Chapter 2 Simple Simulation

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

Chapter 2 Simple Simulation

- In the real world, there are many problems that we can solve by simulating their processes. Such problems are called simulation problems.
- For these problems, solution procedures and rules are showed in problem descriptions.
- Programs must simulate procedures or implement rules based on descriptions.

- There are three kinds of process simulations:
    - Simulation of Direct Statement
    - Simulation by Sieve Method
    - Simulation by Construction

2.1 Simulation of Direct Statement

- For problems for simulation of direct statement, programmers are required to solve these problems strictly following rules showed in the problems' descriptions.
- Programmers must read such problems carefully, and simulate processes based on descriptions.
- A problem for simulation of direct statement gets harder as the number of rules increases. It causes the amount of code to grow up and get more illegible.

2.1.1 Speed Limit

- **Source: ACM Mid-Central USA 2004**
- **IDs for Online Judge: POJ 2017, ZOJ 2176, UVA 3059**

- Bill and Ted are taking a road trip. But the odometer in their car is broken, so they don't know how many miles they have driven. Fortunately, Bill has a working stopwatch, so they can record their speed and the total time they have driven. Unfortunately, their record keeping strategy is a little odd, so they need help computing the total distance driven. You are to write a program to do this computation.

For example, if their log shows

Speed in miles perhour	Total elapsed time in hours
20	2
30	6
10	7

this means they drove 2 hours at 20 miles per hour, then 6-2=4 hours at 30 miles per hour, then 7-6=1 hour at 10 miles per hour. The distance driven is then (2)(20) + (4)(30) + (1)(10) = 40 + 120 + 10 = 170 miles. Note that the total elapsed time is always since the beginning of the trip, not since the previous entry in their log.

- **Input**
- The input consists of one or more data sets. Each set starts with a line containing an integer *n*, 1 <= *n* <= 10, followed by *n* pairs of values, one pair per line. The first value in a pair, *s*, is the speed in miles per hour and the second value, *t*, is the total elapsed time. Both *s* and *t* are integers, 1 <= *s* <= 90 and 1 <= *t* <= 12. The values for *t* are always in strictly increasing order. A value of -1 for *n* signals the end of the input.
- **Output**
- For each input set, print the distance driven, followed by a space, followed by the word "miles".

Analysis

- A simple problem of direct statement.
  - Simulate the stopwatch's running to compute the total distance driven:
    - the last "total elapsed time in hours" is z,
    - the current "speed in miles per hour" is x,
    - the current "total elapsed time in hours" is y,
  - then the current distance driven is $(y-z)*x$, and add it to the total distance driven.

2.1.2 Ride to School

- **Source: ACM Beijing 2004 Preliminary**
- **IDs for Online Judge: POJ 1922, ZOJ 2229**

- Many graduate students of Peking University are living in Wanliu Campus, which is 4.5 kilometers from the main campus – Yanyuan. Students in Wanliu have to either take a bus or ride a bike to go to school. Due to the bad traffic in Beijing, many students choose to ride a bike.
- We may assume that all the students except "Charley" ride from Wanliu to Yanyuan at a fixed speed. Charley is a student with a different riding habit – he always tries to follow another rider to avoid riding alone. When Charley gets to the gate of Wanliu, he will look for someone who is setting off to Yanyuan. If he finds someone, he will follow that rider, or if not, he will wait for someone to follow. On the way from Wanliu to Yanyuan, at any time if a faster student surpassed Charley, he will leave the rider he is following and speed up to follow the faster one.
- We assume the time that Charley gets to the gate of Wanliu is zero. Given the set off time and speed of the other students, your task is to give the time when Charley arrives at Yanyuan.

- **Input**
- There are several test cases. The first line of each case is N ($1 <= N <= 10000$) representing the number of riders (excluding Charley). $N = 0$ ends the input. The following N lines are information of N different riders, in such format
- V_i [TAB] T_i
- V_i is a positive integer <= 40, indicating the speed of the *i*-th rider (kph, kilometers per hour). T_i is the set off time of the *i*-th rider, which is an integer and counted in seconds. In any case it is assured that there always exists a nonnegative T_i .
- **Output**
- Output one line for each case: the arrival time of Charley. Round up (ceiling) the value when dealing with a fraction.

Analysis

- There is no mathematical formula to solve the problem. We can calculate the arrival time of Charley by simulating each student leaves from Wanliu to Yanyuan. For each test case, the time that Charley gets to the gate of Wanliu is zero. From it we calculate the arrival time of each student. Obviously the earliest arrival time is the arrival time of Charley.

Suppose *min* is the earliest arrival time for the first i-1 riders, the speed of the i-th rider is v, and the set off time of the i-th rider is t. Then the time when the i-th rider arrives at Yanyuan $x=t+\frac{4.5*3600}{v}$. If $x<min$, then *min* is adjusted as x. Obviously, after the arrival time of all riders is calculated, *min* is the arrival time of Charley.

There is a trap in the test data. If T_i is a negative integer, we should neglect it. It doesn’t affect the arrival time of Charley.

2.2 Simulation by Sieve Method

- Simulation by sieve method
  - to get constraints in the problem description and such constraints constitute a sieve.
  - all possible solutions are put on the sieve to filter out solutions which do not meet constraints from time to time.
- Finally solutions settling on the sieve are solutions to the problem.

- The structure and idea for simulation by sieve method is concise and clear, but it is also blindly. Therefore maybe the time efficiency is not good.
- The key to simulation by sieve method is to find the constraints.
  - Any errors and omissions will lead to failure.
  - Filtering rules do not need complex algorithm design, such problems are usually simple simulation problems.

2.2.1 Self Numbers

- **Source: ACM Mid-Central USA 1998**
- **IDs for Online Judge: POJ 1316, ZOJ 1180, UVA 640**

- In 1949 the Indian mathematician D.R. Kaprekar discovered a class of numbers called self-numbers. For any positive integer *n*, define *d*(*n*) to be *n* plus the sum of the digits of *n*. (The *d* stands for digitadition, a term coined by Kaprekar.) For example, *d*(75) = 75 + 7 + 5 = 87. Given any positive integer *n* as a starting point, you can construct the infinite increasing sequence of integers *n*, *d*(*n*), *d*(*d*(*n*)), *d*(*d*(*d*(*n*))), ... For example, if you start with 33, the next number is 33 + 3 + 3 = 39, the next is 39 + 3 + 9 = 51, the next is 51 + 5 + 1 = 57, and so you generate the sequence
- 33, 39, 51, 57, 69, 84, 96, 111, 114, 120, 123, 129, 141, ...
- The number *n* is called a generator of *d*(*n*). In the sequence above, 33 is a generator of 39, 39 is a generator of 51, 51 is a generator of 57, and so on. Some numbers have more than one generator: for example, 101 has two generators, 91 and 100. A number with no generators is a self-number. There are thirteen self-numbers less than 100: 1, 3, 5, 7, 9, 20, 31, 42, 53, 64, 75, 86, and 97.

- **Input**
- No input for this problem.
- **Output**
- Write a program to output all positive self-numbers less than 10000 in increasing order, one per line.

Analysis

- Simulation by sieve method is used to solve the problem.
- Suppose the sieve is array g, where $g[y]=x$ means y is a number in ascending sequence for x.
- Based on “$d(x) = x$ + the sum of the digits of x”, a subprogram *generate_sequence*(x) is to generate the ascending sequence [$d(x)$, $d(d(x))$, $d(d(d(x)))$, ...] for x.
- Suppose x is the generation number for every number in the sequence:
- $g[d(x)]=g[d(d(x))]=g[d(d(d(x)))]=....=x$

- If a number is in ascending sequence for *x*, it is not a self-number and should be sieved from sieve *g*. The process will repeat until the generated number ≥1000 or the generated number has been generated before ($g[x] \neq x$). If *x* has been generated, it is not a self-number.

- The algorithm is as follows:
  - $g[i]$ is initialized as i ($1 \le i \le 1000$).
  - *generate_sequence*(1) ... *generate_sequence*(1000) are called to calculate $g[1..1000]$.
  - numbers left in the sieve, that is, $g[x]==x$, are self-numbers.

2.3 Construction Simulation

- Construction simulation is a kind of relatively complex simulation method. It requires a complete and accurate mathematical model to represent and solve a problem. We need to design parameters of the model, and calculate simulation result. Because such mathematical models represent objects and their relationships accurately, the efficiencies are relatively high.

2.3.1 Bee

- **IDs for Online Judge: UVA 11000**

- In Africa there is a very special species of bee. Every year, the female bees of such species give birth to one male bee, while the male bees give birth to one male bee and one female bee, and then they die!
- Now scientists have accidentally found one "magical female bee" of such special species to the effect that she is immortal, but still able to give birth once a year as all the other female bees. The scientists would like to know how many bees there will be after *N* years. Please write a program that helps them find the number of male bees and the total number of all bees after *N* years.

- **Input**
- Each line of input contains an integer *N* (≥ 0). Input ends with a case where *N* = -1. (This case should NOT be processed.)
- **Output**
- Each line of output should have two numbers, the first one being the number of male bees after *N* years, and the second one being the total number of bees after *N* years. (The two numbers will not exceed 2^{32}.)

Analysis

- From the description of bees’ breeding, it is a problem of process simulation. Because bees’ breeding is based on rules, the corresponding mathematical model can be constructed. Therefore it is also a problem of construction simulation.

- There is only one integer for a test case. And -1 marks the end of input. After the first test case is input, there is a *while* repetition statement *while* (n > -1). In the loop body, the calculation process is as follows:
  - Initialize the number of female bees a as 1, and the number of male bees b as 0. Because of the size of operation, the type of a and b is long long.
  - Making a series of recurrences for i from 0 to n-1. After i+1 years, the number of female bees is the number of the last year's male bees +1, and the number of male bees is the number of last year's bees. Therefore formulas are as follows:
  - $c = 1 + b; d = a + b; a = c; b = d;$
  - Output the number of male bees a and the number of bees $a+b$ after N years;
  - Input the next test case;