Introdução e dicas de Programação Competitiva

André Fakhoury

2021

1 Introdução

Neste texto, vou tentar explicar um pouco sobre algumas características de programação competitiva, e também dar algumas dicas mais gerais da linguagem *C++* que normalmente são úteis.

1.1 Dicas gerais de programação competitiva

Para resolver um problema, é importante ler o enunciado com atenção, e se atentar no que o problema pede, e quais as restrições de entrada, e então ir pensando em possíveis soluções. É importante pensar na complexidade e eficiência do código antes de implementar, para não perder tanto tempo. Esse processo vai ficando mais natural conforme o passar dos tempos. Por isso é importante resolver vários exercícios para ganhar experiência.

Normalmente, os códigos implementados para as competições não são os mais bonitos. Por mais que seja importante deixar o código legível, muitas vezes são deixadas de lado várias boas práticas de programação, para que a codificação seja feita mais rapidamente.

Na maioria dos sites, pode-se utilizar várias linguagens de programação, mas as principais são C++, Java e Python (e, dentro dessas, C++ é a mais utilizada).

1.2 Materiais complementares

Existem vários materiais complementares disponíveis na internet, caso queira aprender algum tópico ou sanar algumas dúvidas de programação competitiva. Vou tentar falar de alguns dos mais utilizados.

1.2.1 Sites

Para consulta das funções de C++, pode-se visitar os sites cplusplus e cppreference.

Para aprender, são bem famosos os sites geeksforgeeks e cp-algorithms. Também existem alguns tutoriais interessantes no hackerrank e hackerearth. Também existem vários outros disponíveis, só pesquisar no Google. O site do GEMA também possui materiais em português.

Para ver problemas e tentar solucionar, são famosos os sites codeforces, atcoder, cses.fi (com uma coletânea de problemas "clássicos"), topcoder, codechef, dentre outros.

1.2.2 Livros

Existem dois livros mais famosos para programação competitiva.

O primeiro é conhecido como "CP-3", que é a terceira edição do livro Competitive Programming, escrito por Steven Halim e Felix Halim.

O segundo talvez seja atualmente o mais utilizado: Competitive Programmer's Handbook, de Antti Laaksonen. Ele está disponível gratuitamente neste link.

Além destes, também existem vários livros caso você queira aprender e se familiarizar com *C++*.

1.2.3 Youtubers

Vou deixar aqui alguns canais que eu acho interessantes, e postam conteúdos relacionados à programação competitiva, como aulas, dicas e "screencasts" (gravação deles codando em algum contest).

- GEMA ICMC (aulas desde conteúdos mais básicos até alguns tópicos um pouco mais avançados)
- MaratonUSP (tipo o GEMA do IME)
- Errichto (possui screencasts e algumas aulas)
- Colin Galen (possui screencasts e algumas aulas)
- William Lin (possui mais screencasts de contests "mais básicos")
- Algorithms Live (várias aulas sobre alguns tópicos, em inglês)
- Tourist (o melhor programador competitivo, possui algumas screencasts interessantes)
- Petr Mitrichev (várias screencasts em Java e C++)

1.3 Vereditos da submissão

As submissões costumam ter os seguintes vereditos:

- **Accepted**: resposta correta :)
- **Wrong Answer**: resposta incorreta, mostrou um valor diferente do esperado.
- **Time Limit Exceeded**: excedeu o limite de tempo
- **Run Time Error**: deu algum erro de execução

- **Compilation Error**: deu algum erro de compilação

Caso sua submissão não tenha sido aceita, é interessante analisar o tipo de erro (se foi problema no tempo, resposta inválida, etc). Vou tentar listar alguns dos problemas mais comuns que podem acontecer.

1.3.1 Pensamento incorreto

Talvez a solução que você pensou está incorreta. Nesse caso, vale a pena tentar pensar em alguns casos de teste que possam quebrar a solução, ou até mesmo rever o enunciado para ver se não faltou nenhum detalhe.

1.3.2 Corner cases

Os "casos de borda" são aqueles casos em que tem que se analisar à parte. Por exemplo: o problema pede pra você calcular a área do triângulo, dados seus 3 lados. E se o triângulo não existir? Vai fazendo if...

1.3.3 Overflow

"Um inteiro possui 32 bits". As vezes, os limites do problema são muito altos, e tem que se utilizar tipos de variáveis diferentes. Em C++, temos o "int" que suporta até aproximadamente $2 * 10^9$ e o "long long" que vai até $8 * 10^{18}$ (pelo menos na grande maioria dos computadores atuais). Tomem cuidado com o tipo "long", ele engana...

1.3.4 Erros de precisão

Quando se utiliza variáveis de ponto flutuante (float, double), podem ocorrer alguns erros de precisão, devido à como estes valores são armazenados na memória. Num geral, é importante minimizar o uso de variáveis destes tipos, e fazer tudo com variáveis inteiras. Um exemplo disso é: como calcular o teto da divisão de a por b, se a e b são inteiros? Em vez de fazer 'ceil(double(a) / b)', podemos fazer '(a + b - 1) / b'.

Outro erro que pode acontecer é se você utilizar o 'cout' para imprimir. É importante que você explicite a quantidade de casas decimais para ele imprimir, senão pode acontecer dele imprimir menos que o necessário e ocasionar um WA. Para isso, pode-se fazer o seguinte para imprimir a variável "answer" com 10 casas decimais:

```
cout << fixed << setprecision(10) << answer << "\n";
```

1.3.5 Complexidade muito alta

Se o código excedeu o limite do tempo, talvez a sua solução não esteja na complexidade esperada para o problema. Um cálculo fácil de fazer é que um computador atual realiza cerca de 10^8 operações por segundo em um código de C++. Se os limites do problema são $n \leq 10^5$ e seu programa tem uma

complexidade de $O(n^2)$, é esperado que ele não passe no tempo limite, pois serão realizadas cerca de 10^{10} operações, que demorariam muito tempo.

1.3.6 Erros de execução

Os principais erros de execução que podem acontecer são acessos inválidos à vetores, utilização de variáveis não inicializadas e divisão por zero. No fim do texto eu dou algumas dicas de flags de compilação que podem ajudar com que você veja esses erros antes de submeter.

2 Programação em *C++*

A linguagem mais utilizada em programação competitiva é *C++*, principalmente por ser bem eficiente e possuir uma biblioteca pronta com vários algoritmos e estruturas de dados (a STL). Se você está familiarizado com *C*, não deve sentir tanta dificuldade para ganhar experiência com *C++*.

2.1 Estrutura básica de um código

A utilização de C++ com a STL terá uma estrutura similar à vista em C (com include, int main() e tudo mais). Um exemplo de entrada e saída de valores de um vetor pode ser visto a seguir:

```
#include <iostream> // biblioteca de entrada e saída
#include <vector> // vector

using namespace std; // explico mais ou menos depois

int main() {
   int n;
   cin >> n;
   vector<int> a(n);
   for (int i = 0; i < n; i++) {
      cin >> a[i];
   }

   for (int i = 0; i < n; i++) {
      cout << a[i];
   }
}
```

Como visto, um termo novo que apareceu é o *namespace*, utilizado para definir funções em algum contexto específico.

Toda função da STL é inserida em um namespace denominado *std*. Assim, caso você queira utilizar algo da STL (por exemplo, um vector), terá que fazer algo como: std::vector<int> v;

Para eliminar essa necessidade de colocar “std::” na frente de tudo que a gente for utilizar da STL, é possível simplificar e inserir este “using namespace std;” no início do código. Assim, quando a gente falar simplesment “vector<int>”, o compilador já sabe que a gente está falando do vector da STL.

2.2 A biblioteca bits/stdc++.h

Esta biblioteca é muito utilizada por já incluir várias outras bibliotecas padrões. Por exemplo, em vez de ter que ficar fazendo um include por biblioteca para usar as coisas da STL, basta apenas incluir essa bits, que vem tudo junto.

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

int main() {
   ...
}
```

Porém, infelizmente, a plataforma de correção automática do Run Codes não gosta muito dela, e dará um erro na submissão. Assim, se você estiver utilizando este judge, terá que incluir cada biblioteca separadamente.

2.3 cin, cout e o “fast i/o”

Caso você queira usar ‘cin’ e ‘cout’ para ler e imprimir (em vez de ‘scanf’ e ‘printf’, é importante tomar alguns cuidados adicionais.

Quando a entrada e a saída possuem muitos valores para serem tratados, a utilização de cin e cout pode ser muito lenta, por causa de sincronizações que são feitas entre as streams padrões de C (stdin, stdout, ...) com as streams de C++ (cin, cout, ...). Para contornar isso, pode-se adicionar a seguinte linha:

```
int main() {
   ios::sync_with_stdio(false); cin.tie(NULL);
   ...
}
```

Isso pode salvar um TLE. Porém, tome cuidado, e não fique misturando as funções de C (printf, scanf) com as funções de C++ (cin, cout), pois a ordem pode ficar toda bagunçada. Só escolher um e usar até o final.

2.4 Compilação e algumas flags

Assim como em C, também pode ser utilizado o GCC (ou clang, dentre outros) para compilar os códigos. Porém, como queremos compilar um código em C++, podemos utilizar o comando “g++”:

```
g++ A.cpp -o A
```

Também é possível informar algumas flags a mais para a compilação. Num geral, é sempre bom ser avisado de todos os warnings possíveis (acesso inválido à memória, utilizando variáveis não inicializadas previamente, overflows, etc). Também é bom sermos avisados em que linha o código está dando ruim. Para isso, pode-se utilizar, por exemplo, as seguintes flags:

```
g++ A.cpp -o A -Wall -fsanitize=address,undefined -g
```

Porém, talvez você possa se assustar um pouco com a verbosidade dos erros que irão acontecer sempre que tiver um erro de execução, como visto na figura 1.

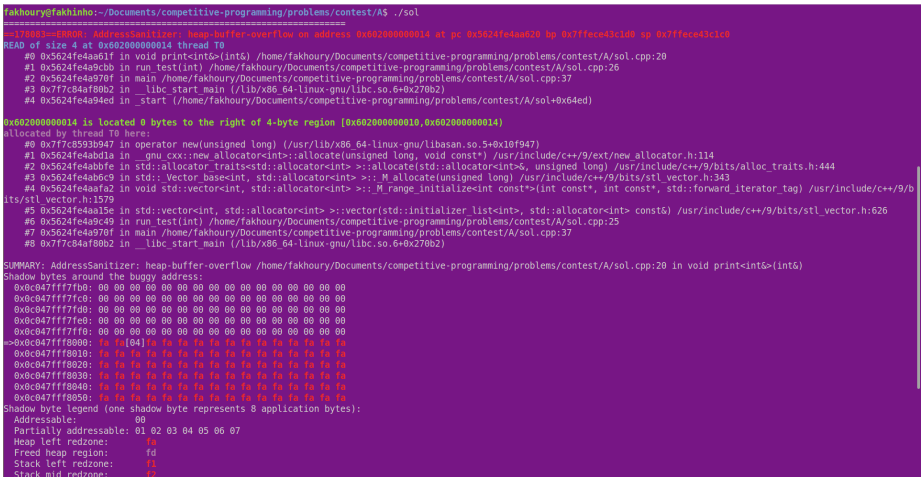

Figure 1: Erro de acesso inválido em um vector

Uma dica que eu dou é sempre olhar as primeiras linhas do erro. No exemplo da figura 1, ele fala que teve um erro na função declarada na linha 20 do meu código, que foi chamada pela linha 26, que foi chamada pela linha 37, etc. O erro muito provavelmente está em alguma dessas.

2.5 Qual IDE utilizar?

Não tem alguma "melhor IDE" para programação, cada pessoa possui suas preferências. As mais famosas, pelo que eu vejo, são Sublime Text, VIM, CLion (o da JetBrains) e VS Code.

Caso você precise de algum compilador online, existe o csacademy workspace que possibilita que você teste o programa para alguns casos de teste. Porém, é bem mais recomendável que você faça tudo em uma máquina local, se possível.