Appalling Architecture

- You were asked to check whether the center of mass lies to the left of the leftmost point on the ground, to the right of the rightmost point, or in between.
- To calculate x-coordinate of the center of mass, calculate

$$\sum_{(x,y) \text{ is the middle of a box}} x$$

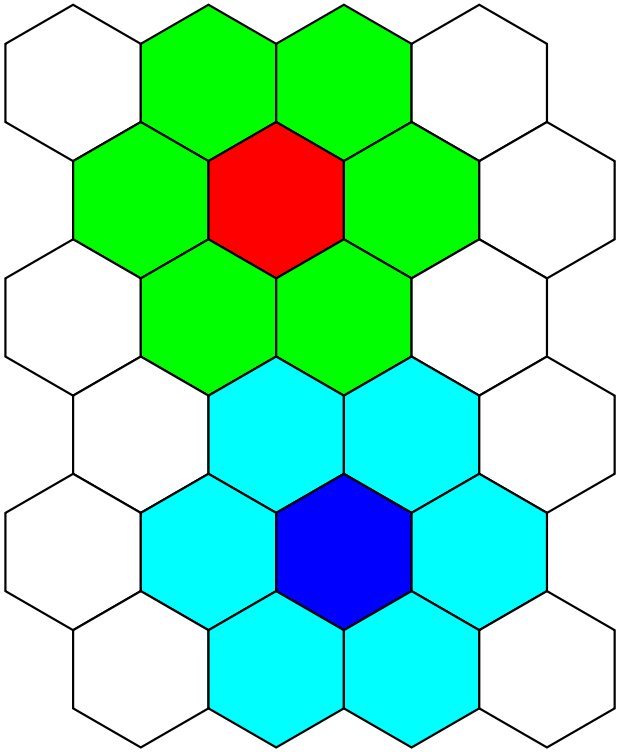
  and divide by the number of boxes the construction is made of.
- Careful: the middle of a box is a half-integer. (Or, the sides of the boxes are half integers.)
- Runtime: $O(\text{area of grid})$, but much slower solutions were accepted.

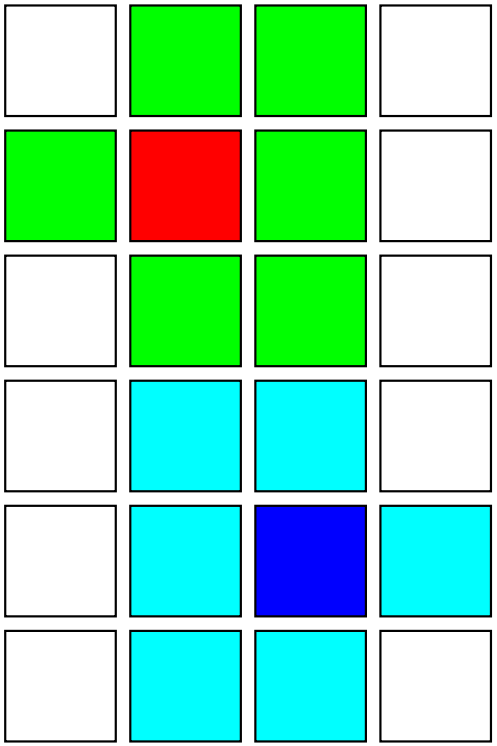
Bee Problem

- Calculate the size of connected components in the grid, then use the largest remaining one until you are out of honey.
- Use a standard flood fill algorithm (DFS or BFS) to calculate the sizes of the components.
- But: the grid is not a square grid, like in your computer, but a hex grid!

Bee Problem

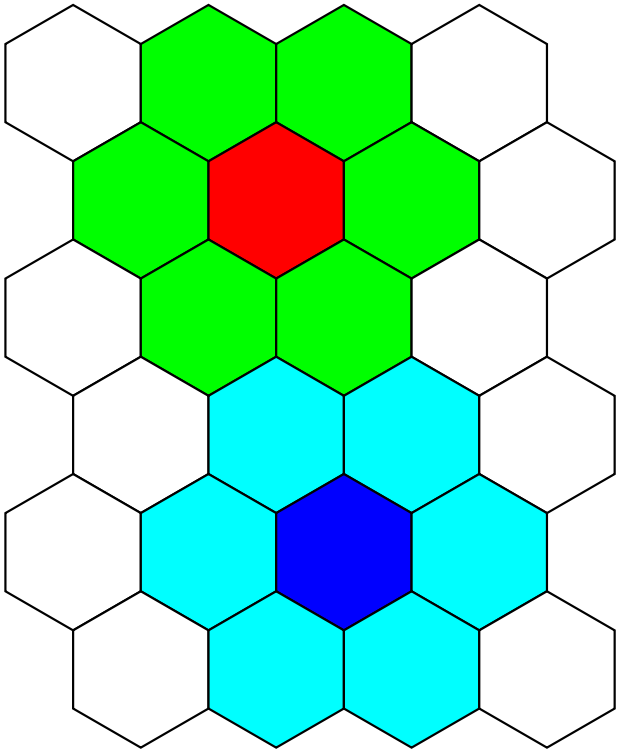
One way of solving this: put it in a rectangular grid and compute neighbours differently based on whether the row is odd or even.

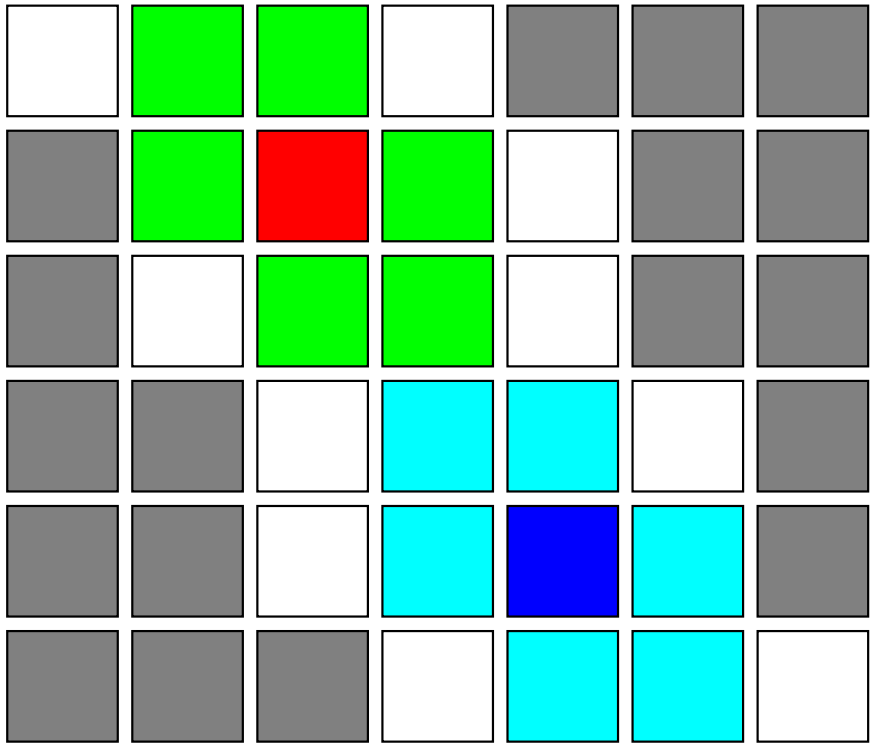




Bee Problem

A second way of solving this: put the grid into a larger rectangular array, but offset the rows. Then the neighbors are always the same.





Criss-Cross Cables

- Goal: find out whether, given an increasing list $x_1, \ldots, x_n$ and a list $l_1, \ldots, l_m$, for every $1 \leq i \leq m$ there is a pair $p_i = (x_{i_1}, x_{i_2})$ $(i_1 < i_2)$ such that $x_{i_2} - x_{i_1} < l_i$ and $p_i \neq p_j$ for all $1 \leq i < j \leq m$.
- First idea: calculate all distances, sort them by length, and check.
- Problem: this is $O(n^2 \log n)$.

Criss-Cross Cables

- Problem: this is $O(n^2 \log n)$.
- Solve this by only calculating the distances you need: make a priority queue of distances, and add all distances $x_{i+1} - x_i$. Then, repeatedly do the following:
  - See if you can make the smallest wire fit on the smallest current distance. If not, the answer is "no".
  - If it does fit, remove the smallest wire and the smallest current distance. Suppose this distance was between x_i and x_j: add distances $x_{j+1} - x_i$ and $x_j - x_{i+1}$ to the queue.

  If you get rid of all wires this way, then the answer is "yes".
- Watch out for $i = 0$ or $j = n$!
- Other nice solutions were possible (or not so nice, if you coded efficiently); see the judge directory.
- Runtime: $O(n \log n)$.

Daily Division (1)

- Question: Given an array $a_0, \ldots, a_{n-1}$, find the index i which minimizes $|(a_0 + \cdots + a_{i-1}) - (a_{i+1} + \cdots + a_{n-1})| - \delta$ where $\delta = 1$ if a_i is odd and the left-half and right-half are not equal, and else $\delta = 0$.

Daily Division (1)

- Question: Given an array $a_0, \ldots, a_{n-1}$, find the index i which minimizes $|(a_0 + \cdots + a_{i-1}) - (a_{i+1} + \cdots + a_{n-1})| - \delta$ where $\delta = 1$ if a_i is odd and the left-half and right-half are not equal, and else $\delta = 0$.
- First, consider no δ. Let $F(i) = a_0 + \cdots + a_{i-1} - (a_{i+1} + \cdots + a_{n-1})$. Then $F(0) < 0, F(n-1) > 0$. Observe that $F(i+1) - F(i) = a_i + a_{i+1} \geq 2$, so F is an strictly increasing function.

Daily Division (1)

- Question: Given an array $a_0, \ldots, a_{n-1}$, find the index i which minimizes $|(a_0 + \cdots + a_{i-1}) - (a_{i+1} + \cdots + a_{n-1})| - \delta$ where $\delta = 1$ if a_i is odd and the left-half and right-half are not equal, and else $\delta = 0$.
- First, consider no δ. Let $F(i) = a_0 + \cdots + a_{i-1} - (a_{i+1} + \cdots + a_{n-1})$. Then $F(0) < 0, F(n-1) > 0$. Observe that $F(i+1) - F(i) = a_i + a_{i+1} \geq 2$, so F is an strictly increasing function.
- With a binary search, determine the smallest index i such that $F(i) >= 0$. Then the index minimizing $|F(j)|$ is i or $i-1$.

Daily Division (1)

- Question: Given an array $a_0, \ldots, a_{n-1}$, find the index i which minimizes $|(a_0 + \cdots + a_{i-1}) - (a_{i+1} + \cdots + a_{n-1})| - \delta$ where $\delta = 1$ if a_i is odd and the left-half and right-half are not equal, and else $\delta = 0$.
- First, consider no δ. Let $F(i) = a_0 + \cdots + a_{i-1} - (a_{i+1} + \cdots + a_{n-1})$. Then $F(0) < 0, F(n-1) > 0$. Observe that $F(i+1) - F(i) = a_i + a_{i+1} \geq 2$, so F is an strictly increasing function.
- With a binary search, determine the smallest index i such that $F(i) >= 0$. Then the index minimizing $|F(j)|$ is i or $i-1$.
- Answering queries takes $O(\log n)$ calculations of F.

Daily Division (2)

- Note, $F(i) = 2\sum_{j=0}^{i-1} a_j + a_i - \sum_{j=0}^{n-1} a_j$. The last two terms can be done in $O(1)$, by updating the sum after a query.

Daily Division (2)

- Note, $F(i) = 2\sum_{j=0}^{i-1} a_j + a_i - \sum_{j=0}^{n-1} a_j$. The last two terms can be done in $O(1)$, by updating the sum after a query.
- For the first term, we have multiple data structures possible which allow fast update and querying:
  - Fenwick tree - $O(\log n)$
  - Segment tree - $O(\log n)$
  - Square-root decomposition - $O(\sqrt{n})$

Daily Division (2)

- Note, $F(i) = 2\sum_{j=0}^{i-1} a_j + a_i - \sum_{j=0}^{n-1} a_j$. The last two terms can be done in $O(1)$, by updating the sum after a query.
- For the first term, we have multiple data structures possible which allow fast update and querying:
  - Fenwick tree - $O(\log n)$
  - Segment tree - $O(\log n)$
  - Square-root decomposition - $O(\sqrt{n})$
- Now consider the δ again. Call the target function $F'(j) = |F(j)| - \delta$. Since $F(j+1) - F(j) \geq 2$. From our binary search, with i smallest such that $F(i) \geq 0$, now either $i-2$, $i-1$ or i is minimal (proof!).

Daily Division (2)

- Note, $F(i) = 2\sum_{j=0}^{i-1} a_j + a_i - \sum_{j=0}^{n-1} a_j$. The last two terms can be done in $O(1)$, by updating the sum after a query.
- For the first term, we have multiple data structures possible which allow fast update and querying:
  - Fenwick tree - $O(\log n)$
  - Segment tree - $O(\log n)$
  - Square-root decomposition - $O(\sqrt{n})$
- Now consider the δ again. Call the target function $F'(j) = |F(j)| - \delta$. Since $F(j+1) - F(j) \geq 2$. From our binary search, with i smallest such that $F(i) \geq 0$, now either $i-2$, $i-1$ or i is minimal (proof!).
- Total runtime: $O(n \log n)$ or $O(n\sqrt{n})$.

Eating Everything Efficiently

- Question: given a DAG with points at every node (but with diminishing returns), what is the best score that can be obtained by walking along the edges?
- Greedy approach is infeasible: sometimes a very lucrative stall may be thousands of nodes away.
- Brute force does not work: way to many paths to check.
- Dynamic programming: if we started in the leaves the answer is simple: just eat the pizza. How do we go from there?

Eating Everything Efficiently

- Dynamic programming: if we started in the leaves the answer is simple: just eat the pizza. How do we go from there?
- The diminishing returns are constructed in a special way: eating at a stall before entering a subgraph S will divide the score you can get in S by 2.
- This gives the following:

$$dp[\mathrm{v}] = \max_{u \text{ child of } v} \left(\max \left(\mathrm{c_v} + \frac{dp[\mathrm{u}]}{2}, dp[\mathrm{u}] \right) \right).$$

- Then the result is in $dp[0]$.
- Runtime: $O(n + m)$.

Floating points

- Given the 2D outline of a ship and some submerged ping pong balls, how many balls remain under water because they are stuck beneath the ship?
- Basic solution: just let the balls float up, and see if they ever get positive y-coordinate. Repeat:
  - Float up. Find the lowest edge you hit.
  - Is the edge horizontal? Then you're stuck. Otherwise float along the edge until a corner.
  - In a corner: do you get stuck, do you float to the connecting edge, or do you float up freely?
  - Repeat until you float off to above the surface or you get stuck.

Floating points

- Floating up once takes $O(n)$ time. Floating up all the way to the top takes $O(n^2)$ time.
- This gives a $O(bn^2)$ solution $\rightarrow$ too slow.
- To improve the solution: Preprocess the ship. First calculate for every edge what the outcome is if the ball hits this edge, which takes $O(n^2)$ time if you use memoization. Then try all balls, which now takes $O(n)$ per ball. Runtime: $O(n^2 + bn)$.

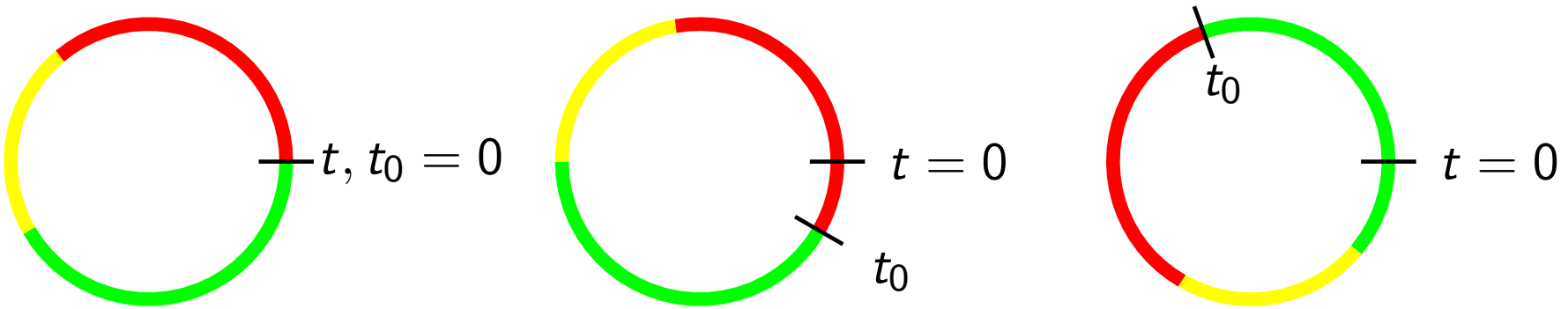
Green Light

Problem

A traffic light is green for T_g seconds, yellow for T_y and red for T_r. Given a series of observations of colors at specific times, compute the probability that the light is color c_q at some time t_q.

Solution

The traffic light goes through a cycle every $T = T_r + T_y + T_g$ seconds. So an observation of, say, green at time t implies that the light is also green at $t + T$, $t + 2T$, $t - T$, etc. So we can model 'time' not as a line, but as a circle. The question is then, how is the color cycle rotated over the circle? (i.e., at what time t_0 modulo T did the cycle start?)



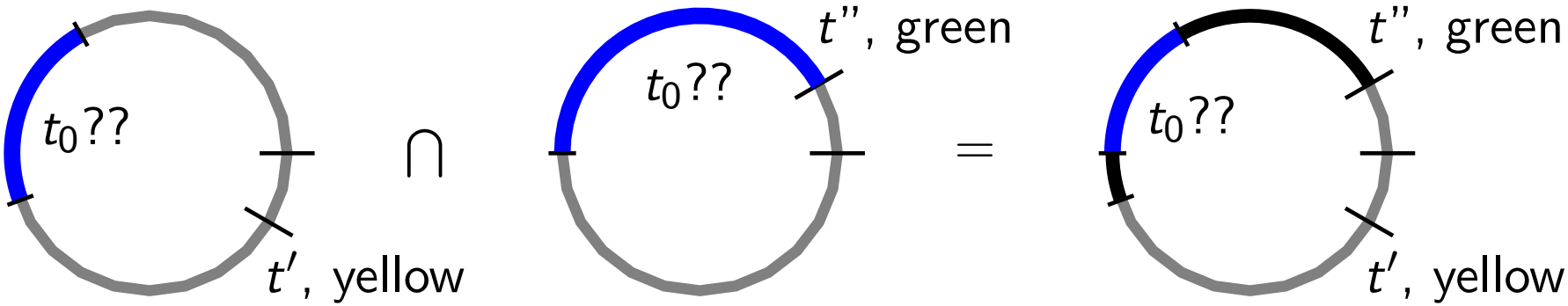
Green Light

Solution

If we observe, say, yellow at time t' (modulo T) this implies the light turns yellow at some point in $(t' - T_y, t']$, and consequently that it turns green at some point in $(t' - T_y - T_g, t' - T_y]$, i.e.:

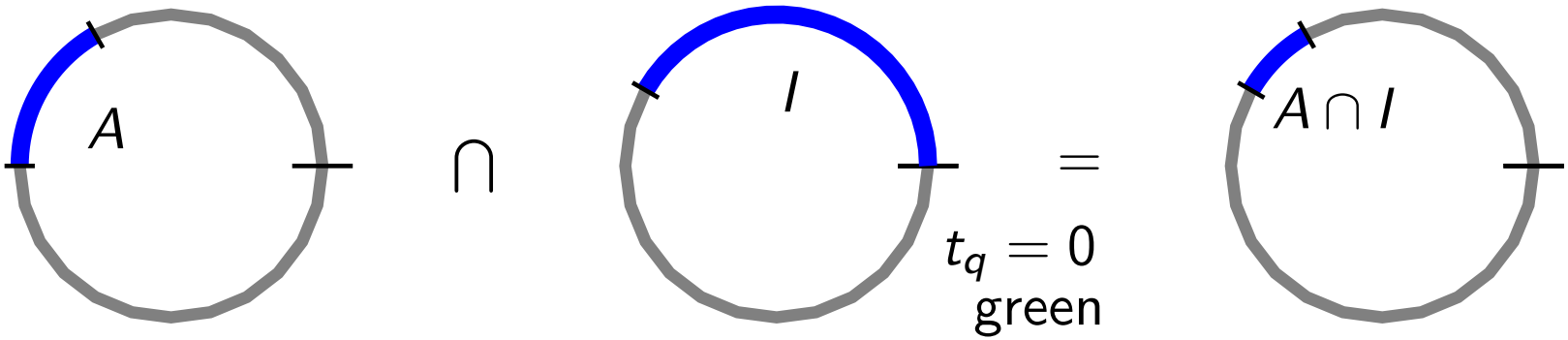
$$t' - T_y - T_g < t_0 \leq t' - T_y$$

Every observation gives rise to an interval on the circle where t_0 must be located. Intersecting all these intervals gives the only admissible region on the circle for t_0.



Solution

Call the resulting admissible circle section A. Finally, the problem asks us the probability that at time t_q the color is c_q. Treating this as an observation gives another interval I. The answer is then simply $|I \cap A|/|A|$ (where $|\cdot|$ gives the length of an interval) - that is, the proportion of A that also results in color c' at time t'.



This sample yields: $|A \cap I|/|A| = 1/2$.

In principle these intersections can be calculated in $O(n \log n)$, but with the given bounds $O(n^2)$ is fine. The tricky part is working with intervals on the circle.

H to O

- Given an input molecule (CH_3OH) and a count of that molecule (5), how many of a target molecule (CH_2) can you make?
- Solution using arrays: One counter array of 26 atoms for each molecule, filled at parsing.

Input atom:

A	B	C	...	H	...	O	...	Z
0	0	1	...	3+1	...	1	...	0

Output atom:

A	B	C	...	H	...	O	...	Z
0	0	1	...	2	...	0	...	0

- Now multiply each number in the first array by 5, and compare the two arrays.
- Don't forget to watch out for division by 0!
- A solution that works on all sample test cases usually works on all inputs.

Isomorphic Inversion

Problem

Given a string s, partition it into a maximal number of contiguous substrings $s = s_0 s_1 \dots s_{k-1}$, such that $s_i = s_{k-i-1}$ for all i.

$$013189301$$

$$01 \,|\, 3 \,|\, 189 \,|\, 3 \,|\, 01$$

Output $k = 5$

Isomorphic Inversion

Solution

Greedy solution: Find the smallest prefix of s that is also a suffix, and remove them. Repeat until $s = \emptyset$.

Proof

Proof by contradiction: suppose that at some point the optimal solution removes a longer prefix/suffix b, whereas we want to remove the shortest prefix/suffix a.

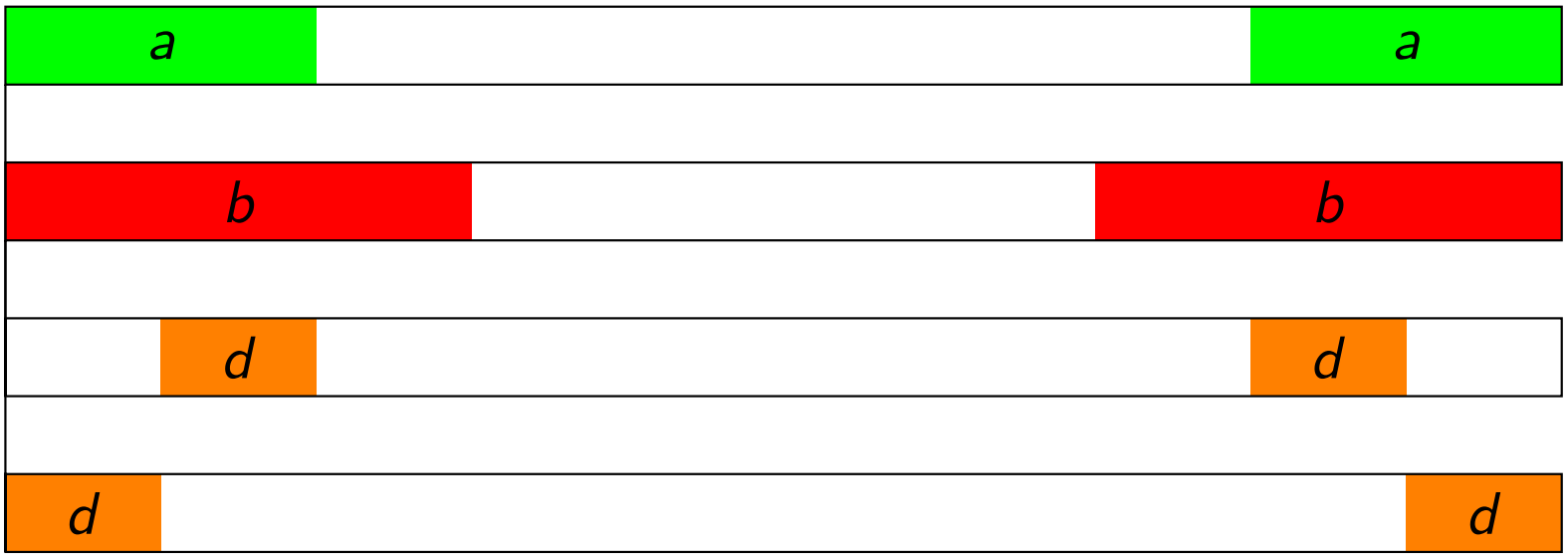
Isomorphic Inversion

Proof

We know that a must be both a prefix and a suffix of b. If $|a| \leq |b|/2$ then we can write $b = aca$ for some c, and in the optimal solution, replace b with $a \mid c \mid a$, contradicting optimality.

Proof

On the other hand, if $|a| > |b|/2$, then the prefix/suffix occurences of a in b overlap. Call this overlap d:



But then the overlap d must also be a prefix and suffix of the whole string s, contradicting the fact that a is the smallest such string.

Isomorphic Inversion

Solution

It takes $O(k)$ time to check if a prefix of length k is also a suffix, so a naive implementation of this solution will have a runtime of $O(n^2)$ and will give `TLE`.

We can speed this up by maintaining rolling hashes of the prefix and suffix as we scan in from both sides, and reset when we find a match, giving $O(n)$.

The hash of a string $t_i t_{i+1} \dots t_j$ is $H_{i,j} = \sum_{k=i}^{j} t_k p^{k-i} \mod M$ for some small prime p and large prime M. By precomputing p^k we can quickly update to the hash of

- $t_i t_{i+1} \dots t_j t_{j+1}$ $(H_{i,j+1} = H_{i,j} + t_{j+1} p^{j-i+1})$
- $t_{i-1} t_i t_{i+1} \dots t_j$ $(H_{i-1,j} = t_{i-1} + p \cdot H_{i,j})$

Jurassic Jigsaw

- Given a set of strings of DNA, what is the "most likely" evolutionary tree?
- Turn the strings into a weighted graph: the weight on an edge between string s and string t is the number of positions i such that $s_i \neq t_i$.
- Then the problem is: compute a minimum spanning tree.
- Use Kruskal or Prim or another MST algorithm. Run time: $\mathcal{O}(n^2 k \log(n))$.

Kallax Construction

- Given a couple of companies that pack inefficiently, what is the pack with the smallest advertised size that still contains at least B pegs?
- For every company, you solve a knapsack problem: having kept track of both the advertised and the real size of the packs of the previous company, use knapsack to find the best packing for a company.
- Take care: you need to take the lowest real size.
- Runtime: $O(Bkl)$.