Ghi chú của một coder

Vũ Anh

Tháng 02 năm 2018

Mục lục

Chương 1

Lời nói đầu

Text will come here

Đọc quyển Deep Learning quá xá hay luôn. Rồi lại đọc SLP 2. Thấy sao các thánh viết hay và chuẩn thế (đấy là lý do các thánh được gọi là ... các thánh chăng =))
Tính đến thời điểm này đã được 2 năm 10 tháng rồi. Quay lại với latex. Thỏa mãn được điều kiện của mình là một tool offline. Mình thích xuất ra pdf (có gì đọc lại hoặc tra cứu cũng dễ).
Hi vọng gắn bó với thằng này được lâu.
Chào từ hồi magizbox.wordpress.com, cái này tồn tại được 77 ngày (hơn 2 tháng) (từ 01/11/2017 đến 17/01/2018)

Chào Khách,
Mình là Vũ Anh. Tính đến thời điểm viết bài này thì đã lập trình được 7 năm (lập trình từ hồi năm 2010). Mình thích viết lách, bằng chứng là đã thay đổi host 2 lần datayo.wordpress.com, magizbox.com. Thành tựu ổn nhất hiện tại chỉ có một project underthesea,
xếp loại tạm được.
Blog này chứa những ghi chép loạn cào cào của mình về mọi thứ. Đúng với phong cách "vô tổ chức" của mình. Chắc chắn nó sẽ không hữu ích lắm với bạn. Tuy nhiên, cảm ơn bạn đã ghé qua.
Nếu Khách quan tâm, thì mình chỉ post bài xàm vào thứ 7 thôi nhé. Những ngày còn lại chỉ post bài nghiêm túc thôi. (bài này quá xàm nhưng được post vào thứ 5 nhé)
Làm sao để thực hiện blog này <ul> Viết markdown và latex hỗ trợ bởi wordpress Server cho phép lưu ảnh động giphy Vấn đề lưu trữ ảnh: sử dụng tính năng live upload của github.com

Bỏ cái này vì quá chậm. Không hỗ trợ tốt latex (công thức toán và reference). Mình vẫn thích một công cụ offline hơn.
Chào từ hồi magizbox.com, cái này tồn tại được 488 ngày (1 năm 4 tháng. wow) (từ 01/07/2016 đến 01/11/2017)

Hello World,
My name is Vu Anh. I'm a developer working at a startup in Hanoi, Vietnam. Coding and writing is fun, so I make this site to share my gists about computer science, data science, and more. It helps me keep my hobby and save information in case I forget. I wish it will be useful for you too.
PS: I always looking for collaboration. Feel free to contact me via email brother.rain.1024[at]gmail.com
Magizbox Stories
Oct 2, 2016: Wow. It's 524th day of my journey. I added some notes in README.md, index.html, changed structure of website. Today I feel like at begin day when I start writing at datayo.wordpress.com blog. In this pass, there are times when I want to make a professional website like tutorialpoints but it doesn't work that way. Because in my heart, I don't want it, I don't to make a professional website. I just love coding, writing and sharing my hobby with people around the world. So today I come back to starting point, I will keep my writing schedule, make some fun stuffs each week.
In July 2016, I turn to use HTML and mkdocs, and opensource magizbox.
In March 2015, I start writing blog with wordpress.

Bỏ cái này vì thời gian build quá lằng nhằng. Quản lý dependencies các kiểu rất lâu. Muốn có một cái gì đó giúp viết thật nhanh và đơn giản.

Chào từ hồi datayo.wordpress.com, cái này tồn tại được 489 ngày (1 năm 4 tháng) (từ 01/03/2015 đến 01/07/2016)

I'm a junior data scientist, working as a researcher in big data team at a company in Vietnam. I love listening music when I'm writing code because it's make me coding better. I love reading books before sleeping because it take me sleep easier and discover the world with data mining via beautiful language R.
I write this blog because I want to share my hobbies with everybody. I hope you will enjoy it. Feel free to contact me via twitter @rain$_1$024 $or email brother.rain.1024@gmail.com (I will answer all emails for sure) for anything you want to hel$
In case you find one of my posts could be better, don't hesitate to drop me a line in comment. I'm very appreciated and I will try my best to make it better and better.

Bỏ cái này. Bỏ wordpress. Vì muốn một site interative hơn.

Phần I

Lập trình

Chương 2

Lập trình là gì?

2.1 Các vấn đề lập trình

Các vấn đề lập trình với từng ngôn ngữ

2.1.1 Nhập môn

code/

1. introduction
2. syntax
3. data structure
4. oop
5. networking
6. os
7. parallel
8. event based
9. error handling
10. logging
11. configuration
12. documentation
13. test
14. ui
15. web
16. database
17. ide
18. package manager
19. build tool
20. make module
21. production (docker)

2.2 Introduction

Installation (environment, IDE)
Hello world

Courses
Resources

2.3 Syntax

variables and expressions
conditional
loops and Iteration
functions
define, use
parameters
scope of variables
anonymous functions
callbacks
self-invoking functions, inner functions
functions that return functions, functions that redefined themselves
closures
naming convention
comment convention

2.4 Cấu trúc dữ liệu

Number
String
Collection
DateTime
Boolean
Object

2.5 Lập trình hướng đối tượng

Classes Objects
Inheritance
Encapsulation
Abstraction
Polymorphism
For OOP Example: see Python: OOP

2.5.1 Bài tập

Quản lý tài khoản ngân hàng

2.6 Networking

REST (example with chat app sender, receiver, message)

2.6.1 Bài tập

Guess My Number Game

2.7 GUI - Giao diện

Quản lý hot girl
Quản lý truyện tranh
Create Analog Clock
Chương trình lịch âm dương
Chương trình học từ tiếng Anh

2.8 Game

- Create Pong Game
- Create flappy bird
- Create Bouncing Game

2.9 Cơ sở dữ liệu

2.9.1 Thử thách

2.10 How to ask a question

Focus on questions about an actual problem you have faced. Include details about what you have tried and exactly what you are trying to do.
Ask about...
Specific programming problems
Software algorithms
Coding techniques
Software development tools
Not all questions work well in our format. Avoid questions that are primarily opinion-based, or that are likely to generate discussion rather than answers.
Don't ask about...
Questions you haven't tried to find an answer for (show your work!)
Product or service recommendations or comparisons
Requests for lists of things, polls, opinions, discussions, etc.
Anything not directly related to writing computer programs

2.11 Các nguyên tắc lập trình

Generic
KISS (Keep It Simple Stupid)
YAGNI
Do The Simplest Thing That Could Possibly Work
Keep Things DRY

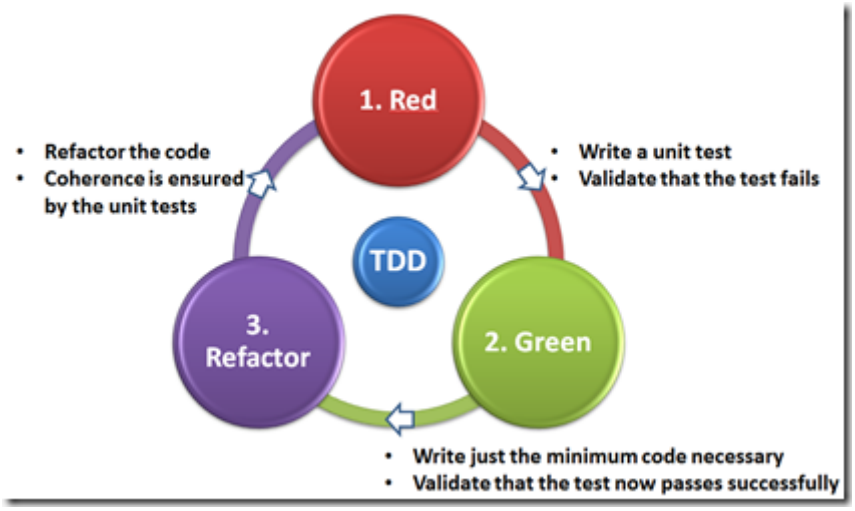
Code For The Maintainer
Avoid Premature Optimization
Inter-Module/Class
Minimise Coupling
Law of Demeter
Composition Over Inheritance
Orthogonality
Module/Class
Maximise Cohesion
Liskov Substitution Principle
Open/Closed Principle
Single Responsibility Principle
Hide Implementation Details
Curly's Law
Software Quality Laws
First Law of Software Quality

2.12 Các mô hình lập trình

Main paradigm approaches 1
1. Imperative
Description:
Computation as statements that directly change a program state (datafields)
Main Characteristics:
Direct assignments, common data structures, global variables
Critics: Edsger W. Dijkstra, Michael A. Jackson
Examples: Assembly, C, C++, Java, PHP, Python
2. Structured
Description:
A style of imperative programming with more logical program structure
Main Characteristics:
Structograms, indentation, either no, or limited use of, goto statements
Examples: C, C++, Java, Python
3. Procedural
Description:
Derived from structured programming, based on the concept of modular programming or the procedure call
Main Characteristics:
Local variables, sequence, selection, iteration, and modularization
Examples: C, C++, Lisp, PHP, Python
4. Functional
Description:
Treats computation as the evaluation of mathematical functions avoiding state and mutable data
Main Characteristics:
Lambda calculus, compositionality, formula, recursion, referential transparency, no side effects
Examples: Clojure, Coffeescript, Elixir, Erlang, F, Haskell, Lisp, Python, Scala, SequenceL, SML

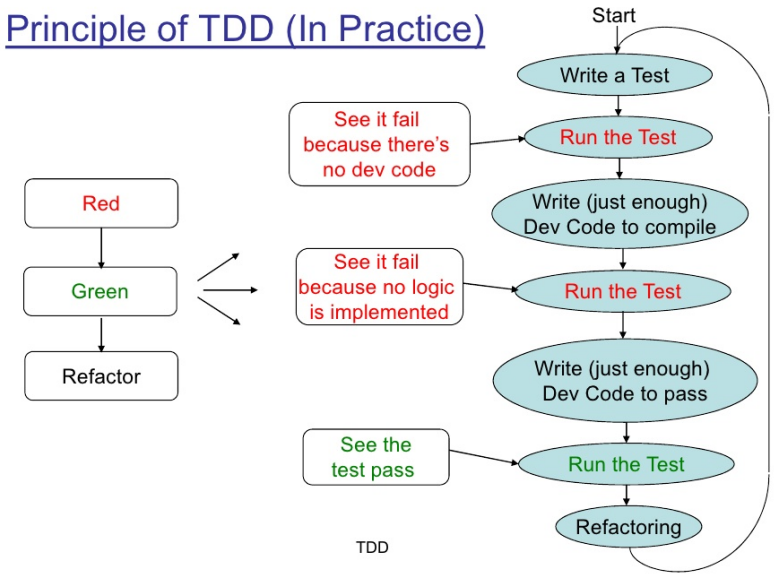
5. Event-driven including time driven
Description:
Program flow is determined mainly by events, such as mouse clicks or interrupts including timer
Main Characteristics:
Main loop, event handlers, asynchronous processes
Examples: Javascript, ActionScript, Visual Basic
6. Object-oriented
Description:
Treats datafields as objects manipulated through pre-defined methods only
Main Characteristics:
Objects, methods, message passing, information hiding, data abstraction, encapsulation, polymorphism, inheritance, serialization-marshalling
Examples: Common Lisp, C++, C, Eiffel, Java, PHP, Python, Ruby, Scala
7. Declarative
Description:
Defines computation logic without defining its detailed control flow
Main Characteristics:
4GLs, spreadsheets, report program generators
Examples: SQL, regular expressions, CSS, Prolog
8. Automata-based programming
Description:
Treats programs as a model of a finite state machine or any other formal automata
Main Characteristics:
State enumeration, control variable, state changes, isomorphism, state transition table
Examples: AsmL

2.13 Testing

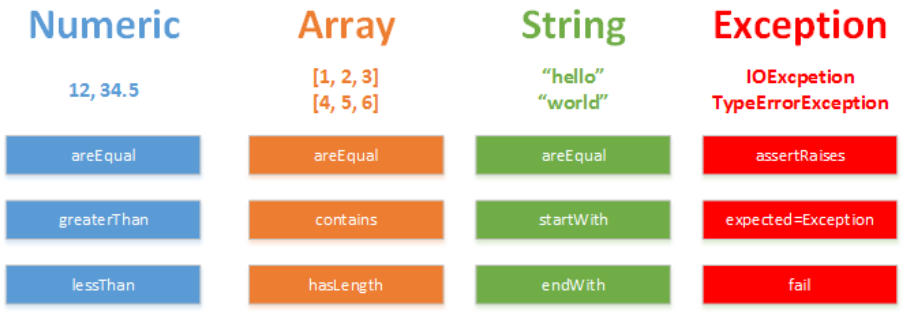


1. Definition 1 2

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle:

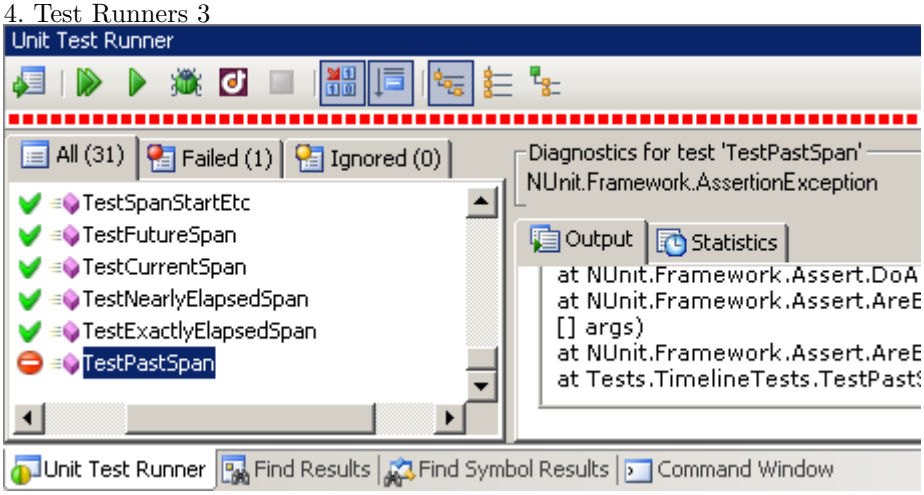


Step 1: First the developer writes an (initially failing) automated test case that defines a desired improvement or new function,
Step 2: Then produces the minimum amount of code to pass that test,
Step 3: Finally refactors the new code to acceptable standards.
Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.
2. Principles 2
Kent Beck defines
Never with a single line of code unless you have a failing automated test. Eliminate duplication Red: (Automated test fail) Green (Automated test pass because dev code has been written) Refactor (Eliminate duplication, Clean the code)
3. Assertions Assert Framework



Assert that the expected results have occurred. [code lang="java"] @Test public void test() assertEquals(2, 1 + 1); [/code]

4. Test Runners 3



When testing a large real-world web app there may be tens or hundreds of test cases, and we certainly don't want to run each one manually. In such as scenario we need to use a test runner to find and execute the tests for us, and in this article we'll explore just that.

A test runner provides the a good basis for a real testing framework. A test runner is designed to run tests, tag tests with attributes (annotations), and provide reporting and other features.

In general you break your tests up into 3 standard sections; setUp(), tests, and tearDown(), typical for a test runner setup.

The setUp() and tearDown() methods are run automatically for every test, and contain respectively:

The setup steps you need to take before running the test, such as unlocking the screen and killing open apps. The cooldown steps you need to run after the test, such as closing the Marionette session.

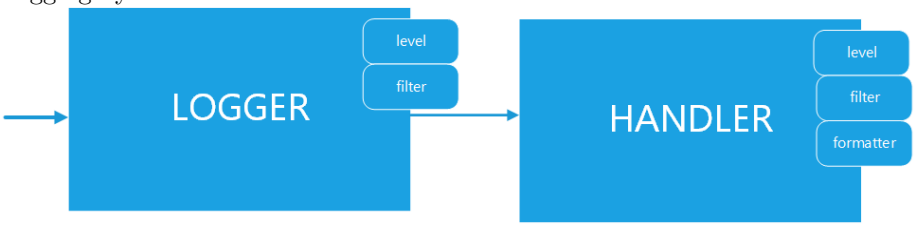
5. Test Frameworks

Language Test Frameworks C++/VisualStudio C++: Test Web Service rest-assured Web UI SeleniumHQ

2.14 Logging

Logging is the process of recording application actions and state to a secondary interface.

Logging System



Levels
Level When it's used DEBUG Detailed information, typically of interest only when diagnosing problems. INFO Confirmation that things are working as expected. WARNING An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
ERROR
Due to a more serious problem, the software has not been able to perform some function. CRITICAL A serious error, indicating that the program itself may be unable to continue running. Best Practices 2 4 5 Logging should always be considered when handling an exception but should never take the place of a real handler. Keep all logging code in your production code. Have an ability to enable more/less detailed logging in production, preferably per subsystem and without restarting your program. Make logs easy to parse by grep and by eye. Stick to several common fields at the beginning of each line. Identify time, severity, and subsystem in every line. Clearly formulate the message. Make every log message easy to map to its source code line. If an error happens, try to collect and log as much information as possible. It may take long but it's OK because normal processing has failed anyway. Not having to wait when the same condition happens in production with a debugger attached is priceless.

2.15 Lập trình hàm

Functional Without mutable variables, assignment, conditional
Advantages 1 Most functional languages provide a nice, protected environment, somewhat like JavaLanguage. It's good to be able to catch exceptions instead of having CoreDumps in stability-critical applications. FP encourages safe ways of programming. I've never seen an OffByOne mistake in a functional program, for example... I've seen one. Adding two lengths to get an index but one of them was zero-indexed. Easy to discover though. – AnonymousDonor Functional programs tend to be much more terse than their ImperativeLanguage counterparts. Often this leads to enhanced programmer productivity. FP encourages quick prototyping. As such, I think it is the best software design paradigm for ExtremeProgrammers... but what do I know. FP is modular in the dimension of functionality, where ObjectOrientedProgramming is modular in the dimension of different components. Generic routines (also provided by CeePlusPlus) with easy syntax. ParametricPolymorphism The ability to have your cake and eat it. Imagine you have a complex OO system processing messages - every component might make state changes depending on the message and then forward the message to some objects it has links to. Wouldn't it be just too cool to be able to easily roll back every change if some object deep in the call hierarchy decided the message is flawed? How about having a history of different states? Many housekeeping tasks made for you: deconstructing data structures (PatternMatching), storing variable bindings (LexicalScope with closures), strong typing (TypeInference), * GarbageCollection, storage allocation, whether to use boxed (pointer-to-value) or unboxed (value directly) representation... Safe multithreading! Immutable data structures are not subject to data race conditions, and consequently don't have to be protected by locks. If you are always allocating new objects, rather than destructively manipulating existing ones, the locking can be hidden in the allocation and GarbageCollection system.

2.16 Lập trình song song

Paralell/Concurrency Programming 1. Callback Pattern 2 Callback functions are derived from a programming paradigm known as functional programming. At a fundamental level, functional programming specifies the use of functions as arguments. Functional programming was—and still is, though to a much lesser extent today—seen as an esoteric technique of specially trained, master programmers.
Fortunately, the techniques of functional programming have been elucidated so that mere mortals like you and me can understand and use them with ease. One of the chief techniques in functional programming happens to be callback functions. As you will read shortly, implementing callback functions is as easy as passing regular variables as arguments. This technique is so simple that I wonder why it is mostly covered in advanced JavaScript topics.
[code lang="javascript"] function getN() return 10;
var n = getN();
function getAsyncN(callback) setTimeout(function() callback(10); , 1000);
function afterGetAsyncN(result) var n = 10; console.log(n);
getAsyncN(afterGetAsyncN); [/code]

2. Promise Pattern 1 3 What is a promise? The core idea behind promises is that a promise represents the result of an asynchronous operation.
A promise is in one of three different states:
pending - The initial state of a promise. fulfilled - The state of a promise representing a successful operation. rejected - The state of a promise representing a failed operation. Once a promise is fulfilled or rejected, it is immutable (i.e. it can never change again).

```
function aPromise(message){
  return new Promise(function( fulfill , reject ){
    if (message == "success"){
       fulfill ("it is a success Promise");
    } if (message == "fail"){
      reject ("it is a fail Promise");
    }
  });
}
```

Usage:

```
aPromise("success").then(function(successMessage){
  console.log(successMessage) }, function(failMessage){
  // it is a success Promise
  console.log(failMessage)
})
```

```
aPromise("fail").then(function(successMessage){
  console.log(successMessage) }, function(failMessage){
  console.log(failMessage)
}) // it is a fail Promise
```

2.17 IDE - Môi trường phát triển tích hợp

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion.
1. Navigation
Word Navigation Line Navigation File Navigation
2. Editing
Auto Complete Code Complete Multicursor Template (Snippets)
3. Formatting
Debugging Custom Rendering for Object

Chương 3

PHP

PHP là ngôn ngữ lập trình web dominate tất cả các anh tài khác mà (chắc là) chỉ dịu đi khi mô hình REST xuất hiện. Nhớ lần đầu gặp bạn Laravel mà cảm giác cuộc đời sang trang.
Cuối tuần này lại phải xem làm sao cài được xdebug vào PHPStorm cho thằng em tập tành lập trình. Haizzz

3.1 Tương tác với cơ sở dữ liệu

Liệt kê danh sách các bản ghi trong bảng groups

```
$sql = "SELECT * FROM `groups`";
$groups = mysqli_query($conn, $sql);
```

Xóa một bản ghi trong bảng groups

```
$sql = "DELETE FROM `groups` WHERE id = `5`";
mysqli_query(\$conn, $sql);
```

3.2 Cài đặt debug trong PHPStorm với Windows

https://www.youtube.com/watch?v=mEJ21RB0F14
(1) XAMPP
- Download XAMPP (cho PHP 7.1.x - do XDebug chưa chính thức hỗ trợ 7.2.0) https://www.apachefriends.org/xampp-files/7.1.12/xampp-win32-7.1.12-0-VC14-installer.exe - Install XAMPP xampp-win32-7.1.12-0-VC14-installer.exe
- Truy cập vào địa chỉ http://localhost/dashboard/phpinfo.php để kiểm tra cài đặt đã thành công chưa
(2) Tải và cài đặt PHPStorm
- Download PHPStorm https://download-cf.jetbrains.com/webide/PhpStorm-2017.3.2.exe - Install PHPStorm
(3) Tạo một web project trong PHPStorm - Chọn interpreter trỏ đến PHP trong xampp
(4) Viết một chương trình add.php

“‘php $a = 2$;b = 3; c =a + b;
echo c; “‘
Click vào ‘add.php‘, chọn Debug, PHPStorm sẽ báo chưa cài XDebug
(5) Cài đặt XDebug theo hướng dẫn tại https://gist.github.com/odan/1abe76d373a9cbb15bed
Click vào add.php, chọn Debug
(6) Cài đặt XDebug với PHPStorm Marklets Vào trang https://www.jetbrains.com/phpstorm/marklets/
Trong phần Zend Debugger - chọn cổng 9000 - IP: 127.0.0.1 Nhấn nút Generate
Bookmark các link quot;Start debuggerquot;, quot;Stop debuggerquot; lên trình duyệt
(7) Debug PHP từ trình duyệt
* Vào trang http://localhost/untitled/add.php * Click vào bookmark Start debugger * Trong PHPStorm, nhấn vào biểu tượng quot;Start Listening for PHP Debug Connectionsquot; * Đặt breakpoint tại dòng thứ 5 * Refresh lại trang http://localhost/untitled/add.php, lúc này, breakpoint sẽ dừng ở dòng 5

3.3 Cấu hình remote debug trong docker container

Kiểm tra xdebug đã chạy chính xác

```
tcpdump -i any port 9000
```

Chú ý: Chẳng hiểu tại sao tcpflow lại không nghe được package. Hic. Mất nguyên buổi sáng.

Phần II

Toán

Chương 4

Xác suất

Xác suất là công cụ để mô hình hóa các sự vật tưởng như ngẫu nhiên.

19/01/2018 Nay tìm được khóa CS 109 của bác Chris Piech quá hay

4.1 Probability Distributions

When we use the word "probability" in day-to-day life, we refer to a degree of confidence that an event of an uncertain nature will occur. For example, the weather report might say "there is a low probability of light rain in the afternoon." Probability theory deals with the formal foundations for discussing such estimates and the rules they should obey. Before we discuss the representation of probability, we need to define what the events are to which we want to assign a probability. These events might be different outcomes of throwing a die, the outcome of a horse race, the weather configurations in California, or the possible failures of a piece of machinery.

4.1.1 Event Spaces

event Formally, we define events by assuming that there is an agreed upon space of possible outcomes, outcome space which we denote by Ω. For example, if we consider dice, we might set Ω = 1, 2, 3, 4, 5, 6. In the case of a horse race, the space might be all possible orders of arrivals at the finish line, a much larger space.
measurable event In addition, we assume that there is a set of measurable events S to which we are willing to assign probabilities. Formally, each event S is a subset of Ω. In our die example, the event 6 represents the case where the die shows 6, and the event 1, 3, 5 represents the case of an odd outcome. In the horse-race example, we might consider the event "Lucky Strike wins," which contains all the outcomes in which the horse Lucky Strike is first. Probability theory requires that the event space satisfy three basic properties: • It contains the empty event , and the trivial event Ω. • It is closed under union. That is, if , S, then so is . • It is closed under complementation. That is, if S, then so is Ω . The requirement that the event space is closed under union and complementation implies that it is also closed under other Boolean operations, such as intersection and set dierence.

4.1.2 Probability Distributions

Definition 2.1 A probability distribution P over (Ω, S) is a mapping from events in S to real values that satisfies probability distribution the following conditions: • P() 0 for all S. • P(Ω) = 1. • If , S and = , then P( ) = P() + P(). The first condition states that probabilities are not negative. The second states that the "trivial event," which allows all possible outcomes, has the maximal possible probability of 1. The third condition states that the probability that one of two mutually disjoint events will occur is the sum of the probabilities of each event. These two conditions imply many other conditions. Of particular interest are P() = 0, and P( ) = P() + P() P( ).

1.3 Interpretations of Probability Before we continue to discuss probability distributions, we need to consider the interpretations that we might assign to them. Intuitively, the probability P() of an event quantifies the degree of confidence that will occur. If P() = 1, we are certain that one of the outcomes in occurs, and if P() = 0, we consider all of them impossible. Other probability values represent options that lie between these two extremes. This description, however, does not provide an answer to what the numbers mean. There are two common interpretations for probabilities. frequentist The frequentist interpretation views probabilities as frequencies of events. More precisely, the interpretation probability of an event is the fraction of times the event occurs if we repeat the experiment indefinitely. For example, suppose we consider the outcome of a particular die roll. In this case, the statement P() = 0.3, for = 1, 3, 5, states that if we repeatedly roll this die and record the outcome, then the fraction of times the outcomes in will occur is 0.3. More precisely, the limit of the sequence of fractions of outcomes in in the first roll, the first two rolls, the first three rolls, . . ., the first n rolls, . . . is 0.3.

The frequentist interpretation gives probabilities a tangible semantics. When we discuss concrete physical systems (for example, dice, coin flips, and card games) we can envision how these frequencies are defined. It is also relatively straightforward to check that frequencies must satisfy the requirements of proper distributions. The frequentist interpretation fails, however, when we consider events such as "It will rain tomorrow afternoon." Although the time span of "Tomorrow afternoon" is somewhat ill defined, we expect it to occur exactly once. It is not clear how we define the frequencies of such events. Several attempts have been made to define the probability for such an event by finding a reference class reference class of similar events for which frequencies are well defined; however, none of them has proved entirely satisfactory. Thus, the frequentist approach does not provide a satisfactory interpretation for a statement such as "the probability of rain tomorrow afternoon is 0.3." An alternative interpretation views probabilities as subjective degrees of belief. Under subjective interpretation this interpretation, the statement P() = 0.3 represents a subjective statement about one's own degree of belief that the event will come about. Thus, the statement "the probability of rain tomorrow afternoon is 50 percent" tells us that in the opinion of the speaker, the chances of rain and no rain tomorrow afternoon are the same. Although tomorrow afternoon will occur only once, we can still have uncertainty about its outcome, and represent it using numbers (that is, probabilities). This description still does not resolve what exactly it means to hold a particular degree of belief. What stops a person from stating that the probability that Bush will win the election is 0.6 and the probability that he will lose is 0.8?

The source of the problem is that we need to explain how subjective degrees of beliefs (something that is internal to each one of us) are reflected in our actions. This issue is a major concern in subjective probabilities. One possible way of attributing degrees of beliefs is by a betting game. Suppose you believe that P() = 0.8. Then you would be willing to place a bet of $1 against $3. To see this, note that with probability 0.8 you gain a dollar, and with probability 0.2 you lose $3, so on average this bet is a good deal with expected gain of 20 cents. In fact, you might be even tempted to place a bet of $1 against $4. Under this bet the average gain is 0, so you should not mind. However, you would not consider it worthwhile to place a bet $1 against
4 and 10 cents, since that would have negative expected gain. Thus, by finding which bets you are willing to place, we can assess your degrees of beliefs. The key point of this mental game is the following. If you hold degrees of belief that do not satisfy the rule of probability, then by a clever construction we can find a series of bets that would result in a sure negative outcome for you. Thus, the argument goes, a rational person must hold degrees of belief that satisfy the rules of probability.1 In the remainder of the book we discuss probabilities, but we usually do not explicitly state their interpretation. Since both interpretations lead to the same mathematical rules, the technical definitions hold for both interpretations.

4.2 Basic Concepts in Probability

4.2.1 Conditional Probability

To use a concrete example, suppose we consider a distribution over a population of students taking a certain course. The space of outcomes is simply the set of all students in the population. Now, suppose that we want to reason about the students' intelligence and their final grade. We can define the event $\alpha to denote \imath all students with grade A, \jmath and the event to denote \imath all students with high intelligence. \jmath Using our distrib$
$P()P()(2.1) That is, the probability that is true given that we know is the relative proportion of outcomes satisfying$
$0.) The conditional probability given an event (say) satisfies the properties of definition 2.1 (see exercise 2.4), and$

4.3 Chain Rule and Bayes Rule

From the definition of the conditional distribution, we immediately see that P( ) = P()P(|). (2.2) chain rule This equality is known as the chain rule of conditional probabilities. More generally, if 1, . . . , k are events, then we can write P(1 . . . k) = P(1)P(2 | 1) · · · P(k | 1 . . . k1). (2.3) In other words, we can express the probability of a combination of several events in terms of the probability of the first, the probability of the second given the first, and so on. It is important to notice that we can expand this expression using any order of events — the result will remain the same. Bayes' rule Another immediate consequence of the definition of conditional probability is Bayes' rule P(|) = P(|)P() P()
A more general conditional version of Bayes' rule, where all our probabilities are conditioned on some background event , also holds: P(|) = P(|)P(|) P(|) . Bayes' rule is important in that it allows us to compute the conditional probability P(|) from the "inverse" conditional probability P(|). Example

2.1 Consider the student population, and let Smart denote smart students and GradeA denote students who got grade A. Assume we believe (perhaps based on estimates from past statistics) that P(GradeA | Smart) = 0.6, and now we learn that a particular student received grade A. Can we estimate the probability that the student is smart? According to Bayes' rule, this depends on prior our prior probability for students being smart (before we learn anything about them) and the prior probability of students receiving high grades. For example, suppose that P(Smart) = 0.3 and P(GradeA) = 0.2, then we have that P(Smart | GradeA) = 0.6 0.3/0.2 = 0.9. That is, an A grade strongly suggests that the student is smart. On the other hand, if the test was easier and high grades were more common, say, P(GradeA) = 0.4 then we would get that P(Smart | GradeA) = 0.6 0.3/0.4 = 0.45, which is much less conclusive about the student. Another classic example that shows the importance of this reasoning is in disease screening. To see this, consider the following hypothetical example (none of the mentioned figures are related to real statistics). Example 2.2 Suppose that a tuberculosis (TB) skin test is 95 percent accurate. That is, if the patient is TB-infected, then the test will be positive with probability 0.95, and if the patient is not infected, then the test will be negative with probability 0.95. Now suppose that a person gets a positive test result. What is the probability that he is infected? Naive reasoning suggests that if the test result is wrong 5 percent of the time, then the probability that the subject is infected is 0.95. That is, 95 percent of subjects with positive results have TB. If we consider the problem by applying Bayes' rule, we see that we need to consider the prior probability of TB infection, and the probability of getting positive test result. Suppose that 1 in 1000 of the subjects who get tested is infected. That is, P(TB) = 0.001. What is the probability of getting a positive test result? From our description, we see that 0.001 · 0.95 infected subjects get a positive result, and 0.999·0.05 uninfected subjects get a positive result. Thus, P(Positive) = 0.0509. Applying Bayes' rule, we get that P(TB | Positive) = 0.001·0.95/0.0509 0.0187. Thus, although a subject with a positive test is much more probable to be TB-infected than is a random subject, fewer than 2 percent of these subjects are TB-infected.

4.4 Random Variables and Joint Distributions

4.4.1 Motivation

Our discussion of probability distributions deals with events. Formally, we can consider any event from the set of measurable events. The description of events is in terms of sets of outcomes. In many cases, however, it would be more natural to consider attributes of the outcome. For example, if we consider a patient, we might consider attributes such as "age,"
"gender," and "smoking history" that are relevant for assigning probability over possible diseases and symptoms. We would like then consider events such as "age $>$ 55, heavy smoking history, and suers from repeated cough." To use a concrete example, consider again a distribution over a population of students in a course. Suppose that we want to reason about the intelligence of students, their final grades, and so forth. We can use an event such as GradeA to denote the subset of students that received the grade A and use it in our formulation. However, this discussion becomes rather cumbersome if we also want to consider students with

grade B, students with grade C, and so on. Instead, we would like to consider a way of directly referring to a student's grade in a clean, mathematical way. The formal machinery for discussing attributes and their values in dierent outcomes are random variable random variables. A random variable is a way of reporting an attribute of the outcome. For example, suppose we have a random variable Grade that reports the final grade of a student, then the statement P (Grade = A) is another notation for P (GradeA).
n the statement P (Grade = A) is another notation for P (GradeA).

4.4.2 What Is a Random Variable?

Formally, a random variable, such as Grade, is defined by a function that associates with each outcome in Ω a value. For example, Grade is defined by a function fGrade that maps each person in Ω to his or her grade (say, one of A, B, or C). The event Grade = A is a shorthand for the event Ω : fGrade() = A. In our example, we might also have a random variable Intelligence that (for simplicity) takes as values either "high" or "low." In this case, the event "Intelligence = high" refers, as can be expected, to the set of smart (high intelligence) students. Random variables can take dierent sets of values. We can think of categorical (or discrete) random variables that take one of a few values, as in our two preceding examples. We can also talk about random variables that can take infinitely many values (for example, integer or real values), such as Height that denotes a student's height. We use Val(X) to denote the set of values that a random variable X can take. In most of the discussion in this book we examine either categorical random variables or random variables that take real values. We will usually use uppercase roman letters X, Y, Z to denote random variables. In discussing generic random variables, we often use a lowercase letter to refer to a value of a random variable. Thus, we use x to refer to a generic value of X. For example, in statements such as "P (X = x) 0 for all x Val(X)." When we discuss categorical random variables, we use the notation x1, . . . , xk, for k = |Val(X)| (the number of elements in Val(X)), when we need to enumerate the specific values of X, for example, in statements such as kX i =1 P (X = xi) = 1. multinomial The distribution over such a variable is called a multinomial. In the case of a binary-valued distribution random variable X, where Val(X) = false, true, we often use x1 to denote the value true for X, and x0 to denote the value false. The distribution of such a random variable is called a Bernoulli Bernoulli distribution. distribution We also use boldface type to denote sets of random variables. Thus, X, Y , or Z are typically used to denote a set of random variables, while x, y, z denote assignments of values to the
variables in these sets. We extend the definition of Val(X) to refer to sets of variables in the obvious way. Thus, x is always a member of Val(X). For Y X, we use xhY i to refer to the assignment within x to the variables in Y . For two assignments x (to X) and y (to Y), we say that x y if they agree on the variables in their intersection, that is, xhX Y i = yhX Y i. In many cases, the notation P(X = x) is redundant, since the fact that x is a value of X is already reported by our choice of letter. Thus, in many texts on probability, the identity of a random variable is not explicitly mentioned, but can be inferred through the notation used for its value. Thus, we use P(x) as a shorthand for P(X = x) when the identity of the random variable is clear from the context. Another shorthand notation is that Px refers to a sum over all possible values that X can

take. Thus, the preceding statement will often appear as Px P(x) = 1. Finally, another standard notation has to do with conjunction. Rather than write P((X = x) (Y = y)), we write P(X = x, Y = y), or just P(x, y).
3.3 Marginal and Joint Distributions Once we define a random variable X, we can consider the distribution over events that can be marginal described using X. This distribution is often referred to as the marginal distribution over the distribution random variable X. We denote this distribution by P(X). Returning to our population example, consider the random variable Intelligence. The marginal distribution over Intelligence assigns probability to specific events such as P(Intelligence = high) and P(Intelligence = low), as well as to the trivial event P(Intelligence high, low). Note that these probabilities are defined by the probability distribution over the original space. For concreteness, suppose that P(Intelligence = high) = 0.3, P(Intelligence = low) = 0.7. If we consider the random variable Grade, we can also define a marginal distribution. This is a distribution over all events that can be described in terms of the Grade variable. In our example, we have that P(Grade = A) = 0.25, P(Grade = B) = 0.37, and P(Grade = C) = 0.38. It should be fairly obvious that the marginal distribution is a probability distribution satisfying the properties of definition 2.1. In fact, the only change is that we restrict our attention to the subsets of S that can be described with the random variable X. In many situations, we are interested in questions that involve the values of several random variables. For example, we might be interested in the event "Intelligence = high and Grade = A." joint distribution To discuss such events, we need to consider the joint distribution over these two random variables. In general, the joint distribution over a set X = X1, . . . , Xn of random variables is denoted by P(X1, . . . , Xn) and is the distribution that assigns probabilities to events that are specified in terms of these random variables. We use to refer to a full assignment to the variables in X , that is, Val(X). The joint distribution of two random variables has to be consistent with the marginal distribution, in that P(x) = P y P(x, y). This relationship is shown in figure 2.1, where we compute the marginal distribution over Grade by summing the probabilities along each row. Similarly, we find the marginal distribution over Intelligence by summing out along each column. The resulting sums are typically written in the row or column margins, whence the term "marginal distribution." Suppose we have a joint distribution over the variables X = X1, . . . , Xn. The most fine-grained events we can discuss using these variables are ones of the form "X1 = x1 and X2 = x2, . . ., and Xn = xn" for a choice of values x1, . . . , xn for all the variables. Moreover,
Intelligence low high A 0.07 0.18 0.25 Grade B 0.28 0.09 0.37 C 0.35 0.03 0.38 0.7 0.3 1 Figure 2.1 Example of a joint distribution P(Intelligence, Grade): Values of Intelligence (columns) and Grade (rows) with the associated marginal distribution on each variable. any two such events must be either identical or disjoint, since they both assign values to all the variables in X . In addition, any event defined using variables in X must be a union of a set of canonical such events. Thus, we are eectively working in a canonical outcome space: a space where each outcome space outcome corresponds to a joint assignment to X1, . . . , Xn. More precisely, all our probability computations remain the same whether we consider the original outcome space (for example, all students), or the canonical space (for example, all combinations of intelligence and grade). atomic outcome We use to denote these atomic outcomes: those assigning a value to each variable in X . For example, if we let X = Intelligence, Grade,

there are six atomic outcomes, shown in figure 2.1. The figure also shows one possible joint distribution over these six outcomes. Based on this discussion, from now on we will not explicitly specify the set of outcomes and measurable events, and instead implicitly assume the canonical outcome space.
3.4 Conditional Probability The notion of conditional probability extends to induced distributions over random variables. For conditional example, we use the notation P (Intelligence | Grade = A) to denote the conditional distribution distribution over the events describable by Intelligence given the knowledge that the student's grade is A. Note that the conditional distribution over a random variable given an observation of the value of another one is not the same as the marginal distribution. In our example, P (Intelligence = high) = 0.3, and P (Intelligence = high | Grade = A) = 0.18/0.25 = 0.72. Thus, clearly P (Intelligence | Grade = A) is dierent from the marginal distribution P (Intelligence). The latter distribution represents our prior knowledge about students before learning anything else about a particular student, while the conditional distribution represents our more informed distribution after learning her grade. We will often use the notation P (X | Y) to represent a set of conditional probability distributions. Intuitively, for each value of Y , this object assigns a probability over values of X using the conditional probability. This notation allows us to write the shorthand version of the chain rule: P (X, Y) = P (X)P (Y | X), which can be extended to multiple variables as P (X1, . . . , Xk) = P (X1)P (X2 | X1) · · · P (Xk | X1, . . . , Xk1). (2.5) Similarly, we can state Bayes' rule in terms of conditional probability distributions: P (X | Y) = P (X)P (Y | X) P (Y) . (2.6)

4.5 Independence and Conditional Independence

4.1 Independence As we mentioned, we usually expect P(|)P(|) to be different from P()P(). That is, learning that is true changes our probability over . However, in some situations equality can occur, so that P(|)=P()P(|)=P(). That is, learning that occurs did not change our probability of .
Definition independent events
We say that an event is independent of event in PP, denoted P()P(), if P(|)=P()P(|)=P() or if P()=0P()=0.
We can also provide an alternative definition for the concept of independence:
Proposition 2.1
A distribution PP satisfies ()() if and only if P()=P()P()P()=P()P().
PROOF Consider first the case where P()=0P()=0; here, we also have P()=0P()=0, and so the equivalence immediately holds. When P()0P()0, we can use the chain rule; we write P()=P(|)P()P()=P(|)P(). Since is independent of , we have that P(|)=P()P(|)=P(). Thus, P()=P()P()P()=P()P(). Conversely, suppose that P()=P()P()P()=P()P(). Then, by definition, we have that P(|)=P()P()=P()P()P()=P(). P(|)=P()P()=P()P()P()=P(). As an immediate consequence of this alternative definition, we see that independence is a symmetric notion. That is, ( ) implies ( ). Example 2.3 For example, suppose that we toss two coins, and let be the event "the first toss results in a head" and the event "the second toss results in a head." It is not hard to convince ourselves that we expect that these two events to be independent. Learning that is true would not change our probability of . In this case, we see two dierent physical

processes (that is, coin tosses) leading to the events, which makes it intuitive that the probabilities of the two are independent. In certain cases, the same process can lead to independent events. For example, consider the event denoting "the die outcome is even" and the event denoting "the die outcome is 1 or 2." It is easy to check that if the die is fair (each of the six possible outcomes has probability 1 6), then these two events are independent.

4.2 Conditional Independence While independence is a useful property, it is not often that we encounter two indepen- dent events. A more common situation is when two events are independent given an additional event. For example, suppose we want to reason about the chance that our student is accepted to graduate studies at Stanford or MIT. Denote by Stanford the event "admitted to Stanford" and by MIT the event "admitted to MIT." In most reasonable distributions, these two events are not independent. If we learn that a student was admitted to Stanford, then our estimate of her probability of being accepted at MIT is now higher, since it is a sign that she is a promising student

Now, suppose that both universities base their decisions only on the student's grade point average (GPA), and we know that our student has a GPA of A. In this case, we might argue that learning that the student was admitted to Stanford should not change the probability that she will be admitted to MIT: Her GPA already tells us the information relevant to her chances of admission to MIT, and finding out about her admission to Stanford does not change that. Formally, the statement is P(MIT | Stanford, GradeA) = P(MIT | GradeA). In this case, we say that MIT is conditionally independent of Stanford given GradeA. Definition 2.3 We say that an event is conditionally independent of event given event in P, denoted conditional independence P |= (|), if P(|) = P(|) or if P( ) = 0. It is easy to extend the arguments we have seen in the case of (unconditional) independencies to give an alternative definition. Proposition 2.2 P satisfies (|) if and only if P(|) = P(|)P(|).

4.3 Independence of Random Variables Until now, we have focused on independence between events. Thus, we can say that two events, such as one toss landing heads and a second also landing heads, are independent. However, we would like to say that any pair of outcomes of the coin tosses is independent. To capture such statements, we can examine the generalization of independence to sets of random variables. Definition 2.4 Let X, Y , Z be sets of random variables. We say that X is conditionally independent of Y given conditional independence Z in a distribution P if P satisfies (X = x Y = y | Z = z) for all values x Val(X), y Val(Y), and z Val(Z). The variables in the set Z are often said to be observed. If the set observed variable Z is empty, then instead of writing (X Y |), we write (X Y) and say that X and Y are marginally independent. marginal independence Thus, an independence statement over random variables is a universal quantification over all possible values of the random variables. The alternative characterization of conditional independence follows immediately: Proposition 2.3 The distribution P satisfies (X Y | Z) if and only if P(X, Y | Z) = P(X | Z)P(Y | Z). Suppose we learn about a conditional independence. Can we conclude other independence properties that must hold in the distribution? We have already seen one such example: symmetry • Symmetry: (X Y | Z) = (Y X | Z). (2.7) There are several other properties that hold for conditional independence, and that often provide a very clean method for proving important properties about distributions. Some key properties are:

• Decomposition: (X Y , W | Z) = (X Y | Z). (2.8) weak union • Weak union:

(X Y , W | Z) = (X Y | Z, W). (2.9) contraction • Contraction: (X W | Z, Y)(X Y | Z) = (X Y , W | Z). (2.10) An additional important property does not hold in general, but it does hold in an important subclass of distributions. Definition 2.5 A distribution P is said to be positive if for all events S such that 6= , we have that positive distribution P() > 0. For positive distributions, we also have the following property: intersection • Intersection: For positive distributions, and for mutually disjoint sets X, Y , Z, W : (X Y | Z, W)(X W | Z, Y) = (X Y , W | Z). (2.11) The proof of these properties is not dicult. For example, to prove Decomposition, assume that (X Y, W | Z) holds. Then, from the definition of conditional independence, we have that P(X, Y, W | Z) = P(X | Z)P(Y, W | Z). Now, using basic rules of probability and arithmetic, we can show P(X, Y | Z) = X w P(X, Y, w | Z) = X w P(X | Z)P(Y, w | Z) = P(X | Z) X w P(Y, w | Z) = P(X | Z)P(Y | Z). The only property we used here is called "reasoning by cases" (see exercise 2.6). We conclude that (X Y | Z).

4.6 Querying a Distribution

Our focus throughout this book is on using a joint probability distribution over multiple random variables to answer queries of interest.

4.6.1 5.1 Probability Queries

probability query Perhaps the most common query type is the probability query. Such a query consists of two parts: evidence • The evidence: a subset E of random variables in the model, and an instantiation e to these variables; query variables • the query variables: a subset Y of random variables in the network. Our task is to compute P(Y | E = e), posterior that is, the posterior probability distribution over the values y of Y , conditioned on the fact that distribution E = e. This expression can also be viewed as the marginal over Y , in the distribution we obtain by conditioning on e.

4.6.2 5.2 MAP Queries

A second important type of task is that of finding a high-probability joint assignment to some subset of variables. The simplest variant of this type of task is the MAP query (also called MAP assignment most probable explanation (MPE)), whose aim is to find the MAP assignment — the most likely assignment to all of the (non-evidence) variables. More precisely, if we let W = X E, our task is to find the most likely assignment to the variables in W given the evidence E = e: MAP(W | e) = argmax w P(w, e), (2.12) where, in general, argmaxx f(x) represents the value of x for which f(x) is maximal. Note that there might be more than one assignment that has the highest posterior probability. In this case, we can either decide that the MAP task is to return the set of possible assignments, or to return an arbitrary member of that set. It is important to understand the dierence between MAP queries and probability queries. In a MAP query, we are finding the most likely joint assignment to W . To find the most likely assignment to a single variable A, we could simply compute P(A | e) and then pick the most likely value. However, the assignment where each variable individually picks its most likely value can be quite dierent from the

most likely joint assignment to all variables simultaneously. This phenomenon can occur even in the simplest case, where we have no evidence. Example 2.4 Consider a two node chain A → B where A and B are both binary-valued. Assume that: a0 a1 0.4 0.6 A b0 b1 a0 0.1 0.9 a1 0.5 0.5 (2.13) We can see that P(a1) > P(a0), so that MAP(A) = a1. However, MAP(A, B) = (a0, b1): Both values of B have the same probability given a1. Thus, the most likely assignment containing a1 has probability 0.6 × 0.5 = 0.3. On the other hand, the distribution over values of B is more skewed given a0, and the most likely assignment (a0, b1) has the probability 0.4 × 0.9 = 0.36. Thus, we have that argmaxa,b P(a, b) 6= (argmaxa P(a),argmaxb P(b)).

4.6.3 5.3 Marginal MAP Queries

To motivate our second query type, let us return to the phenomenon demonstrated in example 2.4. Now, consider a medical diagnosis problem, where the most likely disease has multiple possible symptoms, each of which occurs with some probability, but not an overwhelming probability. On the other hand, a somewhat rarer disease might have only a few symptoms, each of which is very likely given the disease. As in our simple example, the MAP assignment to the data and the symptoms might be higher for the second disease than for the first one. The solution here is to look for the most likely assignment to the disease variable(s) only, rather than the most likely assignment to both the disease and symptom variables. This approach suggests marginal MAP the use of a more general query type. In the marginal MAP query, we have a subset of variables Y that forms our query. The task is to find the most likely assignment to the variables in Y given the evidence E = e: MAP(Y | e) = arg max y P(y | e). If we let Z = X Y E, the marginal MAP task is to compute: MAP(Y | e) = arg max Y X Z P(Y , Z | e). Thus, marginal MAP queries contain both summations and maximizations; in a way, it contains elements of both a conditional probability query and a MAP query. Note that example 2.4 shows that marginal MAP assignments are not monotonic: the most likely assignment MAP(Y1 | e) might be completely dierent from the assignment to Y1 in MAP(Y1, Y2 | e). Thus, in particular, we cannot use a MAP query to give us the correct answer to a marginal MAP query.

4.7 Continuous Spaces

In the previous section, we focused on random variables that have a finite set of possible values. In many situations, we also want to reason about continuous quantities such as weight, height, duration, or cost that take real numbers in IR. When dealing with probabilities over continuous random variables, we have to deal with some technical issues. For example, suppose that we want to reason about a random variable X that can take values in the range between 0 and 1. That is, Val(X) is the interval [0, 1]. Moreover, assume that we want to assign each number in this range equal probability. What would be the probability of a number x? Clearly, since each x has the same probability, and there are infinite number of values, we must have that P(X = x) = 0. This problem appears even if we do not require uniform probability.

6.1 Probability Density Functions How do we define probability over a continuous random variable? We say that a function density function p : IR 7→ IR is a probability density function or (PDF) for X if it is a nonnegative integrable function such that Z Val(X) p(x)dx = 1. That is, the integral over the set of possible values of X is 1. The PDF defines a distribution for X as follows: for any x in our event space: P(X a) = aZ p(x)dx. cumulative The function P is the cumulative distribution for X. We can easily employ the rules of distribution probability to see that by using the density function we can evaluate the probability of other events. For example, P(a X b) = bZa p(x)dx. Intuitively, the value of a PDF p(x) at a point x is the incremental amount that x adds to the cumulative distribution in the integration process. The higher the value of p at and around x, the more mass is added to the cumulative distribution as it passes x. The simplest PDF is the uniform distribution. Definition 2.6 A variable X has a uniform distribution over [a, b], denoted X Unif[a,b] if it has the PDF uniform distribution p(x) = 0 otherwise b 1a b x a. Thus, the probability of any subinterval of [a, b] is proportional its size relative to the size of [a, b]. Note that, if b a $<$ 1, then the density can be greater than 1. Although this looks unintuitive, this situation can occur even in a legal PDF, if the interval over which the value is greater than 1 is not too large. We have only to satisfy the constraint that the total area under the PDF is 1. As a more complex example, consider the Gaussian distribution. Definition 2.7 A random variable X has a Gaussian distribution with mean µ and variance 2, denoted X Gaussian distribution N µ; 2, if it has the PDF p(x) = 21 e (x2 µ 2)2 . standard A standard Gaussian is one with mean 0 and variance 1. Gaussian A Gaussian distribution has a bell-like curve, where the mean parameter µ controls the location of the peak, that is, the value for which the Gaussian gets its maximum value. The variance parameter 2 determines how peaked the Gaussian is: the smaller the variance, the
more peaked the Gaussian. Figure 2.2 shows the probability density function of a few dierent Gaussian distributions. More technically, the probability density function is specified as an exponential, where the expression in the exponent corresponds to the square of the number of standard deviations that x is away from the mean µ. The probability of x decreases exponentially with the square of its deviation from the mean, as measured in units of its standard deviation.
6.2 Joint Density Functions
The discussion of density functions for a single variable naturally extends for joint distributions of continuous random variables. Definition 2.8 Let P be a joint distribution over continuous random variables X1, . . . , Xn. A function p(x1, . . . , xn) joint density is a joint density function of X1, . . . , Xn if • p(x1, . . . , xn) 0 for all values x1, . . . , xn of X1, . . . , Xn. • p is an integrable function. • For any choice of a1, . . . , an, and b1, . . . , bn, P(a1 X1 b1, . . . , an Xn bn) = b1 Za1 · · · b nZ a n p(x1, . . . , xn)dx1 . . . dxn. Thus, a joint density specifies the probability of any joint event over the variables of interest. Both the uniform distribution and the Gaussian distribution have natural extensions to the multivariate case. The definition of a multivariate uniform distribution is straightforward. We defer the definition of the multivariate Gaussian to section 7.1. From the joint density we can derive the marginal density of any random variable by integrating out the other variables. Thus, for example, if p(x, y) is the joint density of X and Y
then p(x) = Z p(x, y)dy. To see why this equality holds, note that the event

a X b is, by definition, equal to the event “a X b and Y .” This rule is the direct analogue of marginalization for discrete variables. Note that, as with discrete probability distributions, we abuse notation a bit and use p to denote both the joint density of X and Y and the marginal density of X. In cases where the distinction is not clear, we use subscripts, so that pX will be the marginal density, of X, and pX,Y the joint density.
6.3 Conditional Density Functions As with discrete random variables, we want to be able to describe conditional distributions of continuous variables. Suppose, for example, we want to define P(Y | X = x). Applying the definition of conditional distribution (equation (2.1)), we run into a problem, since P(X = x) = 0. Thus, the ratio of P(Y, X = x) and P(X = x) is undefined. To avoid this problem, we might consider conditioning on the event x X x + , which can have a positive probability. Now, the conditional probability is well defined. Thus, we might consider the limit of this quantity when → 0. We define P(Y | x) = lim →0 P(Y | x X x +). When does this limit exist? If there is a continuous joint density function p(x, y), then we can derive the form for this term. To do so, consider some event on Y , say a Y b. Recall that P(a Y b | x X x +) = P(a Y b, x X x +) P(x X x +) = Ra b Rx x + p(x0, y)dydx0 Rx x + p(x0)dx0 . When is suciently small, we can approximate x+ Z x p(x0)dx0 2p(x). Using a similar approximation for p(x0, y), we get P(a Y b | x X x +) Ra b 2p(x, y)dy 2p(x) = bZa p(x, y) p(x) dy. We conclude that p(x,y) p(x) is the density of P(Y | X = x).
Let p(x, y) be the joint density of X and Y . The conditional density function of Y given X is conditional density function defined as p(y | x) = p(x, y) p(x) When p(x) = 0, the conditional density is undefined. The conditional density p(y | x) characterizes the conditional distribution P(Y | X = x) we defined earlier. The properties of joint distributions and conditional distributions carry over to joint and conditional density functions. In particular, we have the chain rule p(x, y) = p(x)p(y | x) (2.14) and Bayes' rule p(x | y) = p(x)p(y | x) p(y) . (2.15) As a general statement, whenever we discuss joint distributions of continuous random variables, we discuss properties with respect to the joint density function instead of the joint distribution, as we do in the case of discrete variables. Of particular interest is the notion of (conditional) independence of continuous random variables. Definition 2.10 Let X, Y , and Z be sets of continuous random variables with joint density p(X, Y , Z). We say conditional that X is conditionally independent of Y given Z if independence p(x | z) = p(x | y, z) for all x, y, z such that p(z) > 0.

4.8 Kì vọng và phương sai

4.8.1 Kì vọng

kì vọng

Cho X là một biến rời rạc nhận các giá trị số, khi đó kì vọng của X dưới phân phối P là

$$\mathbb{E}_P[X] = \sum_x x \cdot P(x).$$

Nếu X là một biến liên tục, khi đó sử dụng hàm mật độ

$$\mathbb{E}_P[X] = \int x \cdot p(x).$$

Ví dụ, nếu X là số chấm hiện ra khi tung một xúc sắc với xác suất xuất hiện mỗi mặt là $\frac{1}{6}$. Khi đó, kì vọng của $\mathbb{E}[X] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6} = 3.5$. Nếu trong trường hợp xúc sắc không cân bằng $P(X = 6) = 0.5$ và $P(X = x) = 0.1$) với $x < 6$, khi đó, $\mathbb{E}[X] = 1 \cdot 0.1 + 2 \cdot 0.1 + \cdots + 5 \cdot 0.1 + 6 \cdots 0.5 = 4.5$.

Often we are interested in expectations of a function of a random variable (or several random variables). Thus, we might consider extending the definition to consider the expectation of a functional term such as X2 + 0.5X. Note, however, that any function g of a set of random variables X1, . . . , Xk is essentially defining a new random variable Y : For any outcome Ω, we define the value of Y as g(fX1(), . . . , fXk()). Based on this discussion, we often define new random variables by a functional term. For example Y = X2, or Y = eX. We can also consider functions that map values of one or more categorical random variables to numerical values. One such function that we use quite often is indicator function the indicator function, which we denote 11X = x. This function takes value 1 when X = x, and 0 otherwise. In addition, we often consider expectations of functions of random variables without bothering to name the random variables they define. For example IEP [X + Y]. Nonetheless, we should keep in mind that such a term does refer to an expectation of a random variable. We now turn to examine properties of the expectation of a random variable. First, as can be easily seen, the expectation of a random variable is a linear function in that random variable. Thus, IE[a · X + b] = aIE[X] + b. A more complex situation is when we consider the expectation of a function of several random variables that have some joint behavior. An important property of expectation is that the expectation of a sum of two random variables is the sum of the expectations. Proposition 2.4 IE[X + Y] = IE[X] + IE[Y]. linearity of This property is called linearity of expectation. It is important to stress that this identity is true expectation even when the variables are not independent. As we will see, this property is key in simplifying many seemingly complex problems. Finally, what can we say about the expectation of a product of two random variables? In general, very little: Example 2.5 Consider two random variables X and Y , each of which takes the value +1 with probability 1/2, and the value 1 with probability 1/2. If X and Y are independent, then IE[X · Y] = 0. On the other hand, if X and Y are correlated in that they always take the same value, then IE[X · Y] = 1. However, when X and Y are independent, then, as in our example, we can compute the expectation simply as a product of their individual expectations: Proposition 2.5 If X and Y are independent, then IE[X · Y] = IE[X] · IE[Y]. conditional We often also use the expectation given some evidence. The conditional expectation of X expectation given y is IEP [X | y] = X x x · P(x | y).

4.8.2 Phương sai

phương sai

Kì vọng của X chỉ giá trị trung bình của X. Tuy nhiên, nó không chỉ sự khác nhau giữa các giá trị mà X có thể nhận.

$$\mathbb{V}ar_P[X] = \mathbb{E}_P\left[(X - \mathbb{E}_P[X])^2\right].$$

A measure of this deviation is the variance of X. VVarP [X] = IEP h(X IEP [X])2i. Thus, the variance is the expectation of the squared dierence between X and its expected value. It gives us an indication of the spread of values of X around the expected value. An alternative formulation of the variance is VVar[X] = IEX2 (IE[X])2 . (2.16) (see exercise 2.11). Similar to the expectation, we can consider the expectation of a functions of random variables. Proposition 2.6 If X and Y are independent, then VVar[X + Y] = VVar[X] + VVar[Y]. It is straightforward to show that the variance scales as a quadratic function of X. In particular, we have: VVar[a · X + b] = a2VVar[X]. For this reason, we are often interested in the square root of the variance, which is called the standard standard deviation of the random variable. We define deviation X = pVVar[X]. The intuition is that it is improbable to encounter values of X that are farther than several standard deviations from the expected value of X. Thus, X is a normalized measure of "distance" from the expected value of X. As an example consider the Gaussian distribution of definition 2.7. Proposition 2.7 Let X be a random variable with Gaussian distribution N(µ, 2), then IE[X] = µ and VVar[X] = 2. Thus, the parameters of the Gaussian distribution specify the expectation and the variance of the distribution. As we can see from the form of the distribution, the density of values of X drops exponentially fast in the distance xµ . Not all distributions show such a rapid decline in the probability of outcomes that are distant from the expectation. However, even for arbitrary distributions, one can show that there is a decline. Theorem 2.1 (Chebyshev inequality): Chebyshev's inequality P (|X IEP [X]| t) VVarP [X] t2 .
We can restate this inequality in terms of standard deviations: We write t = kX to get P(|X IEP [X]| kX) 1 k2. Thus, for example, the probability of X being more than two standard deviations away from IE[X] is less than 1/4.

4.9 Các hàm phân phối thông dụng

Phần này có thêm khảo Goodfellow u.a. (2016) và giáo trình xác suất thống kê của thạc sỹ Trần Thiện Khải, đại học Trà Vinh [1]

17/01/2018 Lòng vòng thế nào hôm nay lại tìm được blog của bạn Đỗ Minh Hải, rất hay

4.9.1 Biến rời rạc

Tổng kết các phân phối rời rạc tham khảo slide 3, chapter 9 của khóa CS 109 [2]
Phân phối Bernoulli:

- xác suất xuất hiện mặt ngửa $X \sim Ber(p)$

Phân phối Bernoulli:

- xác suất xuất hiện mặt ngửa $X \sim Ber(p)$

Phân phối Binomial:

- xác suất thành công n lần $X \sim B(n, p)$

Phân phối Poisson:

[1] http://www.ctec.tvu.edu.vn/ttkhai/xacsuatthongke_dh.htm
[2] Slide 3, chapter 9, CS109 Stanford

- xác suất thành công n lần $X \sim Poi(\lambda)$

Phân phối Geometric:

- số lần thử đến khi thành công $X \sim Geo(p)$

Phân phối Negative Binomial:

- số lần thử đến khi r thành công $X \sim NegBin(r, p)$

Phân phối Hyper Geometric:

- số bóng trắng lấy được trong N bóng chứa m bóng trắng $X \sim HypG(n, N, m)$

4.9.2 Biến ngẫu nhiên Bernoulli

Phép thử Bernoulli là một phép thử chỉ có hai khả năng xảy ra "thành công" hoặc "thất bại". Trong thực tế, các ví dụ của phép thử Bernoulli xuất hiện rất phổ biến như *tung đồng xu*, *sự kiện một ổ đĩa bị hỏng*, *sự kiện một ai đó thích xem một bộ phim trên Netflix*. iên Định nghĩa X là một biến ngẫu nhiên, nhận giá trị bằng 1 nếu sự kiên thành công, nhận giá trị bằng 0 nếu sự kiện thất bại. Xác suất sự kiện thành công (X nhận giá trị bằng 1) là $P(X = 1) = p(1) = p$, xác suất sự kiên thất bại $P(X = 0) = p(0) = 1 - p$
X là biến ngẫu nhiên Bernoulli

$$X \sim Ber(p)$$

Khi đó, kì vọng của X

$$E[X] = p$$

Phương sai của X

$$Var(X) = p(1 - p)$$

4.9.3 Phân phối Bernoulli

Như đã đề cập về phép thử Béc-nu-li rằng mọi phép thử của nó chỉ cho 2 kết quả duy nhất là A với xác suất p và $\bar{A}$ với xác suất $q = 1 - p$ Biến ngẫu nhiên X tuân theo phân phối Béc-nu-li

$$X \sim B(p)$$

với tham số $p \in \mathbb{R}, 0 \leq p \leq 1$ là xác suất xuất hiện của A tại mỗi phép thử

Định nghĩa		Giá trị
PMF	p(x)	$p(x) \mid p^x(1-p)^{1-x}, x \in \{0, 1\}$
CDF	$F(x; p)$	$\begin{cases} 0 & \text{for } x < 0 \\ 1 - p & \text{for } 0 \leq x < 1 \\ 1 & \text{for } x \geq 1 \end{cases}$
Kỳ vọng	$E[X]$	p
Phương sai	$Var(X)$	$p(1-p)$

4.9.4 Phân phối Binomial

- xác suất thành công n lần $X \sim B(n, p)$

4.9.5 Phân phối Poisson

- xác suất thành công n lần $X \sim Poi(\lambda)$

4.9.6 Phân phối Geometric

- số lần thử đến khi thành công $X \sim Geo(p)$

4.9.7 Phân phối Negative Binomial

- số lần thử đến khi r thành công $X \sim NegBin(r, p)$

4.9.8 Phân phối Hyper Geometric

- số bóng trắng lấy được trong N bóng chứa m bóng trắng $X \sim HypG(n, N, m)$

4.9.9 Biến liên tục

4.9.10 Phân phối đều

phân phối đều — Là phân phối mà xác suất xuất hiện của các sự kiện là như nhau.

Biến ngẫu nhiên X tuân theo phân phối đều rời rạc

$$X \sim Unif(a, b)$$

với tham số $a, b \in \mathbb{Z}; a < b$ là khoảng giá trị của X, đặt $n = b - a + 1$

Ta sẽ có:

Định nghĩa	Giá trị
PMF	$p(x) \mid \frac{1}{n}, \forall x \in [a, b]$
CDF - $F(x; a, b)$	$\frac{x - a + 1}{n}, \forall x \in [a, b]$
Kỳ vọng - $E[X]$	$\frac{a + b}{2}$
Phương sai - $Var(X)$	$\frac{n^2 - 1}{12}$

Ví dụ 1:

bài toán chờ xe buýt — Lịch chạy của xe buýt tại một trạm xe buýt như sau: chiếc xe buýt đầu tiên trong ngày sẽ khởi hành từ trạm này vào lúc 7 giờ, cứ sau mỗi 15 phút sẽ có một xe khác đến trạm. Giả sử một hành khách đến trạm trong khoảng thời gian từ 7 giờ đến 7 giờ 30. Tìm xác suất để hành khách này chờ:

a) Ít hơn 5 phút.

b) Ít nhất 12 phút.

Giải

Gọi X là số phút sau 7 giờ mà hành khách đến trạm.
Ta có: $X \sim R[0;30]$.
a) Hành khách sẽ chờ ít hơn 5 phút nếu đến trạm giữa 7 giờ 10 và 7 giờ 15 hoặc giữa 7 giờ 25 và 7 giờ 30. Do đó xác suất cần tìm là:

$$P(0 < X < 15) + P(25 < X < 30) = \frac{5}{30} + \frac{5}{30} = \frac{1}{3}$$

b) Hành khách chờ ít nhất 12 phút nếu đến trạm giữa 7 giờ và 7 giờ 3 phút hoặc giữa 7 giờ 15 phút và 7 giờ 18 phút. Xác suất cần tìm là:

$$P(0 < X < 3) + P(15 < X < 18) = \frac{3}{30} + \frac{3}{30} = \frac{1}{5}$$

Chương 5

Thống kê

Theo Nguyễn Tiến Dũng (2015), **thống kê toán học** có thể coi là tổng thể các phương pháp toán học, dựa trên lý thuyết xác suất và các công cụ khác, nhằm đưa ra được những thông tin mới, kết luận mới, có giá trị, từ những bảng số liệu thô ban đầu, và nhằm giải quyết những vấn đề nào đó nảy sinh từ thực tế.

5.1 Các vấn đề thống kê

Một số mục đích chính của thống kê

- Mô tả số liệu.
- Ước lượng và dự đoán các đại lượng.
- Tìm ra các mối quan hệ giữa các đại lượng.
- Kiểm định các giả thuyết.

5.2 Ước lượng bằng thống kê

5.2.1 Phương pháp hợp lý cực đại

phương pháp hợp lý cực đại

Theo Nguyễn Tiến Dũng (2015), một trong những phương pháp phổ biến nhất để ước lượng phân bố xác suất của X bằng một phân bố xác suất trong một họ nào đó gọi là **phương pháp hợp lý cực đại** (maximum likelyhood - dễ xảy ra nhất)

Ý tưởng của phương pháp này: những gì mà thấy được trong thực nghiệm, thì phải dễ xảy ra hơn là những gì không thấy. Ví dụ, khi một giáo viên hỏi một học sinh 4 câu hỏi ngẫu nhiên về một môn học nào đó mà học sinh đều trả lời được, thì giáo viên sẽ "ước lượng" đấy là một học sinh giỏi, vì khi giỏi thì mới nhiều khả năng trả lời được cả 4 câu hỏi, còn nếu không giỏi sẽ có nhiều khả năng không trả lời được ít nhất 1 trong 4 câu hơn là khả năng "ăn may" trả lời" được cả 4 câu.
Chúng ta sẽ tìm phân phối xác suất của biến ngẫu nhiên X sao cho mẫu thực nghiệm $(x_1, ..., x_n)$ có nhiều khả năng xảy ra nhất.

$$h_{MLE} = \underset{h \in H}{argmax} P(h|D)$$

Ví dụ. Giả sử ta muốn tìm thấy xác suất của một sự kiện A nào đó (ví dụ như sự kiên: say rượu khi lái xe). Gọi X là *hàm chỉ báo* của A: $X = 0$ nếu A không xảy ra, $X = 1$ nếu A xảy ra. Khi đó X có phân bố Bernoulli với tham số $p = P(A)$. Để ước lượng p, ta làm n phép thử ngẫu nhiên đọc lập, và được một mẫu $x_1, ..., x_n$ của X. Các số $x_1, ..., x_n$ chỉ nhận hai giá trị 0 và 1. Gọi k là số số 1 trong dãy số $x_1, ..., x_n$. Khi đó, hàm độ hợp lý là:

$$L(p) = p^k(1-p)^{n-k}$$

Đạo hàm của $L(p)$ là $L'(p)$

$$L'(p) = n\left(\frac{k}{n} - p\right)p^{k-1}(1-p)^{n-k-1}$$

Từ đó, hàm $L(p)$ đạt cực đại trên khoảng $[0, 1]$ tại điểm $p = \frac{k}{n} = \sum_{i=1}^{n} \frac{x_i}{n}$
Như vậy, theo phương pháp hợp lý cực đại, ta có ước lượng sau đây của xác suất $p = p(A)$

$$\hat{p} = \sum_{i=1}^{n} \frac{x_i}{n}$$

5.2.2 Phương pháp moment

5.2.3 Phương pháp hậu nghiệm cực đại

phương pháp hậu nghiệm cực đại

Với một tập các giải thiết có thể H, hệ thống học sẽ tìm **giả thiết có thể xảy ra nhất** h ($h \in H$) đối với dữ liệu quan sát được D.

Giả thiết này được gọi là giả thiết có xác suất hậu nghiệm cực đại (**maximum a posteriori - MAP**)

$$h_{MAP} = \underset{h \in H}{argmax} P(h|D)$$

Do định lý Bayes,

$$h_{MAP} = \underset{h \in H}{argmax} \frac{P(D|h)P(h)}{P(D)}$$

Do $P(D)$ không phụ thuộc vào h, nên

$$h_{MAP} = \underset{h \in H}{argmax} P(D|h)P(h) \tag{5.1}$$

Ví dụ. Dữ liệu **chơi tennis**. Dữ liệu được giới thiệu trong quyển Mitchell (1997) chứa thông tin thời tiết và quyết định chơi/không chơi tennis của một người. Được trình bày trong bảng 5.2.3
Tập H bao gồm 2 giả thiết có thể

- h_1: Anh ta chơi tennis

ngày	ngoài trời	nhiệt độ	độ ẩm	gió	chơi tennis
n1	nắng	nóng	cao	yếu	không
n2	nắng	nóng	cao	mạnh	không
n3	âm u	nóng	cao	yếu	có
n4	mưa	bình thường	cao	yếu	có
n5	mưa	mát mẻ	bình thường	yếu	có
n6	mưa	mát mẻ	bình thường	mạnh	không
n7	âm u	mát mẻ	bình thường	mạnh	có
n8	nắng	bình thường	cao	yếu	không
n9	nắng	mát mẻ	bình thường	yếu	có
n10	mưa	bình thường	bình thường	yếu	có
n11	nắng	bình thường	bình thường	mạnh	có
n12	âm u	bình thường	cao	mạnh	có
n13	âm u	nóng	bình thường	yếu	có
n14	mưa	bình thường	cao	mạnh	không

Bảng 5.1: Dữ liệu chơi tennis

- h_2: Anh ta không chơi tennis

D: Tập dữ liệu (các ngày) mà trong đó trời nắng và gió mạnh (ngoài trời = nắng)
Dựa vào phương pháp MAP, có thể trả lời câu hỏi với thời tiết như vậy, anh ta có chơi tennis hay không?
Nhắc lại công thức 5.1

$$h_{MAP} = \underset{h \in H}{argmax} P(D|h)P(h) \tag{5.2}$$

Do đó, ta cần tính hai giá trị $P(D|h_1)P(h_1)$ và $P(D|h_2)P(h_2)$

- $P(D|h_1)P(h_1) = P(D, h_1)$
  $= P($ ngoài trời=nắng, chơi tennis=có $) = \frac{2}{14}$
- $P(D|h_2)P(h_2) = P(D, h_2)$
  $= P($ ngoài trời=nắng, chơi tennis=không $) = \frac{3}{14}$

Do đó, với phương pháp MAP, có thể đưa ra kết luận với thời tiết như vậy, anh ta sẽ **không** chơi tenins

5.3 unbiased estimation

5.4 Kiểm định các giả thuyết

5.5 Tài liệu tham khảo

22/01/2018 Hôm nay tìm được quyển Nhập môn hiện đại xác suất và thống kê của hai tác giả Đỗ Đức Thái và Nguyễn Tiến Dũng quá vui.

Phần III

Khoa học máy tính

Chương 6

Data Structure and Algorithm

View online http://magizbox.com/training/danda/site/

6.1 Introduction

Algorithms + Data Structures = Programs
In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently. Data structures can implement one or more particular abstract data types (ADT), which specify the operations that can be performed on a data structure and the computional complexity of those operations. In comparison, a data structure is a concrete implementation of the specification provided by an ADT.
In mathematics and computer science, an algorithm is a self-contained step-by-step set of operations to be performed. Algorithms perform calculation, data processing, and/or automated reasoning tasks.
Software engineering is the study of ways in which to create large and complex computer applications and that generally involve many programmers and designers. At the heart of software engineering is with the overall design of the applications and on the creation of a design that is based on the needs and requirements of end users. While software engineering involves the full life cycle of a software project, is includes many different components - specification, requirements gathering, design, verification, coding, testing, quality assurance, user acceptance testing, production, and ongoing maintenance.
Having an in-depth understanding on every component of software engineering is not mandatory, however, it is important to understand that the subject of data structures and algorithms is concerned with the coding phase. The use of data structures and algorithms is the nuts-and-blots used by programmers to store and manipulate data.
This article, along with the other examples in this section focuses on the essentials of data structures and algorithms. Attempts will be made to understand how they work, which structure or algorithm is best in a particular situation in an easy to understand environment.

Data Structures and Algorithms - Defined A data structure is an arrangement of data in a computer's memory or even disk storage. An example of several common data structures are arrays, linked lists, queues, stacks, binary trees, and hash tables. Algorithms, on the other hand, are used to manipulate the data contained in these data structures as in searching and sorting.
Many algorithms apply directly to a specific data structures. When working with certain data structures you need to know how to insert new data, search for a specified item, and deleting a specific item.
Commonly used algorithms include are useful for:
Searching for a particular data item (or record). Sorting the data. There are many ways to sort data. Simple sorting, Advanced sorting Iterating through all the items in a data structure. (Visiting each item in turn so as to display it or perform some other action on these items)

6.1.1 Greedy Algorithm

Greedy Algorithms An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen.
Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.
Counting Coins This problem is to count to a desired value by choosing the least possible coins and the greedy approach forces the algorithm to pick the largest possible coin. If we are provided coins of 1, 2, 5 and 10 and we are asked to count 18 then the greedy procedure will be
Select one 10 coin, the remaining count is 8
Then select one 5 coin, the remaining count is 3
Then select one 2 coin, the remaining count is 1
And finally, the selection of one 1 coins solves the problem
Though, it seems to be working fine, for this count we need to pick only 4 coins. But if we slightly change the problem then the same approach may not be able to produce the same optimum result.
For the currency system, where we have coins of 1, 7, 10 value, counting coins for value 18 will be absolutely optimum but for count like 15, it may use more coins than necessary. For example, the greedy approach will use 10 + 1 + 1 + 1 + 1 + 1, total 6 coins. Whereas the same problem could be solved by using only 3 coins (7 + 7 + 1)
Hence, we may conclude that the greedy approach picks an immediate optimized solution and may fail where global optimization is a major concern.
Examples Most networking algorithms use the greedy approach. Here is a list of few of them
Travelling Salesman Problem Prim's Minimal Spanning Tree Algorithm Kruskal's Minimal Spanning Tree Algorithm Dijkstra's Minimal Spanning Tree Algorithm Graph - Map Coloring Graph - Vertex Cover Knapsack Problem Job Scheduling Problem

6.1.2 Divide and Conquer

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.
Broadly, we can understand divide-and-conquer approach in a three-step process.
Divide/Break This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible. At this stage, sub-problems become atomic in nature but still represent some part of the actual problem.
Conquer/Solve This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.
Merge/Combine When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively and conquer merge steps works so close that they appear as one.
Examples The following computer algorithms are based on divide-and-conquer programming approach
Merge Sort Quick Sort Binary Search Strassen's Matrix Multiplication Closest pair (points) There are various ways available to solve any computer problem, but the mentioned are a good example of divide and conquer approach.

6.1.3 Dynamic Programming

Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.
Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to achieve the best solution.
So we can say that
The problem should be able to be divided into smaller overlapping sub-problem. An optimum solution can be achieved by using an optimum solution of smaller sub-problems. Dynamic algorithms use memorization. Comparison In contrast to greedy algorithms, where local optimization is addressed, dynamic algorithms are motivated for an overall optimization of the problem.
In contrast to divide and conquer algorithms, where solutions are combined to achieve an overall solution, dynamic algorithms use the output of a smaller sub-problem and then try to optimize a bigger sub-problem. Dynamic algorithms use memorization to remember the output of already solved sub-problems.

Example The following computer problems can be solved using dynamic programming approach
Fibonacci number series Knapsack problem Tower of Hanoi All pair shortest path by Floyd-Warshall Shortest path by Dijkstra Project scheduling Dynamic programming can be used in both top-down and bottom-up manner. And of course, most of the times, referring to the previous solution output is cheaper than recomputing in terms of CPU cycles.

6.1.4 7 Steps to Solve Algorithm Problems

Today, I viewed the video "7 Steps to Solve Algorithm Problems" by Gayle Laakmann McDowell - the author of Cracking the Coding Interview book. In this video, Gayle describe her method for solve algorithms problems which consists 7 steps: listen carefully, example, brute force, optimize, walk through your algorithms, code and test. In this article, I will summary these steps base on what I learned from this video.
Step 1: Listen carefully Every single detail in a question is necessary to solve it.
The first step is to listen carefully to the problem. So, generally speaking every single detail in a question is necessary to solve that problem - either to solve it all or to solve it optimally. So if there's some detail you haven't used in the question in your algorithm so far think about how you can put that to use because it might be necessary to solve the problem optimally.
Let me give you an example.
You have two arrays, sorted and distinct How did you find the number of elements in common between the two arrays? A lot of people solve this problem and they'll get kind of stuck for awhile and what they'll do is they'll be solving the problem and they'll know the arrays are sorted but they haven't actually used the fact that it's sorted.
This sorting detail - it's not necessary just to find an algorithm but it is necessary to solve the problem optimally.
So remember every single detail in the problem and make sure you use it.
Step 2: Example Make example big, no special cases
The second piece is to come up with a good example, so the last problem that I gave two arrays sorted and distinct compute the number of elements in common, most people's examples look like this.
too small and special case A: 1, 5, 15, 20 B: 2, 5, 13, 30 Yes technically if it's a problem but it's not very useful.
As soon as you glance at this example you notice that there's only one element common and you know exactly what it is and it's obvious because this example is so small and it's actually kind of a special case.
A better example is something like this
larger and avoid special cases A: 1, 5, 15, 20, 30, 37 B: 2, 5, 13, 30, 32, 35, 37, 42 It's much larger and you've avoided some special cases. One of the easiest ways of improving your performance on algorithm questions is just make your examples larger and really avoid special cases.
Step 3: Brute force Better to have a brute force than nothing at all
The third step is to come up with a brute force algorithm. Now I'm not saying you need to go out of your way to come up with something slow, I'm really just saying, hey if the first thing you have is only something really really slow and terrible that's okay. It is so much better to start off with something slow then

to start off with nothing at all. So it's fine if your first algorithm is slow and terrible whatever. However, and this is very very very important, I'm not saying to code the brute force. I'm saying just state your brute force algorithm, state its runtime, and then immediately go to optimizing.
A good chunk of the time on algorithm interview question will often be spent on optimizations. So that's step 4 and spend some good time on it.
Step 4: Optimize The fourth step is optimize and spend some good time on it.
Step 5: Walk through your algorithms Know exactly what you're going to do before coding
what variables data structures? how, why, why do they change? what is the structure of your code Then once you have an optimal algorithm or you're ready to start coding take a step back and just make sure you know exactly what you're going to do in your code.
So many people code prematurely when they aren't really really comfortable with what they're about to do and it ends in disaster. An eighty percent understanding of what you're about to write is really not enough for a whiteboard especially. So take a moment and walk through your algorithm and make sure you know exactly what you're about to do.
Step 6: Code Use space wisely, coding style matters, modularize
Step 6 is to start coding and I'm gonna go into this in a bit of detail. So a couple things to keep in mind particularly when you're coding on a whiteboard. The first couple tips are kind of whiteboard specific but try to write your lines straight. I'm not gonna be judging you on your handwriting and things like that but when people start writing their lines and sharp angles they start to lose track over whether this if statement under this for loop or not. The second thing is use your board space wisely. If you don't need stuff up on the board anymore just erase it. Try to write in this top left corner etc.
Basically give yourself as much space as you possibly can to write your code. If you do run out of space though, it's ok to use arrows, that's fine, I'm really not gonna be judging you on this kind of stuff. So more important things.
Coding style matters (consistent braces, consistent variable naming, consistence spaces, descriptive variables)
Coding style matters even on a whiteboard but on a computer as well, so that means things like braces, naming conventions, or using camel case or underscores, things like that. Those kind of style things absolutely matter. I'm not that concerned over which style you pick, I don't care if you write braces on the same line or the next line but I do care a lot that you have a style and you stick to it. So be consistent in your style. When it comes to variable names, yeah I know it's an annoying to write long variable names on a whiteboard but descriptive variable names are important to good style. So one compromise here is write the good descriptive variable name first and then just ask your interviewer, hey is it okay if I abbreviate this the next time. So that'll be a nice little compromise - you'd show that you care about good variable names but you also don't waste a lot of time.
Modularize (before. not after)
Last thing I want to talk about is modularization. Modularize your code up front and just any little conceptual chunks of code, push that off to another function. So suppose you have three steps in your algorithm - process the first string, process the second string, and then compare the results. Don't start writing these for loops that walk through each string in the very beginning.

Instead write this overall function that wraps these three steps. So step one, step two, step three, and then start drilling in and going into all the details there. Remember any conceptual chunks of code push those off to other functions, don't write them in line.
Step 7: Test Analyse: think about each line, double check things that look weired/risky (for-loop that decrement, math)
Use test cases (smaller test-cases first (faster to run, you will problably be more through, edge cases, big test cases)
Then once you're done with the coding you have to start testing your code. One of the mistakes a lot of people do here is they take their big example from step 2 and throw that in as a test case. The problem with that is it's very large so it will take you a long time to run through but also you just used that to develop your code, so if here's an oversight there, the problem will probably repeat itself here.
What's a better step to do, what's a better process to do, is just walk through your code line by line and just think about each line up front not with the test case but just consider, is it really doing the right thing?
Double check anything that looks weird, so for loops that decrement instead of increment and any math at all is a really common place for errors. Just think, look at your code analytically and think what are the most likely places for errors to be and double-check those.
Start with small rather than big
Then once you start with actual test cases start with small test cases rather than big ones. Small test cases work pretty much as effectively as big test cases but they are so much faster to run through, and in fact because they're faster people tend to be much more thorough so you're much more likely to actually find bugs with small test cases than big test cases. So start with small test cases then go in to edge cases after that and then if you have time maybe throw in some big test cases. A couple last techniques with testing. The first one is make sure that when you're testing you're really thinking about what you're doing. A lot of people when they're testing they're just walking through their code almost like they're a bot, and they only look to see if things made sense at the very end when they look at their output. It's much better to really think as you're testing, this way you find the bug as soon as it happens rather than six lines later at the very bottom.
Test your code not your algorithm
The second thing is when you're testing make sure that you're actually testing your code and not your algorithm. An amazing number of people will just take their example and like just walk through it again as though they're just walking through their algorithm but they're never even looking at their code, they're not looking at the exact calculations their code actually did. So make sure that you're really testing your code.
Find bugs
Then the last thing is when you find in a bug, don't panic. Just really think about what caused the bug. A lot of times people will panic and just try to make the first fix that fixes it for that output but they haven't really given it some thought and then they're in a much worse position because if you make the wrong fix to your code, the thing that just fixed the output but didn't fix a real bug you've not fixed the actual bug, you've made your code more complex, and you potentially introduced a brand new bug and you're in a much worse

position. It's much better to just when you find the bug, it's ok, it's not that big of a deal to have a bug it's very normal just really think through what the actual bug, where the actual plug came from.
Remember
think as you test (don't be a bot) test your code, not your algorithm think before you fix bugs. Don't panic! (wrong fixes are worse than no fix) Suggested Reading 7 Steps to Solve Algorithm Problems. Gayle Laakmann McDowell

6.2 Data Structures

6.2.1 Array

Arrays An array is an aggregate data structure that is designed to store a group of objects of the same or different types. Arrays can hold primitives as well as references. The array is the most efficient data structure for storing and accessing a sequence of objects.
Here is the list of most important array features you must know (i.e. be able to program)
copying and cloning insertion and deletion searching and sorting You already know that the Java language has only two data types, primitives and references. Which one is an array? Is it primitive? An array is not a primitive data type - it has a field (and only one), called length. Formally speaking, an array is a reference type, though you cannot find such a class in the Java APIs. Therefore, you deal with arrays as you deal with references. One of the major diffeences between refeences and primituives is that you cannot copy arrays by assigning one to another:
int[] a = 9, 5, 4; int[] b = a; The assignment operator creates an alias to the object, like in the picture below
Since these two references a and b refer to the same object, comparing them with the double equal sign "==" will always return true. In the next code example, int [] a = 1,2,3; int [] b = 1,2,3; a and b refer to two different objects (though with identical contents). Comparing them with the double equal sign will return false. How would you compare two objects with identical contents? In short, using the equals method. For array comparison, the Java APIs provides the Arrays class. The Arrays class The java.util.Arrays class is a convenience class for various array manipulations, like comparison, searching, printing, sorting and others. Basically, this class is a set of static methods that are all useful for working with arrays. The code below demonstrates a proper invocation of equals:
int[] a = 1,2,3; int[] b = 1,2,3; if(Arrays.equals(a, b)) System.out.println("arrays with identical contents"); Another commonly used method is toString() which takes care of of printing
int[] a = 1,2,3; System.out.println(Arrays.toString(a)); Here is the example of sorting
int[] a = 3,2,1; Arrays.sort(a); System.out.println(Arrays.toString(a)); In addition to that, the class has other utility methods for supporting operations over multidimensional arrays.
Copying arrays There are four ways to copy arrays
using a loop structure using Arrays.copyOf() using System.arraycopy() using clone() The first way is very well known to you

int[] a = 1, 2, 3; int[] b = new int[a.length]; for(int i= 0; i < a.length; i++) b[i] = a[i]; The next choice is to use Arrays.copyOf()
int[] a = 1, 2, 3; int[] b = Arrays.copyOf(a, a.length); The second parameter specifies the length of the new array, which could either less or equal or bigger than the original length.
The most efficient copying data between arrays is provided by System.arraycopy() method. The method requires five arguments. Here is its signature
public static void arraycopy(Object source, int srcIndex, Object destination, int destIndex, int length) The method copies length elements from a source array starting with the index srcIndex to a new array destination at the index destIndex.The above code example can be rewritten as it follows
int[] a = 1, 2, 3; int[] b = new int[a.length]; System.arraycopy(a, 0, b, 0, 3) And the last copying choice is the use of cloning. Cloning involves creating a new array of the same size and type and copying all the old elements into the new array. The clone() method is defined in the Object class and its invocation is demonstrated by this code segment
int[] a = 1, 2, 3; int[] b = (int[]) a.clone(); Note, that casting (int[]) is the must. Examine the code in ArrayCopyPrimitives.java for further details.
Insertion and Deletion Arrays in Java have no methods and only one immutable field length. Once an array is created, its length is fixed and cannot be changed. What do you do to resize the array? You allocate the array with a different size and copy the contents of the old array to the new array. This code example demonstrates deletion from an array of primitives
public char[] delete(char[] data, int pos) if(pos >= 0 pos < data.length) char[] tmp = new char[data.length-1]; System.arraycopy(data, 0, tmp, 0, pos); System.arraycopy(data, pos+1, tmp, pos, data.length-pos-1); return tmp; else return data; The first arraycopy copies the elements from index 0 to index pos-1, the second arraycopy copies the elements from index pos+1 to data.length. Examine the code in ArrayDemo.java for further details.
The ArrayList class The java.util.ArrayList class supports an idea of a dynamic array - an array that grows and shrinks on demand to accomodate the number of elements in the array. Below is a list of commonly used methods
add(object) - adds to the end add(index, object) - inserts at the index set(index, object) - replaces at the index get(index) - returns the element at that index remove(index) - deletes the element at that index size() - returns the number of elements The following code example will give you a heads up into how some of them are used.
/* ADD */ ArrayList<Integer> num = new ArrayList<Integer>(); for(int i = 0; i < 10; i++) num.add(i); System.out.println(num);
/* REMOVE even integers */ for(int i = 0; i < num.size(); i++) if(num.get(i)System.out.println(num);
Copying arrays of objects This topic is more complex for understanding.. Let us start with a simple loop structure
Object[] obj1 = new Integer(10), new StringBuffer("foobar"), new Double(12.95); Object[] obj2 = new Object[obj1.length]; for(int i = 0; i ‹ obj1.length; i++) obj2[i] = obj1[i]; At the first glance we might think that all data is copied. In reality, the internal data is shared between two arrays. The figure below illustrates the inner structure
The assignment operator obj2[i] = obj1[i] is a crucial part of understanding the concept. You cannot copy references by assigning one to another. The assignment creates an alias rather than a copy. Let us trace down changes in the above

picture after execution the following statements
obj1[0] = new Integer(5);
and ((StringBuffer) obj1[1]).append('s');
As you see, obj1[0] and obj2[0] now refer to different objects. However, obj1[1] and obj2[1] refer to the same object (which is "foobars"). Since both arrays shares the data, you must be quite careful when you modify your data, because it might lead to unexpected effects.
The same behavior will take place again, if we use Arrays.copuyOf(), System.arraycopy() and clone(). Examine the code example ArrayCopyReferences.java for further details.
Multi-dimensional arrays In many practical application there is a need to use two- or multi-dimensional arrays. A two-dimensional array can be thought of as a table of rows and columns. This creates a table of 2 rows and 4 columns:
int[][] ar1 = new int[2][4]; You can create and initialize an array by using nested curcly braces. For example, this creates a table of 3 rows and 2 columns:
int[][] ar2 = 1,2,3,4,5,6; Generally speaking, a two-dimensional array is not exactly a table - each row in such array can have a different length. Consider this code fragment
Object[][] obj = new Integer(1),new Integer(2), new Integer(10), "bozo", new Double(1.95); The accompanying picture sheds a bit of light on internal representation
From the picture you clearly see that a two-dimensional array in Java is an array of arrays. The array obj has two elements obj[0] and obj[1] that are arrays of length 2 and 3 respectively.
Cloning 2D arrays The procedure is even more confusing and less expected. Consider the following code segment
Object[][] obj = new Integer(1),new Integer(2), new Integer(10), "bozo", new Double(1.95);
Object[][] twin = (Object[][]) obj.clone(); The procedure of clonig 2d arrays is virtually the same as cloning an array of references. Unfortunately, built-in clone() method does not actualy clone each row, but rather creates references to them Here is a graphical interpretation of the above code
Let us change the value of obj[1][1]
obj[1][1] = "xyz"; This assignment effects the value of twin[1][1] as well
Such a copy is called a "shallow" copy. The default behavior of clone() is to return a shallow copy of the object. If we want a "deep" copy instead, we must provide our own implementation by overriding Object's clone() method.
The idea of a "deep" copy is simple - it makes a distinct copy of each of the object's fields, recursing through the entire object. A deep copy is thus a completely separate object from the original; changes to it don't affect the original, and vise versa. In relevance to the above code, here is a deep clone graphically
Further, making a complete deep copy is not always needed. Consider an array of immutable objects. As we know, immutable objects cannot be modified, allowing clients to share the same instance without interfering with each other. In this case there is no need to clone them, which leads to the following picture
Always in this course we will create data structures of immutable objets, therefore implementing the clone method will require copying a structure (a shape) and sharing its internal data. We will discuss these issues later on in the course.
Challenges "Arrays: Left Rotation". hackerrank. 2016 References "Array Data Structure". Victor S.Adamchik, CMU. 2009

6.2.2 Linked List

A linked list is a sequence of data structures, which are connected together via links.
Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.
Link Each link of a linked list can store a data called an element. Next Each link of a linked list contains a link to the next link called Next. LinkedList A Linked List contains the connection link to the first link called First. Representation Linked list can be visualized as a chain of nodes, where every node points to the next node.
As per the above illustration, following are the important points to be considered.
Linked List contains a link element called first. Each link carries a data field(s) and a link field called next. Each link is linked with its next link using its next link. Last link carries a link as null to mark the end of the list. Types of Linked List Following are the various types of linked list.
Simple Linked List Item navigation is forward only. Doubly Linked List Items can be navigated forward and backward. Circular Linked List Last item contains link of the first element as next and the first element has a link to the last element as previous. Basic Operations Following are the basic operations supported by a list.
Insertion Adds an element at the beginning of the list. Deletion Deletes an element at the beginning of the list. Display Displays the complete list. Search Searches an element using the given key. Delete Deletes an element using the given key. Insertion Operation Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.
Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode). Then point B.next to C
NewNode.next > RightNode; It should look like this
Now, the next node at the left should point to the new node.
LeftNode.next > NewNode;
This will put the new node in the middle of the two. The new list should look like this
Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.
Deletion Operation Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.
The left (previous) node of the target node now should point to the next node of the target node
LeftNode.next > TargetNode.next;
This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.
TargetNode.next > NULL;

We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.
Reverse Operation This operation is a thorough one. We need to make the last node to be pointed by the head node and reverse the whole linked list.
First, we traverse to the end of the list. It should be pointing to NULL. Now, we shall make it point to its previous node
We have to make sure that the last node is not the lost node. So we'll have some temp node, which looks like the head node pointing to the last node. Now, we shall make all left side nodes point to their previous nodes one by one.
Except the node (first node) pointed by the head node, all nodes should point to their predecessor, making them their new successor. The first node will point to NULL.
We'll make the head node point to the new first node by using the temp node. The linked list is now reversed.

6.2.3 Stack and Queue

An array is a random access data structure, where each element can be accessed directly and in constant time. A typical illustration of random access is a book - each page of the book can be open independently of others. Random access is critical to many algorithms, for example binary search.
A linked list is a sequential access data structure, where each element can be accesed only in particular order. A typical illustration of sequential access is a roll of paper or tape - all prior material must be unrolled in order to get to data you want.
In this note we consider a subcase of sequential data structures, so-called limited access data sturctures.
Stacks A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top. A stack is a recursive data structure. Here is a structural definition of a Stack:
a stack is either empty or it consistes of a top and the rest which is a stack;
Applications The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack. Another application is an "undo" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack. Backtracking. This is a process when you need to access the most recent data element in a series of elements. Think of a labyrinth or maze - how do you find a way from an entrance to an exit? Once you reach a dead end, you must backtrack. But backtrack to where? to the previous choice point. Therefore, at each choice point you store on a stack all possible choices. Then backtracking simply means popping a next choice from the stack.
Language processing: space for parameters and local variables is created internally using a stack. compiler's syntax check for matching braces is implemented by using stack. support for recursion Implementation In the standard library of

classes, the data type stack is an adapter class, meaning that a stack is built on top of other data structures. The underlying structure for a stack could be an array, a vector, an ArrayList, a linked list, or any other collection. Regardless of the type of the underlying data structure, a Stack must implement the same functionality. This is achieved by providing a unique interface:
public interface StackInterface<AnyType> public void push(AnyType e);
public AnyType pop();
public AnyType peek();
public boolean isEmpty(); The following picture demonstrates the idea of implementation by composition.
Another implementation requirement (in addition to the above interface) is that all stack operations must run in constant time O(1). Constant time means that there is some constant k such that an operation takes k nanoseconds of computational time regardless of the stack size.
Array-based implementation
In an array-based implementation we maintain the following fields: an array A of a default size (1), the variable top that refers to the top element in the stack and the capacity that refers to the array size. The variable top changes from -1 to capacity - 1. We say that a stack is empty when top = -1, and the stack is full when top = capacity-1. In a fixed-size stack abstraction, the capacity stays unchanged, therefore when top reaches capacity, the stack object throws an exception. See ArrayStack.java for a complete implementation of the stack class.
In a dynamic stack abstraction when top reaches capacity, we double up the stack siz
Linked List-based implementation
Linked List-based implementation provides the best (from the efficiency point of view) dynamic stack implementation. See ListStack.java for a complete implementation of the stack class.
Queues A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle. An excellent example of a queue is a line of students in the food court of the UC. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed enqueue and dequeue. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.
Implementation In the standard library of classes, the data type queue is an adapter class, meaning that a queue is built on top of other data structures. The underlying structure for a queue could be an array, a Vector, an ArrayList, a LinkedList, or any other collection. Regardless of the type of the underlying data structure, a queue must implement the same functionality. This is achieved by providing a unique interface.
interface QueueInterface‹AnyType> public boolean isEmpty();
public AnyType getFront();
public AnyType dequeue();
public void enqueue(AnyType e);

public void clear(); Each of the above basic operations must run at constant time O(1). The following picture demonstrates the idea of implementation by composition.
Circular Queue Given an array A of a default size (1) with two references back and front, originally set to -1 and 0 respectively. Each time we insert (enqueue) a new item, we increase the back index; when we remove (dequeue) an item - we increase the front index. Here is a picture that illustrates the model after a few steps:
As you see from the picture, the queue logically moves in the array from left to right. After several moves back reaches the end, leaving no space for adding new elements
However, there is a free space before the front index. We shall use that space for enqueueing new items, i.e. the next entry will be stored at index 0, then 1, until front. Such a model is called a wrap around queue or a circular queue
Finally, when back reaches front, the queue is full. There are two choices to handle a full queue:a) throw an exception; b) double the array size.
The circular queue implementation is done by using the modulo operator (denoted
See ArrayQueue.java for a complete implementation of a circular queue.
Applications The simplest two search techniques are known as Depth-First Search(DFS) and Breadth-First Search (BFS). These two searches are described by looking at how the search tree (representing all the possible paths from the start) will be traversed.
Deapth-First Search with a Stack In depth-first search we go down a path until we get to a dead end; then we backtrack or back up (by popping a stack) to get an alternative path.
Create a stack Create a new choice point Push the choice point onto the stack while (not found and stack is not empty) Pop the stack Find all possible choices after the last one tried Push these choices onto the stack Return Breadth-First Search with a Queue In breadth-first search we explore all the nearest possibilities by finding all possible successors and enqueue them to a queue.
Create a queue Create a new choice point Enqueue the choice point onto the queue while (not found and queue is not empty) Dequeue the queue Find all possible choices after the last one tried Enqueue these choices onto the queue Return We will see more on search techniques later in the course.
Arithmetic Expression Evaluation An important application of stacks is in parsing. For example, a compiler must parse arithmetic expressions written using infix notation:
1 + ((2 + 3) * 4 + 5)*6 We break the problem of parsing infix expressions into two stages. First, we convert from infix to a different representation called postfix. Then we parse the postfix expression, which is a somewhat easier problem than directly parsing infix.
Converting from Infix to Postfix. Typically, we deal with expressions in infix notation
2 + 5 where the operators (e.g. +, *) are written between the operands (e.q, 2 and 5). Writing the operators after the operands gives a postfix expression 2 and 5 are called operands, and the '+' is operator. The above arithmetic expression is called infix, since the operator is in between operands. The expression
2 5 + Writing the operators before the operands gives a prefix expression

+2 5 Suppose you want to compute the cost of your shopping trip. To do so, you add a list of numbers and multiply them by the local sales tax (7.25
70 + 150 * 1.0725 Depending on the calculator, the answer would be either 235.95 or 230.875. To avoid this confusion we shall use a postfix notation
70 150 + 1.0725 * Postfix has the nice property that parentheses are unnecessary. Now, we describe how to convert from infix to postfix.
Read in the tokens one at a time If a token is an integer, write it into the output If a token is an operator, push it to the stack, if the stack is empty. If the stack is not empty, you pop entries with higher or equal priority and only then you push that 1. token to the stack. If a token is a left parentheses '(', push it to the stack If a token is a right parentheses ')', you pop entries until you meet '('. When you finish reading the string, you pop up all tokens which are left there. Arithmetic precedence is in increasing order: '+', '-', '*', '/'; Example. Suppose we have an infix expression:2+(4+3*2+1)/3. We read the string by characters. '2' - send to the output. '+' - push on the stack. '(' - push on the stack. '4' - send to the output. '+' - push on the stack. '3' - send to the output. '*' - push on the stack. '2' - send to the output. Evaluating a Postfix Expression. We describe how to parse and evaluate a postfix expression.
We read the tokens in one at a time. If it is an integer, push it on the stack If it is a binary operator, pop the top two elements from the stack, apply the operator, and push the result back on the stack. Consider the following postfix expression
5 9 3 + 4 2 * * 7 + * Here is a chain of operations
Stack Operations Output ———————————— push(5); 5 push(9); 5 9 push(3); 5 9 3 push(pop() + pop()) 5 12 push(4); 5 12 4 push(2); 5 12 4 2 push(pop() * pop()) 5 12 8 push(pop() * pop()) 5 96 push(7) 5 96 7 push(pop() + pop()) 5 103 push(pop() * pop()) 515 Note, that division is not a commutative operation, so 2/3 is not the same as 3/2.
Challenges Stacks: Balanced Brackets Queues: A Tale of Two Stacks References "Stacks and Queues". Victor S.Adamchik, CMU. 2009

6.2.4 Tree

Binary Tree Fundamentally, a binary tree is composed of nodes connected by edges (with further restrictions discussed below). Some binary tree, tt, is either empty or consists of a single root element with two distinct binary tree child elements known as the left subtree and the right subtree of tt. As the name binary suggests, a node in a binary tree has a maximum of 22 children.
The following diagrams depict two different binary trees:
Here are the basic facts and terms to know about binary trees:
The convention for binary tree diagrams is that the root is at the top, and the subtrees branch down from it. A node's left and right subtrees are referred to as children, and that node can be referred to as the parent of those subtrees. A non-root node with no children is called a leaf. Some node aa is an ancestor of some node bb if bb is located in a left or right subtree whose root node is aa. This means that the root node of binary tree tt is the ancestor of all other nodes in the tree. If some node aa is an ancestor of some node bb, then the path from aa to bb is the sequence of nodes starting with aa, moving down the ancestral chain of children, and ending with bb. The depth (or level) of some node aa is its distance (i.e., number of edges)

from the tree's root node. Simply put, the height of a tree is the number of edges between the root node and its furthest leaf. More technically put, it's 1+max(height(leftSubtree),height(rightSubtree))1+max(height(leftSubtree),height(rightSubtree))) (i.e., one more than the maximum of the heights of its left and right subtrees). Any node has a height of 11, and the height of an empty subtree is 11. Because the height of each node is 1+1+ the maximum height of its subtrees and an empty subtree's height is 11, the height of a single-element tree or leaf node is 00. Let's apply some of the terms we learned above to the binary tree on the right:

The root node is AA.

The respective left and right children of AA are BB and EE. The left child of BB is CC. The respective left and right children of EE are FF and DD.

Nodes CC, FF, and DD are leaves (i.e., each node is a leaf).

The root is the ancestor of all other nodes, BB is an ancestor of CC, and EE is an ancestor of FF and DD.

The path between AA and CC is A→B→CA→B→C. The path between AA and FF is A→E→FA→E→F. The path between AA and DD is A →E→D→E→D.

The depth of root node AA is 00. The depth of nodes BB and EE is 11. The depth of nodes CC, FF, and DD, is 22.

The height of the tree, height(t)height(t), is 22. We calculate this recursively as height(t)=1+(max(height(root.leftChild),height(root.rightChild)))height(t)=1+(max(height(root.leftChild),

Because this is long and complicated when expanded, we'll break it down using an image of a slightly simpler version of tt whose height is still 22:

Binary Search Tree A Binary Search Tree (BST), tt, is a binary tree that is either empty or satisfies the following three conditions:

Each element in the left subtree of tt is less than or equal to the root element of tt (i.e., max(leftTree(t).value)t.valuemax(leftTree(t).value)t.valuemax(leftTree(t).value)t.valuemax(leftTree(t).v

Each element in the right subtree of tt is greater than the root element of tt (i.e., max(rightTree(t).value)t.valuemax(rightTree(t).value)t.valuemax(rightTree(t).value)t.valuemax(rightTree(t).

Both leftTree(t)leftTree(t) and rightTree(t)rightTree(t) are BSTs.

You can essentially think of it as a regular binary tree where for each node parent having a leftChildleftChild and rightChildrightChild, leftChild.valueparent.valuerightChild.valueleftChild.valu

In the first diagram at the top of this article, the binary tree of integers on the left side is a binary search tree.

Advantages and Drawbacks Searching for elements is very fast. We know that each node has a maximum of two children and we know that the items are always in the left subtree and the » items are always in the right subtree. To search for an element, we simply need to compare the value we want against the value stored in the root node of the current subtree and work our way down the appropriate child subtrees until we either find the value we're looking for or we hit null (i.e., an empty subtree) which indicates the item is not in the BST. Inserting or searching for a node in a balanced tree is O(logn)O(logn) because you're discarding half of the possible values each time you go left or right.

It can easily become unbalanced. Depending on the insertion order, the tree can very easily become unbalanced (which makes for longer search times). For example, if we create a new tree where the sequence of inserted nodes is 2→1→3→4→5→62→1→3→4→5→6, we end up with the following unbalanced tree:

Observe that the root's left subtree only has one node, whereas the root's right subtree has four nodes. For this reason, inserting or searching for a node in an

unbalanced tree is O(n)O(n).
Challenges "Trees: Is This a Binary Search Tree?". hackerrank. 2016 References
"Binary Trees and Binary Search Trees". AllisonP, hackerrank. 2016

6.2.5 Binary Search Tree

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties
The left sub-tree of a node has a key less than or equal to its parent node's key. The right sub-tree of a node has a key greater than to its parent node's key. Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as
$left_subtree(keys)node(key)right_subtree(keys)RepresentationBSTisacollectionofnodesarrangedinawaywh$
Following is a pictorial representation of BST
We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.
Basic Operations Following are the basic operations of a tree
Search Searches an element in a tree. Insert Inserts an element in a tree. Pre-order Traversal Traverses a tree in a pre-order manner. In-order Traversal Traverses a tree in an in-order manner. Post-order Traversal Traverses a tree in a post-order manner. Node Define a node having some data, references to its left and right child nodes.
struct node int data; struct node *leftChild; struct node *rightChild; ; Search Operation Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.
Algorithm
struct node* search(int data) struct node *current = root; printf("Visiting elements: ");
while(current->data != data)
if(current != NULL) printf("
//go to left tree if(current->data > data) current = current->leftChild; //else go to right tree else current = current->rightChild;
//not found if(current == NULL) return NULL; return current; Insert Operation Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.
Algorithm
void insert(int data) struct node *tempNode = (struct node*) malloc(sizeof(struct node)); struct node *current; struct node *parent;
tempNode->data = data; tempNode->leftChild = NULL; tempNode->rightChild = NULL;
//if tree is empty if(root == NULL) root = tempNode; else current = root; parent = NULL;
while(1) parent = current;
//go to left of the tree if(data < parent->data) current = current->leftChild; //insert to the left

if(current == NULL) parent->leftChild = tempNode; return; //go to right of the tree else current = current->rightChild;
//insert to the right if(current == NULL) parent->rightChild = tempNode; return;

6.3 Heaps

A heap is just what it sounds like — a pile of values organized into a binary tree-like structure adhering to some ordering property. When we add elements to a heap, we fill this tree-like structure from left to right, level by level. This makes heaps really easy to implement in an array, where the value for some index ii's left child is located at index 2i+12i+1 and the value for its right child is at index 2i+22i+2 (using zero-indexing). Here are the two most fundamental heap operations:
add: Insert an element into the heap. You may also see this referred to as push.
poll: Retrieve and remove the root element of the heap. You may also see this referred to as pop. Max Heap This type heap orders the maximum value at the root.
When we add the values 1→2→3→41→2→3→4 to a Max heap, it looks like this:
When we poll the same Max heap until it's empty, it looks like this:
Min Heap This type of heap orders the minimum value at the root.
When we add the values 1→2→3→41→2→3→4 to a Min heap, it looks like this:
When we poll the same Min heap until it's empty, it looks like this:
Applications The heap data structure has many applications.
Heapsort: One of the best sorting methods being in-place and with no quadratic worst-case scenarios. Selection algorithms: A heap allows access to the min or max element in constant time, and other selections (such as median or kth-element) can be done in sub-linear time on data that is in a heap. Graph algorithms: By using heaps as internal traversal data structures, run time will be reduced by polynomial order. Examples of such problems are Prim's minimal-spanning-tree algorithm and Dijkstra's shortest-path algorithm. Priority Queue: A priority queue is an abstract concept like "a list" or "a map"; just as a list can be implemented with a linked list or an array, a priority queue can be implemented with a heap or a variety of other methods. Order statistics: The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array. Challenges "Heaps: Find the Running Median". hackerrank. 2016 References "Heaps". AllisonP, hackerrank. 2016 "Heap (data structure)". wikipedia. 2016

6.4 Sort

6.4.1 Introduction

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Following are some of the examples of sorting in real-life scenarios
Telephone Directory The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily. Dictionary The dictionary stores words in an alphabetical order so that searching of any word becomes easy. In-place Sorting and Not-in-place Sorting Sorting algorithms may require some extra space for comparison and temporary storage of few data elements. These algorithms do not require any extra space and sorting is said to happen in-place, or for example, within the array itself. This is called in-place sorting. Bubble sort is an example of in-place sorting.
However, in some sorting algorithms, the program requires space which is more than or equal to the elements being sorted. Sorting which uses equal or more space is called not-in-place sorting. Merge-sort is an example of not-in-place sorting.
Stable and Not Stable Sorting If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called stable sorting.
If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called unstable sorting.
Stability of an algorithm matters when we wish to maintain the sequence of original elements, like in a tuple for example.
Adaptive and Non-Adaptive Sorting Algorithm A sorting algorithm is said to be adaptive, if it takes advantage of already 'sorted' elements in the list that is to be sorted. That is, while sorting if the source list has some element already sorted, adaptive algorithms will take this into account and will try not to re-order them.
A non-adaptive algorithm is one which does not take into account the elements which are already sorted. They try to force every single element to be re-ordered to confirm their sortedness.
Important Terms Some terms are generally coined while discussing sorting techniques, here is a brief introduction to them
Increasing Order
A sequence of values is said to be in increasing order, if the successive element is greater than the previous one. For example, 1, 3, 4, 6, 8, 9 are in increasing order, as every next element is greater than the previous element.
Decreasing Order
A sequence of values is said to be in decreasing order, if the successive element is less than the current one. For example, 9, 8, 6, 4, 3, 1 are in decreasing order, as every next element is less than the previous element.
Non-Increasing Order
A sequence of values is said to be in non-increasing order, if the successive element is less than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 9, 8, 6, 3, 3, 1 are in non-increasing order, as every next element is less than or equal to (in case of 3) but not greater than any previous element.
Non-Decreasing Order
A sequence of values is said to be in non-decreasing order, if the successive element is greater than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 1, 3,

3, 6, 8, 9 are in non-decreasing order, as every next element is greater than or equal to (in case of 3) but not less than the previous one.

6.4.2 Bubble Sort

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2)(n2) where n is the number of items.
How Bubble Sort Works? We take an unsorted array for our example. Bubble sort takes O(n2)(n2) time so we're keeping it short and precise.
Bubble sort starts with very first two elements, comparing them to check which one is greater.
In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.
We find that 27 is smaller than 33 and these two values must be swapped.
The new array should look like this
Next we compare 33 and 35. We find that both are in already sorted positions. Then we move to the next two values, 35 and 10.
We know then that 10 is smaller 35. Hence they are not sorted.
We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this
To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this
Notice that after each iteration, at least one value moves at the end.
And when there's no swap required, bubble sorts learns that an array is completely sorted.
Now we should look into some practical aspects of bubble sort.
Algorithm We assume list is an array of n elements. We further assume that swap function swaps the values of the given array elements.
begin BubbleSort(list)
for all elements of list if list[i] > list[i+1] swap(list[i], list[i+1]) end if end for return list
end BubbleSort Pseudocode We observe in algorithm that Bubble Sort compares each pair of array element unless the whole array is completely sorted in an ascending order. This may cause a few complexity issues like what if the array needs no more swapping as all the elements are already ascending.
To ease-out the issue, we use one flag variable swapped which will help us see if any swap has happened or not. If no swap has occurred, i.e. the array requires no more processing to be sorted, it will come out of the loop.
Pseudocode of BubbleSort algorithm can be written as follows
procedure bubbleSort(list : array of items)
loop = list.count;
for i = 0 to loop-1 do: swapped = false
for j = 0 to loop-1 do:
/* compare the adjacent elements */ if list[j] > list[j+1] then /* swap them */ swap(list[j], list[j+1]) swapped = true end if
end for
/*if no number was swapped that means array is sorted now, break the loop.*/

if(not swapped) then break end if
end for
end procedure return list Implementation One more issue we did not address in our original algorithm and its improvised pseudocode, is that, after every iteration the highest values settles down at the end of the array. Hence, the next iteration need not include already sorted elements. For this purpose, in our implementation, we restrict the inner loop to avoid already sorted values.

6.4.3 Insertion Sort

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.
The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of (n2), where n is the number of items.
How Insertion Sort Works? We take an unsorted array for our example.
Insertion sort compares the first two elements.
It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.
Insertion sort moves ahead and compares 33 with 27.
And finds that 33 is not in the correct position.
It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.
By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.
These values are not in a sorted order.
So we swap them.
However, swapping makes 27 and 10 unsorted.
Hence, we swap them too.
Again we find 14 and 10 in an unsorted order.
We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.
This process goes on until all the unsorted values are covered in a sorted sub-list.
Now we shall see some programming aspects of insertion sort.
Algorithm Now we have a bigger picture of how this sorting technique works, so we can derive simple steps by which we can achieve insertion sort.
Step 1 If it is the first element, it is already sorted. return 1; Step 2 Pick next element Step 3 Compare with all elements in the sorted sub-list Step 4 Shift all the elements in the sorted sub-list that is greater than the value to be sorted Step 5 Insert the value Step 6 Repeat until list is sorted Pseudocode procedure insertionSort(A : array of items) int holePosition int valueToInsert
for i = 1 to length(A) inclusive do:
/* select value to be inserted */ valueToInsert = A[i] holePosition = i
/*locate hole position for the element to be inserted */
while holePosition > 0 and A[holePosition-1] > valueToInsert do: A[holePosition] = A[holePosition-1] holePosition = holePosition -1 end while

/* insert the number at hole position */ A[holePosition] = valueToInsert
end for
end procedure

6.4.4 Selection Sort

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.
The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.
This algorithm is not suitable for large data sets as its average and worst case complexities are of O(n2)(n2), where nn is the number of items.
How Selection Sort Works? Consider the following depicted array as an example.
For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.
So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.
For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.
We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.
After two iterations, two least values are positioned at the beginning in a sorted manner.
The same process is applied to the rest of the items in the array.
Following is a pictorial depiction of the entire sorting process
Now, let us learn some programming aspects of selection sort.
Algorithm Step 1 Set MIN to location 0 Step 2 Search the minimum element in the list Step 3 Swap with value at location MIN Step 4 Increment MIN to point to next element Step 5 Repeat until list is sorted Pseudocode procedure selection sort list : array of items n : size of list
for i = 1 to n - 1 /* set current element as minimum*/ min = i
/* check the element to be minimum */
for j = i+1 to n if list[j] < list[min] then min = j; end if end for
/* swap the minimum element with the current element*/ if indexMin != i then swap list[min] and list[i] end if
end for
end procedure

6.4.5 Merge Sort

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being O(nlogn)(nlogn), it is one of the most respected algorithms.
Merge sort first divides the array into equal halves and then combines them in a sorted manner.

How Merge Sort Works? To understand merge sort, we take an unsorted array as the following
We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.
This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.
We further divide these arrays and we achieve atomic value which can no more be divided.
Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.
We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.
In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.
After the final merging, the list should look like this
Now we should learn some programming aspects of merge sorting.
Algorithm Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.
Step 1 if it is only one element in the list it is already sorted, return. Step 2 divide the list recursively into two halves until it can no more be divided. Step 3 merge the smaller lists into new list in sorted order. Pseudocode We shall now see the pseudocodes for merge sort functions. As our algorithms point out two main functions divide merge.
Merge sort works with recursion and we shall see our implementation in the same way.
procedure mergesort(var a as array) if (n == 1) return a
var l1 as array = a[0] ... a[n/2] var l2 as array = a[n/2+1] ... a[n]
l1 = mergesort(l1) l2 = mergesort(l2)
return merge(l1, l2) end procedure
procedure merge(var a as array, var b as array)
var c as array
while (a and b have elements) if (a[0] > b[0]) add b[0] to the end of c remove b[0] from b else add a[0] to the end of c remove a[0] from a end if end while
while (a has elements) add a[0] to the end of c remove a[0] from a end while
while (b has elements) add b[0] to the end of c remove b[0] from b end while
return c
end procedure

6.4.6 Shell Sort

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

This algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval. This interval is calculated based on Knuth's formula as
Knuth's Formula h=h3+1h=h3+1
where
hh is interval with initial value 1 This algorithm is quite efficient for medium-sized data sets as its average and worst case complexity are of O(n)(n), where nn is the number of items.
How Shell Sort Works? Let us consider the following example to have an idea of how shell sort works. We take the same array we have used in our previous examples. For our example and ease of understanding, we take the interval of 4. Make a virtual sub-list of all values located at the interval of 4 positions. Here these values are 35, 14, 33, 19, 42, 27 and 10, 44
We compare values in each sub-list and swap them (if necessary) in the original array. After this step, the new array should look like this
Then, we take interval of 2 and this gap generates two sub-lists - 14, 27, 35, 42, 19, 10, 33, 44
We compare and swap the values, if required, in the original array. After this step, the array should look like this
Finally, we sort the rest of the array using interval of value 1. Shell sort uses insertion sort to sort the array.
Following is the step-by-step depiction
We see that it required only four swaps to sort the rest of the array.
Algorithm Following is the algorithm for shell sort.
Step 1 Initialize the value of h Step 2 Divide the list into smaller sub-list of equal interval h Step 3 Sort these sub-lists using insertion sort Step 3 Repeat until complete list is sorted Pseudocode Following is the pseudocode for shell sort.
procedure shellSort() A : array of items
/* calculate interval*/ while interval < A.length /3 do: interval = interval * 3 + 1 end while
while interval > 0 do:
for outer = interval; outer < A.length; outer ++ do:
/* select value to be inserted */ valueToInsert = A[outer] inner = outer;
/*shift element towards right*/ while inner > interval -1 A[inner - interval] >= valueToInsert do: A[inner] = A[inner - interval] inner = inner - interval end while
/* insert the number at hole position */ A[inner] = valueToInsert
end for
/* calculate interval*/ interval = (interval -1) /3;
end while
end procedure

6.4.7 Quick Sort

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quick sort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst case complexity are of (n2), where n is the number of items.
Partition in Quick Sort Following animated representation explains how to find the pivot value in an array.
The pivot value divides the list into two parts. And recursively, we find the pivot for each sub-lists until all lists contains only one element.
Quick Sort Pivot Algorithm Based on our understanding of partitioning in quick sort, we will now try to write an algorithm for it, which is as follows.
Step 1 Choose the highest index value has pivot Step 2 Take two variables to point left and right of the list excluding pivot Step 3 left points to the low index Step 4 right points to the high Step 5 while value at left is less than pivot move right Step 6 while value at right is greater than pivot move left Step 7 if both step 5 and step 6 does not match swap left and right Step 8 if left right, the point where they met is new pivot Quick Sort Pivot Pseudocode The pseudocode for the above algorithm can be derived as
function partitionFunc(left, right, pivot) leftPointer = left rightPointer = right - 1
while True do while A[++leftPointer] < pivot do //do-nothing end while
while rightPointer > 0 A[–rightPointer] > pivot do //do-nothing end while
if leftPointer >= rightPointer break else swap leftPointer,rightPointer end if
end while
swap leftPointer,right return leftPointer
end function Quick Sort Algorithm Using pivot algorithm recursively, we end up with smaller possible partitions. Each partition is then processed for quick sort. We define recursive algorithm for quicksort as follows
Step 1 Make the right-most index value pivot Step 2 partition the array using pivot value Step 3 quicksort left partition recursively Step 4 quicksort right partition recursively Quick Sort Pseudocode To get more into it, let see the pseudocode for quick sort algorithm
procedure quickSort(left, right)
if right-left <= 0 return else pivot = A[right] partition = partitionFunc(left, right, pivot) quickSort(left,partition-1) quickSort(partition+1,right) end if
end procedure

6.5 Search

6.5.1 Linear Search

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.
Algorithm Linear Search (Array A, Value x)
Step 1: Set i to 1 Step 2: if i > n then go to step 7 Step 3: if A[i] = x then go to step 6 Step 4: Set i to i + 1 Step 5: Go to Step 2 Step 6: Print Element x Found at index i and go to step 8 Step 7: Print element not found Step 8: Exit
Pseudocode procedure $\text{linear}_s earch(list, value)$

for each item in the list
if match item == value
return the item's location
end if
end for
end procedure

6.5.2 Binary Search

Binary search is a fast search algorithm with run-time complexity of (log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.
Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero.
How Binary Search Works? For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.
First, we shall determine half of the array by using this formula
mid = low + (high - low) / 2 Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.
Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.
We change our low to mid + 1 and find the new mid value again.
low = mid + 1 mid = low + (high - low) / 2 Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.
The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.
Hence, we calculate the mid again. This time it is 5.
We compare the value stored at location 5 with our target value. We find that it is a match.
We conclude that the target value 31 is stored at location 5.
Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.
Pseudocode The pseudocode of binary search algorithms should look like this
Procedure $\text{binary}_s earchA_, sortedarrayn_, sizeofarrayx_, valuetobesearched$
Set lowerBound = 1 Set upperBound = n
while x not found if upperBound < lowerBound EXIT: x does not exists.
set midPoint = lowerBound + (upperBound - lowerBound) / 2
if A[midPoint] < x set lowerBound = midPoint + 1
if A[midPoint] > x set upperBound = midPoint - 1
if A[midPoint] = x EXIT: x found at location midPoint
end while
end procedure

6.5.3 Interpolation Search

Interpolation search is an improved variant of binary search. This search algorithm works on the probing position of the required value. For this algorithm to work properly, the data collection should be in a sorted form and equally distributed.
Binary search has a huge advantage of time complexity over linear search. Linear search has worst-case complexity of (n) whereas binary search has (log n).
There are cases where the location of target data may be known in advance. For example, in case of a telephone directory, if we want to search the telephone number of Morphius. Here, linear search and even binary search will seem slow as we can directly jump to memory space where the names start from 'M' are stored.
Positioning in Binary Search In binary search, if the desired data is not found then the rest of the list is divided in two parts, lower and higher. The search is carried out in either of them.
Even when the data is sorted, binary search does not take advantage to probe the position of the desired data.
Position Probing in Interpolation Search Interpolation search finds a particular item by computing the probe position. Initially, the probe position is the position of the middle most item of the collection.
If a match occurs, then the index of the item is returned. To split the list into two parts, we use the following method
mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo]) where
A = list Lo = Lowest index of the list Hi = Highest index of the list A[n] = Value stored at index n in the list If the middle item is greater than the item, then the probe position is again calculated in the sub-array to the right of the middle item. Otherwise, the item is searched in the subarray to the left of the middle item. This process continues on the sub-array as well until the size of subarray reduces to zero.
Runtime complexity of interpolation search algorithm is O(log(logn))(log(logn)) as compared to O(logn)(logn) of BST in favorable situations.
Algorithm As it is an improvisation of the existing BST algorithm, we are mentioning the steps to search the 'target' data value index, using position probing
Step 1 Start searching data from middle of the list. Step 2 If it is a match, return the index of the item, and exit. Step 3 If it is not a match, probe position. Step 4 Divide the list using probing formula and find the new midle. Step 5 If data is greater than middle, search in higher sub-list. Step 6 If data is smaller than middle, search in lower sub-list. Step 7 Repeat until match. Pseudocode
A → Array list N → Size of A X → Target Value
Procedure Interpolation$_S earch()$
Set Lo → 0 Set Mid → -1 Set Hi → N-1
While X does not match
if Lo equals to Hi OR A[Lo] equals to A[Hi] EXIT: Failure, Target not found end if
Set Mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo])
if A[Mid] = X EXIT: Success, Target found at Mid else if A[Mid] $<$ X Set Lo to Mid+1 else if A[Mid] $>$ X Set Hi to Mid-1 end if end if
End While
End Procedure

6.5.4 Hash Table

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.
Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.
Hashing Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.
(1,20) (2,70) (42,80) (4,25) (12,44) (14,32) (17,11) (13,78) (37,98) Sr. No. Key Hash Array Index 1 1 1 2 2 2 3 42 42 4 4 4 5 12 12 6 14 4 7 17 7 8 13 3 9 37 7 Linear Probing As we can see, it may happen that the hashing technique is used to create an already used index of the array. In such a case, we can search the next empty location in the array by looking into the next cell until we find an empty cell. This technique is called linear probing.
Sr. No. Key Hash Array Index After Linear Probing, Array Index 1 1 1 2 2 2 3 42 42 4 4 4 5 12 12 6 14 14 7 17 17 8 13 13 9 37 37 Basic Operations Following are the basic primary operations of a hash table.
Search Searches an element in a hash table. Insert inserts an element in a hash table. delete Deletes an element from a hash table. DataItem Define a data item having some data and key, based on which the search is to be conducted in a hash table.
struct DataItem int data; int key; ; Hash Method Define a hashing method to compute the hash code of the key of the data item.
int hashCode(int key) return key Search Operation Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code.
Example
struct DataItem *search(int key) //get the hash int hashIndex = hashCode(key);
//move in array until an empty while(hashArray[hashIndex] != NULL)
if(hashArray[hashIndex]->key == key) return hashArray[hashIndex];
//go to next cell ++hashIndex;
//wrap around the table hashIndex
return NULL; Insert Operation Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing for empty location, if an element is found at the computed hash code.
Example
void insert(int key,int data) struct DataItem *item = (struct DataItem*) malloc(sizeof(struct DataItem)); item->data = data; item->key = key;
//get the hash int hashIndex = hashCode(key);
//move in array until an empty or deleted cell while(hashArray[hashIndex] != NULL hashArray[hashIndex]->key != -1) //go to next cell ++hashIndex;
//wrap around the table hashIndex

hashArray[hashIndex] = item; Delete Operation Whenever an element is to be deleted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing to get the element ahead if an element is not found at the computed hash code. When found, store a dummy item there to keep the performance of the hash table intact.
Example
struct DataItem* delete(struct DataItem* item) int key = item->key;
//get the hash int hashIndex = hashCode(key);
//move in array until an empty while(hashArray[hashIndex] !=NULL)
if(hashArray[hashIndex]->key == key) struct DataItem* temp = hashArray[hashIndex];
//assign a dummy item at deleted position hashArray[hashIndex] = dummyItem; return temp;
//go to next cell ++hashIndex;
//wrap around the table hashIndex
return NULL;

6.6 Graph

6.6.1 Graph Data Structure

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.
Formally, a graph is a pair of sets (V,E)(V,E), where VV is the set of vertices and EE is the set of edges, connecting the pairs of vertices. Take a look at the following graph
In the above graph,
V=a,b,c,d,eV=a,b,c,d,e
E=ab,ac,bd,cd,deE=ab,ac,bd,cd,de
Definitions Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms Vertex Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on. Edge Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0. Adjacency Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on. Path Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.
Basic Operations Following are basic primary operations of a Graph

Add Vertex Adds a vertex to the graph. Add Edge Adds an edge between the two vertices of the graph. Display Vertex Displays a vertex of the graph.

6.6.2 Depth First Traversal

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.
As in the example given above, DFS algorithm traverses from A to B to C to D first then to E, then to F and lastly to G. It employs the following rules.
Rule 1 Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack. Rule 2 If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.) Rule 3 Repeat Rule 1 and Rule 2 until the stack is empty. Algorithms Step Traversal Description 1. Initialize the stack. 2. Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order. 3. Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both Sand D are adjacent to A but we are concerned for unvisited nodes only. 4. Visit D and mark it as visited and put onto the stack. Here, we have B and C nodes, which are adjacent to D and both are unvisited. However, we shall again choose in an alphabetical order. 5. We choose B, mark it as visited and put onto the stack. Here Bdoes not have any unvisited adjacent node. So, we pop Bfrom the stack. 6. We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find D to be on the top of the stack. 7. Only unvisited adjacent node is from D is C now. So we visit C, mark it as visited and put it onto the stack.

6.6.3 Breadth First Traversal

Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.
As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.
Rule 1 Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it * in a queue. Rule 2 If no adjacent vertex is found, remove the first vertex from the queue. Rule 3 Repeat Rule 1 and Rule 2 until the queue is empty. Algorithms Step Traversal Description 1. Initialize the stack. 2. Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order. 3. Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both Sand D are adjacent to A but we are concerned for unvisited nodes only. 4. Visit D and mark it as visited and put onto the stack. Here, we have B and C nodes, which are adjacent to D and both are unvisited. However, we shall again choose in an alphabetical order. 5. We choose B, mark it as visited and put onto the stack. Here Bdoes not have any unvisited adjacent node. So, we pop Bfrom the stack. 6. We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find D to be on the top of the stack. 7. Only unvisited adjacent

node is from D is C now. So we visit C, mark it as visited and put it onto the stack. At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over

6.7 String

String manipulation is a basic operation of many algorithms and utilities such as data validation, text parsing, file conversions and others. The Java APIs contain three classes that are used to work with character data:
Character – A class whose instances can hold a single character value. String – An immutable class for working with multiple characters. StringBuffer and StringBuilder – Mutable classes for working with multiple characters. The String and StringBuffer classes are two you will use the most in your programming assignments. You use the String class in situations when you want to prohibit data modification; otherwise you use the StringBuffer class.
The String class In Java Strings can be created in two different ways. Either using a new operator
String demo1 = new String("This is a string");
char[] demo2 = 's','t','r','i','n','g'; String str = new String(demo2); or using a string literal
String demo3 = "This is a string"; The example below demonstrates differences between these initializations
String s1 = new String("Fester"); String s2 = new String("Fester"); String s3 = "Fester"; String s4 = "Fester"; Then
s1 == s2 returns false s1 == s3 returns false s3 == s4 returns true Because of the importance strings in real life, Java stores (at compile time) all strings in a special internal table as long as you create your strings using a string literal String s3 = "Fester". This process is called canonicalization - it replaces multiple string objects with a single object. This is why in the above example s3 and s4 refer to the same object. Also note that creating strings like s3 and s4 is more efficient. Review the code example StringOptimization.java that demonstrates time comparisons between these two ways of string creation.
Here are some important facts you must know about strings:
1. A string is not an array of characters. Therefore, to access a particular character in a string, you have to use the charAt() method. In this code snippet we get the fourth character which is 't':
String str = "on the edge of history"; char ch = str.charAt(3); 2. The toString() method is used when we need a string representation of an object.
The method is defined in the Object class. For most important classes that you create, you will want to override toString() and provide your own string representation.
3. Comparing strings content using == is the most common mistake beginners do. You compare the content using either equals() or compareTo() methods.
Basic String methods The String class contains an enormous amount of useful methods for string manipulation. The following table presents the most common String methods:
str.charAt(k) returns a char at position k in str. str.substring(k) returns a substring from index k to the end of str s.substring(k, n) returns a substring from

index k to index n-1 of str str.indexOf(s) returns an index of the first occurrence of String s in str str.indexOf(s, k) returns an index of String s starting an index k in str str.startsWith(s) returns true if str starts with s str.startsWith(s, k) returns true if str starts with s at index k str.equals(s) returns true if the two strings have equal values str.equalsIgnoreCase(s) same as above ignoring case str.compareTo(s) compares two strings s.compareToIgnoreCase(t) same as above ignoring case Examine the code in BasicStringDemo.java for further details.

The StringBuffer class In many cases when you deal with strings you will use methods available in the companion StringBuffer class. This mutable class is used when you want to modify the contents of the string. It provides an efficient approach to dealing with strings, especially for large dynamic string data. StringBuffer is similar to ArrayList in a way that the memory allocated to an object is automatically expanded to take up additional data.

Here is an example of reversing a string using string concatenation

public static String reverse1(String s) String str = "";
for(int i = s.length() - 1; i>=0; i–) str += s.charAt(i);
return str; and using a StringBuffer's append
public static String revers2(String s) StringBuffer sb = new StringBuffer();
for(int i = s.length() - 1; i>=0; i–) sb.append(s.charAt(i));
return sb.toString(); Another way to reverse a string is to convert a String object into a StringBuffer object, use the reverse method, and then convert it back to a string:
public static String reverse3(String s) return new StringBuffer(s).reverse().toString();

The performance difference between these two classes is that StringBuffer is faster than String when performing concatenations. Each time a concatenation occurs, a new string is created, causing excessive system resource consumption. Review the code example StringOverhead.java that demonstrates time comparisons of concatenation on Strings and StringBuffer.

StringTokenizer This class (from java.util package) allows you to break a string into tokens (substrings). Each token is a group of characters that are separated by delimiters, such as an empty space, a semicolon, and so on. So, a token is a maximal sequence of consecutive characters that are not delimiters. Here is an example of the use of the tokenizer (an empty space is a default delimiter):

String s = "Nothing is as easy as it looks"; StringTokenizer st = new StringTokenizer(s); while (st.hasMoreTokens()) String token = st.nextToken(); System.out.println("Token [" + token + "]"); Here, hasMoreTokens() method checks if there are more tokens available from the string, and nextToken() method returns the next token from the string tokenizer.

The set of delimiters (the characters that separate tokens) may be specified in the second argument of StringTokenizer. In the following example, StringTokenizer has a set of two delimiters: an empty space and an underscore:

String s = "Every$_s$olution$_b$reedsnewproblems"; $StringTokenizerst = newStringTokenizer(s, "_"); while(st.h$
$handcolumnspecifiestheregularexpressionconstructs, whiletheright-handcolumndescribestheconditions$

Character Classes

[abc] a, b, or c (simple class) $[^abc]Anycharacterexcepta, b, orc(negation)[a-zA-Z]athroughz, orAthroughZ, inclusive(range)[a-d[m-p]]athroughd, ormthroughp : [a-dm-p](union)[a-z[def]]d, e, orf(intersection)[a-z[^bc]]athroughz, exceptforbandc : [ad-z](subtraction)[a-z[^m-p]]athroughz, andnotmthroughp : [a-lq-z](subtraction)$
$danydigitfrom0to9$

$wanywordcharacter(a-z, A-Z, 0-9and_)$
$Wanynon-wordcharacter$
$sanywhitespacecharacter?appearingonceornotatall*appearingzeroormoretimes+$
$appearingoneormoretimesTheJavaStringclasshasseveralmethodsthatallowyoutoperformanoperationusin$
The matches() method The matches("regex") method returns true or false depending whether the string can be matched entirely by the regular expression "regex". For example,
"abc".matches("abc") returns True, but
"abc".matches("bc") returns False. In the following code examples we match all strings that start with any number of dots (denoted by *), followed by "abc" and end with one or more underscores (denoted by +).
String regex = ".*"+"abc"+"$_+$";
"..abc$_{\text{".matches(regex);}}$
"abc$_{\text{".matches(regex);}}$
"abc".$matches(regex); ThereplaceAll()methodThemethodreplaceAll("regex","replacement")replaceseach$
$lettersfromagivenstring$
String str = "Nothing 2is as <> easy AS it $+_{=}looks$!"; $str = str.replaceAll("[^{a}-zA-$
$Z]","");Thepattern"[a-zA-Z]"describesallletters(inupperandlowercases).Nextwenegatethispattern,toge$
$letters"[^{a}-zA-Z]"$.
In the next example, we replace a sequence of characters by "-"
String str = "aabfooaaaabfooabfoob"; str = str.replaceAll("a*b", "-"); The star "" in the pattern "ab" denotes that character "a" may be repeated zero or more times. The output: "-foo-foo-foo-";
The split() method The split("regex") splits the string at each "regex" match and returns an array of strings where each element is a part of the original string between two "regex" matches.
In the following example we break a sentence into words, using an empty space as a delimiter:
String s = "Nothing is as easy as it looks"; String[] st = s.split(" "); Tokens are stored in in an array of strings and could be be easily accessible using array indexes. In the next code example, we choose two delimiters: either an empty space or an underscore:
String s = "Every$_{s}olution_{b}reedsnewproblems"; $String[]st = s.split("_|");Whatifastringcontainsseveralund$
$", thatdenotesarepetitivepattern$
String s = "Every$_{s}olution_{b}^{reeds_{n}ew_{p}roblems";String[]st=s.split("_+");It'simportanttoobservethatsplit()mightreturnsemptytokens.Inth$
String[] st = "Tomorrow".split("r"); we have three tokens, where the second token is empty string. That is so because split() returns tokens between two "regex" matches.
One of the widely use of split() is to break a given text file into words. This could be easily done by means of the metacharacter "" (any non-word character), which allows you to perform a "whole words only" search using a regular expression. A "word character" is either an alphabet character (a-z and A-Z) or a digit (0-9) or a underscore.
"Let's go, Steelers!!!".split("
W"); returns the following array of tokens
[Let, s, go, Steelers] Examine the code in Split.java for further details.
Pattern matching Pattern matching in Java is based on use of two classes
Pattern - compiled representation of a regular expression. Matcher - an engine that performs match operations. A typical invocation is the following, first we

create a pattern
String seq = "CCCAA"; Pattern p = Pattern.compile("C*A*"); In this example we match all substrings that start with any number of Cs followed by any number of As. Then we create a Matcher object that can match any string against our pattern
Matcher m = p.matcher(seq); Finally, we do actual matching
boolean res = m.matches(); The Matcher class has another widely used method, called find(), that finds next substring that matches a given pattern. In the following example we cound the number of matches "ACC"
String seq = "CGTATCCCACAGCACCACACCCAACAACCCA"; Pattern p = Pattern.compile("A1C2"); Matcher m = p.matcher(seq); int count = 0; while(m.find()) count++; System.out.println("there are " + count + " ACC"); Examine the code example Matching.java for further details.
Pattern matching in Computational Biology
The DNA (the genetic blueprint) of any species is composed of about 4 billion ACGT nucleotides. DNA forms a double helix that has two strands of DNA binding and twisting together. In pattern matching problems we ignore the fact that DNA forms a double helix, and think of it only as a single strand. The other strand is complimentary. Knowing one strand allows uniquely determine the other one. Thus, DNA is essentially a linear molecule that looks like a string composed out of only four characters A, C, G, and T:
CGTATCCCACAGCACCACACCCAACAACCC Each nucleotides (also called a base) strongly binds to no more than two other bases. These links provides a linear model of DNA strand. The particular order of ACGT nucleotides is extremely important. Different orders generate humans, animals, corn, and other organisms. The size of the genome (a genome is all the DNA in an organism) does not necessarily correlate with the complexity of the organism it belongs to. Humans have less than a third as many genes as were expected.
Pattern matching in computational biology arises from the need to know characteristics of DNA sequences, such as
find the best way to align two sequences. find any common subsequences determine how well a sequence fits into a given model. Comparing various DNA sequencesn provide many uses. Current scientific theories suggest that very similar DNA sequences have a common ancestor. The more similar two sequences are, the more recently they evolved from a single ansestor. With such knowledge, for example, we can reconstruct a phylogenetic tree (known as a "tree of life".) that shows how long ago various organisms diverged and which species are closely related.
Challenges Strings: Making Anagrams References "Strings". Victor S.Adamchik, CMU. 2009

6.7.1 Tries

Introduction There are many algorithms and data structures to index and search strings inside a text, some of them are included in the standard libraries, but not all of them; the trie data structure is a good example of one that isn't.
Let word be a single string and let dictionary be a large set of words. If we have a dictionary, and we need to know if a single word is inside of the dictionary the tries are a data structure that can help us. But you may be asking yourself, "Why use tries if set and hash tables can do the same?"

There are two main reasons:
The tries can insert and find strings in O(L)O(L) time (where L represent the length of a single word). This is much faster than set , but is it a bit faster than a hash table. The set and the hash tables can only find in a dictionary words that match exactly with the single word that we are finding; the trie allow us to find words that have a single character different, a prefix in common, a character missing, etc.
The tries can be useful in TopCoder problems, but also have a great amount of applications in software engineering. For example, consider a web browser. Do you know how the web browser can auto complete your text or show you many possibilities of the text that you could be writing? Yes, with the trie you can do it very fast. Do you know how an orthographic corrector can check that every word that you type is in a dictionary? Again a trie. You can also use a trie for suggested corrections of the words that are present in the text but not in the dictionary.
What is a Tree? You may read about how wonderful the tries are, but maybe you don't know yet what the tries are and why the tries have this name. The word trie is an infix of the word "retrieval" because the trie can find a single word in a dictionary with only a prefix of the word. The main idea of the trie data structure consists of the following parts:
The trie is a tree where each vertex represents a single word or a prefix. The root represents an empty string (""), the vertexes that are direct sons of the root represent prefixes of length 1, the vertexes that are 2 edges of distance from the root represent prefixes of length 2, the vertexes that are 3 edges of distance from the root represent prefixes of length 3 and so on. In other words, a vertex that are k edges of distance of the root have an associated prefix of length k. Let v and w be two vertexes of the trie, and assume that v is a direct father of w, then v must have an associated prefix of w. The next figure shows a trie with the words "tree", "trie", "algo", "assoc", "all", and "also."
Note that every vertex of the tree does not store entire prefixes or entire words. The idea is that the program should remember the word that represents each vertex while lower in the tree.
Coding a Trie The tries can be implemented in many ways, some of them can be used to find a set of words in the dictionary where every word can be a little different than the target word, and other implementations of the tries can provide us with only words that match exactly with the target word. The implementation of the trie that will be exposed here will consist only of finding words that match exactly and counting the words that have some prefix. This implementation will be pseudo code because different coders can use different programming languages.
We will code these 4 functions:
addWord. This function will add a single string word to the dictionary. countPreffixes. This function will count the number of words in the dictionary that have a string prefix as prefix. countWords. This function will count the number of words in the dictionary that match exactly with a given string word. Our trie will only support letters of the English alphabet. We need to also code a structure with some fields that indicate the values stored in each vertex. As we want to know the number of words that match with a given string, every vertex should have a field to indicate that this vertex represents a complete word or only a prefix (for simplicity, a complete word is considered also a prefix) and

how many words in the dictionary are represented by that prefix (there can be repeated words in the dictionary). This task can be done with only one integer field words.
Because we want to know the number of words that have like prefix a given string, we need another integer field prefixes that indicates how many words have the prefix of the vertex. Also, each vertex must have references to all his possible sons (26 references). Knowing all these details, our structure should have the following members:
structure Trie integer words; integer prefixes; reference edges[26]; And we also need the following functions:
initialize(vertex) addWord(vertex, word); integer countPrefixes(vertex, prefix); integer countWords(vertex, word); First of all, we have to initialize the vertexes with the following function:
initialize(vertex) vertex.words=0 vertex.prefixes=0 for i=0 to 26 edges[i]=NoEdge
The addWord function consists of two parameters, the vertex of the insertion and the word that will be added. The idea is that when a string word is added to a vertex vertex, we will add word to the corresponding branch of vertex cutting the leftmost character of word. If the needed branch does not exist, we will have to create it. All the TopCoder languages can simulate the process of cutting a character in constant time instead of creating a copy of the original string or moving the other characters.
addWord(vertex, word) if isEmpty(word) vertex.words=vertex.words+1 else vertex.prefixes=vertex.prefixes+1 k=firstCharacter(word) if(notExists(edges[k])) edges[k]=createEdge() initialize(edges[k]) cutLeftmostCharacter(word) addWord(edges[k], word) The functions countWords and countPrefixes are very similar. If we are finding an empty string we only have to return the number of words/prefixes associated with the vertex. If we are finding a non-empty string, we should to find in the corresponding branch of the tree, but if the branch does not exist, we have to return 0.
countWords(vertex, word) k=firstCharacter(word) if isEmpty(word) return vertex.words else if notExists(edges[k]) return 0 else cutLeftmostCharacter(word) return countWords(edges[k], word);
countPrefixes(vertex, prefix) k=firstCharacter(prefix) if isEmpty(word) return vertex.prefixes else if notExists(edges[k]) return 0 else cutLeftmostCharacter(prefix) return countWords(edges[k], prefix) Complexity Analysis In the introduction you may read that the complexity of finding and inserting a trie is linear, but we have not done the analysis yet. In the insertion and finding notice that lowering a single level in the tree is done in constant time, and every time that the program lowers a single level in the tree, a single character is cut from the string; we can conclude that every function lowers L levels on the tree and every time that the function lowers a level on the tree, it is done in constant time, then the insertion and finding of a word in a trie can be done in O(L) time. The memory used in the tries depends on the methods to store the edges and how many words have prefixes in common.
Other Kinds of Tries We used the tries to store words with lowercase letters, but the tries can be used to store many other things. We can use bits or bytes instead of lowercase letters and every data type can be stored in the tree, not only strings. Let flow your imagination using tries! For example, suppose that you want to find a word in a dictionary but a single letter was deleted from the word. You can modify the countWords function:

countWords(vertex, word, missingLetters) k=firstCharacter(word) if isEmpty(word) return vertex.word else if notExists(edges[k]) and missingLetters=0 return 0 else if notExists(edges[k]) cutLeftmostCharacter(word) return countWords(vertex, word, missingLetters-1) Here we cut a character but we don't go lower in the tree else We are adding the two possibilities: the first character has been deleted plus the first character is present r=countWords(vertex, word, missingLetters-1) cutLeftmostCharacter(word) r=r+countWords(edges[k], word, missingLetters) return r The complexity of this function may be larger than the original, but it is faster than checking all the subsets of characters of a word.
Challenges "Tries: Contacts". hackerrank. 2016 References "Using Tries – Topcoder". Topcoder.com. N.p., 2016. Web. 11 Oct. 2016.

6.7.2 Suffix Array and suffix tree

A suffix tree T is a natural improvement over trie used in pattern matching problem, the one defined over a set of substrings of a string s. The idea is very simple here. Such a trie can have a long paths without branches. If we only can reduce these long paths into one jump, we will reduce the size of the trie significantly, so this is a great first step in improving the complexity of operations on such a tree. This reduced trie defined over a subset of suffixes of a string s is called a suffix tree of s
For better understanding, let's consider the suffix tree T for a string s = abakan. A word abakan has 6 suffixes abakan , bakan, akan, kan, an, n and its suffix tree looks like this:
There is a famous algorithm by Ukkonen for building suffix tree for s in linear time in terms of the length of s. However, because it may look quite complicated at first sight, many people are discouraged to learn how it works. Fortunately, there is a great, I mean an excellent, description of Ukkonen's algorithm given on StackOverflow. Please refer to it for better understanding what a suffix tree is and how to build it in linear time.
Suffix trees can solve many complicated problems, because it contain so many information about the string itself. Fo example, in order to know how many times a pattern P occurs in s, it is sufficient to find P in T and return the size of a subtree corresponding to its node. Another well known application is finding the number of distinct substrings of s, and it can be solved easily with suffix tree, while the problem looks very complicated at first sight.
The post I linked from StackOverflow is so great, that you simple must read it. After that, you will be able to identify problems solvable with suffix trees easily. If you want to know more about when to use a suffix tree, you should read this paper about the applications of suffix trees.
Suffix Array Suffix array is a very nice array based structure. Basically, it is a lexicographically sorted array of suffixes of a string s. For example, let's consider a string s = abakan. A word abakan has 6 suffixes abakan , bakan, akan, kan, an, n and its suffix tree looks like this:
Of course, in order to reduce space, we do not store the exact suffixes. It is sufficient to store their indices.
Suffix arrays, especially combined with LCP table (which stands for Longest Common Prefix of neighboring suffixes table), are very very useful for solving many problems. I recommend reading this nice programming oriented paper

about suffix arrays, their applications and related problems by Stanford University.
Suffix arrays can be build easily in O(nlog2n)O(nlog2n) time, where n is the length of s, using the algorithm proposed in the paper from the previous paragraph. This time can be improved to O(nlogn)O(nlogn) using linear time sorting algorithm.
However, there is so extraordinary, cool and simple linear time algorithm for building suffix arrays by Kärkkäinen and Sanders, that reading it is a pure pleasure and you cannot miss it.
Correspondence between suffix tree and suffix array
It is also worth to mention, that a suffix array can be constructed directly from a suffix tree in linear time using DFS traversal. Suffix tree can be also constructed from the suffix array and LCP table as described here.

6.7.3 Knuth-Morris-Pratt Algorithm

The problem:
given a (short) pattern and a (long) text, both strings, determine whether the pattern appears somewhere in the text.
We'll go through the Knuth-Morris-Pratt (KMP) algorithm, which can be thought of as an efficient way to build these automata. I also have some working C++ source code which might help you understand the algorithm better.
First let's look at a naive solution.
suppose the text is in an array: char T[n]T[n] and the pattern is in another array: char P[m]P[m]. One simple method is just to try each possible position the pattern could appear in the text.
Naive string matching:
for (i=0; T[i] != ”; i++) for (j=0; T[i+j] != ” P[j] != ” T[i+j]==P[j]; j++) ; if (P[j] == ”) found a match There are two nested loops; the inner one takes O(m) iterations and the outer one takes O(n) iterations so the total time is the product, O(mn). This is slow; we'd like to speed it up.
In practice this works pretty well – not usually as bad as this O(mn) worst case analysis. This is because the inner loop usually finds a mismatch quickly and move on to the next position without going through all m steps. But this method still can take O(mn) for some inputs. In one bad example, all characters in T[] are "a"s, and P[] is all "a"'s except for one "b" at the end. Then it takes m comparisons each time to discover that you don't have a match, so mn overall.
Here's a more typical example. Each row represents an iteration of the outer loop, with each character in the row representing the result of a comparison (X if the comparison was unequal). Suppose we're looking for pattern "nano" in text "bananananobano".
0 1 2 3 4 5 6 7 8 9 10 11 T: b a n a n a n o b a n o
i=0: X i=1: X i=2: n a n X i=3: X i=4: n a n o i=5: X i=6: n X i=7: X i=8: X i=9: n X i=10: X
Some of these comparisons are wasted work! For instance, after iteration i=2, we know from the comparisons we've done that T[3]="a", so there is no point comparing it to "n" in iteration i=3. And we also know that T[4]="n", so there is no point making the same comparison in iteration i=4.
Skipping outer iterations The Knuth-Morris-Pratt idea is, in this sort of situation, after you've invested a lot of work making comparisons in the inner loop of

the code, you know a lot about what's in the text. Specifically, if you've found a partial match of j characters starting at position i, you know what's in positions T[i]...T[i+j-1]. You can use this knowledge to save work in two ways. First, you can skip some iterations for which no match is possible. Try overlapping the partial match you've found with the new match you want to find:
i=2: n a n i=3: n a n o Here the two placements of the pattern conflict with each other – we know from the i=2 iteration that T[3] and T[4] are "a" and "n", so they can't be the "n" and "a" that the i=3 iteration is looking for. We can keep skipping positions until we find one that doesn't conflict:
i=2: n a n i=4: n a n o Here the two "n"'s coincide. Define the overlap of two strings x and y to be the longest word that's a suffix of x and a prefix of y. Here the overlap of "nan" and "nano" is just "n". (We don't allow the overlap to be all of x or y, so it's not "nan"). In general the value of i we want to skip to is the one corresponding to the largest overlap with the current partial match:
String matching with skipped iterations:
i=0; while (i<n) for (j=0; T[i+j] != ” P[j] != ” T[i+j]==P[j]; j++) ; if (P[j] == ”) found a match; i = i + max(1, j-overlap(P[0..j-1],P[0..m])); Skipping inner iterations The other optimization that can be done is to skip some iterations in the inner loop. Let's look at the same example, in which we skipped from i=2 to i=4:
i=2: n a n i=4: n a n o In this example, the "n" that overlaps has already been tested by the i=2 iteration. There's no need to test it again in the i=4 iteration. In general, if we have a nontrivial overlap with the last partial match, we can avoid testing a number of characters equal to the length of the overlap. This change produces (a version of) the KMP algorithm:
KMP, version 1:
i=0; o=0; while (i<n) for (j=o; T[i+j] != ” P[j] != ” T[i+j]==P[j]; j++) ; if (P[j] == ”) found a match; o = overlap(P[0..j-1],P[0..m]); i = i + max(1, j-o); The only remaining detail is how to compute the overlap function. This is a function only of j, and not of the characters in T[], so we can compute it once in a preprocessing stage before we get to this part of the algorithm. First let's see how fast this algorithm is.
KMP time analysis We still have an outer loop and an inner loop, so it looks like the time might still be O(mn). But we can count it a different way to see that it's actually always less than that. The idea is that every time through the inner loop, we do one comparison T[i+j]==P[j]. We can count the total time of the algorithm by counting how many comparisons we perform. We split the comparisons into two groups: those that return true, and those that return false. If a comparison returns true, we've determined the value of T[i+j]. Then in future iterations, as long as there is a nontrivial overlap involving T[i+j], we'll skip past that overlap and not make a comparison with that position again. So each position of T[] is only involved in one true comparison, and there can be n such comparisons total. On the other hand, there is at most one false comparison per iteration of the outer loop, so there can also only be n of those. As a result we see that this part of the KMP algorithm makes at most 2n comparisons and takes time O(n).
KMP and finite automata If we look just at what happens to j during the algorithm above, it's sort of like a finite automaton. At each step j is set either to j+1 (in the inner loop, after a match) or to the overlap o (after a mismatch). At each step the value of o is just a function of j and doesn't depend on other infor-

mation like the characters in T[]. So we can draw something like an automaton, with arrows connecting values of j and labeled with matches and mismatches.
The difference between this and the automata we are used to is that it has only two arrows out of each circle, instead of one per character. But we can still simulate it just like any other automaton, by placing a marker on the start state (j=0) and moving it around the arrows. Whenever we get a matching character in T[] we move on to the next character of the text. But whenever we get a mismatch we look at the same character in the next step, except for the case of a mismatch in the state j=0.
So in this example (the same as the one above) the automaton goes through the sequence of states:
j=0 mismatch T[0] != "n" j=0 mismatch T[1] != "n" j=0 match T[2] == "n" j=1 match T[3] == "a" j=2 match T[4] == "n" j=3 mismatch T[5] != "o" j=1 match T[5] == "a" j=2 match T[6] == "n" j=3 match T[7] == "o" j=4 found match j=0 mismatch T[8] != "n" j=0 mismatch T[9] != "n" j=0 match T[10] == "n" j=1 mismatch T[11] != "a" j=0 mismatch T[11] != "n" This is essentially the same sequence of comparisons done by the KMP pseudocode above. So this automaton provides an equivalent definition of the KMP algorithm. As one student pointed out in lecture, the one transition in this automaton that may not be clear is the one from j=4 to j=0. In general, there should be a transition from j=m to some smaller value of j, which should happen on any character (there are no more matches to test before making this transition). If we want to find all occurrences of the pattern, we should be able to find an occurrence even if it overlaps another one. So for instance if the pattern were "nana", we should find both occurrences of it in the text "nanana". So the transition from j=m should go to the next longest position that can match, which is simply j=overlap(pattern,pattern). In this case overlap("nano","nano") is empty (all suffixes of "nano" use the letter "o", and no prefix does) so we go to j=0.
Alternate version of KMP The automaton above can be translated back into pseudo-code, looking a little different from the pseudo-code we saw before but performing the same comparisons.
KMP, version 2:
j = 0; for (i = 0; i < n; i++) for (;;) // loop until break if (T[i] == P[j]) // matches? j++; // yes, move on to next state if (j == m) // maybe that was the last state found a match; j = overlap[j]; break; else if (j == 0) break; // no match in state j=0, give up else j = overlap[j]; // try shorter partial match The code inside each iteration of the outer loop is essentially the same as the function match from the C++ implementation I've made available. One advantage of this version of the code is that it tests characters one by one, rather than performing random access in the T[] array, so (as in the implementation) it can be made to work for stream-based input rather than having to read the whole text into memory first. The overlap[j] array stores the values of overlap(pattern[0..j-1],pattern), which we still need to show how to compute.
Since this algorithm performs the same comparisons as the other version of KMP, it takes the same amount of time, O(n). One way of proving this bound directly is to note, first, that there is one true comparison (in which T[i]==P[j]) per iteration of the outer loop, since we break out of the inner loop when this happens. So there are n of these total. Each of these comparisons results in increasing j by one. Each iteration of the inner loop in which we don't break out of the loop results in executing the statement j=overlap[j], which decreases

j. Since j can only decrease as many times as it's increased, the total number of times this happens is also O(n).
Computing the overlap function Recall that we defined the overlap of two strings x and y to be the longest word that's a suffix of x and a prefix of y. The missing component of the KMP algorithm is a computation of this overlap function: we need to know overlap(P[0..j-1],P) for each value of j>0. Once we've computed these values we can store them in an array and look them up when we need them. To compute these overlap functions, we need to know for strings x and y not just the longest word that's a suffix of x and a prefix of y, but all such words. The key fact to notice here is that if w is a suffix of x and a prefix of y, and it's not the longest such word, then it's also a suffix of overlap(x,y). (This follows simply from the fact that it's a suffix of x that is shorter than overlap(x,y) itself.) So we can list all words that are suffixes of x and prefixes of y by the following loop:
while (x != empty) x = overlap(x,y); output x; Now let's make another definition: say that shorten(x) is the prefix of x with one fewer character. The next simple observation to make is that shorten(overlap(x,y)) is still a prefix of y, but is also a suffix of shorten(x). So we can find overlap(x,y) by adding one more character to some word that's a suffix of shorten(x) and a prefix of y. We can just find all such words using the loop above, and return the first one for which adding one more character produces a valid overlap:
Overlap computation:
z = overlap(shorten(x),y) while (last char of x != y[length(z)]) if (z = empty) return overlap(x,y) = empty else z = overlap(z,y) return overlap(x,y) = z
So this gives us a recursive algorithm for computing the overlap function in general. If we apply this algorithm for x=some prefix of the pattern, and y=the pattern itself, we see that all recursive calls have similar arguments. So if we store each value as we compute it, we can look it up instead of computing it again. (This simple idea of storing results instead of recomputing them is known as dynamic programming; we discussed it somewhat in the first lecture and will see it in more detail next time.) So replacing x by P[0..j-1] and y by P[0..m-1] in the pseudocode above and replacing recursive calls by lookups of previously computed values gives us a routine for the problem we're trying to solve, of computing these particular overlap values. The following pseudocode is taken (with some names changed) from the initialization code of the C++ implementation I've made available. The value in overlap[0] is just a flag to make the rest of the loop simpler. The code inside the for loop is the part that computes each overlap value.
KMP overlap computation:
overlap[0] = -1; for (int i = 0; pattern[i] != ''; i++) overlap[i + 1] = overlap[i] + 1; while (overlap[i + 1] > 0 pattern[i] != pattern[overlap[i + 1] - 1]) overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1; return overlap; Let's finish by analyzing the time taken by this part of the KMP algorithm. The outer loop executes m times. Each iteration of the inner loop decreases the value of the formula overlap[i+1], and this formula's value only increases by one when we move from one iteration of the outer loop to the next. Since the number of decreases is at most the number of increases, the inner loop also has at most m iterations, and the total time for the algorithm is O(m). The entire KMP algorithm consists of this overlap computation followed by the main part of the algorithm in which we scan the text (using the overlap values to speed up the scan). The first part

takes O(m) and the second part takes O(n) time, so the total time is O(m+n).

Chương 7

Object Oriented Programming

View online http://magizbox.com/training/object$_o$riented$_p$rogramming/site/
Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another. There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.
Many of the most widely used programming languages (such as C++, Java, Python etc.) are multi-paradigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming. Significant object-oriented languages include Java, C++, C, Python, PHP, Ruby, Perl, Delphi, Objective-C, Swift, Scala, Common Lisp, and Smalltalk.

7.1 OOP

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A distinguishing feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OO programming, computer programs are designed by making them out of objects that interact with one another.[1][2] There is significant diversity in object-oriented programming, but most popular languages are class-based, meaning that objects are instances of classes, which typically also determines their type. 1. A First Look
Procedural vs Object Oriented 1
Procedural Approach

Focus is on procedures All data is shared: no protection More difficult to modify Hard to manage complexity Advantages of Object Orientation
People think in terms of object OO models map to reality OO models are: Easy to develop Easy to understand. 2. Principles encapsulation, inheritance, abstraction, polymorphism 2
Fundamental Principles of OOP In order for a programming language to be object-oriented, it has to enable working with classes and objects as well as the implementation and use of the fundamental object-oriented principles and concepts: inheritance, abstraction, encapsulation and polymorphism.
2.1 Encapsulation 3 4 5
Encapsulation is the packing of data and functions into a single component. The features of encapsulation are supported using classes in most object-oriented programming languages, although other alternatives also exist. It allows selective hiding of properties and methods in an object by building an impenetrable wall to protect the code from accidental corruption.
What it do? We will learn to hide unnecessary details in our classes and provide a clear and simple interface for working with them.
Example: A popular example you'll hear for encapsulation is driving a car. Do you need to know exactly how every aspect of a car works (engine, carburettor, alternator, and so on)? No - you need to know how to use the steering wheel, brakes, accelerator, and so on.
2.2 Inheritance 6 7
Inheritance is when an object or class is based on another object (prototypal inheritance) or class (class-based inheritance), using the same implementation (inheriting from an object or class) specifying implementation to maintain the same behavior (realizing an interface; inheriting behavior).
inherit everything, add data or functionality, override functions, super
What it do? We will explain how class hierarchies improve code readability and enable the reuse of functionality.
Example: A real-world example of inheritance is genetic inheritance. We all receive genes from both our parents that then define who we are. We share qualities of both our parents, and yet at the same time are different from them.
Example: we might classify different kinds of vehicles according to the inheritance hierarchy. Moving down the hierarchy, each kind of vehicle is both more specialized than its parent (and all of its ancestors) and more general than its children (and all of its descendants). A wheeled vehicle inherits properties common to all vehicles (it holds one or more people and carries them from place to place) but has an additional property that makes it more specialized (it has wheels). A car inherits properties common to all wheeled vehicles, but has additional, more specialized properties (four wheels, an engine, a body, and so forth). The inheritance relationship can be viewed as an is-a relationship. In this relationship, the objects become more specialized the lower in the hierarchy you go.
Look at the image above you will get a point.8 Yes, the derived class can access base class properties and still the derived class has its own properties.
2.3 Abstraction
In computer science, abstraction is a technique for managing complexity of computer systems. It works by establishing a level of complexity on which a person interacts with the system, suppressing the more complex details below the current level. The programmer works with an idealized interface (usually well

defined) and can add additional levels of functionality that would otherwise be too complex to handle.
What it do? We will learn how to work through abstractions: to deal with objects considering their important characteristics and ignore all other details.
Example: You'll never buy a "device", but always buy something more specific : iPhone, Samsung Galaxy, Nokia 3310... Here, iPhone, Samsung Galaxy and Nokia 3310 are concrete things, device is abstract.
2.4 Polymorphism 9
Polymorphism is the provision of a single interface to entities of different types. A polymorphic type is one whose operations can also be applied to values of some other type, or types.
What it do? We will explain how to work in the same manner with different objects, which define a specific implementation of some abstract behavior.
Example: All animal can speak, but dogs woof, cats meow, and ducks quack
There are two types of polymorphism
Overloading (compile time polymorphism): methods have the same name but different parameters. Overriding (run time polymorphism): the implementation given in base class is replaced with that in sub class.
Example 10: Let us Consider Car example for discussing the polymorphism. Take any brand like Ford, Honda, Toyota, BMW, Benz etc., Everything is of type Car. But each have their own advanced features and more advanced technology involved in their move behavior.
3. Concepts Learn Object Oriented Programming though Mario Game
[embed]https://www.youtube.com/watch?v=HBbzYKMfx5Y[/embed]
How Mario get 1up
3.1. Object 11
Objects are key to understanding object-oriented technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle. In mario world, Mario is an object.
Goomba is an object. Koopa is also an object. Even a coin and a pile are objects
Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state in fields (variables in some programming languages) and exposes its behavior through methods (functions in some programming languages). Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication. Hiding internal state and requiring all interaction to be performed through an object's methods is known as data encapsulation — a fundamental principle of object-oriented programming. In Mario world, Mario has some fields like position (which indicate where Mario stands), state (which indicate whether Mario alive), and some methods like walk , fire or jump.
Goomba has some fields like position (which indicate where Goomba stands), state (which indicate whether Goomba die), and direction (which indicate the direction Goomba moves). Goomba has move method, and jumped$_o$nmethod(whichoccurswhenitisjumpedonby
Mario Objects, real scene
3.2 Class 12
In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that

your bicycle is an instance of the class of objects known as bicycles. A class is the blueprint from which individual objects are created.
In Mario world, each coin object come from Coin class, and every Koomba come from Koomba class
3.3. Inheritance 13
Inheritance is a mechanism in OOP to design two or more entities that are different but share many common features.
Feature common to all classes are defined in the superclass The classes that inherit common features from the superclass are called subclasses In Mario World, Goomba and Koopa is in
AND MANY, MANY MORE
3.4. Association, Aggregation and Composition
13
Association:
Whenever two objects are related with each other the relationshiop is called association between object
Aggregation:
Aggregation is specialized from of association. In aggregation objects have their own life-cycle but there is ownership and child object can not belongs to another parent object. But this is only an ownership not the life-cycle control of child control through parent object.
Example: Student and Teacher, Person and address
Composition
Composition is again specialize form of aggregation and we can call this as ‘life and death“ relationship. It is a strong type of aggregation. Child object does not have their life-cycle and if parent object is deleted, all child object will also be deleted.
Example: House and room
3.5 Polymorphism 13
Polymorphism indicates the meaning of "many forms"
Polynorphism present a method that can have many definitions. Polymorphism is related to "over loading" and "over ridding".
Overloading indicates a method can have different definitions by defining different type of parameters.
[code] getPrice(): void getPrice(string name): void [/code]
3.6 Abstraction 13
Abstraction is the process of modelling only relevant features
Hide unnecessary details which are irrelevant for current purpose. Reduces complexity and aids understanding.
Abstraction provides the freedom to defer implementation decisions by avoiding commitments to details.
3.7 Interface 13
An interface is a contract consisting of group of related function prototypes whose usage is defined but whose implementation is not:
An interface definition specifies the interface’s member functions, called methods, their return types, the number and types of parameters and what they must do.
These is no implementation associated with an interface.
4. Coupling and Cohesion 13

4.1 Coupling Coupling defines how dependent one object on another object (that is uses).
Coupling is a measure of strength of connection between any two system components. The more any one components knows about other components, the tighter (worse) the coupling is between those components.
4.2 Cohesion Cohesion defines how narrowly defined an object is. Functional cohesion refers measures how strongly objects are related.
Cohesion is a measure of how logically related the parts of an individual components are to each other, and to the overall components. The more logically related the parts of components are to each other higher (better) the cohesion of that components.
4.3 Object Oriented Design Low coupling and tight cohesion is good object oriented design.
Challenge Object Task 1: With boiler plate code, make an gif image (32x32) Mario fire ball and jump to get coins
5. NEXT Design Principles Design Patterns

7.2 UML

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.
http://www.yuml.me/ Use UML with IntellIJ: UML Designer Architecture 1
Design of a system consists of classes, interfaces and collaboration. UML provides class diagram, object diagram to support this. Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support implementation perspective. Process defines the flow of the system. So the same elements as used in Design are also used to support this perspective. Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective. Modelling Types 2
Diagrams Usecase Diagram 3 4 5
A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.
A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.
Use case diagrams depict:
Use cases. A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse. (example) Actors. An actor is a person, organization, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures. (example) Associations. Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modeled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicating the direction of the initial invocation of the relationship or to indicate the primary actor within

the use case. The arrowheads are typically confused with data flow and as a result I avoid their use. (example) Extend: Extend is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional) extending use case can be inserted into the behavior defined in the extended use case. (example) Include is a directed relationship between two use cases which is used to show that behavior of the included use case (the addition) is inserted into the behavior of the including (the base) use case. (example) System boundary boxes (optional). You can draw a rectangle around the use cases, called the system boundary box, to indicates the scope of your system. Anything within the box represents functionality that is in scope and anything outside the box is not. System boundary boxes are rarely used, although on occasion I have used them to identify which use cases will be delivered in each major release of a system. (example) Packages (optional). Packages are UML constructs that enable you to organize model elements (such as use cases) into groups. Packages are depicted as file folders and can be used on any of the UML diagrams, including both use case diagrams and class diagrams. I use packages only when my diagrams become unwieldy, which generally implies they cannot be printed on a single page, to organize a large diagram into smaller ones. (example) Class Diagram 6

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

3.3.1 UML Association 9 10

Association

Association is reference based relationship between two classes. Here a class A holds a class level reference to class B. Association can be represented by a line between these classes with an arrow indicating the navigation direction. In case arrow is on the both sides, association has bidirectional navigation.

Aggregation

Aggregation (shared aggregation) is a "weak" form of aggregation when part instance is independent of the composite:

the same (shared) part could be included in several composites, and if composite is deleted, shared parts may still exist. Shared aggregation is shown as binary association decorated with a hollow diamond as a terminal adornment at the aggregate end of the association line. The diamond should be noticeably smaller than the diamond notation for N-ary associations. Shared aggregation is shown as binary association decorated with a hollow diamond.

Composition

Composition (composite aggregation) is a "strong" form of aggregation. Composition requirements/features listed in UML specification are:

it is a whole/part relationship, it is binary association part could be included in at most one composite (whole) at a time, and if a composite (whole) is deleted, all of its composite parts are "normally" deleted with it. Note, that UML does not define how, when and specific order in which parts of the composite are created. Also, in some cases a part can be removed from a composite before the composite is deleted, and so is not necessarily deleted as part of the composite.

Aggregation vs Composition

11

Sequence Diagram 7

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart.
A sequence diagram shows object interactions arranged in time sequence.
It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.
Activity Diagram 8
Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control.
UML - Architecture
UML - Modeling Types
UML - Use Case Diagrams
Use Case Diagram
UML Association Between Actor and Use Case
Class diagram
Sequence diagram
Activity diagram
Aggregation
UML Class Diagram: Association, Aggregation and Composition
Lecture Notes on Object-Oriented Programming: Object Oriented Aggregation

7.3 SOLID

SOLID Principles In computer programming, SOLID (single responsibility, open-closed, Liskov substitution, interface segregation and dependency inversion) is a mnemonic acronym introduced by Michael Feathers for the "first five principles" named by Robert C. Martin in the early 2000s that stands for five basic principles of object-oriented programming and design. The intention is that these principles, when applied together, will make it more likely that a programmer will create a system that is easy to maintain and extend over time. The principles of SOLID are guidelines that can be applied while working on software to remove code smells by providing a framework through which the programmer may refactor the software's source code until it is both legible and extensible. It is part of an overall strategy of agile and Adaptive Software Development.
"Dependency Management is an issue that most of us have faced. Whenever we bring up on our screens a nasty batch of tangled legacy code, we are experiencing the results of poor dependency management. Poor dependency management leads to code that is hard to change, fragile, and non-reusable."
Uncle Bob talk about several different design smells in the PPP book, all relating to dependency management. On the other hand, when dependencies are well managed, the code remains flexible, robust, and reusable. So dependency

management, and therefore these principles, are at the foundation of the -ilities that software developers desire.
SRP - Single Responsibility A class should have one, and only one, reason to change.
A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)
Original Paper
OCP - Open/Closed You should be able to extend a classes behavior, without modifying it.
Software entities ... should be open for extension, but closed for modification."
Original Paper
LSP - Liskov Substitution Derived classes must be substitutable for their base classes.
Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program. See also design by contract.
Original Paper
ISP - Interface Segregation Make fine grained interfaces that are client specific.
Many client-specific interfaces are better than one general-purpose interface.
Original Paper
DIP - Dependency Inversion Depend on abstractions, not on concretions.
One should "depend upon abstractions, not concretions."
Original Paper
References The Principles of OOD

7.4 Design Patterns

Design Patterns
Creational design patterns These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.
Structural design patterns These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.
Behavioral design patterns These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.
Design Pattern QA Examples of GoF Design Patterns in Java's core libraries Dependency Injection vs Factory Pattern What is Inversion of Control? What is so bad about singletons? What is the basic difference between Factory and Abstract Factory Patterns? When would you use the Builder Pattern? What is the difference between Builder Design pattern and Factory Design pattern? How do the Proxy, Decorator, Adapter, and Bridge Patterns differ? Abstract Factory Pattern Creates an instance of several families of classes Intuitive 1
Volkswagen Transparent Factory in Dresden

What is it? 2 The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes.
In normal usage, the client software creates a concrete implementation of the abstract factory and then uses the generic interface of the factory to create the concrete objects that are part of the theme. The client doesn't know (or care) which concrete objects it gets from each of these internal factories, since it uses only the generic interfaces of their products.
This pattern separates the details of implementation of a set of objects from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface.
Design
Example Code
The most interesting factories in the world Abstract factory pattern Observer Pattern Intuitive
Definition 1 The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems. The Observer pattern is also a key part in the familiar model–view–controller (MVC) architectural pattern. The observer pattern is implemented in numerous programming libraries and systems, including almost all GUI toolkits. Structure 2
Subject
knows its observers. Any number of Observer objects may observe a subject. provides an interface for attaching and detaching Observer objects Observer
defines an updating interface for objects that should be notified of changes in a subject. ConcreteSubject
stores state of interest to ConcreteObserver objects. sends a notification to its observers when its state changes. ConcreteObserver
maintains a reference to a ConcreteSubject object. stores state that should stay consistent with the subject's. implements the Observer updating interface to keep its state consistent with the subject's. Examples Example 1: Blog Manager Application
In this application, each user is an Observer, each blog is a Subject. When a blog post a new article (state change), user get an update. When users get update, they update their articles.
[code lang="java"] Blog sportBlog = new Blog("SPORT"); User user1 = new User("Fan1"); User user2 = new User("Fan2");
sportBlog.attach(user1); sportBlog.attach(user2);
sportBlog.post(new Article("football")); sportBlog.post(new Article("swimming"));
user1.getArticles(); user2.getArticles();
sportBlog.detach(user1);‘ [/code]
Real Implementations Broadcast Receiver 3 4 on Android
More Articles http://javapapers.com/design-patterns/observer-design-pattern/
Comparison Observer/Observable pattern vs Publisher/Subscriber pattern 5 Observer/Observable pattern is mostly implemented in a synchronous way, i.e. the observable calls the appropriate method of all its observers when some event occurs. The Publisher/Subscriber pattern is mostly implemented in an asynchronous way (using message queue). In the Observer/Observable pattern, the

observers are aware of the observable. Whereas, in Publisher/Subscriber, publishers and subscribers don't need to know each other. They simply communicate with the help of message queues. Observer pattern
Broadcast Receiver
Design Patterns: Elements of Reusable Object-Oriented Software
Which design patterns are used on Android?
stackoverflow, Difference between Observer, Pub/Sub, and Data Binding

Chương 8

Database

View online http://magizbox.com/training/computer$_s$*cience*/*site*/*database*/

8.1 Introduction

Relational DBMS: Oracle, MySQL, SQLite
Key-value Stores: Redis, Memcached
Document stores: MongoDB
Graph: Neo4j
Wide column stores: Cassandra, HBase
Design and Modeling (a.k.a Data Definition) 1.1 Schema A database schema of a database system is its structure described in a formal language supported by the database management system (DBMS) and refers to the organization of data as a blueprint of how a database is constructed (divided into database tables in the case of Relational Databases). The formal definition of database schema is a set of formulas (sentences) called integrity constraints imposed on a database. These integrity constraints ensure compatibility between parts of the schema. All constraints are expressible in the same language. A database can be considered a structure in realization of the database language. The states of a created conceptual schema are transformed into an explicit mapping, the database schema. This describes how real world entities are modeled in the database.
1.1.1 Type In computer science and computer programming, a data type or simply type is a classification identifying one of various types of data, such as real, integer or Boolean, that determines the possible values for that type; the operations that can be done on values of that type; the meaning of the data; and the way values of that type can be stored.
TEXT, INT, ENUM, TIMESTAMP
1.2 Cardinality (a.k.a Relationship) Foreign key, Primary key
1.2 Indexing A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns

of a database table, providing the basis for both rapid random lookups and efficient access of ordered records. Why Indexing is important?
Indexing in MySQL
CREATE INDEX NameIndex ON Employee (name) SELECT * FROM Employee WHERE name = 'Ashish' 2. Data Manipulation Create - Read - Update - Delete Create or add new entries Read, retrieve, search, or view existing entries * Update or edit existing entries * Delete/deactivate existing entries /* create */
CREATE TABLE Guests (id INT(6) UNSIGNED AUTO$_INCREMENTPRIMARYKEY, firstnameVAR$
$create(insert)*/INSERTINTOGuests(firstname, lastname, email)VALUES('John','Doe','john@exam$
$read*/SELECT*FROMGuestsWHEREid = 1/*update*/UPDATEGuestsSETlastname ='$
$Doe'WHEREid = 1/*delete*/DELETEFROMGuestsWHEREid = 1`3.DataRetrieveTransaction3.1Da$
Get user id, user name and number of post of this user
SELECT user.id, user.name, COUNT(post.*) AS posts FROM user LEFT OUTER JOIN post ON post.owner$_id = user.idGROUPBYuser.id`Selectuserwhoonlyorderonetime.$
SELECT name, COUNT(name) AS c FROM orders GROUP BY name HAVING c = 1; Calculate the longest period (in days) that the company has gone without a hiring or firing anyone.
SELECT x.date, MIN(y.date) y$_date, DATEDIFF(MIN(y.date), x.date)daysFROM(SELECThiredatedat$
$x.dateGROUPBYx.dateORDERBYdaysDESCLIMIT1; DataRetrieveAPI$
API Description get get single item Get dog by id
Dog.get(1) find find items
@see collection.find()
Find dog name "Max"
Dog.find("name": "Max") sort sort items
@see cursor.sort
Get 10 older dogs
Dog.find().sort("age", limit: 10) aggregate sum, min, max items
@see collection.aggregate
Get sum of dogs' age
Dog.find().aggregate("sum$_age" : sum: "age") 3.2 Transaction A transaction symbolizes a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in database. Example: Transfer $900 from Account$
Bob to Alice
start transaction select balance from Account where Account$_Number =' Bob'; selectbalancefromAccountwhe$
$Alice'; updateAccountsetbalance = balance-900hereAccount_Number =' Bob'; updateAccountsetbalance =$
$balance+900hereAccount_Number =' Alice'; commit; //ifallsqlqueriessuccedrollback; //ifanyofSqlqueries$
In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably. In the context of databases, a single logical operation on the data is called a transaction.
For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction. ![][16]
4. Backup and Restore Sometimes it is desired to bring a database back to a previous state (for many reasons, e.g., cases when the database is found corrupted due to a software error, or if it has been updated with erroneous data). To achieve this a backup operation is done occasionally or continuously, where each desired database state (i.e., the values of its data and their embedding in

database's data structures) is kept within dedicated backup files (many techniques exist to do this effectively). When this state is needed, i.e., when it is decided by a database administrator to bring the database back to this state (e.g., by specifying this state by a desired point in time when the database was in this state), these files are utilized to restore that state.
5. Migration In software engineering, schema migration (also database migration, database change management) refers to the management of incremental, reversible changes to relational database schemas. A schema migration is performed on a database whenever it is necessary to update or revert that database's schema to some newer or older version. Example: Android Migration by droid-migrate
droid-migrate init -d my$_{d}atabasedroid-migrategenerateupdroid-migrategeneratedownExample$: $DatabaseSeedingwithLaravel$
6. Active record pattern | Object-Relational Mapping (ORM) Object-relational mapping in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.
Example
php
$employee = newEmployee()$;employee->setName("Joe"); $employee- > save()$; $Android$
public class User @DatabaseField(id = true) String username; @DatabaseField String password; @DatabaseField String email; @DatabaseField String alias; public User() Implementations
Android: [ormlite-android] PHP: [Eloquent]

8.2 SQL

SQL SELECT * FROM WORLD
INSERT INTO
SELECT * FROM girls

8.3 MySQL

MySQL
MySQL is an open-source relational database management system (RDBMS); in July 2013, it was the world's second most widely used RDBMS, and the most widely used open-source client–server model RDBMS. It is named after co-founder Michael Widenius's daughter, My. The SQL abbreviation stands for Structured Query Language. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. For proprietary use, several paid editions are available, and offer additional functionality.

MySQL: Docker Docker Run docker pull mysql docker run -d -p 3306:3306 –env MYSQL$_{R}OOT_{P}ASSWORD = docker - -envMYSQL_{D}ATABASE = docker - -envMYSQL_{U}SER = docker - -envMYSQL_{P}ASSWORD = docker mysqlNote : OnWindows, viewyour0.0.0.0IPbyrunningbelowcommandline(oryoucanturnonKitema
Docker Compoose Step 1: Clone Docker Project
git clone https://github.com/magizbox/docker-mysql.git mv docker-mysql mysql
Step 2: Docker Compose
version: "2"
services: mysql: build: ./mysql/. ports: - 3306:3306 environment: - MYSQL$_{R}OOT_{P}ASSWORD = docker−MYSQL_{D}ATABASE = docker−MYSQL_{U}SER = docker−MYSQL_{P}ASSWORD = dockervolumes : −./data/mysql : /var/lib/mysqlDockerFoldermysql/configmy.cnfDockerfileVerifydoc machinelsNAMEACTIVEDRIVERSTATEURLSWARMdefault∗virtualboxRunningtcp : //192.168.99.100 : 2376Youcanaddphpmyadmintoseemysqlworks
phpmyadmin: image: phpmyadmin/phpmyadmin links: - mysql environment: - PMA$_{A}RBITRARY = 1ports : −80 : 80Seeitworks
Go to localhost Login with Server=mysql, Username=docker, Password=docker

8.4 Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as database, cache and message broker. 1
It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.
Redis: Client Python Client pipy/redis
Installation
pip install redis Usage
import redis r = redis.StrictRedis(host='localhost', port=6379, db=0) r.set('foo', 'bar') -> True
r.get('foo') -> 'bar'
r.delete('foo')
after delete r.get('foo') -> None Java Client https://redislabs.com/redis-java
Redis: Docker Docker Run docker run -d -p 6379:6379 redis Docker Compose
version: "2"
services: redis: image: redis ports: - 6379:6379 Redis.io

8.5 MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
MongoDB provides high performance data persistence. In particular,
Support for embedded data models reduces I/O activity on database system. Indexes support faster queries and can include keys from embedded documents and arrays. MongoDB is 1 in the Document Store according to db-engines

Client Mongo Shell The mongo shell is an interactive JavaScript interface to MongoDB and is a component of the MongoDB package. You can use the mongo shell to query and update data as well as perform administrative operations.
Start Mongo
Once you have installed and have started MongoDB, connect the mongo shell to your running MongoDB instance. Ensure that MongoDB is running before attempting to launch the mongo shell.
mongo Interact with mongo via shell
Show list database > show dbs
Create or use a database > use <database$_name$ >> $usetestexample$
List collection > show collections
Create or use a collection > db.<collection$_name$ >> $db.new_collectionexample$
Read document > db.new$_c$$ollection.find$()
Insert new document > db.new$_c$$ollection.insertOne$("$a$" : "$b$")
Update document > db.new$_c$$ollection.update$("$a$" : "$b$", set: "a": "bcd")
Remove document > db.new$_c$$ollection.remove$("$a$" : "$b$")$PyMongo-PythonClientPyMongoisaPythondistr$
Installation We recommend using pip to install pymongo on all platforms:
pip install pymongo Usage import pymongo create connection client = pymongo.MongoClient('127.0.0.1', 27017) -> MongoClient(host=['127.0.0.1:27017'], document$_c$$lass = dict, tz_aware = False, connect = True$)
create database db = client.db$_test$$- > Database(MongoClient(host = [$'127.0.0.1 : 27017'], $document_class = dict, tz_aware = False, connect = True), u'db_test')$
create collection (collection is the same with table in SQL) collection = db.new$_c$$ollection$
insert document to collection (document is the same with rows in SQL) db.collection.insert$_one$("c" : "d")$- ><$ $pymongo.results.InsertOneResultat0x7f7eab3c9f00 >$
read document of collection db.new$_c$$ollection.find_one$("$c$" : "$d$")$- > u'_id' : ObjectId('589a8195f23e627a973c$
update documents of collection db.new$_c$$ollection.update$("$c$" : "$d$","set": "c": "def") -> u'n': 1, u'nModified': 1, u'ok': 1.0, 'updatedExisting': True
remove document of collection db.new$_c$$ollection.remove$("$c$" : "$def$")$- > u'n' : 1, u'ok' : 1.0DockerDockerRu$
docker run -p 27017:27017 mongo:latest

Chương 9

Hệ điều hành

Những phần mềm không thể thiếu

- Adblock extension
- Trình duyệt Google Chrome (với các extensions Scihub, Mendeley Desktop, Adblock)
- Terminal (Oh-my-zsh)
- IDE Pycharm để code python
- Quản lý phiên bản code Git
- Bộ gõ ibus-unikey trong Ubuntu hoặc unikey (Windows) (Ctrl-Space để chuyển đổi ngôn ngữ)
- CUDA (lập trình trên GPU)

Xem thông tin hệ thống
Phiên bản 'ubuntu 16.04'

```
sudo apt-get  install  sysstat
```

Xem hoạt động (%) của các core cpu

```
mpstat -A
```

CPU của mình có bao nhiều core, bao nhiêu siblibngs

```
cat /proc/cpuinfo

processor       : 23
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-2430 v2 @ 2.50GHz
stepping        : 4
microcode       : 0x428
cpu MHz         : 1599.707
```

```
cache size      : 15360 KB
physical id     : 1
siblings        : 12
core id         : 5
cpu cores       : 6
apicid          : 43
initial apicid  : 43
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
    ↪ mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
    ↪ pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon
    ↪ pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf
    ↪ eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2
    ↪ ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic popcnt
    ↪ tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm ida arat
    ↪ xsaveopt pln pts dtherm tpr_shadow vnmi flexpriority ept vpid
    ↪ fsgsbase smep erms
bogomips        : 5005.20
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

Kết quả cho thấy cpu của 6 core và 12 siblings

Xem chương trình nào tốn ram

```
ps aux | awk '{print $2, $4, $11}' | sort -k2rn | head -n 20
```

https://www.garron.me/en/go2linux/how-find-which-process-eating-ram-memory-linux.html

Chương 10

Ubuntu

27/12/2017: Lại dính lỗi không thể login. Lần này thì lại phải xóa bạn KDE đi. Kể cũng hơn buồn. Nhưng nhất quyết phải enable được tính năng Windows Spreading (hay đại loại thế). Hóa ra khi ubuntu bị lỗi không có lancher hay toolbar là do bạn unity plugin chưa được enable. Oài. Sao người hiền lành như mình suốt ngày bị mấy lỗi vớ vẩn thế không biết.

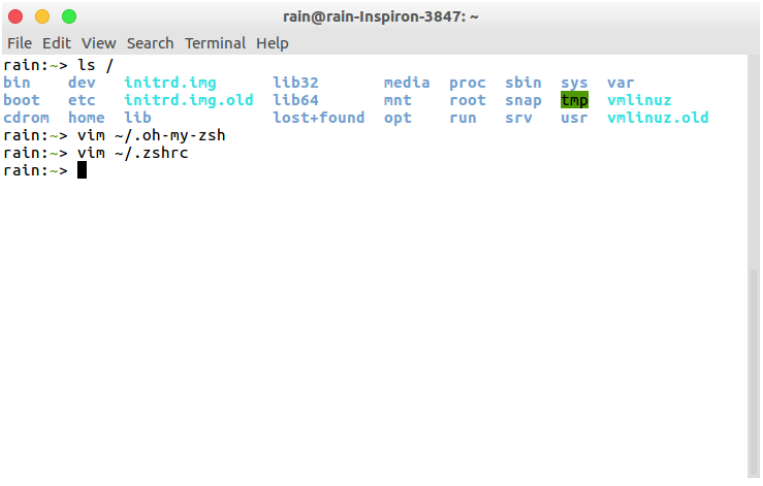
20/11/2017: Hôm nay đen thật, dính lỗi login loop. Fix mãi mới được. Thôi cũng kệ. Cảm giác bạn KDE này đỡ bị lỗi ibus-unikey hơn bạn GNOME. Hôm nay cũng đổi bạn zsh theme. Chọn mãi chẳng được bạn nào ổn ổn, nhưng không thể chịu được kiểu suggest lỗi nữa rồi. Đôi khi thấy default vẫn là tốt nhất.

21/11/2017: Sau một ngày trải nghiệm KDE, cảm giác giao diện mượt hơn GNOME. Khi overview windows với nhiều màn hình tốt và trực quan hơn. Đặc biệt là không bị lỗi ibus nữa. Đổi terminal cũng cảm giác ổn ổn. Không bị lỗi suggest nữa.

Chuyện terminal

Terminal là một câu chuyện muôn thuở của bất kì ông coder nào thích customize, đẹp, tiện (và bug kinh hoàng). Hiện tại mình đang thấy combo này khá ổn Terminal (Ubuntu) (Color: Black on white, Build-in schemes: Tango) + zsh + oh-my-zsh (fishy-custom theme). Những features hay ho

- Làm việc tốt trên cả Terminal (white background) và embedded terminal của Pycharm (black background)
- Hiển thị folder dạng ngắn (chỉ ký tự đầu tiên)
- Hiển thị brach của git ở bên phải



Chuyện bộ gõ

Làm sao để khởi động lại ibus, thỉnh thoảng lại chết bất đắc kì tử

```
ibus-daemon &
ibus  restart
```

Chuyện lỗi login loop
Phiên bản: ‘ubuntu 16.04‘
https://askubuntu.com/questions/389903/ibus-doesnt-seem-to-restart
Hot Corner và Workspace
Cần cài đặt ngay **gnome-tweak-tool**

```
sudo apt install gnome-tweak-tool
```

Cài đặt hot corner với
https://askubuntu.com/questions/975348/how-to-get-all-hot-corner-in-ubuntu-17-10
Cài đặt Workspace

Chương 11

Networking

View online http://magizbox.com/training/computer$_s$*cience*/*site*/*networking*/
TCP/IP TCP/IP is the protocol that has run the Internet for 30 years.
How TCP/IP works
Read More
Happy 30th Anniversary, Internet and TCP/IP!!! P2P Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.
Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts.[1] Peers are both suppliers and consumers of resources, in contrast to the traditional client-server model in which the consumption and supply of resources is divided. Emerging collaborative P2P systems are going beyond the era of peers doing similar things while sharing resources, and are looking for diverse peers that can bring in unique resources and capabilities to a virtual community thereby empowering it to engage in greater tasks beyond those that can be accomplished by individual peers, yet that are beneficial to all the peers.
bridge vs NAT When you create a new virtual machine, you have one of many options when it comes to choosing your network connectivity. Two common options are to use either bridged networking or network address translation (NAT). So, what exactly does that look like? Take a look at the figure below.
NAT: In this diagram, the vertical line next to the firewall represents the production network and you can see that 192.168.1.1 is the IP address of the company's firewall that connects them to the Internet. There is also a virtual host with three virtual machines running inside it. The big red circle represents the virtual adapter to which NAT-based virtual machines connect (172.16.1.1). You can see that there are two such virtual machines with IP addresses of 172.16.1.2 and 172.16.1.3. When you configure a virtual machine as using NAT, it doesn't see the production network directly. In fact, all traffic coming from the virtual machine will share the VM host's IP address. Behind the scenes, traffic from the virtual machines is routed on the virtual host and sent out via the host's physical adapter and, eventually, to the Internet.
bridge: The third virtual machine (192.168.1.3) is configured in "bridged" mode

which basically means that the virtual network adapter in that virtual machine is bridged to the production network and that virtual machine operates as if it exists directly on the production network. In fact, this virtual machine won't even be able to see the two NAT-based virtual machines since they're on different networks.
Read more: NAT vs. bridged network: A simple diagram

Chương 12

UX - UI

View online http://magizbox.com/training/computer$_s$*cience*/*site*/*ux*/
1. Design Principles UI Design Do's and Don'ts Android Design Principles
2. Design Trends 2.1 Material Design 1 components
We challenged ourselves to create a visual language for our users that synthesizes the classic principles of good design with the innovation and possibility of technology and science. This is material design. This spec is a living document that will be updated as we continue to develop the tenets and specifics of material design.
Tools
materialpalette.com Icon: fa2png UI Components
Data Binding Transclusion Directive - Fragments
Messaging Intent Android 1
Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.
Intents are objects of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.
An intent can contain data via a Bundle. This data can be used by the receiving component.
Style Theme Android Development: Explaining Styles and Themes, https://m.youtube.com//watch?v=MXp
Responsive Design Support Multi Screen 2
Intent Android
Support Multi Screen

Chương 13

Service-Oriented Architecture

View online http://magizbox.com/training/computer$_s$*cience*/*site*/*software*$_a$*rchitecture*/
A service-oriented architecture (SOA) is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology. 2 Generally accepted view 1 Boundaries are explicit Services are autonomous Services share schema and contract, not class Service compatibility is based on policy Microservices In computing, microservices is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs. These services are small building blocks, highly decoupled and focussed on doing a small task, facilitating a modular approach to system-building. One of concepts which integrates microservices as a software architecture style is dew computing. 1 Properties 2 Each running in its own process Communicating with lightweight mechanisms, often an HTTP resource API Build around business capabilities Independently deployable fully automated deployment Maybe in a different programming language and use different data storage technologies Monolith vs Microservice Monolith Microservice Simplicity Partial Deployment Consistency Availability Inter-module refactoring Preserve Modularity Multiple Platforms Benefits 4 Their small size enables developers to be most productive. It's easy to comprehend and test each service. You can correctly handle failure of any dependent service. They reduce impact of correlated failures. Web Service RESTful API
REST Client Sense (Beta)
A JSON aware developer console to ElasticSearch.
API Document and Client Generator http://swagger.io/swagger-editor/
API Client CRUD Pet
API Client Method URL Body Return Body Method GET /pets [Pet] PetApi.list()
POST /pets/ Pet Pet PetApi.create(pet) GET /pets/pet$_i$*dPetPetApi.get*(*pet*$_i$*d*)*PUT*/*pets*/*pet*$_i$*dPetPetPetA*
CRUD Store
GET /stores StoreApi.list() ... ... Relationships
Many to many

Example [https://api.facebook.com/method/links.getStats?urls=Microservices
Slide 11/42, Micro-servies
Martin Fowler, Microservices, youtube
Rick E. Osowski, Microservices in action, Part 1: Introduction to microservices, IBM developerworks

Chương 14

License

View online http://magizbox.com/training/computer$_s$*cience*/*site*/*licenses*/
Licenses More Licenses
More Open Source Licenses Choose A License Top 20 Open Source Licenses

Chương 15

Semantic Web

View online http://magizbox.com/training/semantic$_web/site/$
The Semantic Web is an extension of the Web through standards by the World Wide Web Consortium (W3C). The standards promote common data formats and exchange protocols on the Web, most fundamentally the Resource Description Framework (RDF).
According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries". The term was coined by Tim Berners-Lee for a web of data that can be processed by machines. While its critics have questioned its feasibility, proponents argue that applications in industry, biology and human sciences research have already proven the validity of the original concept.

15.1 Web 3.0

Tim Berners-Lee has described the semantic web as a component of "Web 3.0".
People keep asking what Web 3.0 is. I think maybe when you've got an overlay of scalable vector graphics – everything rippling and folding and looking misty – on Web 2.0 and access to a semantic Web integrated across a huge space of data, you'll have access to an unbelievable data resource ...
—Tim Berners-Lee, 2006
"Semantic Web" is sometimes used as a synonym for "Web 3.0", though the definition of each term varies.

15.2 RDF

15.3 SPARQL

SPARQL (pronounced "sparkle", a recursive acronym for SPARQL Protocol and RDF Query Language) is an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web. On 15 January

2008, SPARQL 1.0 became an official W3C Recommendation, and SPARQL 1.1 in March, 2013.
SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.
A SPARQL query
Anatomy of a query
SPARQL has four query forms. These query forms use the solutions from pattern matching to form result sets or RDF graphs. The query forms are:
SELECT Returns all, or a subset of, the variables bound in a query pattern match. CONSTRUCT Returns an RDF graph constructed by substituting variables in a set of triple templates. ASK Returns a boolean indicating whether a query pattern matches or not. DESCRIBE Returns an RDF graph that describes the resources found. Example
Query Result Data filename: ex008.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook>
SELECT ?person WHERE ?person ab:homeTel "(229) 276-5135" Offline query example GET CRAIG EMAILS PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> PREFIX owl: <http://www.w3.org/2002/07/owl> PREFIX xsd: <http://www.w3.org/2001/XMLSchema> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema> PREFIX : <http://www.semanticweb.org/lananh/ontologies/2016/10/untitled-ontology-3>
SELECT ?craigEmail WHERE :craig :email ?craigEmail . Online query example PREFIX ab: <http://learningsparql.com/ns/addressbook>
SELECT ?craigEmail WHERE ab:craig ab:email ?craigEmail . Query in dbpedia.org Example
SELECT * WHERE ?a ?b ?c . LIMIT 20

Phần IV

Khoa học dữ liệu

Chương 16

Học sâu

Tài liệu cũ: http://magizbox.com/training/deep_learning/site/
Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence.

24/11/2017 - Từ hôm nay, mỗi ngày sẽ ghi chú một phần (rất rất nhỏ) về Deep Learning

22/11/2017 - Phải nói quyển này hơi nặng so với mình. Nhưng thôi cứ cố gắng vậy.

16.1 Deep Feedforward Networks

deep feedforward networks

Deep feedforward networks, (hay còn gọi là feedforward neural networks) hoặc multilayer perceptrons (MLPs) là mô hình deep learning cơ bản nhất. Mục tiêu của một feedforward network là xấp xỉ một hàm $f*$. Ví dụ, với bài toán phân loại, $y = f*(x)$ chuyển đầu vào x thành một nhãn y. Một feedforward network định nghĩa một ánh xạ $y = f(x, \theta)$ và học giá trị của θ để xấp xỉ hàm $f*$ tốt nhất.

Mô hình được gọi là feedforward vì giá trị của x đang lan truyền qua mạng đến output, mà không có chiều ngược lại. Các mô hình neural có sự lan truyền ngược lại được gọi là recurrent neural network.

Feedforward là nền tảng cho rất nhiều mô hình mạng neural hiện đại. Ví dụ, các ứng dụng trong việc nhận diện ảnh thường áp dụng mô hình convolutional, là một dạng đặc biệt của feedforward network. Feedforward network cũng là nền tảng cho mạng recurrent network, có nhiều ứng trọng xử lý ngôn ngữ.

Feedforward neural networks được gọi là network vì nó thường được biểu diễn bởi một đồ thị gồm nhiều tầng, định nghĩa các hàm được kết hợp với nhau như thế nào. Ví dụ, chúng ta có 3 hàm f(1)(x), f(2)(x), f(3)(x) được đặt với nhau thành một chuỗi, hình thành hàm f(x) = f(3)(f(2)(f(1)(x))). Cấu trúc chuỗi này là dạng phổ biến nhất trong mạng neural. Trong trường hợp này f(1)(x) được gọi là layer 1, f(2)(x) được gọi là layer 2, và cứ tiếp tục như vậy. Độ dài của chuỗi thể hiện độ sâu của mô hình. Thuật ngữ "deep learning" xuất phát từ điều này. Layer cuối cũng được goi là output layer. Trong mạng neural, mục tiêu là học hàm f*(x). Ứng với mỗi giá trị đầu vào x, sẽ có tương ứng một giá trị y. Mục tiêu của mạng là tìm các tham số để y xấp xỉ với f*(x). Nhưng việc

học của hàm không hoàn toàn phụ thuộc vào x, y, mà do learning algorithm quyết định. Vì dữ liệu huấn luyện không chỉ rõ giá trị ở những layer ở giữa, nên chúng được gọi là các hidden layers Cuối cùng, mạng neural được gọi là neural vì nó lấy cảm hứng từ ngành khoa học thần kinh. Mỗi hidden layer trong mạng thường là các giá trị vector. Mỗi phần tử trong vector sẽ quyết định việc hành xử thế nào với các input đầu vào và đưa ra output.
Ta có thể coi mỗi layer chứa rất nhiều units hoạt động song song, đóng vai trò như một hàm ánh xạ vector sang giá trị số. Mặc dù các kiến trúc của mạng neural lấy rất nhiều ý tưởng từ ngành khoa học thần kinh, tuy nhiên, mục tiêu của mạng neural không phải để mô phỏng bộ não. Một cách nhìn tốt hơn là xem mạng neural như một máy học, được thiết kế với các điểm nhấn từ những gi chúng ta biết về não người. Một cách để hiểu mạng neural là bắt đầu với các mô hình linear, và xem xét các hạn chế của các mô hình này. Linear models, như logistic regression hay linear regression, hấp dẫn ở chỗ chúng có thể fit rất hữu quả và đang tin cậy, với các close form và tối ưu hóa lồi. Một hàn chế hiển nhiên của linear là các hàm linear, nên model không thể hiểu tương tác giữa các input. Để mở rộng lienar model để biểu diễn hàm nonlinear của x, chúng ta có thể áp dụng linear model không phải cho x mà cho (x), trong đó là một nonlinear transformer. Có thể xem như một tập các feature mô tả x, hoặc một cách để biểu diễn x. Vấn đề là chọn mapping . Có 3 cách thông dụng, chọn một generic như RBF, thiết kế bằng tay, hoặc học .
Nội dung chương này:

- 1. Ví dụ cơ bản củ feedforward network
- 2. Design decisions cho việc thiết kế mạng feedforward network
- 3. Choose activation functions trong hidden layer
- 4. Thiết kế mạng neural, bao nhiêu layers, các layer kết nối với nhau như thế nào, có bao nhiêu unit trong một layer
  - chosing the optimizer
  - cost function
  - form of the output units
- 5. Thuật toán Back-propagation
- 6. Góc nhìn lịch sử

Tham khảo Chapter 6, Deep Learning Book Neural Network Zoo http://www.asimovinstitute.org/neural-network-zoo/
word2vec Example Step 1: Download word2vec example from github
dir
02/06/2017 11:45 DIR . 02/06/2017 11:45 DIR .. 02/06/2017 10:12 9,144 word2vec$_b asic.py Step 2 : Run word2vec$_b asic example$
$activate tensorflow -gpu$ python word2vec$_b asic.py$
Found and verified text8.zip Data size 17005207 Most common words (+UNK) [['UNK', 418391], ('the', 1061396), ('of', 593677), ('and', 416629), ('one', 411764)] Sample data [5241, 3082, 12, 6, 195, 2, 3136, 46, 59, 156] ['anarchism', 'originated', 'as', 'a', 'term', 'of', 'abuse', 'first', 'used', 'against'] 3082 originated -> 5241 anarchism 3082 originated -> 12 as 12 as -> 6 a 12 as -> 3082 originated

6 a -> 195 term 6 a -> 12 as 195 term -> 2 of 195 term -> 6 a Initialized Average loss at step 0 : 288.173675537 Nearest to its: nasl, tinkering, derivational, yachts, emigrated, fatalism, kingston, kochi, Nearest to into: streetcars, neglecting, deutschlands, lecture, realignment, bligh, donau, medalists, Nearest to state: canterbury, exceptions, disaffection, crete, westernmost, earthly, organize, richland,

16.2 Tài liệu Deep Learning

Lang thang thế nào lại thấy trang này My Reading List for Deep Learning! của một anh ở Microsoft. Trong đó, (đương nhiên) có Deep Learning của thánh Yoshua Bengio, có một vụ hay nữa là bài review "Deep Learning" của mấy thánh Yann Lecun, Yoshua Bengio, Geoffrey Hinton trên tạp chí Nature. Ngoài ra còn có nhiều tài liệu hữu ích khác.

16.3 Các layer trong deep learning

16.3.1 Sparse Layers

nn.Embedding (hướng dẫn)
grep code: Shawn1993/cnn-text-classification-pytorch
Đóng vai trò như một lookup table, map một word với dense vector tương ứng

16.3.2 Convolution Layers

nn.Conv1d, nn.Conv2d, nn.Conv3d)
grep code: Shawn1993/cnn-text-classification-pytorch, galsang/CNN-sentence-classification-pytorch
Các tham số trong Convolution Layer
* *kernel_size* (hay là filter size)
Đối với NLP, $kernel\_size$ thường bằng $region\_size * word\_dim$ (đối với conv1d) hay $(region\_size, word\_dim)$ đối với conv2d
<small>Quá trình tạo feature map đối với region size bằng 2</small> ![](https://media.giphy.com/media/l2Q
* ‘in$_c$*hannels*‘,‘*out*$_c$*hannels*‘(*lslng*‘*featuremaps*‘)
Kênh (channels) là các cách nhìn (view) khác nhau đối với dữ liệu. Ví dụ, trong ảnh thường có 3 kênh RGB (red, green, blue), có thể áp dụng convolution giữa các kênh. Với văn bản cũng có thể có các kênh khác nhau, như khi có các kênh sử dụng các word embedding khác nhau (word2vec, GloVe), hoặc cùng một câu nhưng biểu diễn ở các ngôn ngữ khác nhau.
* ‘stride‘
Định nghĩa bước nhảy của filter.

Hình minh họa sự khác biệt giữa các feature map đối với stride=1 và stride=2. Feature map đối với stride = 1 có kích thước là 5, feature map đối với stride = 3 có kích thước là 3. Stride càng lớn thì kích thước của feature map càng nhỏ.
Trong bài báo của Kim 2014, ‘stride = 1‘ đối với ‘nn.conv2d‘ và ‘stride = word$_d$*im*‘*Øivi*‘*nn.conv*1*d*‘
Toàn bộ tham số của mạng CNN trong bài báo Kim 2014,

| Description | Values | |———————|———————| | input word vectors | Google word2vec | | filter region size | (3, 4, 5) | | feature maps | 100 | | activation function | ReLU | | pooling | 1-max pooling | | dropout rate | 0.5 | | *latexlamp*; $s =$ 22 norm constraint | 3 |
Đọc thêm:
* [Lecture 13: Convolutional Neural Networks (for NLP). CS224n-2017](http://web.stanford.edu/class/cs224n 2017-lecture13-CNNs.pdf) * [DeepNLP-models-Pytorch - 8. Convolutional Neural Networks](https://nbviewer.jupyter.org/github/DSKSD/DeepNLP-models-Pytorch/blob/master/notebooks/08.CNN-for-Text-Classification.ipynb) * [A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. Zhang 2015](https://arxiv.org/pdf/1510.03820.pdf)

16.4 Recurrent Neural Networks

16.4.1 What are RNNs?

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later). Here is what a typical RNN looks like:
A recurrent neural network and the unfolding in time of the computation involved in its forward computation
A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature The above diagram shows a RNN being unrolled (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word. The formulas that govern the computation happening in a RNN are as follows:
xtxt is the input at time step tt. For example, x1x1 could be a one-hot vector corresponding to the second word of a sentence. stst is the hidden state at time step tt. It's the "memory" of the network. stst is calculated based on the previous hidden state and the input at the current step: st=f(Uxt+Wst1)st=f(Uxt+Wst1). The function ff usually is a nonlinearity such as tanh or ReLU. s1s1, which is required to calculate the first hidden state, is typically initialized to all zeroes. otot is the output at step tt. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. ot=softmax(Vst)ot=softmax(Vst). There are a few things to note here:
You can think of the hidden state stst as the memory of the network. stst captures information about what happened in all the previous time steps. The

output at step otot is calculated solely based on the memory at time tt. As briefly mentioned above, it's a bit more complicated in practice because stst typically can't capture information from too many time steps ago. Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (UU, VV, WW above) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn. The above diagram has outputs at each time step, but depending on the task this may not be necessary. For example, when predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word. Similarly, we may not need inputs at each time step. The main feature of an RNN is its hidden state, which captures some information about a sequence.

16.4.2 What can RNNs do?

RNNs have shown great success in many NLP tasks. At this point I should mention that the most commonly used type of RNNs are LSTMs, which are much better at capturing long-term dependencies than vanilla RNNs are. But don't worry, LSTMs are essentially the same thing as the RNN we will develop in this tutorial, they just have a different way of computing the hidden state. We'll cover LSTMs in more detail in a later post. Here are some example applications of RNNs in NLP (by non means an exhaustive list).

Language Modeling and Generating Text Given a sequence of words we want to predict the probability of each word given the previous words. Language Models allow us to measure how likely a sentence is, which is an important input for Machine Translation (since high-probability sentences are typically correct). A side-effect of being able to predict the next word is that we get a generative model, which allows us to generate new text by sampling from the output probabilities. And depending on what our training data is we can generate all kinds of stuff. In Language Modeling our input is typically a sequence of words (encoded as one-hot vectors for example), and our output is the sequence of predicted words. When training the network we set ot=xt+1ot=xt+1 since we want the output at step tt to be the actual next word.

Research papers about Language Modeling and Generating Text:

Recurrent neural network based language model Extensions of Recurrent neural network based language model Generating Text with Recurrent Neural Networks

Machine Translation Machine Translation is similar to language modeling in that our input is a sequence of words in our source language (e.g. German). We want to output a sequence of words in our target language (e.g. English). A key difference is that our output only starts after we have seen the complete input, because the first word of our translated sentences may require information captured from the complete input sequence.

RNN for Machine Translation

RNN for Machine Translation. Image Source: http://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Research papers about Machine Translation:

A Recursive Recurrent Neural Network for Statistical Machine Translation Sequence to Sequence Learning with Neural Networks Joint Language and Translation Modeling with Recurrent Neural Networks Speech Recognition Given an

input sequence of acoustic signals from a sound wave, we can predict a sequence of phonetic segments together with their probabilities.
Research papers about Speech Recognition:
Towards End-to-End Speech Recognition with Recurrent Neural Networks Generating Image Descriptions Together with convolutional Neural Networks, RNNs have been used as part of a model to generate descriptions for unlabeled images. It's quite amazing how well this seems to work. The combined model even aligns the generated words with features found in the images.
Deep Visual-Semantic Alignments for Generating Image Descriptions. Source: http://cs.stanford.edu/people/karpathy/deepimagesent/

16.4.3 Training RNNs

Training a RNN is similar to training a traditional Neural Network. We also use the backpropagation algorithm, but with a little twist. Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps. For example, in order to calculate the gradient at t=4t=4 we would need to backpropagate 3 steps and sum up the gradients. This is called Backpropagation Through Time (BPTT). If this doesn't make a whole lot of sense yet, don't worry, we'll have a whole post on the gory details. For now, just be aware of the fact that vanilla RNNs trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem. There exists some machinery to deal with these problems, and certain types of RNNs (like LSTMs) were specifically designed to get around them.
RNN Extensions Over the years researchers have developed more sophisticated types of RNNs to deal with some of the shortcomings of the vanilla RNN model. We will cover them in more detail in a later post, but I want this section to serve as a brief overview so that you are familiar with the taxonomy of models.
Bidirectional RNNs are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements. For example, to predict a missing word in a sequence you want to look at both the left and the right context. Bidirectional RNNs are quite simple. They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs.
Deep (Bidirectional) RNNs are similar to Bidirectional RNNs, only that we now have multiple layers per time step. In practice this gives us a higher learning capacity (but we also need a lot of training data).
Deep Bidirectional RNN LSTM networks are quite popular these days and we briefly talked about them above. LSTMs don't have a fundamentally different architecture from RNNs, but they use a different function to compute the hidden state. The memory in LSTMs are called cells and you can think of them as black boxes that take as input the previous state ht1ht1 and current input xtxt. Internally these cells decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input. It turns out that these types of units are very efficient at capturing long-term dependencies. LSTMs can be quite confusing in the beginning but if you're interested in learning more this post has an excellent explanation.

Conclusion So far so good. I hope you've gotten a basic understanding of what RNNs are and what they can do. In the next post we'll implement a first version of our language model RNN using Python and Theano. Please leave questions in the comments!
[1] : [*UnderstandingConvolutionalNeuralNetworksforNLP*](*http* : *//www.wildml.com/*2015/11/*understo*
convolutional−neural−networks−for−nlp)[2] : [*http* : *//pytorch.org/docs/master/nn.html*](*http* :
//pytorch.org/docs/master/nn.html)

Chương 17

Xử lý ngôn ngữ tự nhiên

Bản lưu cũ http://magizbox.com/training/natural_language_processing/site/
Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages.

17.1 Introduction to Natural Language Processing

Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages.
NLP is related to the area of human–computer interaction. Many challenges in NLP involve: natural language understanding, enabling computers to derive meaning from human or natural language input; and others involve natural language generation.
The input and output of an NLP system can be either speech or written text.
Components of NLP
There are two components of NLP as given
Natural Language Understanding (NLU): this task mapping the given input in natural language into useful representations and analyzing different aspects of the language. Natural Language Generation (NLG): In the process of producing meaningful phrases and sentences in the form of natural language form some internal representation. It involves text planning retrieve the relevant content from knowledge base, sentence planning choose required words, forming meaningful phrases, setting tone of the sentence, text realization map sentence plan into sentence structure. Difficulties
Natural Language has an extremely rich form and structure. It is very ambiguous. There can be different levels of ambiguity
Lexical ambiguity: it is at very primitive level such as word-level. For example, treating the word “board” as noun or verb? Syntax level ambiguity: A sentence be parsed in different ways. For example, “He lifted the beetle with the red cap?” - did he use cap to lift the beetle or he lifted a beetle that had red cap? Referential ambiguity: referring to something using pronouns. For example, Rima went to

Gauri. She said "I am tired". - Exactly who is tired? One input can mean different meanings. Many inputs can mean the same thing.

17.2 Natural Language Processing Tasks

The analysis of natural language is broken into various board levels such as phonological, morphological, syntactic, semantic, pragmatic and discourse analysis.

Phonological Analysis Phonology is analysis of spoken language. Therefore, it deals with speech recognition and generation. The core task of speech recognition and generation system is to take an acoustic waveform as input and produce as output, a string of words. The phonology is a part of natural language analysis, which deals with it. The area of computational linguistics that deals with speech analysis is computational phonology

Example: Hans Rosling's shortest TED talk

Original Sound

0:00 / 0:52

Text X means unknown but the world is pretty known it's seven billion people have seven stones. One billion can save money to fly abroad on holiday every year. One billion can save money to keep a car or buy a car. And then three billion they save money to pay the by be a bicycle or perhaps a two-wheeler. And two billion they are busy saving money to buy shoes. In the future they will get rich and these people we move over here, these people will move over here, we will have two billion more in the world like this and the question is whether the rich people over there are prepared to be integrated in the world with 10 bilions people. Auto generated sound

0:00 / 0:36

Morphological Analysis

It is the most elementary phase of NLP. It deals with the word formation. In this phase, individual words are analyzed according to their components called "morphemes". In addition, non-word taken such as punctuation, etc. are separated from words. Morpheme is basic grammatical building block that makes words.

The study of word structure is refereed to as morphology. In natural language processing, it is done in morphological analysis. The task of breaking a word into its morphemes is called morphological parsing. A morpheme is defined as minimal meaningful unit in a language, which cannot be further broken into smaller units.

Example: word fox consists a single morpheme, as it cannot be further resolved into smaller units. Whereas word cats consists two morphemes, the morpheme "cat" and morpheme "s" indicating plurality.

Here we defined the term meaningful. Though cat can be broken in "c" and "at", but these do not relate with word "cat" in any sense. Thus word "cat" will be dealt with as minimum meaningful unit.

Morphemes are traditionally divided into two types

(i) "free morphemes", that are able to act as words in isolation (e.g., "thing", "permanent", "local") (ii) "bound morphemes", that can operate only as part of other words (e.g., "is" 'ing' etc) The morpheme, which forms the center part of the world, is also called "stem". In English, a word can be made up of one or more

morphemes, e.g., word - thing -> stem "think" word - localize -> stem "local", suffix "ize" word - denationalize -> prefix "de", stem "nation", suffix "al", "ize" The computational tool to perform morphological parsing is finite state transducer. A transducer performs it by mapping between the two sets of symbols, and a finite state transducer does it with finite automaton. A transducer normally consists of four parts: recognizer, generator, translator, and relator. The output of the transducer becomes a set of morphemes.

Lexical Analysis In this phase of natural language analysis, validity of words according to lexicon is checked. Lexicon stands for dictionary. It is a collection of all possible valid words of language along with their meaning.

In NLP, the first stage of processing input text is to scan each word in sentence and compute (or look-up) all the relevant linguistic information about that word. The lexicon provides the necessary rules and data for carrying out the first stage analysis.

The details of words, like their type (noun, verb and adverb, and other details of nouns and verb, etc.) are checked.

Lexical analysis is dividing the whole chunk of text into paragraphs, sentences, and words.

Syntactic Analysis Syntax refers to the study of formal relationships between words of sentences. In this phase the validity of a sentence according to grammar rules is checked. To perform the syntactic analysis, the knowledge of grammar and parsing is required. Grammar is formal specification of rules allowable in the language, and parsing is a method of analyzing a sentence to determine its structure according to grammar. The most common grammar used for syntactic analysis for natural languages are context free grammar (CFG) also called phase structure grammar and definite clause grammar. These grammars are described in detail in a separate actions.

Syntactic analysis is done using parsing. Two basic parsing techniques are: top-down parsing and bottom-up parsing.

Semantic Analysis In linguistics, semantic analysis is the process of relating syntactic structures, from the levels of phrases, clauses, sentences and paragraphs to the level of the writing as a whole, to their language-independent meanings. It also involves removing features specific to particular linguistic and cultural contexts, to the extent that such a project is possible.

The elements of idiom and figurative speech, being cultural, are often also converted into relatively invariant meanings in semantic analysis. Semantics, although related to pragmatics, is distinct in that the former deals with word or sentence choice in any given context, while pragmatics considers the unique or particular meaning derived from context or tone. To reiterate in different terms, semantics is about universally coded meaning, and pragmatics the meaning encoded in words that is then interpreted by an audience

Discourse Analysis The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentence.

Topics of discourse analysis include:

The various levels or dimensions of discourse, such as sounds, gestures, syntrax, the lexicon, style, rhetoric, meanings, speech acts, moves, strategies, turns, and other aspects of interaction Genres of discourse (various types of discourse in politics, the media, education, science, business, etc.) The relations between text (discourse) and context The relations between discourse and power The relations

between discourse and interaction The relations between discourse and cognition and memory Pragmatic Analysis During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.
Sentiment Analysis
MetaMind, @RichardSocher
Named Entity Recognition KDD 2015 Tutorial: Automatic Entity Recognition and Typing from Massive Text Corpora - A Phrase and Network Mining Approach
Relationship Extraction AlchemyAPI

17.3 Natural Language Processing Applications

Information Retrieval (IR) Information retrieval (IR) is the activity of obtaining information resources relevant to an information need from a collection of information resources. Searches can be based on metadata or on full-text (or other content-based) indexing.
Information Extraction (IE) Information extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents. In most of the cases this activity concerns processing human language texts by means of natural language processing (NLP).
Machine Translation Machine translation, sometimes referred to by the abbreviation MT (not to be confused with computer-aided translation, machine-aided human translation (MAHT) or interactive translation) is a sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another.
Question Answering (QA) Question answering (QA) is a computer science discipline within the fields of information retrieval and natural language processing (NLP), which is concerned with building systems that automatically answer questions posed by humans in a natural language.

17.4 Spelling Correction

For instance, we may wish to retrieve documents containing the term carrot when the user types the query carot. Google reports (http://www.google.com/jobs/britney.html) that the following are all treated as misspellings of the query britney spears: britian spears, britney's spears, brandy spears and prittany spears
We look at two steps to solving this problem: the first based on edit distance and the second based on k-gram overlap. Before getting into the algorithmic details of these methods, we first review how search engines provide spell-correction as part of a user experience.
Implementing spelling correction There are two basic principles underlying most spelling correction algorithms.
Of various alternative correct spellings for a mis-spelled query, choose the nearest one. This demands that we have a notion of nearness or proximity between a pair of queries. When two correctly spelled queries are tied (or nearly tied), select the one that is more common. For instance, grunt and grant both seem

equally plausible as corrections for grnt. Then, the algorithm should choose the more common of grunt and grant as the correction. The simplest notion of more common is to consider the number of occurrences of the term in the collection; thus if grunt occurs more often than grant, it would be the chosen correction. A different notion of more common is employed in many search engines, especially on the web. The idea is to use the correction that is most common among queries typed in by other users. The idea here is that if grunt is typed as a query more often than grant, then it is more likely that the user who typed grnt intended to type the query grunt. Corpus Birkbeck spelling error corpus
References How to Write a Spelling Corrector. Peter Norvig. 2007 Statistical Natural Language Processing in Python. Peter Norvig. 2007 Spelling correction. Introduction to Information Retrieval. 2008

17.5 Word Vectors

Discrete Representation Use a taxonomy like WordNet that has hypernyms (is-a) relationships
from nltk.corpus import wordnet as wn panda = wn.synset("panda.n.01") hyper = lambda s: s.hypernyms() list(panda.closure(hyper)) [Synset(‘procyonid.n.01’), Synset(‘carnivore.n.01’), Synset(‘placental.n.01’), Synset(‘mammal.n.01’), Synset(‘vertebrate.n.01’), Synset(‘chordate.n.01’), Synset(‘animal.n.01’), Synset(‘organism.n.01’), Synset(‘living$_thing.n.01'), Synset(‘w$
Great as resource but missing nuances, e.g. synonyms: adept, expert, good, practiced, proficient, skillful? Missing new words (impossible to keep up to date): wicked, badass, nifty, crack, ace, wizard, genius, ninjia Subjective Requires human labor to create and adapt Hard to compute accurate word similarity Word2Vec Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.
Word2vec was created by a team of researchers led by Tomas Mikolov at Google. The algorithm has been subsequently analysed and explained by other researchers. Embedding vectors created using the Word2vec algorithm have many advantages compared to earlier algorithms like Latent Semantic Analysis.
Main Idea of Word2Vec
Instead of capturing cooccurrence counts directly,, Predict surrounding words of every word Both are quite similar, see “Glove: Global Vectors for Word Representation” by Pennington et at. (2014) and Levy and Goldberg (2014)... more later. Faster and can easily incorporate a new sentence/document or add a word to the vocabulary. Detail of Word2Vec
Predict surrounding words in a window of length m of every word. Objective function: Maximize the log probability of any context word given the current cenetr word: J()=1Tt=1Tmjm,j0log p(wt+j|wt) J()=1Tt=1Tmjm,j0log p(wt+j|wt) where represents all variables we optimize
Predict surrounding words in a window of length m of every word For p(wt+j|wt)p(wt+j|wt)
the simplest first formulation is p(o|c)=exp(uTovc)Ww=1exp(uTwvc) p(o|c)=exp(uoTvc)w=1Wexp(uwTvc)

where o is the outside (or output) word id, c is the center word id, u and v are "center" and "outside" vectors of o and c
Every word has two vectors! This is essentially "dynamic" logistic regression Linear Relationships in word2vec
These representations are very good at encoding dimensions of similarity!
Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space Syntactically
xapplexapplesxcarxcarsxfamilyxfamiliesxapplexapplesxcarxcarsxfamilyxfamilies
Similarly for verb and adjective morphological forms Semantically (Semeval 2012 task 2)
xshirtxclothingxchairxfurniturexshirtxclothingxchairxfurniture xkingxmanxqueenx-womanxkingxmanxqueenxwoman GloVe Project
Highlights Training Model Overview GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.
Pre-trained Model fastText
Pre-trained word vectors for 294 languages, trained on Wikipedia using fastText. These vectors in dimension 300 were obtained using the skip-gram model described in Bojanowski et al. (2016) with default parameters.
glove
Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl
Language: English
Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, 300d vectors, 822 MB download): glove.6B.zip Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, 200d vectors, 1.42 GB download): glove.twitter.27B.zip
word2vec-GoogleNews-vectors
Language: English
Pre-trained Google News corpus (3 billion running words) word vector model (3 million 300-dimension English word vectors).
Word Analogies Test for linear relationships, examined by Mikolov et at. (2014)
Suggested Readings Simple Word Vector representations: word2vec, GloVe. cs224d.stanford.edu. Last Accessed: 2017-02-01. FastText and Gensim word embeddings. rare-technologies.com. Last Accessed: 2016-08-31. Distributed Representations of Words and Phrases and their Compositionality. papers.nips.cc. Last Accessed: 2013-12-05. Efficient Estimation of Word Representations in Vector Space. arxiv.org. Last Accessed: 2013-01-16

17.6 Conditional Random Fields in Name Entity Recognition

In this tutorial, I will write about how to using CRF++ to train your data for name entity recognition task.

Environment:
Ubuntu 14.04 Install CRF++ Download CRF++-0.58.tar.gz
Extact CRF++-0.58.tar.gz file
Navigate to the location of extracted folder through
Install CRF++ from source
./configure make sudo make install ldconfig Congratulations! CRF++ is install
crf$_learnTrainingCRFTotrainaCRFusingCRF++, youneed2things$:
A template file: where you define features to be considered for training A training data file: where you have data in CoNLL format crf$_learn-ttemplate_filetrain_data_filemodel$
crf$_learn - ttemplatetrain.txtmodelAbinaryofmodelisproduce.$
To test this model, on a testing data
crf$_test - mmodeltestfile > output.txt$
crf$_test - mmodeltest.txt > output.txtReferencesConditionalRandomFields$:
$InstallingCRF++onUbuntuConditionalRandomFieldsTrainingandTestingusingCRF++$

17.7 Entity Linking

In natural language processing, entity linking, named entity linking (NEL), named entity disambiguation (NED), named entity recognition and disambiguation (NERD) or named entity normalization (NEN) is the task of determining the identity of entities mentioned in text. More precise, it is the task of linking entity mentions to entries in a knowledge base (e.g., DBpedia, Wikipedia)
Entity linking requires a knowledge base containing the entities to which entity mentions can be linked. A popular choice for entity linking on open domain text are knowledge-bases based on Wikipedia, in which each page is regarded as a named entity. NED using Wikipedia entities has been also called wikification (see Wikify! an early entity linking system]). A knowledge base may also be induced automatically from training text or manually built.
NED is different from named entity recognition (NER) in that NER identifies the occurrence or mention of a named entity in text but it does not identify which specific entity it is
Examples Example 1:
For example, given the sentence “Paris is the capital of France”, the idea is to determine that “Paris” refers to the city of Paris and not to Paris Hilton or any other entity that could be referred as “Paris”.
Example 2:
Give the sentence “In Second Debate, Donald Trump and HIllary Clinton Spar in Bitter, Personal Terms”, the idea is to determine that “Donald Trump” refer to an American politician, and “Hillary Clinton” refer to 67th United States Secretary of State from 2009 to 2013.
Architecture
Mention detection: Identification of text snippets that can potentially be linked to entities Candidate selection: Generating a set of candidate entities for each mention Disambiguation: Selecting a single entity (or none) for each mention, based on the context Mention detection
Goal: Detect all “linkable” phrases
Challenges:

Recall oriented: Do not miss any entity that should be link Find entity name variants (e.g. “jlo” is name variant of [Jennifer Lopez]) Filter out inappropriate ones (e.g. “new york” matches >2k different entities) COMMON APPROACH Build a dictionary of entity surface forms entities with all names variants Check all document n-grams against the dictionary the value of n is set typically between 6 and 8 Filter out undesired entities Can be done here or later in the pipeline Examples
Candidate Selection
Goal: Narrow down the space of disambiguation possibilities
Balances between precision and recall (effectiveness vs. efficiency)
Often approached as ranking problem: keeping only candidates above a score/rank threshold for downstream processing.
COMMONNESS Perform the ranking of candidate entities based on their overall popularity, i.e., “most command sense”
Examples
Commonness can be pre-computed and stored in the entity surface form dictionary. Follows a power law with a long tail of extremely unlikely senses; entities at the tail end of distribution can be safely discarded (e.g., 0.001 is sensible threshold)
Disambiguation
Baseline approach: most common sense
Consider additional types of evidence: prior importance of entities and mentions, contextual similarity between the text surrounding the mention and the candidate entity, coherence among all entity linking decisions in the document.
Combine these signals: using supervised learning or graph-based approaches
Optionally perform pruning: reject low confidence or semantically meaning less annotations.
References “Entity Linking”. wikipedia “Entity Linking”. Krisztian Balog, University of Stavanger, 10th Russian Summer School in Information Retrieval. 2016 “An End-to-End Entity Linking Approach for Tweets”. Ikuya Yamada, Hideaki Takeda, Yoshiyasu Takefuji. 2015

17.8 Chatbot

28/01/2018 Hôm nay thấy khoá này hay quá Build Your Own Chatbot Cognitive Class CB0103EN

24/01/2018 Hôm qua vừa nhận kèo cà phê với CEO của Rabiloo. Thấy thú vị quá. Hôm nay hỏi anh Vũ qua qua về chatbot. Hehe

17.8.1 3 loại chatbot

Bài này là dịch từ bài của một bác ở IBM. Nóng hổi vừa thổi vừa dịch.

Chatbot hỗ trợ - Support Chatbots

Những còn này có xu hướng **nắm rõ về một lĩnh vực**, giống như các kiến thức của một công ty.
Có lẽ Rabiloo, otonhanh, và rất nhiều bot trên facebook thuộc loại này.

Chatbot chức năng - Skill chatbot

Các chatbot chức năng thường là loại **một câu lệnh**, và không cần phải quá chú ý về ngữ cảnh.
Ví dụ, nó có thể thực hiện các câu "Bật đèn lên"
Mấy con bot này có thể ở các nhà thông minh hay các bot trong công nghiệp.

Trợ lý ảo

Các trợ lý ảo có thể kết hợp của hai loại trên. Hoạt động tốt nào biết một chút kiến thức của mỗi lĩnh vực.
Một ví dụ là bạn Siri của Apple.
Kết
Không cần biết loại chatbot bạn muốn xây dựng là gì, điều quan trọng là hãy **đưa cho nó một cuộc sống, một tính cách riêng**, **khiến nó trở nên hữu ích**, và **dễ dàng sử dụng**.
Mọi người sử dụng chatbot vì họ muốn có một cách giao tiếp tự nhiên hơn so với những cách trước đó. Có thể đó là công việc đơn giản như bật một chiếc đèn, hay đó là công việc phức tạp như cho vay thế chấp. Mỗi một công việc có những đặc tính cụ thể, chắc chắn chú bot của bạn tỏa sáng với thiết kế của nó. **Các phương pháp là không thể đếm xuể**.

17.8.2 Các câu hỏi mà chatbot cần chuẩn bị

Tham khảo bài viết này 36 Questions to Ask Your Chatbot

17.8.3 Ý tưởng chatbot

Mình muốn chatbot giao tiếp bằng ngôn ngữ tự nhiên chứ không theo kiểu mệnh lệnh. *cười -> ra chuyện cười.* Thế thì không khác gì search engine.
Mà sao không có quan bot nào khai thác tâm trạng, thông tin của người dùng. Để tìm ra nhu cầu của họ nhỉ?
Vài chức năng hữu ích

- Hỏi đáp thông tin cá nhân của bot. Nhiều bot fail bởi trò đơn giản này.
- Yêu đương an ủi tán tỉnh
- Dự báo thời tiết. Lấy thông tin địa điểm người dùng. Bình luận về thời tiết hiện tại
- Tìm ảnh gái xinh, đọc truyện cười
- Khả năng học

Chán chẳng cần làm

- Tra mã số karaoke

17.8.4 Một số chatbot

Chatbot tiếng anh

1. Mitsuku (web) (2000-). Chiến thắng Loebner Prize vào năm 2016-2017
2. Rose (web). Chiến thắng Loebner Prize vào năm 2014-2015
3. Cleverbot (1997)
4. ELIZA (1964-1966) tại MIT bởi Joseph Weizenbaum
5. poncho (messenger). Con này chỉ chuyên về thời tiết
6. Melody (Baidu). Chuyên tư vấn về bác sỹ
7. Do Not Pay (2017). Chatbot về tư vấn luật
8. meditatebot (messenger) - Hướng dẫn thiền
9. bots duolingo (ios) - Bot hướng dẫn học ngoại ngữ

Nền tảng tiếng Anh

- api.ai
- DialogFlow

Cuộc thi tiếng Anh
Loebner Prize. Được tổ chức lần đầu vào năm 1990. Mục tiêu là kiểm tra xem chương trình có vượt qua được Turning Test hay không.
Chatbot tiếng việt

1. troly.bedieu (messenger) (2016) Có mấy chức năng. Tán ngẫu. Tìm ảnh gái xinh. Ổn phết
2. Bob (skype) Cũng khá vui đấy
3. Người Bạn Tốt Miki (2016). Xịt rồi
4. hana.ai. Nghe nói có vẻ khủng

5. Simsimi. Thấy bảo "con" chatbot này bậy lắm. Không biết có thật không?
6. Sumichat Hỗ trợ cả tiếng Việt đây
7. VisualFriend (2007) bởi HungCode. Nghe nói là thần thánh lắm nhưng chưa kiểm chứng.

Nền tảng tiếng việt

1. hekate.ai (2017). 66 triệu doanh nghiệp. 1.2 tỷ người. 68 tỷ message mỗi ngày
2. harafunnel.com

Cuộc thi trên nền tảng tiếng Việt
fpt.ai: bot of the year
Đề bài: Xây dựng Chatbot để nâng cao trải nghiệm của người dùng cá nhân và doanh nghiệp
Thời gian: 22/11/2017 – 30/07/2018
Đối tượng: Cá nhân và doanh nghiệp quan tâm đến lĩnh vực Chatbot trên khắp Việt Nam.

Phần V

Linh tinh

Chương 18

Nghiên cứu

01/11/2017
Không biết mình có phải làm nghiên cứu không nữa? Vừa kiêm phát triển, vừa đọc paper mỗi ngày. Thôi, cứ (miễn cưỡng) cho là nghiên cứu viên đi.

18.1 Các công cụ

18.1.1 Google Scholar & Semantic Scholar

Google Scholar vẫn là lựa chọn tốt

- Tìm kiếm tác giả theo lĩnh vực nghiên cứu và quốc gia: sử dụng filter label: + đuôi
  - ví dụ: danh sách các nhà nghiên cứu Việt Nam thuộc lĩnh vực xử lý ngôn ngữ tự nhiên
- danh sách này đã sắp xếp theo lượng trích dẫn

Bên cạnh đó còn có semanticscholar (một project của allenai) với các killer features

- Tìm kiếm các bài báo khoa học với từ khóa và filter theo năm, tên hội nghị
- Xem những người ảnh hưởng, ảnh hưởng bởi một nhà nghiên cứu, cũng như xem co-author, journals và conferences mà một nhà nghiên cứu hay gửi bài

18.1.2 Mendeley

Mendeley rất tốt cho việc quản lý và lưu trữ. Tuy nhiên điểm hạn chế lại là không lưu thông tin về citation

18.1.3 Hội nghị và tạp chí

Các hội nghị tốt về xử lý ngôn ngữ tự nhiên

- Rank A: ACL, EACL, NAACL, EMNLP, CoNLL

- Rank B: SemEval

Các tạp chí

- Computational Linguistics (CL)

18.1.4 Câu chuyện của Scihub

Sci-Hub được tạo ra vào ngày 5 tháng 9 năm 2011, do nhà nghiên cứu đến từ Kazakhstan, Alexandra Elbakyan
Hãy nghe chia sẻ của cô về sự ra đời của Sci-Hub
> Khi tôi còn là một sinh viên tại Đại học Kazakhstan, tôi không có quyền truy cập vào bất kỳ tài liệu nghiên cứu. Những bai bao tôi cần cho dự án nghiên cứu của tôi. Thanh toán 32 USD thi thât la điên rô khi bạn cần phải đọc lướt hoặc đọc hàng chục hoặc hàng trăm tơ để làm nghiên cứu. Tôi có được những bai bao nhơ vao trôm chúng. Sau đó tôi thấy có rất nhiều và rất nhiều nhà nghiên cứu (thậm chí không phải sinh viên, nhưng các nhà nghiên cứu trường đại học) giống như tôi, đặc biệt là ở các nước đang phát triển. Họ đã tạo ra các cộng đồng trực tuyến (diễn đàn) để giải quyết vấn đề này. Tôi là một thành viên tích cực trong một cộng đồng như vậy ở Nga. Ở đây ai cần có một bài nghiên cứu, nhưng không thể trả tiền cho nó, có thể đặt một yêu cầu và các thành viên khác, những người có thể có được những giấy sẽ gửi nó cho miễn phí qua email. Tôi có thể lấy bất cứ bai nao, vì vậy tôi đã giải quyết nhiều yêu cầu và người ta luôn rất biết ơn sự giúp đỡ của tôi. Sau đó, tôi tạo Sci-Hub.org, một trang web mà chỉ đơn giản là làm cho quá trình này tự động và các trang web ngay lập tức đã trở thành phổ biến.
Về phần mình, là một nhà nghiên cứu trẻ, đương nhiên phải đọc liên tục. Các báo cáo ở Việt Nam về xử lý ngôn ngữ tự nhiên thì thường không tải lên các trang mở như arxiv.org, các kỷ yếu hội nghị cũng không public các proceedings. Thật sự scihub đã giúp mình rất nhiều.
Scihub bị chặn
Vào thời điểm này (12/2017), scihub bị chặn quyết liệt. Hóng được trên page facebook của scihub các cách truy cập scihub. Đã thử các domain khác như .tw, .hk. Mọi chuyện vẫn ổn cho đến hôm nay (21/12/2017), không thể truy cập vào nữa.
Đành phải cài tor để truy cập vào scihub ở địa chỉ http://scihub22266oqcxt.onion.
Và mọi chuyện lại ổn.

18.2 Làm sao để nghiên cứu tốt

- Làm việc mỗi ngày
- Cập nhật các kết quả từ các hội nghị, tạp chí
- Viết nhật ký nghiên cứu mỗi tuần (tổng kết công việc tuần trước, các ý tưởng mới, kế hoạch tuần này)

18.3 Sách giáo khoa

Machine Learning Yearning, by Andrew Ng

18.4 Lượm lặt

Review các khóa học Deep Learning

Chương 19

Trở thành giảng viên

Hơi bị hay Ai cũng có thể trở thành giảng viên giỏi nếu của anh Cường tại techmaster. Tưởng phải mua, hóa ra lại được set free. A hihi.

- Lên ý tưởng
- Chuẩn bị nội dung
- Code mẫu
- Xây dựng kịch bản
- Trình bày đơn giản, dễ hiểu

19.1 Dạy và Học

19.1.1 Khác biệt giữa dạy online và offline

Dạy online
Ưu điểm

1. Khả năng mở rộng rất tốt
2. Chi phí thấp
3. Số lượng học viên không giới hạn

Nhược điểm

- Tỷ lệ bỏ học rất cao
- Tương tác chưa thực sự tốt

19.1.2 Làm sao để giảm tỷ lệ bỏ học trực tuyến?

- Video ngắn $<$ 10 phút
- Cấp chứng chỉ 70-90 USD lấy 1 chứng chỉ cho khóa học 4-6 tuần
- Chấm bài tập

- Facebook group kết nối giảng viên học viên
- Bài tập đủ dễ: chia nhỏ dự án khó ra
- Gamification: học như chơi
- Chịu khó email thông báo cho lớp

19.1.3 Flip Learning

Lớp học đảo ngược
Vấn đề học online

- Bỏ học cao
- Tương tác face 2 face kém
- Thực hành không có, không kiểm soát

Vấn đề học offline

- Chi phí cao
- Giao thông không thuận tiện
- Tuyển sinh khó

Flip Learning lớp học đảo ngược, kết hợp ưu điểm học trực tuyến và thực hành phòng lab.
- Khuyến khích học viên xem trước bài giảng video hướng dẫn giảng viên
- Đọc tìm hiểu thêm, trả lời trắc nghiệm qua Internet
- Tại buổi học, dành tối đa thời gian để **thực hành, hỏi đáp, hợp tác**
Khi đến lớp, học viên đã có kiến thức, biết rõ mình sẽ làm gì.
- Không thụ động, rèn tính tự học, tự đọc
- Không chống cằm nhìn giảng viên, không ghi chép
- Hỏi, làm, nói, giúp đỡ nhau
Flip Learning:
- Giảng viên phải chuẩn bị bài giảng
- Đến lớp trao đổi, hướng dẫn

19.2 Sai lầm cố hữu của giảng viên

1- Nghĩ ai cũng biết như mình.
Học viên 80% từ nông thôn, 40% thất nghiệp
Giảng viên dùng toàn từ chuyên ngành tiếng Anh mà không giải thích cặn kẽ
2- Nói nhiều, lý thuyết nhiều, ít có ví dụ, thí nghiệm minh họa. Học viên đã rất chán kiểu dạy ở đại học
3- Khô cứng: giải thích về kế thừa
4- Ba hoa, bốc phét như bán hàng đa cấp. Lesson online giới hạn 10 phút, học offline chỉ có 140 phút.
5- Dùng nhiều tính từ, đại ngôn: rất, cực.. mà không có con số.
6- Cẩu thả: không chuẩn bị kỹ slide, slide toàn chữ 12 trang.
7- Mã nguồn, dự án ví dụ vô duyên, khó hiểu

```
*
**
***
****
*****

*     *
*   *
* *
*
```

8- Chia nhỏ các bài tập

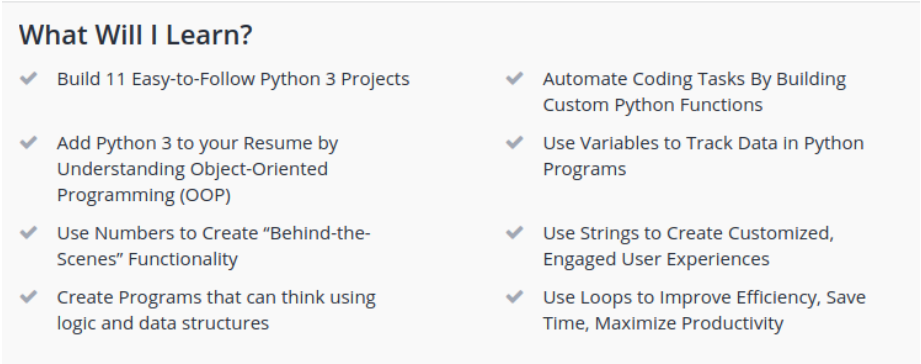
19.3 Bước 1: Xây dựng khung giáo trình

Các thông tin cơ bản của khóa học cần xác định khi thiết kế giáo trình cần có

- Mục tiêu của khóa học
- Đối tượng của khóa học
- Thời lượng của khóa học
- Học xong khóa học này, học sinh có thể làm gì?
- Các gợi ý cho khóa học tiếp theo

19.3.1 Xác định mục tiêu của khóa học

Trong phần này, có thể tham khảo sách ebook, các khóa học được đánh giá cao trên Udemy, PluralSights, Lynda
Hãy xem một mục tiêu cực kỳ hấp dẫn của một khóa học về Python rất thành công trên udemy.com



Có lẽ điểm cộng quan trọng nhất trong khóa học này là **Xây dựng 11 dự án Python 3** - học xong khóa này học viên sẽ thực hiện được rất nhiều dự án

19.3.2 Xác định đối tượng của khóa học

Khi dạy một khóa học, điều quan trọng nhất để đạt được mục tiêu học tập đó là xác định đối tượng của khóa học là ai.
Có rất nhiều đối tượng tiềm năng

- Sinh viên năm đầu: Họ rất cần kiến thức cơ bản, kinh nghiệm từ những người đi trước.
- Lập trình viên cần nâng cao kỹ năng: khi tiếp cận một công nghệ mới, lập trình viên muốn thông qua video có thể tiếp cận nhanh hơn với vấn đề

Học viên Techmaster là ai?

1. Sinh viên CNTT năm đầu, cần thực hành: 30%
2. Người thất nghiệp: 40%
3. Lập trình cần nâng cao kỹ năng 30%
4. Hầu hết là từ nông thôn 70%
5. Thời gian có hạn, cần tìm việc ngay

Họ cần gì?

1. Bài giảng thực tế, có nhiều ví dụ sinh động
2. Rất khác với ở trường đại học:
    - Không phải thi
    - Lý thuyết ít, thực hành nhiều, trừu tượng ít
    - Ví dụ phải cool
    - Có người hỗ trợ ngay
3. Trung tâm giới thiệu việc làm sau khi học

19.3.3 Thời lượng khóa học

Thời lượng khóa học được tính bằng số lượng module, số lượng video, tổng thời lượng video.
Cần xác định với đối tượng học tập hiện tại, liệu họ có thể theo hết khóa học không?

19.3.4 Khung giáo trình

Mục tiêu của bước này là xây dựng khung giáo trình cho toàn bộ khóa học.
Các điều cần ghi nhớ

- Top Down tốt hơn Bottom Up

Một công cụ hữu ích là MindMup, có thể dễ dàng tích hợp trong Google Drive
Người bình thường không thể nhớ quá 5 mục trong một buổi học:

- Offline: 1 buổi dạy 1 chủ đề chia thành 4 minitask
- Online: 1 section gồm 4-5 video, mỗi video $<$ 10 phút

Sử dụng MindMap
Tài liệu tham khảo

- Xây dựng khung giáo trình, Trịnh Minh Cường

19.4 Bước 2: Xây dựng nội dung từng module

19.4.1 Thiết kế cấu trúc từng module

Đầu và cuối mỗi module cần có một video overvew và summary.
Mỗi module nên có một sản phẩm cuối cùng
Review 1: Khoá Machine Learning của thầy Andrew Ng

- Cả khóa học gồm 11 tuần
- 1 tuần có từ 2-3 section
- Mỗi section có từ 5-9 videos
- Mỗi section có từ 3-5 bài tập lập trình

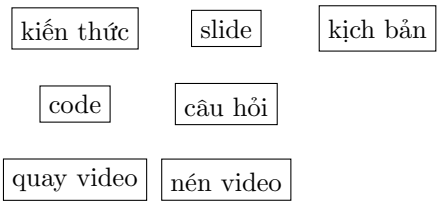
Review 2: Khoá Web cơ bản HTML5, CSS3 và Javascript của bạn Đặng Quang Huy

- Cả khóa học có 79 bài học
- Có tất cả 19 module
- Mỗi module gồm 3-9 video
- Mỗi module sẽ có một bài tập
- Mỗi video từ 5-9 phút

19.4.2 Thiết kế bài tập lập trình

Bài tập lập trình cần **giống một trò chơi**, hay là **một ứng dụng thực tế**.
Học viên sẽ thấy phấn khích và thử thách khi thấy thành quả cuối cùng.

19.5 Bước 3: Xây dựng một bài giảng pro



19.5.1 Thiết kế để học trực tuyến

1- Một video < 10 phút. Tốt nhất 3 - 5 phút
2- Ví dụ là một dự án mẫu chạy được. Đơn giản tối đa
3- Học viên phấn khích khi thấy kết quả cuối cùng
4- Chia nhỏ task top-down:
+ Sản phẩm này làm gì?
+ Có chức năng gì? chỉ nên 1 hoặc 2
+ Màn hình chính

19.5.2 Tính logic cấu trúc bài giảng

Không nên:
- Tùy tiện theo ý thích hoặc kinh nghiệm cá nhân giảng viên - Giập khuôn theo sách hoặc khóa học có sẵn Hậu quả học nhiều, nhưng kỹ năng, kiến thức lộn xộn. Thà học ít mà dùng được nhiều, hiệu quả còn hơn
Nên:
- Thiết kế top down dùng Mindmap - Các mẩu kiến thức có liên kết, sâu chuỗi
- Trước - Sau: thủ tục -> hướng đối tượng -> design pattern array -> dictionary -> linked list
SQL raw query -> Objection Relation Mapping
- Kế thừa: Cha - Con, Chị - Em
UIView > UIScollView > UITableView, UICollectionView
- Đối lập - so sánh:
Postgresql <> MongoDB, FireBase <> RethinkDB Apple Push Notification <> Google Cloud Messaging
- Phân nhóm theo tiêu chí: Dễ học, dùng thường xuyên, học viên sướng. Sướng quyết định tất cả
Ví du:
- Lẻ tẻ, minh họa cụ thể từng cú pháp, API function. Học viên không thu hoạch được nhiều - Một dự án kết hợp các bước sẽ thú vị hơn, dài hơn. Hay lúc đầu, buồn ngủ lúc sau.

19.5.3 Tạo slide

Slide ngắn gọn, xúc tích
Tự nhiên hôm nay (22/01/2018) lại đọc được bài You suck at PowerPoint, thấy hay quá.
Sau đây là 10 lỗi thường gặp khi làm bài thuyết trình

1. Quá nhiều chữ trong 1 slide.
2. Màu chữ và màu nền không tương phản với nhau.
3. Dùng clip art, word art.
4. Hình ảnh sử dụng trong slide chất lượng kém, scale sai tỉ lệ.
5. Sử dụng nhiều font chữ trong 1 slide.
6. Lạm dụng quá nhiều hiệu ứng (animation/transition).
7. Bài presentation không có cấu trúc.

8. Slide không ăn nhập gì với nội dung trình bày.
9. Không ghi rõ nguồn khi sử dụng tài liệu, hình ảnh của người khác.
10. Ý thức của người làm slide

19.5.4 Viết kịch bản nói

1- Viết chi tiết chính xác đến từng câu, rồi đọc lại + Nói chưa lưu loát > viết chi tiết 2- Liệt kê ý chính, vừa demo, vừa nói + Nói tốt + lười + Hậu kỳ phải cắt bỏ nhiều đoạn nói nhịu, ề à Câu từ dài dòng, rườm ra, tỷ lệ tiếp thu càng thấp Học trực tuyến, học viên dùng mắt đọc - xem là chính 70%, nghe là phụ, 30%. Video qua 5 phút gây buồn ngủ.
Nhắc kịch bản
- Không nên in ra giấy ! - Màn hình rộng Dell Utra, để mở cửa sổ nhắc kịch bản - Mở rộng thêm 1 màn hình mới - Gõ kịch bản ra iPad, dựng iPad lên thành màn hình phụ

19.5.5 Đặt câu hỏi cho học viên

Nếu không đặt câu hỏi:

- Học viên thụ động, buồn ngủ, sớm rời bỏ khóa học
- Giảng viên không biết học viên có hiểu bài hay không?

Đặt câu hỏi như thế nào?

1. Tránh Yes / No Question
2. Câu hỏi để thức tỉnh khả năng suy nghĩ, động não
   (a) AJAX được ai phát minh ra? : chẳng có gì là quan trọng
   (b) AJAX là viết tắt của cụm từ nào? : câu hỏi xem học viên có theo dõi video không?
   (c) Một trang web hiển thị giá cổ phiếu cập nhật 15 giây một lần. Có 4000 khách hàng cùng truy cập, vậy có bao nhiêu AJAX gửi về máy chủ trong 1 giây? : đây là câu hỏi hay
3. Câu hỏi ôn lại kiến thức
   - Ý nghĩa của http verb GET, POST, PUT, DELETE là gì, khác nhau như thế nào? Câu hỏi rất rộng, không tốt
   - Nhưng khi nào dùng POST và khi nào dùng PUT sẽ tập trung hơn và dễ tạo lựa chọn để trả lời.
   - Sử dụng GET truyền id có thể thay thế DELETE được không? Tại sao không nên.
4. Hãy nhìn vào mắt học viên khi hỏi. (offline)
   - Hỏi những học viên có dấu hiệu buồn ngủ

19.6 Bước 4.1: Quay video

19.6.1 Chú ý khi quay video

1- Camstasia 3 đã ra mắt
2- Ink2 Go
3- xScap
4- Terminal: font Roboto Mono for Powerline 18pt
5- Powerpoint 720p

19.6.2 Các lỗi giảng viên thường mắc phải khi làm video

1. Nói nhiều, thích ôm đồm nhiều thông tin trong một video > Độ dài tuyệt nhất là 3 phút
2. Chữ bé, các chi tiết khó nhìn khiến người dùng bị mỏi mắt > Thường sau 3-4 phút là khá mệt
3. Thiếu dòng title phút đầu tiên của video clip -> người dùng khó định hình ngày giảng viên muốn nói gì
4. Không điểm danh các mục nhỏ, nội dung trình bày trong video > Không biết sẽ đi về đâu
5. Không có điểm nhấn, phân cách giữa các mục nhỏ trong hướng dẫn > Mình đang ở mục mấy rồi nhỉ? Còn mấy mục nữa thì hết
6. Ít hình minh họa > Một hình ảnh dùng đúng hiệu quả hơn 1000 câu giải thích
7. Chỉ gõ lệnh nhưng ít khi nói mục đích để làm gì? Kết quả mong muốn là gì
8. Không giải thích các thuật ngữ mới
9. Chưa tính đến môi trường thực hành của học viên có thể khác so với giảng viên

19.6.3 Đồ nghề tạo giáo trình trực tuyến

1- Web Cam
+ Logitech C920 hoặc Xioami Camera loại 650,000VND. + Tỷ lệ bỏ lửng bài video giảm 30% nếu học viên thấy mặt giảng viên. + Mặt giảng viên đẹp trai, hấp dẫn, hài hước càng tốt + Mở sẵn cửa sổ ứng dụng cần demo. Giảm tối đa thời gian chết
2- Camtasia
+ Camtasia Studio trên Windows có nhiều chức năng hơn bản Camtasia Mac + Phiên bản Camtasia Mac không nén được video định dạng H264 phải dùng FFmpeg để nén + Thu ở khung hình 1280 * 720 pixels. Trên Mac có công cụ xScope để căn kích thước màn hình thu
3- Phòng thu âm cần yên tĩnh
+ Đóng chặt cửa sổ, cửa phòng khi quay chấp nhật nóng trong 10 phút quay. + Bật chế noise remove trong Camtasia
4- Sublime Text 5- Ink2Go 6- Lucid Chart 7- Adobe Photoshop 8- Skitch 9- PowerPoint 10- Slides.com

19.6.4 Kinh nghiệm khi quay video

Phòng cần yên tĩnh: nên tắt quạt hay các thiết bị điện tử gây ồn. Tắt điện thoại tránh cuộc gọi đến khi đang ghi hình Để micro xa bàn phím để không lẫn tiếng lạch cạch
Tuyệt đối không nên để background wallpaper lòe loẹt, học viên không chú ý vào demo của bạn mà xem background thì hiệu quả truyền đạt rất tệ. Tốt nhất để màn hình background là đen 100
Tắt Skype, chat, loại bỏ các icon không cần thiết trên màn hình desktop
Nên dùng web camera Logitech C920 để xa mồm để không ghi tiếng thở hổn hển của giảng viên
Nên quay cả mặt giảng viên, nhưng cut crop để không quay background của phòng thu. Hãy để học viên tập trung vào bài giảng
Không sử dụng âm thanh nên có tiếng hát hoặc giai điệu nhanh, phức tạp khiến học viên không tập trung
Lưu ý rằng:
Học viên thích xem demo sinh động và đọc chữ trên màn hình rất nhanh, đừng nói quá dài dòng, phức tạp.

19.6.5 Nén video chuẩn H.264

Áp dụng đối với hệ điều hành Mac và Linux. Sau khi ghi hình ở độ phân giải 1280x720 pixels, và xuất ra video mp4. Nếu kích thước video tính theo Megabytes $>$ 2 * số phút (Mb) thì video đó nén chưa tốt. Chúng ta cần phải nén video với chuẩn H264.
Các bước thực hiện
Vào web site này https://www.ffmpeg.org/ tải phần mềm ffmpeg về
Copy vào thư mục /usr/local/bin để có thể gọi ffmpeg ở mọi nơi Tạo một bash script có tên v264 trong thư mục /usr/local/bin cd /usr/local/bin nano v264
Paste đoạn script này vào file v264

```
#!/bin/bash
if [ -z "$1" ]; then
echo "You must input video file name"
exit
fi
if [ -f $1 ]; then
output="$(echo $1 | sed 's/\.[^.]*$//')"
ffmpeg -i $1 -c:a copy -x264-params crf=30 -b:a 64k "$output-264.mp4
    ↪ "
else
echo "File $1 does not exist"
fi
```

Để file v264 thực thi được cần gõ lệnh để bổ xung quyền execution

```
chmod +x v264
```

Khi nén video gõ lệnh này

```
v264 yourvideo.mp4
```

19.6.6 Upload video và tạo bài giảng

Tạo một lesson như thế nào?
- Có một video dài < 10 phút
- Tối thiểu 3 câu hỏi quiz
- Nên bổ sung mã nguồn hoặc slide

19.7 Bước 4.2: Thất bại khi thực hành

Mọi lỗi sơ suất đều khiến giảng viên cực kỳ bối rối, áp lực Mồ hôi toát ra như tắm, học viên xì xào, chán nản vì sốt ruột. Buổi học thất bại
Mọi demo cần chuẩn bị tập dượt, dù là chi tiết nhỏ nhất.
Thực hành tối thiểu 3 lần, ghi lại các bước thành Hand-On-Lab. Ghi lại những lỗi giảng viên đã gặp phải:
- Cắm nhầm cực dương - âm tụ hóa
- Thiếu tụ lọc nhiễu
- Máy học viên có thể bị thiếu RAM, cấu hình sai
Giảng viên phát trước Hand-On-Lab cho học viên xem. Tạo câu hỏi quiz để đảm bảo học viên hiểu bài trước khi thực hành.
Đối với những bài thực hành khó, hãy yêu cầu học viên cài sẵn môi trường thực hành bởi:
1- Thời gian tải phần mềm ở lớp học có thể lâu 2- Tự học viên phát hiện lỗi khi cài phần mềm ở nhà 3- Học viên được làm quen trước với bài lab đơn giản
Đối với bài thực hành khó, hãy mạnh dạn chọn ra 1-2 học viên giỏi để chuẩn bị trước. Đây là đặc ân mà học viên giỏi sẽ thích thú và cố gắng tham gia.

19.8 Bước 5: Học viên mất căn bản - Trả lời hỏi đáp

Question: Trong một lớp học có quá nhiều học viên không có căn bản để học, nhưng lại có vài học viên giỏi, tiếp thu rất nhanh.
Phục vụ học viên mất căn bản hay phục vụ học viên giỏi?
Trả lời: Đừng chia thành 2 nhóm học viên giỏi và học viên kém. Nhóm giỏi sẽ tiếp tục đi nhanh hơn và nhóm kém sẽ bỏ cuộc.
Chia nhóm để ghép học viên giỏi và kém. Học viên giỏi sẽ giúp đỡ học viên kém. Lớp 12 học viên chia thành 4 nhóm. Tạo ra cuộc thi đua nhỏ giữa các nhóm. Nếu có học viên bỏ học, sắp xếp lại nhóm.
Question: Trong một lớp học thực hành, nếu có nhiều câu hỏi ở mức độ khó khác nhau? Giảng viên chỉ được chọn trả lời một. Vậy chọn câu trả lời nào?
Answer: Hãy trả lời có khả năng hài lòng số đông học viên nhất.
Câu hỏi khó trả lời buổi sau, qua Facebook group của lớp
Câu hỏi ngoài lề không trả lời
Câu hỏi dễ, nhờ học viên khá trong lớp trả lời hộ

19.9 Marketing

Các hình thức marketing

1. Mạng xã hội: Facebook, forum Phải boost quảng cáo. 500000 - 2000000 VND cho mỗi khóa học
2. SEO - Google Search
3. Qua hội thảo, meetup hiệu quả thấp do số lượng người tham gia $<$ 40 người
Conversation rate: số người mua hàng / số người tham quan
Nếu đến từ organic search CR = 1 / 200
Nếu qua chia sẻ bài viết mạng xã hội CR = 1 / 4000
Nếu qua giới thiệu, tư vấn người quen CR = 1 / 2

19.9.1 Chia sẻ bài học trên Facebook

Hoàn thành bài học
Một bài viết chia sẻ lên Facebook có ảnh sẽ được người dùng click vào cao gấp 2-3 lần bài viết không có ảnh. Một bài viết có ảnh thumbnail có nút play xem video thì sẽ có số người dùng click vào cao hơn 1.3 lần so với ảnh thuần túy
Một khóa học online không giới thiệu qua Facebook, YouTube thì chắc chắn không có học viên dù giảng viên giỏi đến mấy
Bài giới thiệu trên facebook có nút play có lượt click lớn hơn nhiều so với không có nút play

19.9.2 Sử dụng Photoshop

Sử dụng Photoshop để làm poster cho khóa học

Chương 20

Nghề lập trình

Chân kinh con đường lập trình: [Teach Yourself Programming in Ten Years. Peter Norvig](http://norvig.com/21-days.html)
Trang web hữu ích
* Chia sẻ thú vị: [15 năm lập trình ở Việt Nam](https://vozforums.com/showthread.php?t=3431312) của Blanic (vozfourm) * Trang web chứa cheatsheet so sánh các ngôn ngữ lập trình và công nghệ http://hyperpolyglot.org/
01/11/2017
Vậy là đã vào nghề (đi làm full time trả lương) được 3 năm rưỡi rồi. Thời gian trôi qua nhanh như *ó chạy ngoài đồng thật. Tâm đắc nhất với câu trong một quyển gì đó của anh lead HR google. Có 4 level của nghề nghiệp. 1 là thỏa mãn được yêu cầu cả bản. 2 là dự đoán được tương lai. 3 là cá nhân hóa (ý nói là tận tình với các khách hàng). 4 là phiêu diêu tự tại. Hay thật! Bao giờ mới được vào mức 4 đây.

Chương 21

Latex

15/12/2017:
Hôm nay tự nhiên nổi hứng vẽ hình trên latex. Thấy blog này là một guide line khá tốt về viết blog phần mềm. Quyết định cài latex
Theo [hướng dẫn này](http://milq.github.io/install-latex-ubuntu-debian/)
“‘ sudo apt-get install texlive-full sudo apt-get install texmaker “‘
Tìm được ngay bên này https://www.overleaf.com/ có vẻ rất hay luôn
Hướng dẫn cực kì cơ bản http://www.math.uni-leipzig.de/ hellmund/LaTeX/pgf-tut.pdf
Chương trình đầu tiên, vẽ diagram cho LanguageFlow

```
\documentclass[border=10pt]{standalone}
\usepackage{verbatim}
\begin{comment}
\end{comment}
\usepackage{tikz}
\begin{document}
\begin{tikzpicture}
    \node[draw] (model) at (0, 0) {Model Folder};
    \node[draw] (analyze) at (6, 0) {Analyze Folder};
    \node[draw] (board) at (3,2) {Board};
    \node[draw] (logger) at (3, -2) {Logger};

    \path[->, densely dotted] (board.east)
        edge [out=0, in=90]
        node[ fill =white, pos=.5] {\tiny (1) init }
        (analyze.north) ;
    \path[->, densely dotted] (board.south)
        edge [out=-90, in=180]
        node[ fill =white, pos=.3] {\tiny (2) serve}
        (analyze.west) ;
        \path[->, densely dotted] (logger.west)
        edge [out=180, in=-90]
        node[ fill =white, pos=.7] {\tiny (1) read}
        (model.south) ;
        \path[->, densely dotted] (logger.east)
```

```
        edge [out=0, in=-90]
        node[ fill =white, pos=.7] {\tiny (2) write}
        (analyze.south) ;
\end{tikzpicture}
\end{document}
```

Doc! Doc! Doc! https://en.wikibooks.org/wiki/LaTeX/PGF/TikZ

Chương 22

Chào hàng

16/01/2018 Bố khỉ. Hôm nay gửi lời mời kết bạn đến một thằng làm research về speech mà nó "chửi" mình không biết pitch. Tổ sư. Tuy nhiên, nó cũng dạy mình một bài học hay về pitch.
Chửi nó là vậy nhưng lần sau sẽ phải đầu tư nhiều hơn cho các lời pitch.
Vẫn không ưa Huyền Chíp như ngày nào, nhưng [bài này](https://www.facebook.com/notes/huyen-chip/k
Tóm lại skill này có 4 phần
1. Ngôn ngữ không trau chuốt 2. Giới thiệu bản thân không tốt 3. Không chỉ ra cho người nhận rằng họ sẽ được gì 4. Không có phương án hành động
Đối với email, thì cần triển khai thế này
* [Chào hỏi] * [Giới thiệu bản thân một cách nào đó để người đọc quan tâm đến bạn] * [Giải thích lý do bạn biết đến người này và bạn ấn tượng thế nào với họ – ai cũng thích được nghe khen] * [Bạn muốn gì từ người đó và họ sẽ được gì từ việc này] * [Kết thúc]

Chương 23

Phát triển phần mềm

* Phát triển phần mềm là một việc đau khổ. Từ việc quản lý code và version, packing, documentation. Dưới đây là lượm lặt những nguyên tắc cơ bản của mình.
Quản lý phiên bản
Việc đánh số phiên bản các thay đổi của phần mềm khi có hàm được thêm, lỗi được sửa, hay các phiên bản tiền phát hành cần thống nhất theo chuẩn của [semversion]. Điều này giúp nhóm có thể tương tác dễ hơn với người dùng cuối.
![](https://raw.githubusercontent.com/magizbox/magizbox/master/wordpress/phat$_t rien_p han_m em/version.$
Đánh số phiên bản
Phiên bản được đánh theo chuẩn của [semversion](https://semver.org/).
* Mỗi khi một bug được sửa, phiên bản sẽ tăng lên một patch * Mỗi khi có một hàm mới được thêm, phiên bản sẽ tăng lên một patch. * Khi một phiên bản mới được phát hành, phiên bản sẽ tăng lên một minor. * Trước khi phát hành, bắt đầu với x.y.z-rc, x.y.z-rc.1, x.y.z-rc.2. Cuối cùng mới là x.y.z * Mỗi khi phiên bản rc lỗi, khi public lại, đặt phiên bản alpha x.y.z-alpha.t (một phương án tốt hơn là cài đặt thông qua github)
Đánh số phiên bản trên git
Ở nhánh develop, mỗi lần merge sẽ được đánh version theo PATCH, thể hiện một bug được sửa hoặc một thay đổi của hàm
Ở nhánh master, mỗi lần release sẽ được thêm các chỉ như x.y1.0-rc, x.y1.0-rc.1, x.y1.0-rc, x.y1.0
Vẫn còn lăn tăn:
* Hiện tại theo workflow này thì chưa cần sử dụng alpha, beta (chắc là khi đó đã có lượt người sử dụng mới cần đến những phiên bản như thế này)
Tải phần mềm lên pypi
Làm theo hướng dẫn [tại đây](http://peterdowns.com/posts/first-time-with-pypi.html)
1. Cấu hình file '.pypirc' 2. Upload lên pypi
"' python setup.py sdist upload -r pypi "'

Chương 24

Phương pháp làm việc

Xây dựng phương pháp làm việc là một điều không đơn giản. Với kinh nghiệm 3 năm làm việc, trải qua 2 project. Mà vẫn chưa produce được sản phẩm cho khách hàng. Thiết nghĩ mình nên viết phương pháp làm việc ra để xem xét lại. Có lẽ sẽ có ích cho mọi người.
Làm sao để làm việc hiệu quả, hay xây dựng phương pháp làm việc hữu ích? Câu trả lời ngắn gọn là "Một công cụ không bao giờ đủ".
<!–more–>
Nội dung
1. [Làm sao để đánh giá công việc trong khoảng thời gian dài hạn?](section1) 2. [Làm sao để quản lý project?](section2) 3. [Làm sao để công việc trôi chảy?](section3) 4. [Làm sao để xem xét lại quá trình làm việc?](section4)
nbsp;
Làm sao để đánh giá công việc trong khoảng thời gian dài hạn?
Câu trả lời OKR (Objectives and Key Results)
 *OKR Framework*
Dầu mỗi quý , nên dành vài ngày cho việc xây dựng mục tiêu và những kết quả quan trọng cho quý tới. Cũng như review lại kết quả quý trước.
Bước 1: Xây dựng mục tiêu cá nhân (Objectives)
Bước 2: Xây dựng các Key Results cho mục tiêu này
Bước 3: Lên kế hoạch để hiện thực hóa các Key Results
nbsp;
Làm sao để quản lý một project
Meistertask
 $*MeisterTask*$
nbsp;
Làm sao để công việc trôi chảy?
Có vẻ trello là công cụ thích hợp
Bước 1: Tạo một team với một cái tên thật ấn tượng (của mình là Strong Coder)
Trong phần Description của team, nên viết Objectives and Key Results của quý này
Sau đây là một ví dụ
“‘ Objectives and Key Results

-> Build Vietnamese Sentiment Analysis -> Develop underthesea -> Deep Learning Book “‘
Bước 2: Đầu mỗi tuần, tạo một board với tên là thời gian ứng với tuần đó (của mình là ‘2017 | Fight 02 (11/12 - 16/12)‘)
Board này sẽ gồm 5 mục: "TODO", "PROGRESSING", "Early Fight", "Late Fight", "HABBIT", được lấy cảm hứng từ Kanban Board
*
$TrelloBoardexample*$
* Mỗi khi không có việc gì làm, xem xét card trong "TODO" * [FOCUS] tập trung làm việc trong "PROGRESSING" * Xem xét lại thói quen làm việc với "HABBIT"
Một Card cho Trello cần có
* Tên công việc (title) * Độ quan trọng (thể hiện ở label xanh (chưa quan trọng), vàng (bình thường), đỏ (quan trọng)) * Hạn chót của công việc (due date)
Sắp xếp TODO theo thứ tự độ quan trọng và Due date
nbsp;
Làm sao để xem xét lại quá trình làm việc?
Nhật lý làm việc hàng tuần . Việc này lên được thực hiện vào đầu tuần . Có 3 nội dung quan trọng trong nhật ký làm việc (ngoài gió mây trăng cảm xúc, quan hệ với đồng nghiệp...)
* Kết quả công việc tuần này * Những công việc chưa làm? Lý do tại sao chưa hoàn thành? * Dự định cho tuần tới
Đang nghiên cứu
Làm sao để lưu lại các ý tưởng, công việc cần làm?: Dùng chức năng checklist của card trong meister. Khi có ý tưởng mới, sẽ thêm một mục trong checklist
Làm sao để tập trung vào công việc quan trọng?: Dùng chức năng tag của meister, mỗi một công việc sẽ được đánh sao (với các mức 5 sao, 3 sao, 1 sao), thể hiện mức độ quan trọng của công việc. Mỗi một sprint nên chỉ tập trung vào 10 star, một product backlog chỉ nên có 30 star.
Tài liệu của dự án: Sử dụng Google Drive, tài liệu mô tả dự án sẽ được link vào card tương ứng trong meister.

Chương 25

Làm sao để thành công?

Vòng tròn thành công gồm có 3 phần : ĐAM MÊ, NĂNG LỰC, LỢI NHUẬN

27/01/2018: Hôm nay xem Shark Tank Việt Nam có vụ nói về các yếu tố để thành công hay quá.

ĐAM MÊ: là cái mình yêu thích làm

NĂNG LỰC: là cái mình làm tốt

LỢI NHUẬN: là cái đem lại giá trị cho người khác

Chương 26

Đặt tên email

Email hiện tại là anhv.ict91@gmail.com. Chắc tốt hơn là email brother.rain.1024@gmail.com. Vụ này đã được anh Lê Hồng Phương nhắc một lần rồi.

Chương 27

Agile

View online http://magizbox.com/training/agile/site/

27.1 Introduction

What is Agile? 2 Agile methodology is an alternative to traditional project management, typically used in software development. It helps teams respond to unpredictability through incremental, iterative work cadences, known as sprints. Agile methodologies are an alternative to waterfall, or traditional sequential development.
Agile Manifesto
Scrum Framework
A manifesto for small teams doing important work 4
What is the difference between Scrum and Agile Development? 1 Scrum is just one of the many iterative and incremental agile software development method. You can find here a very detailed description of the process.
In the SCRUM methodology a sprint is the basic unit of development. Each sprint starts with a planning meeting, where the tasks for the sprint are identified and an estimated commitment for the sprint goal is made. A Sprint ends with a review or retrospective meeting where the progress is reviewed and lessons for the next sprint are identified. During each sprint, the team creates finished portions of a product.
In the Agile methods each iteration involves a team working through a full software development cycle, including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders.
Agile Stories and Teasers 2011, PRESENTATION: HOW OUR TEAM LIVES SCRUM 2010, The real life of a Scrum team – with photos 2009, How Scrum Helped Our Team Agile Tools 3 Agilefant (4/ 3/ 3) What is the difference between Scrum and Agile Development?
Agile Methodology
agile-tools.net/
A manifesto for small teams doing important work

27.2 Team

Scrum Team 1 Within the Scrum Framework three roles are defined:
Product Owner Scrum Master Development team Each of these roles has a defined set of responsibilities and only if they fulfill these responsibilities, closely interact and work together they can finish a project successfully.
Scrum Roles Stakeholders
Product Owner One of the most important things for the success of scrum is the role of the Product Owner, who serves as an interface between the team and other involved parties (stakeholders). It can be said that in companies that use scrum, the tasks and responsibilities of the particular Product Owner are never the same. Starting with the choice of that person provided with the proper and necessary skills, make them take specific trainings, up to the responsibility they take; the role of the Product Owner –short PO- is the most complex one regarding that procedure.
Often the PO has to "fight" on both sides. Whereas the team can work a certain fraction of time (time boxed) "protected" by the Scrum Master, the Product Owner often needs to deal with marketing, management or the customers in order to be able to present the software requirements (User Stories) quite precisely to the team (see the box "criteria for User Stories).
Furthermore the Product Owner is responsible for the return on investment (ROI). He validates the solutions and verifies whether the quality is acceptable or not from the end-users' point of view. He also has to decide over the importance of single features in order to prioritize these in their treatment and he has to tell the team what the product should look like in the end (product vision). Since one of the teams' tasks is to work effectively, the Product Owner must react fast on call-backs. Hence he fulfills the role of a communicator, as he must be in contact with all stakeholders, sponsors and last but not least the team throughout a project. After all it is his task to coordinate the financial side of the product development, which is successful through his continuous work and prioritizing the advancing tasks (Product Backlog). All these diverse requests demonstrate how important the selection of the "right" person for the role of the PO is for the success of a project.
The nomenclature is definite. The Product Owner is at Scrum not only the manager of a product, but also the Owner and therefore he is the one responsible for the correct creation of a product. Being a Product Owner means:
You are responsible for the success of the outcome of the product delivered by the team. You make Business decisions of importance and priorities. You deliver the vision of the product to the team. You prepare the User Stories for the team of development. You should possess severe domain knowledge. You validate the outcomes and test them for their quality. You react promptly on callbacks. You communicate on a continual base with all Stakeholders, financiers and the team. You control the financial side of the project.
Scrum Master The most obvious difference between a team leader and a Scrum Master is represented by the name itself though. Whereas one is leading the team and sets the tasks, the other one is in charge of observing that the team obeys the rules and realizes the method of Scrum entirely. The Scrum Master does not interfere into the decisions of the team regarding specifically the development, but rather is there for the team as an advisor. He only interferes actively when anybody of the team or any other participant of a project (Stakeholder) does not

obey the rules of Scrum. Whereas a team leader often gives requirements and takes responsibility for the completion of those, an experienced Scrum Master gives only impulses and advises to the team to lead the correct way, to use the right method or to choose the right technology. In fine the Scrum Master acts more like a Team Coach than a team leader.
ScrumMaster and Impediments Another important task of the Scrum Master is to get rid of all possible impediments that might disturb the work of the team. Usually problems can be classified in three different categories. The first one is problems the team cannot solve. E.g. the team cannot do any kind of performance-tests because the hardware is not in place, the IT-department does not provide Bug tracker, or the ordered software just still did not reach the team. Another impediment could be that the marketing or sales manager was there again demanding that another feature gets integrated "quickly".
The second one regards impediments that result through the organizational structure or strategic decisions. Maybe the office is not capable of handling the important meetings or teamwork – e.g. because there is no media. One mistake that occurs quite often regards the problem that the Scrum Master is seen as the personnel responsible for the team members. This is often because of the classical role of a project leader, but using Scrum it only leads to conflicts of interests and is strongly against its major principle: The team owns a management role in the method of Scrum and is therefore coequal with the Scrum Master and the Product Owner. Another aspect can be the insufficient bandwidth of the internet for the new project.
The third problem refers to the individuals. Someone needs a hand with the debugging. Another one cannot solve a task alone and needs someone else for the pair programming. Someone else has to reset his computer....
Even though a Scrum Master can't and shouldn't realize some requirements himself, he is still responsible for solving and getting rid of problems and needs to give proper criteria. This task often takes up a lot of time and requires great authority and backbone. The Scrum Master has to create an optimal working-condition for the team and is responsible for this condition to be retained, in order to meet the goals of every sprint – i.e. for a short sprint the defined requirements.
Scrum Development Team Different from other methods, in Scrum a team is not just the executive organ that receives its tasks from the project leader, it rather decides self dependent, which requirements or User Stories it can accomplish in one sprint. It constructs the tasks and is responsible for the permutation of those – the team becomes a manager. This new self-conception of the team and the therewith aligned tasks and responsibilities necessarily change the role of the team leader/project leader. The Scrum Master does not need to delegate all the work and to plan the project, he rather takes care that the team meets all conditions in order to reach the self-made goals. He cleans off any impediments, provides an ideal working environment for the team, coaches and is responsible for the observation of Scrum-rules – he becomes the so-called Servant Leader.
The changed role perception is one of the most important aspects, when someone wants to understand Scrum and with the intent to introduce it in their own company.
Scrum Roles

27.3 Artifacts

Product Backlog 1
Force-ranked list of desired functionality Visible to all stakeholders Any stakeholder (including the Team) can add items Constantly re-prioritized by the Product Owner Items at top are more granular than items at bottom Maintained during the Backlog Refinement Meeting
Product Backlog Item (PBI) 1
Specifies the what more than the how of a customer-centric feature Often written in User Story form Has a product-wide definition of done to prevent technical debt May have item-specific acceptance criteria Effort is estimated by the team, ideally in relative units (e.g., story points) Effort is roughly 2-3 people 2-3 days, or smaller for advanced teams Sprint Backlog 1
Consists of committed PBIs negotiated between the team and the Product Owner during the Sprint Planning Meeting
Scope commitment is fixed during Sprint Execution
Initial tasks are identified by the team during Sprint Planning Meeting
Team will discover additional tasks needed to meet the fixed scope commitment during Sprint execution
Visible to the team
Referenced during the Daily Scrum Meeting
Sprint Task 1
Specifies how to achieve the PBI's what
Requires one day or less of work
Remaining effort is re-estimated daily, typically in hours
During Sprint Execution, a point person may volunteer to be primarily responsible for a task
Owned by the entire team; collaboration is expected
Personal Sprint Board with Sticky Notes (Windows)
Sprint Burndown Chart 1
Indicates total remaining team task hours within one Sprint Re-estimated daily, thus may go up before going down Intended to facilitate team self-organization Fancy variations, such as itemizing by point person or adding trend lines, tend to reduce effectiveness at encouraging collaboration Seemed like a good idea in the early days of Scrum, but in practice has often been misused as a management report, inviting intervention. The ScrumMaster should discontinue use of this chart if it becomes an impediment to team self-organization.
Product / Release Burndown Chart
Tracks the remaining Product Backlog effort from one Sprint to the next May use relative units such as Story Points for Y axis Depicts historical trends to adjust forecasts scrum-reference-card

27.4 Meetings

Meetings Scrum Meetings 1 Sprint Planning Daily Scrum Sprint Review Sprint Retrospective Product Backlog Refinement
Sprint Planning 1
Goals – Discuss and make sure the whole team understands the upcoming Work Items to deliver (quantity, complexity) – Select the work items to achieve during

the Sprint (Create the Sprint Backlog) – Rationally forecast the amount of work and commit to accomplish it – Plan how this work will be done
Attendees 1st part (what?): Whole team 2nd part (how?): Even though the product owner does not attend, he should remain to answer questions. Duration The meeting is time-boxed: 2 hours / week of sprint duration.
Typical meeting roadmap In Scrum, the sprint planning meeting has two parts:
1. What work will be done?
– the Product Owner presents the ordered product backlog items to the development team – the whole Scrum team collaborates to understand the work and select work items starting from the top of the product backlog
2. How will the work be accomplished?
– the development team discusses to define how to produce the next product increment in accordance with the current Definition of Done – The team does just the sufficient design and planning to be confident of completing the work during the sprint – The upcoming work to be done in the early days is broken down into small tasks of one day or less – Work to be done later may be left in larger units to be decomposed later (this is called Just-In-Time planning in Lean) – Final commitment to do the work
Important to know / Good practices – The development team is alone responsible to determine the number of product backlog items that will be "pulled" to the sprint. Nobody else should interfere in that decision, based on the current state of the project, the past performance and the current availability of the team. – It is a good practice to set a sprint goal to keep focus on the big picture and not on the details. – It is necessary for the Sprint planning meeting success that the product backlog is well ordered and refined. This is the Product Owner's responsibility. – The development team is responsible to decide how to do the work (self-organization). – There is no point, especially at the beginning, to try to make hourly estimates of the work and compare them to availability. This habit is inherited from traditional PM methods and may be counterproductive, as reduces ownership of the team's commitments. The best for the team is to intuitively estimate its own capacity to do the work, reduce the amount of committed work to deliver, and get experienced at estimating during the first sprints.
Daily Scrum 1
Goals Ensure, every day, at the same palace, that the team is on track for attaining the sprint goal and that team members are all on the same page. Spot blocks and problems. The Scrum Master can resolve impediments that are beyond the team's control. This is NOT a management reporting meeting to anyone. The team members communicate together as a team. Based on what comes up in the meeting, the development team reorganizes the work as needed to accomplish the sprint goal. Create transparency, trust and better performance. Build the team's self-organization and self-reliance. Attendees Development Team + Scrum Master. The Product Owner presence is not required but it I almost always interesting for him / here to be present, especially to clarify requirements if needed. Other stakeholders can attend to get a good and quick overview of the progress and project status, although having managerial presence may cause a "trigger" effect and pollute the meeting's effectiveness. Duration No more than 15 minutes. A good practice is to allow 2 minutes to each team member. Typical meeting roadmap Every team member answers 3 questions:
What I have accomplished since the last daily Scrum What I plan to accomplish

between now and the next daily Scrum What (if anything) is impeding my progress No discussion during the meeting. Only brief questions to clarify the previous list. Any subsequent discussion should take place after the meeting with the concerned team members.
Important to know / Good practices The Daily Scrum is sometimes called "Daily Stand Up". This name gives a good overview of the tone and shortness of this meeting. Each team member moves the tasks in the taskboard while speaking (if not done before) The Sprint Burndown chart can be updated by the Scrum Master at the end of the meeting Having a "being blocked" task list is useful. Personally I add a dedicated column in the taskboard Only team members speak during the daily Scrum. Nobody else. TheDaily Scrum is a proof of the team's sef-organizing capacity as it shows how much team members collaborate together as they address themselves to the whole team (and not only the Scrum Master) It is quite common to uncover additional tasks during the Sprint to achieve the Sprint Goal Sprint Review Meeting 1
Goals After Sprint execution, the team holds a Sprint Review Meeting to demonstrate a working product increment to the Product Owner and everyone else who is interested. The meeting should feature a live demonstration, not a report. Attendees Product Team Product Owner When After Sprint execution Duration 4 hours given a 30-day Sprint (much longer than anyone recommends nowadays), the maximum time for a Sprint Review Meeting is 4 hours Roadmap Demonstration After the demonstration, the Product Owner reviews the commitments made at the Sprint Planning Meeting and declares which items he now considers done. For example, a software item that is merely "code complete" is considered not done, because untested software isn't shippable. Incomplete items are returned to the Product Backlog and ranked according to the Product Owner's revised priorities as candidates for future Sprints. The ScrumMaster helps the Product Owner and stakeholders convert their feedback to new Product Backlog Items for prioritization by the Product Owner. Often, new scope discovery outpaces the team's rate of development. If the Product Owner feels that the newly discovered scope is more important than the original expectations, new scope displaces old scope in the Product Backlog. The Sprint Review Meeting is the appropriate meeting for external stakeholders (even end users) to attend. It is the opportunity to inspect and adapt the product as it emerges, and iteratively refine everyone's understanding of the requirements. New products, particularly software products, are hard to visualize in a vacuum. Many customers need to be able to react to a piece of functioning software to discover what they will actually want. Iterative development, a value-driven approach, allows the creation of products that couldn't have been specified up front in a plan-driven approach.
Sprint Retrospective Meeting 1 2
Goals At this meeting, the team reflects on its own process. They inspect their behavior and take action to adapt it for future Sprints.
When Each Sprint ends with a retrospective.
Duration 45 minutes
Roadmap
Dedicated ScrumMasters will find alternatives to the stale, fearful meetings everyone has come to expect. An in-depth retrospective requires an environment of psychological safety not found in most organizations. Without safety, the retrospective discussion will either avoid the uncomfortable issues or deteriorate

into blaming and hostility.
A common impediment to full transparency on the team is the presence of people who conduct performance appraisals.
Another impediment to an insightful retrospective is the human tendency to jump to conclusions and propose actions too quickly. Agile Retrospectives, the most popular book on this topic, describes a series of steps to slow this process down: Set the stage, gather data, generate insights, decide what to do, close the retrospective. (1) Another guide recommended for ScrumMasters, The Art of Focused Conversations, breaks the process into similar steps: Objective, reflective, interpretive, and decisional (ORID). (2)
A third impediment to psychological safety is geographic distribution. Geographically dispersed teams usually do not collaborate as well as those in team rooms.
Retrospectives often expose organizational impediments. Once a team has resolved the impediments within its immediate influence, the ScrumMaster should work to expand that influence, chipping away at the organizational impediments.
ScrumMasters should use a variety of techniques to facilitate retrospectives, including silent writing, timelines, and satisfaction histograms. In all cases, the goals are to gain a common understanding of multiple perspectives and to develop actions that will take the team to the next level.
Product Backlog Refinement Meeting 1
How to: A Great Product Backlog Refinement Workshop
PRESENTATION: HOW OUR TEAM LIVES SCRUM

Chương 28

Docker

View online http://magizbox.com/training/docker/site/
Docker is an open platform for building, shipping and running distributed applications. It gives programmers, development teams and operations engineers the common toolbox they need to take advantage of the distributed and networked nature of modern applications.

28.1 Get Started

Docker Promise Docker provides an integrated technology suite that enables development and IT operations teams to build, ship, and run distributed applications anywhere.
Build: Docker allows you to compose your application from microservices, without worrying about inconsistencies between development and production environments, and without locking into any platform or language.
Composable You want to be able to split up your application into multiple services.
Ship: Docker lets you design the entire cycle of application development, testing and distribution, and manage it with a consistent user interface.
Portable across providers You want to be able to move your application between different cloud providers and your own servers, or run it across several providers.
Run: Docker offers you the ability to deploy scalable services, securely and reliably, on a wide variety of platforms.
Portable across environments You want to be able to define how your application will run in development, and then run it seamlessly in testing, staging and production.
1. Docker Architecture
Docker Containers vs Virtual Hypervisor Model
In the Docker container model, the Docker engine sits atop a single host operating system. In contrast, with the traditional virtualization hypervisor mode, a separate guest operating system is required for each virtual machine.
Docker Images and Docker Containers
2.1 Docker Images Docker images are the build component of Docker.
For example, an image could contain an Ubuntu operating system with Apache and your web application installed. Images are used to create Docker containers.

Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created.
Built by you or other Docker users
Stored in the Docker Hub or your local Registry
Images Tags
Images are specified by repository:tag The same image my have multiple tags The default tag is lastest Look up the repository on Docker to see what tags are available 2.2 Docker Containers Docker containers are the run component of Docker.
Docker containers are similar to a directory. A Docker container holds everything that is needed for an application to run. Each container is created from a Docker image. Docker containers can be run, started, stopped, moved, and deleted. Each container is an isolated and secure application platform.
2.3 Registry and Repository Docker registries are the distribution component of Docker.
Docker registries
Docker registries hold images. These are public or private stores from which you upload or download images. The public Docker registry is provided with the Docker Hub. It serves a huge collection of existing images for your use. These can be images you create yourself or you can use images that others have previously created. Docker Hub
Docker Hub is the public registry that contains a large number of images available for your use.
2.3 Docker Networking Docker use DNS instead etc/hosts 3. Docker Orchestration Three tools for orchestrating distributed applications with Docker Docker Machine Tool that provisions Docker hosts and installs the Docker Engine on them Docker Swarm Tool that clusters many Engines and schedules containers Docker Compose Tool to create and manage multi-container applications Installation Docker only supports CentOS 7, Windows 7.1, 8/8.1, Mac OSX 10.8 64bit
Windows, CentOS
Usage Lab: Search for Images on Docker Hub Installation: Ubuntu Installation ubuntu 14.04
Prerequisites
apt-get update apt-get install apt-transport-https ca-certificates
apt-key adv –keyserver hkp://p80.pool.sks-keyservers.net:80 –recv-keys 58118E89F3A912897C070ADBF7622
Edit /etc/apt/sources.list.d/docker.list
deb https://apt.dockerproject.org/repo ubuntu-trusty main apt-get update apt-get purge lxc-docker apt-cache policy docker-engine Install docker
apt-get update apt-get install -y –force-yes docker-engine service docker start
Run Docker
docker run hello-world docker info docker version Installation https://www.docker.com/products/docker-toolbox
Go to the Docker Toolbox page. Click the installer link to download. Install Docker Toolbox by double-clicking the installer. Docker: CentOS 7 Installation
Install Docker
make sure your existing yum packages are up-to-date. sudo yum -y update
add docker repo sudo tee /etc/yum.repos.d/docker.repo «-'EOF' [dockerrepo]
name=Docker Repository baseurl=https://yum.dockerproject.org/repo/main/centos/$releasever/enabled = 1$gpgcheck = 1$gpgkey = https : //yum.dockerproject.org/gpgEOF

install the Docker package. sudo yum install -y docker-engine
start the Docker daemon sudo service docker start
verify sudo docker run hello-world Install Docker Compose
sudo yum install epel-release sudo yum install -y python-pip
sudo pip install docker-compose Docker 1.10.3
Kitematic Port Forwarding Open Virtualbox
Volumes Install Docker for Windows
Self Paced Training, Docker Fundamentals
Understand the architecture
Understand the architecture
Docker Compose and Networking
https://www.docker.com/
Orchestrating Docker with Machine, Swarm and Compose

28.2 Dev

Create New Image Create new image by commit docker commit <container ID> <yourname>/curl:1.0
Build image docker build -t ubuntu/test:1.0 . docker build -t ubuntu/test:1.0 –no-cache . Import/Export Container 1 2 Export the contents of a container's filesystem as a tar archive
docker export [OPTIONS] CONTAINER docker export $red_panda > latest.tar$
docker import file|URL|- [REPOSITORY[:TAG]] docker import http://example.com/exampleimage.tgz
cat exampleimage.tgz | docker import - exampleimagelocal:new Save/Load Image 3 4 Save one or more images to a tar archive
docker save [OPTIONS] IMAGE [IMAGE...] docker save busybox > busybox.tar.gz
docker load [OPTIONS] docker load < busybox.tar.gz Push Images to Docker Hub Login to docker hub docker login –username=yourhubusername –email=youremail@company.com
push docker docker push [repo:tag]
tag an image into a repository docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG] Windows 5 Useful Docker Tips and Tricks on Windows
Commandline Reference: export
Commandline Reference: import
Commandline Reference: save
Commandline Reference: load

28.3 Dockerfile

Build a song with Dockerfile
A dockerfile is a configuration file that contains instructions for building a Docker image.
Provides a more effective way to build images compared to using docker commit
Easily fits into your continuous integration and deployment process. Command Line
'FROM' instruction specifies what the base image should be FROM java:7

‘RUN‘ instruction specifies a command to execute RUN apt-get update apt-get install -y curl vim openjdk-7-jdk
CMD Defines a default command to execute when a container is created CMD ["ping", "127.0.0.1", "-c", "30"]
ENTRYPOINT Defines the command that will run when a container is executed ENTRYPOINT ["ping"] Volumes
Configuration Files and Directories**
COPY <src>... <dest> COPY hom* /mydir/ adds all files starting with "hom" COPY hom?.txt /mydir/ ? is replaced with any single character, e.g., "home.txt" Networking
Ports still need to be mapped when container is executed EXPOSE Configures which ports a container will listen on at runtime FROM ubuntu:14.04 RUN apt-get update RUN apt-get install -y nginx
EXPOSE 80 443
CMD ["nginx", "-g", "daemon off;"] 2.4 Linking Containers Example: Web app container and Database container
Linking is a communication method between containers which allows them to securely transfer data from one to another
Source and recipient containers Recipient containers have access to data on source containers Links are established based on container names Create a Link Create the source container first Create the recipient container and use the –link option
Best practice - give your containers meaningful names
Create the source container using the postgres docker run -d –name database postgres
Create the recipient container and link it docker run -d -P –name website –link database:db nginx Uses of Linking
Containers can talk to each other without having to expose ports to the host Essential for micro service application architecture Example: Container with the Tomcat running Container with the MySQL running Application on Tomcat needs to connect to MySQL Operating Systems for Docker 1 2 Operating System Features CoreOS Service Discovery, Cluster management, Auto-updates CoreOS is designed for security, consistency, and reliability. Instead of installing packages via yum or apt, CoreOS uses Linux containers to manage your services at a higher level of abstraction. A single service’s code and all dependencies are packaged within a container that can be run on one or many CoreOS machines The New Minimalist Operating Systems
Top 5 operating systems for your Docker infrastructure

28.4 Container

Lifecycle docker create creates a container but does not start it. docker rename allows the container to be renamed. docker run creates and starts a container in one operation. docker rm deletes a container. docker update updates a container’s resource limits. Starting and Stopping docker start starts a container so it is running. docker stop stops a running container. docker restart stops and starts a container. docker pause pauses a running container, "freezing" it in place. docker unpause will unpause a running container. docker wait blocks

until running container stops. docker kill sends a SIGKILL to a running container. docker attach will connect to a running container. Info docker ps shows running containers. docker logs gets logs from container. (You can use a custom log driver, but logs is only available for json-file and * journald in 1.10). docker inspect looks at all the info on a container (including IP address). docker events gets events from container. docker port shows public facing port of container. docker top shows running processes in container. docker stats shows containers' resource usage statistics. docker diff shows changed files in the container's FS. Import / Export docker cp copies files or folders between a container and the local filesystem. docker export turns container filesystem into tarball archive stream to STDOUT. Example
docker cp foo.txt mycontainer:/foo.txt docker cp mycontainer:/foo.txt foo.txt Note that docker cp is not new in Docker 1.8. In older versions of Docker, the docker cp command only allowed copying files from a container to the host
Executing Commands docker exec to execute a command in container. To enter a running container, attach a new shell process to a running container called foo
docker exec -it foo /bin/bash. Suggest Readings: docker cheatsheet

28.5 Compose

Compose a symphony
Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration. To learn more about all the features of Compose see the list of features. Docker Compose Describes the components of an application
Box your component
.component/ docker-compose.yml app-1/ app/ file-1 file-2 Dockerfile run.sh app-2/ app/ file-1 file-2 Dockerfile run.sh Example docker-compose.yml example from docker-birthday-3 3
version: "2"
services: voting-app: build: ./voting-app/. volumes: - ./voting-app:/app ports: - "5000:80" networks: - front-tier - back-tier
result-app: build: ./result-app/. volumes: - ./result-app:/app ports: - "5001:80" networks: - front-tier - back-tier
worker: image: manomarks/worker networks: - back-tier
redis: image: redis:alpine $container_name$: $redisports$: ["6379"]$networks$: $-back-tier$
db: image: postgres:9.4 $container_name$: $dbvolumes$: $-"db-data$: $/var/lib/postgresql/data"networks$: $-back-tier$
volumes: db-data:
networks: front-tier: back-tier: Networks $network_mode, ports, external_hosts, link, net$
$network_mode$: "$bridge"network_mode$: "$host"network_mode$: "$service$: [$service_name$]"$network_mode$: "$container$: [$containername/id$]"$VolumesOpsrebuildyourimagesdocker-composebuild$
build or rebuild services docker-compose build [SERVICE...] docker-compose build –no-cache database
start all the containers docker-compose up -d docker-compose up

stops the containers docker-compose stop [CONTAINER] Keep a container running in compose 2
You can use one of those lines
command: "top" command: "tail -f /dev/null" command: "sleep infinity" Docker Compose Files Version 2
github issue, Keep a container running in compose
docker-compose.yml

28.6 Swarm

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts 1
Architecture 2
Swarm Manager: Docker Swarm has a Master or Manager, that is a pre-defined Docker Host, and is a single point for all administration. Currently only a single instance of manager is allowed in the cluster. This is a SPOF for high availability architectures and additional managers will be allowed in a future version of Swarm with 598.
Swarm Nodes: The containers are deployed on Nodes that are additional Docker Hosts. Each Swarm Node must be accessible by the manager, each node must listen to the same network interface (TCP port). Each node runs a node agent that registers the referenced Docker daemon, monitors it, and updates the discovery backend with the node's status. The containers run on a node.
Scheduler Strategy: Different scheduler strategies (binpack, spread, and random) can be applied to pick the best node to run your container. The default strategy is spread which optimizes the node for least number of running containers. There are multiple kinds of filters, such as constraints and affinity. This should allow for a decent scheduling algorithm.
Node Discovery Service: By default, Swarm uses hosted discovery service, based on Docker Hub, using tokens to discover nodes that are part of a cluster. However etcd, consul, and zookeeper can be also be used for service discovery as well. This is particularly useful if there is no access to Internet, or you are running the setup in a closed network. A new discovery backend can be created as explained here. It would be useful to have the hosted Discovery Service inside the firewall and 660 will discuss this.
Standard Docker API: Docker Swarm serves the standard Docker API and thus any tool that talks to a single Docker host will seamlessly scale to multiple hosts now. That means if you were using shell scripts using Docker CLI to configure multiple Docker hosts, the same CLI would can now talk to Swarm cluster and Docker Swarm will then act as proxy and run it on the cluster.
Create Swarm Simple Swarm Swarm with CentOS Swarm with Windows Swarm and Compose 5 Create docker-compose.yml file
Example
wget https://docs.docker.com/swarm/swarm$_{a}t_{s}cale/docker-compose.ymlDiscovery4$
Scheduling 3 Docker Swarm: CentOS Prerequisites Docker 1.10.3 CentOS 7
Create cluster 1 Receipts: consult
Configure daemon in each host 6

Edit /etc/systemd/system/docker.service.d/docker.conf
[Service] ExecStart= ExecStart=/usr/bin/docker daemon -H fd:// -D -H tcp://<node$_ip$>: 2375 $--cluster-store=consul://<consul_ip>:8500 --cluster-advertise=<node_ip>:2375 Restart docker$
systemctl daemon-reload systemctl restart docker Verify docker run with config
ps aux | grep docker | grep -v grep Run cluster
In discovery host
run consul docker run -d -p 8500:8500 –name=consul progrium/consul -server -bootstrap In swarm manage host
run swarm master docker run -d -p 2376:2375 swarm manage consul://<consul$_ip$>: 8500/$swarm In swarm node host$
open 2375 port firewalld-cmd –get-active-zones firewall-cmd –zone=public –add-port=2375/tcp –permanent firewall-cmd –reload
run swarm node docker run -d swarm join –addr=<node$_ip$>: 2375$consul://<consul_ip>:8500/swarm Question: Why manager and consul cannot run in the same host? (because docker-proxy?)$
Verify 2
manage docker -H :2376 info export DOCKER$_HOST = tcp://<ip>:12375 docker info docker run -d -P redis$
debug a swarm docker swarm –debug manage consul://<consul$_ip$>: 8500$Docker Swarm: Simple Swarm Simple Swarm in one node with token 1 Change file vi /etc/default/docker$
DOCKER$_OPTS =$ "$-H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock$"
Restart docker service docker restart
create swarm token docker run –rm swarm create > token$_id$
Run swarm node docker run -d swarm join –addr=ip:2375 token://token$_i$ddocker ps
Run swarm manager docker run -d -p 12375:2375 swarm manage token://token$_id$
Swarm info docker -H :12375 info
export DOCKER$_HOST = tcp://ip:12375 docker info docker run -d -P redis Docker Swarm: Windows Prerequisites Docker 1.10.3 Giithiuvchyth Docker Swarm$
Docker-swarm, Cannot connect to the docker engine endpoint
Docker Swarm Part 3: Scheduling
Docker Swarm Part 2: Discovery
Deploy the application
Configuring and running Docker on various distributions

28.7 UCP

Docker Universal Control Plane
Docker Universal Control Plane provides an on-premises, or virtual private cloud (VPC) container management solution for Docker apps and infrastructure regardless of where your applications are running.
[embed]https://www.youtube.com/watch?v=woDzgqtZZKc[/embed]

28.8 Labs

Lab 1:
1. Create a container from an Ubuntu image and run a bash terminal docker run -it ubuntu /bin/bash 2.Inside the container install curl apt-get install curl

3. Exit the container terminal Ctrl-P 4. Run ‘docker ps -a‘ and take not of your container ID 5. Save the container as a new image. For the repository name use <yourname>/curl. Tag the image as 1.0 6. Run ‘docker images‘ and verify that you can see your new image Lab 2: Run Ubntu
FROM ubuntu:14.04 RUN apt-get update apt-get install -y curl vim 1. Go into the test folder and open your Dockerfile from the previous exercise
2. Add the folloing line to the end CMD ["ping", "127.0.0.1", "-c", "30"]
3. Build the image docker build -t <yourname>/testimage:1.1 .
4. Execute a container from the image and observe the output docker run <yourname>/testimage:1.1
5. Execute another container from the image and specify the echo command docker run <yourname>/testimage:1.1 echo "hello world"
6. Observe how the container argument overrides the CMD instruction Lab:
1) In your home directory, create a folder called test 2) In the test folder, create a file called ‘Dockerfile‘ 3) In the file, specify to use Ubuntu 14.04 as the base image FROM ubuntu:14.04 4) Write an instruction to install curl and vim after an apt-get update RUN apt-get update apt-get install -y curl vim 5) Build an image from Dockerfile. Give it the repository <yourname>/testimage and tag it as 1.0 docker build -t johnnytu/testimage:1.0 . 6) Create a container using your newly build image and verify that curl and vim are installed. Lab: Run a container and get Terminal Access
* Create a container using the ubuntu image and connect to STDIN and a terminal docker run -i -t ubuntu /bin/bash * In your container, create a new user using your first and last name as the username addusername username * Exit the container exit * Notice how the container shutdown * Once again run: docker run -i -t ubuntu /bin/bash * Try and find your user ‘cat /etc/passwd‘ * Noice that it does not exist. Lab: Run a web application inside a container
The -P flag to map container ports to host ports docker run -d -P tomcat:7 docker ps check your image details runing go to <server url>:<port number> verify that you can see the Tomcat page Lab Create a Docker Hub Account
Lab Push Images to Docker Hub
Login to docker hub $docker login--username = yourhubusername--email = youremail@company.comPassword : WARNING : logincredentialssavedinC : \Q docker.jsonLoginSucceeded$
Use ‘docker push‘ command docker push [repo:tag]
Local repo must have same name and tag as the Docker Hub repo
Used to rename a local image repository before pushing to Docker Hub docker tag [image ID] [repo:tag] docker tag [local repo:tag] [Docker Hub repo:tag]
> tag image with ID (trainingteam/testexample is the name of repository on Docker hub) docker tag edfc212de17b trainingteam/testexample:1.0
> tag image using the local repository tag docker tag johnnytu/testimage:1.5 trainingteam/testexample

28.9 GUI

Environment
Ubuntu 14.04 Docker Engine 1.11 note: I can’t make it run on CentOS.
Dockerfile 1 [code]
Base docker image FROM debian:sid MAINTAINER Jessica Frazelle jess@docker.com

ADD https://dl.google.com/linux/direct/google-talkplugin$_{c}urrent_{a}md64.deb/src/google-talkplugin_{c}urrent_{a}md64.deb$
Install Chrome RUN echo 'deb http://httpredir.debian.org/debian testing main' » /etc/apt/sources.list apt-get update apt-get install -y ca-certificates curl hicolor-icon-theme libgl1-mesa-dri libgl1-mesa-glx libv4l-0 -t testing fonts-symbola –no-install-recommends curl -sSL https://dl.google.com/linux/linux$_{s}igning_{k}ey.pub|apt-keyadd- echo"deb[arch = amd64]http : //dl.google.com/linux/chrome/deb/stablemain" > /etc/apt/sources.list.d/google.list apt − getupdateapt − getinstall − y google − chrome−stable −−no−install−recommends dpkg−i′/src/google−talkplugin_{c}urrent_{a}md64.deb′ apt−getpurge−−auto−remove−ycurl rm−rf/var/lib/apt/lists/ rm−rf/src/.deb
COPY local.conf /etc/fonts/local.conf
RUN rm /etc/apt/sources.list.d/google-chrome.list
RUN apt-get update RUN apt-get install -y libcanberra-gtk-module RUN apt-get install -y dbus libdbus-1-dev libxml2-dev dbus-x11
Autorun chrome ENTRYPOINT ["google-chrome"] CMD ["–user-data-dir=/data"] [/code]
Build Image [code] docker build -t chrome . [/code]
Docker Run [code] xhost + docker run -it –net host –cpuset-cpus 0 -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY$ -v HOME/chrome/data/Downloads:/root/Downloads -v $HOME/chrome/data/ : /data−v/deb/shm : /dev/shm−−device/dev/sndchrome−−user − data − dir = /datawww.google.com[/code]
Known Issue Issue 1: Failed to load module "canberra-gtk-module" 2
[code] Gtk-Message: Failed to load module "canberra-gtk-module" [/code]
Issue 2: Chrome crash 3
Solution: add -v /dev/shm:/dev/shm

- https://hub.docker.com/r/jess/chrome/ /dockerfile/
- http://fabiorehm.com/blog/2014/09/11/running-gui-apps-with-docker/comment-2515573749

Chương 29

Lean

View online http://magizbox.com/training/lean/site/
Lean startup is a method for developing businesses and products first proposed in 2008 by Eric Ries. Based on his previous experience working in several U.S. startups, Ries claims that startups can shorten their product development cycles by adopting a combination of business-hypothesis-driven experimentation, iterative product releases, and what he calls validated learning. Ries' overall claim is that if startups invest their time into iteratively building products or services to meet the needs of early customers, they can reduce the market risks and sidestep the need for large amounts of initial project funding and expensive product launches and failures.

29.1 Lean Canvas

29.2 Workflow

Life's too short to build something nobody wants. What separates successful startups from unsuccessful ones is that they find a plan that works before running out of resources.
Build-Measure-Learn The fundamental activity of a startup is to turn ideas into products, measure how customers respond, and then learn whether to pivot or persevere. All successful startup processes should be geared to accelerate that feedback loop.
Step 1: Document your Plan A. Writing down the initial vision using Lean Canvas and then sharing it with at least one other person
Step 2: Identify the Riskiest Parts Formulate Falsifiable Hypotheses
Falsifiable Hypothesis = [Specific Repeatable Action] will [Expected Measurable Action] The biggest risk is building something nobody wants. The risks can be divided in:
Stage 1: Problem/Solution Fit. Do I have a problem worth solving?
Answer these questions:
Is it something customers want? (must-have) Will they pay for it? If not, who will? (viable) Can it be solved? (feasible) Stage 2: Product/market Fit. Have I built something people want?"

Qualitative metrics (interviews) Quantitative metrics for measuring product/-market fit. Stage 3: Scale. How do I accelerate growth?
After product/market fit, raise a big round of funding to scale the business. Traction is needed! Seek External advice and ask specific questions.
Systematically test your Plan. Based on the scientific method run experiments. First build hypothesis that can be easily disproved and use artifacts in front of customers to "measure" their response using qualitative and quantitative data to derive "learning" to validate or refute the hypothesis.
The Lean Startup is also about being efficient. In all the stages we can take actions to be more efficient: eliminate don't-needs at the MVP, deploy continuously, make feedback easy for customers, don't push for features, etc.
The Lean Startup methodology is strongly rooted in the scientific method, and running experiments is a key activity. Your job isn't just building the best solution, but owning the entire business model and making all the pieces fit.

29.3 Validated Learning

Startups exist not to make stuff, make money, or serve customers. They exist to learn how to build a sustainable business. This learning can be validated scientifically, by running experiments that allow us to test each element of our vision.

Tài liệu tham khảo

Goodfellow, Ian / Bengio, Yoshua / Courville, Aaron (2016): *Deep Learning.* , MIT Press.

Mitchell, Thomas M. (1997): *Machine Learning.* New York, NY, USA, McGraw-Hill, Inc., 1 Auflage.

Nguyễn Tiến Dũng, Đỗ Đức Thái (2015): *Nhập môn hiện đại: Xác suất và thống kê.* , Tủ sách SPUTNIK, first Auflage.

Chỉ mục

Ghi chú