**Codeforces Compiler Tips**

SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Fri 5/10/2024 2:18 PM

If I ever feel restricted by the max-size of long, I could build my own rudimentary BigInt size by just tracking a vector of base-10 digits. Or could implement a base MAX\_INT bignum system, like in Python.



ReplyForward

SJ

Shen, James

Fri 5/10/2024 2:10 PM

(No message text)

SJ

Shen, James

Fri 5/10/2024 2:07 PM

(No message text)



You forwarded this message on Fri 5/10/2024 2:07 PM

You forwarded this message on Fri 5/10/2024 2:07 PM

SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Fri 5/10/2024 12:36 PM

####

When doing union find on edges (maybe after a topological sort), make sure to remember the case of both vertices on the edge already belong to sets, MAYBE THE SAME SET!!! (I have made mistake of remember case where they already belong to different sets, but forgot case of already belonging to the same set!)

###

When borderline TLE, and a bunch of helper function calls, see if can just make it in time if "inline" all the functions, just copy the function block right into the larger scope! Will make code super long and messy, last ditch resort, but keep it as a resource!

####

my union find codes tend to be very long, very fine logic with keys, erase and insert keys. Should abstract out some of the nitty gritty logic by making additional helper functions such as ensureKeyExists(unordered\_map<int, unordered\_set<int>>& hm, int key), if key doesn't exist, then insert {key, *empty set*} pair into hm. This helper function will assist in encapsulating behavior for composite maps

###

When borderline TLE, passes most test cases, if using huge 3D dynamic vectors, see if can find a way to reduce the dimensions. 1D faster than 2D, 2D faster than 3D vectors. e.g. if keeping track of a bunch of index spans, {{0, 4}, {6, 8}, {9, 12}} e.x, if the spans can be indexed as {0, 1, 2}, then a dimension can be saved, will run faster! Fastest of all is string, then raw array, then vector. A bunch of small vectors or maps is TLE or MLE demise!

If implementation has complex 3D vectors, try to simplify it down to fewer dimensions!

### make a list of large primes, know a few off the top of your head e.g. 10^9 + 7, 10^9 + 9, 2^31 - 1 (2147483647) to spam for Rabin-Karp key collision emergencies!

###

when Rabin-Karp or string hashing returns correct answer on small tests, but fails on large tests, logic seems correct, consider hashing collision false positive/false negatives! Use a double-hash technique. One method is to do the same hashing operations, but do everything twice: one with MOD1, second with different MOD2. Count the total number of matches. Must be 2 matches if the original strings represented by the two compared keys are equal. If originally not equal, still a tiny chance, 1/10^18 rather than 1/10^9 chance that falsely reports 2 matches. But if 1 or 0 matches, then must mean originally they are not equal. Or choose bigger mod, custom data type \_\_uint128 or something. Or we can do a true double-hash, %= MOD1, then \*= PRIME2, then %= MOD3? This would be an "atomic" operation made up of composite hashing, rather than checksum-type " "pseudo double hashing"

#######

LACK OF C++ UNDERSTANDING! BE CREATIVE, BUT DON'T MAKE DANGEROUS ASSUMPTIONS OR MESS WITH THINGS YOU DON'T UNDERSTAND! Thought that if(statement1) if(statement2) return false; was the same as if(statement1){ if(statement2) return false;}  sometimes, they behave differently! When there are additional else/else if statements immediately proceeding, the else/else if will pair with statement2 rather than statement1 if the curly braces are not included!

### when getting ready to submit, make sure to comment out all couts and debugging logic!

####### REFERENCES CANNOT BE REASSIGNED IN C++!!!!!!!

####### TRY TO ABSTRACT OUT AND REUSE AS MUCH CODE AS POSSIBLE, MAKE SURE YOU WRITE IT CORRECT THE FIRST TIME! Rather than copy pasting the same subroutine a bunch of times, when you modify one chunk and forget to modify the chunk in a different location, BAD THINGS HAPPEN!

####### WHEN HAVE VOID FUNCTION MEANT TO MUTATE, MAKE SURE YOU INCLUDE AMPERSAND WHEN YOU INTEND TO MODIFY IT IN-MEMORY DIRECTLY RATHER THAN MAKE A COPY!!!!!!! Single missing ampersand bug has caused me incorrect submissions!!!!!!!

### C++ gcd function is \_\_gcd(), lcm function is lcm(), no trailing double underscore

### be careful when using built-in C++ functions, such as \_\_gcd, does it support finding gcd of long long type?

### BE CAREFUL! Made the stupid mistake of using .rbegin() instead of .begin() for C++ map! Know what you are doing, 110% focus! Know when you want to iterate in forwards direction or backwards direction, don't get confused!

# When going extremely bruty, can try to cheese through the large test cases, instead of going for loop from 0 -> n, if we see that we are stuck on a test case, but pass the test case or get stuck on a different test case when we loop from n -> 0, opposite direction, we can try hybrid approach, going 0, n, 1, n - 1, 2, n - 2, ....... etc., this will "cheese" those cases that are designed to exhaust a brute force approach into TLE

#### if looking for vector optimization, can't think of algorithmic improvement, if not using dynamic arrays, try converting to raw C array implementation, static array size value reset using memset(ptr, val, bytes) is much faster than resetting vector to all zeros!

###

If there are unexpected bugs in algo w modular arithmetic, check for negative mods!

#######

Don't mistakenly use vector v.clear() when intention is to reset to all zeros! v.clear() erases all elements, makes it into empty vector, have to manually use for i loop to reset each index position to a 0. Have made this mistake multiple times, very irritating when the rest of the algo is correct, but this bug remains!

### for Boolean functions, do not name them isNotGood, returning true when it is not good. Very confusing, just name it as isGood, return true when is good, return false when not. Have made mistake of returning false when I mean it to be true, true when I mean it to be false!

####

NEVER COMPARE NEGATIVE signed int WITH UNSIGNED INT!!!!!!! Finnicky!!!!!!! compared .size() > -1, does not behave as you would expect!

  vector<int> v;

    if(v.size() > -1) cout << "yes";

    else cout << "no";

    cout << endl;

\*\*\* prints counterintuitively prints "no"

###

Frobenius number! Good technique for partitioning, integer splitting, number theory, grouping

####

C++ set, map operations:

lower\_bound(x), returns smallest key where key >= x

upper\_bound(x), returns smallest key where key > x

###

When TLE, check for while loops! More often than not, while loop rather than for loop is culprit when time complexity seems reasonable

####

If stuck in a lull, seemingly most test cases passed, try to think of edge cases that your logic does not cover, or your logic TLEs. Emergency scenario, cannot find better algorithmic time complexity, try to optimize what you have using heuristics, or use type optimizations (e.g. string rather than vector)

###

if C++ vector v.erase(v.begin() + i, v.end()) does not behave as expected, e.g. appends an extra zero in the vector, then the i variable is > v.size()

#### Must be very meticulous with types in C++! Type OD when dealing with .size(), .length() attributes, these return size\_t type variable, underflow potential since unsigned. Also make sure to have types match in built-in functions, e.g. max(), min(), pow(), should have int and int, not long long and unsigned int or something

####

Pay attention to if you declared empty vector, or fully initialized it with all 0's already! Do not make the mistake of absent-mindedly push\_back() when fully initialized already!

#### monotonic stack algorithm, can find and update in O(log n) time, binary search (since monotonic is either increasing or decreasing!!!), we can pop out everything like normal, and replace afterwards, but we can optimize out the costly popping stage, just find the index, update at that index, store the index in auxiliary index variable, and forget everything to the right!!!!!!! Updating index is like deleting everything to the right, without actually spending the time to do so! Application: Longest Increasing Subsequence LIS problem, Knuth Dancing Links/Dancing Cells

### bisection search (cartesian coordinate points, integer equation solutions) with negative values may mess things up! if lo = -1, hi = 0, mid = (lo + hi) / 2 calculated to be 0, since (-1) / 2 = 0 in C++, one would expect to round down, expect -1, but evaluates to 0 instead!!!!!!! very tricky, keep in mind! In Python, (-1) // 2 evaluates to -1, as expected, rounds down.

# for binary search, explore updating lo = mid + 1, rather than lo = mid, these small details might make a difference in a complicated variation of binary search

# sorting intervals by start, end events, {start, 0} {end, 1} is a very elegant way, well-formed valid parenthesis, "depth int stack"

### When doing comparison pattern match problems, abbreviations, "a3e" matching "apple" and "12eee" matching "abcdefghijkleee", can either do it the more elegant way, skipping indices by parsing out numbers and incrementing index simultaneously, or can do multi-stage, first generate a matching template by actually building a new string, "a---e" and "------------eee", KISS, Keep It Simple, Stupid! whatever does the job in tight contest time contraints.

### DO NOT CAST (int) on C++ pow function, since pow returns a double. Use arbitrary fixed precision bignums, and square and multiply fast O(log n) exponentiation algo for exact! floating point numbers are finnicky! never compare 2 floating point numbers for exactness, use epsilons!

### if there is a DP or prefix array solution, see if there is a greedy solution, rather than trying all index locations, and speeding up with O(1) prefix pre-compute lookup, see if we can just find the best index using a greedy strategy!

### in C++, do not do stuff like:   if(a == b == c), DOES NOT WORK AS HUMAN INTERPRETS IT!!!!!!! must do    if(a == b && a == c), sneaky logic error due to careless language oversights

####

for problems where brute force is extremely computationally taxing, even for small inputs, e.g. dealing with 100-digit numbers, or large integers in string form, involving permutations and multisets, combinatorics with digits, play around with the smallest test cases, try to find an inductive pattern, DP, or if the solutions to small cases can be "built on" using a clever constructive observation, "reusing" the small case solutions by padding zeros into the larger solutions, etc. TRY THINGS, DON'T SIT AROUND! Abusing the same pattern for all inputs, be quick to detect reusable opportunities! (constructive algos when trying to find a permutation that satisfies problem condition, or generate a set of numbers sharing same digit multiset, check if changing the location of padded zeros helps!)

####

BELOW INCOMPLETE -

beautiful tree visualization strategies: think of tree leaves as the beaches of the island, further away a node is from a leaf, the more inland it is.... post-midnight thoughts in Honors Village of tree visualization. Some nodes are junctions, places where "branching" occurs, other nodes are just a part of a single branch, comparable to a linked list node (only a "prev" and "next," no branching)

cactus graphs:

sqrt decomp/segtree on a linear input (array or linked list):



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 1/16/2024 3:15 AM

///////

ONE OF THE MOST SLEEPER, DIFFICULT, HEISENBUGS, HARD-TO-CATCH logic bugs is converting integers to string forms, for purposes of creating a key to hash in a hashmap. Used i, j of topleft and bottomright cells of submatrices as keys to store information about each submatrix in a hashtable, thought I was being clever when I abused Python's str() and int() elegant conversion mechanisms, but instead overlooked a pitfall. Input matrix dimension constraints <= 50, used str(si) + str(sj) + str(fi) + str(fj), (si, sj) topleft cell, (fi, fj) bottomright cell, as key, my intention was to have each coordinate take up 2 places in the string. What I missed was single-digit integers are converted to single-digit strings, instead of 00094901, converts to 09491, instead of 01022304, converts to 12234. but 12234 is not unique, it can correspond to 0102234 or 12020304, so there is uniqueness hashing issues!!!!!!! ALWAYS BE ATTENTIVE, NATURALLY SKEPTICAL AND PARANOID WHEN CONVERTING BETWEEN INTEGER AND STRING! Keep in mind digit count. LEADING ZEROS ARE REMOVED, EVEN WHEN YOU WANT THEM!!!!!!!

/////// when you misread the constraints, and hardcode in based on the wrong constraints, MAKE SURE TO CHANGE YOUR CODE!!!!!!!

/////// DO NOT cin >> a, b, c;

/////// dumbest mistake, should ALWAYS BE cin >> a >> b >> c; !!!!!!!

///////

division by 2 is bitwise right shift by 1!

///////

double sqrt decomposition, double jump array? Would make things even faster? getting to another level of efficiency, getting closer to logarithmic seg tree?

/////// can make a prefix array, suffix array of a sqrt jump array, multilayered data structures! 2D segtrees, binary tree within segtree, multidimensional data structures with different structure in each layer!

/////// When using direct indexing by convenience, such as index [i] to check data stored for the number having value i, make sure to "pad" the beginning of vector before the push\_back() operations. i.e. when working with values >= 1, we want to initialize vector with {0} placeholder for index 0, before push\_back() for value 1, or else we would be 1 index off when accessing.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 12/30/2023 5:31 PM

\*\*\*\*\*\*\*

for huge system tests involving big numbers, think of worst case. Possible to TLE even if algorithm is correct, since factoring is O(sqrt(n)) time complexity. If numbers are on order of magnitude of 10^9, factoring single number will require ~10^4 to ~10^5 operations, so if 10^4 test cases, will surely TLE. So when factoring huge numbers, memoize them, don't want to repeatedly factor the same exact huge number, just search it up if hard work done before! Just like log or trig tables in old days! Hard work already done, just open up the book, lookup! Also, if we only need the smallest factors, just break out of the loop early if small factor is already obtained. These are practical optimizations to help reduce runtime, since even if pass small pretests, may fail large system tests, and other participants can come up with some diabolical test cases to make it TLE!

\*\*\*\*

Some problems do a precomputation before all the test cases, similar to test cases just being queries, and the brunt of the work done, e.g. DP precalculation of Alice and Bob optimal play, up to 10^5 size, guaranteed permissible runtime, then read in inputs and calculate each query.

### to prevent getting hacked when using CodeForces C++ unordered\_map or Python dict(), if want to use hashtable, should just use C++ map instead, since backed by a Red Black balanced binary search tree, doesn't have the issue of hashtable key collisions. Have to sacrifice some runtime, since O(log n) operations rather than O(1). Or implement own middle square method using hashing.

### prefix array data structure. Also think about building prefix tree data structure to store size of subtree efficiently!

### log of 0 is UNDEFINED!!!!!!! Sneaky runtime error! Make sure to never take logs of non-positives!

### watch out for edge cases! Make sure you are not going index out of bounds. For example, I made prefix array for even and odd index sums, but input has case where only input vector only size 1, so odd index sum doesn't have an initialization, seg fault when I try to do prefixOdd[1] = v[1], when the 1 index doesn't exist, only has size 1, max index is 0.

### vector index OutOfBounds when trying to check v[v.size()] !!!!!!! if you want to check v.size(), DFS until the entire length of the vector is traversed, substring partition, then should allocate the "searched" vector lookup table to have size N + 1, so that it has an index for the length of the string.

-Number of children of a tree node is NOT THE NUMBER OF DIRECT ADJACENT CHILDREN (all adjacent nodes, minus 1 unless the node is the root of the tree), BUT THE SUM OF THE NUMBER

-Using O(1) memory in primitive variables types is faster than storing O(n) memory by building vector!

-In Linux, can get # physical cores by std::thread::hardware\_concurrency(); or sysconf(\_SC\_NPROCESSORS\_CONF); sysconf(\_SC\_NPROCESSORS\_ONLN);, those might be logical cores though (due to hyperthreading in a single core)



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 12/21/2023 11:06 PM

C++ custom sort comparator function, must comply with strict weak ordering criteria! If two elements equal, FUNCTION MUST RETURN FALSE, so return e1 < e2 is correct, not return e1 < e2. (the same cannot be said for your own sort implementation, this is regarding the built-in sort(begin, end, comparator) paradigm in stl). I've had MLE, TLE, wrong answer from this!

-If the function called in a thread passes in argument by reference, then must use ref(argument) syntax in thread instantiation!

thread th1(funcname, x, y);

-if funcname is a member function in a class, should be a static member function, and should pass in address of function, i.e. &funcname,  or just funcname itself (itself is an address!). But if not a member function (just a free function), does not have to be static.

### Be cognizant of what your code is doing! E.g. if have a 1000-element list of 10-character English names as strings, bubble sort, and trying to debug, isn't good to concatenate all elements as a huge string, then print out after each iteration! Up to 1 million iterations, print out a 10^4 char-long string each iteration, no bueno! Trying it on small cases is fine, but for big cases, TLE, or even worse, MLE and program terminated/crashes!

**From:** Shen, James <jamesshen@ufl.edu>  
**Sent:** Thursday, December 21, 2023 4:20 AM  
**To:** Shen, James <jamesshen@ufl.edu>  
**Subject:** Re: Codeforces Compiler Tips

many problems can be mentally visualized as a collection of bars in a bar graph! Values, cost, ... minimizing cost to "flatten", plateau out the bars so that all have equal height. Intuitive to think about some sort of average, that is somewhat at a middle, centralized distance from all bar heights. BUT WHEN THINK OF MEAN, SHOULD ALSO CONSIDER THE MEDIAN!!!!!!! Median is extremely important and elegant as well! My idea was a 3-way binary search to find the centralized height such that sum of abs distances of bar heights to centralized height is minimized. This is a monotonic function (nonincreasing or nondecreasing) on both sides of the minimal solution. If solution is nonintegral, then integral best solutions are either floor or ceiling, can be both, a tie! But impossible to have more than 2-way tie, so 3-way binary search on a = [mid], b = [mid + 1], c = [mid + 2], comparing cases: a == b or b == c, a < b < c, a > b and b < c, a > b > c. If there is a question where a 3-way tie optimum is possible on adjacent positions, then should try a, b, c, d = [mid], [mid + 1], [mid + 2], [mid + 3], more casework! a < b == c < d, a < b < c < d, a > b == c == d, a > b == c < d, more crazy stuff! Median is a more elegant solution if shown to be correct!

-more about elegance: try to visualize, picture in your mind. Bar graph/histogram of values in an array. ICPC boat problem, envision a bird's eye view of ships travelling in cargo lane, different directions. Same speed or different speed? Group all the ones going same direction together, then do a "phase shift", like sine or cosine wave phase shift sliding across x-axis, to convert it to the same exact moment in time, then merge arrays. ELEGANCE! "phase shift" technique is the most elegant technique I know!

- Another elegant idea: Think about partitioning array of relationships as connected components, when each node can get to any other node in the connected component, but impossible to reach outside of it. e.g., can swap 2 numbers if they have common factor > 1, i.e. GCD >= 2. This would constitute connected components! e.g. [2, 4, 6, 3, 5, 35] would be [2, 4, 6, 3], [5, 35] as connected components. If can move the numbers within these, all that is required is adjacent "bubbling" a la bubble sort, bubbling can reach ANY permutation, pretty sure furthest permutation in # of moves is O(n^2) with n-length permutation, optimal moves being made. (BFS is a brute force way of finding. Is there a faster way)?



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 12/21/2023 4:20 AM

many problems can be mentally visualized as a collection of bars in a bar graph! Values, cost, ... minimizing cost to "flatten", plateau out the bars so that all have equal height. Intuitive to think about some sort of average, that is somewhat at a middle, centralized distance from all bar heights. BUT WHEN THINK OF MEAN, SHOULD ALSO CONSIDER THE MEDIAN!!!!!!! Median is extremely important and elegant as well! My idea was a 3-way binary search to find the centralized height such that sum of abs distances of bar heights to centralized height is minimized. This is a monotonic function (nonincreasing or nondecreasing) on both sides of the minimal solution. If solution is nonintegral, then integral best solutions are either floor or ceiling, can be both, a tie! But impossible to have more than 2-way tie, so 3-way binary search on a = [mid], b = [mid + 1], c = [mid + 2], comparing cases: a == b or b == c, a < b < c, a > b and b < c, a > b > c. If there is a question where a 3-way tie optimum is possible on adjacent positions, then should try a, b, c, d = [mid], [mid + 1], [mid + 2], [mid + 3], more casework! a < b == c < d, a < b < c < d, a > b == c == d, a > b == c < d, more crazy stuff! Median is a more elegant solution if shown to be correct!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 12/21/2023 4:20 AM

-in text files, beware of whitespace, tabs, newlines, and carriage returns!

* sometimes, just separated by \n, othertimes, \r\n, when looking at mama's input files for FDM project, the file was separated by commas, and each line by \r\n
* if \r\n, and read in by newlines, getline(cin, s) default delimiting char is '\n', so the carriage return will still be parsed into the argument string s. BE EXTREMELY CAREFUL! CARRIAGE RETURN DURING cout PRINTING WILL MAKE IT SEEM LIKE THE cout WAS CORRUPTED, BUT ACTUALLY THE OUTPUT LOCATION JUST SHIFTED TO THE BEGINNING OF THE LINE, AND OVERWROTE THE PREVIOUS CHARACTERS AT THOSE LOCATIONS! Extremely tricky!
* Whenever see input separated by lines, be sure to figure out if \r\n, or \n !!!!!!!

Java JUnit testing, each unit test is done in a premediated order. The previous code results from the previous tests are maintained through the current test, if I want fresh start for a test, clean slate, I have to do the clearing and resetting logic in the setup() function, with the @before each decorator so that it clears it before each unit test, similar to Slang enhanced test suite ability (test config files, Zebra compute farm, unit tests, master test scripts).

Make sure to initialize variables in Java! Unlike in C++, where I can do stuff like vector<int> myVec;   in Java have to ALWAYS initialize using new, List<Integer> myList = new ArrayList<>();  or else have null pointer problems in Java. In C++, that just creates an empty vector! Different languages, different intricacies.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 12/17/2023 1:43 AM

using #include <bits/stdc++.h> increases compilation time by a significant amount.

when doing Dijkstra's don't make the mistake of assigning node = pq.top()[0]; d = pq.top()[1];. should be other way around, since priority queue inner vector elements should be [d, node] rather than [node, d], ordered by shortest distance so far from starting!

///////EXTREMELY HELPFUL!!!!!!!:::::::#######@@@@@@@///////

\*\*\*\*\*\*\*Think about algebraic manipulation, especially when doing arrays. Calculate difference array i.e. diff[i] = arr[i + 1] - arr[i].

Think about mathy property stuff like b - a = diff,

b - diff = a. Always consider if we are allowed to repeat choosing indices.

e.g. b - a = diff, again choose b - a = diff, then diff - diff = 0 if can repeat choosing same b and a in separate operations

Also whenever arrays, think about prefix/suffix arrays, RLE, if (i, j), rearrange equation so that all i's on same side, all j's on same side!

# Remember to parse in all input before early continue, or else will fuck up the input! Input must be read in



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Wed 12/6/2023 3:44 AM

meet-in-the-middle for subset sum brute force improvement, instead of exponential to the N, it is exponential to the N/2, which is better.

For my builder arrays brute force method, can improve the subset sum calculation runtime from O(n \* exponential) to O(1 \* exponential), constant amortized when using subset sum from previous iteration to update current subset sum, rather than throwing out previous work, and starting new calculation from scratch. Standard binary representation amortized 2 constant operations to update the 1111..1111 to 10000..0000, gray code better with just 1 constant operation, but more difficult to implement.

Try DP for special subset sum problems, standard subset sum can be done in pseudo-polynomial time using DP (back-to-front repeated 1-D DP (memory optimization), or traditional 2D DP), but choice of positive and negative sum has a nice algebraic manipulation. Call x our chosen positive subset sum, and -y our chosen negative subset sum. x + y = totalsum, we want to find cases where x - y = target. y = totalsum - x. x - y = 2x - totalsum. We want to find 2x - totalsum = target, can rearrange to just have a single subset variable x on one side, and (target + totalsum) / 2 on right side. Similar in nature to the i and j pair of substring having equal # consonants and vowels, can use prefix sum and hashmap counting cumulative vowels/consonants at index, rearrange prefix\_c[j] - prefix\_c[i] == prefix\_v[j] - prefix\_v[i] into all j's on same side, and all i's on same side, so can go from O(n^2) to O(n)



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Wed 12/6/2023 3:36 AM

custom comparator for 3rd parameter in std::sort call must be STRICT WEAK ORDERING!!!!!!! Or else will result in segfault sometimes. Will pass some test cases without problem, but then mysteriously segfault for other test cases! Make sure the custom comparator logic is foolproof.

C++ v.clear() function for vectors destroys all elements in the vector (doesn't necessarily change values at the prior memory locations, just empties it to size 0. Vector doesn't get relocated I think, old data that was there before, should still be there. If before there was 001010101100 10101010 0101010010 in old bit locations, should still be there, but the memory should be "freed", able to be allocated for future variables.

Must manually use for loop to reset all elements to 0 in the vector, don't mistakenly use .clear() for that!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 11/21/2023 4:27 PM

Python woes:

variable name typos can be overlooked.

In following code, the *helloword*variable name *typo* is not caught!

helloworld = 3

if helloworld == 3:

    print("typo not caught")

else:

    if helloword == "hello":

        print("zzz")

Well, maybe not that big of a deal since that block of code is not executed, during the run. Seems like the interpreter does not do a complete scan of every branch of logic, it just runs what it needs to, doesn't check semantics for everything. But when test cases do not comprehensively test each conditional branch, if large code cyclomatic complexity, and there are variable typos, upon deployment, issues involving typo bugs may manifest!

In MySQL, when comparing with NULL, use IS operator, rather than = operator,

similar to Python is None, rather than == None

**std::**[**unordered\_map**](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Fcplusplus.com%2Freference%2Funordered_map%2Funordered_map%2F&data=05%7C01%7Cjamesshen%40ufl.edu%7C181f130f27bf49ed2df508dbf636610e%7C0d4da0f84a314d76ace60a62331e1b84%7C0%7C0%7C638374485639561105%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=9517b5bjV8nUKhPVcAfece7njtDG4X7A0kksaY5WXoY%3D&reserved=0)**::insert**

*Inserts new elements in the*[*unordered\_map*](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Fcplusplus.com%2Funordered_map&data=05%7C01%7Cjamesshen%40ufl.edu%7C181f130f27bf49ed2df508dbf636610e%7C0d4da0f84a314d76ace60a62331e1b84%7C0%7C0%7C638374485639561105%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=Cp75lKT07Nn5mOLsttdWkAjVqY1zX%2FRLT0%2BVIzSYnrI%3D&reserved=0)*.*

*Each element is inserted ​\*\*\*\*****only if its key is not equivalent to the key of any other element already in the container (keys in an***[***unordered\_map***](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Fcplusplus.com%2Funordered_map&data=05%7C01%7Cjamesshen%40ufl.edu%7C181f130f27bf49ed2df508dbf636610e%7C0d4da0f84a314d76ace60a62331e1b84%7C0%7C0%7C638374485639561105%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=Cp75lKT07Nn5mOLsttdWkAjVqY1zX%2FRLT0%2BVIzSYnrI%3D&reserved=0)***are unique).****\*\*\*\**

be cognizant of what the type() of the input is!!!!!!! I sometimes make type errors such as check if x == 0 rather than if x == "0", the input is "01010111" but I am checking if == 0, if == 1

don't forget to comment out your print statements before submitting

when question does not say to MOD divide 1000000007, and you use huge numbers in your implementation, make sure to MOD on your own without reminders, or else will have overflow runtime error, catch you by surprise!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 10/21/2023 3:15 PM

way to think about Pascal's triangle row DP: connection between DP and combinatorics n choose x, nCx, is that you can go from n-1Cx-1 OR n-1Cx to nCx (the most recently considered item, we can choose to either select or not select). So number of ways to choose x from n, is number of ways to choose x-1 from n-1 (and then select current item) PLUS number of ways to choose x from n - 1 (and then NOT SELECT current item).

Lots of overhead for memory allocation:

Time taken to...

allocate 10^6 int vectors of size 1: ~0.2 sec

allocate 1 int vector of size 10^6: ~0.01 sec

allocate 10^6 strings of size 1: ~0.01 sec

allocate 1 string of size 10^6: ~0.001 sec



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 10/15/2023 1:02 AM

LeetCode contest problem: given sorted input array nums, what is the minimum size of the array after any number of the following operation is done: Choose i, j such that i < j and nums[i] < nums[j], delete both elements, and re-index nums. What is the smallest size that can be achieved after any number of operations? Greedy: take top 2 max frequency elements, delete them, think of it as bar graph, we want to balance the heights of the bars! We do this using a heap!

Whenever intermediates can overflow int, make the variable a long long (even when the final answer is within int, since my algo could do a final /= 2 division, meaning before the division, could be larger than INT\_MAX).

For computer binary number problems, keep in mind that magnitude of INT\_MIN is 1 greater than the magnitude of INT\_MAX!!!!!!!

TIL: using 1D vector, [a \* i + j], is faster than using 2D vector [i][j], and using string of chars is even faster than using 1D vector, especially if we are initializing a bunch of vectors. Initializing vector<int> v (100, 0); takes 10x longer than initializing string s = string(100, '0'); so if we don't need to store very large ints (if all we need is char-size int, < 256, then string of chars is very efficient)

LeetCode Combination Sum IV, when doing huge DP problems with huge intermediates, even if the answer to the target value is tiny, value 1, the intermediate values can get HUGE, surpass int, unsigned int, even surpass long long! unsigned long long allowed all values to be surpassed. So keep in mind that the possible size of intermediates can be HUGE, so either check if I can mod intermediate values, or if I should build my own bignum class in C++ (using base 10 vector arithmetic by digits, easy to implement, albeit slow)

If built-in C++ hash is slow, see if I can build a faster one using simple modular division, take matters into my own hands when I TLE using hashing, but time complexity seems sufficient! C++ hashing operations are 100x slower than vector lookup, see if I can avoid modular division! If using the same set of numbers, and you know for sure, then sort, them, order them!



SJ

Shen, James











A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 10/15/2023 12:59 AM

\*\*\*\*\*\*\* Can make a trie like a traditional tree, with vector of children, but could also make the children storage a hashmap. key is char c, value is pointer pointing to next node (has c as its character)

In LeetCode, when using pointers, making your own classes, always make dynamically allocated memory, not statically allocated on the stack, must use the new operator keyword to allocate a pointer of an object on the heap. When doing stuff like ListNode\* node = head; that I always do, the copy constructor must be doing deep copy and allocation on the heap dynamically allocated behind the scenes. So when I am not using copy constructor, but rather constructor, I should use new operator, or else will be created on the stack

\*\*\*\*\*\*\* SOME LINKED LIST PROBLEMS DO NOT HAVE DEFAULT empty constructor, that does not set ->val of node, only new Node(0), must initialize with value. Always use new operator!

can use for(int val : hs) when iterating through an unordered\_set<int>, can use for(pair<int, string> : hm) when iterating through an unordered\_map<int, string>, iterating using for() foreach loop takes approximately the same time as iterating through using unordered\_map<int, string>::iterator it; for(it = hm.begin().....)

When doing quant trading mental math practice, BE EXTREMELY ATTENTIVE, I frequently forget to include the negative sign, remember to always include it if the first operand is smaller than the second operand for a subtraction problem! Also if negative number multiplied/divided by positive number! Remember to carry correctly



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 10/15/2023 12:59 AM

// for codeforces contest problems, remember to READ IN THE #testcases !!!!!!!

// I often declare a variable T for total testcases, and forget to cin >> T;

// some kind of sorting, topo-sort-esque, then DFS, is a good greedy alternative way to doing DP approach. LeetCode Longest String Chain can be done via DP, via adjacency graph construction, sorting words by length of word, then by DFS.

// When doing iterative while loop DFS, could be multiple connected components, each connected component must start with depth = 0, then when push new root into stack, depth++, now depth = 1, if I make depth as an outer variable, using same depth variable for all connected components (rather than in same scope as a single connected component DFS), then I must remember to initialize depth = 0, then ++ when push new root. Made the mistake of initializing depth = 1, works correctly for the connected component of the first DFS, then depth gets decremented to 0 when the root of 1st DFS is popped from stack, now depth = 0, subsequent connected components do not have a depth++ when pushing their root to the stack, so incorrect after 1st DFS. So, initialize depth = 0 in outer scope, for each connected component depth++ when pushing root, or else ans for max depth will be off by 1 if ans not found within 1st connected component.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 10/15/2023 12:58 AM

when using C++ map, set (the ordered kind), lower\_bound/upper\_bound returns iterator to *pair*type for map. Both lower\_bound and upper\_bound return iterator to .next key in sorted order, lower\_bound returning duplicate if exists, upper\_bound returns *strictly*next greater, use prev if want to access previous value!

// some problems can be done with either DP or combinatorics

// when thinking about ordered sequences of numbers, with rules based on comparison

of size of numbers, visualize and think about the numbers as bars in a bar graph, maybe we are trying to "plateau", smooth out, even out the bars, trying to make them all same height, or maybe we think of a solid bar, and bars to the right can have varying height, but height less than that solid bar



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 10/15/2023 12:58 AM

// be aware of types! I commonly make the mistake of initializing a vector of ints, rather than long long, I put long long value into int, but typecast, mods over, mutates value into incorrect one!

// multiplying and dividing some float A by another float B, e.g. A \*= B; A /= B, the value of A may be off from original value of A before the mult/div operations by magnitude of 10^-12, 10^-17, or some tiny amount, even if B is an int. Be wary of the precision errors, make sure the precision loss is within the tolerance of the problem! Since when multiply or divide, we only keep a finite number of accurate significant digits, we are forced to discard less significant digits (truncated), so when we do the inverse operation, we may not get exactly the same thing if enough significant digits are created from the multiplication/division

// when using max() and min() functions in C++, any numerical function in general, really, make sure that the types of both operands are identical. If the types of both operands are different, then compile error. e.g. should cast 2nd operand of int type to (long long) if 1st operand is long long

// I keep on messing up by trying to access .top() of empty stack or popping .pop() from empty stack, I can pop from stack without checking size IF I INITIALIZE STACK WITH ELEMENT IN IT TO BEGIN WITH, like my personal implementation of tree or graph traversal, immediately push root to DFS stack, but when I initialize stack as empty, then must CHECK SIZE STACK > 0 FIRST!!!!!!!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 10/15/2023 12:58 AM

I often make the mistake of declaring:

int n = matrix.size();

and use int j = n - 1; j >= 0; --j

when j should be iterating through a single row of matrix, but n is NUMBER of rows in matrix!

Frequently mix up i and j. Do not have lapses in concentration!

INT\_MAX + 1 = INT\_MIN i.e. (2147483647 + 1 overflows to -2147483648)

INT\_MIN - 1 underflows to INT\_MAX

maximum unsigned int + 1 overflows to minimum unsigned int (which is 0)

minimum unsigned int - 1 underflows to max unsigned int

if we declare variable as INT\_MIN on line 1, then decrement on different line, compiler does not produce warning. If INT\_MIN - 1 is an expression that creates a temporary variable, then compiler produces overflow warning message.

When dealing with huge products, modular dividing with some MOD, product of 2D matrix except

cell [i][j], we could multiply everything EXCEPT value, using prefix and suffix arrays (separating case of row i and rows != i), or we could try using long long overflow detection technique, and divide out.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 10/15/2023 12:58 AM

* Made mistake during UF programming team tryouts, infinite k-ary trees, with edge weight equal to the "label" of the edge, number of paths such that edge weight sum == n, and at least one edge has weight of at least d. "At least one" is a KEY PHRASE that should hint towards taking the complement. Solved this problem w/ dynamic programming, integer partitions, with k as bound for normal, with d - 1 as bound for contraint, subtracted the two arr.back() values, but since I was modding, had cases where subtracting resulted in negative number. In these cases, when early cases pass, and case 16 does not pass, then must add the MOD again to it (since we know for sure contrained d - 1 bound ways is smaller than k bound value ways.
* a lot of sliding window frequency problems can also be solved using prefix array of frequencies
* my brute force check to see if single char/digit insertion in string/number is found in hashmap can be optimized by making all prefix + suffix concatenations with a single element not included, reduces the complexity from O(d^2 \* n) to O(d \* n)
* ALWAYS USE {} WITH IF CONDITIONAL STATEMENTS! Countless times I have written a multi-line conditional block without the {}, so only the top line was executed, 2-line conditional block, second line was continue; so each time it would continue (as continue was not inside the if block), logic bug, but program guaranteed to never have a runtime error, so hard bug to catch!
* many arr[i] arr[j] arr[k] problems with dependencies, comparison, and indexing can be done using topological sort, then starting from global minimum/maximum and going from there



SJ

Shen, James











A blue and orange logo

Description automatically generated





To:​Shen, James​

Mon 9/4/2023 2:48 PM

# time complexity of hashing a string is not O(1) time, but rather O(L) time, with L being the length of the string. Doubling the length of string ~ doubles the runtime taken to hash the string. Rabin-Karp rolling hash algorithm?

# when doing contest programming problems with % mod on the final answer, since too many ways, huge combinatorial number, independent ways, etc., sometimes I forget to do a final mod outside of the for loop, REMEMBER TO DO A FINAL MOD IF DOING FINAL MULTIPLY (or add/subtract) OPERATIONS OUTSIDE FO THE FOR LOOP RIGHT BEFORE return



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Mon 7/24/2023 10:48 PM

-pay utmost attention, be extremely meticulous about EVERY KEYSTROKE when coding! Lapse in focus, typed   mid = (lo + hi);   rather than   mid = (lo + hi) / 2;   for binary search mid mid index update, only mistake in the code, and that caused runtime error, index out of bounds! A single mistake can ruin an otherwise correct and efficient program! So be careful! Think through in your mind what you want to type, and have a meta-focus to ensure that you catch your bugs, or better yet, don't make the bugs in the first place! EVERY CHARACTER IS IMPORTANT! like tennis, every point is important, piano, every note is important, put care and thought into it!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Mon 7/24/2023 2:14 AM

middle square method for implementing hash function, one-way function.

always pay attention of all possible edge cases! Like popping from an empty stack empty list, accessing empty list, NEVER DO THAT! When deleting/popping items, always think of the possibility that the list is already empty!

More edge cases: size input array is 0, is 1, is maximum size, when values are maximum/minimum, when input node is None/nullptr (tricky edge case, intuitively expect there to be at least one node, but not necessarily the case!)

When changing variable name, make sure to change ALL INSTANCES of the variable name, don't miss any places where it is used! If you forget to take out old variable name in RValue, then compiler will pick it up, but in a dynamically-typed language such as Python, old variable name as an LValue (previously used to update it) will just be treated as an initialization of that variable name, and the logic will be bogus. e.g. if old variale name was my\_val, it is used in a few places, and my\_val = 10 is left unchanged, when my\_val is intended to be changed to val\_1, then val\_1 will not be updated to 10, and my\_val will be initialized to 10, but never used!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 7/20/2023 8:50 PM

1. **/\* stuff you should look for**
2. **\* int overflow, array bounds**
3. **\* special cases (n=1?)**
4. **\* do smth instead of nothing and stay organized**
5. **\* WRITE STUFF DOWN**
6. **\* DON'T GET STUCK ON ONE APPROACH**
7. **\*/**

// don't fucking copy-paste your previously-written code! just proliferating bugs that were in that line, or could have some index oversight, forget to change nums[0] special case outside fo the for loop to nums[i] inside the for loop, so the program has bogus logic. Re-think through the logic whenever you write the code again, this will help to reduce bugs per line rate! Index copy-paste bugs or careless i and j mix-ups are the WORST bugs, humiliating

// ALWAYS MAKE SURE TO USE THE SAME EXACT INDICES i, j in the for loop. The worst bugs, hardest to find, are when you use i instead of j in the for loop, and similar bugs. Single-character variable names, very subtle, but result in garbage logic!

// Be careful about not double-counting tree node visits! Each node in a binary search tree can be visited up to three times! Very easy to double-count if not paying attention to this possibility

// Goldman Sachs Slang language for loop i variable scope is shared by its outside scope, pretty sure For loops and If condition blocks do not create their own scope, unlike in C++. variable i in Slang can be outside the For loop, C++ for(int i = 0; i < v.size(); ++i) the i index variable is destroyed immediately upon exiting the for loop

// When using C++ for each loop style, if the container being iterated is a 2D vector, and each element is 1D vector, remember to pass the element by reference, or else have to make deep copy of each vector, WHICH BLOWS UP RUNTIME!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 7/9/2023 2:44 PM

\* An empty Python string is 49 bytes! Each character in the string adds 1 byte!

\* import sys *then*sys.getsizeof( my\_var ) returns the number of bytes of my\_var !!!!!!!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 7/9/2023 2:44 PM

FOR BIG fixed integer values, even LONG LONG, make sure to modular divide in between multiplications to keep the values low, or else even long long can overflow when two big unsigned ints are multiplied with one another. Multiplying long long with unsigned int just underflows, wraps back around, so that is a common error that is hard to track down. Try to use SIGNED data types, can show compiler error for overflows! Long long multiplied by large int can overflow, compiler shows.

If I ever feel restricted by the max-size of long, I could build my own rudimentary BigInt size by just tracking a vector of base-10 digits. Or could implement a base MAX\_INT bignum system, like in Python.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 7/9/2023 2:43 PM

####### When using the

"ios\_base::sync\_with\_stdio(false); cin.tie(0);"

When program runs on my local machine, providing correct answer, but fails on the target machine grader, then probably too much memory allocated! Even though memory allocated fails at 13 MB / "~16 MB limit", I surmise 3 of those MB are already taken up.

A real strategy for reducing RAM usage, or reducing memory requirements when memory is the limiting factor, is to write temporary data to disk, then read from disk again.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

​\*\*\*\*\*\* \*Virtually every program's output is in the form of "lines". Since this is a UNIX environment, lines in all input/output files are terminated with a single newline "\n", rather than a carriage return plus newline "\r\n" (although properly-designed programs generally should not care which convention is being used). If your output does not contain a newline at the end of every line, it may be graded as incorrect. Note that the last line in the input file and output file should also end with a newline "\n" --- this is a common source of bugs: if you are testing your code locally, make sure your input files end with "\n" at the end of the last line, particularly if you are using split("\n") to separate individual lines in a language like Python. This is perhaps the number one reason for emails we receive saying "it works on my system but not the contest server".



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 7/9/2023 2:43 PM

**\* When getting values from I/O, cin, fin, DO NOT BREAK EARLY! GET ALL OF THE VALUES IN THE CORRECT ORDER, if break early, then will not get all the values, messes up the input order formatting, will likely make the program crash, reading values into the wrong variables when cin order is corrupted! BEEN HERE BEFORE!**

\***Pay attention to how you initialize vector. If already initialized with size, DO NOT FORGET and accidentally use .push\_back() instead of updating using index operator []! Or else vector size will be messed up,**reference binding to null pointer of type 'int'

**\*Before submitting answer, MAKE SURE ALL OF THE DEBUGGING PRINT STATEMENTS ARE COMMENTED OUT! Don't carelessly lose submission points!**

* how many distinct DFS traversals, without entering in cycle, that traverses all nodes?
* or, go the other way around, calculate the complement by figuring out all the double-counting cases.

SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 7/9/2023 2:43 PM

For my brute force counting method, can do custom capacities, meaning different number of symbols/states to choose from at each position/place value. So unit place could be base 2, tens place could be base 8, etc.

In the binary case, for the purpose of creating the powerset, all possible subsets of a set. What I have been doing is using builder vector, [1, 0, 1, 1, 1, 1] for instance, and incrementing using standard binary encoding incrementing ---> [1, 1, 0, 0, 0, 0]. However,

it takes amortized 2 operations (bit flips) on average for every increment, based on converging infinite series of Summation Sigma( n = 1 to +infty of   n/2^n), converges to 2. Calculated using brute force programming script, floating point arithmetic, and algebraically by taking derivative of the classic Power Series, Summation( n = 1 to +infty of r^n), we choose r to be 1/2.

Gray code has each adjacent binary representation differ in only 1 bit position, so a single bit flip, which better than using the traditional binary representation scheme. Hamming distance = 1. Gray Code bit position to flip can be generated as: 1 2 1 3 1 2 1 4...

1

1 2

1 2 1 3

1 2 1 3 1 2 1 3, similar to how the permutation builder position incrementer is generated, by concatenating itself, and then incrementing the last index, and similar to how the bitcounts of all binary numbers is generated. Number systems follow the same order, except that for larger numbers, you add a new bit for the new place value, which adds 1 to the bitcount of all previous bitcounts of the smaller numbers. Similar to what Donald Knuth was doing in his analysis of how many times the maximum value is expected to update (EV calculation) in the simple linear algorithm of finding the maximum in an array of unique numbers (no duplicates). Permutational analysis.

####### Python fout.write() file write must be str type! And reading file into memory must

####### go into str variable type!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 7/9/2023 2:42 PM

Common mistake when doing while(true) binary search loop:

For Cutting Ribbons CodeSignal Q4, remember to set binary search hi variable as max(ribbon\_lengths) + 1, rather than max(ribbon\_lengths), to prevent the edge case of k = len(ribbon\_lengths), (k being the minimum number of ribbon cuts of length L that we have to obtain) with each ribbon length being the same, so each ribbon length is max(ribbon\_lengths), we can keep the length of the ribbons (without actually cutting, we already have k ribbons of length L, in this case L is max(ribbon\_lengths) ) results in TLE infinite while loop, ending up as lo == hi - 1, but never breaking out, since my if conditions compare k, and only breaks out of while loop (by returning a value) if mid, mid + 1 calculations differ from k. The condition logic has it that if we can create >= k ribbons of length L, L being mid + 1, then that means we can keep making the cut size bigger. In this case, with hi being max(ribbon\_length), our mid, calculated from (lo + hi) / 2, can only go up to hi - 1, mid can never reach hi. So if we have k = 10, and len(ribbon\_lengths) = 10, and all ribbons are 100 long, our mid eventually becomes 99, mid + 1 becomes 100. For cut size L of both (mid) and (mid + 1), we can make 10 ribbons of size L, so the binary search keeps on going, anticipating that we can make the cut larger, and find the specific boundary of (mid), (mid + 1) where cut size L of (mid + 1) finally < k, in which case we return mid. So we set lo = mid, and calculate new mid = (lo + hi) / 2, BUT WAIT, OUR MID STAYS THE SAME, WE ARE STUCK AT MID = HI - 1, FOREVER! And since (mid + 1) == hi, we can always make k cuts of (mid + 1) cut size, in this edge case. We can initialize hi to be max(ribbon\_lengths) + 1, to ensure that the (mid + 1) > max(ribbon\_lengths), ensuring that there will be 0 cuts of size (mid + 1). This will finally terminate the while loop in this edge case! TOOK ME 9 MONTHS TO FIGURE OUT THIS PROBLEM!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 7/9/2023 2:42 PM

!!!!!!! STRIVE TO THINK ABOUT COMPUTERS AND PROGRAMS AND ALGOS AT ALL LEVELS EVERY DAY!!!!!!! When you get better at chemistry/physics, can think about the electrons and materials level, energy and matter and philosophy and the state of representation, yin, yang, data and information, then think about circuits, then logic gates, then the processor level, what the architecture of the CPU is, the datapath in in the CPU, the instruction fetch cycle, think of the current as moving in the clockwise direction of the cartoon CPU datapath diagram, then think of the controller logic. CPU is the microprogramming level, have program counter logic, the multiplexer choice of having the constant "4" added in at the hardware level, as a choice for ALU operand, to increment the program counter, which has its special register? (or if it doesn't have special register, then stored in memory, but probably special register). Then think of the machine instructions level, OPcodes and instruction types (I, R, J-type instructions), learn about pipelining soon, think about computer I/O, writing words into memory (either data for the program to process, or the program instructions themselves) using the inport (special memory address), get program output through the outport (special memory address). Then think about the assembly code level. How is assembly code converted to machine instructions (conversion through assembly, just a code conversion, 1-to-1 correspondence, perfect information, reversible, unlike compiler, difficult to see how the compiler generates the machine instructions. Runtime errors are things caught by the C++ executor, maybe division by zero (error probably handled at the hardware level?), or segfault, things that are impossible to know at compiler time, since we don't know what data is coming into the program. These are things the human has to figure out, what data can/cannot be entered into the program. A program can work perfectly fine, as long as there are no negative numbers for instance, depending on how the program was written!

Think about how arrays, variables in a high-level programming language become registers and RAM memory at the assembly level. How do we optimize the usage of registers, using as many of the general-purpose registers as possible, much faster than spending additional CPU cycles to read intermediate calculations in memory.

Think about the stack pointer, think about the stack in memory (is it a special place to store the memory location of the instruction to jump to?), think about how a nested for loop, nested 40 levels deep, would be represented in a program in assembly!

How is a hashmap implemented in assembly code?

Think about how hashing, data structures, are implemented, how is a metal C structure converted into assembly code, do each of the attributes become ordered sequence in memory, a bunch of pointers to other memory locations, each pointer 8 bytes, so structure with 4 attributes has the first 4 bytes being pointers? So struct in C with attributes int size, int max, int weight, string name, float volume, would be array of pointers, indices 0, 1, 2, 3, during code compilation, RegEx performed to match each attribute with the corresponding index in the array of points of attributes for the struct?

\*\*\*\*\*\*\*Instruction set is the interface between hardware and software!!!!!!! Never forget that!!!!!!!

The technique of writing intermediate results to the disk is a great one, if the system being used is equipped with only a limited amount of RAM. Parallels the CPU architectural paradigm of writing intermediate results to memory! (RAM in this generation of computers)

-Buddy Strings Leetcode: Lots of edge cases, THINK OF THEM ALL! Break them down into casework. Imagine if there are multiple index swaps allowed, not just 1. What would the time complexity be? Would it be a combinatorics problem, could it still be O(n) time complexity by treating the underlying subproblem of chars at indices not being combinatorics, but by using is\_anagram hashmap technique?



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 6/15/2023 9:57 PM

ALWAYS BREAK / RETURN FROM while(true) loop, or infinite loop!

if program keeps on running without terminating, and does not produce output, then consider and infinite while loop being a potential culprit!

\*\*\*\*\*\*\*

\*\*\*\*\*\*\* READ EACH QUESTION CAREFULLY!!! MAKE SURE THAT QUESTION IS ASKING FOR POSITIVE VS NON-NEGATIVE, MAKES A HUGE DIFFERENCE IN THE ANSWER TO A PROBLEM

\*\*\*\*\*\*\*

Problem Solving Checklist Strategy when solved: (Algorithm to practice and memorize as a human ("human algorithm"))

####### Ideally for iterating permutations, iterate by calling the next\_permutation function. Pretty sure it is O(n) for next\_permutation, so iterating through all permutations would take O(n \* n!) time using next\_permutation. Using my recursive permutation lister, for each element in a single permutation, have to extract that element, then take the left and right slices (prefix and suffix, not including extracted element), so ~1/2 \* n^2 time complexity for single permutation of length n, so O(n^2 \* n!) for listing all permutations.

####### I wrote that recursive permutation almost a year after I wrote my first permutation lister, which was an iterative one. Used modular division, variation of my counting builder method, to select the index of the chosen element. Then deleted the chosen element from the element array. So again, O(n^2 \* n!), n^2 because of the deletions. An improvement from my first iterative, modular division permutation lister algorithm was using current count and a capacity in addition to a builder. No division, so faster, but still deleted elements from arrays (deleting from middle of array takes O(n) time). Then, another improvement, more elegant, a builder for the builder! Use a "genetic increment sequence", 121213 121213.... to indicate the index from the right to increment in builder array, all other elements to the right become 0. But still O(n^2).

####### Key breakthrough, just today: Instead of making a temp array of our current elements, and choosing elements without replacement (deleting) to build the permutation FOR EVERY PERMUTATION, can just memoize all subsets of the items! Use my counting method to generate all subsets of elements, since total number of subsets is O(2^n), which is insignificant compared to O(n!). Exponential growth is slower than permutational growth. I still create all of the permutation builders O(n!) amortized, since most of the time I'm iterating through very few elements in builder when incrementing. Pretty rare to iterate through entire builder (it happens). But for each permutation builder, I still have to build every permutation from an empty string, so algorithm is O(n \* n!), even without deletion in the middle. Deleting in the middle can be avoided if the power set memoization is used, and XOR operator is used.

#next permutation algorithm, repeated n! times, is a great algorithm to achieve O(n \* n!)

#swap-based permutation listing algorithms are good, I should think about them. They are faster, truly O(n!) time, since building each permutation is O(1) amortized, as swapping avoids building every permutation from scratch (what I do, which is wasteful)

#it would be awesome to have an O(n!) permutation listing algorithm, by getting a closed-form, pure mathematical relationship between each permutation and the next one, sequences of the indices. Instead of using my without-replacement system of 0000, 0010, 0100, 0110, 0200, etc., this system would be 0123, 0132, 0213, 0231 WITH REPLACEMENT system. I've taken the integer differences between successive permutations and analyzed them, but was unable to unlock the pattern.

0. Your personal brute force counting method! Can count through and list out permutations (either using capacity, count, builder by finding MOD pattern from last summer, or by using new 121213 121213 121213 121214 ? increment index pattern for modifying builder), combinations (in sorted order), sequence of bitcount of numbers in range (0112 1223...) self-copying and bulk incrementing pattern.

1. read problem, if it is numerical and can be expressed

        //    as a function, see if function can be honed in on using bisection

        //    search. Sometimes hard to find closed-form quick exact solution,

        //    so bisection algorithm works well. For exact solutions to the decimal

        //    point, bisection search can never find true answer, but if the question

        //    only deals with integers, AND THE FUNCTION MUST BE MONOTONICALLY INCREASING/DECREASING, then bisection search can narrow down to the

        //    optimal solution!

  \*\*\*\*\*\*\* If the function is not monotonically increasing/decreasing, then find a way to partition it so that it is monotonic! Can then do bisection search on each of the groups!

If one of the variables is time (only increasing), or if list is sorted, then consider binary search for quick searching!

2. "Elegant insight" problems: Ship traffic ICPC problem, instead of ships moving,

move the point in time (rewind, fast-forward the video as the onlooker). Group all ships as eastward or westward. Consider the ships moving towards the east as a whole, and the ships moving towards the west as a whole. "time-shift" the ships into the same lane (ships further away are transposed a greater distance). Then in this new "single lane", merge the same-direction ships if they overlap in their lengths into one jumbo ship (like merging intervals). Then, figure out the times when each direction is clear (the complement of the times when each direction has ships in the path), then merge opposite direction time intervals to see if there is overlap. If no overlap, then impossible. Rather than do a bunch of O(n) merge two arrays of intervals (two pointers method), just represent each interval as open or close. Then you have two types of events, "ship begin", and "ship end", and we put ALL ship intervals in all lanes of the same direction into ONE 2D array, then sort by x-position, with 0 representing "open" or "begin" event, and 1 representing "close" or "end" event, so that when sorted, ships always begin before they end. Keep a "depth stack" of how many overlapping ships there are, when depth stack count (stored as int) goes to 0, we know the jumbo ship overlap interval ends.

Another elegant insight problem is the famous ants on a plank, what time when the last ant falls off the plank (other version is robots on a number line, move left, right, colide, change direction, find the sums of all distances between all pairs of robots). When ants collide, change direction, instantaneous change in direction whilst keeping same speed. Instead of keeping track of each ant's IDENTITY, we can think of the ants as "moving through" all the other ants, we ignore every other ant, ant keeps on marching in same direction. This is because we can just "swap" the identities of the two ants. Don't actually have to consider the collisions, unless we want to keep track of EXACTLY WHICH ANT is at which position and preserve identity of the ants.

//  (although I wonder if I can write up a way to track the location of each ant by IDENTITY, know exactly where each ant is from start to finish, rather than simply knowing the positions of all the ants at the end, with identity scrambled).

3. Always consider sorting to improve time complexity! Order is most useful than disorder!

4. Is question a hidden graph problem? (convert fraction to repeating decimal is a cycle-detection graph problem, USACO prefix was hidden graph problem, that is how I approached it, even though intended solution was DP. Sometimes there are multiple ways to do problems) DFS? BFS? Depth-Limited Search (DLS)? bridges? cycles? directed, undirected? Trees? Parents? Children? Leaves? Degrees?

5. Shortest path? Floyd-Warshall? Dijkstra? Bellman-Ford? Newest negative weights shortest path (2021 randomized algo)? Just let the graphs keep running for the maximum amount of time (check if small enough to be able to brute it) to check if cycle reached?

6. Connected components, islands, union finds?

7. My personal consecutive ints in a data stream union find, keep track of left bound, right bound, check if new num connects two blocks, or left/right extends one block, or makes new singleton block, most number of consecutive flowers in bouquet of flowers on certain day after blooming is a variation of this problem.

8. "Spring cleaning" personal algo, can allocate fragmented array, and do "spring cleaning" once in a while to tighten the array again, move elements to beginning of array after drifting and fragmenting to the right. Alternative to doubly-linked list in Design Least Recently Used (LRU) Cache. Figure out how Least Frequently Used (LFU) Cache adds onto LRU.

9. Consider using Monotonic Stack, Queue, deque! Deque implemented in both directions, make an array with capacity DOUBLE that of intended max\_size, and push first element to the middle index, then push\_back left-appends, push\_front right-appends.

10. When memory is very tight, consider using BITPACKING! In USACO Prefix problem, converted adjacency list (containing 0 to 9, representing relative how many index positions to increment when going to that adj node) into a single INT, with 2^0 representing 0, 2^9 representing 9, unique way to represent the set of adj nodes. Reduced memory usage 7x

11. Two pointers method

12. Sliding window

13. Working from the back of the array/string, instead of from the front.

14.Dynamic Programming

15. Greedy, can I prove that the greedy approach is optimal?

16. MATH. Prime sieve tricks, prime factors efficient sieving, all proper factors efficient sieving, LCM of a range sieving, GCD of a pair calculated in O(log(n)) time, for GCD of a set of numbers, find GCDs of the smaller numbers first, this will speed up the whole calculation by making everything smaller. Euler totient. MEMOIZATION IS VERY IMPORTANT. Gaussian elimination.

17. GEOMETRIC ALGOS. Sweepline algo for closest points, 2D. Divide and conquer algo, even more important.

Convex Hull algo! Figure it out yourself, farmer fencing LeetCode!

18. Jump array (sqrt(n) block sizes, num\_blocks is sqrt(n) as well) for O(nsqrtn) range sum optimization.

19. Prefix array, Suffix array.

20. Learn how to build a heap from pointers, AVL, red-black tree, learn seg tree algo, minimum spanning tree, Knuth-Morris-Pratt algo, Tarjan's algo, skiplist, bloom filter, etc.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

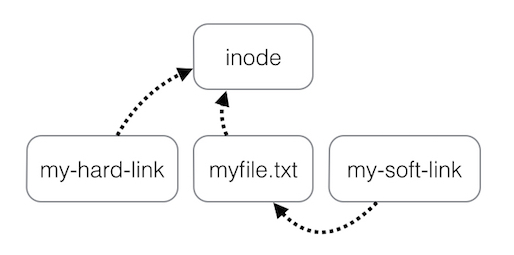
Description automatically generated





To:​Shen, James​

Thu 6/8/2023 12:19 AM





SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Wed 6/7/2023 4:40 PM

HAVE A MIND FOR YOURSELF! Don't get swayed soich by other people. Be more jing guay. Be sharper. I.e. pay with corporate card with tip, rather than give cash! Think things through carefully, pay attention, navigation, street smart, communication. Time management.

Get [Outlook for Android](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Faka.ms%2FAAb9ysg&data=05%7C01%7Cjamesshen%40ufl.edu%7C757546fbdaca4baad3f308db67d78a83%7C0d4da0f84a314d76ace60a62331e1b84%7C0%7C0%7C638217947672420193%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=ETgboV%2BhZt4kNqy0ybK1QIhOfe6jdtOeRhY5XOQJ%2BZQ%3D&reserved=0)



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 6/4/2023 1:04 AM

Benq competitive programming tips

int overflow, array bounds special cases (n=1?) do smth instead of nothing and stay organized WRITE STUFF DOWN DON'T GET STUCK ON ONE APPROACH



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Wed 5/31/2023 4:44 PM

For Leetcode Can Place Flowers problem, my approach was to split the flowerbed into streaks of empty pots and pots with flowers using RLE, and then check for edge cases (only one streak in RLE, only one streak and length only 1 (just 1 empty pot in entire input), right and left blocks of RLE, and center blocks surrounded by other blocks). Using divisibility and number of adjacent neighbors that need to be empty, we can solve the maximum number of flowers we can place. O(N) to build RLE, and then O(num streaks), but using division and modular division is very slow! Instead, we can use greedy algorithm approach. Place a flower at the EARLIEST POT you can place it in. For small k, where k is number of adj neighbor pots that must be empty, we can just check the 2 \* k neighbors (remember to check for out of bounds for indices near the array boundaries) at each index to see if there is a flower already planted. But when k is large, same magnitude as array length N, the algorithm will balloon to O(N^2). Instead, keep track of index of last planted flower to the left, and do a "pre-scan" or a "scout recon" look-ahead search to find the next already-planted flower to the right. Then this becomes O(N).



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Wed 5/31/2023 1:44 PM

In Python:

arr1 = [1, 2, 3]

arr2 = arr1 # alias

arr3 = arr1[:] # actual slice copy, changing arr1 won't change arr3



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Wed 5/31/2023 10:49 AM

THIS HAS HAPPENED MULTIPLE TIMES:

remember to put a (   ) parenthesis around bitwise operations!  a & b == b will be interpreted as just the calculation a & 1, since the == is evaluated first, by C++ operator precedence order!

ALWAYS REMEMBER TO PUT PARENTHESIS AROUND BITWISE OPERATION EXPRESSION BEFORE CHECKING FOR EQUALITY!



SJ

Shen, James











A blue and orange logo

Description automatically generated





To:​Shen, James​

Wed 5/31/2023 10:49 AM

Great self-discovered permutation builder generator (by indices) and bitcount generator. For permutation, the first few permutation builders for 4! are: 0000, 0010, 0100, 0110, 0200, 0210, 1000. Each builder becomes the next by passing through an incrementer function, the indices that are incremented has the sequence 1,2,1,2,1,3,1,2,1,2,1,3.... The sequence can be generated from scratch like this: start with [1]. Then, copy over EVERYTHING in seq so far COUNT # times. COUNT is 1. Seq = [1, 1]. Increment last element by 1, increment COUNT by 1. Seq = [1, 2]. Copy everything COUNT num times, increment last element, increment COUNT, etc. The increment sequence can be used by the incremnt function to create the next builder. Each builder describes the indices to pick from the remaining permuted objects (with no replacement)..

The bitcount generator is as follows. Start with [0] (0 bits in 0), then copy over EVERYTHING in list so far, add all copied elements by 1, append to end of list. [0, 1]. Repeat. [0, 1, 1, 2]. Repeat. [0, 1, 1, 2, 1, 2, 2, 3], etc. etc. Each iteration grows the length of list by power of 2, makes sence for bitcount generator.

AGAIN, COMMENT OUT YOUR cout AND print() STATEMENTS! I/O and STDIN/STDOUT

ARE VERY SLOW! MANY TIMES I HAVE FORGOTTEN TO COMMENT OUT PRINT

STATEMENTS BEFORE THE SUBMISSION, WHICH COST ME AN ACCEPTED ANSWER!

And numerous borderline runtime cases where commenting out a print statement

turns a TLE into accepted.

LEARN THE FAST I/O methods in C++ !!!!!!!

To check if two numbers/strings are permutations of one another, just see if number/string

is anagram of the other, using either digit freq vec of size 10, letter freq vec of size 26,

or hashmaps to track symbol key and freq of that symbol.

SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 5/30/2023 5:15 PM

Python fin.read(), and then parsing out the "\n" myself is a lot slower than using fin.readline() and then concatenating the individual lines without the "\n".

REMEMBER TO USE fin.readline().strip()  and maybe  .split() !!!!!!!

ALWAYS REMEMBER TO USE .strip() since the "\n" char is included in the readline return!

I DRILLED THIS INTO MY HEAD LAST SUMMER, DON'T FORGET IT!

* do clever things like if (*certain input or test case), return 0* at different locations throughout the code to figure out where the error is. "Binary search"
* intermittent printing in a for/while loop if(*count % 10000 == 0*)
* remember to check if indexing is correct, seg fault, index bounds off by 1?
* am I reading in the correct input/output files? Do the file names exist on my computer? etc. Can cause bad\_alloc() if try to read from file that does not exist.
* If allocate a static raw array on the stack, if array size is too big, can cause program to simply terminate without error note!
* can use bitpacking to reduce memory!

C++ raw array parameter pass in function:

prefix\_cpp.cpp:25:34: error: multidimensional array must have bounds for all dimensions except the first

Remember to comment out your own input and output file names on your own computer when you submit code to USACO, input/output file names should be question name!

raw C 2D arrays are quite tricky to use. But the syntax is shown below.

int\*\* graph = new int\*[S\_length + 1]; // do I need to allocate this on heap, too large for stack?

      for(int i = 0; i < S\_length + 1; ++i){

            graph[i] = new int[10];

      }

      for(int i = 0; i < S\_length + 1; ++i){

            for(int j = 0; j < 10; ++j){

                  graph[i][j] = 0;

            }

      }

// pass in by parameter

bool index\_in\_graph(int i, int\*\* graph){ // int graph[][10]



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 5/30/2023 2:37 PM

USACO Prefix problem-solving attempt log:

1. Started this problem last summer/fall, coded it up in Python first, program ran correctly on my computer and on USACO Training server, gave correct output, but too slow. Passed 5/6 test cases, last testcase with 200,000 S string length failed, ran in ~1.4 sec.
2. Programmed same algorithm in C++, ran quickly, but had bad\_alloc memory error. Passed 5/6 test cases, but 6th case failed for different reason than Python. Tried looking at all of the indexing, looked at the graph DFS traversal algo, did not find error.
3. Utilized cout statements to try to see where my error was.
4. Used cerr statements (show up when runtime error, when cout does not) and "binary search" style printing to locate which lines of code were the culprit.
5. Turns out that the graph-building code was to blame for the bad\_alloc. Used cerr to intermittently print out index i in the for loop, to see at which point the program terminates. At i = 198,000, program terminated. Originally used unordered\_map hm to represent graph, but unordered\_map uses more memory than vector, so switched to 2D vector to represent graph. But still too much memory.
6. USACO training server runtime statistics report ~13500 KB memory used, purportedly 16 MB limit, so a little confusing. I think there is already some memory used to start up a C++ program, so less than 16 MB is actually available for use, like how 128 GB laptop only has 100 GB free for user's personal storage.
7. The previous 5 test cases passed, so logic and algorithm is correct, but memory was the issue. More proof that memory was the issue presented itself when I initialized a 10^5 size vector, 2 \* 10^5 size vector, and 10^6 size vector right before the graph-building code block. The program terminated at a much smaller index i when iterating through S, so when building the graph, even with 2D vector, used too much memory.
8. Makes sense that too much memory was used, since the test case has many consecutive 'A' chars, and the substrings are "A", "AA", "AAA", "AAAA", "AAAAA", etc. So most adj list lengths were 7.
9. With adj list length 7, capacity is ~double actual length of vector, with sizeof(int) being 4, 14 \* 4 = 56 bytes per index, 56 \* 200,000 ~= 11 MB, not including extra capacity of outer graph vector, and overhead, so lots of memory allocation.
10. Instead, in graph could store *indices that are not adjacent*, so an "inverse adj list". But there are 13 substrings in total, so even inverse, each graph node would have ~6 in adj list, not too much of an improvement.
11. So instead, could store start,end indices for "A", "AA", "AAA"... substrings, would turn 7 elements in adj list to 2.
12. For Python algorithm, try to improve runtime by doing a huge fin.read(), read entire file, instead of .readline() one at a time, and then parse out '\n' myself, to see if faster. Since Python reading in file already takes 1 sec. The O(n^2) DFS algo can be improved to O(n) by storing last i index left off at in each adj list, so we don't have to start from i = 0 in each adj\_list each time. But the O(n^2) that it is right now runs in 0.5 sec, so my file reading should be the first thing I try to fix.
13. The Python memory definitely surpassed 16 MB, since Python uses much more memory than C++, and my algo was the same in both my C++ and Python implementations. This just indicates that USACO training server allows more memory usage in Python solutions than in C++ solutions.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 5/30/2023 1:05 AM

Interesting observation:

I was working on the USACO Prefix problem and implemented two approaches: 1. unordered\_map representation of an adjacency list of graph 2. vector<vector<int>> repr. of adj. list of graph.

Both implementations had used exactly the same amount of memory, so I am wondering if the C++ compiler is analyzing the code to see what key values are inserted into the unordered\_map. Maybe the compiler sees that the keys are within a limited range, and quite dense, so converts into a vector<vector<int>> internally and automatically, saving memory and improving runtime. That would require replacing every instance of hashmap variable name and function with respective vector function! Compilers are crazy.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Mon 5/29/2023 6:16 PM

2 options for priority\_queue syntax in C++:

old syntax (supported for more C++ versions):

priority\_queue<T, vector<T>, comparator class> pq;

\*\* must be comparator class, not just a function. Make a comparator class, and in public attributes overload the () operator, so the function comparator overloads comparator class name.

new syntax (leaving out template arguments for priority queue, compiler deduces this,

        is only supported from C++ 17 and beyond):

vector<int> vec;

priority\_queue pq(vec.begin(), vec.end(), bool\_comparator\_func());

\*\* third argument does not have to be class overloaded, can just be a bool comparator.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Mon 5/29/2023 3:12 PM

Retyping algorithms frequently is good practice. Sometimes a hastle. Make a library of common functions you use for Project Euler (bool is\_prime, prime sieves, consecutive full prime factorization sieves, consecutive distinct prime factorization sieves, consecutive all factors factorization sieve, consecutive LCM sieve, GCD, counter "builder" array, coprime Euler totient function, etc.)

    // be on the lookout for overflows. When the C++ program terminates

    // in the middle, without outputting anything, that is a sign

    // that there is a memory bug, potentially due to undefined behavior

    // involving numeric types.

  // Or could be index out of bounds, accessing memory past array/vector bounds

REMEMBER TO COMMENT OUT cout print statements before running.

But sometimes, paradoxically the runtime is reduced when a cout statement is left within the code. This could be due to avx,avx2,fma compiler flags



You forwarded this message on Fri 5/10/2024 2:10 PM

You forwarded this message on Fri 5/10/2024 2:10 PM

SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 5/28/2023 11:21 PM

REMEMBER, USING C++ bool IS QUITE SLOW! FOR SIEVE OF ERATOSTHENES, USING bool WITH SIEVE RANGE 1e8 RUNS IN 20 SEC, WHILE USING INTS RUNS IN 1 SEC (with #pragma GCC optimize("Ofast") #pragma GCC target("avx,avx2,fma")).

USE SEGMENTED SIEVE OF ERATOSTHENES FOR MEMORY-SENSITIVE QUESTIONS.

Allocating huge (1e8 elements) raw C++ arrays (not vector objects) can only be done as a static global variable (outside of main function). Storing on stack results in memory error.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 5/28/2023 10:17 PM

REMEMBER, IN SIEVE OF ERATOSTHENES, CAN MAKE INNER J LOOP TO CROSS OUT MULTIPLES OF PRIME NUMBERS, START WITH k \* k as the smallest prime! 2 \* k is slower, since 2 \* k was already crossed out at the very beginning when 2 was the prime being dealt with!



SJ

Shen, James











A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 5/28/2023 4:51 PM

prime number list generator method 2. can be optimized by calculating square roots ahead of time.

Instead of calculating square roots every time, "calculate" by using multiplication (but then again, if implementation has the square as hashmap key and the root as value, must deal with added runtime constant multiple of hashing!)

--- complexity from O(n^2 / log(n)) to O(n sqrt(n / log(n)))

--- 4. To reduce sieving memory usage, can split the sieve into multiple parts. Instead of one big array of size n = 10\*\*9 (too much memory to allocate all at once), can split into multiple sieve chunks of array length 1000, or array length 1,000,000. Experiment to see what the effect of sieve subchunk size on number of total operations, the time/space complexity tradeoff.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 5/28/2023 4:18 PM

prime number list generator:

1. brute force: check all numbers in range, check each number for divisors <= sqrt(itself), runs O(n root n)
2. start with prime list 2: Then for each num in range starting with 3, check if divisible by any of the previously found primes. If prime, then add to prime list. Since around x/log(x) primes in range x, complexity is O(n^2 / log(n))
3. basic sieves: Sieve of Eratosthenes O(n log log n), Sieve of Euler O(n)
4. More advanced sieves

prime factorization in range can also be brute-forced by checking each number individually,

but it can also be sieved efficiently. Similarly, all proper divisors of nums in a range can be sieved,

as well as LCM of range of numbers.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 5/28/2023 2:52 PM

Seems like Python interpreter has undefined variable name closest match algorithm.

In code below, printed out ans1, even though print statement was for ans, but ans was not defined.

import time

import math

# compare str conversion method with logarithm method

def logarithm\_method():

    n = 2

    a = 1

    b = 1

    while True:

        c = a + b

        n += 1

        a = b

        b = c

        num\_digits = int(math.floor(math.log10(c))) + 1

        if num\_digits >= 9:

            break

    print("b:", b)

    print("c:", c)

    while True:

        c = a + b

        n += 1

        if n % 1000 == 0:

            print("n:", n)

        a = b

        b = c

        last\_9 = c % 1000000000

        num\_digits = int(math.floor(math.log10(c))) + 1

        first\_9 = c // 10\*\*(num\_digits - 9) # can continuously multiply with each iteration rather than wasteful exp

##        if n == 541:

##            print("first\_9:", first\_9)

##            print("last\_9:", last\_9)

##            print("c:", c)

##            print("num\_digits:", num\_digits)

##            break

        ok = True

        for x in [last\_9, first\_9]:

            present = [False for \_ in range(10)]

            rem = x

            while rem != 0:

                present[rem % 10] = True

                rem //= 10

            for d in range(1, len(present)):

                if present[d] is False:

                    ok = False

                    break

            if ok is False:

                break

        if ok:

            print("ans:", n)

            print("first\_9:", first\_9)

            print("last\_9:", last\_9)

            break

##

##def string\_method():

##

def main():

    begin = time.time()

    ans1 = logarithm\_method()

    finish = time.time()

    elapsed1 = finish - begin

##    begin = time.time()

##    ans2 = string\_method()

##    finish = time.time()

##    elapsed2 = finish - begin

##

##    if ans1 == ans2:

##        print("The methods return the same")

##    else:

##        print("Check your algorithms")

    print("ans:", ans)

    print("logarithm:", elapsed1, "sec")

    print("string:", elapsed2, "sec")

if \_\_name\_\_ == "\_\_main\_\_":

    main()



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 5/7/2023 1:11 AM

Although it is quite convenient and concise to not use curly braces for a single-line conditional statement/loop statement, BE EXTREMELY SURE that the statement is a single-line statement! OTHERWISE IT IS A TOUGH BUG TO CATCH, WILL READ THE CODE AGAIN AND AGAIN AND NOT SEE ANYTHING WRONG!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 4/8/2023 11:00 PM

REMEMBER TO MAKE VECTOR TYPE FOR PREFIX AND SUFFIX ARRAYS LONG LONG, OR ELSE WILL FUCKING OVERFLOW! BITCH!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 4/1/2023 11:46 PM

Remember to PASS BY REFERENCE, NOT BY VALUE, for complex function parameters such as vectors, objects, structures, etc.

Passing by value is fine for primitives, however.

\*\*\*\*\*\*\* In a LeetCode contest, I TLE'd when I passed by value in my custom static bool comparator 2D vector sort. Passing by reference resulted in the accepted solution. TIL (Today I Learned), DON'T FORGET TO PASS BY REFERENCE!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 3/28/2023 2:53 PM

Don't simply make a vector with {  } curly braces each time! Compiler does not know what I am doing sometimes. EXPLICITLY make an empty vector, then push\_back!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 2/2/2023 10:31 PM

size() returns unsigned int. So when size() is zero, size() - 1 becomes UINT\_MAX instead of -1. It's one of the most annoying things.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 2/2/2023 10:18 PM

Sometimes in C++, make loop condition .length() or .size(), cast it to int using C-style (int) or preferred static\_cast<int>() on the return values of those methods, returns a size\_t type value! Should consider casting all to int if there is a mysterious stack buffer overflow Address Sanitizer! William Lin came across this while doing Make the String Great LeetCode 1544. in weekly contest 201.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 2/2/2023 4:11 PM

For LeetCode trapping rainwater and similar questions, have the option of O(n) monotonic stack (and apparently two pointers O(1) memory), O(n log n) topological sort, find the highest elevations and assume all filled with water in between, and then "subtract out" the land when dealing lower elevation lands, since between the taller elevations we originally assumed filled with water, but later we find out exactly how much land is filled between it. Also, have the option of adding up the amount of water in horizontal stretches, one depth at a time, although this seems to have the poorest time complexity, if there are tiny isolated, but very deep, ponds of water.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 1/31/2023 4:33 PM

Remember that copying container in C++ seems like a deepcopy, from the code below, while

in Python it seems like an alias. In Python either use import copy copy.deepcopy(<my\_container>) or element-by-element insert into new empty container.

a = set()

a.add(1)

a.add(2)

a.add(3)

b = a

a.add(4)

a.add(5)

print("a:", a)

print("b:", b)

unordered\_set<int> set1;

  unordered\_set<int> set2;

    set1.insert(1); set1.insert(2); set1.insert(3);

    unordered\_set<int> set3 = set1; // this assignment seems like a straight up deep copy, not alias like in

                                                             // Python

    set1.insert(4); set1.insert(5);

    unordered\_set<int>::iterator it;

    cout << "set3" << '\n';

    for(it = set3.begin(); it != set3.end(); ++it){

        cout << \*it << ' ';

    } cout << "\n\n";

    cout << "set1" << '\n';

    for(it = set1.begin(); it != set1.end(); ++it){

        cout << \*it << ' ';

    } cout << "\n\n";



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 1/22/2023 9:16 PM

Total ways to partition a string S into substrings: 2^(S.length() - 1)

* To generate all unique partitions, could write out all numbers from 0 to total\_ways in binary, and then quickly parse each binary representation. Each streak of successive 0's or successive 1's is a substring. But would have to check for duplicates, since 110011 would split S into 3 substrings of length 2 each, as would 001100, i.e. there is a duplicate for each unique way when generated in this manner.
* Instead, we realize that the duplicate of a unique partition is just the binary inverse of the partition.

1 -> 0, and 0 -> 1.  110011 -> 001100. So the duplicate partition, the inverse, is just the max binary value of S.length() number of digits, 111111 in this case, subtracted by the partition. So duplicate inverse of 0000 is 1111, inverse of 0001 is 1110, inverse of 0010 is 1101, etc. So instead of generating all possible binary numbers from 0 to 2^S.length(), we only need to list out from 0 to 2^(S.length() - 1).

* Another way to think about this is: at each position in string S, we can choose to start a new substring, or continue the previous. This is a binary choice. However, at the first position, index 0, we HAVE to start a new substring, we cannot choose to continue the previous, since it is the first position. Therefore, for a string S of length L, there are 2^(L - 1) ways to partition into substrings.
* Can also use backtracking (a la ClimbingStairs CodeSignal, and ways to sum to 100 Project Euler, although for that problem 1, 1, 2 and 2, 1, 1 are not unique.)
* LeetCode 131 can be solved by above binary generating using Jache Builder or Jache Bitmasks, or by finding all palindromes and then using a graph DFS traversal, or by finding all palindromes and then using back-to-front Dynamic Programming



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 1/14/2023 12:16 PM

JAMES'S COMMON GRAPH DFS ERRORS!

-for iterative graph DFS, remember to pass the status ('w', 'g', 'b') hashmap/vector by reference, so it can be updated. Also pass the curr\_index hashmap/vector by reference, so it can be updated too (to store where we are in each node's adjacency list, to prevent unnecessary repeated traversals, like in my old DFS method).

-REMEMBER TO MARK THE ROOT AS 'g', OR ELSE IT WILL MESS UP THE TRAVERSAL! I had this bug in LeetCode "Lexicographically Smallest Equivalent String."

-A graph can be a single connected component or a collection of multiple disjoint connected components! In that case, must do a DFS for each connected component. For each input node, check to see if 'w'. If not, then a previous DFS already took care of it, but if 'w', then make that our new root, MARK IT TO 'g'!!!!!!!

-Do not include the node in its own adjacency list.

-For a k-ary tree, a special graph, knowing the parent of a node makes traversal very easy. Do not even

have to use a for loop within the *while root is not black*to find the next index of a 'w' node. Can just go in order, and increment, since there is only one path to each node! Unlike in a general graph, where there can be multiple paths, so nodes in adj list can be marked as 'g' or 'b' out of order.

-REMEMBER TO BREAK AFTER FINDING THE NEXT UNDISCOVERED NODE AND COLORING IT GRAY! WE ONLY WANT TO FIND ONE UNDISCOVERED NODE AT A TIME. NOT BREAKING WOULD GO ALL THE WAY DEEP, multiple nodes per while loop iteration, BUT WE ONLY WANT ONE NEW NODE PER WHILE LOOP ITERATION! More predictable.

After iterating through adj\_list and finding next 'w' node, starting from curr\_i in adj\_list:

1.  Reassign boolean var *deadend* to false, new node found.

2.  Increment curr\_i for next time when I return to node.

3.  Reassign newly discovered node to *node*var.

4.  Set node from 'w' to 'g'.

5.  Push node to stack *stk*.

6.  Add node to connected component group if the problem needs connected components.

7.  break;

For iterative merge sort, double the group size to be compared after each pass, and within each pass,

increment the i and j two pointers by 2 \* curr\_group\_size after merging each block. Try to figure out quicksort, KMP, quickselect, divide and conquer, linear regression, Tarjan's algorithm, Kosaraju's algorithm, Djikstra's algorithm, Bellman-Ford algorithm, Depth-Limited-Search (DFS and BFS hybrid), etc.

Learn recursive functions iteratively to deeply understand what is happening, and what is being updated, but in competition scenarios, it would be more concise to just implement the algorithm recursively. But in my training, implement iteratively!

Go into machine learning and computer vision a little.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Mon 1/9/2023 12:04 AM

When dealing with slopes in a 2D cartesian coordinate problem, if x-coor and y-coor are of magnitude 10^4,

then using double is fine, precise enough for LeetCode max points on a line problem. But for Olympus Mans Kattis problem with 10^9 x-coor and y-coor magnitude, even C++ long double was not precise enough. Instead, use absolute fixed precision slope expressed as a simplified ratio (if we want to hash it, convert to string form).

In stream of integers types of problems, some sort of union merging left and right sections works. When finding the maximum length of a streak of consecutive integers given in a stream (3,6,4,1,2,5,10,11,12,7 --> best streak is 1,2,3,4,5,6,7), a hashmap and keeping track of leftmost edge and rightmost edge of contiguous block is important. Each new int from the stream can either extend a block leftwards, extend a block rightwards, or merge, "glue", two blocks together.

In C++ and Java, floor division (integer division, or truncating division) is the default when dividing two ints. If we want to have the value output floating point type, switch to float division when dividing two ints by first multiplying by static\_cast<double>(1) \* integer1 / integer2; in C++,

or

double d = 1;

d \* integer1 / integer 2; in Java.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sun 1/8/2023 3:36 PM

Consecutive streak complete factorizer (factorize all numbers from 1 --> 1,000,000), consecutive streak distinct prime factors finder (do not have to find the total number of terms of factor, e.g. for 8 just find 2, instead of 2^3), streak LCM finder (LCM of all numbers 1 --> 1,000,000) using products of p^(floor(log p (1,000,000))) p means prime number, GCD finder for multiple numbers (sort and then find the smallest gap, GCD will be a factor of the value of the smallest gap, Chinese Remainder Theorem, similar to the CodeForces Koxia and Number Theory Goodbye 2022 contest). Generate all prime numbers using a sieve, Sieve of Eratosthenes, Sieve of Euler.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 1/7/2023 8:20 PM

 // can use vector to store factors in sorted order, as in

    // and then use two pointers technique to merge

    // (or find intersection for GCD or union for LCM) the factors of 2 numbers.

        int i = 0;

        int j = 0;

        vector<int> merged\_factors;

        while(i < factors[k].size() && j < factors[k + 2].size()){

            if(factors[k][i] < factors[k + 2][j]){

                merged\_factors.push\_back(factors[k][i]);

                merged\_factors.push\_back(factors[k][i + 1]);

                i += 2;

            }else if(factors[k + 2][j] < factors[k][i]){

                merged\_factors.push\_back(factors[k + 2][j]);

                merged\_factors.push\_back(factors[k + 2][j + 1]);

                j += 2;

            }else if(factors[k][i] == factors[k + 2][j]){

                merged\_factors.push\_back(factors[k][i]);

                merged\_factors.push\_back(factors[k][i + 1] + factors[k + 2][j + 1]);

                i += 2;

                j += 2;

            }

        }

        while(i < factors[k].size()){

            merged\_factors.push\_back(factors[k][i]);

            merged\_factors.push\_back(factors[k][i + 1]);

            i += 2;

        }while(j < factors[k + 2].size()){

            merged\_factors.push\_back(factors[k + 2][j]);

            merged\_factors.push\_back(factors[k + 2][j + 1]);

            j += 2;

        }



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated







To:​Shen, James​

Sat 1/7/2023 11:34 AM

// sqrt() is 4x slower than simple rapid multiplication, or continue  
                                       // statement, but still pretty fast!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 1/7/2023 11:34 AM

 // MAKE SURE YOU FUCKING PUT PARENTHESIS AROUND BITWISE OPERATIONS!  
 // Or else the equality == operator is evaluated first

  BITWISE TECHNIQUE: vector<int> positions (32, 0); // how many numbers have a high bit in the position

 // instead of jump array,

        // use prefix array to make quick sums, HUGE query in O(1) time!

        // range sum query is best done using PREFIX ARRAY!

        // Although jump array works too

// REMEMBER, min() function takes int as parameter!  
// But vector .size() returns size\_t type

// LSB in positions on left for bitwise, convenient



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 1/7/2023 2:10 AM

A computer code with white text

Description automatically generated

Be wary of the INCREMENTER of the FOR LOOP. Most of the times just a ++i, ++j, --k traditional increment/decrement by 1, but sometimes a special i += 2 if you want to have a jump gap size of 2, or even

j += k, as in the Sieve of Eratosthenes inner for loop. BE CAREFUL! Pay attention to what you are doing.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Sat 1/7/2023 1:32 AM

A screenshot of a computer program

Description automatically generated



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Fri 1/6/2023 2:27 PM

AGAIN, MAKE SURE YOU INCREMENT while loop counter i  IN EACH ITERATION! You forget to increment i TOO OFTEN. Also if early break from while loop, make sure you increment the loop counter and any other variables if you should.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Fri 1/6/2023 1:56 PM

    // C++ .substr() function is quite safe, as long as start

    // index is <= S.length(), length of acquired substring

    // will be min(second param, S.length() - first param)



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Fri 1/6/2023 12:39 AM

Always be mindful of matrix.size() vs matrix[0].size() ! I made this mistake, was controlling the j index variable for each row of the matrix, but used matrix.size() - 1 as the starting value for j, but SHOULD HAVE USED matrix[0].size() - 1 !!!!!!! Be very mindful! This question was from the Uber parkingLot question on CodeSignal company challenges! Always be mindful of for loop conditions, initializations, increment/decrement/special += k crement. (kay-crement LOL)



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 1/5/2023 11:56 PM

cout a bunch when you are debugging! Use "binary search" cout similar to what USACO Training tips say, adn cout vec.size() since seeing if the size of a vector is the culprit of the issue is useful. An error I have made is initializing a 2D vector of a nonzero size, and then using push\_back()  instead of reassigning by index operator. So the initial initialized elements are blank, and can run into segmentation faults if I mistakenly access by index within the empty 1D vector elements of the 2D vector.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 1/5/2023 11:43 PM

     // remember, in C++ there is abs and fabs,

            // atoi and atol, BE WARY OF the TYPES OF THE

            // ARGUMENTS IN FUNCTIONS!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 1/5/2023 11:18 PM

In C++, use call by reference for static bool comparator(vector<int>& a, vector<int>& b){}

to prevent deep copying over the arguments by value. Just passing in two references to the vector

objects is much faster!

SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 1/5/2023 8:43 PM

In C++ you can use address operator on a container such as an unordered\_set<int>, and you can create a pointer variable for the set! Very useful!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Thu 1/5/2023 6:36 PM

Whenever you include an early break or continue in your code, make sure that you did not forget to do anything! In my first implementation of merge sort (merge\_sort.py), I  used continue statement without first updating group\_size variable. A few lines below, I had group\_size \*= 2, but the early continue skipped those lines of code, so I only did a single iteration of merge\_sort.

SO ALWAYS CHECK IF YOU NEED TO UPDATE THE VARIABLES YOU NEED TO UPDATE BEFORE YOU DO

EARLY CONTINUE OR BREAK!

AND FOR CODING CONTESTS, MAKE SURE YOU cin OR fin ALL THE INPUT YOU NEED !!!!!!!

I'VE MADE THE MISTAKE OF EARLY BREAKING AND CONTINUING WITHOUT UPDATING VARIABLES AND GETTING ALL INPUT TOO MANY TIMES !!!!!!! BE METICULOUS !!!!!!!

// again, use the chronological events technique.

        // sort by open, close events, OPEN BEFORE CLOSE for the same x-position.

        // Improvement in the algorithm, previously, I had duplicates, and

        // gave all events with the same time (or x-position) a pair,

        // {time, bool open or close}. Now I will count frequency, should be cleaner,

        // require simpler sorting. Python default sort is increasing by both elements

        // in a 2D sort, C++ default also sorts them all, double ascending!

        // C++ default sort implementation should be much, much faster than

        // using your own custom comparator!

SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 1/3/2023 4:31 PM

Remember, sometimes, a bug at a line in code can be due to a true cause of the bug later on in the code! This is possible with memory/pointer bugs. Holistic program! Look everywhere if there is a bug, also use USACO Training's model of "binary search placement" of print statements to hone in on where exactly the bug is.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 1/3/2023 4:08 PM

BE AWARE OF POSSIBLE OVERFLOWS! If an int can overflow to a long, maybe it is multiplied by a number, and then it overflows, MAKE IT long IN THE FIRST PLACE! Also use atol (ASCII to long) instead of atoi (ASCII to int) if string digit input can overflow std::numeric\_limits<int>::max()

default magic number e.g. 212032912 representation in C++ seems to be int, so cannot multiply two large magic numbers together, there will be overflow! Initialize long variable with int magic number value, then multiply again by what value needs to be multiplied, this way there is no overflow!

In Python, I have made multiple Memory Limit Exceeded/TLE due to a huge range() in a for loop. for i in range(10\*\*9) will result in MLE in LeetCode; after all, a for loop range in Python is different from a for loop with counter variable in Java and C++!

Remember, if statements and for statements are their own scopes. In C++, curly braces designate a scope, variables initialized inside a pair of curly braces are destroyed (thus variable name can be reused) after the closing curly brace.

new C++ keyword allocates memory to the heap. otherwise, just a local variable. REMEMBER, C++ GLOBAL VARIABLES GO OUTSIDE THE MAIN FUNCTION! USUALLY GO RIGHT BELOW THE #include <header file> statements.

Pass by reference with large, complex objects, large vectors for speed increase, and also to "return" and modify multiple objects! Instead of a single return object using return statement. In Python, no pass by reference, so workaround is to return a list/tuple in a "bulk delivery" sort of form, return my\_delivery, and I specify and remember what my\_delivery[0] is, my\_delivery[1] is, my\_delivery[2] is, and then update values based on the multiple elements that are returned.

REMEMBER, Python import typing is a module that allows type hints, which may improve readability to programmers, and allow IDEs and linters more effectively analyze the code. But if programmer mistakenly declares an incorrect type hint, or declared tuple instead of list, I am not sure what the repercussions will be.

IN AN INTERVIEW, DON'T BE WISHY-WASHY, BE VERY FIRM AND CONFIDENT IN YOUR ANSWERS! Don't give general answers, go straight to the point, go in-depth, answer the question elaborately, cleanly.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 1/3/2023 3:37 PM

DON'T FORGET TO cin IN ALL THE INPUT FROM THE TEST CASE IF I PLAN ON USING AN EARLY BREAK OR CONTINUE TO ACCELERATE THE CODE! Catastrophic cin afterwards if I forget, since I would not cin as intended for the following test cases!



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 1/3/2023 3:36 PM

To speed up implementation, write out some vomit header files, commonly-used function, shortcuts, similar to Benq. The test cases, inputting should be done by the int main() function, your own function should be void source(), which should be much faster. Most codeforces questions, after all, have multiple test cases, each test case has some common inputs, n elements in array, then n following elements....

A good trick is to use tourist's bool ok variable. My variable name of choices are: bool status, bool is\_valid, bool is\_prime, bool is\_<some description>, bool ok. Adopt bool ok!

Remember, to reduce runtime, memoize things, also precompute things. e.g compute all primes once using sieve from k = 2 --> limit + 1 (inclusive --> exclusive), then put primes in a vector, uses extra storage, but there are much fewer primes in that range than every integer in that range, so maybe a 10x increase in runtime, if the range is large! Precomputing things, prefix arrays, are good for range sum queries, range add/update queries, jump array, think of efficient block chunking data structures that reduce overall operations/improve time complexity.

Codeforces user "pajenegod" seems to be a good PyPy programmer, Grandmaster rating.

Benq and tourist submissions are probably good to look at, use C++ 17 and C++ 20 respectively, don't forget to try different compilers if TLE, borderline!

Also try to avoid hash-based solutions, STL hash function is well-known, and may be hacked in a codeforces contest. Also, unordered\_map is slow, only ~10^6 operations per second, compared with the ~10^8 vector address lookup, addition, subtraction, multiplication operations (division and mod div are a bit slower).

If there is a question that is about taking all differences between elements at distinct pairs of indices, don't just naively take all pairs of differences, even if O(N^2) seems fine. See if there is an underlying quicker way of doing things, such as in the Codeforces contest. I naively took all pairs of differences, but user tourist counted frequencies. A frequency >= 2 meant that there was a pair with difference! Much faster.



SJ

Shen, James









A black arrow on a yellow background

Description automatically generated

A blue and orange logo

Description automatically generated





To:​Shen, James​

Tue 1/3/2023 2:57 PM

REMEMBER!

If the time complexity of your solution seems good, medium test cases are accepted, but the largest/worst cases TLE, then try a different compiler. If C++ 20 TLE, then try C++ 17. This was the key for Problem D. in Hello 2023, Boris Haircut. The C++ 20 64 bit winlib compiler TLE (over 2000 ms), but when the same code is submitted using C++ 17 compiler, the solution is accepted (~1100 ms). This two-fold reduction in observed runtime is normal, as the source code is compiled differently, there may be a different constant factor in big O notation, SO BE FLEXIBLE, TRY DIFFERENT THINGS, AND DON'T SIT AROUND! Try different compilers (C++ 20, C++ 17, C++ 14). Also, even though PyPy is faster most of the time than normal Python 3.11, I have had cases in Codeforces where the normal interpreted Python 3.9 is accepted over the PyPy, TLE if the JIT compiler slows things down somehow. ALSO REMEMBER pragma is not a cure-all, sometimes it works, it reduces runtime by up to 50%, but sometimes it does nothing, or even slows things down! Keep grinding.

Attendee panel closed