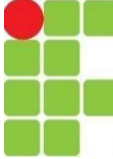


| | | |
|---|--|--|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência artificial | Atividade livro | |
| Professor: Prof. Dr. Murilo Vargues da Silva | Data: 31/07/2023 | |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

Exercícios do livro “Inteligência artificial” do George F. Luger, pg.27, exercícios 4,10,11,12

Exercício 4

R: Apesar de ser um marco importante na história da inteligência artificial, os critérios de Turing apresentam várias limitações que podem dificultar a avaliação precisa da inteligência das máquinas. Ele foca principalmente na imitação e na comunicação, deixando de lado muitos outros aspectos importantes da inteligência e da compreensão.

Exercício 10

R: O problema do aprendizado de máquina é difícil devido a vários motivos. Os dados do mundo real são complexos e multidimensionais, tornando a interpretação complexa. Encontrar o equilíbrio entre ajuste excessivo e subajuste é complicado.

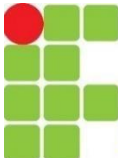
Exercício 11

R: Sim, é possível que um computador compreenda e use a linguagem natural humana, embora com limitações e desafios significativos. A área da Inteligência Artificial (IA) que se concentra nessa capacidade é chamada de Processamento de Linguagem Natural (PLN) ou Processamento de Linguagem Natural.

Exercício 12

R: Viés Algorítmico e Desigualdade: Algoritmos de IA podem amplificar preconceitos presentes nos dados de treinamento, resultando em decisões discriminatórias e reforçando desigualdades sociais.

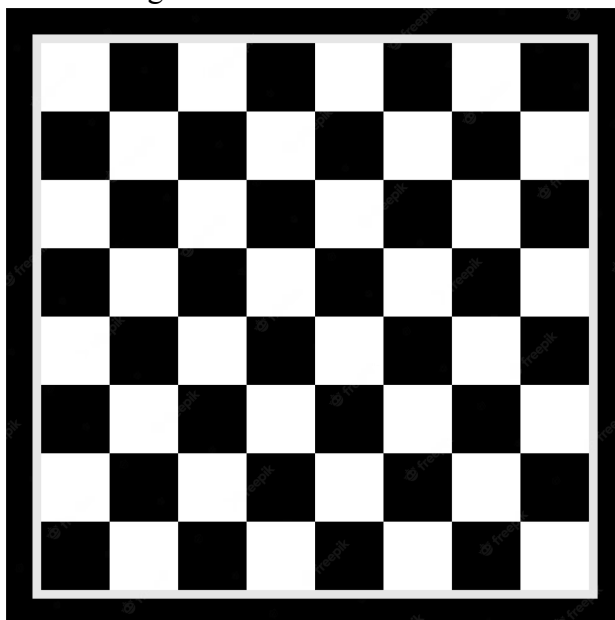
Desemprego Tecnológico: A automação impulsionada pela IA pode substituir empregos, causando desemprego e exigindo adaptação dos trabalhadores para novas funções. Isso pode gerar instabilidade econômica e social.

| | | |
|---|--|--|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência artificial | Atividade 1 | |
| Professor: Prof. Dr. Murilo Vargues da Silva | Data: 07/08/2023 | |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

Atividade das rainhas

O objetivo do problema de oito rainhas é posicionar oito rainhas em um tabuleiro de xadrez de tal forma que nenhuma rainha ataque qualquer outra (uma rainha ataca qualquer peça situada na mesma linha, coluna ou diagonal).

Figura 1 - Tabuleiro de xadrez



Tamanho do espaço de estados: 178.462.987.637.760 possibilidades.

Estados: 0 a 8 rainhas.

Estado inicial: nenhuma rainha posicionada no tabuleiro, se iniciando na primeira posição.

Ações: a partir do estado inicial, inserir uma rainha para cada coluna deslocando duas linhas para baixo a cada inserção. Caso não existam linhas suficientes para se deslocar para baixo, deve-se considerar a primeira linha para iniciar a contagem.

Teste de objetivo: posicionar 8 rainhas sem que elas se ataquem.

Atividade do robô

O objetivo é dirigir o robô para fora de um labirinto. O robô inicia no meio do labirinto em direção ao norte. Você pode virar o robô em direção ao norte, sul, leste ou oeste. O robô pode ser comandado para mover uma certa distância para frente, apesar que irá parar antes de bater no muro.

Formule esse problema. Qual é o tamanho do espaço de estados?

Tamanho do espaço de estados: Considerando um labirinto 10x10, teremos 429 possibilidades.

Estados: Parado, rotacionando e andando.

Estado inicial: iniciar no meio do labirinto virado para o norte.

Ações: o robô poderá se locomover livremente dentro do labirinto. Para isso ele irá poder virar para as 4 direções, norte, sul, leste ou oeste e também será possível dele seguir em frente.

Teste de objetivo: o objetivo deste problema é possibilitar o robô de sair do labirinto.

Ao navegar pelo labirinto, é necessário virar apenas na interseção de dois ou mais corredores. Reformule esse problema usando essa observação. Qual será o tamanho do espaço de estados agora?

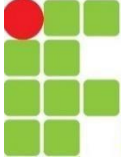
Tamanho do espaço de estados: Considerando o mesmo labirinto 10x10.

Estados: Parado, rotacionando e andando.

Estado inicial: iniciar no meio do labirinto virado para o norte.

Ações: o robô poderá se locomover livremente dentro do labirinto. Para isso ele irá virar apenas quando houver uma intersecção entre dois corredores. Ao atingir a intersecção, ele irá rotacionar para o lado possível para continuar andando pelo labirinto.

Teste de objetivo: o objetivo deste problema permanece o mesmo.

| | | |
|---|--|-------------------------|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência artificial | | Atividade |
| Professor: Prof. Dr. Murilo Vargues da Silva | | Data: 18/08/2023 |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

TRABALHO - BUSCA LABIRINTO

ESTUDO DO FUNCIONAMENTO

Geração de labirinto

A geração do labirinto é realizada pelo código denominado `maze_generator.py`. A inicialização do labirinto é através de um array cheio de 0's, indicando células vazias.

```
# initialize the grid array full of zeros
num_rows = 41
num_columns = num_rows
grid = np.zeros((num_rows, num_columns))
```

A posição inicial será demarcada em verde “2”, sendo então o ponto de partida do labirinto. O ponto que determina a conclusão do labirinto é mostrado como um ponto vermelho “3”. Esses pontos de início e fim do labirinto podem ser alterados no código, porém deve-se tomar cuidado para não quebrar o intuito do labirinto. As cores também podem ser trocadas, mas para manter de uma forma mais intuitiva mantém as mesmas cores.

```
# define start and goal
grid[0, 0] = 2
grid[-1, -1] = 3
```

O programa entra em um loop principal e permanece lá até o labirinto estar completo. A geração do labirinto é feita passo a passo através da função ‘`generate_step`’ que irá determinar o próximo passo na geração. Quando ela recebe a posição atual, o histórico das posições percorridas e o contador de passos de retorno como entrada. Ela determina as próximas posições em que o algoritmo pode avançar. Se houver pelo menos uma posição válida para se mover, o algoritmo escolhe uma delas aleatoriamente e avança para a posição. As posições escolhidas indicarão o caminho em que é possível

caminhar, sendo eles na cor branca “1”. A posição anterior é atualizada para a nova posição e adicionada ao histórico de posições visitadas.

Esse algoritmo conta com uma funcionalidade de dead end, ou também conhecido como beco sem saída. Se caso não houver posições válidas para onde se mover a partir da posição atual, o algoritmo retrocede na trilha do labirinto, voltando para uma posição anterior que tenha caminhos não explorados. Os números de passos do retrocesso são rastreados para evitar retroceder demais.

Mantendo toda essa lógica anteriormente apresentada, o algoritmo continuará gerando passo a passo até que todas as células tenham sido visitadas e o labirinto esteja completamente gerado.

Estratégias: Busca em largura, profundidade e A*

Antes de iniciarmos falando sobre como os algoritmos resolvem esse problema, devemos entender o propósito de cada um. O algoritmo BFS, ou também conhecido como busca em largura, explora os nós em camadas, começando pela posição inicial e expandindo gradualmente para nós vizinhos antes de prosseguir para camadas subsequentes. Garantindo que encontre o caminho mais curto. Para a resolução do labirinto, o algoritmo utiliza o BFS para controlar a fronteira e os nós explorados. Todo o processo de resolução pode ser exibido se caso habilitado, mostrando o caminho em que o BFS percorre.

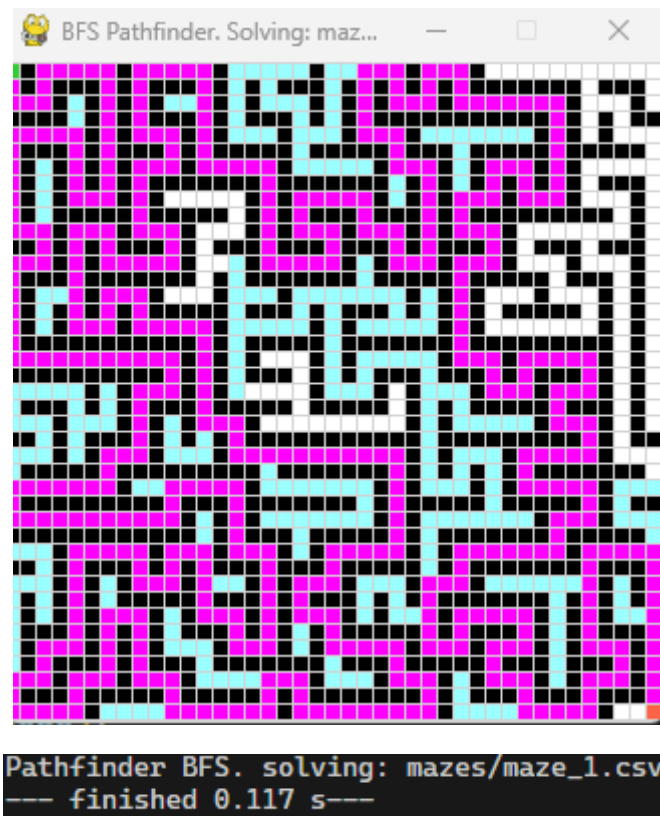
O algoritmo DFS, também conhecido como busca em profundidade, explora o máximo possível em uma direção antes de retroceder. No problema do labirinto, ele começa na posição inicial e explora a ramificação de caminhos até alcançar a posição final. O foco do DFS está na profundidade, o que significa que ele pode seguir um caminho até o final antes de voltar atrás e explorar outros caminhos. A classe DFS implementada mantém uma fronteira entre nós a serem explorados e uma lista de nós já explorados. Ele itera pela fronteira, expandindo os nós para encontrar caminhos até o objetivo.

O algoritmo A* utiliza as técnicas do algoritmo BFS e a heurística para encontrar o melhor caminho possível. Ele avalia os nós com base em uma função de custos estimados que combina o custo acumulado até o nó e uma estimativa do custo restante até o objetivo. O código em que é implementado o A* possui uma classe ‘Node’ para representar os nós e uma classe ‘PriorityQueue’ para manter a fronteira de

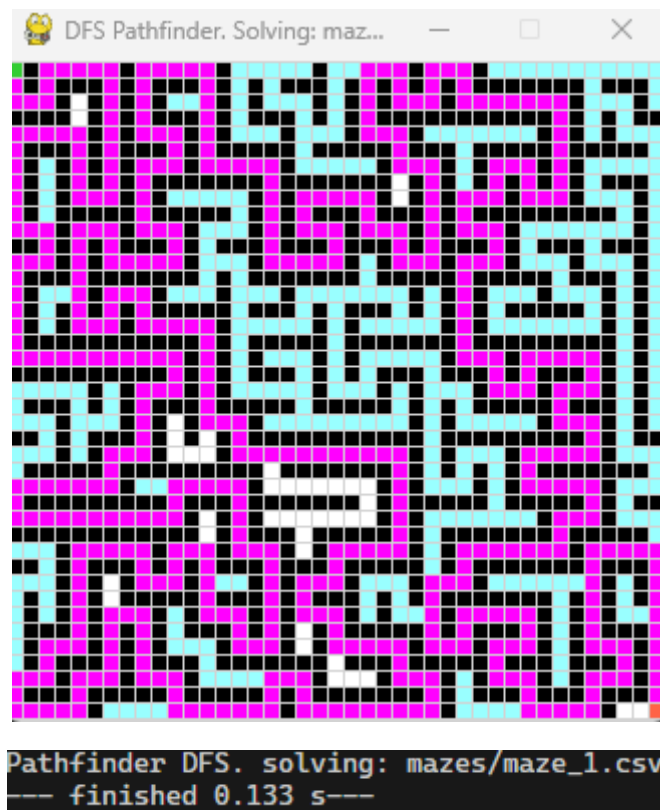
nós a serem explorados.

Estudo com tempo de execução e soluções encontradas cada estratégia de busca

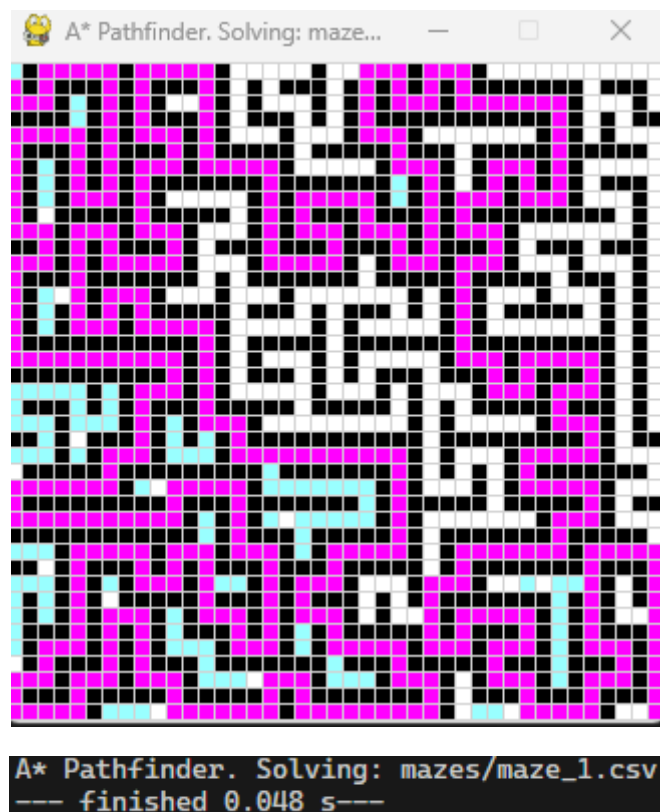
BFS:



DFS:



A*:



Após a realização dos testes e o conhecimento de cada algoritmo por trás da resolução do labirinto, podemos concluir que o algoritmo A* é o mais eficiente entre todos. Em seguida temos o BFS mostrando uma melhor eficiência do que o DFS. O DFS demonstrou o pior desempenho entre os testados, sendo perceptível até no mapa gerado, onde ele explorou uma maior quantidade de nós que não tinham necessidade. Em termos de tempo, no geral todos foram rápidos, porém em problemas mais complexos, os algoritmos que apresentaram o melhor tempo podem ter um desempenho muito superior aos demais.

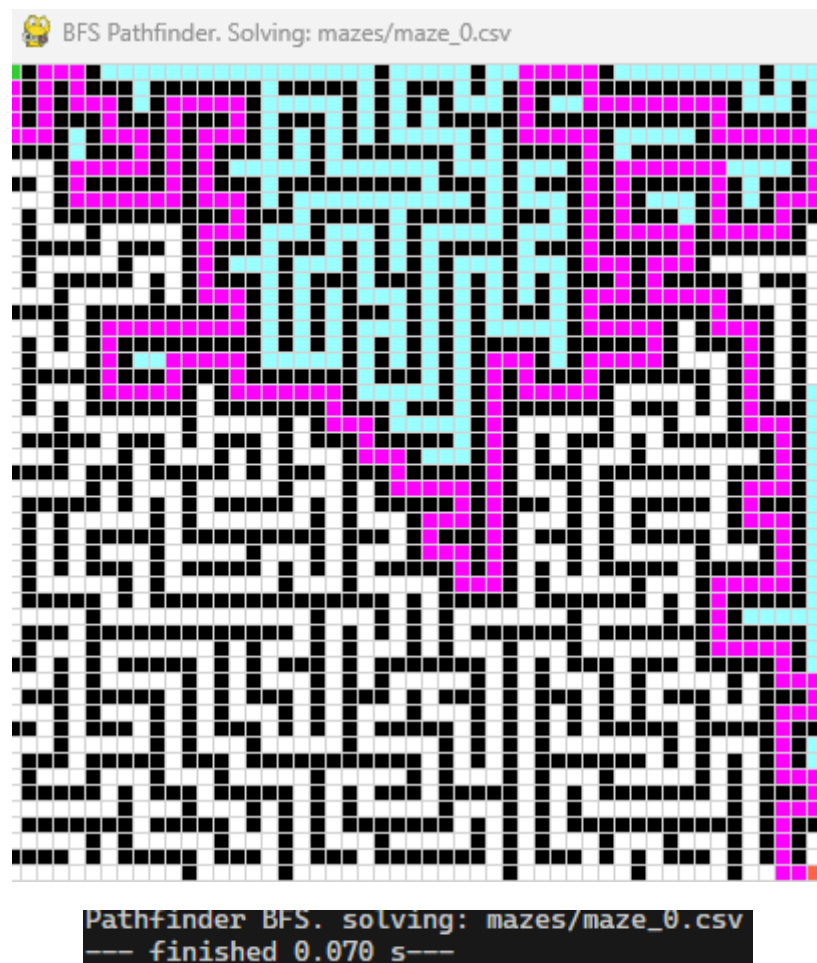
Alterando o tamanho do labirinto e realizando experimentos

Para alterar o tamanho do labirinto a ser gerado deve-se alterar o valor da variável 'num_row', vale ressaltar que os valores inseridos nessa variável tem que ser ímpares, caso contrário, apresentará erro na geração do labirinto.

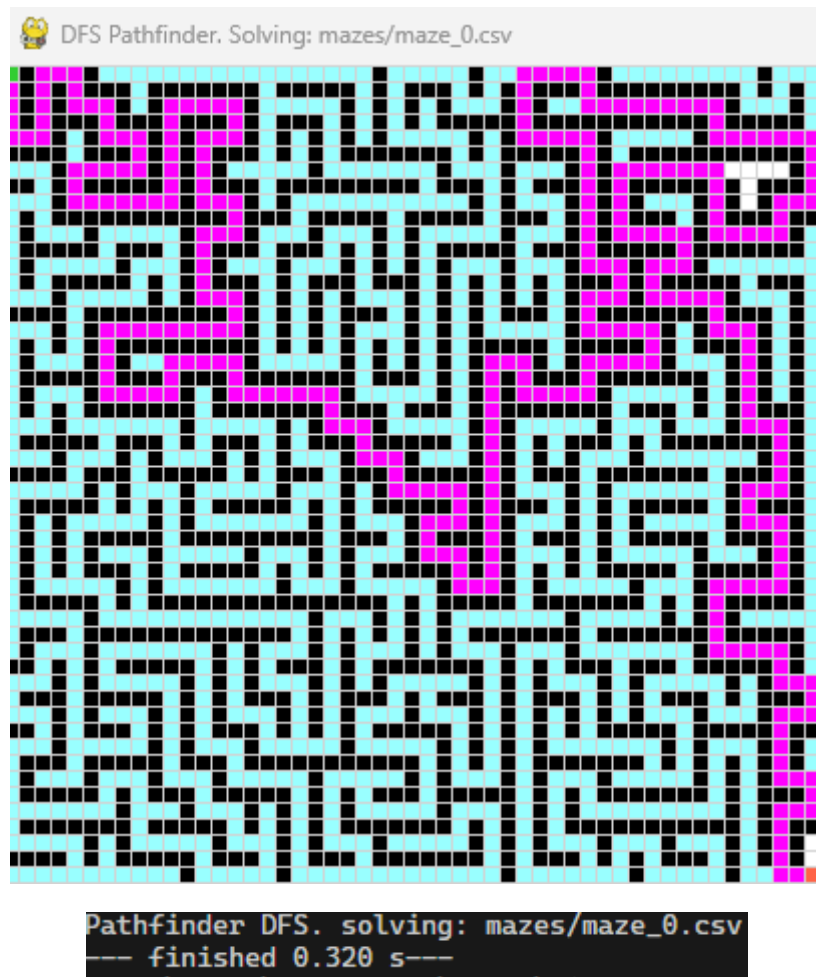
```
# initialize the grid array full of zeros
num_rows = 51
num_columns = num_rows
grid = np.zeros((num_rows, num_columns))
```

O valor em questão foi 51, aumentando em mais 10 linhas e 10 colunas em comparação com o labirinto anterior. A seguir serão apresentados os resultados dos 3 algoritmos.

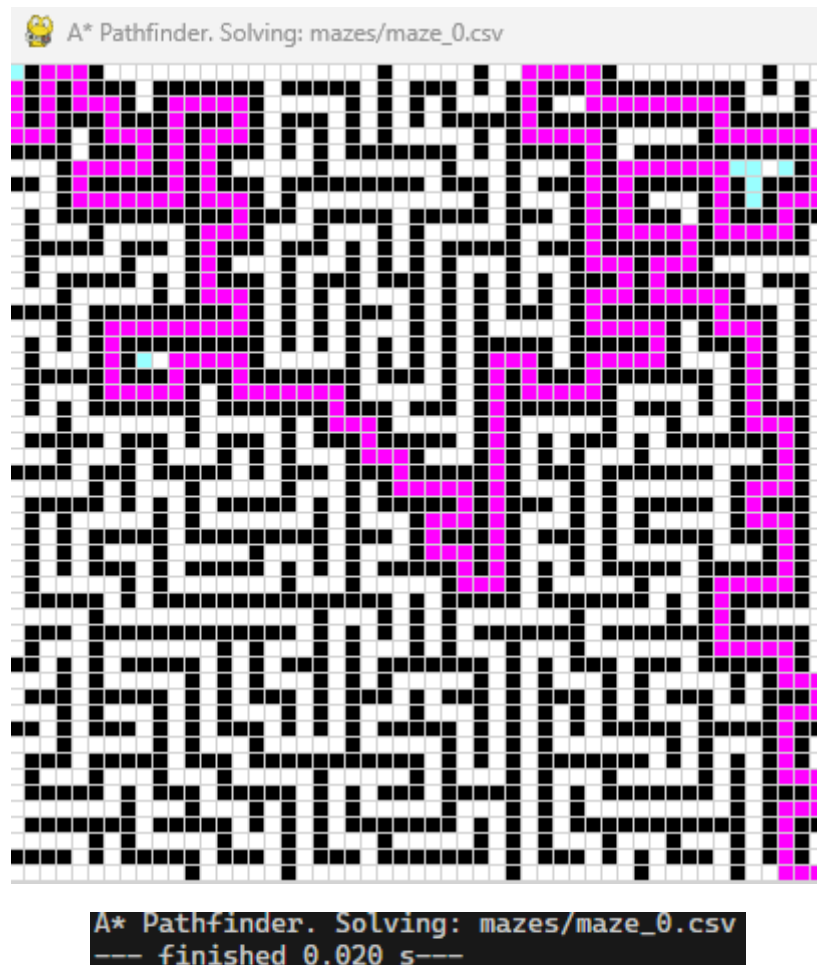
BFS:



DFS:



A*:

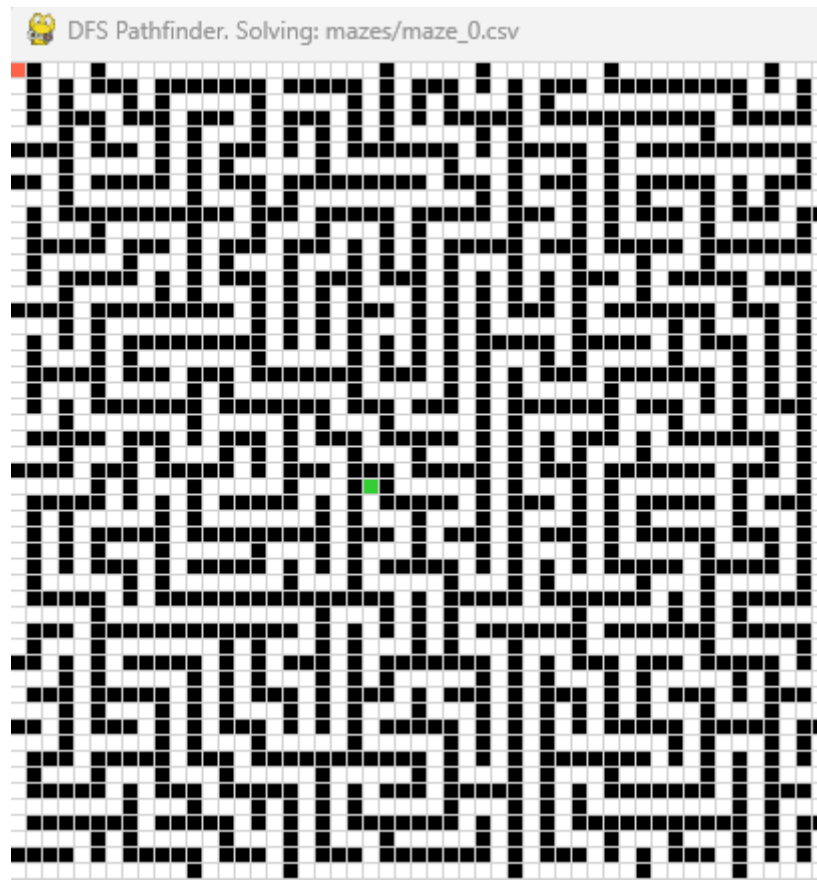


Com os experimentos realizados, constatamos que a eficácia do algoritmo A* é completamente superior aos demais, pois como podemos ver pela **Figura x**, o labirinto foi concluído com um caminho praticamente sem desvio do objetivo. Em contrapartida, os resultados dos outros dois algoritmos foram ineficientes em comparação ao A*. Sendo o BFS em segundo lugar, ele consegue completar em um tempo interessante, porém como visto na **Figura x** ele acabou realizando uns desvios para o canto superior do labirinto. Em último lugar temos o algoritmo DFS, sendo o pior resultado, ele concluiu o labirinto em um tempo muito acima dos demais e em quesito a caminhos explorados, foi praticamente o labirinto inteiro, isso se dá pelo seu comportamento de explorar até o fim um caminho para que vá para os demais.

Alterando a posição de início e fim do labirinto

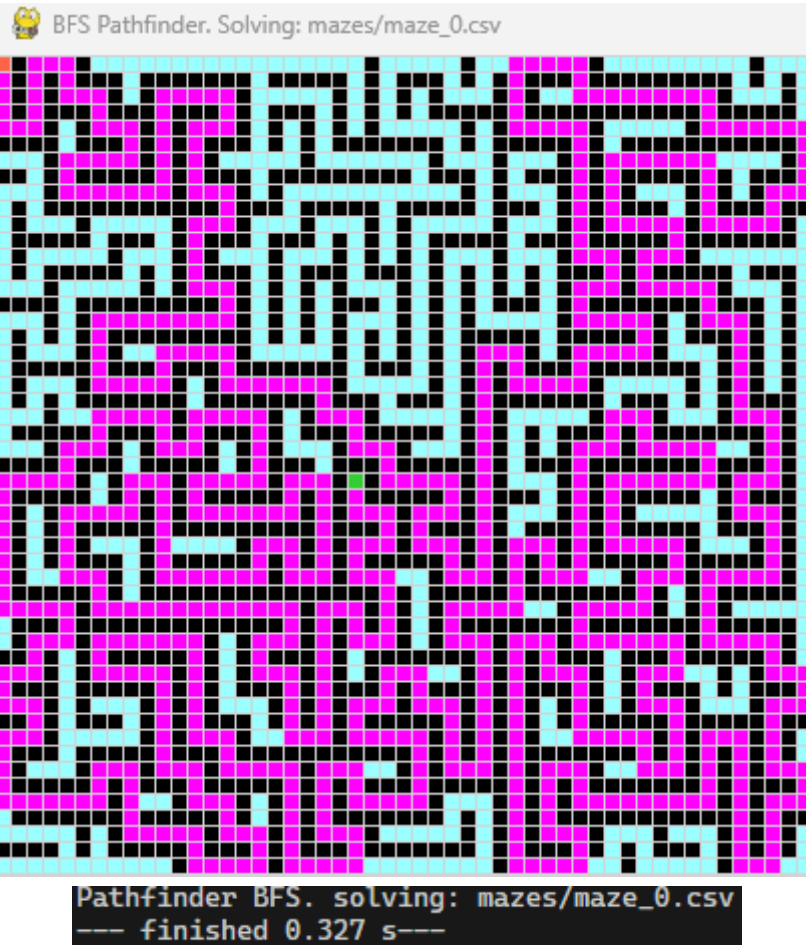
Como experimento, a posição de início e fim do labirinto foi trocado. O início

agora será próximo do centro do labirinto, na linha 26 e coluna 22 e o fim passará a ser a posição inicial original, linha 0 e coluna 0. Com isso obtemos o labirinto da seguinte forma:

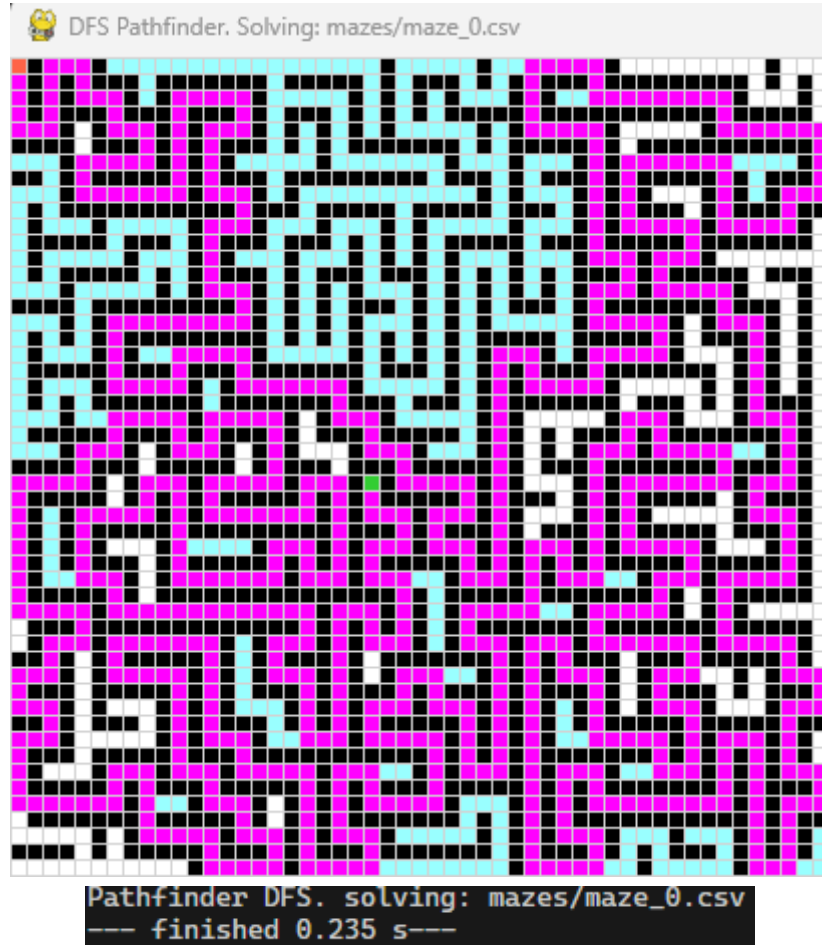


A partir deste labirinto, serão feitos testes com os três algoritmos e seus resultados serão apresentados a seguir:

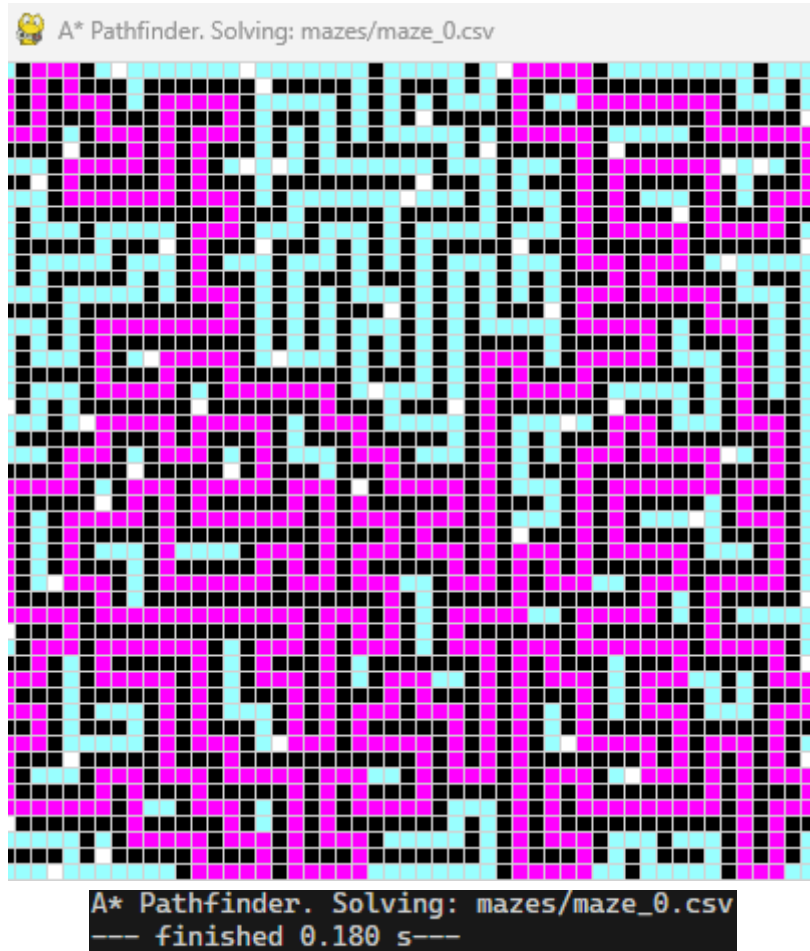
BFS:



DFS:



A*:

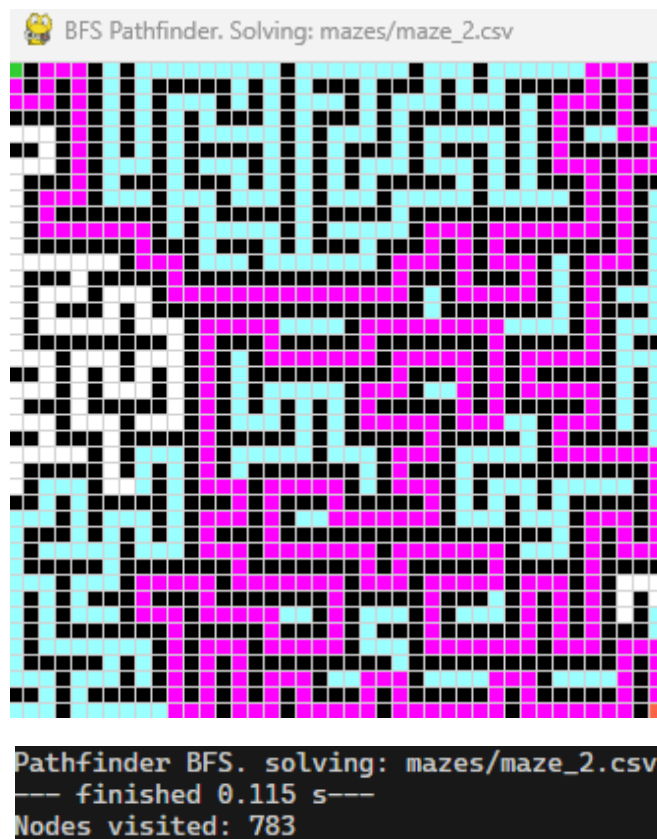


A alteração no ponto de início e fim influenciou de maneira negativa os algoritmos para a conclusão do problema. Como visto, o A* permaneceu sendo o melhor de todos mesmo tendo que percorrer uma grande distância. Em seguida temos o DFS que pela primeira vez não obteve o pior resultado, ele conseguiu ser o algoritmo que percorre a menor distância, visto que há muitos nós não visitados. E por último temos o BFS que teve o pior tempo e visitou todos os nós possíveis.

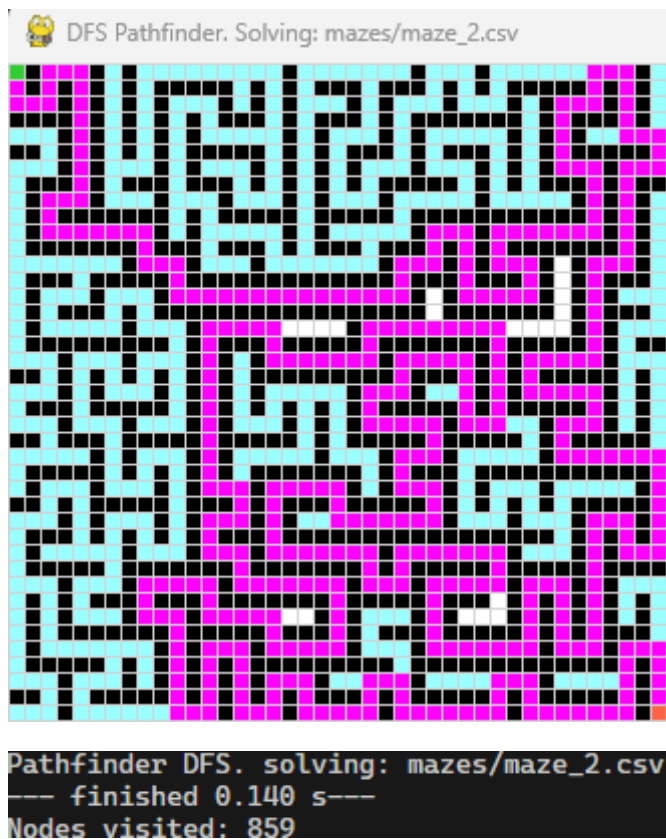
Implementando um contador de nós visitados

Para o teste a seguir, será selecionado um dos mapas gerados no arquivo original. O nome do labirinto se chama “maze2_csv” e nele será feito o teste de quantos nós cada algoritmo tem que visitar para solucionar o problema. A seguir será apresentado o resultado dos algoritmos:

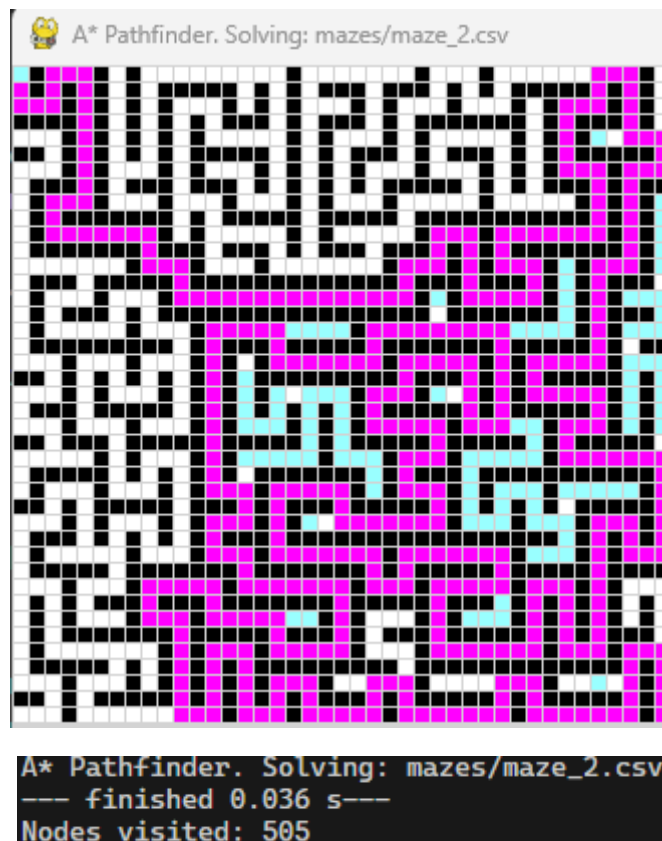
BFS:



DFS:



A*:



Como possível analisar, novamente o A* foi o mais eficiente para encontrar a saída do labirinto devido a sua natureza ele conseguiu percorrer a menor quantidade de nós. Em segundo lugar temos o BFS que visitou uma quantidade maior do que o A* e por fim levou um tempo maior para a conclusão. Por último temos o DFS que percorreu a maior quantidade de nós e levou o maior tempo para a conclusão.

Função para o cálculo de custo originalmente implementado

Dentro dos arquivos é possível encontrar o “helper_aStar.py”, ele é o responsável por toda a lógica envolvendo o algoritmo A*. Nele é disponibilizado a função “compute_node_cost”.

```
def compute_node_cost(pos, goal):  
    """  
    Parameters  
    -----  
    pos : tuple of 2 ints
```

```

        position of node whos cost want to compute.
goal : tuple of 2 ints
        position of goal.
Returns
-----
cost : float
        euclidean distance pos-goal
"""
x, y = pos
x_goal, y_goal = goal
cost = np.sqrt((x_goal-x)**2 + (y_goal-y)**2)
return cost

```

Como podemos ver, ele utiliza de um cálculo de custo conhecido. Utilizando a fórmula euclidiana para calcular a distância percorrida, essa é uma heurística comumente utilizada nos algoritmos A*. Essa fórmula representa a distância direta (em linha reta) entre dois pontos em um espaço bidimensional.

Implementado a distância cityblock na busca A* e realizando testes

Para implementar esse cálculo de distância no algoritmo de busca A* basta importarmos uma biblioteca `scipy.spatial.distance cityblock`. Após isso, é necessário ir a função “`compute_node_cost`” e alterar o código presente para:

```

def compute_node_cost(pos, goal):
    """
    Parameters
    -----
    pos : tuple of 2 ints
        position of node whos cost want to compute.
    goal : tuple of 2 ints
        position of goal.
    Returns
    -----
    cost : float
        euclidean distance pos-goal
    x, y = pos

```

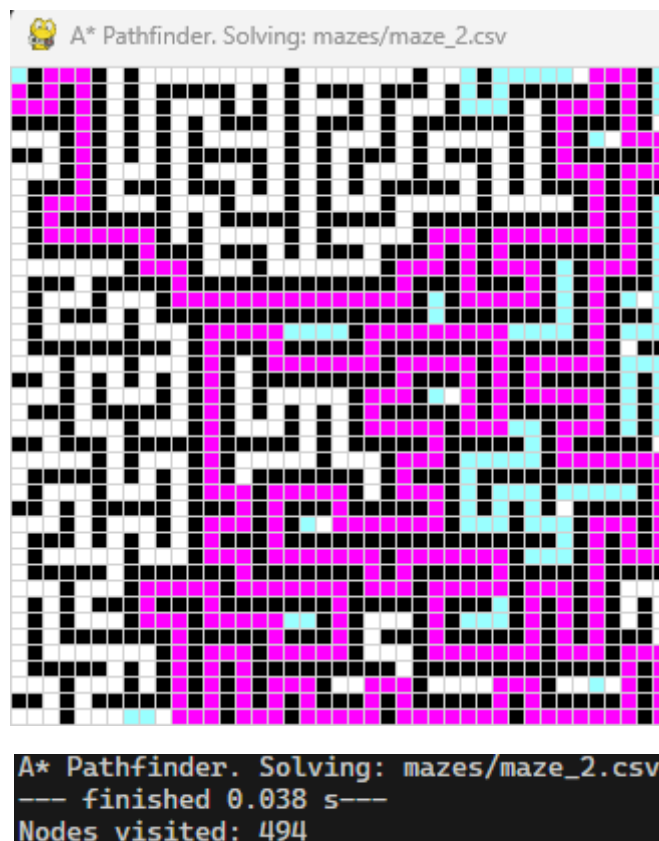
```

x_goal, y_goal = goal
cost = np.sqrt((x_goal-x)**2 + (y_goal-y)**2)
return cost
"""
cost = cityblock(pos, goal)
return cost

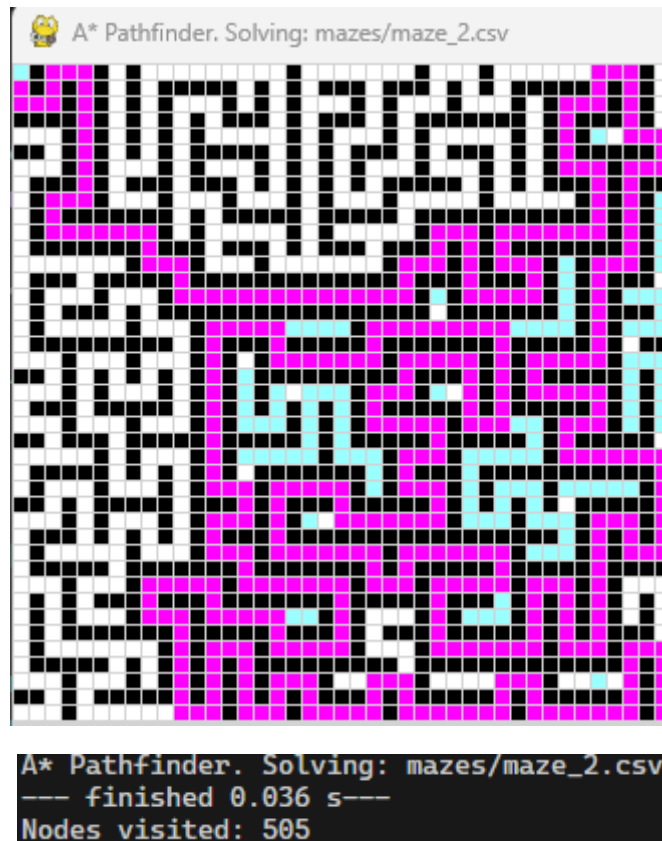
```

A seguir será feita uma comparação entre o método de cálculo da distância original e a cityblock.

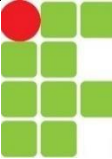
A* cityblock:



A* original:

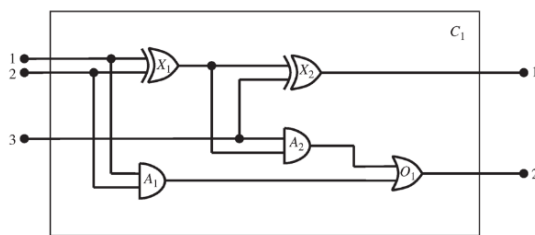


Em primeiras impressões podemos notar que o tempo para resolver é praticamente o mesmo, tendo apenas milésimos de segundos de diferença entre os dois. Mas se olharmos na quantidade de nós visitados é perceptível que o método cityblock conseguiu ser mais eficiente do que o método original, visitando uma menor quantidade de nó para a solução do problema. O cityblock possui como característica a soma das diferenças absolutas das coordenadas x e y entre dois pontos. Em um ambiente onde os movimentos só podem ocorrer na horizontal e na vertical (sem diagonais), a distância de Manhattan representa o número mínimo de movimentos necessários para mover-se de um ponto para outro, seguindo apenas direções perpendiculares.

| | | |
|---|--|-------------------------|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência Artificial | | Lista 1 |
| Professor: Prof. Dr. Murilo Vargues da Silva | | Data: 11/09/2023 |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

ATIVIDADE LÓGICA

Dado o somador lógico representado abaixo:



1. Terminar a codificação da instância geral e instância específica (slide 28);
2. Criar mais três consultar utilizando linguagem escrita (slide 29);
3. Transformar as três consultas novas em consultas de lógica de primeira ordem (slide 30);
4. Testar instância especificada e consultas utilizando prolog;
5. Gerar relatório com o passo a passo e prints e enviar no moodle.

1)

-Geral:

$\forall g \text{ Type}(h) = \text{XOR} \Rightarrow \text{Signal}(\text{Out}(1,h)) = 1 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n,h)) = 1 \wedge \exists m \text{ Signal}(\text{In}(m,h)) = 0$

$\forall g \text{ Type}(h) = \text{XOR} \Rightarrow \text{Signal}(\text{Out}(1,h)) = 0 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n,h)) = 1 \wedge \exists m \text{ Signal}(\text{In}(m,h)) = 1$

$\forall g \text{ Type}(i) = \text{AND} \Rightarrow \text{Signal}(\text{Out}(1,i)) = 1 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n,i)) = 1 \wedge \exists m \text{ Signal}(\text{In}(m,h)) = 1 \wedge m \neq n$

-Específica:

`Connected(Out(1,A1),In(2, O1))`

2) Para saída $C1 = 0$ e $C2 = 1$, podemos considerar as três consultas:

Caso 1

input 1: 1

input 2: 0

input 3: 1

Caso 2

input 1: 0

input 2: 1

input 3: 1

Caso 2

input 1: 0

input 2: 0

input 3: 0

3)

$\exists i_1, i_2, i_3 \text{ Signal(In}(1, C1))=i_1 \wedge \text{Signal(In}(2, C1))=i_2 \wedge \text{Signal(In}(3, C1))=i_3 \wedge$
 $\text{Signal(Out}(1, C1))=0 \wedge \text{Signal(Out}(2, C1)) = 1$

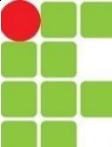
valores para i_1, i_2 e $i_3 = 1, 0$ e 1 , respectivamente

$\exists i_1, i_2, i_3 \text{ Signal(In}(1, C1))=i_1 \wedge \text{Signal(In}(2, C1))=i_2 \wedge \text{Signal(In}(3, C1))=i_3 \wedge$
 $\text{Signal(Out}(1, C1))=0 \wedge \text{Signal(Out}(2, C1)) = 1$

valores para i_1, i_2 e $i_3 = 0, 1$ e 1 , respectivamente

$\exists i_1, i_2, i_3 \text{ Signal(In}(1, C1))=i_1 \wedge \text{Signal(In}(2, C1))=i_2 \wedge \text{Signal(In}(3, C1))=i_3 \wedge$
 $\text{Signal(Out}(1, C1))=0 \wedge \text{Signal(Out}(2, C1)) = 1$

valores para i_1, i_2 e $i_3 = 0, 0$ e 0 .

| | | |
|---|--|--|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência Artificial | Lista 1 | |
| Professor: Prof. Dr. Murilo Vargas da Silva | Data: 25/09/2023 | |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

LISTA 1 – PROLOG

1. Digite o Programa 1.1, incluindo as regras que definem as relações avô e irmão, e realize as seguintes consultas:

Programa 1.1: *Uma árvore genealógica.*

```

pai (adão, cain) .
pai (adão, abel) .
pai (adão, seth) .
pai (seth, enos) .

```

1.3 Regras

Regras nos permitem definir novas relações em termos de outras relações já existentes. Por exemplo, a regra

```
avô(X,Y) :- pai(X,Z), pai(Z,Y) .
```

define a relação *avô* em termos da relação *pai*, ou seja, estabelece que *X* é avô de *Y* se *X* tem um filho *Z* que é pai de *Y*. Com essa regra, podemos agora realizar consultas tais como

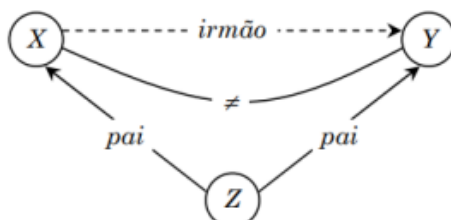
```

?- avô(X, enos) .
X = adão

```

Fatos e regras são tipos de *cláusulas* e um conjunto de cláusulas constitui um *programa lógico*.

Como mais um exemplo, vamos definir a relação *irmão* em termos da relação *pai*, já existente. Podemos dizer que duas pessoas distintas são irmãs se ambas têm o mesmo pai. Essa regra é representada pelo seguinte grafo:



Em Prolog, essa regra é escrita como:

```
irmão(X,Y) :- pai(Z,X), pai(Z,Y), X\=Y.
```

Código utilizado para realizar as consultas:

```
pai(adão,cain) .  
pai(adão,abel) .  
pai(adão,seth) .  
pai(seth,enos) .  
avô(X,Y):-pai(X,Z),pai(Z,Y) .  
irmão(X,Y):-pai(Z,X),pai(Z,Y),X\=Y.
```

a) Quem são filhos de Adão?

```
?- pai(adão,X) .  
X = cain ;  
X = abel ;  
X = seth .
```

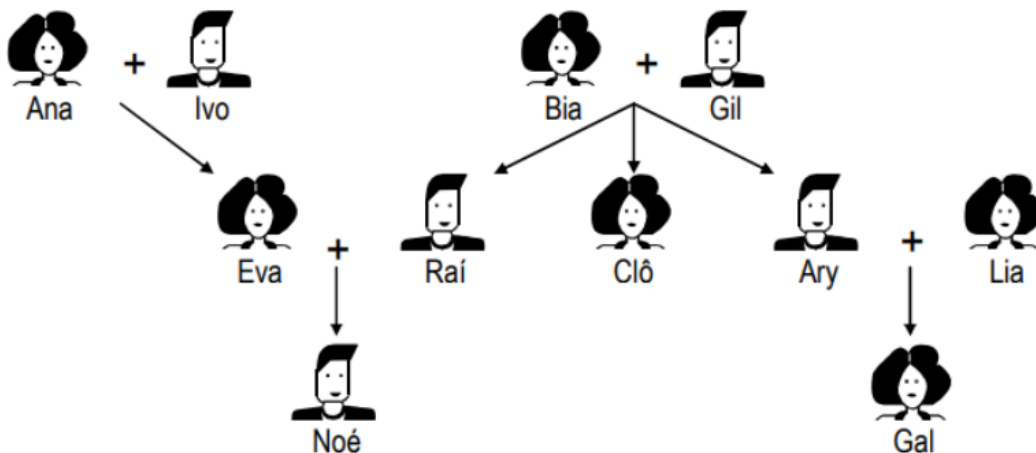
b) Quem são os netos de Adão?

```
?- avô(adão,X) .  
X = enos .
```

c) Quem são os tios de Enos?

```
?- irmão(seth,X) .  
X = cain ;  
X = abel ;  
false.
```

2. Considere a árvore genealógica a seguir:



Código utilizado para as consultas:


```

mãe(ana,eva) .
mãe(bia,rai) .
mãe(bia,clô) .
mãe(bia,ary) .
mãe(eva,noé) .
mãe(lia,gal) .
pai(ivo,eva) .
pai(rai,noé) .
pai(ary,gal) .
pai(gil,rai) .
pai(gil,clô) .
pai(gil,ary) .
mulher(ana) .
mulher(eva) .
mulher(bia) .
mulher(clô) .
mulher(lia) .
mulher(gal) .
homem(ivo) .
homem(rai) .
homem(noé) .
homem(gil) .
homem(ary) .
gerou(X,Y):-mãe(X,Y);pai(X,Y) .
irmão(X,Y):-pai(Z,X),pai(Z,Y),X\=Y,homem(X) .
irmã(X,Y):-pai(Z,X),pai(Z,Y),X\=Y,mulher(X) .
filho(X,Y):-gerou(Y,X),homem(X) .
filha(X,Y):-gerou(Y,X),mulher(X) .
tio(X,Y):-pai(Z,Y),mãe(M,Y),irmão(X,Z);irmão(X,M),Z\=M.
tia(X,Y):-pai(Z,Y),mãe(M,Y),irmã(X,Z);irmã(X,M),Z\=M.
primo(X,Y):-filho(X,Z),tio(Z,Y) .
prima(X,Y):-filha(X,Z),tio(Z,Y) .
avô(X,Y):-homem(X),gerou(X,Z),gerou(Z,Y) .

```

- a) Usando fatos, defina as relações pai e mãe. Em seguida, consulte o sistema para ver se suas definições estão corretas.

```

?- pai(X,noé) .
X = rai .

```

```

?- mãe(X,noé) .
X = eva .

```

- b) Acrescente ao programa os fatos necessários para definir as relações homem e mulher. Por exemplo, para estabelecer que Ana é mulher e Ivo é homem, acrescente os fatos mulher(ana) e homem(ivo).

```
?- mulher(eva).  
true.
```

```
?- homem(rai).  
true.
```

- c) Usando duas regras, defina a relação gerou(X,Y) tal que X gerou Y se X é pai ou mãe de Y. Faça consultas para verificar se sua definição está correta. Por exemplo, para a consulta gerou(X,eva) o sistema deverá apresentar as respostas X = ana e X = ivo.

```
?- gerou(X,eva).  
X = ana ;  
X = ivo.
```

- d) Usando relações já existentes, crie regras para definir as relações filho, filha, tio, tia, primo, prima, avô e avó. Para cada relação, desenhe o grafo de relacionamentos (Efetue leitura da apostila da USP), codifique a regra correspondente e faça consultas para verificar a corretude.

```
?- filho(noé,Y).  
Y = eva ;  
Y = rai.
```

```
?- filha(eva,Y).  
Y = ana ;  
Y = ivo.
```

```
?- tio(X,noé).  
X = ary ;  
false.
```

```
?- tia(X,noé).  
X = clô ;  
false.
```

```
?- primo(X,gal).  
X = noé ;  
false.
```

```
?- prima(X,noé).  
X = gal ;  
false.
```

```
?- avô(X,noé).  
X = ivo ;  
X = gil ;  
false.
```

```
?- avó(X,noé).  
X = ana ;  
X = bia ;  
false.
```

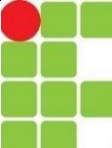
3. Codifique as regras equivalentes às seguintes sentenças:

a) Todo mundo que tem filhos é feliz.

```
feliz(X):-tem_filhos(X).  
tem_filhos(X):-pai(X,_);mãe(X,_).
```

b) Um casal é formado por duas pessoas que têm filhos em comum.

```
casal(X,Y):-tem_filhos_em_comum(X,Y).  
tem_filhos_em_comum(X,Y):-pai(X,Z),pai(Y,Z);mãe(X,Z),mãe(Y,Z).
```

| | | |
|---|--|--|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência Artificial | Lista 2 | |
| Professor: Prof. Dr. Murilo Vargues da Silva | Data: 02/10/2023 | |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

LISTA 2 – PROLOG

- Inclua no Programa 2.1 uma regra para o predicado dens(P,D), que relaciona cada país P à sua densidade demográfica correspondente D. Em seguida, faça consultas para descobrir:

```
% país(Nome, Área, População)
país(brasil, 9, 130).
país(china, 12, 1800).
país(eua, 9, 230).
país(índia, 3, 450).

% dens(País, Densidade)
dens(P,D) :- país(P,A,Q), D is Q/A.
```

- qual a densidade demográfica de cada um dos países;

```
?- dens(P,D).
P = brasil,
D = 14.4444444444444445 ;
P = china,
D = 150 ;
P = eua,
D = 25.555555555555557 ;
P = índia,
D = 150.
```

- se a Índia é mais populosa que a China.

```
?- dens(índia,I), dens(china,C), I>C.
false.
```

- Inclua no Programa 2.2 as informações da tabela abaixo e faça as consultas indicadas a seguir:

| Código | Nome | Salário | Dependentes |
|--------|------|-------------|-------------|
| 4 | Leo | R\$ 2500,35 | Lia, Noé |
| 5 | Clô | R\$ 1800,00 | Eli |
| 6 | Gil | R\$ 1100,00 | ----- |

```
% func(Código, Nome, Salário)
func(1, ana, 1000.90).
func(2, bia, 1200.00).
func(3, ivo, 903.50).
func(4, leo, 2500.35).
func(5, ivo, 1800.00).
func(6, ivo, 1100.00).

% dep(Código, Nome)
dep(1, ary).
dep(3, raí).
dep(3, eva).
dep(4, lia).
dep(4, noé).
dep(5, eli).
```

a. Quem tem salário entre R\$ 1500,00 e R\$ 3000,00?

```
?- func(C,N,S), S>=1500, S<=3000.
C = 4,
N = leo,
S = 2500.35 ;
C = 5,
N = ivo,
S = 1800.0 ;
false.
```

b. Quem não tem dependentes e ganha menos de R\$ 1200,00?

```
?- func(C,N,S), \+dep(C,_), S<1200.
C = 6,
N = ivo,
S = 1100.0.
```

- c. Quem depende de funcionário que ganha mais de R\$ 1700,00?

```
?- dep(C,N), func(C,_,S), S>1700.  
C = 4,  
N = lia,  
S = 2500.35 ;  
C = 4,  
N = noé,  
S = 2500.35 ;  
C = 5,  
N = eli,  
S = 1800.0.
```

3. Inclua no Programa 2.3 as seguintes regras:
- Um filme é longo se tem duração superior a 150 minutos.
 - Um filme é lançamento se foi lançado a menos de 1 ano.

```
% filme(Título, Gênero, Ano, Duração)  
filme('Uma linda mulher', romance, 1990, 119).  
filme('Sexto sentido', suspense, 2001, 108).  
filme('A cor púrpura', drama, 1985, 152).  
filme('Copacabana', comédia, 2001, 92).  
filme('E o vento levou', drama, 1939, 233).  
filme('Carrington', romance, 1995, 130).
```

```
% longo(Filme)  
longo(F):- filme(F,_,_,D), D > 150.
```

```
% lançamento(Filme)  
lançamento(F):- filme(F,_,A,_), A >= 2022.
```

```
?- longo(F).  
F = 'A cor púrpura' ;  
F = 'E o vento levou' ;  
false.
```

```
?- lançamento(F).  
false.
```

4. Codifique um programa contendo as informações da tabela abaixo e faça as consultas indicadas a seguir:

| Nome | Sexo | Idade | Altura | Peso |
|------|------|-------|--------|------|
| Ana | fem | 23 | 1.55 | 56.0 |
| Bia | fem | 19 | 1.71 | 61.3 |
| Ivo | masc | 22 | 1.80 | 70.5 |
| Lia | fem | 17 | 1.85 | 57.3 |
| Eva | fem | 28 | 1.75 | 68.7 |
| Ary | masc | 25 | 1.72 | 68.9 |

```
% pessoa(Nome, Sexo, Idade, Altura, Peso)
pessoa(ana, fem, 23, 1.55, 56.0).
pessoa(bia, fem, 19, 1.71, 61.3).
pessoa(ivo, masc, 22, 1.80, 70.5).
pessoa(lia, fem, 17, 1.85, 57.3).
pessoa(eva, fem, 28, 1.75, 68.7).
pessoa(ary, masc, 25, 1.72, 68.9).
```

a. Quais são as mulheres com mais de 20 anos de idade?

```
?- pessoa(N,S,I,A,P), S=fem, I>20.
N = ana,
S = fem,
I = 23,
A = 1.55,
P = 56.0 ;
N = eva,
S = fem,
I = 28,
A = 1.75,
P = 68.7 ;
false.
```

b. Quem tem pelo menos 1.70m de altura e menos de 65kg?

```
?- pessoa(N,S,I,A,P), A>=1.70, P<65.  
N = bia,  
S = fem,  
I = 19,  
A = 1.71,  
P = 61.3 ;  
N = lia,  
S = fem,  
I = 17,  
A = 1.85,  
P = 57.3 ;  
false.
```

c. Quais são os possíveis casais onde o homem é mais alto que a mulher?


```
?- pessoa(N,H,I,A,P), H=masc, pessoa(Q,W,E,R,T), W=fem, R>A.  
N = ivo,  
H = masc,  
I = 22,  
A = 1.8,  
P = 70.5,  
Q = lia,  
W = fem,  
E = 17,  
R = 1.85,  
T = 57.3  
  
N = ary,  
H = masc,  
I = 25,  
A = 1.72,  
P = 68.9,  
Q = lia,  
W = fem,  
E = 17,  
R = 1.85,  
T = 57.3  
  
N = ary,  
H = masc,  
I = 25,  
A = 1.72,  
P = 68.9,  
Q = eva,  
W = fem,  
E = 28,  
R = 1.75,  
T = 68.7
```


5. O peso ideal para uma modelo é no máximo $62.1 \cdot \text{Altura} - 44.7$. Além disso, para ser modelo, uma mulher precisa ter mais que 1.70m de altura e menos de 25 anos de idade. Com base nessas informações, e considerando a tabela do exercício anterior, defina um predicado capaz de recuperar apenas os nomes das mulheres que podem ser modelos.

```
% pessoa(Nome, Sexo, Idade, Altura, Peso)
pessoa(ana, fem, 23, 1.55, 56.0).
pessoa(bia, fem, 19, 1.71, 61.3).
pessoa(ivo, masc, 22, 1.80, 70.5).
pessoa(lia, fem, 17, 1.85, 57.3).
pessoa(eva, fem, 28, 1.75, 68.7).
pessoa(ary, masc, 25, 1.72, 68.9).

% modelo(Nome)
modelo(N) :- pessoa(N,S,I,A,P), S=fem, A>1.70, I<25, P<=(62.1*A-44.7).
```

```
?- modelo(N).
N = bia ;
N = lia ;
false.
```

| | | |
|---|--|-------------------------|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência Artificial | | Lista 3 |
| Professor: Prof. Dr. Murilo Vargas da Silva | | Data: 16/10/2023 |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

LISTA 3 – PROLOG

1. O programa a seguir associa a cada pessoa seu esporte preferido.

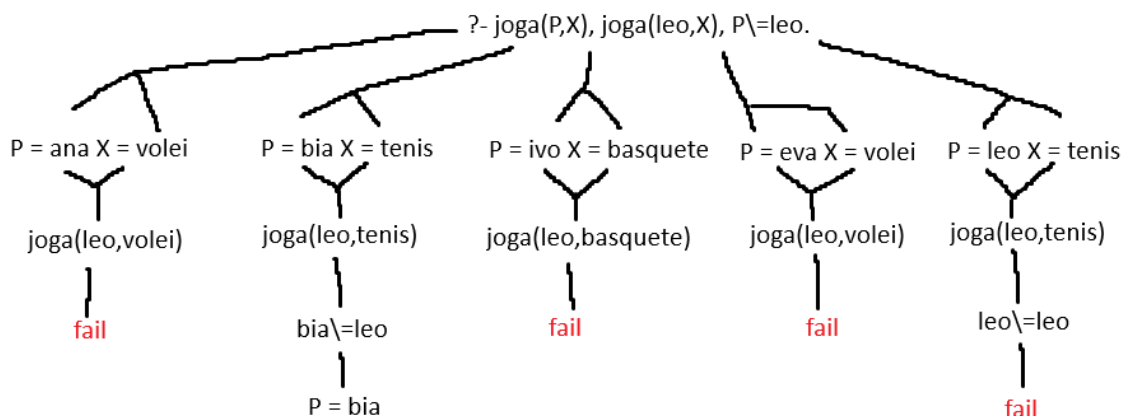
```
joga(ana, volei).
joga(bia, tenis).
joga(ivo, basquete).
joga(eva, volei).
joga(leo, tenis).
```

Suponha que desejamos consultar esse programa para encontrar um parceiro P para jogar com Leo. Então, podemos realizar essa consulta de duas formas:

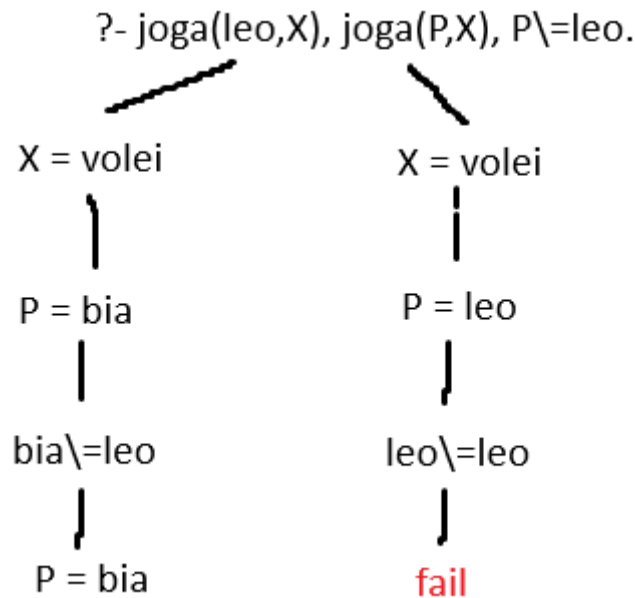
- a. `?- joga(P,X), joga(leo,X), P\=leo.`
b. `?- joga(leo,X), joga(P,X), P\=leo.`

Desenhe as árvores de busca construídas pelo sistema ao responder cada uma dessas consultas. Qual consulta é mais eficiente, por quê?

Resultado da árvore de busca da pesquisa (a):



Resultado da árvore de busca da pesquisa (b):



A consulta (b) é mais eficiente porque ela começa com uma restrição específica (Leo jogando volei) e, em seguida, encontra um parceiro que atende a essa restrição. Isso reduz a quantidade de backtracking necessário. A consulta (a) começa com a variável P e, em seguida, verifica a restrição em relação a Leo, o que pode levar a mais tentativas de unificação e, portanto, ser menos eficiente.

2. O predicado `num` classifica números em três categorias: positivos, nulo e negativos. Esse predicado, da maneira como está definido, realiza retrocesso desnecessário. Explique por que isso acontece e, em seguida, utilize cortes para eliminar esse retrocesso.

```
num(N,positivo) :- N>0.  
num(0,nulo).  
num(N,negativo) :- N<0.
```

Considerando o predicado `num`, ao fazermos uma consulta como por exemplo `num(5, C)`, ele irá achar a solução (`C = positivo`) e, em seguida fará o retrocesso para verificar se há outras soluções, o que é desnecessário para esse caso. O mesmo se aplica se entrarmos com um valor de N negativo, após o Prolog encontrar a solução (`C = negativo`) ele irá retornar e procurar

outras soluções.

Para solucionarmos esses retrocessos desnecessários, basta adicionar o operador de corte '!', que fará com que o Prolog não realize o retrocesso a partir desse ponto. O predicado então ficará da seguinte maneira:

```
num(N,positivo) :- N > 0, !.  
num(0,nulo).  
num(N,negativo) :- N < 0, !.
```

A adição do corte após as condições que satisfaçam as categorias "positivo" e "negativo" garante que o Prolog não faça retrocesso nessas regras após encontrar uma solução. Assim, o retrocesso desnecessário é eliminado, tornando o predicado mais eficiente. No entanto, a regra para o caso "nulo" não tem um corte, o que permite ao Prolog continuar a procurar por mais soluções se N for igual a zero.

3. Suponha que o predicado fail não existisse em Prolog. Qual das duas definições a seguir poderia ser corretamente usada para causar falhas?

- a. falha :- (1=1).
- b. falha :- (1=2).

A definição que ocasionará falha será a (b). Neste caso, a regra falha falhará porque a igualdade $1=2$ é avaliada como falsa, o que levará à falha da regra. Isso é equivalente a usar fail para indicar uma falha, mas sem o uso direto do predicado fail.

4. Considere o programa a seguir:

```
animal(cão).  
animal(canário).  
animal(cobra).  
animal(morcego).  
animal(gaivota).  
voa(canário).  
voa(morcego).  
voa(gaivota).  
dif(X,X) :- !, fail.
```

`dif(_,_).`

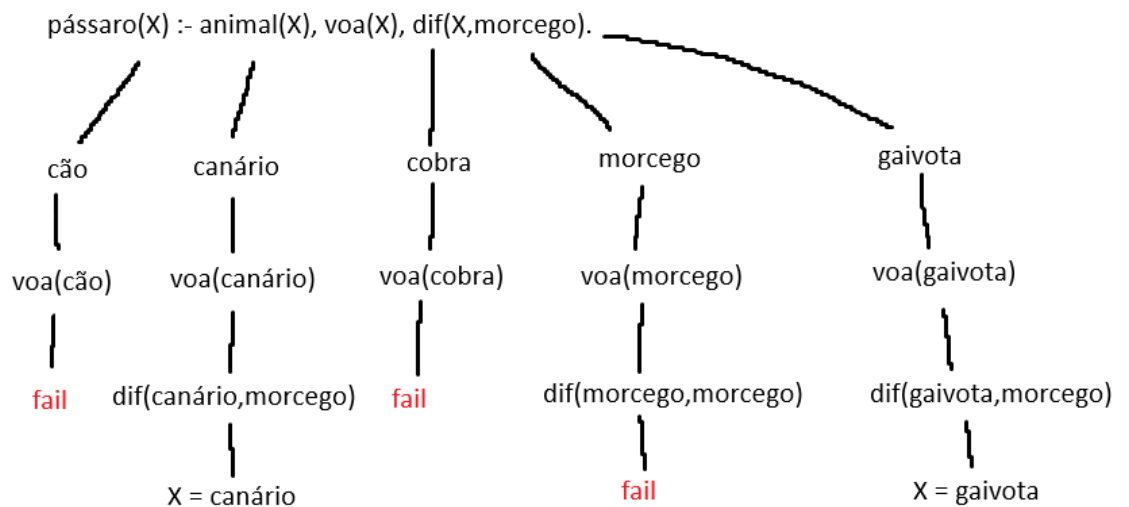
`pássaro(X) :- animal(X), voa(X), dif(X,morcego).`

Desenhe a árvore de busca necessária para responder a consulta

?- pássaro(X).

Em seguida, execute o programa para ver se as respostas do sistema correspondem àquelas que você encontrou.

Desenho da árvore de busca:



Execução do programa:

```
[trace] ?- pássaro(X).  
  Call: (10) pássaro(_30470) ? creep  
  Call: (11) animal(_30470) ? creep  
  Exit: (11) animal(cão) ? creep  
  Call: (11) voa(cão) ? creep  
  Fail: (11) voa(cão) ? creep  
  Redo: (11) animal(_30470) ? creep  
  Exit: (11) animal(canário) ? creep  
  Call: (11) voa(canário) ? creep  
  Exit: (11) voa(canário) ? creep  
  Call: (11) dif(canário, morcego) ? creep  
  Exit: (11) dif(canário, morcego) ? creep  
  Exit: (10) pássaro(canário) ? creep  
X = canário ;  
  Redo: (11) animal(_30470) ? creep  
  Exit: (11) animal(cobra) ? creep  
  Call: (11) voa(cobra) ? creep  
  Fail: (11) voa(cobra) ? creep  
  Redo: (11) animal(_30470) ? creep  
  Exit: (11) animal(morcego) ? creep  
  Call: (11) voa(morcego) ? creep  
  Exit: (11) voa(morcego) ? creep  
  Call: (11) dif(morcego, morcego) ? creep  
  Call: (12) fail ? creep  
  Fail: (12) fail ? creep  
  Fail: (11) dif(morcego, morcego) ? creep  
  Redo: (11) animal(_30470) ? creep  
  Exit: (11) animal(gaivota) ? creep  
  Call: (11) voa(gaivota) ? creep  
  Exit: (11) voa(gaivota) ? creep  
  Call: (11) dif(gaivota, morcego) ? creep  
  Exit: (11) dif(gaivota, morcego) ? creep  
  Exit: (10) pássaro(gaivota) ? creep  
X = gaivota.
```

5. Defina um predicado recursivo para calcular o produto de dois números naturais usando apenas soma e subtração.

Predicado recursivo:

```
produto(0, _, 0). % O produto de 0 com qualquer número é 0.
produto(X, Y, Resultado) :-
    X > 0, % Certifique-se de que X seja um número natural.
    X1 is X - 1, % Reduza X em 1.
    produto(X1, Y, ResultadoParcial), % Chamada recursiva.
    Resultado is ResultadoParcial + Y. % Adicione Y ao resultado parcial.
```

Resultado:

```
?- produto(5, 2, X).
X = 10 ;
false.
```

6. Defina um predicado recursivo exibir um número natural em binário.

Predicado recursivo:

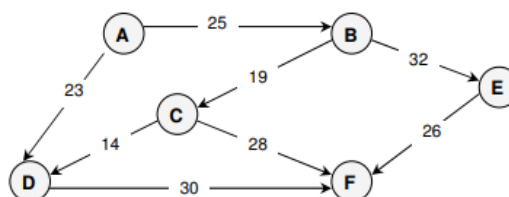
```
decimal_para_binario(0, "0"). % Caso base: 0 em decimal é "0" em binário.
decimal_para_binario(1, "1"). % Caso base: 1 em decimal é "1" em binário.

decimal_para_binario(N, Binario) :-
    N > 1,
    Resto is N mod 2, % Calcula o resto da divisão por 2.
    Quociente is N // 2, % Calcula o quociente da divisão por 2.
    decimal_para_binario(Quociente, BinarioAnterior), % Chamada recursiva.
    atom_concat(BinarioAnterior, Resto, Binario). % Concatena o resto ao resultado anterior.
```

Resultado:

```
?- decimal_para_binario(10, X).
X = '1010' ;
false.
```

7. O grafo a seguir representa um mapa, cujas cidades são representadas por letras e cujas estradas (de sentido único) são representados por números, que indicam sua extensão em km.



- a. Usando o predicado `estrada(Origem, Destino, Km)`, crie um programa para representar esse mapa.

```
%estrada(Origem, Destino, Km)
estrada(a, b, 25).
estrada(a, d, 23).
estrada(b, c, 19).
estrada(b, e, 32).
estrada(c, d, 14).
estrada(c, f, 28).
estrada(d, f, 30).
estrada(e, f, 26).
```

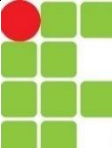
```
?- estrada(a, Destino, Distância).
Destino = b,
Distância = 25 ;
Destino = d,
Distância = 23.
```

- b. Defina a relação transitiva `dist(A,B,D)`, que determina a distância D entre duas cidades A e B.

```
%estrada(Origem, Destino, Km)
estrada(a, b, 25).
estrada(a, d, 23).
estrada(b, c, 19).
estrada(b, e, 32).
estrada(c, d, 14).
estrada(c, f, 28).
estrada(d, f, 30).
estrada(e, f, 26).
```

```
dist(A, B, D) :- estrada(A, B, D).
dist(A, B, D) :- estrada(A, C, D1), dist(C, B, D2), D is D1 + D2.
```

```
?- dist(a, f, D).
D = 72 ;
D = 88 ;
D = 83 ;
D = 53 ;
false.
```


| | | |
|---|--|--|
|  <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui</p> | <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA Campus Birigui Bacharelado em Engenharia de Computação</p> | |
| Disciplina: Inteligência Artificial | Lista 4 | |
| Professor: Prof. Dr. Murilo Vargues da Silva | Data: 23/10/2023 | |
| Nome do Aluno: Henrique Akira Hiraga | Prontuário: BI300838X | |

LISTA 4 – PROLOG

- Defina o predicado `ultimo(L,U)`, que determina o último item U numa lista L. Por exemplo, `ultimo([a,b,c],U)`, resulta em `U=c`.

```
ultimo([X],X).
ultimo([_|Y],X) :- ultimo(Y,X).

?- ultimo([a,b,c],X).
X = c ;
false.
```

- Defina o predicado `tam(L,N)`, que determina o número de itens N existente numa lista L. Por exemplo, `tam([a,b,c],N)`, resulta em `N=3`.

```
tam([],0).
tam([_|X],N) :- tam(X,N1), N is N1+1.

?- tam([a,b,c],N).
N = 3.
```

- Defina o predicado `soma(L,S)` que calcula a soma S dos itens da lista L. Por exemplo, `soma([4,9,1],S)` resulta em `S=14`.

```
soma([],0).
soma([X|Y],S) :- soma(Y,S1), S is X+S1.

?- soma([4,9,1],S).
S = 14.
```

4. Defina o predicado $\text{máx}(L,M)$ que determina o item máximo M na lista L .
Por exemplo, $\text{máx}[4,9,1],M$) resulta em $M=9$.

```
max([X],X) .  
max([X|Y],M) :- max(Y,M), M>=X.  
max([X|Y],X) :- max(Y,M), X>M.  
  
?- max([4,9,1],M) .  
M = 9 ;  
false.
```

5. Usando o predicado anexa , defina o predicado $\text{inv}(L,R)$ que inverte a lista L . Por exemplo, $\text{inv}([b, c, a], R)$ resulta em $R=[a,c,b]$.

```
anexa([],B,B) .  
anexa([X|A],B,[X|C]) :- anexa(A,B,C) .  
  
inv([],[]) .  
inv([X|A],B) :- inv(A,R), anexa(R,[X],B) .  
  
?- inv([b,c,a],R) .  
R = [a, c, b].
```

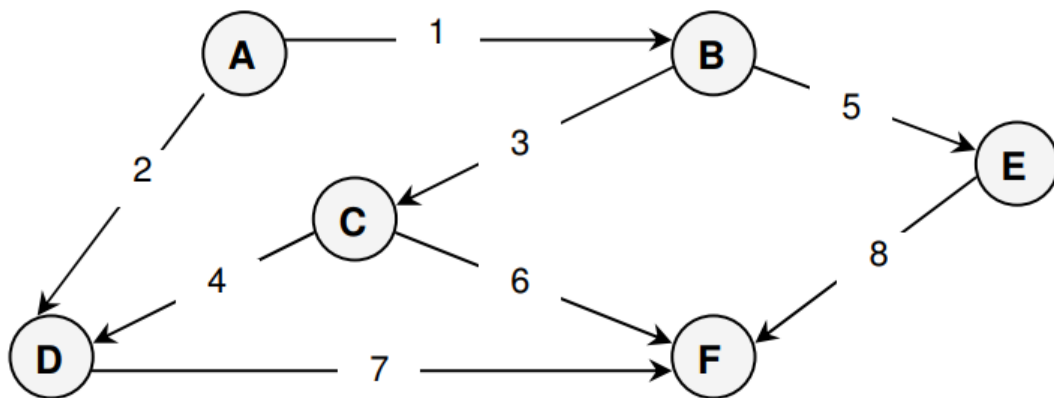
6. Usando o predicado inv , defina o predicado $\text{sim}(L)$ que verifica se uma lista é simétrica. Por exemplo, $\text{sim}([a,r,a,r,a])$ resulta em yes .

```
inv([],[]) .  
inv([X|A],B) :- inv(A,R), anexa(R,[X],B) .  
  
sim([]) .  
sim([X]) :- true,! .  
sim([X|A]) :- inv([X|A],[X|A]) .  
  
?- sim([a,r,a,r,a]) .  
true.
```

7. Usando a tabela $d(0,\text{zero}), d(1,\text{um}), \dots, d(9,\text{nove})$, defina o predicado $\text{txt}(D,P)$ que converte uma lista de dígitos numa lista de palavras. Por exemplo, $\text{txt}([7,2,1],P)$ resulta em $P=[\text{sete},\text{dois},\text{um}]$.

```
d(0, 'zero').  
d(1, 'um').  
d(2, 'dois').  
d(3, 'três').  
d(4, 'quatro').  
d(5, 'cinco').  
d(6, 'seis').  
d(7, 'sete').  
d(8, 'oito').  
d(9, 'nove').  
  
txt([], []).  
txt([X|Y], [X1|Y1]) :- d(X,X1), txt(Y,Y1).  
  
?- txt([7,2,1],P).  
P = [sete, dois, um].
```

8. O grafo a seguir representa um mapa, cujas cidades são representadas por letras e cujas estradas são representados por números.



- Usando o predicado $\text{estrada}(\text{Número}, \text{Origem}, \text{Destino})$, crie um programa para representar esse mapa.
- Defina o predicado $\text{rota}(A,B,R)$, que determina uma rota R (lista de estradas) que leva da cidade A até a cidade B .

```

estrada(1,a,b) .
estrada(2,a,d) .
estrada(3,b,c) .
estrada(4,c,d) .
estrada(5,b,e) .
estrada(6,c,f) .
estrada(7,d,f) .
estrada(8,e,f) .

rota(X,Y,[R]) :- estrada(R,X,Y) .
rota(X,Y,[R1|R2]) :- estrada(R1,X,Z) , rota(Z,Y,R2) .

?- rota(a,f,X) .
X = [1, 3, 6] ;
X = [1, 3, 4, 7] ;
X = [1, 5, 8] ;
X = [2, 7] ;
false .

```

9. Um retângulo é representado pela estrutura **retângulo**(A,B,C,D), cujos vértices são A, B, C e D, nesta ordem.
- Defina o predicado **regular**(R) que resulta em yes apenas se R for um retângulo cujos lados sejam verticais e horizontais.
 - Defina o predicado **quadrado**(R) que resulta em yes apenas se R for um retângulo cujos lados têm as mesmas medidas.

```

% Definir uma relação para verificar se uma linha é vertical
vertical(linha(ponto(X,Y1), ponto(X,Y2))) :- Y1 \= Y2.

% Definir uma relação para verificar se uma linha é horizontal
horizontal(linha(ponto(X1,Y), ponto(X2,Y))) :- X1 \= X2.

% Definir uma relação para verificar se um retângulo é regular (quadrado)
regular(retângulo(A, B, C, D)) :-
    vertical(linha(A, B)),
    horizontal(linha(B, C)),
    vertical(linha(C, D)),
    horizontal(linha(D, A)),
    A \== B,
    B \== C,
    C \== D.

% Definir uma relação para verificar se um retângulo é um quadrado
quadrado(R) :-
    regular(R),
    vertical(linha(A, C)),
    horizontal(linha(B, D)),
    A \== B,
    B \== C,
    C \== D.

```

```
?- regular(retângulo(ponto(0,0),ponto(1,0),ponto(1,1),ponto(0,1))).  
true.
```

10. Supondo que a base de dados esteja inicialmente vazia, indique qual será o seu conteúdo após terem sido executadas as seguintes consultas.

```
?- asserta( metal(ferro) ).  
?- assertz( metal(cobre) ).  
?- asserta( metal(ouro) ).  
?- assertz( metal(zinco) ).  
?- retract( metal(X) ).
```

Os metais serão adicionados para a lista, porém após utilizar o comando retract(metal(x)), toda a lista será apagada.

11. Implemente os predicados liga, desliga e lâmpada para que eles funcionem conforme indicado pelos exemplos a seguir:

```
?- liga, lâmpada(X).  
X = acesa  
Yes  
?- desliga, lâmpada(X).  
X = apagada  
Yes
```

```
:- dynamic lâmpada/1.  
  
lâmpada(apagada) .  
  
liga :-  
    retract(lâmpada(_)),  
    asserta(lâmpada(acesa)) .  
  
desliga :-  
    retract(lâmpada(_)),  
    asserta(lâmpada(apagada)) .
```

```
?- lâmpada(X).  
X = apagada.
```

```
?- liga, lâmpada(X).  
X = acesa.
```

```
?- lâmpada(X).  
X = acesa.
```

```
?- desliga, lâmpada(X).  
X = apagada.
```

```
?- lâmpada(X).  
X = apagada.
```

- 12.O predicado `asserta` adiciona um fato à base de dados, incondicionalmente, mesmo que ele já esteja lá. Para impedir essa redundância, defina o predicado `memorize`, tal que ele seja semelhante a `asserta`, mas só adicione à base de dados fatos inéditos.

```
memorize(X) :-  
    retractall(X), % Remove um fato se já houver um fato igual na base de dados  
    asserta(X).    % Acrescenta o novo fato
```

```
?- memorize(boss(gundyr)).  
true.
```

```
?- memorize(boss(artorias)).  
true.
```

```
?- memorize(boss(gundyr)).  
true.
```

```
?- listing(boss).  
:- dynamic boss/1.
```

```
boss(gundyr).  
boss(artorias).
```

```
true.
```

13. Suponha um robô capaz de andar até um certo local e pegar ou soltar objetos. Além disso, suponha que esse robô mantém numa base de dados sua posição corrente e as respectivas posições de uma série de objetos. Implemente os predicados `pos(Obj,Loc)`, `ande(Dest)`, `pegue(Obj)` e `solte(Obj)`, de modo que o comportamento desse robô possa ser simulado, conforme exemplificado a seguir:

```
?- pos(0,L).
```

```
0 = robô
```

```
L = garagem ;
```

```
0 = tv
```

```
L = sala ;
```

```
No
```

```
?-    pegue(tv),    ande(quarto),    solte(tv),  
    ande(cozinha).
```

```
anda de garagem até sala
```

```
pega tv
```

```
anda de sala até quarto
```

```
solta tv
```

```
anda de quarto até cozinha
```

```
Yes
```

```

:- dynamic pos/2.
:- dynamic posse/1.

% pos(Objeto, Local)
pos(robo, garagem).
pos(tv, sala).
posse([]). %objeto que o robo esta segurando, inicialmente indefinido

pegue(Obj) :-
    pos(robo, L1), %pos(robo, L1) = posição do robô
    pos(Obj, L2), %pos(Obj, L2) = posição do objeto
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %pega objeto
    retract(pos(robo,_)), asserta(pos(robo,L2)), %apaga posição original e define
posição atual
    format('anda de ~w até ~w\n', [L1,L2]), % define deslocamento
    format('pega ~w\n', [Obj]). % define objeto que o robô pegou

ande(Dest) :-
    pos(robo,L1), %pos do robô
    retract(pos(robo,_)), %remove posicao original do robô
    asserta(pos(robo, Dest)), %define nova posicao do robô
    format('anda de ~w até ~w\n', [L1, Dest]), %define deslocamento
    posse(Obj), %verifica objeto em posse do robô
    retract(pos(Obj,_)), %remove posição original do objeto em posse do robô
    asserta(pos(Obj, Dest)). %atualiza posição do objeto do robô

solte(Obj) :-
    posse(Obj), %verifica objeto em posse do robô
    retract(posse(_)), %remove objeto em posse do robô
    asserta(posse([])), %define que não há nenhum objeto em posse do robô
    format('solta ~w\n', [Obj]). %define robô soltando objeto em posse

?- pos(O,L).
O = robo,
L = garagem ;
O = tv,
L = sala.

?- pegue(tv).ande(quarto).solte(tv).ande(cozinha).
anda de garagem até sala
pega tv
anda de sala até quarto
solta tv
anda de quarto até cozinha
false.

```

14. Modifique o programa desenvolvido no exercício anterior de modo que, quando for solicitado ao robô pegar um objeto cuja posição é desconhecida, ele pergunte ao usuário onde está esse objeto e atualize a sua base de dados com a nova informação. Veja um exemplo:

```

?- pos(O,L).
O = robô
L = cozinha ;
O = tv

```


L = quarto ;

No

?- pegue(lixo), ande(rua), solte(lixo),
ande(garagem).

Onde está lixo? quintal

anda de cozinha até quintal

pega lixo

anda de quintal até rua

solta lixo

anda de rua até garagem

Yes

?- pos(0,L).

0 = robô

L = garagem ;

0 = lixo

L = rua ;

0 = tv

L = quarto ;

No

```

:- dynamic pos/2.
:- dynamic posse/1.

% pos(Objeto, Local)
pos(robo, garagem).
pos(tv, quarto).
posse([]). %objeto que o robo esta segurando, inicialmente indefinido

pegue(Obj) :-
    pos(robo, L1), %pos(robo, L1) = posição do robô
    pos(Obj, L2), %pos(Obj, L2) = posição do objeto
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %pega objeto
    retract(pos(robo,_)), asserta(pos(robo,L2)), %apaga posição original e define posição atual
    format('anda de ~w até ~w\n',[L1,L2]), % define deslocamento
    format('pega ~w\n',[Obj]). % define objeto que o robô pegou

pegue(Obj) :-
    format('Onde está ~w? ',[Obj]), %Pergunta qual a localização do objeto
    gets(L2), % Guarda local do objeto em L2
    asserta(pos(Obj,L2)), %Define posição do objeto cuja localização era desconhecida
    pos(robo,L1), %Posição do robô
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %Pega novo objeto
    retract(pos(robo,L1)), %Remove posição original do robô
    asserta(pos(robo,L2)), %Define posição atual do robô
    format('anda de ~w até ~w\n',[L1,L2]), %Define deslocamento
    format('pega ~w\n',[Obj]). %Define objeto que o robô pegou

^ande(Dest) :-
    pos(robo,L1), %pos do robô
    retract(pos(robo,_)), %remove posicao original do robô
    asserta(pos(robo,Dest)), %define nova posicao do robô
    format('anda de ~w até ~w\n',[L1,Dest]), %define deslocamento
    posse(Obj), %verifica objeto em posse do robô
    retract(pos(Obj,_)), %remove posição original do objeto em posse do robô
    asserta(pos(Obj,Dest)). %atualiza posição do objeto do robô

solte(Obj) :-
    posse(Obj), %verifica objeto em posse do robô
    retract(posse(_)), %remove objeto em posse do robô
    asserta(posse([])), %define que não há nenhum objeto em posse do robô
    format('solta ~w\n',[Obj]). %define robô soltando objeto em posse

gets(S) :-
    read_line_to_codes(user_input,C),
    name(S,C).

?- pegue(lixo).ande(rua).solte(lixo).ande(garagem).
Onde está lixo? quintal
anda de garagem até quintal
pega lixo
anda de quintal até rua
solta lixo
anda de rua até garagem
false.

?- pos(O,L).
O = robo,
L = garagem ;
O = lixo,
L = rua ;
O = tv,
L = quarto.

```

15. Acrescente também ao programa do robô o predicado leve(Obj,Loc), que leva um objeto até um determinado local. Por exemplo:

```

?- leve(tv,sala).

```

anda de garagem até quarto

pega tv

anda de quarto até sala

solta tv

Yes

```
:- dynamic pos/2.
:- dynamic posse/1.

% pos(Objeto, Local)
pos(robo, garagem).
pos(tv, quarto).
posse([]). %objeto que o robo esta segurando, inicialmente indefinido

pegue(Obj) :-
    pos(robo, L1), %pos(robo, L1) = posição do robô
    pos(Obj, L2), %pos(Obj, L2) = posição do objeto
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %pega objeto
    retract(pos(robo,_)), asserta(pos(robo,L2)), %apaga posição original e define posição atual
    format('anda de ~w até ~w\n', [L1,L2]), % define deslocamento
    format('pega ~w\n', [Obj]). % define objeto que o robô pegou

pegue(Obj) :-
    format('Onde está ~w? ', [Obj]), %Pergunta qual a localização do objeto
    gets(L2), % Guarda local do objeto em L2
    asserta(pos(Obj,L2)), %Define posição do objeto cuja localização era desconhecida
    pos(robo,L1), %Posição do robô
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %Pega novo objeto
    retract(pos(robo,L1)), %Remove posição original do robô
    asserta(pos(robo,L2)), %Define posição atual do robô
    format('anda de ~w até ~w\n', [L1,L2]), %Define deslocamento
    format('pega ~w\n', [Obj]). %Define objeto que o robô pegou

ande(Dest) :-
    pos(robo,L1), %pos do robô
    retract(pos(robo,_)), %remove posicao original do robô
    asserta(pos(robo,Dest)), %define nova posicao do robô
    format('anda de ~w até ~w\n', [L1,Dest]), %define deslocamento
    posse(Obj), %verifica objeto em posse do robô
    retract(pos(Obj,_)), %remove posição original do objeto em posse do robô
    asserta(pos(Obj,Dest)). %atualiza posição do objeto do robô

solte(Obj) :-
    posse(Obj), %verifica objeto em posse do robô
    retract(posse(_)), %remove objeto em posse do robô
    asserta(posse([])), %define que não há nenhum objeto em posse do robô
    format('solta ~w\n', [Obj]). %define robô soltando objeto em posse

gets(S) :-
    read_line_to_codes(user_input,C),
    name(S,C).

leve(Obj,Loc) :-
    pegue(Obj),
    ande(Loc),

    solte(Obj).
```

```
?- pos(O,L).  
O = robo,  
L = garagem ;  
O = lixo,  
L = rua ;  
O = tv,  
L = quarto.
```

```
?- leve(tv,sala).  
anda de garagem até quarto  
pega tv  
anda de quarto até sala  
solta tv  
true ;
```