Interactive Tasks

Mamnoon Siam

Draft July 2, 2021

1 On Common Pitfalls

1.0.1 Don't miss important intermediate relations

Problem 1.1 (Chocolate Bunny)**.** There's a hidden permutation p of length n. Find it using at most $2n$ queries of following format: $ask(i, j) = p_i \mod p_j$.

Solution. Instant reaction would be to find where n is. Because if we query with $p_j = n$, then `ask` function returns the exact value of p_i. So one possible approach is to find the location of n first, say using n queries (we're just rolling the dice here, nothing much, let's see if we can get a 6-6), and then use $n-1$ queries to find the remaining numbers. After grinding a while we get $(a \mod b > b \mod a) \Leftrightarrow (a < b)$. So we can find the maximum value using $n-1$ comparisons – first make a variable m, we'll store the maximum so far in this. Initially assign some arbitrary index to m. Then go through the rest of the indices $p_1, p_2, \ldots, p_{n-1}$ in some arbitrary order. In the i-the step, compare p_i with m, if $a_{p_i} > a_m$, then do $m := p_i$, otherwise do nothing. At the end, m is the index of n.

But every comparison requires two queries and so we would need $2(n-1)$ queries to find n. After that $n-1$ extra queries, overall $3(n-1)$ queries. Well, that's no good to us. The core of this problem is to notice that in some step i, when we are not changing m, we are actually knowing the exact value of a_{p_i} because $a_{p_i} < a_m \implies a_{p_i} \mod a_m = a_{p_i}$. And when we are changing the value of m, we are getting the exact value of a_m. Now, if you think about it, every number that's not n is getting beaten by at least one other larger number, thus at the end of this algorithm, we have all the information we need! □

2 Generic Approaches

2.1 Find one extreme/helpful object

Then find others via that object.

2.1.1 Pairing up different type of objects using stack

Problem 2.1 (HonestOrUnkind)**.** There are A honest people and B dishonest people. You want to find which of them are honest. Every people know about

others' honesty. You can ask at most $2N$ questions ($N = A + B$) of the follow format: choose two people a and b, and ask a if b is honest or not. Dishonest people will arbitrarily choose to be honest or dishonest, and their decision is independent each time. But the honest people will always tell the truth. Your are allowed to ask at them at most $2N$ questions. Find the set of honest people or report that there will more than one set of honest people regardless of your way of questioning.

Solution. This problem has similar setup as the "Knights and Knaves" puzzles. Familiarity with those puzzles may help thinking about the setup, but it's not needed. Nonetheless, you can always google if you want to.

If the number of honest persons is not more than the number of dishonest persons, then whatever you do, you can't be sure of the set of honest persons. Because, what if A of the dishonest persons just act like a plain honest person and mess up any of your assumptions? So, if $A \leqslant B$, we have to report impossible.

Now, let's see if we can find one honest person, then we can get others' type by asking him about everyone else.

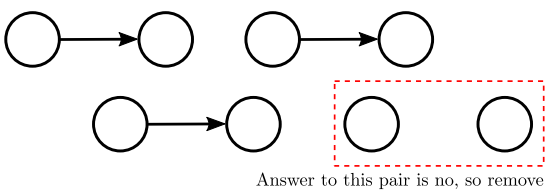
Next up, if you analyze all possible scenarios, namely, "what if a and b both are truthful", "a is truthful but b is dishonest", and so on... total 2^2 scenarios, you will see that

- If the answer is yes (that is, **Q: Hey a, is b honest?** (I'll denote this question with $\mathsf{ask}(a, b)$ A: Yes) then we can only, but surely say that this is not the case where a is truthful and b is dishonest. Other three cases are equally likely.
- If the answer is no, again, we can only, but surely say that this is not the case where a and b both are truthful. But like the 'yes' answer, we cannot distinguish among the other three cases.

We didn't have any other way to tackle this problem other than to grind like this. So our best bet would be to use these two interpretations of the answers and the fact that $A > B$ somehow to find a truthful person.

Say, we arbitrarily picked two persons a and b and asked a about b. If the answer is no, then at least one of them is dishonest, and we can discard them, because even without them the $A > B$ property holds. What is the answer is yes? The only assumption we can make about their honesty is that, it must be one of TT, FT and FF (T means that person is honest and F means he's dishonest, and first letter represents a and second letter represents b).

We may now articulate some sort of process: in the first round, pair up everyone with someone else, for example, pair up $2i-1$-th and $2i$-th person. Now query each pair (arbitrarily choose who will be a). Eliminate each pair whose answer is no. Otherwise, we'll just keep track that "these pairs' answers are yes". Actually, let's draw directed edge from a to b if $\mathsf{ask}(a, b)$ is yes.



Answer to this pair is no, so remove both of them.

Let's call these directed paths *chains* (well, actually we can call single nodes as chains too, they're just one-node chains). In the next step, arbitrarily pair up chains. If there are odd number of them, just ignore one of them for this step. For each pair pick the last node of the first chain as a and first node of the second chain as b (arbitrarily choose which one should be the first chain and which one should be the second). Now query them. One of the two things will happen – either two chains will get merged, or two of the nodes, namely a and b will get removed from our consideration.

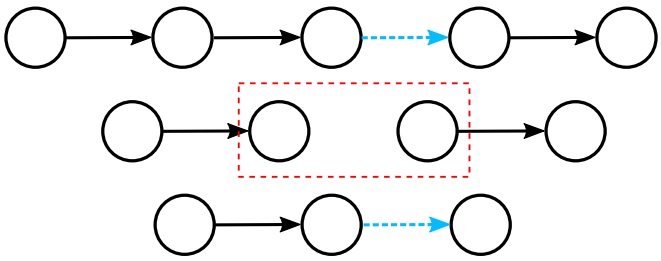


Figure 1: Red box means answer to $\texttt{ask}(a, b)$ is no, and blue edge means the answer is yes.

We keep doing this process until there is only one chain left. What's interesting about these chains is that if you write down the types of people on the chain from left to write, the sequence will be something like this: `FFFFFFFTTT` i.e. everything is `T` after the first one. Also, we know that there is always at more `T`s than `F`s, we can say that the last person on the remaining chain is a honest person.

We don't actually need to pair up everything at each step – just arbitrarily choose two chains and query. To smartly pair things up, we can you stack (forget about the chains completely) – keep a stack of people, try to push each of them one by one, query $\texttt{ask}(\texttt{stack.top}, \texttt{current person})$. If the answer is no, pop the top person from the stack and discard both of them. Otherwise put the new person on the top. At the end, the top person in the stack is honest. □

The next problem is another showcase of the idea about pairing up objects.

Problem 2.2 (IOI 2015 Towns)**.** There is a hidden edge-weighted tree of $N \geqslant 6$ leaves (you don't know the number of internal nodes). It is guaranteed that every internal node has degree greater than 2. Center of the tree is a node which minimizes the maximum distance from it to some other node. You need to find the maximum distance from a center to its furthest vertex. Additionally, you need to figure out if one of the center(s) is also a leaf-centroid or not (that is, each of the subtrees created by removing that center have at most $\left\lfloor \frac{N}{2} \right\rfloor$ leaves. You can ask $|uv|$, the length of the simple path between u and v for any u, v. You are allowed to use at most $\left\lceil \frac{7 \cdot N}{2} \right\rceil$ queries.

2.1.2 Keep beating inferior objects sequentially until you find the extreme one.

Also, note that, every object gets beaten at exactly once except the extreme one.

Problem 2.3. See 1.1.

2.2 Or just find some specific object?

2.2.1 In N/c queries?

Outline —
- Try to eliminate at least c objects from 'potential' set per query.

Problem 2.4 (Find root using lca query)**.** Find root of a tree using no more than $\lfloor n/2 \rfloor$ $\text{lca}(u, v)$ queries.

Solution. Pick two leaves in each iteration. Either one of them is the answer and terminate, or delete them from the tree and continue. □

2.3 Try To Solve for Special Case First

Problem 2.5 (Strange Device)**.** Computer has an array a of n distinct numbers. It can answer queries of the following form: in response to the positions of k different elements of the array, it will return the position and value of the m-th among them in the ascending order. You know k (fixed beforehand by the computer), but m is hidden. Find m by using no more than n queries. $1 \leqslant m \leqslant k < n$.

Solution. Try to solve for $k = n - 1$. Actually there aren't many ways to query i.e. there are n ways. Query $\{1, \ldots, n\} \setminus \{i\}$ $\forall i$. How to find m using the information? Investigate. □

2.4 Meet in the Middle: How To Meet?

Outline — Try searching for the extremest object from both ends – to make sure you meet at the same place.

Problem 2.6 (Down or Right)**.** There's a binary grid of size $n \times n$ – 1's are inaccessible. $\text{ask}(x_1, y_1, x_2, y_2)$ tells if there is a down-right path from (x_1, y_1) to (x_2, y_2), with a restriction that $|x_1 - x_2| + |y_1 - y_2| \geqslant n - 1$. Find a path from $(1, 1)$ to (n, n) by asking no more than $4n$ queries.

Solution. We can find a half-path starting at $(1, 1)$ and another half-path ending at (n, n). How do we make sure that they meet in the middle? While finding those paths we make sure to follow the uppermost ($\equiv$ rightmost) path (extreme object). □

2.5 Mapping/Bijection

Problem 2.7 (Decypher the String)**.** transform(S) performs a sequence of swaps between pairs indices of string S (this sequence is fixed beforehand). You are given transform(P) and you need to find P. You can ask transformed values of at most 3 strings. $|P| \leqslant 10^4$.

Solution. $26^3 > |P|$. □

2.6 Bitmask Trickeries

2.6.1 Iteratively Uncover Prefix/Suffix and use that to find the following bits

Problem 2.8 (Ehab and another another xor problem)**.** Computer has two hidden integers a and b ($0 \leqslant a, b < 2^{30}$). In one query, upon selecting two integer c and d ($0 \leqslant c, d < 2^{30}$), you can compare $a \oplus c$ and $b \oplus d$ (i.e. equal, greater, less). Find a and b using no more than 62 queries.

Solution. If we can keep track of some prefix, we can reset all bits in that prefix of $a \oplus c$ and $b \oplus d$, thus can peacefully work out the remaining bits. So, the only sub-problem we need to solve is to find the most significant bits of a and b. □

Takeaway. At this point, ask yourself if there are MANY reasonable ways of querying? No. It is always helpful to ask yourself this question and if the answer if no, then exhaust all the possibilities.

3 Bunch of Hardcore Tricks

3.1 Disjoint partition using binary representation

Outline — If $a \neq b$, there is at least one bit in which a and b differ.

Suppose we need to find a special pair (a, b) $a \neq b$ (maybe some pair that minimizes some cost fn) and we are allowed to query with two sets A and B as parameters ($A \cap B = \varnothing$) , where, in return we will know if $a \in A \wedge b \in B$ (or maybe minimum cost of some pair (x, y) such that $x \in A$ and $y \in B$).

In such cases, do $\log N$ queries using the following method: i-th query, set $A := \{a \mid a_i = 0\}$ and $B := \{b \mid b_i = 1\}$[a].

[a] ${}^{a}x_i$ denotes the i-th bit of x.

Problem 3.1 (Tree Diameter)**.** Find diameter of a hidden weighted tree using $2 + \log N$ queries of the following form: $\text{ask}(A, B) = \max_{a \in A, b \in B} \text{dist}(a, b)$ (A and B must be disjoint).

Problem 3.2 (The penguin's game)**.** There's a hidden array a (size is at most 1000). Computer has selected two indices i and j such that $a_i = a_j = x$ and $a_{k \notin \{i,j\}} = y$ ($1 \leqslant x, y \leqslant 10^9, x \neq y$). Find i and j using at most 19 queries. Allowed query: $\text{ask}(S) = \bigoplus_{k \in S} a_k$.

Solution. Obviously binary search won't work. So we shouldn't waste our time trying to unlock some hidden quirkiness which will enable binary search. First off, if we ask with a set S, we can deduce the parity of number of x in S. So, using our trick we can determine which bits of i and j are same and which aren't. Then, if we can somehow find one of them, we are done. At least one will differ. Now, we have a set of indices where exactly one of i or j lies. Find where that is

using binary search. Also, this set's size is at most half of the original array's. So one query is saved in bs. □

4 Data Structures in Interactive Problems

4.1 Segment Tree

Problem 4.1 (Minimum and Maximum)**.** Find minimum and maximum of an array using no more than $\left\lceil \frac{3 \cdot n}{2} \right\rceil - 2$ queries of the following form: ask$(i, j) = 0$ if $a_i = a_j$, 1 if $a_i > a_j$ and -1 otherwise.

Problem 4.2. Find smallest two elements in an array using no more than $N + \lceil \log N \rceil$ comparisons.

4.2 Centroid Decomposition

Outline —

Problem 4.3 (Ehab and the Big Finale)**.** You are given a rooted (at 1) tree of n nodes. Computer has a hidden node x and you need to find it. Allowed query format:

1. dist(u): returns distance between x and u, the number of edges.
2. second(u): returns index of the second node on the path from u to x.

Use at most $2 \lceil \log n \rceil$ queries in total.

Solution. First of all, we can find level of x by ask(1) and ignore all vertices with level greater than that, so we are always above x. This will come in handy later. At first our search space is the whole tree. We will reduce it by at least half per iteration. Pick the centroid c of our search space. We can figure out if x is in the subtree of c wrt the original "rooting". If it's not, then we know we can ignore the subtree of c (again, wrt the original rooting at 1) – at least half reduction. Otherwise, just query second(c) and find which way to go, ignore the rest – again, at least half reduction. □

Problem 4.4 (Cactus Search)**.** You are given a cactus graph $G(V, E)$ ($|V| \leqslant 500$), Cmputer has a hidden vertex u and you need to find it. Method of interaction is when you give it a vertex v, it either says "`FOUND`" or "`GO` w", where w is a node adjacent to v which satisfies $|uw| < |uv|$[1]. Use at most 10 interactions.

[1] $|ab|$ means the shortest distance from u to v.

Solution. Let us define $S(x)$ as the sum of distances between x and all nodes in the graph (or in the subgraph under our consideration, as we will see later) i.e. $\sum_{y \in V} |xy|$.

Note that centroid of a tree has the minimum S value. Analogously, we can define "centroid" of a general graph to be the vertex with minimum S value.

Basically, we will maintain a set C of candidate nodes and we will try to shrink the size of it by at least half in each step. When we query a node v and get w in return, how do we decide if a node is still a candidate or not? A node x will still be a candidate *iff*:

$$|xw| < |xv|.$$

Let C' be the set of such nodes that satisfy this condition. We need to prove that, if we pick v as the centroid (as defined above for a general graph) of our search space, then $|C'| \leqslant \frac{1}{2}|C|$.

This can be proved by contradiction using lemma **??**. If more than half of the nodes have $|xw| - |xv| \leqslant -1$, then other nodes wouldn't be able to counter this drop in the S value. As a result we get $S(w) < S(v)$ – our initial assumption was wrong. □

4.3 Heavy-light Decomposition

Problem 4.5 (Ehab and the Big Finale)**.** See 4.3.

Solution. This problem also has a solution which utilizes properties of heavy-light decomposition technique. The main idea is we will reach x explicitly going through the root to x path. But we will skip nodes as many as we can using the heavy paths. As we need to change heavy paths at most $\log n$ times, we can query 2 times per jump in order to make intermediate decisions. Refer to the original contest editorial for more info. □

5 Graphs

Outline — This section contains various techniques to uncover hidden graphs. Few general ideas–

- Uncover one edge at a time.
- Try to use BFS/DFS tree? DFS trees are OP in these cases.
- Maybe find any spanning tree first?

Problem 5.1 (Hidden Bipartite Graph)**.** There is a hidden graph of $n \leqslant 600$ nodes, you have to tell if it's bipartite or not. Also, if it's bipartite, print the two sets, otherwise print an odd cycle. You can use the following utility function: $\mathsf{ask}(S)$ returns number of such edges that have both endpoints in S. You can usethis function at most 20 000 times.

Solution. How can you find an edge? Find a spanning tree (because then you can initially divide them into two sets, then check for odd cycle)? How can we find a spanning tree? □

This reminds me of two olympiad problems.

Problem 5.2 (JOI 18 Library)**.** There are $N \leqslant 1\,000$ books lined up in a bookshelf, from left to right, in an arbitrary order. You have to find that order. You can use the function `query`(S) at most 20 000 times. Let's define an operation to be grabbing a contagious range of books. `query`(S) returns the minimum number of operations needed to grab all the books in S. Note that, you cannot distinguish an ordering of books from its reverse, so it's not needed to specify whether the reported ordering is from left to right or right to left.

Solution. Main idea is to think of it as an line graph. So, you have a line graph, you pick some nodes, and give it to the grader. It returns you the number of connected components of the graph induced by the nodes you have given. Also, note that you can query at most $2n \log n$ times. This suggests that you should find one edge at a time using two binary searches (or maybe one bs, but you need several queries to take a decision).

Let's define few things to make the writing easier. For a graph G and a subset of its vertices S, let's denote G's subgraph that is induced the by S with $G[S]$. Also, let $C(G)$ denote the number of connected component of graph G.

We'll denote the original line graph with L (the one that we have to find). Also, let H denote the subgraph consisting of L's vertices (from 1 to N) and edges that we have found so far (initially no edge).

For some subset S of L's vertices, the grader will return $C(L[S])$ and we can compute $C(H[S])$. If the later one is less than the former one, $L[S]$ must have some extra edge than $H[S]$. Using this, we can find that using two binary search. Find minimum r such that $C(L[\{1, 2, \ldots, r\}]) > C(H[\{1, 2, \ldots, r\}])$. Then find maximum l such that $C(L[\{l, l+1, \ldots, r\}]) > C(H[\{l, l+1, \ldots, r\}])$. Now, you can surely say that there is an edge between l and r, which hasn't been found yet.

This problem can also be solved using the binary representation trick. See this. □

Problem 5.3 (CEOI 2014 Carnival)**.** TODO.

Unlike the previous two problems, sometimes you will need to find edges, but you wouldn't want to the additional edges to interfere. In these cases, try working with several independant sets, and binary search on them.

Problem 5.4 (JOISC 2020 Chameleon's Love)**.** There are $2N$ $(1 \leqslant N \leqslant 500)$ chameleons, numbered from 1 to $2N$. Each of them has three attributes:

1. Y_i, its gender (either 0 or 1, meaning male or female respectively),
2. C_i, its original color (an integer between 1 and N) and
3. L_i, the index of the chameleon it loves.

You know the following additional properties:

- There are exactly N male and N female chameleons.
- For each color $i \in [1, N]$, there are exactly two chameleons with original color i, one male and one female (denote their indices with A_i and B_i).
- No chameleon loves a chameleon of its own gender.
- No chameleon loves a chameleon whose original color is same as its own original color.

- Each chameleon is loved by exactly one other chameleon.

You have to find the N pairs $\{A_i, B_i\}$ (i.e. their order doesn't matter, you just have to report the pairs of chameleons that have the same color). You can use the following function at most 20 000 times.

`Query`(p) p is a list of unique indices. You invite the chameleons from p at a party. Each of them changes their color according to the following rule: if L_i attends the party, i's color is changed to L_i's original color. Otherwise, it remains the same as the its original one. The function returns the number of unique colors in the party. Of course, the change of colors is temporary for each party :) (that is, every query is independent).

Additional constraint	$L_{L_i} = i$, $i \in [1, 2N]$	$N \leqslant 7$	$N \leqslant 50$	$Y_i = 0$, $i \in [1, 2N]$	-
Points	4	20	20	20	36

6 Binary Search

Outline — For the most part, you need to look out for "constraint hints".

6.1 Warm-up Problems

Problem 6.1 (Gues the Maximums)**.** Statement TODO.

Problem 6.2 (Mahmoud and Ehab and the binary string)**.** Statement TODO.

Problem 6.3 (Partition Black and White points by Straight Line)**.** In each iteration of the interaction, you have to give a point in 2D plane and in return interactor will tell you the color of the point. You have to interact exactly n times (so you are forced to generate n points). After that, you will have to find a straight line which separates black and white points. $n \leqslant 30$, points have to be within $[0, 10^9] \times [0, 10^9]$.

Solution. This universe is better off without problems which are set as if they were some elegant combinatorial geometry problem while actually they are just some dumb constructive grind.

Note that, $n \leqslant \log_2(\text{canvas length})$. □

Problem 6.4 (Glad to See You)**.** There is a hidden sequence $1 \leqslant a_1, a_2, a_3, \ldots, a_k \leqslant n$ of distinct integers ($2 \leqslant k \leqslant n \leqslant 10^5$) and you have to find two of the integers by asking at most 60 questions of the following type: $\text{ask}(x, y) = [\min |a_i - x| \leqslant \min |a_i - y|]$.

Solution. Suppose the problem asks us to find one of them. If we sit back a bit and think the formal nature of binary search, we get this– we essentially maintain a range where we know, for sure, that at least one solution exists. Now, how can we say, "Hey, if answer to *this* query is *yes* then *that* part surely has at least one solution."? □

6.2 Doubling Trick to make Binary Search work

Outline — Suppose there is a hidden integer array a such that it satisfies $a_i \in [0, 1]$ and $\exists p$, $(a_i = 1 - a_{i+p}, \forall i)$. Find p by asking at most $2 \log |a|$ values of a.

Pick an arbitrary index s and ask $s + 2^0$, $s + 2^1$, ... Note the first k where $\text{ask}(s) \neq \text{ask}(s + 2^k)$.

Lemma 6.1. There is a "strict transition" in interval $[s, s + 2^k)$ i.e. $\exists i$, $s \leqslant i < s + 2^k \wedge a_s = a_{s+1} = \ldots = a_i = 1 - a_{i+1} = 1 - a_{i+2} = \ldots = 1 - a_{s+2^k}$.

After that, binary search in $[s, s + 2^k)$ (you can, because values of a in this range is "monotonic" e.g. 111111100000 or 000000111).

Problem 6.5 (IOI 2007 Aliens)**.** A stamp is a square grid of length $(2m-1)(2k-1)$ with alternating black (inked) and white (non-inked) square chunks of length $(2m - 1)$. See figure **??**.

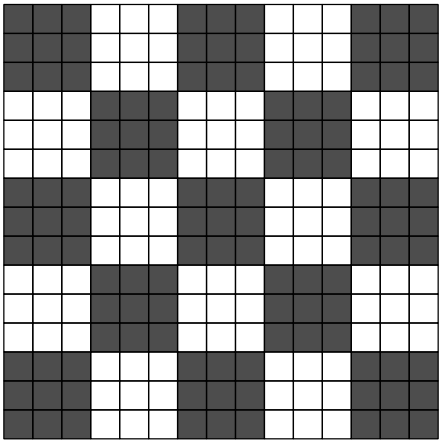


Figure 2: Stamp with $k = 3, m = 2$.

You know neither m nor k. Computer pastes the stamp somewhere on a $N \geqslant (2m - 1)(2k - 1)$ length square grid. If computer tells you the coordinate of an inked cell in the beginning, find the boundary (i.e. upper-left and lower-right points) of the stamp on the grid. (Refer to the actual problem statement for exact details.)

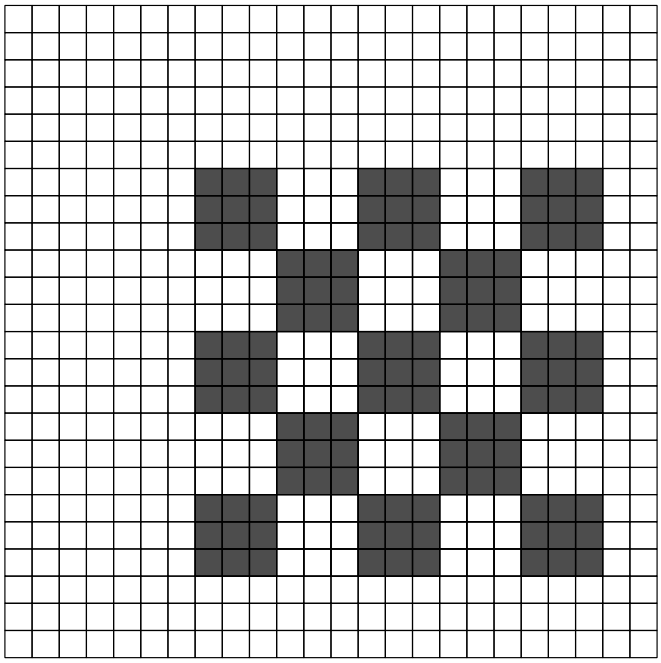


Figure 3: Stamp pasted on a 24×24 grid.

Solution. Just apply the mentioned trick in all four directions. □

Problem 6.6 (Game with Modulo)**.** Computer has a hidden number a. Find it using a total of at most $2\lceil \log U \rceil$ queries of the following form: ask(x, y) lets you know whether $(x \mod a) < (y \mod a)$ or not. $1 \leqslant a \leqslant U$, $0 \leqslant x, y \leqslant 2 \cdot U$.

6.3 Discrete Continuity

Yes, big fan of Pranav A. Sriram's Olympiad Combinatorics.

> **Outline —** As the name suggests, we have a almost continuous function – usually defined on $\mathbb{N}$ and we have to find one (maybe more?) root(s) of it, given that you know two opposite signed y values of that function. Use bisection method.

Problem 6.7 (The hat)**.** n cards (n is even, $2 \leqslant n \leqslant 100\,000$) are arranged around a circle. Each card has some value written on it. Let a_i denote the value written on the i-th card. It is guaranteed that $|a_i - a_{i+1}| = 1$. The function $\mathsf{ask}(i)$ returns the value of a_i. Your task is to find some i such that $a_i = a_{i+\frac{n}{2}}$ or state that there isn't one. You are allowed to use call the ask function at most 60 times.

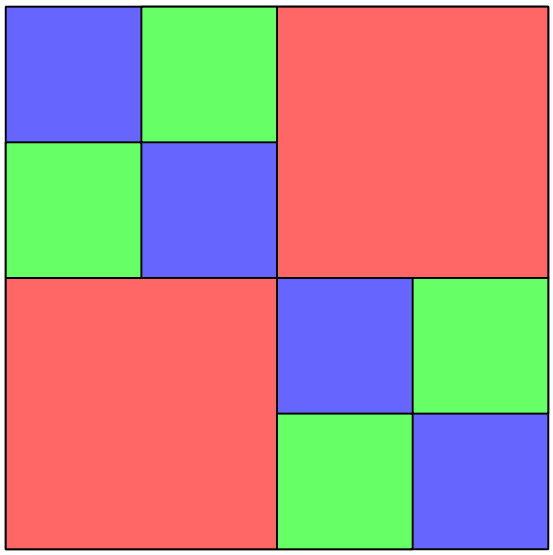
Solution. Let $f(i) = \text{ask}(i) - \text{ask}(i+\frac{n}{2})$. Note that, $f(i) = -f(i+\frac{n}{2})$. Furthermore, $f(i) - f(i+1) \in \{-2, 0, 2\}$, which means that all values of $f(i)$ have the same parity. So, if some $f(i)$ is odd, then print -1. Otherwise pick some arbitrary i and find out $f(i)$ by calling the ask function twice. If $f(i)$ is 0, then we are done. If not, then wlog let's assume that $f(i)$ is positive. Now, as we know that $f(i + \frac{n}{2}) = -f(i)$, if we go from i to $i+1$ and from $i+1$ to $i+2$ and so on,

upto $i + \frac{n}{2}$, we will go from strictly above the x-axis to strictly below the x-axis. This means we must have $y = f(x) = 0$ at some moment! Now we just need to binary search. Initialize l and r with i and $i + \frac{n}{2}$ respectively. The values of l and r strictly follows this property: $\exists x \in (l, r) \mid f(x) = 0$. At each step, pick $m = \left\lfloor \frac{l+r}{2} \right\rfloor$ and evaluate $f(m)$. If it's 0, then again, we're done. Otherwise if it's positive, it means that there is at least one $x \in (m, r)$ such that $f(x) = 0$. If $f(m)$ is negative, that means that there is at least one zero in (l, m). Adjust l and r accordingly. In each step, we are halving our search space. So in total, we need around $2 \log(\frac{n}{2})$ queries. □

7 Divide and Conquer Approaches

Problem 7.1. There's a hidden $n \times n$ matrix M with $M_{i,j} = 0$ and $M_{i,j} \in [0, 10^9]$. For each i, find $\min_{i \neq j} M_{i,j}$. Allowed query: ask$(\{w_1, w_2, \ldots, w_k\})$ returns an array r of n numbers where $r_i = \min_{j=1}^{k} M_{i,w_j}$. Query limit $2 \log n$.

Solution.



Remark. There's another solution using the "Disjoint partition using binary representation" trick. □

Problem 7.2 (IOI 2016 Unscrambling a Messy Bug)**.**

8 Randomized Approaches

Problem 8.1 (Interactive LowerBound)**.** There's a n node linked list, values of which are sorted in increasing order. ask(i) returns $value_i$ and $next_i$ – value of i-th node and its next node's index. Find lower bound index of x (given, you are also given n and the first node's index). $n \leqslant 50\,000$ and you can ask atmost 1999 queries.

Solution. Randomly ask k questions. Pick the node with max value but less than x. Now keep searching linearly from there. Probability that you will get bonked is approximately $\left(1 - \frac{2000-k}{n}\right)^k$. For optimal k,

$$\frac{d}{dk}\left(1 - \frac{2000-k}{n}\right)^k = 0$$

$$\implies \left(1 - \frac{2000-k}{n}\right)^k \left(\frac{k}{n\left(1 - \frac{2000-k}{n}\right)} + \ln\left(1 - \frac{2000-k}{n}\right)\right) = 0$$

$$\implies \left(\frac{k}{n\left(1 - \frac{2000-k}{n}\right)} + \ln\left(1 - \frac{2000-k}{n}\right)\right) = 0$$

. Plugging in $n = 50000$ and solving for k we get,

$$k = 2000\left(25e^{W(\frac{24e}{25})-1} - 24\right) \approx 994.915$$

, where W is inverse function of $y = xe^x$. □

Problem 8.2 (Lost Root)**.** In a perfect k-ary tree, nodes are labeled arbitrarily (hidden from you). You can use this function: $\text{ask}(a, b, c) = [b \in ac]$ (ac is the simple path from a to c). Find the label of the root of the tree using the function at most $60 \cdot n$ times.

Solution. We can find the height of the tree using simple math. Let it be h. We need to find two nodes a and b such that $|ab| = 2h$ (i.e. both are leaf nodes) and then print the middle node. Once we know the list of nodes on the path, we can find the middle node using similar strategy that is used to find k-th order statistics of an unsorted array in linear time (expected). Complexity of this part is $O(\log_k n)$. In practice, this is at most around 30 – far lower than 60.

To find the nodes on the path we need to query $n - 2$ times. So, we still can guess 60 pairs of nodes to get our hands on a pair of leaves. If we randomly choose the pair, in one draw the chance of at least one not being leaf is approximately $\frac{3}{4}$ (can differ drastically for small n, double checking this part is good habit). If we do 60 rounds probability of failure is $(\frac{3}{4})^{60} \approx 3.2 \times 10^{-8}$. □

Problem 8.3 (JOISC 2019 Meetings)**.** There's a hidden tree on N ($3 \leqslant N \leqslant 2000$) nodes, you have to figure out its structure. You know that each node's degree is atmost 18. You can use the following function to do the task:

`query`(u, v, w) This function returns a node x which minimizes $|ux| + |vx| + |wx|$. Note that, x may coincide with one of u, v, w, and while using the function, u, v, w may also coincide with each other.

Let X be the number of times you use the `query` function.

Limit of X	10^5	10^5	10^5	10^5	40 000
N	7	50	300	2000	2000
Points	7	10	12	49	22

9 Tricks Up The Sleeves

9.1 Desperate Times: Shaving 1 or 2 queries off

Problem 9.1 (The Hidden Pair)**.** Given tree on n nodes, hidden two distinct nodes u and v, find them using no more than $\lceil \log n \rceil + 1$ queries. ask$(\{q_1, q_2, q_3, \ldots q_k\})$ returns $\min_{i=1}^{n} |uq_i| + |vq_i|$ and arg min.

Solution. Find one node on the path, root one that node. Find one of the hidden node using $\lceil \log n \rceil$ queries. The use one query to find another (well, argmin will be given in every query). To shave of 1 query, note that distance from one of the hidden nodes will be $\geqslant \frac{n}{2}$. Another way of optimizing: to find first hidden node, **ignore the largest subtree** of one of the root's children. □

> **Takeaway.** Shaving 1 or 2 queries in a binary search problem may sometimes be just about reducing initial search space by half.

9.2 Hardcore Bijections and Injections

Problem 9.2 (Palindromic Paths)**.** There is a hidden binary grid of dimensions $n \times n$ and you have to find values of each cell (n is odd). ask(x_1, y_1, x_2, y_2) tells you whether there is any palindromic down-right path from cell (x_1, y_1) to cell (x_2, y_2). You can ask at most n^2 questions. It is guaranteed that value of cell $(1, 1)$ is 1.

Solution. If we color the board in checkerboard fashion (with $(1, 1)$ being black, we can find values of all the black cells. Similarly, if value of one white cell was fixed, other white values would be uniquely determined. So we have only two probable grids to consider. We simulate both of them and find a pair of cell where ask-value differs (using some local dynamic programming technique) in those two scenarios and ask that pair. We will know which scenario we are dealing with. Now, do we need to worry about whether we will be able to find such differentiating pair or not? Well, no. Because, if this this were to happen, we wouldn't be able to differentiate those two cases regardless what we would query, and this problem wouldn't occur in a contest altogether. □

10 Using Minimum Number of Queries

These type of problems involve calculating the optimal way of querying. Most of the time the solution is to find that optimal way using DP, and sometimes some greedy approach.

11 Honorable Mentions

Problems that are too abstract to be categorized atm.

Problem 11.1 (Intriguing Selection)**.** There are $2n$ players with distinct strength. You can compare them to find the strongest n players in such a way that there are at least two ways to sort them i.e. their ordering cannot be determined using your comparisons.

Solution. Keep two non empty sets such that no comparison is made between them at any point. Eliminate the smallest element from the union of the sets. To find the smallest element, compare the smallest objects of the two sets. In this way, although we are performing a comparison between two sets, one of the compared elements is getting eliminated. Also, we need to keep union of those two sets at least of size $n + 1$, in order to make sure that the eliminated object is not among strongest n ones.

As for the idea, I don't have any clue how one can come up with something like that atm. Seems completely out of the blue. □