Editorial - Vacation Contest 1

A. Precious Gift

Preparation by: @Aniket54
Editorial by: @siddhantnema

Let us see the trivial cases first.

If $a \leq b$, then Bob has enough money to gift Alice the very first day. The answer in this case is simply b.

Now if $a > b$,

- If Bob is creating more (or equal) bugs than he is fixing i.e. $d \geq c$,then he cannot earn enough to reach amount a.
- If $d < c$, then Bob must be earning $d - c$ jewels every day. So he must work for ceil($(a - b)/(c - d)$) days. Now multiplying this with c and adding to b (counting first day) we get our answer.

Code:

```
from math import ceil

def inparr():
    return [int(i) for i in input().split()]

def solve():
    a,b,c,d = inparr()
    if b>=a:
        print(b)
        return

    if d>=c:
        print(-1)
    else:
        a -= b
        res = b + ceil(a/(c-d))*c
        print(res)

for _ in range(int(input())):
    solve()
```

B. Maximal Spice Harvest

Preparation by: @accord
Editorial by: @accord

Clearly, we must harvest spice during the time while we are not under attack. In the intervals that we are not under attack, we must harvest as much spice as possible. Therefore, our answer is simply, (assuming $t_0 = d_0 = 0$)

$$\sum_{i=0}^{n-1} \left\lfloor \frac{t_{i+1} - (t_i + d_i)}{s} \right\rfloor + \left\lfloor \frac{M - (t_n + d_n)}{s} \right\rfloor.$$

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
        int n, M, s; cin >> n >> M >> s;

        int t[n+1], d[n+1]; t[0] = d[0] = 0;
        for(int i = 1; i <= n; i++) {
                cin >> t[i] >> d[i];
        }

        int ans = 0;
        for(int i = 0; i < n; i++) {
                ans += (t[i+1] - (t[i]+d[i]))/s;
        }
        ans += (M - (t[n]+d[n]))/s;

        cout << ans;
}
```

C. Jumbled Numbers

Preparation by: @shxun
Editorial by: @shxun

- What is the sum of the elements in b? It is $x + 2 \cdot sum\ of\ elements\ in\ a$.
- Let A and B be the sum of all elements in a and b respectively.
- Let's sort the array b.
- If A is one of the elements of b there are only two places that A can be found in b:
    1. Add the end of the array in which case x lies somewhere before the last element.
    2. As the second last element, in which case x is the last element.
- For the first case:
    1. We can iterate over the first $n+1$ elements of b.
    2. If $B - b[i]$ is equal to twice the last element, then $b[i]$ is x and the rest of the list b excluding the last element is a possibility for a.
- For the second case:
    1. If the last element is x then the sum of the first n elements should be equal to the second last element.

2. In this case the first n elements are a possibility for a.

- Failing both the above cases it is not possible to generate the list a.

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

void solve() {
    int n; cin >> n;
    vector<int> b(n + 2);
    ll sum = 0;
    for (int i = 0; i < n + 2; i++) {
        cin >> b[i];
        sum += b[i];
    }
    sort(b.begin(), b.end());

    // If the last element is the sum.
    ll last_ele = b[n + 1];
    for (int i = 0; i < n + 1; i++) {
        if (sum - b[i] != 2 * last_ele) continue;
        for (int j = 0; j < n + 1; j++) {
            if (j != i) cout << b[j] << ' ';
        }
        return;
    }

    // If the second last element is the sum, i.e. last_ele is x
    ll second_last_ele = b[n];
    if (sum - last_ele == 2 * second_last_ele) {
        for (int i = 0; i < n; i++) cout << b[i] << ' ';
        return;
    }

    // No possible case.
    cout << -1;
}

int main() {
    int T = 1;
    cin >> T;
    while (T--) {
        solve();
        cout << "\n";
    }
    return 0;
}
```

D: Maximum Good Energy

Preparation by: @siddhantnema
Editorial by: @siddhantnema

The logic behind this question comes under the category of **Number Theory/Brute Force**.

First we address the trivial case where the number initial energy does not contain any 0s nor 2 or 5 as a factor and the energy drink is less than 10 liters. This would mean we can't add any 0s to Ram's energy. So we output $N \cdot M$ as the answer.

Please refer to the code as you read this explanation

Now let us try to extend this logic to any general case.

Initialize a variable prod to 1.

Now keep finding a number that will add another 0 to our current number N.

This number will always be $2, 5, 10$ (trivially).

Let us call this number K. As long as $K \cdot prod \leq M$, we set $prod$ to $prod \cdot K$ and N to $N \cdot K$

At the end we multiply $M/prod$ (floored) to our energy N so as to get the maximal energy with the most number of zeros.

Code:

```
#include <bits/stdc++.h>
using namespace std;
#define siddhantnema ios_base::sync_with_stdio(0); cin.tie(0);
#define ll long long

ll findn0(ll x)
{
    while(x%10 == 0)
        x/=10;
    return x%10;
}

void solve()
{
    ll N,M;
    cin >> N >> M;

    ll prod = 1;
    while(true)
    {
        ll tmp = findn0(N);
        ll K = 10;
        for(ll i = 0; i<10;i++)
```

```
            if((i+1)*tmp  % 10 == 0)
            {
                K = i+1; break;
            }
        if(prod*K <= M)
        {
            prod *= K;
            N*=K;
        }
        else
            break;
    }
    N*= (M/prod);
    cout << N << '\n';

}

signed main()
{
    siddhantnema
    #ifndef ONLINE_JUDGE
        freopen("output.txt","w",stdout);
        freopen("input.txt","r",stdin);
    #endif
    ll t=1;
    cin>>t;
    for(ll _ = 0;_<t;_++)
    {
        // cout << "Test Case #" << _+1 << '\n';
        solve();
    }
}
```

E. Optimal Altar Placements

Preparation by: @accord
Editorial by: @accord

We do not really need to think much. Where there is magic - there is symmetry! And of course, since we aim to have the shortest paths from $(1, 1)$ -> $(4, n)$ and $(4, 1)$ -> $(1, n)$, be the same in number, we can easily accomplish this if we keep symmetry between the left and right, OR also if we keep symmetry between the top and bottom.

Why is symmetry the key to this problem? Because, if the paths $(1, 1)$ -> $(4, n)$ and $(4, 1)$ -> $(1, n)$ are the same under some symmetry, then clearly their shortest paths will also be the same!

Let us use top-bottom symmetry for now, as it is more natural for the question.

Consider the pattern of filling altars show below:

```
.........  .........  .........  .........   .........
.........  .#.......  .##......  .###.....   .####....
.........  .#.......  .##......  .###.....   .####....  and so on...
.........  .........  .........  .........   .........
```

for even numbers. Clearly, this maintains symmetry and will lead to an equal number of shortest paths. For odd numbers, we can implement this idea:

```
.........  .........  .........  .........   .........
....#....  .#..#..#.  .###.....  .####....   .#####...
.........  .........  .#.#.....  .#.##....   .#.###...  and so on...
.........  .........  .........  .........   .........
```

For odd $k > 3$, we can make a "dead-end" for the magic in our altars. No shortest path would go through a dead-end, so it preserves the number of shortest paths. For $k = 1$ and $k = 3$, we have special cases. We use left-right symmetry to solve those. (We leave it as an exercise to the reader to prove that those work.)

And of course, there are no cases where it is impossible to construct a shrine, because nothing is impossible with magic!

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
        int n, k; cin >> n >> k;
        vector<string> v(4);
        for(int i = 0; i < n; i++) {
                v[0] += "."; v[1] += "."; v[2] += "."; v[3] += ".";
        }
        cout << "YES\n";
        if(k==1) {
                v[1][n/2] = '#';
        }
        else if(k == 3) {
                v[1][n/2] = '#';
                v[1][1] = '#';
                v[1][n-2] = '#';
        }
        else {
                for(int i = 0; i < (k+1)/2; i++) {
                        v[1][i+1] = '#';
                        v[2][i+1] = '#';
                }
                if(k%2) v[2][2] = '.';
        }
        for(auto x : v) cout << x << "\n";
}
```

F: Steep Steep Slopes

Preparation by: @siddhantnema
Editorial by: @siddhantnema

The logic behind this question becomes very clear if the reader is familiar with **Dynamic Programming/Combinatorics**.

On any given day there can be 3 kinds of walks Feebo can take:

- Gentle to OAT and Steep Slope back to a hostel
- Gentle to OAT and Gentle Slope back to a hostel
- Steep to OAT and Gentle Slope back to a hostel

Now Feebo can take g_i / s_i number of paths from the i^{th} hostel to the OAT multiplied with the number of Steep/Gentle slopes (depending on which of the 3 cases you have chosen) with any hostel j ($1 \leq j \leq m$).

So for 1 day it is just the sum of all 3 of these cases over all m hostels.

Now, $n - 1$ days remain which can be solved by a simple recursive call (Please refer to the code as you read this explanation) . To further optimize this, we do not compute redundant calls and store each <hostel,days left> pair in a $2D$ Array or a map of pairs (in C++).

Code:

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll RMOD = 1000000007;

vector<vector<ll>> DP(1001,vector<ll>(1001,-1));
ll rec(ll hostel,vector<ll> &gentle_slopes,vector<ll> &steep_slopes,ll
days,vector<vector<ll>> &DP)
{
    if(DP[hostel][days]!= -1)
        return DP[hostel][days];


    ll N = gentle_slopes.size();
    if(days == 0)
        return 1;

    ll gentle_first = 0;
    for(ll i=0;i<N;i++)
        gentle_first += (gentle_slopes[hostel]* (gentle_slopes[i]+steep_slopes[i]) *
(rec(i,gentle_slopes,steep_slopes,days-1,DP)%RMOD))%RMOD;
```

```
    gentle_first %= RMOD;

    ll steep_first = 0;
    for(ll i=0;i<N;i++)
        steep_first += (steep_slopes[hostel]* (gentle_slopes[i]) *
(rec(i,gentle_slopes,steep_slopes,days-1,DP)%RMOD))%RMOD;
    steep_first %= RMOD;


    ll res = (gentle_first+steep_first)%RMOD;
    return DP[hostel][days] = res;
}

void solve()
{
    ll M,N;
    cin >> M >> N;
    vector<ll> gentle_slopes(M),steep_slopes(M);
        for(ll i = 0;i<M;i++)
                cin >> gentle_slopes[i];

        for(ll i = 0;i<M;i++)
                cin >> steep_slopes[i];

    cout << rec(0,gentle_slopes,steep_slopes,N,DP) << '\n';

}

signed main()
{
    solve();
}
```