2019 ICPC North Central USA Regional Contest Solution Outlines

Greg Hamerly, modified for NCNA by Bryce Sandlund

November 9, 2019

F– Pulling their weight

Description

Given a list of weights $(w_1, w_2, \ldots, w_n)$, find the smallest integer t where

$$\sum_{w_i<t} w_i = \sum_{t<w_i} w_i$$

i.e. sum of weights $< t$ equals sum of weights $> t$.

E.g. for $(1, 2, 3, 6)$ the best t is 4.

F– Pulling their weight

Description

Given a list of weights $(w_1, w_2, \ldots, w_n)$, find the smallest integer t where

$$\sum_{w_i < t} w_i = \sum_{t < w_i} w_i$$

i.e. sum of weights $< t$ equals sum of weights $> t$.

E.g. for $(1, 2, 3, 6)$ the best t is 4.

Solution

Insight: t is either a given weight, or a weight plus one.

Consider each unique weight (or weight plus one) as a possible t, in sorted order.

For each possible t, determine if it is the solution.

F– Pulling their weight

Feasible Solutions

When considering a certain t, if any weights are equal to t, we can just ignore them.

Some lists of integers are invalid inputs:

- If we just make up a list of weights, it might not have a solution.
  - E.g. there is no t for $(1, 2, 4)$.
- But the problem guarantees that every given input has a solution (so $(1, 2, 4)$ cannot occur).

F– Pulling their weight

Solution Strategy Details

Define:

- $(s_1, s_2, \ldots, s_k)$ are the sorted *unique* weights
- $M(w)$ is the number of times w appears in the input (multiplicity)
- $P(i) = \sum_{j=1}^{i} s_i \cdot M(s_i)$ is a prefix sum
- $S = P(k)$ is the sum of the whole sequence

For each i in 1 to k:

- if $P(i) = S - P(i) + s_i \cdot M(s_i)$, then $t = s_i$
  - i.e. prefix and suffix sums are the same, ignoring all values equal to t
- if t is not any s_j and $P(i) = S - P(i)$, then $t = s_i + 1$
  - i.e. prefix and suffix sums are the same; no weight is equal to t

Runtime: $O(n \log n)$ for sorting, and $O(n)$ for everything else.

G– Birthday Paradox

Description

Given a list of integers, find the probability of this list occurring.

Here each integer represents the number of people sharing some birthday.

Assume each person has 1/365 chance of any given birthday.

G– Birthday Paradox

Description

Given a list of integers, find the probability of this list occurring.

Here each integer represents the number of people sharing some birthday.

Assume each person has 1/365 chance of any given birthday.

Solution

The solution can be derived using the multinomial probability distribution, and counting the number of possible reorderings.

G– Birthday Paradox

Solution Strategy

Let $p = 1/365$ in what follows, and input values be $X = (x_1, x_2, \ldots, x_k)$.

The multinomial probability distribution P_M answers the question "What is the probability of observing all x_i on *fixed* days?" That is, the first k days of the year.

$$P_M(X) = \frac{n!}{\prod_{i=1}^{k} x_i!} p^n$$

where $n = \sum_{i=1}^{k} x_i$.

However, this isn't enough – we cannot assume these people's birthdays are the first k days of the year.

So how many different ways can we arrange their birthdays?

G– Birthday Paradox

Solution Strategy

There are $\binom{365}{k}$ ways to choose k days out of 365.

Additionally, we can reorder the x_i's:

- if the x_i's are unique, then there are $k!$ orderings;
- if some x_i's are the same, then there are fewer...

Let $C = (c_1, c_2, \ldots, c_j)$ be the *frequency counts* of each x_i value. E.g. if $X = (1, 1, 1, 5, 5)$ then $C = (3, 2)$. Then the number of unique ways to reorder the x_i's is

$$\frac{(\sum c_j)!}{\prod(c_j!)}$$

which also arises from the multinomial distribution.

G– Birthday Paradox

Solution Strategy

Putting this all together, the answer is:

$$P_M(X) \cdot \binom{365}{k} \cdot \frac{(\sum c_j)!}{\prod(c_j!)}$$

We can do all of this in log-space (the `lgamma` function is useful here).

Running time: $O(k)$ for k inputs.

J– This Ain't Your Grandpa's Checkerboard

Description

Given a grid of `B` and `W` characters, is it "valid"?

Invalid: unequal numbers of `B` and `W` values in a row/column, or a run of three of the same values in a row/column.

J– This Ain't Your Grandpa's Checkerboard

Description

Given a grid of `B` and `W` characters, is it "valid"?

Invalid: unequal numbers of `B` and `W` values in a row/column, or a run of three of the same values in a row/column.

Solution

Read the input; check each row/column; report the answer.

Transpose: you can check each row, transpose the whole grid, then check each row (which was a column).

Running time: $O(n)$

H– On Average They're Purple

Description

Given a graph, how can Anna color the edges (red/blue) to force Bob to change colors the maximum number of times? Bob is trying to minimize the number of color changes while going between two designated nodes.

At first glance this feels like a difficult problem. But there's a simple solution hiding inside.

H– On Average They're Purple

Description

Given a graph, how can Anna color the edges (red/blue) to force Bob to change colors the maximum number of times? Bob is trying to minimize the number of color changes while going between two designated nodes.

At first glance this feels like a difficult problem. But there's a simple solution hiding inside.

Solution

Just find the shortest path from source to destination using breadth-first search. If the path length is k then the answer is $k - 1$.

H– On Average They're Purple

Solution Strategy

Proof idea that the shortest path finds the answer:

Consider starting at the source node. Anna colors every edge touching that node red (say), to maximize the chance of changing colors at a subsequent edge.

Now move one hop away, along any edge. Anna wants to color all newly-reachable edges blue, to force a color change.

This argument generalizes to k hops away.

Thus, the shortest path of length k changes colors $k-1$ times, which is the most Anna can force Bob to do.

Runtime for BFS: $O(N+M)$ for N nodes and M edges.

B– Code Names

Description

Given a list ($n \leq 500$) of anagram words with no duplicate letters, find the size of the largest "swap-free" set.

Two words are NOT swap-free if they differ by swapping two letters (e.g. `abcdef` and `aecdbf` are NOT swap-free).

B– Code Names

Description

Given a list ($n \leq 500$) of anagram words with no duplicate letters, find the size of the largest "swap-free" set.

Two words are NOT swap-free if they differ by swapping two letters (e.g. `abcdef` and `aecdbf` are NOT swap-free).

Solution

This appears to be an NP-hard problem – finding a maximum independent set. But there is more structure to the problem.

We can reduce this to a maximum bipartite matching (maximum flow) problem.

Identify the bipartite structure, construct the edges, find a maximum matching using network flow.

B– Code Names

Solution Strategy

Define: two words are neighbors if they are NOT swap-free.

Identify the bipartite graph structure.

- Construct a graph with edges between neighbors.
- Each pair of words are anagrams – think permutations.
- Permutations have *parity* – equal to the parity of the number of *inversions*.
- Swapping (any) two letters changes the parity. Thus,
    - Two words *cannot be neighbors* if they have the same parity.
    - Two words *might* be neighbors if they have different parity.

Construct the graph in $O(N^2)$ time.

B– Code Names

Solution Strategy

Once we've defined the bipartite structure on N words, the maximum independent set can be found by taking $N - M$ where M is the maximum number of matches in the bipartite graph.

Run maximum bipartite matching using any algorithm in $O(N^3)$ time worst-case to find M.

D– Some Sum

Description

Given a number N, is it possible that the sum of N consecutive numbers is even, odd, or both?

D– Some Sum

Description

Given a number N, is it possible that the sum of N consecutive numbers is even, odd, or both?

Solution

Since the bounds on the values of the numbers are small (1 to 100), just try all possible sums of length N. Running time: $O(N)$.

There's also an $O(1)$ time solutions by inspecting the value of $N \bmod 4$:

- if $N \bmod 4 = 0$, the answer is even
- if $N \bmod 4 = 2$, the answer is odd
- otherwise, the answer is either

This is because the sum of 4 consecutive numbers is even.

I– Full Depth Morning Show

Description

You are given a tree with N nodes, where every node has a tax value t_u and each edge has some weight w_i. The cost of a path between nodes u and v is equal to $(t_u + t_v)\,\mathrm{dist}(u, v)$. For each node u, compute the sum of the costs of all paths to all other nodes v.

I– Full Depth Morning Show

Description

You are given a tree with N nodes, where every node has a tax value t_u and each edge has some weight w_i. The cost of a path between nodes u and v is equal to $(t_u + t_v)\,\mathrm{dist}(u, v)$. For each node u, compute the sum of the costs of all paths to all other nodes v.

Solution

The above expression can be broken up into $t_u\,\mathrm{dist}(u, v) + t_v\,\mathrm{dist}(u, v)$. Compute for some arbitrary root in $O(N)$ time. Then computing the answer for a neighboring node can be done in $O(1)$ time.

I– Full Depth Morning Show

Formula

Fix some node u as the root, compute two quantities:

$$a_u = \sum_v \mathrm{dist}(u, v)$$

$$b_u = \sum_v t_v \, \mathrm{dist}(u, v)$$

I– Full Depth Morning Show

Formula

Fix some node u as the root, compute two quantities:

$$a_u = \sum_v \operatorname{dist}(u, v)$$

$$b_u = \sum_v t_v \operatorname{dist}(u, v)$$

Solution Strategy

Then the answer for node u is just $t_u a_u + b_u$.

How do we compute $a_{u'}$ and $b_{u'}$ for some neighbor u' of u?

I– Full Depth Morning Show

Formula

Fix some node u as the root, compute two quantities:

$$a_u = \sum_v \operatorname{dist}(u, v)$$

$$b_u = \sum_v t_v \operatorname{dist}(u, v)$$

Solution Strategy

Then the answer for node u is just $t_u a_u + b_u$.

How do we compute $a_{u'}$ and $b_{u'}$ for some neighbor u' of u?

I– Full Depth Morning Show

Formula

Fix some node u as the root, compute two quantities:

$$a_u = \sum_v \text{dist}(u, v)$$

$$b_u = \sum_v t_v \, \text{dist}(u, v)$$

Solution Strategy (continued)

Let w be the length of the edge between u and u'. For all nodes in the subtree of u' when the tree is rooted at u, their distance to the root decreases by w. For all other nodes, the distance increases by w. If we let size(u) be the size of the subtree rooted at u, then

$$a_{u'} = a_u + w(N - \text{size}(u)) - w \, \text{size}(u)$$

I– Full Depth Morning Show

Formula

Fix some node u as the root, compute two quantities:

$$a_u = \sum_v \mathrm{dist}(u, v)$$

$$b_u = \sum_v t_v \, \mathrm{dist}(u, v)$$

Solution Strategy (continued)

Similarly, if we let $\mathrm{tax}(u)$ be the sum of the tax values of all nodes in the subtree rooted at u, then

$$b_{u'} = b_u + w(\mathrm{tax}(\mathrm{root}) - \mathrm{tax}(u)) - w \, \mathrm{tax}(u)$$

I– Full Depth Morning Show

Formula

Fix some node u as the root, compute two quantities:

$$a_u = \sum_v \operatorname{dist}(u, v)$$

$$b_u = \sum_v t_v \operatorname{dist}(u, v)$$

Runtime

Since these values can be updated in $O(1)$, walking and updating the tree and computing all values takes $O(N)$ time in total.

A– Weird Flecks, But OK

Description

Given N points in 3 dimensions, find the smallest circle that encompasses all of them in one of the three orthogonal planes.

A– Weird Flecks, But OK

Description

Given N points in 3 dimensions, find the smallest circle that encompasses all of them in one of the three orthogonal planes.

Solution

Apply an efficient algorithm for smallest-circle three times (once for each plane).

Welzl's algorithm runs in $O(N)$ time (expected). Other approaches also work.

A– Weird Flecks, But OK

Solution Strategy

Preliminaries: a circle can be uniquely defined by either two antipodal points or three non-collinear points.

The math for finding the circle for three points is left to the reader.

Naive approach: for each plane, try all circles defined by every $O(N^2)$ pair and $O(N^3)$ triple. For each circle, check if all N points are inside.

This approach is $O(N^4)$, too slow since $N \leq 5\,000$.

A– Weird Flecks, But OK

Solution Strategy: Binary Search + Angle Sweep

Once we've reduced the problem to two dimensions, we can binary search on the radius of the minimum enclosing circle.

For any valid circle which covers all the points, we can translate it so at least two input points lie on the boundary of the circle. Then, for each input point, consider all circles with that point on its boundary. Each other point is included in some range of angles. If there exists some angle that includes all points, then this radius is attainable.

Runtime: $O(N^2 \log 1\,000)$.

A– Weird Flecks, But OK

Solution Strategy: Ternary Search for the center

Consider the following function: $f(x, y)$ is the minimum radius of a circle that has center at (x, y) and covers all the points in the input. Evaluating $f(x, y)$ once can be done in $O(N)$ time.

We've now reduced our problem to finding the minimum value of $f(x, y)$. We can do this by doing two nested ternary searches: for a fixed x, the value of $f(x, y)$ is convex.

One heuristic to speed up this solution is to first take the convex hull of the input points before computing values of $f(x, y)$.

Runtime: $O(N \log^2 1\,000)$.

A– Weird Flecks, But OK

Solution Strategy: Welzl's algorithm

Start with all points in set P and empty set R. Here R defines a set of boundary-defining points (up to 3).

Welzl(P, R):

- If P is empty or $|R| = 3$, return smallest circle defined by R.
- Choose a point $p \in P$ at random.
- Recurse with $P \setminus \{p\}$ and R.
- If the resulting circle C contains p, return C.
- Otherwise, try again (recurse) with $P \setminus \{p\}$ and $R \cup \{p\}$, and return that circle.

Running time: $O(N)$ expected time.

C– New Maths

Description

Given a number N, find the smallest integer a such that

$$a \oplus a = N.$$

C– New Maths

Description

Given a number N, find the smallest integer a such that

$$a \oplus a = N.$$

Solution

Recursively backtrack through possible a from most significant digit to least significant digit, or vice versa.

C– New Maths

Solution Strategy

Insight: The most significant digit of N is determined by the most significant digit of a; similarly, the second most significant digit of N is determined by the two most significant digits of a.

Let a_k be the most significant digit of a. Then since $a_k^2 \equiv N_{2k} \pmod{10}$, this limits the choice of a_k to one of at most two choices.

A similar property holds for a_i: the choice of a_i is severely constrained by the choice of a_j for all $j > i$ and modular congruences. Testing can confirm recursive backtracking through feasible digits of a is fast enough.

E– Early Orders

Description

Given a list of integers $x_1, x_2, \ldots, x_n$ and a number k, find the lexicographically smallest subsequence of x that contains each integer from 1 to k exactly once.

E– Early Orders

Description

Given a list of integers $x_1, x_2, \ldots, x_n$ and a number k, find the lexicographically smallest subsequence of x that contains each integer from 1 to k exactly once.

Solution

Iteratively construct the lexicographically smallest subsequence by considering each x_i one at a time.

E– Early Orders

Solution Strategy

There are two cases to consider when considering x_i.

1. The given integer is already present in our subsequence. We do nothing.
2. The given integer is not present in our subsequence. This is the interesting case to consider.

   While our tentative subsequence is not empty, we will compare the given integer to the last integer in our subsequence. If the given integer is larger than the last integer in our subsequence, then append it to the end of our subsequence. Otherwise, it is smaller than the last integer. We can safely remove this integer if and only if there is another appearance of this integer that will be considered later in the sweep. We repeat this removal process until we can no longer remove an integer, at which we perform the append.

K– Solar Energy

Description

Given a set of stars that emit energy with a launch angle a according to the formula $\max(0, T_i - s_i \cdot \mathrm{dist}(a_i, a))$, determine the launch angle a that maximize the total energy.

K– Solar Energy

Description

Given a set of stars that emit energy with a launch angle a according to the formula $\max(0, T_i - s_i \cdot \mathrm{dist}(a_i, a))$, determine the launch angle a that maximize the total energy.

Solution

Consider each star simultaneously by running a sweep of all stars, calculating the total energy at every point. Point the spaceship at the star with the highest energy.

K– Solar Energy

Solution Strategy

The first observation is that since the energy function is (piecewise) linear in the angle, the derivative is constant. This implies the best angle to launch the spacecraft will be directed at a star.

There are 10^5 stars, thus we cannot spend linear time to calculate the energy achieved from launching the spacecraft at a particular star. Instead, we must do a sweep, calculating the energy for all stars in one go. For each star, we can determine at which angle the star starts contributing to the energy of the launch, and similarly when that energy peaks and stops contributing. Again, since the derivative is constant, we can maintain the change in energy per radian moved, making it possible to find all energies in $O(n \log n)$ total time, after sorting by radian. Care must be taken to handle the circle properly, and stars that contribute energy no matter the angle must be special-cased.