Análise de Complexidade

Pior caso, melhor caso, caso médio

Prof. Edson Alves - UnB/FGA

2018

Sumário

Pior caso, melhor caso, caso médio

Definição

- A análise de algoritmos considera três cenários possíveis, os quais relacionam a entrada com o número de iterações do algoritmo
- O pior caso acontece quando o algoritmo é executado o número máximo de iterações
- No melhor caso o algoritmo é executado o número mínimo de iterações possível
- O caso médio representa o cenário esperado quando as entradas possuem determinada distribuição de probabilidade de ocorrência caso
- Em termos formais, a complexidade do caso médio C_M é dada por

  $$C_M = \sum_i p(\text{input}_i)\text{steps}(\text{input}_i)$$

  com $p(\text{input}_i) \geq 0$ e $\sum_i p(\text{input}_i) = 1$
- A fórmula para o caso médio coincide com a definição probabilística de valor esperado

Observações

- O melhor caso tem interesse meramente teórico, não sendo levado em consideração na maior parte das análises
- A maioria das análises se concentram no pior caso, pois ele é uma estimava de como o algoritmo efetivamente vai se comportar
- Embora o caso médio seja mais próximo da realidade, sua análise é mais técnica e depende de conceitos elaborados de matemática e probabilidade
- Além disso, o caso médio tende a ser idêntico ao pior caso no contexto da complexidade assintótica
- A notação mais utilizada é a notação Big-O, seguida pela notação Big-Θ

Exemplos de análise de complexidade assintótica

Exemplo 01

Problema: Determinar a soma dos elementos de um vetor a com n elementos.

Algoritmo:

```
for (i = sum = 0; i < n; i++)
    sum += a[i];
```

Complexidade: 2 atribuições no início, $2n$ no laço. Total de atribuições: $2 + 2n$. Logo o algoritmo é $O(n)$.

Exemplo 02

Problema: Determinar a soma dos subvetores a[0-i] de um vetor a.

Algoritmo:

```
for (i = 0; i < n; i++) {
    for (j = 1, sum = a[0]; j <= i; j++)
        sum += a[j];
    cout << "Soma para o subvetor a[0-" << i << "] = " << soma << endl;
}
```

Complexidade: 1 atribuição no início, $3n$ no laço externo e

$$\sum_{i=1}^{n-1} 2i = n(n-1)$$

no laço interno. Total de atribuições: $1 + 3n + n(n-1)$. O algoritmo é $O(n^2)$.

Exemplo 03

Problema: Determinar a soma dos subvetores de cinco elementos de um vetor a.

Algoritmo:

```
for (i = 4; i < n; i++) {
    for (j = i - 3, sum = a[i - 4]; j <= i; j++)
        sum += a[j];
    cout << "Soma para a[" << i-4 << "-" << i << "] = " << soma << endl;
}
```

Complexidade: 1 atribuição no início, $3(n-4)$ no laço externo e $8(n-4)$ no laço interno. Total de atribuições: $1+11(n-4)$. Logo o algoritmo é $O(n)$.

Exemplo 04

Problema: Determinar o tamanho do maior subvetor ordenado de um vetor a com n elementos.

Algoritmo:

```
for (i = 0, length = 1; i < n - 1; i++) {
    for (i1 = k = i, i2 = i + 1; k < n - 1 && a[k] < a[k+1]; k++, i2++)
         if (length < i2 - i1 + 1)
            length = i2 - i1 + 1;
}
```

Complexidade

Caso 1: Vetor em ordem decrescente. 2 atribuições no início, $4(n-1)$ no laço externo e 0 no laço interno. Total de atribuições: $2+4(n-1)$. O algoritmo é $O(n)$.

Exemplo 04

Caso 2: Vetor em ordem crescente. 2 atribuições no início, $4(n-1)$ no laço externo e

$$(n-1) + \sum_{i=0}^{n-2} [2(n-1-i)] = (n-1) + (n-1)(n-2) = (n-1)^2$$

no laço interno. Total de atribuições:

$$2 + 4(n-1) + (n-1)^2 = 2 + (n-1)(n+3)$$

O algoritmo é $O(n^2)$.

Como determinar o caso médio?

Exemplo 05

Problema: Realizar uma busca binária de um elemento k um vetor ordenado de inteiros a de tamanho n.

Algoritmo:

```
int lo = 0, mid, hi = n - 1;

while (lo <= hi) {
    mid = lo + (hi - lo)/2;

    if (key < a[mid])
        hi = mid - 1;
    else if (key > a[mid])
        lo = mid + 1;
    else return mid;
}

return -1;
```

Exemplo 05

Complexidade

Caso 1: Chave no meio do vetor. 2 atribuições no início, 1 no laço. Total de atribuições: 3. O algoritmo é $O(1)$.

Caso 2: Chave não está no vetor. 2 atribuições no início, laço executado m vezes, onde $n/2^m = 1$. Total de atribuições: $2 + 2\log n$. O algoritmo é $O(\log n)$.

Caso médio?

Exemplo 06

Problema: Busca sequêncial de um elemento k em um vetor a de tamanho n

Algoritmo:

```
for (i = 0; i < n; i++) {
    if (a[i] == k)
        return i;
}

return -1;
```

Complexidade

Melhor caso: k é o primeiro termo. 1 atribuição no início. Total de atribuições: 1. O algoritmo é $O(1)$.

Pior caso: k não se encontra no vetor. 1 atribuição no início, n no laço. Total de atribuições: $1 + n$. O algoritmo é $O(n)$.

Exemplo 06

Caso médio: k *está em alguma das* n *posições ou não se encontra no vetor.* Considere a distribuição de probabilidade uniforme $p(\mathsf{input}_i) = 1/(n+1)$.

Então

$$\begin{aligned} C_M &= \sum_{i=0}^{n} p(\mathsf{input}_i)\mathsf{steps}(\mathsf{input}_i) \\ &= \frac{1}{n+1}\left[\sum_{i=0}^{n-1}(i+1) + (n+1)\right] \\ &= \frac{1}{n+1}\left[\frac{n(n+1)}{2} + (n+1)\right] = \frac{n+2}{2} \end{aligned}$$

Logo, no caso médio, o algoritmo é $O(n)$.

Referências

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.