

University of São Paulo
Institute of Mathematics and Statistics

VIII USP Freshman Try-outs
Editorial

Problem	Author
A - Abducting Nathan!	Lucas Harada
B - Best University ID	Carlos Ferreira and Gabriel Morete
C - CCM	Thiago Oliveira
D - Diary of Happiness	Thiago Oliveira
E - El Clasificador	Nathan Luiz
F - Food Queue	Lucas Harada
G - Grand Meeting	Felipe Noronha
H - Harada Football Club	Thiago Oliveira
I - Irritating Carlinhos	Gabriel Morete
J - Journey through time	Lucas Harada

Table 1: Contest problems.

A. Abducting Nathan!.

Tags : Mathematics.

Author : Gabriel Morete.

First, one should notice that the complete rounds do not matter, i.e., each sequence of $2k$ points. Let $P := N + T$ be the total number of points. We may eliminate the complete rounds by taking the remainder of the division of P by $2k$. Let $P' := P \pmod{2k}$. There are two options. If $P' < k$ then they are playing the first k points of the rounds, hence Thiago is serving. Otherwise, it is Nathan's Turn. We have the following C++ code.

```
1 #include "bits/stdc++.h"
2 using namespace std;
3
4 signed main(){
5     int T, k, t, n;
6
7     cin>>T;
8
9     while (T--){
10         cin>>k>>t>>n;
11         int p = (t + n) % (2*k);
```

```
12         if (p < k)
13             cout<<"Thiago"<<endl;
14         else
15             cout<<"Nathan"<<endl;
16     }
17 }
```

B. Best University ID.

Tags : Mathematics, number theory.

Author : Gabriel Morete.

Given a list of numbers one need to find the one with the largest prime factor. This is a standard task in number theory with multiple ways of doing it. However, this problem is a bit tight and some solutions may not fit the time limit. Here we describe a efficient solution to the problem.

The first observation is that if a positive integer x have a prime factor larger than $\sqrt{x}$ than x is prime and the only prime factor is the number itself (try to prove it yourself! Hint : contradiction). Hence, since no input number is larger than 10^8 , we will preprocess for each number up to 10^4 if it is prime or not (in the simplest way possible) and add each prime to a list.

Now, for each input number, we will run trough the list of primes and check for each prime if it divides the current number and divide the number until it is not a factor anymore. We will save the largest prime that divided the number. If after this process the remaining number is larger than one, than the remaining number is itself a prime and must be considered in the answer. Since there at most $N/\log(N)$ primes up to N (see prime number theorem) the algorithm is $O(N + nN/\log(N))$ which is fast enough. We have the following C++ code.

```
1  vector<int> primes;
2
3  int higest_prime(int val){
4      int ans = 1;
5
6      for (auto x : primes){
7          while ((val % x) == 0){ // x is a prime factor of val
8              ans = max(ans , x);
9              val /= x;
10         }
11     }
12
13     // since val <= 10^8, it is garanteed that val is 1 or a prime
14     return max(val , ans);
15 }
16
17 bool is_prime(int val){
18     for (int i = 2; i < val; i++)
19         if ((val % i) == 0)
20             return false;
21     return true;
22 }
23
24 signed main(){
25     for (int i = 2; i < 10010; i++)
26         if (is_prime(i))
27             primes.push_back(i);
28 }
```

```

29     int n;
30     cin>>n;
31
32     int pans = -1; // Higest prime in the answer
33     string ans = "-1"; // Answer
34
35     for (int i = 0; i < n; i++){
36         string name;
37         int val;
38         cin>>name>>val;
39
40         int hp = highest_prime(val);
41         if (hp > pans){
42             pans = hp;
43             ans = name;
44         }
45     }
46
47     cout<<ans<<endl;
48 }

```

C. CCM.

Tags : Mathematics, Geometry, Binary search.

Autor : Thiago Oliveira.

We describe two ways to solve this question. The first one involves only mathematical reasoning and leads to a closed formula, the second is a solution with binary search.

Let r_{cr}, r_{ca}, r_{ma} be the radius of the balloon of Cris, Carlinhos and Marcel, respectively, at the end of the process. We want that $r_{ca} \geq r_{cr}$ and $r_{ca} \geq r_{ma}$. For the first condition is sufficient and necessary that $V_{ca} \geq V_{cr}$, the second condition is harder.

Let us first compute r_{ca} . Cris' and Carlinhos' balloon will meet at time

$$T = \frac{X}{V_{ca} + V_{cr}}. \quad (1)$$

Hence,

$$r_{ca} = V_{ca} * T = \frac{V_{ca}X}{V_{ca} + V_{cr}}. \quad (2)$$

Since Carlinhos' balloon is larger than Cris' balloon, and Marcel is at the same distance from Cris and Carlinhos, he will blow up his balloon until it meets Carlinhos' balloon (note that Marcel's speed does not matter). If

$$D = \sqrt{(X/2)^2 + Y^2}$$

is the distance from Carlinhos to Marcel, then Marcel's balloon will finish with radius $D - r_{ca}$. Therefore, $r_{ca} \geq D - r_{ca}$ which implies

$$2r_{ca} \geq D. \quad (3)$$

We can combine 2 and 3 to obtain

$$V_{ca} \geq \frac{DV_{cr}}{2X - D}.$$

We conclude that Carlinhos' speed should be the maximum between V_{cr} and $\frac{DV_{cr}}{2X - D}$. The following C++ code implements this solution

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     long long int x, y;
6     double vcr;
7     double D;
8
9     cin >> x >> y;
10    double xx = x; double yy = y;
11
12    D = sqrt((xx/2)*(xx/2) + yy*yy);
13
14    cin >> vcr;
15
16    cout << setprecision(15) << fixed << max(vcr, D*vcr/(2*xx - D)) << endl;
17 }
```

We recommend understanding the first solution before moving on. The second solution is to binary search what is the minimum possible speed for Carlinhos. Note that if Carlinhos achieves his goal with a speed X , then he will surely achieve his goal with a speed greater than X .

We can start with Cris' speed as a lower bound for the solution, and with 10^{18} as an upper bound. To verify if a speed is valid we can first compute the time T when Carlinhos will stop blowing up his balloon, using equation 1. After that we can compute r_{ca} using 2. A speed is valid if and only if r_{ca} is at least half of the distance D , which is equivalent to 3. The following C++ code implements this solution:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  long long int x, y;
5  double vcr;
6  double D;
7
8  bool test(double v){
9      double t = x/(v + vcr);
10
11      if(t*v >= D/2) return true;
12      return false;
13 }
14
15 int main(){
16     cin >> x >> y;
17     double xx = x; double yy = y;
18
19     D = sqrt((xx/2)*(xx/2) + yy*yy);
20
21     cin >> vcr;
22     double l = vcr; double r = 1e18;
23     for(int i = 0; i <= 200; i++){
24         double m = (l + r)/2;
25         if(test(m))
26             r = m;
27         else
28             l = m;
29     }
30     cout << setprecision(15) << fixed << (l + r)/2 << endl;
31 }
```

D. Diary of Hapiness.

Tags : Mathematics.

Autor : Matheus Paulo.

In this question, we need to find the average of the grades and then, decide if its positive, negative or equal to 0. Because n is always positive, the average of the grades has the same sign of their sum, so we can only consider the sum of the grades, working only with integers. Because grades are small, we can store their value in an integer variable, and in the end, print the corresponding answer, depending on its sign. We have the following C++ code.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int n; cin >> n;
6     int s = 0;
7     for(int i=0;i<n;i++){
8         int num; cin >> num;
9         s += num;
10    }
11    if(s > 0) cout << ":\n";
12    else if(s < 0) cout << ":(\n";
13    else cout << ":\n";
14 }
```

E. El Classifier.

Tags : Data Structures.

Autor : Gabriel Caiaffa.

In this question, we have a direct application of a data structure map (which is implemented as a Red Black Tree in C++). The operation described at the beginning of the questions can be done using the `lower_bound` (follow here a link with examples of how to use `lower_bound` in a map: [here](#)). We must add all sneakers and their quantities to the map. As we serve customers, we need update the quantities of sneakers that we sell and if the quantity of sneakers goes to 0, we need to remove it from the map. Follow here an implementation of the solution in C++:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define fastio ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0)
4
5 int main(){
6     fastio;
7     int n,q; cin>>n>>q;
8     int sneakers[n];
9     map<int , int> inventory;
10    int sz;
11    for(int i = 0; i<n; i++){
12        cin >> sz;
13        inventory[sz]++;
14    }
15    int min_client;
16    for(int i = 0; i<q;i++){
17        cin>> min_client;
18        auto desire = inventory.lower_bound(min_client);
19        if(desire == inventory.end())
20            cout<< -1 <<endl;
21        else if( desire->second== 1){
22            cout<< desire->first << endl;
23            inventory.erase( desire->first );
24        }
25        else{
26            cout<< desire->first << endl;
27            inventory[ desire->first ]--;
28        }
29    }
30 }
```

F. Food Queue.

Tags : Greedy.

Autor : João Guilherme.

That's a basic example of a scheduling problem. Since we need the maximum number of students in each cafeteria and the order doesn't matter, realize that the best option is always choose the person that takes less time eating. So, we need storage the times of each cafeteria separately and sort these vectors. After that, we need check how many people we can get, choosing the person with the shortest time at that moment. We will make the cumulative sum of each vector, and count until the total sum of times is smaller than or equal to T . After get this done for the four cafeterias, the sum of these results will be the answer. We have the following C++ code.

```
1  #include "bits/stdc++.h"
2  using namespace std;
3
4  signed main(){
5      long long n, t;
6      cin>>n>>t;
7
8      vector<long long> queues[4]; // C, F, P, Q
9
10     for (int i = 0; i < n; i++){
11         char b; int v;
12         cin >> b >> v;
13         if (b == 'C') queues[0].push_back(v);
14         else if (b == 'F') queues[1].push_back(v);
15         else if (b == 'P') queues[2].push_back(v);
16         else if (b == 'Q') queues[3].push_back(v);
17     }
18
19     int ans = 0;
20
21     for (int i = 0; i < 4; i++){
22         sort(queues[i].begin(), queues[i].end());
23
24         int aux = 0;
25         for (auto x : queues[i])
26             if (aux + x <= t){
27                 ans++;
28                 aux += x;
29             }
30     }
31
32     cout<<ans<<endl;
33 }
```

G. Grand Meeting.

Tags: Mathematics.

Author: Willian Mori.

Let i and j be the stations at which Cássio and Morete embark. A minimal time needed for them to meet is achieved they meet in station $\left\lfloor \frac{i+j}{2} \right\rfloor$, taking $\left\lceil \frac{|i-j|}{2} \right\rceil$ minutes. All that remains is discovering which stations Cássio and Morete embark. This can be done by iterating over s and finding corresponding stations for Cássio and Morete:

```
1 // returns Cassio and Morete's respective stations
2 pair<int, int> get_stations(int n, string s[], string c, string m) {
3     pair<int, int> result;
4     for(int i=1; i<=n; i++) {
5         if(s[i] == c) result.first = i;
6         if(s[i] == m) result.second = i;
7     }
8     return result;
9 }
```

H. Harada Football Club.

Tags : Mathematics.

Author : Nathan Luiz.

We are going to describe two solutions for this problem. The first important observation is that we need at least one player in each of the four positions, and exactly one goalkeeper. Then, the problem is equivalent to: how many ways do we have to distribute $N - 4$ players in 3 positions (defense, midfield or attack)? It is certainly a simpler version of the problem, so let's work with it. Suppose we have x defense players, y midfield players and z attack players. Therefore:

$$x + y + z = N - 4.$$

Now I am going to describe the first solution. Let's fix the value of x . Then, $y + z = N - 4 - x$. As y, z are non-negative integers, the number of possible solutions for this equation is $N - 3 - x$. So, for each possible $0 \leq x \leq N - 4$, we can add $N - 3 - x$ to the answer, leading to a $O(N)$ solution. It is also possible to find the following closed formula in $O(1)$:

$$(N - 3) \cdot (N - 3) - \frac{(N - 4) \cdot (N - 3)}{2}.$$

The second solution is using [stars and bars](#). We have an equation with 3 variables in which each variable belongs to $[0, N - 4]$ and the sum of the variables is $N - 4$. So, the answer is:

$$\binom{(N - 4) + 3 - 1}{2}.$$

An implementation of the first method is below:

```
1 int ans = 0;
2 for (int x = 0; x <= n - 4; x++) {
3     ans += n - 3 - x;
4 }
5 cout << ans << "\n";
```

I. Irritating Carlinhos.

Tags: Implementation, Simulation.

Author: Enrique Junchaya.

Let n be the length of the strings that represent the movements. We want to simulate, for Thiago and Carlinhos, the resulting positions after each of the n moves that they do. We have to start that with Thiago's first move since he always moves first. After that, we must continue with Carlinhos's first move, then with Thiago's second move and so on. Notice that after each of the moves we must check that their positions are not the same because, if they were, then we know that Thiago was caught and we should answer "Rodou!" and stop the program. The code of this part of the program would be like this

```
1 for(int i = 0; i < n; i++) {
2     simulate(tx, ty, s[i]);
3     check();
4     simulate(cx, cy, t[i]);
5     check();
6 }
7 cout << "Quase!" << endl;
```

where `simulate(x, y, c)` modifies positions x and y according to the character c and `check()` checks if Thiago's and Carlinhos's positions are not the same and, if they were, prints "Rodou!" and stops the program.

J. Journey through time.

Tags: Implementation, Simulation, Dynamic Programming.

Author: Antonio Hachisuca.

Note that if $1 < t \leq n$ is the current moment, then independently of the operation made in t the answer of operations 2,3 and 4 for every moment $1 \leq t' < t$ remains the same, because no operations alters the "past". Let $res_{t,2}$, $res_{t,3}$ and $res_{t,4}$ be the results for operations 2,3 and 4 at moment t , respectively. Thus, it is possible to save the results of the three operations in $res_{t,2}$, $res_{t,3}$ and $res_{t,4}$ in the t -th iteration. For the first operation, we have:

1. $res_{1,2} = \max\{x, -\infty\} = x;$
2. $res_{1,3} = \min\{x, \infty\} = x;$
3. $res_{1,4} = x.$

For the rest, if the i -th operation is of type 1, then

1. $res_{i,2} = \max\{x, res_{i-1,2}\};$
2. $res_{i,3} = \min\{x, res_{i-1,3}\};$
3. $res_{i,4} = x + res_{i-1,4}.$

Else,

1. $res_{i,3} = res_{i-1,2};$
2. $res_{i,3} = res_{i-1,3};$
3. $res_{i,3} = res_{i-1,4}.$

It is possible to save all results in an array and the final complexity is $O(n)$, since there are n operations and every operation is solved in $O(1)$. Therefore, we have the following C++ solution:

```
1 #include <iostream>
2
3 using namespace std;
4
5 const int INF = 1e9+7;
6
7 long long res[100001][5];
8
9 signed main(){
10     int n; cin>>n;
11
12     res[0][2] = -INF;
13     res[0][3] = INF;
14     res[0][4] = 0;
15
```

```

16     for (int i=1; i<=n; i++){
17         int op; cin>>op;
18
19         if (op == 1){
20             long long x; cin>>x;
21
22             res[i][2] = max(res[i-1][2], x);
23             res[i][3] = min(res[i-1][3], x);
24             res[i][4] = res[i-1][4] + x;
25         }
26         else{
27             int t; cin>>t;
28
29             res[i][2] = res[i-1][2];
30             res[i][3] = res[i-1][3];
31             res[i][4] = res[i-1][4];
32
33             cout<<res[t][op]<<endl;
34         }
35     }
36
37     return 0;
38 }

```