IDI Open Programming Contest April 17th, 2010

Solution sketches

A Guarding the Border
B Beehive Epidemic
C Mobile Gaming
D Balancing Weights
E Ambulance Antics
F Nurikabe
G Cookie Monster
H Typing Monkey
I The Diligent Cryptographer
J Combat Odds

Problem A

Guarding the Border

Problem author: Eirik Reksten

Imagine you were solving an easier version of the problem, where you were asked whether or not you could achieve a maximal distance of D using at most M towers. This problem can be solved using a simple iterative loop through the segment, adding a new tower at every D'th location. Using this as a subroutine, you can solve the original problem by binary searching on the maximal distance, returning the lowest distance where this is possible. This solution runs in $O(N \times log(L))$ time.

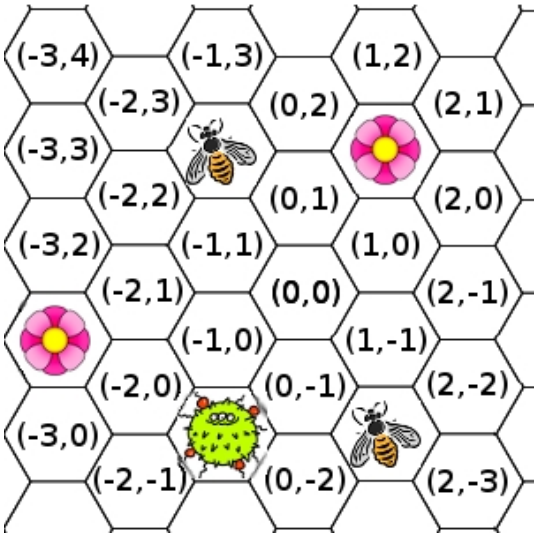
Another approach is using a greedy algorithm. Observe that between any two of the old towers, the new ones will/can be evenly spaced out in an optimal solution. Therefore, the maximal distance for a single segment is defined by its length and amount of new towers. Now, if adding another tower is to improve the solution, it has to be added to the segment with the highest maximal distance (otherwise the solution won't improve, and you'd have to add one to this segment later anyway).

A greedy method thus proceeds by always adding the next tower to the segment with the highest maximal distance. Using a heap (ie. PriorityQueue in Java) to keep track of the segment allows this algorithm to run in $O(M \times log(N))$ time.

Problem B

Beehive Epidemic

Problem author: Eirik Reksten



In this problem, you're trying to match every bee with a corresponding safe zone, aiming to maximize the amount of safe bees. This part is easily solved using a maximal beepartite matching algorithm (such as using max-flow).

To figure out which safe zones a certain bee can reach without infection, you need to run a breadth-first search from that bee. A coordinate system is already shown in the image, so the only problem with this is to check whether or not a cell is infected by a bacteria in sufficient time. In order to do this, you need to precompute the infection times for each cell. That means another breadth-first search, where you start by adding all bacteria locations to the queue. Now you also need to make sure to let the breadth-first searches run sufficiently far out from the objects initially, since a bee might want to take a long detour in order to reach a safe zone safely.

Also note that bees, safe zones and bacteria can all be in the same location at the beginning. A bee starting on a bacteria will never make it (see the notes and constraints section).

Problem C

Mobile Gaming

Problem author: Eirik Reksten

The important observations are that both the cop and robber have constant speed along both dimensions and that during a collision they'll overlap in both dimensions as well. If you know the time interval the sprites overlap in each dimension, you can take the intersection of these. The answer is the start point of this intersection interval.

Calculating the time interval of overlap in one dimension is only a matter of simple calculation and comparison.

Problem D

Balancing Weights (Easy)

Problem author: Eirik Reksten

The important realization in this problem is that since you only need the sign of the answer (`-, + or 0`), and the moment of inertia always will be positive, all that matters is the sum of the torques applied from each weight. Moreover, as all masses are the same, you simply need to add all distances in order to find this sign. If the sum of distances is less than zero print `Left`, if its more than zero print `Right`, or print `Equilibrium` if it is exactly zero.

Problem E

Ambulance Antics

Problem author: Ruben Spaans

Brute force is needed to solve this problem, every combination must be examined. For each trip the ambulance makes, every combination of picking up 1, 2 or 3 patients must be explored.

Standard brute force will time out. In order to get the algorithm fast enough, dynamic programming can be used. The subproblem is a bitmask over all intersections (except the hospital), where bit i is set if the patient at intersection i hasn't been picked up. The asymptotic runtime of this algorithm is $O(2^n \cdot n^2)$.

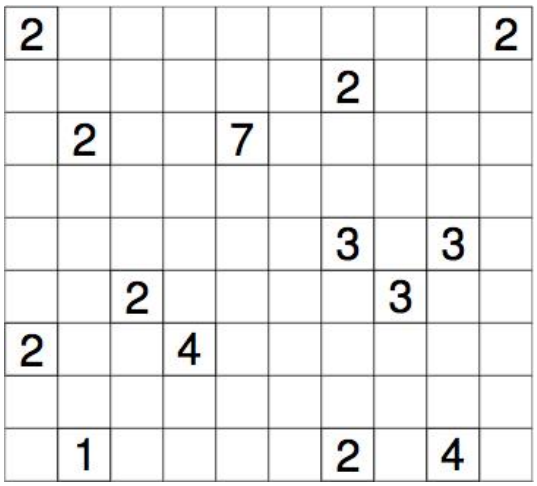
One more optimization should be required in order to pass. Always pick up the lower-numbered untaken patient whenever the ambulance is empty. This will reduce the state space significantly.

In addition, as a first step, one should find the shortest paths between each pair of nodes in the graph, since they will be used throughout the algorithm above. Floyd-Warshall is fine for this step.

Problem F

Nurikabe

Problem author: Ruben Spaans



This is a pretty straightforward problem to solve, one merely has to check if all the rules are followed for a given board. The following is one way to do it:

First, loop over every 2×2 box in the grid and check if there is one that contains only water cells. If it does, we know that the board is not valid.

Then, pick an arbitrary water cell and do a breadth-first search through all neighbouring water cells. If there exist any water cells on the board after this search, there exist disconnected water cells and the board is not valid.

At last, run breadth-first searches from every cell containing a number. If the number of cells reachable from the numbered cell differs from the given number, or more than more numbered cell was found on the same island, the board is not valid. Finally, if there are island cells left after doing the breadth-first searches, the board is not valid since each island must contain a number.

If all of the checks passed, we have a valid board.

Problem G

Cookie Monster (Easy)

Problem author: Eirik Reksten

This was probably the easiest problem in the set, as it's simply asking for the answer to $\lceil N/C \rceil$.

As the constraints were pretty kind as well, it suffices to convert to a floating-point number and use the internal ceil() of the programming language

Some simple Java solutions:

```
System.out.println((N+C-1)/C)
```

```
System.out.println(N/C+(N%C==0?0:1));
```

```
System.out.println(Math.round(Math.ceil(N/C)))
```

Problem H

Typing Monkey

Problem author: Ruben Spaans

The problem can be viewed as a markov chain, since the probability distribution for the next state is only dependent on the current state. A state is defined as a pair of indices (a, b) in the two given strings P and Q, where a means that the last a characters typed by the monkey is the same as the prefix of the first string, and similarly for b and the second string. There are a maximum of $n = |P| + |Q| + 1$ states.

We can create a transition matrix P of size $n \times n$, where P_{ij} is the probability of moving to state j, given that we are currently in state i. We want to define the states $(|P|, \cdot)$ and $(\cdot, |Q|)$ such that we stay in these states with probability 1. It can be shown that P^2 is a transition matrix for moving from one state to another in two time steps. This can be generalized and hence we need to determine

$$\lim_{n \to \infty} P^n.$$

In order to approximate P^∞, we can take P to a huge power using fast matrix exponential: Square the matrix a lot of times. It turns out that $P^{2^{50}}$ is sufficient, which can be calculated by repeatedly squaring P 50 times. When we have an approximation of P^∞, we can simply read off the desired probabilities p and q, the probabilities of typing the first word and the second word, given that we start in state $(0, 0)$. The answer is p, but since we might still have non-zero probabilities of not having found a word yet, we calculate the answer using $\frac{p}{p+q}$.

There is another way of solving this problem which is capable of solving test cases with larger strings. Using generating functions one can set up infinite sequences of typing sequences leading to sequences ending in the desired words. By manipulating the expressions representing these sequences, one eventually arives at a simple sum formula. Interested readers may consult the judge solution, or Concrete Mathematics (Graham, Knuth, Patashnik) for a derivation of this formula for an alphabet with two letters with equal probability.

It is possible to set up the relations between the states as a system of linear equations. However, Gaussian elimination with double precision does not give sufficient numerical stability to solve this problem. Implementing Gaussian elimination with fractions where the numerator and denumerator are arbitrarily sized integers will work.

Problem I

The Diligent Cryptographer

Problem author: Tor Gunnar Houeland

Any cryptographic key is valid for the new system, so the task in this problem is to determine whether the input is valid for the old system, and output `new` or `unknown` accordingly.

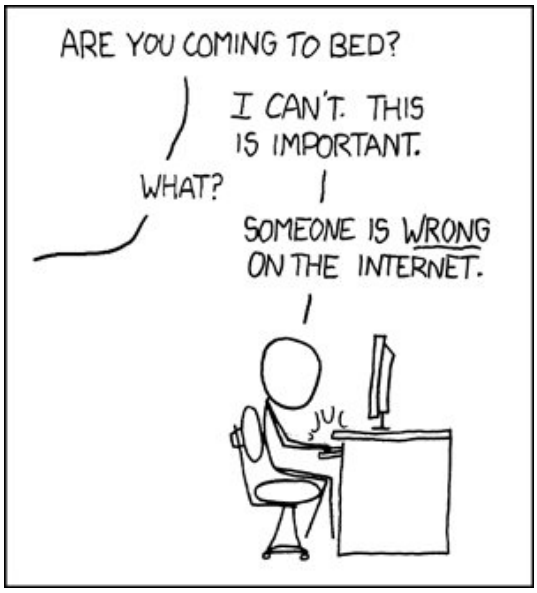
Create a boolean array (or use a bitmask) to store whether a letter has already been used or not, and read through the input one letter at a time. If a repeated letter is found, check whether the previous input was a valid permutation (i.e. the N previously read letters are the letters from `A` to the Nth letter). If it is valid, check whether the key is a repetition of the initial permutation by iterating through the key and comparing each letter to the same index modulo N.

The key is also valid if there are no repeated letters.

Problem J

Combat Odds

Problem author: Eirik Reksten



The intended solution for this problem is using dynamic programming. Lets rephrase the original question to

> What is the probability of observing a sequence of L losses in a sequence of $n + k$ combats, where the k first combats result losses?

We can define the function $P(i, j)$ to be this probability for $i + j$ combats starting with j losses. We know that the next combat will result in a win with probability p or a loss with probability $1 - p$. A win will take us to the state $i - 1, 0$, while a loss will take us to the state $i - 1, j + 1$. This allows us to formulate the following recursive function:

$$P_{i,j} = p \times P_{i-1,0} + (1 - p) \times P_{i-1,j+1}$$

We also formulate the following terminating equations:

$$P_{i,L} = 1.0$$
$$P_{i,j} = 0.0 \text{ if } i + j < L$$

Using these equations, we can solve the problem using memoization or dynamic programming.