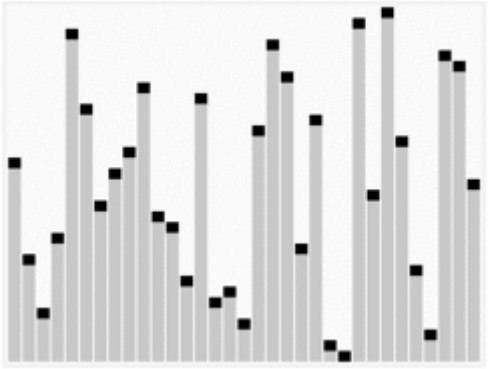
WIKIPÉDIA

Algoritmo

Origem: Wikipédia, a enciclopédia livre.

Algoritmo é uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais devendo ser executadas mecânica ou eletronicamente em um intervalo de tempo finito e com uma quantidade de esforço finita.[1][2] [1]



Uma animação do algoritmo de ordenação quicksort de uma matriz de valores ao acaso. As barras vermelhas marcam o elemento pivô. No início da animação, estando o elemento para o lado direito, é escolhido como o pivô.

O conceito de *algoritmo* existe há séculos e o uso do conceito pode ser atribuído a matemáticos gregos, por exemplo a Peneira de Eratóstenes e o algoritmo de Euclides.

O conceito de algoritmo é frequentemente ilustrado pelo exemplo de uma receita culinária, embora muitos algoritmos sejam mais complexos. Eles podem repetir passos (fazer iterações) ou necessitar de decisões (tais como comparações ou lógica) até que a tarefa seja completada. Um algoritmo corretamente executado não irá resolver um problema se estiver implementado incorretamente ou se não for apropriado ao problema.Jean Luc Chabert

Um algoritmo não representa, necessariamente, um programa de computador,[3] e sim os passos necessários para realizar uma tarefa. Sua implementação pode ser feita por um computador, por outro tipo de autômato ou mesmo por um ser humano. Diferentes algoritmos podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros. Tal diferença pode ser reflexo da complexidade computacionalaplicada, que depende de estruturas de dados adequadas ao algoritmo. Por exemplo, um algoritmo para se vestir pode especificar que você vista primeiro as meias e os sapatos antes de vestir a calça enquanto outro algoritmo especifica que você deve primeiro vestir a calça e depois as meias e os sapatos. Fica claro que o primeiro algoritmo é mais difícil de executar que o segundo apesar de ambos levarem ao mesmo resultado.Algorithmics

O conceito de um algoritmo foi formalizado em 1936 pela Máquina de Turing de Alan Turing e pelo cálculo lambda de Alonzo Church, que formaram as primeiras fundações daCiência da computação

Índice

Etimologia

Os historiadores da palavra *algoritmo* encontraram a origem no sobrenome, Al-Khwarizmi, do matemático persa do século IX Mohamed ben Musa,[4] cujas obras foram traduzidas no ocidente cristão no século XII, tendo uma delas recebido o nome *Algorithmi de numero indorum*, sobre os algoritmos usando o sistema de numeração decimal (indiano). Outros autores, entretanto, defendem a origem da palavra em *Al-goreten* (raiz - conceito que se pode aplicar aos cálculos).[5] "Álgebra" e "algorismo" também formam formas corrompidas da palavra, pois as pessoas esqueciam as derivações originais. O dicionário "Vollständiges Mathematisches Lexicon" (Leipzig, 1747) refere a palavra "Algorithmus"; nesta designação estão combinadas as noções de quatro cálculos aritméticos, nomeadamente a adição, multiplicação, subtração e divisão. A frase "algorithmus infinitesimalis" foi na altura utilizada para significar; "maneiras de calcular com quantidades infinitésimas" (pequenas), uma invenção de Leibnitz. Também é conhecido no meio financeiro, como "algos".[6]

Formalismo

Um programa de computador é essencialmente um algoritmo que diz ao computador os passos específicos e em que ordem eles devem ser executados, como por exemplo, os passos a serem tomados para calcular as notas que serão impressas nos boletins dos alunos de uma escola. Logo, o algoritmo pode ser considerado uma sequência de operações que podem ser simuladas por uma máquina de Turing completa.

Quando os procedimentos de um algoritmo envolvem o processamento de dados, a informação é lida de uma fonte de entrada, processada e retornada sob novo valor após processamento, o que geralmente é realizado com o auxílio de uma ou mais estrutura de dados.

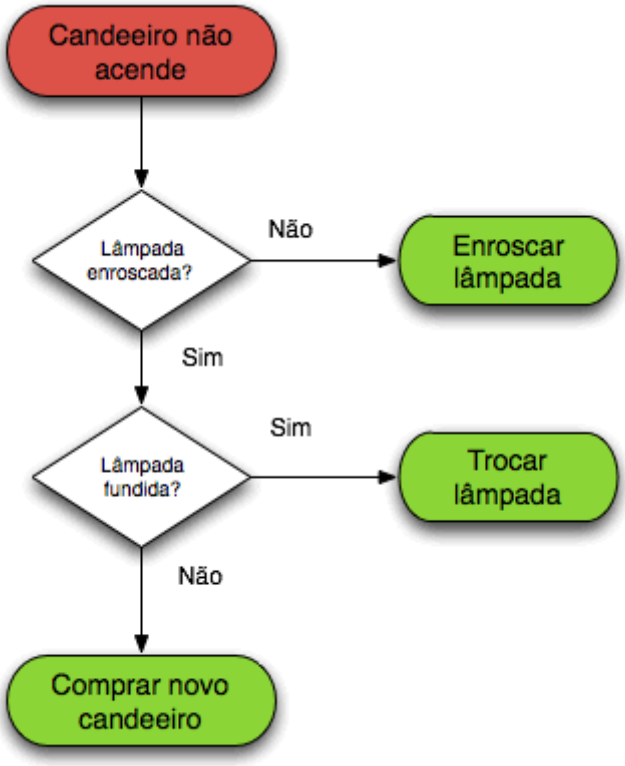
Para qualquer processo computacional, o algoritmo precisa estar rigorosamente definido, especificando a maneira que ele se comportará em todas as circunstâncias. A corretividade do algoritmo pode ser provada matematicamente, bem como a quantidade assintótica de tempo e espaço (complexidade) necessários para a sua execução. Estes aspectos dos algoritmos são alvo da análise de algoritmos.

A maneira mais simples de se pensar um algoritmo é por uma lista de procedimentos bem definida, na qual as instruções são executadas passo a passo a partir do começo da lista, uma ideia que pode ser facilmente visualizada através de um fluxograma. Tal formalização adota as premissas da programação imperativa, que é uma forma mecânica para visualizar e desenvolver um algoritmo. Concepções alternativas para algoritmos variam emprogramação funcionale programação lógica

Término do algoritmo

Alguns autores restringem a definição de algoritmo para procedimentos que eventualmente terminam. Marvin Minsky constatou que se o tamanho de um procedimento não é conhecido de antemão, tentar descobri-lo é um problema indecidível, já que o procedimento pode ser executado infinitamente, de forma que nunca se terá a resposta. Alan Turing provou em 1936 que não existe máquina de Turing para realizar tal análise para todos os casos, logo não há algoritmo para realizar tal tarefa para todos os casos. Tal condição é conhecida atualmente como problema da parada

Para algoritmos intermináveis o sucesso não pode ser determinado pela interpretação da resposta e sim por condições impostas pelo próprio desenvolvedor do algoritmo durante sua execução.



Fluxograma, um exemplo de algoritmo imperativo. O estado em vermelho indica a entrada do algoritmo enquanto os estados em verde indicam as possíveis saídas.

Exemplos

Alguns exemplos genéricos de algoritmos são: uma coreografia, um manual de instruções, uma receita culinária, Técnicas para resolver problemas matemáticos, uma pesquisa na internet, dentre outros.

Torre de Hanói

Um clássico problema que trabalha o desenvolvimento da lógica e do raciocínio matemático é a torre de Hanói, inventado pelo matemático francês Édouard Lucas em 1883.[7] O quebra-cabeça é composto por três hastes e vários discos de tamanhos diferentes, que podem deslizar para qualquer haste. O quebra-cabeça começa com os discos em uma pilha organizada em ordem crescente de tamanho em uma haste, a menor no topo, fazendo assim uma forma cônica.

Neste exemplo, toma-se o seguinte problema: tem-se três hastes. Umas das hastes serve de suporte para três discos. Deseja-se mover todos os discos para outra haste, porém deve-se movimentar um disco de cada vez e um disco maior nunca pode ser colocado sobre um disco de menor tamanho.

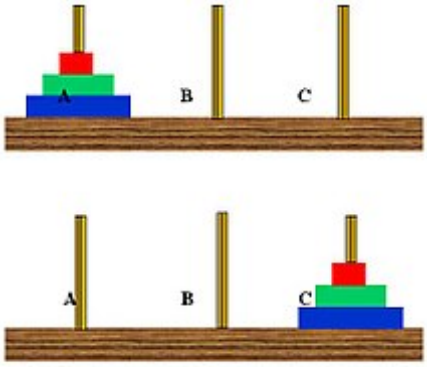


Imagem da torre de Hanói (com três discos), mostrando como estariam as peças no início e no fim da solução.

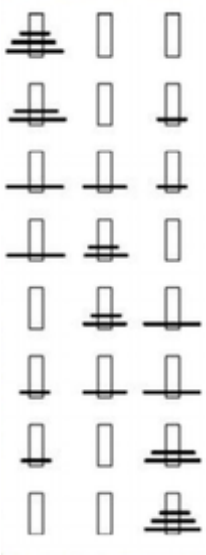
Solução em forma narrativa

Nomeiam-se as hastes como A, B e C e os discos como Vermelho, Verde e Azul. Considera-se que inicialmente os discos estão na haste A. Segue uma sequência de passos para a resolução do quebra-cabeça:

1. move-se o disco Vermelho para a haste C.
2. move-se o disco Verde para a haste B.
3. move-se o disco Vermelho para a haste B.
4. move-se o disco Azul para a haste C.
5. move-se o disco Vermelho para a haste A.
6. move-se o disco Verde para a haste C.
7. move-se o disco Vermelho para a haste C.

Solução em forma gráfica

Podemos também representar a solução em forma gráfica, desenhando as hastes e a posição dos discos a cada movimento (ou passo). Com 3 discos, o quebra-cabeça pode ser resolvido em 7 movimentos. O número mínimo de movimentos necessários para resolver um quebra-cabeça da Torre de Hanói é $2^n - 1$, onde n é o número de discos.



Resolução da torre de Hanói (com três discos).

Essa sequência, ou descrição, finita de passos ou tarefas é a quem chamamos de algoritmos.

Análise de algoritmos

A análise de algoritmos é um ramo da ciência da computação que estuda as técnicas de projeto de algoritmos e os algoritmos de forma abstrata, sem estarem implementados em uma linguagem de programação em particular ou implementadas de algum outro modo. Ela preocupa-se com os recursos necessários para a execução do algoritmo tais como o tempo de execução e o espaço de armazenamento de dados. Deve-se perceber que para um dado algoritmo pode-se ter diferentes quantidades de recursos alocados de acordo com os parâmetros passados na entrada. Por exemplo, se definirmos que o fatorial de um número natural é igual ao fatorial de seu antecessor multiplicado pelo próprio número, fica claro que a execução de `fatorial(10)` consome mais tempo que a execução de `fatorial(5)`.

Um meio de exibir um algoritmo a fim de analisá-lo é através da implementação por pseudocódigo em português estruturado, também conhecido no Brasil como Portugol. Este código pode ser digitado dentro de algum editor de textos como o Bloco de Notas, anotado num caderno ou ainda poder digitado diretamente dentro de um programa interpretador de algoritmos, como é caso do Visualg, que é um editor, interpretador e executor dos algoritmos.

Classificação

Classificação por implementação

Pode-se classificar algoritmos pela maneira pelo qual foram implementados.

- **Recursivo** ou **iterativo** - um algoritmo recursivo possui a característica de invocar a si mesmo repetidamente até que certa condição seja satisfeita e ele é terminado, que é um método comum em programação funcional. Algoritmos iterativos usam estruturas de repetição tais como laços, ou ainda estruturas de dados adicionais tais como pilhas, para resolver problemas. Cada algoritmo recursivo possui um algoritmo iterativo equivalente e vice-versa, mas que pode ter mais ou menos complexidade em sua construção.
- **Lógico** - um algoritmo pode ser visto como uma dedução lógica controlada. O componente lógico expressa os axiomas usados na computação e o componente de controle determina a maneira como a dedução é aplicada aos axiomas. Tal conceito é base para a programação lógica.
- **Serial** ou **paralelo** - algoritmos são geralmente assumidos por serem executados instrução a instrução individualmente, como uma lista de execução, o que constitui um algoritmo serial. Tal conceito é base para a programação imperativa. Por outro lado existem algoritmos executados paralelamente, que levam em conta as arquiteturas de computadores com mais de um processador para executar mais de uma instrução ao mesmo tempo. Tais algoritmos dividem os problemas em subproblemas e o delegam a quantos processadores estiverem disponíveis, agrupando no final o resultado dos subproblemas em um resultado final ao algoritmo. Tal conceito é base para a programação paralela. De forma geral, algoritmos iterativos são paralelizáveis; por outro lado existem algoritmos que não são paralelizáveis, chamados então problemas inerentemente seriais.
- **Determinístico** ou **não-determinístico** - algoritmos determinísticos resolvem o problema com uma decisão exata a cada passo enquanto algoritmos não-determinísticos resolvem o problema ao deduzir os melhores passos através de estimativas sob forma de heurísticas.
- **Exato** ou **aproximado** - enquanto alguns algoritmos encontram uma resposta exata, algoritmos de aproximação procuram uma resposta próxima a verdadeira solução, seja através de estratégia determinística ou aleatória. Possuem aplicações práticas sobretudo para problemas muito complexos, do qual uma resposta correta é inviável devido à sua complexidade computacional.

Classificação por paradigma

Pode-se classificar algoritmos pela metodologia ou paradigma de seu desenvolvimento, tais como:

- **Divisão e conquista**- algoritmos de divisão e conquista reduzem repetidamente o problema em sub-problemas, geralmente de forma recursiva, até que o sub-problema é pequeno o suficiente para ser resolvido. Um exemplo prático é o algoritmo de ordenação *merge sort*. Uma variante dessa metodologia é o *decremento e conquista*, que resolve um sub-problema e utiliza a solução para resolver um problema maior. Um exemplo prático é o algoritmo para pesquisa binária.
- **Programação dinâmica**- pode-se utilizar a programação dinâmica para evitar o re-cálculo de soluções já resolvidas anteriormente.
- **Algoritmo ganancioso**- um algoritmo ganancioso é similar à programação dinâmica, mas difere na medida em que as soluções dos sub-problemas não precisam ser conhecidas a cada passo, uma escolha gananciosa pode ser feita a cada momento com o que até então parece ser mais adequado.
- **Programação linear-** A resolução de um problema através de programação linear envolve a maximização / minimização das entradas de um conjunto de desigualdades lineares.
- **Redução** - a redução resolve o problema ao transformá-lo em outro problema. É chamado também *transformação e conquista*.
- **Busca e enumeração**- vários problemas podem ser modelados através de grafos. Um algoritmo de exploração de grafo pode ser usado para caminhar pela estrutura e retornam informações úteis para a resolução do problema. Esta categoria inclui algoritmos de busca e backtracking.
- **Paradigma heurístico e probabilístico**- algoritmos probabilísticos realizam escolhas aleatoriamente. Algoritmos genéticos tentam encontrar a solução através de ciclos de mutações evolucionárias entre gerações de passos, tendendo para a solução exata do problema. Algoritmos heurísticos encontram uma solução aproximada para o problema.

Classificação por campo de estudo

Cada campo da ciência possui seus próprios problemas e respectivos algoritmos adequados para resolvê-los. Exemplos clássicos são algoritmos de busca, de ordenação, de análise numérica, de teoria de grafos, de manipulação de cadeias de texto, de geometria computacional, de análise combinatória, de aprendizagem de máquina, de criptografia, de compressão de dados e de interpretação de texto.

Classificação por complexidade

Alguns algoritmos são executados em tempo linear, de acordo com a entrada, enquanto outros são executados em tempo exponencial ou até mesmo nunca terminam de serem executados. Alguns ditos problemas tem múltiplos algoritmos enquanto outros não possuem algoritmos para resolução.Jesús Bisbal Riera

Implementação

Algoritmos podem ser implementados em circuitos elétricos ou até mesmo em dispositivos mecânicos (autômatos).Mania dos inventores do século XIX, os autômatos eram máquinas totalmente mecânicas, construídas com a capacidade de serem programadas para realizar um conjunto de atividades autônomas. Em 2011, o filme *A Invenção de Hugo Cabret*(tradução brasileira) do cineasta Martin Scorsese traz a história do ilusionista Georges Méliès precursor do cinema e um colecionador de autômatos, sendo uma de suas máquinas o fio condutor desta história. O autômato específico era capaz de desenhar a cena emblemática do seu filme "Viagem à Lua".

Entretanto, a maioria dos algoritmos são desenvolvidos para programas de computador, para isto, existe uma grande variedade de linguagens de programação, cada uma com características específicas que podem facilitar a implementação de determinados algoritmos ou atender a propósitos mais gerais.

Programa de Computador

Ada Lovelace escreveu o primeiro algoritmo para ser processado por uma máquina, a máquina analítica de Charles Babbage. Um programa de computador é essencialmente um algoritmo que diz ao computador os passos específicos e em que ordem eles devem ser executados.Usando o Pseudocódigo (uma linguagem simples, nativa a quem o escreve, de forma a ser entendida por qualquer

pessoa) que é uma forma genérica de escrever o algoritmo, sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. Um exemplo de pseudocódigo é o Portugol, que utiliza o compilador VisuALG[8]. O VisuAlg é um programa que edita, interpreta e executa algoritmos com uma linguagem próxima do portuguêsestruturado como um programa normal de computador. É um programa de livre uso e distribuição, empregado no ensino de programação em várias escolas e universidades no Brasil e no exterior. Quando os procedimentos de um algoritmo envolvem o processamento de dados, a informação é lida de uma fonte de entrada, processada e retornada sob novo valor após processamento, o que geralmente é realizado com o auxílio de um conjunto de instruções e estrutura de dados.Um exemplo, para ser feito nas escolas é fazer os passos a serem tomados para calcular as notas que serão impressas nos boletins dos alunos de uma escola, informando se o anulo foi aprovado ou reprovado.

Exemplo:

```
// Seção de Declarações

var

NOTA1, NOTA2, NOTA3, NOTA4, MEDIA : REAL

inicio

// Seção de Comandos

ESCREVA("DIGITE A PRIMEIRA NOTA: ")

LEIA(NOTA1)

ESCREVA("DIGITE A SEGUNDA NOTA: ")

LEIA(NOTA2)

ESCREVA("DIGITE A TERCEIRA NOTA: ")

LEIA(NOTA3)

ESCREVA("DIGITE A QUARTA NOTA: ")

LEIA(NOTA4)

MEDIA := (NOTA1 + NOTA2 + NOTA3 + NOTA4) / 4 ;

SE MEDIA <=  6.9 ENTAO

   ESCREVA("A MEDIA DO ALUNO FOI: ", MEDIA)

   ESCREVAL (" - ALUNO REPROVADO ")

FIMSE

SE MEDIA >= 7 ENTAO

   ESCREVA("A MEDIA DO ALUNO FOI: ", MEDIA)

   ESCREVAL (" - ALUNO APROVADO ")

FIMSE

fimalgoritmo
```

Tradutor e Interpretador

Ao receber uma bicicleta no natal Carlinhos precisa ler o manual de instruções e seguir passo a passo as tarefas descritas no documento para poder se divertir com seu presente. Podemos dizer que Carlinhos é um interpretador dos comandos fornecidos pelo manual de instruções. Entretanto seu pai encontrou uma promoção na internet e comprou um produto fabricado na França e o menino ao se deparar com o manual percebeu que o mesmo não poderia ser "interpretado" já que não sabia ler em francês. Para resolver o problema seu pai contratou um tradutor de francês para português, assim, este novo manual pôde ser "interpretado" por Carlinhos e enfim sua bicicleta seria montada.

No computador, o problema de Carlinhos se repete diariamente, havendo a necessidade de softwares básicos para traduzir e interpretar os diversos programas dos usuários escritos em diversas linguagens existentes. Os softwares que convertem um programa de usuário escrito em uma linguagem para outra linguagem são chamados de tradutores. A linguagem na qual o programa original está expresso é chamada de linguagem fonte e a linguagem para a qual ela será convertida é conhecida como linguagem alvo. Tanto a linguagem fonte quanto a linguagem alvo definem níveis de abstração específicos.

Se existir um processador que possa executar diretamente programas escritos na linguagem fonte, não há necessidade de se traduzir o programa fonte para uma linguagem alvo.

O método de tradução é empregado quando há um processador (seja ele implementado em hardware ou por interpretação) disponível para executar programas expressos na linguagem alvo mas não na linguagem fonte. Se a tradução tiver sido feita corretamente, a execução do programa traduzido vai obter exatamente os mesmos resultados que a execução do programa fonte obteria se houvesse um processador que o executasse diretamente.

É importante observar a diferença entre tradução e interpretação. Na tradução, o programa original, expresso na linguagem fonte, não é executado diretamente. Em vez da execução direta, esse programa precisa ser convertido para um programa equivalente, conhecido como programa objeto ou programa binário executável, que será executado após o término do processo de tradução.

Logo, a tradução envolve dois passos distintos:

- Geração de um programa equivalente na linguagem alvo;
- Execução do programa obtido.

No processo de interpretação existe apenas um único passo: a execução do programa original na linguagem fonte.

Tradutores

Os tradutores podem ser divididos em dois grupos, dependendo da relação existente entre a linguagem fonte e a linguagem alvo. Quando a linguagem fonte for essencialmente uma representação simbólica para uma linguagem de máquina numérica, o tradutor é chamado de montador e a linguagem fonte é chamada de linguagem de montagem. Quando a linguagem fonte for uma linguagem de alto nível, como é o caso do Pascal ou do C, e a linguagem alvo for uma linguagem de máquina numérica ou uma representação simbólica desta linguagem (linguagem de montagem), o tradutor é chamado de compilador

Processo de Compilação

Diferente do processo de montagem de um programa em linguagem de montagem para um programa em linguagem de máquina, que é bastante simples, pois existe um mapeamento direto de um para um entre os comandos em linguagem de montagem e os equivalentes em código binário, o processo de compilação de linguagens é muito mais complexo.

Passos da compilação

Considere o comando simples abaixo:

```
A = B + 4;
```

O compilador tem que resolver um número grande de tarefas na conversão deste comando em um ou mais comandos em linguagem de montagem:

1.Reduzir o texto do programa para símbolos básicos da linguagem, como identificadores tais como A e B, demarcações como o valor constante 4 e delimitadores do programa tais como = e +. Esta parte da compilação é chamada de análise léxica.

2. Decodificar os símbolos para reconhecer a estrutura do programa. No comando usado acima, por exemplo, um programa chamado parser deve reconhecer o comando como sendo uma atribuição de valores da forma:

```
<Identificador> “=” <Expressão>
```

onde `<Expressão>` é decodificado na forma:

```
<Identificador> “+” <Constante>
```

Essa tarefa é chamada de análise sintática.

3. Análise de nomes: associar os nomes A e B com variáveis do programa, e associá-los também a posições de memória específicas onde essas variáveis serão armazenadas durante a execução.

4. Análise de tipos: determinar os tipos de todos os dados. No caso anterior, as variáveis A e B e a constante 4 seriam reconhecidas como sendo do tipo int em algumas linguagens. As análises de nome e tipo são também conhecidas como análise semântica: determina o significado dos componentes do programa.

5. Mapeamento de ações e geração de código: associar comandos do programa com uma sequência em linguagem de montagem apropriada. No caso anterior, a sequência em linguagem de montagem poderia ser:

Comando de atribuição.

```
ld[B], %r0, %r1    // Carrege variável B em um registrador.
add %r1, 4, %r2    // Calcule o valor da expressão.
st %r2, %r0, [A]   // Faça a atribuição na variável A.
```

6.Existem passos adicionais que o compilador deve tomar, tais como, alocar variáveis a registradores, usar registradores e, quando o programador desejar, otimizar o programa. O otimizador de código (independente de máquina) é um módulo opcional (presente na grande maioria dos compiladores) que objetiva melhorar o código intermediário de modo que o programa objeto produzido ao fim da compilação seja menor (ocupe menos espaço de memória) e/ou mais rápido (tenha tempo de execução menor). A saída do otimizador de código é um novo código intermediário.

Processo de Montagem

O processo de traduzir um programa em linguagem de montagem para programa em linguagem de máquina é chamado de processo de montagem. Este processo é muito simples, uma vez que existe um mapeamento um para um de comandos em linguagem de montagem para seus correspondentes em linguagem de máquina. Isto é o contrário da compilação, onde um comando em linguagem de alto nível pode ser traduzido em vários comandos em linguagem de máquina.

Por que usar uma Linguagem de Montagem?

Programar em uma linguagem de montagem não é fácil. Além da dificuldade, o desenvolvimento de um programa na linguagem de montagem consome mais tempo do que seu desenvolvimento em uma linguagem de alto nível. A depuração e manutenção dos programas em linguagem de montagem são mais complicados.

Nessas condições, por que alguém escolheria programar em uma linguagem de montagem?

Existem duas razões que justificam esta opção: performance e acesso aos recursos da máquina. Um expert na linguagem de montagem pode produzir um código menor e muito mais eficiente do que o gerado por um programador usando linguagem de alto nível.

Em segundo lugar, certos procedimentos precisam ter acesso total ao hardware. Por exemplo, se a máquina alvo tiver um bit para expressar o overflow de operações aritméticas, um programa em linguagem de montagem pode testar diretamente este bit, coisa que um programa em Java não pode fazer. Além disso, um programa em linguagem de montagem pode executar qualquer uma das instruções do conjunto de instruções da máquina alvo.

Tarefas do montador

Embora a montagem seja um processo simples, é tedioso e passível de erros quando feito manualmente. Montadores comerciais têm ao menos as seguintes características:

- Permitem ao programador especificar posição de valores de dados e programas durante a execução;
- Permitem que o programador de início realize valores de dados na memória antes da execução do programa;
- Implementam mnemônicos em linguagem de montagem para todas as instruções da máquina e modos de endereçamento, e traduzem comandos em linguagem de montagem válidos, nos seus equivalentes em linguagem de máquina;
- Permitem o uso de rótulos simbólicos para representar endereços e constantes;
- Incluem um mecanismo que permite que variáveis sejam definidas em um programa em linguagem de montagem e usadas em outros programas separadamente;
- Possibilitam a expansão de macros, ou seja, rotinas (semelhantes às funções em linguagem de alto nível) que podem ser definidas uma vez e então instanciadas quantas vezes necessário.

Montadores de dois passos

A maioria dos montadores leem textos do programa em linguagem de montagem duas vezes, e são chamados de “montadores de dois passos”. O primeiro passo serve para determinar o endereço de todos os itens de dados e instruções de máquina, e selecionar quais instruções devem ser geradas para cada instrução em linguagem de montagem (mais ainda não gerá-las).

Os endereços dos itens de dados e instruções são determinados por meio do uso de um contador de programa para a montagem, chamado contador de localização. O contador de localização gerencia o endereço da instrução executada e dos itens de dados durante a montagem, que geralmente é inicializada com `0` (zero). No início do primeiro passo, é incrementado de acordo com o tamanho de cada instrução.

Durante este passo, o montador também efetua quaisquer operações aritméticas em tempo de montagem, e insere as definições de todos os rótulos de funções e variáveis e as constantes, em uma tabela chamada a Tabela de Símbolos.

A razão principal para exigir uma segunda passagem é permitir que símbolos sejam usados no programa antes de serem definidos. Após a primeira passagem, o montador terá identificado todos os símbolos e os colocado na Tabela de Símbolos, já durante a segunda passagem, gerará código de máquina, inserindo os identificadores dos símbolos que agora são conhecidos.

Ligação e Carregamento

A maioria dos programas é composto de mais de um procedimento. Os compiladores e os montadores geralmente traduzem um procedimento de cada vez, colocando a saída da tradução em disco. Antes que o programa possa rodar, todos os seus procedimentos precisam ser localizados e ligados uns aos outros de maneira a formarem um único código.

Ligação

A função do ligador é coletar procedimentos traduzidos separadamente e ligá-los uns aos outros para que eles possam executar como uma unidade chamada programa binário executável.

Se o compilador ou o montador lesse um conjunto de procedimentos fonte e produzisse diretamente um programa em linguagem de máquina pronto para ser executado, bastaria que um único comando fonte fosse alterado para que todos os procedimentos fonte tivessem que ser novamente traduzidos.

Usando a técnica do módulo objeto separado, o único procedimento a ser novamente traduzido seria aquele modificado. Havendo a necessidade de realizar apenas a etapa de ligação dos módulos separados novamente, sendo esta tarefa mais rápida que a tradução.

Carregamento

O carregador é um programa que coloca um módulo de carregamento na memória principal. Conceitualmente, a tarefa do carregador não é difícil. Ele deve carregar os vários segmentos de memória com seus valores corretos e inicializar certos registradores, tais como o apontador para pilha do sistema, responsável pelo escopo das rotinas que estarão em execução e o contador de instruções contido no processador, com seus valores iniciais, indicando assim onde o programa deve ser iniciado.

Em Sistemas Operacionais modernos, vários programas estão residentes na memória a todo instante, e não há como o montador ou o ligador saber em quais endereços os módulos de um programa irão residir. O carregador deve relocar estes módulos durante o carregamento adicionando um deslocamento a todos os endereços, permitindo desta forma acessar cada módulo individualmente na memória.

Esse tipo de carregamento é chamado de carregamento com relocação. O carregador simplesmente modifica endereços relocáveis dentro de um único módulo de carregamento para que vários programas passem a residir na memória simultaneamente.

Interpretadores

O software interpretador é um programa de computador que executa instruções escritas em uma linguagem de programação. Por exemplo, as linguagens Basic, Prolog, Python e Java, são frequentemente interpretados. Um interpretador geralmente usa uma das seguintes estratégias para a execução do programa: executar o código fonte diretamente ou traduzir o código fonte em alguma eficiente representação intermediária e depois executar este código.

Para isso, certos tipos de tradutores transformam uma linguagem fonte em uma linguagem simplificada, chamada de código intermediário, que pode ser diretamente "executado" por um programa chamado interpretador. Nós podemos imaginar o código intermediário como uma linguagem de máquina de um computador abstrato projetado para executar o código fonte.

Interpretadores são, em geral, menores que compiladores e facilitam a implementação de construções complexas em linguagens de programação. Entretanto, o tempo de execução de um programa interpretado é geralmente maior que o tempo de execução deste mesmo programa compilado, pois o interpretador deve analisar cada declaração no programa a cada vez que é executado e depois executar a ação desejada, enquanto que o código compilado apenas executa a ação dentro de um contexto fixo, anteriormente determinado pela compilação. Este tempo no processo de análise é conhecido como "overhead interpretativa".

Ver também

- Estrutura de dados
- Autômato
- Complexidade computacional
- Teoria da computabilidade
- Teoria da computação
- Garbage in, garbage out
- Máquina abstrata
- Negociações de alta frequência
- Algoritmo probabilístico
- Algoritmo de Euclides

Referências

1. Cruz, Adriano Joaquim de Oliveira (1 de janeiro de 1997) «Algoritmos» (http://equipe.nce.ufrj.br/adriano/c/apostila/algoritmos.htm). Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro. Consultado em 11 de março de 2017.

2. Linder, Marcelo Santos. «Programação para Computação» (http://www.univasf.edu.br/~marcelo.linder/arquivos_pc/aulas/aula1.pdf) (PDF). Universidade Federal do Vale de São Francisco. Consultado em 12 de janeiro de 2012.
3. Nuno Miguel da Conceição Fernandes Verdasca (janeiro de 2013). *Identificação e Análise de Movimento Humano com Ultrassons* (http://repositorio.ipl.pt/bitstream/10400.21/2540/1/Disserta%C3%A7%C3%A3o.pdf) (PDF) (Dissertação de mestrado) Consultado em 12 de Março de 2015.
4. Loureiro, Computação da UFOP. «Análise de complexidade» (http://www.decom.ufop.br/menotti/paa101/slides/aula-AnaliseAlgoritmos.pdf) (PDF). Consultado em 12 de janeiro de 2012.
5. TAVARES, P. de Campos; **Algoritmo**, *in* "Enciclopédia Verbo Luso-Brasileira da Cultura, Edição Século XXI", Volume II, Editorial Verbo, Braga, Janeiro de 1998 ISBN 972-22-1864-6
6. algoritmos e mercados (http://expresso.sapo.pt/algoritmos-assaltam-mercados-de-icommoditiesi-depois-das-bolsas=f714175) Acedido em 20 de julho 2012
7. Spitznagel, Edward L. (1971). *Selected topics in mathematics* (https://www.worldcat.org/oclc/130674) New York: Holt, Rinehart and Winston. pp. p. 137. ISBN 0030846935. OCLC 130674 (https://www.worldcat.org/oclc/130674)
8. [(http://www.apoioinformatica.inf.br/produtos/visualg) «VisuAlg»] Verifique valor |url= (ajuda)

Bibliografia

- Algoritmos e Programação - Teoria e Prática: para universitários e profissionais de informática Novatec Editora. ISBN 85-7522-073-X

1. Donald E. Knuth (1973) The Art of Computer Programming, Volume 1: Fundamental Algorithms (2ª edição). Addison-Wesley, ISBN 0-201-03809-9 (em inglês)
2. ↑ CHARLES E. LEISERSON, Thomas H. Cormen, RONALD L. RIVEST, CLIFFORD STEIN , Algoritmos: *teoria e prática* , CAMPUS - RJ, 2002 ISBN 8-535-20926-3
3. Donald E. Knuth, *Selected Papers on Analysis of Algorithms* ISBN 1-57586-212-3 Livro (em inglês)
4. ↑ Jean Luc Chabert, *A History of Algorithms: From the Pebble to the Microchip* Springer Verlag, 1999. ISBN 978-3-540-63369-3. (em inglês)
5. ↑ *Algorithmics: The Spirit of Computing* Addison-Wesley. 2004. ISBN 978-0-321-11784-7. (em inglês)
6. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2009 ISBN 978-026-203-384-8 (em inglês)
7. Donald E. Knuth, The Art of Computer Programming
8. ↑ Jon Kleinberg, Éva Tardos, *Algorithm Design*, Addison-Wesley, 2005 ISBN 0-321-29535-8 (em inglês) Livro
9. Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani, Algorithms
10. Richard E. Neapolitan, Kumarss Naimipour , *Foundations of Algorithms Using Java Pseudocode*, Jones & Bartlett Learning, 2004 ISBN 0-763-72129-8 (em inglês)
11. ↑ Laira Vieira Toscani, Paulo A. S. Veloso, *Complexidade de Algoritmos* Série Livros Didáticos Informática UFRGS - Vol. 13 , Bookman ISBN 8-540-70139-1
12. RICARDO LINDEN , *Algoritmos Genéticos* (2a edição) , Brasport ISBN 8-574-52373-9
13. ↑ Jesús Bisbal Riera, *Manual de Algorítmica Recursividad, complejidad y diseño de algoritmos*, Editorial UOC, 2009 ISBN 8-497-88027-7 (em castelhano)
14. Gilberto Farias de Sousa Filho, Eduardo de Santana Medeiros Alexandre, *Introdução a Computação* - 2ª edição. João Pessoa: Editora da UFPB, 2014. ISBN:978-85-237-0892-4

Ligações externas

- O que é algoritmo? (em português)
- Dictionary of Algorithms and Data Structures
- Aprenda a Programar: Série de artigos didáticos ensinando Português Estruturado
- Aplicativo .jar para testes de Algoritmo (www.ucb.br)
- Exercícios resolvidos de algoritmos para estudo
- Site para aprendizado e treinamento de algoritmos e estrutura de dados
- Data Structures and Algorithms: Godfried Toussaint na Universidade McGill, Montréal
- animação de alguns algoritmos (baseado no livro Algorithms in C++ de Robert Sedgewick) (em inglês)
- The Algorithm Design Manual, 2nd Edition
- Projeto de Algoritmos em C
- Análise de Algoritmos

- Minicurso de Análise de Algoritmos
- Dicionário
- A. Broder, J. Stolfi, Pessimal algorithms and simplexity analysis

Compressão de dados
Teoria
Com perda · Sem perda
Tipo de dados de origem
áudio · banda · imagens · vídeo
Métodos
Lista de algoritmos - Algoritmos de Compressão
