Ciência da Computação

Ciência que estuda técnicas, metodologias e instrumentos computacionais, bem como soluções algorítmicas de problemas, projeto, desenvolvimento e programação de computadores, automação de processos e tarefas; processamento, armazenamento e transferência de dados.

Sobre computadores

- Não sabem o que fazem;
- Não "pensam" duas vezes antes de executar uma tarefa;
- Não fazem aquilo que você deseja, mas sim o que você manda;

Algoritmo

Sequência de passos ou instruções bem definidas para a execução de uma tarefa

➔ O algoritmo é como uma receita de bolo;

➔ O conceito de "instrução bem definida" é subjetivo e depende de quem vai executar.

Exemplo de algoritmo

Cálculo de raízes de equação de 2º grau ($ax+bx+c=0$)

Entrada: a, b, c;
Saída: raízes reais r' e r", ou declaração de inexistência de raízes reais.
Passo 1: Calcular Δ segundo a expressão:
Δ : = b^2-4ac
Passo 2: se $\Delta<0$:
Passo 2.1.: Declare inexistência de raízes reais
Passo 2.2.: Pare o algoritmo
Passo 3:Calcular r' segundo a expressão:
r' : =$(-b+\sqrt{\Delta})/2$
Passo 4:Calcular r" segundo a expressão:
r' : =$(-b-\sqrt{\Delta})/2$

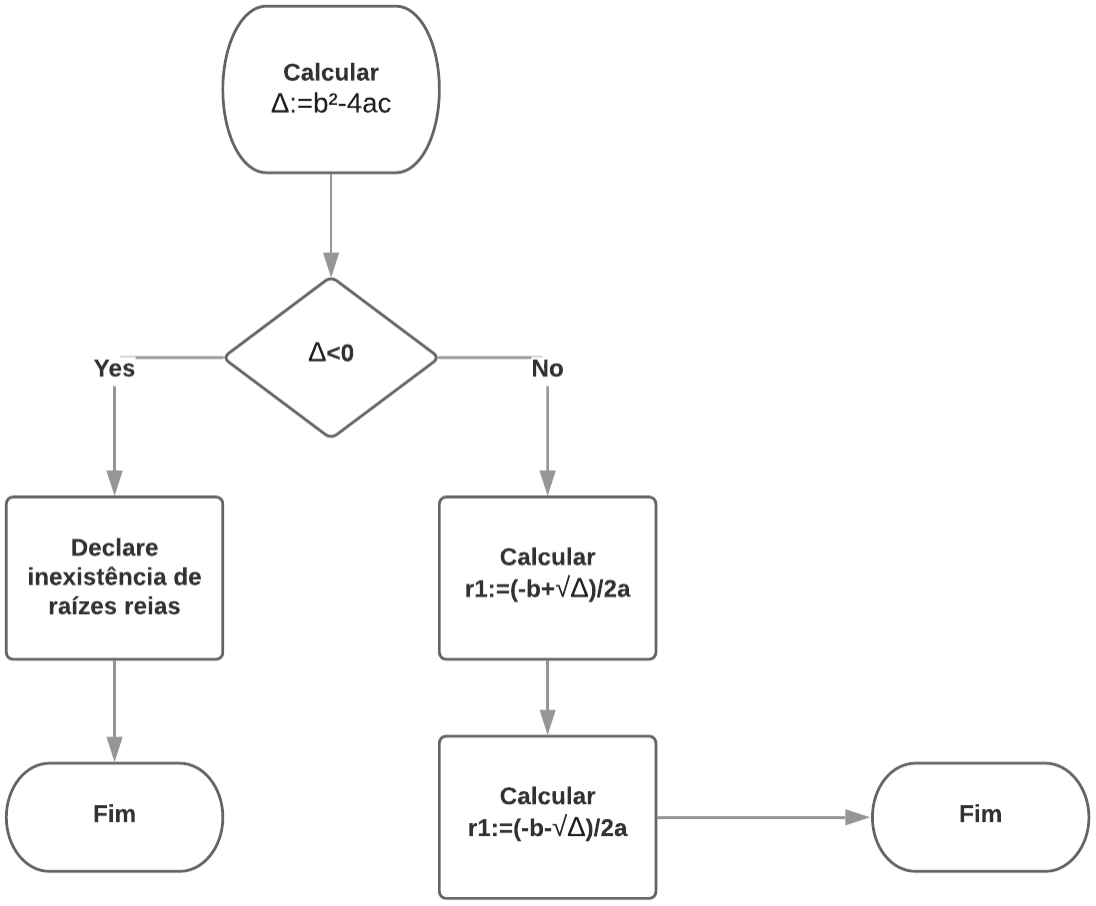
Fim do algoritmo

No contexto computacional, todo algoritmo recebe dados de entrada e produz algum tipo de resultado (saída)
Com um algoritmo em mãos, não é necessário a inteligência para execução de uma tarefa, além da capacidade de seguir a risca as instruções, pois a inteligência necessária está "embutida" no algoritmo.
Dessa forma, através de um algoritmo, é possível utilizar uma máquina sem inteligência para a resolução de problemas, sendo preciso então "traduzir" o

algoritmo em instruções que possam ser reconhecidas e executadas pela máquina

A forma como apresentamos o algoritmo para encontrar raízes de equação de 2º grau é denominada pseudo-código.
Em alternativa, podemos apresentar algoritmos usando fluxogramas.

Ex.: Fluxograma do algoritmo das raízes.



Elementos básicos de algoritmos computacionais:

- Parâmetros (argumentos)
  - De entrada: elementos necessários para a execução do algoritmo (a,b,c no exemplo acima);
  - De saída: a resposta, resultado final produzido pelo algoritmo (r1,r2 e declaração de inexistência de raízes no nosso exemplo);
- Variável: elemento utilizado para armazenar (ou se referir a) resultados intermediários obtidos pelo algoritmo (Δ, r1,r2,a,b,c no nosso exemplo)
- Atribuição: armazenamento de um resultado presente na direita em uma variável que esteja a esquerda. sempre em uma de três formas:
  - <- (seta para onde será atribuído)
    - Ex.: Δ<-b²-4a

    - := (dois pontos, igual)
      Ex.: $\Delta := b^2-4a$
    - = (igual)
        - Ex.: $\Delta = b^2-4ac$
- Condicionais: instruções de desvio. Permitem executar um bloco de instruções se uma determinada condição for verdadeira ou falsa);
- Cláusula de repetição: permitem repetir um bloco de instruções por um determinado número de vezes,ou enquanto uma determinada condição for verdadeira
    - Ex.: Algoritmo para somar dois vetores a e b;
        - c = a(-2,0,1)+(3,4,5)=(1,4,6)
          Entrada: vetores a e b de n elementos
          Saída: vetor c, tal que c=a+b
            I. Seja d, o d-iésimo elemento do vetor d;
            II. Para todo k e {1,2,3,...,n-1,k}
            III. $c_n=a_n+b_n$

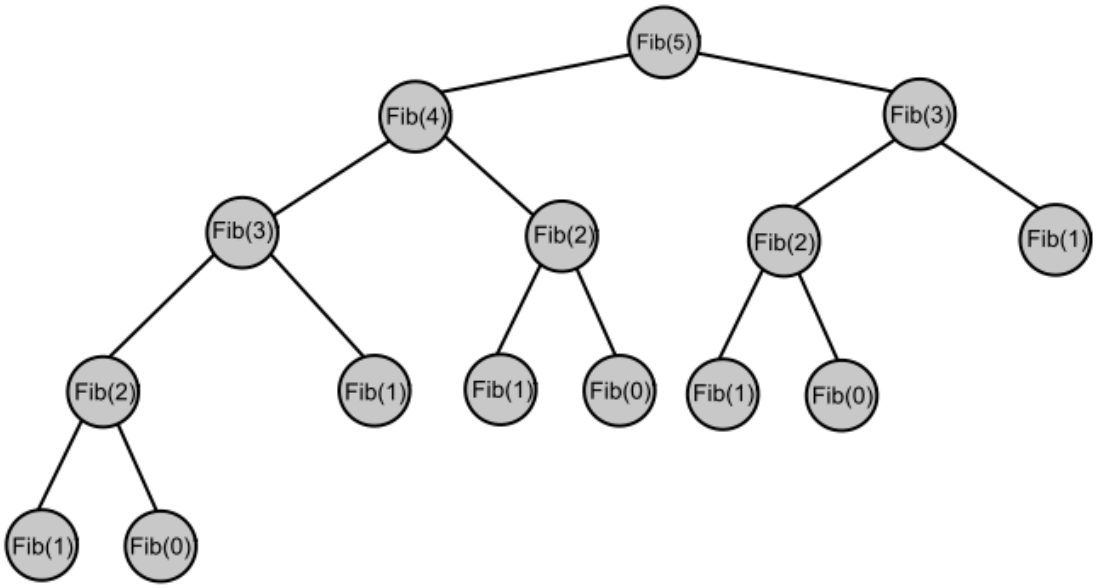
Recursão

_____Algo que use a si mesmo para ser definido. Um algoritmo recursivo utiliza ele mesmo como passo, algoritmos recursivos costumam ser implementados através de funções(recursão com ordem maior que 1(número de casos bases) a recursividade se torna extremamente pesada pois muitas contas são repetidas(exemplo abaixo)).
Exemplo (fatorial):

- Procedimento: fatorial(n);
- Entrada: número natural n;
- Saída: fatn (n!);

1. Se n=0 ou n=1, então;
2. {
3. fatn = 1;
4. }
5. senão
6. {
7. fatn=(n*fatorial(n-1));
8. }
9. retornar fatn;
10. /*a instrução “retornar” serve para enfatizar o que o algoritmo produz como saída também está implícito que o algoritmo encerra sua execução quando esta instrução é executada!*/

Exemplo fibonacci recursivo:



EXERCÍCIOS:

1. Escreva um algoritmo para calcular o comprimento de um vetor de n elementos.
2. Implemente o algoritmo acima em alguma linguagem.
3. Escreva dois algoritmos para calcular o enésimo número da sequência de fibonacci, um recursivo e outro não recursivo, implementar a versão não recursiva em alguma linguagem.

Sistema de Numeração

- Base: número de dígitos diferentes de um sistema de numeração
    - Binário: 0 | 1
    - Octal: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
    - Decimal: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
    - Hexadecimal: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F
- Em sistemas de numeração diferentes do decimal, representam-se assim:
    - 11010(binário)=32(octal)=26(decimal)=1A(hexadecimal)
        - Quanto maior a base, mais é possível representar com a mesma quantidade de dígitos
- Conversão de bases
-

Modelo de Computador

- Unidade de processamento (CPU)
    - Realiza operações lógicas, aritméticas e de comparação;
- Memória principal
    - Espaço de trabalho do processador (temporário) acesso aleatório.

Periféricos

- Entrada/saída de dados;

Hierarquia de Memória

1. Registradores (células de armazenamento interno do processador)
    a. Processos manipulados pelo processador precisam ser carregados para um de seus registradores
2. Memória cache (memória interna do processador)
3. Memória principal (RAM)
4. Memória secundária (Memória de Armazenamento Massivo)

Linguagens de programação

- Facilitam a escrita de programas através de conceitos e um conjunto de expressões válidas que é traduzido para linguagem de máquina.
    - *Linguagens compiladas*
        - Modelo:
          código fonte ======>compilador======>Programa .exe
          entrada saída
        - Um algoritmo é escrito em um ou mais arquivos-fonte (códigos fonte) usando expressões válidas.
          O código fonte é submetido a um programa especial chamado compilador que lê as instruções e as traduz para linguagem de máquina gerando um programa executável. Em geral essa tradução é feita de forma bastante otimizada.
        - Ex.: C, C++, Pascal, Fortran
        - Vantagem: é fácil e rápido de executar o arquivo quando já presente na memória.
    - *Linguagens interpretadas*
        - Modelo:
          código fonte ======> interpretador
          entrada
        - Nesse caso, o interpretador recebe como entrada o código fonte com as instruções e as executa ele próprio linha por linha(e caso haja uma linha errada a linha é ‘pulada’.
          A princípio, o programa interpretado precisa, a cada execução
            1. Traduzir as instruções dos arquivos fonte para linguagem de máquina
            2. Executar essas instruções
        - Ex.:Python, PHP, JavaScript, Mathlab, VisualBasic, R.
        - Vantagens: maior portabilidade (caso o hardware tenha o interpretador, mais fácil decorar, normalmente mais fáceis de

programar (mais alto nível), caso tenha um erro na linha 100, as outras 99 ainda serão interpretadas
        - Desvantagens: Não é gerado um arquivo para executar em outra máquina, sendo necessário enviar o código fonte completo e que o receptor tenha o interpretador instalado
    - *Linguagens híbridas*
        - Modelo
          código fonte =>compilador de código de byte=>código de byte=>Interpretador de código de byte
          entrada saída entrada
        - Nesse esquema, uma espécie de compilador gera um código de byte a partir do arquivo fonte com as instruções. Esse código de byte é então interpretado por um interpretador.
        - Exemplos: Java, Python, C#.

- Linguagens de Programação costumam ser classificadas em gerações
    - 1° Ger: Linguagens de máquina, construções codificadas através de números.
        - Ex.: a instrução de somar os conteúdos do registrador R1 e R2 poderia ser expressa(hipoteticamente) como 35812
        - Muito propenso a erros
        - Precisa manipular endereços de memória diretamente
        - Depende da máquina que executará o programa(baixa portabilidade)

    - 2º Ger:Linguagens de montagem
        - Associa uma sigla mininemônica a cada instrução da linguagem de máquina
        - Ex:35812 ↔ ADD R1,R2
        - os programas ficaram mais “simples” de serem compreendidos
        - Ainda dependente da máquina

    - 3° Ger: Linguagens como Fortran, Algol, C…
        - Primitivas em alto nível. (não se manipula registradores nem endereços de memórias diretamente)
        - Surgiram compiladores e interpretadores para as linguagens

<u>Paradigmas de Programação</u>

- Definem metodologias para o desenvolvimento de software

    - Paradigma imperativo ou procedural
        - Abordagem tradicional onde os programas seguem uma sequência de comandos.

        - Ex.: C, Pascal, Fortran

    - Paradigma declarativo (Lógico)
        - Se descreve o problema a ser solucionado em vez do algoritmo a ser seguido.
        - As linguagens declarativas precisam então implementar um algoritmo para resolução de problemas.
        - Ex.: prolog

    - Paradigma Funcional
        - Programas saem desenvolvidos através do uso de funções matemáticas que podem ser extensamente combinadas entre si.
        - Fortemente baseado em cálculo-lambda, no uso de listas de dados (cargas) e recursão.
        - Ex.: Haskell, LISP, Scheme

    - Paradigma Orientado a Objetos
        - Programas são construídos a partir de objetos
        - Cada objeto é visto como um exemplar/representante de uma classe. A classe por sua vez, modela de forma genérica, uma entidade do mundo real.
        - Cada classe descreve um conjunto de propriedades (atributos) que seus exemplares devem possuir, bem como um conjunto de funções (métodos) que podem ser aplicados a qualquer exemplar da classe.
        - Essas funções são definidas através do paradigma imperativo.
        - Ex.: JAVA, C++, Python, Delph, C#.

Funções

- No sentido mais matemático, é um mapeamento entre um conjunto de possíveis valores de entrada e um conjunto de valores de saída, de forma que cada possível entrada seja associada a uma única saída;

  x---->|f|---->f(x)

- O processo de computar uma função consiste em determinar o valor da saída correspondente a uma determinada entrada
- Funções cujos valores de saída podem ser determinados algoritmicamente são ditas computáveis
- Infelizmente, existem funções que não computam, e assim, estão além das capacidades dos computadores.
  O estudo da computabilidade de funções é importante na ciência da computação pois

    - Ao identificar as capacidades mínimas que permitem que uma máquina compute qualquer função computável, podemos nos assegurar que qualquer máquinas que tenha essas capacidades pode ser usada, em tese, na resolução de problemas computáveis.
    - O estudo das funções computáveis, é, em sua essência o estudo da aplicabilidade das máquinas para resolução de problemas.
    - Ao determinar que um certo problema está intrinsecamente relacionado a uma função não computável, sabemos então que não podemos usar máquinas na sua resolução.
- A função não computável mais ilustre da ciência da computação está relacionada ao problema conhecido como problema da parada:
    - É possível construir um algoritmo Q que receba:
        i. Um código fonte de um programa P
        ii. Uma entrada X válida para P e seja capaz de dizer para qualquer par P e X válidas, se o programa P, ao receber 3 entradas X, irá encerrar sua execução em tempo infinito (parar), ou entrar em laço infinito, apenas analisado o código de P sem chegar a executá-lo?

            Não é possível construir Q pelo fato do problema ser resolvido por Q ser uma função não computável (função de parada)
            A demonstração se dá por contradição.
            Assuma, por contradição, que existe um programa Q que recebe <P, X> e para com a variável w=1, se o programa P parar com a entrada X, ou para com W=0 se P não para com a entrada X

            A partir do programa Q podemos criara um programa S

            ```
            Q              |
            -----          |
            while (w==1)   |
            {              |                    S
                           |
            }              |
            ```

            S entra em laço infinito quando w=1, ou seja, quando G diz que o algoritmo P vai para, note que S para quando G dizer que P não vai parar.
            Rodamos então o programa S passando ele mesmo como ambas as entradas, isto é, rodamos S a partir de P=S e X=S.

Em outras palavras S (e por consequência Q) será usado para verificar se, o próprio S para ao receber ele mesmo como entrada
Se G concluir que S para quando recebe ele mesmo, então W será 1 e S entrará em laço infinito. Sendo assim, S não irá parar e temos como
Por outro lado, se G concluir que S não para quando receber a sí prpróprio, W será 0 e S irá parar. Logo temos outra contradição, pois G disse que S não pararia. Assim, S não pode não parar ao receber ele mesmo como entrada.
Então a única conclusão possível é que não pode existir!

Máquina de Turing

- Modelo abstrato de computador, proposto por Alan TUring (1936) com o objetivo de compreender as capacidades e limitações das máquinas.
  Toda máquina de Turing computa alguma função computável por meio de uma unidade de controle que pode ler e escrever símbolos em uma fita por meio de uma cabeça de leitura/escrita;
  A fita é dividida em células, em cada célula, há um valor pertencente a um conjunto de símbolos denominado como alfabeto da máquina.
  A computação na máquina se dá através de estados. A máquina sempre está em um dos seus estados previstos.
  Para cada estado dependendo do valor da célula da fita sobre a cabeça de leitura/escrita, a máquina pode re-escrever um valor, mover a cabeça para a célula a direita ou à esquerda na fita e alterar seu estado corrente
- Exemplo: máquina para incrementar uma unidade em um número binário
  - Alfabeto: {0,1,*(símbolo especial para delimitar o dado)}
  - Conjunto de estados:
    - START (início)
    - ADD (adição)
    - CARRY (transporte(vai um))
    - OVERFLOW (transbordamento)
    - RETURN (retorno)
    - HALT (parar)
  - Tabela de ações

	Estado atual	Conteúdo	Valor a	Direção	Novo estado

		da célula atual	escrever	a mover	a entrar
1	START	*	*	Esquerda	ADD
2	ADD	0	1	Direita	RETURN
3	ADD	1	0	Esquerda	CARRY
4	ADD	*	*	Direita	RETURN
5	CARRY	0	1	Direita	RETURN
6	CARRY	1	0	Esquerda	CARRY
7	CARRY	*	1	Esquerda	OVERFLOW
8	OVERFLOW	(ignorado)	*	Direita	RETURN
9	RETURN	0	0	Direita	RETURN
10	RETURN	1	1	Direita	RETURN
11	RETURN	*	*	Não se move	HALT

- Vamos supor que o estado inicial da máquina é: {*,1,0,1,*}
  estado atual: START posição atual ^

- Pela linha 1 da tabela: {*,1,0,1,*}
  estado: ADD posição atual ^

- Pela linha 3 da tabela: {*,1,0,1,*}
  estado: CARRY ^ posição atual

- Pela linha 5 da tabela: {*,1,1,0,*}
  estado: RETURN ^ posição atual

- Pela linha 9 da tabela: {*,1,1,0,*}
  estado: RETURN ^ posição atual

- Pela linha 11 da tabela: {*,1,1,0,*}
  estado: HALT posição atual ^
    - FIM

Representação de números negativos

- Para representar números negativos em computação, usa-se um bit para o sinal (0 para positivo e 1 para negativo)
- Ex. com 8 bits temos;
    - 2^8=256 valores diferentes
    - 49(dec)=0011001(bin)
    - -49(dec)=1011001(bin)
- Se para representar o negativo de um n° x, apenas invertêssemos o bit de sinal, teríamos a desvantagem da dupla representação do zero
    - +0=00000000
    - -0=10000000
- Por essa razão, costuma-se representar números negativos usando complemento a dois, que é definido como o complemento em relação a 2^n, onde N é o número de bits.
- Assim, para calcular o negativos de um n° X por complemento a dois, executa-se os seguintes passos:
    - 1 Calcula-se o complemento a um de X invertendo todos os seus bits
    - 2 Soma-se uma unidade ao complemento de X
    - ex:-49
        - passo 1:49(dec) para bin → 00110001
            - note que é preciso inverter os 8bits
            - 00110001 → 11001110
        - Passo 2: 11001111
            - Soma-se uma unidade
        - -49(dec): 11001111(bin)
    - É preciso lembrar de completar o zeros a esquerda para que ele sempre tenha os N bits pré-determinados.
    - Uma vantagem do complemento a dois é que operações de soma podem ser feitas mentalmente sem “deixar” de manter a consistência
        - 0123!=123
        - ex 49-49=49+(-49)
            - 00110001
            - \+ 11001111
            - 00000000
            - .
        - ex 31-49
            - 31/2
              1 15/2
                1 7/2
                  1 3/2
                    1 ½
                      1 0
            - 31(dec) → 00011111(bin)

- Portanto:
  31-49 =
  00011111
  +11001111

  11101110
- Agora basta desfazer o complemento a dois no resultado obtido:
  11101110
  -1

  11101101
- inverte-se os números:
  11101101 → 00010010(bin) = 18(dec)
- Como o 18 estava em complemento a dois, o resultado é negativo (-18), ou seja, 31-49 = -18

pnc do zang

trabalho icc www.facom.ufu.br/~wendelmelo

p/ dia 20/12(por email(wendelmelo@ufu.br)

individual