Brazilian ICPC Summer School 2022: 5^{th} Contest Solution Outlines

Fidel I. Schaposnik M.

January 29, 2022

This document presents solution outlines for problems included in the 5^{th} contest of the first week of the Brazilian ICPC Summer School of 2022. Problems are loosely classified in difficulty ranging from 1 to 10, but keep in mind this is obviously my subjective evaluation which you may not share. The discussion presented here does not aim to be either detailed or exhaustive, but should be considered only as a rough guide helping you to think about these problems. As usual, please send any comments or questions to `fidel.s+maratona@gmail.com`.

1 Lightbulb Testing

- **Subject:** parsing
- **Difficulty:** 8/10
- **Solutions during contest:** 1

This problem requires parsing the input to simulate the process of turning the lamp on and off. The grammar is simple, but to avoid implementation difficulties it is convenient to be extra-tidy and define a structure representing a sequence of operations, including parameters for "total on-time", "total off-time" and "lamp state changes after execution".

We can naturally define operations for concatenating and repeating these structures, and use binary search to determine how many repetitions of a given sequence can be achieved without going over a given total on-time for the lamp. Execution time shouldn't be a limiting factor for our solution if parsing is performed in linear time on the length of the input.

2 Vigenère Cipher Encryption

- **Subject:** ad-hoc
- **Difficulty:** 1/10
- **Solutions during contest:** 28

This problem asks us to encrypt a string using the Vigenère cypher. We should simply apply the procedure described in the problem statement, and execution time is then linear in the input size.

3 Vigenère Cipher Analysis

- **Subject:** strings
- **Difficulty:** 9/10
- **Solutions during contest:** 1

This problem asks us to determine whether a string S, encrypted using the Vigènere cypher, can be decrypted without having access to the key (whose length is bounded by K) if we know two words W_1 and W_2 which should be present in the original message.

To solve this problem, we consider all possible key lengths $k = 1, \ldots, K$ and generate all possible keys of length k which are compatible with W_1 and W_2, considering all possible positions in which these words can appear in S. The valid keys must be compatible with both W_1 and W_2, and the occurrences of these in S should be non-overlapping.

An important detail to keep in mind is that we may find various different keys decrypting S into the same original message (*e.g.* "A" and "AA"), so that in order to check if the answer is ambiguous it is not sufficient to count the number of different valid keys we find. Instead, we should explicitly check that there are at least two keys which result in different strings when they are repeated as many times as needed to cover the message length $|S|$.

If we use hashing to compare and rotate strings in $\mathcal{O}(1)$ time, as well as to repeat strings n times in $\mathcal{O}(\log n)$ time, the solution's execution time is then $\mathcal{O}(K|S| \log |S|)$.

4 Flowery Trails

- **Subject:** graphs
- **Difficulty:** 2/10
- **Solutions during contest:** 26

The problem consists in finding the sum of the weights of all edges that belong to at least one minimum length path joining two given nodes in the input graph. An edge (u, v) of weight w belongs to a minimum length path from node S to node T if and only if

$$\text{dist}(S, u) + w + \text{dist}(v, T) = \text{dist}(S, T)\,,$$

where $\text{dist}(x, y)$ is the minimum distance between nodes x and y. Since our graph is undirected, we also have $\text{dist}(v, T) = \text{dist}(T, v)$, so that we can use the above formula to check whether an edge belongs to a minimum length path from S to T in time $\mathcal{O}(1)$, as long as we have precalculated all possible values of $\text{dist}(S, x)$ and $\text{dist}(T, x)$. We can achieve this make use twice of Dijkstra's algorithm, starting from S and T. The solution then runs in $\mathcal{O}(N \log N + E)$ time, where N is the number of nodes in the graph and E is its number of eges.

5 The Imp

- **Subject:** dynamic programming
- **Difficulty:** 3/10
- **Solutions during contest:** 12

If we consider boxes one-by-one in some arbitrary order to be defined later, in each step we have two options:

- either we buy the box, in which case the Imp makes sure to minimize our winnings so that the final value is the minimum between that box's value (if he allows us to take it) and the value of all the remaining boxes when the Imp is left only $k - 1$ box destructions (if he destroys the box we have chosen), minus the cost of the destroyed box;

- or we do not buy the box, and we move on to the next one without affecting the Imp's number of box destructions k.

Since the decision above is up to us, we should maximize between both cases. Then, we are only left to decide the optimal order in which it is convenient to consider the boxes, and it is not hard to convince ourselves that we should do this in order of increasing value. The solution therefore runs in $\mathcal{O}(N \log N + NK)$ time.

6 Pork Barrel

- **Subject:** graphs + data structures
- **Difficulty:** 10/10
- **Solutions during contest:** 2

This problem can be understood as a dynamic minimum spanning tree (MST), in which for each range $[l, h]$ we are to find the weight of the MST using only the input edges with weights ranging from l to h, inclusive.

We cannot apply a standard MST algorithm (*e.g.* Kruskal's algorithm) to answer each query separately, since this would be too slow. We should note, however, that if we order the edges according to their weight w_i, so that $w_1 > w_2 > \cdots > w_{M-1} > w_M$, then the MST using edges of weight greater or equal to w_i (let's call this MST$[w_i]$) is practically identical to MST$[w_{i+1}]$, *i.e.* the MST which uses edges of weights greater or equal to w_{i+1}. In fact, the only difference between them is that if edge $i+1$ connects two nodes u and v which are already connected in MST$[w_i]$, then to obtain MST$[w_{i+1}]$ we should remove the largest edge in the path from u to v in MST$[w_i]$, replacing it with the edge of weight w_{i+1}. On the other hand, if u and v are not connected in MST$[w_i]$, then we simply need to add this edge to obtain MST$[w_{i+1}]$.

Applying the algorithm above we can generate all MST$[w_i]$ in time $\mathcal{O}(MN)$, using DFS to find in each step the heaviest edge in the corresponding path. To answer the queries, we then need to add the weights of all edges of weight less than or equal to h which belong to MST$[l]$. If we store the MST$[w_i]$ as arrays $m_i[1 \ldots M]$ such that

$$m_i[j] = \begin{cases} 0 & \text{if edge } j \notin \text{MST}[w_i] \\ w_j & \text{if edge} j \in \text{MST}[w_i] \end{cases},$$

then the answer to each query can be obtained in logarithmic time using a segment tree to compute sums over intervals. Since we will be needing M arrays of this type, we should use persistance to save memory in the representation of the segment trees, taking us from $\mathcal{O}(M^2)$ to $\mathcal{O}(M \log M)$ required memory. The final solution then runs in $\mathcal{O}(MN + M \log M + Q \log M)$ time.

7 Wedding Hall

- **Subject:** computational geometry
- **Difficulty:** 7/10
- **Solutions during contest:** 0

Let L be the length of the sides of each of the three squares forming the hall. Note that if it is possible to build a hall of side-length L, then it is also possible to build a hall of any side-length L' such that $L' < L$. This property allows us to simplify the problem using binary search on the maximum allowed size: in each step of the search we need to answer a question of the form "Is it possible to build a hall of side-length L?" We also note that without any loss of generality we can assume the hall we build always has at least one point on its left and lower sides, since otherwise we could displace the hall continuously to the left or downwards, respectively.

As an intermediate step to reach the solution let us first consider a simpler problem, namely: is it possible to find a renctangle of dimensions $L \times 2L$ which does not contain any input points, but has at least one of them on its left side? This problem can be solved using a right-to-left sweep line, where the events we need to consider correspond to the crossing of the sweeping line with the input points. If all during the sweeping process we keep track of the set of points that are at most a distance L to the right of the sweep line, when we cross a new point we can easily determine whether we can fit a rectangle of dimensions $L \times 2L$ having this point on its left side. To do this, we look for the first point in our set that is above the one we are crossing, as well as the first point below it, and if their separation is greater than $2L$ then it is possible to fit the $L \times 2L$ rectangle between them. Moreover, we can even determine a vertical segment on which we need to place the lower-left corner of this rectangle!

To solve the original problem, observe that a hall can be thought of as being composed of two overlapping rectangles, one of dimensions $L \times 2L$ and the other of dimensions $2L \times L$. We therefore need to perform two sweep lines, one in the right-to-left direction and the other in the the top-to-bottom direction, to find two sets of segments (vertical and horizontal, respectively) on which the lower-left corner of both rectangles needs to be placed. Since we want these rectangles to overlap, their lower-left corner must coincide and therefore we are interested in the intersections of these sets of segments: if there is at least one intersecion point, then there exists at least one position that is compatible with a hall of side-length L, otherwise (*i.e.* if the sets of horizontal and vertical segments don't intersect) it is not possible to build this particular hall.

8 Stapled Intervals

- **Subject:** maths
- **Difficulty:** 5/10
- **Solutions during contest:** 11

The key observation to solve this problem is that a "stapled interval" can never contain a prime number p, for if this were the case it should also contain another number that is not coprime with it, of which the lowest possibility is $2p$. However, Bertrand's postulate[1] states that there is at least one prime number q in the range from p to $2p$. We would therefore be forced to repeat this procedure extending the interval at least up to $2q$, and this would go on *ad infinitum* so that our interval could never be finite, which is a necessary condition to be a stapled interval.

Since the number of primes less than or equal to n is

$$\pi(n) \sim \frac{n}{\log n},$$

we can expect their separation to be of order $n/\pi(n) \sim \log n$, and the number of intervals to consider in the range $[0, N]$ will be $\mathcal{O}(N \log N)$. A brute-force check of an interval of size n would take $\mathcal{O}(n^2 \log N)$, where the logarithmic

[1] `https://en.wikipedia.org/wiki/Bertrand's_postulate`

factor comes from using Euclid's algorithm to compute greatest common divisors. Thus, since $n \sim \log N$ a brute-force solution will take $\mathcal{O}(N \log^4 N)$ time, which is too slow (though it could serve to precompute all necessary stapled intervals offline, to later explicitly include them in your source code).

To speed up the process, we note that

- we can avoid using Euclid's algorithm noting that the closest non-coprime numbers to m are $m \pm p$, where p is the smallest prime dividing m (which we can precompute while executing the sieve of Eratosthenes);
- we may reuse the checks we performed for an interval when checking some other interval that contains it.

With these observations, it is not too hard to speed up the solution to $\mathcal{O}(N \log^2 N)$ or even $\mathcal{O}(N \log N)$ execution time, which will be sufficient to pass the test cases without using any precomputation tricks.

9 It Can Be Arranged

- **Subject:** graphs
- **Difficulty:** 6/10
- **Solutions during contest:** 13

To solve this problem we binary search for the answer (number of rooms to book), given that if we can satisfy the constraints using K rooms we can certainly also do so for any number of rooms $K' > K$. To determine whether it is possible or not to solve the constraints with a given value of K, we construct a flow network as follows:

- Create two nodes for each assignment, one source (s_i for $i = 1, \ldots, N$) and one target (t_i for $i = 1, \ldots, N$).
- Create a global source S connected to every source node s_i with edges having a capacity equal to the number of rooms required by each subject (c_i for $i = 1, \ldots, N$).
- Creamos a global sink T connected to every target node t_i with edges having capacities c_i.

- If we can reuse the rooms of the i-th subject for the j-th subject (that is, if $b_i + \text{clean}_{ij} < a_j$), we add an edge of infinite capacity from the source node s_i to the target node t_j.

To impose the condition that the non-reused rooms should be at most K, we have to make sure that the flow from the set of source nodes $\{S, s_i\}$ to the set of sink nodes $\{t_i, T\}$ which does not go through the infnite-capacity edges (*i.e.* does not correspond to reused rooms) is less than or equal to K. To achieve this:

- Create a node L connected to every source node s_i with infinite capacity edges.
- Create a node R connected to every target node t_i with infinite capacity edges.
- Connect L and R with an edge of capacity K.

In this way, the edge from L to R represents all those rooms that need to be reserved because they cannot be reused from some other subject, and is in fact the only part of the flow network that needs to be modified as we perform the binary search. Evidently, it is possible to satisfy the problem's conditions with a given value of K if the flow from S to T in the corresponding network is equal to the total number of required rooms, namely $\sum_{i=1}^{N} c_i$, and because the number of nodes and edges is $\mathcal{O}(N)$ we do not need to use any particularly efficient flow algorithm to solve this problem.

10 The Agency

- **Subject:** greedy
- **Difficulty:** 4/10
- **Solutions during contest:** 22

To go from planet S to planet E we need to take a certain minimum number of steps P, each of which changes the bit at a given position of S that differs with the bit in the same position of E. There are then two tipes of operations: those in which we turn off a bit in S (that is, we make it $1 \mapsto 0$) and those in which we turn it on (*i.e.*, we have $0 \mapsto 1$).

Since we pay a tax every time a given bit is on, it's clear that it is always convenient to first perform the "turn off" operations first, and only later the "turn on" ones. Moreover, when turning off bits it is always convenient to choose as the next bit the one with a highest tax among all those that need turning off. Similarly, when turning on bits it is always convenient to choose in each step the next bit as the one whose tax is the least costly among all those that need turning on.

With the observations above, given a set of steps we know the optimal order in which they need to be taken, and can compute the minimum cost of the whole process. However, it is not necessarily optimal to perform only P steps: if a bit is on both in S and in E, but its tax is very high, it may actually be more convenient to turn it off before doing the P steps, and turn it back on once we have finished. In general, we then need to take $P + 2 \times M$ steps, the additional $2 \times M$ steps corresponding to turning off and later on again the M most costly bits that are on both in S and E.

If the total number of bits in S and E is N, simulating the whole process takes $\mathcal{O}(N)$ time and therefore we can simply check all $\mathcal{O}(N)$ possible values of M. The solution then runs in $\mathcal{O}(N^2 + N \log N)$ time, where the $N \log N$ factor comes from sorting (only once for the whole problem) the costs of all the taxes.

11 Bases

- **Subject:** parsing + maths
- **Difficulty:** 8/10
- **Solutions during contest:** 4

We need to find all the bases in which two mathematical expressions involving sums and products of natural numbers are equal. In an arbitrary base x, a natural number is a polynomial $P(x) = \sum_i d_i x^i$, where the d_i are the number's digits with $i = 0, 1, 2, \ldots$ going from right to left. Any expression therefore involves sums and products of polynomials, and in turn evaluates to a polynomial, so that our problem is to analyze the solutions of $P_l(x) = P_r(x)$, where $P_l(x)$ and $P_r(x)$ are the polynomials resulting from evaluating the left- and right-hand-side expressions, respectively.

If all the coefficients in $P_l(x)$ and $P_r(x)$ are equal, there is an infinite number of solutions starting at $x = m+1$, where m is the highest digit appearing in either expression (since for any base B the valid digits are $0, 1, \ldots, B-1$). Otherwise, let k be the rightmost position where both polynomials differ, then

$$0 = P_l(x) - P_r(x) = \sum_{i=k}^{N} (l_i - r_i)x^i = x^k \sum_{i=0}^{N-k} (l_{i+k} - r_{i+k})x^i \,,$$

where N is the position of the highest non-zero coefficient of either polynomial. Since $x = 0$ is not a valid base, this means

$$l_k - r_k = -x \sum_{i=0}^{N-k-1} (l_{i+k+1} - r_{i+k+1})x^i \,,$$

where we moved the first term outside of the sum and extracted a common x factor from all the remaining terms. Therefore, x divides $|l_k - r_k|$ and since the input expressions are at most 17 characters long we have that coefficients are bounded by $9 \times 9 \times \cdots = 9^8 < 5 \times 10^7$; therefore, we can factorize $|l_k - r_k|$ and find all of its divisors.[2]

In order to check whether a certain divisor x of $|l_k - r_k|$ satisfies $P_l(x) = P_r(x)$ we need to parse both expressions while evaluating them in base x, taking care of avoiding overflows (because *e.g.* $99999999 \times 99999999 > 2^{64}$ already for base 16). One trick to do this is to use a probabilistic check: randomly choose a few different primes p and check $P_l(x) = P_r(x) \mod p$; then the evaluation can easily be kept under control, and the probability of obtaining a "false positive" $P_l(x) = P_r(x)$ result are very small.

[2]For an estimate of the maximum number of divisors, observe that $2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 > 5 \times 10^7$ and has $2^9 = 512$ divisors.