

# Tópicos Avançados I

Joaquim Madeira

09/06/2020

# Sumário

- Recap
- Problemas tratáveis vs não-tratáveis
- Algoritmos determinísticos vs não-determinísticos
- Tipos de problemas
- Problemas de Decisão : As Classes P, NP e NP-Completo
- Determinação de soluções aproximadas
- Sugestão de leitura

Let's  
RECAP

# Recapitulação

# Procura Exaustiva

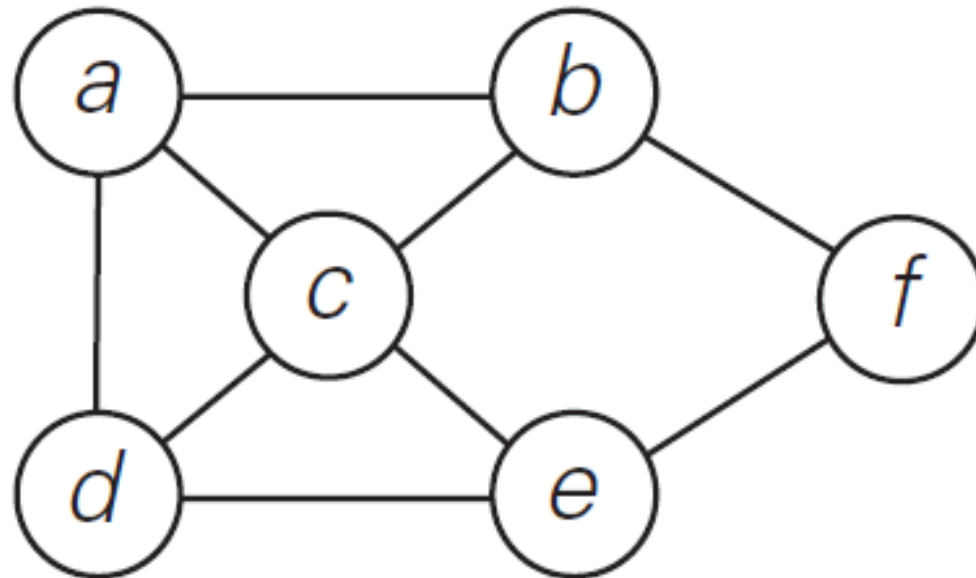
- Estratégia de força-bruta aplicada a **problemas combinatórios**
  - I.e., há um conjunto finito de soluções admissíveis
- Algoritmo
  - Enumerar **todas** as possíveis **soluções candidatas**
  - Verificar se cada uma **satisfaz** as restrições do problema
  - Se necessário, **escolher uma solução** do conjunto de soluções admissíveis
- Como assegurar que foram **verificadas todas as soluções candidatas** ?

# Procura Exaustiva

- Características
  - Muitas vezes é simples de implementar
  - Irá **sempre** encontrar uma solução, caso exista (**?!?**)
- MAS, em tempo proporcional ao número de soluções candidatas
  - **Explosão combinatória !**
  - Só praticável para instâncias “muito pequenas” **!!**

# Problema do Ciclo Hamiltoniano

- Este grafo tem um **Ciclo Hamiltoniano** ?
- **Como fazer** ?



[Levitin]

# Problema do Ciclo Hamiltoniano

- Como fazer ?
- Escolher um vértice qualquer de  $G$  como vértice inicial
- Gerar, sucessivamente, cada uma das  $(n - 1)!$  permutações possíveis dos vértices intermédios
- Acrescentar-lhe o vértice inicial e vértice final
- Verificar se a sequência de vértices define um ciclo em  $G$

# O Problema do Caixeiro Viajante

- Determinar o **caminho mais curto** que atravessa **n cidades**
- MAS, visitando cada cidade uma só vez !
- E retornando à cidade inicial !
- Problema de **otimização** combinatória



[Wikipedia]



# O Problema do Caixeiro Viajante

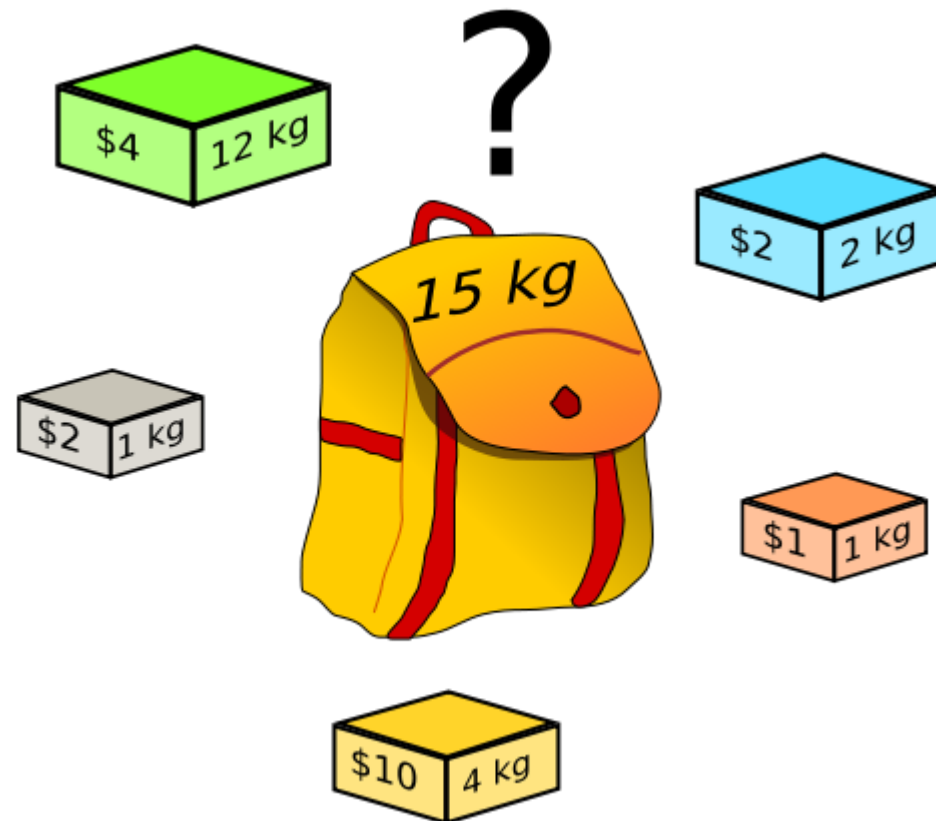
- Como fazer ?
- Escolher um vértice qualquer como vértice inicial
- Gerar, sucessivamente, cada uma das  $(n - 1)!$  permutações possíveis dos vértices intermédios
- Acrescentar-lhe o vértice inicial e vértice final
- Verificar se a sequência de vértices define um ciclo em  $G$
- Calcular o seu custo / distância
- Guardar o ciclo mais económico / mais curto

# O Problema do Caixeiro Viajante

- Questões
  - Como armazenar o grafo ?
  - O grafo é **completo** ?
  - Como gerar todas as **permutações** ?
- Desempenho computacional
  - **$O(n!)$**
  - A procura exaustiva só pode ser aplicada a instâncias muito pequenas !!
  - Alternativas ?

# O Problema da Mochila

- Determinar o **subconjunto mais valioso de itens**, que cabe na mochila



[Wikipedia]

# O Problema da Mochila

- Como fazer ?
- Gerar, sucessivamente, os  $2^n$  subconjuntos do conjunto de  $n$  itens
- Para cada um dos subconjuntos admissíveis, calcular o seu peso total e o seu valor
- E guardar o subconjunto mais valioso

# O Problema da Mochila

- Questões
  - Como gerar todos os subconjuntos ?
  - A ordem é importante ?
- Desempenho computacional
  - $O(2^n)$
  - A procura exaustiva só pode ser aplicada a instâncias muito pequenas !!
  - Alternativas ?
    - Soluções exatas vs. aproximadas

# Problemas Tratáveis **vs** Problemas Não-Tratáveis

# Algoritmos – Limitações

- Os algoritmos resolvem problemas
- MAS :
  - Alguns problemas não são resolúveis por um algoritmo
  - Outros problemas são resolúveis por um algoritmo, mas não em tempo polinomial
  - Mesmo quando os problemas são resolúveis em tempo polinomial, há habitualmente um limite inferior para a ordem de complexidade dos algoritmos usados

# Teoria da Complexidade

- Classificar os problemas de acordo com a sua ordem de complexidade
- Classificação principal :
  - Problemas tratáveis → Resolúveis em tempo polinomial
  - Problemas não-tratáveis → **Não podem ser resolvidos em tempo polinomial !!**  
**OU**  
**Não se sabe, se poderão ser resolvidos em tempo polinomial !!**



# Teoria da Complexidade

- A Teoria da Complexidade está centrada nos **Problemas de Decisão**
  - Respostas SIM / NÃO
- **Problema de decisão não-decidível**
  - Não é resolúvel por um algoritmo !!
  - **The Halting Problem**
- Algoritmos deterministas vs não-deterministas
  - **Guessing** + Verification !!
- Como **classificar** os problemas de decisão ?

# Algoritmos Deterministas **vs** Algoritmos Não-Deterministas

# Algoritmos Deterministas

- Um algoritmo determinista
  - Devolve sempre **o mesmo resultado**, qualquer que seja o número de vezes que é executado com **os mesmos dados de entrada**.
  - Executa sempre **a mesma sequência de instruções** quando é executado com **os mesmos dados de entrada**.
- O tipo mais habitual de algoritmo !
- Há uma definição mais formal em termos de máquinas de estado...

# Algoritmos Não-Deterministas

- Um algoritmo não-determinista
  - Pode ter um **comportamento diferente**, para **os mesmos dados de entrada**, em **diferentes execuções**
  - Ao contrário de um algoritmo determinista !
- Habitualmente usados para obter **soluções aproximadas** para instâncias de problemas de otimização combinatória
  - Quando é **demasiado oneroso** determinar **soluções exatas** usando um algoritmos determinista

# Algoritmos Não-Deterministas

- Como obter um comportamento diferente em cada execução ?
- Causas de **comportamento não-determinista**
  - Comportamento / estado externo que não os dados de entrada
    - Inputs do utilizador / timer values / **valores (pseudo-)aleatórios**
  - Operações “timing-sensitive” em sistemas com múltiplos processadores
  - Erros de hardware podem originar mudanças de estado inesperadas

# Tipos de Problemas

# Problemas de Decisão

- Resposta : **SIM** / **NÃO**
- Dado um número natural  $n$ ,  **$n$  é um número primo** ?
- Dados dois números naturais,  $x$  e  $y$ ,  **$x$  é divisível por  $y$**  ?
- ...
- Dado um grafo  $G(V,E)$ ,  $G$  tem um **Circuito Euleriano** ?
- Dado um grafo  $G(V,E)$ ,  $G$  tem um **Ciclo Hamiltoniano** ?
- ...

# Problemas de Decisão

- ...
- Dado um conjunto  $A$ , com  $n$  elementos inteiros e positivos, e um valor inteiro positivo  $S$ , a soma dos elementos de algum dos subconjuntos de  $A$  é igual a  $S$  ?
- ...
- Quais dos problemas são mais “difíceis” ?
- Como resolver os problemas mais “difíceis” ?



# Problemas de Procura / Identificação

- Identificar / Listar uma solução, **caso exista**
- Dado um grafo  $G(V,E)$ , listar um **Circuito Euleriano** de  $G$
- Dado um grafo  $G(V,E)$ , listar um **Ciclo Hamiltoniano** de  $G$
- Dado um **conjunto**  $A$ , com  **$n$  elementos inteiros e positivos**, e um valor inteiro positivo  **$S$** , listar os elementos de um dos **subconjuntos** de  $A$  cuja **soma** é igual a  **$S$**
- Quais dos problemas são mais “difíceis” ?
- Como **resolver** os problemas mais “difíceis” ?

# Problemas de Otimização

- Determinar a(uma) **solução ótima**, caso exista
- Dado um grafo  $G(V,E)$ , determinar o(um) **Ciclo Hamiltoniano** de **menor custo** de  $G$
- Dado um **conjunto de  $n$  itens**, e uma **mochila** de capacidade  **$W$** , determinar o(um) **subconjunto** de **maior valor** que cabe na mochila
- Como **resolver** ?

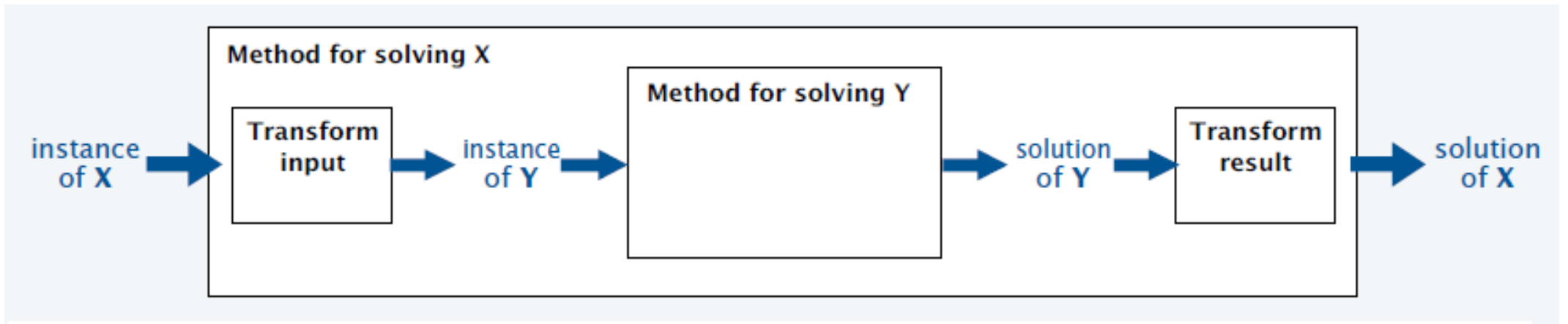
# As Classes P, NP e NP-Completo

# Probs. de Decisão : Classes de Complexidade

- **Classe P** – Polynomial-Time
  - Problemas de decisão resolúveis em tempo polinomial
  - Por **algoritmos deterministas** !!
- **Classe NP** – Non-deterministic Polynomial-Time
  - Problemas de decisão que podem ser resolvidos por algoritmos **não-deterministas**
  - **Soluções geradas aleatoriamente** podem ser **verificadas em tempo polinomial** !!
- Exemplos ?

# A classe NP-Completo

- Problema **NP- Completo**
  - Pertence à classe NP
  - **Todos** os outros problemas em NP são **polinomialmente redutíveis** nele



[Sedgewick & Wayne]

# Redução em tempo-polinomial

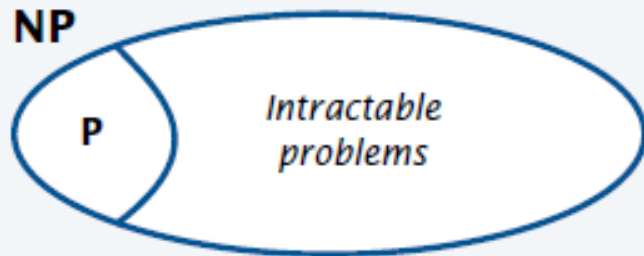
- Problema do Ciclo Hamiltoniano (HCP)
  - O grafo  $G$  tem um ciclo Hamiltoniano ?
- Problema do Caixeiro Viajante (TSP)
  - O grafo completo  $G'$  tem um **ciclo Hamiltoniano** de **custo** quando muito  **$m$**  ?
- HCP  $\alpha$  TSP
- Implicação ?

# P = NP ?

- $P \subseteq NP$
- $P = NP$  ??  $\rightarrow$  “The Million Dollar Question !!”

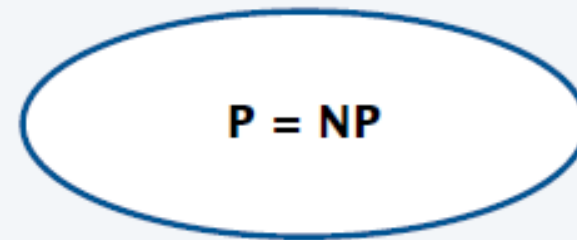
## $P \neq NP$

- Intractable search problems exist.
- Brute force search may be the best we can do for some problems.



## $P = NP$

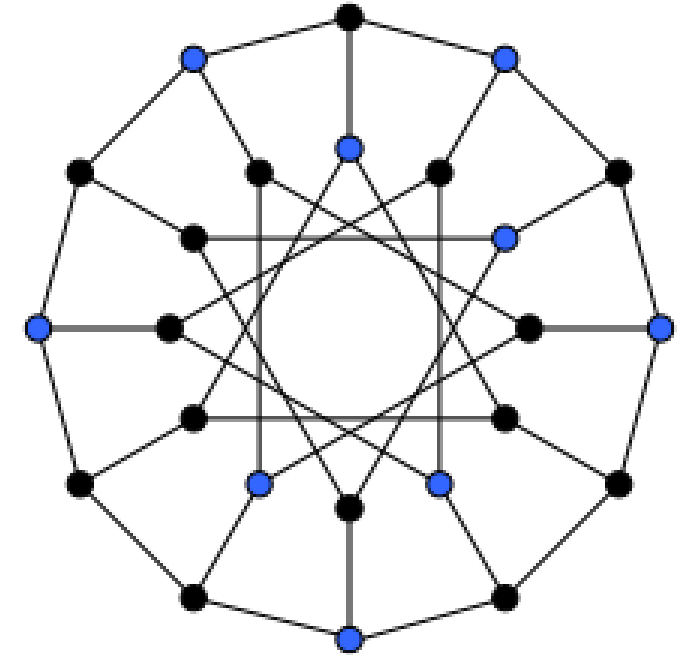
- All search problems are tractable.
- Efficient algorithms exist for IP, SAT, FACTOR ... *all* problems in **NP**.



[Sedgewick & Wayne]

# Conjunto independente de vértices

- $V' \subseteq V : v, w \text{ in } V' \rightarrow (v,w) \text{ not in } E$ 
  - $v$  e  $w$  são **não adjacentes** em  $G$
- O grafo  $G$  tem um conjunto independente de vertices de cardinalidade  $\geq k$  ?
  - **NP-Completo** !
- Determinar o **conjunto independente de vértices de cardinalidade máxima** do grafo  $G$ 
  - **NP-Difícil** !



[Wikipedia]

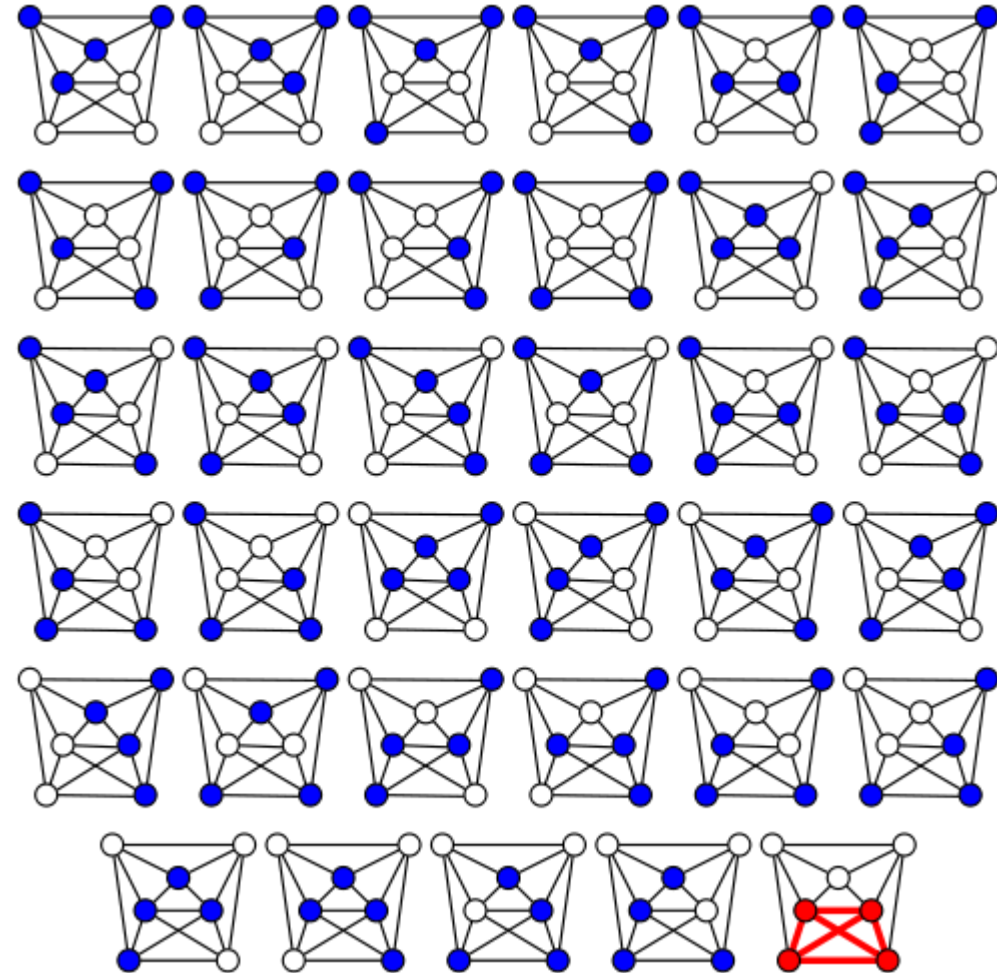


# Clique

- $V' \subseteq V : v, w \text{ in } V' \rightarrow (v, w) \text{ in } E$ 
  - $v$  e  $w$  são adjacentes em  $G$
- Clique vs. Conjunto Independente de Vértices ?
- O grafo  $G$  tem uma clique de cardinalidade  $\geq k$  ?
  - NP-Completo !
- Determinar a clique de cardinalidade máxima do grafo  $G$ 
  - NP-Difícil !

# Clique

- Determinar uma 4-clique usando a **procura exaustiva**
- **Algoritmos de aproximação** para tipos particulares de grafos !



[Wikipedia]

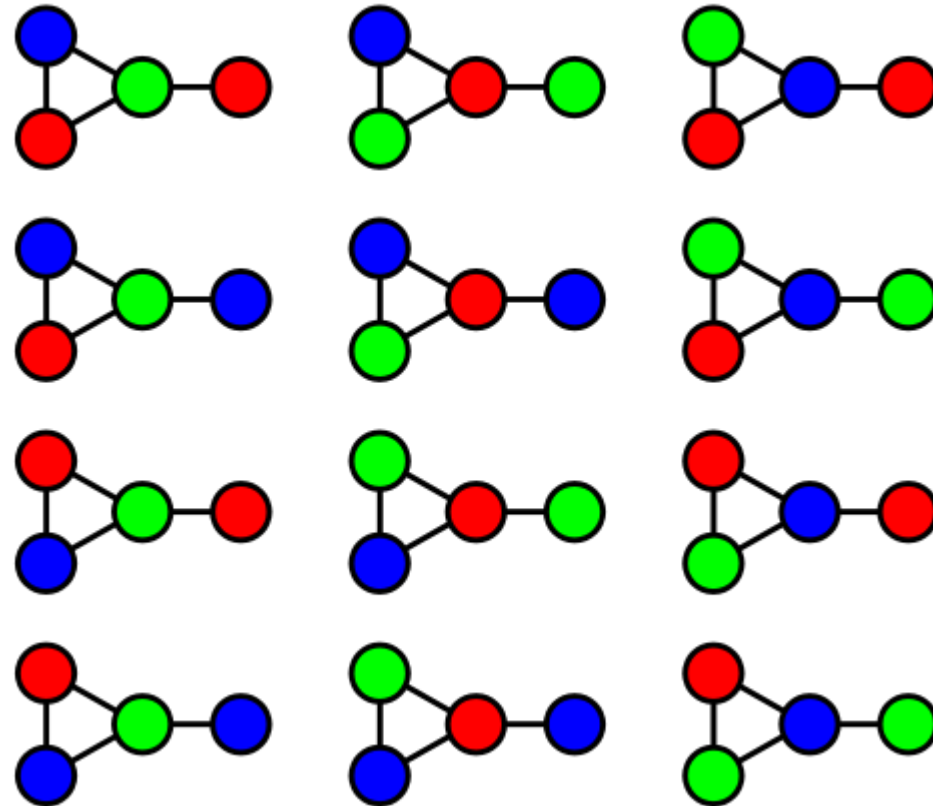
# Clique

- Aplicação : **Social Networks**
  - Determinar o maior subconjunto de pessoas em que todos se conhecem
- Outras áreas de aplicação
  - Bioinformática
  - Química Computacional
  - ...

# Coloração de Vértices

- **Coloração própria dos vértices** de  $G(V,E)$ 
  - Vértices adjacentes têm cores distintas (**rótulos**)
- O grafo  $G$  tem uma coloração própria de vértices usando, quando muito,  $k$  cores ( $\leq k$ ) ?
  - **NP-Completo**, exceto para  $k=1$  ou  $k=2$
- Determinar o **menor número de cores** definindo uma coloração própria dos vértices de  $G$ 
  - **Número cromático** de  $G$

# Coloração de Vértices



[Wikipedia]

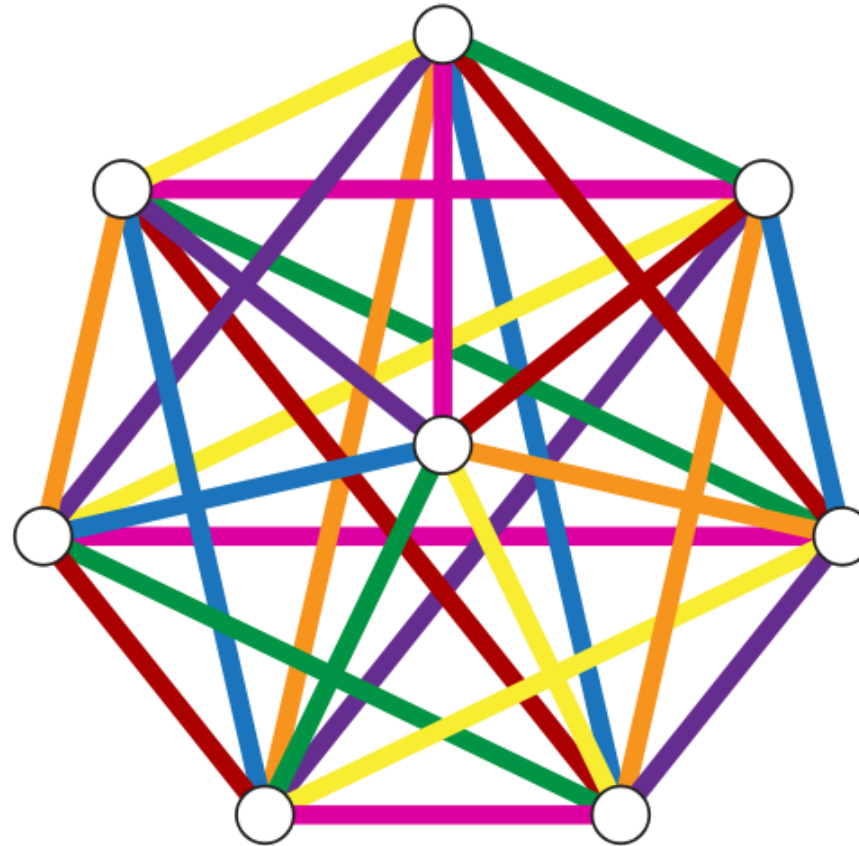
- 12 maneiras diferentes de colorir os vértices usando 3 cores

# Coloração de Arestas

- **Coloração própria das arestas** de  $G(V,E)$ 
  - Arestas adjacentes têm cores distintas (**rótulos**)
- O grafo  $G$  tem uma coloração própria de arestas usando, quando muito,  $k$  cores ( $\leq k$ ) ?
  - **NP-Completo**, exceto para  $k=1$  ou  $k=2$
- Determinar o **menor número de cores** definindo uma coloração própria das arestas de  $G$ 
  - **Índice cromático** de  $G$

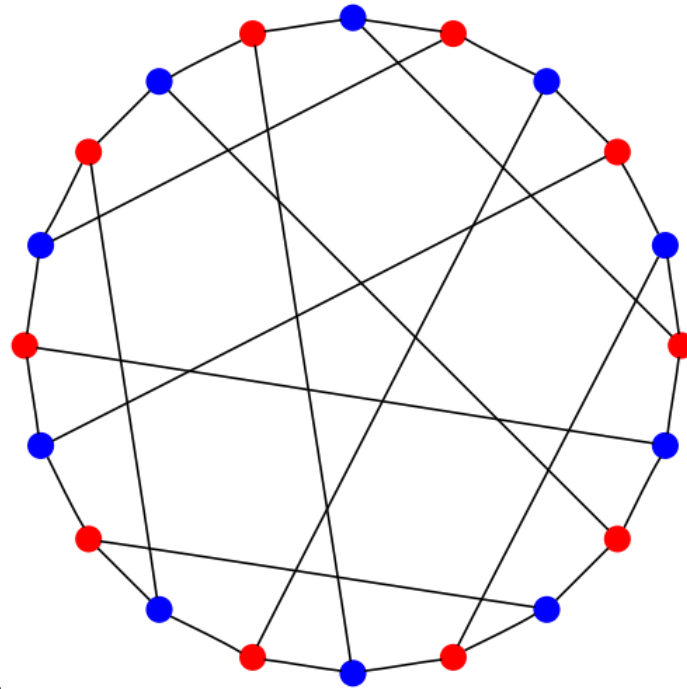
# Coração de Arestas

- $K_8$
- 8 vértices, 28 arestas
- raio 1
- diâmetro 1
- cintura 3
- 7-regular
- número cromático 8
- Índice cromático 7



[Wikipedia]

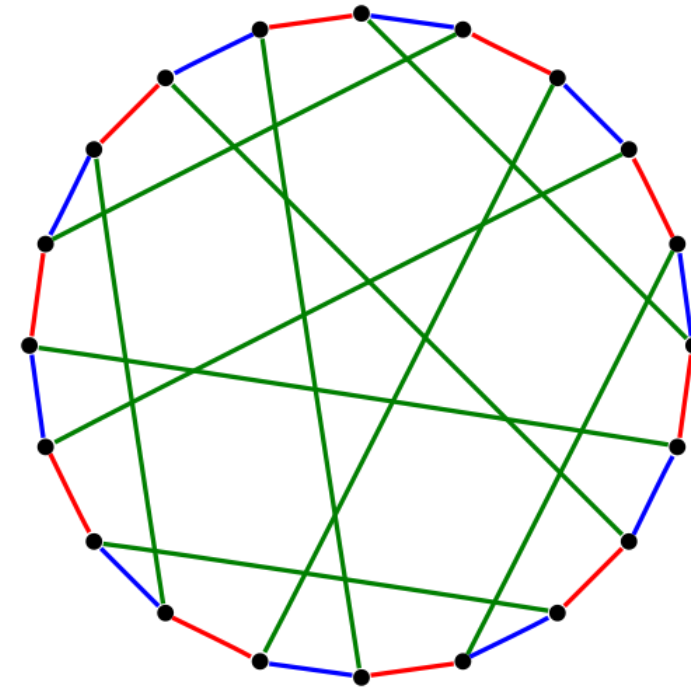
# Grafo de Desargues



20 vértices, 30 arestas

raio 5, diâmetro 5, cintura 6

número cromático 2, índice cromático 3



[Wikipedia]



# Aplicações

- Coloração de Vértices
  - Escalonamento
    - Aviões para voos; Alocação de frequências de rádio
  - Compiladores : otimização
  - Sudoku
- Coloração de Arestas
  - Escalonamento
    - Competições de todos-contra-todos
  - Comunicações óticas

# Grafos Isomorfos

- Dados dois grafos, definidos pelas suas listas de adjacências, esses grafos são isomorfos ?
- Pertence à classe NP
- MAS não se sabe
  - Se é resolúvel em tempo-polinomial para qualquer classe de grafos
  - Se pertence à classe NP-Completo

# Grafos Isomorfos

- Resolúvel em tempo-polinomial para
  - Árvores
  - Grafos planares
  - ...
- Aplicações
  - Identificar um composto químico numa base de dados química
  - Verificar circuitos elétricos (esquema vs. circuito)

# Soluções Aproximadas

# Soluções Aproximadas

- Não tentar determinar **soluções exatas** para problemas difíceis de otimização combinatória
  - **Pode demorar demasiado tempo !!**
  - Mundo-real : **dados imprecisos**
  - Soluções aproximadas poderão ser suficientes !!
- Calcular **soluções aproximadas**
  - P.ex., usando heurísticas **vorazes / greedy !!**
  - Avaliar a **exatidão** das soluções aproximadas
    - Performance ratio :  $R_A$

# Approximation Accuracy – Min Prob

- **Minimizar** uma função  $f()$
- Solução aproximada :  $s_a$
- Solução exata :  $s^*$
- Erro relativo :  $re(s_a) = ( f(s_a) - f(s^*) ) / f(s^*)$
- Exatidão / Accuracy ratio :  $r(s_a) = f(s_a) / f(s^*)$
- Performance ratio :  $R_A$ 
  - O menor majorante para os possíveis valores  $r(s_a)$
  - Deverá ser tão **próximo de 1** quanto possível
  - Expressa a **qualidade do algoritmo de aproximação**

# Approximation Accuracy – Max Prob

- **Maximizar** uma função  $f()$
- Solução aproximada :  $s_a$
- Solução exata :  $s^*$
- Erro relativo :  $re(s_a) = ( f(s^*) - f(s_a) ) / f(s^*)$
- Exatidão / Accuracy ratio :  $r(s_a) = f(s^*) / f(s_a)$
- Performance ratio :  $R_A$ 
  - O maior minorante para os possíveis valores  $r(s_a)$
  - Deverá ser tão **próximo de 1** quanto possível

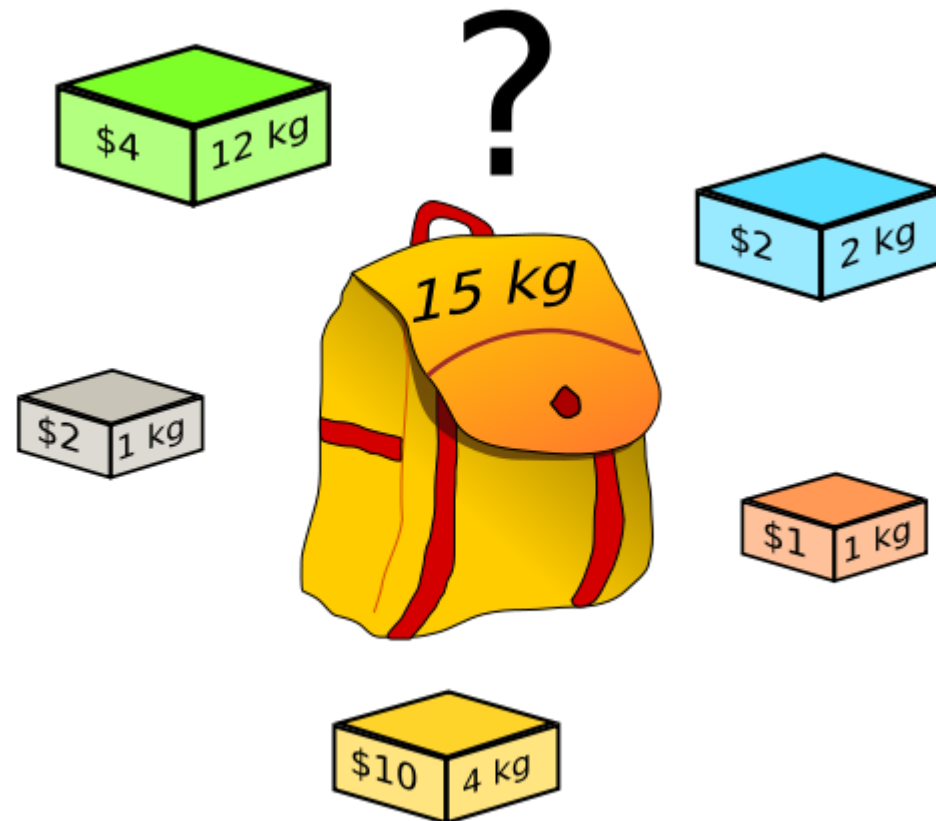
# Aplicação – Problemas de Otimização

- Vamos analisar algumas **heurísticas vorazes / greedy** simples
- O Problema da Mochila – The **0-1 Knapsack** Problem
- O Problema do Caixeiro Viajante – The **Traveling Salesperson** Problem



# O Problema da Mochila

- Determinar o **subconjunto mais valioso de itens**, que cabe na mochila



[Wikipedia]

# O Problema da Mochila

- Dados  **$n$  itens**
  - Com **peso**  $w_1, w_2, \dots, w_n$
  - Com **valor**  $v_1, v_2, \dots, v_n$
- Uma mochila de **capacidade**  $W$
- Qual é o (um) **subconjunto mais valioso de itens**, que cabe na mochila ?
- Problema **NP-difícil !!**

# O Problema da Mochila

- **Heurística voraz / greedy**
  - Selecionar os itens pela **ordem decrescente** dos seus **rácios  $v / w$**
- **Calcular** os quocientes valor / peso :  **$r_i = v_i / w_i$**
- **Ordenar** os itens em ordem **não-crescente** dos seus rácios  **$r_i$**
- **Repetir** até que não exista mais nenhum item na lista ordenada
  - Retirar o próximo item da lista
  - Se o **item cabe**, colocá-lo na **mochila**
  - Caso contrário, **esquecê-lo**



# O Problema da Mochila

- Mochila de capacidade  $W = 10$
- 4 itens
  - Item 1 :  $w = 7$  ;  $v = \$42$  :  $v / w = 6$  : 2º
  - Item 2 :  $w = 3$  ;  $v = \$12$  :  $v / w = 4$  : 4º
  - Item 3 :  $w = 4$  ;  $v = \$40$  :  $v / w = 10$  : 1º
  - Item 4 :  $w = 5$  ;  $v = \$25$  :  $v / w = 5$  : 3º
- Solução ?
  - Item 3 + Item 4 : **\$65**
  - Solução ótima

# O Problema da Mochila

- É uma **heurística simples**...
  - Existem outras...
- **Poderá ser sempre ótima ?**
- Qual seria a consequência de uma resposta positiva ?

# O Problema da Mochila

- Mochila de capacidade  $W = 50$
- 3 itens
  - Item 1 :  $w = 10$  ;  $v = \$60$  :  $v / w = 6$
  - Item 2 :  $w = 20$  ;  $v = \$100$  :  $v / w = 5$
  - Item 3 :  $w = 30$  ;  $v = \$120$  :  $v / w = 4$
- Resultado da heurística voraz
  - Item 1 + Item 2 : \$160 
- Solução ótima
  - Item 2 + Item 3 : \$220 !! 

# O Problema da Mochila

- Mochila de capacidade  $W > 2$
- 2 items
  - Item 1 :  $w = 1$  ;  $v = \$2$  :  $v / w = 2$  : 1º !!
  - Item 2 :  $w = W$  ;  $v = \$W$  :  $v / w = 1$  : 2º
- Solução ?
  - Item 1 !!!
  - Optimal solution : Item 2
  - $R_A = \infty$  !!



# O Problema do Caixeiro Viajante

- Determinar o **caminho mais curto** que atravessa **n cidades**
- MAS, visitando cada cidade uma só vez !
- E retornando à cidade inicial !
- Problema de **otimização** combinatória



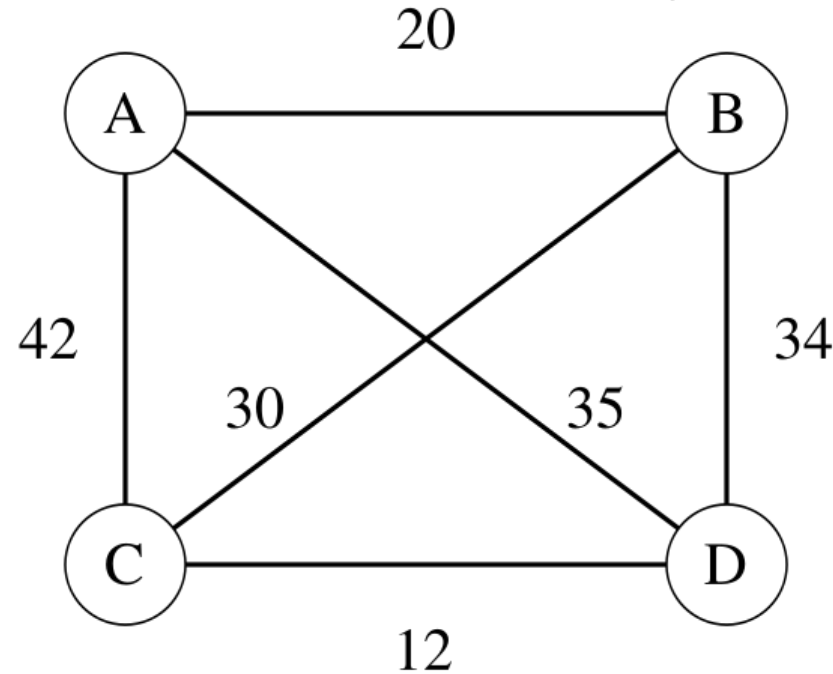
[Wikipedia]



# O Problema do Caixeiro Viajante

- Heurística do **Vizinho-Mais-Próximo / Nearest-neighbor – Greedy !!**
  - Prosseguir para a cidade mais próxima e ainda não visitada
- Escolher uma qualquer cidade como inicial
- Repetir até que todas as cidades tenham sido visitadas
  - Avançar para a cidade mais próxima e ainda não visitada
- Voltar à cidade inicial
- Heurística simples
- Mas  $R_A = \infty$  !!

# O Problema do Caixeiro Viajante



- Qual é a solução ótima ?
- Aplicar a heurística do vizinho-mais-próximo !
- Exatidão ?

# O Problema do Caixeiro Viajante

- Há outras heurísticas simples, por exemplo:
- Bidirectional-Nearest-Neighbor
- Shortest-Edge
- Aplicá-las ao exemplo anterior !!

# Sugestão de Leitura

# Sugestão de leitura

- A. Levitin, “*Design and Analysis of Algorithms*”, 3rd. Ed., Pearson, 2012
  - Chapter 11, Chapter 12