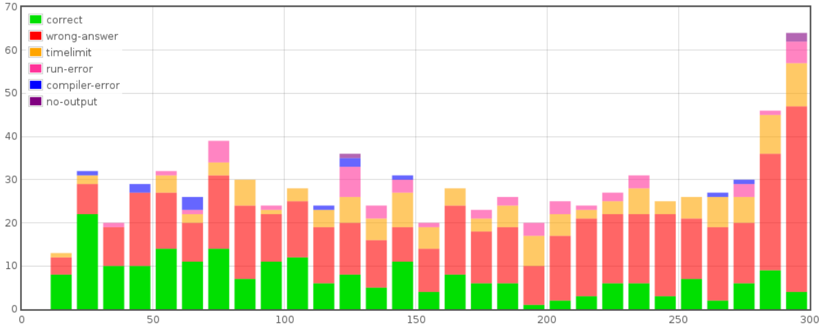
GCPC 2015
Presentation of solutions

acm International Collegiate Programming Contest

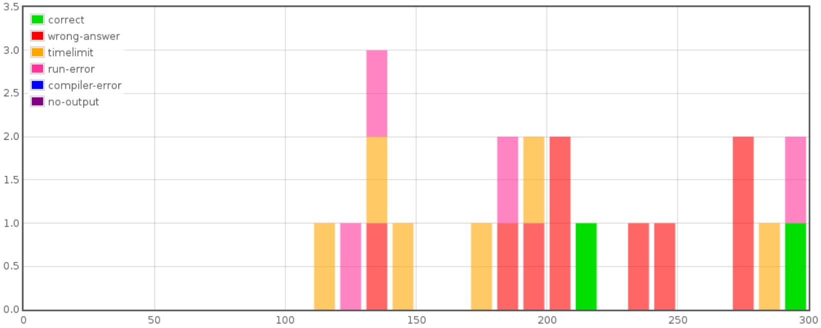
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Statistics


correct
wrong-answer
timelimit
run-error
compiler-error
no-output
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300

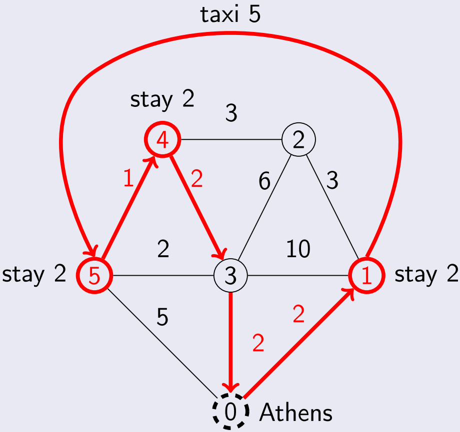
A - Greece – Statistics

Statistics: 21 submissions, 2 accepted



A - Greece

Problem



- Start in Athens (0), visit all sites and return to Athens within a given time
- Taxi ticket as a one-time short cut

⇒ Essentially TSP, but you only have to visit P nodes, all other are optional

A - Greece

Solution without taxi ticket

- Insight 1: You can always take the shortest path between any two sites $p(a, b)$
- Insight 2: TSP must only be calculated on P nodes connected by edges with $w(a, b) = |p(a, b)|$

A - Greece

Solution without taxi ticket

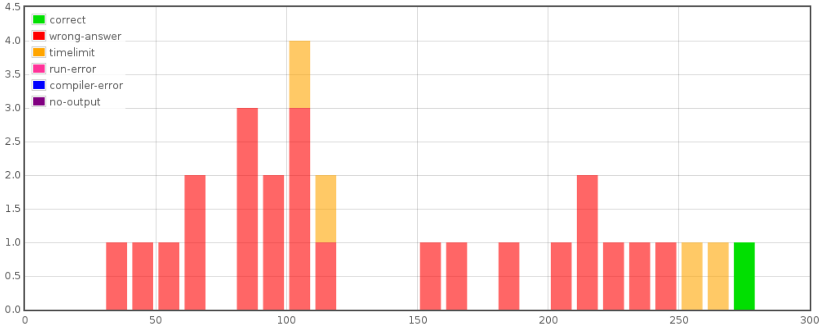
- Insight 1: You can always take the shortest path between any two sites $p(a, b)$
- Insight 2: TSP must only be calculated on P nodes connected by edges with $w(a, b) = |p(a, b)|$

Implementation

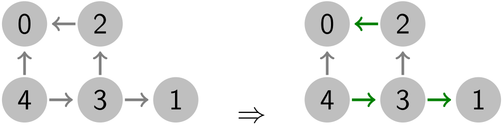
- Compute shortest paths by P-times Dijkstra: $\mathcal{O}(P \cdot N \log N)$
- Run 2^P DP solution for TSP — $P!$ will be to slow
- Add extra dimension to the DP to account for the taxi ticket
- Compare the two values with $G - \sum t_i$

B - Bounty Hunter II – Statistics

Statistics: 28 submissions, 1 accepted



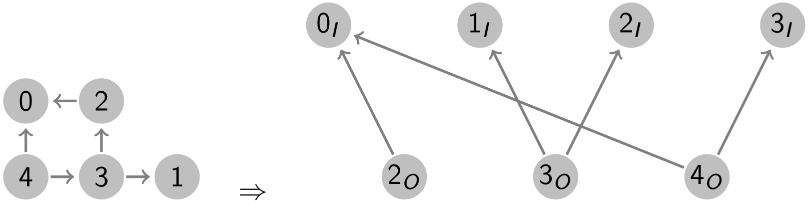
B - Bounty Hunter II



Problem

Given a DAG with N nodes find the minimum number of vertex-disjoint paths to cover each vertex.

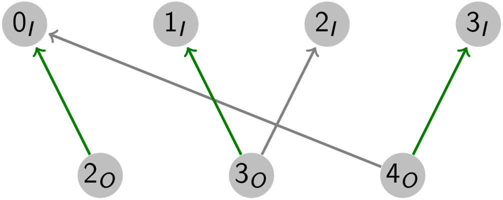
B - Bounty Hunter II



Solution

Construct bipartite graph from DAG. Set O contains all vertices with their outgoing edges, set I contains all vertices with their incoming edges.

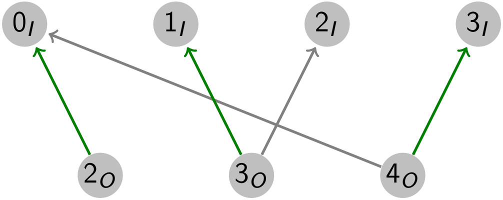
B - Bounty Hunter II



Solution

- Compute maximal matching M on bipartite graph with augmenting paths or similar algorithm.

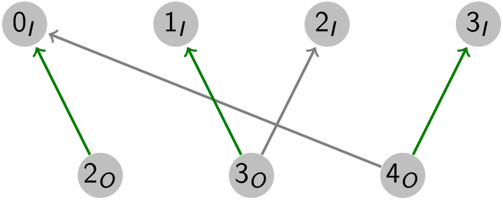
B - Bounty Hunter II



Solution

- Compute maximal matching M on bipartite graph with augmenting paths or similar algorithm.
- Idea: Start with N zero length paths in every node.

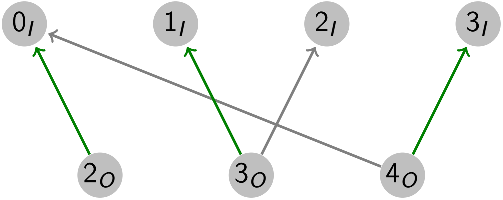
B - Bounty Hunter II



Solution

- Compute maximal matching M on bipartite graph with augmenting paths or similar algorithm.
- Idea: Start with N zero length paths in every node. Edge $(a_O, b_I) \in M \triangleq$ Path arriving at a continues to b

B - Bounty Hunter II

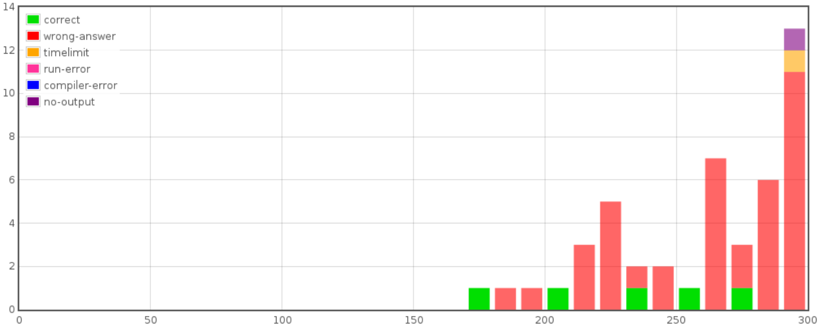


Solution

- Compute maximal matching M on bipartite graph with augmenting paths or similar algorithm.
- Idea: Start with N zero length paths in every node. Edge $(a_O, b_I) \in M \triangleq$ Path arriving at a continues to b
- Number of necessary paths is then $N - |M|$.

C - Cake – Statistics

Statistics: 46 submissions, 5 accepted



C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio a of allowed weight of the cake and an algorithm to reduce weight.

Weight reduction algorithm

- Choose a real number $s \geq 2$
- For each vertex
  - for both incident edges mark where $1/s$ of the edge's length is
  - cut directly between the two markings and remove the part with the current vertex

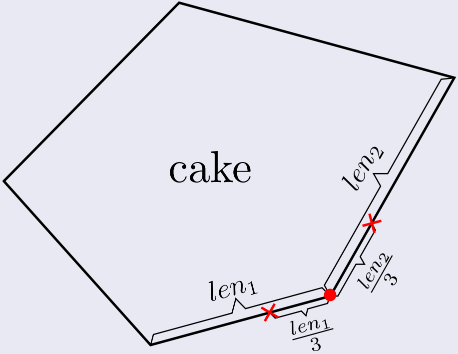
Problem

Compute the maximal s such that the area of the remaining polygon has proportion less or equal than a.

C - Cake

Weight reduction algorithm

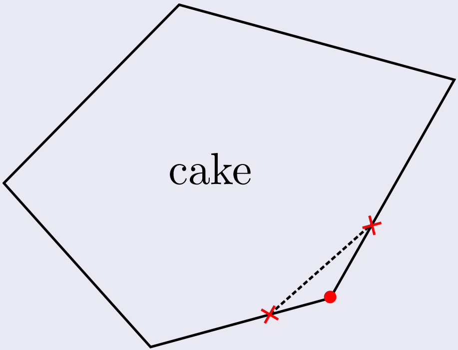
- $s := 3$
- for each vertex
  - for both incident edges mark where $1/s$ of the edge's length is
  - cut directly between the two markings and remove the part with the current vertex



C - Cake

Weight reduction algorithm

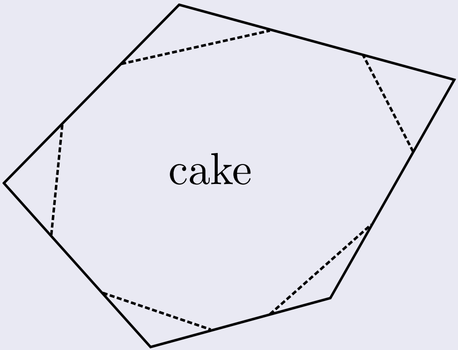
- $s := 3$
- for each vertex
  - for both incident edges mark where $1/s$ of the edge's length is
  - cut directly between the two markings and remove the part with the current vertex



C - Cake

Weight reduction algorithm

- $s := 3$
- for each vertex
  - for both incident edges mark where $1/s$ of the edge's length is
  - cut directly between the two markings and remove the part with the current vertex



C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio a of allowed weight of the cake and an algorithm to reduce weight.

Problem

Compute the maximal s such that the area of the remaining polygon has proportion less or equal than a.

(Possible) Solution

- Fix parameter s

C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio *a* of allowed weight of the cake and an algorithm to reduce weight.

Problem

Compute the maximal *s* such that the area of the remaining polygon has proportion less or equal than *a*.

(Possible) Solution

- Fix parameter *s*
- Generate reduced shape
- Calculate area of complete / reduced shape and their ratio

C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio a of allowed weight of the cake and an algorithm to reduce weight.

Problem

Compute the maximal s such that the area of the remaining polygon has proportion less or equal than a.

(Possible) Solution

- Fix parameter s
- Generate reduced shape
- Calculate area of complete / reduced shape and their ratio

The ratio depends on s in a strictly increasing manner.
$\Rightarrow$ Binary search is possible.

C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio a of allowed weight of the cake and an algorithm to reduce weight.

Problem

Compute the maximal s such that the area of the remaining polygon has proportion less or equal than a.

(Possible) Solution

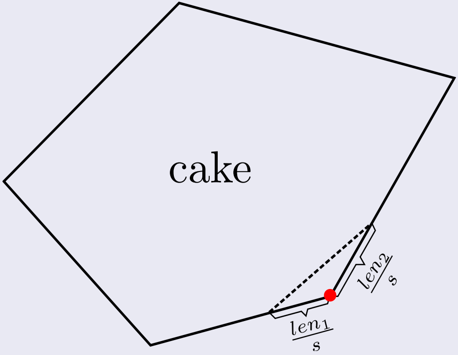
- Fix parameter s
- Generate reduced shape
- Calculate area of complete / reduced shape and their ratio

The ratio depends on s in a strictly increasing manner.
$\Rightarrow$ Binary search is possible.
WARING: Precision is a huge issue!

C - Cake

Insight

The removed area grows proportional with $\frac{1}{s^2}$.



C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio a of allowed weight of the cake and an algorithm to reduce weight.

Problem

Compute the maximal s such that the area of the remaining polygon has proportion less or equal than a.

Better Solution

- The removed area grows proportional with $\frac{1}{s^2}$.
- Compute the reduced area for some value of s and scale.

C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio *a* of allowed weight of the cake and an algorithm to reduce weight.

Problem

Compute the maximal s such that the area of the remaining polygon has proportion less or equal than a.

Better Solution

- The removed area grows proportional with $\frac{1}{s^2}$.
- Compute the reduced area for some value of s and scale.
- Use $s = 2$ (we call the reduced area $A_{s=2}$).
- Use only 64 bit integers to avoid precision issues.

C - Cake

Input

Given a convex polygon (the surface of a cake), a ratio a of allowed weight of the cake and an algorithm to reduce weight.

Problem

Compute the maximal s such that the area of the remaining polygon has proportion less or equal than a.

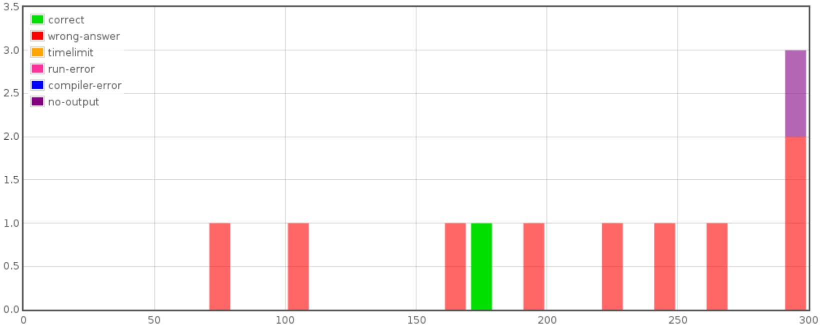
Better Solution

- The removed area grows proportional with $\frac{1}{s^2}$.
- Compute the reduced area for some value of s and scale.
- Use $s = 2$ (we call the reduced area $A_{s=2}$).
- Use only 64 bit integers to avoid precision issues.

$(A_{full} - A_s) = (A_{full} - A_{s=2}) \cdot 2^2/s^2$ and
$s_{max} = 2 \cdot \sqrt{(A_{full} - A_{s=2})/(A_{full} \cdot (1 - a))}$

D - Carpets – Statistics

Statistics: 11 submissions, 1 accepted



D - Carpets

Problem

Decide whether a rectangular room can be covered by a given set of smaller rectangular carpets (count ≤ 7).

D - Carpets

Problem

Decide whether a rectangular room can be covered by a given set of smaller rectangular carpets (count ≤ 7).

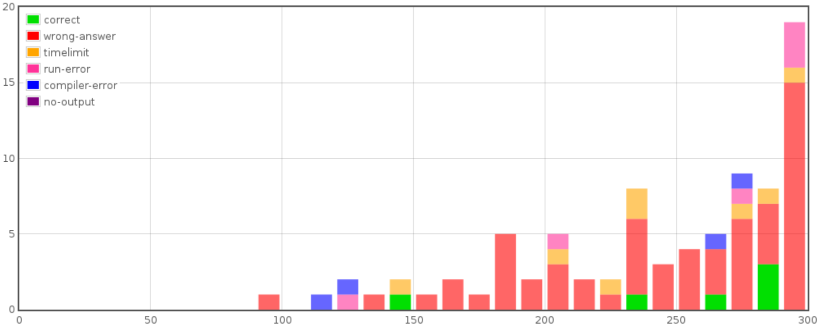
Solution: Backtracking + obvious optimizations

Try filling a 2D-array of booleans representing the room:

1. Find the topmost row with free cells and pick the leftmost cell
2. If no cell free any more, return "`yes`"
3. For any carpet in stock & rotation and fitting at the given cell:
    1. Put the carpet
    2. If recursive call to (1) successful, return "`yes`"
    3. Put the carpet back to stock
4. If all available carpets tried, return "`no`"

E - Change of Scenery – Statistics

Statistics: 79 submissions, 6 accepted



E - Change of Scenery

Problem

Given a shortest path between node S and T in a graph.
Is there is a different path of the same length between S and T?

E - Change of Scenery

Problem

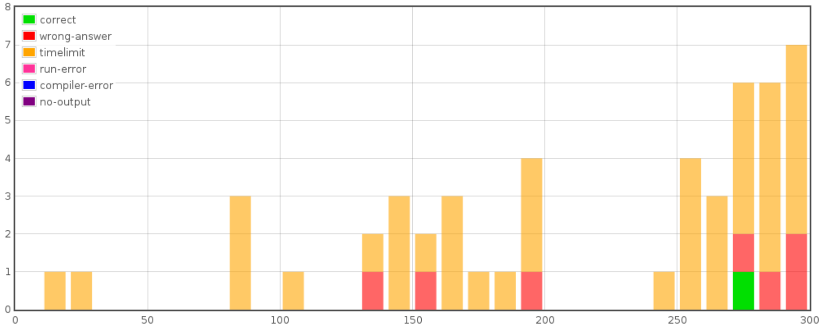
Given a shortest path between node S and T in a graph.
Is there is a different path of the same length between S and T?

Solution

- Dijkstra with minor adjustments.
- Keep track of the set of nodes N that can be reached via multiple shortest paths.
- Add a node to N if
  - you reach it a second time without updating OR
  - you update it from a node in N.
- Don't forget to remove it from N if you update it from a node not in N!
- Finally, report whether the target is in N.

F - Divisions – Statistics

Statistics: 49 submissions, 1 accepted



F - Divisions

Problem

Given: a positive integer N, $(1 \leq N \leq 10^{18})$
Output the number of positive integral divisors D_N of N.

F - Divisions

Problem

Given: a positive integer N, $(1 \leq N \leq 10^{18})$
Output the number of positive integral divisors D_N of N.

(Naive) Solution: Standard factorization in $O(\sqrt{N})$

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, e.g. $288 = 2^5 * 3^2$
where p_i are the prime factors of N.
$\Longrightarrow D_N = \prod_{i=1}^{k}(n_i + 1)$, e.g. $D_{288} = (5 + 1) \cdot (2 + 1) = 18$.

F - Divisions

Problem

Given: a positive integer N, $(1 \leq N \leq 10^{18})$
Output the number of positive integral divisors D_N of N.

(Naive) Solution: Standard factorization in $O(\sqrt{N})$

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, e.g. $288 = 2^5 * 3^2$
where p_i are the prime factors of N.
$\implies D_N = \prod_{i=1}^{k}(n_i + 1)$, e.g. $D_{288} = (5+1) \cdot (2+1) = 18$.

Insight

- Do factorization only for prime factors up to $\sqrt[3]{N}$.
- $N = C \cdot \prod_{i=1}^{m} p_i^{n_i}$, where $p_i \leq \sqrt[3]{N}$ are prime factors of N.
- C contains no divisor less than $\sqrt[3]{N}$.

F - Divisions

Solution

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, then result is $\prod_{i=1}^{k}(n_i + 1)$.

- $N = C \cdot \prod_{i=1}^{m} p_i^{n_i}$, where C contains no prime factor $\leq \sqrt[3]{N}$.

F - Divisions

Solution

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, then result is $\prod_{i=1}^{k}(n_i + 1)$.

- $N = C \cdot \prod_{i=1}^{m} p_i^{n_i}$, where C contains no prime factor $\leq \sqrt[3]{N}$.
- Insight: C is either 1, prime, or the product of two primes.

F - Divisions

Solution

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, then result is $\prod_{i=1}^{k}(n_i + 1)$.

- $N = C \cdot \prod_{i=1}^{m} p_i^{n_i}$, where C contains no prime factor $\leq \sqrt[3]{N}$.
- Insight: C is either 1, prime, or the product of two primes. This results in the following few cases:
  - $C = 1$: output $\prod_{i=1}^{m}(n_i + 1)$, check in $O(1)$.

F - Divisions

Solution

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, then result is $\prod_{i=1}^{k}(n_i + 1)$.

- $N = C \cdot \prod_{i=1}^{m} p_i^{n_i}$, where C contains no prime factor $\leq \sqrt[3]{N}$.
- Insight: C is either 1, prime, or the product of two primes. This results in the following few cases:
  - $C = 1$: output $\prod_{i=1}^{m}(n_i + 1)$, check in $O(1)$.
  - C is prime: output $2 \cdot \prod_{i=1}^{m}(n_i + 1)$, check in $O(\log N)$.

F - Divisions

Solution

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, then result is $\prod_{i=1}^{k}(n_i + 1)$.

- $N = C \cdot \prod_{i=1}^{m} p_i^{n_i}$, where C contains no prime factor $\leq \sqrt[3]{N}$.
- Insight: C is either 1, prime, or the product of two primes. This results in the following few cases:
  - $C = 1$: output $\prod_{i=1}^{m}(n_i + 1)$, check in $O(1)$.
  - C is prime: output $2 \cdot \prod_{i=1}^{m}(n_i + 1)$, check in $O(\log N)$.
  - C is a product of two equal primes / square: output $3 \cdot \prod_{i=1}^{m}(n_i + 1)$, check in $O(\log N)$, e.g. use sqrt.

F - Divisions

Solution

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, then result is $\prod_{i=1}^{k}(n_i + 1)$.

- $N = C \cdot \prod_{i=1}^{m} p_i^{n_i}$, where C contains no prime factor $\leq \sqrt[3]{N}$.
- Insight: C is either 1, prime, or the product of two primes. This results in the following few cases:
  - $C = 1$: output $\prod_{i=1}^{m}(n_i + 1)$, check in $O(1)$.
  - C is prime: output $2 \cdot \prod_{i=1}^{m}(n_i + 1)$, check in $O(\log N)$.
  - C is a product of two equal primes / square: output $3 \cdot \prod_{i=1}^{m}(n_i + 1)$, check in $O(\log N)$, e.g. use sqrt.
  - C is a product of two different primes: output $2 \cdot 2 \cdot \prod_{i=1}^{m}(n_i + 1)$, no further check necessary.

F - Divisions

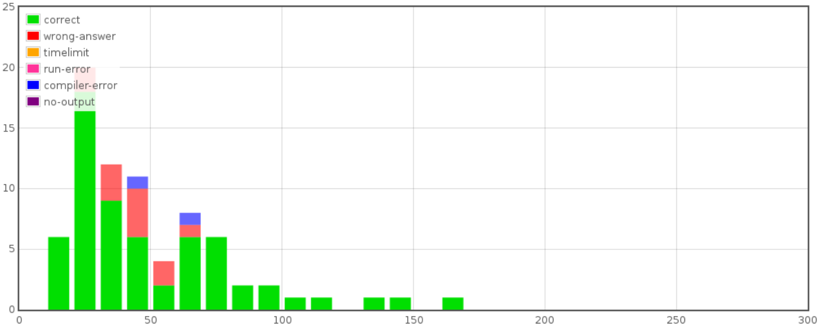
Solution

Let $N = \prod_{i=1}^{k} p_i^{n_i}$, then result is $\prod_{i=1}^{k}(n_i + 1)$.
Alternative solution:

- Use pollard ρ algorithm for factorization.
- Beware of overflows.

G - Extreme Sort – Statistics

Statistics: 74 submissions, 62 accepted



G - Extreme Sort

Problem

Check whether input sequence is correctly sorted in ascending order.

G - Extreme Sort

Problem

Check whether input sequence is correctly sorted in ascending order.

Solution 1

- Check whether $x_i \leq x_{i+1}$ for all i.
- $\Rightarrow \mathcal{O}(n)$

G - Extreme Sort

Problem

Check whether input sequence is correctly sorted in ascending order.

Solution 1

- Check whether $x_i \leq x_{i+1}$ for all i.
- $\Rightarrow \mathcal{O}(n)$

Solution 2

- Don't think, just calculate the *extreme property*, i.e. calculate all $x_{i,j}$ and print "`no`" if any of those is less than 0, otherwise print "`yes`".
- $\Rightarrow \mathcal{O}(n^2)$ - fast enough.

G - Extreme Sort

Solution 3

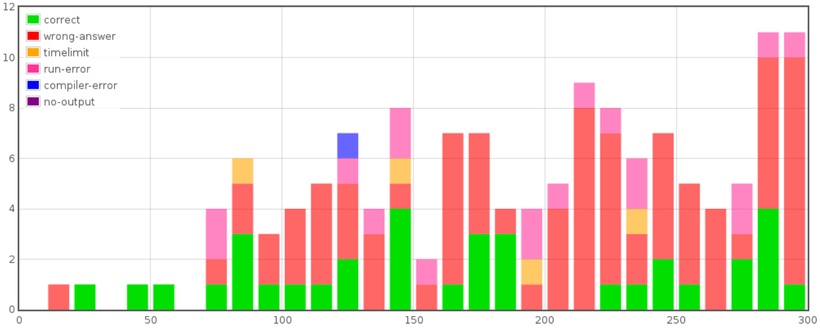
- Copy the input sequence, sort it, compare to original.
- Python:
```
print("yes" if data == sorted(data) else "no")
```
- $\Rightarrow \mathcal{O}(n \log n)$

Solution 4

- Know your standard library.
- C++:
```
std::cout <<
(std::prev_permutation(begin(data), end(data))
? "no" : "yes") << std::endl;
```
- $\Rightarrow \mathcal{O}(n)$

H - Legacy Code – Statistics

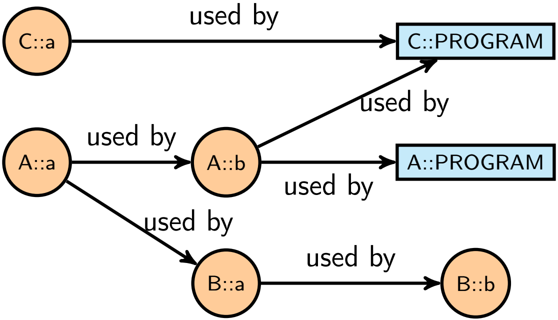
Statistics: 138 submissions, 34 accepted



Legacy Code

Problem

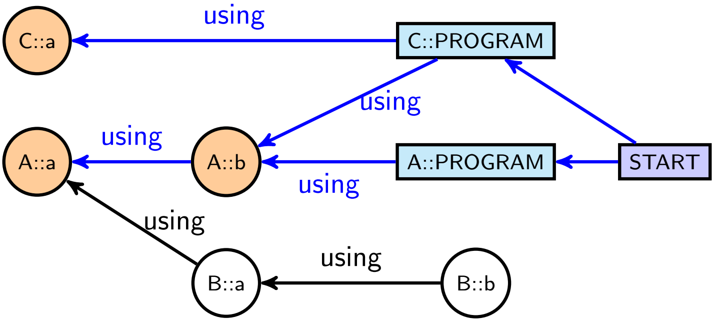
For every method in a program all callers are given.
Find the number of unused methods no matter which program is run.



Legacy Code

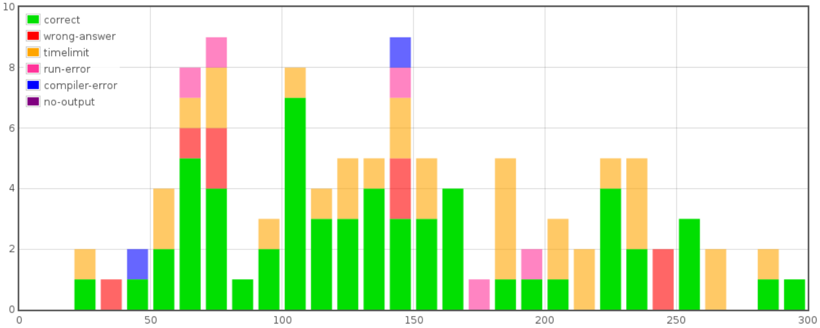
Solution

- Transpose directions in the “used by”-graph to a “using”-graph.
- Explore the resulting graph with help of your favorite algorithm (bfs, dfs) choosing `XXX::PROGRAM` as starting nodes.
- Count unvisited nodes.



I - Milling Machines – Statistics

Statistics: 100 submissions, 57 accepted



I - Milling Machines

Problem

Given a large number of work pieces and a large number of milling steps.

I - Milling Machines

Problem

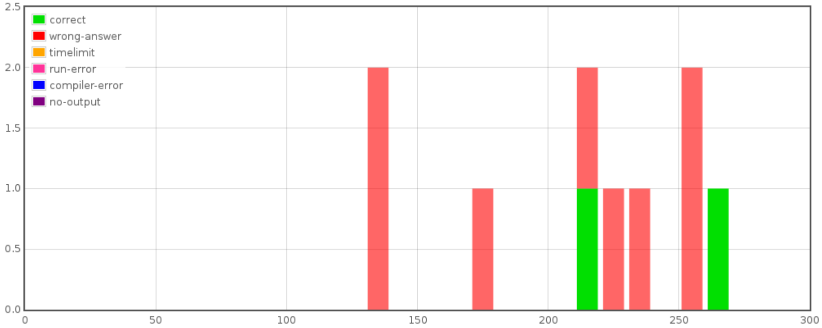
Given a large number of work pieces and a large number of milling steps.

Solution

- Naive solution is way too slow ($10^4 \cdot 10^4 \cdot 100^2 = 10^{12}$).
- Insight: Milling steps may be combined using a maximum operation.
- $\Rightarrow$ Combine milling steps and apply combined step on every workpiece.
- Reduces complexity to $10^4 \cdot 100^2 = 10^8$.

J - Souvenirs – Statistics

Statistics: 10 submissions, 2 accepted



J - Souvenirs

Problem

Two type of coins. Buy as many souvenirs as possible. Merchants have different prices, different methods of rounding and different values to round towards.

J - Souvenirs

Problem

Two type of coins. Buy as many souvenirs as possible. Merchants have different prices, different methods of rounding and different values to round towards.

Solution

```
function buy(gold, silver, m)







 return best
```

J - Souvenirs

Problem

Two type of coins. Buy as many souvenirs as possible. Merchants have different prices, different methods of rounding and different values to round towards.

Solution

```
function buy(gold, silver, m)



 best = buy(gold, silver, m+1) //don't buy





 return best
```

J - Souvenirs

Problem

Two type of coins. Buy as many souvenirs as possible. Merchants have different prices, different methods of rounding and different values to round towards.

Solution

```
function buy(gold, silver, m)



 best = buy(gold, silver, m+1) //don't buy
 if(silver >= price[m]) //buy with silver
   best = max(best, 1+buy(gold, silver-price[m], m+1))




 return best
```

J - Souvenirs

Problem

Two type of coins. Buy as many souvenirs as possible. Merchants have different prices, different methods of rounding and different values to round towards.

Solution

```
function buy(gold, silver, m)



 best = buy(gold, silver, m+1) //don't buy
 if(silver >= price[m]) //buy with silver
   best = max(best, 1+buy(gold, silver-price[m], m+1))
 (else) if(gold >= 1) //buy with gold
   ret = roundCorrectly(goldInSilver - price[m])
   best = max(best, 1+buy(gold-1, silver + ret, m+1))
 return best
```

J - Souvenirs

Problem

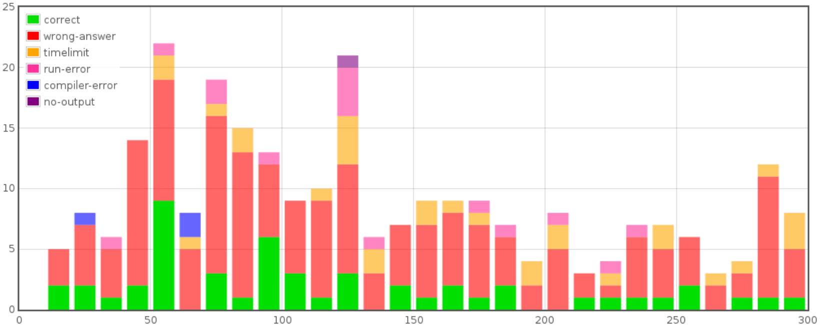
Two type of coins. Buy as many souvenirs as possible. Merchants have different prices, different methods of rounding and different values to round towards.

Solution

```
function buy(gold, silver, m)
 if(dp[gold][silver][m] != UNDEF)
   return dp[gold][silver][m]
 best = buy(gold, silver, m+1) //don't buy
 if(silver >= price[m]) //buy with silver
   best = max(best, 1+buy(gold, silver-price[m], m+1))
 (else) if(gold >= 1) //buy with gold
   ret = roundCorrectly(goldInSilver - price[m])
   best = max(best, 1+buy(gold-1, silver + ret, m+1))
 return best = dp[gold][silver][m]
```

K - Upside Down Primes – Statistics

Statistics: 260 submissions, 50 accepted



K - Upside down primes

Problem

Given an integer N, check if N is prime and still a prime after N is rotated by 180 degrees.

K - Upside down primes

Problem

Given an integer N, check if N is prime and still a prime after N is rotated by 180 degrees.

Solution

- `no` if N contains 3, 4 or 7
- `no` if N is composite (use standard primality test)
- `no` if rotated N is composite (use standard primality test)
- otherwise: `yes`

K - Upside down primes

Problem

Given an integer N, check if N is prime and still a prime after N is rotated by 180 degrees.

Solution

- `no` if N contains 3, 4 or 7
- `no` if N is composite (use standard primality test)
- `no` if rotated N is composite (use standard primality test)
- otherwise: `yes`

Common mistakes

- 1 is not prime, 2 is prime (so are 3 and 5)
- replace 6 to 9 and vice versa in parallel!
- square root might be a prime factor
- use 64 bit ints all the way