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 25, 2020)

Version 2020.1

Week 1 – Part 1: Class Introduction

Outline

Part I: Class Summary

- In this class, you will practice your algorithm skills by writing many programs;
- Key Idea: Solve challenges with programs
  - Read the problem and choose the right algorithm;
  - Choose the implementation and data structure;
  - Run the program and check the correct result;
- In the 1st and 2nd year, we learn the theory of many algorithm. In this lecture, we learn the practice of these algorithms.

Algorithms: Theory x Implementation

It is very different to implement an algorithm in the classroom, and to implement an algorithm to solve a problem:

- Data Structure: You need to implement the function to read the input and store it in a data structure;
- Special Cases: Special cases in the input can cause bugs or make the problem more complex;
- Large Input: If the input is very large, the implementation needs to be efficient, or the program will run forever;
- Debugging: It can be hard to debug if the input is very hard, you need to learn how to make a test input;

Automated Judging

In this lecture we use **Automated Judges (AJ)** to check the correctness of your programming assignments.

- An AJ is a website that receives code from users, automatically compile it, and check its output against an expected output;
- AtCoder, Aizu Online Judge, Topcoder, UVA Online Judge;
- You can submit your code many times;
  - But you don't receive debug information: you have to debug by yourself.
- Grading is automatic and instantaneous;

Every lecture, you have 8 programming assignments, and you have to complete a minimum of 2.

What this class expects of you

Estimated classwork:

- You need basic programming knowledge in C++ or Java;
- Complete 2-4 program assignments per week;
  - (Atcoder difficulty: 300 to 500)
  - Average of 4 hours per week
- First assignments are easy, but later assignments are hard
  - Spend a lot of time debugging and improving code;
- No final exam: only assignments!

Hint: Do your homework early!

Part II: What is a Programming Challenge?

A "Programming Challenge" is a problem where you write a program to find the solution. Think of it as a **programming puzzle**:

Simple Example

You want to hold dancing classes. You receive a list of possible dates and times from all your friends (ex: Fri: 14-16). Find a set of class times which put most of your friends together. Don't forget to make pairs!

- Usually Programming challenges have a "story";
- Usually there is a maximum running time, so you must choose the most efficient algorithm;
- Sometimes there are special input cases;
- In general, you can solve the challenge with a small program (< 200 lines);

Programming Challenges in the world

- **School Competitions**: Programming contests started as a way for schools to compete early in 1980s. ICPC - University
- **Self Improvement**: After 2000, many people use online Automated Judges to improve their programming skills and compete against each other (TopCoder, HackerRank, AtCoder, etc);
- **Company Recruitment**: More recently, several companies (including Google, Facebook, etc) have started using programming challenges to test the technical knowledge of their candidates.

A challenge example: The 3n+1 problem

What can we expect from a programming challenge?

- Description;
- Input and Output format;
- Input and Output examples;

Let's show how we would solve this problem (we will see more details later in the class).

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

Consider the following algorithm:

1. input n
2. print n
3. if $n = 1$ then STOP
4. if n is odd then $n \longleftarrow 3n + 1$
5. else $n \longleftarrow n/2$
6. GOTO 2

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed before and including the 1 is printed. For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between and including both i and j.

Input

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.

Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
1 10
100 200
201 210
900 1000
```

Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

A challenge example: The 3n+1 problem

Solving the problem

This problem requires that we calculate the longest sequence generated from the following algorithm:

1. if $n = 1$ then STOP
2. if n is odd, then $n = 3n + 1$
3. else $n = n/2$
4. GOTO 1

For example, from 1 to 4:

- 1: 1 END
- 2: 2 1 END
- 3: 3 10 5 16 8 4 2 1 END
- 4: 4 2 1 END

Maximum **Cycle length** is 8 (for n = 3)

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

Consider the following algorithm:

```
1.  input n
2.  print n
3.  if n = 1 then STOP
4.      if n is odd then n ⟵ 3n + 1
5.      else n ⟵ n/2
6.  GOTO 2
```

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed before and including the 1 is printed. For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between and including both i and j.

Input

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.

Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
1 10
100 200
201 210
900 1000
```

Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

A challenge example: The 3n+1 problem

A simple solution

```
int main() {
  int min = 1;
  int max = 10;
  int maxcycle = 0;
  for (int i = min; i <= max; i++) {
    int cycle = 1;
    int n = i;
    while (n != 1) {
      if (n % 2 == 0) { n = n / 2; }
      else { n = n*3 + 1; }
      cycle++;
    }
    if (cycle > maxcycle) maxcycle = cycle;
  }
  cout << min << " " << max << " " << maxcycle << "\n";
  return 0;
}
```

A challenge example: The 3n+1 problem

Simple solutions, simple problems

The solution proposed in the last slide has some problems. One problem is that it can be very slow! Consider the following situation:

Let the input be 1 10:

- 1: 1 END
- 2: 2 1 END ...
- 7: 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 END

  ...
- 9: 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 END
- 10: 10 5 16 8 4 2 1 END

This simple program will repeat a lot of work. If the input is large, this extra work will be very slow! How can we improve this?

A challenge example: The 3n+1 problem

Memoization

To solve this "repeated work" issue, we can use a technique called **Memoization**.

The basic idea is very simple: Every time you finish a calculation, store the result of this calculation in the memory. If you ever have to do that calculation again, load it from memory.

This technique can be very useful to reduce the amount of repeated work.

In this course, we will review many techniques like this to make programs more efficient, and you will implement these techniques in the programming assignments.

Topics in this class

We will study the following topics in this semester:

1. Ad Hoc Problems
2. Data Structures
3. Search Problems
4. Dynamic Programming
5. Graphs Problems (Graph Structure)
6. Graph Problems (Graph Search and Flow)
7. String Manipulation
8. Math Problems
9. Geometry Problems
10. Final Remix

Class Format

Weekly Lecture Contents

- PDFs with information about algorithms
- Videos with explanation about the classes
- Programming assignments on "onlinejudge.org"

Evaluation

- No final examination;
- Weekly programming assignments;

More details about these two points in the next section.

About the Lecturer

- Name: Claus Aranha;
- Country: Brazil;
- Research Topics:
  - Evolutionary Algorithms;
  - Artificial Life;
- Hobbies:
  - Game Programming;
  - Astronomy;
- webpage:

  `http://conclave.cs.tsukuba.ac.jp`

Extra: Join the Tsukuba ICPC Team!

What is ICPC?

If you like these contests, and want an extra challenge, please consider joining the Tsukuba ICPC team!

ICPC (International Collegiate Programming Contest) is the largest and most traditional programming competition between universities.

More than 50.000 students from all over the world participate in this competition every year.

Contest Website: `https://icpc.baylor.edu/`

Extra: Join the Tsukuba ICPC Team!

Program and see the world!

- Requirements: Team of 3 students, any course;
- Schedule:
  - National Preliminary Competition in July
  - Japanese Regional Competition in October
  - Asian Semi-final in December
  - World Final April next year

  (Dates may change this year because of nCov-19)
- Contact me if you're interested!

Week 1 – Part 2: How this lecture is organized

Outline

What you will do every week

- (manaba) Get the week PDF and study the lecture;
- (manaba) Watch the lecture video;
- (OnlineJudge) Read the programming assignments;
- Solve the programming assignments;
- (OnlineJudge) Test your programs until they are correct;
- (manaba) Submit your correct programs;

Class Dates and Deadlines

Class Dates

- 4/28, 5/12, 5/16, 5/19, 5/26, 6/2, 6/6, 6/9, 6/16, 6/23;
- No final exam;
- Classes will asynchronous (download the video);

Deadlines

- The deadline for assignments: **10 days after each week**
- The deadline for late assignments: 7/7
- Final Grades will be published by 7/13

Dates subject to changes.

Lecture Notes and manaba

manaba system

- Use it to access News, Surveys, and Forums;
- Use it to submit assignments;
- URL: `https://manaba.tsukuba.ac.jp/ct/course_1322213`
- Self-registration Code: 6910967 (non-credit students)

github page

- Has the same material as Manaba (Lecture notes, videos);
- Can be accessed by anyone;
- CANNOT access news, grades or assigments;
- URL: `https://caranha.github.io/Programming-Challenges/`

Websites for programming Assignment

- **Online Judge**: `https://onlinejudge.org`
  This website hosts many programming challenges and an Automated Judge. In this class, you will submit your programming assignment to this site to check its correctness.

- **uMonitor**: `https://caranha.github.io/Programming-Challenges/uMonitor/monitor.html`
  This website is a javascript that collects information from Online Judge. You can use this site to check the assignments for each week, and which assigments you have already submitted.

Course Language

Natural Language

- The materials in this course will be in English;
- Announcements and Video Lectures will be in Japanese;
- You can use any language you want;
- If you are interested in helping translate the materials, contact me!

Programming Language

- The online judge accepts: C, C++, Java, python, PASCAL;
- I recommend that you use C++, but you can use other languages;
- Code examples in this class will usually be C++;

Reference Books

Textbook:

- **textbook:** Steven Halim, Felix Halim,"Competitive Programming", 3rd edition. `https://cpbook.net/`

Other references:

- Steven S. Skiena, Miguel A. Revilla,"Programming Challenges", Springer, 2003
- 秋葉拓哉、 岩田陽一、 北川宜稔,『プログラミングコンテストチャレンジブック』
- 渡部有隆、『オンラインチャレンジではじめるC/C++プログラミング入門、Online Programming Challenge!』(ISBN978-4-8399-5110-8)
- 渡部有隆、『プログラミングコンテスト攻略のためのアルゴリズムとデータ構造』（ISBN978-4-8399-5295-2）

Submitting Assignments

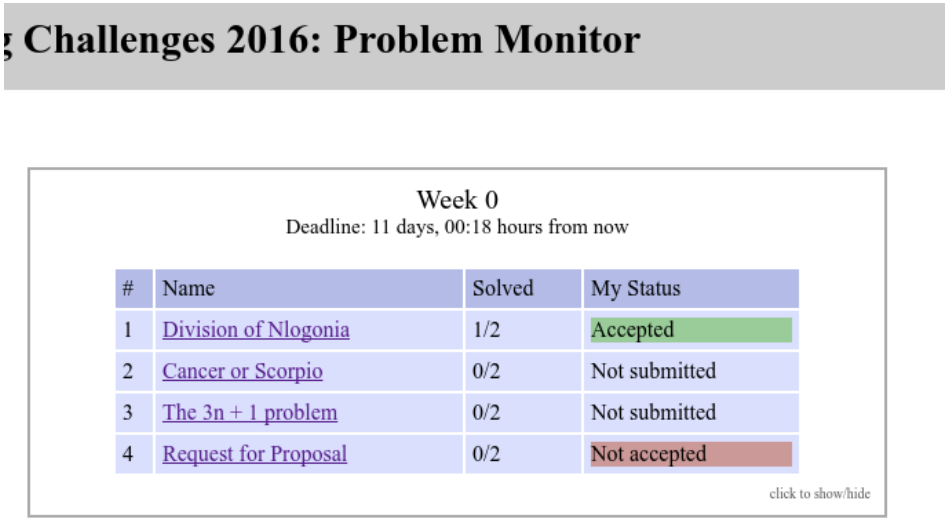
Let's learn how to submit assignments in this lecture. These are the main steps:

- Check the assigment on the monitor page;
- Read the assignments on onlinejudge.org and solve them;
- Submit the assignment to onlinejudge.org;
- Check the results: onlinejudge.org and monitor page;
- Submit your code to Manaba;

Submitting Assignments 1

Check the monitor page

https://caranha.github.io/Programming-Challenges/uMonitor/monitor.html



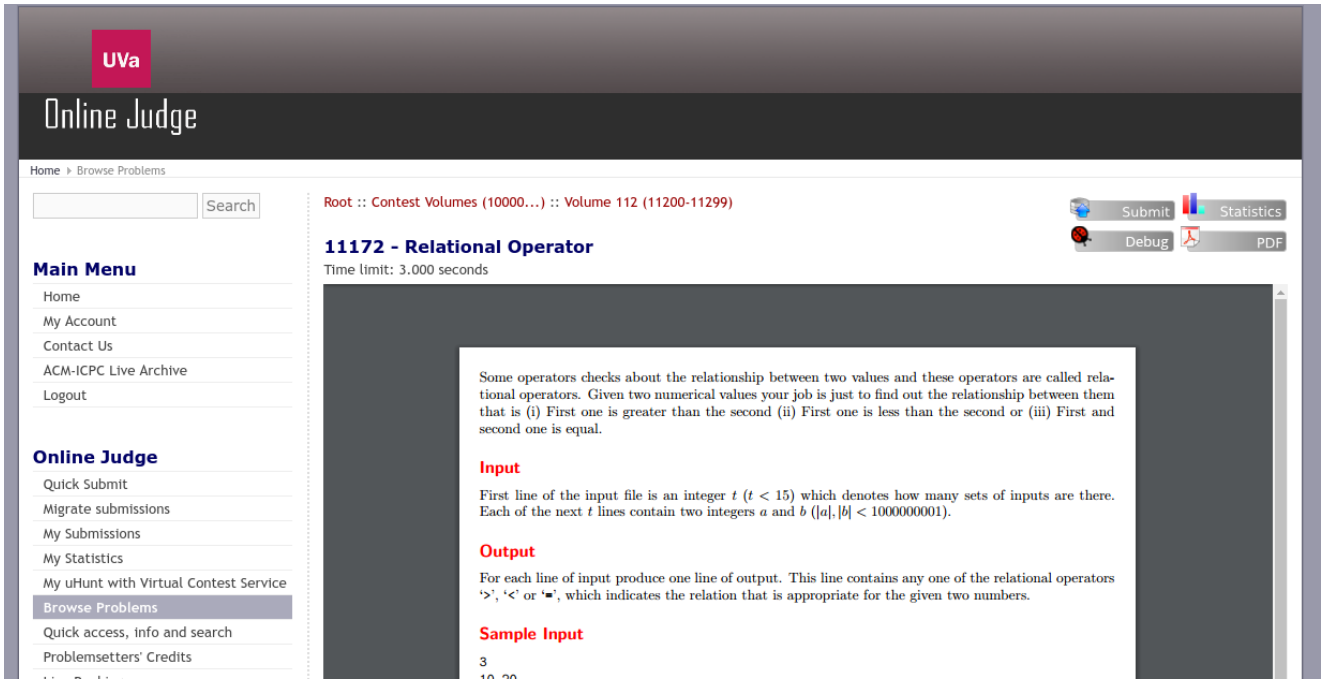
Check the assignments for each week at the monitor webpage:

- The programming assignments for each week;
- The deadlines for each week;
- Total submission from the students;
- Status of your submissions;

Submitting Assignments 2

Submitting an assignment to Online Judge

Click on a problem name to go to the "Online Judge" page:

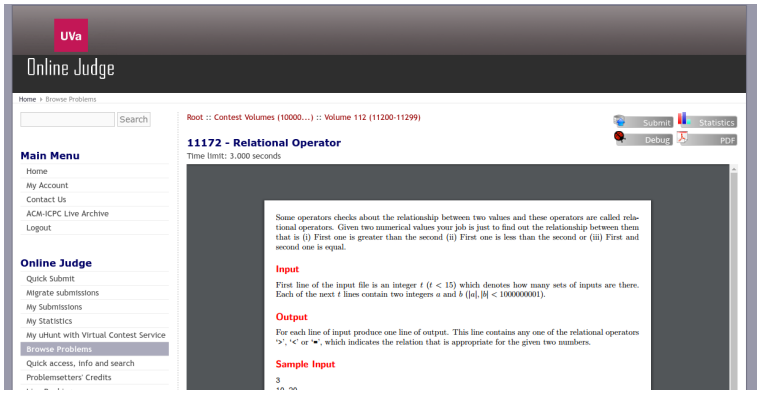


In this page you can get more information about the problem, and submit your code. Click on the "submit" button to submit your code.

Submitting Assignments 2

Submitting an assignment to Online Judge

What information can you get from the Online Judge?



- Problem Description – Read carefully;
- Sample Input and Output – Use this to debug your program;
- Time limit – Make an efficient program!

Submitting Assignments 2

What is the Online Judge?

What is the Online Judge?

- Created by the University of Valladolid (Spain);
- Now an independent website;
- Over 20.000 Programming challenges;

Important 1: Make an account on Online Judge!

You need an account in Online Judge for this course. When you make your account, **submit your account name on the survey on manaba!**

Important 2: What to do if the judge is offline

Sometimes the Online Judge is offline for maintenance. Please be patient. If the Online Judge is offline for a long period, there will be a deadline extension. This will be announced on manaba.

Submitting Assignments 3

Reading the Judge Results

My Submissions

#	Problem	Verdict	Language	Run Time	Submission Date
17182419	1124 Celebrity jeopardy	Accepted	C++	0.000	2016-04-11 06:42:30
17181459	10141 Request for Proposal	Compilation error	C++11	0.000	2016-04-11 01:36:46
17181444	11498 Division of Nlogonia	Accepted	C++11	0.000	2016-04-11 01:30:43
17071417	102 Ecological Bin Packing	Compilation error	C++11	0.000	2016-03-23 09:21:55
17070667	161 Traffic Lights	Accepted	C++	0.000	2016-03-23 07:24:56
16607686	489 Hangman Judge	Accepted	C++	0.349	2015-12-20 03:52:45
16607670	489 Hangman Judge	Wrong answer	C++	0.335	2015-12-20 03:47:01
16607649	489 Hangman Judge	Runtime error	C++	0.000	2015-12-20 03:40:51

- After you submit your program, the online judge will evaluate it in 2-5 minutes (or more if it is busy).
- You can see the result by clicking on the "my submissions" link;
- These are the possible results:
  - Accepted: Congratulations!
  - Presentation Error: Small mistake. Check the output!
  - Wrong Answer: Your program is incorrect. Time to debug.
  - Time Limit Exceeded: Your program is too slow. Optimize!
  - Memory limit exceeded: Your program uses too much memory.
  - Runtime Error: Your program crashed! (segmentation fault!)

Attention: Java and Python users

Java Users

- All code must be in the one file;
- The static main method must be in Main class.
- Do not use public classes. Even Main must be non public.
- Use Buffered I/O for faster input/output.

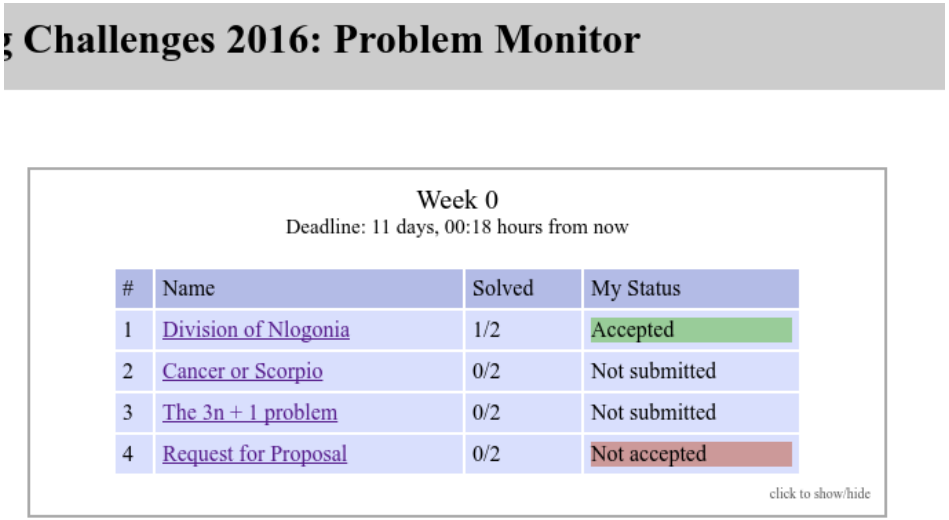
Python Users

Python was added recently to Online Judge. For some of the problems, python might be too slow, and maybe you cannot solve in time with the same algorithm.

If you receive too many "Time Limit Exceeded" results, try solving using C++;

Submitting Assignments 3

Reading the Monitor Results



; Challenges 2016: Problem Monitor

Week 0
Deadline: 11 days, 00:18 hours from now

#	Name	Solved	My Status
1	Division of Nlogonia	1/2	Accepted
2	Cancer or Scorpio	0/2	Not submitted
3	The 3n + 1 problem	0/2	Not submitted
4	Request for Proposal	0/2	Not accepted

click to show/hide

- You can check the overall status of your submissions on the problem monitor;
- This information also include the deadlines for each week, and if your submissions are late or not;
- This information will only be available after your complete the **manaba survey**;

Submitting Assignments 4

Submitting the code to Manaba

- You must submit your assignment to manaba after you check your code is correct;
- Put all your program files in a single zip file;
- Avoid using non-ascii characters for the filenames;
- Attention: Do not forget to submit to manaba!

s2020999_tsukubataro_w01.zip

- program1.cpp
- program2.cpp
- program3.java
- program5.cpp

Grading

Your grades are calculated based on the number of programs you submit

Grade Algorithm: **Base Grade** -Late Penalty + Rank Bonus

Grading

Base Grade

Every week, there are 8 programming assignments. You don't need to solve every one (but please try!). Your base grade is based on how many problems you solved.

- **Base Grade**:
  - "C": Minimum 2 problems per week;
  - "B": Minimum 3 problems per week;
  - "A": Minimum 4 problems per week;

Warning 1: Only "Accepted" problems count. Not "Wrong Answer"
Warning 2: The grade is based on minimum problems. Not average.

Grading

Late Penalty

You can still submit your assignment after the weekly deadline. However, if you submit too many assignments late, you will receive a penalty.

The penalty reduces your grade by one level. A to B, B to C. The penalty **will not** reduce your grade below C.
You will receive a penalty if more than 25% of your assignments were submitted late.

Grading

Rank Bonus

For each base grade (C, B, A), 10% of the students with the most **total problems** will receive a bonus.

The bonus will raise your base grade. C to B, B to A, A to A+.

In very special cases, the professor may give a bonus for a student that contributes greatly to the class.

Grading

Plagiarism

The assignments are individual. You must write your programs by yourself.

You can do this

- Ask for ideas to your friends;
- Ask for ideas in the MANABA forum;
- Ask for help with a bug;

You can NOT do this

- Copy a solution from the internet;
- Copy a solution from your friends;
- Give your code to a friend;

Students who do plagiarism will fail the course, and possibly suffer penalties from the university.

Teacher Communication

- The teacher will use the manaba "News" systems to send important news. Make sure to check if you get a message from manaba;
- New class materials will be posted on the github page, and on manaba before the lecture time;
- This semester we do not have "real time" lectures, but the professor will reserve the lecture time (Tue 3,4) for "office hours";
- Use the manaba "Forums" systems to ask questions about the assignments;
- Send an e-mail to the professor with any questions about the course that are not about assignments;
- You can send e-mail and forum message in English or Japanese;

Programming Environment

In this lecture, you will need to write and compile programs to complete the assignments and get your grade;

If you do not have a programming environment (do not have a computer at home), and you are a student from the College of Information Science (COINS), the college will help you. Please contact the teacher by e-mail;

If you are a student from another college, please contact the professor. I cannot promise that I can help you, but I will do my best.

Important Reminder

For this class, you need an account on the Online Judge website.

Please make an account on that website, and submit your username on the manaba survey.

If you don't complete the survey with your username, you will not be graded!

Week 1 – Part 3: AdHoc Problems

What are Ad-hoc problems?

Ad-hoc is an expression used to mean something used for a single purpose. In programming challenges, an ad-hoc problem is a simple problem that you can solve by simply following the instructions.

Although ad-hoc problems are simple, it is good to practice them seriously, as we can learn skills that are useful for more complex problems too:

- Extracting important information from the problem text;
- Understanding the input and output;
- Generating debug and worst case data;
- Analysing problems;

In this lecture, we will see some hints that will be useful for the entire course.

Revisiting the 3n+1 problem

Let's revisit the 3n+1 problem, that we introduced in the beginning of this class.

This is a great problem because it is very simple, but you still need to be careful when implementing the solution.

Revisiting the 3n+1 problem

Problem outline

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

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

Size: 16
```

A simple solution for the 3n+1 problem

```
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: n = 3 * n + 1
        else: n = n / 2
        count += 1
      if count > max: max = count
    print (line, max)
  except EOFError: break
```

A Simple Solution has a Simple problem

- If you submit the simple solution in the previous slide, you will receive the result Wrong Answer from the judge.
- You look at the example input and output, and your program gets the correct solution for all the sample input!
- You even try a few more values of i and j to make sure.
- What is wrong with the simple solution? (Answer in the next slide)

Trick Hidden Inputs

- The first solution solves all the sample input correctly, but there are some inputs that can cause problems.
- For example, think of this input: $i = 20, j = 10$
- The output will be nothing!
- Here is the code that causes this problem:

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

- But in the input there are no examples with "$i > j$" is this allowed?

Trick Hidden Inputs

Reading the input carefully

The problem page has a description that include an **input** section:

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.

In the input section, there is no rule that forces $j > i$!

If it is not forbidden by the rules, then maybe it will happen!

Trick Hidden Inputs

General Rules

- First rule of Programming Challenges:

  If it is not written, assume the worst case

- Some common tricky inputs to be careful:
  - A number is negative; a number is zero; a number is maximum;
  - The input is out of order (or in order!);
  - The input is repeated;
  - A graph is unconnected; a graph is fully connected;
  - Lines are parallel; points are in the same place;
  - An area is 0; an angle is 0;
  - The input is very long;
  - The input is very short;

- If it is not against the rules, it may (will?) happen;
- This first rule is also important for the real world!

Problem 3n+1

A fixed solution for the input

```
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
```

This solution has a problem too!

- This time we are sure that the solution is correct for all inputs;
- But you still do not get "accepted" in the online judge...
- Now the judge says "Time limited exceeded".
- What happened? (Think a little bit on this one)

Solution 2: Be careful of computing costs!

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 minimum and maximum?

```
1 10000 262
```

- A sequence with 262 steps does not seem very long.
- But remember we calculate all sequences between 1, 10000!
- Estimate the cost: $10000 \times 262 \approx 2,000,000$ steps!

- This is one query. The input can have repeated queries!

Avoiding repeated work

- Think about the cycle calculation (let's call it A(n));

```
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
```

- Can we avoid all this recalculation?

Good Technique: Memoization

Store the result of a function that we know will be calculated again in the future. This is useful for many problems!

Avoiding Repeated Work

Memoization Idea

To implement Memoization, we use a table or dictionary to store partial results.

```
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]
```

In a future class, we will focus on Dynamic Programming, which generalizes this technique.

A Programming Challenge Workflow

When solving a programming challenge, you want to follow these common steps:

- 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

Reading and understanding the problem is very important! Try to understand completely the problem before starting to program.

General Tips

- Find the rules of the problem, separate from the story;
- Sometimes it is easy to read the input/output part first;
- Problems with a lot of story are usually not very hard.;

Reading English is Hard!

- Don't worry!
- Focus on the keywords;
- Understanding a little English is important for CS!
- Get help from professor Google Translate;

Example: Problem 11559 – Event Planning

Story:

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.

How do you tell the difference between Story and Rules? Look for the keywords!

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

Hints for Reading Problems

- First, look at the sample input and output;
- Write the key ideas 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!

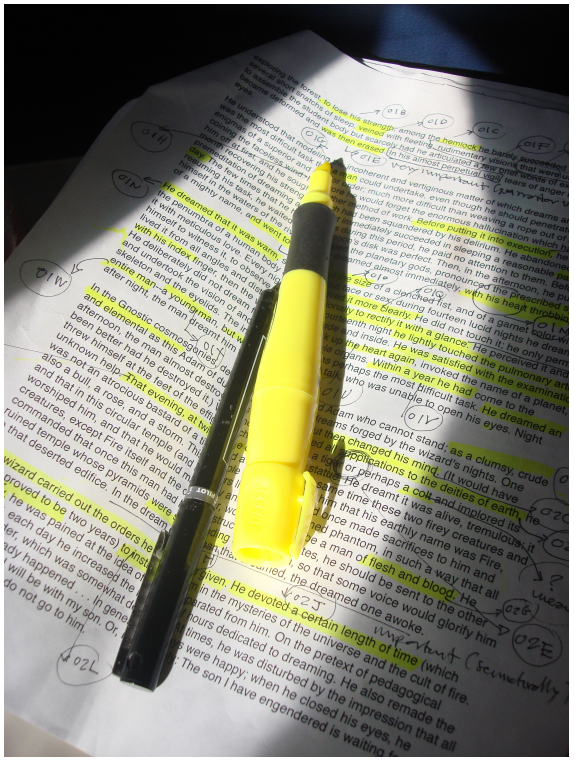


Image by Guido “random” Alvarez, released as CC-BY-2.0

Task 2: Reading the Input/Output

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

- 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?

Task 2: Estimate time limit from input size

The input size shows how big the problem gets.

- Small Problem: You can use brute force algorithms;
- Big Problem: You need to use more complex algorithms; Maybe prune the input;

Estimate the "Time Limit Exceeded" limit

- expect around 10,000,000 operations per second on the online judge;
- most problems have 1-3 seconds of time limit;
- python can be a bit slower!

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)

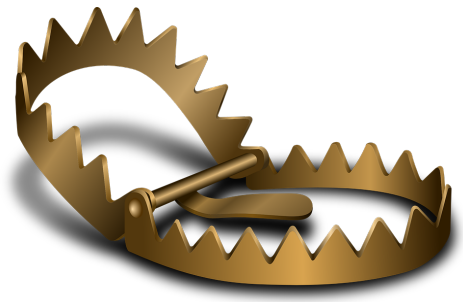
Task 2: Tricks in the input/output

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.

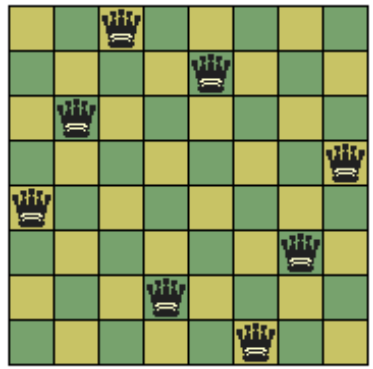
One way to reduce the cost of a program is to use pruning

Task 3: Example of Pruning – 8 queen problem

Pruning is when you quickly remove bad cases to reduce the computational cost of an algorithm.

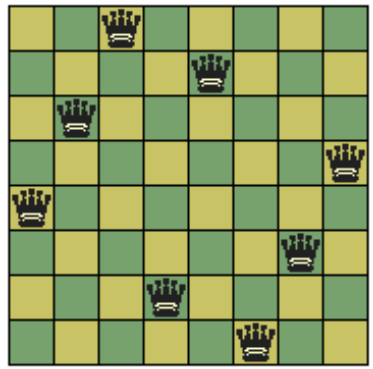
The "8-queens" problem is a good problem to understand pruning. We want to find all possible positions of 8 queens in a chessboard, where they don't attack each other.

The brute force algorithm is: Try all possible positions, and check if they are valid. But how we list all positions make a big difference.



Task 3: Example of Pruning – 8 queen problem

Approach 1 – all squares



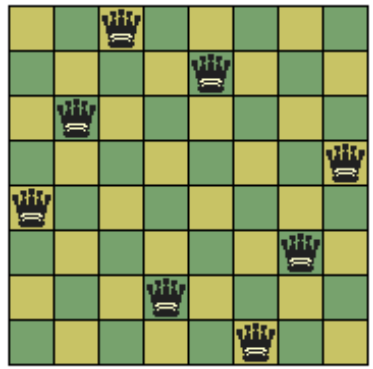
Approach 1: For each queen, test all possible squares;

```
Queen1: a1 or a2 or a3 or a4 ... or h5 or h6 or h7 or
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

Approach 2 – columns



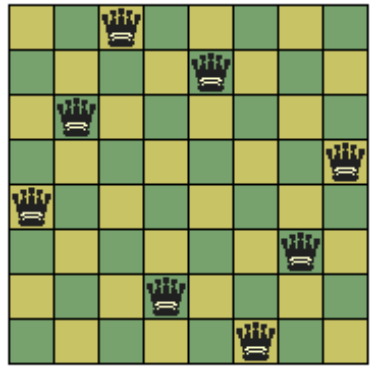
Approach 2: Choose one column for each queen, test all rows.

```
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

Approach 3 – permutation



Choose one column for each queen, test a row not used.

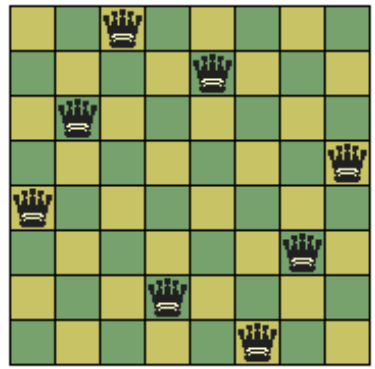
```
If Q1 is a1, Q2 in b2, Q3 must be c3-7...

A solution is the order of rows: Ex: 1-3-5-2-7-4-8-6

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

Task 3: Example of Pruning – 8 queen problem

Pruning comparison



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

Same algorithm (brute force), but pruning changes the efficiency!

Task 4: Coding

After you understand what the problem requires you to do, and after you decide which algorithm to use, then you can begin programming.

Avoid beging programming before you have a good idea of what you are going to do. Early programming might lead to bugs, and you may decide that you want to change your algorithm or data structure;

Hints when coding

Hint 1: Coding Library

As you solve many programs, save functions and patterns that you use many times:

- Code for different types of Input;
- Common data structures;
- Difficult algorithms;

Hint 2: Use paper

When you write your idea on paper (diagrams, code), it helps you clarify parts that are not well defined in your mind.

Sometimes this helps you find problems in your idea early.

Hints when Coding

Hint 3: "Programmer Efficiency"

We often think about **CPU efficiency** (program is fast), or **memory efficiency** (program uses little memory).

But another very important efficiency is **Programmer Efficiency**: If the program is too complex, the programmer will get tired or confused!

- Complex programs take longer to complete;
- Complex programs are harder to debug;
- You can't reuse code from complex programs easily;

How to improve programmer efficiency:

- Learn and use the standard library and macros;
- Create your own library of programming challenge patterns;
- Focus on the minimal features for this problem;
- Use clear names for variables and functions;

Sample data and Hidden Data

Remember that the sample data in the program is not all data! The Online Judge will use a set of **hidden data**.

Generate your own set of data to test your program before submitting.

Sample Data

- Useful to test the input/output functions;
- You can read the sample data to understand the problem;
- Does NOT include tricky cases;

Hidden Data

- Data of the online judge;
- Much bigger cases, uses maximum and minimum values;
- Data includes tricky cases and worst cases;

Generating testing data cases

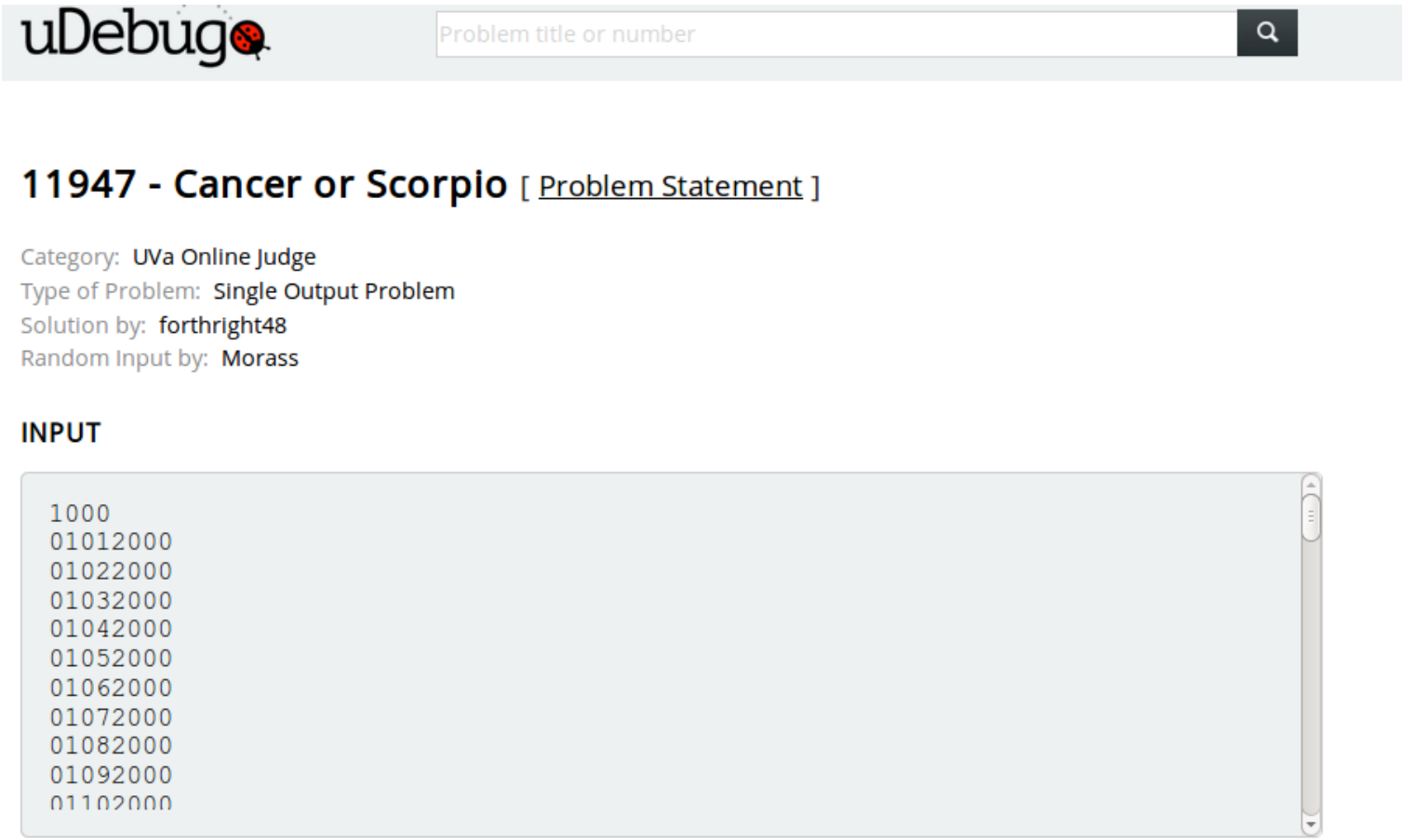
"Be mean. Generate data that will show bugs in your program, not data that you expect to work."

- Repeat one case many times in the same file (check for initialization);
- Random data with maximum size/values (test for array limits and general performance);
- Border cases (maximum and minimum values);
- Worst cases (what data would make the algorithm run slowest? If you don't know the worst case, maybe you don't understand the algorithm yet!);

In particular for random and large test data sets, I recommend that you create a simple script that generates random data for you. Random data is great for finding "crash" bugs.

Test data website: uDebug

The udebug website `https://www.udebug.com/` has test cases for many problems in online-judge. However, note that it does not necessarily has all test cases!



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!

Send questions to the manaba forums!