UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

Inteligência Artificial

Resolução de Problemas Por Meio de Busca

Docentes: Eng Roxan Cadir
Eng Ruben Manhiça

Maputo, 28 de agosto de 2024

Conteúdo da Aula

1. Resolução de Problemas Por meio de Buscas

Resolução de problemas por meio de busca

Capítulo 3 – Russell & Norvig

Secções 3.1, 3.2 e 3.3

Agentes de resolução de problemas

- Agentes reativos não funcionam em ambientes para quais o número de regras condição-acção é grande demais para armazenar.
- Nesse caso podemos construir um tipo de agente baseado em objetivo chamado de agente de resolução de problemas.

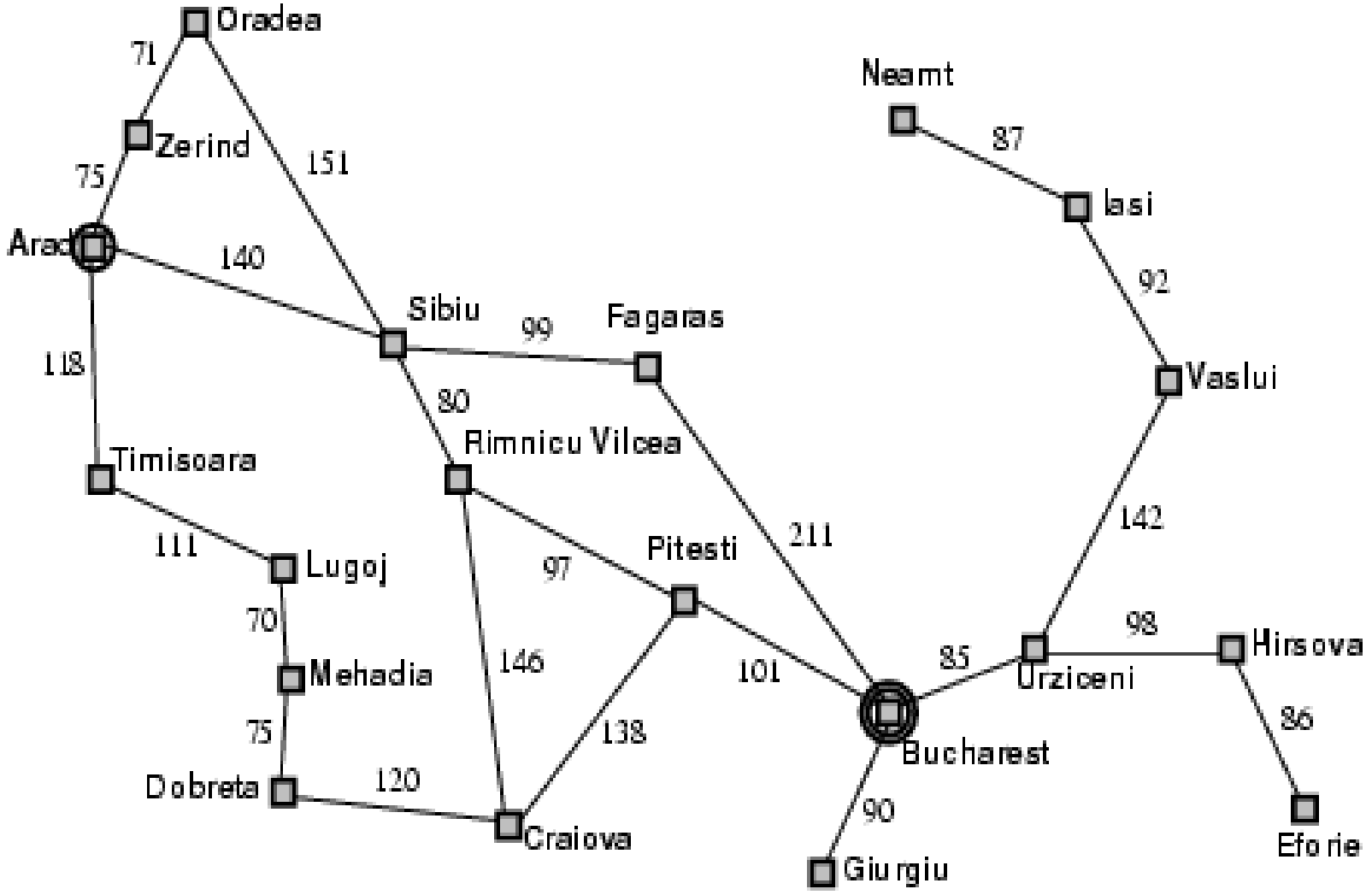
Busca

- Um agente com várias opções imediatas pode decidir o que fazer comparando diferentes sequências de acções possíveis.
- Esse processo de procurar pela melhor sequência é chamado de busca.
- Formular objetivo → buscar → executar

Exemplo: Romênia

- De férias na Romênia; actualmente em Arad.
- Vôo sai amanhã de Bucareste.
- Formular objetivo:
  - Estar em Bucareste
- Formular problema:
  - estados: Cidades
  - acções: Conduzir entre as cidades
- Encontrar solução:
  - sequência de cidades, ex., Arad, Sibiu, Fagaras, Bucareste.

Exemplo: Romênia



Formulação de problemas

Um problema é definido por quatro itens:

1. Estado inicial ex., “em Arad"
2. Acções ou função sucessor *S(x)* = conjunto de pares estado-acção
   - ex., *S(Arad)* = {<*Arad* → *Zerind, Zerind*>, … }
3. Teste de objetivo, pode ser
   - explícito, ex., *x* = “em Bucharest"
   - implícito, ex., *Cheque-mate(x)*
   -
4. Custo de caminho (aditivo)
   - ex., soma das distâncias, número de acções executadas, etc.
   - *c(x,a,y)* é o custo do passo, que deve ser sempre ≥ 0

- Uma solução é uma sequência de acções que levam do estado inicial para o estado objetivo.
- Uma solução ótima é uma solução com o menor custo de caminho.

Agente de Resolução de Problemas

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

Supõe que ambiente é estático, observável, discreto e determinístico.

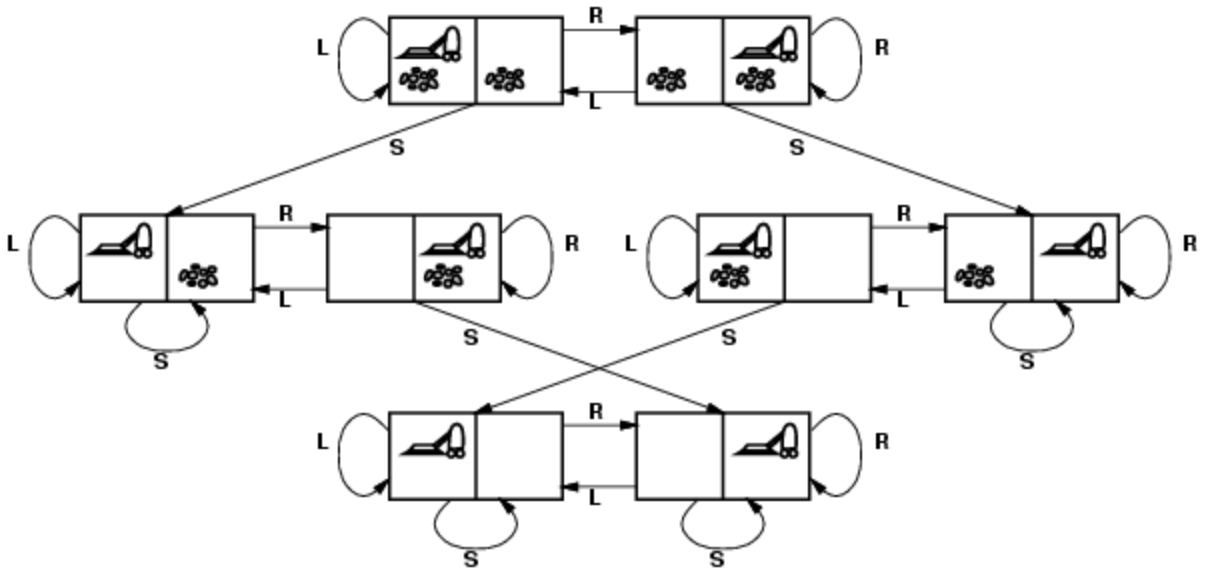
Espaço de estados

- O conjunto de todos os estados acessíveis a partir de um estado inicial é chamado de espaço de estados.
  - Os estados acessíveis são aqueles dados pela função sucessora.
- *O espaço de estados pode ser interpretado como um grafo em que os nós são estados e os arcos são acções*.

Selecionando um espaço de estados

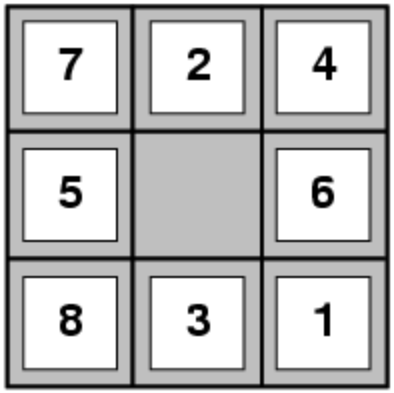
- O mundo real é absurdamente complexo
  → o espaço de estados é uma abstração
- Estado (abstrato) = conjunto de estados reais
- Acção (abstrata) = combinação complexa de acções reais
  - ex., "Arad → Zerind" representa um conjunto complexo de rotas, desvios, paragens, etc.
  - Qualquer estado real do conjunto "em Arad" deve levar a algum estado real "em Zerind".
- Solução (abstrata) = conjunto de caminhos reais que são soluções no mundo real
- A abstração é útil se cada acção abstrata é mais fácil de executar que o problema original.

Exemplo 1: Espaço de Estados do Mundo do Aspirador de Pó

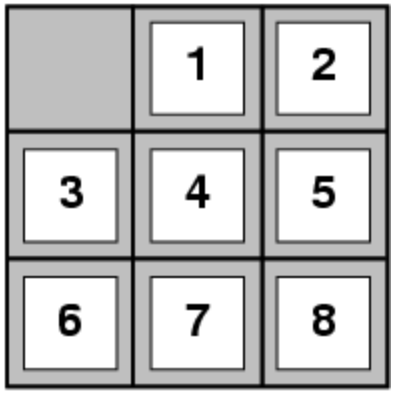


- **Estados:** Definidos pela posição do robô e sujidade (8 estados)
- **Estado inicial:** Qualquer um
- **Função sucessor:** pode-se executar qualquer uma das acções em cada estado (esquerda, direita, aspirar)
- **Teste de objetivo:** Verifica se todos os quadrados estão limpos
- **Custo do caminho:** Cada passo custa 1, e assim o custo do caminho é o número de passos do caminho

Exemplo 2:
O quebra-cabeça de 8 peças



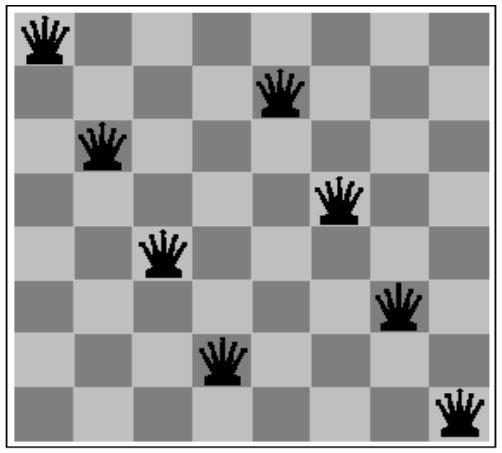
Start State



Goal State

- **Estados:** Especifica a posição de cada uma das peças e do espaço vazio
- **Estado inicial:** Qualquer um
- **Função sucessor:** gera os estados válidos que resultam da tentativa de executar as quatro acções (mover espaço vazio para esquerda, direita, acima ou abaixo)
- **Teste de objetivo:** Verifica se o estado corresponde à configuração objetivo.
- **Custo do caminho:** Cada passo custa 1, e assim o custo do caminho é o número de passos do caminho

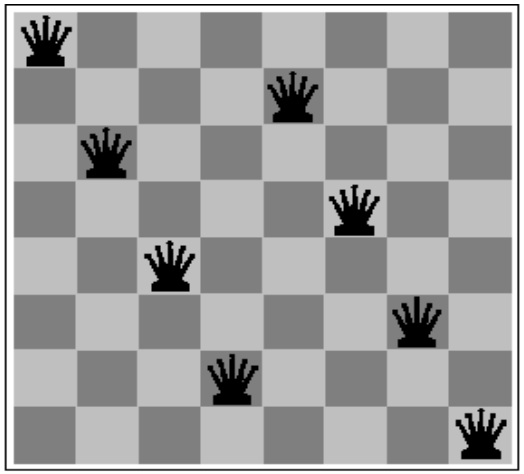
Exemplo 3:Oito rainhas
Formulação incremental



Quase solução

- **Estados:** qualquer disposição de 0 a 8 rainhas
- **Estado inicial**: nenhuma rainha
- **Função sucessor**: colocar 1 rainha em qualquer vazio
- **Teste**: 8 rainhas no tabuleiro, nenhuma atacada
- 64x63x...57 = $3x10^{14}$ sequências para investigar

Exemplo 3:Oito rainhas
Formulação de estados completos



Quase solução

- **Estados:** disposições de n rainhas, uma por coluna, nas n colunas mais a esquerda sem que nenhuma rainha ataque outra
- **Função sucessor**: adicionar uma rainha a qualquer quadrado na coluna vazia mais à esquerda, de tal modo que ela não seja atacada
- Tamanho do espaço de estados: 2.057

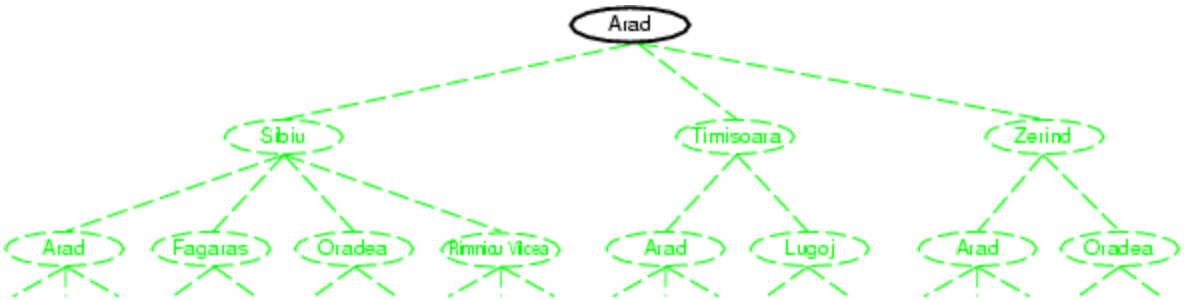
Problemas do mundo real

- **Problema de roteamento**
  - encontrar a melhor rota de um ponto a outro (aplicações: redes de computadores, planeamento militar, planeamento de viagens aéreas)
- **Problemas de tour**
  - visitar cada ponto pelo menos uma vez
- **Caixeiro viajante**
  - visitar cada cidade exatamente uma vez
  - encontrar o caminho mais curto
- **Layout de VLSI**
  - posicionamento de componentes e conexões em um chip
- **Projeto de proteínas**
  - encontrar uma sequência de aminoácidos que serão incorporados em uma proteína tridimensional para curar alguma doença.
- **Pesquisas na Web**
  - é fácil pensar na Web como um grafo de nós conectados por links

Busca de soluções

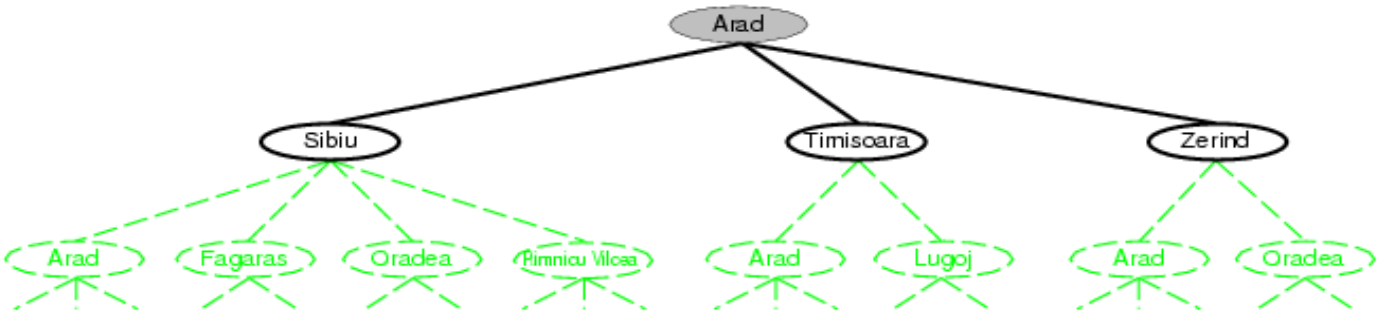
- Ideia: Percorrer o espaço de estados a partir de uma ***árvore de busca***.
- *Expandir* o estado actual aplicando a função sucessor, *gerando* novos estados.
- Busca: seguir um caminho, deixando os outros para depois.
- A *estratégia de busca* determina qual caminho seguir.

Exemplo de árvore de busca



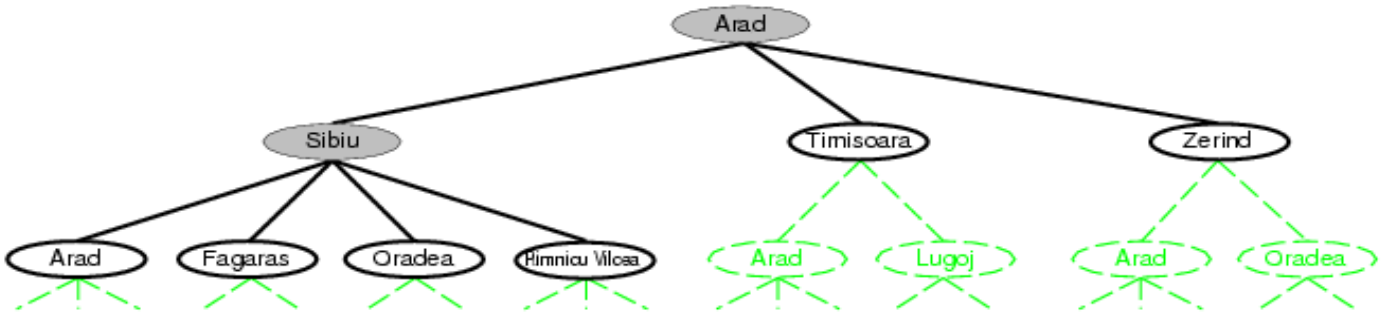
Estado inicial

Exemplo de árvore de busca



Depois de expandir Arad

Exemplo de árvore de busca



Depois de expandir Sibiu

Descrição informal do algoritmo geral de busca em árvore

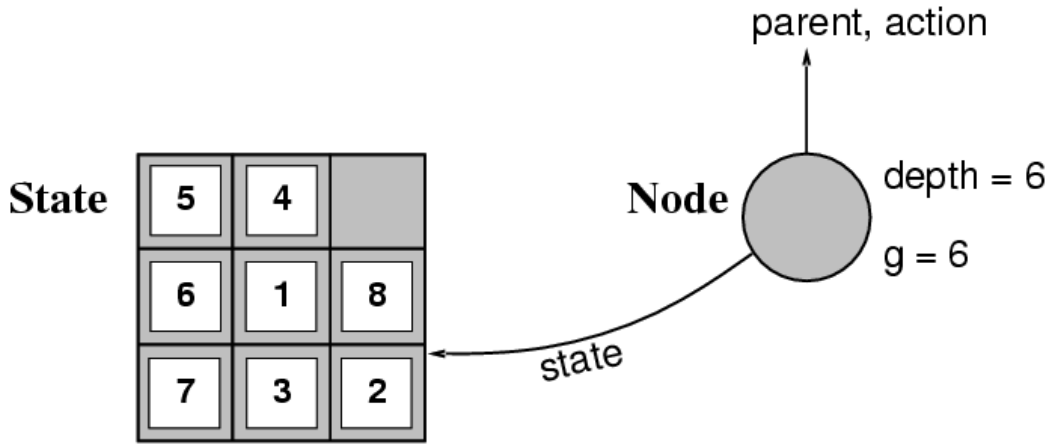
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

Árvore de busca não é equivalente a espaço de estados!

- Há 20 estados no mapa da Romênia (espaço de estados), mas infinitos caminhos a percorrer. Portanto a árvore de busca, neste caso, tem tamanho infinito.
  - Caminho infinito: Arad-Sibiu-Arad-Sibiu-Arad-...

Estados vs. nós

- Um estado é uma (representação de) uma configuração física
- Um nó é uma estrutura de dados que é parte da árvore de busca e inclui estado, nó pai, acção, custo do caminho *g(x)*, profundidade



- A função `Expand` cria novos nós, preenchendo os vários campos e usando a função sucessor do problema para gerar os estados correspondentes.
- A coleção de nós que foram gerados, mas ainda não foram expandidos é chamada de borda (ou fringe)
  - Geralmente implementados como uma fila.
  - A maneira como os nós entram na fila determina a estratégia de busca.

Algoritmo geral de busca em árvore

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Estratégias de busca

- Uma estratégia de busca é definida pela escolha da ordem da expansão de nós
- Estratégias são avaliadas de acordo com os seguintes critérios:
  - completeza: o algoritmo sempre encontra a solução se ela existe?
  - complexidade de tempo: número de nós gerados
  - complexidade de espaço: número máximo de nós na memória
  - otimização: a estratégia encontra a solução ótima?
- Complexidade de tempo e espaço são medidas em termos de:
  - *b:* máximo fator de ramificação da árvore (número máximo de sucessores de qualquer nó)
  - *d:* profundidade do nó objetivo menos profundo
  - *m*: o comprimento máximo de qualquer caminho no espaço de estados (pode ser ∞)

TPC

- Ler as secções do capitulo 3
- Investigar o que são Problemas NP (Equivalente, Fácil, Completo, Difícil)

FIM!!!

Duvidas e Questões?