# Solution outlines

## BAPC 2010 – Leiden

October 23, 2010

# D - Collatz

- With $N = 10^9$ a linear solution will be too slow
- Compute the number of ropes from $m$ to $2m$ with $m \leq N$ and $N < 2m$
- Compute the number of ropes from $n$ to $3n+1$ with $n \leq N$ and $N < 3n+1$
- In total $\lceil \frac{N}{2} \rceil + \lceil \frac{N}{2} \rceil - \lceil \frac{\lfloor \frac{N-1}{3} \rfloor}{2} \rceil$ ropes are cut
- This leads to an $O(1)$ solution

# I - Keylogger

- Simulate the keypresses, implementing each operation in $O(1)$
- Use a *Linked List* of characters, *or*
- Use 2 stacks
- This leads to an $O(N)$ solution, where $N = Length(L)$
- Note that using a vector or array will lead to a $O(N^2)$ solution

# A - Gene Shuffle

- Create an *inverse permutation* for $B$, so set $B\_pos[B_i] = i$
- Walk through $A$ from left to right, keep the start of current segment and maximum $B\_pos[A_i]$
- If the maximum position in $B$ equals the current position, print this segment and set start to $current + 1$
- This leads to an $O(N)$ solution

# A - Gene Shuffle

## Alternative solution

- Use 2 boolean arrays to indicate which numbers you have seen in each gene sequence
- Keep track of the number D of differences between these arrays
- Walk from left to right in both gene sequences and update D
- Whenever D $==$ 0, we found a new segment

# G - Snooker

- Simulate the game, keep the players' scores and remaining scores, stop when score difference is bigger than remaining
- Note that the number of points remaining may be different for both players.
- Tricky cases:
  - Player 1 wins when player 2 misses a coloured ball
  - Player 1 wins when he scores a red ball, because at that moment player 2 cannot play the following coloured ball anymore
  - The extra black ball rule
- This leads to an $O(N)$ solution

# B - Top 2000

- Solve with *Dynamic Programming*
- Let $DP[i]$ be the minimum penalty for a complete schedule of singles $1 \ldots i$
- To calculate $DP[i+1]$ from $1 \ldots i$, step by step take the previous single, the single before that, etc. in the current block
- Calculate the answers using the DP table
- Take minimum over these answers
- Stop when the length of the current block exceeds M. At that point, the only way to get an even better schedule may be to have one block with all singles $1 \ldots i+1$.
- This leads to an $O(N \cdot M)$ solution

# F - Maze Recognition

- Use a *Depth First Search* to discover the maze
- Make sure you handle every room only once
- Construct the graph from this
- Run a shortest path algorithm (for example Dijkstra's or Floyd-Warshall) to find the answer
- This leads to an $O(N)$ solution, where $N$ is the size (number of rooms) of the maze.

# J - Wrong Answer

- Let us call 2 overlapping words a *conflict*
- Notice that conflicts can only happen between a horizontal and a vertical answer
- Create a graph with a node for each answer, and an edge of weight = 1 for each conflict
- This graph is *bipartite*
- From the *Max-flow min-cut theorem* we know that the minimum number of answers that are wrong is equal to maximum bipartite matching in the graph
- Answer is H + V - maximum_matching()
- This leads to an $O((H + V)^2)$ solution

# C - The Twin Tower

- Represent a floor as a bitmask, an integer between 0 and $2^9$, where a 1-bit means that this room is already paired
- Use *Dynamic Programming* to find the number of combinations to pair one floor for each bitmask
- Again use DP to find the answer for every height for every set of rooms that are not paired yet
- Precompute this only once for all heights
- This leads to an $O(N \cdot 2^{18})$ solution, which is roughly $1.3 \cdot 10^9$ steps, after which every testcase can be handled in $O(1)$

# C - The Twin Tower

## Alternative solution

- Construct a $2^9 \times 2^9$ matrix of state transitions
- Use matrix exponentiation for finding the answer in $O(\log N)$ time
- However, $(2^9)^3$ operations is a bit too much, so use symmetries to make the matrix smaller
- This will lead to a matrix of approximately $100 \times 100$
- An odd $N$ always has answer 0
- Squaring the matrix and eliminating states that cannot be reached from state 0 leads to a $23 \times 23$ matrix
- This leads to an $O(\log N)$ solution, which can easily handle cases up to $N = 10^{18}$

# C - The Twin Tower

## Short solution

Based on the previous idea and some other mathematical insights, the following program solves the testset almost instantly:

```
#include <iostream>
using namespace std;
int i,j,k,a[23]={679,3879,6670,8547,8552,6663,7423,4872,9740,5808,9447,560,
                 4199,267,5135,2584,3344,1455,1460,3337,6128,9328,1},
    f[2501]={1,229,7728,2069,4990,1182,8338,3409,2588,4676,5205,8420,6749,
             6107,1784,4701,3574,6252,4989,1032,8473,8155,2806};
int main()
{
  for(j=23;j<5001;j++)
    for(i=0;i<23;i++)
      f[j]=(f[j]+f[j-1-i]*a[i])%10007;
  cin>>j;
  for(i=0;i<j;i++) { cin>>k; cout<<(k&1?0:f[k/2])<<endl; }
  return 0;
}
```

# E - Clocks

- Binary search on the radius of the clock
- To check if a clock with radius $nr$ fits, there are 3 possibilities for its centre-point:
  - Place it in the corner of the wall, so at $(nr, nr)$, $(W - nr, nr)$, $(nr, H - nr)$ or $(W - nr, H - nr)$
  - New clock touching the wall and a clock $i$, at most 2 possible centre-points per pair of wall and clock
    - For wall with $y = 0$, the clock must be at $y = nr$, so solve the equation $(x_i - x)^2 + (y_i - nr)^2 == (r_i + nr)^2$ for x
  - New clock touching 2 clocks $i$ and $j$, at most 2 possible centre-points per pair of clocks
    - Solve $(x_i - x)^2 + (y_i - y)^2 == (r_i + nr)^2$ and $(x_j - x)^2 + (y_j - y)^2 == (r_j + nr)^2$ for x and y
- For each possible centre-point, check if it does not overlap with the walls or other clocks
- This leads to an $O(C^3 \log(\min(W, H)))$ solution

# E - Clocks

## Alternative solution

- It is not too hard to see that in the optimal placement the clock touches at least 3 other clocks/walls
- There are 4 possibilities for the placement of the final clock:
  - New clock touching 3 walls
  - New clock touching 2 walls and 1 existing clock
  - New clock touching 1 wall and 2 existing clock
  - New clock touching 3 existing clocks
- This problem is known as *Apollonius' Tangency Problem* and can be solved algebraically
- This leads to an $O(C^4)$ solution

# H - Pie Division

- Basic idea: a dividing line that always goes through exactly 2 pieces, rotate this line and check all possibilities
- Start with a dividing line equally dividing the pieces
- Then, while we are not back to the initial configuration:
  - Check if the current division is a valid division (place the first piece on the line on the left and the second one on the right)
  - If there are more pieces on the left side, rotate the dividing line clockwise around the first piece until another piece is "hit"
  - Otherwise, rotate the dividing line clockwise around the other piece until another piece is "hit"
- Because there are no 3 pieces in a line, we can be sure that there is always a unique next piece, and that our algorithm will return to the initial configuration in $N$ steps
- This leads to an $O(N^2)$ solution