Aula 01 - Apresentação

Tópicos da aula

Considerações iniciais

Se você está aqui, é porque se diverte escrevendo códigos (doravante *codando*). No entanto, *codar* casualmente não tem tanta aplicação, até porque seu código pode demorar muito ou não está num formato profissionalmente interessante. E que tal ver quem faz o melhor código? :)

Por "melhor", entenda-se um código claro, organizado, curto e otimizado. A prática com a programação cuidará das duas primeiras partes. Este treinamento ajudará com as outras duas.

Como treinar na programação competitiva

Para desenvolver uma boa prática de programação, especialmente competitiva, nada melhor do que se pôr à prova!

E é para isso que existem os juízes online, vastos repertórios de problemas de programação que contêm verificadores automáticos de resposta adequados a cada linguagem interessante. O mais tradicional, e indicado no treinamento de C, é o *URI Online Judge*, mas seguem alguns outros (até mais) relevantes:

- *Sphere Online Judge* (SPOJ)

Como treinar na programação competitiva

Para desenvolver uma boa prática de programação, especialmente competitiva, nada melhor do que se pôr à prova!

E é para isso que existem os juízes online, vastos repertórios de problemas de programação que contêm verificadores automáticos de resposta adequados a cada linguagem interessante. O mais tradicional, e indicado no treinamento de C, é o *URI Online Judge*, mas seguem alguns outros (até mais) relevantes:

- *Sphere Online Judge* (SPOJ)
- *UVa Online Judge*

Como treinar na programação competitiva

Para desenvolver uma boa prática de programação, especialmente competitiva, nada melhor do que se pôr à prova!

E é para isso que existem os juízes online, vastos repertórios de problemas de programação que contêm verificadores automáticos de resposta adequados a cada linguagem interessante. O mais tradicional, e indicado no treinamento de C, é o *URI Online Judge*, mas seguem alguns outros (até mais) relevantes:

- *Sphere Online Judge* (SPOJ)
- *UVa Online Judge*
- *CodeForces*

Como treinar na programação competitiva

Para desenvolver uma boa prática de programação, especialmente competitiva, nada melhor do que se pôr à prova!

E é para isso que existem os juízes online, vastos repertórios de problemas de programação que contêm verificadores automáticos de resposta adequados a cada linguagem interessante. O mais tradicional, e indicado no treinamento de C, é o *URI Online Judge*, mas seguem alguns outros (até mais) relevantes:

- *Sphere Online Judge* (SPOJ)
- *UVa Online Judge*
- *CodeForces*
- *TopCoder*

Como treinar na programação competitiva

O URI é tradicionalmente recomendado por ter problemas bem separados por assunto geral e nível de dificuldade, sendo, portanto, mais didático. Mas um outro ponto forte dele é que nele se aplicam os *contests* de aquecimento para a OBI :)

Não explicamos antes, mas *contests* são pequenas provas de programação com problemas dos mais variados assuntos e níveis de dificuldade para serem resolvidos, em geral, em 4 horas. Neles, o tempo de submissão é usado como critério de desempate, então saber escrever pouco ou rápido faz a diferença, que pode ser a diferença entre o primeiro e o terceiro lugares (inclusive, as classificações são exibidas em tempo real).

Em geral, os *contests* de aquecimento da OBI, que são oficiais, são mais difíceis do que a prova mesmo, então não se assustem com o nível deles.

Conhecendo melhor a programação competitiva - OBI

Finalmente, daremos mais detalhes de como funciona a Olimpíada Brasileira de Informática. Ela é um evento individual composto de 3 fases, na sucessão interna $\rightarrow$ estadual $\rightarrow$ nacional. A cada transição de fase, apenas 30% dos participantes da fase anterior são aprovados, mas não desanime! Não é tão difícil assim passar de fase.

A primeira fase seleciona os participantes dentro de cada instituição de ensino participante e, em 2018, foram 3 problemas dos seguintes assuntos: matemática, vetores (mas dava pra fazer apenas com *while*) e algoritmo de *Dijkstra*. Este último é um algoritmo clássico de grafos, um assunto que será visto adiante.

Felizmente, a correção da OBI, embora automatizada, não é binária, então é possível conseguir pontuação parcial p pontos, com $0<p<100$.

Conhecendo melhor a programação competitiva - OBI

A segunda fase envolve problemas mais difíceis, porém é possível passar para a terceira fase, após a qual o comitê responsável divulga as pontuações finais de cada participante, com eventual premiação a depender da classificação nacional. O resultado final sai normalmente em outubro, e as fases (de duração máxima de 5h) ocorrem, respectivamente, em: maio, junho e agosto.

Se você está curioso pra saber como é o estilo da prova, confira o máximo possível de problemas da seção pratique, no site da OBI. Nele, também estão disponíveis alguns gabaritos de problemas de anos anteriores, bem como os resultados finais (na seção Resultados - Quadros de Medalhas) dos últimos anos.

Conhecendo melhor a programação competitiva - Maratona

Já a Maratona de Programação, esta é um evento do qual participam trios de programadores e é composta de apenas duas fases, regional e nacional. No entanto, não mais do que dois times por instituição de ensino podem participar da fase nacional, o que a torna ainda mais desafiadora. A prova em si consiste num total de 13 questões a serem resolvidas em 5 h.

No caso do ITA, uma terceira fase, a seleção interna, faz parte do processo. No ano passado, 4 times foram selecionados para participar da fase regional (no nosso caso, englobando o Vale do Paraíba), então as coisas já começam interessantes.

Também, na maratona em si você pode ganhar balão em tempo real para cada problema resolvido, com placar automaticamente atualizado e tudo mais (fora viajar para outros lugares :D). Não detalharei muito pois, embora seja um evento muito bom, não é o enfoque principal deste treinamento.

Problemas recorrentes com códigos

A partir de agora, serão comuns as expressões **Accepted**, **Wrong Answer**, **Time Limit Exceeded** e **Memory Limit Exceeded**, mais conhecidas como, respectivamente, AC, WA, TLE e MLE. As causas mais comuns desses problemas, incluindo-se a possibilidade de lógica errada, são:

- WA: Erro de precisão devido a *double* ou ocorreu *overflow* do limite do inteiro de 32 bits (o int clássico), i.e, alguma operação resultou num número pelo menos igual a 2^{31} e o ajuste com *long long int* (inteiro de 64 bits) foi esquecido, fora o óbvio.

Problemas recorrentes com códigos

A partir de agora, serão comuns as expressões **Accepted**, **Wrong Answer**, **Time Limit Exceeded** e **Memory Limit Exceeded**, mais conhecidas como, respectivamente, AC, WA, TLE e MLE. As causas mais comuns desses problemas, incluindo-se a possibilidade de lógica errada, são:

- WA: Erro de precisão devido a *double* ou ocorreu *overflow* do limite do inteiro de 32 bits (o int clássico), i.e, alguma operação resultou num número pelo menos igual a 2^{31} e o ajuste com *long long int* (inteiro de 64 bits) foi esquecido, fora o óbvio.
- TLE: Lógica insuficientemente boa, ou falta de otimização na leitura/impressão de dados (este problema só se torna relevante com o C++).

Problemas recorrentes com códigos

A partir de agora, serão comuns as expressões **Accepted**, **Wrong Answer**, **Time Limit Exceeded** e **Memory Limit Exceeded**, mais conhecidas como, respectivamente, AC, WA, TLE e MLE. As causas mais comuns desses problemas, incluindo-se a possibilidade de lógica errada, são:

- WA: Erro de precisão devido a *double* ou ocorreu *overflow* do limite do inteiro de 32 bits (o int clássico), i.e, alguma operação resultou num número pelo menos igual a 2^{31} e o ajuste com *long long int* (inteiro de 64 bits) foi esquecido, fora o óbvio.
- TLE: Lógica insuficientemente boa, ou falta de otimização na leitura/impressão de dados (este problema só se torna relevante com o C++).
- MLE: Memória reservada insuficiente para abranger o escopo dos casos-teste.

Noções de complexidade

A forma mais segura de evitar dores de cabeça por causa de TLE, que é o principal desafio da OBI, é elaborar um código cuja complexidade extrema não ultrapasse 1 segundo de execução. Para isso, faremos uso da notação O( ) (*big-O*) para denotar a complexidade de um código.

Vamos explicar, então, o que é complexidade de um código, para essa notação: é a quantidade máxima de operações que seu código pode executar dentro do escopo de possibilidades. Ela é tradicionalmente expressa por "funções imediatas" das variáveis pertinentes. Exemplos muito comuns de complexidades são: $O(1)$, $O(n)$, $O(n^2)$ e $O(n\log_2 n)$.

Considere, para fins práticos, que executar 10^8 operações equivale $\approx$ a 1 segundo de execução. Logo, um algoritmo de complexidade $O(n^2)$ de variável principal limitada até 10^4 ainda está ok.

Noções de complexidade

Para deixar mais claro o conceito, vejamos os seguintes exemplos:

Código 1

```
#include <stdio.h>


int main( ){
    int n;
    scanf("%d", &n);
    printf("%d\n", n*(n+1)/2);
    return 0;
}
```

Noções de complexidade

Código 2

```
#include <stdio.h>


int main( ){
    int n, soma=0;
    scanf("%d", &n);
    for(int i=1; i<=n; i++) soma+=i;
    printf("%d\n", soma);
    return 0;
}
```

Para cada código, avalie a complexidade e indique uma possível (e relevante) situação em que falhe.

Aquecimento

Vamos começar com o problema Floresta, da OBI de 2010.