PROGRAMAÇÃO E ALGORITMOS (LEII)

Universidade da Beira Interior, Departamento de Informática
Hugo Pedro Proença, 2016/2017

Resumo

- Complexidade Computacional
  - Problemas
  - Instâncias
  - Algoritmos
    - Número de Passos Necessários
  - Notação “Big-O”

Complexidade Computacional

- O termo Complexidade Computacional envolve a classificação de **problemas**, de acordo com a sua inerente tratabilidade, isto é, em relação à "facilidade" com que são resolvidos.
- Por outro lado, é também referente à forma de como um **algoritmo** resolve **instâncias** de 1 problema.
  - A complexidade computacional de 1 algoritmo é uma medida do total de passos necessários (na pior das hipóteses) para resolver 1 instância de 1 problema de determinado tamanho.
  - O número de passos é medido em função desse tamanho.

Problemas, Algoritmos e Instâncias

- Um **problema** é uma descrição em abstracto de algo que necessita resposta.
  - Por exemplo: Dado um grafo com "n" vértices a "k" arestas, qual o caminho de custo mínimo que passa por todos os vértices apenas 1 vez?
- Uma **instância** será um caso concreto do problema em questão. Por exemplo: Seja G(V,A) um grafo, com V={1,2 3 4}, A={(1,2), (1,4), (2,3), (3,4), (2,4)}. Qual o caminho de custo mínimo que passa por todos os vértices apenas 1 vez?

Problemas, Algoritmos e Instâncias

- Um **algoritmo** para um dado problema é um conjunto de instruções que garantidamente encontram uma solução correcta para qualquer instância do problema num número finito de passos.
- No domínio das ciências da computação, a complexidade computacional é modelada pela noção de máquina de Turing.
- Um modelo introdutório (mais prático) consiste na noção de "**passo**". Um passo consiste numa das seguintes operações: adição, subtracção, multiplicação, divisão, comparação e atribuição.

Notação "Big-O"

- Ao analisar a tratabilidade de um problema estamos particularmente interessados na análise **assintótica**
  - De que forma o número de passos necessários varia quando a instância do problema fica muito grande?
- **Notação "Bio-O"**
  - Para 2 funções f(t) e $g(t)$ de um parâmetro "t" não negativo, diz-se que f(t)=**O(g(t))** se existir uma constante c>0 tal que, para "T" suficientemente grande, $f(t) \leq cg(t)$.
- A função $cg(t)$ é uma assíntota superior de f.
  - Por exemplo, $100(t^2 + t) = O(t^2)$, uma vez que com c = 101 a relação é satisfeita para t>=100.
  - No entanto, $0.0001\ t^3$ não tem limite assintótico $O(t^2)$. Porquê?

Pior caso, Limite assintótico

- Tal como verificado, o número de passos varia em função do tamanho do input e também dos seus elementos.
- Assim, para cada input, éxistirá potencialmente um número de passos diferentes.
- Desta forma, é comum estarmos interessados no **pior caso**, isto é, no cálculo de um assíntota superior para o número de passos necessários à resolução do problema.
- Mas, qual o pior dos casos? É claro que podemos sempre aumentar o tamanho do input, por forma a que o tempo de execução aumente.
- As questões fundamentais são:
  - Qual o pior caso, para 1 input de tamanho "n"?
  - **Quão rápido** o tempo de execução **cresce**, ao **aumentarmos** o tamanho do input?

Pior Caso: Exemplo

- Considere o seguinte algoritmo

```
void funcao(int *A, int N){
        for (int i=0; i<N; i++)
                for (int j=i+1; j<N; j++)
                        if (A[i] > A[j])
                                swap( A[i], A[j] );
}
```

- Quantos passos são executados, para as seguintes instâncias de problema?
  - A=[1 2 5 8];
  - A=[8 5 2 1];
  - A=[2 6 3 8 3 4 1 10]

Notação "Big-O"

- Diz-se que um algoritmo executa **em tempo polinomial**, se este tiver tempo de execução $f(t) = O(P(t))$, onde $P(t)$ é uma função polinomial do tamanho do input.
- Os algoritmos que executam em tempo polinomial são geral e formalmente considerados eficientes.
- Considere-se o exemplo da função anterior. Qual o pior caso, para um input de tamanho "N"?
  - A primeira linha é executada N vezes
  - A 2ª linha é executada $N(N - 1)/2$ vezes
  - A 3ª linha é também executada $N(N - 1)/2$ vezes
  - Se os elementos do vector estiverem em ordem descrescente a função swap vai sempre ser executada ($N(N - 1)/2$ vezes).
  - Assim, neste caso, o algoritmo vai fazer $3N(N - 1)/2 + N = 1.5N^2 - 0.5N$ passos.

Pré-Processador: Condições

- Assim, o algoritmo do exemplo apresentado tem $f(N) = 1.5N^2 - 0.5N$.
- É usual retirar os termos sem grande expressão no tempo de execução final. Neste caso, $-0.5N$ pode ser retirado, uma vez que será tendencialmente muito menor que o termo $1.5N^2$ e não afecta significativamente o valor final.
- Na prática, retiram-se os termos de crescimento mais lento, e diz-se que o algoritmo tem complexidade "+/- $1.5N^2$"
- De facto, até o termo 1.5 pode ser omitido, **bastando apenas dizer que a função atrás mostrada tem complexidade quadrática: $O(N^2)$.**
- **Exercício:** Considere dois algoritmos, em que 1 requer N^2 passos na execução e outro $0.001N^3$. Para que tamanhos de input sería preferível utilizar cada um deles?

Complexidade Computacional

- A seguinte tabela sumariza de forma empírica as gamas de valores plausíveis para o tamanho da entrada de cada tipo de algoritmo, por forma a que seja resolvido em "*poucos segundos*".

complexity	maximum N
$\Theta(N)$	100 000 000
$\Theta(N \log N)$	40 000 000
$\Theta(N^2)$	10 000
$\Theta(N^3)$	500
$\Theta(N^4)$	90
$\Theta(2^N)$	20
$\Theta(N!)$	11