UAPC 2024

Solution Sketches

Thanks!

Problem setters, testers, and judges:

- Ian DeHaan
- Answer
- Grayden Price
- Noah Weninger
- Maple

Kattis Team: Greg Hamerly, Fredrik Niemelä

UACS - Event site organization and point-of-contact for sponsors.

And Thanks To Our Sponsors!

Division 2 Exclusive Problems

Keys, Phone, Wallet - Zac Friggstad

D2 Solves: 49

Fastest Solve:
plugged in 0:02:38

Many ways to do it. Here's one.

```
items = {"keys", "phone", "wallet"}
N = int(input())

taken = [input() for _ in range(N)]

items.difference_update(taken)

if items: print(*sorted(items), sep="\n")
else: print("ready")
```

Candy Store - Noah Weninger

D2 Solves: 44

Fastest Solve:
sigmunf freud: freddy fast bear har har harhar 0:11:59

A few ways as well.

One is to create a dictionary mapping initials to a list of names with those initials.

Then for each query, look up the initials in the dictionary and look at the corresponding list (if any) to answer the query.

Tip - one of many way to get the initials of a name:

```
first, last = fullname.split()
initials = first[0]+last[0]
```

Birthday Candles - Zac Friggstad

D2 Solves: 12

Fastest Solve:
Singh++ 1:03:31

1) Sort each guest's candles.

2) Blow out the 1st candles from each guest first, then the 2nd candles from each guest next, then the 3rd and so on.

3) If you can't blow out all of the jth candles of the guests, sort these jth candles and blow out what you can in this order.

Running time: O(N log N) where N is the total number of candles.

Problems Common to Both Divisions

Pencil Crayons - Zac Friggstad

D1 Solves: 23

D2 Solves: 43

Fastest Solve:
cannot cpp
0:02:04

You only have to remove duplicates from each box.

Trick: can count the number of duplicates in a list `L` easily:

```
len(L) - len(set(L))
```

But, again, there are many other ways to do it.

Dinner Time - Zac Friggstad

D1 Solves: 17

D2 Solves: 11

Fastest Solve:
Blahaj 0:22:45

Needs some thought.

When you pass the potatoes K steps, if it is passed to the person who holds the gravy (even if just momentarily) then add 1 to the final answer. Check quickly with:

```
potatoes_pos < gravy_pos and gravy_pos <= potatoes_pos + K
```

When you pass the gravy K steps, if the potatoes are currently futher than the gravy then the number of new people that get gravy on their potatoes is

```
min(K, potatoes_pos - gravy_pos)
```

Ray Chasing - Zac Friggstad

D1 Solves: 14

D2 Solves: 14

Fastest Solve:
ln(x) 0:18:17

Tip: translate the coordinates so the ray is emitted from the origin.

Then you can tell if it hits the left/right wall and top/bottom wall of the box simply by looking to see if the coordinates the ray passes through are positive/negative.

If a ray would hit two sides according to this check, which one would it hit first? You can solve for the “time” it intersects a wall and see which is smaller (or if the same, it hits a corner).

Alternatively (my solution), can use 2D cross products to see if the corners lie to the left/right of the ray (or on the ray).

Evacuation - Grayden Price

D1 Solves: 6

D2 Solves: 1

Fastest Solve: Brandon Fuller 0:35:22

For each bridge, compute the time it becomes unusable by stepping through the tornado's path.

Then run Dijkstra's algorithm except you can use a bridge (i.e. enqueue it in the heap) if it remains usable throughout the entire crossing.

Dijkstra's algorithm is a well-known algorithm and is common in programming. It finds the fastest way to go between two locations in an graph where edges have different times to traverse.

Triangle Trees - Noah Weninger

D1 Solves: 4

D2 Solves: 1

Fastest Solve:
Blahaj 1:02:17

Can solve in linear time using depth-first search or breadth-first search.

During the search, when you reach a new vertex then assign it the smallest number that has not been assigned to one of its neighbours.

If the graph has no triangles, this will use 2 colours (unless the graph has no edges, in which case it just uses 1 colour).

If the graph has triangles, you can prove that the search never has more than 2 coloured neighbours of a node when you first visit it using the property there are no cycles of length > 3 (basically because the graph looks like nodes and 3-cycles glued together in a tree-like fashion) - so only 3 colours will be used.

Note that the graph might be disconnected!

Lines of X - Zac Friggstad

D1 Solves: 2

D2 Solves: 0

Fastest Solve: ln(x) 2:34:57

Trying all ways of filling in the dots is far too slow.

For a line L that does not contain an 0, there are $2^{y(L)}$ ways to fill in dots if there are **y(L)** dots not spanned by L. If we sum this over all lines, we overcount the number of ways to get at least one line of x.

In general, Let **S** be a nonempty subset of lines. There are at most 18 lines (8 + 8 + 2) so there are less than 2^{18} subsets to try.

Let c(S) be 0 if the lines in **S** span a grid cell with an o. Otherwise, let it be $2^{y(S)}$ where **y(S)** is the number of empty cells not covered by a line in S. The answer is then the sum of (-1)^{|S| + 1} * c(S) over nonempty subsets **S** by the inclusion/exclusion principle.

Special case: if the grid already has a line of X then it is just 2^d where d is the number of dots.

Power String Matching - Zac Friggstad

D1 Solves: 2

D2 Solves: 1

Fastest Solve: ln(x) 1:54:20

Hardest in Div 2, really hard even for Div 1.

Suppose we can answer the following query in O(1) time:

Does `s[i1:j1] == t[i2:j2]`?

We can do this by computing for every `i,j` the longest common prefix of `s[i:]` and `t[j:]` by just using a simple loop through the characters: $O(n^3)$ total for all `i,j` (there are faster ways, but they aren't needed).

Using this, we can solve the problem with dynamic programming.

Let **f(i,j)** be true iff the first i characters of `s` can generate the first `j` characters of t. Base cases i = 0 and j = 0 are immediate.

Power String Matching - Zac Friggstad

D1 Solves: 2

D2 Solves: 1

Fastest Solve: ln(x) 1:54:20

For the recurrence:

- First try just ignoring the i'th character of s[:i] (i.e. check **f(i-1,j)**).
- Then try having a suffix of **s[:i]** cover a suffix of **t[:j]** (perhaps with repetitions). That is, for each 1 <= k <= min(i,j) we try all p = k, 2k, 3k, ...
- If s[i-k:i] matches t[j-p,j-p+k] we recursively check **f(i-k,j-p)**, otherwise we quit the loop.

Running time (if N = M):
$O(N^2)$ subproblems, O(N/k) different values of p per k for a running time of
$O(N^2 * sum_{k=1}^N N/K) = O(N^3 * log(N))$

Division 1 Exclusive Problems

Rerouting Rapids - Ian DeHaan And Noah Weninger

D1 Solves: 3

Fastest Solve: Brandon Fuller 1:39:16

Binary search the answer.

To answer a query if a value of Q is possible, do a DFS of the tree starting at the end of the river. The DFS will reroute rapids within a subtree so no node has > Q incoming rapids in a way that minimizes the number of rapids that must leave the subtree.

To process a node, get all rapids that must leave the subtrees rooted at its children. Have Q of them end at this node (or all of them if there are < Q). The rest plus one more for this node go outside of this node.

The query Q is successful if at most Q rapids reach the end of the river in this DFS.

D1 Solves: 2

Fastest Solve: Brandon Fuller 1:33:09

Number Magic - Answer and Maple

Bidirectional search

For some value 1 <= **T** <= 32, generate all $O(2^{T})$ values that *can be generated from* the original number **N** by recursively trying all ways to apply **T** steps of magic.

Then for each query **M**, generate all values that *can generate* **M** by applying at most 32-**T** steps of "reverse magic". That is a, number **X** can be generated by 2**X** and 2**X**+1 and also by at most one number **Y** where we add the all-1 digit string to **Y**. Be careful about overflow in C++. See if any of these numbers was in the set of numbers generated from N. This generates $O(3^{32-T})$ numbers.

Running time: $O(2^{T} + \mathbf{Q} * 3^{32-T})$. The optimal value to set **T** is 24 or so.

Gopher Residence - Ian DeHaan

D1 Solves: 0

Fastest Solve:

First, consider the part where the gophers have already decided whether they actually want to live and the city holds the lottery.

We claim the lottery will always pick the same number of gophers no matter the random choices that are made, so all that matters is the gopher's random choice about whether they actually want to live in the residence.

Can prove this as follows: For any choices of gophers S, T that are feasible where $|S| < |T|$, there exists a gopher in T but not in S that could be added to S while keeping it feasible. Not hard to prove.

Gopher Residence - Ian DeHaan

D1 Solves: 0

Fastest Solve:

To compute this number from a given set of interested gophers, we do it recursively. For each room, compute the maximum number of gophers that can be put in the rooms below it. Add the interested gophers to this room. Then while this room's capacity is violated, arbitrarily discard gophers until it is ok. This takes a proof similar to the statement on the previous slide.

Back to the original problem. For each room we use dynamic programming to compute the probability that exactly k gophers living under this residence (including this residence). When combining these probabilities in a recursive way, just clamp the values k to the room's capacity.