Complexidade Computacional

Introdução

Pretende-se com esta página dar uma introdução a todos os interessados sobre Complexidade respondendo a certas questões chave tais como:

- O que é a Complexidade?
- Porquê o estudo da Complexidade?
- Tipos de Complexidade.
- Upper e Lower Bound.

O que é a Complexidade?

A Complexidade de um algoritmo consiste na quantidade de " trabalho " necessária para a sua execução, expressa em função das operações fundamentais, as quais variam de acordo com o algoritmo, e em função do volume de dados.

Porquê o estudo da Complexidade?

Muitas vezes as pessoas quando começam a estudar algoritmos perguntam-se qual a necessidade de desenvolver novos algoritmos para problemas que ja têm solução. A performance é extremamente importante na informática pelo que existe uma necessidade constante de melhorar os algoritmos. Apesar de parecer contraditório, com o aumento da velocidade dos computadores, torna-se cada vez mais importante desenvolver algoritmos mais eficientes, devido ao aumento constante do "tamanho" do problemas a serem resolvidos.

Devido a este factor surge a Complexidade Computacional, pois e através dela que se torna possivel determinar se a implementação de determinado algoritmo é viavel.

A importância da complexidade pode ser observada no exemplo abaixo, que mostra 5 algoritmos A_1 a A_5 para resolver um mesmo problema, de complexidades diferentes. Supomos que uma operação leva 1 ms para ser efectuada. A tabela seguinte dá o tempo necessário por cada um dos algoritmos.

$T_k(n)$ é a complexidade do algoritmo.

n	**A_1** $T1(n)= n$	**A_2** $T_2(n)=n\log n$	**A_3** $T_3(n)=n^2$	**A_4** $T_4(n)=n^3$	**A_5** $T_5(n)=2^n$
16	0.016s	0.064s	0.256s	4s	1m4s
32	0.032s	0.16s	1s	33s	46 Dias
512	0.512s	9s	4m22s	1 Dia 13h	10^{137} Séculos

Tipos de Complexidade.

Espacial - Este tipo de complexidade representa o espaço de memória usado para executar o algoritmo, por exemplo.

Temporal - Este tipo de complexidade é o mais usado podendo dividir-se em dois grupos:

Tempo (real) necessário à execução do algoritmo.

Número de instruções necessárias à execução.

Para o estudo da complexide são usados três perspectivas :

- Caso Pior
- Caso Melhor
- Caso Médio

Caso Pior

Este método é normalmente representado por **O ( )**, por exemplo se dissermos que um determinado algoritmo é representado por g(x) e a sua complexidade Caso Pior é n, será representada por g(x) = **O**(n).

Consiste basicamente em assumir o pior dos casos que podem acontecer, sendo muito usado

e sendo normalmente o mais fácil de determinar.

Exemplos:

Se por exemplo existirem cinco baús sendo que apenas um deles tem algo dentro e os outros estão vazios a complexidade caso pior será O(5) pois eu no pior dos casos acerto no baú cheio á quinta tentativa.

Caso Melhor

Representa-se por Ω ( )

Método que consiste em assumir que vai aconter o melhor. Pouco usado. Tem aplicação em poucos casos.

Exemplos:

Se tivermos uma lista de numeros e quisermos encontrar algum deles assume-se que a complexidade caso melhor é Ω (1) pois assume-se que o numero estaria logo na cabeça da lista.

Caso Médio

Representa-se por θ().

Este método é dos três o mais dificil de determinar pois necessita de análise estatistica e como tal muitos testes. No entanto é muito usado pois é também o que representa mais correctamente a complexidade do algoritmo.

Upper Bound

Seja dado um problema, por exemplo, multiplicação de duas matrizes quadradas de ordem n (n*n). Conhecemos um algoritmo para resolver este problema(pelo método trivial) de complexidade $O(n^3)$. Sabemos assim que a complexidade deste problema não deve superar $O(n^3)$, uma vez que existe um algoritmo desta complexidade que o resolve. Um limite superior (upper bound) deste problema é $O(n^3)$. O limite superior de um algoritmo pode mudar se alguém descobrir um algoritmo melhor. Isso de facto aconteceu com o algoritmo de Strassen que é de $O(n^{\log 7})$. Assim o limite superior do problema de multiplicação de matrizes passou a ser $O(n^{\log 7})$. Outros pesquisadores melhoraram ainda este resultado. Actualmente o melhor resultado é o de Coppersmith e Winograd de $O(n^{2.376})$.

O limite superior de um algoritmo é parecido com o record mundial de uma modalidade de atletismo. Ela é estabelecida pelo melhor atleta (algoritmo) do momento. Assim como o record mundial o limite superior pode ser melhorado por um algoritmo (atleta) mais veloz.

Lower Bound

Às vezes é possivel demonstrar que para um dado problema, qualquer que seja o algorimo a ser usado o problema requer pelo menos um certo número de operações. Essa complexidade

é chamada Limite inferior (Lower Bound). Veja que o limite inferior dependo do problema mas não do particular algoritmo. Usamos a letra Ω em lugar de O para denotar um limite inferior.

Para o problema de multiplicação de matrizes de ordem *n,* apenas para ler os elementos das duas matrizes de entrada leva $O(n^2)$. Assim uma cota inferior trivial é $\Omega(n^2)$.

Na analogia anterior, um limite inferior de uma modalidade de atletismo não dependeria mais do atleta. Seria algum tempo minimo que a modalidade exige, qualquer que seja o atleta. Um limite inferior trivial para os 100 metros seria o tempo que a velocidade da luz leva a percorrer 100 metros no vácuo.

Se um algoritmo tem uma complexidade que é igual ao limite inferior do problema então o algoritmo é óptimo.

O algoritmo de CopperSmith e Winograd é de $O(n^{2.376})$ mas o limite inferior é de $\Omega(n^2)$. Portante não é óptimo. Pode ser que este limite superior possa ainda ser melhorado.

Exemplo De Cálculo de Complexidade:

Complexidade de um algoritmo de busca

```
Algoritmo Busca( elemento:elemento, lista:lista):
        Se ListaVazia(lista)
          então  Falso          // Instrucão 1
       Senão
            Se Cabeça(lista) = Elemento
              então Verdadeiro            // Instrução 2
            Senão
                Busca:= Busca (Elemento , Resto (lista))      // Instrução Recursiva
```

Vamos chamar a este Algoritmo β e à sua complexidade O (β).

Podemos chamar a instruçao 1 e 2 f(x) e à instrução recursiva g(x).

A complexidade do algoritmo pode-se calcular:

$O(\beta) = O(f(x)) * O(g(x))$

$O(\beta) = O(2 * n)$

$O(\beta) = O(2n)$

$\Omega(\beta) = O(n)$

Links Relacionados:

www.mat.uc.pt/~rosalia/AED/tp.recomp.html

www.cs.hmc.edu/~keller/courses/cs60/slides/Complexity.html

www.dca.fee.unicamp.br/~ting/Courses/ea869/faq1.html#item2

www.inf.ufrgs.br/pos/SemanaAcademica/Semana2000/MarcoBarbosa/

Página Realizada por:

Gonçalo Madeira nº 12874

email: goncalo.p.madeira@clix.pt

Página optimizada para:
1152x864