Régua-Puzzle

Matheus Rubio

Giovane Nilmer

UFJF
Universidade Federal de Juiz de Fora

Departamento de
Ciência da Computação

Problemática (Régua-Puzzle)

O jogo das fichas(régua-puzzle) consiste em um jogo de tabuleiro com uma única linha, um espaço vazio e um número igual de fichas pretas e brancas.

Exemplo de objetivo

Colocar as peças pretas entre as peças brancas.

Operações Permitidas

- Deslizar uma ficha para um espaço vazio.
- Saltar uma ficha sobre outra em direção ao espaço vazio.

Implementação

Linguagem utilizada: Typescript

Principais classes:

- **Logger.ts**
  - Auxilia nos logs da aplicação.
- **Tree.ts**
  - Árvore de solução
- **Node.ts**
  - Representação do nó da árvore
- **Edge.ts**
  - Representação das arestas presentes na árvore
- **PriorityQueue.ts**
  - Funções para a fila de prioridades (utilizada na Busca Gulosa e A*)

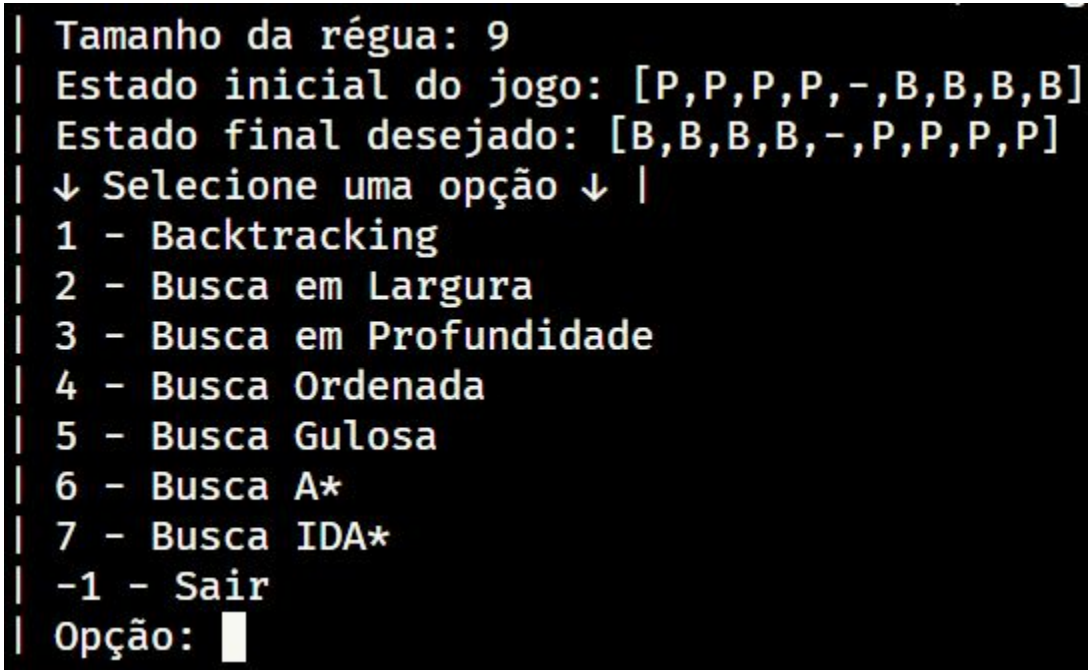
Formato de entrada

Para poder executar qualquer um dos algoritmos somente 4 parâmetros são necessários:

- **Tamanho da "régua"**
  - Deve ser sempre um número ímpar e maior que 3, que é o mínimo possível.
- **Estado inicial do jogo**
  - String separada por vírgula onde cada caractere entre as vírgulas só poderá ser 'P'(Preto), 'B'(Branco) e '-'(Vazio) **EX:** `P,P,P,P,-,B,B,B,B`
  - O número de P's e B's deve ser igual
  - Somente poderá possuir um único espaço vazio '-'
- **Estado final desejado**
  - Mesma regra do estado inicial mas também deve ser diferente do estado inicial
- **Tamanho máximo da árvore de solução**
  - Sem muitas restrições, só precisa ser maior que 0

Menu

Após preencher as informações de entrada o menu abaixo é apresentado



Menu

Para todos os algoritmos no menu disponíveis para execução, mais um parâmetro é solicitado ao usuário que é o tamanho máximo da árvore de solução, a presença desse parâmetro é necessária principalmente para evitar estouros de memórias em alguns casos.

Resultado da Execução

Ao fim da execução do algoritmo a seguinte tela é apresentada para o usuário:

```
----------------Resultado do Backtracking-----------------
| Nós visitados: 26492
| Nós expandidos: 6732
| Custo da solução: 8
| Caminho da solução: P,P,-,B,B -> P,-,P,B,B -> P,B,P,-,B -> P,B,P,B,- -> P,B,-,B,P -> -,B,P,B,P -> B,-,P,B,P -> B,B,P,-,P -> B,B,-,P,P
| Fator de ramificação: 2.94
| Tempo de execução: 0.271s
----------------------------------------------------------
```

Principais funções

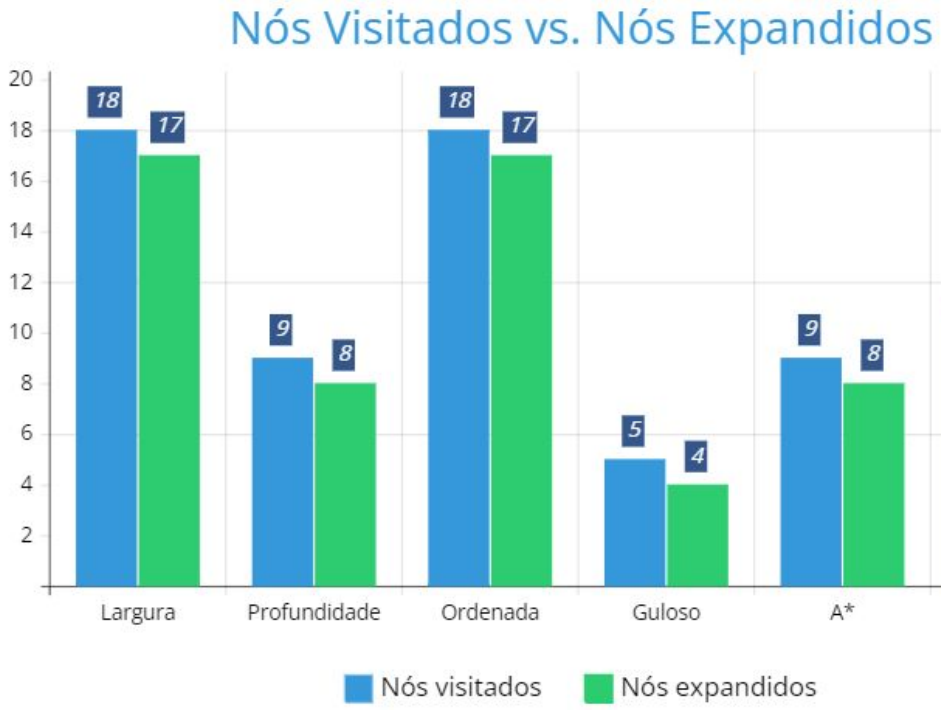
`main` - Contém o menu e um *loop* que, enquanto a opção selecionada for diferente de -1, permitirá ao usuário escolher um algoritmo para aplicar de acordo com as entradas fornecidas, e uma verificação da validade destas.

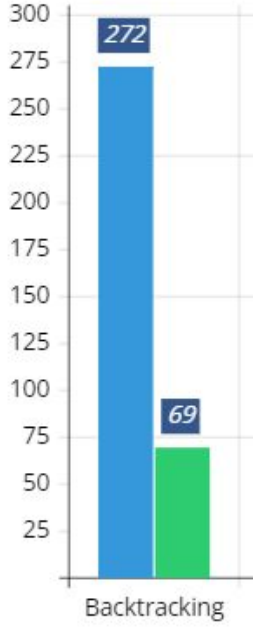
`getAllPossibilitiesFromNode` - A partir da posição do espaço vazio no array, verifica possíveis movimentos de 1 espaço ex.: [P,-,P,B,B] → [P,P,-,B,B]

ou 2 espaços ex.: [P,-,B,P,B] → [P,P,B,-,P]

`getSelectedMaxDepth` - Permite ao usuário escolher um número inteiro positivo para definir a profundidade máxima que a árvore poderá atingir em sua execução.

Estatísticas de Execução - Régua de 5

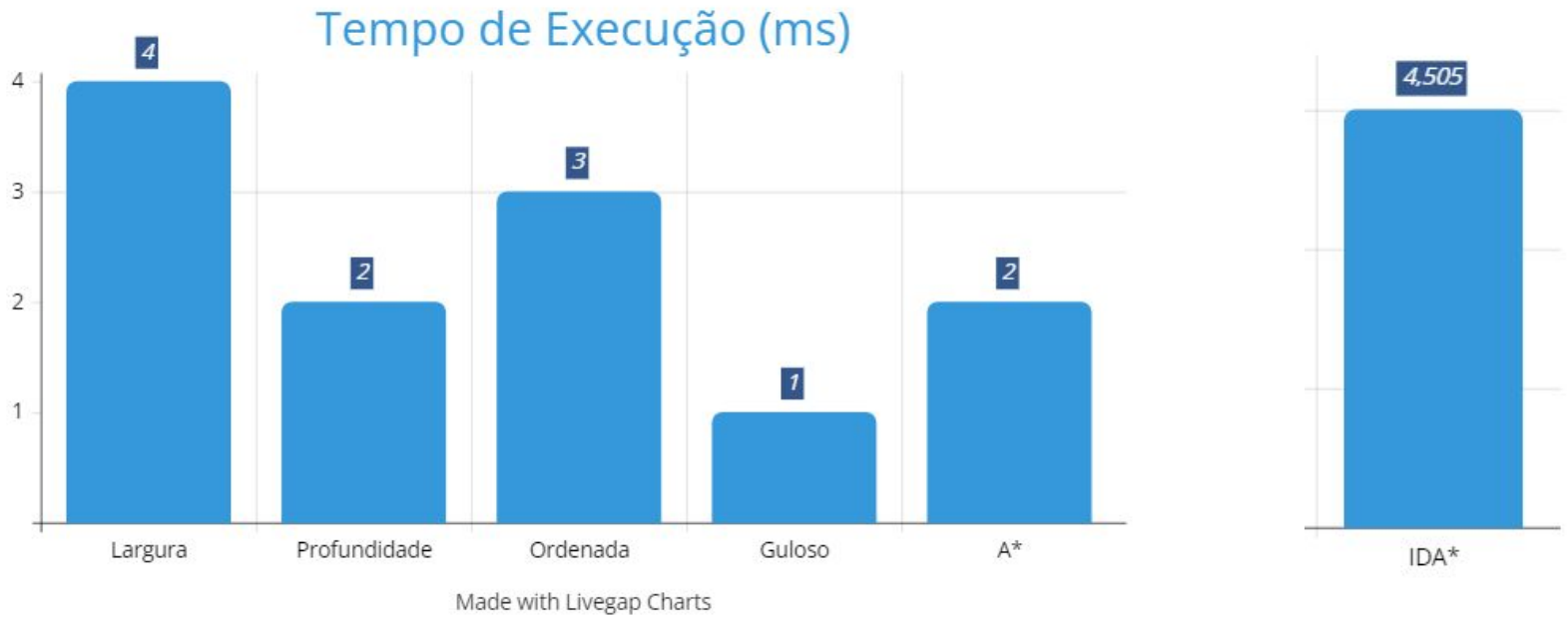




Estado inicial → [-,P,B,B,P]

Objetivo → [B,B,-,P,P]

Estatísticas de Execução - Régua de 7



Estado inicial → [P,-,B,B,P,P,B]

Objetivo → [B,B,B,-,P,P,P]

Principais dificuldades

- Definição da *Heurística* (para os algoritmos *Guloso*, *A**, *IDA**;
- "Estouro" de memória a depender do tamanho da régua;
- Tentativa de no backtracking não gerar estados repetidos 🥲

OBRIGADO!