J224 - Digit Implant Strategy

Ethen Yuen {ethening}

2022-01-29

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Background

Problem Idea by ethening

Preparation by ethening, VCLH

Problem Restatement

Given integers **S, T** (1 ≤ length of S, T ≤ 10^6), and digit **x** (1 ≤ x ≤ 9)

Insert **x** into **T** to construct **T'**

Output **T'** such that **abs(S - T')** is minimized

E.g. T = 146, x = 3,

T' can be <u>3</u>146, 1<u>3</u>46, 14<u>3</u>6, 146<u>3</u>

If S is 1459, then 1463 should be outputted

87663 521 8	8521

99000 9999 8	98999

Statistics

0 points	9	+ 3	+ 0	+ 0	= 12
6 points	2	+ 2	+ 0	+ 0	= 4
11 points	6	+ 3	+ 6	+ 1	= 16
17 points	0	+ 2	+ 2	+ 0	= 4
23 points	0	+ 0	+ 1	+ 0	= 1
29 points	1	+ 2	+ 1	+ 2	= 6
40 points	1	+ 2	+ 4	+ 4	= 11
100 points	0	+ 0	+ 0	+ 1	= 1

First solved by **cwong** at **2h 46m 17s**

SUBTASK

For all cases:
$1 \leq \text{Length of } S, \text{Length of } T \leq 10^6$
$1 \leq x \leq 9$

	Points	Constraints
1	11	$1 \leq \text{Length of } S, \text{Length of } T \leq 8$
2	6	$(\text{Length of } S) < (\text{Length of } T) + 1$ x and the digits of S and T is either 3 or 7
3	12	$(\text{Length of } S) > (\text{Length of } T) + 1$ x and the digits of S and T can only be 3, 5, or 7
4	11	$(\text{Length of } S) \neq (\text{Length of } T) + 1$
5	24	The first digits of S and T are different.
6	36	No additional constraints

Subtask 1

Subtask 1 (11%): $1 \leq$ Length of S, T ≤ 8.

- The numbers are **small** enough to be stored using 32-bit integer.
- There are at most 9 possible T’.
- **Exhaust** all and find the T’ that achieved minimum difference.

Insertion of digit

Suppose length of T is k.

For discussion purpose, we denoted the different possible T’ by
T0, T1, ..., Tk, where **Ti** is produced by inserting **x** before the **i-th digit of T**.
(Tk means inserting x after all digits)

E.g. T = 146, x = 3,
Then T0 = <u>3</u>146, T1 = 1<u>3</u>46, T2 = 14<u>3</u>6, T3 = 146<u>3</u>

Subtask 1

Subtask 1 (11%): 1 ≤ Length of S, T ≤ 8.

- **Exhaust** all and find the T' that achieved minimum difference.
- Ti could be calculated by some integer division and modulo.

Score: 11

```
int pwr = 1000'000'000;

for (int i = 0; i <= 8; i++) {
    int l = T / pwr;
    int r = T % pwr;

    /* Ti = inserting x between l and r */
    int Ti = l * pwr * 10 + x * pwr + r;

    /* Update answer with Ti here */
    /* ... */

    pwr /= 10;
}
```

Subtask 2

Subtask 2 (6%): (Length of S) < (Length of T) + 1,
x and the digits of S and T is either 3 or 7

- Length of T’ always greater than S.
- Value of T’ always greater than S.
- To achieve minimum abs(S - T’), we have to **minimize T’**. (S is not important in this subtask)

Subtask 2

- To achieve minimum abs(S - T'), we have to minimize T'
- Let's observe:

e.g. T = 377373773737

when x = 3, ans = <u>3</u>377373773737; when x = 7, ans = 377373773737<u>7</u>

e.g. T = 77777373

when x = 3, ans = <u>3</u>77777373; when x = 7, ans = 77777373<u>7</u>

It seems that we always want to insert 3 **at front**, and 7 **at back**.

Subtask 2

Subtask 2 (6%): (Length of S) < (Length of T) + 1,
x and the digits of S and T is either 3 or 7

- We always want to insert 3 at front, and 7 at back.
- This make sense since 3 is smallest digit in the number and should be put in front to minimize; similar argument for 7.

Subtask 2

Subtask 2 (6%): (Length of S) < (Length of T) + 1,
x and the digits of S and T is either 3 or 7

- Just print **'3' + T** for x = 3 and **T + '7'** for x = 7.
- As S and T can be large, we would like to store them in C++ string.
  (array of int / array of char works fine, however we could use C++ string function to our advantages, which will be shown later)

Score: 6 (Cumulative: 17)

Subtask 3

Subtask 3 (12%): (Length of S) > (Length of T) + 1,
x and the digits of S and T is either 3, 5, or 7

- Opposite to Subtask 2, this time we have to maximize T'.
- We always want to insert 3 at back, and 7 at front. What about 5?

Subtask 3

- To achieve minimum abs(S - T'), we have to maximize T'
- Let's observe when x = 5:
  - e.g. T = 77777373, ans = 777775373
  - e.g. T = 55575535, ans = 555755535
  - e.g. T = 577575755, ans = 5775757555
- Some rules can be concluded from these:
  - We never want to insert 5 in front of 7, because inserting after that 7 would always yield a larger number; insert 5 in front of another 5 is the same as inserting after;
  - **Inserting 5 in front of a 3 always yield a larger number** than inserting after that 3.

Subtask 3

- We would insert 5 **in front of the first occurrence of 3**; if no 3's are present in the number, insert 5 at the back.
  - If the inserted x is followed by 5, or 7, there are always a larger alternative.
  - Why should it be inserted in front of first occurrence of 3?
- e.g. T = 575535337

  Inserting right before first occurrence of 3: 5755535337

  Inserting after first occurrence of 3: 57553xxxxx (must be smaller)

Subtask 3

Subtask 3 (12%): (Length of S) > (Length of T) + 1,
x and the digits of S and T is either 3, 5, or 7

- We always want to insert 3 at back, and 7 at front.
- We would insert 5 **in front of the first occurrence of 3** (or at the back).
    - If first occurrence of 3 is the **i-th digit of T**, the answer would be **T_i**.

Subtask 3

Subtask 3 (12%): (Length of S) > (Length of T) + 1,
x and the digits of S and T is either 3, 5, or 7

```
/* Inserting x before the i-th digit of T*/
string Ti = T;
Ti.insert(i, x);
```

- We always want to insert 3 at back, and 7 at front.
- For x = 5, if first occurrence of 3 is the **i-th digit of T**, the answer would be **Ti** (or Tk if there are no 3).
    - Ti can be constructed either by looping manually or with insert function.

Score: 12 (Cumulative: 29)

Subtask 4

Subtask 4 (11%): (Length of S) ≠ (Length of T) + 1,

- We have to combine Subtask 2 and 3 ideas and generalize it to tackle general S, T (without digits constrained to be some particular values).
- Let's tackle the case where (Length of S) > (Length of T) + 1 first:
  - In Subtask 3, we would insert 5 in front of the first occurrence of 3, because it give us the largest number.

Subtask 4

Subtask 4 (11%): (Length of S) ≠ (Length of T) + 1,

- Let's tackle the case where (Length of S) > (Length of T) + 1 first:
  - In Subtask 3, we would insert 5 in front of the first occurrence of 3, because it give us the largest number. We should **insert x in front of the first occurrence of y (where y < x)** for getting the largest number.
  - Suppose **T = abcdeyqrstu…** (abcde are digits ≥ x)
    Inserting right before y: **abcdexyqrstu…**
    Inserting after y: **abcdey…** (must be smaller)

(y ≤ x) does not work for case like T = 573, x = 5, where the optimal answer is 5753.

Subtask 4

Subtask 4 (11%): (Length of S) ≠ (Length of T) + 1,

- For (Length of S) > (Length of T) + 1:
  - We should **insert x in front of the first occurrence of y (where y < x)**
- For (Length of S) < (Length of T) + 1:
  - We should **insert x in front of the first occurrence of y (where y > x)**

Score: 29 (Cumulative: 40)

Remember to handle cases where all the digits are ≥ x / ≤ x

Subtask 5

Subtask 5 (24%): The first digits of S and T are different.

- You should have noticed now, the real challenge of the problem is when T' is in equal length with S.
- For when the lengths are not equal, just run the solution of Subtask 4.

- What is so special about the first digits of S and T???

Subtask 5

Subtask 5 (24%): The first digits of S and T are different.

- Let’s suppose S[0] = ‘7’, T[0] = ‘3’.
- If we are inserting x at any position other than at the front,
  - S would be **7abc……def**, and T’ would be **3qxs…tuv**, where S and T’ are equal length.
  - S must be larger than T’ since the first digit is larger.
  - We want to **maximize T’ to minimize abs(S - T’)**. ← Same as **Subtask 4**
- Useful problem-solving technique: **Reduce to known problem**

Subtask 5

Subtask 5 (24%): The first digits of S and T are different.

- Let's suppose S[0] > T[0].
- **Case 1:** we are inserting x at any position other than at the front,
  - We want to **maximize T'** using **Subtask 4** idea, let's suppose the result is **Ta**
- **Case 2:** we are inserting x at the front (T0).
- We only have two candidate answers, Ta and T0.
  - Just compare **abs(S - Ta)** and **abs(S - T0)**, to see which is smaller.

Subtask 5

- Compare **abs(S - Ta)** and **abs(S - T0)**, to see which is smaller.
  - We need to do **High Precision Arithmetic (HPA)** manually, using string for big number.
  - Luckily, S and T' are of same length which makes it a bit less complicated.
- We only need to implement two functions: one for **comparing** two big numbers, one for calculating the **difference** between two big numbers.
  - Plan is: Compare S & Ta and subtract the smaller one from the bigger one; Do the same for S & T0.
    Then compare the two differences to see which is smaller.

Subtask 5

```
// return true if x >= y, suppose x and y are of
same length
bool cmp(const string& x, const string& y) {
    int len = x.length();
    for (int i = 0; i < len; i++) {
        if (x[i] > y[i]) return 1;
        if (x[i] < y[i]) return 0;
    }
    return 1;
}
```

C++ have built-in lexicographical comparator with string, which you could use in this scenario (because x and y are of same length). Basically do `return x >= y;`

```
// return x - y, given x >= y and they are of same
length
string subtract(string x, string y) {
    int len = x.length();
    for (int i = len - 1; i >= 0; i--) {
        x[i] -= (y[i] - '0');
        if (x[i] < '0') {
            x[i] += 10;
            --x[i - 1];
        }
    }
    return x;
}
```

Subtask 5

Subtask 5 (24%): The first digits of S and T are different.

- We have handled S[0] > T[0].
- For S[0] < T[0], the only difference is that you should try to **minimize Ta**.

Score: 24 (Cumulative: 64)

Full Solution

Subtask 6 (36%): No additional constraints.

- What about the case where there is a common prefix of S and T?
  - e.g. S = 3141539897, T = 3141555091, x = 2
- Could we just ignore the prefix and perform Subtask 5 solution?
  - Ta = 3141550291 ← Optimal T' that you could get inserting after '5'.
  - Tb = 3141525091 ← Similar to T0 in **Subtask 5**
- In **most** cases, this will give you the optimal answer. By ignoring the prefix, it will eliminate itself in abs(S - T') and yield a small difference.

Full Solution

- In **most** cases, this will give you the optimal answer.
- There are some cases that will mess this up, one of them are given to you in the samples.

99000 9999 8	98999

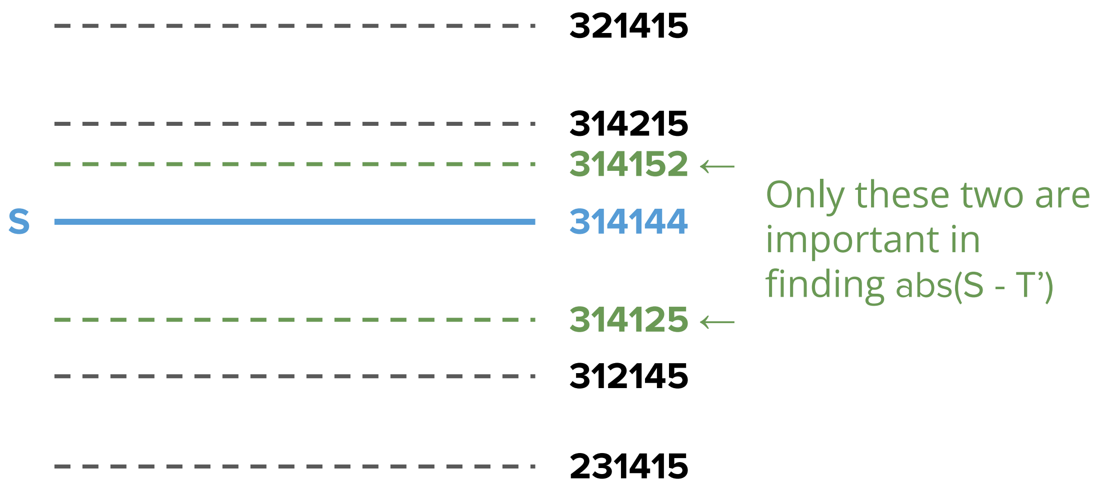
- Ta = 99989, abs(S - Ta) = 989
- Tb = 99899, abs(S - Tb) = 899
- Topt = 98999, **abs(S - Topt) = 1**

Full Solution

- **Case 1:** Inserting x after the prefix.
  - Reduce to only 2 candidates to try by Subtask 5.
- **Case 2:** Inserting x in between / before the prefix.
  - Try all possibilities? Would let to TLE.
  - Can we reduce the candidates as well? Most insertion seems would make abs(S - T') a lot bigger, especially if x is inserted in relatively front.

Reforming the Problem

- Let's study the problem statement again.
- "Output **T'** such that **abs(S - T')** is minimized"



Reforming the Problem

- “Output **T’** such that **abs(S - T’)** is minimized”

- Find the **minimum T’ that T’ ≥ S** &&
- Find the **maximum T’ that T’ ≤ S**.
- Then take the one that yields a smaller absolute difference.

Full Solution

- **Case 1:** Inserting x after the prefix.
  - Reduce to only 2 candidates to try by Subtask 5.
- **Case 2:** Inserting x in between / before the prefix.
  - Find the minimum T’ that T’ > S, we denoted that by **Tc**
  - Find the maximum T’ that T’ < S, we denoted that by **Td**

    (The T’ = S part must be handled by Case 1)
  - How?

Full Solution

- e.g. S = 614152abc……, T = 614152qrs……, x = 4
  - And we only consider inserting in the prefix part

$\underline{4}$614152qrs...	Inserting before 6	< S
6$\underline{4}$14152qrs...	Inserting before 1	> S
61$\underline{4}$4152qrs...	Inserting before 4	We could always ignore inserting x before the same digit, because it is actually the same number as below
614$\underline{4}$152qrs...	Inserting before 1	> S
6141$\underline{4}$52qrs...	Inserting before 5	< S
61415$\underline{4}$2qrs...	Inserting before 2	> S

Full Solution

- Inserting x before y (y < x) would make T' > S

6414152qrs...	Inserting before 1	> S

- Inserting x before z (z > x) would make T' < S

4614152qrs...	Inserting before 6	< S

- Because the part prior to the insertion is the same for S and T'. While the most significant digit that is different is the insertion position (x and the original digit there).

Full Solution

- Inserting x before y (y < x) would make T' > S

6414152qrs...	Inserting before 1	> S
6144152qrs...	Inserting before 1	> S
6141542qrs...	Inserting before 2	> S

- Inserting x before the **last occurence of y (that y < x)** would yield the minimum T'. (The **Tc** that we are looking for!)
    - The reason of this should be easy to seen from the aligned numbers from the above table.

Full Solution

- **Case 1:** Inserting x after the prefix.
  - Fixed the first different digit, and perform Subtask 5 solution to get **Ta**.
  - Insert x right after prefix to get **Tb**.
- **Case 2:** Inserting x in between / before the prefix.
  - Find the minimum T' that T' > S (**Tc**) .
    - Inserting x before the last occurence of y (that y < x) would yield the minimum T'.
  - Find the maximum T' that T' < S (**Td**).
    - Inserting x before the last occurence of y (that y > x) would yield the maximum T'.

Full Solution

Subtask 6 (36%): No additional constraints.

- e.g. S = 3141539897, T = 3141555091, x = 2
  - Ta = 3141550291
  - Tb = 3141525091
  - Tc = 3142155091
  - Td = 3141255091
- Calculate all of **abs(S - T{a, b, c, d})** and output the one who achieve the smallest difference.

Score: 100

Aftermath

The full solution considers four candidates:

- Insert some position after common prefix **Ta (a>b)**
- Insert x **right after** common prefix to get **Tb**.
- Find the minimum T’ such that T’ > S **Tc (c<b)**
- Find the maximum T’ such that T’ < S **Td (d<b)**

Aftermath

The full solution considers four candidates:

- Insert some position after common prefix **Ta (a>b)**
- Insert x **right after** common prefix to get **Tb**.
- Find the minimum T’ such that T’ > S **Tc (c<b)**
- Find the maximum T’ such that T’ < S **Td (d<b)**

⇒ 1. You can reduce the no. of candidates from 4 to 3.

Aftermath

The full solution considers four candidates:

- Insert some position after common prefix **Ta ($a>b$)**
- Insert x **right after** common prefix to get **Tb**.
- Find the minimum T’ such that $T' > S$ **Tc ($c<b$)** **only if $Tb < S$**
- Find the maximum T’ such that $T' < S$ **Td ($d<b$)** **only if $Tb > S$**

⇒ 1. You can reduce the no. of candidates from 4 to 3.

Aftermath

The full solution considers four candidates:

- Insert some position after common prefix **Ta (a>b)**
- Insert x **right after** common prefix to get **Tb**.
- Find the minimum T' such that T' > S **Tc (c<b)** **only if Tb < S**
- Find the maximum T' such that T' < S **Td (d<b)** **only if Tb > S**

⇒ 1. You can reduce the no. of candidates from 4 to 3.

⇒ 2. There isn't much choice for **c/d**. In fact we only need to consider 1.

Aftermath

The full solution considers four candidates:

- Insert some position after common prefix **Ta (a>b)**
- Insert x **right after** common prefix to get **Tb**.
- Insert x into common prefix **T(b-1)**
- ~

⇒ 1. You can reduce the no. of candidates from 4 to 3.

⇒ 2. There isn't much choice for **c/d**. In fact we only need to consider 1. (Why?)