CSC165 Week 7

Larry Zhang, October 21, 2014

an informal, anonymous survey of your learning experience so far

http://goo.gl/forms/AyC01bEk7g

Exclusive for Tuesday evening section!

today's outline

➔ proof by cases

➔ review of proofs

➔ algorithms

proof by cases

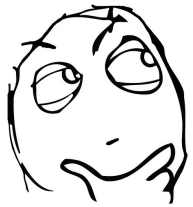
proof by cases

- ➔ **split** your argument into differences cases
- ➔ prove the conclusion **for each** case

$$\forall n \in \mathbb{N}, n^2 + n \text{ is even}$$

thoughts

$$n^2 + n = n(n + 1)$$



Proof:

$$\forall n \in \mathbb{N}, n^2 + n \text{ is even}$$

one more practice

Define predicate $T(n)$ by

$$\forall n \in \mathbb{N}, \quad T(n) \Leftrightarrow \exists i \in \mathbb{N}, n = 7i + 1$$

Prove:

$$S1 : \forall n \in \mathbb{N}, T(n) \Rightarrow T(n^2)$$

Disprove:

$$S2 : \forall n \in \mathbb{N}, T(n^2) \Rightarrow T(n)$$

$$\forall n \in \mathbb{N}, \quad T(n) \Leftrightarrow \exists i \in \mathbb{N}, n = 7i + 1$$

thoughts Prove: $S1 : \forall n \in \mathbb{N}, T(n) \Rightarrow T(n^2)$

$$n = 7i + 1$$

$$n^2 = 7k + 1$$

$$\forall n \in \mathbb{N}, \quad T(n) \Leftrightarrow \exists i \in \mathbb{N}, n = 7i + 1$$

Proof: Prove: $S1 : \forall n \in \mathbb{N}, T(n) \Rightarrow T(n^2)$

Define predicate $T(n)$ by

$$\forall n \in \mathbb{N}, \quad T(n) \Leftrightarrow \exists i \in \mathbb{N}, n = 7i + 1$$

Prove:

$$S1 : \forall n \in \mathbb{N}, T(n) \Rightarrow T(n^2)$$

Disprove:

$$S2 : \forall n \in \mathbb{N}, T(n^2) \Rightarrow T(n)$$

uniqueness (math prerequisite)

if ***m*** and ***n*** are natural numbers, with $n \neq 0$, then there is **exactly one** pair natural numbers ***(q, r)*** such that:

$$m = qn + r, \qquad n > r \geq 0$$

divide *m* by *n*, the quotient and remainder are unique

$$\forall n \in \mathbb{N}, \quad T(n) \Leftrightarrow \exists i \in \mathbb{N}, n = 7i + 1$$

Disprove: $S2 : \forall n \in \mathbb{N}, T(n^2) \Rightarrow T(n)$

thoughts

$\neg S_2$:

$$\forall n \in \mathbb{N}, \quad T(n) \Leftrightarrow \exists i \in \mathbb{N}, n = 7i + 1$$

Disproof: Disprove: $S2 : \forall n \in \mathbb{N}, T(n^2) \Rightarrow T(n)$

a review of proof patterns

patterns of inference

what's known

---

what can be inferred

Two categories of inference rules

- ➔ ***introduction****: smaller statement => larger statement*
- ➔ ***elimination****: larger statement => smaller statement*

negation introduction

assume A

…

contradiction

---

conjunction introduction

A

B

---

c is red

c is fast

---

c is red and fast

disjunction introduction

A

(nothing here)

c is red

c is red or fast

c is fast or red

(nothing)

it's a boy or a girl

implication introduction

assume A

…

B

---

assume ¬B

…

¬A

---

equivalence introduction

$A \Rightarrow B$

$B \Rightarrow A$

---

n odd $\Rightarrow n^2$ odd

n^2 odd $\Rightarrow n$ odd

---

universal introduction

assume $a \in D$

...

$P(a)$

---

assume a generic car c

...

c is red

---

all cars are red

existential introduction

$P(a)$

$a \in D$

---

c is red

c is a car

---

there exists a car that is red

negation elimination

$\neg\neg A$

"the car is not red"
is false

---

the car is red

negation elimination

$$\frac{A \qquad \neg A}{\bot}$$

$$\frac{\text{there are 51 balls} \qquad \text{there are not 51 balls}}{\text{contradiction!}}$$

conjunction elimination

$$\frac{A \wedge B}{}$$

$$\frac{\text{the car is red and fast}}{\text{the car is red}}$$

$$\frac{}{\text{the car is fast}}$$

disjunction elimination

$A \vee B$

$\neg A$

---

it’s a boy or a girl

it’s not a boy

---

it’s a girl

implication elimination

$$\frac{A \Rightarrow B \qquad A}{B}$$

$$\frac{\text{If you work hard, you get A+} \qquad \text{You work hard}}{\text{You get A+}}$$

implication elimination

$A \Rightarrow B$

$\neg B$

If you work hard, you get A+

You don't get A+

You don't work hard

equivalence elimination

$$\frac{A \Leftrightarrow B}{}$$

$$\frac{n \text{ odd} <=> n^2 \text{ odd}}{\begin{array}{l} n \text{ odd} => n^2 \text{ odd} \\ n^2 \text{ odd} => n \text{ odd} \end{array}}$$

universal elimination

$\forall\ x \in D, P(x)$

$a \in D$

$P(a)$

all cars are red

X is a car

X is red

existential elimination

$\exists\ x \in D,\ P(x)$

$\exists\ n \in N,\ n$ *divides* 32

Let $m \in N$ *such that* m *divides* 32

how to be good at proofs

- ➔ become familiar with these patterns, by **lots of practice**.
- ➔ recognize these patterns in your proof, use the manipulation rules to get closer to your target

Chapter 4

Algorithm Analysis and Asymptotic Notation

Computer scientists talk like…

"The worst-case runtime of bubble-sort is $O(n^2)$."

"I can sort it in $O(n \log n)$."

"That's too slow, make it linear-time."

"That problem cannot be solved in polynomial time."

compare two sorting algorithms

bubble sort

merge sort

demo at *http://www.sorting-algorithms.com/*

Observations

→

“running time”: what do we really mean?

- ➔ It does **NOT** mean how many **seconds** are spent in running the algorithm.
- ➔ It means **the number of steps** that are taken by the algorithm.
- ➔ So, the running time is **independent of the hardware** on which you run the algorithm.
- ➔ It only depends on the algorithm itself.

but, sometimes we don't really care about the number of steps...

- ➔ what we really care: how the number of steps **grows** as the size of input **grows**
- ➔ we **don't care** about the **absolute** number of steps
- ➔ we care about: "*when input size **doubles**, the running time **quadruples***"
- ➔ so, ***0.5n²*** and ***700n²*** are **no different**!
- ➔ **constant factors do NOT matter!**

constant factor does not matter, when it comes to growth

$$T_1(n) = 0.5\,n^2 \qquad T_2(n) = 700\,n^2$$

$$\frac{T_1(2n)}{T_1(n)} = \frac{0.5\,(2n)^2}{0.5\,n^2} = \frac{2n^2}{0.5n^2} = 4$$

$$\frac{T_2(2n)}{T_2(n)} = \frac{700\,(2n)^2}{700\,n^2} = \frac{2800\,n^2}{700\,n^2} = 4$$

we care about large input sizes

- ➔ we don't need to study algorithms in order to sort **two** elements, because different algorithms make no difference
- ➔ we care about algorithm design when the input size ***n*** is very large
- ➔ so, n^2 and n^2+n+2 are no different, because when n is really large, $n+2$ is negligible compared to n^2
- ➔ ***only the highest-order term matters***

low-order terms don't matter

$$T_1(n) = n^2 \qquad T_2(n) = n^2 + n + 2$$

$$T_1(10000) = 100,000,000$$

$$T_2(10000) = 100,010,002$$

$$\text{difference} \approx 0.01\%$$

summary of running time

- ➔ we count the number of steps
- ➔ constant factors don't matter
- ➔ only the highest-order term matters

*so, the followings are **no different**...*

$$n^2 \qquad 2n^2 + 3n \qquad \frac{n^2}{165} + 1130n + 3.14159$$

so we can say, they are all $O(n^2)$

$O(n^2)$ is called asymptotic notation

$O(n^2)$ is called asymptotic upper-bound

"growing no faster than n^2"

$\Omega(n^2)$ is called asymptotic lower-bound

"growing no slower than n^2"

will be discussed in more detail later

asymptotic notation

*It is a **simplification** of the "real" running time*

➔ *it does not tell the whole story about how fast a program runs in real life.*

- ***in real world applications, constant factor matters! hardware matters! implementation matters!***

➔ *this simplification makes possible the development of the whole **theory of computational complexity**.*

- ***HUGE idea!***

a quick note

In CSC165, sometimes we use **asymptotic notations** such as ***O(n²)***, and sometimes, when we want to be more accurate, we use the **exact forms**, such as ***3n² + 2n***

It depends on what the question asks.

analyse the time complexity of a program

linear search

```
def LS(A, x):
""" Return index i, x == A[i].
    Otherwise, return -1 """
1.  i = 0
2.  while i < len(A):
3.      if A[i] == x:
4.          return i
5.      i = i + 1
6.  return -1
```

What's the running time of this program?

linear search

```
def LS(A, x):
""" Return index i, x == A[i].
    Otherwise, return -1 """
1.  i = 0
2.  while i < len(A):
3.      if A[i] == x:
4.          return i
5.      i = i + 1
6.  return -1
```

Count time complexity

`LS([2, 4, 6, 8], 4)`

linear search

```
def LS(A, x):
""" Return index i, x == A[i].
    Otherwise, return -1 """
1.  i = 0
2.  while i < len(A):
3.      if A[i] == x:
4.          return i
5.      i = i + 1
6.  return -1
```

Count time complexity

`LS([2, 4, 6, 8], 6)`

linear search

```
def LS(A, x):
""" Return index i, x == A[i].
    Otherwise, return -1 """
1.  i = 0
2.  while i < len(A):
3.      if A[i] == x:
4.          return i
5.      i = i + 1
6.  return -1
```

what is the running time of **LS(A, x)**

if the first index where ***x*** is found is ***k***

i.e., **A[k] == x**

linear search

```
def LS(A, x):
""" Return index i, x == A[i].
    Otherwise, return -1 """
1.  i = 0
2.  while i < len(A):
3.      if A[i] == x:
4.          return i
5.      i = i + 1
6.  return -1
```

Count time complexity

`LS([2, 4, 6, 8], 99)`

linear search

```
def LS(A, x):
""" Return index i, x == A[i].
    Otherwise, return -1 """
1.  i = 0
2.  while i < len(A):
3.      if A[i] == x:
4.          return i
5.      i = i + 1
6.  return -1
```

what is the running time of **LS(A, x)**

if ***x*** is not in **A** at all

let **n** be the size of **A**

takeaway

➔ program running time varies with inputs

➔ there is a worst case in which the running time is the longest

worst-case time complexity

$t_P(x)$: running time of program P with input x

the worst-case time complexity of P
with input $x \in I$ of size n

$$W_P(n) = \max\{\, t_P(x) \mid x \in I \wedge \text{size}(x) = n \,\}$$

worst-case: performance in the worst situation, what we typically do in CSC165

best-case: performance in the best situation, not very interesting, rarely studied

average-case: the expected performance under random inputs following certain probability distribution, will study in CSC263

next week

➔ more on asymptotic notations

➔ more algorithm analyses