Introdução

- **Algoritmo**: conjunto claramente especificado de instruções a seguir para resolver um problema
- Análise de algoritmos:
  - provar que um algoritmo está correto
  - determinar recursos exigidos por um algoritmo (tempo, espaço, etc.)
  - comparar os recursos exigidos por diferentes algoritmos que resolvem o mesmo problema (um algoritmo mais eficiente exige menos recursos para resolver o mesmo problema)
  - prever o crescimento dos recursos exigidos por um algoritmo à medida que o tamanho dos dados de entrada cresce

Complexidade espacial e temporal

- Complexidade espacial de um programa ou algoritmo: espaço de memória que necessita para executar até ao fim
  - ***S(n)*** - espaço de memória exigido em função do tamanho *(n)* da entrada
- Complexidade temporal de um programa ou algoritmo: tempo que demora a executar (tempo de execução)
  - ***T(n)*** - tempo de execução em função do tamanho *(n)* da entrada
- Complexidade ↑ versus Eficiência ↓
- Por vezes estima-se a complexidade para o "melhor caso" (pouco útil), o "pior caso" (mais útil) e o "caso médio" (igualmente útil)

Notação de *O* grande

- Na prática, é difícil (senão impossível) prever com rigor o tempo de execução de um algoritmo ou programa
  - Para obter o tempo precisamos, ao menos, de:
    - constantes multiplicativas (normalmente estas constantes são tempos de execução de operações atômicas)
    - parcelas menos significativas para valores grandes de *n*
  - Identificam-se as operações dominantes (mais freqüentes ou muito mais demoradas) e determina-se o número de vezes que são executadas (e não o tempo de cada execução, que seria uma constante multiplicativa)
- Exprime-se o resultado com a notação de *O* grande

O Conceito de Complexidade

- A Complexidade Computacional é um ramo da Matemática Computacional que estuda a eficiência dos algoritmos.
- Para medir a eficiência de um algoritmo freqüentemente usamos um tempo teórico que o programa leva para encontrar uma resposta em função dos dados de entrada.
- Se a dependência do tempo com relação aos dados de entrada for polinomial, o programa é considerado rápido, pois dado um polinômio p(x) sabemos que |p(x)| cresce para $+\infty$ com |x|. Se, entretanto, a dependência do tempo for exponencial o programa é considerado lento.

Algoritmos P e NP

- A classe de algoritmos **P** é formada pelos procedimentos para os quais existe um polinômio *p(n)* que limita o número de passos do processamento se este for iniciado com uma entrada de tamanho *n*.
- Os algoritmos **NP** não se referem a procedimentos não polinomiais (na verdade isto é uma conjectura). A leitura correta para procedimentos NP é dizer que se referem a algoritmos "não-determinísticos-polinomiais" no tempo.

Complexidade em Algoritmos Computacionais

- **O número 4.294.967.297 é um número primo?**
- Solução pelo crivo de Erastóstenes – Lenta!
- Procurar fatorar o número – Menos lenta – Euler resolve a questão em 1732.

Complexidade em Algoritmos Computacionais

- **O QUE SÃO OS ALGORITMOS NP AFINAL ?**
- A classe dos problemas NP é aquela para as quais podemos verificar, em tempo polinomial, se uma possível solução é correta.
- Os problemas de classe P estão contidos nos de classe NP. De fato, se um algoritmo pode ser executado em tempo polinomial e tivermos em mãos um possível candidato *S* à solução, é possível executar o programa, obter uma solução correta *C* e comparar *C* com *S* para certificar que *S* é de fato solução, tudo em tempo polinomial.

Complexidade em Algoritmos Computacionais

- **COMO RECONHECER SE UM ALGORITMO É OU NÃO É EFICIENTE ?**
- Um algoritmo é eficiente precisamente quando sua complexidade for um polinômio no tamanho de sua entrada.
- De acordo com a definição, um problema seria considerado tratável, exibindo-se algum algoritmo de complexidade polinomial que o resolvesse.
- Para verificar que é intratável, há necessidade de provar que ***todo*** possível algoritmo que o resolva não possui complexidade polinomial.

Complexidade em Algoritmos Computacionais

- **CLASSIFICAÇÃO DOS PROBLEMAS**
- Problema algorítmico: conjunto de dados +objetivos+instância
- Tipos:
  - **Problema de Decisão**: Existe estrutura S que satisfaça a propriedade P ?
  - **Problema de Localização**: Encontrar uma estrutura S que satisfaça uma propriedade P.
  - **Problema de Otimização**: Encontrar uma estrutura S que satisfaça critérios de otimização.

Complexidade em Algoritmos Computacionais

- *"Suponha que um caixeiro viajante tenha de visitar n cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. Suponha, também, que não importa a ordem com que as cidades são visitadas e que de cada uma delas pode-se ir diretamente a qualquer outra. O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total".*
- *PROBLEMA DO CAIXEIRO VIAJANTE*

Complexidade em Algoritmos Computacionais

- O problema do caixeiro viajante é um problema de otimização combinatória.
  - Como transforma-lo num problema de enumeração ?
  - Como determinar todas as rotas do caixeiro ?
  - Como saber qual delas é a menor ?
- SOLUÇÃO: São (n-1)! Rotas
  - É um trabalho fácil para a máquina ?

Complexidade em Algoritmos Computacionais

A quantidade (n - 1)! cresce com uma velocidade alarmante

n	Rotas por segundo	(n - 1)!	Cálculo total
5	250 milhões	24	insignificante
10	110 milhões	362 880	0.003 seg
15	71 milhões	87 bilhões	20 min
20	53 milhões	1.2×10^{17}	73 anos
25	42 milhões	6.2×10^{23}	470 milhões de anos

Complexidade em Algoritmos Computacionais

- Se descobrirmos como resolver o problema do caixeiro viajante em tempo polinomial, seremos capazes de resolver, também em tempo polinomial, uma grande quantidade de outros problemas matemáticos importantes.
- Se um dia alguém provar que é impossível resolver o problema do caixeiro em tempo polinomial no número de cidades, também se terá estabelecido que uma grande quantidade de problemas importantes não tem solução prática.
- Costuma-se resumir essas propriedades do problema do caixeiro dizendo que ele pertence à categoria dos problemas **NP - completos**.