# Relatório Agente de Busca Inteligência Artificial

**Daniel Rosendo de Souza**[1]

[1]Ciência da Computação – Instituto Federal do Ceará (IFCE)

daniel.rosendos@hotmail.com

***Resumo.*** *Este relatório descreve problemas da área de inteligência artificial, como problema de busca do mapa da Romênia, problema das 8-rainhas e problema do mundo do aspirador de pó, utilizando algoritmos de busca para achar soluções dos problemas.*

## 1. Introdução

Durante o estudo da cadeira de inteligência artificial, foram mostrados em sala de aula alguns problemas de busca. Inicialmente foi mostrado o problema do aspirador de pó, que consiste em um pequeno mundo de dois lugares, onde um agente, aspirador de pó, tem a tomar decisões sobre se movimentar a esquerda ou direita e decidir se o local está limpo ou não, fazendo assim a limpeza ou não do local. Segundamente foi mostrado o problema das 8 rainhas, este problema consiste em, como dispor 8 rainhas em um tabuleiro de xadrez 8 por 8, onde nenhuma das rainhas se ataque. Por fim foi mostrado o problema de percurso ou caminho, entre dois pontos, no nosso caso utilizando o mapa da Romênia.

Para resolver esses problemas, é mostrado como resolver esses problemas utilizando algoritmos de busca, podendo citar alguns como busca em largura, busca em profundidade e entre outros. Cada resolução de problemas pode ser ótima ou não, dependerá de qual algoritmo de busca será utilizado para resolver o problema.

## 2. Contexto

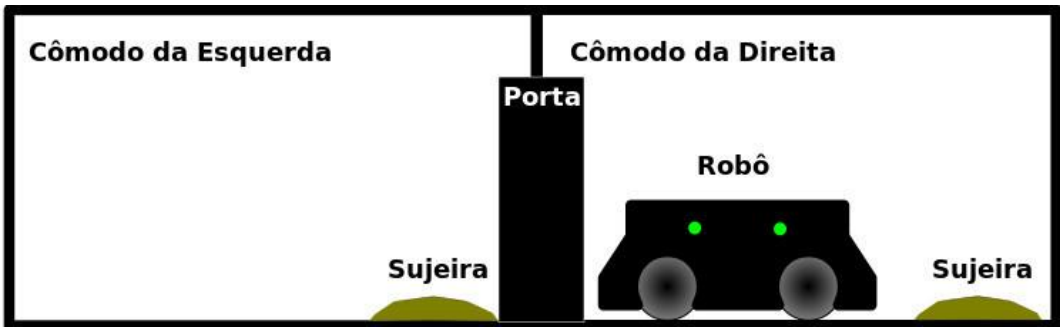
Foi proposto pelo professor, aos alunos, a implementação dos agentes de resolução de problemas por meio de busca, busca em profundidade, busca em profundidade com lista de visitados, busca em profundidade limitada, busca em profundidade iterativa, busca em largura e busca de custo uniforme. Feita as implementações utilizar esses agentes para resolver os problemas do mundo do aspirador de pó, problemas das oito rainhas e problema do mapa da Romênia.

As implementações foram feitas em linguagem python em sua versão 2.7, utilizando a IDE pycharm e este trabalho será dividido da seguinte forma: Na próxima sessão será explicado com mais detalhes os problemas de busca, na sessão 4 será explicado a organização dos arquivos, na sessão 5 será explicado sobre os algoritmos de busca e sua implementação, na sessão 6 será mostrado uma tabela comparativa dos testes e na sessão 7 será explicado os agentes de buscas e sua implementação e por fim uma breve conclusão.

## 3. Problemas de Busca

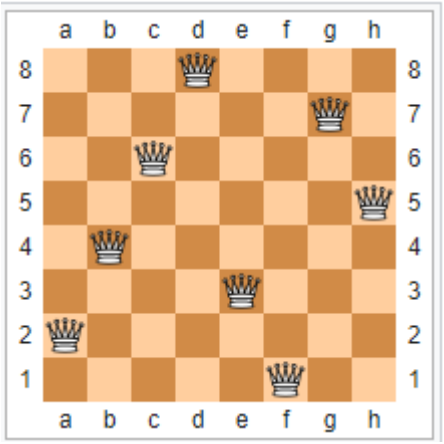
### 3.1. Mundo Aspirador de Pó

Nesse problema, há um pequeno mundo onde um aspirador de pó autônomo é colocado em um lar com apenas 2 cômodos, e seu objetivo é manter sempre os 2 cômodos limpos na maior parte do tempo em que o aspirador estiver sendo avaliado. O robô só pode estar em um desses cômodos em cada instante de tempo e ele tem como objetivo final manter os dois cômodos limpos. Para alcançar seu objetivo, o robô pode executar três ações: Mover-se para o cômodo da esquerda, mover-se para o cômodo da direita e limpar/aspirar ao cômodo atual e, além disso, supõe-se que os cômodos não voltam a ficar sujos. A imagem abaixo ilustra esse problema:



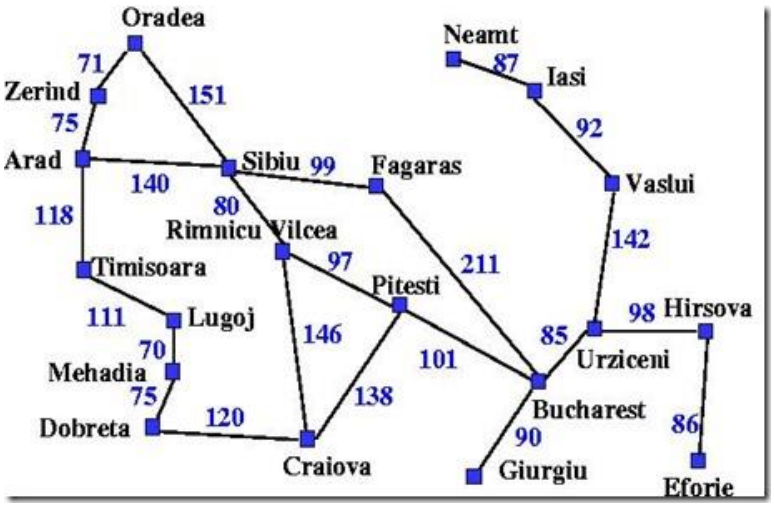

### 3.2. Problema das 8-Rainhas

O problema das oito rainhas é o problema matemático de dispor oito rainhas em um tabuleiro de xadrez de dimensão 8 por 8, de forma que nenhuma delas seja atacada por uma outra. Para tanto, é necessário que duas rainhas quaisquer não estejam numa mesma linha, coluna ou diagonal. A imagem abaixo ilustra uma das soluções possíveis para o problema das 8-Rainhas:



### 3.3. Problema Mapa da Romênia

O problema mapa da Romênia, pode-se caracterizar como um problema de roteamento, que é definido em termos de posições especificas e transições ao longo de ligações entre elas. Este problema consiste em como se deslocar entre duas cidades, de forma que o viajante consiga viajar em um menor caminho possível ou encontrar uma rota de uma cidade A para uma cidade B.




### 4. Organização

Nossos arquivos estão organizados da seguinte forma:

- Agente_Teste: Arquivo que contém os teste das implementações feitas dos agentes de busca.
- Classes: Arquivo que contém as classes do nosso projeto, classe problema classe que inicializa e faz a leitura do problema, classe Nó ela é quem faz a caracterização do objeto Nó e nossa Classe ataques utilizada para resolução do problema das 8-rainhas.
- Rainha: Arquivo que contém as funções de dar output no terminal ao final da resolução do problema das rainhas.
- Solver: Arquivo que contém a função de resolução de problemas das rainhas.
- Worlds: Arquivo que contém o mundo do aspirador de pó e o mapa da Romênia, criação de suas ligações.

## 5. Resolução de Problemas por Meio de Agentes de Busca

Um agente de busca é um processo que tenta encontrar uma sequência de ações que leva de um estado inicial até um estado objetivo, uma vez encontrada a solução, o agente pode executar a sequência de ações para chegar a seu objetivo.

### 5.1. Busca em Largura

Busca em largura é um método de busca sem informação que expande e examina sistematicamente todos os vértices de um grafo direcionado ou não-direcionado. Podemos dizer que o algoritmo realiza uma busca exaustiva num grafo passando por todas as arestas e vértices do grafo. Utiliza-se de uma estrutura de dados fila para garantir a ordem de chegada dos vértices. Com isso, as visitas aos vértices são realizadas através da ordem de chegada à estrutura fila.

Sua implementação é feita da seguinte forma, inicialmente o algoritmo de busca irá receber o problema, onde o nó inicial recebe o ponto de partida, se esse ponto de partida já for o nosso objetivo, retorne esse nó caso não, adicionamos na nossa borda esse nó e enquanto a borda não estiver vazia, remove a borda da fila e para cada ação dentro das ações do problema de seu estado, o filho recebe a ação e o nó, se o filho não estiver na borda e o estado do filho não estiver explorado, então faça, se esse estado do filho for o objetivo então me retorne a solução caso não acrescente o filho na borda.

## 5.2. Busca em Profundidade

Busca em profundidade é um método de busca sem informação que progride através da expansão do primeiro nó filho da árvore de busca, e se aprofunda cada vez mais, até que o alvo de busca seja encontrado ou até que ele se depare com um nó que não possuí filhos. Em implementações não recursivas, todos os nós expandidos são adicionados em uma pilha.

Sua implementação é feita da seguinte forma, inicialmente o algoritmo de busca receberá o problema, onde o nó inicial recebe o inicio do problema, ou ponto de partida, se esse ponto de partida já for o nosso objetivo, retorna esse nó como resultado, caso não adiciona esse nó a borda, enquanto a borda não for vazia, nó recebe da pilha o nó da borda, para cada ação dentro das ações do problema, do estado do nó, faça o filho receber a ação e o no, e compare se o filho não estiver na borda faça a comparação, se o estado daquele filho for o objetivo, retorne esse filho como solução, caso não acrescente o filho a borda.

### 5.3. Busca em Profundidade com Lista de Visitados

A busca em profundidade com lista de visitados, parte da mesma definição que uma busca em profundidade, porém, este algoritmo de busca possuí uma lista de explorados, fazendo com que a borda não receba nós repetidos, deixando apenas nós não visitados pelo algoritmo.

Sua implementação é feita da seguinte forma, inicialmente o algoritmo de busca receberá o problema, onde o nó inicial recebe o inicio do problema, ou ponto de partida, e pergunta, se esse ponto de partida já for o nosso objetivo, retorne esse nó como resultado, caso não cria-se um vetor borda, uma lista de visitados e acrescenta o nó a borda. Enquanto a borda não estiver vazia, um nó recebe da pilha o nó da borda, e é adicionado na lista de visitador o nó anterior, para cada ação dentro das ações do problema, do estado do nó, faça o filho receber a ação e nó e comparar, se o estado do filho não estiver na lista de explorados e o filho não estiver na borda, então faça as seguintes comparações, se esse estado do filho for o nosso objetivo ele me retorna a

solução caso não verifica se o esse estado não está na lista de explorados e adiciona esse filho a borda.

### 5.4 Busca em Profundidade Limitada

A busca em profundidade limitada, parte da mesma definição que uma busca em profundidade, porém, o número de iterações é delimitado, fazendo com que force um limite de quantas vezes o algoritmo irá rodar, até achar a solução, podendo ou não achar solução dependendo do limite estabelecido.

Inicialmente nosso algoritmo recebe como entrada o nosso problema, um limite de iterações, onde o nó inicial recebe o inicio do problema, é chamada posteriormente uma função recursiva, passando como parâmetros o nó inicial, o problema e o limite. Nessa função recursiva, iremos perguntar se, esse nó recebido já é o nosso objetivo, caso seja retorne a solução, caso se o nosso limite seja zero, retorne um corte fora e caso qualquer outra coisa faça, as ocorrências de corte fora ser falso, e para cada ação dentro das ações do problema, do estado do nó, faça o filho receber a ação e o nó, onde o resultado é uma chamada recursiva da função. Se o resultando chegar a um corte fora, faça nossa ocorrência de corte fora virar verdade, caso resultado não seja vazio retorne o resultado, e se nossa ocorrência de corte for verdade retorne corte fora, caso não retorne falso.

### 5.5 Busca Iterativa

A busca iterativa, é uma busca em profundidade limitada, que utiliza-se de recursos para incrementar o limite de forma automática, até que seja encontrado o resultado.

A sua implementação é bem simples, basta criar uma variável profundidade e enquanto for verdade, chamar a busca em profundidade limitada até que o resultado não seja um corte fora, retornando o resultado.

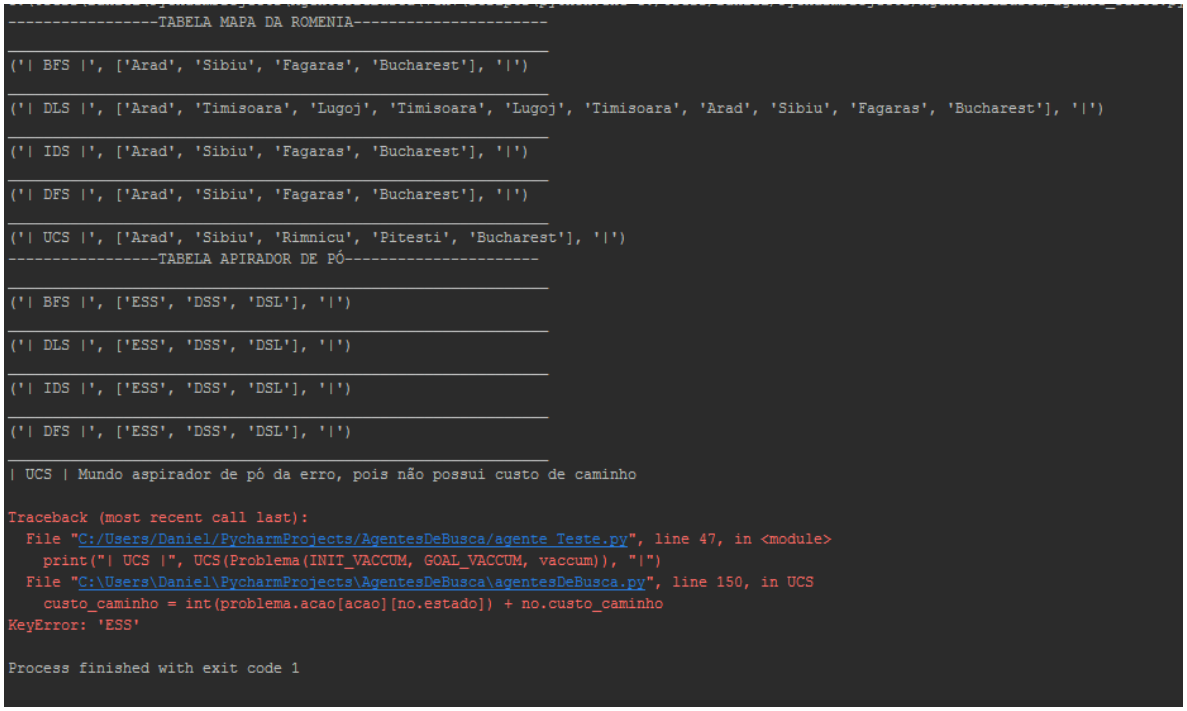
### 5.6 Busca de Custo Uniforme

A busca de custo uniforme, é uma estratégia de busca baseada na busca em largura, mas ao invés de pegar o primeiro nó expandido que está na lista, o nó que possuí menor custo será escolhido para ser expandido.

Inicialmente nosso algoritmo recebe como entrada nosso problema, um no recebe o inicio de nosso problema, ou ponto de partida, a borda recebe uma fila de prioridade, adiciona-se a borda o custo do caminho de cada nó, e é criado uma lista de visitados. Enquanto a borda não for vazia, um auxiliar recebe o nó da borda e nó recebe o auxiliar, e vemos se esse nó é o nosso objetivo, caso seja retorne esse nó como solução, caso não adicione o nó na lista de explorado e para cada ação dentro das ações de nosso problema, dos estados do nó faça, o custo do caminho receba o custo do caminho de um ponto a outro, onde o filho recebe a ação, o nó e o custo do caminho entre os nós, se o custo do caminho e o filho não estiverem dentro da borda e o estado do filho não estiver na lista de explorado coloque o custo do caminho e o filho dentro da borda.

## 6. Testes

Abaixo segue a tabela comparativa dos agentes de busca em relação ao problema de busca do mapa da Romênia e ao problema de busca do mundo aspirador de pó:




Para mundo aspirador de pó, a busca UCS – Custo Uniforme, apresenta erro devido ao mundo do aspirador de pó não possuir custo no seu pecurso.

## 7. Conclusão

Este trabalho teve como intuito aumentar o aprendizado de agentes de resolução de problema, por meio da implementação desses agentes por alguma linguagem de programação, demonstrando de forma concreta o funcionamento desses algoritmos.