



SUGGESTED SOLUTIONS OF SHARIF INTERNET CONTEST 2008

**Aideen NasiriShargh
Nima Ahmadi Pour Anari
SaeedReza Seddighin**

PROBLEM A: A DECIMAL NUMBER

- The range of an **int** is

[-2,147,483,648, 2,147,483,647]

- **Note:** `abs(MIN_INT)` (that is `2,147,483,648`) does not fit in an **int**!

- **Another Solution:** read it as a **string**!



PROBLEM B: BOLD, ITALIC, UNDERLINE

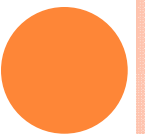
- A Good Solution: Recursive Parsing
 - `string fix(string input, int mask)` can take an input string and fix it. Mask can be a 3 bit int, standing on B, U or I-ness of the **exactly** (adjacent) outer tag.
 - If input is something like “<x>[inner]</x>” (where these X’s are **matching** tags) call `fix([inner], mask | (1<< (x’s id)))`.
 - Otherwise, if input is not of that kind and mask is not zero, write mask’s implementation around the input. E.g. `fix(“a<u>hi</u>z”, 1)` may return

```
“<SPAN STYLE=\“FONT-WEIGHT:BOLD\”>” + “A” +  
FIX(“<U>HI</U>”,0) + “Z” +  
“</SPAN>”;
```



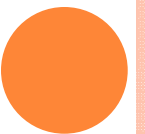
PROBLEM C: CAPSLOCK

- A boolean variable to hold if it CapsLock is ON or off.
- Walk through the text and write non-! Ones.



PROBLEM D: DMAIL!

- Why making the priority twice and hold only powers of 2? Just keep $\log(\text{priority})$ that would be a number between 1 and 126 and works properly!
- `set<pair<int,string> >` in C++ can handle it pretty fine!



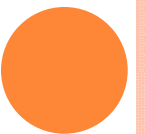
PROBLEM E: EMPIRE

- You are asked to create 2 components (with railroads) and then connect them with an airline.
- Postponing airline construction to the last step and thinking on Kruskal algorithm to get Minimum Spanning Tree, the problem is:
 - **Find an MST, discard the heaviest edge of it!**



PROBLEM F: FFS

- BackTracking, of course!
- Storing achievable states (holding everyone's cash and value and owner) in a set<State>, you can avoid repeated states.
- More bounds?



PROBLEM G: GHANDAK AND KEY

- Knowing the carry's in the sum of $a+b$, results in equations with digits of a, b as variables.
- Solutions to these equations (i.e. first digit of $a +$ first digit of $b = 12$) can be found by a pre-calculation.
- **More clever solution:** Dynamic programming. $\text{Dynamic}[k][c]$ stores the number of ways to choose the leftmost digits of a, b (except the k rightmost ones) so that their sum is what should be, assuming a carry of c on the k^{th} position.



PROBLEM H: HOW TO DANCE ROBOT

○ BFS!

- Each node has the 3 field: Xposition, Yposition, AccumulatorValue.
- We have 100x100x255 nodes and each of them is connected to at most 12 nodes.
- Explore them in “ENSW” order and then “*+” operations to reach the lexicographically-first path before any other!
- Keep parent’s path (like “E+” or “N*”) in 2 characters and write the path recursively.



PROBLEM I: ICALCULATE

- **Note:** Intermediate values may be very large.
- Wrong solution: Use Java's BigInteger. It's not going to help, because the results of operations may be too long to store or do computations with.
- Correct solution: Choose a prime P (larger than 2 times the magnitude of the result), and do computations mod P . If none of the denominators are divisible by P , we're good. Else find another random P , and start over.



PROBLEM J: JULIET

- For each leaf find all intersections of the line joining Juliet to the shore, and the perimeter of the leaf.
- Amongst these points, the point like P whose $\text{inner_product}(P, \text{Juliet's position})$ is maximum or minimum is the entering or exiting point respectively. Put these two inside a list L .
- Add Juliet's position and the origin to L and then sort the points in L with respect to the function $\text{inner_product}(-P, \text{Juliet})$.
- Iterate over L . Whenever you reach an entering point add 1 to a counter. Whenever you reach an exiting point add -1 to it. Whenever the counter is not 0 and we're between Juliet and the origin, we are on a leaf.
- This way we can find the length of the path we're on a leaf.

