Kattis

Introduction

Bjarki Ágúst Guðmundsson
Tómas Ken Magnússon

Árangursrík forritun og lausn verkefna

School of Computer Science
Reykjavík University

Welcome

- T-414-AFLV, Árangursrík forritun og lausn verkefna

- Bjarki Ágúst Guðmundsson, bjarkig12@ru.is
- Tómas Ken Magnússon, tomasm12@ru.is

Goal

- Given a problem, we want to
  - solve it efficiently
  - by using algorithms and data structures,
  - convert our solution into a program,
  - do it as quickly as possible (under pressure)
  - and do it correctly (without bugs)

- This course will exercise this process

How?

- Study common types of problems
- Show common applications of algorithms and data structures you already know from
  - Reiknirit (the algorithms course)
  - Gagnaskipan (the data structures course)
- Introduce other common algorithms and data structures
- Go over some commonly used theory
- Practice problem solving
- Practice programming
- More practice
- More practice

Course book

- Competitive Programming by Steven Halim
- First edition can be downloaded from the book homepage:
- **https://sites.google.com/site/stevenhalim/**

- We will loosely follow the first edition
- There's also a 2nd and 3rd edition (both should be compatible with our course), but they need to be ordered online

Piazza

- Piazza can be used to ask questions
- **https://piazza.com/class/in9iwmhczfd1dv**

Course schedule

Class no.	Date	Topics	Activities
1	25.04	Introduction	
2	26.04	Data structures and libraries	
3	27.04	Data structures	
4	28.04	Problem solving paradigms	
5	29.04	Greedy algorithms	Problem session I
	30.04		
	01.05		Problem sets week 1
6	02.05	Dynamic programming	
7	03.05	Unweighted graphs	
8	04.05	Graphs	
9	05.05	Network flow	
10	06.05		Problem session II
	07.05		
	08.05		Problem sets week 2
11	09.05	Mathematics	
12	10.05	Strings	
13	11.05	Geometry	
14	12.05		
15	13.05		Final exam
	14.05		
	15.05		Problem sets week 3, Bonus problems

Problem sets

- Each class covers a topic
- A talk about the topic before noon
- After noon you get a set of problems about that topic
- Groups of up to three people can discuss the problems, but each individual must write and hand in their own code
    - We will check for similar submissions, and take action if we think that people are cheating

Problem sets

- Each problem set has 5 problems
- Each problem is assigned some amount of points
- To get a perfect score you need to get at least a certain amount of points
- The grade follows linearly from the number of points you get
  - If you get 85% of the required points, your grade is 8.5
  - If you get 50% of the required points, your grade is 5.0
  - If you get 100% or more of the required points, your grade is 10.0

- The deadline for a problem set is the following Sunday
- Except for Friday's problem set, which is handed in the Sunday next week

Bonus problems

- Each problem set contains two challenging bonus problems
- Deadline for all bonus problems is the same as the deadline for the last problem set
- Bonus problems are only taken into account if the student would pass the course before they're taken into account

Late handins, partial grading

- Late handins will not be accepted
  - There should be more than enough time for each problem set
- Solutions will not be partially graded

Problem sessions

- On each Friday is a programming contest
- Problems will be related to topics covered so far
- Teams of up to three people compete together
- Each team can only use a single computer

Final exam

- Will be held on the last Friday
- Similar to a problem set from all the topics

- Need to pass the exam to pass the course

Course evaluation

Problem sets	70%
Problem sessions	10%
Final exam	20%
Bonus problems	20%
Total	120%

- Remember that bonus problems are only considered if the student passes the course, and is only used to raise the final grade
- A final grade greater than 10 will be reduced down to 10

Introduction

The problems

- Typical programming contest problems
- Usually consists of
  - Problem description
  - Input description
  - Output description
  - Example input/output
  - A time limit in seconds
  - A memory limit in bytes
- You are asked to write a program that solves the problem for all valid inputs
- The program must not exceed time or memory limits

Example problem

Problem description

Write a program that multiplies pairs of integers.

Input description

Input starts with one line containing an integer T, where $1 \leq T \leq 100$, denoting the number of test cases. Then T lines follow, each containing a test case. Each test case consists of two integers A, B, where $-2^{20} \leq A, B \leq 2^{20}$, separated by a single space.

Output description

For each test case, output one line containing the value of $A \times B$.

Example problem

Sample input	Sample output
4 3 4 13 0 1 8 100 100	12 0 8 10000

Example solution

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

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

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

Example solution

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

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

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- Is this solution correct?

Example solution

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

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

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- Is this solution correct?
- What if $A = B = 2^{20}$?

Example solution

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

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

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- Is this solution correct?
- What if $A = B = 2^{20}$? The output is 0...

Example solution

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

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

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- Is this solution correct? No!
- What if $A = B = 2^{20}$? The output is 0...

Example solution

- When $A = B = 2^{20}$, the answer should be 2^{40}

Example solution

- When $A = B = 2^{20}$, the answer should be 2^{40}
- Too big to fit in a 32-bit integer, so it overflows

Example solution

- When $A = B = 2^{20}$, the answer should be 2^{40}
- Too big to fit in a 32-bit integer, so it overflows
- Using 64-bit integers should be enough

Example solution

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

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

    for (int t = 0; t < T; t++) {

        long long A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

Example solution

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

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

    for (int t = 0; t < T; t++) {

        long long A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- Is this solution correct?

Example solution

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

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

    for (int t = 0; t < T; t++) {

        long long A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- Is this solution correct? Yes!

Automatic judging

- The problems will be available on Kattis:
- **https://ru.kattis.com/**

- Kattis is an online judge, similar to Mooshak
- You will submit your solutions to Kattis, and get immediate feedback about the solution
- You can submit in any of the supported languages:
  - C
  - C++
  - Java
  - Python 2
  - Python 3
  - C#
  - and others

Judge verdicts

- Feedback about solutions is limited
- You will (usually) receive one of:
  - Accepted
  - Wrong Answer
  - Compile Error
  - Run Time Error
  - Time Limit Exceeded
  - Memory Limit Exceeded
- We will not reveal which test cases Kattis uses to test your solution

Tips

- There are a couple of tips and guidelines you can keep in mind towards becoming a more effective programmer and better problem solver

Tip 0: Faster typing

- Become a faster/better typist
- Don't let your fingers be the limiting factor of solving problems quickly
- Good problem solvers have simple solutions; they don't have to type as much, but it's still important to type in quickly

- TypeRacer is a fun and effective way to practice:
- **http://play.typeracer.com/**

Tip 1: Quickly classify problems

- Practice quickly identifying problem types
- Rate of appearance of different problem types in recent ICPC Asia Regional problem sets (which usually consists of 7-11 problems):

Category	Sub-Category	Frequency
Ad Hoc	Straightforward	1-2
Ad Hoc	Simulation	0-1
Complete Search	Iterative	0-1
Complete Search	Backtracking	0-1
Divide & Conquer		0-1
Greedy	Classic	0
Greedy	Original	1
Dynamic Programming	Classic	0
Dynamic Programming	Original	1-3
Graph		1-2
Mathematics		1-2
String Processing		1
Computational Geometry		1
Harder Problems		0-1

Tip 2: Do Algorithm Analysis

- When solving a problem, our solution has to be fast enough and can not use too much memory
- We also want our solution to be as simple as possible
- We can use Algorithm Analysis to determine if a solution will run within the time limit
- Rule of thumb: 10^8 operations per second

Tip 2: Do Algorithm Analysis

- When solving a problem, our solution has to be fast enough and can not use too much memory
- We also want our solution to be as simple as possible
- We can use Algorithm Analysis to determine if a solution will run within the time limit
- Rule of thumb: 10^8 operations per second
- We want to sort $n \leq 10^6$ integers, and we have 3 seconds.
  - Can we use a simple $O(n^2)$ bubble sort?
  - What about a more complex $O(n \log n)$ merge sort?

Tip 2: Do Algorithm Analysis

- When solving a problem, our solution has to be fast enough and can not use too much memory
- We also want our solution to be as simple as possible
- We can use Algorithm Analysis to determine if a solution will run within the time limit
- Rule of thumb: 10^8 operations per second
- We want to sort $n \leq 10^6$ integers, and we have 3 seconds.
  - Can we use a simple $O(n^2)$ bubble sort?
  - What about a more complex $O(n \log n)$ merge sort?
- We want to sort $n \leq 10^3$ integers, and we have 3 seconds.
  - Can we now use the simple $O(n^2)$ bubble sort?

Tip 2: Do Algorithm Analysis

- When solving a problem, our solution has to be fast enough and can not use too much memory
- We also want our solution to be as simple as possible
- We can use Algorithm Analysis to determine if a solution will run within the time limit
- Rule of thumb: 10^8 operations per second
- We want to sort $n \leq 10^6$ integers, and we have 3 seconds.
  - Can we use a simple $O(n^2)$ bubble sort?
  - What about a more complex $O(n \log n)$ merge sort?
- We want to sort $n \leq 10^3$ integers, and we have 3 seconds.
  - Can we now use the simple $O(n^2)$ bubble sort?
- Always go for the simplest solution that will pass the time limit

Tip 2: Do Algorithm Analysis

- You should practice doing approximate mental calculations
- Rule of thumb: $2^{10} \approx 10^3$

- Sometimes you have a solution that you're not sure is correct
- Try to prove it's correct!
- Even if you don't manage to prove or disprove it, you will probably get a better understanding of the problem

Tip 2: Do Algorithm Analysis

n	Slowest Accepted Algorithm	Example
≤ 10	$O(n!), O(n^{\mathbf{6}})$	Enumerating a permutation
≤ 15	$O(2^n \times n^{\mathbf{2}})$	DP TSP
≤ 20	$O(2^n), O(n^{\mathbf{5}})$	DP + bitmask technique
≤ 50	$O(n^{\mathbf{4}})$	DP with 3 dimensions + $O(n)$ loop, choosing ${}_nC_{\mathbf{4}}$
$\leq 10^{\mathbf{2}}$	$O(n^{\mathbf{3}})$	Floyd Warshall's
$\leq 10^{\mathbf{3}}$	$O(n^{\mathbf{2}})$	Bubble/Selection/Insertion sort
$\leq 10^{\mathbf{5}}$	$O(n \log_{\mathbf{2}} n)$	Merge sort, building a Segment tree
$\leq 10^{\mathbf{6}}$	$O(n), O(\log_{\mathbf{2}} n), O(1)$	Usually, contest problems have $n \leq 10^{\mathbf{6}}$ (to read input)

Tip 3: Master Programming Languages

- You should know your programming language like the back of your hand
- This includes your programming language's library
  - C++'s Standard Template Library
  - The Java Class Library
- If it's already implemented in the standard library, you usually don't need to implement it yourself

Tip 4: Test your solution

- You want to make sure your solution is correct and runs within the time limit
- Or you already know it's wrong, but don't know why

- Try to break your solution by finding a counterexample (an input for which your solution gives incorrect output, or takes too long to compute an answer)
- Try edge cases, large inputs, ...

Tip 5: Practice and more practice

- Problem solving and programming skills come with practice
- Lots of online judges that let you solve problems from past contests
- Some of these online judges also hold contests frequently
- Open Kattis, Codeforces, HackerRank, Codechef, UVa, TopCoder, ...

Ad Hoc Problems

Ad Hoc problems

- The simplest kind of problem
- Just do what the problem description tells you
- Straightforward or a simulation
- Time limit is not an issue
- Sometimes long and misleading problem descriptions
- Sometimes tricky edge cases
- Complex problems can be hard to implement

Problem: Cost Cutting

Company XYZ have been badly hit by recession and is taking a lot of cost cutting measures. Some of these measures include giving up office space, going open source, reducing incentives, cutting on luxuries and issuing pink slips.

They have got three (3) employees working in the accounts department and are going to lay-off two (2) of them. After a series of meetings, they have decided to dislodge the person who gets the most salary and the one who gets the least. This is usually the general trend during crisis like this. You will be given the salaries of these 3 employees working in the accounts department. You have to find out the salary of the person who survives.

Problem: Cost Cutting

Input

The first line of input is an integer T ($T < 20$) that indicates the number of test cases. Each case consists of a line with 3 distinct positive integers. These 3 integers represent the salaries of the three employees. All these integers will be in the range $[1000, 10000]$.

Output

For each case, output the case number followed by the salary of the person who survives.

Problem: Cost Cutting

Sample input	Sample output
3 1000 2000 3000 3000 2500 1500 1500 1200 1800	Case 1: 2000 Case 2: 2500 Case 3: 1500

Cost Cutting: Solution

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

int main() {
    int T;
    scanf("%d", &T);

    for (int t = 0; t < T; t++) {

        int salary[3];
        scanf("%d", &salary[0]);
        scanf("%d", &salary[1]);
        scanf("%d", &salary[2]);

        sort(salary, salary + 3);

        printf("Case %d: %d\n", t + 1, salary[1]);
    }

    return 0;
}
```

Problem: SMS Typing

Cell phones have become an essential part of modern life. In addition to making voice calls, cell phones can be used to send text messages, which are known as SMS for short. Unlike computer keyboards, most cell phones have limited number of keys. To accommodate all alphabets, letters are compacted into single key. Therefore, to type certain characters, a key must be repeatedly pressed until that character is shown on the display panel.

In this problem we are interested in finding out the number of times keys on a cell phone must be pressed to type a particular message.

Problem: SMS Typing

In this problem we will assume that the key pad of our cell phone is arranged as follows.

	abc	def
ghi	jkl	mno
pqrs	tuv	wxyz
	<SP>	

In the above grid each cell represents one key. Here <SP> means a space. In order to type the letter ‘a’, we must press that key once, however to type ‘b’ the same key must be repeatedly pressed twice and for ‘c’ three times. In the same manner, one key press for ‘d’, two for ‘e’ and three for ‘f’. This is also applicable for the remaining keys and letters. Note that it takes a single press to type a space.

Problem: SMS Typing

Input

The first line of input will be a positive integer T where T denotes the number of test cases. T lines will then follow each containing only spaces and lower case letters. Each line will contain at least 1 and at most 100 characters.

Output

For every case of input there will be one line of output. It will first contain the case number followed by the number of key presses required to type the message of that case. Look at the sample output for exact formatting.

Problem: SMS Typing

Sample input	Sample output
2 welcome to ulab good luck and have fun	Case #1: 29 Case #2: 41

SMS Typing: Solution

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

string keys[12] = {
    "",     "abc", "def",
    "ghi",  "jkl", "mno",
    "pqrs", "tuv", "wxyz",
    "",     " ",   ""
};

int main() {
    int T;
    scanf("%d\n", &T);

    for (int t = 0; t < T; t++) {

        // Each test case is handled here
    }

    return 0;
}
```

SMS Typing: Solution

```
// Each test case:

string line;
getline(cin, line);

int cnt = 0;
for (int i = 0; i < line.size(); i++) {
    int cur;
    for (int j = 0; j < 12; j++) {
        for (int k = 0; k < keys[j].size(); k++) {
            if (line[i] == keys[j][k]) {
                cur = k + 1;
            }
        }
    }

    cnt += cur;
}

printf("Case #%d: %d\n", t + 1, cnt);
```

Problem set 1

- The first problem set is already online
- Deadline is next Sunday
- We urge you to start right away nonetheless