Problem Solutions

Contour

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.

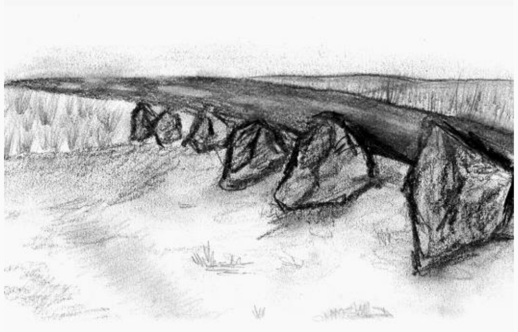
Contour- Solution

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:
  - $v_d = v_0 + gt \times \cos(\theta)$
  - $d = v_0t + ½gt^2 \times \cos(\theta)$ ... *+ C*
- Solve for t:
  - $½gt^2 \times \cos(\theta) + v_0t - d = 0$
  - $t = (-v_0 \pm \sqrt{}(v_0^2 + 2gd \times \cos(\theta))) / (g \times \cos(\theta))$
  - Substitute back in, iterate over line segments
- Or:
  - **Potential energy** $E_p = mgh$
  - **Kinetic energy** $E_k = ½mv^2$
    - $v_\infty = \mathrm{sqrt}(2 \times g \times h)$



First Counter

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

First 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.

Hungover

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

Hungover - Solution

Techniques

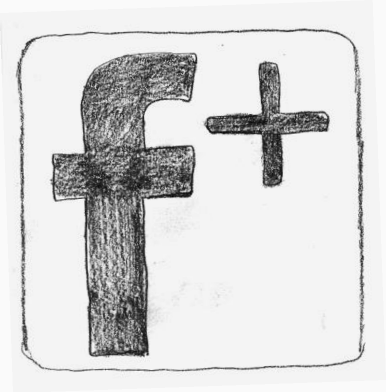
- Fixed-point arithmetic
- Knapsack problem
- Depth-first search
- Memoisation

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]
    ```



KeyWord Log

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.

KeyWord 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!