Como aprender a programar

Programar é uma das mais difíceis atividades humanas. É preciso ser um bom resolvedor de problemas, mas é necessário ir mais além: Tem que ser também um bom meta-resolvedor de problemas. Ou seja, não basta saber como resolver uma questão, é preciso ensinar uma máquina a resolvê-la.

Se ela fosse inteligente, seria mais fácil. Mas a danada é meio burrinha, sem nenhum jogo de cintura. A especificação precisa descer a detalhes mínimos, ainda que o uso desses detalhes seja pouco comum. Mas, ao encontrar qualquer situação inesperada e não prevista, o programa (qualquer programa) estaca e entra em situação de estupor.

Há mais de 50 anos a academia discute como se deve ensinar e aprender a programar. A essência da programação está mais para arte do que para engenharia e é aí que a coisa complica: como ensinar a alguém a pintar um quadro ou a compor uma sinfonia ?

Há alguns consensos entre os autores: um deles é que aprender a programar passa por construir dezenas ou centenas de programas de complexidade crescente. Parece que ao construir programas simples primeiro e não tão simples depois o cérebro humano vai absorvendo as técnicas usadas e passa a pensar (meta-pensar) através delas. E sendo assim, está aberto o caminho para ser um belo programador.

Nós temos uma vantagem sobre os demais artistas. Trata-se do computador. Ele pode e deve ser usado como um auxiliar de ensino-aprendizagem. Os programas são feitos para rodar em um computador e ele pode nos dizer sem nenhuma dificuldade se os programas funcionam bem ou não.

Uma outra vantagem sobre um pintor ou sobre um compositor é que um programa ou funciona (e foi bem feito) ou não funciona (e é inútil). A decisão é exata e pode ser obtida com facilidade.

A figura do computador (e neste caso da Internet) apresenta ainda caminhos adicionais para aprender a programação, (há mais de 5000 no site), escreve um programa (em C, Java ou C++) que o resolve e submete ao site o programa fonte que escreveu.

Para auxiliá-lo nesta tarefa o site UVA entrega, para cada programa da lista, um conjunto preparado de dados de entrada e associados a cada um, o conjunto de dados de saída que o programa supostamente deveria produzir. Com essas 3 coisas: definição, dados de entrada e dados de saída esperados, o candidato a programador tem tudo o que precisa para escrever e enviar ao site o programa pedido.

O site, ao receber tal programa, compila-o, e o executa com uma instância nova (inédita) de dados.

Os resultados que o site entrega alguns segundos após a submissão são:

OK o programa que você submeteu foi rodado com dados inéditos e produziu exatamente o resultado esperado.

Limite tempo O programa não terminou no intervalo de tempo a ele alocado. Ou o programa está em loop devido a algum erro nas estruturas de repetição, ou o que é menos comum, é necessário um algoritmo mais eficiente para atender a este problema.

Erro de compilação O nome diz tudo, o site não conseguiu compilar o programa que você submeteu. Ou ocorreu um equívoco na linguagem selecionada ou a linguagem instalada no seu computador não é compatível com aquela usada pelo site. Ou ainda, o que é mais frequente, você cometeu um erro de escrita no seu código.

Resultado diferente O seu programa funcionou, mas produziu resultados distintos dos que o site esperava para aquele problema. Em outras palavras, sua solução é incorreta. A chave para resolver este tipo de erro é uma leitura cuidadosa da definição, sobretudo nos detalhes (chamadas condições de contorno).

☞ Para você fazer

Entre no site UVA e cadastre-se fornecendo nome e senha. Agora você tem uma conta no servidor e pode começar a submeter programas. Olhe o programa 100 e estude sua definição, dados de entrada e de saída. Defina, no seu computador uma solução. Se tiver dificuldade copie este código

```
#include <stdio.h>
long acha (long m) {
   int ciclo;
   ciclo = 1;
   while (m != 1) {
      if (0 == (m%2)) {
       m = m / 2;
      }
      else
      {m = (3*m)+1;
   }
      ciclo++;
   }
   return ciclo;
}

int main() {
   long i,j,aux,trocou,maior,ini,ago;
   while (scanf("%ld %ld",&i,&j) !=EOF) {
      trocou = 0;
      maior = 0;
      if (i > j) {
      aux = i;
      i = j;
      j = aux;
      trocou = 1;
      }
      ini = i;
      while (ini <= j) {
      ago = acha(ini);
      if  (ago > maior) {
         maior = ago;
      }
      ini++;
      }
      if (trocou == 0) {
       printf("%ld %ld %ld\n",i,j,maior);
      }
      else {
         printf("%ld %ld %ld\n",j,i,maior);
      }
   }
   return 0;
}
```

Projeto Euler

Este é outro esquema muito interessante para aprender a programar e se desenvolver na nobre arte. Ao contrário da UVA, no projeto Euler não se envia um programa e sim um resultado numérico que geralmente é muito difícil de se obter. O site é `http://projecteuler.net`.

Aqui, há razões de ordem prática para se buscar algoritmos e programas eficientes. Alguns pedidos se programados sem cuidado podem demorar meses ou anos em um computador comum. Ninguém precisa esperar tanto e segundo os criadores do site, nenhum problema demanda mais do que 1 segundo de CPU.

A questão é que para muitos problemas, tal limite de tempo impõe severas restrições e especialização no algoritmo procurado.

Um caso real que aconteceu comigo. Ao programar um problema (o 104 na lista do Euler) que diz

A sequencia de Fibonacci é definida pela relação de recorrência $F_n = F_{n-1} + F_{n-2}$ *onde* $F_1 = 1$ *e* $F_2 = 1$. *Deve-se notar que* F_{541} *que contém 113 dígitos é o primeiro número de Fibonacci para o qual os últimos 9 dígitos são pandigitais 1-9 (ou seja contém todos os dígitos de 1 a 9 não necessariamente em ordem). E,* F_{2749} *que contém 575 dígitos é o primeiro número de Fibonacci para o qual os primeiros 9 dígitos são pandigitais 1-9. Dado* F_k *que é o primeiro número de Fibonacci para o qual os primeiros 9 dígitos E os últimos 9 dígitos são pandigitais 1-9, ache k.*

Minha primeira opção demorou 5 dias rodando para achar o resultado (certo). Essa demora se justificava pois lida-se com números de Fibonacci com mais de 5000 dígitos cada um. Daí, quebrei a cabeça para otimizar o programa e alguns dias depois cheguei a uma versão equivalente que demorou menos de 10 segundos de tempo de relógio.

Seja como for, como você só vai fornecer um resultado, não importa quanto vai demorar, nem qual linguagem/compilador ou ambiente operacional você vai usar. Em tese, poderia até usar lápis e papel apenas. Há usuários cadastrados usando mais de 100 linguagens diferentes na obtenção das soluções.

O site não dá nenhuma dica ou auxílio até você resolver certo o problema. Nessa hora você ganha acesso a uma lista de discussão de todas as pessoas que já resolveram o mesmo problema. Muita da otimização posterior vem das idéias postadas nessa lista.

Alguns problemas são muito difíceis. Pode-se dizer que são adequados apenas para poucas centenas de habitantes deste planeta. A coisa é, como se dizia quando eu era pequeno, *às veras*.

☞ Para você fazer

Entre no site Euler e cadastre-se fornecendo nome e senha. Agora você tem uma conta no servidor e pode começar a submeter problemas. Escolha o problema 1, cuja definição é *Se listarmos todos os números naturais abaixo de 10 que são múltiplos de 3 ou 5, obteremos 3, 5, 6 e 9. A soma desses múltiplos é 23. Ache a soma de todos os múltiplos de 3 ou 5 abaixo de 1000.*

Tente escrever um programa de computador que ache a resposta.

Se não conseguir, pode oferecer a resposta certa que é 233168.

Pronto, você já é membro da coletividade mundial do projeto Euler.

Sexta Brilhante

Aqui na UP, temos um subconjunto em português do site Euler, que funciona nas mesmas regras. Os problemas são parecidos, mas não exatamente iguais.

☞ Para você fazer

Entre no site da sexta brilhante, `http://www.hamerski.com.br/sexta` e se cadastre. Olhe o primeiro problema, que diz:

A seguinte sequência iterativa é definida para todos os inteiros positivos: $n \to n/2$ *(quando n é par) e* $n \to 3n+1$ *(quando n é ímpar). Usando esta regra e começando com 13, gera-se a seguinte sequência:* $13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$. *Pode-se ver que esta sequência (começando em 13 e terminando em 1) contém 10 termos. Ainda que não tenha sido provado, espera-se que todas as sequências terminem em 1. Qual o número inicial, abaixo de 800000, que produz a maior sequência em comprimento? Note que depois que a cadeia começa, os termos podem passar de 800000.*

A resposta é 626331.