UKIEPC 2015

Post-Contest Presentation

rgl@google.com

Some words

First, apologies for the judge lag in the first two hours.

UKIEPC has previously been hosted alongside the NCPC, a larger contest. Most of the people working on this event had no experience with hosting any kind of programming contest until a few months ago.

This was a painful lesson, but somewhat necessary to go through to ensure it doesn't happen again in the bigger contests we plan to hold in future.

Thanks to **Rob Perkins** and **Jaap Eldering** for rescuing the servers.

Some numbers

2012: **0** teams
2013: **52** teams; **5** sites
2014: **61** teams; **9** sites
2015: **142** teams; **12** sites

First correct submission: **00:18:32 – C, DoCThors** (Imperial College London)
Last correct submission: **05:14:56 – G, Ariel** (Trinity College Dublin)
Number of submissions: **959**

742 lines of code to solve the whole set.

Some names

Organisers: **Max Wilson**, **James Davenport**, **Christian Ledig**

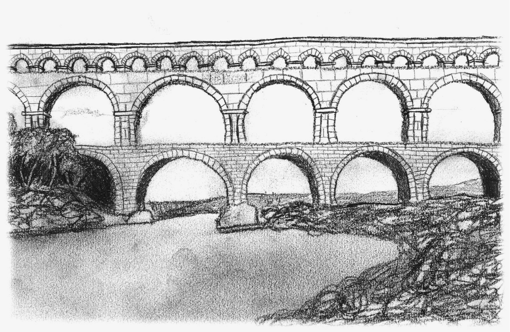
Writers: **Sander Alewijnse**, **Jaap Eldering**, **Swen Gaudl**, **Jim Grimmett**

Reviewers: **Rowan Lee**, **Nicolas Prevot**, **James Stanley**

SysAdmins: **Jaap Eldering**, **Rob Perkins**

Illustrator: **Lisa Abose**

Problem Solutions



A - Aqueducts

2 correct • solved at: **04:26** by
EE Dragons (University of Cambridge)

Author: **Jim**

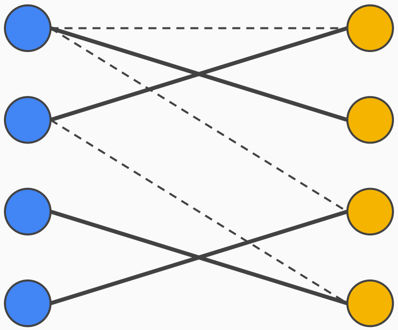
Overview

- Given a graph which is:
  - weighted (by distances)
  - directed (downhill)
  - acyclic
- And has:
  - up to 40 source points, **S**
  - up to 40 sink points, **T**
- Find a way to pair up elements from **S** and **T** so that:
  - every item from **T** has an item from **S**.
  - there is a downhill route between each **S** and **T** pair.
- Minimise the cost of this matching.

Aqueducts - Solution

Techniques

- Dijkstra's algorithm
- Breadth-first search
- Minimum cost flow



Algorithm

- We are only interested in hills from S and T.
- Make a new graph of vertices from {S,T} where edge cost is their distance in the original graph (according to Dijkstra's algorithm)
- This will be a bipartite graph.
- Look for a minimum-cost matching.
    - Hungarian algorithm (classical weighted matching method)
        - $O(S^3)$ on 40 vertices is very fast.
        - Overall complexity will be $O(S^3+S \cdot N^2 \cdot \log N)$
    - Minimum cost maximum flow
        - Can work directly on the original graph, as long as it's well-optimised.
        - $O(S \cdot N^2 \cdot \log N)$

B - Biking

75 correct • solved at: **00:28** by

Boole's Fools (University of Cambridge)

Author: **Robin**

Overview

- We have a series of up to 4 sections of a hill, with various inclines and sloped distances.
- Each section starts from where the last left off.
- Given a formula for acceleration, find the final speed of a bike if it starts at the top of any of the segments.

Mountain Biking - Solution

Techniques

- Trigonometry
- Mechanics

Algorithm

- Say we start off at speed v_0 and finish at speed v_d (after D metres).
- Integrate the formula for acceleration:
  - $v_d = v_0 + gt \times \cos(\theta)$
  - $d = v_0 t + \frac{1}{2} g t^2 \times \cos(\theta)$ *... + C*
- Solve for t:
  - $\frac{1}{2} g t^2 \times \cos(\theta) + v_0 t - d = 0$
  - $t = (-v_0 \pm \sqrt{(v_0^2 + 2gd \times \cos(\theta))}) / (g \times \cos(\theta))$
  - Substitute back in, iterate over line segments
- Or:
  - **Potential energy** $E_p = mgh$
  - **Kinetic energy** $E_k = \frac{1}{2} m v^2$
    - $v_\infty = \mathrm{sqrt}(2 \times g \times h)$

C - Conversation

58 correct • solved at: **00:18** by
DoCThors (Imperial College London)

Author: **Jim**

Overview

- Given a set of specifications like:
  - key1 value$_1$ value$_2$ value$_3$
  - key2 value$_4$ value$_5$ value$_6$
- Find the values that belong to every single key.
- Among these values, sort them:
  - By frequency descending.
  - Break ties lexicographically.

Conversation Log - Solution

Techniques

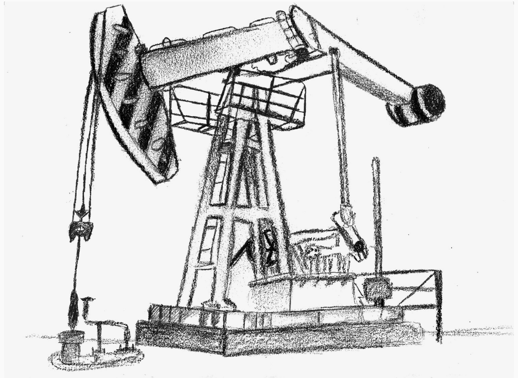
- String chopping
- Hash maps
- Sort by key
- Schwartzian transform

Algorithm

- We need two pieces of information about each word:
    - Which users it was associated with (for filtering)
    - How many times it appeared (for sorting)
- Map each username to an integer
    - Every time we encounter a new word, initialise a structure:

```
struct Word {
    string text;
    int freq = 0;
    set<UserId> users;
    bool operator < (Word const &other) const {
        return freq != other.freq ? freq > other.freq :
                                    text < other.text;
    }
}
```

- Update each word on a line by adding the userId to its set
- Filter for users.count() == MAX_USER_ID, sort, and print!



D - Drilling

3 correct • solved at: **02:41** by
DoCThors (Imperial College London)

Author: **Robin**

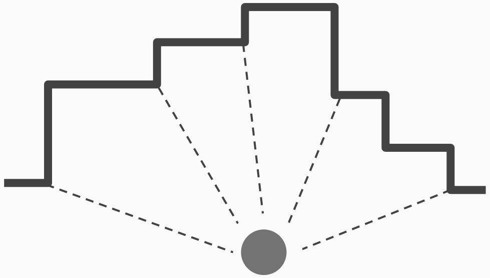
Overview

- Given a 3D surface
  - As a set of polygon-shaped contour lines
- Compute the point p
  - On the surface
  - With shortest distance to origin

Slant Drilling - Solution

Techniques

- Geometry
- Point-in-polygon



Algorithm

- It's always best to drill either straight down, or from the lower edge of a contour.
- Find the closest point to the origin on each contour segment, and calculate sloped distance.
  - Iterate over every segment
  - Create a cost function $C = (ax+(bx-ax) \times i)^2 + (ay+(by-ay) \times i)^2$
    - Differentiate and solve for $d(C)/d(i) = 0$
- Find which contour contains the origin
  - Cast a ray in some arbitrary direction. If and only if the origin is inside a contour the ray will cross that contour an odd number of times.
  - Take the candidate contour with the smallest area.

E - Rainfall

1 correct • solved at: **04:40** by
EE Dragons (University of Cambridge)

Author: **Jaap**

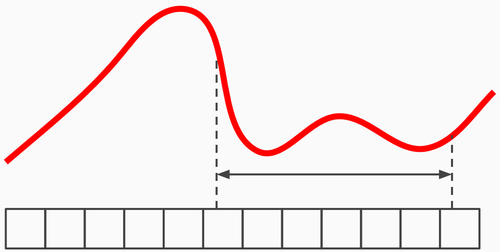
Overview

- Balance two cost functions for the same situation:
  - The rate of sweating, proportional to speed^2
  - The amount of rain across the journey, a function of start time and speed, decreasing as speed increases
- Choose a start time and speed to minimise the total cost.

Rainfall - Solution

Techniques

- Calculus
- Cumulative sums
- Ternary search
- Dynamic programming



Algorithm

- Algebra shows it's always best to keep a constant speed
- Try every possible starting and ending minute
  - $O(N) \times O(N) = O(N^2)$
- If we are going to chop off some fractional time ***x*** to decrease rain (at the expense of cycling faster), we'll chop off part of the rainier of the start and end minutes.
  - Reducing time by more than 1 falls within another [start, end] pair so we can ignore that case.
- cost = sum(rain[S:T]) - x × rain[edge] + constant/(T-S-x)
- **Differentiate** cost and solve for dCost/dX = 0
  - $0 \le x \le 1$
  - Don't forget x = 0 and x=1

F - Physiognomy

0 correct

Author: **Robin**

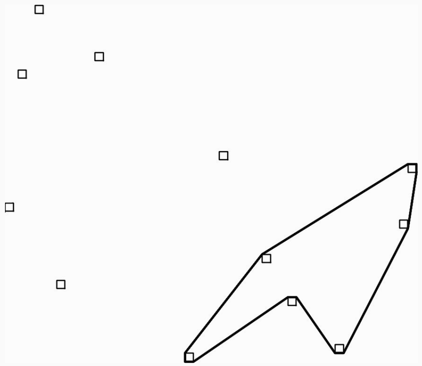
Overview

- Given up to 12 weighted squares of equal size,
- Make a loop around some subset of them such that:
  - The loop is continuous.
  - The sum of weights inside the loop is equal to the sum of weights outside.
- Make this loop as short as possible.

Physiognomy - Solution

Techniques

- Topology
- Point-in-polygon
- Bitmasks
- Travelling salesman
- Dynamic programming



Algorithm

- Assume we already constructed the loop
- In this case, we can find whether a lamp is inside by the same means as ***drilling***: if and only if it's inside, the number of loop segments crossed in any direction will be odd.
- Let's make that part of our state:
    - minimum_loop[start][pos][n_1,n_2,...,n_7,n_8]... n^n possibilities?
    - We only need to know the parity, not the exact number.
    - **minimum_loop[start][pos][2^n]**
- How many possibilities for **start** and **pos**?
    - A minimal loop always touches only **corner vertices**, of which there are at most **n × 4** = 48.
- Time complexity: **$O(n^3 \times 2^n)$**

G - Drinking

18 correct • solved at: **01:53** by
Y U NO ACK (Imperial College London)

Author: **Jim**

Overview

- We have a collection of beers
  - Various costs
  - Various alcohol contents
  - Various sizes of glass
- We have targets:
  - Spend a certain amount of money
  - Drink a certain amount of alcohol
- We need to find a way of meeting these targets exactly by choosing a list of orders
  - Some can be chosen several times
  - Some can be ignored

Drink Responsibly - Solution

Techniques

- Fixed-point arithmetic
- Knapsack problem
- Depth-first search
- Memoisation

Algorithm

- Imagine a straightforward depth-first search:
  - ```
    def solve(i, units_left, money_left):
        if units_left <= 0 or money_left <= 0 or i >= n:
            return [] if (units_left | money_left) == 0 else None
        sol_with = solve(i, units_left-units[i], money_left-price[i])
        sol_without = solve(i+1, units_left, money_left)
        if sol_with is not None:
            return [beer] + sol_with
        elsif sol_without is not None:
            return sol_without
        else:
            return None
    ```
- Q: How many possible sets of parameters can this take?
  - A: O(N) × O(U) × O(M) = **O(NUM)**
- Memoise answers to overlapping subproblems:
  - ```
    if already_solved[i][units_left][money_left]:
        return answer_for[i][units_left][money_left]
    ```

H - Sunlight

9 correct • solved at: **01:48** by
Beuler (University of Cambridge)

Author: **Robin**

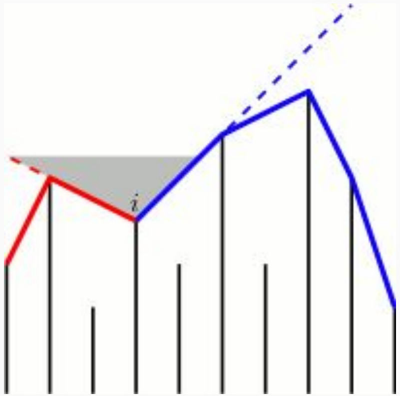
Overview

- Given N columns in 2D
- Find the proportion of angles above each column which aren't occluded by other columns
- For example:
  - 2 1 3
  - y[0], 2× the height of y[1], occludes 45° of the view
  - y[3], 3× the height of y[1], occludes 63.4° of the view
- N is quite large, so find a method more efficient than brute-force.

Sunlight - Solution

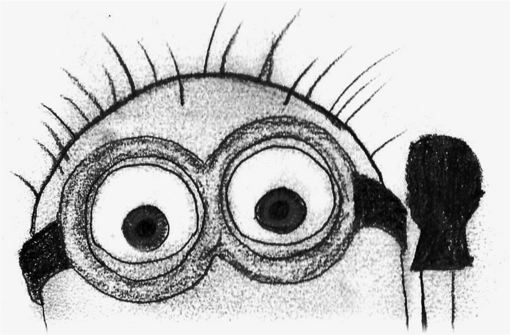
Techniques

- Convex hull
- Andrew's algorithm
- Trigonometry



Algorithm

- Some first-pass observations:
  - When computing the angle for a building **i**, we can safely ignore all buildings not in the convex hull of buildings **[0, i]**, and not in the convex hull of buildings **[i, n-1]**.
  - In fact, the building defining the angle on right side of **i** comes directly after **i** in the convex hull of **[i, n-1]**.
  - Similarly, the building defining the angle on the left side of **i** comes directly before **i** in the convex hull of **[0, i]**.
- Compute convex hull twice using Andrew's algorithm
  - Maintain convex hulls on stack for [0, i] and for [i, n-1]
  - Left to right: Top of stack defines the left angle
  - Right to left: Top of stack defines the right angle



I - Nimionese

69 correct • solved at: **00:33** by

Exception: teamName not found.
(University of Warwick)

Author: **Max**

Overview

- We have a string made of words, each composed of several syllables.
- Three rules to apply:
    - First letter must be "hard" ie. member of a certain subset of consonants.
    - For the subsequent syllables all "hard" consonants must match the first letter
    - Each word must end in "ah", "oh", or "uh".

Nimionese - Solution

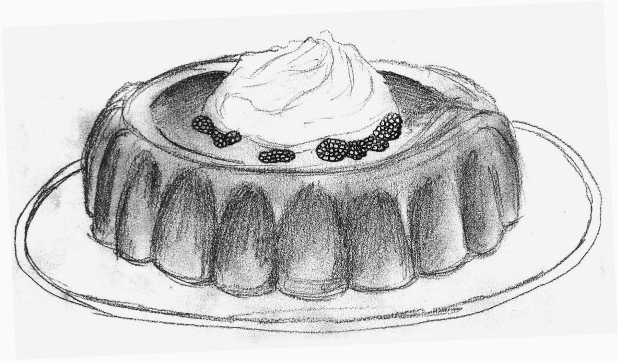
Techniques

- Regular expressions
- String chopping

Algorithm

- Read in each word separately
- Use your language’s split() function to get an array of syllables
- Three regexes:
  - ```
    let hard = “bcdgknpt”
    let soft = “aou”

    syll[    0].sub(“^[^$hard]”, [x] ⟶ closest(x, hard))
    syll[1...$].sub(“[$hard]”,   [x] ⟶ syll[0][0])
    syll[  $-1].sub(“[$hard]\$”, [x] ⟶ x + closest(x, soft) + ‘h’)
    ```
- For the security-minded:
  - `[...] .sub(“^[^$hard]”,,) [...]`
  - **Please don’t do this** in real life!
    ```



J - Jelly Raid

5 correct • solved at: **03:05** by
EE Dragons (University of Cambridge)

Author: **Swen**

Overview

- Given a 60x60 floor plan with walkable and blocked cells,
- Locations of dormitory and kitchen,
- And paths of 200 patrolling masters
  - (where every path contains at most 7 cells and masters follow it back and forth indefinitely)
- Find the shortest path from the dormitory to the kitchen so that you are not seen by the patrolling masters
  - (where two people can see one another if they are in the same row or column and there are no blocked cells between them).

Jelly Raid - Solution

Techniques

- Least common multiple
- Breadth first search
- Sneakiness

Algorithm

- Just running Dijkstra won't work.
    - What if we get trapped somewhere?
- Plugging time in as part of state makes for a slow solution
    - $60 \times 60 = 3600$
    - $O(N^4)$
- The patrol periods for {1, 2, 3, 4, 5, 6, 7} will be {1, 2, 4, 6, 8, 10, 12}
    - LCM = 120
- Let's incorporate the current progress through the cycle into state
- min_dist[120][rows][cols]
    - $O(C \times N^2)$

K - Call a Cab

0 correct

Author: **Sander**

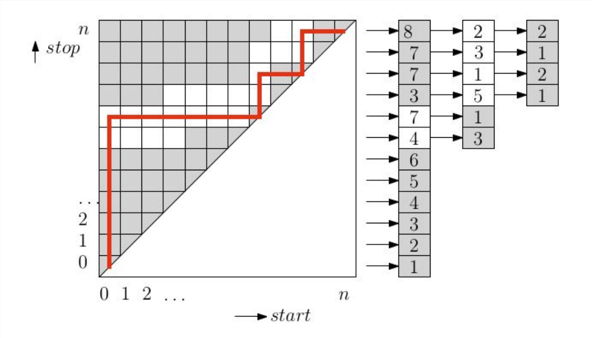
Overview

- Given:
  - N Points Of Interest
  - Restrictions on whether we can travel from POI **i** to **j** by taxi **t**
    - Minimal distance
    - Maximal variation in angle
- Compute how to get from 0 to n in minimal number of hops using any kind of taxi.

Call a Cab - Solution

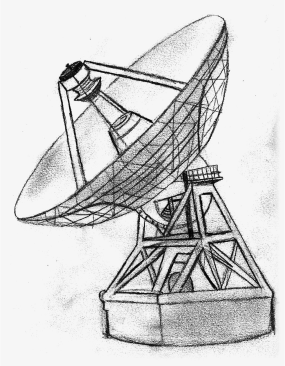
Techniques

- Two pointer algorithms
- Multisets
- Segment trees
- Dynamic programming



Algorithm

- For a given transportation type:
    - Farthest reachable POI is **non-decreasing**.
    - Nearest reachable POI is **also non-decreasing**.
    - Everything in between is accessible.
- Execute two pointer algorithm for nearest and farthest. Moving a pointer takes O(1) and O(log(n)) time resp.
- **Minimum distance**: keep a running total of distance travelled.
- **Maximum range**: keep an auxiliary multiset of differences between all angles travelled, in sorted order. The heading range is given by 360.000 minus the largest angular difference.
- Now we can run an efficient dynamic programming algorithm with a segment tree (or sliding window)



L - Telescope

2 correct • solved at: **03:41** by
Me[N]tallica (University of Cambridge)

Author: **Robin**

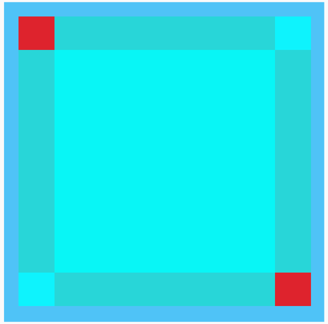
Overview

- A grid of boolean values has had a mean filter applied.
- This means that where previously a value was cell[i][j]. it will become:
  - sum(cell[a][b]) ÷ n^2 for a,b where `max(abs(a-i), abs(b-j)) < ½ n`
- We need to reverse this filter to count the number of connected components.

Telescope - Solution

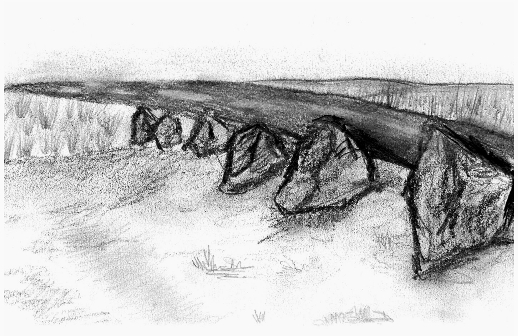
Techniques

- Cumulative sums
- Inclusion-exclusion
- Flood fill



Algorithm

- Multiply any blurred pixel by N^2 to get the number of white pixels in an NxN square around that pixel
  - 0xFFFF / (100^2) ≈ 6.5 so no precision has been lost
- What if we subtract two squares?
  - ```
    pixels(a..b,c..d) - pixels(a-1..b-1,c..d)
     = pixels(b,c..d) - pixels(a-1,c..d)
    ```
- How about four squares?
  - ```
    pixels(a..b,c..d) - pixels(a-1..b-1,c..d)
                      - pixels(a..b,c-1..b-1)
                      + pixels(a-1..b-1,c-1..d-1)
     = pixel(b,d) - pixel(a-1,d) - pixel(b,c-1) + pixel(a-1,c-1)
    ```
- We know that all pixels outside the boundaries are completely black, so let's work in from the edges restoring cells.
- Complexity: **O(RC)**



M - Milestones

30 correct • solved at: **00:57** by
DoCThors (Imperial College London)

Author: **Robin**

Overview

- Given
  - 1 list **A** of observations of an event at one time scale factor
  - 1 list **B** of when all events happened at another time scale factor
- Find all of the scale factors that could plausibly be applied to **B** to get a substring that equals **A**.
- Example:
  - 1,2,3
  - 3,4,5,7,9
    - 3,4,5 = 1,2,3 × 1 + 2
    - 5,7,9 = 1,2,3 × 2 + 1

Milestone Counter - Solution

Techniques

- String matching
- Fractions

Algorithm

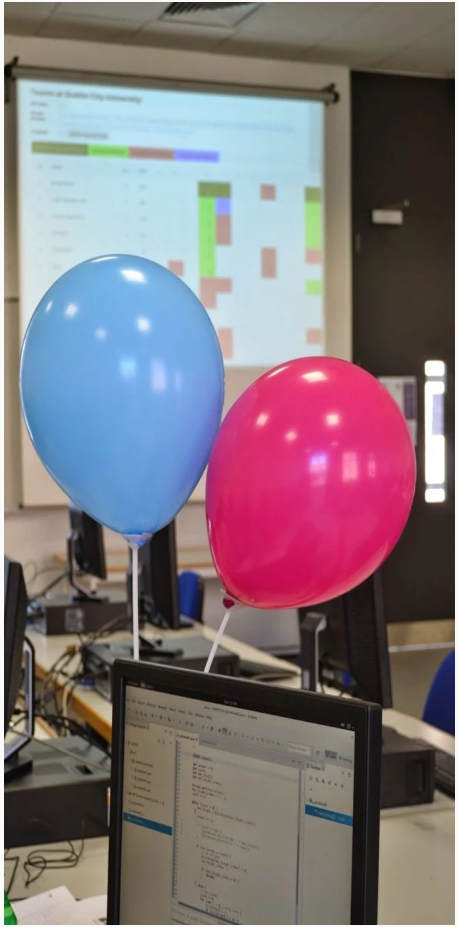
- Let's look at a base case: checking N times against N distances.
  - We can work out the speed from $(d_1 - d_0) \div (t_1 - t_0)$
  - Now we need to compare the speed for every pair:

    $$(d_1 - d_0) \div (t_1 - t_0) = (d_{x+1} - d_x) \div (t_{x+1} - t_x)$$

    or

    $$(t_{x+1} - t_x) \div (t_x - t_{x-1}) = (d_{x+1} - d_x) \div (d_x - d_{x-1})$$
  - What's important is the **ratio** between current distance and previous distance.
- The strings of M and N symbols are equivalent to strings of M-2 and N-2 fractions which should have exact matches.
- From here it's regular string comparison
  - Knuth Morris Pratt / Boyer Moore / Rabin Karp
  - Or since N is so small, brute force works too.






Questions?

Or comments?

Final Standings

http://ukiepc-2015.bath.ac.uk/