Verão 2019 - TÓPICOS DE PROGRAMAÇÃO

Prof. Dr. Leônidas O. Brandão (coordenador)

Profa. Dra. Patrícia Alves Pereira (ministrante)

Prof. Bernardo (Monitor)

Introdução: Algoritmos ...

estão por toda parte em nosso dia-a-dia:

- instruções para o uso de medicamentos;
- instruções para fazer comidas (receitas de culinárias);
- instruções para montagem de móveis;
- instruções para programar aparelhos eletrodomésticos (cafeteira, panela elétrica, micro-ondas, máquina de lavar roupas, ...);
- instruções ...
- instruções ...

Todos são exemplos de algoritmos.

Um algoritmo...

nada mais é do que uma **sequência de ações executáveis** para a obtenção de uma **solução para um determinado tipo de problema**. (Dijkstra)

Um pouco de história ...

Sabe-se que a origem do termo “**algoritmo**” deve-se ao matemático persa **Abu Já’far Mohamed ibn Musa al Khowarizmi**

que é o possível o autor dos primeiros algoritmos conhecidos, ou registrados, na história (em torno do ano 825 D.C.)

Veja... algoritmos existiam há muitos anos antes da construção do primeiro computador na década de 1940.

Voltados, primordialmente à solução de problemas aritméticos.

Vejam ...

algoritmos existiam há muitos anos antes da construção do primeiro computador na década de 1940.

Voltados, primordialmente à solução de problemas aritméticos.

Atualmente temos também algoritmos para solucionar problemas computacionais que não tem natureza numérica.

Mas para os nossos estudos ...

- um problema será uma pergunta de caráter geral a ser respondida.
- normalmente, um problema tem vários parâmetros cujos valores são variáveis
- tem uma descrição geral de todos os seus parâmetros, e um enunciado de quais propriedades, ou solução, que deve satisfazer.

Uma instância de um problema ...

é obtida ao se fixarem valores particulares de todos os parâmetros do problema.

Por exemplo...

para o problema de ordenar (crescente) uma lista finita de ***n*** números inteiros [A1, A2,...,An],

a solução consiste nesses mesmos números em ordem crescente,

que nada mais é do que uma permutação de índices, p:(1,...,n) → (1,...,n),

tal que [Ai(1),Ai(2),...,Ai(n)]

satisfaça Ai(j) ≤ Ai(j+1), para todo j, 1 ≤ j ≤ n-1.

Por exemplo...

Uma instância deste problema poderia ser a lista [15,3,0,-7] onde n = 4, e a solução: [-7,0,3,15].

Outra instância poderia ser a lista [1,-2,13,1,7,5]

onde n = 6, e a solução: [-2,1,1,5,7,13].

Um algoritmo é...

uma descrição passo a passo de como um problema é solucionável.

- com uma descrição finita
- passos bem definidos (sem ambiguidades)
- executáveis computacionalmente.

Diz-se que um algoritmo ...

resolve um problema P se este algoritmo recebe qualquer instância I de P e sempre produz uma solução para esta instância I.

Diz-se, então, que o algoritmo resolve a instância I, e que I é um dado de entrada do algoritmo, e a solução é um dado de saída do algoritmo.

Desse modo, para qualquer dado de entrada I, o algoritmo deverá ser executável em tempo finito e, produzir uma solução correta para o problema P.

Mas quando...

Temos uma descrição passo a passo com todas as propriedades de um algoritmo, exceto unicamente da garantia de que é executável em tempo finito para qualquer instância I.

Então, tal descrição será chamada de procedimento.

Mas quando...

Temos uma descrição passo a passo com todas as propriedades de um algoritmo, exceto unicamente da garantia de que é executável em tempo finito para qualquer instância I.

Então, tal descrição será chamada de procedimento.

Assim ...

Algoritmo é definido como um procedimento (ou função) que consiste de um conjunto de regras não ambíguas que especificam, para cada entrada, uma sequência finita de operações, terminando com uma saída correspondente.

Um algoritmo resolve um problema quando, para qualquer entrada, produz uma resposta correta, se forem concedidos tempo e memória suficientes para sua execução.

Mas ...

O fato de um algoritmo resolver (teoricamente) um problema não significa que seja aceitável na prática.

Os recursos de espaço e tempo requeridos têm grande importância em casos práticos.

Às vezes, o algoritmo mais imediato está longe de se razoável em termos de eficiência.

Analisar um algoritmo …

significa prever os recursos de que o algoritmo necessitará.

como:

- memória,
- velocidade de processamento (tempo de computação)
- largura de banda de comunicação ou hardware de computador

Analisar um algoritmo …

Em geral, pela análise de vários algoritmos candidatos para um problema, pode-se identificar facilmente um algoritmo mais eficiente.

Essa análise pode indicar mais de um candidato viável, mas vários algoritmos de qualidade inferior em geral são descartados no processo.

Com o crescente avanço tecnológico ...

as máquinas estão cada vez mais rápidas, e pode, ingenuamente, parecer ofuscar a importância da complexidade de um algoritmo.

Entretanto, acontece exatamente o inverso, pois as máquinas ao se tornarem mais rápidas, passam a ter capacidade para solucionar problemas maiores e é a complexidade do algoritmo que determina o novo tamanho máximo de problema resolvível.

Complexidade

A primeira medida da dificuldade de um problema se relaciona com o tempo necessário para resolvê-lo.

Para um dado problema, a complexidade de tempo é uma função que relaciona o tamanho de uma entrada (ou instância) ao tempo necessário para resolvê-la.

Resumidamente, queremos saber em quanto tempo podemos resolver todas as instâncias de um problema.

Mas especificamente...

associamos a cada entrada um valor inteiro chamado tamanho que é a medida da quantidade de dados de entrada.

Procuramos então uma função do tamanho da instância, que expresse o tempo que o algoritmo necessita para resolver aquela instância.

Essa função é chamada complexidade de tempo.

Vejamos...

Para um dado problema, podemos ter vários algoritmos para resolvê-lo, cada um deles com uma complexidade diferente.

A menor dessas complexidades é chamada cota superior de complexidade *n* do problema considerado.

Esta cota nos diz que, para instâncias arbitrárias de tamanho *n* , podemos resolver o problema em tempo menor ou igual a cota superior(n) .

Ou seja, …

nós não devemos ficar satisfeitos com um algoritmo de complexidade pessimista maior que a cota superior(n) , pois um outro algoritmo de complexidade pessimista com cota superior(n) já existiria.

Por esta razão, algoritmos são analisados para se determinar a sua complexidade, deixando-se em aberto a possibilidade de se reduzir ainda mais a cota superior(n) , se for descoberto um novo algoritmo cuja complexidade pessimista seja menor do que qualquer algoritmo já conhecido.

Por outro lado, …

podemos estar interessados em saber quão rapidamente podem-se resolver todas as instâncias de um dado problema, independentemente do algoritmo que exista ou que se possa descobrir no futuro.

Isto é, estamos interessados na complexidade inerente ou intrínseca do problema que é chamada cota inferior de complexidade n do problema.

Esta cota nos diz ...

que nenhum algoritmo pode resolver o problema com complexidade pessimista menor do que cota inferior(n) , para entradas arbitrárias de tamanho n .

Por exemplo, para o problema da ordenação de ***n*** números inteiros, demonstra-se que sua cota inferior é uma função proporcional a ***n*lg*n***, ou seja, qualquer algoritmo para resolver este problema necessitará de um tempo maior ou igual ao valor desta função, para entradas de tamanho ***n*** , na situação mais desfavorável.

Enfim ...

cota inferior(n) é o mínimo, sobre todos os algoritmos possíveis da complexidade máxima;

enquanto que cota superior(n) é o mínimo sobre todos os algoritmos existentes, da complexidade máxima.

Aprimorar uma cota superior significa descobrir um algoritmo que tenha uma cota inferior melhor do que a existente, nós nos concentramos em técnicas que nos permitam aumentar a precisão com a qual o mínimo, sobre todos os algoritmos possíveis, pode ser limitado. Esta distinção nos leva às diferenças nas técnicas desenvolvidas em análise de complexidade.

Anexo

Quanto vale S após a execução do algoritmo abaixo,

para n = 0, 1, 2, 3, 4, 5, 6, ... n ?

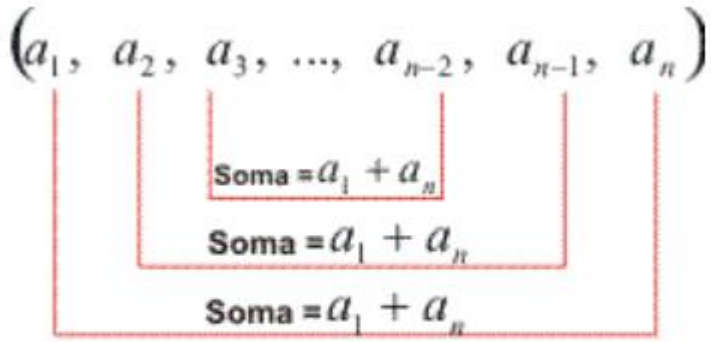
```
S ← 0
para i ← 2 até n-2 faça
        para j ← i até n faça
                S ← S + 1
```

Solução:

Se n ≥ 4, então ao final da execução,

temos

S = 3 + 4 + ...+ (n-3) + (n-2) + (n-1)



$$S_n = (a_1 + a_n)\frac{n}{2}$$

Primeiro termo = 3

Último termo = (n-1)

Número de termos (último – primeiro +1)

= (n-1) -3 +1 = n-3

Solução:

Se n ≥ 4, então ao final da execução, temos

$S = (n-1) + (n-2) + (n-3) + \ldots + 4 + 3 =$

$S = (n+2)(n-3)/2$

$S = n^2/2 - n/2 - 3$

função associada ao tamanho da instância ***n*** .