Documento Explicativo: Projeto AI Programming Benchmark

1. Em Que Consiste o Projeto?

- **Objetivo Principal:**
    - Este projeto consiste numa ferramenta de software desenhada para realizar uma avaliação comparativa e objetiva das capacidades de diferentes modelos de Inteligência Artificial (como Claude, ChatGPT e Gemini) na área específica da programação. O foco é entender quão bem estas IAs conseguem gerar código Python correto e apropriado para resolver problemas algorítmicos padronizados.
- **Problema Abordado (Motivação):**
    - As IAs generativas estão a tornar-se ferramentas comuns no desenvolvimento de software, mas avaliar a sua real proficiência em programação é um desafio. Muitas avaliações são subjetivas ou baseadas em exemplos limitados. Este projeto procura resolver isso, fornecendo uma metodologia de benchmark padronizada e quantificável para comparar modelos de forma justa e identificar as suas verdadeiras capacidades e limitações em tarefas de codificação.
- **O Que a Ferramenta Faz (Visão Geral Funcional):**
    - A ferramenta automatiza o processo de apresentar um conjunto diversificado de desafios de programação a cada IA.
    - Recolhe as soluções (código) propostas por cada uma.
    - Analisa a qualidade básica desse código (através de ferramentas como Flake8, embora a análise em si não seja o foco deste documento).
    - **Crucialmente, executa o código gerado contra um conjunto de testes predefinidos para verificar se a solução funciona corretamente e produz os resultados esperados.**
    - Agrega os resultados (sucesso/falha nos testes, tempo de resposta da API, etc.) para permitir uma comparação informada entre os modelos de IA.
    - O objetivo final é gerar dados que mostrem qual IA é mais eficaz, consistente ou rápida para diferentes tipos de problemas de programação.

2. Os Desafios de Programação: Técnicas e Finalidades

- **Introdução à Seleção de Desafios:**

    1. A seleção dos desafios foi feita com o intuito de cobrir um espectro variado de conceitos fundamentais da ciência da computação, estruturas de dados e técnicas algorítmicas. A inclusão de desafios com diferentes níveis de dificuldade (Fácil, Médio, Difícil) permite avaliar as IAs em cenários diversos. A seguir, detalha-se cada desafio, a técnica de programação tipicamente

associada à sua resolução e a finalidade de incluir esse teste específico no benchmark.

- **Descrição Detalhada de Cada Desafio:**

    1. **Fibonacci Sequence**

        - **Descrição Breve:** Gerar os primeiros N números da sequência de Fibonacci (0, 1, 1, 2, 3...).
        - **Técnica Esperada:** Iteração com duas variáveis para guardar os dois números anteriores, ou recursão (com potencial risco de ineficiência para N grande se não houver memoização).
        - **Finalidade do Teste:** Avaliar a compreensão de sequências matemáticas básicas e a implementação de algoritmos iterativos ou recursivos simples.
    2. **Binary Search**

        - **Descrição Breve:** Encontrar o índice de um valor alvo numa lista *ordenada* de inteiros.
        - **Técnica Esperada:** Algoritmo de divisão e conquista, comparando o alvo com o elemento do meio da lista e descartando metade da lista a cada passo.
        - **Finalidade do Teste:** Avaliar a capacidade de implementar algoritmos de busca eficientes (complexidade O(log n)) que requerem o pré-requisito de dados ordenados.
    3. **Merge Sort**

        - **Descrição Breve:** Ordenar uma lista de inteiros em ordem crescente devolvendo uma *nova* lista ordenada.
        - **Técnica Esperada:** Algoritmo de divisão e conquista recursivo: divide a lista ao meio repetidamente, ordena as metades e depois junta (merge) as metades ordenadas.
        - **Finalidade do Teste:** Avaliar a implementação de algoritmos de ordenação recursivos clássicos e eficientes (complexidade O(n log n)), incluindo a importante etapa de "merge".
    4. **Count Islands in Grid**

        - **Descrição Breve:** Contar o número de "ilhas" (grupos de 1s conectados horizontal ou verticalmente) numa grelha 2D, onde 0s representam água.
        - **Técnica Esperada:** Algoritmos de travessia de grafo aplicados à grelha, como Busca em Profundidade (DFS - Depth-First Search) ou Busca em Largura (BFS - Breadth-First Search), para explorar e marcar as células pertencentes a uma ilha já visitada.
        - **Finalidade do Teste:** Testar a compreensão e aplicação de algoritmos de travessia (grafo/matriz), essenciais para problemas de conectividade e exploração espacial.
    5. **Palindrome Check**

- **Descrição Breve:** Verificar se uma string é um palíndromo (lê-se da mesma forma de trás para frente), ignorando maiúsculas/minúsculas e caracteres não alfanuméricos.
- **Técnica Esperada:** Filtrar e normalizar a string (remover não alfanuméricos, converter para minúsculas) e depois comparar a string resultante com a sua versão invertida, ou usar a técnica de dois ponteiros (um no início, outro no fim) que se movem em direção ao centro.
- **Finalidade do Teste:** Avaliar a capacidade de manipulação básica de strings (filtragem, normalização, inversão) e lógica de comparação simples.

6. **Valid Parentheses**

    - **Descrição Breve:** Determinar se uma string contendo apenas parênteses '()', '{}', '[]' é válida (os parênteses abertos são fechados na ordem correta e pelo tipo correspondente).
    - **Técnica Esperada:** Uso de uma estrutura de dados de Pilha (Stack). Empilha-se um parêntese de abertura quando encontrado. Ao encontrar um parêntese de fecho, verifica-se se o topo da pilha é o parêntese de abertura correspondente; se for, desempilha-se, senão a string é inválida.
    - **Finalidade do Teste:** Testar a compreensão e o uso correto de pilhas para resolver problemas de validação de sequências e correspondência de pares.

7. **LRU Cache (Least Recently Used Cache)**

    - **Descrição Breve:** Implementar uma cache com capacidade fixa que remove o item menos recentemente usado quando a capacidade é excedida. Deve suportar operações `get` (obter valor e marcar como usado) e `put` (inserir/atualizar valor e potencialmente remover o LRU).
    - **Técnica Esperada:** Combinação de um Dicionário (Hash Map) para acesso rápido (O(1)) aos valores pela chave, e uma estrutura para manter a ordem de uso (tipicamente uma Lista Duplamente Ligada ou um `OrderedDict` em Python) para permitir a identificação e remoção do item menos usado também em O(1).
    - **Finalidade do Teste:** Avaliar a capacidade de projetar e implementar estruturas de dados customizadas mais complexas que combinam diferentes primitivas para otimizar operações comuns.

8. **Two Sum**

    - **Descrição Breve:** Dada uma lista de números e um alvo, encontrar os *índices* de dois números na lista que somam exatamente o alvo.
    - **Técnica Esperada:** Uso eficiente de um Dicionário (Hash Map). Itera-se pela lista e, para cada número, verifica-se se o "complemento" (`target - numero_atual`) já foi visto (ou seja, se

está no dicionário). Se sim, encontraram-se os dois números. Se não, adiciona-se o número atual e o seu índice ao dicionário.
    - **Finalidade do Teste:** Avaliar o uso inteligente de dicionários/hash maps para otimizar buscas, transformando um problema potencialmente O(n^2) (se comparássemos todos os pares) num problema O(n).
9. **Longest Substring Without Repeating Characters**
    - **Descrição Breve:** Encontrar o comprimento da maior substring dentro de uma string que não contenha caracteres repetidos.
    - **Técnica Esperada:** Técnica da Janela Deslizante (Sliding Window). Mantém-se uma "janela" (uma substring candidata) e um conjunto (Set) ou dicionário para rastrear os caracteres presentes na janela atual. Expande-se a janela para a direita; se um caractere repetido for encontrado, contrai-se a janela pela esquerda até que a repetição seja removida. O comprimento máximo da janela é registado.
    - **Finalidade do Teste:** Testar a compreensão e aplicação da técnica de janela deslizante, um padrão comum para resolver problemas de otimização em sequências/arrays.
10. **Minimum Edit Distance**
    - **Descrição Breve:** Calcular o número mínimo de operações de edição (inserção, deleção, substituição de um caractere) necessárias para transformar uma palavra noutra.
    - **Técnica Esperada:** Programação Dinâmica, especificamente o algoritmo de Levenshtein. Usa-se uma matriz 2D onde cada célula `(i, j)` armazena a distância mínima entre os primeiros `i` caracteres da primeira palavra e os primeiros `j` caracteres da segunda. Cada célula é calculada com base nos valores das células vizinhas (esquerda, cima, diagonal) e na igualdade dos caracteres correspondentes.
    - **Finalidade do Teste:** Avaliar a capacidade de aplicar programação dinâmica, uma técnica poderosa para resolver problemas de otimização que podem ser divididos em subproblemas sobrepostos.

3. Conclusão Geral sobre os Testes

A variedade dos desafios selecionados garante que as IAs sejam testadas numa gama de competências essenciais para um programador: desde algoritmos básicos e manipulação de dados até estruturas de dados mais complexas e técnicas avançadas como programação dinâmica e janela deslizante. A avaliação não se foca apenas na geração de código sintaticamente correto, mas sim na sua **capacidade funcional** de resolver o problema proposto, conforme validado pelos casos de teste. Isso fornece uma medida mais realista da utilidade prática das IAs em tarefas de programação do mundo real.