

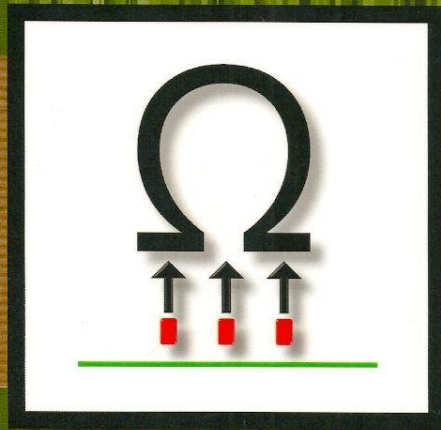
Introdução

Prof. **João Paulo** R. R. Leite
joapaulo@unifei.edu.br

ECOE44 & ECOE45 – Maratona de Programação

Competitive Programming 3

The New Lower Bound of Programming Contests.



Steven Halim
Felix Halim

HANDBOOK FOR ACM ICPC AND IOI CONTESTANTS
2013

Competitive Programming

Steven & Felix Halim

National University of Singapore

<https://cpbook.net/>

URI ONLINE JUDGE



O URI Online Judge é um projeto que está sendo desenvolvido pelo Departamento de Ciência da Computação da URI. O principal objetivo é promover a prática de programação e o compartilhamento de conhecimento.

CRÉDITOS

REPOSITÓRIO DE PROBLEMAS

URI

ONLINE JUDGE

PROBLEMS & CONTESTS

ENTRAR

EMAIL

SENHA

☐ LEMBRAR-ME (7 DIAS)

ENTRAR

COMPETIÇÕES E RANKING

Resolva os problemas disponíveis utilizando as 11 linguagens de programação, competindo com os outros usuários. Como desafio, melhore seu ranking, resolvendo o máximo de problemas e aperfeiçoando seu código fonte.

CONFIRA O RANKING



URI ONLINE JUDGE ACADEMIC

www.urionlinejudge.com.br

URI Online Judge: Problems & Contests
Tarefas, Exercícios e Treinamento.

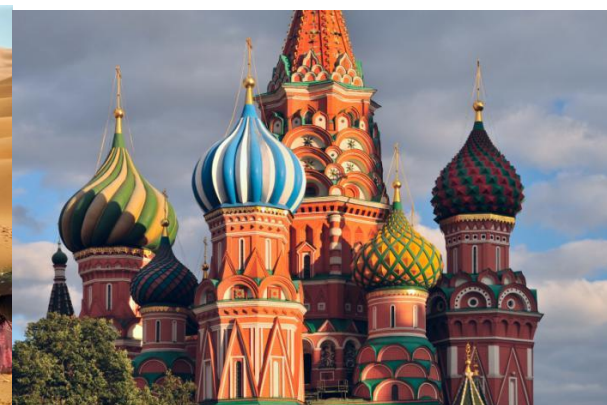
Pré-Requisitos

- A linguagem de programação básica do curso será o C++.
 - É preciso ter conhecimento sobre C++ para solução de exercícios em competição e tarefas para casa.
 - Aprendido em disciplinas de programação básica, estruturas de dados e POO.
 - **Este NÃO será um curso básico!**
- Conhecimento básico em análise de algoritmos é desejável.
- É necessário conhecer STL.
 - Explicarei brevemente sobre a *Standard Template Library*, mas preciso que você estude bastante sozinho!
- A disciplina é para alunos que gostam de programação e desejam competir ou melhorar suas habilidades.

Oportunidade de representar a UNIFEI nas Maratonas de Programação



Ou em uma final mundial da ACM-ICPC!



E... certamente, conseguir um belo emprego!



Dado um problema de competição, nós queremos:

- Resolvê-lo com **eficiência** (baixo custo computacional)...
- Utilizando **algoritmos** conhecidos e **estruturas de dados**.
- Sermos capazes de traduzir nossa solução em uma **linguagem de programação**;
- Fazê-lo o mais **rápido** possível;
- E **corretamente**!

Este curso te ajudará com esse processo!

- Não estamos falando de **engenharia de software** por aqui, ou de excelentes práticas de programação.
- Também não resolveremos **problemas de pesquisa**, com solução ainda indefinida.
 - Todos os problemas de competições foram formulados por alguém (às vezes uma pessoa também com problemas! Hehe) e já foram resolvidos antes. Precisamos apenas **identificar as técnicas** e **pensar fora da caixa** (mas que clichê!).

E como faremos isso?

- Estudando algumas **classes típicas** de problemas.
- **Aprendendo novas técnicas** e algoritmos;
- Identificando **cenários apropriados** para cada algoritmo e estrutura de dados que estudarmos (ou que já conhecíamos previamente).
- Aprendendo muito com nossos **programadores/competidores mais experientes**.
- **Praticando** a solução de problemas **autonomamente**:
 - URI Online Judge (www.urionlinejudge.com.br)
 - ICPC Live Archive (icpcarchive.ecs.baylor.edu/)
 - CodeForces (codeforces.com)
 - Uva (uva.onlinejudge.org)
 - SPOJ (br.spoj.com)

E também:

- Praticando.
- Praticando.
- E praticando mais um pouco.

Os Problemas

Um problema de competição é normalmente composto por quatro partes:

1. A **descrição do problema**, que varia bastante em complexidade e pode tanto pedir alguma coisa diretamente quanto contar uma bela história sobre o assunto (e, geralmente, desviar sua atenção do objetivo).
2. A **descrição da Entrada**, que é importantíssima e deve ser respeitada. Nessa seção temos noção da dimensão dos dados e começamos a fazer estimativas para verificar se nossa solução resolve o problema em um tempo razoável.
3. A **descrição da saída**, que contém a maneira exata como sua resposta deve ser formatada. De crucial importância, uma vez que os juízes irão rejeitar uma resposta que, mesmo correta, esteja mal formatada.
4. **Exemplos de entrada e saída** vem por último, facilitando testes do programa desenvolvido.

Algumas vezes teremos também informação sobre o **limite de tempo**. Fique esperto no enunciado.

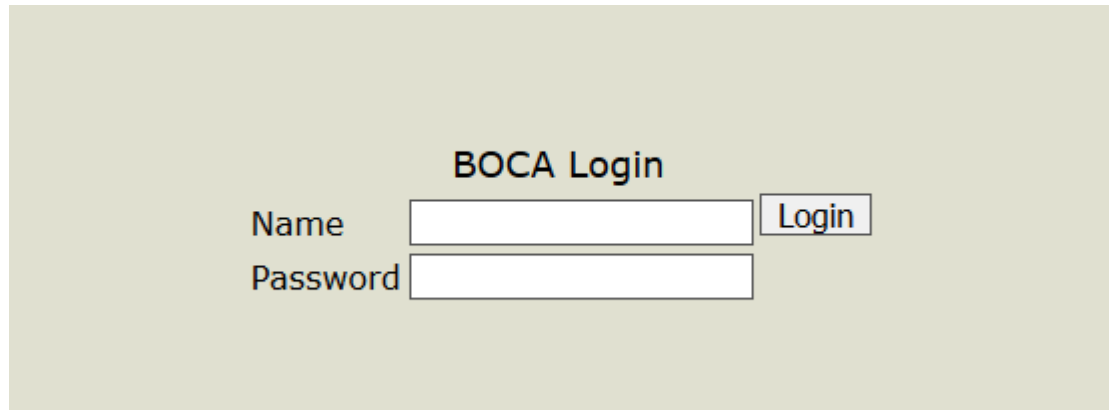
O Julgamento

Seu código é enviado para os juízes e pode receber as seguintes respostas:

- **Accepted:** Passou e foi aceito. Todos os casos de teste utilizados em seu programa deram respostas idênticas ao gabarito dos juízes. Há muitos outros casos de teste além dos exemplos da folha de prova. Tenha consciência disso.
- **Wrong Answer:** Resposta errada. Há um ou mais casos de teste (geralmente mais), que não obtiveram a resposta desejada (de acordo com o gabarito).
- **Compile Error:** O sistema de julgamento não conseguiu nem compilar seu programa, que está com erro de sintaxe. Não façam isso, pfvr.
- **Runtime Error:** Erro em tempo de execução. Acontece normalmente quando entramos em um loop infinito, acessamos vetor fora dos limites e variáveis não alocadas, etc. Erro em tempo de execução.
- **Time Limit Exceeded:** Todos os problemas devem executar por um período aceitável de tempo que, normalmente, está na casa de um ou poucos segundos.

Nossa Plataforma

boca.unifei.edu.br

A screenshot of a web login form titled "BOCA Login". The form is set against a light beige background. It contains two input fields: one for "Name" and one for "Password". To the right of the "Name" field is a button labeled "Login".

BOCA Login

Name

Password

Login

Hoje trabalharemos em **duplas**. Explore bem a plataforma e revezem-se na frente do computador. Muitas atividades durante o semestre serão individuais. Observação: Mesmo que haja computadores vagos, HOJE não será permitido à dupla utilizar mais de um PC para codificar.

Exemplo: Multiplicando

Descrição do Problema

Escreva um programa que multiplique pares de inteiros.

Descrição da Entrada

A entrada começa com uma linha contendo um inteiro T , onde $1 \leq T \leq 100$, representando o número de casos de teste. Seguem T linhas, cada uma contendo um caso de teste. Cada caso de teste consiste em dois inteiros, A e B , onde $-2^{20} \leq A, B \leq 2^{20}$, separados por um espaço.

Descrição da Saída

Para cada caso de teste, escreva na saída uma única linha contendo o valor de A multiplicado por B .

Exemplo de Entrada	Exemplo de Saída
4	12
3 4	0
13 0	8
1 8	10000
100 100	

```
1  #include <stdio>
2
3  int main()
4  {
5      int t, a, b;
6
7      scanf("%d", &t);
8
9      for(int i = 0; i < t; i++)
10     {
11         scanf("%d %d", &a, &b);
12         printf("%d\n", a*b);
13     }
14
15     return 0;
16 }
```

A solução está correta?

Vamos realizar um teste:

Façamos **$A = B = 2^{20} = 1048576$**

Quanto deu? 0? Errado! É preciso dar 2^{40} , ou 1099511627776.

Entrada **grande demais** para caber em um inteiro de 32-bits.

Precisamos de um **long long**, de 64-bits.

São essas **condições de contorno** que precisamos testar antes de enviar o problema. É nisso que, muitas vezes, a pessoa que elaborou o problema estava com a cabeça. Também podemos chamá-las de **condições de limite**.

Não tomemos WA por falta de atenção.

Fique de olho nos limites!

```
1  #include <stdio>
2
3  int main()
4  {
5      int t;
6      long long a, b;
7
8      scanf("%d", &t);
9
10     for(int i = 0; i < t; i++)
11     {
12         scanf("%lld %lld", &a, &b);
13         printf("%lld\n", a*b);
14     }
15
16     return 0;
17 }
```

Agora sim!

Exercício: Sequência de Fitromacci

Descrição do Problema

Uma sequência de números é dada pela seguinte relação de recorrência: os k primeiros números são iguais a 1; o n ésimo ($n > k$) valor é determinado pela soma dos k elementos anteriores. Sua tarefa, neste exercício, será determinar o n ésimo termo de Fitromacci. Por exemplo, para $k=3$, e $n=7$, temos a sequência 1 1 1 3 5 9 17.... e portanto, o sétimo número, neste caso é 17.

Descrição da Entrada

A primeira linha da entrada contém um inteiro T ($1 \leq T \leq 1000$) indicando o número de instâncias. Cada instância é composta por uma linha contendo o inteiro k ($1 \leq k \leq 7$), seguido por um inteiro n ($1 \leq n \leq 40$).

Descrição da Saída

Para cada instância, imprima, em uma linha, o n ésimo termo da sequência.

Exemplo de Entrada	Exemplo de Saída
2	17
3 7	24097
7 20	

Dicas

Algumas coisas para se **ter em mente**,
buscando tornar-se um programador
cada vez **melhor** e **mais competitivo**.

Dica 1: Digite Rápido

Parece brincadeira, mas em alguns casos, quando a competição está muito acirrada, a diferença entre duas equipes pode ser de apenas alguns minutos.

Para saber como anda sua agilidade para digitação, visite o site <http://www.typingtest.com/>.

O Prof. João Paulo fez o teste e, em média, escreve **45 palavras** por minuto – um desempenho bom, mas que poderia ser ainda melhor. Uma pessoa é considerada rápida quando consegue digitar **mais de 50 palavras corretas** em um minuto.



Dica 2: Identifique os Problemas

Obviamente que a experiência trará muito benefício para este quesito. No entanto, há uma taxa de aparecimento de alguns tipos de questões clássicos em competições. São eles:

- Ad Hoc (2-3 problemas)
- Busca Completa (0-1 problema)
- Divisão e Conquista (0-1 problema)
- Algoritmos Gulosos (0-1 problema)
- Programação Dinâmica (2-3 problemas)
- Grafos (~2 problemas)
- Matemática (~2 problemas)
- Processamento de Strings (~1 problema)
- Geometria Computacional. (~1 problema)

Nas competições há
entre 10 e 12
problemas, em média.

Dica 2: Identifique os Problemas

Importante lembrar que a tabela anterior é apenas um indicativo, baseado no histórico das competições.

Muitos problemas mais difíceis podem ser classificados em mais de uma categoria. Por exemplo, o algoritmo de **Kruskal** pode ser utilizado para encontrar a árvore geradora mínima de um **grafo**. **Dijkstra** pode ser utilizado para encontrar o caminho mínimo de um vértice a todos os outros também de um **grafo**. Ambos são **algoritmos gulosos**.

O algoritmo de **Floyd-Warshall** encontra os caminhos mínimos entre todos os pares de vértices de um **grafo**. É implementado com **Programação Dinâmica**.

Dica 3: Faça Análise de Algoritmo

De acordo com as regras das competições, não é suficiente que nosso programa resolva o problema: é preciso que ele resolva dentro de um **limite de tempo**, estabelecido para cada problema individualmente.

Sempre pode existir mais de uma solução para cada problema. Mas como escolher a melhor? Devemos observar o intervalo dos dados de entrada e discutir com nossa equipe.

Ao final, implementamos o algoritmo **mais simples possível**, mas que tenha **maior probabilidade de funcionar corretamente**, dentro dos **requisitos de tempo**.

É importante observar o tamanho da entrada.

Imagine que o tamanho máximo de um dado de entrada é 10^5 . Caso seu algoritmo seja $O(n^2)$, a sua intuição dirá que 10^{10} parece ser um valor muito alto... Portanto, vale a pena gastar um tempo tentando **encontrar um algoritmo de ordem reduzida**, por exemplo, $O(n \lg n)$, reduzindo para 1.7×10^6 operações (alguém pode, enquanto isso, codificar o algoritmo mais simples, caso não encontrem outra solução. Tempo é fundamental!)

Agora, imaginemos um caso onde você e sua equipe criaram um algoritmo de ordem $O(n^4)$. Podemos, inicialmente, pensar que o algoritmo é terrível, mas, ao observar o dado de entrada, percebemos que ele é sempre < 50 . **Mande ver!** Se a entrada for pequena, algoritmos de ordem elevada não fazem tanta diferença.

Dica 3: Faça Análise de Algoritmo

Podemos assumir algo em torno de 10^9 operações por segundo, para fins de cálculo aproximado.

Além disso, $2^{10} \approx 10^3$.

Imagine que queiramos ordenar $n \leq 10^6$ inteiros, e o limite de tempo para o problema é de 3 segundos.

- Podemos utilizar um algoritmo trivial $O(n^2)$ como Insertion Sort?
- E se usássemos um algoritmo mais sofisticado como o Merge Sort, de $O(n \log n)$?

Dica 3: Faça Análise de Algoritmo

Sempre escolha a implementação mais simples possível que resolva o problema dentro dos limites de tempo.

n	Slowest Accepted Algorithm	Example
≤ 10	$O(n!), O(n^6)$	Enumerating a permutation
≤ 15	$O(2^n \times n^2)$	DP TSP
≤ 20	$O(2^n), O(n^5)$	DP + bitmask technique
≤ 50	$O(n^4)$	DP with 3 dimensions + $O(n)$ loop, choosing ${}_nC_k = 4$
$\leq 10^2$	$O(n^3)$	Floyd Warshall's
$\leq 10^3$	$O(n^2)$	Bubble/Selection/Insertion sort
$\leq 10^5$	$O(n \log_2 n)$	Merge sort, building a Segment tree
$\leq 10^6$	$O(n), O(\log_2 n), O(1)$	Usually, contest problems have $n \leq 10^6$ (to read input)

(Fonte: Reykjavík University, 2015)

Dica 4: Domine a Linguagem

Em nossa disciplina, daremos maior ênfase para **C e C++**. As competições costumam permitir várias outras, como Java e Python. Esperamos que, com nossa ajuda e **muito treino**, sua habilidade como programador dessas linguagens tenha uma grande melhoria.

Java também pode ser importante, e não deve ser deixado de lado.

Ex.: BigInteger, Processamento de Strings, GregorianCalendar.

Saiba como a palma de sua mão:

- C++ STL Template Library
- The Java Class Library



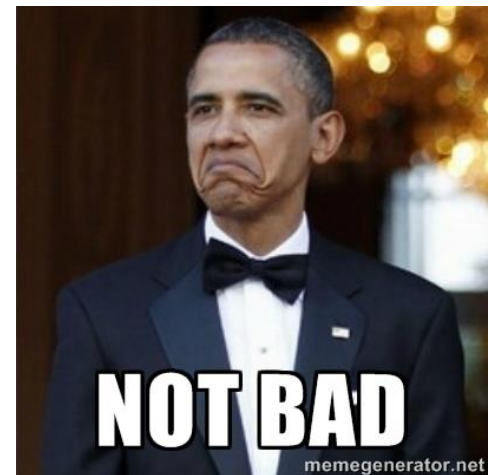
Dica 4: Domine a Linguagem

Imagine, por exemplo, que seja necessário escrever um programa que calcule o fatorial de 25. A resposta é 15.511.210.043.330.985.984.000.000 .

A resposta está MUITO acima do limite do inteiro primitivo **unsigned long long** que é $2^{64}-1 = 18.446.744.073.709.551.615$.

Veja a facilidade, utilizando a classe **BigInteger** de Java:

```
1  import java.util.Scanner;
2  import java.math.BigInteger;
3
4  class Main {
5
6      public static void main(String[] args)
7      {
8          BigInteger fat = BigInteger.ONE;
9
10         for(int i = 2; i <= 25; i++)
11             fat = fat.multiply(BigInteger.valueOf(i));
12         System.out.println(fat);
13     }
14 }
```



Dica 5: Teste seu Código

Os exemplos de entrada e saída exibidos na folha de prova são normalmente triviais e não representam condições de limite.

Teste seu código! Ninguém que ganhar 20 minutos de punição à toa. Pode significar **10 posições** no ranking final da competição.

Teste entradas grandes, valores limite, etc.

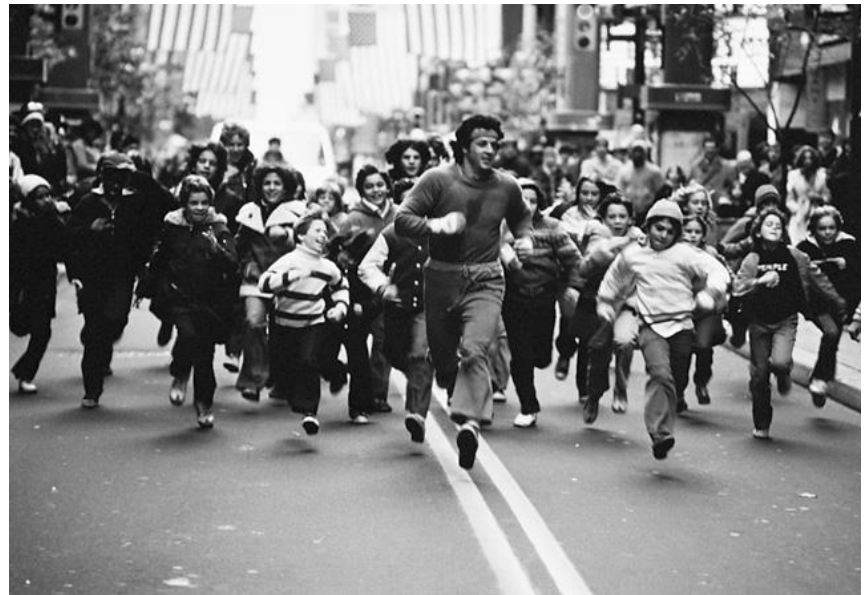
Tente encontrar contra exemplos para sua solução.

Seus testes deram certo? **SUBMIT!**

Dica 6: Treine, treine e treine mais

Como um atleta maratonista, é preciso **manter a sua forma**. Não deixe de resolver problemas sempre que possível. Reserve um tempo todos os dias. Utilize os juízes online que temos: URI, Uva, ICPC Live Archive, TopCoder, SPOJ, Codeforces e muitos outros.

Eu, como professor, ficarei satisfeito e orgulhoso se vocês me superarem como programadores de competição. Vá atrás de problemas cada vez mais difíceis e **compartilhe suas soluções conosco**.



Problemas Ad-Hoc

Significa “destinado a essa finalidade”.
É geralmente empregada sobretudo em contexto jurídico,
também no sentido de "**para um fim específico**".

- É o **tipo mais simples** de problema;
- É apenas necessário fazer o que o problema pede, tanto de maneira direta quanto através de alguma simulação;
- Em alguns casos, possuem **enunciados longos**, com objetivo de distrair do objetivo principal;
- Em alguns casos, possuem **casos limite perigosos**. É preciso ser muito cauteloso.
- Podem, também, ser bem difíceis. É necessário ter uma “**sacada genial**” para resolvê-lo.

Exercício: T9 Spelling

Descrição do Problema

The Latin alphabet contains 26 characters and telephones only have ten digits on the keypad. We would like to make it easier to write a message to your friend using a sequence of keypresses to indicate the desired characters. The letters are mapped onto the digits as shown below. To insert the character 'B' for instance, the program would press "22". In order to insert two characters in sequence from the same key, the user must pause before pressing the key a second time. The space character ' ' should be printed to indicate a pause. For example, "2 2" indicates "AA" whereas "22" indicates "B".



Exercício: T9 Spelling

Descrição da Entrada

The first line of input gives the number of cases, N , $1 \leq N \leq 100$. N test cases follow. Each case is a line of text containing the desired message, which will be at most 1000 characters long. Each message will consist of only lowercase characters 'a'–'z' and space characters ' '. Pressing zero emits a space.

Descrição da Saída

For each test case, output one line containing "Case #x: " followed by the message translated into the sequence of key presses.

Exemplo de Entrada	Exemplo de Saída
4 hi yes foo bar hello world	Case #1: 44 444 Case #2: 999337777 Case #3: 333666 6660 022 2777 Case #4: 4433555 555666096667775553