GB20602 - Programming Challenges

Week 1 - Programming Challenge Tips

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Sciences

(last updated: April 13, 2024)

Version 2022

Programming Challenges Tips and Tricks

Week 1: Tips for Programming Challenges

First week: 7 Easy problems (1 long problem)! What do you need to learn?

1. How to read a program challenge? (Problem style)
2. How to use the Automated Judge? (Input and Output)
3. How to debug? (Thinking and Debug Data)

The class one material contains General Tips for this course.

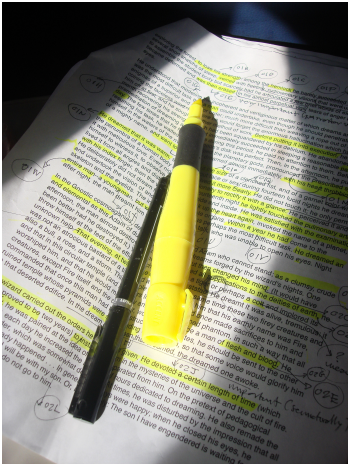
These Tips will be useful for the rest of the course. Learn them well!

Topics for today

- Tip 1: Reading the problem;
- Tip 2: Problem size and algorithm analysis;
- Tip 3: Writing test cases;
- Tip 4: Input/Output;
- Tip 5: How to ask for help;

Tip 1: Reading, Thinking, then Coding

- It is very important to read and understand the problem before start coding.
- If you completely understand the problem, you can avoid small mistakes.
- Maybe read the Input and Output first!
- Try to solve the problem on paper
- If you don't understand the problem, ask for help!
- Quickly read all 8 problems before starting. Different people like different problems.



Hint 2: Be careful of problem size!

Some of the secret input sizes are very big.

If your program is too slow, you cannot solve big input sizes.

When you read the problem, pay attention to the input size.

Think about the the number of loops that your program uses!

Program is too slow: Example 1

Calculate the Maximum Execution Length between any two numbers *i* and *j*.

```
Execution of (n)
1. print n
2. if n == 1 then STOP
3. if n is odd then n = 3n + 1
4. else n = n/2
5. GOTO 2

Example: Execution of (22)
22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 -- Execution Length = 16
```

For ONE input, the problem is small and easy.

A simple solution for the 3n+1 problem

Simple solution: loop until n == 1 and count the loops (lines 9-12).

However, for large input, this program is too slow.

Remember: You have to execute the program for every input between *i* and *j*

```
 1 while true:
 2  try:
 3    line = input()
 4    max = 0
 5    tk = line.split()
 6    i, j = int(tk[0]), int(tk[1])
 7    for n in range(i, j+1):
 8      count = 1
 9      while n != 1:
10        if n % 2 == 1: n = 3 * n + 1
11        else: n = n / 2
12        count += 1
13      if count > max: max = count
14    print (line, max)
15  except EOFError: break
```

Large input in the 3n+1 problem

- Objective: Calculate the largest execution between *i* and *j*. For a large input:

```
Input: 1 10000 <-- Largest between 1 and 10000?
Output: 262    <-- Largest is 262 steps
```

- 262 steps for 1 execution is short.
  But for 10000 inputs: $262 \times 10000 = 2,000,000$ steps!
- Can we make it faster?

Make 3n+1 faster by removing repetition

- Some examples of calculating (A(n)) – Note the repetition:

```
A(22):  22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
A(11):  11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
A(17):  17 52 26 13 40 20 10 5 16 8 4 2 1
A(13):  13 40 20 10 5 16 8 4 2 1
A(10):  10 5 16 8 4 2 1
A(8) :  8 4 2 1
```

- To avoid the repetition, we store partial results in an array.
- This technique is called Memoization. It is very useful for many problems!

Memoization for 3n+1

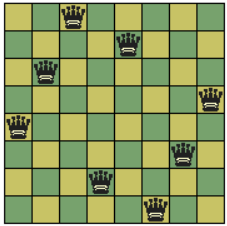
Use a list or dictionary to memorize past results (memo table). Check the memo table before calculating results.

```
memotable = {}
memotable[1] = 1

def A(n):
  if n in table.keys(): return table[n]
  else:
    if n % 2 == 1: table[n] = 1 + A(3*n + 1)
    else: table[n] = 1 + A(n/2)
  return table[n]
```

More about Memo Tables in Lecture 4 (Dynamic Programming)

Problem Size Example: 8 Queen Problem



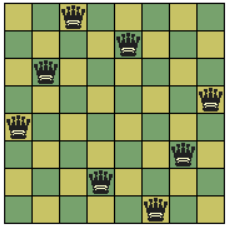
8-Queen problem: Place 8 queens in a chess board, so that all columns, all rows, and all diagonals are different

Algorithm (1): Eight loops size 64: Try all rows and all columns for queen 1, queen 2, queen 3, queen 4, ... etc.

```
Choices for Queen1: a1, a2, a3, a4, ..., h6, h7, h8
Choices for Queen2: a1, a2, a3, a4, ..., h6, h7, h8
Choices for Queen3: a1, a2, a3, a4, ..., h6, h7, h8
...
Choices for Queen8: a1, a2, a3, a4, ..., h6, h7, h8

Total Number of choices: 64 x 64 x 64 x 64 = 64^8 ~ 10^14
                                              TOO MANY!
```

Reducing the size of 8-Queen (1)



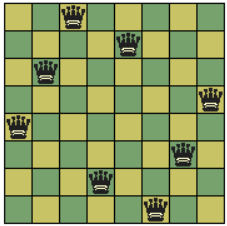
To reduce the number of choices, we force each queen to be on a different row, and only choose the column.

Algorithm (2): Eight loops size 8: Try all columns for queen 1 (fix row 1), all columns for queen 2 (fix row 2), ..., etc.

```
Choices for Queen1: a1, b1, c1, d1, e1, f1, g1 or h1
Choices for Queen1: a2, b2, c2, d2, e2, f2, g2 or h2
Choices for Queen1: a3, b3, c3, d3, e3, f3, g3 or h3
...
Choices for Queen8: a8, b8, c8, d8, e8, f8, g8 or h8

Total Solutions: 8 x 8 x 8 ... = 8^8 ~ 10^7
                                  Still Big!
```

Reducing the size of 8-Queen (2)



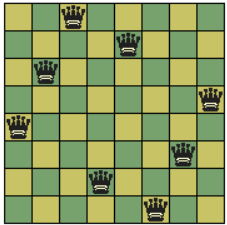
We can reduce the number of choices even more, if we force the columnd and the row of each queen to be different.

Algorithm (3): Fix the row for each queen, 1 loop through the permutation of columns.

```
Choice 1: {a1, b2, c3, d4, e5, f6, g7, h8}
Choice 2: {a1, b2, c3, d4, e5, f6, g8, h7}
Choice 3: {a1, b2, c3, d4, e5, f7, g6, h8}
Choice 4: {a1, b2, c3, d4, e5, f7, g8, h6}
...

Total Solutions: 8 x 7 x 6 x 5 ... = 40320  --  OK!
```

Reducing the size of 8-Queen: Summary



- Algorithm 1: All rows and columns: 10^{14} steps;
- Algorithm 2: Fix rows, All columns: 10^{7} steps;
- Algorithm 3: Permutation: 40320 steps;

Hint 3: Write testcases!

The description of a problem includes Sample Input.

However, the judge checks your program with Test Input.

Example: 3n+1 Problem

Sample Input: always $i < j$.

Hidden Input: sometimes $i > j$!

The Test Input is more difficult than the Sample Input!

The Test Input is more difficult than the Sample Input!

- Input is Negative; Input is Minimum; Input is Maximum;
- Input is out of order, Input is in reverse order;
- Input is repeated;
- Graph is disconnected; Graph is fully connected;
- Lines are parallel; Points are in the same place;
- Angle is 0; Area is 0;

Read the input rule carefully.

Create your own Debug Input

Creating Debug Input

Create and test a Debug Input before submitting the program!

Many wrong programs can be fixed if you create debug input before submitting.

Again: Create Debug Input!

Ideas for Debug Input (checklist)

- Simple Debug Input;
- Debug Input with maximum value;
- Debug Input with tricky cases;
- Random Debug Input Input (use scripts!);

Creating debug input is an important skill for research and work!

Tip 4: Read the Input and Output description carefully

- What is the stop condition?
  - The number of inputs is given in the beginning;
  - Special value to indicate the end of the input;
  - Input ends at the end of the file (EOF);
- What is the format of the input/output?
  - Very important for output (floating point precision)
- What is the size of the input?
  - What is the maximum number of inputs?
  - What are the maximum and minimum values?
  - Are there special conditions?

There are 3 common stop conditions

1. Read N, then read N queries;

```
int n; cin >> n;
for (; n > 0; n--) { // Do something }
```

2. Read until a special condition; (example, until $n = 0$)

```
int n; cin >> n;
while (n != 0) { // Do something
cin >> n; }
```

3. Continue reading until no more Input (Until EOF);

```
int a, b;
while (cin >> a >> b;) { // Do something! }
```

Output: You need to return EXACTLY what the problem requires

Be careful with the common mistakes:

- REMOVE PRINT DEBUG.
- UPPERCASE vs lowercase
- Singular vs plural
- Start with 1 vs start with 0
- Number of float digits: 0.34 x 0.340
- Round up or round down?: 0.34 x 0.35
- Solution order;
- etc...

Tip 5: How to ask for help

When you are stuck, it is okay to ask for help from the teacher or from your friends.

When you ask for help, add as much information as you can about your problem.

(This is a training for bug reporting as well!)

Help Checklist

1. Send your current code. (Do not send screenshot!)
2. Explain the idea of your program.
3. Explain the Debug Input you used.

Final message

Thank you for listening!

Have fun writing programs.

Be faster than your friends!

Ask questions on manaba.

About these Slides

Individual images in some slides might have been made by other authors. Please see the following pages for details.

Image Credits I