Problem A

Help Chelsea!(easy)

Problem author: Truls A. Bjørklund

This is considered to be the simplest task in this problem set. You should essentially find the largest value in a list of integers. If you are able to parse integers and strings in a programming language, all you need is a for-loop to find the maximum number and print the associated name.

Problem B

Virus

Problem author: Øyvind Grotmol, Medallia

It can take as long as $2 \cdot 10^{10}$ for the virus bomb to go off, so we cannot simply simulate the virus growth time step by time step. The crucial idea is that we can represent the number of viruses of each form and the number of tritium atoms by a vector of length $N + 1$, and that the action in one time step is simply multiplying this vector with the matrix given in the input augmented with a row 0 0 ... 0 1. The state in time t is thus this matrix to the power t multiplied by the initial vector 1 0 ... 0 0. Since calculating the power of the matrix can be done in time logarithmic in t, we can find the number of tritium atoms at any given time pretty fast. Thus, we can use binary search to find out how long time it takes before the bomb goes off (if it does). The easiest way to avoid overflow during the calculations, is to use longs and cap them at L.

Problem C

Frogger

Problem author: Øyvind Grotmol, Medallia

There are only 32 possible test cases, so we can pre-generate the answers. There's a number of approaches to have the computer search for the solutions, but with some ingenuity we can do it all with mathematics.

Let's first establish a lower bound for the number of frogs needed by finding an invariance in the form of a weight associated with each frog such that the sum over all the frogs cannot increase. $k^{d(x,y)}$ where $d(x,y) = |x - X| + |y|$ is the Manhattan distance from (x, y) to the fly at $(X, 0)$ and $k = \frac{\sqrt{5}-1}{2}$ solves the equation $1 = k + k^2$ is convenient because a jump in the direction of the fly will exactly preserve the total sum (the weight of the exploded frog plus the previous weight of the jumping frog equals the new weight of the jumping frog), and any jump away from the fly will decrease the sum. If $X = 5$, and you imagine filling the entire half-plane with non-positive x coordinates with frogs, the sum of all their weights is exactly 1. Thus, it's impossible for the frogs to ever reach $(5, 0)$, since no matter how many frogs you start with, their sum will always be slightly less than 1, which would be the weight of a frog sitting at $(5, 0)$. Trying to reach the fly in this case will only lead to a mayhem of exploding frogs. Of course, it's not possible to eat the fly for $X > 5$ either.

We already have the answer for $0 \leq X \leq 3$ since it's given in the input (it's easy to find working frog configurations with the given sizes, and the bounds are tight with respect to the invariance).

So only $X = 4$ remains. Using the invariance above, you'll find that you need to add up the 19 highest-weighted positions to reach 1. However, no configuration of 19 frogs will actually be able to get to the fly (remember that the invariance only gives a lower bound, not an upper one). It is possible with 20. To figure this out, you could either just convince yourself using pen and paper, or write a computer program to solve this case.

Problem D

Conquistador

Problem author: Nils Grimsmo

The main obstacle in this problem is the parsing. If you are using Java, the easiest approach is to use `string.split(regex)`. For the costs, split the input string first on ”`,`”, and then on ”`:`”. You could put the mapping from criterion names to costs in a `HashMap<String,Integer>`. For the string of combinations, split first on ”`\\|`” and then ”`&`”.

What you are searching for is the minimum cost among the combinations, where the cost of a combination is the maximum of the costs of the criteria it requires.

Problem E

Party

Problem author: Truls A. Bjørklund

This problem is also considered to have medium difficulty. When reading the problem text it should be obvious that we have a bipartite graph here, consisting of nodes representing the nurses on one side, and the geeks who needs a date on the other. There is an edge between a nurse and the geeks she thinks it is OK to date.

As is quite common in bipartite graphs, it will be helpful to add two extra nodes, one source and one sink. Every geek has an edge to the sink, and from the source, there is an edge with capacity x to every girl.

If we set $x = 1$ we have a flow network. If we find the max-flow in this network we will actually be able to determine whether it is possible to find a date for all geeks in one party. In that case, the max-flow will be equal to the amount of geeks. If we set $x = 2$, we find the same with two parties.

Hence, the only thing we have to do is to vary x until we find the smallest value of x that gives a max-flow equal to the number of geeks. If that does not happen even when x is similar to the number of geeks, it is not possible, and we should print so.

To vary x can be done in several ways, and a binary search is probably an intuitive approach. With most max-flow-routines however, it will be more efficient to just do a linear search, because we can keep the current flow between the runs. But, a solution with binary search will also be accepted.

Problem F

Save the computer!

Problem author: Truls A. Bjørklund

This is one of the problems with medium difficulty in this problem set. Because we are allowed to assume that the probability that one component fails is independent of the failure of other components, we can easily find a formula for the probability that our computer will survive one semester. Let us first define sc_i to be the amount of spare components no i we have. This leads to the formula given in equation 1.

$$P = \Pi_{i=1}^{c} P_i(k \leq sc_i) \tag{1}$$

We must also obey the following constraint:

$$\sum_{i=1}^{c} sc_i * p_i \leq b$$

We note that equation 1 requires us to calculate the cumulative probabilities, by summing up.

One possible solution to this problem is to try all possibilities, but that will lead to a solution that is not efficient enough. We are thus lead to think about solutions based on dynamic programming. There are several possible ways to use dynamic programming here, but not all of them leads to solutions that are efficient enough. For a solution to be accepted, it should have a running time of $O(cb)$. An example of such as solution is outlined here.

Let us define $sc_{i,b}$ to be the optimal number of components of type i to buy when we have a budget of b. We can then find the optimal probability of having a working computer throughout the semester from the following recursive formula:

$$S(i) = \begin{cases} 0.0 & \text{if } b < 0 \\ \Pi_{i=1}^{c} P_i(k \leq 0) & \text{if } b = 0 \\ \max\left(S(i-1), \max_{0 \leq j < c}\{S(i-p_j) * P(k \leq sc_{j,i-p_j}+1)/P(k \leq sc_{j,i-p_j})\}\right) & \text{else} \end{cases}$$

As is probably obvious from the formula, we need to maintain two tables in a solution based on this formula, one for S(i)'s of size c, and one for the $sc_{i,b}$'s, of size cb.

We also need to calculate the cumulative probabilities. There are several possible ways to do this. The fastest one in the sample solutions is to memoise them.

Problem G

Fridge of Your Dreams

Problem author: Nils Grimsmo

The main obstacle in this problem is reading the binary number. In decimal notation, if the ith digit counting from the *right* is x, this digit contributes $x \cdot 10^i$ to the total value of the number. In binary notation, it contributes $x \cdot 2^i$. If B is a string of bits of length m, indexed from the *left*, the total value represented is

$$\sum_{i=0}^{m-1} B[i] \cdot 2^{m-1-i}$$

A simple program calculating this is

```
$v \leftarrow 0$;
**for** $i \leftarrow 0$ **to** $m - 1$ **do**
    $v \leftarrow 2 \cdot v + B[i]$
**end**
```

If you use Java, a simpler solution is to use built-in methods:

```
Integer.parseInt(myString, 2)
```

Problem H

Scorched earth

Problem author: Truls A. Bjørklund

This problem is a physics problem, and the programming challenges are quite moderate. By analysing the problem, we come up with the following equations:

$$tv\sin(d) - \frac{1}{2}gt^2 = y_o - y_u$$

$$tv\cos(d) + \frac{1}{2}wt^2 = x_o - x_u$$

The simplest approach from here on is to find v from one of the equations and replace all occurrences of v in the other.

$$v = \frac{2(y_o - y_u) + gt^2}{2t\sin(d)}$$

$$\frac{2(y_o - y_u) + gt^2}{2\tan(d)} + \frac{1}{2}wt^2 = x_o - x_u$$

$$t^2 = \frac{2\left(x_o - x_u + \frac{y_u - y_o}{\tan(d)}\right)}{\frac{g}{\tan(d)} + w}$$

We obviously only want positive values of t:

$$t = \sqrt{\frac{2\left(x_o - x_u + \frac{y_u - y_o}{\tan(d)}\right)}{\frac{g}{\tan(d)} + w}}$$

We will be able to determine that it is impossible to hit the opponent if t is an imaginary number. Otherwise, we can calculate v from:

$$v = \frac{y_o - y_u + \frac{1}{2}gt^2}{t\sin(d)}$$

If we end up with $0.0 \leq v \leq 300.0$, we have a solution we should output. Otherwise, we should output “impossible”.

Problem I

Free Willy

Problem author: Øyvind Grotmol, Medallia

The state space here can be as large as $26! = 4 \cdot 10^{26}$ permutations of 26 letters. We can reach at most 10^{10} of those with 10 moves, but it's still way too many alternatives to explore. The crucial idea here is that we can search backwards from the target as well, applying the permutations in reverse. Then we only need to generate the 10^5 words that can be reached in 5 steps from the starting word, and the 10^5 words that can reach the target word in 5 steps. We can check if these sets overlap in constant time per word by using hashing.