ICPC Camp Solutions

days 3 and 4

pageno

Task: count all ways to go from “1234” to “page 12 of 34”

Catch: watch out for leading zeros

“0123” -> no solutions

“1099” -> the only solution is “10 of 99”, “1 of 099” is invalid

Special case: strings of even length

“4742” -> “47 of 42” is invalid (just use standard string comparison, first<=second)

mechsort

task: what is the heaviest inversion?

solution in O(n): for each element: what is the heaviest element before me, and is it heavier than I am?

sumlendiv

task: given a sequence and k, count all contiguous subsequences such that both lenght and sum is divisible by k

main trick: prefix sums

If S(x) is the sum of the first x elements, then the sum of the subsequence with indices [a,b) is S(b)-S(a)

For each x, compute the pair (x mod k, S(x) mod k).
Good subsequences are such that the pair for the beginning and the end is the same.

warehouse

task: find the diameter of the set of stores -- answer is ceil(diameter/2)

one possible solution: first, iteratively discard all leaves without a store until you have none; then, in each round discard all leaves and increase the answer by 1

easier to implement solution: we want two stores that are farthest from each other

1. start at any store A
2. using bfs find the store B that is farthest from A
3. using the same bfs find the store C that is farthest from B
4. BC is a diameter

interconnect

task: compute expected time to connect a graph when adding random edges

state = a **sorted** sequence of sizes of connected components

in other words, states = integer partitions of the number of vertices

useful observation: the number of integer partitions of n grows **exponentially in n**, but **the base of the exponent is very small**: n=30 only has 5604 partitions

solution: recursion with memoization over all states
to solve a state try all ways of adding an edge and then solve a linear equation

bipartite

task: remove at most half of a graph's edges to make it bipartite

a lovely task! it's hard because it's open-ended -- "how do I even start?"

the task is actually very easy:

there is always a solution

a simple greedy algorithm works:
add the vertices one at a time; when adding a vertex, count the number of edges leading into each partition, put it in the one with the smaller count and erase those

robot

task: find shortest lex smallest periodic program that makes a robot fall off a table

main solution idea: fix the cell (x,y) reached after the program is executed once

the program has to bring the robot from (0,0) to (x,y), then from (x,y) to (2x,2y), ...

for each i, cut out a rectangle of the map with corners (ix,iy) and ((i+1)x,(i+1)y)
overlay those rectangles (OR-ing obstacles)
find whether there is a valid path that does not hit any obstacle in any iteration

complete solution: iterate over all (x,y) in increasing order of x+y

badsquare

task: find a rotation of a sequence that maximizes total area of rectangles

for each k, we want:
- to compute sum(a[i]*b[k+i])
- to make sure that no term is 47*47

the first thing can be done in O(n log n) time using FFT
(it is called a “cyclic convolution”, best resource http://e-maxx.ru/algo/fft_multiply, use google translate)
the second thing can also be done, just define c[i]=(a[i]==47) and d[i]=(b[i]==47)

onthefarm

task: find lex min solution to a system of Diophantine equations

observation: cow+chicken = 2 pianos + 2 cabbages

lex min valid quadruple (pianos,cabbages,cows,chickens)
cannot have pianos>=2 and cabbages>=2

that leaves four easily solvable cases

excell

task: convert between two representations of Excel cells

easy-looking task with many pitfalls, mostly off-by-one errors

make sure to use 64-bit integers **everywhere**

test on large inputs for which you know the answer by hand

brazil

task: simulate a bureaucracy

Each request can be handled in constant (or expected constant) time.

Two possible approaches:

1. use linked lists, for each document remember the one below, for each stack remember the top and bottom (instead of pointers all info can also be stored in hashmaps)
2. use actual stacks and implement a way to push a whole stack into another (e. g., push “-stack_id”)

bdpq

task: check whether two strings have the same geometric shape

ugly task, avoid/postpone in real contest -- many places to make a mistake

catch: some symmetries are not stated explicitly and must be deduced

one possible approach: hard-wire a lot of constants; another approach:

```
char letters[7][120] = {
"....*.........*......**.....*...........*....*..............................*.........................",
"....*.........*.....*.......*...........*....*.........................***.***.....................***.",
"..*.***.***.***.**..**..***.***..*....*.*.*..*..***.***.***.***.***.*....*..*.*.*.*.*.*.*.*.*.*...*.",
".**.*.*.*...*.*.**..*...*.*.*.*..*....*.**...*..***.*.*.*.*.*.*.*.*.*...***..*..*.*.*.*.***..*..*.*.***.",
"***.***.***.***.***.*....**.*.*.***...*.*.*.***.*.*.*.*.***.***.***.*.....*..**.***..*..***.*.*..**.*...",
".........................*...........*.....................*.....*.....***.....................*.***.",
".......................***.........**.....................*.....*.............................*......",};
```

concave

task: count concave quadrilaterals

observations:
convex hull is a triangle
the shape of a triangle is determined by two vectors
<3000 possibilities for each vector

for each possible triangle shape:
count ways how to place the shape into the grid in O(1)
count inner grid points using Pick’s theorem -- using three calls to gcd()

colors

task: count the min number of colors needed for a special planar graph

the answer is always 1 or 2

incorrect approaches based on Euler’s theorem and similar results -- won’t work

easy solution: all unit squares adjacent to the line are vertices, adjacent vertices = line doesn’t pass between them.
find connected components, check whether two finite components touch

wireless

task: find Euclidean MST for random points

for small inputs, use Kruskal directly

for large inputs, divide the square into a grid of buckets, distribute the points into the buckets, and only generate edges between nearby buckets

(either guess/compute a good bucket size, or keep on adding longer edges until everything becomes connected)

convex

task: count convex quadrilaterals

try all possibilities for the axes-parallel bounding box

given the bounding box, try all possibilities for the placement of the quads' corners
solve each possibility separately

```
for (int tl=0; tl<2; ++tl) for (int tr=0; tr<2; ++tr) for (int br=0; br<2; ++br) for (int bl=0; bl<2; ++bl)
for (int t =0; t <3; ++t ) for (int r =0; r <3; ++r ) for (int b =0; b <3; ++b ) for (int l =0; l <3; ++l ) {
    int corners = tl+tr+br+bl, total = corners + t+r+b+l;
    ...
}
```

eatpizza

task: count valid ways to eat a pizza without it falling off a table

first step of a solution: where is the center of mass of a single piece?

either solve an integral, or solve numerically by averaging a lot of thin triangles

then the solution is a simple DP over all segments:
- try all possibilities which piece to eat next
- verify that remaining pieces did not drop
- solve each piece separately and multiply by a suitable binomial coefficient

vacuum

task: compute expected time until a Roomba vacuums the whole flat

solution: the implementation is a bit long but the idea is clear: DP over the whole state space

state = (which corridors have been vacuumed, which room am I in)

Gaussian elimination needed to solve multiple states at once.

stickers

task: find the number where we run out of stickers

starting observations:
- we will always run out of stickers (from 10k-digit numbers we are using more than we get)
- the first digit to be missing is 1

solution:
- write a function “how many 1s are in numbers 1..n?”
- use it to do a “compressed simulation”: e.g., if I have k=1, 100 spare “1”s and I’m in 3-digit numbers, I’m losing at most two “1”s per number => jump by 50.

party

task: a generalization of 2-SAT (with both existential and universal quantifiers)

without the universal quantifiers (people who ignore implications):
- represent each clause as two symmetric implications
- construct a directed graph on literals
- verify that no SCC contains both x and not-x

with the universal quantifiers: additional check needed:
- can we reach one universally-quantified vertex from another?