Algorithm Analysis
Data Structure Practice

Yonghui Wu
School of Computer Science, Fudan University
WeChat: 13817360465
yhwu@fudan.edu.cn

Chapter 1 Practice for Simple Computing

- 1.1 Improving programming Style
- 1.2 Multiple Test Cases
- 1.3 Precision of Real Number
- 1.4 Improving Time Complexity by Dichotomy

Goals for Chapter 1

- Master programming languages.
- Be familiar with online judge systems and programming environments.
- Begin to learn how to transfer a practical problem into a computing process, implement the computing process by a program, and debug the program to pass all test data.

1.1 Improving programming Style

- The pattern of a program
  - Input-Process-Output
- A problem for simple computing is a problem whose process is simple

1.1.1 Financial Management

- **Source:** ACM Mid-Atlantic 2001
- **IDs for Online Judge:** POJ 1004, ZOJ 1048, UVA 2362

- Larry graduated this year and finally has a job. He's making a lot of money, but somehow never seems to have enough. Larry has decided that he needs to grab hold of his financial portfolio and solve his financing problems. The first step is to figure out what's been going on with his money. Larry has his bank account statements and wants to see how much money he has. Help Larry by writing a program to take his closing balance from each of the past twelve months and calculate his average account balance.

- **Input**
- The input will be twelve lines. Each line will contain the closing balance of his bank account for a particular month. Each number will be positive and displayed to the penny. No dollar sign will be included.
- **Output**
- The output will be a single number, the average (mean) of the closing balances for the twelve months. It will be rounded to the nearest penny, preceded immediately by a dollar sign, and followed by the end-of-line. There will be no other spaces or characters in the output.

Analysis

The problem's pattern "Input-Process-Output" is very simple :

Firstly the income of 12 months $a[0..11]$ is inputted by a for statement

for(i=0;i<12;i++), and the total income $sum=\sum_{i=0}^{11} a[i]$ is calculated.

Then the average monthly income $avg=\frac{sum}{12}$ is calculated.

Finally *avg* is outputted in accordance with the problem's requirement.

Program

- The input and output of the program must meet formats for input and output
- A program should be readable
  - The style of a program should be serration based on logical level.
- The program annotations should be given.

1.2 Multiple Test Cases

- In order to guarantee correctness of a program, for most problems there are multiple test cases.
  - the number of test cases is given;
  - the mark of input end is given.

1.2.1 Doubles

- Source: ACM Mid-Central USA 2003
- IDs for Online Judge: POJ 1552, ZOJ 1760, UVA 2787

- As part of an arithmetic competency program, your students will be given randomly generated lists of from 2 to 15 unique positive integers and asked to determine how many items in each list are twice some other item in the same list. You will need a program to help you with the grading. This program should be able to scan the lists and output the correct answer for each one. For example, given the list
- 1 4 3 2 9 7 18 22
- your program should answer 3, as 2 is twice 1, 4 is twice 2, and 18 is twice 9.

- **Input**
- The input file will consist of one or more lists of numbers. There will be one list of numbers per line. Each list will contain from 2 to 15 unique positive integers. No integer will be larger than 99. Each line will be terminated with the integer 0, which is not considered part of the list. A line with the single number -1 will mark the end of the file. The example input below shows 3 separate lists. Some lists may not contain any doubles.
- **Output**
- The output will consist of one line per input list, containing a count of the items that are double some other item.

Analysis

- There are multiple test cases for the problem.
- A loop statement is used to deal with multiple test cases.
  - The loop enumerates every test case.
  - -1 marks the end of input.
- -1 is the end condition of the loop.

Analysis

- In the loop statement, there are two steps:
  - A loop inputs a test case into array a, and accumulates the number of elements n in the test case. 0 marks the end of the test case.
  - A double loop enumerates all pairs of $a[i]$ and $a[j]$ ($0 \le i < n-1$, $i+1 \le j < n$) in the test case, and determines whether ($a[i]*2==a[j]$ || $a[j]*2==a[i]$) holds.

Program

- The number of test cases and the size of a test case are unknown.
- A double loop statement is used for the program structure:
  - the outer loop is used to enumerate every test case;
  - the inner loop is used to deal with a test case;

off-line method

- In some problems, the size of test data is larger, all test cases are dealt with by same method and the result area is known, its time complexity can be improved by off-line method.
  - All solutions within the specified range are calculated and are stored in a constant array.
  - Program deals with the constant array directly for each test case.
- It can avoid duplication of computing.

1.2.2 Sum of Consecutive Prime Numbers

- Source: ACM Japan 2005
- IDs for Online Judge: POJ 2739, UVA 3399

- Some positive integers can be represented by a sum of one or more consecutive prime numbers. How many such representations does a given positive integer have? For example, the integer 53 has two representations 5 + 7 + 11 + 13 + 17 and 53. The integer 41 has three representations 2 + 3 + 5 + 7 + 11 + 13 , 11 + 13 + 17 , and 41 . The integer 3 has only one representation, which is 3. The integer 20 has no such representations. Note that summands must be consecutive prime numbers, so neither 7 + 13 nor 3 + 5 + 5 + 7 is a valid representation for the integer 20. Your mission is to write a program that reports the number of representations for the given positive integer.

- **Input**
- The input is a sequence of positive integers each in a separate line. The integers are between 2 and 10000, inclusive. The end of the input is indicated by a zero.
- **Output**
- The output should be composed of lines each corresponding to an input line except the last zero. An output line includes the number of representations for the input integer as the sum of one or more consecutive prime numbers. No other characters should be inserted in the output.

Analysis

- The program needs to deal with consecutive prime numbers for each test case
  - off-line method can be used to solve the problem
  - all prime numbers less than 10001 are gotten, and are stored in array *prime*[1.. *total*] in ascending order.

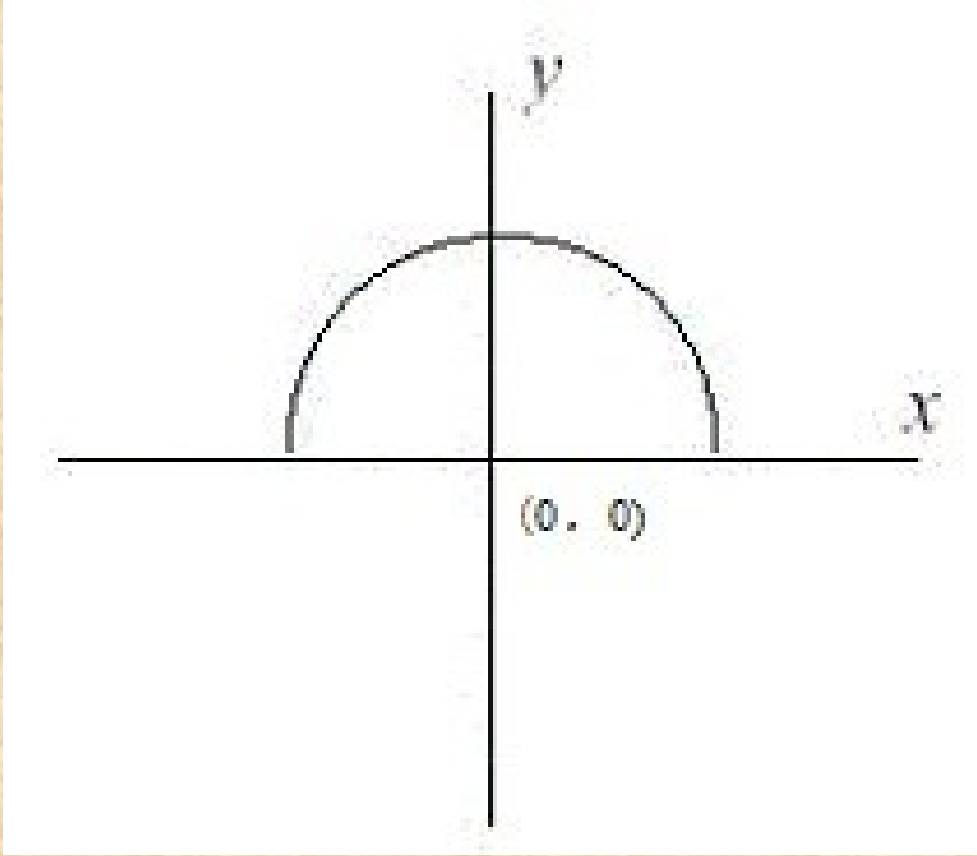
- Deal with test cases one by one:
  - Suppose the input number is *n*; the sum of consecutive prime numbers is *cnt*; the number of representations for *cnt*==*n* is *ans*.
  - A double loop is used to get the number of representations for *n*:
    - The outer loop *i*: for (int *i*=0; *n*>=*prime*[*i*]; *i*++) enumerates all possible minimum *prime*[*i*].
    - The inner loop *j*: for (int *j*=*i*; *j* < *total* && *cnt*<*n*; *j*++) *cnt* += *prime*[*j*]; is to calculate the sum of consecutive prime numbers. If *cnt*≥*n*, then the loop ends; and if *cnt*==*n*, then the number of representations *ans*++.
  - When the outer loop ends, *ans* is the solution to the test case.

1.3 Precision of Real Numbers

1.3.1 I Think I Need a Houseboat

- Sourse: ACM Mid-Atlantic 2001
- IDs for Online Judge: POJ 1005, ZOJ 1049, UVA 2363

- Fred Mapper is considering purchasing some land in Louisiana to build his house on. In the process of investigating the land, he learned that the state of Louisiana is actually shrinking by 50 square miles each year, due to erosion caused by the Mississippi River. Since Fred is hoping to live in this house the rest of his life, he needs to know if his land is going to be lost to erosion.
- After doing more research, Fred has learned that the land that is being lost forms a semicircle. This semicircle is part of a circle centered at (0, 0), with the line that bisects the circle being the X axis. Locations below the X axis are in the water. The semicircle has an area of 0 at the beginning of year 1. (Semicircle illustrated in the Figure 1.1.)


y
x
(0. 0)

- **Input**
- The first line of input will be a positive integer indicating how many data sets will be included (*N*).
- Each of the next *N* lines will contain the X and Y Cartesian coordinates of the land Fred is considering. These will be floating point numbers measured in miles. The Y coordinate will be non-negative. (0, 0) will not be given.

- **Output**
- For each data set, a single line of output should appear. This line should take the form of:
- Property *N*: This property will begin eroding in year *Z*.
- where *N* is the data set (counting from 1), and *Z* is the first year (start from 1) this property will be within the semicircle AT THE END OF YEAR *Z*. *Z* must be an integer. After the last data set, this should print out `END OF OUTPUT.'.

Analysis

The number of test cases n is given. Therefore a *for* repetition statement is used to deal with all test cases. Each teat case only contains X and Y Cartesian coordinates. The i-th test case (X_i, Y_i) and the center of the circle (0, 0) constitute the semi-circle that will be eroded. Each year 50 square miles' land is eroded. And the number of years is an integer. When (X_i, Y_i) is in water, the number of years must be the least integer which is greater than $\frac{\text{the area of the semi-circle}}{50}$, and function $ceil(x)$ is used to round up the fare.

1.4 Improving Time Complexity by Dichotomy

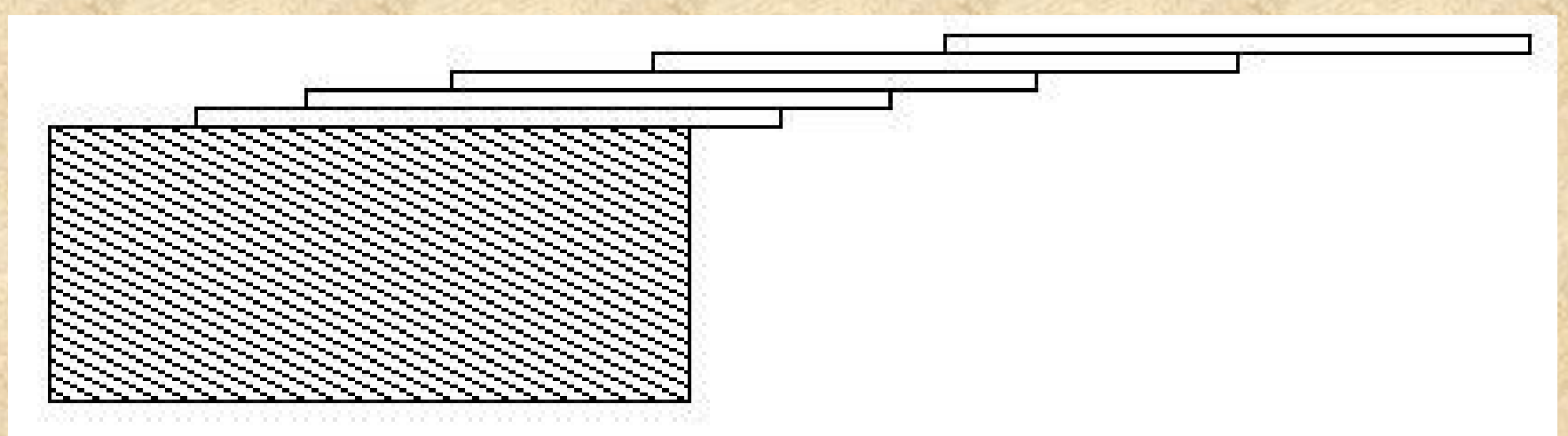
- In some cases, data area of a problem is an ordered interval. Dichotomy is used to divide the interval into two sub-intervals, and then determine the process of computation is in the left sub-interval or in the right sub-interval. If the solution isn't gotten, then repeat above steps. For a problem whose time complexity is $O(n)$, if dichotomy can be used to solve it, its time complexity can be improved to $O(\log_2(n))$.

- In real arithmetics, sometime we need to determine whether real number *x* and real number *y* is equal or not. Using *y*−*x*==0 as the condition maybe results in error of precision. The method avoiding error of precision is to set a constant of precision *delta*. If |*y*−*x*|<*delta*, then we can judge *x* and *y* are equal.

1.4.1 Hangover

- Source: ACM Mid-Central USA 2001
- IDs for Online Judge: POJ 1003, UVA 2294

- How far can you make a stack of cards overhang a table? If you have one card, you can create a maximum overhang of half a card length. (We're assuming that the cards must be perpendicular to the table.) With two cards you can make the top card overhang the bottom one by half a card length, and the bottom one overhang the table by a third of a card length, for a total maximum overhang of 1/2 + 1/3 = 5/6 card lengths. In general you can make n cards overhang by 1/2 + 1/3 + 1/4 + ... + 1/(n + 1) card lengths, where the top card overhangs the second by 1/2, the second overhangs the third by 1/3, the third overhangs the fourth by 1/4, etc., and the bottom card overhangs the table by 1/(n + 1). This is illustrated in the figure below.



- **Input**
- The input consists of one or more test cases, followed by a line containing the number 0.00 that signals the end of the input. Each test case is a single line containing a positive floating-point number c whose value is at least 0.01 and at most 5.20; c will contain exactly three digits.
- **Output**
- For each test case, output the minimum number of cards necessary to achieve an overhang of at least c card lengths. Use the exact output format shown in the examples.

Analysis

The problem's data area is little. Therefore firstly lengths that cards achieve are calculated, and the length is at most 5.20 card lengths. Suppose *total* is the number of cards, *len*[*i*] is the length that *i* cards achieve. That is, $len[i] = len[i-1] + \frac{1}{i+1}$, where $i \geq 1$ and $len[0]=0$. Obviously array *len* is in ascending order.

Because elements of *len* and *x* are real numbers, the accuracy error must be controlled. Suppose *delta*=1e-8, and function *zero*(*x*) marks *x* is a positive real number, a negative real number, or a zero. And function *zero*(*x*) is defined as follow:

$$zero(x)=\begin{cases} 1 & x > delta \\ -1 & x < -delta \\ 0 & Otherwise \end{cases}$$

- Initially *len*[0]=0. Array *len* can be gotten through the following loop:
- for(*total*=1; *zero*(*len*[*total*-1]-5.20)<0; *total*++)
- *len*[*total*]=*len*[*total*-1]+1.0/double(*total*+1);
- After array *len* is gotten, the program inputs the first test data *x* and enters the loop of *while* (*zero*(*x*)). In each loop dichotomy is used to get the minimum number of cards necessary to achieve an overhang of at least *x* card lengths, and then the next test data *x* is inputted. The loop terminates when *x*=0 .00.

The procedure of dichotomy is as follow:

The initial interval $[l, r]=[1, total]$ and $mid=\left\lfloor \frac{l+r}{2} \right\rfloor$. If $zero(len[mid] - x) <$ 0, then search the right half ($l=mid$); otherwise search the left half ($r=mid$). Repeat above steps in interval $[l, r]$ until $l+1 \geq r$. And r is the minimum number of cards.

Dichotomy can be used not only in data search, but also in function calculation. Suppose there are variables x_1, x_2 and x_3; and function $x_1=f(x_2, x_3)$ holds. Recursive halving method can be used to calculate x_3 when x_1 and x_2 are known. The method is as follow.

"Halving" is to halve data area of the problem (such as data area of x_3), and the property of problem (such as $x_1=f(x_2, x_3)$) is not changed. Suppose the size of data area for the problem is n. We can firstly make use of some methods to change the original problem into c sub-problems with half data area (c is a constant, is related to the problem, and is not related to data area), then solve the problem by solving sub-problems whose size of data area is $\frac{n}{2}$. Properties for these sub-problems are the same as the original problem, but the size of data area for these sub-problems are smaller.

"Recursion" is to repeat above "halving" steps. A problem whose size of data area is $\frac{n}{2}$ is changed into c sub-problems whose size is $\frac{n}{4}$; and so on. Repeat above steps until sub-problems can be solved easily.

1.4.2 Humidex

- **Source: Waterloo Local Contest, 2007.7.14**
- **IDs for Online Judge: POJ 3299**

- The humidex is a measurement used by Canadian meteorologists to reflect the combined effect of heat and humidity. It differs from the heat index used in the United States in using dew point rather than relative humidity.
- When the temperature is 30 C (86 F) and the dew point is 15 C (59 F), the humidex is 34 (note that humidex is a dimensionless number, but that the number indicates an approximate temperature in C). If the temperature remains 30 C and the dew point rises to 25 C (77 F), the humidex rises to 42.3.
- The humidex tends to be higher than the U.S. heat index at equal temperature and relative humidity.
-

- The current formula for determining the humidex was developed by J.M. Masterton and F.A. Richardson of Canada's Atmospheric Environment Service in 1979.
- According to the Meteorological Service of Canada, a humidex of at least 40 causes "great discomfort" and above 45 is "dangerous." When the humidex hits 54, heat stroke is imminent.
- The record humidex in Canada occurred on June 20, 1953, when Windsor, Ontario hit 52.1. (The residents of Windsor would not have known this at the time, since the humidex had yet to be invented.) More recently, the humidex reached 50 on July 14, 1995 in both Windsor and Toronto.

- The humidex formula is as follows:
- humidex = temperature + *h*
- *h* = (0.5555)*(e − 10.0)
- *e* = 6.11 * *exp* [5417.7530 * ((1/273.16) − (1/(dewpoint+273.16)))]
- where *exp*(*x*) is 2.718281828 raised to the exponent *x*.
- While humidex is just a number, radio announcers often announce it as if it were the temperature, e.g. "It's 47 degrees out there ... [pause] .. with the humidex,". Sometimes weather reports give the temperature and dew point, or the temperature and humidex, but rarely do they report all three measurements. Write a program that, given any two of the measurements, will calculate the third.
- You may assume that for all inputs, the temperature, dew point, and humidex are all between −100 C and 100 C.

- **Input**
- Input will consist of a number of lines. Each line except the last will consist of four items separated by spaces: a letter, a number, a second letter, and a second number. Each letter specifies the meaning of the number that follows it, and will be either T, indicating temperature, D, indicating dew point, or H, indicating humidex. The last line of input will consist of the single letter E.

- **Output**
- For each line of input except the last, produce one line of output. Each line of output should have the form:
- T number D number H number
- where the three numbers are replaced with the temperature, dew point, and humidex. Each value should be expressed rounded to the nearest tenth of a degree, with exactly one digit after the decimal point. All temperatures are in degrees celsius.

Analysis

- Based on humidex formula *humidex* = *temperature* + *h*, *h* is proportional to dew point. If dew point and temperature (or humidex) are known, the value of *h* can be inferred, and humidex or temperature can be calculated by the humidex formula. If temperature and humidex are known, recursive halving method can be used to calculate dew point:

- The program sets initial value 0 to the dew point, and enters a loop: the initial value of increment for dew point is 100, halve the increment value each time as the loop is performed. If humidex gotten from the formula is larger than the announced humidex, then the value of dew point decreases an increment (that is, $h\searrow$, decrease the humidex to be close to the announced humidex.); otherwise the value of dew point increases an increment (that is, $h\nearrow$, increase the humidex to be close to the announced humidex.). The repetition condition is the increment value greater than 0.0001. When the loop ends, the dew point is the answer.