

Universidade de São Paulo

Instituto de Matemática e Estatística

VIII Simulado dos Bixos

Editorial

Problem	Author
A - Ajude o Nathan	Lucas Harada
B - Buscando Primos	Carlos Ferreira e Gabriel Morete
C - CCM	Thiago Oliveira
D - Dias Felizes	Thiago Oliveira
E - El Clasificador	Nathan Luiz
F - Fila do Bandeco	Lucas Harada
G - Grande Encontro	Felipe Noronha
H - Harada Futebol Clube	Thiago Oliveira
I - Irritando Carlinhos	Gabriel Morete
J - Jogo do Noronha	Lucas Harada

Table 1: Problemas da prova.

A. Ajude o Nathan.

Tags : Matemática, Aritmética modular.

Autor : Gabriel Morete.

Primeiramente, basta notar que as rodadas completas não importam. Seja $P = N + T$ o total de pontos, podemos eliminar as rodadas completas tirando o resto de divisão por $2k$. Seja $P' := P \pmod{2k}$ temos duas opções. Se $P' < k$ então Thiago está sacando. Caso contrário Nathan está sacando. Temos o seguinte código em C++.

```
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. Buscando Primos.

Tags : Matemática, teoria dos números.

Author : Gabriel Morete.

Dada uma lista de números queremos encontrar o que tem o maior fator primo. Esse é um exercício bem comum de teoria dos números e há múltiplas formas de fazê-lo. Contudo, esse problema é um pouco apertado e algumas soluções podem não passar no tempo limite. Descreveremos uma solução eficiente para o problema.

A observação principal é que se um inteiro positivo x tem um fator primo maior que $\sqrt{x}$ então x é primo e o fator é o próprio número (tente provar esse fato! Dica : contradição). Como nenhum número é maior que 10^8 podemos pré-processar todos os primos até 10^4 e colocá-los em uma lista.

Agora, para cada número da entrada, vamos percorrer a lista e testar para cada primo se o mesmo divide o número e dividir o mesmo enquanto o primo for um fator. Nós vamos guardar o maior fator primo encontrado nesse processo. Se após isso o número resultante for maior que um o mesmo é primo e deve ser considerado na resposta. Como há no máximo $N/\log(N)$ primos até N (veja *prime number theorem*) o algoritmo consome tempo $O(N + nN/\log(N))$, o que é rápido o suficiente. Temos o seguinte código em C++.

```
1  vector<int> primes;
2
3  int highest_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 guaranteed 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 : Matemática, Geometria, Busca Binária.

Autor : Thiago Oliveira.

Nós descrevemos duas maneiras de resolver a questão. A primeira envolve somente raciocínio matemático para se chegar em uma fórmula fechada, e a segunda solução é com busca binária.

Seja r_{cr}, r_{ca}, r_{ma} os raios dos balões de Cris, Carlinhos e Marcel, respectivamente, ao final do processo. Queremos que $r_{ca} \geq r_{cr}$ e $r_{ca} \geq r_{ma}$. Para a primeira condição é necessário e suficiente que $V_{ca} \geq V_{cr}$, a segunda condição é mais difícil.

Primeiro vamos computar r_{ca} . Os balões de Cris e Carlinhos irão se encontrar no tempo

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

Então,

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

Como o balão de Carlinhos é maior que o de Cris, e Marcel está a mesma distância dos dois outros professores, ele vai encher seu balão até que este encontre o de Carlinhos (note que a velocidade de Marcel não importa). Se

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

é a distância entre Carlinhos e Marcel, então o balão de Marcel terminará com raio $D - r_{ca}$. Portanto, $r_{ca} \geq D - r_{ca}$ o que implica

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

Podemos combinar 2 e 3 para obter

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

Concluimos que a velocidade de Carlinhos deve ser o máximo entre V_{cr} e $\frac{DV_{cr}}{2X - D}$. O seguinte código em C++ implementa esta solução:

```
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 }
```

Recomendamos entender a primeira solução antes de continuar. A segunda solução é uma busca binária na velocidade mínima que Carlinhos pode encher seu balão. Note que se Carlinhos atinge seu objetivo com velocidade X , ele certamente irá atingir com uma velocidade superior à X .

Podemos começar com a velocidade da Cris como uma cota inferior para a resposta e com 10^{18} como uma cota superior. Para verificar que uma velocidade é válida podemos primeiro computar o tempo T em que Carlinhos para de encher o balão, usando a equação 1. Depois disso podemos computar r_{ca} usando 2. Por último, uma velocidade é válida se e somente se r_{ca} é pelo menos metade da distância D , o que é equivalente à 3. O seguinte código em C++ implementa esta solução:

```
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. Dias Felizes.

Tags : Matemática.

Autor : Matheus Paulo.

Nessa questão, devemos calcular a média das notas e depois decidir se é maior, menor ou igual a 0. Como o valor de n é sempre positivo, então a média tem o mesmo sinal da soma das notas, então podemos apenas utilizar a soma das notas e não nos preocupar com números decimais. Como as notas são pequenas, podemos guardar a soma das notas em uma variável de tipo inteiro, e no final, imprimir a resposta correspondente, baseada no seu sinal. Temos o seguinte código em C++.

```
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 Classificador.

Tags : Estrutura de Dados.

Autor : Gabriel Caiaffa.

Nessa questão, temos uma aplicação direta da estrutura de dados map (que no C++ é implementada como uma Árvore Rubo Negra). A operação descrita no início do enunciado pode ser feita usando o `lower_bound` (segue aqui um link com exemplos de como usar o `lower_bound` em map: [aqui](#)). Devemos adicionar todos os tamanhos de tenis no map e suas quantidades também. A medida que vamos atendendo os clientes, devemos ir retirando da quantidade de tenis daquele tamanho no estoque. Caso nao tenhamos mais o tenis no estoque, ele deve ser retirado do map. Segue abaixo uma implementação em C++ do problema:

```
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. Fila do bandeco.

Tags : Guloso.

Autor : João Guilherme.

Esse é um exemplo básico de problema de escalonamento. Como precisamos da maior quantidade de pessoas possível e a ordem da entrada das pessoas não importa, percebe-se que o melhor é sempre escolher pessoas com tempo pequeno. Assim, precisamos guardar os tempos de cada bandeirão de maneira separada e ordenar esses vetores. Após isso, precisaremos olhar quantas pessoas conseguimos encaixar escolhendo sempre a pessoa com menor tempo. Para fazer isso, basta fazer uma soma de prefixo enquanto a mesma é menor que t . O número de elementos somados é o máximo de pessoas que podem comer nesse bandeirão. Após isso, basta somar a resposta de cada bandeirão. Temos o seguinte código em C++.

```
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 }
34 }
```

G. Grande Encontro.

Tags: Matemática.

Author: Willian Mori.

Sejam i e j as estações em que Cássio e Morete embarcam. Um tempo mínimo necessário para eles se encontrarem é realizado quando os dois se encontram na estação $\left\lfloor \frac{i+j}{2} \right\rfloor$, levando $\left\lceil \frac{|i-j|}{2} \right\rceil$ minutos. Resta descobrir em que estações Cássio e Morete embarcam. Isso pode ser feito iterando sobre s e encontrando as estações correspondentes de Cássio e Morete:

```
1 // retorna as respectivas estacoes de Cassio e Morete
2 pair<int, int> encontra_estacoes(int n, string s[], string c, string m) {
3     pair<int, int> resultado;
4     for(int i=1; i<=n; i++) {
5         if(s[i] == c) resultado.first = i;
6         if(s[i] == m) resultado.second = i;
7     }
8     return resultado;
9 }
```

H. Harada Futebol Clube.

Tags : Matemática.

Author : Nathan Luiz.

Nós vamos descrever duas soluções para este problema. A primeira observação importante é que precisamos de pelo menos um jogador em cada uma das quatro posições e exatamente um goleiro. Ou seja, o problema é equivalente a: quantas maneiras temos de distribuir $N - 4$ jogadores em 3 posições (defesa, meio-campo, ataque)? Esta com certeza é uma versão mais simples do problema, então vamos trabalhar nela. Suponha que temos x jogadores na defesa, y jogadores meio-campo e z atacantes. Então:

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

Agora vou descrever a primeira solução. Vamos fixar o valor de x . Então, $y + z = N - 4 - x$. Como y, z são inteiros não negativos, então o número de solução desta equação é $N - 3 - x$. Então, para cada possível $0 \leq x \leq N - 4$, nós podemos adicionar $N - 3 - x$ na resposta, sendo uma solução em $O(N)$. Também é possível chegar na fórmula fechada:

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

A segunda solução utiliza [stars and bars](#). Nós temos uma equação com 3 variáveis em que cada variável pertence a $[0, N - 4]$ e a soma das variáveis é $N - 4$. Ou seja, a resposta é:

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

Uma implementação do primeiro método abaixo:

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

I. Irritando o Carlinhos.

Tags: Implementação, Simulação.

Author: Enrique Junchaya.

Seja n o tamanho das *strings* que representam os movimentos. Queremos simular, para Thiago e Carlinhos, as posições que resultam após de cada um dos n movimentos que eles fazem. Devemos fazer isso começando com o primeiro movimento do Thiago já que ele sempre se movimenta primeiro. Depois seguimos com o primeiro movimento do Carlinhos, logo com o segundo do Thiago e assim por diante. Note que após cada um dos movimentos devemos verificar que a posição deles não seja a mesma porque, se for, então sabemos que Thiago já foi pego e devemos responder "Rodou!" e parar o programa. O código dessa parte do programa ficaria assim

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

onde `simula(x, y, c)` modifica as posições x e y segundo o indicado pelo caráter c e `verifica()` verifica se as posições de Thiago e Carlinhos são iguais e, se são iguais, imprime "Rodou!" e para o programa.

J. Jogo do Noronha.

Tags: Implementação, Simulação, Programação Dinâmica.

Autor: Antonio Hachisuca.

Note que se $1 < t \leq n$ é o momento atual, então independentemente da operação feita em t o resultado das operações 2,3 e 4 para qualquer momento $1 \leq t' < t$ não muda, pois nenhuma operação altera o "passado". Sejam $res_{t,2}$, $res_{t,3}$ e $res_{t,4}$ os resultados das operações 2,3 e 4 no momento t , respectivamente. Logo, é possível salvar os resultados das três operações em $res_{t,2}$, $res_{t,3}$ e $res_{t,4}$ na t -ésima iteração. Para a primeira operação, temos:

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

Para as restantes, se a i -ésima operação for do tipo 1, então

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

Caso contrário,

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

Como todo $i \leq n \leq 10^5$, então é possível salvar os resultados em um vetor e a complexidade final é $O(n)$, pois são n operações e cada operação é resolvida em $O(1)$. Segue uma solução:

```
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 }

```