Data Structures and Algorithms 2019
Review for Screening Test

Thai Dinh, Kien Nguyen,Y Nguyen, Huy Pham

January 6, 2019

Basic Data Types

- Built-in Data Types
- Derived Data Types
- User-defined Data Types

Built-in Data Types

- Predefined and can be used directly
- Examples:
    - Character: char (1 byte)
    - Integral: int (4 byte), short (2 byte), long (8 byte), byte (1 byte)
    - Floating: float (4 byte), double (8 byte)
    - Boolean: true or false value (1 byte)

Examples of Built-in Data Types

Listing 1: C++/Java (static-typed)

```
char c = 'a';
int i  = 5;
double d = 1.6;
```

Listing 2: Python (dynamic-typed)

```
c = "aaa"
i = 5
d = 1.6
```

Derived Data Types

- Data types that are derived from fundamental data types
- Don't create a new data type, but instead add some functionalities to the basic data types
- Examples:
  - Array: collection of variables of the same type
  - Pointer (C++): variable that holds a memory address of another variable

Examples of Derived Data Types

Listing 3: C++ pointer and reference

```
int c  = 9;
int* pointer_c = &c;
int& ref_c = c;
```

Listing 4: C-style array

```
int arr[5] = {1, 2, 3, 4 ,5};
```

User-defined Data Types

- Create your own customized data types
- Examples:
  - Class
  - Structure
  - Enumeration

Examples of User-defined Data Types

Listing 5: Enum

```
enum Color { red, green, blue };
Color r = red;
```

Examples of User-defined Data Types

Listing 6: Struct example in C++

```
struct Complex {
    double real;
    double imag;
};
struct Student {
    std::string name;
    int age;
};
Complex c = {1.0, 2.0};
Student good = {"Hello", 9};
```

Listing 7: Class example in Java

```
class Complex {
    double real;
    double imag;
    public Complex(double real, double img)
      {
        this.real = real;
        this.imag = img;
    }
}
Complex c = new Complex(1.0, 2.0);
```

Examples of User-defined Data Types

Listing 8: Class example in Python

```
class Complex:
    def __init__(self, real, img):
        self.real = real
        self.imag = img

c = Complex(1.0, 2.0)
```

Logical Operators

C++	**Java**	**Python**	**Operator**
! exp	! exp	**not** exp	Logical NOT
exp && exp	exp && exp	exp **and** exp	Logical AND
exp || exp	exp || exp	exp **or** exp	Logical OR

Equality & Comparison Operators

C++	Java	Python	Operator
`exp < exp`	`exp < exp`	`exp < exp`	Less than
`exp > exp`	`exp > exp`	`exp > exp`	Greater than
`exp <= exp`	`exp <= exp`	`exp <= exp`	Less than or equal
`exp >= exp`	`exp >= exp`	`exp >= exp`	Greater than or equal
`exp == exp`	`exp == exp`	`exp == exp`	Equal to
`exp != exp`	`exp != exp`	`exp != exp`	Not Equal to
		`is`	Same identity
		`is not`	Different identity

Arithmetic Operators

C++	Java	Python	Operator
+	+	+	Addition
-	-	-	Subtraction
*	*	*	Multiplication
/	/	/ or //	Division (*)
%	%	%	Modulo operator

(*): In Python 3, we use / for True division and // for Integer division.

```
>>> 7/10
0.7
>>> 7//10
0
```

Bitwise Operators

C++	Java	Python	Operator
~	~	~	Bitwise **complement**
&	&	&	Bitwise **and**
|	|	|	Bitwise **or**
^	^	^	Bitwise **exclusive-or**
`x << y`	`x << y`	`x << y`	Shift bits left, filling in with zeros
`x >> y`	`x >> y`	`x >> y`	Shift bits right, filling in with sign bit
	`x >>> y`		Shift bits right, filling in with zeros

Sequence Operators

Python supports built-in sequence types, which is different from C++/Java

C++ data types	Java data types	Python data types	Description
std::list	java.util.List	list	**mutable** sequence of objects
No equivalent	No equivalent	tuple	**immutable** sequence of objects
std::string	String	str	character string

Sequence Operators

Sequence Operators	Description
`s[i]`	Element at index i
`s[start:stop]`	Slice including indices [start, stop)
`s[start:stop:step]`	Slice including indices start, start + step, start + 2*step,... up to but not equaling or stop
`s + t`	Concatenation of sequences
`k * s`	Shorthand for s + s + s + … (k times)
`val in s`	Containment check
`val not in s`	Non-containment check

Set Operators

key in *s*	containment check
key not in *s*	non-containment check
$s == t$	s is equivalent to t
$s! = t$	s is not equivalent to t
$s <= t$	s is a subset of t
$s < t$	s is a proper subset of t
$s >= t$	s is a superset of t
$s > t$	s is a proper superset of t
$s\|t$	the union of s and t
$s\&t$	the intersection of s and t
$s - t$	the set elements in s but not in t
$s\ \hat{}\ t$	the set of elements in precisely s or t

Dictionaries/Map

C++	Java	Python	Description
d[key]	d.get(key)	d[key]	value associated with given key
d[key] = val	d.put(key, val)	d[key] = val	set (or reset) the value

Dictionaries/Map

C++	**Java**	**Python**	**Description**
d.erase(key)	d.remove(key)	del d[key]	remove key and its associated value
d.find(key) != d.end()	d.containsKey (key)	key in d	containment check
no simple equivalent	d1.equals(d2)	d1 == d2	d1 is equivalent to d2

Extended Assignment Operators for C++

```
int i = 10;
int j = 5;
string s = "yes";

i += 4; // i = i + 4 = 14
j *= -2; // j = j*(-2) = -10
s += " or no" // s = s + " or no" = "yes or no"
```

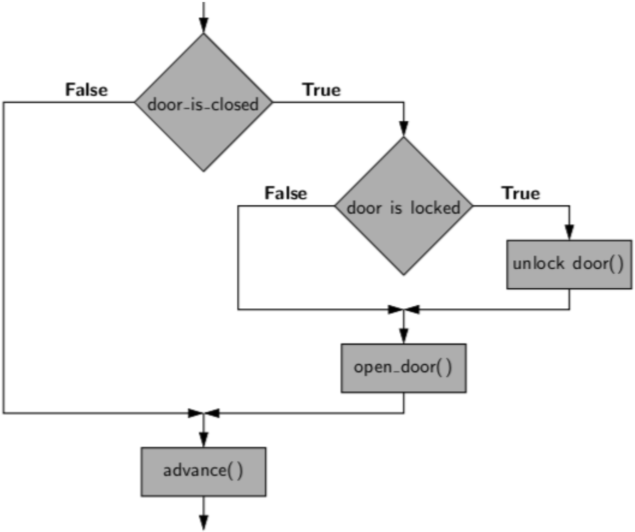
Extended Assignment Operators for Python

```python
a = [1, 2, 3]
b = a # an alias for a
b += [4, 5] # extends the original with 2 elements
b = b + [6, 7] # reassign to a new list
print(a) # [1, 2, 3, 4, 5]
print(b) # [1, 2, 3, 4, 5, 6, 7]
```

Compound Expressions and Operator Precedence

	Operators	Symbol		Operators	Symbol
1	Member access	sd	9	Bitwise-xor	^
2	Function/method calls Container subscripts/slices	exp(...) exp[...]	10	Bitwise-or	|
3	Exponentiation	**	11	Comparisons/containments	is, == ,!=,...
4	Unary operators	+exp, -epx, ~exp	12	Logical-not	**not**
5	Multiplication, division	*, /, //, %	13	Logical-and	**and**
6	Addition, subtraction	+, -	14	Logical-or	or
7	Bitwise-shifting	<<, >>	15	Conditional	val1 **if** cond **else** val2
8	Bitwise-and		16	Assignments	=, +=, ...

An Example Flowchart



Conditionals

<table>
<tr><th>C++</th><th>Java</th><th>Python</th></tr>
<tr>
<td><pre>if (condition1){
    first_body;
} else if (condition2){
    second_body;
} else {
    third_body;
}</pre></td>
<td><pre>if (condition1){
    first_body;
} else if (condition2){
    second_body;
} else {
    third_body;
}</pre></td>
<td><pre>if condition1:
    first_body
elif condition2:
    second_body
else:
    third_body</pre></td>
</tr>
</table>

While Loops

C++	Java	Python

`while (condition){`
`    body_statement;`
`}`
 	
`do {`
`    body_statement;`
`} while (condition)`
	`while (condition){`
	`    body_statement;`
	`}`
	
	`do {`
	`    body_statement;`
	`} while (condition)`
		`while condition:`
		`    body_statement`

For Loops

<table>
<tr><th>C++</th><th>Java</th><th>Python</th></tr>
<tr><td colspan="2"><code>for (initialization, condition, increment) {</code>
<code>   body_statement;</code>
<code>}</code></td><td><code>for element in iterable:</code>
<code>   body</code></td></tr>
</table>

Break Statements

Listing 9: Break statement example in Python

```
found = False
for student_name in dsa_class_list:
    if student_name == "Y␣Nguyen":
        found = true
        break
```

Listing 10: Continue statement example Python

```
count = 0
for student_name in dsa_class_list:
    if student_name == "Y␣Nguyen":
        continue
    count += 1
```

Listing 11: Function example in Python

```
def count_students_by_name(
    dsa_class_list,
    name_of_interest
    ):
    count = 0
    for student_name in dsa_class_list:
        if student_name == name_of_interest:
            count += 1
    return count
```

Functions

- Information passing
- Mutable parameters
- Default parameter values
- Keyword parameters

High-level Languages

- Represent a giant leap towards easier programming
- Syntax similar to the English language
- Divided into 2 groups
  - Procedural languages
  - Object-oriented languages

Procedural Languages

- Specify the sequence of steps that implements a particular Algorithms
- Revolves around keeping code as concise as possible
- Focus on a very specific end result to be achieved
- Examples:
  - C
  - Fortran
  - Pascal

Object-Oriented Languages

- Focus not on structure, but on modeling data
- Programmers code using blueprints of data models called classes
- Examples
  - Java
  - C++

Object-Oriented Programming

- A design program philosophy
- Key idea: the real world can be accurately described as a collection of objects that interact
- Everything is grouped as objects $\rightarrow$ enhance reusability

OOP Basic Terminology

- Object
- Method
- Attribute
- Class

Classes and Objects

- A class is a prototype, idea, and **blueprint** for creating objects
- An object is an **instance** of a class
- A class has a constructor for creating objects
- A class is composed of 3 things
  - Name
  - Attributes
  - Methods

Formal Definition of Objects

- A computational entity that:
  - Encapsulates some state
  - Is able to perform actions/methods on its state
  - Communicates with other objects

Listing 12: Class example C++

```
class Human {
private:
    std::string name;
public:
    void Human(std::string name) {
        this->name = name;
    }
}
    std::string getName() {
        return name;
    }
}
Human thai =  Human("Thai␣Dinh");
```

Classes and Objects Examples

Listing 13: Class example Java

```
class Human {
    private String name;
    pulic Human(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
Human thai = new Human("Thai␣Dinh");
```

Listing 14: Class example Python

```
class Human:
    def __init__(self, name):
        self.name = name

    def getName(self):
        return self.name

thai = Human("Thai␣Dinh")
```

OOP Basic Concepts

- Encapsulation
- Inheritance
- Abstraction

Encapsulation

- Inclusion of properties and methods within a class/object
- Enable reusability

Encapsulation

- Inclusion of properties and methods within a class/object
- Enable reusability

Encapsulation Examples

Listing 15: Java access example

```
Human thai = new Human("Thai␣Dinh");
System.out.println(thai.name); // error
    because  attribute name is private
System.out.println(thai.getName()); // valid
     because method getName is public
```

Note: Python is not a strongly-encapsulated language (no real access modifier mechanism).

Inheritance

- Allow class hierarchy
- Enable reusability

Inheritance Examples

Listing 16: Java inheritance example

```
class SuperHero extends Human {
    private String hero_name;
    public SuperHero(String normal_name,
        String hero_name) {
         super(normal_name);
         this.hero_name = hero_name;
    }
    public String getNick() {
         return hero_name;
    }
    public void doGoodThing {}
}
```

Listing 17: Java inheritance example

```
class Villain extends Human {
    private String villain_name;
    public Villain(String normal_name,
        String villain_name) {
         super(normal_name);
         this.villain_name = villain_name;
    }
    public String getNickname {
         return villain_name;
    }
    public void doBadThing {}
}
```

Abstraction

- Allows programmers to represent complex real world in the simplest manner
- When we design abstract classes, we define the framework for later extensions

Listing 18: Java abstract example

```
abstract class Animal {
    abstract void move();
}
class Dog extends Animal {
    void move() {}
}
class Cat extends Animal {
    void move() {}
}
```

There are no instance of general animal. It must be a specific type of animal.

Advantages of OOP

- Code reuse and recycling
- Improved productivity
- Improved maintainability
- Faster development
- Higher quality, lower cost of software development

Disadvantages of OOP

- Steep learning curve
- Could lead to larger programs
- Could produce slower programs

Exponents

$$
\begin{aligned}
X^A X^B &= X^{A+B} \\
\frac{X^A}{X^B} &= X^{A-B} \\
\left(X^A\right)^B &= X^{AB} \\
X^N + X^N &= 2X^N \neq X^{2N} \\
2^N + 2^N &= 2^{N+1}
\end{aligned}
$$

Logarithms

All logarithms are to the base 2 unless specified otherwise.

Definition

$X^A = B$ if and only if $log_X B = A$.

Theorem

$$log_A B = \frac{log_C B}{log_C A} \quad \textit{where } A, B, C > 0,\ A \neq 1.$$

Theorem

$$log(AB) = logA + logB \quad A, B > 0$$

Some other useful formulas

$$log(A/B) = logA - logB$$
$$log(A^B) = B\ logA$$
$$logX < X \quad \forall X > 0$$

Geometric series

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$

$$\sum_{i=0}^{N} A^i = \frac{A^{N+1} - 1}{A - 1}$$

If $0 < A < 1$ then

$$\sum_{i=0}^{N} A^i < \frac{1}{1 - A}$$

Series

$$\sum_{i=0}^{\infty} A^i = \frac{1}{1-A}$$

$$\sum_{i=1}^{\infty} \frac{i}{2^i} = \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \ldots = 2$$

Example: Toss a fair coin ($P(H) = P(T) = 0.5$) until it turns Head. Calculate the expected number of tosses.

Arithmetic series

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3}$$

$$\sum_{i=1}^{N} i^k \approx \frac{N^{(k+1)}}{|k+1|} \quad k \neq -1$$

Harmonic series

$$\sum_{i=1}^{N} \frac{1}{i} \approx log_e N$$

Modular Arithmetic

- We say that A is congruent to B modulo N, written $A \equiv B \ (mod \ \ N)$, if N divides $A - B$.
- This means the remainders are the same when A and B are divided by N.
- For example, $7 \equiv 22 \ (mod \ 5)$.
- If $A \equiv B \ (mod \ \ N)$ then $A + C \equiv B + C \ (mod \ \ N)$ and $AD \equiv BD \ (mod \ \ N)$.
- If N is prime, then $ab \equiv 0 \ (mod \ \ N)$ if and only if $a \equiv 0 \ (mod \ \ N)$ or $b \equiv 0 \ (mod \ \ N)$.

Proof Techniques

- Direct proof (proof by construction)
- Proof by induction
- Proof by counter-example
- Proof by contradiction
- Proof by contrapositive

Direct Proof

In order to prove $P \Rightarrow Q$, we assume P is true and use P to show that Q must be true.

Example: Prove that if a and b are consecutive integers, then the sum $a + b$ is odd.

Proof. b can be written as $b = a + 1$. Thus $a + b = 2a + 1$. Therefore $a + b$ is odd. $\square$

Proof by induction

- Prove the statement for a base case (or some base cases)
- Prove that if the statement holds for all the cases up to k (which is finite), then it also holds for the case $k+1$.

Example: Prove that $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$ for $N \geq 1$
Proof. It is easy to show that it holds when $N = 1$. Now suppose that it holds for $N = k$ $(k \geq 1)$ or $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$. We than have $\sum_{i=1}^{k+1} i = \frac{k(k+1)}{2} + (k+1) = \frac{(k+1)(k+2)}{2}$. Thus the statement also holds for $N = k+1$. $\square$

Proof by counter-example

Prove that a statement is false by giving a counter-example.
Example: Disprove the statement "All prime numbers are odd numbers."
Proof. Counter-example: 2 is a prime number but is not an odd number. □

Proof by contradiction

In order to prove $P \Rightarrow Q$, we assume P is true, $\neg Q$ is true and demonstrate a contradiction.

Example: (Pigeonhole principle) If n items are placed into m containers, and $n > m$, then there exists at least one container with more than one item.

Proof. Assume $n > m$ and there is no container with more than one item. Then the number of items is less than or equal to m, or $n \leq m$. This is contradictory to the assumption $n > m$. $\square$

Proof by contrapositive

In order to prove $P \Rightarrow Q$, we prove $\neg Q \Rightarrow \neg P$.
Example: Prove that if x^2 is odd then x must be odd.
Proof. If x is even then x^2 is even. $\square$

- A recursive function is a function that is defined in terms of itself
- May have one or more base cases where the return value can be calculated / are known without resorting to recursion.

Example: Write a function to calculate $n!$ where n is a non-negative integer.

Listing 19: Recursive function to calculate $n!$

```
def my_factorial(n):
    # Sanity check the input, not shown
    if n > 0:
        return n*my_factorial(n-1)
    # base case
    else:
        return 1
```

Permutations

Result

Number of different ordered arrangements (permutations) of n distinct objects

$$P_n = n(n-1)(n-2)\dots 3.2.1 = n!$$

Example 1: How many ways can you arrange 4 students in a row of 4 chairs?

Example 2: How many different letter arrangements can we get from the letters B, E, T, T, E, R?

Combinations

Result

Number of different groups of r objects taken from n objects $(0 \le r \le n)$

$$\binom{n}{r} = \frac{n!}{(n-r)!r!}$$

Example 1: An attends a Data Structures and Algorithms class at the VEF Academy which has a total of 28 students (including An). An wants to find 3 more students in the class to form a group of 4 (which An plans to name "Fantastic Four") for the term projects. How many ways can An pick the 3 students?

Example 2: (Binomial Theorem) Expand the polynomial $(x+y)^n$ and explain the coefficients!

Conditional Probability

$P(E|F)$ is the conditional probability that E occurs given that F has occurred.

Definition

If $P(F) > 0$ then

$$P(E|F) = \frac{P(EF)}{P(F)}$$

Example: (from Ross [1]) A fair coin is tossed twice. Assuming that the two coin tosses are independent, what is the conditional probability that both tosses land on heads, given that (a) the first flip lands on heads? (b) at least one flip lands on heads?

Example: (a) Let $B = \{(H, H)\}$ be the event that both tosses land on heads, $F = \{(H, H), (H, T)\}$ be the event that the first toss land on heads.

$$P(B|F) = \frac{P(BF)}{P(F)} = \frac{1/4}{1/2} = 1/2$$

Independent Events

Definition

Two events E and F are said to be independent if $P(EF) = P(E)P(F)$. Two events E and F that are not independent are said to be dependent (Ross [1]).

Example: A card is drawn at random from an ordinary deck of 52 playing cards. If E is the event that the selected card is a Queen and F is the event that it is a diamond, then E and F are independent. We have that $P(EF) = 1/52$, whereas $P(E) = 4/52$ and $P(F) = 13/52$.

Random Variables

- A real-valued function defined on the outcome of a probability experiment is called a random variable (r.v.).
- $F(x) = P\{X \leq x\}$ is called the distribution function (a.k.a. cumulative distribution function, or cdf) of X.
- A random variable whose set of possible values is either finite or countably infinite is called discrete.
- If X is a discrete r.v. then the function $p(x) = P\{X = x\}$ is called the probability mass function of X.
- The expected value of X (also mean or expectation of X) is given by

$$E[X] = \sum_{x:p(x)>0} xp(x)$$

Random Variables

- For a function g,

$$E[g(X)] = \sum_{x:p(x)>0} g(x)p(x)$$

- The variance of X

$$var(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2$$

- The standard deviation of X: $SD(X) = \sqrt{Var(X)}$
- X is the random value whose value is equal to the value from rolling a die. Calculate $E(X)$, $var(X)$, and $SD(X)$.

Discrete Uniform Random Variables

unif $\{a, b\}$ $(a \leq b)$: The pmf, expected value, and variance are given by

$$
\begin{aligned}
\mathbb{P}(i) &= \frac{1}{b-a+1}, \; i = a, \ldots, b \\
E[X] &= \frac{a+b}{2} \\
Var(X) &= \frac{(b-a+1)^2 - 1}{12}
\end{aligned}
$$

Binomial Random Variables

Binomial(n, p)$(0 < p < 1, n \geq 1)$: The pmf, expected value, and variance are given by

$$\begin{aligned}
\mathbb{P}(i) &= \binom{n}{i} p^i (1-p)^{(n-i)} \\
&= \frac{n!}{i!(n-i)!}\, p^i (1-p)^{(n-i)} \\
&\quad \text{where } 0 \leq i \leq n \\
E[X] &= np \\
Var(X) &= np(1-p)
\end{aligned}$$

Note that when $n = 1$, this becomes a Bernoulli random variable.

pmf of Binomial(10, 1/2)

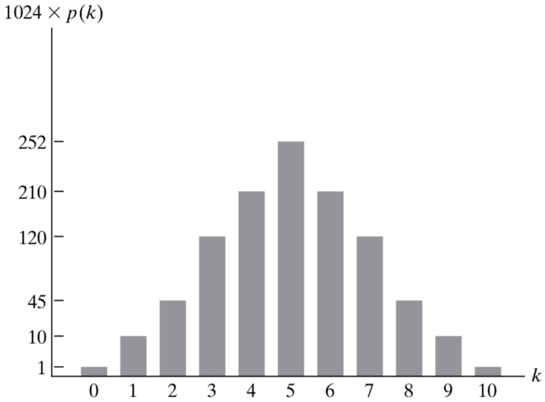


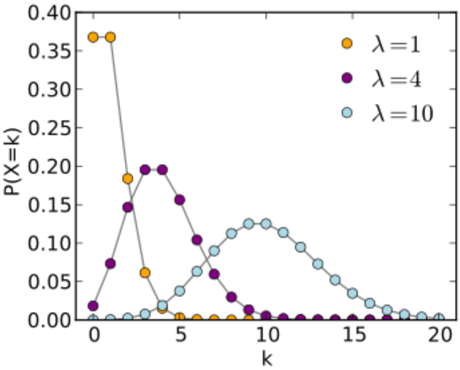
FIGURE 4.5 Graph of $p(k) = \binom{10}{k}\left(\frac{1}{2}\right)^{10}$

Poisson Random Variables

Poisson(λ): ($\lambda \geq 0$): The pmf, expected value, and variance are given by

$$\begin{aligned} \mathbb{P}(i) &= \frac{e^{-\lambda}\lambda^i}{i!} \qquad i \geq 0 \\ E[X] &= Var(X) = \lambda \end{aligned}$$

pmf of Poisson(λ)



Geometric Random Variables

Geo(p): $(0 < p < 1)$: The pmf, expected value, and variance are given by

$$\begin{aligned} \mathbb{P}(i) &= p(1-p)^{i-1} \\ E[X] &= \frac{1}{p} \\ Var(X) &= \frac{1-p}{p^2} \end{aligned}$$

References

[1] M. H. Goldwasser, M. T. Goodrich, and R. Tamassia – Data Structures and Algorithms in Python, John Wiley & Sons, 2013
[2] Oracle Java Tutorials
[3] S. Ross, A First Course in Probability, 6th Ed, Prentice Hall, 2002
[4] M. A. Weiss. Data Structures and Algorithm Analysis in Java. Mark Allen Weiss. Pearson Education, 2012.