CS 2305: Discrete Mathematics for Computing I

Lecture 17

- KP Bhat

Some Important Functions

The *floor* function, denoted

$$f(x) = \lfloor x \rfloor$$

is the largest integer less than or equal to *x*.

The *ceiling* function, denoted

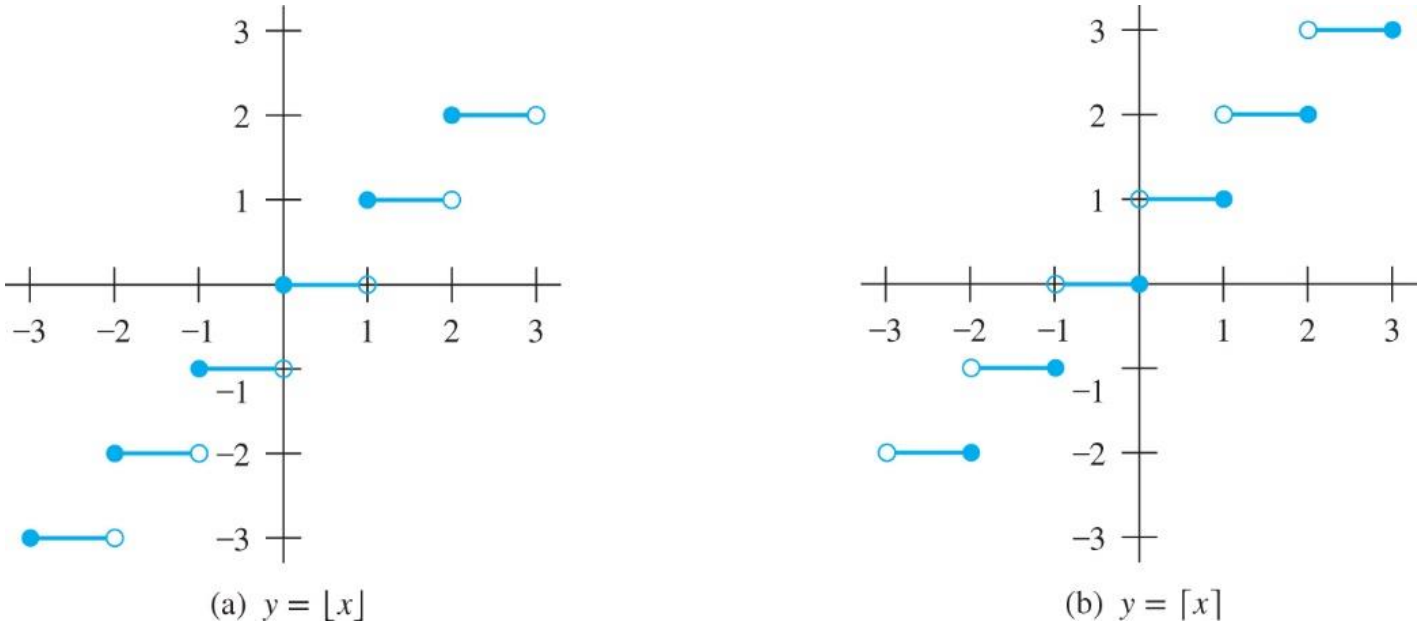
$$f(x) = \lceil x \rceil$$

is the smallest integer greater than or equal to *x*

Example: $\lceil 3.5 \rceil = 4$ $\lfloor 3.5 \rfloor = 3$

$\lceil -1.5 \rceil = -1$ $\lfloor -1.5 \rfloor = -2$

Floor and Ceiling Functions 1



Graph of (a) Floor and (b) Ceiling Functions

Floor and Ceiling Functions 2

TABLE 1 **Useful Properties of the Floor and Ceiling Functions.**
(n is an integer, x is a real number)

(1a)	$\lfloor x \rfloor = n$ if and only if $n \le x < n+1$
(1b)	$\lceil x \rceil = n$ if and only if $n-1 < x \le n$
(1c)	$\lfloor x \rfloor = n$ if and only if $x-1 < n \le x$
(1d)	$\lceil x \rceil = n$ if and only if $x \le n < x+1$
(2)	$x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$
(3a)	$\lfloor -x \rfloor = -\lceil x \rceil$
(3b)	$\lceil -x \rceil = -\lfloor x \rfloor$
(4a)	$\lfloor x+n \rfloor = \lfloor x \rfloor + n$
(4b)	$\lceil x+n \rceil = \lceil x \rceil + n$

Proving Properties of Functions

Example: Prove that x is a real number, then

$$\lfloor 2x \rfloor = \lfloor x \rfloor + \lfloor x + ½ \rfloor$$

Solution: Let $x = n + \varepsilon$, where n is an integer and $0 \le \varepsilon < 1$.

Case 1: $\varepsilon < ½$

- $\lfloor 2x \rfloor = \lfloor 2(n + \varepsilon) \rfloor = \lfloor 2n + 2\varepsilon \rfloor = 2n$, since $2\varepsilon < 1$.
- $\lfloor x \rfloor = \lfloor n + \varepsilon \rfloor = n$, since $\varepsilon < ½$
- $\lfloor x + ½ \rfloor = \lfloor n + \varepsilon + ½ \rfloor$ Now $\varepsilon < ½$ so $\varepsilon + ½ < ½ + ½$ or $\varepsilon + ½ < 1$ $\therefore \lfloor x + ½ \rfloor = n$
- Hence, $\lfloor 2x \rfloor = 2n$ and $\lfloor x \rfloor + \lfloor x + ½ \rfloor = n + n = 2n$.

Case 2: $\varepsilon \ge ½$

- $\lfloor 2x \rfloor = \lfloor 2(n + \varepsilon) \rfloor = \lfloor 2n + 2\varepsilon \rfloor = 2n + 1$, since $2\varepsilon \ge 1$.
- $\lfloor x \rfloor = \lfloor n + \varepsilon \rfloor = n$, since $\varepsilon < 1$
- $\lfloor x + ½ \rfloor = \lfloor n + \varepsilon + ½ \rfloor$ Now $\varepsilon \ge ½$ so $\varepsilon + ½ \ge ½ + ½$ or $\varepsilon + ½ \ge 1$ $\therefore \lfloor x + ½ \rfloor = n+1$
- Hence, $\lfloor 2x \rfloor = 2n + 1$ and $\lfloor x \rfloor + \lfloor x + ½ \rfloor = n + (n + 1) = 2n + 1$.

Factorial Function

Definition: f: $\mathbf{N} \rightarrow \mathbf{Z}^+$ **,** denoted by $f(n) = n!$ is the product of the first n positive integers when n is a nonnegative integer.

$$f(n) = 1 \cdot 2 \cdots (n-1) \cdot n, \qquad f(0) = 0! = 1$$

Stirling's Formula:

$$n! \sim \sqrt{2\pi n}(n/e)^n$$

$$\lim_{n\to\infty} f(n)/g(n) = 1$$

Examples:

$$f(1) = 1! = 1$$

$$f(2) = 2! = 1 \cdot 2 = 2$$

$$f(6) = 6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$$

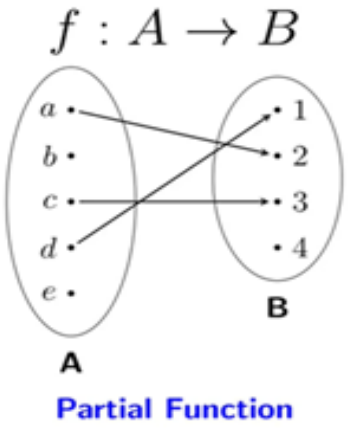
$$f(20) = 2,432,902,008,176,640,000.$$

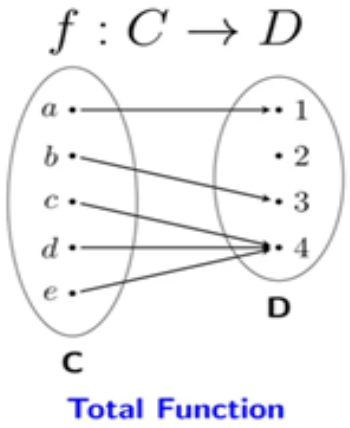
Partial Functions

Definition: A *partial function f* from a set *A* to a set *B* is an assignment to each element *a* **in a subset of *A*,** called the *domain of definition* of *f*, of a unique element *b* in *B*.

- The sets *A* and *B* are called the *domain* and *codomain* of *f*, respectively.
- We day that *f* is ***undefined*** for elements in *A* that are not in the domain of definition of *f*.
- When the domain of definition of *f* equals *A*, we say that *f* is a *total function*.
  - Domain of definition is equal to the domain

Example: *f*: **Z** → **R** where $f(n) = \sqrt{n}$ is a partial function where the domain of definition is the set of nonnegative integers. Note that *f* is undefined for negative integers.





Sequences and Summations

Section 2.4

Introduction

Sequences are ordered lists of elements.

- 1, 2, 3, 5, 8
- 1, 3, 9, 27, 81, …….

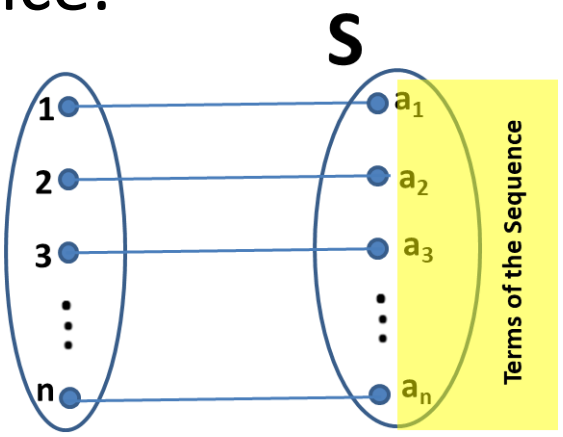
The notion of position is important in sequences. It is the index at which a certain value appears in the sequence

Sequences arise throughout mathematics, computer science, and in many other disciplines, ranging from botany to music.

Sequences 1

Definition: A *sequence* is a function from a subset of the integers (usually either the set {0, 1, 2, 3, 4, …..} or {1, 2, 3, 4, ….}) to a set *S*.

The notation a_n is used to denote the image of the integer *n*. We can think of a_n as the equivalent of *f(n)* where *f* is a function from {1,2,…..} to *S*. We call a_n a *term* of the sequence.



Sequences 2

Example: Consider the sequence $\{a_n\}$ where

$$a_n = \frac{1}{n} \qquad \{a_n\} = \{a_1, a_2, a_3 \ldots\}$$

Terms of the sequence: $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \cdots$

Geometric Progression

Definition: A *geometric progression* is a sequence of the form: $a, ar, ar^2, \ldots, ar^n, \ldots$

where the *initial term a* and the *common ratio r* are real numbers.

Examples :

1. Let $a = 1$ and $r = -1$. Then :

$$\{b_n\} = \{b_0, b_1, b_2, b_3, b_4, \ldots\} = \{1, -1, 1, -1, 1, \ldots\}$$

2. Let $a = 2$ and $r = 5$. Then :

$$\{c_n\} = \{c_0, c_1, c_2, c_3, c_4, \ldots\} = \{2, 10, 50, 250, 1250, \ldots\}$$

3. Let $a = 6$ and $r = 1/3$. Then :

$$\{d_n\} = \{d_0, d_1, d_2, d_3, d_4, \ldots\} = \left\{6, 2, \frac{2}{3}, \frac{2}{9}, \frac{2}{27}, \ldots\right\}$$

Arithmetic Progression

Definition: A arithmetic progression is a sequence of the form: $a,\ a+d,\ a+2d,\ldots,\ a+nd,\ldots$

where the *initial term a* and the *common difference d* are real numbers.

Examples :

1. Let $a=-1$ and $d=4$:

$$\{s_n\}=\{s_0,\ s_1,\ s_2,\ s_3,\ s_4,\ldots\}=\ \{-1,\ 3,\ 7,\ 11,\ 15,\ \ldots\}$$

2. Let $a=7$ and $d=-3$:

$$\{t_n\}=\{t_0,\ t_1,\ t_2,\ t_3,\ t_4,\ldots\}=\{7,\ 4,\ 1,-2,\ -5,\ldots\}$$

3. Let $a=1$ and $d=2$:

$$\{u_n\}=\{u_0,\ u_1,\ u_2,\ u_3,\ u_4,\ldots\}=\{1,\ 3,\ 5,\ 7,\ 9,\ldots\}$$

Summations

Sum of the terms $a_m, a_m + 1, \ldots, a_n$

from the sequence $\{a_n\}$

The notation:

$$\sum_{j=m}^{n} a_j \text{ or } \sum_{j=m}^{n} a_j \text{ or } \sum_{m \le j \le n} a_j$$

represents

$$a_m + a_{m+1} + \cdots + a_n$$

The variable *j* is called the *index of summation*. It runs through all the integers starting with its *lower limit m* and ending with its *upper limit n*.

Geometric Series [1]

Sums of terms of geometric progressions

$$\sum_{j=0}^{n} ar^j = \begin{cases} \dfrac{ar^{n+1} - a}{r - 1} & r \neq 1 \\ (n+1)a & r = 1 \end{cases}$$

Proof: Let

$$S_n = \sum_{j=0}^{n} ar^j$$

$$rS_n = r\sum_{j=0}^{n} ar^j$$

$$= \sum_{j=0}^{n} ar^{j+1}$$

To compute S_n, first multiply both sides of the equality by r and then manipulate the resulting sum as follows:

Geometric Series 2

$$= \sum_{j=0}^{n} ar^{j+1}$$

From previous slide.

$$= \sum_{k=1}^{n+1} ar^{k}$$

Replacing the index of summation with $k = j + 1$.

$$= \left(\sum_{k=1}^{n} ar^{k} \right) + ar^{n+1}$$

Removing $k = n + 1$ term from the summation sign

$$= a + \left(\sum_{k=1}^{n} ar^{k} \right) + ar^{n+1} - a$$

Adding and subtracting "a" (i.e. $k = 0$ term).

$$= \left(\sum_{k=0}^{n} ar^{k} \right) + (ar^{n+1} - a)$$

Bringing the $k = 0$ term under the summation sign by changing the lower limit of the summation

Geometric Series 3

$$= \left(\sum_{k=0}^{n} ar^k\right) + (ar^{n+1} - a)$$ From previous slide.

$$= S_n + \left(ar^{n+1} - a\right)$$ Substituting *S* for summation formula

$$\therefore \qquad rS_n = S_n + \left(ar^{n+1} - a\right)$$

$$S_n = \frac{ar^{n+1} - a}{r - 1}$$ if r ≠1

Special Case: r = 1

$$S_n = \sum_{j=0}^{n} ar^j = \sum_{j=0}^{n} a = (n+1)a$$ if r = 1

Some Useful Summation Formulae

TABLE 2 Some Useful Summation Formulae.

Sum	***Closed From***
$\sum_{k=0}^{n} ar^k \ (r \neq 0)$	$\frac{ar^{n+1}-a}{r-1},\ r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k, \lvert x \rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=0}^{\infty} kx^{k-1}, \lvert x \rvert < 1$	$\frac{1}{(1-x)^2}$

Geometric Series: We just proved this.

Later we will prove some of these by induction.

Proof in text (requires calculus)

Algorithms

Chapter 3

Problems and Algorithms

The first step in solving many computational problems is to precisely state the problem, using the appropriate structures to specify the input and the desired output.

We then solve the general problem by specifying the steps of a procedure that takes a valid input and produces the desired output. This procedure is called an *algorithm*.

Algorithms[1]

Abu Ja'far Mohammed
Ibin Musa Al-Khowarizmi
(780-850)

Definition: An *algorithm* is a finite set of precise instructions for performing a computation or for solving a problem.

- Concept of algorithms has existed for ever but it started becoming part of common parlance only after the invention of the digital computer, in the 1940s
  - Word did not appears in the Webster's New World Dictionary till 1957
- Origin of the word is disputed but many people now believe that the word was derived from the name of famous Persian mathematician al-Khwarizmi
  - Lived in the 9th century and wrote a famous book that gave algorithmic solutions to many mathematical problems
    - The word "algebra" is derived from the title of that book!

Algorithms 2

Abu Ja'far Mohammed
Ibin Musa Al-Khowarizmi
(780-850)

Example: Describe an algorithm for finding the maximum value in a finite sequence of integers.

Solution: Perform the following steps:

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum.
   - If it is larger than the temporary maximum, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integers. If not, stop.
4. When the algorithm terminates, the temporary maximum is the largest integer in the sequence.

Specifying Algorithms

Algorithms can be specified in different ways.

- Human language
- Flowchart
- *Pseudocode*

Pseudocode is an intermediate step between an English language description of the steps and a coding of these steps using a programming language

- employs constructs supported by a typical programming language (especially looping, branching and functions/subroutines), together with informal English statements

Programmers can use the description of an algorithm in pseudocode to construct a program in a particular language.

Pseudocode helps us analyze the time required to solve a problem using an algorithm, independent of the actual programming language used to implement algorithm.

Flowchart

- Graphical representation of an algorithm
- Operations, instructions and series of instructions are represented by boxes of different shapes
- The flow of control is represented by directed lines connecting the boxes
- Unwieldy for algorithms of even moderate complexity
  - Great learning/teaching tool

Some Commonly Used Flowchart Symbols

Oval

Stadium

Terminator: Start / Stop

Rectangle

Calculation or process, other than a decision

Diamond

Decision

Parallelogram

Input or Output

Small Circle

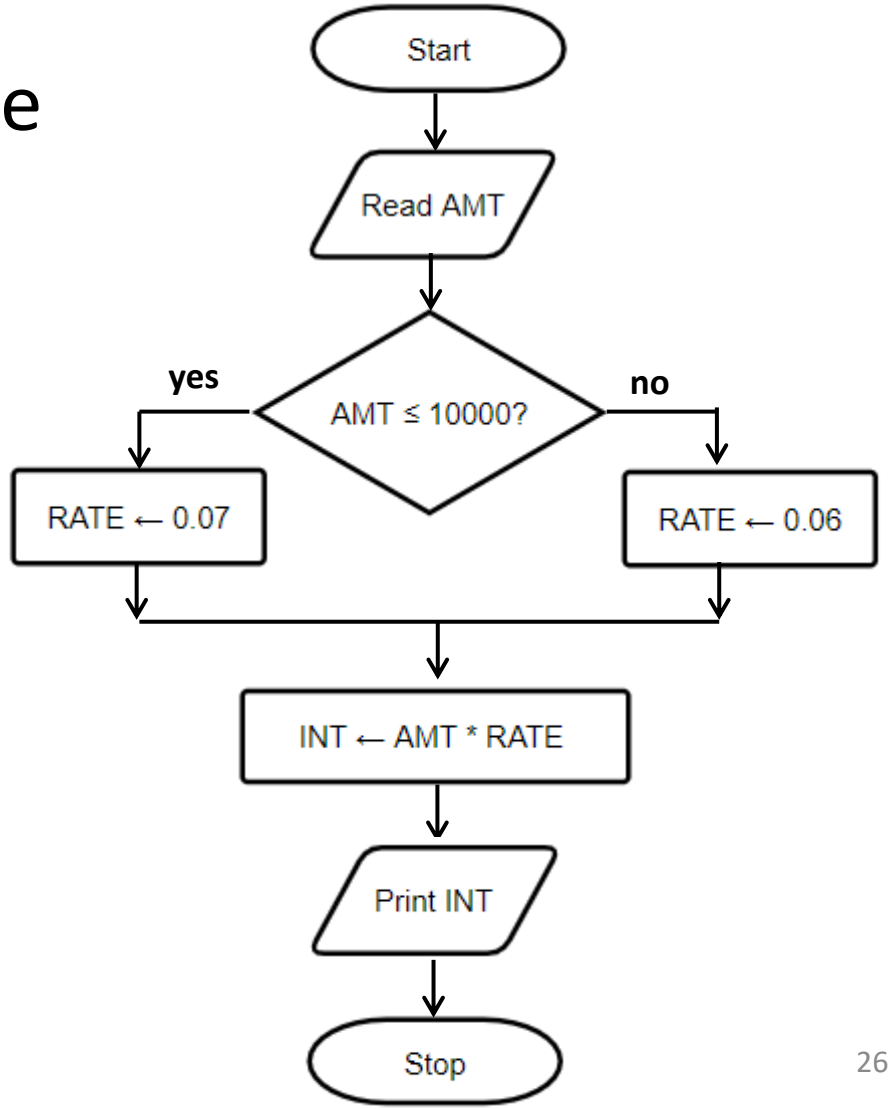
Connection

Hexagon

Start of loop body

Sample Flowchart

Flowchart that shows the computation of interest on a certain principal amount. The rate of interest is 7% if the principal amount is greater than $10,000 and 6% otherwise



Properties of Algorithms

Input: An algorithm typically has input values from a specified set.

Output: The algorithm performs an action or produces the output values from a specified set (the solution).

Correctness: An algorithm should produce the correct output values for each set of input values.

Finiteness: An algorithm should produce the output after a finite number of steps for any input.

Effectiveness: It must be possible to perform each step of the algorithm correctly and in a finite amount of time.

Generality: The algorithm should work for all problems of the desired form.

Finding the Maximum Element in a Finite Sequence

The algorithm in pseudocode:

procedure *max*(a_1, a_2, …., a_n: integers)

max := a_1

for *i* := 2 to *n*

    if *max* < a_i **then** *max* := a_i

return *max* {*max* is the largest element}

Some Example Algorithm Problems

Three classes of problems will be studied in this chapter.

1. *Searching Problems*: finding the position of a particular element in a list.
2. *Sorting problems*: putting the elements of a list into increasing order.
3. *Optimization Problems*: determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs.
    - An algorithm that quickly produces good, but not necessarily optimal solutions is called a *heuristic*. You will study them in depth in more advanced CS courses.