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 24, 2020)

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

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 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
```

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

Good Technique: Memoization

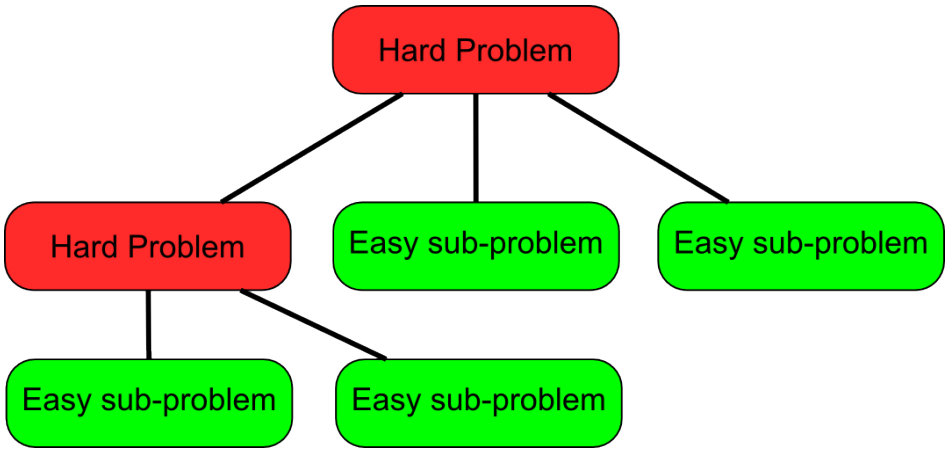
If we know that we will use a result again in the future, we should store this result in the memory.

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!

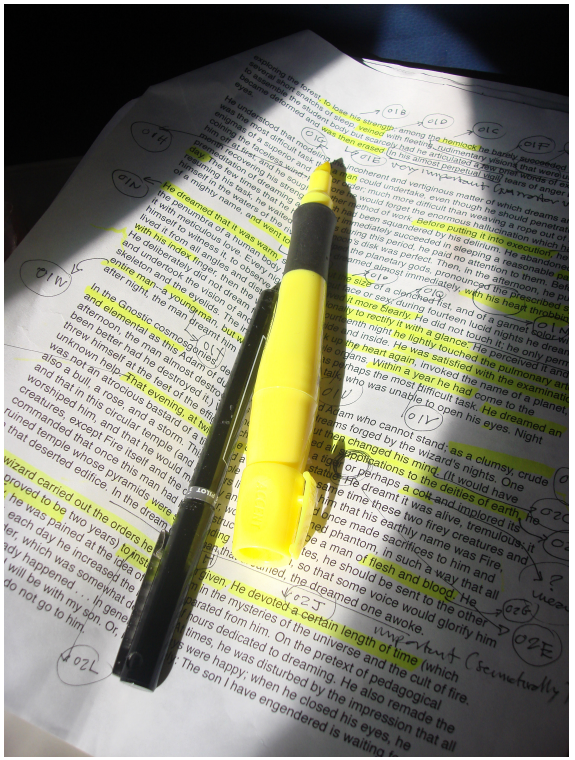


Image by Guido “random” Alvarez, released as CC-BY-2.0

Task 2: Reading the Input/Output

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

- What is the 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)

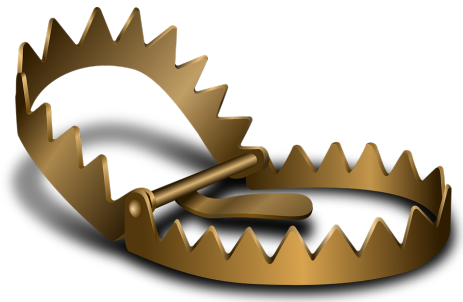
Task 2: Input/Output – Traps!

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;



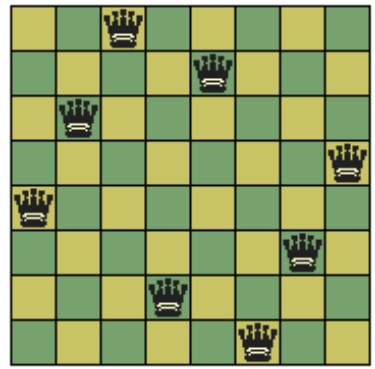
Task 3: Choosing the algorithm

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

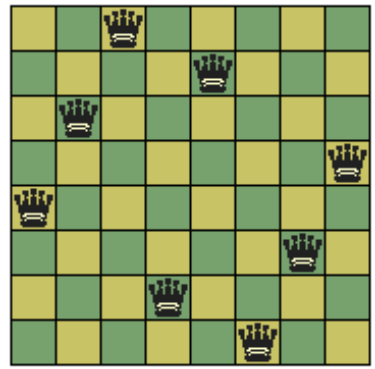


You want to find a position for 8 queens where no queen attack another queen.

```
Solution 1: All options:
Queen1: a1 or a2 or a3 or a4 ... or h5 or h6 or h7 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

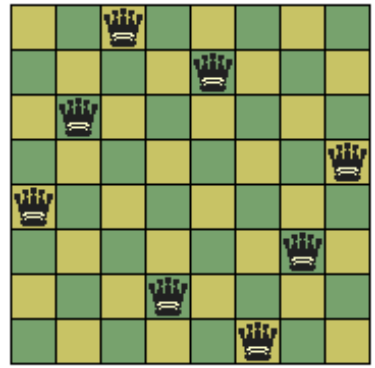


You want to find a position for 8 queens where no queen attack another queen.

```
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: 1-3-5-2-7-4-8-6

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

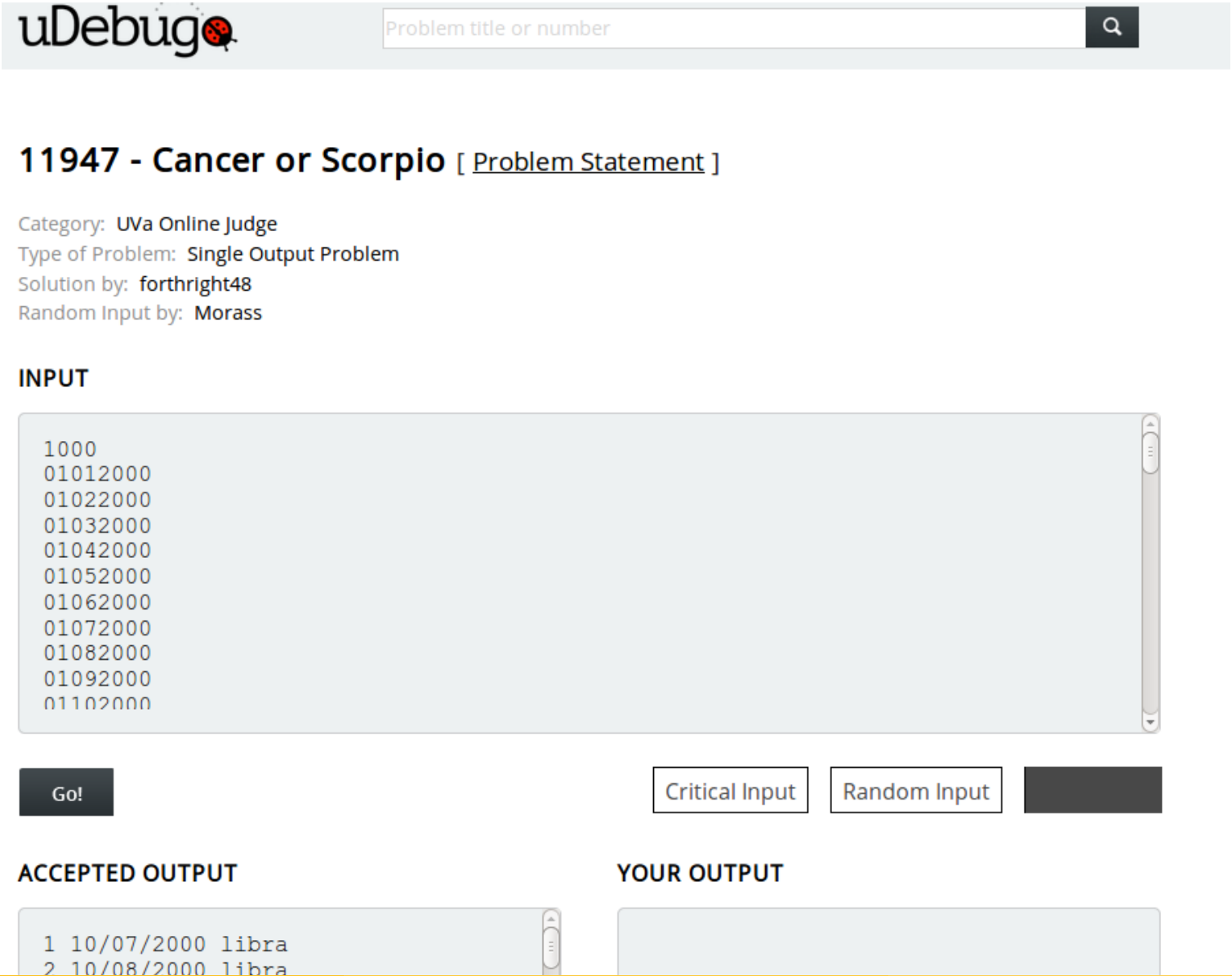
- Used by the UVA judge;
- Contain bigger data sets;
- Includes special cases;

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)
- $A(5,n) =$!!!!!!!!!!!