IDI Open
Programming Contest
April 21th, 2012

Solution sketches

Problem A

Boss Rush

Problem author: Ruben Spaans

The main observation is that the three weapon categories are independent of each other. We can then do bipartite matching for each category separately. Create a graph with the bosses on the left side and the weapons on the other side, two of each. There is an edge between a boss and a weapon (both copies) if the weapon can be used to defeat the boss. Then, for each boss (starting with the first one), try to find an augmenting path to an unused weapon. Abort when either all bosses are matched, or a boss is encountered for which no augmenting path exists. Do this for all three weapon categories. The answer is the minimum of the results from the three weapon categories.

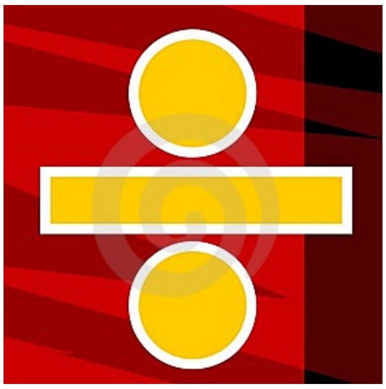
It is also possible to use a regular max flow algorithm. Create a graph as described above, and add a source with edges to each boss, and add a sink with edges from each weapon. Here, only one of each weapon is needed. Each edge has a capacity of 1, except from the edges from the weapons to the sink, which should have capacity 2. Add edges from the source to bosses in increasing order, and find an augmenting path for each edge added. Abort when no augmenting path can be found.

A map from string to integer can be used for converting the strings in the input into node identifiers.

Problem B

Decode the Message (Easy)

Problem author: Geir-Arne Fuglstad



This problem can be solved by reading in the string and then following the instructions given. Initialize a variable with value equal to 0, then use this variable to keep track of the value of the current word. Each time a lower-case character is encounter its value is added to the variable, and each time a space is encountered calculate the value modulo 27, output the corresponding character and set the variable to zero. Finally, there is no space in the end of the string so the character corresponding to the last word is outputed after iterating through the string.

Problem C

Troublesome Tools

Problem author: Jon Marius Venstad

The tools will always be divided into disjoint subsets, where the tools in each subset cannot be discriminated by current knowledge:

1. Initially all tools are in the same subset.
2. A subset of tools j that are picked from a subset i will form a new one, and i will be replaced by $i \setminus j$.
3. The picking in (2) can be done in $\binom{n_i}{k_i}$ ways for each subset i, for a total of $\Pi_i \binom{n_i}{k_i}$ These may be large numbers, so Pascal's Triangle modulo $2^{31} - 1$ is advisable. (Alternatively division modulo $2^{31} - 1$ works fine, since this is a prime group.)
4. Add the results from (3), modulo $2^{31} - 1$.

For a fast representation of the subsets, let each tool keep track of which subset it belongs to, and let each subset only keep track of the split-off subset for each step (2), and its size.

Problem D

Civilization (Easy)

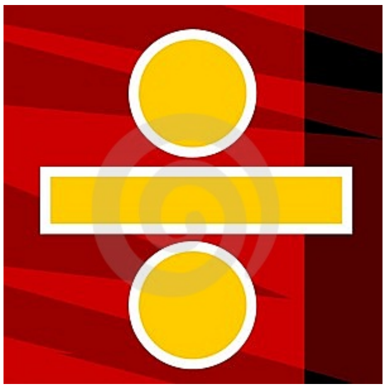
Problem author: Ruben Spaans

The maximal input is 18 regions, and each can be either included or excluded. Trying all 2^{18} combinations will run well within the time limit. This can either be done with a backtracking algorith that either picks or doesn't pick a region in each step, or one can do a loop with 2^{18} iterations where each of the 18 bits indicate whether the region should be included.

Problem E

Inverse Divisor Sums

Problem author: Ruben Spaans



The solution is backtracking: Try all ways of factoring the number N into factors $\frac{p_i^{a_i}-1}{p_i-1}$, and at the same time reconstruct the original number M by multiplying together $p_i^{a_i}$. Call the backtrack function with the initial values $m = 1, n = N, p = 2$. For each step in this function, try all powers of p such that the above expression divides n. If it does, divide n by this expression and multiply m by p^a where a is the power found, then call the backtrack function with p equal to the next prime. Whenever $n = 1$, m contains a number such that sum of divisors$(m) = n$.

Some optimizations are required in order to pass the time limit. The two following will suffice:

- When the backtrack routine is called with a p satisfying $\frac{p^2-1}{p-1} > n$ there can only be one factor left. If $n-1$ is prime, then $n-1$ is the remaining factor and we can add $m(n-1)$ to the list of outputs.
- There is only a limited set of values for p, a such that $\frac{p^a-1}{p-1}$ divides N. Precalculate all such values before doing backtracking, and only use these values during the backtrack.

Problem F

Longest Common Path

Problem author: Ruben Spaans

The main observation is that Per and Pål's common path is always at the beginning of their paths. In an optimal common path, they'll never rejoin once they split.

Run one-to-all shortest path from school, and from each home. Then try all nodes and check if they could have departed from each other at this node. If going via this node doesn't make any of their paths longer, the path from school to this node is a candidate for the longest common path. Check the length, and update the length of the longest common path seen so far.

Floyd-Warshall isn't fast enough for this problem, Dijkstra was needed in order to pass within the time limit.

Problem G

Birthday Party

Problem author: Geir-Arne Fuglstad

This problem can be solved by inclusion-exclusion type of counting. Start by initially counting too much, then remove too much, include too much again and so on. The problem can be rephrased as a questions about graphs, what is the probability that there is one or more cycles of length k in a graph with N nodes where every node has exactly one directed edge to another node?

Let

$$T_1 = \binom{N}{k}(k-1)!\cdot(N-1)^{N-k} = \frac{N(N-1)\cdots(N-k+1)}{k}(N-1)^{N-k},$$

that is the number of ways to choose k people and put them in a cycle times the number of ways to choose the edges for the remaining nodes. This necessarily counts a lot of things too many times. Configurations with exactly one cycle is counted once, configurations with exactly two cycles are counted twice, configurations with exactly three cycles are counted thrice and so on.

We then compute

$$T_2 = \frac{1}{2!}\binom{N}{k}(k-1)!\binom{N-k}{k}(k-1)!\cdot(N-1)^{N-2k} = \frac{N(N-1)\cdots(N-2k+1)}{2!k^2}(N-1)^{N-k},$$

which counts the number of ways to choose $2k$ nodes from N nodes and put them in two cycles times the number of ways to choose the edges for the remaining nodes. Now we have counted configurations with exactly one cycle 0 times, configurations with exactly two cycles ones, configurations with exactly three cycles $\binom{3}{2} = 3$ times and so on.

We countinue this l times until $N - (l+1)k < 0$ and take

$$S = T_1 - T_2 + T_3 - T_4 + \ldots + (-1)^{l+1}T_l.$$

We can confirm that each configuration with exactly a cycles is counted

$$\binom{a}{1} - \binom{a}{2} + \binom{a}{3} - \binom{a}{4} + \ldots + (-1)^{l+1}\binom{a}{a} = 1 - (1-1)^a = 1$$

times. So our total answer is

$$P = S/(N-1)^N.$$

For computational purposes the $P_i = T_i/(N-1)^N$ should first be computed in order, where each P_i is calculated by multiplying P_{i-1} by some additional terms, and then summed in an alternating sum to get P.

Problem H

Holey Road

Problem author: Ruben Spaans

This is a shortest path problem. Before running Dijkstra or something similar, it is necessary to find out what the graph is. It is not sufficient to let a hole be a node in the graph, as the car can arrive at different positions at the boundary of the circle.

When the car travels from one hole to another, it will travel along one of four possible tangents. These can be calculated by setting up the necessary equations[1]. When the car travels to the next hole, it needs to travel along the boundary of the circle. This distance can be calculated by finding the angles of the two tangents on the current circle, taking their difference and multiply by πr. It is only possible to travel along a tangent if it does not intersect with other holes in the road.

Construct a graph where the nodes are the start position, destination and all possible tangent points. Create an edge for each tangent that does not cross any other hole, and between each tangent point on a single circle. Then, run your favourite shortest-path algorithm (Dijkstra, for instance).

[1]See `http://en.wikipedia.org/wiki/Tangent_lines_to_circles#Algebraic_solutions`

Problem I

Candy Store

Problem author: Ruben Spaans (idea by Magnus Lie Hetland)

This problem can be solved with dynamic programming. This is a variation of the knapsack problem. But instead of maximizing the value of the items we wish to pick, we want to calculate the number of ways we can pick candy so that the total cost is at least C kroner.

Let a_i be the cost of candy type i, and $T(i, c)$ be the number of ways to spend exactly c kroner by buying candy from the first i types (inclusive). To arrive at $T(i, c)$ we either bought or didn't buy the candy of type $i - 1$; the number of ways to do so is $T(i - 1, c - a_i)$ and $T(i - 1, c)$, respectively. Hence, we get the recurrence

$$T(i, c) = T(i - 1, c) + T(i - 1, c - a_i).$$

There are two essentially different ways to calculate the answer: Calculate

$$\sum_{i=C}^{\infty} T(n, i)$$

or

$$2^N - \sum_{i=0}^{C-1} T(n, i).$$

The second way is faster, but both ways would pass within the time limit. Naturally ∞ in the first sum would be replaced by the total cost of all candy types, as there are no ways to spend more money than that.

Problem J

Rotating Penguin Maze

Problem author: Jon Marius Venstad

1. Find the path using BFS or DFS, while storing the pressure plates along the path.
2. Then loop through the path, keeping track of the number of pressure platess activated.
3. "Rotate" the directions accordingly when printing them.