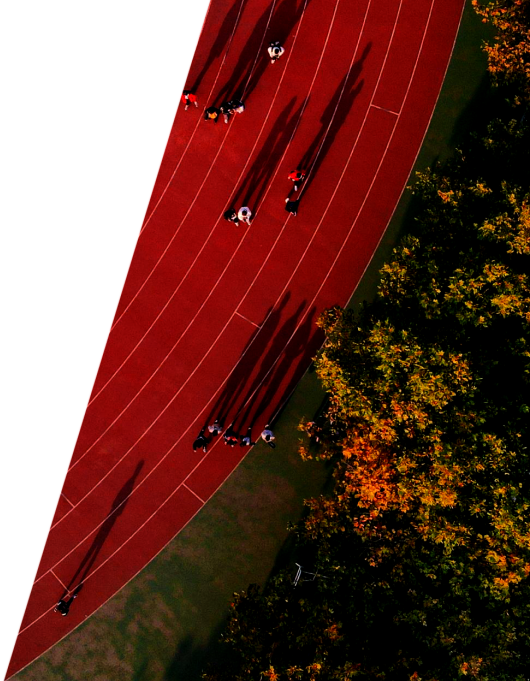
数据结构与算法
Data Structure and Algorithm

Introduction

ZHANG Luping

2023 年 6 月 24 日



This template is a secondary creation of SINTEF Presentation template from Federico Zenith

Following a brief introduction written by Federico Zenith about how to use LaTeX and beamer to prepare slides. All rights reserved by him

Table of Contents

- You have knowledge of the most common abstractions for data collections (e.g., stacks, queues, lists, trees, maps).
- You understand algorithm strategies for producing efficient realizations of common data structures.
- You can analyze algorithmic performance, both theoretically and experimentally, and recognize common trade-offs between competing strategies.
- You can wisely use existing data structures and algorithms found in modern programming language libraries.
- You have experience working with concrete implementations for most foundational data structures and algorithms.
- You can apply data structures and algorithm to solve complex problems.

Table of Contents

Declarative knowledge vs. Imperative knowledge

All knowledge can be thought of as either **declarative or imperative**.

- **Declarative knowledge** is composed of statements of fact.

Declarative knowledge vs. Imperative knowledge

All knowledge can be thought of as either **declarative or imperative**.

- **Declarative knowledge** is composed of statements of fact.
  - *The square root of x is a number y such that $y \times y = x$,*

Declarative knowledge vs. Imperative knowledge

2 Algorithm

All knowledge can be thought of as either **declarative or imperative**.

- **Declarative knowledge** is composed of statements of fact.
  - *The square root of x is a number y such that $y \times y = x$,*
  - *It is possible to travel by train from Qingdao to Beijing*

Declarative knowledge vs. Imperative knowledge

2 Algorithm

All knowledge can be thought of as either **declarative or imperative**.

- **Declarative knowledge** is composed of statements of fact.
  - *The square root of x is a number y such that $y \times y = x$,*
  - *It is possible to travel by train from Qingdao to Beijing*
- **Imperative knowledge** is "how to" knowledge, or recipes for deducing information.

A way to compute the square root of a number

2 Algorithm

A way to compute the square root of a number

2 Algorithm

1. Start with a guess, g.
2. If $g \times g$ is close enough to x, stop and say that g is the answer.
3. Otherwise, create a new guess by averaging g and $\frac{x}{g}$, i.e., $\frac{(g+\frac{x}{g})}{2}$.
4. Using this new guess, which we again call g, repeat the process until $g \times g$ is close enough to x.

Definition of algorithm

- Note that the description of the method is **a sequence of simple steps, together with a flow of control** that specifies when to execute each step. Such a description is called an **algorithm**.

Definition of algorithm

- Note that the description of the method is **a sequence of simple steps, together with a flow of control** that specifies when to execute each step. Such a description is called an **algorithm**.
- More formally, *an algorithm is a finite list of instructions describing a set of* ***computations*** *that when executed on a set of inputs will proceed through a sequence of well-defined states and eventually produce an output.*

Examples: Perfect cube root

2 Algorithm

This code prints the integer cube root, if it exists, of an integer. If the input is not a perfect cube, it prints a message to that effect.

```
# Find the cube root of a perfect cube
x = int(input("Enter an integer: "))
ans = 0
while ans**3 < abs(x):
    ans += 1
if ans**3 != abs(x):
    print(x, "is not a perfect cube")
else:
    if x < 0:
        ans = -ans
    print("Cube root of" , x, "is", ans)

```

Examples: Prime number

2 Algorithm

This code tests whether an integer is a prime number and returning the smallest divisor if it is not.

```
# Test if an int > 2 is prime. If not, print smallest divisor
x = int(input("Enter an integer greater than 2: "))
smallest_divisor = None
for guess in range(2, x):
    if x%guess == 0:
        smallest_divisor = guess
        break
if smallest_divisor != None:
    print("Smallest divisor of", x, "is", smallest_divisor)
else:
    print(x, "is a prime number")

```

Examples: Approximation to the square root of x

2 Algorithm

```
1       epsilon = 0.01
2       step = epsilon**2
3       num_guesses = 0
4       while abs(ans**2 - x) >= epsilon and ans <= x:
5           ans += step
6           num_guesses += 1
7       print("number of guesses = ', num_guesses)
8       if abs(ans**2 - x) >= epsilon:
9           print("Failed on square root of", x)
10      else:
11          print(ans, "is close to square root of", x)
12
```

Q&A

Thank you for listening!
Your feedback will be highly appreciated!