Lecture 11 - Data Structures (Part 1)

DSE 511

Drew Schmidt
2022-09-29

Announcements

- Nothing unresolved from last time
- Homework is live!
  - Due date moved to Wed Oct 5 (midnight)
  - `fetch` vs `pull`
  - New homework *not* next week!
- Next week is fall break!
  - No class Thursday Oct 6
- Questions?

Content

- Intro to Data Structures and Algorithms
- Fundamental Type
- Arrays

Intro to Data Structures and Algorithms

Data Structures and Algorithms

- Multi-course CS topic
- CS talk endlessly about sorting algorithms
- Also obsessed with trees for some reason
- For this course
    - Staying mostly high level (arrays are an exception)
    - Only what's (mostly) relevant for data science

Some Important Data Structures for Data Science

- Arrays
- Dataframe
- Hash tables
- Network/tree structures

What's The Point?

- There are often *good* and *bad* ways to do things on a computer
- How do you know?
- Define "good" and "bad"?
- How can you even tell?

Complexity

- "Big O"
- Describes asymptotic behavior of a function
- Used in CS to understand runtime behavior as data grows
- More on this later

Specification vs Implementation

- Some data structures are "abstract"
- Implemented using other data structures!
- Example: a queue
  - Receives data FIFO
  - `pop()` removes the first element
  - `push()` adds element to the back
  - Can implement via linked list, hash table, array, …
- Example: a dictionary
  - Key/value storage
  - Can be implemented via hash table (and others)

Fundamental Type

Fundamental Type

- How is data *actually* stored in memory?
- Computers don't know fancy things
  - Dataframes
  - Lists
  - Hash tables
  - Graphs
- Computers know *blocks of memory*
- But blocks of *what*?

Important Fundamental Types

- `int` - integer, at least 4 bytes
- `float` - floating point number, exactly 4 bytes
- `double` - floating point number, exactly 8 bytes
- `char` - a character (part of a string), usually 1 byte

Fundamental Types in R and Python

- Most computation (R, Python) automatically in double precision
- 32-bit Float:
  - half the memory
  - twice as fast (roughly)
  - not as accurate
- Python
  - Supported in numpy
  - `np.random.rand(3).astype('f')`
  - `np.array([1, 2, 3, 4], dtype='f')`
- R
  - float package https://cran.r-project.org/package=float
  - fmlr package https://hpcran.org/packages/fmlr/index.html

Fundamental Types in R and Python

R

```
typeof(1)
```

```
## [1] "double"
```

```
typeof(2)
```

```
## [1] "double"
```

```
typeof(c(1, 2))
```

```
## [1] "double"
```

```
typeof(1:2)
```

```
## [1] "integer"
```

Python

```
type(1)
```

```
## <class 'int'>
```

```
type(1.0)
```

```
## <class 'float'>
```

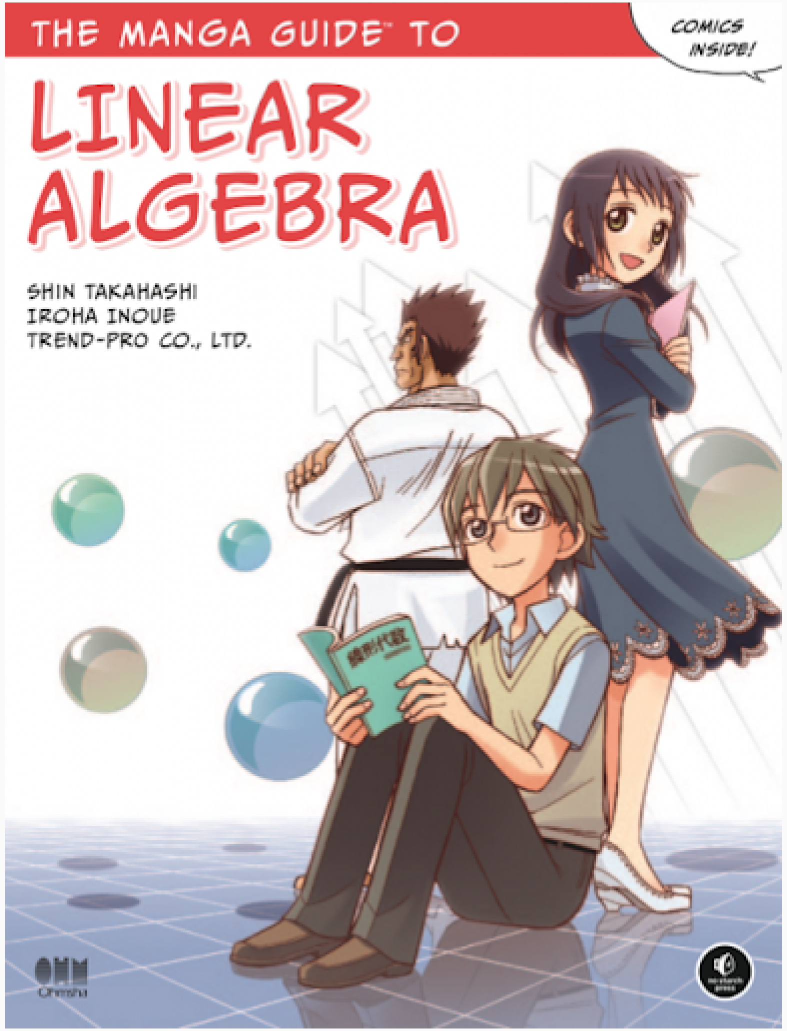
Arrays

What Is An Array?

- A contiguous block of memory
- A "collection" of data of *the same fundamental type*
    - An array of numbers
    - An array of strings
    - An array of *pointers* (lists in R!)
- The data structure that powers linear algebra
    - Numeric arrays

Linear Algebra

- LA dominates scientific and data computing
- Some uses in data:
  - PCA - SVD
  - Linear Models - QR
  - Covariance/correlation - gemm/syrk
  - Inverse - Cholesky, LU
- 1970's: LINPACK (not that one)
- 1980's: BLAS, LAPACK
- 1990's: ScaLAPACK
- 2000's: PLASMA, MAGMA
- 2010's: ~~DPLASMA~~ SLATE



Some Terminology

- **gemm** - matrix-matrix multiply
- **BLAS** - Basic Linear Algebra Subprograms; matrix library
- **FLOPS** - Floating Point Operations Per Second (adds and multiplies)
- **LINPACK** - Solve $Ax = b$
- **TOP500** - list of computers ranked by LINPACK benchmark

Examples of Arrays

- A numeric vector
- A matrix
- A "tensor" (multi-dimensional array)

```
set.seed(1234)
x = rnorm(3)
x
```

```
## [1] -1.2070657  0.2774292  1.0844412
```

```
x[2]
```

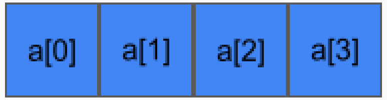
```
## [1] 0.2774292
```

Arrays

- How do we get from an array to a matrix?
- Recall: arrays are *contiguous blocks of memory*

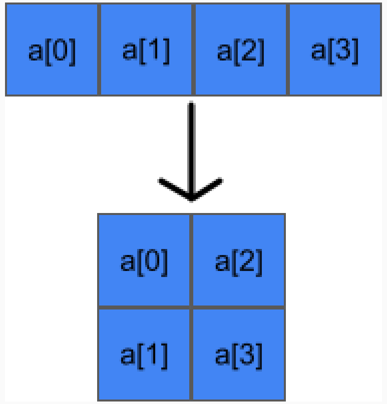
Answer: by convention

Arrays



a[0] a[1] a[2] a[3]

Arrays

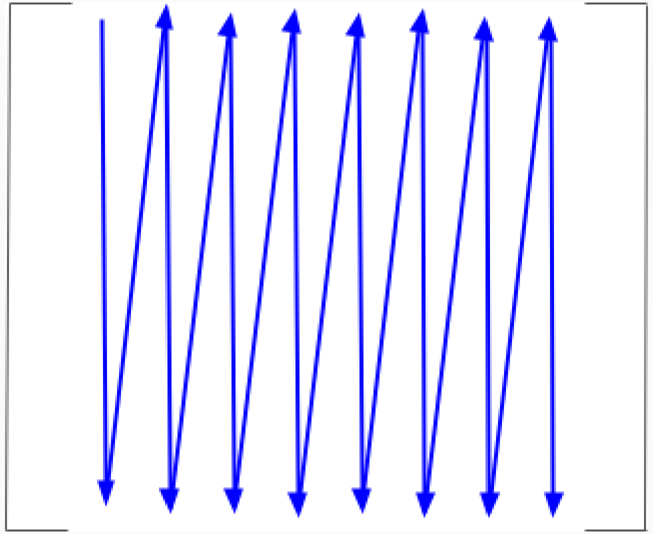


a[0]	a[1]	a[2]	a[3]

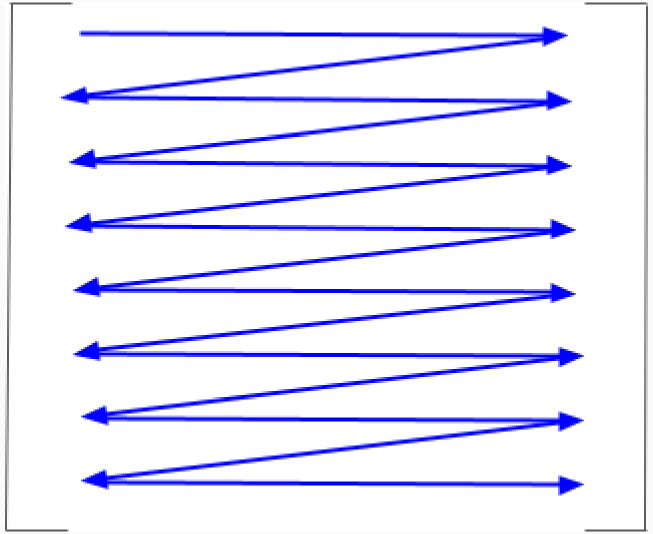
a[0]	a[2]
a[1]	a[3]

Major Ordering

Column Major



Row Major



Major Ordering

Column-Major

- R
- Fortran

Row-Major

- Python
- C/C++ (typically)

Matrices

- Assume x is $m \times n$ matrix
- Use 0-based indexing

Column Major

$X[i, j] = X[i + m \cdot j]$

```
for (int j=0; j<n; j++) {
  for (int i=0; i<m; i++) {
    X[i + m*j]
  }
}
```

Row major

$X[i, j] = X[i \cdot n + j]$

```
for (int i=0; i<m; i++) {
  for (int j=0; j<n; j++) {
    X[i*n + j]
  }
}
```

Matrices

```
x = matrix(1:20, 5)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```
x[4, 3]
```

```
## [1] 14
```

```
x[(4-1) + (3-1)*5 + 1]
```

```
## [1] 14
```

Multi-Dimensional Arrays

- Assume x is an $n_1 \times n_2 \times n_3$ array
- Use 0-based indexing

Column Major

$X[i, j, k] = X[i + n_1 \cdot j + (n_1 \cdot n_2) \cdot k]$

```
for (int k=0; k<n3; k++) {
  for (int j=0; j<n2; j++) {
    for (int i=0; i<n1; i++) {
      X[i + j*n1 + k*(n1*n2)])
    }
  }
}
```

Multi-Dimensional Arrays

```
n1 = 2; n2 = 3; n3 = 2
x = array(
  1:(n1*n2*n3),
  dim = c(n1, n2, n3)
)
x
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

```
x[1, 2, 2]
```

```
## [1] 9
```

```
x[(1-1) + (2-1)*n1 + (2-1)*n1*n2 + 1]
```

```
## [1] 9
```

Wrapup

Wrapup

- Fundamental types
  - There are many
  - For data science, you'll probably stick to `int`, `float`, and `double`
- Arrays power almost everything in data science!
  - Linear models
  - Dimension reduction
  - Fancy neural network models
  - ...

Questions?