UKIEPC 2024  

Summary and solution outlines

Problem Solutions

Amalgram

?? correct • solved at: ??:?? by
??
??

Overview

- Create a word containing:
  - All of the letters of word1
  - All of the letters of word2
- If a letter occurs:
  - N times in word1
  - M times in word2
  - Then it should occur at least max(N,M) times in word3

Amalgram

Techniques

- Hashmaps
- Sorting

Algorithm

- We need a way to take the superset of both strings
- One method inspired by merge-sort:
  - Sort both of the strings
  - Create two pointers i,j. For as long as i<n, j<m:
    - If a[i] == b[j]: Print a[i] and advance **both** i++, j++
    - Otherwise if a[i] < b[j]: Print a[i] and advance i++
    - Otherwise: print b[j] and advance j++
- Other methods exist
  - Like counting frequency in a hashmap for each string

Budget Analysis

0 correct • solved at: ??:?? by
??
??

Overview

- You are given a formula for linear regression and must optimise the parameters to the formula
- The optimisation takes place on many different subarrays of the same list
- Invent a way to do this efficiently (faster than O(N) in the size of the list)

Budget Analysis

Techniques

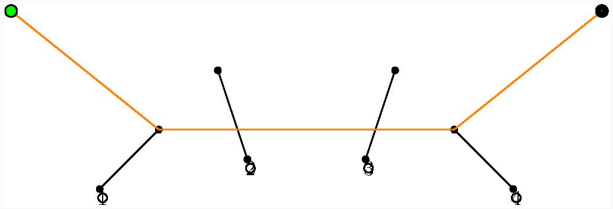
- Calculus
- Prefix sums

Algorithm

- Taking the partial derivatives and solve the resulting linear equations system.
- One could see that for the linear regression of y on x (even if it is regularized) one needs to know the following:
  - sum(x_i), sum(y_i), sum(x_i^2) and sum(x_i * y_i).

- Given that the given x_i and y_i are constant, we could precompute the prefix sums and get the regression coefficients for each query in O(1). Computing the prediction also takes constant time.

Overall time complexity of the solution is O(n+m)



Cross Country

0 correct • solved at: ??:?? by
??
??

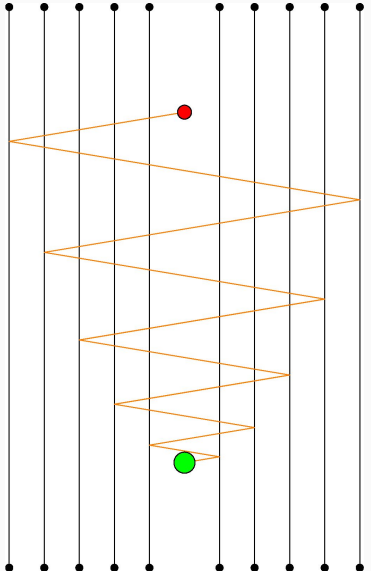
Overview

- Draw the shortest line going through some line segments A, B, C, D, … Z
- Going through a line segment multiple times is OK, but you must go through the segment in the right order too.

Cross Country

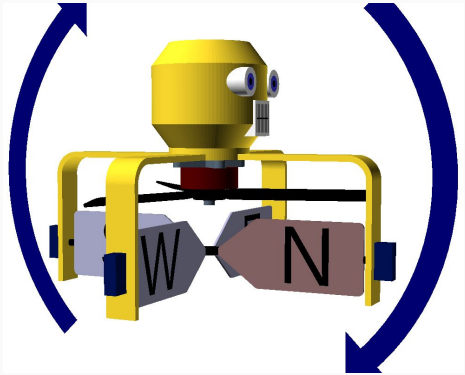
Techniques

- Geometry
- Dynamic programming



Algorithm

- When we encounter a line segment on our path, there are three potentially-optimal moves:
  - Keep going in the same direction
  - Keep going in a **reflection of** the same direction
  - Completely change direction
- The shortest path will only **completely** change direction if it touches the endpoint of a line
  - Otherwise the path could be shortened by changing angles
- Eliminate the reflection aspect by generating 2^N scenarios of “virtual line segments” where reflection is already simulated.
- Then it’s a shortest path problem:
  - Two segment endpoints have a connecting edge **iff** the line between them goes through all intermediate segments.



Drone Operator

0 correct • solved at: ??:?? by
??
??

Overview

- Given is a system of 3 equations in 4 variables “n”, “e”, “s”, “w”
  - This system is underspecified, but relates all of the variables to each other.
- The objective is to make the maximum of |n|, |e|, |s|, |w| as small as possible under the constraints.

Drone Operator

Techniques

- Greedy algorithms
- Sorting

Algorithm

- Our equations have one spare degree of freedom:
  - `p = e - w` ::: **`w = e - p`**
  - `r = n - s` ::: **`s = n - r`**
  - `y = n + e + s + w` ::: **`y = 2n + 2e - r - p`**
- In a solution where max(|n|,|e|,|s|,|w|)=T, **two** variables have |v|=T.
  - Otherwise, max(|v|) could be decreased without increasing T.
  - The options are: (|n|=|e|, |n|=|s|, |n|=|w|, |e|=|s|, |e|=|w|)
- So either 1. Add each possibl equation into the system and solve algebraically, taking the minimum solution as the answer.
- Or 2. Binary search on the value of T, and for every pair in (n,e,s,w) set both elements to T then check if this gives a valid set of equations.

Eradication Sort

0 correct • solved at: ??:?? by ?? ??

Overview

- Remove some items from an unsorted list, to make it sorted
- For each gap of width W you leave in the resulting list, you pay a cost of W^2
  - What is the minimum total cost you can achieve?

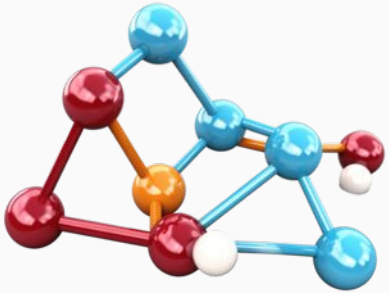
Eradication Sort

Techniques

- Dynamic programming
- Binary indexed trees
- Convex line functions

Algorithm

- Straightforward dynamic programming gives a (too slow) solution:
  - cost[x] = the cost in case the last item **kept** so far is y[x]
  - cost[x] = min(cost[i] + $(x-i)^2$ for-all [i < x] st. y[i] ≤ y[x])
    - Complexity of implementation is $O(N^2)$
- We need ways to find the candidate values, and to get the minimum among them, quickly.
  - cost[x] = cost[i]+$(x-i)^2$ can be rearranged as a line equation:
  - cost[x]-x^2 = cost[i]+i^2 - (2i)x
  - If we ignore (y[i] ≤ y[x]), we can store the lines in a list sorted by gradient, and use binary/two-pointers search to find the best line for a given x.
- To address the (y[i] ≤ y[x]) constraint, we'll create a binary-indexed tree of these lists. After calculating cost[x], update lines[y[x]...inf].



Finding Proteins

0 correct • solved at: ??:?? by
??
??

Overview

- The coordinates of some points with high dimensionality are given
- We will build a subset of these points and call it K.
  - K is built by repeatedly taking the point which is **farthest** from the **closest** point in K
- Find a way to build the set K quickly.

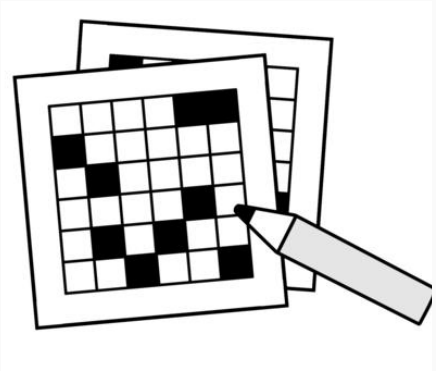
Finding Suspicious Proteins

Techniques

- Ad-hoc optimisation
- Precalculation

Algorithm

- When choosing the next value V (which happens K times):
  - We need to calculate min(dist(V, k[i]) forall existing k
  - This is (O(L*K))) - too slow
    - However, min(dist(V, k[0...i])) is the same as:
      - min(dist(V, k[0...i]), dist(V, k[i+1]))
      - The only thing that changes is the addition of the new element of K, and this can be updated later.
  - We can maintain the current minimum distances to so-far-unselected items as dist[0...n-1]
    - Every time we choose a new element of K, update all the distances with min(cur, dist(k))



Grid Search

0 correct • solved at: ??:?? by
??
??

Overview

- You are given a 2D word and a 2D grid in which to search for that word
  - Highlight the places where the word occurs in the grid.

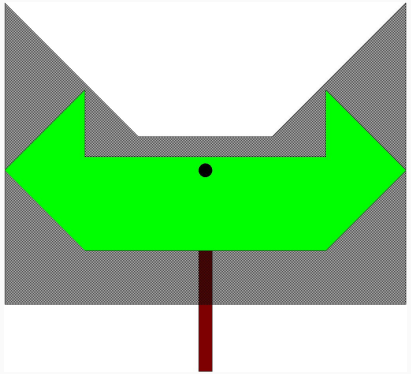
Grid Search

Techniques

- Hashing
- FFT

Algorithm

- String-matching algorithms based on hashing (Rabin-Karp) and FFT translate well over to multiple dimensions
  - For example, using a polynomial hash we can use 2 passes:
    - For each row, save the running hash of the last N items
    - Then for each column, make a running hash of the last M hashes (using P^N or some other prime for rows)
  - Knowing the size of the word in advance helps with this, eg. building prefix sum tables for each row.
- Standard string-matching with Aho-Corasick is adaptable enough to work too.
  - First, for each row find which string(s) match at each column
  - Then, look in each column for the right pattern of match IDs
  - Both steps are one-dimensional string matching problems.



Hedge Topiary

0 correct • solved at: ??:?? by ?? ??

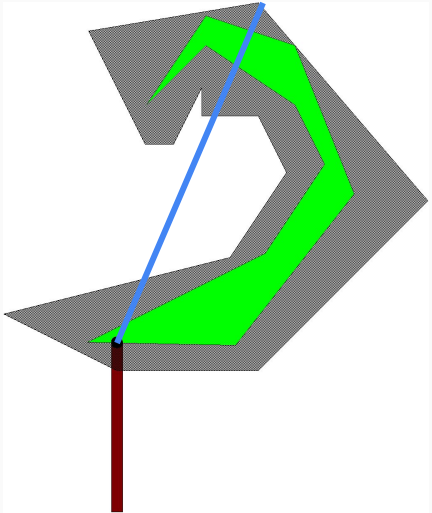
Overview

- You have two shapes, A and B.
  - The shapes may be convex.
  - However, the shapes are simple polygons with integer-aligned vertices.
- Shape A needs to fit inside B.
  - How big can you scale A (around the origin) so that it fits inside B?

Hedge Topiary

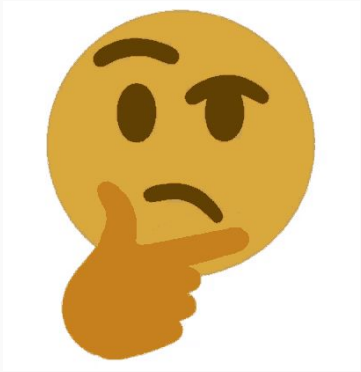
Techniques

- Ray-ray intersection
- Sweepline
- Rational arithmetic



Algorithm

- Imagine slowly scaling shape A up. At times, the boundaries of A cross inside/outside the boundaries of B (and vice versa)
  - Such intersections happen at **vertex/edge boundaries** (a vertex of A/B touches an edge of B/A).
- We're going to solve the problem individually for a "radial slice" in the direction of each vertex of A+B and combine solutions later.
  - For each vertex in the input, find the one-dimensional "slices" of A and "slices" of B in the direction of that vertex.
  - We need to align these slices so that they overlap.
    - For each slice attached to a vertex, find the set of feasible ranges of scale factors where it can "fit" into/around the slices of the other shape
- Finally, use a sweepline to intersect the sets of feasible regions.



Inconsistent

0 correct • not solved

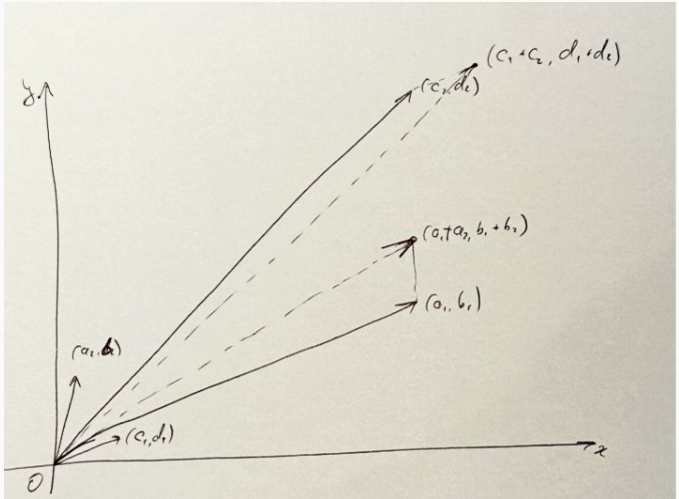
Overview

- We want to demonstrate Simpson's Paradox.
- Two teams have been training equally hard across N categories of problem
  - Team1 solved **a[i]/b[i]** of category i
  - Team2 solved **c[i]/d[i]** of category i
- Both teams attempted M problems in total.
- Generate a table of results where team1 has done better than team2 on each individual category, but team2 has solved more tasks overall.

Inconsistent Patterns

Techniques

- Construction



Algorithm

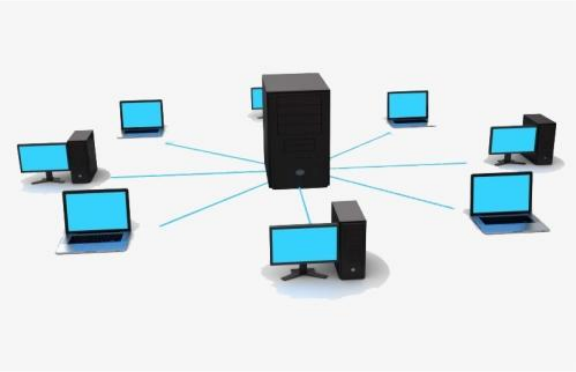
- Given the M >= 4N one construction that works is as follows:
  - - (M-3*(N-1)-1) / (M-3*(N-1)) < 1/1
  - - 1/3 < 1/2: repeated (N-2) times
  - - 1/3 < x / (M-1-2*(N-2))
    - where x is the smallest integer such that this inequality holds, ie floor(M-1-2*(N-2))

The intuition behind this construction is looking at (a_i, b_i) and (c_i, d_i) as two-dimensional vectors.

We need pairs of vectors such that:

- the slope of vector1 is greater than the slope of vector2
- while this order reverses when we add up all first elements is pairs and all second

(sum(a_i) / sum(b_i) correspond to the vector sum of (a_i, b_i)).



Jabber Network

0 correct • solved at: 02:53 by
??
??

Overview

- You start with a connected, unrooted binary tree of computers (nodes) and connections (edges)
- Distances between nodes are defined using “demands”
- Periodically an edge will be removed from the tree, and you need to re-connect the graph in the cheapest way possible according to demands x distance.

Jabber Network

Techniques

- DFS on trees
- Precomputation

Algorithm

- Once an old link is removed, the graph is split into two connected components, and you need to reconnect them again.
- You can search for the endpoints of a new edge independently in each of these 2 components
- …

Jabber Network

Techniques

- DFS on trees
- Precomputation

Algorithm

- Our way of searching for an endpoint in a component:
  - Identify all demands (s_i, t_i, d_i) that leave that component
    i. (s_i being a vertex in the current component)
  - The contribution of the endpoint p is the sum of products of demand d_i times the distance between s_i to p
  - This contribution can be recomputed efficiently when you move p to one of its neighbours
  - to do that, precompute sums of demand products in each subtree using a DFS
  - This makes it possible to find the best endpoint using another DFS
- Complexity: O(n(n + d)): n iterations, each iteration takes O(n + d)

Knitting

0 correct • solved at: ??:?? by ?? ??

Overview

- Complete the design of a scarf made out of patchwork stripes
  - Within a range of K stripes, the same colour shouldn't occur more than once
  - Otherwise, make sure that no colour appears too often (minimise the frequency of the most-frequent colour)

Knitting

Techniques

- Greedy algorithms
- Priority queues

Algorithm

- **If** a scarf can be constructed, we can build it with a greedy algorithm
  - Go from left-to-right filling in spaces.
  - Scan the last K-1 items of the existing scarf, looking for colours that we can't use for now.
    - Put the remaining colours in a priority queue where the **least-used** colour is on top (it's always best to use this if available)
  - Every time we add a colour from the priority queue, remove it from the queue and add scarf[i-k+1] to the queue instead.
- Make sure to check the resulting scarf is valid! (And that the priority queue never ran out of items)

Leg Day

0 correct • solved at: ??:?? by
??
??

Overview

- Given a pattern of up to 31 strings, represent those strings using emojis and repeat those emojis up to 31 times to make a nice calendar
- The emojis you pick should match the strings, eg. “todayislegday” should be represented by a “🦵”

Leg Day

Techniques

- Unicode
- Grit

Algorithm

- Make sure to print the week numbers at the start of each line!
- For each string, use your standard library's string-in-string functions to check which rule it matches first (**in the order specified in the input**)
- C++ and Python have no problem outputting UTF-8 characters in a non-UTF8 context. When printing in Java, make sure to set the encoding on Strings properly.

Final Standings

http://ukiepc2024.cloudcontest.org/