Waterloo Trainings Selection 1

Problem analysis

Artem Vasilev Pavel Krotkov

Brazilian ICPC Summer School, 2016

A. Tic Tac Toe

Problem statement

- Tic Tac Toe game on the $n \times n$ board
- Finishes when there are m similar next to each other
- Might be on a row, column or diagonal
- $n \leq 1000$
- Determine whether game is finished

A. Tic Tac Toe

Problem solution

- Let's develop solution for rows
- Other cases are similar
- $left_{i,j}$ – amount of cells equal to cell $[i, j]$ to the left from $[i, j]$
- $$left_{i,j} = \begin{cases} 0, \text{if } [i, j] \text{ is empty} \\ 1, \text{if } [i, j] \text{ differs from } [i, j-1] \\ left_{i,j-1} + 1, \text{if } [i, j] \text{ is equal to } [i, j-1] \end{cases} \tag{1}$$
- Game is finished, if $\exists i, j : left_{i,j} = m$
- All the error situations also can be found out

B. Nice Prefixes

Problem statement

- Count the number of strings with *nice prefixes* of length L over an alphabet of K symbols.
- A prefix of a string is *nice* if $\mid count(x) - count(y) \mid \leq 2$ for all characters x and y, where $count(x)$ is the number of occurences of x in the given string.

B. Nice Prefixes

Slow solution

- dp[n][x][y][z] is the number of strings of length n which have x characters occuring t times, y characters occuring $t+1$ times and z characters occuring $t+2$ times, where t is the minimal number of times any character occurs.
- $x + y + z = K$
- $xt + y(t+1) + z(t+2) = N \in [tK, (t+2)K)$. From this we can derive that $t \in [\lfloor \frac{N}{K} \rfloor - 2, \lfloor \frac{N}{K} \rfloor]$.
- Use matrix exponentiation to get dp[N] in $O(S^3 \log N)$ time. $\binom{K+2}{2} \leq 1326$ states, too large to fit in TL.

B. Nice Prefixes

Optimizing

- Consider a moment when the minimal amount of times any character occurs (x in definitions from the last slide) increased. Notice that for this string $z = 0$. Number of states with $z = 0$ is $K + 1$. We'll call such a state *interesting*.
- Number of interesting states is small enough so we can use matrix exponentiation to calculate the number of strings ending with a particular interesting state and visiting t interesting states inbetween for all $t \in [\lfloor \frac{N}{K} \rfloor - 2, \lfloor \frac{N}{K} \rfloor]$.
- Bruteforce the last state of our string ($O(K^2)$) and the last interesting state ($O(K)$). From these states we can derive t (the minimum of all $count(x)$). Sum over all possible pairs of these states will give the answer.

B. Nice Prefixes

Full algorithm

-
  1. Find the number of ways from all interesting states to all states.
  2. Find A^t, where A is the transition matrix between interesting states, for all $t \in [\lfloor \frac{N}{K} \rfloor - 2, \lfloor \frac{N}{K} \rfloor]$.
  3. Iterate over all possible pairs (last state, last interesting state), take the precomputed results from steps 1 and 2; add it to the answer.
- Total runtime: $O(K^3 \log N)$

C. Slalom

Problem statement

- Need to pass N pairs of gates
- Gates have different Y-coordinates
- Gates are shifted over each other along X-axis
- Vertical speed is constant for every pair of ski
- Horizontal speed $\in [-v_h; v_h]$

C. Slalom

Solution idea

- If we can pass all gates at speed V we can pass all gates at any speed $v < V$
- If we can't pass all gates at speed V we can't pass all gates at any speed $v > V$
- We can order all pairs of ski by speed and do binary search
- All we need to do is checking whether we can pass all gates at particular speed

C. Slalom

Checking of particular speed

- We start at point $(0, 0)$, our speed is s
- At the first gate $y = y_1$, $x \in [-\frac{y_1}{s} \times v_h, \frac{y_1}{s} \times v_h]$
- Since we need to pass the gate $x \in [max(x_1, -\frac{y_1}{s} \times v_h), min(x_1 + W, \frac{y_1}{s} \times v_h)]$
- We can store current range of possible x coordinate and update it gate-by-gate
- If at some point we can't pass the gate, this speed doesn't fit

D. Celebrity Split

Problem statement

- n items, each worth w_i
- $n \leq 24$
- $10^6 \leq w_i \leq 4 \times 10^7$
- We need to find two subsets with equal worth and maximize this worth

D. Celebrity Split

Solution idea

- Let's divide all items on two halves (maximum size of each – 12 items)
- For every half we'll calculate all possible partitions onto three parts (3^{12} variants)
- For every partition we need to know difference between Jack's and Jill's parts and total worth of sold property
- For every partition of the first half we'll find partition of the second half with same difference and minimum worth of sold property
- One of the combinations is the answer

E. Knight's Trip

Problem statement

- Knight can go for two cells along one of the axis and for one cell along another axis
- We need to find shortest path from $[0, 0]$ to $[x, y]$

E. Knight's Trip

Obvious case

- What are the constraints on x and y for us to know exact shortest path?
- $T = min(|x|, |y|) - ||x| - |y|| = 0 \pmod 3$
- In this case we are doing $\frac{T}{3}$ pairs of corresponding moves (like $(2, 1) + (1, 2)$) and then $||x| - |y||$ steps to create the difference between $|x|$ and $|y|$

E. Knight's Trip

Not obvious case

- What if it's not the case?
- Precalculate space around $[0, 0]$ for 10 cells in each direction
- Calculate distance from all precalculated cells satisfying the condition to $[x, y]$
- Answer is one of the calculated distances

F. Paintball

Problem statement

- Square paintball field 1000×1000
- n circles on in we can't go in ($n \leq 1000$)
- Cross the field from west to east

F. Paintball

Solution idea

- We go along nothern border
- If we meet a circle, we go along it's border counter-clockwise
- Going that way until we meet nothern/southern border or another circle
- We can precalculate all intersection points of all pairs of circles

G. Fire!

Problem statement

- Maze on a grid
- Some cells are on fire
- Fire spreads with $1 \frac{\text{cell}}{\text{minute}}$ speed
- We need to find an exit

G. Fire!

Solution idea

- Let's say we have a 3-D maze
- $[t, x, y]$ is $[x, y]$ cell after t minutes
- Now our fire doesn't spread
- Every move we go into next time level
- Let's do BFS and find an exit

G. Fire!

Implementation details

- Maze size: $1000 \times 1000 \times T$
- A lot of time, a lot of space
- But actually, we don't need to store all the information
- We need to store nearest fire location for every cell at the beginning (another BFS) to check whether $[t, x, y]$ is on fire
- Total amount of cells we are interested is not bigger then 1000×1000, because we don't need to go into the same cell twice

H. Alaska

Problem statement

- Automobile can ride 200 miles without charging
- We need to ride 1422 miles
- There are some charging stations along the way
- We need to check whether we can do it

H. Alaska

Problem solution

- The easiest problem of the contest
- Order all charging stations
- Check if all distances between consequent stations are less then or equal to 200 miles

I. Driving Range

Problem statement

- We have a network of cities and roads
- Automobile can ride driving range (x) without charging
- Automobile should be able to get from any city to any other city through any amount of cities
- x should be minimized

I. Driving Range

Problem solution

- If the driving range is x, we have only roads which are not longer, then x
- If the driving range is satisfying, then any range that is longer is also satisfying
- If the driving range is not satisfying, then any range that is sharter is also not satisfying
- We can do binary search to find optimal driving range

I. Driving Range

Implementation details

- When checking driving range x we do BFS on our graph to ensure it's connected
- During BFS we use only roads no longer then x

J. Buzzwords

Problem statement

- String S is given
- $|S| \leq 1000$
- We need to find the most popular x-letter combination for every x

J. Buzzwords

Solutions overview

- Several different solutions
- Suffix structures (for example suffix tree)
- Polynomial hashing – easiest for implementation

J. Buzzwords

Polynomial hashing: task overview

- We need to create function $f : String \rightarrow Integer$
- This function should give distributed values on different strings
- We should be able to calculate this function for all substrings of S in $O(|S|^2)$ time

J. Buzzwords

Polynomial hashing: method idea

- The following function is considered good at most cases
- $f(S) = (\sum_{i=0}^{|S|-1} S_i \times P^i) \mod M$
- P – some prime number
- M – some modulo

J. Buzzwords

Polynomial hashing: method features

- Probability of collision is considerable for $\sqrt{M}$ strings
- A lot of collisions on Thue-Morse strings when $M = 2^x$

J. Buzzwords

Polynomial hashing: calculating substrings hashes

- $f(s_{i..j}) = f(s_{i..j-1}) \times P + s_j$
- We can calculate hashes of all substrings starting at s_i for $O(|S|)$

J. Buzzwords

Problem solution

- We calculate hashes of all substrings with big enough modulo
- For every possible substring length we store a map from hash to amount of occurences
- Can easily find the most popular string

K. Ferry Loading

Problem statement

- We have n cars ($n \leq 100$)
- Each car has it's own weight (real number, ≤ 100)
- We need to divide them onto two subsets of *almost* equal total weight
- Weights are considered *almost* equal if their difference is less then 2%

K. Ferry Loading

Simplifying the problem

- Let's multiply all car weights on some number X
- We need to achieve the situation, when sum of the fraction parts is less then 1% of total car weight
- It can be achieved if sum of the integer parts is around 10 000
- Now we can throw out all fraction parts

K. Ferry Loading

Problem solution

- Now our problem became standard knapsack problem
- Knapsack problem can be solved with dynamic programming