Sorting numbers from 1 to 999,999,999 in words as strings

Interesting programming puzzle:

> If the integers from 1 to 999,999,999 are written as words, sorted alphabetically, and concatenated, what is the 51 billionth letter?
>
> To be precise: if the integers from 1 to 999,999,999 are expressed in words (omitting spaces, 'and', and punctuation - see note below for format), and sorted alphabetically so that the first six integers are
>
> - eight
> - eighteen
> - eighteenmillion
> - eighteenmillioneight
> - eighteenmillioneighteen
> - eighteenmillioneighteenthousand
>
> and the last is
>
> - twothousandtwohundredtwo
>
> then reading top to bottom, left to right, the 28th letter completes the spelling of the integer "eighteenmillion".
>
> The 51 billionth letter also completes the spelling of an integer. Which one, and what is the sum of all the integers to that point?
>
> *Note:* For example, 911,610,034 is written "ninehundredelevenmillionsixhundredtenthousandthirtyfour"; 500,000,000 is written "fivehundredmillion"; 1,709 is written "onethousandsevenhundrednine".

I stumbled across this on a programming blog 'Occasionally Sane', and couldn't think of a neat way of doing it, the author of the relevant post says his initial attempt ate through 1.5GB of memory in 10 minutes, and he'd only made it up to 20,000,000 ("twentymillion").

Can anyone ~~think of~~ *~~come up with~~* **share with the group** a novel/clever approach to this?

algorithm

edited Sep 29 '09 at 21:51

community wiki
3 revs, 2 users 99%
Dominic Rodger

14 Yes, I can think of a clever approach. – chrispy Sep 29 '09 at 21:44

3 Fourty-Two..... – Mike Robinson Sep 29 '09 at 21:51

5 Just ruined my evening (xkcd.com/356) – Alan Jackson Sep 29 '09 at 22:04

2 Actually, this problem hardly needs a program at all. It's readily solvable with a spreadsheet or even by hand, if you're patient. – RBarryYoung Sep 29 '09 at 23:32

6 I have discovered a truly marvelous approach to this, which this comment field is too narrow to contain. – Thomas Padron-McCarthy Sep 30 '09 at 13:13

show **10** more comments

20 Answers

Edit: Solved!

You can create a generator that outputs the numbers in sorted order. There are a few rules for comparing concatenated strings that I think most of us know implicitly:

- a < a+b, where b is non-null.
- a+b < a+c, where b < c.
- a+b < c+d, where a < c, and a is not a subset of c.

If you start with a sorted list of the first 1000 numbers, you can easily generate the rest by appending "thousand" or "million" and concatenating another group of 1000.

Here's the full code, in Python:

```
import heapq

first_thousand=[('', 0), ('one', 1), ('two', 2), ('three', 3), ('four', 4),
                ('five', 5), ('six', 6), ('seven', 7), ('eight', 8),
                ('nine', 9), ('ten', 10), ('eleven', 11), ('twelve', 12),
                ('thirteen', 13), ('fourteen', 14), ('fifteen', 15),
                ('sixteen', 16), ('seventeen', 17), ('eighteen', 18),
                ('nineteen', 19)]
tens_name = (None, 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
             'seventy','eighty','ninety')
for number in range(20, 100):
    name = tens_name[number/10] + first_thousand[number%10][0]
    first_thousand.append((name, number))
for number in range(100, 1000):
    name = first_thousand[number/100][0] + 'hundred' + first_thousand[number%100][0]
    first_thousand.append((name, number))

first_thousand.sort()

def make_sequence(base_generator, suffix, multiplier):
    prefix_list = [(name+suffix, number*multiplier)
                   for name, number in first_thousand[1:]]
    prefix_list.sort()
    for prefix_name, base_number in prefix_list:
        for name, number in base_generator():
            yield prefix_name + name, base_number + number
    return

def thousand_sequence():
    for name, number in first_thousand:
        yield name, number
    return

def million_sequence():
    return heapq.merge(first_thousand,
                       make_sequence(thousand_sequence, 'thousand', 1000))
```

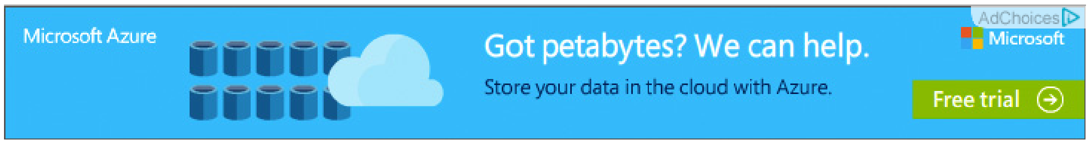
It took a while to run, about an hour. The 51 billionth character is the last character of sixhundredseventysixmillionsevenhundredfortysixthousandfivehundredseventyfive, and the sum of the integers to that point is 413,540,008,163,475,743.

edited Oct 1 '09 at 14:11

community wiki
2 revs
Mark Ransom

+1 for complete working solution. It can be made faster of course. – Don Roby Jul 22 '11 at 2:36

add a comment


Microsoft Azure
AdChoices
Microsoft
Got petabytes? We can help.
Store your data in the cloud with Azure.
Free trial

I'd sort the names of the first 20 integers and the names of the tens, hundreds and thousands, work out how many numbers start with each of those, and go from there.

For example, the first few are `[ eight, eighteen, eighthundred, eightmillion, eightthousand, eighty, eleven, ... .`

The numbers starting with "eight" are 8. With "eighthundred", 800-899, 800,000-899,999, 800,000,000-899,999,999. And so on.

The number of letters in the concatenation of words for 0 (represented by the empty string) to 99 can be found and totalled; this can be multiplied with "thousand"=8 or "million"=7 added for higher ranges. The value for 800-899 will be 100 times the length of "eighthundred" plus the length of 0-99. And so on.

edited Sep 29 '09 at 23:01 — community wiki
5 revs
Pete Kirkham

+1. This is the only sort of approach that's going to work as the numbers get large; everyone's idea of actually storing a billion strings is crazy! – ShreevatsaR Sep 29 '09 at 23:16

Agreed, this is one of only two correct approaches that also have a chance of ever finishing (the other is elaboration streaming, which is both very difficult to get right and much slower). Still, the answer given is very abstract and the real devil is in the details. – RBarryYoung Sep 30 '09 at 16:59

Given the question appears to be still live for selecting candidates for interview, I didn't think it sporting to post a fully worked out answer. – Pete Kirkham Sep 30 '09 at 17:16

This is part of a job application?!? Heh, I guess the test is to see who's smart enough to realize that it's actually harder to solve it with a program than it is with Excel of a paper and pencil. fair enough...(+1) – RBarryYoung Oct 1 '09 at 4:11

3 Dominic: nope, this really is easier for a person to solve than most folks here realize. What Pete describes above is a 5000-foot description of the best and fastest way to solve. Using a program for this is harder because there is a significant amount of one-time special casing involved. By the time you figure all of that out, code it up, test it, debug it, and then finally run it, then yes you could have done it by hand. – RBarryYoung Oct 1 '09 at 14:35

show **1** more comment

This guy has a solution to the puzzle written in Haskell. Apparently Michael Borgwardt was right about using a Trie for finding the solution.

answered Sep 29 '09 at 22:28 — community wiki
Anax

5 Yes, it "uses" a trie, but that solution does not actually store all billion strings, or anywhere close to that. Instead, it generates only the relevant parts of the trie after the question is asked, cleverly skipping over *most* of the billion numbers using a bit of **automatic differentiation** and other ideas. Your link is the fourth (and presumably final) part in the series; look at Part 2 for the crucial ideas: conway.rutgers.edu/~ccshan/wiki/blog/posts/WordNumbers2 As such, despite sharing a keyword ("trie"), it is closer to Pete Kirkham's answer than Michael Borgwardt's. :-) – ShreevatsaR Sep 30 '09 at 13:58

Thanks for taking the trouble and sharing this information. – Anax Sep 30 '09 at 14:19

add a comment

Those strings are going to have lots and lots of common prefixes - perfect use case for a trie, which would drastically reduce memory usage and probably also running time.

answered Sep 29 '09 at 21:52 — community wiki
Michael Borgwardt

1 So that'd work - at the cost of greater time-complexity you could spin through the trie after each insertion to discard any elements after the 51 billionth character too, so 51 billion characters + a bit would be the upper bound on the data needed to be stored in memory. – Dominic Rodger Sep 29 '09 at 22:02

@Dominic: And how much time would generating all 999,999,999 strings and inserting them into a trie take? – MAK Sep 30 '09 at 14:29

@MAK - unless I've misunderstood tries, I believe it'd be O(nlogn). As to absolute time, I've no idea. An awfully long time I'd expect. – Dominic Rodger Oct 1 '09 at 11:55

add a comment

(The first attempt at this is wrong, but I will leave it up since it's more useful to see mistakes on the way to solving something rather than just the final answer.)

I would first generate the strings from 0 to 999 and store them into an array called thousandsStrings. The 0 element is "", and "" represents a blank in the lists below.

The thousandsString setup uses the following:

```
Units: "" one two three ... nine

Teens: ten eleven twelve ... nineteen

Tens: "" "" twenty thirty forty ... ninety
```

The thousandsString setup is something like this:

```
thousandsString[0] = ""

for (i in 1..10)
   thousandsString[i] = Units[i]
end

for (i in 10..19)
   thousandsString[i] = Teens[i]
end

for (i in 20..99)
   thousandsString[i] = Tens[i/10] + Units[i%10]
end

for (i in 100..999)
   thousandsString[i] = Units[i/100] + "hundred" + thousandsString[i%100]
end
```

Then, I would sort that array alphabetically.

Then, assuming t1 t2 t3 are strings taken from thousandsString, all of the strings have the form

t1

OR

t1 + million + t2 + thousand + t3

OR

t1 + thousand + t2

To output them in the proper order, I would process the individual strings, followed by the millions strings followed by the string + thousands strings.

```
foreach (t1 in thousandsStrings)

   if (t1 == "")
     continue;

   process(t1)

   foreach (t2 in thousandsStrings)
      foreach (t3 in thousandsStrings)
         process (t1 + "million" + t2 + "thousand" + t3)
      end
   end

   foreach (t2 in thousandsStrings)
       process (t1 + "thousand" + t2)
   end
end
```

where process means store the previous sum length and then add the new string length to the sum and if the new sum is >= your target sum, you spit out the results, and maybe return or break out of the loops, whatever makes you happy.

==

Second attempt, the other answers were right that you need to use 3k strings instead of 1k strings as a base.

Start with the thousandsString from above, but drop the blank "" for zero. That leaves 999 elements and call this uStr (units string).

Create two more sets:

```
tStr = the set of all uStr + "thousand"

mStr = the set of all uStr + "million"
```

Now create two more set unions:

```
mtuStr = mStr union tStr union uStr

tuStr = tStr union uStr
```

Order uStr, tuStr, mtuStr

Now the looping and logic here are a bit different than before.

```
foreach (s1 in mtuStr)
   process(s1)

   // If this is a millions or thousands string, add the extra strings that can
   // go after the millions or thousands parts.

   if (s1.contains("million"))
      foreach (s2 in tuStr)
         process (s1+s2)

         if (s2.contains("thousand"))
            foreach (s3 in uStr)
               process (s1+s2+s3)
            end
         end
      end
   end

   if (s1.contains("thousand"))
      foreach (s2 in uStr)
         process (s1+s2)
      end
   end
end
```

edited Sep 30 '09 at 13:53

community wiki
3 revs
John

Almost. First problem, 8000000 becomes "eightmillionthousand", but this is easily fixed. Second problem, you produce an incorrect order: "eight", "eightmillion", "eightthousand", "eighthundred". – Mark Ransom Sep 30 '09 at 5:57

Ah, you're right about needing to ignore the blanks, and I see you're right about the eighthundred vs eightmillion ordering. So it looks like the answer is the one given by someone else using 3k strings instead of 1k strings so you use the answers given by Pete Kirkham or John W – John Sep 30 '09 at 12:50

+1: not 1005, but still far better than the two Trie answers that everyone is upvoting for some reason (tries are NOT the way to solve this problem). – RBarryYoung Sep 30 '09 at 17:00

sigh "Not 100%", stupid fingers... – RBarryYoung Sep 30 '09 at 17:01

add a comment

Here's my python solution that prints out the correct answer in a fraction of a second. I'm not a python programmer generally, so apologies for any egregious code style errors.

```
#!/usr/bin/env python

import sys

ONES=[
    "",         "one",      "two",      "three",        "four",
    "five",     "six",      "seven",    "eight",        "nine",
    "ten",      "eleven",   "twelve",   "thirteen",     "fourteen",
    "fifteen",  "sixteen",  "seventeen","eighteen",     "nineteen",
]
TENS=[
    "zero",     "ten",      "twenty",   "thirty",       "forty",
    "fifty",    "sixty",    "seventy",  "eighty",       "ninety",
]

def to_s_h(i):
    if(i<20):
        return(ONES[i])
    return(TENS[i/10] + ONES[i%10])

def to_s_t(i):
    if(i<100):
        return(to_s_h(i))
    return(ONES[i/100] + "hundred" + to_s_h(i%100))

def to_s_m(i):
    if(i<1000):
        return(to_s_t(i))
    return(to_s_t(i/1000) + "thousand" + to_s_t(i%1000))

def to_s_b(i):
    if(i<1000000):
        return(to_s_m(i))
    return(to_s_m(i/1000000) + "million" + to_s_m(i%1000000))
```

It works by precomputing the length of the numbers from 1..999 and 1..999999 so that it can skip entire subtrees unless it knows that the answer lies somewhere within them.

edited Jan 16 '13 at 16:57

community wiki
2 revs
Electric Wig

That's insanely quick - finished in 6ms on my fairly run of the mill PC. Kudos, sir. – Dominic Rodger Jan 15 '13 at 17:18

Thanks - I'd missed the part where it's supposed to compute the sum of the numbers to that point. Fixed now :) – Electric Wig Jan 16 '13 at 16:58

add a comment

What I did: 1) Iterate through 1 - 999 and generate the words for each of these. As we generate: 2) Create 3 data structures where each node has a pointer to children and each node has a character value, and a pointer to Siblings. (A binary tree, in fact, but we don't want to think of it that way necessarily - for me it's easier to conceptualise as a list of siblings with lists of children hanging off, but if you think about it {draw a pic} you'll realise it is in fact a Binary Tree). These 3 data structures are created cocurrently as follows: a) first one with the word as generated (ie 1-999 sorted alphabetically) b) all the values in the first + all the values with 'thousand' appended (ie 1-999 and 1,000 - 999,000 (step 1000) (1998 values in total) c) all the values in B + all the values in a with million appended (2997 values in total) 3) For every leaf node in(b) add a Child as (a). For every leaf node in (c) add a child as (b). 4) Traverse the tree, counting how many characters we pass and stopping at 51 Billion.

NOTE: This doesn't sum the values (I didn't read that bit when I originally did it), and runs in just over 3 minutes (about 192 secs usually, using c++). NOTE 2: (in case it isn't obvious) there are only 5,994 values stored, but they are stored in such a way that there are a billion paths through the tree

I did this about a year or two ago when I stumbled accross it, and have since realised there are many optimisations (the most time consuming bit is traversing the tree - by a LONG WAY). There are a few optimisations that I think would significantly improve this approach, but I could never be bothered taking it further, other than to optimise redundant nodes in the tree slightly, so they stored strings rather than characters

I have seen people claim on line that they've solved it in less than 5 seconds....

edited Jul 22 '11 at 0:24

community wiki
2 revs
Johnny

I have java code that solves it in about 8 seconds. But it took lots longer to write the code. – Don Roby Jul 22 '11 at 1:28

I'm basically thinking that it's possible to work out the number of characters in each tree/subtree. So when we add trees on the leaves, we just add the number of characters and the sum below that point (rather than the characters themselves). Then we only need to generate the actual branch we're on to work out the exact answer (if that makes sense). Because the traversal is the real killer, it's the obvious step to minimise – Johnny Jul 22 '11 at 1:54

Yes, that's basically the approach. See my just posted answer. Also +1! – Don Roby Jul 22 '11 at 2:13

add a comment

weird but fun idea.

build a sparse list of the lengths of the number from 0 to 9, then 10-90 by tens, then 100, 1000, etc etc, to billion, indexes are the value of the integer part who's lenght is stored.

write a function to calculate the number as a string length using the table. (breaking the number into it's parts, and looking up the length of the aprts, never actally creating a string.)

then you're only doing math as you traverse the numbers, calculating the length from the table afterward summing for your sum.

with the sum, and the value of the final integer, figure out the integer that's being spelled, and volia, you're done.

answered Sep 29 '09 at 22:25

community wiki
Eric Brown - Cal

add a comment

Yes, me again, but a completely different approach.

Simply, rather than storing the "onethousandeleventyseven" words, you write the sort to use that when comparing.

Crude java POC:

```
public class BillionsBillions implements Comparator {
    public int compare(Object a, Object b) {
        String s1 = (String)a; // "1234";
        String s2 = (String)b; // "1235";

        int n1 = Integer.valueOf(s1);
        int n2 = Integer.valueOf(s2);

        String w1 = numberToWords(n1);
        String w2 = numberToWords(n2);

        return w1.compare(w2);
    }

    public static void main(String args[]) {
        long numbers[] = new long[1000000000]; // Bring your 64 bit JVMs

        for(int i = 0; i < 1000000000; i++) {
            numbers[i] = i;
        }

        Arrays.sort(numbers, 0, numbers.length, new BillionsBillions());

        long total = 0;

        for(int i : numbers) {
            String l = numberToWords(i);
            long offset = total + l - 51000000000;

            if (offset >= 0) {
                String c = l.substring(l - offset, l - offset + 1);
                System.out.println(c);
                break;
            }
         }
    }
```

"numberToWords" is left as an exercise for the reader.

answered Sep 29 '09 at 22:39 community wiki
Will Hartung

add a comment

Do you need to save the entire string in memory?

If not, just save how many characters you've appended so far. For each iteration, you check the length the next number's textual representation. If it exceeds the nth letter you are looking for, the letter must be in that string, so extract it by it's index, print it, and stop execution. Otherwise, add the string length to the character count and move to the next number.

answered Sep 29 '09 at 21:53 community wiki
Phil M

1 How would that work? You'd have to generate all the strings before you sort them, right? So you'd still have significantly greater than 51 billion characters in memory, which is an awful lot of data. – Dominic Rodger Sep 29 '09 at 21:55

The problem is coming up with the sorted list in the first place... – DJ. Sep 29 '09 at 21:57

1 This is what I was thinking, until I saw the "sorted alphabetically" part – Neil N Sep 29 '09 at 21:59

You don't need to sort the list. I think the key to this is thinking how you would programmatically work out just the first or last string. Then how to work out the second, etc etc, and keep going until you get to 51 billion or whatever. – DisgruntledGoat Sep 29 '09 at 23:14

I totally missed the sorting part of the question! :(– Phil M Sep 30 '09 at 0:56

show **1** more comment

All the strings are going to start with either one, ten, two, **twenty**, three, thirty, four, etc so I'd start with figuring out how many are in each of the buckets. Then you should at least know which bucket you need to look closer at.

Then I'd look at **subdividing the buckets** further based on the possible prefixes. For example, within ninehundred, you are going to have all the same buckets that you had to start off with, just for numbers starting with 900.

answered Sep 29 '09 at 22:03 community wiki
Alan Jackson

add a comment

The question is about efficient data storage not string manipulation. Create an enum to represent the words. the words should appear in sorted order so that when it comes time to sort it is a simplish compare. Now generate the list and sort. use the fact that you know how long each word is in conjunction with the enum to add up to the character you need.

answered Sep 29 '09 at 22:06 community wiki
stonemetal

add a comment

Code wins...

```
#!/bin/bash
n=0
while [ $n -lt 1000000000 ]; do
   number -l $n | sed -e 's/[^a-z]//g'
   let n=n+1
done | sort > /tmp/bignumbers
awk '
BEGIN {
    total = 0;
}
{
    l = length($0);
    offset = total + l - 51000000000;
    print total " " offset
    if (offset >= 0) {
        c = substr($0, l - offset, 1);
        print c;
        exit;
    }
    total = total + l;
}' /tmp/bignumbers
```

Tested for a much smaller range ;-). Requires a LOT of diskspace, a compressed filesystem would be, umm, valuable, but not so much memory.

Sort has options to compress work files as well, and you could toss in gzip to directly compress data.

Not the zippiest solution.

But it does work.

answered Sep 29 '09 at 22:24 community wiki
Will Hartung

5 It would be interesting to see if someone could come up with a better algorithm while waiting for this to finish. – Mark Ransom Sep 29 '09 at 22:30

Mine is still running...sort is 25MB heap so far, and has two, 16ishMB work files, but it will destroy my drive if it actually ever finishes, I can't hold the sorted file on this drive....I have no hope that it will get that for however. :-) – Will Hartung Sep 29 '09 at 23:04

1 Mine is finished. How about yours? – Mark Ransom Oct 1 '09 at 14:44

add a comment

Honestly I would let an RDBMS like SQL Server or Oracle do the work for me.

1. Insert the billion strings into an indexed table.
2. Compute a string length column.
3. Start pulling off the top X records at a time with a SUM, until I get to 51 billion.

Might beat up the server for a while as it would need to do a lot of Disk IO, but overall I think I could find an answer faster than someone who would write a program to do it.

Sometimes just getting it done is what the client really wants, and could care less what fancy design pattern or data structure you used.

edited Sep 29 '09 at 22:25

community wiki
2 revs
Neil N

1 Where can I sign up to work for the client that asks for problems like this to be solved? – Fragsworth Sep 29 '09 at 22:38

Well, it's better than the orginal approach, but that's still an awful lot of diskspace, even by today's standards and it'll take forever to run. – RBarryYoung Sep 30 '09 at 16:56

@Fragsworth - ITA Software in Boston, MA. They've removed this problem from the list, but I solved it when I applied there in 2008. – Don Roby Jul 22 '11 at 1:32

add a comment

figure out lengths for 1-999 and include length for 0 as 0.

so now you have an array for 0-999 namely uint32 sizes999[1000]; (not going to get into the details of generating this) also need an array of thousand last letters last_letters[1000] (again not going to get into the details of generating this as it is even easier even hundreds d even tens y except 10 which is n others cycle though last of on **e** through nin **e** zero is irrelavant)

```
uint32 sizes999[1000];

uint64 totallen = 0;
strlen_million = strlen("million");
strlen_thousand = strlen("thousand");
for (uint32 i = 0; i<1000;++i){
    for (uint32 j = 0; j<1000;++j){
        for (uint32 j = 0; j<1000;++j){
           total_len += sizes999[i]+strlen_million +
                        sizes999[j]+strlen_thousand +
                        sizes999[k];
           if totallen == 51000000000 goto done;
           ASSERT(totallen <51000000000);//he claimed  51000000000 was not 
intermediate
        }
    }
}
done:
```

//now use i j k to get last letter by using last_letters999

//think of i,j,k as digits base 1000

//if k = 0 & j ==0 then the letter is n millio**n**

//if only k = 0 then the letter is d thousan**d**

//other wise use the array of last_letters since

//the units digit base 1000, that is k, is not zero

//for the sum of the numbers i,j,k are the digits of the number base 1000 so

```
n = i*1000000 + j*1000 + k;
```

//represent the number and use

```
sum = n*(n+1)/2;
```

if you need to do it for number other than 51000000000 then also calculate sums_sizes999 and use that in the natural way.

total memory: 0(1000);

total time: 0(n) where n is the number

answered Sep 29 '09 at 22:57

community wiki
pgast

This doesn't sort the number names in alphabetical order. – Will Hartung Sep 29 '09 at 23:04

your are absolutely correct – pgast Sep 30 '09 at 3:12

add a comment

This is what I'd do:

1. Create an array of 2,997 strings: "one" through "ninehundredninetynine", "onethousand" through "ninehundredninetyninethousand", and "onemillion" through "ninehundredninetyninemillion".
2. Store the following about each string: length (this can be calculated of course), the integer value represented by the string, and some enum to signify whether it's "ones", "thousands", or "millions".
3. Sort the 2,997 strings alphabetically.

With this array created, it's straightforward to find all 999,999,999 strings in order alphabetically based on the following observations:

1. Nothing can follow a "ones" string
2. Either nothing, or a "ones" string, can follow a "thousands" string
3. Either nothing, a "ones" string, a "thousands" string, or a "thousands" string then a "ones" string, can follow a "millions" string.

Constructing the words basically involves creating one- to three-letter "words" based on these 2,997 tokens, making sure that the order of the tokens makes a valid number according to the rules above. Given a particular "word", the next "word" is found like this:

1. Lengthen the "word" by adding the token first alphabetically, if possible.
2. If this can't be done, advance the rightmost token to the next one alphabetically, if possible.
3. If this too is not possible, then remove the rightmost token, and advance the second-rightmost token to the next one alphabetically, if possible.
4. If this too is not possible, you're done.

At each step you can calculate the total length of the string and the sum of the numbers by just keeping two running totals.

edited Sep 29 '09 at 23:14

community wiki
2 revs
John

add a comment

It's important to note that there is a lot of overlapping and double counting if you iterate over all 100 billion possible numbers. It's important to realize that the number of strings that start with "eight" is the same number of numbers that start with "nin" or "seven" or "six" etc...

To me, this begs for a dynamic programming solution where the number of strings for tens, hundreds, thousands, etc are calculated and stored in some type of look up table. Ofcourse, there will be special cases for one vs eleven, two vs twelve, etc

I'll update this if I can get a quick running solution.

answered Sep 29 '09 at 23:24

community wiki
llamaoo7

add a comment

WRONG!!!!!!!!! I READ THE PROBLEM WRONG. I thought it meant "what's the last letter of the alphabetically last number"

what's wrong with:

```
public class Nums {

// if overflows happen, switch to an BigDecimal or something
// with arbitrary precision
public static void main(String[] args) {
    System.out.println("last letter: " + lastLetter(1L, 51000000L);
    System.out.println("sum: " + sum(1L, 51000000L);
}

static char lastLetter(long start, long end) {
   String last = toWord(start);
   for(long i = start; i < end; i++)
      String current = toWord(i);
      if(current.compareTo(last) > 1)
         last = current;

   return last.charAt(last.length()-1);
}

static String toWord(long num) {
   // should be relatively easy, but for now ...
   return "one";
}

static long sum(long first, long n) {
   return (n * first + n*n) / 2;
}
}
```

haven't actually tried this :/ LOL

answered Sep 29 '09 at 23:50 community wiki les2

add a comment

You have one billion numbers and 51 billion characters - there's a good chance that this is a trick question, as there are an average of 51 characters per number. Sum up the conversions of all the numbers and see if it adds up to 51 billion.

Edit: It adds up to 70,305,000,000 characters, so this is the wrong answer.

edited Sep 30 '09 at 13:02 community wiki 2 revs Mark Ransom

the problem doesnt state that you have 51 billion characters, it just asks what the 51 billionth character is. – aepurniet Sep 30 '09 at 3:08

And I'm stating that there's a chance the 51 billionth character is also the last character. Testing that possibility is a simpler problem than an all-out solution. – Mark Ransom Sep 30 '09 at 6:45

Huh - that was a closer call than I thought it might be - good effort! Out of interest, how did you compute the character counts? Using some kind of common-subsequence counting? – Dominic Rodger Sep 30 '09 at 13:37

I created the names with the code I posted to another answer. I tried two different computation methods to make sure they matched, one with common subsequences and one with brute force. – Mark Ransom Sep 30 '09 at 16:06

add a comment

I solved this in Java sometime in 2008 as part of an application to work at ITA Software.

The code is long, and it now being three years later, I look at it with a bit of horror... So I'm not going to post it.

But I'll post quotes from some notes that I included with the application.

> The problem with this puzzle is of course the size. The naïve approach would be to sort the list in word number order and then to iterate through the sorted list counting characters and summing. With a list of size 999,999,999 this would of course take a rather long time and the sort could likely not be done in memory.
>
> But there are natural patterns in the ordering which allow shortcuts.
>
> Immediately following any entry (say the number is X) ending in “million” will come 999,999 entries starting with the same text, representing all the numbers from X +1 to X + 10^6 -1.
>
> The sum of all these numbers can be computed by a classic formula (an “arithmetic series”), and the character count can be computed by a similarly simple formula based on the prefix (X above) and a once-computed character count for the numbers from 1 to 999,999. Both depend only on the “millions” part of the number at the base of the range. Thus if the character count for the entire range will keep the entire count below the search goal, the individual entries need not be traversed.
>
> Similar shortcuts apply for “thousand”, and indeed could be applied to “hundred” or “billion” though I didn’t bother with shortcuts at the hundreds level and the billions level is out of range for this problem.
>
> In order to apply these shortcuts, my code creates and sorts a list of 2997 objects representing the numbers:
>
> 1 to 999 stepping by 1 1000 to 999000 stepping by 1000 1000000 to 999000000 stepping by 1000000
>
> The code iterates through this list, accumulating sums and character counts, recursively creating, sorting and traversing similar but smaller lists as needed.
>
> Explicit counting and adding is only needed near the end.

I didn't get the job, but later used the code as a "code sample" for another job, which I did get.

The Java code using these techniques for skipping much of the explicit counting and adding runs in about 8 seconds.

answered Jul 22 '11 at 2:09 community wiki
Don Roby

add a comment

Not the answer you're looking for? Browse other questions tagged algorithm or ask your own question.