SCC0218 - Projeto 1 - Futoshiki

Integrantes: **Nº USP:**

- ❏ Víctor de Aquino Amorim 9277642
- ❏ Zoltán Hirata Jetsmen 9293272

1. Introdução

Futoshiki é um jogo de *puzzle* baseado em um tabuleiro. É jogado sobre um tabuleiro quadrado com um determinado tamanho fixo (4x4, por exemplo, podendo chegar até 9x9).

O objetivo do jogo é descobrir os dígitos escondidos no interior das células do tabuleiro; cada célula é preenchida com um número entre 1 e o tamanho do tabuleiro. Em cada linha e coluna cada dígito aparece exatamente uma vez. No início do jogo algumas células podem já estar preenchidas. O jogo também pode conter algumas restrições entre as células do tabuleiro, essas desigualdades devem ser respeitadas e podem ser usadas como pistas para descobrir as células restantes.

2. Implementação

O jogo Futoshiki foi implementado em linguagem C/C++, e teve como principal componente uma *struct* exemplificada na imagem a seguir.

```
typedef struct table{
    int num; // Número da célula
    int qtdRestrictionsMin; // Quantidade de restrições existentes em que (i, j) < (i', j')
    int qtdRestrictionsMax; // Quantidade de restrições existentes em que (i, j) > (i', j')
    int min[4][2]; // Coordenadas em que (i, j) < (i', j')
    int max[4][2]; // Coordenadas em que (i, j) > (i', j')
    int definitive; // Define se o número é definitivo ou não
    list<int> * values; // Lista de possíveis valores para uma determinada coordenada
    list<int> * removedValues; // Guarda os valores removidos
    unordered_multimap<int,int> * coordinates; // Guarda as coordenadas dos valores removidos
}TABLE;
```

Figura 1 - Struct com as informações contidas em cada coordenada do tabuleiro

Foram feitas 3 implementações para a resolução do jogo, sendo uma complementar a outra: *Backtracking*, *Backtracking* + Verificação Adiante, *Backtracking* + Verificação Adiante + MVR.

- ***Backtracking*** – A função testa todas as possibilidades possíveis para encontrar a solução. Ela insere um número ‘n’ $0 < n \leq$ ‘d’ (dimensão do tabuleiro) e vai para a próxima coordenada. Caso alguma restrição ocorra (o número já exista na linha, ou na coluna ou ele é maior ou menor a uma determinada coordenada) a função soma um no valor de ‘n’. Se as possibilidades foram testadas e nenhuma foi válida, o *backtracking* é ativado e a função zera o número da coordenada atual e retorna '1' que representa um erro adiante no tabuleiro, e assim, testa outros números nas coordenadas anteriores. Quando o laço que percorre o tabuleiro chega na linha ‘d’, a função retorna 0 que representa sucesso e que o jogo foi concluído.

- ***Backtracking* + Verificação Adiante** – Logo na leitura da entrada, uma lista guarda todos os possíveis valores para a coordenada, sendo representada pela variável '*values*', já eliminado valores que entram em conflito com os já definidos no tabuleiro. Com essa lista, a função somente insere os números presentes nela, na coordenada respectiva e assim vai para próxima. Ao chamar a próxima coordenada, existe uma função auxiliar que remove todos os números das listas que entram em conflito com o colocado anteriormente no tabuleiro. Se alguma lista ficar vazia, representa que não existe nenhuma possibilidade de número para aquela coordenada e assim realiza o *backtracking*. Além de remover os números que entram em conflito, a função auxiliar '*FCVerification*' insere as coordenadas do número removido em um *unordered_multimap* para que caso ocorra o *backtracking*, a inserção ocorra somente nas coordenadas corretas. Para as restrições, como normalmente mais de um número é removido, para saber correntemente quais inserir em um suposto *backtracking*, a variável '*removedValues*' guarda esses valores. A inserção deles ocorre logo após o retorno de um conflito pela função '*insertRemoved*'.
- ***Backtracking* + Verificação Adiante + MVR** – Utiliza da mesma técnica que a Verificação Adiante, porém, não percorre o tabuleiro sequencialmente e sim nas coordenadas que possuem menos opções de números válidos.

3. Desempenho

Para avaliar-se o desempenho foram utilizadas 4 entradas distintas:

- **Tabuleiro 5x5**

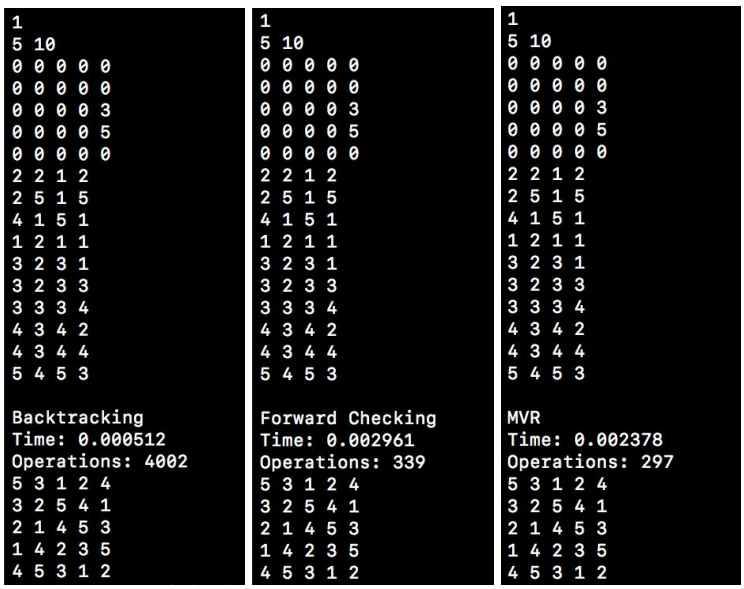

Figura 2 - Testes realizados para um tabuleiro 5x5

Como podemos observar, para tabuleiros pequenos (4x4 e 5x5) as 3 heurísticas resolvem o jogo em tempo baixo e pode-se observar a redução da quantidade do número de atribuições com a evolução da heurística. Em casos

que o *backtracking* resolve com tempo menor que as outras heurísticas, nestas ocorre um gasto de tempo devido ao grande número de manipulações que podem ocorrer com as estruturas de dados auxiliares (*list* e *multimap*).

- **Tabuleiro 6x6**

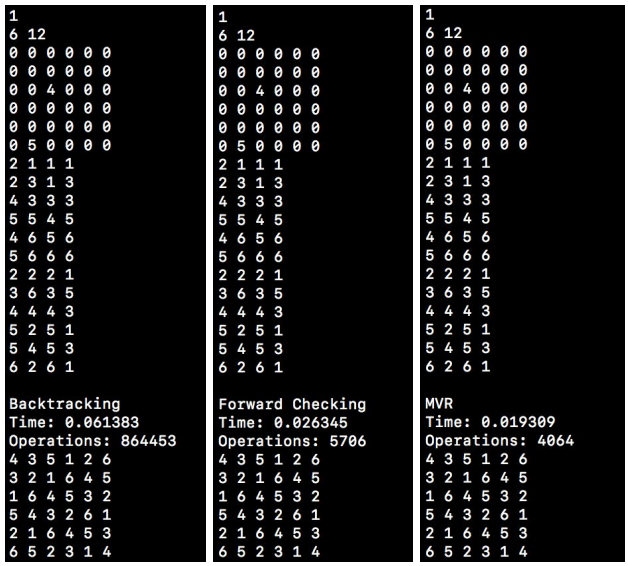

Figura 3 - Testes realizados para um tabuleiro 6x6

A partir de tabuleiros um pouco maiores começa-se a observar a necessidade de heurísticas mais avançadas, em um tabuleiro 6x6 o MVR continua apresentando menos operações e agora mesmo com as manipulações de estruturas auxiliares se torna mais rápido.

- **Tabuleiro 7x7**

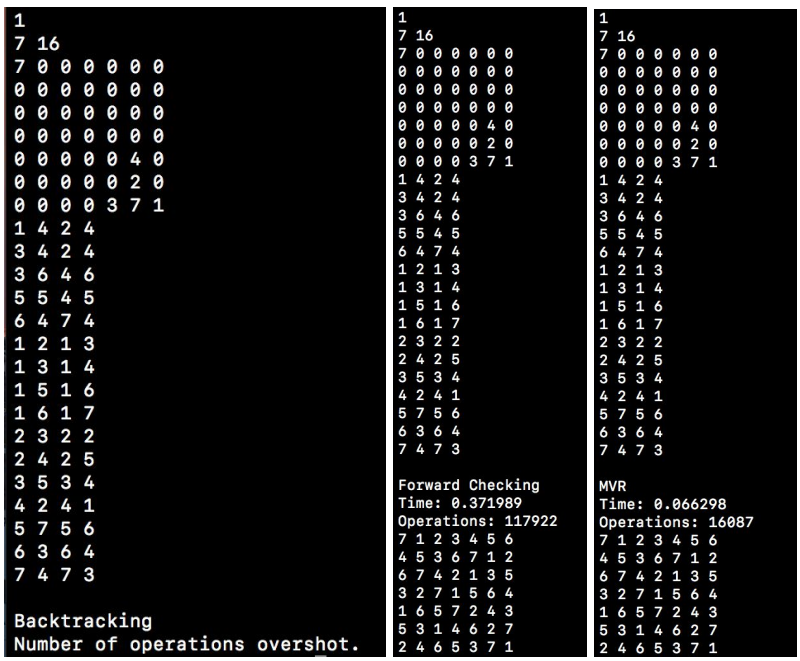

Figura 4 - Testes realizados para um tabuleiro 7x7

O *backtracking* se torna cada vez mais ineficiente com o aumento da dimensão do tabuleiro, isso se deve a falta de podas que são realizadas pelas heurísticas mais avançadas. Para o caso apresentado o número de operações realizadas no *backtracking* excede 10^6 operações e o processo é finalizado para aquele tabuleiro, enquanto o número de atribuições cai com o avanço da heurística e o tempo também diminui.

- **Tabuleiro 9x9**

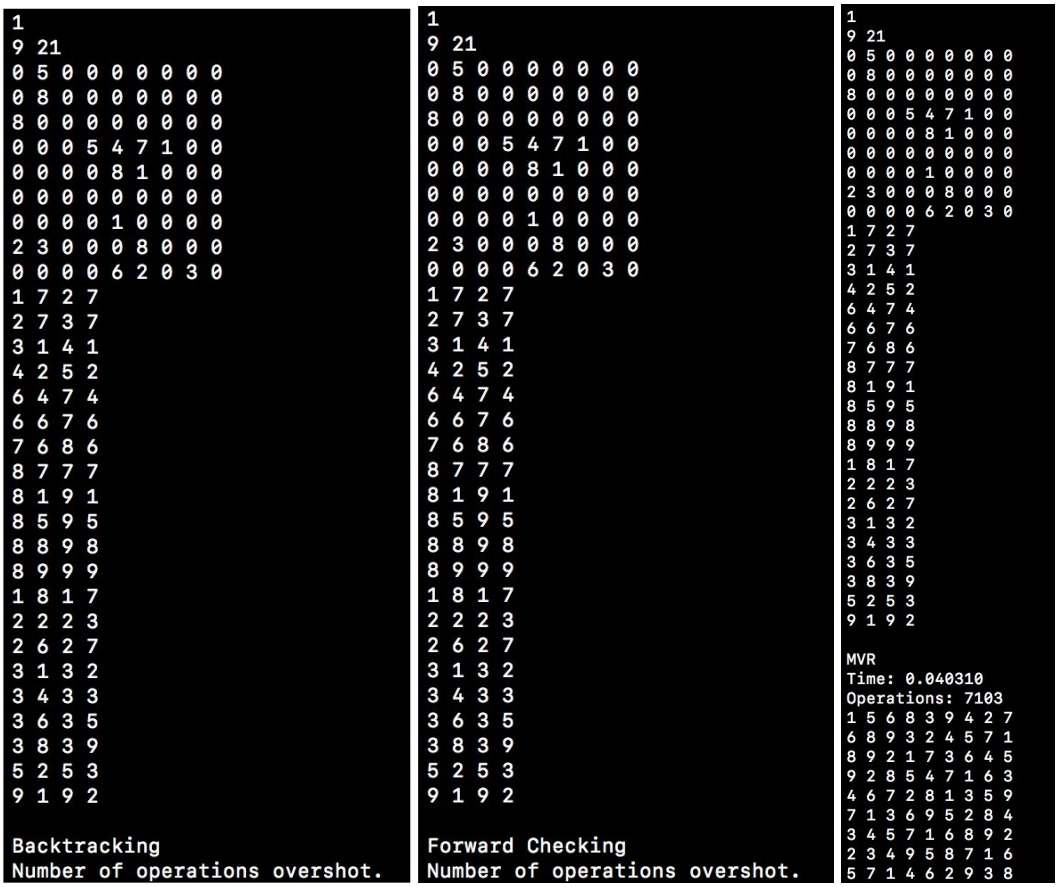

Figura 5 - Testes realizados para um tabuleiro 9x9

Podemos observar um caso em que apenas o MVR consegue resolver de forma eficiente, visto que o tempo levado é baixo e a quantidade de operações não ultrapassa e nem mesmo se aproxima do limite, enquanto as duas heurísticas anteriores ultrapassam 10^6 operações.