GCC1734 – Inteligência Artificial
Notas de Aula
Agentes, Espaço de Estados e Formulação de Problemas de Busca

Prof. Eduardo Bezerra

Sumário

1 Agentes **3**
1.1 O que é um agente? . . . 3
1.2 Racionalidade . . . 4

2 Agentes Reflexos e Agentes Planejadores **5**
2.1 Agente Reflexo . . . 5
2.2 Agente Planejador . . . 5
2.3 Por que planejamento supera reflexo? . . . 5

3 Ambiente **7**
3.1 O que o ambiente define formalmente . . . 7
3.2 Utilidade Esperada . . . 8

4 Tipos de Ambientes **10**
4.1 Observabilidade . . . 11
4.2 Determinismo . . . 11
4.3 Dinamismo . . . 11
4.4 Número de Agentes . . . 12
4.5 Conhecimento da Física do Ambiente . . . 12
4.6 Resumo . . . 13

5 Estado do Mundo vs. Estado de Busca **13**
5.1 Estado do Mundo . . . 13
5.2 Estado de Busca . . . 14
5.3 Exemplo: dois problemas no Pac-Man . . . 14

6 Problemas de Busca **15**
6.1 Formulação . . . 15
6.2 Planos e Utilidade . . . 17
6.3 Exemplo: Pac-Man (Eat-All-Dots) . . . 18
6.4 Exemplo: Quebra-cabeça de 8 Peças . . . 20

7 Resolução de um Problema de Busca **23**
7.1 Grafo de Espaço de Estados 23
7.2 Árvore de Busca (Search Tree) 24
7.3 Algoritmo de Busca 26

8 Atividades intra-classe **33**
8.1 Carro Autônomo 33
8.2 Labirinto 34
8.3 Perguntas extras (para discussão em sala de aula) 34

Objetivos

Ao final da leitura, você deve ser capaz de:

- Definir formalmente um agente racional.
- Caracterizar tipos de ambientes.
- Formular corretamente um problema de busca.
- Calcular tamanho de espaço de estados usando o *princípio multiplicativo*.
- Diferenciar estado do mundo e estado de busca.

1 Agentes

O problema central da Inteligência Artificial é a construção de um *agente racional*. Antes de definir o que significa ser *racional*, precisamos entender o que é um agente e como ele se relaciona com o mundo ao seu redor.

1.1 O que é um agente?

Um *agente* é qualquer entidade que percebe o ambiente em que está inserida e toma decisões sobre como agir nele. Essa definição é deliberadamente ampla: um agente pode ser um robô físico, um programa de computador, um veículo autônomo, ou até mesmo um personagem em um jogo.

O que caracteriza um agente é a presença de três elementos:

- **Sensores**: mecanismos pelos quais o agente percebe o ambiente. Para um robô, isso pode ser uma câmera ou um sensor de distância. Para um programa, pode ser a leitura de um arquivo ou o recebimento de dados de uma API.
- **Atuadores**: mecanismos pelos quais o agente age sobre o ambiente. Para um robô, rodas ou braços mecânicos. Para um programa, a escrita em um banco de dados ou o envio de uma requisição.
- **Ambiente**: tudo aquilo que está fora do controle direto do agente, mas que pode influenciar (e ser influenciado por) suas ações.

A relação entre esses três elementos define o ciclo fundamental de operação de qualquer agente: *perceber* o estado do ambiente por meio dos sensores, *decidir* qual ação tomar, e *agir* por meio dos atuadores, o que geralmente altera o *estado do mundo* e gera novas percepções.

Pergunta 1

Escolha um agente do seu cotidiano (ex.: Waze, filtro de spam, aspirador robô). Liste, nesse exemplo: (i) quais são as **percepções** (sensores), (ii) quais são as **ações** (atuadores), e (iii) o que você considera parte do **ambiente** (tudo que está fora do controle direto do agente).

A Figura 1 ilustra o ciclo fundamental de operação de um agente. O agente e o ambiente são entidades distintas que interagem continuamente: o agente recebe *percepções* do ambiente por meio de seus sensores, processa essas informações para *decidir* qual ação é mais adequada, e então *age* sobre o ambiente por meio de seus atuadores. Essa ação, por sua vez, altera o estado do mundo, o que gera novas percepções e reinicia o ciclo. Observe que o ciclo não tem fim natural: um agente em operação percebe, decide e age continuamente, adaptando seu comportamento à medida que o ambiente evolui.

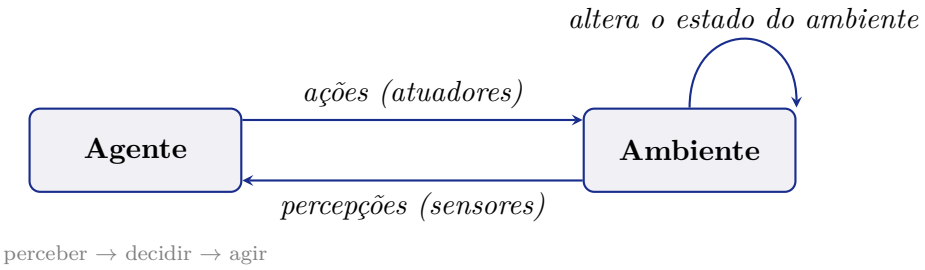

Figura 1: Ciclo fundamental de operação de um agente: o agente percebe o ambiente por meio de sensores, decide qual ação tomar, e age por meio de atuadores, alterando o ambiente e gerando novas percepções.

1.2 Racionalidade

Dado que um agente pode tomar diferentes ações, surge a pergunta: *o que distingue um agente que age bem de um que age mal?* Essa é exatamente a questão que o conceito de racionalidade responde.

Um agente é **racional** quando seleciona ações que maximizam sua **utilidade esperada**, isto é, quando suas decisões são as melhores possíveis dado o que ele sabe sobre o ambiente e sobre as consequências de suas ações. Dois aspectos dessa definição merecem atenção.

- Primeiro, racionalidade diz respeito à *decisão tomada*, não ao processo cognitivo que a gerou. Um agente pode chegar à decisão certa por um caminho simples ou por um raciocínio elaborado. O que importa é que a decisão seja a melhor disponível dadas as informações que o agente possui.
- Segundo, racionalidade não é o mesmo que perfeição. Um agente racional faz o melhor que pode *com o que sabe*. Se o ambiente é incerto ou as informações são incompletas, um agente racional pode tomar uma decisão que se revela equivocada, e ainda assim ter agido racionalmente, pois sua escolha era a melhor possível dado o conhecimento disponível no momento.

Pergunta 2

Um médico prescreve o melhor tratamento disponível com base em um diagnóstico correto, mas o paciente não melhora por razões imprevisíveis. O médico agiu de forma irracional? Explique sua resposta destacando a diferença entre **qualidade da decisão** e **resultado obtido**.

2 Agentes Reflexos e Agentes Planejadores

Definido o que é um agente racional, surge uma questão prática: *como* o agente deve escolher suas ações? Existem duas abordagens fundamentalmente diferentes.

2.1 Agente Reflexo

Um **agente reflexo** escolhe sua próxima ação baseando-se exclusivamente no estado atual percebido. Ele não considera o futuro, não simula consequências e não mantém nenhum modelo interno do ambiente. A decisão é imediata: dado o estado s, execute a ação a.

Essa abordagem é simples e computacionalmente barata. Em certos ambientes (especialmente os muito simples ou onde a melhor ação em cada situação pode ser determinada localmente) ela funciona bem.

O problema aparece quando a melhor ação em um dado momento não é aquela que parece mais atraente imediatamente, mas sim aquela que abre as melhores possibilidades no futuro. Um agente reflexo, por definição, não tem como perceber essa diferença.

2.2 Agente Planejador

Um **agente planejador** mantém um modelo interno do ambiente e, antes de agir, simula mentalmente as consequências de diferentes sequências de ações. Ele não pergunta apenas "qual é a melhor ação agora?", mas sim "qual sequência de ações me leva ao melhor resultado final?"

Para fazer isso, o agente precisa de:

- Um modelo de como o ambiente responde a cada ação (o que chamamos de *modelo de transição* ou *função sucessora*);
- Um critério para avaliar resultados (o que chamamos de *utilidade* ou *custo*);
- Uma forma de explorar sistematicamente as possibilidades (o que chamamos de *algoritmo de busca*).

Esses três elementos juntos formam a base de tudo que será estudado nas próximas aulas.

Pergunta 3

No contexto deste texto, dizer que o agente tem um "modelo interno" significa que ele consegue **simular** as consequências de ações antes de agir. Explique isso com um exemplo: descreva o que a função $T(s, a)$ codifica em um cenário simples (por exemplo, um robô em uma grade ou um labirinto). Em seguida, explique por que um agente **sem** esse modelo (como o agente reflexo) não consegue simular nada antes de agir.

2.3 Por que planejamento supera reflexo?

Considere um labirinto simples com uma bifurcação (célula $[1, 1]$ da Figura 2). O caminho para baixo parece mais curto, mas leva a um beco sem saída. O caminho da direita parece mais longo, mas conduz à saída.

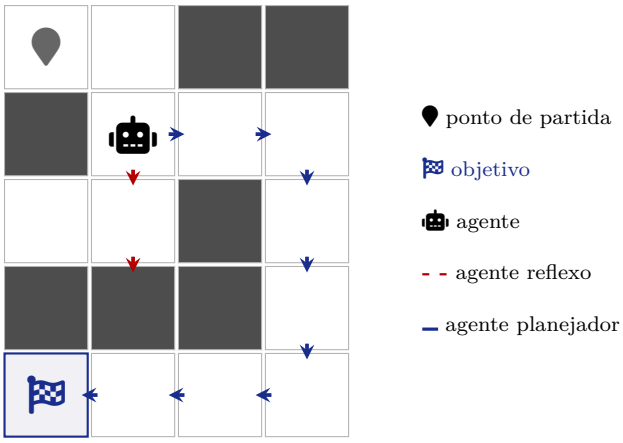

Figura 2: Labirinto com bifurcação em $(1, 1)$. O agente reflexo (tracejado vermelho) desce pela coluna 1, que parece mais próxima do objetivo, mas encontra um beco sem saída em $(1, 3)$. O agente planejador (azul) antecipa esse bloqueio e escolhe o caminho pela direita, contornando o labirinto até o objetivo (🏁).

A Figura 2 ilustra por que o agente reflexo toma a decisão errada na bifurcação em $(1, 1)$. Naquele ponto, o agente enxerga duas opções: descer para $(1, 2)$ ou virar à direita para $(2, 1)$. Suponha que o agente conheça a posição da saída em $(0, 4)$ e use a distância até ela como critério de decisão, uma estratégia intuitiva e razoável. Nesse caso, $(1, 2)$ parece melhor: está mais perto da saída do que $(2, 1)$. O agente reflexo escolhe descer.

O problema é que essa avaliação considera apenas o passo imediato, sem simular o que vem depois. O agente não sabe (porque não verifica) que a célula $(1, 3)$ é uma parede e que não há como chegar à saída por essa coluna. A informação sobre o objetivo existe, mas não é suficiente sem a capacidade de antecipar consequências.

O agente planejador, ao contrário, simula ambos os caminhos antes de agir. Ele descobre que o caminho pela coluna 1 termina em beco, enquanto o caminho pela coluna 3, apesar de parecer mais longo no primeiro passo, conduz à saída. Com essa informação, ele escolhe virar à direita em $(1, 1)$, não porque essa direção pareça melhor *agora*, mas porque é melhor *ao final*.

Esse exemplo captura a limitação fundamental do agente reflexo: **avaliar bem o passo imediato não é suficiente para garantir decisões globalmente corretas**. O agente pode conhecer o objetivo e ainda assim falhar, não por falta de informação sobre *onde* quer chegar, mas por não simular *como* chegar lá.

Pergunta 4

No labirinto da Figura 2, o agente reflexo usa a distância até a saída como critério e escolhe descer na bifurcação em $(1, 1)$. Por que essa decisão falha? Identifique a informação que o agente **não** considerou e explique como o agente planejador a utilizaria.

Pergunta 5

Considere um agente reflexo que joga xadrez capturando sempre a peça de maior valor disponível no momento. Dê um exemplo (pode ser descrito em palavras, sem diagrama) em que essa estratégia leva a uma desvantagem (por exemplo, perder a rainha na jogada seguinte). O que um agente planejador faria de diferente nesse caso?

3 Ambiente

Um agente nunca atua no vazio. Toda decisão que ele toma, toda ação que executa, ocorre em relação a algo externo a ele, algo que o influencia e que é influenciado por ele. Esse algo é o **ambiente**. O ambiente pode ser definido como tudo aquilo que:

- está fora do controle direto do agente;
- pode influenciar (e ser influenciado por) as ações do agente.

Essa definição é mais rica do que parece. Ela implica que o ambiente não é apenas o espaço físico em que o agente se encontra, mas qualquer fonte de influência externa. Para um robô, o ambiente inclui o chão, os obstáculos e a gravidade. Para um programa de xadrez, o ambiente é o tabuleiro e as regras do jogo. Para um agente de compras online, o ambiente inclui preços, estoques e o comportamento de outros compradores.

3.1 O que o ambiente define formalmente

Do ponto de vista do agente, o ambiente é o que determina as *regras do jogo*. Formalmente, ele especifica três coisas:

- **O conjunto de estados possíveis**: todas as configurações que o ambiente pode assumir. É o universo dentro do qual o agente existe e age.
- **Como ações transformam estados**: dado um estado atual e uma ação do agente, o ambiente determina qual será o estado resultante. Essa relação é o que chamamos de *modelo de transição*, e será formalizada adiante como uma função T.
- **Se há ou não incerteza nas transições**: em alguns ambientes, a mesma ação sempre produz o mesmo resultado. Em outros, o resultado é imprevisível; a mesma ação pode levar a estados diferentes dependendo de fatores que o agente não controla.

Essa última distinção (entre ambientes previsíveis e imprevisíveis) é particularmente importante, pois influencia diretamente o que significa agir racionalmente. Um agente em um ambiente totalmente previsível pode planejar com precisão. Um agente em um ambiente incerto precisa raciocinar sobre probabilidades e consequências esperadas. As duas situações exigem estratégias fundamentalmente diferentes, como veremos a seguir.

Ambiente Determinístico

Um ambiente é **determinístico** quando, para cada estado s e ação a, existe exatamente um estado sucessor s'. Formalmente:

$$s' = T(s, a)$$

O mesmo par (s, a) sempre produz o mesmo resultado, sem exceção. Isso significa que um agente que conhece o estado atual e o modelo de transição T pode prever com exatidão todos os estados futuros antes de executar qualquer ação, uma propriedade muito valiosa para o planejamento.

Exemplos de ambientes determinísticos incluem o quebra-cabeça de 8 peças, onde cada movimento tem exatamente um resultado, e a busca em grafos sem aleatoriedade, onde cada aresta leva a um único nó sucessor.

Pergunta 6

Em um ambiente determinístico, conhecer o estado atual e a função T permite prever exatamente o resultado de cada sequência de ações. Que vantagem isso traz para o planejamento? Qual é um limite prático dessa vantagem (por exemplo, quando o número de possibilidades a explorar é muito grande)?

Ambiente Estocástico

Um ambiente é **estocástico** quando uma mesma ação pode levar a diferentes estados sucessores, cada um ocorrendo com certa probabilidade. O resultado deixa de ser único e passa a ser descrito por uma distribuição de probabilidade:

$$\Pr(s' \mid s, a)$$

Um exemplo simples: um robô executa a ação "avançar", mas o chão é escorregadio. Com 80% de probabilidade, ele avança como esperado; com 20%, ele permanece no mesmo estado. O robô não tem como saber de antemão qual dos dois resultados ocorrerá.

Outros exemplos de ambientes estocásticos incluem o mercado financeiro, onde decisões de investimento têm resultados incertos, e jogos com dados, onde o resultado de cada lance é probabilístico.

Nesse tipo de ambiente, o agente não pode mais planejar com base em resultados certos. Em vez disso, ele precisa raciocinar sobre o *valor esperado* de suas ações, exatamente o conceito de utilidade esperada que será desenvolvido na próxima seção.

Pergunta 7

Em um ambiente estocástico, um agente pode simplificar e tratar apenas o resultado mais provável como se fosse certo. Segundo a definição de racionalidade deste curso (maximizar utilidade esperada), essa simplificação pode ser considerada racional? Em que tipos de situação ela pode ser aceitável e em quais pode ser perigosa?

3.2 Utilidade Esperada

Em ambientes determinísticos, o agente sabe exatamente qual estado resultará de cada ação, de modo que escolher a ação que maximiza $U(s')$ é suficiente. O problema aparece

em ambientes estocásticos: quando a ação a pode levar a múltiplos estados sucessores, cada um ocorrendo com certa probabilidade, não existe um único $U(s')$ a maximizar. O agente precisa ponderar os valores de todos os possíveis resultados por suas probabilidades de ocorrência.

Essa ponderação é exatamente o que a **utilidade esperada** captura. Se uma ação a tomada no estado s pode levar a diferentes estados sucessores s', cada um com probabilidade $\Pr(s' \mid s, a)$, então a utilidade esperada de a é:

$$EU(a) = \sum_{s'} \Pr(s' \mid s, a)\, U(s')$$

A fórmula tem uma interpretação direta: cada estado possível s' contribui com sua utilidade $U(s')$, ponderada pela probabilidade de que a realmente leve a s'. Um agente racional escolhe a ação que maximiza $EU(a)$: não a ação mais segura, nem a que tem maior chance de sucesso, mas a de maior valor esperado.

Recompensa e Utilidade

Para aplicar a fórmula acima, o agente precisa de uma maneira concreta de atribuir valores numéricos aos estados. Essa é a função da **recompensa**: $R(s)$ é uma medida direta do quão desejável é estar no estado s, expressa em unidades naturais do problema — pontos em um jogo, quilômetros poupados em uma rota, lucro em uma decisão financeira.

A **utilidade** $U(s)$ é uma função que o agente usa para *avaliar* os estados ao tomar decisões. No modelo mais simples, os dois conceitos coincidem:

$$U(s) = R(s)$$

Isso significa que o agente avalia cada estado exatamente pelo seu valor de recompensa, sem nenhuma transformação adicional. Um agente com essa função de utilidade é chamado de **neutro ao risco**: ele trata cada ponto de recompensa da mesma forma, independentemente de quanto já acumulou ou de quão incerta é a situação.

Essa coincidência entre U e R não é a única possibilidade. Em situações reais, agentes podem ser **avessos ao risco** (preferindo um ganho menor e certo a uma aposta com valor esperado maior) ou **propensos ao risco** (aceitando apostas mesmo quando o valor esperado é desfavorável). Essas atitudes surgem quando U é uma transformação não-linear de R: por exemplo, $U(s) = \sqrt{R(s)}$ penaliza ganhos altos incertos em favor de ganhos menores e garantidos. Neste curso, adotamos o modelo linear $U = R$ para manter o foco em busca e planejamento.

Exemplo

Para ilustrar, considere um agente que escolhe entre duas ações:

- a_1: ganho *certo* de 10 pontos.
- a_2: 50% de chance de ganhar 25 pontos, 50% de ganhar 0.

Adotando $U(s) = R(s)$ e aplicando a fórmula:

$$EU(a_1) = 1{,}0 \cdot 10 = 10 \qquad EU(a_2) = 0{,}5 \cdot 25 + 0{,}5 \cdot 0 = 12{,}5$$

Embora a_1 seja mais segura — nenhuma chance de ganhar zero —, a_2 tem utilidade esperada maior. Um agente racional neutro ao risco escolheria a_2.

Pergunta 8

No exemplo apresentado, temos duas ações: a_1 (ganho certo de 10 pontos) e a_2 (50% de chance de 25 pontos, 50% de chance de 0 pontos). (i) Calcule $EU(a_1)$ e $EU(a_2)$ usando a fórmula $EU(a) = \sum_{s'} \Pr(s' \mid s, a)\, U(s')$. (ii) Qual ação um agente racional neutro ao risco escolheria, e por quê? (iii) Se o ganho de a_2 fosse 15 pontos (em vez de 25), a resposta mudaria?

Probabilidade de sucesso vs. utilidade esperada

Um equívoco comum é confundir maximizar utilidade esperada com maximizar a probabilidade de obter algum resultado positivo. As duas coisas são diferentes. Considere:

- a_1: 90% de chance de ganhar 10 pontos.
- a_2: 40% de chance de ganhar 30 pontos.

Calculando:

$$EU(a_1) = 0{,}9 \cdot 10 = 9 \qquad EU(a_2) = 0{,}4 \cdot 30 = 12$$

A ação a_1 tem probabilidade de sucesso muito maior (90% contra 40%). Ainda assim, a_2 tem utilidade esperada maior. Um agente racional neutro ao risco escolheria a_2.

Racionalidade não significa "escolher o mais seguro". Significa escolher a ação de maior valor esperado segundo a função de utilidade adotada. Segurança e valor esperado são critérios distintos e, em geral, levam a escolhas diferentes.

Checkpoint

Os conceitos desta seção distinguem três critérios de decisão que frequentemente são confundidos: maximizar a recompensa imediata do estado atual, maximizar a probabilidade de obter algum resultado positivo, e maximizar a utilidade esperada. Os dois exemplos numéricos acima já ilustram a diferença entre o segundo e o terceiro critério. Antes de continuar, certifique-se de que consegue explicar com suas próprias palavras por que os três critérios são distintos e dar um exemplo simples em que cada um levaria a uma decisão diferente dos outros dois.

4 Tipos de Ambientes

Nem todo ambiente impõe os mesmos desafios a um agente racional. Duas características já foram discutidas em detalhe: determinismo e estocasticidade. Mas existem outras dimensões igualmente importantes que influenciam diretamente como o agente deve agir e que tipo de algoritmo é adequado para cada situação.

4.1 Observabilidade

Um ambiente é **totalmente observável** quando o agente tem acesso completo ao estado atual do mundo a cada instante; seus sensores capturam tudo que é relevante para a decisão. O quebra-cabeça de 8 peças é um exemplo: o agente sempre sabe exatamente onde cada peça está.

Um ambiente é **parcialmente observável** quando o agente enxerga apenas uma parte do estado do mundo. O jogo Pac-Man com campo de visão limitado é um exemplo: o agente não sabe onde os fantasmas estão além do seu raio de visão. Nesse caso, o agente precisa lidar com incerteza sobre o próprio estado em que se encontra, o que torna o problema substancialmente mais difícil.

Impacto no agente: em ambientes totalmente observáveis, o estado de busca pode ser definido com precisão. Em ambientes parcialmente observáveis, o agente pode precisar manter uma *estimativa* do estado atual, o que exige técnicas além da busca clássica.

Pergunta 9

Em um ambiente **totalmente observável**, o agente sabe exatamente em que estado se encontra a cada instante. Em um ambiente **parcialmente observável**, isso deixa de ser verdade. Dê um exemplo concreto: cite uma informação que o Pac-Man *não consegue observar* quando os fantasmas estão fora do seu campo de visão, e explique em uma frase por que isso dificulta a escolha da próxima ação.

4.2 Determinismo

Essa dimensão já foi discutida em detalhe na Seção 3. Recapitulando:

- **Determinístico**: dado o estado atual e a ação, o próximo estado é único e previsível.
- **Estocástico**: a mesma ação pode levar a diferentes estados, cada um com certa probabilidade.

O modelo de busca que formalizamos neste documento assume determinismo. Ambientes estocásticos exigem extensões do modelo, como o uso de distribuições de probabilidade no lugar da função de transição T.

4.3 Dinamismo

Um ambiente é **estático** quando ele não se altera enquanto o agente delibera. O labirinto do Pac-Man sem fantasmas é estático: nada muda enquanto o agente planeja seu próximo movimento.

Um ambiente é **dinâmico** quando ele pode mudar independentemente das ações do agente, seja por fatores externos, seja pela passagem do tempo. O trânsito urbano é dinâmico: mesmo que o carro autônomo não se mova, o estado das vias ao redor muda continuamente.

Impacto no agente: em ambientes estáticos, o agente pode planejar com calma antes de agir. Em ambientes dinâmicos, um plano calculado no instante t pode estar desatualizado no instante $t + 1$, exigindo replanejamento contínuo.

Pergunta 10

Qual é o risco de executar um plano fixo em um ambiente dinâmico (que pode mudar enquanto o agente delibera)? Dê um exemplo em que replanejar é essencial *mesmo conhecendo* T (isto é, mesmo sabendo como ações mudam estados), porque o ambiente muda por fatores externos.

4.4 Número de Agentes

Um ambiente é **monoagente** quando apenas um agente atua nele. É **multiagente** quando outros agentes também tomam decisões que afetam o ambiente.

A distinção importa porque agentes adicionais podem ser:

- **Cooperativos**: trabalham junto com o agente principal em direção a um objetivo comum (ex: robôs em uma linha de montagem).
- **Competitivos**: têm objetivos conflitantes com o agente principal (ex: adversário em um jogo de xadrez).

Em ambientes multiagente competitivos, o agente não pode tratar os outros como parte fixa do ambiente; ele precisa antecipar e reagir às decisões dos adversários, o que exige técnicas específicas como teoria dos jogos e busca adversarial.

Pergunta 11

Em um ambiente **monoagente**, o agente planeja sem se preocupar com as intenções de outros. Em um ambiente **multiagente competitivo**, isso muda. Considere um jogo de xadrez: por que o agente não pode simplesmente ignorar o adversário e executar o plano que seria ótimo se jogasse sozinho? Responda em uma ou duas frases.

4.5 Conhecimento da Física do Ambiente

Um ambiente tem **física conhecida** quando o agente sabe de antemão como suas ações transformam estados, ou seja, conhece o modelo de transição T. Nesse caso, o agente pode planejar sem precisar experimentar: basta simular internamente as consequências de cada ação.

Quando a física é **desconhecida**, o agente não tem esse modelo a priori. Ele precisa aprendê-lo por meio da interação com o ambiente, tentando ações e observando os resultados. Essa situação é o ponto de partida para técnicas de *aprendizado por reforço*, que serão abordadas em aulas futuras.

Impacto no agente: todos os algoritmos de busca que estudaremos nas próximas aulas assumem física conhecida: o agente sabe exatamente o que acontecerá ao executar cada ação. Isso é uma hipótese simplificadora importante, e vale tê-la em mente.

Pergunta 12

Até aqui, assumimos que o agente conhece T e consegue simular as consequências de suas ações. Agora imagine o oposto: o agente *não* conhece T e, portanto, não consegue prever com certeza qual estado resultará de cada ação. Explique por que, nesse caso, o agente precisa **interagir com o ambiente** (experimentar ações e observar resultados) antes de conseguir planejar de forma confiável. Dê um exemplo simples de que tipo de "regra" sobre o ambiente ele precisaria descobrir (isto é, uma parte de T).

4.6 Resumo

A tabela abaixo sintetiza as cinco dimensões discutidas.

Dimensão	**Variantes**	**Exemplo**
Observabilidade	Total / Parcial	Xadrez / Pac-Man com névoa
Determinismo	Determinístico / Estocástico	8-puzzle / Robô com falhas
Dinamismo	Estático / Dinâmico	Labirinto / Trânsito urbano
Número de agentes	Monoagente / Multiagente	Labirinto / Xadrez
Física	Conhecida / Desconhecida	Busca em grafo / Robô novo

Pergunta 13

Considere o Pac-Man *com fantasmas* e escolha **duas** das cinco dimensões da tabela (Observabilidade, Determinismo, Dinamismo, Número de agentes, Física). Para cada uma das duas dimensões escolhidas, indique em qual variante o Pac-Man se enquadra e justifique em uma frase.

5 Estado do Mundo vs. Estado de Busca

Antes de formalizar um problema de busca, é necessário responder a uma pergunta aparentemente simples: *o que deve estar representado em um estado?*

5.1 Estado do Mundo

O **estado do mundo** é uma descrição completa de tudo que existe no ambiente em um dado instante: posições de todos os objetos, valores de todos os sensores, histórico de ações passadas, condições físicas do agente, e assim por diante.

Em princípio, essa descrição completa seria ideal; ela nunca omite informação relevante. Na prática, porém, ela é inutilizável diretamente em busca, por duas razões.

Primeiro, o estado do mundo pode conter uma quantidade enorme de variáveis, a maioria delas irrelevante para o problema que o agente precisa resolver. Segundo, o tamanho do espaço de estados cresce explosivamente com o número de variáveis representadas: cada variável adicional multiplica o número de estados possíveis.

5.2 Estado de Busca

O **estado de busca** é uma abstração do estado do mundo. Ele retém apenas as informações necessárias para que o agente possa:

- Decidir quais ações são disponíveis;
- Determinar como as ações transformam o estado;
- Verificar se o objetivo foi atingido.

Tudo que não influencia essas três decisões pode, e deve, ser omitido. Essa omissão deliberada é chamada de **abstração**, e é ela que torna o problema computacionalmente tratável.

Observação

A abstração não é uma simplificação ingênua do problema. É uma escolha de modelagem: o projetista decide conscientemente o que incluir e o que descartar. Uma abstração mal feita pode tornar o problema irresolúvel ou produzir soluções incorretas.

Pergunta 14

Dê um exemplo de **abstração ruim** na formulação de um problema de busca: cite uma informação que, se for omitida do estado de busca, pode levar o agente a tomar decisões erradas (isto é, produzir um plano que parece válido no modelo, mas falha no problema real). Explique em uma frase por que essa informação é necessária.

5.3 Exemplo: dois problemas no Pac-Man

Considere duas versões de um problema em um labirinto do Pac-Man.

Problema 1: levar o Pac-Man da posição (x_1, y_1) até a posição (x_2, y_2).

Para resolver esse problema, o agente precisa saber apenas *onde está*. O estado de busca pode ser representado simplesmente por:

$$s = (x, y)$$

Informações como quais pílulas existem no labirinto são completamente irrelevantes para essa tarefa e não precisam constar no estado.

Problema 2: comer todas as pílulas no menor número de passos.

Agora, saber apenas a posição do Pac-Man não é suficiente. Para decidir o que fazer a seguir, o agente precisa saber também *quais pílulas ainda não foram comidas*. O estado de busca passa a ser:

$$s = (x, y, d_1, d_2, \ldots, d_k)$$

onde cada $d_i \in \{0, 1\}$ indica se a i-ésima pílula ainda está presente. Essa informação adicional é indispensável: dois estados com a mesma posição (x, y) mas configurações diferentes de pílulas exigem decisões distintas do agente.

O exemplo ilustra uma lição geral: **o estado de busca depende do problema**, não apenas do ambiente. O mesmo labirinto físico dá origem a dois espaços de estados

completamente diferentes conforme o objetivo muda. Definir o estado corretamente é, portanto, a primeira e mais importante decisão na formulação de um problema de busca.

Pergunta 15

No Problema 2, o estado inclui a posição do Pac-Man e o vetor de pílulas restantes. Cite **duas** informações do mundo que foram omitidas (por exemplo, o caminho percorrido até agora ou o tempo absoluto) e explique por que elas não são necessárias para encontrar uma solução ótima *neste modelo*.

6 Problemas de Busca

Um agente racional precisa escolher ações que maximizem sua utilidade esperada. Para isso, ele precisa responder a uma pergunta fundamental:

> "Dado o estado atual do mundo, quais ações devo executar para atingir meu objetivo da melhor maneira possível?"

Responder essa pergunta exige transformar o ambiente em um modelo matemático. Esse modelo é chamado de **problema de busca**.

6.1 Formulação

A formulação de um problema de busca corresponde a decidir *como modelar* o problema. Isso envolve escolher:

- Como definir um estado;
- Quais ações são possíveis em cada estado;
- Como as ações transformam estados;
- Como medir qualidade de soluções (c ou utilidade);
- Qual é exatamente a condição de sucesso. i.e., que estados correspondem à solução do problema.

Observe que essa etapa envolve *abstração*. Abstrair significa ignorar detalhes irrelevantes do mundo real para manter o espaço de estados tratável. O mundo real pode ser extremamente complexo, mas o estado de busca deve conter apenas o necessário para planejamento. Uma má formulação pode:

- Tornar o espaço de estados explosivo;
- Impedir a distinção entre estados relevantes;
- Produzir soluções incorretas.

Formalmente, um problema de busca pode ser descrito como a tupla:

$$\langle S, A, T, c, s_0, GoalTest \rangle$$

onde:

1. S é o **espaço de estados**: conjunto de configurações relevantes.

2. A é o conjunto de **ações** disponíveis ao agente. Em geral, as ações possíveis dependem do estado, então definimos:

$$\text{ACTIONS} : S \to 2^A$$

   onde ACTIONS(s) retorna o conjunto de ações permitidas em s.

3. $T : S \times A \to S$ é o **modelo de transição** (função sucessora). Dado um estado s e uma ação a, produz o estado sucessor:

$$s' = T(s, a)$$

   Note que T é determinística: o mesmo par (s, a) produz sempre o mesmo estado sucessor. Isso significa que, neste modelo, não há incerteza nas transições. Embora ambientes estocásticos existam e tenham sido discutidos anteriormente, o foco das próximas aulas será o caso determinístico, que já apresenta desafios computacionais significativos.

4. $c : S \times A \to \mathbb{R}_{\geq 0}$ é a **função custo** da ação:

$$c(s, a) \geq 0$$

   O custo representa o "preço" pago pelo agente ao executar uma ação. Ele está diretamente ligado ao critério de racionalidade: como veremos a seguir, minimizar custo total é, neste modelo, a forma operacional de maximizar utilidade.

5. $s_0 \in S$ é o **estado inicial**.

6. $GoalTest : S \to \{\texttt{True}, \texttt{False}\}$ é o **teste de objetivo**. Ele indica se um estado satisfaz o objetivo do agente.

Uma forma equivalente é definir o conjunto de estados objetivo $G \subseteq S$ e usar:

$$GoalTest(s) \equiv (s \in G)$$

Pergunta 16

Escolha um problema simples (ex.: achar uma sala no campus, resolver um labirinto, ordenar um vetor). Escreva: (i) um exemplo de estado inicial s_0, (ii) um exemplo de ação possível em ACTIONS(s), e (iii) como seria o teste de objetivo $GoalTest(s)$.

Pergunta 17

Em um problema real (por exemplo, roteamento urbano), cite uma informação do mundo real que você *não* colocaria no estado de busca por ser irrelevante para decidir ações ou testar o objetivo (por exemplo, a cor do carro). Explique por quê.

A escolha do estado influencia não apenas a correção da solução, mas também o desempenho do agente. Dois agentes que resolvem o "mesmo" problema podem ter desempenhos radicalmente diferentes dependendo de como formularam esse problema e de qual estratégia de busca adotaram.

Pergunta 18

Dois agentes podem resolver o "mesmo" problema com desempenhos muito diferentes. Explique por quê. Na sua opinião, a diferença pode vir da **formulação do problema** (como o estado foi definido), da **estratégia de busca** (qual nó da fronteira é expandido primeiro), ou de ambos? Justifique em 2–3 frases.

6.2 Planos e Utilidade

Resolver um problema de busca significa determinar um **plano** a partir de sua formulação. Um plano é uma sequência de ações:

$$\pi = (a_1, a_2, \ldots, a_k)$$

que, aplicada a partir de s_0, gera uma sequência de estados $s_0, s_1, \ldots, s_k$ tal que:

$$s_{i+1} = T(s_i, a_i) \quad \text{e} \quad a_i \in \text{ACTIONS}(s_i)$$

O plano é uma **solução** se o estado final satisfaz o teste de objetivo:

$$GoalTest(s_k) = \texttt{True}$$

O custo total do plano é a soma dos custos de cada ação executada:

$$C(\pi) = \sum_{i=1}^{k} c(s_{i-1}, a_i)$$

Pergunta 19

Considere o grafo do mini-desafio do Carro Autônomo (Seção 8.1) e o plano $\pi = (\text{A} \to \text{D}, \text{D} \to \text{E}, \text{E} \to \text{H})$. (i) Verifique que π é uma solução, isto é, que $GoalTest(s_k) = \texttt{True}$. (ii) Calcule $C(\pi) = \sum_i c(s_{i-1}, a_i)$ para esse plano. (iii) Existe algum plano com custo menor? Se sim, qual?

Aqui podemos estabelecer a ponte com o critério de *racionalidade* definido anteriormente. Em ambientes determinísticos com custo acumulado, definimos a utilidade de um plano como o negativo do seu custo:

$$U_\pi(\pi) = -C(\pi)$$

Com essa definição, maximizar utilidade equivale exatamente a minimizar custo, e o critério geral de racionalidade (maximizar utilidade esperada) se reduz, neste caso determinístico, a encontrar o plano de menor custo. Logo, um agente racional deve buscar um **plano ótimo**:

$$\pi^* = \arg \min_{\pi \,:\, GoalTest(s_k) = \texttt{True}} C(\pi)$$

Ou seja, resolver esse problema de busca é exatamente o mecanismo pelo qual o agente implementa sua racionalidade. Essa equivalência é o que justifica o uso de algoritmos de busca como mecanismo de decisão racional: resolver um problema de busca com custo mínimo *é* agir racionalmente dentro deste modelo.

6.3 Exemplo: Pac-Man (Eat-All-Dots)

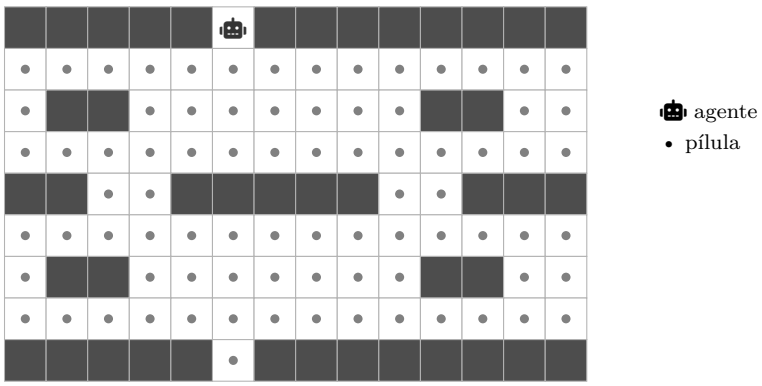

Figura 3: Representação estilizada do labirinto do Pac-Man no estilo de grade. O agente ( ) deve percorrer o labirinto coletando todas as pílulas pelo caminho de menor custo.

Considere uma versão simplificada do jogo Pac-Man:

- Não há fantasmas.
- O labirinto é fixo.
- O objetivo é comer todas as pílulas no menor número possível de passos.

Queremos formular esse problema como um problema de busca, isto é, definir explicitamente a tupla:

$$\langle S, \text{ACTIONS}, T, c, s_0, GoalTest \rangle$$

Espaço de Estados (S)

Um estado deve conter toda informação necessária para decidir os próximos passos.

Aqui, precisamos saber:

- A posição atual do Pac-Man.
- Quais pílulas ainda não foram comidas.

Uma representação adequada é:

$$s = (x, y, d_1, d_2, \ldots, d_k)$$

onde:

- (x, y) é a posição do agente no labirinto;
- $d_i \in \{0, 1\}$ indica se a i-ésima pílula ainda está presente.

Observe que o estado não armazena histórico do caminho percorrido. Apenas o necessário para decidir o futuro.

Ações (ACTIONS(s))

As ações possíveis dependem da geometria do labirinto.

$$\text{ACTIONS}(s) \subseteq \{\text{North, South, East, West}\}$$

Uma ação é válida apenas se não levar a uma parede.

Modelo de Transição (T)

A função de transição descreve como o estado evolui após uma ação.
Se $a \in \text{ACTIONS}(s)$, então:

$$s' = T(s, a)$$

onde:

- A nova posição (x', y') é obtida movendo na direção a;
- Se houver pílula na nova posição, o boolean correspondente é atualizado para 0.

Formalmente:

$$(x, y, d_1, \ldots, d_k) \xrightarrow{a} (x', y', d'_1, \ldots, d'_k)$$

Função Custo (c)

Cada movimento tem custo unitário:

$$c(s, a) = 1$$

Logo, o custo total de um plano é o número de passos executados.
Minimizar custo equivale a minimizar o tempo para comer todas as pílulas.

Estado Inicial (s_0)

$$s_0 = (x_0, y_0, 1, 1, \ldots, 1)$$

onde todas as pílulas ainda estão presentes.

Teste de Objetivo ($GoalTest$)

O objetivo é alcançado quando todas as pílulas foram comidas:

$$GoalTest(s) = \begin{cases} \texttt{True}, & \text{se } d_1 = \cdots = d_k = 0 \\ \texttt{False}, & \text{caso contrário} \end{cases}$$

Tamanho do Espaço de Estados

Suponha que:

- O labirinto possui n posições válidas;
- Existem k pílulas.

A posição do agente pode assumir n valores distintos.
Cada pílula pode estar em dois estados: presente (1) ou ausente (0). Logo, existem:

$$2^k$$

configurações possíveis das pílulas.
Pelo princípio multiplicativo[1], o tamanho do espaço de estados é:

$$|S| = n \cdot 2^k$$

Mesmo para valores moderados de k, esse número cresce rapidamente.

Pergunta 20

Se o Pac-Man pudesse estar em qualquer célula, inclusive em paredes, o número n de posições possíveis seria maior. O que isso revela sobre por que o espaço de estados S costuma incluir apenas **posições válidas/alcançáveis** no modelo?

Exemplo numérico.
Se:

- $n = 100$ posições;
- $k = 20$ pílulas;

então:

$$|S| = 100 \cdot 2^{20} \approx 1{,}05 \times 10^8$$

ou seja, aproximadamente 100 milhões de estados. Esse número representa apenas os estados possíveis. Ainda não consideramos as transições entre eles. Mesmo um problema aparentemente simples pode gerar um espaço de estados enorme.

6.4 Exemplo: Quebra-cabeça de 8 Peças

O quebra-cabeça de 8 peças é um dos problemas de busca mais estudados na literatura de IA, e por boas razões: ele é simples o suficiente para ser compreendido imediatamente, mas complexo o suficiente para exigir algoritmos inteligentes. O tabuleiro é uma grade 3×3 com oito peças numeradas e um espaço vazio. A única ação disponível é deslizar uma peça adjacente para o espaço vazio. O objetivo é atingir uma configuração ordenada a partir de uma configuração embaralhada.

[1] O princípio multiplicativo declara que, se uma variável admite n valores e outra admite m valores independentes, a combinação admite $n \cdot m$ configurações.

Esse problema ilustra algo importante: o espaço de estados não é definido pelo tamanho do tabuleiro, mas pelo número de configurações distintas que ele pode assumir. Uma grade 3×3 parece pequena, mas o número de arranjos possíveis das 9 posições (8 peças mais o espaço vazio) é $9! = 362.880$, o que já representa um espaço considerável para exploração.

No 8-puzzle, o **estado** é a configuração completa das peças na grade. Uma **ação** desliza uma peça para o espaço vazio, produzindo um novo estado. A Figura 4 mostra um exemplo: a partir do estado s, a ação "Mover 7 para cima" produz o estado $s' = T(s, \text{Mover 7 para cima})$.

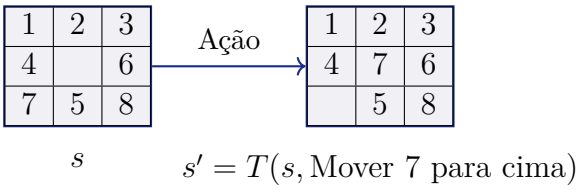

Figura 4: Exemplo de transição de estados no 8-puzzle. O espaço vazio está na posição central em s; após mover a peça 7 para cima, o espaço vazio ocupa a posição inferior esquerda em s'.

Pergunta 21

Quantos estados *diferentes* existem no 8-puzzle se considerarmos todas as permutações das 9 posições (8 números + 1 espaço vazio)? Considere que o espaço vazio conta como uma peça para fins de permutação. Mostre o raciocínio e confirme o valor $9! = 362.880$.

Com o estado e as ações definidos, podemos especificar a tupla correspondente à formulação do problema de busca para o 8-puzzle ($\langle S, \text{ACTIONS}, T, c, s_0, GoalTest \rangle$).

Espaço de Estados (S)

Para decidir qual ação tomar, o agente precisa saber exatamente como as peças estão dispostas no tabuleiro: a posição do espaço vazio determina quais movimentos são válidos, e a posição de cada peça determina o quanto o estado atual se afasta do objetivo. Por isso, o estado é definido como a *configuração completa* do tabuleiro, isto é, a atribuição de um valor (peça ou espaço vazio) a cada uma das 9 células da grade.

O espaço de estados S é, portanto, o conjunto de todas as configurações possíveis. Como há 9 posições a preencher com 9 elementos distintos, temos $|S| = 9! = 362.880$. Vale observar, porém, que nem todas essas configurações são alcançáveis a partir de um dado estado inicial: pode-se mostrar que exatamente metade das permutações são atingíveis por sequências válidas de movimentos.

Ações (ACTIONS(s))

As ações disponíveis em cada estado dependem de onde o espaço vazio se encontra. Se ele está no centro, há quatro movimentos possíveis (norte, sul, leste, oeste). Se está em uma borda, há três. Se está em um canto, apenas dois. Em todos os casos:

$$\text{ACTIONS}(s) \subseteq \{\text{Norte, Sul, Leste, Oeste}\}$$

onde cada direção indica qual peça adjacente será deslizada para o espaço vazio.

Modelo de Transição (T)

O modelo de transição é determinístico: dado o estado atual e uma ação válida, o estado resultante é único e completamente determinado pela geometria do movimento. Executar "Norte", por exemplo, significa trocar o espaço vazio com a peça imediatamente acima dele; o resultado não depende de nenhum fator externo.

Função custo

A função custo trata todos os movimentos igualmente:

$$c(s, a) = 1$$

Isso significa que o custo total de um plano é simplesmente o número de movimentos executados, e a solução ótima é aquela que resolve o *puzzle* com o menor número de passos.

Estado Inicial (s_0)

O estado inicial é qualquer configuração embaralhada do tabuleiro a partir da qual o agente deve encontrar o caminho até a solução. Por exemplo:

$$s_0 = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 4 & & 6 \\ \hline 7 & 5 & 8 \\ \hline \end{array}$$

Em geral, s_0 é fornecido como entrada do problema. A dificuldade da instância depende diretamente de quão distante s_0 está do estado objetivo, medido pelo número mínimo de movimentos necessários para resolvê-la.

Teste de Objetivo ($GoalTest$)

O estado objetivo s^* é a configuração ordenada:

$$s^* = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & \\ \hline \end{array}$$

O teste de objetivo verifica simplesmente se o estado atual coincide com s^*:

$$GoalTest(s) = \begin{cases} \text{Verdadeiro}, & \text{se } s = s^* \\ \text{Falso}, & \text{caso contrário} \end{cases}$$

A função custo define o que significa "ótimo"

A escolha de $c(s, a) = 1$ faz com que otimizar o plano signifique minimizar o número de movimentos. Mas essa não é a única escolha possível. Em outros domínios, o custo de uma ação pode representar distância percorrida, energia consumida ou tempo gasto. Note que a solução ótima muda conforme o critério adotado. Dois agentes que resolvem o mesmo puzzle com a mesma função de transição T, mas com funções de custo diferentes, podem produzir planos completamente distintos. Isso ilustra uma lição geral: a função custo não descreve o ambiente, ela descreve *o que o agente quer otimizar*.

Pergunta 22

Suponha que alguém proponha definir o custo de cada ação no 8-puzzle como o valor impresso na peça movida, isto é, mover a peça 7 custaria 7 e mover a peça 2 custaria 2. Essa seria uma escolha de modelagem apropriada? Explique por que sim ou por que não, considerando o que uma função custo deve representar.

7 Resolução de um Problema de Busca

Uma vez definida a formulação $\langle S, A, T, c, s_0, GoalTest \rangle$, resta responder à pergunta prática: como um algoritmo a resolve? Essa seção introduz as duas estruturas centrais que tornam isso possível, o *grafo de espaço de estados* e a *árvore de busca*, e descreve como elas são exploradas durante a execução de um algoritmo de busca.

7.1 Grafo de Espaço de Estados

O espaço de estados, junto com as transições definidas por T, induz naturalmente uma estrutura matemática chamada grafo:

$$G = (S, E)$$

onde:

- Os vértices são estados $s \in S$;
- Existe uma aresta dirigida (s, s') se existe uma ação a tal que $T(s, a) = s'$.

Essa estrutura é chamada de **grafo de espaço de estados**.
Note que:

- Cada estado aparece exatamente uma vez como *vértice* no grafo.
- O grafo é definido implicitamente por T.
- Na prática, ele é grande demais para ser armazenado integralmente.

Como o grafo não pode ser construído de uma vez, o agente precisará percorrê-lo de forma incremental: partindo do estado inicial, gerando apenas os estados que forem necessários à medida que a busca avança. Para isso, será necessário introduzir uma

estrutura auxiliar que organize quais estados já foram visitados e quais ainda precisam ser explorados. Essa estrutura será apresentada mais adiante nesta seção.

Pergunta 23

Como o grafo de espaço de estados é grande demais para ser armazenado explicitamente, o agente precisa explorá-lo incrementalmente. Que informações mínimas ele deve manter para: (i) evitar revisitar estados desnecessariamente e (ii) conseguir reconstruir o caminho da solução ao final?

7.2 Árvore de Busca (Search Tree)

Até agora, definimos o espaço de estados e observamos que ele induz um grafo de espaço de estados. Entretanto, esse grafo pode ser enorme. Não é viável construí-lo explicitamente. Para resolver o problema, o agente constrói uma estrutura diferente: a **árvore de busca**.

Definição

Uma *árvore de busca* é uma estrutura construída dinamicamente durante o processo de exploração. Ela possui:

- Raiz: corresponde ao estado inicial s_0;
- Nós filhos: correspondem à aplicação de ações;
- Cada caminho da raiz até um nó representa um plano parcial.

Nó de Busca

Um **nó de busca** não é apenas um estado. Ele contém informações adicionais. Formalmente, podemos representá-lo como:

$$n = \langle s, parent, action, g \rangle$$

onde:

- s é o estado;
- $parent$ é o nó pai;
- $action$ é a ação que levou ao estado;
- g é o custo acumulado desde a raiz.

Observe:

- Dois nós diferentes podem conter o mesmo estado s;
- Isso ocorre quando há múltiplos caminhos até s.

Grafo vs. Árvore

As duas estruturas representam coisas fundamentalmente diferentes, e confundi-las é um dos erros mais comuns ao estudar algoritmos de busca.

No grafo de espaço de estados, cada estado aparece exatamente uma vez como vértice. O grafo descreve a estrutura do problema: quais configurações existem e como as ações as conectam. Ele é uma propriedade do ambiente, não do algoritmo.

Na árvore de busca, o que está sendo representado não são estados, mas *caminhos*. Cada nó da árvore corresponde a uma sequência de ações que leva do estado inicial até algum estado, e se existem dois caminhos distintos que chegam ao mesmo estado s, então s aparecerá duas vezes na árvore, como dois nós distintos. A árvore é uma propriedade da execução do algoritmo, não do ambiente.

Essa assimetria tem uma consequência direta sobre o tamanho das duas estruturas. Como um mesmo estado pode ser alcançado por múltiplos caminhos, o número de nós na árvore de busca satisfaz:

$$|\text{nós da árvore}| \;\geq\; |S|$$

e frequentemente é muito maior do que $|S|$. Em ambientes com ciclos, a diferença pode ser ilimitada: o algoritmo pode gerar infinitos nós mesmo quando o espaço de estados é finito, simplesmente percorrendo o mesmo ciclo repetidas vezes. Isso explica por que controlar quais estados já foram visitados (por meio do conjunto *explored*) é essencial para garantir a terminação da busca.

Expansão

Expandir um nó folha da árvore de busca corresponde ao ato de gerar os sucessores desse nó e adicioná-los à árvore. Concretamente, isso envolve:

- Selecionar um nó folha;
- Aplicar todas as ações em ACTIONS(s);
- Gerar novos nós filhos.

A busca consiste em repetir esse processo até que:

$$GoalTest(s) = \texttt{True}$$

Interpretação Conceitual

A árvore de busca representa a capacidade do agente de:

> Simular possíveis futuros antes de agir.

Cada ramo é um cenário hipotético. Cada profundidade representa decisões sucessivas. A estratégia de busca determina *qual ramo explorar primeiro*.

Pergunta 24

A árvore de busca pode crescer indefinidamente mesmo quando S é finito. Em que situação isso ocorre? Explique relacionando (i) a existência de ciclos no grafo e (ii) a busca **em árvore** (sem conjunto de estados explorados).

7.3 Algoritmo de Busca

Uma vez definida a formulação

$$\langle S, A, T, c, s_0, GoalTest \rangle,$$

resta resolver o problema.

Resolver um problema de busca significa executar um **algoritmo** que explora o grafo de espaço de estados até encontrar um plano que satisfaça $GoalTest$.

Ideia Geral

O grafo de espaço de estados raramente pode ser construído por completo antes da busca começar. Para problemas de tamanho realista, ele é simplesmente grande demais: o 8-puzzle tem 362.880 estados, e problemas como o Pac-Man com pílulas chegam a centenas de milhões. A solução é explorar o grafo *incrementalmente*: o algoritmo parte do estado inicial e vai gerando novos estados apenas quando necessário, expandindo o espaço conhecido à medida que avança.

Para organizar essa exploração incremental, o algoritmo mantém três estruturas de dados ao longo de sua execução.

A primeira é a **árvore de busca**, que registra os caminhos percorridos até cada estado visitado. Cada nó da árvore carrega o estado em que o agente se encontra, o nó pai, a ação que gerou esse nó e o custo acumulado desde a raiz. A árvore cresce à medida que novos estados são gerados, mas nunca é construída por inteiro de antemão; ela emerge progressivamente durante a execução.

A segunda é a **fronteira** (também chamada de *open list*), que contém os nós gerados mas ainda não expandidos. Esses são os candidatos imediatos para a próxima expansão: o agente já sabe que eles existem, mas ainda não examinou o que é possível fazer a partir deles. A fronteira é a região de contato entre o espaço já explorado e o espaço ainda desconhecido, e a escolha de *qual* nó da fronteira expandir a seguir é exatamente o que distingue um algoritmo de busca de outro.

A terceira estrutura é o **conjunto de estados explorados** (também chamado de *closed list*), que registra os estados cujos nós já foram expandidos. Sua função é evitar que o algoritmo expanda o mesmo estado mais de uma vez, o que seria redundante e, na presença de ciclos, poderia fazer a busca não terminar. Essa estrutura é opcional: algoritmos que não a mantêm são chamados de busca em árvore, e algoritmos que a mantêm são chamados de busca em grafo. A diferença entre os dois será discutida mais adiante nesta seção.

Nós vs. Estados: Exemplo Visual

Um erro comum é confundir o **estado** (onde o agente está) com o **nó** (o rastro do caminho percorrido até ali). A Figura 5 ilustra uma das distinções mais críticas para a compreensão da busca em IA: a diferença conceitual entre **estado** e **nó**. Enquanto o grafo de espaço de estados (parte superior) representa a conectividade "física"do ambiente, onde cada estado é único, a árvore de busca (parte inferior) mapeia as trajetórias hipotéticas exploradas pelo agente. Observe que o estado C aparece duplicado na árvore através dos nós 3 e 4; isso ocorre porque existem dois caminhos distintos para alcançá-lo. Para o algoritmo, embora o estado final seja o mesmo, esses nós são objetos distintos na memória, pois carregam históricos, ancestrais e custos acumulados diferentes. Compreender essa redundância é o

que permite ao aluno entender a necessidade de estruturas como o conjunto *explored* para evitar o crescimento exponencial e redundante da árvore de busca.

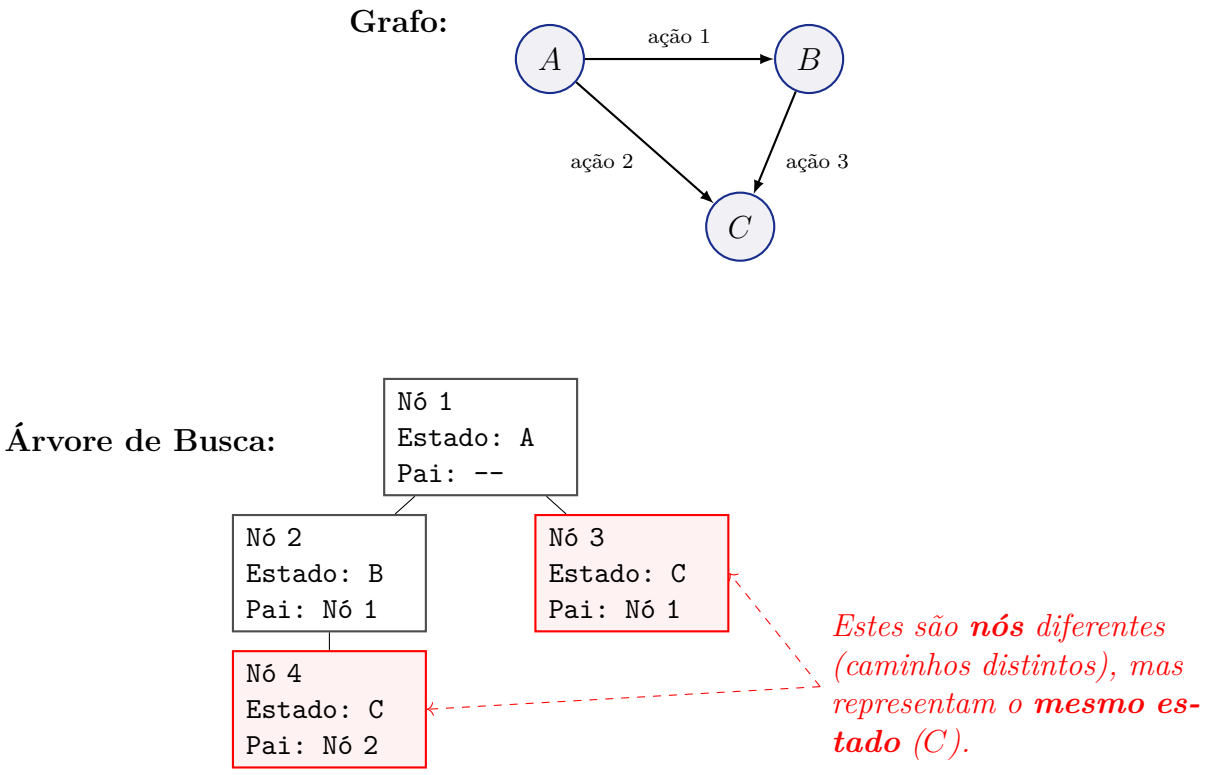

Figura 5: Diferença entre a estrutura do mundo (Grafo) e a estrutura da busca (Árvore).

Conclusão Importante

Um **Estado** não tem pai nem custo acumulado; ele é apenas uma configuração do mundo. Um **Nó** é uma estrutura de dados que guarda o estado, quem é seu antecessor e qual o custo total para chegar até ali.

Pergunta 25

Um **estado** e um **nó de busca** são objetos distintos. (i) Liste os campos que um nó $n = \langle s, parent, action, g \rangle$ carrega e que o estado s *não* carrega. (ii) No grafo da Figura 5, os nós 3 e 4 contêm o mesmo estado C, mas são nós distintos. O que os diferencia concretamente, isto é, quais campos de n têm valores diferentes entre eles?

Fronteira e Nós Explorados

A Figura 6 ilustra o estado intermediário da árvore de busca durante a execução de um algoritmo.

Podemos distinguir três conjuntos de nós:

- **Nós explorados** (pretos): já foram removidos da fronteira e expandidos. Seus sucessores já foram gerados.

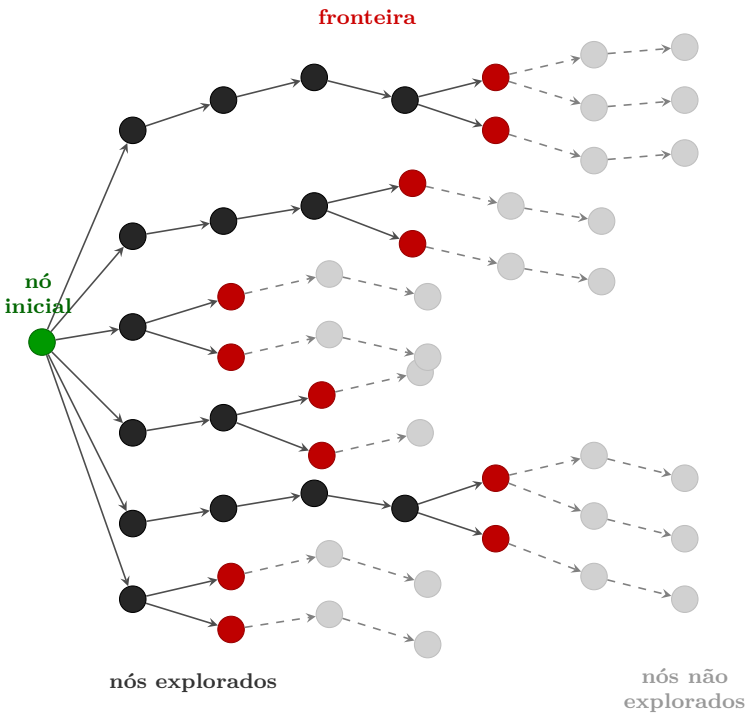

Figura 6: Estado intermediário da árvore de busca. O nó inicial (verde) é a raiz. Nós pretos já foram expandidos. Nós vermelhos compõem a fronteira em profundidades variadas e correspondem aos nós folhas da árvore atual; note que alguns ramos atingem a fronteira mais cedo que outros, o que reflete a estratégia de busca em uso. Nós cinzas representam estados do espaço de estados cujos nós correspondentes ainda não foram gerados na árvore. Arestas tracejadas indicam expansões ainda não realizadas.

- **Fronteira** (vermelho): contém os nós gerados, mas ainda não expandidos. São as extremidades dos caminhos atualmente sob consideração.
- **Nós não explorados** (cinza): ainda não foram gerados na árvore de busca. Os estados correspondentes existem no espaço de estados, mas ainda não possuem nós associados na árvore.

A fronteira define o limite entre:

- A parte já examinada do espaço;
- A parte ainda desconhecida.

Cada iteração do algoritmo consiste em:

1. Selecionar um nó da fronteira;
2. Removê-lo da fronteira;
3. Expandir seus sucessores;
4. Inserir novos nós na fronteira.

A estratégia de busca determina *qual nó da fronteira será selecionado*. Essa decisão influencia diretamente:

- Tempo de execução;
- Uso de memória;
- Qual solução será encontrada primeiro;
- Se a solução será ótima.

Observação

A árvore de busca cresce apenas na região da fronteira. Estados que ainda não foram gerados não ocupam memória.

Estrutura Geral

A estrutura geral é:

1. Inicializar a fronteira com um nó contendo s_0.
2. Enquanto a fronteira não estiver vazia:
   (a) Selecionar um nó n da fronteira;
   (b) Se $GoalTest(n.state)$ for verdadeiro, retornar o plano associado a n;
   (c) Caso contrário, expandir n:
   - Gerar todos os sucessores via T;
   - Inserir os novos nós na fronteira.
3. Se a fronteira esvaziar, não há solução.

A única diferença essencial entre algoritmos de busca clássicos está na regra usada para selecionar o próximo nó da fronteira.

Pseudocódigo Formal

O Algoritmo 1 implementa um esquema genérico de busca em grafo parametrizado por um *problem*, isto é, por uma instância da tupla $\langle S, \text{ACTIONS}, T, c, s_0, GoalTest\rangle$. Inicialmente, a fronteira é criada vazia e recebe um nó contendo o estado inicial $problem.s_0$. A cada iteração, o algoritmo remove da fronteira um nó n segundo a estratégia adotada (fila, pilha, fila de prioridade etc.). Se o estado associado a esse nó satisfaz $problem.GoalTest$, o caminho é reconstruído por meio dos ponteiros *parent* e retornado como solução. Caso contrário, o estado é marcado como explorado e todos os seus sucessores são gerados aplicando $problem.ACTIONS$ e $problem.T$. Cada sucessor dá origem a um novo nó, cujo custo acumulado g é atualizado com base na função de custo. Para evitar expansão redundante, o algoritmo só insere na fronteira estados que ainda não foram explorados nem já estão aguardando expansão. O processo continua até que uma solução seja encontrada ou a fronteira se esvazie, caso em que o algoritmo conclui que não há solução.

Pergunta 26

No Algoritmo 1, cada nó n' criado armazena apenas o campo *parent* (ponteiro para o nó pai) e o campo *action* (ação que gerou n'). Como, a partir dessas informações, é possível reconstruir o caminho completo desde s_0 até n' ao final da busca? Descreva o procedimento em duas ou três frases.

Algorithm 1 Generic-Search(problem)

1: $frontier \leftarrow$ estrutura vazia
2: inserir nó contendo $problem.s_0$ em $frontier$
3: $explored \leftarrow \emptyset$
4: **while** $frontier \neq \emptyset$ **do**
5: $n \leftarrow$ remover nó da $frontier$ segundo a estratégia
6: **if** $problem.GoalTest(n.state)$ **then**
7: **return** caminho associado a n
8: **end if**
9: adicionar $n.state$ a $explored$
10: **for all** $a \in problem.ACTIONS(n.state)$ **do**
11: $s' \leftarrow problem.T(n.state, a)$
12: **if** $s' \notin explored$ e não existe nó em $frontier$ com estado s' **then**
13: criar novo nó n' com:

$$n'.state = s', \quad n'.parent = n, \quad n'.g = n.g + problem.c(n.state, a)$$

14: inserir n' em $frontier$
15: **end if**
16: **end for**
17: **end while**
18: **return** falha

Importante: Busca em Árvore vs. Busca em Grafo

Embora utilizemos uma **árvore de busca** para representar os caminhos, o algoritmo apresentado no Algoritmo 1 é tecnicamente uma **Busca em Grafo**.
A diferença reside no tratamento de estados repetidos:

- **Busca em Árvore:** Não mantém o conjunto *explored*. Se houver ciclos no grafo de estados (ex: o Pac-Man pode ir e voltar entre duas células), a árvore de busca pode crescer infinitamente, mesmo que o espaço de estados seja pequeno.
- **Busca em Grafo:** Utiliza o conjunto *explored* (lista fechada) para garantir que cada estado seja expandido **no máximo uma vez**.

Por que isso importa? Na busca em grafo, garantimos a terminação em espaços de estados finitos. No entanto, o custo de memória aumenta, pois precisamos armazenar todos os estados já visitados para evitar a redundância.

Visualizando a Exploração: Grafo vs. Árvore

Na Figura 7, comparamos como um ambiente simples com um ciclo (o agente pode voltar para o estado anterior) é representado no espaço de estados e como ele se expande em uma árvore de busca.

- **No Grafo**: Cada estado é único. O ciclo $s_0 \leftrightarrow s_1$ é apenas um par de arestas entre dois vértices.

Figura 7: Comparação entre grafo de estados (esq.) e árvore de busca (dir.) em um ambiente com ciclo. Na esquerda, temos um grafo de espaço de estados com ciclo entre s_0 e s_1. Na direita, temos uma árvore de busca; note que o estado s_0 reaparece em um novo nó.

- **Na Árvore**: Cada nó representa um *caminho* (ou plano parcial). Se o algoritmo não utilizar o conjunto `explored` (Busca em Árvore), ele poderá oscilar infinitamente entre os nós que contêm os estados s_0 e s_1.

Fronteira como Estrutura de Controle

A fronteira não é apenas um detalhe de implementação: ela é o mecanismo que determina *qual* nó será expandido a seguir e, portanto, a ordem em que o espaço de estados é explorado. Algoritmos de busca que parecem conceitualmente distintos são, na prática, instâncias do mesmo esquema genérico diferindo apenas na estrutura de dados usada para implementar a fronteira.

Se a fronteira for implementada como uma **fila** (*FIFO: first in, first out*), o nó mais antigo é sempre selecionado primeiro. Isso significa que todos os nós a uma certa profundidade são expandidos antes de qualquer nó mais profundo, resultando na **busca em largura**. Essa estratégia garante encontrar a solução de menor profundidade, mas pode consumir muita memória, pois mantém todos os nós de um nível antes de avançar ao próximo.

Se a fronteira for implementada como uma **pilha** (*LIFO: last in, first out*), o nó gerado mais recentemente é expandido primeiro. O algoritmo mergulha o mais fundo possível em cada ramo antes de retroceder, resultando na **busca em profundidade**. Essa estratégia usa pouca memória (apenas o caminho atual precisa estar na memória de uma vez), mas não garante encontrar a solução de menor custo, nem mesmo terminar, se houver caminhos de profundidade infinita.

Se a fronteira for implementada como uma **fila de prioridade**, o nó selecionado a cada iteração é aquele com menor valor segundo algum critério numérico. Quando esse critério é o custo acumulado g desde a raiz, o algoritmo sempre expande o nó de menor custo conhecido, resultando na **busca de custo uniforme**. Essa estratégia garante encontrar a solução de menor custo total, desde que os custos das ações sejam não-negativos. Outros critérios de prioridade (como combinar g com uma estimativa do custo restante até o objetivo) darão origem a algoritmos mais sofisticados, que serão estudados em aulas futuras.

A tabela abaixo resume a correspondência entre estrutura de dados e comportamento resultante.

Estrutura da fronteira	Algoritmo	Garante solução ótima?
Fila (*FIFO*)	Busca em largura	Sim (custo uniforme por passo)
Pilha (*LIFO*)	Busca em profundidade	Não
Fila de prioridade (por g)	Busca de custo uniforme	Sim

Pergunta 27

A tabela ao final desta subseção relaciona três estruturas de dados a três algoritmos de busca. Para cada estrutura (fila, pilha, fila de prioridade), explique em uma frase **por que** ela produz o comportamento de busca correspondente, isto é, qual propriedade da estrutura determina a ordem em que os nós são expandidos.

Correção e Otimalidade

Nem todo algoritmo de busca é igualmente útil. Antes de escolher qual algoritmo aplicar a um problema, é necessário entender o que ele garante, e o que não garante. Duas propriedades capturam as garantias mais importantes.

A primeira é a **completude**. Um algoritmo é completo quando garante encontrar uma solução sempre que ela existir. Isso pode parecer uma exigência óbvia, mas nem sempre é satisfeita: um algoritmo que explora o espaço de estados de forma inadequada pode entrar em ciclos, mergulhar indefinidamente em ramos sem saída, ou simplesmente nunca alcançar a região do espaço onde a solução se encontra. Um algoritmo incompleto pode falhar mesmo quando a solução existe e está acessível a partir do estado inicial.

A segunda é a **otimalidade**. Um algoritmo é ótimo quando, se existir solução, garante encontrá-la e a solução retornada possui custo mínimo dentre todas as soluções possíveis. Observe que otimalidade é uma propriedade mais forte do que completude: todo algoritmo ótimo é necessariamente completo, pois não pode garantir que retornou a melhor solução se não garante sequer encontrar uma. A implicação inversa, porém, não vale: um algoritmo pode ser completo sem ser ótimo, encontrando sempre alguma solução, mas sem garantia de que é a de menor custo. A busca em profundidade é um exemplo clássico dessa situação.

As duas propriedades podem, portanto, ser combinadas de três formas relevantes na prática: o algoritmo pode ser completo e ótimo (a situação ideal), completo mas não ótimo (encontra sempre uma solução, mas não necessariamente a melhor), ou nem completo nem ótimo (pode falhar mesmo havendo solução). Um algoritmo ótimo mas não completo é, estritamente, uma contradição.

Se essas propriedades são satisfeitas depende de três fatores combinados. O **tipo de ambiente** importa porque ciclos e espaços infinitos podem impedir a terminação de algoritmos sem controle de estados visitados. A **formulação do problema** importa porque uma abstração que colapsa estados distintos pode tornar impossível distinguir caminhos de custo diferente. E a **estratégia de seleção da fronteira** importa porque a ordem em que os nós são expandidos determina diretamente se a primeira solução encontrada será a de menor custo. Nenhum desses fatores isolado determina as propriedades do algoritmo: os três interagem.

Pergunta 28

O texto afirma que "um algoritmo ótimo mas não completo é, estritamente, uma contradição". Explique com suas palavras por que essa afirmação é verdadeira, isto é, por que otimalidade implica completude. Em seguida, explique por que a implicação inversa não vale: por que um algoritmo pode ser completo sem ser ótimo?

Conexão com Racionalidade

Ao longo deste documento, definimos racionalidade como a escolha da ação que maximiza a utilidade esperada do agente. Nos ambientes determinísticos com custo acumulado que estamos estudando, essa definição geral se traduz em algo concreto e computacionalmente operacional: encontrar o plano de menor custo total.

A ponte entre os dois é direta. Definimos a utilidade de um plano π como o negativo do seu custo:

$$U_\pi(\pi) = -C(\pi)$$

Com esse sinal, maximizar utilidade é matematicamente idêntico a minimizar custo. O agente racional (i.e., aquele que maximiza U_π) é exatamente o agente que encontra o plano mais barato. Não é uma coincidência de notação: é a formalização de que, neste modelo, agir bem significa desperdiçar o mínimo possível de recursos para atingir o objetivo.

Isso revela o papel de cada peça do modelo. A função de utilidade especifica *o que* o agente quer: atingir o objetivo ao menor custo possível. O algoritmo de busca especifica *como* o agente tenta satisfazer esse critério: explorando sistematicamente o espaço de estados em busca do plano ótimo. Racionalidade e busca não são conceitos separados neste modelo; a busca ótima *é* a implementação operacional da racionalidade.

Uma consequência prática imediata: nem todo algoritmo de busca implementa racionalidade. Um algoritmo que encontra *uma* solução qualquer, sem garantia de que seja a de menor custo, não realiza plenamente o critério racional. Para que o algoritmo seja racional no sentido definido aqui, ele precisa ser *ótimo*: a primeira solução que ele retorna deve ter custo mínimo entre todas as soluções possíveis. Como vimos na subseção anterior, isso depende diretamente da estratégia de seleção da fronteira.

Pergunta 29

Suponha que não há ciclos e que o espaço de estados é finito. Que regra de seleção da fronteira garante que a **primeira** solução encontrada terá custo total mínimo? Explique a ideia de expandir nós em **ordem crescente de custo acumulado** g e por que isso é suficiente para garantir otimalidade nesse cenário.

8 Atividades intra-classe

8.1 Carro Autônomo

O grafo abaixo representa uma malha rodoviária simplificada com oito cidades. As arestas são bidirecionais e os valores indicam a distância em quilômetros entre cidades vizinhas.

Considere o problema: *partir da cidade A e chegar à cidade H pelo caminho de menor custo.*

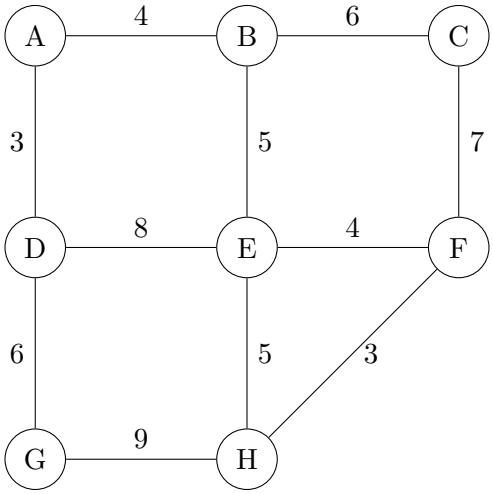

Figura 8: Malha rodoviária fictícia utilizada no mini-desafio.

Formule esse problema de busca definindo explicitamente cada componente da tupla $\langle S, \text{ACTIONS}, T, c, s_0, GoalTest \rangle$. Para cada componente, justifique brevemente sua escolha; em particular, o que você decidiu incluir e o que decidiu omitir do estado de busca, e por quê.

Por fim, responda: quantas rotas distintas existem entre A e H nesse grafo? Todas elas são soluções válidas? O que distingue a solução ótima das demais?

8.2 Labirinto

Considere o seguinte labirinto representado como uma grade 3×3:

S	·	·
■	■	·
·	·	G

O agente começa na célula marcada com S (linha 1, coluna 1) e deve chegar à célula marcada com G (linha 3, coluna 3). Células marcadas com ■ são paredes intransponíveis. O agente pode se mover para células adjacentes (norte, sul, leste, oeste), desde que não haja parede. Cada movimento tem custo 1.

Formule esse problema de busca definindo explicitamente cada componente da tupla $\langle S, \text{ACTIONS}, T, c, s_0, GoalTest \rangle$. Para cada componente, justifique brevemente sua escolha. Em seguida, calcule o tamanho do espaço de estados e identifique quantas soluções ótimas existem para esse problema.

8.3 Perguntas extras (para discussão em sala de aula)

Pergunta 30

Em um ambiente parcialmente observável, por que a posição (x, y) pode não ser um estado de busca suficiente? Dê um exemplo do tipo de informação adicional que o agente precisaria manter (mesmo que aproximada), como uma **crença** sobre onde estão obstáculos ou adversários que ele não consegue observar diretamente.

Pergunta 31

Por que um oponente "inteligente" (que escolhe ações para te atrapalhar) não é bem representado apenas como aleatoriedade em uma distribuição $\Pr(s' \mid s, a)$? Explique, em uma frase, qual dificuldade extra aparece quando o "ambiente" inclui agentes com objetivos próprios.

Pergunta 32

Classifique o jogo Pac-Man *com fantasmas* segundo cada uma das cinco dimensões da tabela. Para cada dimensão, justifique sua escolha em **uma frase**. Em seguida, diga qual dimensão representa o maior desafio para um agente planejador nesse ambiente e explique por quê.

Pergunta 33

Ao criar um nó n', o algoritmo armazena apenas um ponteiro para o nó pai (*parent*) e a ação realizada, em vez de armazenar uma lista com o caminho completo desde s_0 até n'. Por que isso economiza memória? Explique mencionando que muitos nós compartilham **prefixos iguais** de caminho.