# Solution Outlines

SWERC Judges

SWERC 2009

## Statistics

| Problem | 1st team solving | Time |
|---|---|---|
| A - Trick or Treat | UPC-2 | 11 |
| B - Restaurant | Exceptional Prog From Lausanne | 72 |
| C - Lights | | |
| D - Darts | UPC-2 | 68 |
| E - Genetics | | |
| F - Haunted Graveyard | ENS Ulm 1 | 142 |
| G - Slalom | Lusco Fusco | 107 |
| H - Routing | | |
| I - Happy Telephones | ENS Ulm 4 | 17 |
| J - Stammering Aliens | UPC-2 | 98 |

## Statistics

| Problem | AC | Total | Success Rate |
|---|---|---|---|
| A - Trick or Treat | 18 | 39 | 46% |
| B - Restaurant | 25 | 85 | 29% |
| C - Lights | 0 | 0 | |
| D - Darts | 5 | 13 | 38% |
| E - Genetics | 0 | 1 | 0% |
| F - Haunted Graveyard | 8 | 146 | 5% |
| G - Slalom | 6 | 35 | 17% |
| H - Routing | 0 | 0 | |
| I - Happy Telephones | 30 | 54 | 56% |
| J - Stammering Aliens | 4 | 35 | 11% |

# Working at the restaurant

## Solution

*Categories:* ad-hoc, data structures.

### Solution

**Simulating a queue with two stacks.**

- We drop plates onto the same stack, and take them from the other one.
- When we run out of plates in the stack from where we take plates, we move all plates from the other stack there.
- Amortized analysis: pay 2 when you drop a plate (actual cost 1), and pay 1 when you receive a take request (actual cost can be much larger, but has already been paid for).
- We output $\leq 2N$ lines and $\leq 3M$ movements; bounds in the statement are not tight.

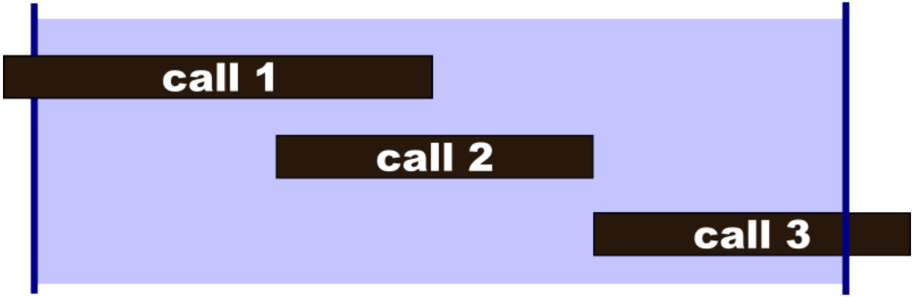
## Common mistakes

- Misunderstanding about order:
    - plates moved, dropped, and taken one by one
    - take before move
- Not meeting restrictions:
    - Do not move, drop, or take 0 plates.
    - Do not move more than $6M$ plates ($M$ = sum of plates from waiter).
    - Do not output more than $6N$ lines ($N$ = events).

# Happy Telephones I

## Solution

*Category:* straightforward.



## Solution

Because of the constraints ( $1 \leq N < 10\,000$ and $1 \leq M < 100$), the brute force approach is fine. Therefore a trivial check for all the calls for each interval works. Complexity: $O(MN)$.

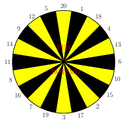
## A better solution

Sorting the vectors of ending and starting times of all telephone calls and using binay search yields an $O(\ (M+N)\ log\ N)$ solution.

# Darts

## Solution

*Category:* Dynamic Programming



- $d_i = 20, 1, 18, 4, ...$ score of each sector.
- $p_A(n, m) =$ probability of A win if it is A's turn.
  $n =$ score of A
  $m =$ score of B
- $p_B(n, m) =$ probability of B win if it is B's turn.

**Problem:** compute $p_A(N, N)$ and $p_B(N, N)$.

# Darts

## Solution

**Remember:** A plays in a unique way, but B plays maximizing his probability of winning.

### Recurrence

$$p_A(n, m) = \frac{1}{20} \sum_{i=1}^{20} (1 - p_B(n - d_i, m))$$

$$p_B(n, m) = \max_{1 \le i \le 20} \frac{1}{3} \sum_{j=-1}^{1} (1 - p_A(n, m - d_{i+j}))$$

Only valid if $n >= 20$ and $m >= 20$!!

# Darts

## Solution

### Trouble

Writing a similar recurrence for $n < 20$ and $m < 20$:
If $n < 20$, $p_A(n, m)$ depends on $p_B(n, m)$, and if $m < 20$, $p_B(n, m)$ depends on $p_A(n, m)$.

Why are we in trouble? We have two equations and two variables, $p_A(n, m)$ and $p_B(n, m)$, just solve the system!
**Not so fast! B maximizes winning probability while playing**
Where does B aim at? We have to solve the system for every possible number B aims at, and then $p_B(n, m)$ will be the maximum of all the possible outcomes.

Solve 20 of these systems and take the one maximizing $p_B(n, m)$.

# Darts

## Solution

### Solution

Precompute $p_A(n, m)$ and $p_B(n, m)$ for all $n, m <= 501$, solving system of equations if necessary.
Complexity: $O(maxN^2)$.

### Comments

Systems of equations can be solved approximately using iterative methods (convergence is good in this case).
There is no need to solve 20 systems, 4 is enough.
Think about it!

## Genetics

### Solution

*Categories:* Greedy, backtracking, randomized or math.

**Idea**

Try to reduce the size of the string at every step $\Rightarrow$ use surgeries 1, 2 and 3 whenever possible.
Use surgery 4 (cut and paste) until surgeries 1, 2 or 3 can be applied.
Several cuts may be needed!

**Question**

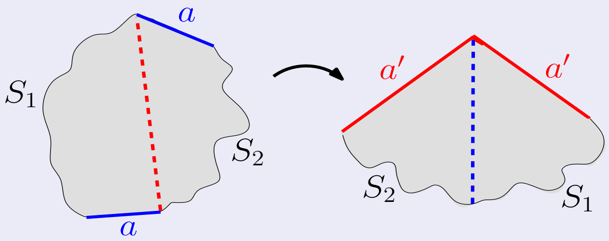
How to cut and paste properly to arrive to a reducible string?

# Genetics

## Solution

### First Solution - Special Cut and Paste

If a nucleotide (namely a/A) appears both times with the same face (a), do the following cut and paste



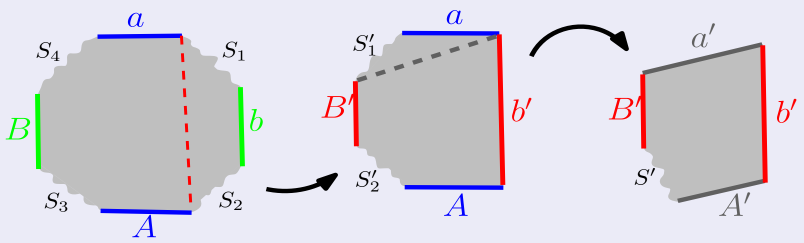
and the result is reducible using surgery 2 (+1 arm)

# Genetics

Solution

## First Solution - Special Cut and Paste (Continuation)

If two nucleotides (a/A and b/B) appear with different faces and alternatively, as in the figure, do the following two cut and paste operations



and the result is reducible using surgery 3 (+1 leg)
Convince yourself that these two rules are always applicable. **We are finished!** AC

# Genetics

## Solution

### Second Solution - Exhaustive search for a reducible string

Be careful, there are many possibilities for cut and paste ⇒ many reachable strings.
Normalization to reduce the number of strings is not enough.

It is not always possible to reach a reducible string with only one cut and paste!

Even a BFS limited to two levels will give TLE .
**DFS - Backtracking** works much better and it is a possible solution AC

# Genetics

Solution

## Third Solution - Randomized cut and paste

Idea: since the empty string is reachable from all valid strings, and a constant number of cuts suffice to reduce the string length $n$, taking random cuts will reduce $n$ in a polynomial expected number of steps.
AC
It works even better than DFS.

## Mathematical approach - Classification of closed surfaces

The string codifies a **closed surface** and we are asking you for **its Euler characteristic and orientability**.
Compute it using the well-known formula: $\chi = v - e + f$. AC

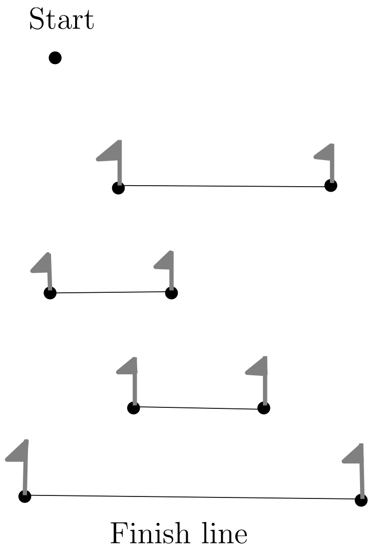
## Slalom
### Solution

*Categories:* Geometry + dynamic programming

### Task
Compute the minimum-length path from the starting point to the finish line.

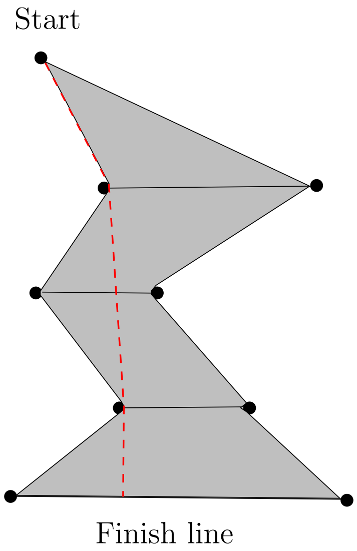
### First Reduction
The optimum path will be **piecewise linear**, composed of segments between vertices (poles), and maybe one final perpendicular segment And it will be contained in the polygon having all the poles as vertices.

## Slalom
### Solution



# Slalom

## Solution



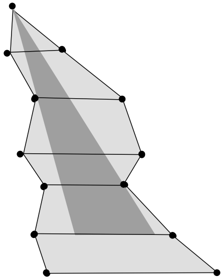
# Slalom

## Solution

### Question to deal with

Computing visibility between vertices inside the polygon in $O(n)$.

Take a vertex and compute visibility to the lower gates. Keep track of the angles determining the sector of visibility.



## Slalom
### Solution

#### Finish line

Be careful! You may finish with a segment **perpendicular to the finish line**, so you may need to compute visibility from any vertex to the finish line.

#### Got it!

It's almost done. **Run Dijkstra or simple DP** (the graph is a DAG) to compute the shortest path.

#### Complexity

Algorithm is $O(n^2)$ provided visibility is computed in $O(n)$.
Try to think about an $O(n)$ solution!

# Trick or Treat

## Solution

*Categories:* Geometry + Binary Search

**Problem:** Given a set of points $(x_i, y_i),\ 1 \leq i \leq n$ compute:

$$d = \min_x \left\{ \max_i \sqrt{(x - x_i)^2 + y_i^2} \right\} = \sqrt{\min_x \left\{ \max_i \{(x - x_i)^2 + y_i^2\} \right\}}$$

and the coordinate $x$ minimizing $d$.

## Trick or Treat

### Solution

#### First reduction

Minimum attained at a vertex of a parabola or at the intersection of two of them.
Solution: Compute all of them in $O(n^2)$ and check which one is the maximum. Total running time: $O(n^3)$. TLE

#### Second reduction

Each parabola has at most one intersection with another parabola, and once it stops being the topmost parabola, it never is again.
Solution: Keep track of the topmost parabola and compute the next intersection with the rest of the parabolas until we reach $x = \infty$. Total running time for the worst case: $O(n^2)$. Testcases designed for TLE.

## Trick or Treat

### Solution

**Heuristic**

Try to reduce the set of points by taking only the convex hull of them. Total running time: $O(n \log n + k^2)$, where $k$ is the number of points in the convex hull. Worst case: $O(n^2)$. Again, testcases designed for TLE.

# Trick or Treat

Solution

## First solution

**Binary search on the answer**

We can compute for a given distance $d$, and for each parabola, the set of points on $y = 0$ that are at a distance $d$ or less from the parabola. If empty, $d$ is too small; if the intersection has more than one point, $d$ is too large.

The intersection of these intervals can be computed in $O(n)$ time, thus the total time is $O(n \log(x_{max}/\epsilon))$. AC

## Second solution

Also a divide and conquer approach. Complexity $O(n \log n)$. AC.

## Third solution

Use **ternary search** to find $x$: the maximum of the parabolas is a function which is first decreasing then increasing. Again, the total time is $O(n\log(x_{max}/\epsilon))$. AC

## Fourth solution

**Clever search in the position x** moving the solution point at each step:

- if the farthest point from x is on the right, do $x = x + d$
- if it's on the left, do $x = x - d$
- else reduce the size of the steps, $d = d/2$.

Also $O(n\log(x_{max}/\epsilon))$: AC.

# Lights

## Solution

*Categories:* math, dynamic programming.

### Equivalent Task

Given $v \in \{0,1\}^n$, how many sets of $m$ distinct vectors in $\{0,1\}^n$ such that $v_1 \oplus v_2 \oplus \ldots \oplus v_m = v$ (addition is bitwise XOR).

Call the answer $f_m(v)$, let $g_m(v) = m! f_m(v)$ (focus instead on sequences, where order does matter).

# Lights

## DP solution

### Recurrence relation

- Choose the first $m-1$ distinct vectors; $(m-1)!\binom{2^n}{m-1}$ ways. $v_m$ is now determined.
- The only thing that can go wrong is $v \oplus v_1 \oplus \ldots \oplus v_{m-1}$ is again one of $v_1, \ldots, v_m$, e.g. $v_1$.
- But this implies $v_2 \oplus \ldots v_{m-1} = v$!
- $g_m(v) = (m-1)!\binom{2^n}{m-1} - (m-1)(2^n - m + 2)g_{m-2}(v)$; nothing is subtracted twice.
- One-dimensional dynamic program ($n$ is *not* a parameter!)
- Base case only depends on whether $v = 0$ or not.
- $f_m(v) = \frac{1}{m}\binom{2^n}{m-1} - \frac{1}{m}(2^n - m + 2)f_{m-2}(v)$.

$\Theta(m \log p)$ solution, if binomial coefficients mod $p$ are computed in $O(\log p)$ time (e.g. precompute factorials and divide mod $p$).

## Lights

### Closed-formula solution

Try to find a relationship between $f_m(0)$ and $f_m(v)$ for some $v \neq 0$.
Easy for odd $m$: $\sum_{i=1}^{m} v_i = 0$ if and only if $\sum_{i=1}^{m}(v_i + v) = v$. So the answer is $\binom{2^n}{m}/2^n$, regardless of $v$.

For even $m$: let $a = f_m(0)$, $b = f_m(v)$. If we add $v$ to $v_i$, the sum is 0 and all $v$'s are distinct iff for all $j$, $v_i + v_j \neq v$.
The only case when such $i$ does not exist is when $v_1, \ldots, v_m$ can be split into pairs of sum $v$. Therefore,

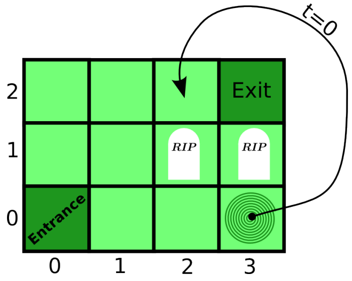
$$a - b = (-1)^{m/2}\binom{2^{n-1}}{\frac{m}{2}}$$
$$a + (2^n - 1)b = \binom{2^n}{m}$$

Solve for $a$ and $b$!

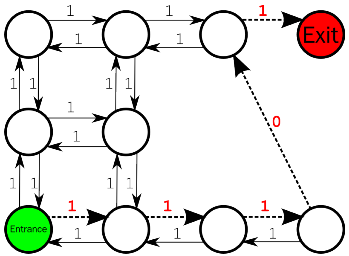
# Haunted Graveyard I

## Solution

*Categories:* graphs.



$\Rightarrow$



Single-Source Shortest Paths (edges may have negative weights).
**Algorithm:** *Bellman-Ford* $O(VE) = O((W \cdot H)^2)$.
Pitfalls:

- Negative cycles only matter if they are reachable.
- Even reachable cycles that do not lead to the exit matter.
- There are no edges leaving the exit cell.

## Stammering Aliens

### Solution

*Categories:* binary search + hashing, suffix trees, suffix arrays.

### Task

Given an string $s$, and an int $m$, find the size of the biggest substring repeated $m$ times (and rightmost position of such a substring)

### First approach

- Binary search in the size $s$ of the substring.
- For a given $s$, use **hashing** to count repeated substrings of size $s$. If hash is good, all hashes for these $n - s + 1$ substrings will be distinct with high probability.
- **Problem:** Hashing takes $O(n)$, this would be $O(n^2)$ TLE .
- If we go through all substrings of length $s$ in order, we can update the hash in $O(1)$!
- Complexity $O(nlogn)$ AC.

## Stammering Aliens

### Solution

#### Other approaches

- Suffix tree: the answer is the deepest node having $\geq m$ children. Naive $O(n^2)$ construction will TLE .

  Build in $O(n)$ with Ukkonen's construction: AC.
- Suffix array: the answer is the maximum, over $i$, of the longest common prefix between suffix $i$ and suffix $i + m - 1$ in the sorted array.

  Naive $O(n^2 \log n)$ sort construction will TLE.

  Use $O(n \log n)$ construction: AC .

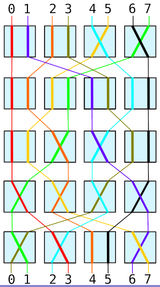
## Routing

### Solution

*Category:* divide and conquer, bicoloring

### Exploit recursive definition of Benes network

- If we knew how to select the state of the switches of the first and last row, we could recurse on the left and right parts of the newtwork.
- Which computer will follow a path in the left subnetwork and which in the right one?



## Routing
### Solution

- Computers that are inputs to the same switch in top row go into different subnetworks (left and right) because of perfect shuffle.
- Same for the bottom row.

**Bicoloring**

Build a graph: computers as vertices and edges between them if they must go into different subnetworks. Always bipartite!
Run bicoloring to divide computers and select switches appropriately (be careful with lexicographical smallest condition).

Complexity: linear in the size of the output AC .