# GB21802 - Programming Challenges

## Week 1 - Ad Hoc Problems

Claus Aranha
caranha@cs.tsukuba.ac.jp

Department of Computer Science

2020/4/28
(last updated: April 23, 2020)

# Week 1 – Part 1: Class Introduction

# Outline

1. Outline and goal of the Class
2. What are programming Challenges?
3. Initial Example
4. Class Program
5. Lecturer Introduction
6. Extra: ICPC

# Class goal: Improve our programming skills!

# Learning Algorithms and Practicing Algorithms

# Automated Judging

# Class Expectations

# What is this course about?

You have learned many programming techniques...

...but can you use them?

## Course Objective: Learning by Practice

- Every week: Solve 8 programming challenges;
- Choose and implement the **best algorithm** for each problem;
- Be careful with max time, and max memory;
- We will discuss algorithms, techniques and tricks;

## Course Goal:

Improve programming abilities, techniques and familiarity.

# What are Programming Challenges

# Programming Challenges in the world

# Example Challenge

# Class Topics

# Class Format

# About the Lecturer

- Name: Claus Aranha;
- Country: Brazil;
- Research:
  - Evolutionary Algorithms;
  - Artificial Life;
- Hobbies:
  - Programming Games;
  - Watching Stars;
- webpage:
  `http://conclave.cs.tsukuba.ac.jp`

# Join the Tsukuba ICPC Team!

# Join the Tsukuba ICPC Team!

# Join the Tsukuba ICPC Team!

# Week 1 – Part 2: How this lecture is organized

# Outline

1. Class Schedule
2. Class Materials
3. How to submit problems
4. Grading
5. Office Hours and Teacher Communication
6. Special Distance Learning in 2020

# What you will do every week

# Class Dates and Deadlines

# Lecture Notes and Manaba

# Online Judge Site

# Course Language

# Reference Books

# Submitting Assignments: Outline

# What is OnlineJudge.org?

# Submitting Problems to OnlineJudge.org

# Attention: Java users

# Reading the Judge Results

# Submitting your code to Manaba

# Grading Outline

# Base Grade

# Late Penalty

# Best of Class Bonus

# Plagiarism Warning

# Teacher Communication

# Using Manaba

# Special Considerations for 2020

# Video Lectures and Lecture times

# Submission Deadline and Programming Environment

# Week 1 – Part 3: AdHoc Problems

# Today's Class: Ad Hoc Problems

- Ad Hoc: Problems that don't have a single algorithm;
- Common forms of Input and Output
- Debugging and Creating Test Cases
- Some Problem Analysis

# Reading the Problem: 3n+1

For any two numbers *i* and *j*, calculate the Maximum Cycle Length.

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

Size: 16
```

# Problem 3n+1 – Solution 1

```
while true:
  try:
    line = input()
    max = 0
    tk = line.split()
    i, j = int(tk[0]), int(tk[1])
    for n in range(i, j+1):
      count = 1
      while n != 1:
        if n % 2 == 1: x = 3 * x + 1
        else: n = n / 2
        count += 1
      if count > max: max = count
      print (line, max)
  except EOFError: break
```

# Solution 1: Problem!

- Solution 1 solves all Sample Input correctly.
- But we still get Wrong Answer!
- Why??? :-(

# Solution 1: Traps!

- Solution 1 solves all Sample Input correctly.
- If we try the input: 20 10 – output is nothing!

```
for x in range(i, j+1):    <-- Error is here!
  ...
  print (line, max)
```

- The sample input has no examples with "$i > j$"!

# Solution 1: Traps!

## Input Description

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 10,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.

You can assume that no operation overflows a 32-bit integer.

The only rules are those expressely written!

# Solution 1: Traps!!!!!

- First rule of Programming Challenges:
  Assume Worst Case
- Always try the worst case input!
  - Number is Negative; Number is zero;
  - Number is out of order (or in order);
  - Number is repeated;
  - Graph is unconnected; Graph is Fully connected;
  - Lines are parallel; Points are in the same place;
  - Area is 0; Angle is 0;
  - Input is very long;
  - Input is very short;
- If it is not against the rules: It will happen!
- This also happens in programs in the real world.

# Problem 3n+1 – Solution 2

```
while true:
  try:
    line = input()
    max = 0
    tk = line.split()
    i, j = int(tk[0]), int(tk[1])
    for n in range(min(i, j), max(i, j)+1): # FIXED
      count = 0
      while n != 1:
        if n % 2 == 1: n = 3 * n + 1
        else: n = n / 2
        count += 1
      if count > max: max = count
      print (line, max)
  except EOFError: break
```

# Solution 2: Time Limited Exceeded!

- All the inputs are correct.
- But now we have Time Limited Exceeded
- Why?

# Solution 2: Cost Calculation

**Input Description**

All integers will be less than 10,000 and greater than 0.

You can assume that no operation overflows a 32-bit integer.

- What happens when the input is 1 10000?

```
1 10000 262
```

- The longest sequence in 1, 10000 has 262 steps.
- But we calculate all sequences between 1, 10000.
- Worst case: 10000*262 = 2,000,000 steps!

- For only one query!

# Solution 2: Memoization

- Let's think about A();

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

- Do we need to Recalculate every time we see a new number?

**Good Technique: Memoization**

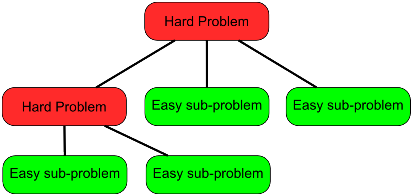
If we know that we will use a result again in the future, we should store this result in the memory.

# Solution 2: Memoization Idea

```
table = {}
table[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]
```

# A Programming Challenge Workflow




Trick to solve problems without bugs: Break the problem down

- How to calculate the function $A()$;
- Imagine the worst possible cases ($i > j$);
- Calculate cost of solution;
- Improve speed with memoization;

# A Programming Challenge Workflow

Common Steps for Programming challenges:

- Task 1: Read the problem description;
- Task 2: Read the input/output;
- Task 3: Think about the algorithm;
- Task 4: Write the Code;
- Task 5: Test the program on example data;
- Task 6: Test the program on hidden data;

# Task 1: Understanding the Problem Description

The English description is so hard!

**Don't Worry:**

1. Separate the text into flavor and rules;
2. Sometimes it is easy to read the input/output first, and then the text;
3. Problems with a lot of flavor are usually not very hard.;

# Example: Problem 11559 – Event Planning

**Flavor:**

As you didn't show up to the yearly general meeting of the Nordic Club of Pin Collectors, you were unanimously elected to organize this years excursion to Pin City. You are free to choose from a number of weekends this autumn, and have to find a suitable hotel to stay at, preferably as cheap as possible.

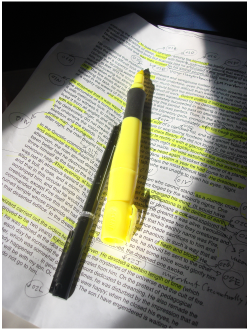
**rules**

You have some constraints: The total cost of the trip must be within budget, of course. All participants must stay at the same hotel, to avoid last years catastrophe, where some members got lost in the city, never being seen again.

**Keywords:** constraints, minimum, maximum, cost, rules, number, etc...

# Hints for hard to read problems

- First, look at the sample input and output;
- Write the idea of the problem on paper
- Use the Paper: mark keywords;
- Use the Paper: cut flavor;
- Read the problem again!;
- Do not begin programming until you understand the problem!



# Task 2: Reading the Input/Output

The Input Description is very important (as we saw in 3n+1)

- What is the size of the data?
- What are the limits of the data?
- What is the format of the data?
- What is the stop condition?

# Task 2: Input Size

The input size shows how big the problem gets.

- Small Problem: Full Search; Simulation;
- Big Problem: Complex Algorithms; Pruning;

**Keep in Mind!**

- The average time limit in UVA is 1-3 seconds.
- Expect maybe 10.000.000 operations per second.

# Task 2: Input Size – Examples

**n $<$ 24**

Exponential algorithms will work ($O(2^n)$).
Or sometimes you can just calculate all solutions.

**n = 500**

Cubic algorithms don't work anymore ($O(n^3) = 125.000.000$)
Maybe $O(n^2 \log n)$ will still work.

**n = 10.000**

A square algorithm ($O(n^2)$) might still work.
But beware any big constants!

**n = 1.000.000**

$O(n \log n) = 13.000.000$
We might need a linear algorithm!

# Task 2: Input Format

Three common patterns for input format:

- Read $N$, then read $N$ queries;
- Read until a special condition;
- Read until EOF;

# Task 2: Input Format

Read $N$, and then read $N$ queries;

Remember $N$ when calculating the size of the problem!

Example: Cost Cutting

```
#include <iostream>
using namespace std;

int main()
{
    int n;
    cin >> n;

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

# Task 2: Input Format

Read Until a Special Condition.
Be careful: You can have many queries before the condition!

Example: Request for Proposal:
The input ends with a line containing two zeroes.

```
int main()
{
   cin >> n >> p;
   while (n!=0 || p!=0)
   {
      // do something!
      cin >> n >> p;
   }
}
```

# Task 2: Input Format

Read until EOF.

Functions in C and Java return FALSE when they read EOF.
Python requires an exception. Very common in UVA.

Example: 3N+1 Problem, Jolly Jumpers

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

# Task 2: Output Format

The UVA judge decides the result based on a simple diff.

Be very careful that the output is exactly right!

*The Judge is like an angry client. It wants the output EXACTLY how it stated.*

# Task 2: Output Format – Checklist

1. DID YOU REMOVE DEBUG OUTPUT?
2. DID YOU REMOVE DEBUG OUTPUT?
3. Easy mistakes: UPPERCASE x lowercase, spleling mitsakes;
4. Boring mistakes: plural: 1 hour or 2 hours;
5. What is the precision of float? (3.051 or 3.05)
6. Round up or Round down? (3.62 $\rightarrow$ 3 or 4)
7. Multiple solutions: Which one do you output (usually ortographical sort)

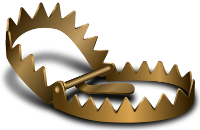
# Task 2: Input/Output – Traps!

## Example: 3n+1 Problem

- *i* and *j* can come in any order.

## Common Traps

- Negative numbers, zeros;
- Duplicated input, empty input;
- No solutions, multiple solutions;
- Other special cases;



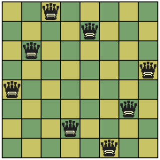
# Task 3: Choosing the algorithm

The important part of choosing the algorithm is counting the time

- An algorithm with $k$-nested loops of and $n$ commands has $O(nk)$ complexity;
- A recursive algorithm with $b$ recursive calls per level, and $L$ levels, it should have $O(bL)$ complexity;
- An algorithm with $p$ nested loops of size $n$ is $O(n^p)$
- An algorithm processing a $n * n$ matrix in O(k) per cell runs in $O(kn^2)$ time.

Use pruning to reduce the complexity of your algorithm!
Also don't forget the number of queries!

# Task 3: Example of Pruning – 8 queen problem

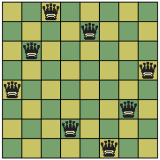


You want to find a position for 8 queens where no queen attack another queen.

```
Solution 1: All options:
Queen1: a1 or a2 or a3 or a4 ... or h5 or h6 or h7
Queen2: Same as Queen 1
Queen3: Same as Queen 2
...
Queen8: Same as Queen 7

Total Solutions: 64 x 64 x 64 x 64 = 64^8 ~ 10^14
```

# Task 3: Example of Pruning – 8 queen problem



You want to find a position for 8 queens where no queen attack another queen.

Solution 2: One queen / column

Queen1: a1 or a2 or a3 ...

Queen2: b1 or b2 or b3 ...

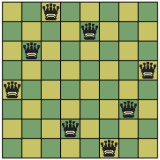
Queen3: c1 or c2 or c3 ...

...

Queen8: h1 or h2 or h3 ...

Total Solutions: 8 x 8 x 8 ... = 8^8 ~ 10^7

# Task 3: Example of Pruning – 8 queen problem



You want to find a position for 8 queens where no queen attack another queen.

```
Solution 3: One queen / row x column
If Q1 is a1, Q2 in b2, Q3 must be c3-7...

A solution is the order of rows: Ex: 1-3-5-2-7-4-8-

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

# Task 4: Coding

Do you understand The Problem and The Algorithm?

**NOW** you can start writing your program.

If you start your program before you understand the solution, you will create many more bugs.

# Task 4: Coding

### Hint 1: "The Library"

Create a file with code examples that you often use.

- Input/output functions;
- Common data structures;
- Difficult algorithms;

### Hint 2: Use paper

- Writing your idea on paper help you visualize;
- Sometimes you can find bugs this way;

# Task 4: Coding

## Hint 3: Programmer Efficiency

Everyone knows about CPU efficiency or Memory efficiency.

But Programmer Efficiency is very important too: Don't get tired/confused!

- Use standard library and macros;
- Your program just need to solve THIS problem;
- Use simple structures and algorithms;
- TDD mindset;

# Task 5,6: Test and Hidden Data

The example data **is not** all the data!

**Example Data**

- Useful to test input/output;
- Read the example data to understand the problem;

**Hidden Data**

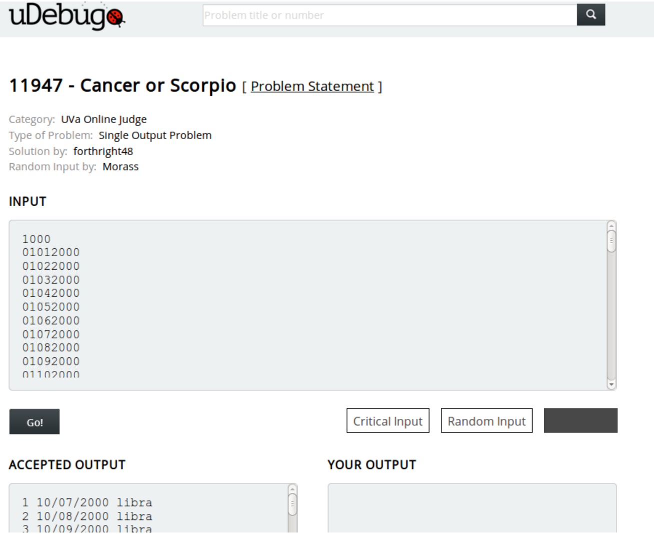
- Used by the UVA judge;
- Contain bigger data sets;
- Includes special cases;

Generate your own set of hidden data before submitting!

# What data to generate?

- Datas with multiple entries (to check for initialization);
- Datas with maximum size (can be trivial cases);
- Random data;
- Border cases (maximum and minimum values in input range);
- Worst cases (depends on the problem);

# The uDebug Website



# To Summarize

Mental framework to solve problems:

1. Read the problem carefully to avoid traps;
2. Think of the algorithm and data structure;
3. Keep the size of the problem in mind;
4. Keep your code simple;
5. Create special test cases;

Now go solve the other problems!

# Thanks for Listening!

Any questions?

# BONUS – Calculating Complexity in Research Experiments

# BONUS – A very simple program

## Ackermann's Function

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0, n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0, n > 0 \end{cases}$$

- $A(0,n) = 1$ step
- $A(1,n) = 2n+2$ steps
- $A(2,n) = n^2$ steps
- $A(3,n) = 2^{n+3} - 3$ !
- $A(4,n) = 2^{2^{\cdots^{2}}} - 3$ !!! (exponential tower of n+3)
- $A(5,n) =$ !!!!!!!!!!!!