Induction and Recursion

李昂生

Discrete Mathematics
U CAS
3, May, 2018

Outline

General view

- Understanding the various closely related, but different concepts, such as **definition**, **proofs**, **induction**, **recursion** and **computation**.
  What the is **essence** of the **computation** and **proof**?
- The challenges for the future

General approach of proof by induction

- Base step
  To establish the initial results as the base of the induction.
- Inductive step
  To prove that the result for a new term can be established from the established propositions

Inductive proofs

Inductive proofs naturally apply to:

- The order n of recursive sequences
- The time step of the execution of an algorithmic procedure
- The complexity of a structure such as:
  – the length of strings,
  – the number of operations used in an object
- Inductive proof is one of the main proving methods in CS. It works by induction on the time step of an algorithmic procedure or on the complexity of a structure.

The key for inductive proofs

- The key to an inductive proof is to fully understand the **structure** of the proofs
- The essence of inductive proof is to establish new things from the existing propositions, which occurs in various forms in Nature and Society
- Induction establishes the results one by one in **order**. So order is the key to induction.

Defining an object or a system

An object such as functions, sets, sequences or systems, is usually complex, for which we define it inductively.
An inductive definition of an object consists of the following steps:
Base step. Determine the initial elements of an object.
Inductive step Describe the rules to generate new elements of the objects from the existing components.
It is natural to prove by induction that the object defined by induction has certain properties.

In this case, the result is proved by induction on the **order** of the construction of the objects.

Recurrence

A **recurrence** is the function that determines the inductive step of a function, together with some initial values.
Examples:
The Fibonacci numbers are defined by

$$\begin{cases} f_0 = 0, f_1 = 1 \\ f_n = f_{n-1} + f_{n-2}. \end{cases} \tag{1}$$

For recurrence, we are interested in finding the **explicit expression** of the general term such as the f_n in the Fibonacci sequences.

The roles of recursion

- Induction is a **key** to proof
- Induction is a method for constructing systems
- Recurrence is a method for representing of a function
- Recursion is an idea for designing algorithms

Consequently, **recursion** is a key to proofs, expressions, constructions and algorithms

Definition of Algorithm

Turing machines capture the machine computability. Classic algorithms satisfy **5 principles**, that is, the **definition of algorithm**: A set of instructions satisfying:

(i) Input
(ii) Effectiveness
(iii) Finiteness
(iv) Determinism
(v) Output

The algorithm for the greatest common divisor

Our algorithm for computing the **greatest common divisor** is a **recursive algorithm**: For natural number a, b,
Suppose that

$$a = qb + r, 0 \leq r < b. \tag{2}$$

Then

$$(a, b) = (b, r), \tag{3}$$

which gives the recursive procedure for computing the greatest common divisor of a and b.

Computing $a^n \bmod m$

Let $n = a_l \cdots a_1 a_0$.
Define

$$c_0 = a \bmod m$$

For $j < l$,
Define

$$c_{j+1} = (c_j)^2 \bmod m.$$

Let $s = 1$.
For j from 0 to l, in increasing order:
Set

$$s = \begin{cases} sc_j, & \text{if } a_j = 1 \\ s, & \text{otherwise.} \end{cases} \tag{4}$$

Binary search

Given an increasing sequence

$$a_1 < a_2 < \cdots < a_n,$$

and a number a.
Find the index i such that

$$a_i \leq a < a_{i+1}.$$

Merge sort - I

Given a sequence of natural numbers

$$S = a_1, a_2, \cdots, a_n,$$

design a recursive algorithm to sort the sequence in increasing order of numbers.
The idea is as follows:

(1) Let S_1 be the sequence of the first half of S, and S_2 be the sequence of the remaining numbers. That is, for $l = \lfloor \frac{n}{2} \rfloor$,

$$S_1 = a_1, a_2, \cdots, a_l,$$

and

$$S_2 = a_{l+1}, \cdots, a_n.$$

Merge sort - II

(2) Suppose that we are able to sort S_1 and S_2 as L_1 and L_2, respectively.

(3) Merging L_1 and L_2 to form an increasing order of S.

(4) For sorting S_1 and S_2, we use the steps (1) - (3) recursively.

- Binary splitting
- Merging two sorted lists

Merge sort - III

Suppose that L_1 and L_2 are two sequences

$$L_1 = c_1 < c_2 < \cdots < c_l$$

$$L_2 = d_1 < d_2 < \cdots < d_r.$$

Clearly, the least element is

$$m = \min\{c_1, d_1\}.$$

Each time, we delete the least element among the two sequences. This gives an algorithm of time complexity

$$O(l + r).$$

Splitting

Given an input instance sequence $a_1, a_2, \cdots, a_n$.
Define a tree T as follows:

1. Let $\lambda = \emptyset$ be the root of T, and associate the root node with

$$L_\lambda = a_1, a_2, \cdots, a_n.$$

2. Suppose that α is a node of T such that L_α has length greater than 2. Then introduce $\alpha\hat{}\langle 0\rangle$ and $\alpha\hat{}\langle 1\rangle$ to be the immediate successors of α such that
   – $\alpha\hat{}\langle 0\rangle$ is to the left of $\alpha\hat{}\langle 1\rangle$,
   – define $L_{\alpha\hat{}\langle 0\rangle}$ to be the first half of L_α, and $L_{\alpha\hat{}\langle 1\rangle}$ to be the remaining part of L_α.
3. Noting that for each α, L_α is expressed by a pair (l, r) such that $L_\alpha = a_l, \cdots, a_r$, for which $2 \log n$ bits are sufficient.

The time complexity of the construction of T is:

$$O(n \cdot \log n).$$

Sorting in T

Suppose that the *splitting tree* T has been defined.
For every node $\alpha \in T$, if L_α contains at most two elements, then Define

$$O_\alpha$$

to be the increasing order of L_α
For every α, if $|L_\alpha| > 2$ and both $O_{\alpha\hat{}\langle 0\rangle}$ and $O_{\alpha\hat{}\langle 1\rangle}$ are defined, then

$$O_\alpha$$

to be the *merging ordering* of $O_{\alpha\hat{}\langle 0\rangle}$ and $O_{\alpha\hat{}\langle 1\rangle}$.
Finally
O_λ is the sorted ordering of $a_1, a_2, \cdots, a_n$.

The complexity

- The time complexity of the algorithm above is

$$O(n \cdot \log n).$$

- What is the parallel time complexity?
- The space complexity:

$$O(n \cdot \log n).$$

Tree method

- The idea of the recursive algorithm above is a tree method
- Tree is a **general method** for data structure and algorithms

KNAPSACK

Given n weights w_i, $i = 1, 2, \cdots, n$, a weight limit W and n values v_i, $i = 1, 2, \cdots, n$.
Find a subset $S \subset [n]$ such that

$$\begin{aligned} &\text{maximize} \sum_{i \in S} v_i \\ &\text{subject to} \sum_{i \in S} w_i \leq W. \end{aligned} \tag{5}$$

Algorithm I

Define $V(w, i)$ to be the largest value attainable by selecting some among the first i items so that their total weight is exactly w.
Then, for every w,

$$V(w, 0) = 0 \tag{6}$$

and

$$V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}. \tag{7}$$

Solving the recurrence, let V be the maximum $V(w, n)$ for all $w \leq W$.

Complexity

- **The time complexity**

$$O(W \cdot n),$$

  a pseudo-polynomial time algorithm.
- **The space complexity**

$$O(W + n).$$

Algorithm - II

Theorem 1

For any $\epsilon > 0$*, there is an algorithm that finds the value* $\geq (1-\epsilon) \cdot \mathrm{OPT}$ *in time complexity* $O(\frac{n^3}{\epsilon})$.

Let $V = \max\{v_1, v_2, \cdots, v_n\}$, each v with $0 \leq v \leq nV$, define

$$W(i, v)$$

to be the minimum weight attainable by selecting some among the first i items so that their value is exactly v.

Then, $W(0, v) = \infty$.

$$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}.$$

In the end, we pick the largest v such that $W(n, v) \leq W$.

The time complexity:

$$O(n^2 V),$$

which is polynomial in n, if $V = poly(n)$.

Approximation

Assume $V >> poly(n)$. Given instance

$$x = (w_1, w_2, \cdots, w_n, W, v_1, v_2, \cdots, v_n).$$

Let

$$x' = (w_1, w_2, \cdots, w_n, W, v_1', v_2', \cdots, v_n'),$$

each $v_j' = 2^b \lfloor \frac{v_j}{2^b} \rfloor$, $j = 1, 2, \cdots, n$.
Solving x' to find an optimal solution with set S' in time $O(n^2 \cdot \frac{V}{2^b})$.

Proof

Let S be the set with optimum solution for instance x. Then

$$\begin{aligned}
\sum_{i \in S'} v_i &\geq \sum_{i \in S'} v_i' \geq \sum_{i \in S} v_i' \\
&\geq \sum_{i \in S} (v_i - 2^b) \\
&= \sum_{i \in S} v_i - |S| \cdot 2^b \\
&\geq (1 - \epsilon)\mathrm{OPT},
\end{aligned}$$

setting $\epsilon = \frac{n2^b}{V}$, noting that $V \leq \mathrm{OPT} \leq n \cdot V$.
Then:
Time complexity:

$$O(n^2 \frac{V}{2^b}) = O(\frac{n^3}{\epsilon}).$$

Divide-and-conquer recurrence relation

If a problem of size n can be divided into subproblems of size $\frac{n}{b}$, then the time complexity of the problem is usually characterised by a recurrence relation of the following form:

$$f(n) = a \cdot f(\frac{n}{b}) + g(n), \tag{8}$$

allowing us to establish the upper bound of the recursive algorithms.
The idea of the algorithm is called **divide-and-conquer** method. It works for many problems.

Complexity of Binary Search

Given

$$a_1 < a_2 < \cdots < a_n,$$

and a, find i such that

$$a_i \leq a < a_{i+1}.$$

The recurrence of the number of comparisons:

$$f(n) = f(\frac{n}{2}) + 1.$$

Solving the recurrence,

$$f(n) = \log_2 n.$$

Finding the Maximum

Given

$$a_1, a_2, \cdots, a_n,$$

find the maximum of the a_i's.
The recurrence of the number of comparisons is:

$$f(n) = 2 \cdot f(\frac{n}{2}) + 1.$$

Solving the recurrence,

$$f(n) = 2n - 1.$$

Merge Sort

For $a_1, a_2, \cdots a_n$,
the recurrence is:

$$M(n) = 2 \cdot M(\frac{n}{2}) + n.$$

Solving the recurrence,

$$M(n) = n \log n.$$

Fast Multiplication of Integers

Given

$$a = a_{2n-1}a_{2n-2}\cdots a_{n+1}a_n\cdots a_1a_0,$$

$$b = b_{2n-1}b_{2n-2}\cdots b_{n+1}a_n\cdots b_1b_0.$$

Let A_0 and A_1 be lower and higher n-bits of a, respectively, and B_0 and B_1 be the lower and higher n-bits of b, respectively. Then,

$$\begin{aligned} ab &= (2^nA_1 + A_0)\cdot(2^nB_1 + B_0) \\ &= (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0, \end{aligned}$$

giving

$$f(2n) = 3f(n) + cn,$$

for some constant c.

Strassen Fast Matrix Multiplication

Volker Strassen, 1969:
The multiplication of two $n \times n$ matrices can be reduced to:

- 7 multiplications of two $\frac{n}{2} \times \frac{n}{2}$ matrices
- 15 additions of $\frac{n}{2} \times \frac{n}{2}$ matrices.

Therefore,

$$f(n) = 7f(\frac{n}{2}) + 15\frac{n^2}{4}.$$

The idea

What are the computable functions?

- Recursion is a mechanism of computation
- Recursion is the essence of computation （递归刻画了计算的内涵）

Encoding discrete objects by natural numbers

Any discrete or symbolic computation can be encoded as functions over natural numbers.
Therefore, we focus on the functions on natural numbers $\mathbb{N} = \{0, 1, 2, \cdots\}$.

Characteristic function

For every set $X \subset \mathbb{N}$, X is identical to its characteristic function χ_X, defined by

$$\chi_X(x) = \begin{cases} 1, & \text{if } x \in X, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Sets are the special case of functions.

Pairing function

There exist computable functions π, π^{-1} such that

- $\pi : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is a one-to-one map such that for every $x \in \mathbb{N}$,

  $$\pi(x) = \langle y, z \rangle,$$

  for some y, z

- $$\pi^{-1}\pi(x) = x.$$

Consequence

We regard $\mathbb{N}$ and $\mathbb{N}^2$ as identical.

The Basic Functions

1. Constant functions

$$f(x) = c,$$

for some constant $c \in \mathbb{N}$.

2. The successor function

$$f(x) = x + 1.$$

3. Projection functions
For every $n \geq 1$, every i with $1 \leq i \leq n$,

$$U_i^n(x_1, x_2, \cdots, x_n) = x_i.$$

Intuition All the basic functions are computable.

Composition

Given $f(x_1, x_2, \cdots, x_n), g_1(y), \cdots, g_n(y)$,

$$f(g_1(y), g_2(y), \cdots, g_n(y))$$

is the composition of $f, g_1, \cdots, g_n$.
Intuition If $f, g_1, \cdots, g_n$ are computable, then so is the composition of the functions.

Recursion

It is an **operator** defined below.
Given $f(x)$, $g(x, y, z)$, define h by:

$$\begin{cases} h(x, 0) = f(x) \\ h(x, y + 1) = g(x, y, h(x, y)). \end{cases} \tag{10}$$

Intuition If both f and g are computable, then so is h.

Search operator

Given $f(x, y)$, define

$$g(x) = \text{the least } y \text{ such that } f(x, y) = 0. \tag{11}$$

We denote g by

$$g(x) = \mu y[f(x, y) = 0].$$

g is called *search operator* or *minimisation operator* or *μ-operator*.

Remarks:

(i) $g(x)$ may not be defined, due to the fact that there is no y such that $f(x, y) = 0$.

In this case, the search for y with $f(x, y) = 0$ is unbounded, without giving any output.

This leads to the *halting problem*.

(ii) If for every x, there is y such that $f(x, y) = 0$, and if f is computable, then so is g.

Definition of recursive function

Definition 2

It is an inductive definition.

Base Step: The basic functions are recursive functions.

Inductive Step: Any function defined by one of the composition operator, recursion operator and search operator from recursive functions is a recursive function.

Recursive function

For some functions f defined in the inductive step, f may not be total, in the sense that on some input x, $f(x)$ is undefined, meaning that the search enters a dead loop or unbounded searching. Those functions are called *recursively partial functions*. Usually, a function is called **recursive** if it is defined by the inductive definition above as a total function.

The algorithm of the inductive definition

The inductive definition naturally gives an algorithm that generates all recursive or recursively partial functions. This means that there is an algorithm $\mathcal{A}$ that lists all the functions of the inductive definition as follows:

$$\phi_0, \phi_1, \cdots, \tag{12}$$

where ϕ_i is the ith function defined by the inductive definition or algorithm $\mathcal{A}$, each of which is a partial recursive function.

Inductive definition implies an algorithm

There is no algorithm to generate exactly all the recursive functions

Suppose to the contrary that this is an algorithm $\mathcal{B}$ that generates all the recursive functions by

$$\psi_0, \psi_1, \cdots, \tag{13}$$

where ψ_i is the ith function defined by algorithm $\mathcal{B}$
Then define

$$\psi(x) = \psi_x(x) + 1.$$

Then:

- ψ is recursive
  (This can be proven.)
  Let $\psi = \psi_n$
- $\psi(n) = \psi_n(n)$ and $\psi(n) = \psi_n(n) + 1$. A contradiction.

Turing Machine

A Turing machine is four-tuple (H, Σ, Q, Δ) of finite sets Σ, Q and Δ, satisfying:

(1) H is the location of the *reading head* of M.

(2) Σ is the alphabet representing the inputs and outputs

(3) Q is the set of states, denoted $q_0, q_1, \cdots, q_l$, with q_0 and q_l being the initial and final states, respectively.

(4) Δ is the finite set of *instructions* of the form

$$(q_i, s; q_j, s', X), \tag{14}$$

where $X = L$ or R, or M.

Turing Machine - continued

(5) Instruction $(q_i, s; q_j, s', X)$ means if
the current state is q_i, and the symbol scanned by the reading head is s, then
– change the state to q_j,
– change the symbol s to s' in the cell scanned by the reading head,
– move the reading head one cell to the left if $X = L$, to the right if $X = R$, and keep unchanged if $X = M$.

Turing computation

Given a Turing machine M, the computation of M proceeds as follows:

(i) Input
It is a string of the form $s_1, s_2, \cdots s_n$ for symbols s_i's in Σ.

(ii) The initially, the state is q_0, and the reading head points at the first symbol s_1 of the input

(iii) During the procedure of the computation, all the instructions are executed in a working type.

(iv) The machine halts when is reaches its last state q_l.

(v) The output is the configuration written on the working type (or a separate output type) when it halts.

Nondeterministic Turing Machine

An instruction is a set of the following form:

$$(q, s; I)$$

where I is a set of triples (q', s', X').

Functions computable by Turing Machines

Theorem 3

For any function $f : \mathbb{N} \to \mathbb{N}$, if it is computed by a Turing machine, then f is a function generated by the inductive definition of recursive functions.

- **Recursive functions = Turing computable functions**

Is recursive equal to computable?

- **Recursive functions are computable**.
- Church-Turing thesis:
  Recursion = Computation

Universal Turing Machine

There exists an algorithm to encode a Turing machine M to a natural number e, which generates all the Turing machines as follows:

$$M_1, M_2, \cdots, \tag{15}$$

where M_i is the Turing machine with code i, or the i-th Turing machine.
For each i, let ϕ_i be the function that is computed by M_i.
Then define U:

1. On input x, y,
2. Decode M_x,
3. Compute and output $M_x(y)$, i.e., $U(x, y) = M_x(y)$.

U is a Turing machine that simulates all M_i for all the i's., which is hence called the **universal Turing machine** - which becomes the model of the computers in the real world

Halting Problem

Let U be a universal Turing machine, define

$$K = \{x \mid U(x, x) \downarrow = 0\}.$$

Theorem 4

There is no Turing machine that computes K.

If χ_K is computed by M_n, then $M_n = K$. However,
Case 1. If $n \notin K$, then either $M_n(n) = K(n) = 0$, so that $n \in K$.
A contradiction.
Case 2. If $n \in K$, then $M_n(n) = K(n) = 1$, so that $n \notin K$.
In either case, there is a contradiction.

This is a typical **diagonal method**, which is a general method in wide range of disciplines.

Relative Computation

We may introduce an extra type for representing the information, referred to **oracle**, provided to a Turing machine from outside source.
In this case, we may develop the Turing machine M_x^A, which may query the information in A, called an oracle A, stored in the oracle type.
This leads to the direction of Computability Theory.
Oracle could be a new resource for algorithms

Conclusions

- Turing machine provides the mathematical model for computers, and for computer science. This is the reason why we have computer science.
- Turing machines focus only on one of the aspects of computation, that is, the **effectiveness**, or the **machine mechanism**.
- Juris Hartmanis defined the *time complexity* of a Turing computation to be the number of times that the instructions are executed, and the *space complexity* to be the number of cells used in the working type of a Turing machine. This leads to Complexity Theory.
  This feature makes Turing machine essentially different from the characterisation of recursive functions, opening the new world of human civilization.

Recursion vs Computation

- Recursion is the **closing definition** of computation
- Computation is an **open world phenomenon**

Church-Turing Thesis

The current state of the art suggests

- Every function that is computed by any device can be computed by a Turing machine.
- Every function that is computed by any device in time polynomial in n, can be computed by a Turing machine in time $n^{O(1)}$, where n is the size of the input.

New Algorithms

(i) Input
input could be large, computation may not even read the inputs

(ii) Effectiveness

(iii) Finiteness
This must be the same

(iv) Determinism

(v) Output
Output may not be unique, like the answers from internet queries

Open Question:

1. **How to realise intelligence?**
2. **Randomness is useful in real world algorithms**

Exercise - 1

(1) Show that if n is a positive integer, then the sequence

$$2 \mod n,\ 2^2 \mod n,\ 2^{2^2} \mod n,\ \cdots \qquad (16)$$

converges to a constant.

(2) Give a recursive definition of the set of bit strings that contain twice as many 0s as 1s.

(3) Give the exact form of functions π and π^{-1} in page 35.

Exercise - 2

(4) Find the maximum number of regions of a plane that is divided by n straight lines.

(5) Find the maximum number of regions of a three-dimensional ball that is divided by n planes.

谢谢！