Introdução à disciplina de algoritmos

Rafael Beserra Gomes, *Universidade Federal do Rio Grande do Norte*

Nas aulas introdutórias estudamos o conceito de algoritmos, o funcionamento geral de um computador e uma noção básica de como os algoritmos podem ser executados computacionalmente. Esse texto[1] complementa alguns pontos importantes dessas aulas.

1 Introdução

Escrever um **algoritmo** significa definir uma sequência de passos para resolver um problema. Nesta disciplina vamos aprender como escrever **algoritmos** em um contexto computacional. Dessa forma é necessário entender ao menos um pouco sobre um sistema computacional.

Discutimos que os computadores têm como finalidade processar dados e, de forma geral, os sistemas computacionais são capazes de adquirir, armazenar, processar e exibir os dados. Em um jogo digital, todos os itens abaixo a título de exemplo, o computador:

- adquire alguns dados do usuário: posição da tela onde o usuário quer apontar (através do mouse) e caracteres ou funções específicas (teclado);

[1] Todos os textos enviados para a turma ainda são rascunhos. Alerta de erros e sugestões podem ser enviados por email - o autor agradece bastante qualquer colaboração.

- armazena dados de forma duradoura: as maiores pontuações do jogo, onde o jogador estava na última vez que jogou; ou ainda de forma temporária: qual a posição atual do personagem, um número aleatório;
- processa dados: qual a posição do personagem no próximo instante, qual a velocidade de um objeto que foi lançado em função do tempo de jogo;
- exibe dados: informações geométricas de um modelo tridimensional do personagem são convertidas em uma imagem que é visualizável no monitor; um som é tocado quando o personagem finaliza uma fase.

Observe que há possibilidade de um sistema não precisar apresentar uma ou mais dessas características. Por exemplo, um sistema de irrigação pode adquirir dados (umidade do solo) para acionar ou interromper a irrigação do solo. O resultado do processamento não foi exibido para um ser humano, embora também possa ter sido, mas serviu como fonte de informação para um dispositivo mecânico que iniciará ou interromperá a irrigação.

Não é objetivo dessa disciplina estudar em detalhes o funcionamento de um computador e seus diversos componentes. Entretanto, são necessários alguns estudos específicos para uma maior compreensão sobre o que ocorre na programação de computadores em geral.

Os computadores possuem alguns componentes essenciais, dos quais são importantes para nosso estudo: memória, dispositivos de entrada, dispositivos de saída e CPU. Através da **memória** é possível armazenar os dados e os programas. Os **dispositivos de entrada e os de saída** são os responsáveis, respectivamente, pela aquisição e exibição dos dados. A **CPU** (um processador) é a unidade central de processamento, um componente que executa instruções básicas como: operações aritméticas e modificação dos dados na memória. Apesar de básicas, o conjunto de instruções que existem nos processadores aliado à velocidade de processamento permitem a execução dos programas como conhecemos, por mais complexos que possam parecer.

1.1 Observações

É essencial ter em mente que os algoritmos dependem do problema e, uma vez que os problemas que surgem na prática são os mais diversos, não há como memorizar algoritmos para resolver cada novo problema. Dessa forma você será avaliado quanto à sua capacidade em elaborar um algoritmo - isto exigirá bastante do seu raciocínio. Você notará que há até certos padrões entre um algoritmo e outro. Entretanto, a sugestão é que não tente de forma alguma memorizar algoritmos, mas sim tente compreender sua lógica.

Ainda assim, nesta disciplina a compreensão dos algoritmos é uma etapa. Não é suficiente para garantir que você seja capaz de elaborar algoritmos. Você poderá compreender todos os algoritmos explicados pelo professor, mas não ser capaz de fazê-los sozinho. Para que você possa adquirir essa habilidade, é necessário então que você **pratique bastante sem ajuda externa**.

Essa prática deve ser feita desde as primeiras lições de forma que essa habilidade possa ser desenvolvida gradativamente. Alunos que não mantém regularidade nos estudos desta disciplina costumam ter maior dificuldade no decorrer dela.

2 Como as informações são armazenadas na memória

Estamos habituados a utilizar computadores e ver informações em vários formatos, como texto, imagem, vídeo, som, um ambiente virtual, para somente citar alguns. Todos esses dados são representados no computador de forma digital em um sistema de **base binária**. Assim, todas essas informações são codificadas como uma sequência de zeros e uns, denominados **bits**. Vide Figura 1.

Os dados mais básicos em um sistema computacional são números inteiros, reais e caracteres. Vários outros dados são armazenados em função desses. Por exemplo: um texto é codificado como uma sequência de caracteres; uma imagem em tons de cinza pode ser representada como uma sequência de números que representam o tom de cinza de cada pixel que compõe a imagem; um som pode ser representado como uma sequência de números reais que indicam a amplitude de um sinal sonoro em função do tempo decorrido.

O ideal é que esses dados mais básicos possuam uma forma padrão de representação, o que é compreensível, já que esses dados devem eventualmente passar por diversos computadores e, portanto, não seria bom que cada computador interpretasse a sequência de bits de uma forma diferente.

Os números reais, por exemplo, seguem o padrão IEEE 754; os números inteiros são escritos em complemento de 2 (uma técnica que permite representar também números negativos), geralmente, em 4 bytes e os caracteres geralmente são escritos de acordo com o padrão ASCII ou o padrão Unicode.

2.1 Limites dos dados

Como os padrões estabelecem um limite da quantidade de bits para cada dado, naturalmente há um número limitado de informações representáveis. Por exemplo, a representação de um caractere com 8 bits permite que apenas 256 (2^8) símbolos sejam representáveis. Um inteiro com 4 bytes pode representar

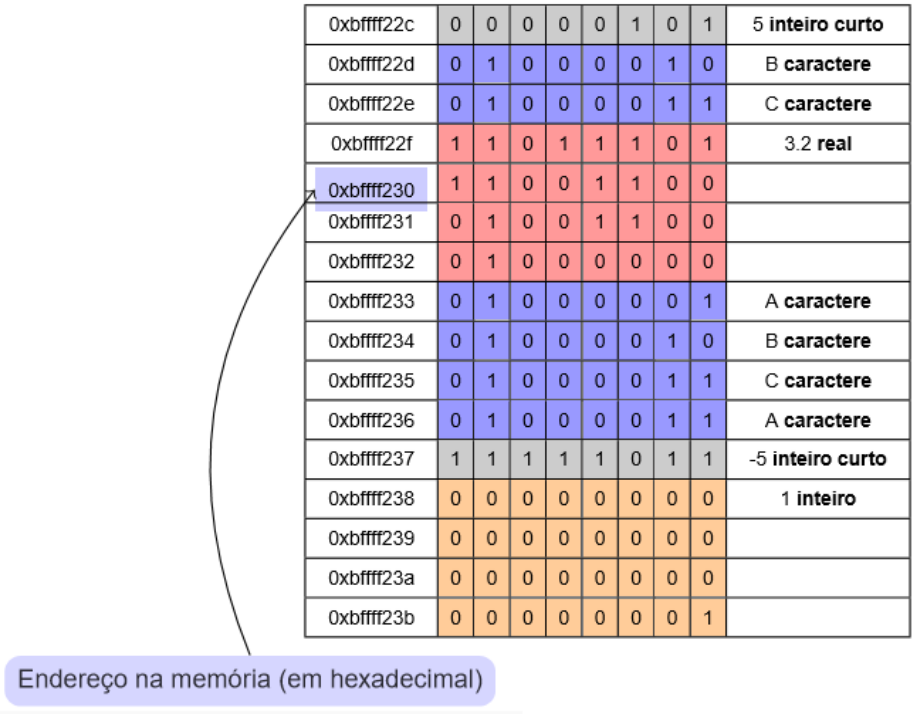

Figure 1: *Uma pequena parte da memória RAM em determinado momento de um computador fictício. A coluna à esquerda representa o endereço de memória. A coluna à direita representa a informação que o grupo de bits à esquerda representa.*

números inteiros de –2147483648 a 2147483647. Como o número de bits para representar um número real também é limitado, este pelo padrão IEEE 754, não é possível representar com exatidão determinados números. Apesar dos limites, os processadores são capazes de lidar com esses números de forma extremamente eficiente, pois há um circuito digital específico para as operações.

Há soluções para contornar essas limitações, mas exigem, ou hardwares específicos para isso, ou soluções via software que consomem várias instruções do processador (o que deve ocorrer na maioria dos casos)[2]. Se você utilizar uma calculadora do seu sistema operacional, provavelmente permitirá que você trabalhe com números fora do intervalo [–2147483648, 2147483647]. Isso é

[2]O interpretador Python já soluciona as limitações numéricas, então você não precisa se preocupar quanto aos limites de um número inteiro. Já a linguagem C não soluciona esse problema de forma nativa.

possível, por exemplo, utilizando vários números inteiros para representar um inteiro maior.

2.2 Extensão de arquivos

As extensões de arquivo devem indicar como os dados estão armazenados em um arquivo. Por exemplo, por um arquivo *txt* codificado em ASCII, espera-se como conteúdo grupos de 8 bits (1 byte), cada qual representando um caractere. Já um arquivo *txt* codificado em Unicode, espera-se como conteúdo grupos de 8 a 32 bits (1 a 4 bytes), cada qual representando um caractere.

Os programadores podem armazenar os dados da forma que bem entenderem. Obviamente, se algum programa precisar das informações desse arquivo, precisará saber como esses dados foram codificados. Dessa forma, costuma-se utilizar os padrões já bem estabelecidos na comunidade.

3 Como os algoritmos são implementados computacionalmente

Os algoritmos são escritos como uma sequência de passos, os quais computacionalmente são, em última análise, convertidos em uma sequência de instruções. Essas instruções em um computador são especificadas como uma sequência de bits de acordo com uma **linguagem de máquina**.

Entretanto, essa linguagem é específica demais e pouco produtiva para escrever algoritmos. Para ilustrar, eis um exemplo fictício. Imagine um processador com instruções que ocupam 16 bits, onde os 5 primeiros bits representem que instrução será executada, os 3 bits seguintes o endereço do registrador e os 8 bits restantes um valor. Essa instrução pode ser, por exemplo, somar 1 ao valor que está no registrador especificado. Essas instruções são elementares demais para escrever um programa considerado por muitos como simples. Por exemplo, são necessárias várias dessas instruções em linguagem de máquina para escrever um programa que exibe na tela os números de 0 a 9.

As instruções que compõem os nossos algoritmos são geralmente armazenados em um arquivo normal. Entretanto, ao solicitar a execução, a sequência de instruções é transferida para a memória RAM do computador (a mesma que armazena os dados, como na Figura 1). Logo, cada instrução também possui um endereço na memória.

É importante notar que as instruções da CPU são tais que é possível, após a execução da instrução atual, executar a próxima instrução ou a instrução em qualquer outro endereço de memória. Assim é possível para o computador executar

os algoritmos passo a passo e desviar o fluxo de execução, conforme aprenderemos mais adiante na disciplina com condicionais, estruturas de repetição e funções.

4 Linguagens de programação

As **linguagens de programação** permitem que escrevamos um algoritmo em uma linguagem que representa melhor o que desejamos fazer. Em geral, escrevemos o algoritmo de acordo com essa linguagem de programação em um arquivo, denominado **código-fonte**, o qual passará por outras etapas antes de ser executado. Há algumas linguagens de programação em que o algoritmo não é necessariamente escrito em um arquivo código-fonte, mas expresso passo a passo durante a execução do próprio programa.

Os algoritmos escritos nos códigos-fonte são executados de formas diferentes, dependendo da linguagem de programação. Em uma **linguagem compilada**, um software específico chamado **compilador**, traduz (denominamos essa ação de compilar) o código para linguagem de máquina. O programa em código de máquina (**executável**) somente poderá ser executado, como requisito mínimo, em máquinas capazes de executar as instruções nele contidas. Dessa forma, um programa compilado para um computador desktop não será executado nativamente em um smartphone.

Os algoritmos em **linguagens interpretadas** não são convertidos para linguagem de máquina mas interpretadas por um programa **interpretador** que executa as instruções correspondentes de acordo com o que foi interpretado. Isso confere às linguagens interpretadas uma grande vantagem: não há necessidade de recompilar os códigos para executar em diferentes plataformas. Um algoritmo escrito na linguagem Python, uma linguagem interpretada, será executado diretamente em qualquer computador que tenha interpretador Python.

Observe que em ambos os casos, o programa é executado por uma sequência de instruções em código de máquina. Mas, no caso das linguagens interpretadas, há a intermediação de um software interpretador. Enquanto nas linguagens compiladas, as instruções já estão prontas no arquivo executável.

Por esse último fato, as linguagens interpretadas tendem a ser mais lentas: o processador perde tempo com instruções com a finalidade de interpretar o código-fonte **enquanto executa o programa**. Nas linguagens compiladas, o compilador também perde tempo interpretando o código-fonte. Mas uma diferença é que a interpretação do código-fonte ocorre **durante a compilação do programa** e não durante a execução do programa. Uma vez compilado, as instruções já estão presentes no executável e não há necessidade de interpretação

de código-fonte.

Há o caso também de linguagens que apresentam as duas formas discutidas. O código em linguagem Java, por exemplo, é compilado para um arquivo em um formato chamado *byte-code*. Este arquivo pode ser então interpretado por um software chamado JVM. Esse software contém uma otimização (chamada JIT) que permite que trechos do código possam ser compilados durante a execução do programa por questões de eficiência.

4.1 As linguagens de programação

Há várias linguagens de programação disponíveis. Para citar algumas populares: C, C++, Java, Python, Pascal, Portugol, Ruby, Basic, PHP, Lua, R, JavaScript, C#, Scheme, Haskell, Perl, Lisp.

Cada uma dessas linguagens possui vantagens e desvantagens, de acordo com o contexto. A linguagem PHP, por exemplo, é excelente para execução em páginas web. Com a linguagem Python é possível programar rapidamente. JavaScript é uma linguagem para acrescentar elementos dinâmicos às páginas web. C é uma linguagem que permite a construção de programas extremamente eficientes. R é bastante utilizado para aplicações científicas. Essa é uma discussão bastante abrangente. É preciso experimentar e conhecer cada uma das linguagens para conhecer as potencialidades e limitações de cada uma.

O nível de abstração de uma linguagem de programação está de acordo com um critério subjetivo. Quanto mais próxima uma linguagem está da linguagem humana, maior o seu nível. Quanto mais próxima uma linguagem está da linguagem de máquina, menor o seu nível.

Veja a seguir um exemplo de código-fonte que exibe na tela os números de 0 a 9 em duas linguagens de programação. Observe como o código em Python é bem mais enxuto e fácil de ler do que o código em C.

```
1 for i in xrange(10):
2     print i
```

Código 1: *exibindo os números de 0 a 9 em Python*

```
1 #include <stdio.h>
2
3 int main() {
4     int i;
5     for(i = 0; i < 10; i++) {
6         printf("%d\n", i);
7     }
8     return 0;
9 }
```

Código 2: *exibindo os números de 0 a 9 em C*

Assim como os idiomas, cada uma dessas linguagens de programação possui uma **sintaxe** e uma **semântica**. Ambas devem ser seguidas à risca, sem flexibilização. Os programadores novatos devem se incomodar com isso, mas essa intransigência permite que os códigos não possam ser ambíguos. Caso não fosse assim, cada compilador/interpretador poderia interpretar de uma forma e gerar programas diferentes.

Se a intenção é apenas comunicar um algoritmo, é possível ainda escrevê-lo em **pseudo-código**, no qual há total liberdade de escrita. Se a linguagem utilizada é próxima de uma linguagem de programação, haverá maior facilidade de implementação computacional desse algoritmo. O uso de pseudo-código permite que o leitor possa focar na lógica do algoritmo ao invés de detalhes técnicos de cada linguagem de programação. Com tal liberdade, cabe ao autor do pseudo-código avaliar a linguagem utilizada e o que deve escrever em termos mais abstratos ou não.

4.2 Como programar

Cada linguagem de programação possui formas diferentes com a qual é possível programar. A maioria, seja interpretada ou compilada, possui a possibilidade de escrever o código-fonte em qualquer editor de texto em texto puro. De fato, o código-fonte é um arquivo textual, mas esse arquivo não deve ter formatação textual, uma vez que esta inclui informações adicionais no texto que não são reconhecidas pelo compilador ou interpretador.

Programas específicos categorizados como **IDE** (*integrated development environment*) existem com a finalidade de facilitar a programação em computadores. Uma IDE pode ter como recursos, para citar os mais básicos: coloração automática do código, facilitando a leitura; indentação automática; organização de códigos-fonte; configurações para alterar parâmetros de compilação; fácil acesso a compilação e execução dos programas; consulta à documentação; autocomplemento de código.