
  - it comes in contact with another particle ⇒ sticks to *aggregate*
  - it enters *kill zone*
- Repeat.



Two Dimensional Arrays in Java

Two dimensional arrays.

```
double a00, a01, a02, a03, a10, a11, a12, a13;
```

vs.

```
double[][] a = new double[2][4];
```

- To access element (i, j) of array named `a`, use `a[i][j]`.

```
public class DLA {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      StdDraw.create(N, N);
      int launch = N - 10;

      boolean[][] dla = new boolean[N][N];
      for (int x = 0; x < N; x++)
         dla[x][0] = true;
```

← N-by-N grid

← launch near top row

← is site i-j occupied?

← initialize aggregate at row 0

Diffusion Limited Aggregation

```
while (!done) {
   int x = (int) (N * Math.random());
   int y = launch;

   while (x < N - 2 && x > 1 && y < N - 2 && y > 1) {
      double r = Math.random();
      if      (r < 0.25) x--;
      else if (r < 0.50) x++;
      else if (r < 0.65) y++;
      else               y--;

      if (dla[x-1][y] || dla[x+1][y] || dla[x][y-1] || dla[x][y+1]) {
         dla[x][y] = true;
         StdDraw.go(x, y);
         StdDraw.spot();
         StdDraw.pause(10);
         if (y > launch) done = true;
         break;
      }
   }
}
```

← launch from random column near top row

particle not in kill zone

← random step

check for contact with a neighboring particle

aggregate reaches top

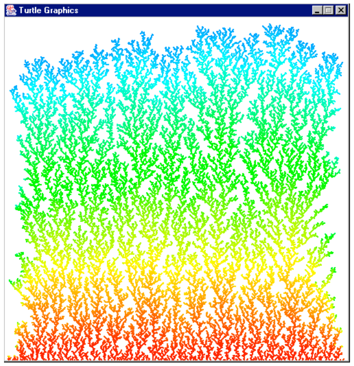
breaks out of innermost while loop

Diffusion Limited Aggregation

Refinements.

- Use diagonals as neighbors, instead of just horizontal and vertical.
- Color particles in launch order, according to rainbow.

```
% java DLA 500
```



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*