2016 ACM-ICPC North America Qualification Contest Solution Outlines

The Judges

September 24, 2016

North America Qualifier 2016
acm International Collegiate Programming Contest

IBM | event sponsor

A – A New Alphabet – First solved at 0:03:08

Problem

Translate a string from one alphabet to another.

A – A New Alphabet – First solved at 0:03:08

Problem

Translate a string from one alphabet to another.

Solution

- Manually convert the translation table to code. E.g. a function that takes a single character, and returns the translated character by
  - checking a sequence of `if` statements,
  - using one large `switch` statement, or
  - querying a `map` data structure.
- Uppercase letters should be translated to the same character as their lowercase counterparts. This can be handled by
  - creating a second translation table for uppercase letters, or
  - converting uppercase letters to lowercase letters by using a library function, or by manipulating their underlying ASCII values.
- Finally a single pass through the text, outputting the translated characters.

J – Quick Estimates – First solved at 0:04:33

Problem

Given a large integer, output the number of digits required to represent it (in base 10).

J – Quick Estimates – First solved at 0:04:33

Problem

Given a large integer, output the number of digits required to represent it (in base 10).

Solution

- The given integers will not fit in 32- or 64-bit integer types.
- Instead, treat the integers as strings.
- The answer is simply the length of each string.

- Easiest problem in the problem set.

C – Big Truck – First solved at 0:09:18

Problem

Out of all shortest paths from vertex 1 to vertex n, find the path that maximizes the sum of values of visited vertices.

C – Big Truck – First solved at 0:09:18

Problem

Out of all shortest paths from vertex 1 to vertex n, find the path that maximizes the sum of values of visited vertices.

Solution

Two solution approaches:

1. Dijkstra's with an appropriate order on vertices
2. Longest path in a directed acyclic graph

C – Big Truck – First solved at 0:09:18

Approach 1: Dijkstra's

- All edge weights are positive, so Dijkstra's algorithm will easily find shortest paths. The graph is small enough that sparse or dense Dijkstra will both work.
- Whenever two paths reach vertex v, Dijkstra's prefers the one that has the least cost.
- For this problem, if two paths reach v with the same minimum cost, then we break the tie with the path which has picked up the most items thus far.
- For each path, keep two pieces of information: path cost c and number of items picked up i.
- Order the priority queue with respect to the tuple $(c, -i)$.
- Still standard Dijkstra; runs in $O(V^2)$ or $O(E \log V)$.

C – Big Truck – First solved at 0:09:18

Approach 2: Longest Path in a DAG

- We can simplify the original graph by turning it into a DAG containing only edges on some shortest path from node 1 to n.
- Edge (a, b) with weight d is in this DAG iff.

$$\mathrm{dist}(1, a) + d + \mathrm{dist}(b, n) = \mathrm{dist}(1, n)$$

- This yields a DAG, since if edge (a, b) is chosen, then b is strictly closer to n than a. So, there can't be any cycles.
- The problem reduces to finding the longest path in this DAG, with length defined by the number of items at each vertex. This can be solved in linear time using dynamic programming.

H – Nine Packs – First solved at 0:12:02

Problem

Buy the minimum number of packs of buns and hotdogs, so that there are equally many hotdogs and buns.

H – Nine Packs – First solved at 0:12:02

Problem

Buy the minimum number of packs of buns and hotdogs, so that there are equally many hotdogs and buns.

Solution

- Considering only hotdog packs $h_1, \ldots, h_i$, let $\mathrm{opt}_h(i,k)$ denote the minimum number of packs you need to buy to have exactly k hotdogs, or $+\infty$ if impossible.
- Similarly, let $\mathrm{opt}_b(i,k)$ denote the corresponding value for the packs of buns $b_1, \ldots, b_i$.
- If $C = \min\left(\sum_{i=1}^{H} h_i, \sum_{i=1}^{B} b_i\right)$, the answer is

$$\min_{1 \le k \le C} \mathrm{opt}_b(B,k) + \mathrm{opt}_h(H,k)$$

- Note that $C \le 100 \cdot 1\,000 = 10^5$.

H – Nine Packs – First solved at 0:12:02

Solution

- Computing $\mathrm{opt}_h(i,k)$ is an instance of the Knapsack problem (or a variant thereof), and can be solved in a similar manner as follows.
- Based on whether we buy the i-th pack of hotdogs (h_i) or not, we get the following recurrence:

$$\mathrm{opt}_h(i,k) = \min(1 + \mathrm{opt}_h(i-1, k-h_i), \mathrm{opt}_h(i-1,k))$$

- We also have the base cases:

$$\mathrm{opt}_h(i,k) = \begin{cases} 0 & \text{if } k = 0 \\ +\infty & \text{if } i = 0 \text{ or } k < 0 \end{cases}$$

- We have a similar recursion for $\mathrm{opt}_b(i,k)$.
- Finally, $\mathrm{opt}_h(H,k)$ and $\mathrm{opt}_b(B,k)$ can be computed for all $1 \le k \le C$ in $O((B+H)C)$ using dynamic programming (or memoization).

G – Inverse Factorial – First solved at 0:14:40

Problem

Given $n!$, find n.

G – Inverse Factorial – First solved at 0:14:40

Problem

Given $n!$, find n.

Solution

- It is guaranteed that the number of digits of $n!$ is at most 10^6. This implies that $n < 3 \cdot 10^5$ (found by experimentation).
- A naive approach computing factorials will be too slow, due to the overhead of big integer arithmetic.
- Instead we can notice that, when $n \geq 10$, each factorial can be uniquely identified by its length (i.e. number of digits).
- The length of an integer x can be computed as $\lfloor \log_{10}(x) \rfloor + 1$.

G – Inverse Factorial – First solved at 0:14:40

Solution

- Let
$$L_k := \log_{10}(k!) = \log_{10}\left(\prod_{i=1}^{k} i\right) = \sum_{i=1}^{k} \log_{10}(i)$$
- Then the length of $k!$ is $\lfloor L_k \rfloor + 1$.
- Using the fact that $L_{k+1} = L_k + \log_{10}(k+1)$, we can successively compute $L_1, L_2, \ldots$, until we find the factorial with the required length.
- Each step takes $O(1)$ time, and the answer will be found in at most $3 \cdot 10^5$ steps.
- Handle $n < 10$ as special cases. As $0! = 1! = 1$, make sure you output 1 when the input is 1 (the output should be positive).

K – Robotopia – First solved at 0:26:54

Problem

There are two types of robots, each with a specific number of arms and legs. Given the total number of arms and legs, determine the number of robots of each type.

K – Robotopia – First solved at 0:26:54

Problem

There are two types of robots, each with a specific number of arms and legs. Given the total number of arms and legs, determine the number of robots of each type.

Solution

- We're looking for positive integers x, y such that
$$l_1 x + l_2 y = l_t$$
$$a_1 x + a_2 y = a_t$$
- This is a system of linear equations, but we want integer solutions.
- The constraints are small enough that complete search works:
  - Loop through all values $x = 1, 2, \dots$ while $l_1 x < l_t$.
  - Solve for the other variable: $y = (l_t - l_1 x)/l_2$.
  - Check if x, y are positive integers that satisfy the system of equations.
- Output '?' if zero or multiple solutions were found.

D – Brackets – First solved at 0:40:42

Problem

Given a sequence of opening and closing brackets, determine if it's possible to invert at most one contiguous subsequence of the brackets, so that the brackets are correctly nested in the resulting sequence.

D – Brackets – First solved at 0:40:42

Problem

Given a sequence of opening and closing brackets, determine if it's possible to invert at most one contiguous subsequence of the brackets, so that the brackets are correctly nested in the resulting sequence.

Solution

- Let $s_1, \ldots, s_n$ denote the sequence of brackets.
- An alternative way to characterize a valid bracket sequence:
  1. For each prefix of the sequence, there are at least as many left brackets as there are right brackets, and
  2. there are equally many left and right brackets in the entire sequence.

D – Brackets – First solved at 0:40:42

Solution

- Basic idea:
  - Step through the sequence from left to right, keeping track of the number of left and right brackets.
  - At some point start inverting brackets. At some later point, stop inverting brackets.
  - At each step, make sure there are at least as many left brackets as there are right brackets (condition 1).
  - When at the end, make sure there are equally many left and right brackets (condition 2).
- How to decide when to start/stop inverting? Dynamic programming.

D – Brackets – First solved at 0:40:42

Solution

- Let $\mathrm{dp}(i, l, t)$ be `true` iff. it's possible to step through $s_i, \ldots, s_n$ such that conditions 1 and 2 are satisfied, assuming that l is the number of left brackets we've seen so far, and
  - $t =$ `Before` if we have not started inverting brackets yet,
  - $t =$ `Invert` if we are currently inverting brackets, and
  - $t =$ `After` if we have stopped inverting brackets.
- Note that $i - 1 - l$ is the number of right brackets we've seen so far.
- Base case: $\mathrm{dp}(n + 1, l, t) =$ `true` iff. $l = (n + 1) - 1 - l$

D – Brackets – First solved at 0:40:42

Solution

- Recurrence:

$$\mathrm{dp}(i,l,t)=\begin{cases}\texttt{false} & \text{if } l<i-1-l\\[1em] \quad(t=\texttt{Before}\wedge \mathrm{dp}(i,l,\texttt{Invert}))\\ \vee\,(t=\texttt{Invert}\wedge \mathrm{dp}(i,l,\texttt{After})) & \text{otherwise}\\ \vee\,\mathrm{dp}(i+1,l+L(i,t),t)\end{cases}$$

  where

$$\mathrm{L}(i,t)=\begin{cases}1 & \text{if } s_i=\text{`('} \text{ and } t\neq\texttt{Invert}\\ 1 & \text{if } s_i=\text{`)'} \text{ and } t=\texttt{Invert}\\ 0 & \text{otherwise}\end{cases}$$

- Answer is $\mathrm{dp}(1,0,\texttt{Before})$.
- Can be computed in $O(n^2)$ using dynamic programming.

B – Arcade – First solved at 1:03:55

Problem

Compute the expected payout from an arcade game.

B – Arcade – First solved at 1:03:55

Problem

Compute the expected payout from an arcade game.

Solution

- This (zero-player) game is an example of a Markov chain.
- For background, see Grinstead and Snell, Ch. 11 [1, 2].
- Three solution approaches:
  1. Compute expected values via system of equations
  2. Compute absorption probabilities
  3. Simulate Markov chain for finite number of steps

B – Arcade – First solved at 1:03:55

Approach 1: Directly Computing Expected Values

- Let E_i denote the expected payout if the ball is currently at hole i, and let c_i denote the payout for dropping into hole i. Then

  $$E_i = p_4 c_i + p_0 E_a + p_1 E_b + p_2 E_c + p_3 E_d$$

  where a, b, c, d are the neighbors of hole i.
- If $H = N(N+1)/2$ is the number of holes, this gives a system of H linear equations in H variables, which can be solved using Gaussian elimination in $O(H^3)$.
- The input constraints imply that the system is consistent and has a unique solution.
- Can be derived without knowing Markov chain theory.

B – Arcade – First solved at 1:03:55

Approach 2: Computing Absorption Probabilities

- Uses Markov chain theory directly.
- Model each hole as a transient state, model falling into a hole as absorbing state. Have $H = N(N+1)/2$ transient and H absorbing states.
- Build $H \times H$ transition matrix $\mathbf{Q}$, use Gaussian elimination to compute fundamental matrix $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$.
- Compute absorption probabilities $\mathbf{B} = \mathbf{NR}$ where $\mathbf{R}$ is the $H \times H$ diagonal matrix $\mathbf{R}_{i,j}$ of being absorbed (falling into the hole) j when hovering over hole i.
- Compute expected value as dot product of first row of $\mathbf{B}$ with expected payout. (Optimize to compute only first row of $\mathbf{B}$.)

B – Arcade – First solved at 1:03:55

Approach 3: Simulation

- Keep track of probability of being over hole i after k steps: vector of $p_{k,i}$ elements.
- For all i, compute $p_{k+1,i}$ from $p_{k,i}$ by simulating the next event of ball bouncing from hole i to possible neighbors.
- Compute contribution to expected payout based on probability of falling into hole i at step k.
- Stop when probability of not having been absorbed is small enough.
- Caveat: if probabilities of falling into holes are very small, this could take many iterations: exploit problem constraint that the probability of not having been absorbed drops below threshold.
- Mathematically equivalent to computing first row of $\mathbf{NR} = (\mathbf{I} - \mathbf{Q})^{-1}\mathbf{R} = (\mathbf{I} + \mathbf{Q} + \mathbf{Q}^2 + \mathbf{Q}^3 + \cdots)\mathbf{R}$ — but will time out if done using dense matrix!

L – Unusual Darts – First solved at 1:24:48

Problem

Find the order in which Alice's darts were thrown so that the probability that Bob's three darts land within her heptagon is the probability given.

L – Unusual Darts – First solved at 1:24:48

Problem

Find the order in which Alice's darts were thrown so that the probability that Bob's three darts land within her heptagon is the probability given.

Solution

- Loop through all permutations of the seven points.
- (Optimize) Restrict w.l.o.g. to permutations that start with "1".
- For each permutation, check that it forms a simple polygon. If not, discard it.
- If simple, compute the area, K, and the probability $p = \left(\frac{K}{2^2}\right)^3$
- If generating permutations in lex. order, print the first one that works. Otherwise, select the lex'ly least one from among the 14 possible orderings of that heptagon.

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

- Start with the permutation 1 2 3 4 5 6 7

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

• Start with the permutation	1 2 3 4 5 6 7
• Iterate: given a permutation	1 2 6 4 7 5 3

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

• Start with the permutation	1 2 3 4 5 6 7
• Iterate: given a permutation	1 2 6 4 7 5 3
• Find the maximal-length decreasing suffix	1 2 6 4 7 5 3

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

• Start with the permutation	1 2 3 4 5 6 7
• Iterate: given a permutation	1 2 6 4 7 5 3
• Find the maximal-length decreasing suffix	1 2 6 4 7 5 3
• Swap the prior term with the first subsequent term that's greater	1 2 6 4 7 5 3

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

- Start with the permutation
- Iterate: given a permutation
- Find the maximal-length decreasing suffix
- Swap the prior term with the first subsequent term that's greater

```
1 2 3 4 5 6 7
1 2 6 4 7 5 3
1 2 6 4 7 5 3
1 2 6 4 7 5 3
1 2 6 5 7 4 3
```

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

- Start with the permutation
- Iterate: given a permutation
- Find the maximal-length decreasing suffix
- Swap the prior term with the first subsequent term that's greater

1 2 3 4 5 6 7

1 2 6 4 7 5 3

1 2 6 4 7 5 3

1 2 6 4 7 5 3

1 2 6 5 7 4 3

1 2 6 5 7 4 3

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

- Start with the permutation
- Iterate: given a permutation
- Find the maximal-length decreasing suffix
- Swap the prior term with the first subsequent term that's greater
- Reverse the suffix

```
1 2 3 4 5 6 7
1 2 6 4 7 5 3
1 2 6 4 7 5 3
1 2 6 4 7 5 3
1 2 6 5 7 4 3
1 2 6 5 7 4 3
1 2 6 5 3 4 7
```

L – Unusual Darts – First solved at 1:24:48

Looping through all Permutations

• Start with the permutation	1 2 3 4 5 6 7
• Iterate: given a permutation	1 2 6 4 7 5 3
• Find the maximal-length decreasing suffix	1 2 6 4 7 5 3
• Swap the prior term with the first subsequent	1 2 6 4 7 5 3
term that's greater	1 2 6 5 7 4 3
• Reverse the suffix	1 2 6 5 7 4 3
• And done	1 2 6 5 3 4 7
	1 2 6 5 3 4 7

L – Unusual Darts – First solved at 1:24:48

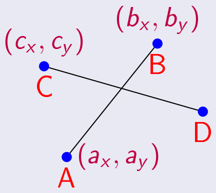
Looping through all Permutations

• Start with the permutation	1 2 3 4 5 6 7
• Iterate: given a permutation	1 2 6 4 7 5 3
• Find the maximal-length decreasing suffix	1 2 6 4 7 5 3
• Swap the prior term with the first subsequent	1 2 6 4 7 5 3
term that's greater	1 2 6 5 7 4 3
• Reverse the suffix	1 2 6 5 7 4 3
• And done	1 2 6 5 3 4 7
• Terminate after 7! (or 6!) iterations	1 2 6 5 3 4 7

L – Unusual Darts – First solved at 1:24:48

Checking to See if a Polygon is Simple

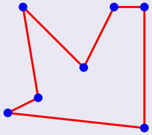
- The path $(a_x, a_y) \to (b_x, b_y) \to (c_x, c_y)$ is a left turn iff. $(b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x) > 0$.
- We use this to write `boolean left(P, Q, R)`
- $\overline{AB}$ crosses $\overline{CD}$ iff. left$(A, B, C) \neq$ left(A, B, D) and left$(C, D, A) \neq$ left(C, D, B)
- To check for simplicity, test all 14 pairs of non-adjacent edges.



L – Unusual Darts – First solved at 1:24:48

Area and Probability (by example)

- Write coordinates in order, in a column, repeating the first

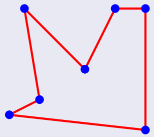


(0.0, 0.4)
(0.4, 0.6)
(0.2, 1.8)
(1.0, 1.0)
(1.4, 1.8)
(1.8, 1.8)
(1.8, 0.2)
(0.0, 0.4)

L – Unusual Darts – First solved at 1:24:48

Area and Probability (by example)

- Write coordinates in order, in a column, repeating the first
- Take 2×2 determinants

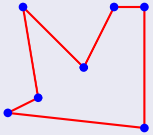


$$\begin{array}{l}
(0.0,\ 0.4) \\
(0.4,\ 0.6) \rightarrow -0.16 \\
(0.2,\ 1.8) \rightarrow +0.60 \\
(1.0,\ 1.0) \rightarrow -1.60 \\
(1.4,\ 1.8) \rightarrow +0.40 \\
(1.8,\ 1.8) \rightarrow -0.72 \\
(1.8,\ 0.2) \rightarrow -2.88 \\
(0.0,\ 0.4) \rightarrow +0.72
\end{array}$$

L – Unusual Darts – First solved at 1:24:48

Area and Probability (by example)

- Write coordinates in order, in a column, repeating the first
- Take 2×2 determinants
- Sum the determinants
- Area is half the abs. value, 1.82

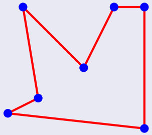


(0.0, 0.4)
(0.4, 0.6)$\rightarrow -0.16$
(0.2, 1.8)$\rightarrow +0.60$
(1.0, 1.0)$\rightarrow -1.60$
(1.4, 1.8)$\rightarrow +0.40$
(1.8, 1.8)$\rightarrow -0.72$
(1.8, 0.2)$\rightarrow -2.88$
(0.0, 0.4)$\rightarrow +0.72$
Sum = -3.64

L – Unusual Darts – First solved at 1:24:48

Area and Probability (by example)

- Write coordinates in order, in a column, repeating the first
- Take 2×2 determinants
- Sum the determinants
- Area is half the abs. value, 1.82
- Probability of each dart is 1.82/4
- So total probability $(1.82/4)^3$



(0.0, 0.4)	
(0.4, 0.6)$\rightarrow$	-0.16
(0.2, 1.8)$\rightarrow$	$+0.60$
(1.0, 1.0)$\rightarrow$	-1.60
(1.4, 1.8)$\rightarrow$	$+0.40$
(1.8, 1.8)$\rightarrow$	-0.72
(1.8, 0.2)$\rightarrow$	-2.88
(0.0, 0.4)$\rightarrow$	$+0.72$

Sum = -3.64

E – Dots and Boxes – First solved at 2:09:32

Problem

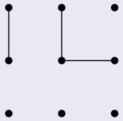
Given an $N \times N$ mesh of points with some adjacent points already joined by line segments. What is the maximum number of additional adjacent pairs of points we can join together without forming a unit square?

E – Dots and Boxes – First solved at 2:09:32

Problem

Given an $N \times N$ mesh of points with some adjacent points already joined by line segments. What is the maximum number of additional adjacent pairs of points we can join together without forming a unit square?

Solution



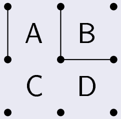
E – Dots and Boxes – First solved at 2:09:32

Problem

Given an $N \times N$ mesh of points with some adjacent points already joined by line segments. What is the maximum number of additional adjacent pairs of points we can join together without forming a unit square?

Solution

- Consider the possible unit squares. Each of them can be surrounded by at most 3 line segments, as 4 surrounding line segments form a unit square.



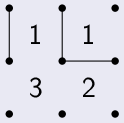
E – Dots and Boxes – First solved at 2:09:32

Problem

Given an $N \times N$ mesh of points with some adjacent points already joined by line segments. What is the maximum number of additional adjacent pairs of points we can join together without forming a unit square?

Solution

- Consider the possible unit squares. Each of them can be surrounded by at most 3 line segments, as 4 surrounding line segments form a unit square.
- Each unit square starts out with a budget of 3 allowed surrounding line segments, subtract the number of existing line segments that surround it.



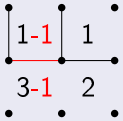
E – Dots and Boxes – First solved at 2:09:32

Problem

Given an $N \times N$ mesh of points with some adjacent points already joined by line segments. What is the maximum number of additional adjacent pairs of points we can join together without forming a unit square?

Solution

- Consider the possible unit squares. Each of them can be surrounded by at most 3 line segments, as 4 surrounding line segments form a unit square.
- Each unit square starts out with a budget of 3 allowed surrounding line segments, subtract the number of existing line segments that surround it.
- Adding a new line segment will reduce the budgets of two adjacent unit squares by one.



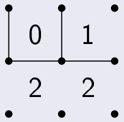
E – Dots and Boxes – First solved at 2:09:32

Problem

Given an $N \times N$ mesh of points with some adjacent points already joined by line segments. What is the maximum number of additional adjacent pairs of points we can join together without forming a unit square?

Solution

- Consider the possible unit squares. Each of them can be surrounded by at most 3 line segments, as 4 surrounding line segments form a unit square.
- Each unit square starts out with a budget of 3 allowed surrounding line segments, subtract the number of existing line segments that surround it.
- Adding a new line segment will reduce the budgets of two adjacent unit squares by one.



E – Dots and Boxes – First solved at 2:09:32

Solution

- Adding a line segment can be seen as matching the corresponding adjacent unit squares together.
- Each unit square can be matched multiple times (given by its budget), and we want the maximum matching.
- This is a standard generalization of maximum matching in a graph.
- Key observation: the graph is bipartite (think of the colors on a chessboard).

E – Dots and Boxes – First solved at 2:09:32

Solution

- This type of generalized bipartite matching can be solved using network flow. Similar to the network flow formulation of ordinary bipartite matching, but with modified capacities.
- Note: Need to add 1 to answer, as the original problem is posed slightly differently.
- Time complexity is $O(N^4)$ if Ford-Fulkerson is used.

I – Primonimo – First solved at 2:17:14

Problem

Find a sequence of moves that wins a game of Primonimo.

I – Primonimo – First solved at 2:17:14

Problem

Find a sequence of moves that wins a game of Primonimo.

Solution

- Subtracting 1 from each square allows us to think of the problem in terms of modulo-p arithmetic.
- Since addition is commutative in modulo-p arithmetic, we can see that the order in which we select squares doesn't matter.
- Let $x_{i,j}$ denote the number of times square (i,j) is selected, and let $v_{i,j}$ denote the initial value of square (i,j). Then we want

$$v_{i,j} + \left(\sum_{k=1}^{n} x_{k,j} + \sum_{k=1}^{m} x_{i,k} - x_{i,j} \right) \equiv p - 1 \pmod{p}$$

I – Primonimo – First solved at 2:17:14

Solution

- Assume for a moment that $x_{i,j} \in \mathbb{R}$ and that we wanted (for some $K \in \mathbb{R}$)
$$v_{i,j} + \left(\sum_{k=1}^{n} x_{k,j} + \sum_{k=1}^{m} x_{i,k} - x_{i,j} \right) = K$$
- Then we have a system of $n \times m$ linear equations in $n \times m$ variables, which we could solve using Gaussian elimination.
- It turns out that Gaussian elimination works over any field (which includes modulo-p arithmetic), so we can use the same method to solve our problem! Just perform all operations of the Gaussian elimination modulo p.
- Performing a division a/b modulo p is the same as multiplying a by the modular multiplicative inverse of b modulo p, which can be computed with the Euclidean algorithm.

I – Primonimo – First solved at 2:17:14

Solution

- All $x_{i,j}$ can be taken modulo p, which means that, if there is a solution, it will not use more moves than allowed by the problem statement.
- Some care has to be taken when there are multiple solutions, or no solutions, as described by the Rouché-Capelli theorem [3]. For the case of multiple solutions, it suffices to set all free variables to zero.
- Time complexity is $O(n^3 \times m^3)$.

F – Free Desserts – First solved at 4:58:33

Problem

Given a large integer P, count the number of pairs of integers a, b such that $a + b = P$, $0 < a < b$ and a, b, P do not mutually share any digits. Also output a few of these pairs.

F – Free Desserts – First solved at 4:58:33

Problem

Given a large integer P, count the number of pairs of integers a, b such that $a + b = P$, $0 < a < b$ and a, b, P do not mutually share any digits. Also output a few of these pairs.

Solution

- Use dynamic programming over the digits of a and b, from left to right, to count the number of solutions.
- Using a recursive function with memoization (as opposed to an iterative DP), simplifies reconstruction of the pairs of integers we need to output.
- The solution proceeds in two steps:
  1. Compute the number of solutions to the whole problem and relevant subproblems.
  2. To produce the pairs, repeat the recursion and use the memoization table to avoid processing empty branches.

F – Free Desserts – First solved at 4:58:33

Solution

- Let's define the subproblem of size k as the original problem restricted to the k least significant digits of P.
- Let a', b', P' denote the numbers a, b, P restricted to the k least significant digits, including possible leading zeros in a and b.
- A subproblem of size k also needs the following input:
  - $LZa(k)$, indicating if there is a leading zero in a' at position k,
  - $Cy(k)$, the carry (0 or 1) at position k produced by $a' + b'$,
  - $Fa(k)$, the set of forbidden digits for a', and
  - $Fb(k)$, the set of forbidden digits for b'.
- Note that $Fb(k)$ is uniquely determined by the other parameters and the digits in P.

F – Free Desserts – First solved at 4:58:33

Solution

- Let's label each recursion level by its corresponding value k.
- The solution of a subproblem of size k can be constructed using the solutions of the subproblems of size $k-1$ as follows:
  - At the kth level of recursion, loop through all acceptable digits of a' and b' at position k and through all acceptable values of $Cy(k)$ and $LZa(k)$.
  - Inside the loop construct the corresponding values of $Fa(k-1)$, $Fb(k-1)$, $Cy(k-1)$ and $LZa(k-1)$ and recurse to the $(k-1)$th level providing these values as input to the recursive call.
  - Use the values returned from the $(k-1)$th level and combine them with the acceptable digits at the kth level to obtain the solution at the kth level. Or, when computing the number of solutions, just sum up the returned values.

F – Free Desserts – First solved at 4:58:33

Solution

- The complete solution of the original problem is composed of the solutions of two subproblems with parameters:
    - k = length of P,
    - (1) $LZa(k) = \texttt{true}$ or (2) $LZa(k) = \texttt{false}$,
    - $Cy(k) = 0$,
    - $Fa(k) = Fb(k)$ = the set of digits in P.
- The memoization table keeps the number of solutions of subproblems for all combinations of $k, LZa(k), Cy(k), FA(k), FB(k)$.
- The size of the table is at most $18 \cdot 2 \cdot 2 \cdot 1024 \cdot 1024 = 75\,497\,472$.
- The complexity is $O(\log P)$. However, the big constant factor of $\approx 10^6$ plays a significant role in any implementation.
- As there are only 10 digits available, use bitmaps to represent subsets of forbidden digits in a and b.

References

[1] http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html.

[2] https://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/Chapter11.pdf.

[3] https://en.wikipedia.org/wiki/Rouch%C3%A9%E2%80%93Capelli_theorem.