CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

1 de agosto de 2020

Resumo

Números primos

Objetivos da aula

- Desenvolver um dos mais completos programas desta fase do curso
- Praticamente integrando diversas técnicas básicas
- Discutir princípios de eficiência dos algorimos

Números primos

- Um número natural positivo A é dito primo se não possui divisores próprios, isto é, não existem divisores d no intervalo $2 \leq d \leq A - 1$.

Subproblemas

- Se soubermos quantos divisores um dado número *a* possui a solução ficaria trivial, não?

```
1  begin
2      read (n);
3      num_div_proprios:= ``calcula o numero de divisores proprios de
           n'';
4      if num_div_proprios = 0 then
5          writeln ('SIM')
6      else
7          writeln ('NAO');
8  end.
```

Rascunho 1

```
1  program primos_v0;
2  var n, i, cont: longint;
3  begin
4      read (n);
5      cont:= 0;
6      i:= 2;
7      while i <= n-1 do
8      begin
9          if n mod i = 0 then
10             cont:= cont + 1;
11         i:= i + 1;
12     end;
13     if cont = 0 then
14         writeln ('SIM')
15     else
16         writeln ('NAO')
17 end.
```

Problema

Ineficiente!

- Executa $n - 2$ iterações sempre
- Poderia parar se achar algum divisor!!!

Rascunho 2: para no primeiro divisor encontrado

```
1  program primos_v1;
2  var n, i, cont: longint; eh_primo: boolean;
3  begin
4      read (n);
5      cont:= 0;
6      eh_primo:= true;
7      i:= 2;
8      while eh_primo and (i <= n-1) do
9      begin
10         if n mod i = 0 then
11             eh_primo:= false;
12         i:= i + 1;
13     end;
14     if eh_primo then
15         writeln ('SIM')
16     else
17         writeln ('NAO')
18 end.
```

Ainda pode ser melhor

Ineficiente!

- É um pouquinho melhor do que o primeiro
- Mas, no pior caso, ainda executa $n - 2$
- Isso ocorre quando n é primo

Como melhorar?

- Só existe um único número par que é primo, o 2
- Todos os outros primos são ímpares
- Então vamos testar se n é par. Se for, resta saber se é o 2 ou outro par qualquer
- Depois resta testar todos os ímpares
- Assim, no pior caso, temos um primo ímpar e o algoritmo *só* vai iterar na ordem de metade das vezes

Rascunho 3: separa pares de ímpares

```
1  program primos_v2;
2  var n, i, cont: longint; eh_primo: boolean;
3  begin
4      read (n);
5      cont:= 0;
6      if n mod 2 = 0 then (* eh par *)
7          if n = 2 then eh_primo:= true
8          else eh_primo:= false
9      else (* eh impar *)
10     begin
11         eh_primo:= true;
12         i:= 3;
13         while eh_primo and (i <= n-1) do
14         begin
15             if n mod i = 0 then
16                 eh_primo:= false;
17             i:= i + 2;
18         end;
19     end;
20     if eh_primo then writeln ('SIM')
21     else writeln ('NAO')
22 end.
```

Pode ser *BEM* melhor

Ineficiente!

- É, de novo, um pouquinho melhor do que o anterior
- Mas, no pior caso, ainda executa $\frac{n-2}{2}$
- Matematicamente, dado um número n, o maior divisor que ele pode ter é $\sqrt{n}$

Versão final: vai até $\sqrt{n}$

```
1  program primos_final;
2  var n, i, cont: longint; eh_primo: boolean;
3  begin
4      read (n);
5      cont:= 0;
6      if n mod 2 = 0 then (* eh par *)
7          if n = 2 then eh_primo:= true
8          else eh_primo:= false
9      else (* eh impar *)
10     begin
11         eh_primo:= true;
12         i:= 3;
13         while eh_primo and (i <= sqrt(n)) do
14         begin
15             if n mod i = 0 then
16                 eh_primo:= false;
17             i:= i + 2;
18         end;
19     end;
20     if eh_primo then writeln ('SIM')
21     else writeln ('NAO')
22 end.
```

Comparação das funções

n	$\frac{n}{2}$	$\sqrt{n}$
1000000	500000	1000
1000000000	500000000	31622
1000000000000	500000000000	1000000

- Para n da ordem de 10^{12}
  - a versão do rascunho 3 vai levar muito tempo
  - a versão final seis ordens de grandeza a menos

Conclusão

- O melhor teria sido analisar completamente o problema *antes* de fazer qualquer implementação
- Neste caso, felizmente, a adaptação foi simples
- Existem casos em que toda a implementação tem que ser jogada fora para implementar outra, totalmente diferente
- A busca pela eficiência é um diferencial em Ciência da Computação!

Conclusão

- Com isso terminamos o capítulo 7
- Passaremos às questões de modularidade
- Depois, finalmente, às estruturas de dados

Exercícios

- Fazer os exercícios 1 a 17 da seção 7.12 do livro [1]
- Os exercícios de 18 a 22 são desafios

[1] `http://www.inf.ufpr.br/cursos/ci055/livro_alg1.pdf`

Fim do tópico

- este material está no livro no capítulo 7, seção 7.11

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`