GB21802 - Programming Challenges

Week 0B - Solving Problems

Claus Aranha
caranha@cs.tsukuba.ac.jp

Department of Computer Science

2016/4/15
(last updated: April 18, 2016)

Survey Results (Sunday)

About 15 respondents:

- Students: Regular: 40%, Tech School: 40%, Intl: 20%
- Contest Experience: 10%
- 50% programs for fun. Only 1 to save the world.
- C/C++ are favorite languages, Only 1 likes Python :-(. One wants to use Ruby, One wants to use LUA!
- Expectations: Programming Contest (1 person), Improving programming skill (5 people), Worried about English: 2 People;

UVA Username

Currently, 24 out of 30 people sent me their username.
Don't forget to submit your UVA username!

Submission Results (Sunday)

- Division of Nlogonia: 7 Tried, 6 Succeeded
- Cancer or Scorpio: 4 tried, 4 succeeded
- 3n+1 Problem: 5 tried, 4 suceeded
- Request for Proposal: 2 tried, 2 succeeded

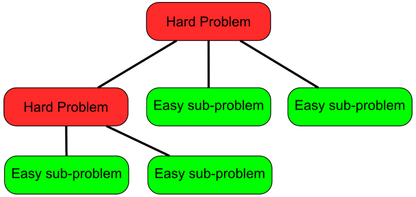
Time to make questions about the problems!

Today's Class: Solving Problems

Hints and ideas on how to solve programming challenges

- Reading Problems
- Considerations when programming
- Input/Output
- Debugging
- Types of Problems

A Programming Challenge Workflow



First trick of solving a hard problem: break it into simpler ones:

- Task 1: Understand the problem description;
- Task 2: Understand the input/output;
- Task 3: Choose the Algorithm;
- Task 4: Write the Code;
- Task 5: Test the program on example data until it works;
- Task 6: Test the program on hidden data;

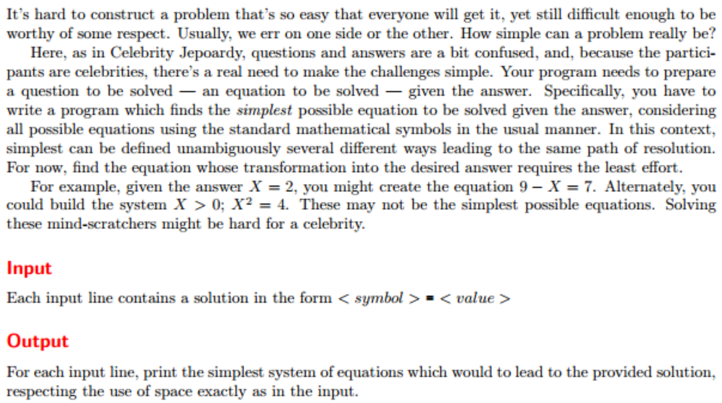
Task 1: Understanding the Problem Description

Program Challenge descriptions can be hard to understand sometimes:

- Easy problems usually have extra “flavor” text;
- Problems with little text are usually harder;

Of course, there are always exceptions!

Extreme example: UVA 1124, Celebrity Jeopardy



It's hard to construct a problem that's so easy that everyone will get it, yet still difficult enough to be worthy of some respect. Usually, we err on one side or the other. How simple can a problem really be?

Here, as in Celebrity Jeopardy, questions and answers are a bit confused, and, because the participants are celebrities, there's a real need to make the challenges simple. Your program needs to prepare a question to be solved — an equation to be solved — given the answer. Specifically, you have to write a program which finds the *simplest* possible equation to be solved given the answer, considering all possible equations using the standard mathematical symbols in the usual manner. In this context, simplest can be defined unambiguously several different ways leading to the same path of resolution. For now, find the equation whose transformation into the desired answer requires the least effort.

For example, given the answer $X = 2$, you might create the equation $9 - X = 7$. Alternately, you could build the system $X > 0$; $X^2 = 4$. These may not be the simplest possible equations. Solving these mind-scratchers might be hard for a celebrity.

Input

Each input line contains a solution in the form $<$ *symbol* $>$ = $<$ *value* $>$

Output

For each input line, print the simplest system of equations which would to lead to the provided solution, respecting the use of space exactly as in the input.

Real Problem: Copy the input into the output.

Hints for reading text

- Read the Input/Output sections first, to know the important keywords of a problem;
- Read the problem one time, and mark keywords;
- Read the problem a second time, and cut phrases that are not important;
- For harder problem descriptions, copy the important information to another file.

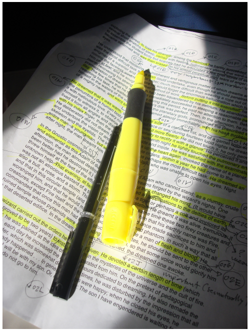


Image by Guido “random” Alvarez, released as CC-BY-2.0

Understanding the Input

Checklist of things we want to learn from the input:

- Problem Size
- Input Format

Input Size

Input Size might be the most important information in the text. It will strongly influence your choice of algorithms.

Keep in Mind!

- Most problems have time limits around 1 – 3 seconds
- Around 10.000.000 calculations per second
- Don’t forget multiplicative effects!

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!

Input Types (1)

N lines

The input tells the number of lines to be read in advance.
Example: 11947 – Cancer or Scorpio

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

int main()
{
    int n;
    string input;

    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> input;
        // Do your work!
    }
}
```

Input Types (2)

Conditional Ending

The input starts with a condition (number of elements, etc), and a special case indicates the end of the input.
Example: 10141 – Request for Proposal

```
int main()
{
   cin >> n >> p;
   cin.ignore(1000, '\n');
   // throws away the \n for a future getline
   while (n!=0 || p!=0)
   {
       // do your work!
       cin >> n >> p;
       cin.ignore(1000, '\n');
   }
}
```

Input Types (3)

End of File

The input goes on until the end of the file. No explicit terminating value.
Example: 100 – The N+1 Problem

```
#include<string>
int main()
{
    string n;
    while (getline(cin,n))
    {
        // Solving your problem
    }
}
```

Reading the Output – Correctness

The UVA judge will evaluate your code based on a simple *diff*. Be very careful to write your output exactly like stated.

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

Easy to Miss errors in the output

- Are the words uppercase, or lower case? (Case or case)
- Are there any mistakes in the words? (case or caes)
- Singular or plural? (1 hours or 1 hour)
- What is the precision of floating numbers? (3.051 or 3.05)
- Do you round up or round down? (3.62 $\rightarrow$ 3 or 4)

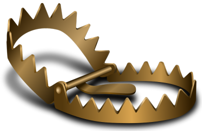
Other things to pay attention in the output

- Do you need to output the entire solution or just the solution cost?
- Are there multiple solutions? Which one must you output?

Reading the problem: Identify Traps!

Be very careful to limits not stated in the problem:

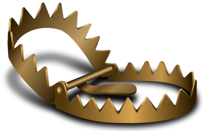
- 3n+1: The input gives i and j, but there is no guarantee that $i > j$.
- Cancer or Scorpio: You have to take leap years into account.



Reading the problem: Identify Traps!

Other common traps:

- Negative numbers, No maximum limit;
- Repeated Numbers;
- (in graphs) circular edges, duplicate edges;
- (in geometry) duplicate points;
- (in paths) two paths with same cost, no solutions;



Task 3: Choosing the algorithm

3.1 What type of problem it is?

- Full Search: Tests all possible solutions;
- Greedy: Break the solution, and try each best subsolution;
- Simulation: Execute a series of steps, record the results;
- Dynamic Programming: Table of partial solutions;
- Counting: Calculate the number of possible solutions;
- Ad-hoc: No fixed category;
- And many others;

We will see examples of each category during the course.

Task 3: Choosing the algorithm

Basic considerations for recursive and iterative algorithms:

- An algorithm with k-nested loops of about n iterations each has O(nk) complexity;
- A recursive algorithm with b recursive calls per level, and L levels, it should have O(bL) complexity;
- This complexity can be reduced with pruning;
- An algorithm processing a $n * n$ matrix in O(k) per cell runs in $O(kn^2)$ time.

Task 3: Choosing the algorithm

We will talk about data structures next class, but see this example:

3.2: What data structure should we use?

Number of Solutions for the 8-queen problem:

- 8x8 matrix, with 1 for each queen: 64*63*62... = 1.78e+14
- 1 queen per column: 8*8*8*... = 1.67+e8
- 1 queen per column/row: 8*7*6... = 40320

You can remove even more solutions by pruning simetries.

Task 4: Coding

Programmer Efficiency: Different from time efficiency, programmer efficiency measures how much a code taxes your mind.

- Understandable program! (not a lot of boiler plate)
- Simpler structures to avoid hard bugs (avoid memory allocation)
- Using “Good Enough” algorithms (avoid overcomplexity)
- Use standard libraries and Macros

Hint: Library File

Create an extra file with macros, functions and structures you use often. Add to this file as you learn new techniques.

Task 4: Coding

Hint 1: Use paper

- Think first
- Code second

Hint 2: Type fast

Practice typing in one of many online typing challenges.

- `www.typingtest.com`
- `https://typing.io` – for programmers!

Test and Hidden Data

The problem description in UVA shows the Example Data. But the actual judge also uses Hidden Data.

Example Data

- Useful to test input/output;
- Read the example data to understand the problem;
- Minimal condition to say your program work;

Hidden Data

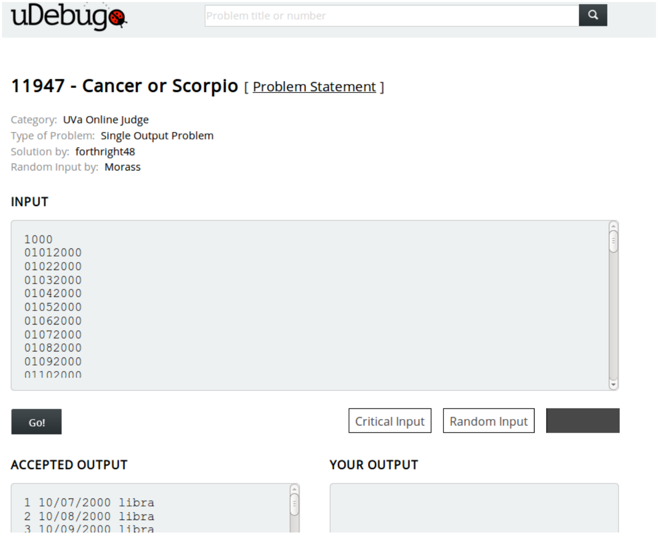
- Actual judging will happen with different data;
- Usually harder and longer data sets;
- Including bully and worst cases;

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



uDebug

Problem title or number

11947 - Cancer or Scorpio [Problem Statement]

Category: UVa Online Judge
Type of Problem: Single Output Problem
Solution by: forthright48
Random Input by: Morass

INPUT

```
1000
01012000
01022000
01032000
01042000
01052000
01062000
01072000
01082000
01092000
01102000
```

ACCEPTED OUTPUT

```
1 10/07/2000 libra
2 10/08/2000 libra
3 10/09/2000 libra
```

YOUR OUTPUT

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 I – The most expensive program ever?

Ackermann's Function

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

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

BONUS II – More Info on the ICPC Programming Challenge

- Application deadline: 06/10 (Fri) – 3-people team
- Internet based contest: 06/24 (Fri) – At Tsukuba
- Webpage: `http://icpc.iisf.or.jp/2015-tsukuba/?lang=ja`

Ask me for details!