Problemas, instâncias, algoritmos, tempo

Esta página introduz o conceito de *instância* de um problema computacional e o conceito de *tamanho* de uma instância. Também discute, brevemente, a noção de *consumo de tempo* (ou *desempenho*, ou *eficiência*) de um algoritmo e as ideias de *pior caso* e *melhor caso*.

Uma das principais missões da análise de algoritmos é prever o consumo de tempo de algoritmos. Dado um algoritmo, procuramos estimar, a relação entre o consumo de tempo do algoritmo e o tamanho de sua "entrada".

Problemas e suas instâncias

Todo problema computacional é uma coleção de "casos particulares" que chamaremos *instâncias* (= *instances*). Uma instância é especificada quando atribuímos valores aos parâmetros do problema. Em outras palavras, uma instância é especificada por um particular conjunto de dados do problema. [A palavra *instância* é um neologismo, importado do inglês. Ela está sendo empregada aqui no sentido de *exemplo*, *exemplar*, *espécime*, *amostra*, *ilustração*.] Veja alguns exemplos:

- Problema da multiplicação de números naturais: Dados números naturais u e v, encontrar a expansão decimal do produto $u \times v$. (Veja mais na página *Multiplicação de números*.)

  Cada instância do problema é definida por dois números naturais. Por exemplo, os números 31415926535897932 e 14142135623730950488 definem uma instância.

- Problema da equação inteira do segundo grau: Dados números inteiros a, b e c, encontrar um número inteiro x tal que $ax^2 + bx + c = 0$.

  Cada instância do problema é definida pelos valores de a, b e c. Por exemplo, os números 472, −311 e 57281 definem uma instância do problema; essa instância consiste em encontrar um número inteiro x tal que $472x^2 - 311x + 57281 = 0$.

- Problema da ordenação de vetor inteiro: Rearranjar (ou seja, permutar) os elementos de um vetor $A[1\,..\,n]$ de números inteiros de modo que ele fique crescente. (Veja mais na página *Ordenação por inserção*.)

  Cada instância do problema é definida por um número natural n e um vetor $A[1\,..\,n]$ de inteiros. Por exemplo, o número 5 e o vetor (876, 145, 323, 112, 221) definem uma instância do problema; essa instância consiste em rearranjar o vetor (876, 145, 323, 112, 221) em ordem crescente.

- Problema da subsequência crescente máxima: Encontrar uma subsequência crescente de comprimento máximo de uma sequência $(a_1, a_2, \ldots, a_n)$ de números naturais. (Veja mais na página *Subsequência crescente máxima*.)

  Uma das instâncias do problema consiste em encontrar uma subsequência crescente máxima de (876, 145, 323, 112, 221).

- Problema do ciclo máximo num digrafo: Encontrar um ciclo de comprimento máximo num digrafo.

  Cada instância do problema é especificada por um digrafo. Por exemplo, o digrafo com vértices a, b, c, d e arcos ab, bc, cd, da, bd especifica uma instância do problema.

Em geral, nem toda instância de um problema tem solução. Assim, por exemplo, a instância (1, 2, 3) do problema da equação inteira de segundo grau não tem solução, pois não existe um número inteiro x tal que $1x^2 + 2x + 3 = 0$.

Em discussões informais é razoável confundir "problema" com "instância" e chamar tudo de "problema". Mas em discussões mais técnicas é importante distinguir os dois conceitos.

Exercícios 1

1. Dê uma solução da instância (1, 2, 1) do problema da equação inteira do segundo grau. Dê uma solução da instância (1, ¾, 0) do problema da equação inteira do segundo grau.
2. Dê uma solução da instância (4, −2, 8, 6) do problema da ordenação de vetor inteiro.

Tamanho de uma instância

O *tamanho* (= *size*) de uma instância de um problema é a quantidade de dados necessária para descrever a instância (ou seja, o "espaço de memória" necessário para especificar os valores dos parâmetros que definem a instância). Em geral, o tamanho de uma instância é descrito por um único número natural, mas às vezes é mais conveniente usar dois ou mais números.

O conceito de tamanho está sendo apresentado de maneira propositalmente vaga para que possamos escolher, em cada caso, a definição mais apropriada.

- No problema da multiplicação de números naturais, por exemplo, parece razoável dizer que toda instância têm tamanho 2 (pois toda instância é especificada por dois números). Mas é mais apropriado dizer que o tamanho de uma instância é o número de *caracteres* necessário para especificar os dois números. Sob esta segunda definição, o tamanho da instância mencionada acima é 37.
- No problema da equação inteira do segundo grau, parece razoável dizer que todas as instâncias têm tamanho 3 (pois cada instância é especificada por três números). Dependendo das circunstâncias, entretanto, pode ser mais apropriado dizer que o tamanho de uma instância é o número de caracteres necessário para descrever a instância. Sob esta definição, o tamanho da instância $(472, -311, 57281)$ é 12 (ou, quem sabe, 18).
- O tamanho de uma instância do problema de ordenação é n. Mas poderia também ser definido como o número total de caracteres necessário para escrever os valores dos elementos de $A[1\,..\,n]$.
- O tamanho de uma instância do problema do ciclo máximo é $n+m$, sendo n o número de vértices e m o número de arcos do digrafo. Poderiamos também dizer que o tamanho da instância é o par (n, m).

Exercícios 2

1. Considere o digrafo que tem vértices a, b, c, d e arcos ab, bc, cd, da, bd. Esse digrafo define uma instância do problema do ciclo máximo. Qual o tamanho dessa instância?

Algoritmos para problemas

A solução de uma instância de um dado problema pode ser um número, um vetor, um valor booleano, etc., dependendo da natureza do problema. Já a solução de um *problema* é sempre um *algoritmo*.

Dizemos que um algoritmo *resolve* um dado problema se, ao receber a descrição de *qualquer* instância do problema, devolve uma solução da instância ou informa que a instância não tem solução. Essa exigência é deveras pesada, pois obriga o algoritmo a resolver não só as instâncias que aparecem em aplicações práticas como também aquelas instâncias "patológicas" que nem parecem "razoáveis".

Exercícios 3

1. Dê uma solução do problema da equação inteira do segundo grau.
2. Descreva informalmente uma solução do problema da ordenação de vetor inteiro.

Desempenho de um algoritmo

"Não confunda *consumo de tempo* com *tempo de consumo*!"
— um aluno

Suponha que A é um algoritmo para um certo problema. Seja $t(I)$ a quantidade de tempo que A consome para processar uma instância I do problema. Queremos estudar o comportamento de t em função do tamanho das instâncias. É preciso lembrar que, via de regra, um problema tem muitas instâncias diferentes de um mesmo tamanho e A consome um tempo diferente para cada uma. Para cada n, sejam

$$T_*(n) \ \text{ e } \ T^*(n)$$

o mínimo e o máximo, respectivamente, de $t(I)$ para todas as instâncias I que têm tamanho n. Assim, $T_*(n) \leq t(I) \leq T^*(n)$ para toda instância I de tamanho n. Diremos que as funções T_* e T^* medem o consumo de tempo do algoritmo A *no melhor caso* (= *best case*) e *no pior caso* (= *worst case*), respectivamente. O melhor caso corresponde às instâncias mais "fáceis" e o pior caso corresponde às instâncias mais "difíceis".

Por exemplo, o consumo de tempo do algoritmo pode ser $200n$ no melhor caso e $3n^2$ no pior caso. (Como se faz usualmente, estamos ignorando os valores pequenos de n, para os quais $200n$ é *maior* que $3n^2$.)

Em geral, o consumo de tempo é proporcional ao número operações elementares que o algoritmo executa ao processar a instância. Tipicamente, uma *operação elementar* é uma operação aritmética entre variáveis do algoritmo, ou uma comparação entre duas variáveis, ou uma atribuição de valor a uma variável. Não é necessário contar *todas* as operações elementares: basta escolher uma operação específica de modo o consumo de tempo do algoritmo seja proporcional ao número de execuções dessa operação. No problema da ordenação, por exemplo, parece óbvio que a operação relevante é a comparação entre elementos do vetor. Já no problema da equação inteira do segundo grau, parece claro que todas as operações aritméticas que envolvem a, b e c são relevantes.

Exercícios 4

1. Suponha que n mede o tamanho das instâncias de um certo problema. A documentação de um algoritmo para o problema diz que o algoritmo consome $10^3 n + 10^6$ unidades de tempo no melhor caso e $2n^2/10$ unidades de tempo no pior caso. Isso faz sentido?
2. Determine o maior n para o qual um problema pode ser resolvido em uma unidade de tempo (segundo, minuto, etc.), supondo que o algoritmo gasta $f(n)$ microssegundos.

$f(n)$	seg	min	hora	dia	mês	ano	século
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Consumo de tempo assintótico

"Algorithm analysis usually means 'give a big-*O* figure for the running time of an algorithm'. (Of course, a big-*Theta* bound is even better.)"
— Ian Parberry, *Problems on Algorithms*

"We want a statement about software, not hardware."

Suponha que A é um algoritmo para um problema P. Em termos de minutos e segundos, a quantidade de tempo que A consome para processar uma dada instância de P depende da máquina usada para executar A. Mas essa dependência se resume a uma constante multiplicativa: se A consome tempo t numa determinada máquina, consumirá tempo $2t$ numa máquina duas vezes mais lenta e $t/2$ numa máquina duas vezes mais rápida. Para eliminar a dependência da máquina, basta discutir o consumo de tempo de A *a menos de constantes multiplicativas*. A notação assintótica (O, Ω, Θ) é ideal para fazer isso.

Suponha que T^* e T_* são as funções que medem, em termos do tamanho das instâncias, o consumo de tempo de A no pior e no melhor caso, respectivamente.

- Se $T^*(n)$ está em $O(n^2)$, por exemplo, dizemos que A consome $O(n^2)$ unidades de tempo no pior caso. Como O( ) tem sabor de "≤", a frase "no pior caso" é redundante pois $T_*(n)$ também estará em $O(n^2)$. Portanto, devemos dizer, simplesmente, que A *consome* $O(n^2)$ *unidades de tempo*.
- Se $T^*(n)$ está em $\Omega(n^2)$ dizemos que A consome $\Omega(n^2)$ unidades de tempo *no pior caso*. Como Ω( ) tem sabor de "≥", a frase "no pior caso" não é redundante.

Se $T^*(n)$ está em $O(n^2)$ e em $\Omega(n^2)$, e portanto em $\Theta(n^2)$, podemos dizer que A consome tempo *proporcional a* n^2 no pior caso, embora isso constitua um abuso do significado usual de "proporcional".

- Se $T_*(n)$ está em $\Omega(n)$, por exemplo, dizemos que A consome $\Omega(n)$ unidades de tempo no melhor caso. A frase "no melhor caso" é redundante: devemos dizer, simplesmente, que A *consome* $\Omega(n)$ *unidades de tempo*.
- Se $T_*(n)$ está em $O(n)$, dizemos que A *consome* $O(n)$ *unidades de tempo no melhor caso*. Desta vez, a frase "no melhor caso" não é redundante.

Exercícios 5

1. Suponha que um algoritmo para um certo problema consome $O(n^2)$ unidades de tempo, sendo n o tamanho de uma instância. Meu amigo diz ter um outro algoritmo para o mesmo problema que consome apenas $\Omega(n)$ unidades de tempo. Devo ficar impressionado?
2. Meu algoritmo para um certo problema consome $O(n^2)$ unidades de tempo, sendo n o tamanho de uma instância. Meu amigo diz ter um algoritmo para o mesmo problema que consome $O(n \lg n)$ unidades de tempo. Devo ficar impressionado?
3. Suponha que dois algoritmos, A e B, resolvem um mesmo problema. O tamanho das instâncias do problema é medido por um parâmetro n. Em quais dos seguintes cenários podemos afirmar que A é mais rápido que B? Cenário 1: No pior caso, A consome $O(\lg n)$ unidades de tempo e B consome $O(n^3)$ unidades de tempo. Cenário 2: No pior caso, A consome $O(n^2 \lg n)$ unidades de tempo e B consome $\Omega(n^2)$ unidades de tempo. Cenário 3: No pior caso, A consome $O(n^2)$ unidades de tempo e B consome $\Omega(n^2 \lg n)$ unidades de tempo.
4. ★ Explique por que a seguinte afirmação não faz sentido: "o algoritmo A consome pelo menos $O(n^2)$ unidades de tempo".
5. O algoritmo abaixo recebe um vetor $P[1\,..\,n]$ com $n \geq 0$ e devolve um vetor $X[0\,..\,n]$. Ao receber argumentos P e i, a função TESTE devolve 1 ou 0 e consome $O(i)$ unidades de tempo para dar a resposta. Calcule detalhadamente o consumo de tempo do ALGORITMO no pior e no melhor casos.

```
ALGORITMO (P, n)
1  para k crescendo de 0 até n
2      i := k
3      enquanto i ≥ 1 e TESTE (P, i) = 0
4          i := i − 1
5      X[k] := i
6  devolva X
```

6. Como o consumo de tempo do seguinte fragmento de código varia com n?

```
1  c := 0
2  i := 1
3  enquanto i ≤ n
4      para j crescendo de 1 até n
5          c := c + 1
6      i := 2·i
```

7. Considere o fragmento de código abaixo. Suponha que as linhas 3 e 4 consomem $O(1)$ unidades de tempo (ou seja, consomem tempo constante, independente de n). Suponha que cada chamada AUX(n) consome $O(n^3)$ unidades de tempo. Quanto tempo consome o algoritmo todo?

```
1  a := 0
2  para k := 1 até n²
3      a := a + 1
4      se k ≤ n
5          b := AUX(k)
```

8. ★ Considere os algoritmos ALGOA e ALGOB abaixo. Que números ALGOA e ALGOB devolvem ao receber um número natural n? Calcule o consumo de tempo de cada um dos algoritmos em função de n. (Para $i = 1, 2, 4$, suponha que cada execução da linha i do código consome 1 unidade de tempo. Justifique sua resposta por indução matemática.) Traduza o consumo de tempo de cada algoritmo para notação Θ (diga "Θ(f)" para alguma função f simples) e justifique a tradução.

```
ALGOA (n)
1  se n = 0
2      devolva 0
3  x := 2·ALGOA (n−1) + 4
4  devolva x
```

```
ALGOB (n)
1  se n = 0
2      devolva 0
3  x := ALGOB (n−1) + ALGOB (n−1) + 4
4  devolva x
```

Veja meu *Minicurso de Análise de Algoritmos*: seção 1.2 e seção A.4.

Departamento de Ciência da Computação
Instituto de Matemática e Estatística da USP