Interactives and Randomised Algorithms

Max Godfrey 12/04/2022

Table of Contents

Interactive Problems

1. Module Tasks
   - Ask the grader some questions and then tell it your answer once you are confident
   - Constrained by the number of queries you can ask (commonly denoted Q)
   - Eg. *Find my Number* and *Jealousy*

2. Communication/Two-Step Tasks
   - Write two routines which communicate with one another
   - Routine 1's output is passed to Routine 2 (usually via parameters)
   - It is common for Routine 1's input to be obfuscated before being given to Routine 2
   - Constrained by what you are allowed to send, for example:
     - Not allowed to send more than k things
     - Each thing you send must conform to some constraints (eg. Integer in $[1, 10^5]$)

Module Task Tips

- Think about where Q comes from:
  - Commonly Q is $\log_2 N$, or $N \log_2 N$ (or slightly greater than those values)
  - The value of Q hints at the intended solution
  - Module tasks are often solvable with DnC or Binary Searching

- Query Conservation:
  - Think about the implications of one of your queries returning false
  - Eg. If you are told that something you're looking for isn't in the left half of the data, you *know* that it has to be in the right half of the data
  - Don't waste queries verifying things which must be true as implied by other queries' results

- Don't be afraid to use up all your queries:
  - You can't afford to spend time coming up with a solution using $0.5Q$ queries; you won't be penalised if you use exactly Q

Two-Step Tips

- Mappings are your friend:
  - It is possible that we can map every possible encoding to a unique value according to the constraints, and then send that value
  - To decode, work out the original input corresponding to your mapping
  - Tends to feature in problems where you are allowed to send more things than there are in the input

- Find a way to compress the data:
  - The input you are given will be bloated in some way
  - Often by finding a good compression, you can get subtasks (or sliding-scale points)

Cave Statement

- You are given N switches which may either be flipped *up* or *down*
  - $1 \leq N \leq 5000$
- Each switch controls one of N doors in a line (numbered from 1 to N), and the i-th switch may or may not control the i-th door
  - You do not know which switch controls each door
- A switch's *correct position* (ie. The position it must be in to open its respective door) is either *up* or *down*, and you must determine:
  - The *correct position* of all switches, and
  - The door controlled by each switch

- You are allowed to call $tryCombination(p)$ up to 70000 times:
  - p is a permutation of N integers, the i-th of which is 1 if the i-th switch is flipped *down*, otherwise 0 if it is flipped *up*
  - The function returns the index of the first closed door, or -1 if all doors are open

Cave Subtasks

1. [12 Points]: The i-th switch is connected to the i-th door
   - All you need to do is determine the correct combination (this hint is specifically given in the task description)
2. [13 Points]: The *correct position* of each switch is 0
   - All you need to do is determine which switch connects to each door (ditto)
3. [21 Points]: $N \leq 100$
4. [30 Points]: $N \leq 2000$
5. [24 Points]: No further constraints ($N \leq 5000$ as stated earlier)

Full Solution

- My tip on Slide 4 suggested that we should look for logarithmic factors
- For the first door:
  - Set all switches to 0
  - If $tryCombination > 1$, we binary search with all switches *up* (0)
  - At each level, set the left range to 1 and see if $tryCombination > 1$
    - If the condition is true, then the switch controlling Door 1 isn't in the left range
    - This means it *has* to be in the right range – don't waste a query verifying this!
  - Undo the range set (and set everything in the right range to 1), or traverse the right range depending on the result
  - Repeat until we have set all switches to 1 except one of them

- For all successive doors, do the same as above, but maintain all *correct positions* we know about so far
- $O(N \log_2 N)$ queries gets us AC on an IOI problem – not bad!
  - This is actually better than a lot of Bronze Medallists performed on this task

Combo Statement

- A video game controller has four buttons: A, B, X, and Y
- You get points for making *Combo Moves*, which are done by pressing sequences of buttons
- The game has a secret sequence of button presses S, of length N
  - The first character of S never reappears in it (ie. S cannot be "BXYABA")
- You are allowed to call $press(p)$, where p is a string:
  - $press(p)$ returns an integer: the length of the largest prefix of S which is also a substring of p
  - For instance, if $S =$ "ABXYY" and $p =$ "$\mathrm{XXYYABYABXAY}$", $press(p) = 3$
    - This is because "ABX" is the largest prefix of S appearing in p
  - $|p| \leq 4N$

Combo Subtasks

1. [5 Points]: $N = 3$
2. [95 Points]: $N \leq 2000$
   - Let q be the number of calls your program makes to $press$
   - If $q \leq N + 2$, your score is 95
   - If $N + 1 < q \leq N + 10$, your score is $95 - 3(q - N - 2)$
   - If $N + 10 < q \leq 2N + 1$, your score is 25
   - If $\max(N + 10, 2N + 1) < q \leq 4N$, your score is 5
   - Otherwise, your score is 0

Brute Force – 10 Points

- For $N = 3$:
  - Our score is not determined based on the number of times we call $press$
  - Try all possible strings

- For $N \leq 2000, Q = 4N$:
  - We can brazenly guess all four characters for each position of S
  - Don't be afraid to use all the queries we are allowed!
  - 5 Points on Subtask 2

Brute Force – 20 More Points

- For $N \leq 2000$, $Q = 4N$:
  - Suppose that k is some known prefix of S
  - We will denote the three characters that S is built from after the first character 1, 2, and 3
  - We determine the i-th character of S by guessing $k + 1$, and (if that fails) $k + 2$
    - From this, we can infer that the i-th character is 3 in the instance that both of our checks failed
  - We use similar logic to determine the first character of S in three guesses
  - $3 + (N - 1) * 2 = 2N + 1$ guesses in the worst case
  - This gives us 25 points on Subtask 2

- At the 2018 IOI, the 107th participant (one of the highest-scoring Bronze Medallists) scored 30 points on this problem
- Is it possible to determine the first character of S in two guesses?

Full Solution – 70 More Points

- Observe that the fact that the first character of S is different to the remainder of its characters allows us to make multiple guesses at once
- Can we come up with a way to determine the i-th character of S in one guess?
- For $N \leq 2000$, $Q \leq N + 2$:
  - Suppose that k is some prefix of S
  - We will denote the three characters that S is built from after the first character 1, 2, and 3
  - Spend at most **two** guesses to determine the first character
  - For all remaining characters bar the last one, guess "$k + 1 + k + 21 + k + 22 + k + 23$":
    - If $press$ returns $|k| + 1$, the next character is 1
    - If $press$ returns $|k| + 2$, the next character is 2
    - Otherwise, the next characters is 3
  - Spend at most two guesses to determine the final character
  - $2 + (N - 2) + 2 = N + 2$ guesses in the worst case – AC!

Parrots Statement

- We have a message M of N integers between 0 and 255 (inclusive) which we wish to send to a faraway land
- We have K indistinguishable parrots who will carry the message for us; each parrot can carry an integer between 0 and R (inclusive)
- However, the parrots will not necessarily arrive at their destination in the order that we send them off in
- We must write an encoder given N and M:
  - This function can call $send(x)$ up to K times, which will send a parrot to its destination
  - All parrots will arrive at their destination and correctly convey the integer which they carry
- We must also write a decoder given N, L, and X:
  - L is the number of parrots we originally sent
  - X is the message we sent, but reordered
  - This function must call $output(x)$ for each x in M (in order)

Subtask 1 – 17 Points

- $N = 8$, and each integer in M is either 0 or 1
- Each encoded integer must be between 0 and 65535, inclusive
  - $R = 2^{16} - 1$
- You are allowed to call $send(x)$ at most $10N$ times

Subtask 1 – 17 Points

- $N = 8$, and each integer in M is either 0 or 1
- Each encoded integer must be between 0 and 65535, inclusive
  - $R = 2^{16} - 1$
- You are allowed to call $send(x)$ at most $10N$ times

The solution:

- According to my tip regarding two-step tasks, the input is bloated in some way and we can exploit this
- A neat way to provide an easily re-constructible sequence is to send the positions of all indices in M where $M_i = 1$

Subtask 2 – 17 More Points

- $N = 8$
- Each encoded integer must be between 0 and 65535, inclusive
  - $R = 2^{16} - 1$
- You are allowed to call $send(x)$ at most $10N$ times
- Subtask 1, except that $0 \leq M_i \leq 255$

Subtask 2 – 17 More Points

- $1 \leq N \leq 16$
- Each encoded integer must be between 0 and 65535, inclusive
  - $R = 2^{16} - 1$
- You are allowed to call $send(x)$ at most $10N$ times
- Since there are no further constraints on M's values, $0 \leq M_i \leq 255$

The solution:

- Observe that we get to send 16 bits' worth of information per parrot
  - Each M_i requires 8 bits to be stored as $\lceil \log_2 255 \rceil = 8$
  - Each position in M requires 4 bits to be stored $\log_2 16 = 4$
- With each number we send:
  - The first 8 bits represent M_i (the value)
  - The following 4 bits represent i (the position of the value)

Subtask 3 – 18 More Points

- $1 \leq N \leq 16$
- Each encoded integer must be between 0 and 255, inclusive
- You are allowed to call $send(x)$ at most $10N$ times
- Incentive: If you solve this subtask, you will have equalled the score achieved by a former Director of Training on this task when he competed in the 2011 IOI

Subtask 3 – 18 More Points

- $1 \leq N \leq 16$
- Each encoded integer must be between 0 and 255, inclusive
- You are allowed to call $send(x)$ at most $10N$ times

The solution:

- Observe that this time we are only allowed to send 8 bits per parrot
- Since there are $16 * 8 = 128$ bits total in the message, we can send 128 integers
- In a similar manner to our Subtask 2 solution, in each number we send:
  - The first 7 bits represent the position of the i-th bit in the bit-string produced by concatenating all binary representations within M
  - The final bit represents the value of the i-th bit

Congratulations! You are now all on the way to one day becoming the AIOC Director of Training!

Subtask 4 – 18 More Points

- $1 \leq N \leq 32$
- Each encoded integer must be between 0 and 255, inclusive
- You are allowed to call $send(x)$ at most $10N$ times

Subtask 4 – 18 More Points

- $1 \leq N \leq 32$
- Each encoded integer must be between 0 and 255, inclusive
- You are allowed to call $send(x)$ at most $10N$ times

The solution:

- We combine our Subtask 1 and Subtask 2 observations
- The bit-string now has $32 * 8 = 256$ bits total
  - Instead of sending the position and value of every bit, we just send the positions of each set bit (ie. The locations of all the 1s)
- Alternative binary formats:
  - The first 5 bits represent the index of M we will modify
  - The final 3 bits represent the position of the bit in M_i that we wish to set
- There are many ways to structure our numbers!

Subtask 5 – up to 19 More Points

- $16 \leq N \leq 64$
- Each encoded integer must be between 0 and 255, inclusive
- You are allowed to call $send(x)$ at most $15N$ times
- For the i-th test case, $P_i = \frac{L_i}{N_i}$ (ie. The ratio between the number of times you call $send(x)$ and the length of M)
- Let $P = \max P_i$ across all cases:
  - If $P \leq 5$, you get 19 points
  - If $5 < P \leq 6$, you get 18 points
  - If $6 < P \leq 7$, you get 17 points
  - If $7 < P \leq 15$, you get $\lfloor 1 + 2(15 - P) \rfloor$ points
  - Otherwise, you get 0 points

The Subtle Art of Giving Up

- The bit-string now has $64 * 8 = 512$ bits total, so we cannot afford to send the information pertaining to each bit
- Instead, we can send information pertaining to each pair of bits
- Pair $i = (M_{2i}, M_{2i+1})$ – note that there are 4 possible values of this pair
- Then, for each pair in the bit-string, we send its position and value
  - Problem: encoding the position takes 8 bits
- Instead of encoding the value, we just send the position multiple times corresponding to the values of the pair:
  - Sending the i-th pair no times corresponds to 00
  - Sending it once corresponds to 01
  - Sending it twice corresponds to 10
  - Sending it three times corresponds to 11

- This achieves $P = 12$ and we get 7 points for it

The Subtle Art of Giving Up

- There are some observations that can be made to improve the previous solution and achieve $P \approx 6$, which will give us 17 points for this subtask
  - I leave all further improvements to these algorithms as an exercise to you

- The problem with sliding-scale tasks is that the closer you get to the full solution, the more time you spend trying to get a very small amount of points
  - They can be a real time-sink in contests, and there comes a point where they are no longer a good use of your time
  - If you have some easy subtasks you haven’t solved, those take priority over a small amount of points on a problem
- Knowing when to give up on a problem is very important, and can be learned by practicing good exam technique
  - Use the trial exams to your advantage!

Randomised Algorithms

- In some cases, it is possible to solve a problem using a *Randomised Algorithm*, even if the problem is not related to any random processes
- A Randomised Algorithm is some kind of process which depends on one or more random variables

- Las Vegas Algorithms:
  - Keep running until it finds the optimal solution
  - Solution is optimal, but runtime can vary hugely

- Monte Carlo Algorithms:
  - Runs according to some random variables and finds *a* solution
  - Solution has a chance of being optimal, but runtime is fixed
- Since we tend to be constrained by time in Informatics, we prefer Monte Carlo Algorithms

Monte Carlo Algorithms

- Suppose that a Monte Carlo Algorithm has a probability P of returning an optimal solution
- If we submit this algorithm, on average we will only pass a proportion of the subtasks equal to P
- However, if we run the algorithm N times in the same programme, the probability of success is $1 - (1 - P)^N$
  - For large N, there is a great chance that our algorithm will produce an optimal solution

Problem: Colinear Points

- Given N points on a 2D plane, find the equation of a line passing through at least $0.25N$ of the points ($N \leq 10^6$)
- You are guaranteed that such a line exists

Problem: Colinear Points

- Given N points on a 2D plane, find the equation of a line passing through at least $0.25N$ of the points ($N \leq 10^6$)
- You are guaranteed that such a line exists

- If we select a point randomly, there is a probability of at least 0.25 that the point exists on the line
- If we select two points randomly, there is a 0.0625 chance that they are both on the line
  - This can be verified in $O(N)$
- We have an $O(N)$ Monte Carlo algorithm
- Repeated 20 times: $\Pr(success) \approx 0.7249$
- Repeated 100 times: $\Pr(success) \approx 0.9984$

Writing a Randomised Algorithm

- Monte Carlo Algorithms generally take the form of selecting some data points randomly, building a solution based off their values, and verifying it
- You must determine the probability of each iteration of your algorithm succeeding
- Consider the most “Realistic Pessimistic Case”
  - ie. How many times should I have to run this algorithm in the worst case before it comes up with an optimal solution?

The Birthday Problem

- How many people would you need to gather together for there to be a reasonable chance that at least two of them have the same birthday?
  - For simplicity, assume that there are 365 days in a year and that the day someone is born on can be modelled by a continuous uniform distribution

- The probability that a group of n people have no birthdays in common is:

$$\frac{365}{365} \times \frac{364}{365} \times \cdots \times \frac{365 - n + 1}{365}$$

- It happens that when $n \approx 23$, there is a $\sim 50\%$ chance that there will be a double-up of birthdays in the group

More broadly…

- If you are given N data points as part of your input, and you need to find two points satisfying some property, there is a moderate probability that you will find a pair by sampling around $\sqrt{N}$ points
- Further reading: https://en.wikipedia.org/wiki/Birthday_problem#Square_approximation

Warning: Adversarial Graders

- Some interactive tasks have graders which will modify their data according to what you ask them
  - They will never lie to you, but they will make everything as difficult for you as possible
  - Imagine playing I-Spy with a sibling who has a list of things beginning with their chosen letter, and will only let you win when they have nothing else to make your ‘interaction’ any more difficult for you
- You **cannot** use randomised algorithms on these tasks: the grader will make them fail
- Chances are that only a deterministic solution will pass these problems
- For instance, Spies III on Orac 2 is adversarial – the same problem on Orac 1 can be solved using a Monte Carlo Algorithm

Warning: Setting a Random Seed

- For local testing and submitting to all competitions in the Australian Informatics Olympiad Program, manually setting your seed is good
  - It allows your computer's Pseudo-Random Number Generator to come up with the same numbers every time you run it
  - This is great for testing

- However, if you submit to a platform like CF where Hacking is a part of the competitions, you run the risk of being hacked if you use a Randomised Algorithm:
  - People have figured out how to break solutions which generate pseudo-random numbers from a custom seed
  - Apparently in some cases it is very easy to come up with a case that breaks specific random seeds on certain problems

How to Randomise your Seed

- `srand(time(NULL))`
  - Sets your random seed based on the current second
  - For most intents and purposes, this is fine
  - However, there are some very smart people out there who have probably figured out how seeding works based on the current second on the Codeforces platform; we need to do better

- `srand(chrono::steady_clock::now().time_since_epoch().count())`
  - Sets your random seed based on the current millisecond/nanosecond
  - Much better!

Your Problemset

1. Cave
2. Combo
3. Parrots
4. River
5. Quicksearch
6. Pizza Problems