LAS VEGAS

ALGORITMOS PROBABILÍSTICOS

PROBLEMA DAS 9 DAMAS: ALGORITMO LAS VEGAS

- O Problema das N-Rainhas (N=9)
  - Restrição: Posicionar 9 rainhas em um tabuleiro 9x9 de modo que nenhuma rainha ataque a outra (na mesma linha, coluna ou diagonal).
  - **Natureza do Problema:** É um problema de **satisfação de restrições** (NP-Completo).
- Por que Las Vegas?
  - A prioridade é encontrar qualquer solução válida (Resposta Determinística).
  - Não nos importamos com o tempo que isso levará (Tempo Probabilístico).

O PASSO A PASSO DO LAS VEGAS

Bloco	Propósito	Função no Código
Bloco I: Tentativa Aleatória	Gerar uma configuração do tabuleiro com decisões aleatórias em cada coluna.	while True (Loop externo de repetição)
Bloco II: Verificação	**Checar a validade** de cada nova rainha colocada.	is_safe(board, row, col) (Função de checagem)

Se a verificação falhar (failed = True), o algoritmo **ABORTA** a tentativa atual e **COMEÇA DO ZERO** (volta ao while True).

DEFINIÇÃO FORMAL DO EQUILÍBRIO DE NASH

- Uma combinação de estratégias constitui um equilíbrio de Nash quando :
- Cada estratégia é a **melhor resposta possível** às estratégias dos demais jogadores.
- Essa condição é verdadeira para **todos os jogadores.**
- Jogo de prevenção de entrada no mercado nacional, que não pode ser resolvido pela EIEED.
- **Estratégias: Empresa dominante:** Investe ou não em expansão.
- **Entrante potencial:** Não exporta, ou exporta em pequena escala, ou exporta em larga escala.

DETALHE DA FUNÇÃO DE VERIFICAÇÃO
A FUNÇÃO IS_SAFE(BOARD, ROW, COL)

Esta função é crítica. Ela garante que, se o algoritmo retornar, a solução é **determinística (correta)**.

Checagem	Lógica (Restrições)	Explicação do Código
Ataque na Coluna	Não é necessária. O algoritmo coloca **uma rainha por coluna**.	Estrutura da board garante que não há duas rainhas na mesma coluna.
Ataque na Linha	Verifica se a row escolhida já foi ocupada por uma rainha em uma prev_col anterior.	if board[prev_col] == row:
Ataque na Diagonal	O ataque ocorre se a **distância entre as linhas** for igual à **distância entre as colunas**.	if abs(board[prev_col] - row) == abs(prev_col - col):

O MECANISMO ALEATÓRIO (O LOOP PRINCIPAL) O CORAÇÃO PROBABILÍSTICO

A aleatoriedade é usada para guiar a busca, mas a **validade é verificada a cada passo**.

Etapa	Explicação	Linhas do Código
1. Início da Tentativa	Zera o tabuleiro (zera a lista board) e entra no loop while True.	board = [-1] * n
2. Aleatoriedade	Para cada coluna (col), o algoritmo gera uma lista de linhas (available_rows) e a **embaralha** (random.shuffle).	random.shuffle(available_rows)

O MECANISMO ALEATÓRIO (O LOOP PRINCIPAL) O CORAÇÃO PROBABILÍSTICO

Etapa	Explicação	Linhas do Código
3. Posicionamento	Tenta colocar a rainha na primeira row aleatória que satisfaz is_safe().	if is_safe(...)
4. Falha	Se o algoritmo verificar todas as linhas na coluna e **NENHUMA** for segura, a tentativa falhou.	if not found_safe_position:
5. Recomeço (Las Vegas)	Se falhar, a rainha na coluna col não pode ser colocada, então o algoritmo **ABORTA** a tentativa atual (break) e volta ao while True para iniciar uma **nova tentativa aleatória**.	while True (Loop externo)

CARACTERÍSTICAS DA EXECUÇÃO TEMPO VS. RESPOSTA NA PRÁTICA

Característica	Las Vegas (9 Damas)	Implicação para o Engenheiro
Resposta	Determinística (Sempre Correta)	O vetor de saída é uma **solução 100% válida**. A verificação foi feita exaustivamente.
Tempo	Probabilístico (Incerteza)	O tempo de execução varia muito a cada vez que o programa é rodado, dependendo da **sorte** da escolha das posições aleatórias.
Estratégia	Encontrar *qualquer* solução válida é suficiente.	Não busca a solução ótima (mais rápida), mas sim a **primeira solução correta** que for encontrada.

AGORA VAMOS VER OUTRA APLICAÇÃO DE LAS VEGAS

- Antes de buscarmos caminhos, vamos lembrar o que um algoritmo Las Vegas nos promete (baseado na aula das N-Rainhas):
- Resposta (Saída): Determinística (Sempre Correta).
- Tempo (Execução): Probabilístico (Incerteza).
- O algoritmo nunca retorna uma resposta errada.
- Ele só retorna quando verifica que a solução é válida (ex: "Nenhuma rainha ataca outra"). Porém, não sabemos quando ele vai encontrar essa solução.

*O MÉTODO CLÁSSICO: O CAMINHO "ÓTIMO" (DIJKSTRA / A)**

- Em Engenharia de Computação, o método padrão para encontrar o "menor custo" (seja distância, combustível ou tempo) é um algoritmo determinístico como o Dijkstra.
- **Objetivo:** Encontrar o **caminho ótimo** (o melhor caminho possível).
- **Processo:** Sistemático. Explora todos os vizinhos de um nó, atualiza os custos e, metodicamente, constrói a melhor rota.
- **Garantia:** Ele **garante** que o caminho encontrado é o de *menor custo absoluto*.
- **Então, qual é o problema?**

O "PROBLEMA" DO CAMINHO ÓTIMO EM REDES DINÂMICAS

- O método de Dijkstra é perfeito para redes *estáticas* (como um mapa de ruas de uma cidade).
- Mas em redes *dinâmicas*, ele falha.
  - **Redes Dinâmicas:** Grafos que mudam constantemente.
  - **Exemplos:** Roteamento de pacotes na Internet (links caem), malha aérea (aeroportos fecham por clima), trânsito de Waze (ruas congestionam).

O "PROBLEMA" DO CAMINHO ÓTIMO EM REDES DINÂMICAS

- **Os Problemas:**
  - **Custo Computacional:** Em grafos gigantes (a Internet, a malha aérea global), rodar Dijkstra é **extremamente lento**.
  - **Obsolescência da Resposta:** O pior problema. O cálculo de Dijkstra pode demorar tanto que, no momento em que ele termina, o grafo já mudou!
- **Exemplo:** Dijkstra calcula a rota aérea ótima, mas 10 minutos depois, o aeroporto de conexão (o "nó" principal) fecha por neblina. O "caminho ótimo" calculado tornou-se **inválido**.

A SOLUÇÃO LAS VEGAS: O CAMINHO "VÁLIDO" E "RÁPIDO"

- Em redes dinâmicas, o objetivo muda:
  - **NÃO é:** "Qual o melhor caminho possível?"
  - **É:** "Qualquer rota funcional que eu possa usar *agora*?"
- Esta é uma tarefa de **Satisfação de Restrições**, perfeita para Las Vegas.
- **A Abordagem Las Vegas (Roteamento Rápido):**
  - **Objetivo:** Encontrar um caminho de A para B que seja *funcional*.
  - **Aleatoriedade:** O algoritmo não explora *todos* os vizinhos. Ele começa a seguir um caminho aleatório (ou semi-aleatório, usando uma heurística simples).
  - **Verificação:** A cada passo, ele verifica: "Este link está ativo? O custo até agora é aceitável?"
- **Retorno:** No momento em que encontra o *primeiro* caminho que chega ao destino e passa nas verificações, ele retorna.

COMPARATIVO: DIJKSTRA VS. LAS VEGAS NO ROTEAMENTO"

Característica	Dijkstra / A* (Determinístico)	Las Vegas (Probabilístico)
Objetivo	Caminho **ÓTIMO**	Caminho **VÁLIDO / SATISFATÓRIO**
Velocidade	Lento (Calcula todas as possibilidades)	Rápido (na média) (Encontra a primeira solução)
Garantia	Garante a **melhor** rota (se o grafo não mudar)	Garante uma rota **correta** (mas não a melhor)
Ideal para	Redes Estáticas (Mapas de GPS)	Redes Dinâmicas (Roteamento na Internet, Voo)

EXERCÍCIO PROPOSTO: ROTA LAS VEGAS (QUATRO BARRAS ✈ BOCA RATON)

- **Cenário:** Uma aeronave de pequeno porte (monomotor) precisa ir do Aeródromo de Quatro Barras (PR, Brasil) para Boca Raton (FL, EUA). Esta aeronave tem **alcance limitado** e precisa de múltiplas paradas para reabastecer.
- **O Problema (Dijkstra):** Calcular o caminho de *menor custo de combustível possível* (o caminho ótimo) exigiria analisar todas as conexões de aeroportos entre Brasil e EUA, um cálculo lento e complexo.
- **Seu Desafio (Las Vegas):** Em vez do "ótimo", queremos um caminho "bom o suficiente" e rápido, usando o Las Vegas.

EXERCÍCIO: AS REGRAS DO VOO (A VERIFICAÇÃO)

- Seu algoritmo Las Vegas deve encontrar um caminho do PONTO_A ao PONTO_B que satisfaça as seguintes restrições (a função de verificação):
- **Restrição de Alcance (Arestas):** O avião não pode voar mais de **2.000 km** entre duas paradas (o custo de cada *aresta* não pode ser > 2000).
- **Restrição de Eficiência (Caminho):** A rota inteira não pode ter mais que **7 paradas** (escalas).
- **Restrição de Custo (Caminho):** O custo total de combustível (soma dos custos) deve ser **menor que 15.000 unidades**.

EXERCÍCIO: A TAREFA DE IMPLEMENTAÇÃO

- Implementar buscar_rota_las_vegas(origem, destino)
- **O Grafo:** Você receberá um grafo (dicionário Python) representando a malha aérea, onde cada aeroporto (nó) tem conexões com outros aeroportos (arestas) e o custo de combustível/distância (peso).
- **O Algoritmo:**
  - Inicie um while True: (O *loop* do Las Vegas).
  - Dentro do loop, inicie uma **nova tentativa de busca aleatória** (ex: *Random Depth-First Search* ou *Random Walk*).
  - A cada nó visitado (aeroporto), escolha aleatoriamente o próximo nó, **desde que** ele respeite a Restrição 1 (Alcance).
  - Se a busca "travar" (chegar a um aeroporto sem saídas válidas), a tentativa falhou.

EXERCÍCIO: A TAREFA DE IMPLEMENTAÇÃO

- **A Verificação:**
  - Se a busca aleatória encontrar o destino (Boca Raton):
  - **Verifique** se o caminho encontrado satisfaz as Restrições 2 e 3 (Paradas < 7 e Custo Total < 15.000).
  - Se sim, return caminho (Resposta Determinística).
  - Se não (ou se a tentativa falhou), o while True garante que o algoritmo **recomece** do zero.
- **Reflexão Final:** Por que este é um algoritmo Las Vegas? Porque garantimos uma resposta *correta* (um caminho que satisfaz todas as restrições), mas não temos ideia de *quanto tempo* (quantas tentativas) levará para encontrá-lo.