UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Projeto Backtracking
SuDoku

Venilton FalvoJr – 7902878

1. Introdução

Este relatório documenta o primeiro trabalho desenvolvido para a disciplina de Projetos de Algoritmos (SCC5900), que consiste na implementação de um programa capaz de resolver problemas do tipo SuDoku usando o algoritmo Backtracking (BT) e as heurísticas de Forward Checking (FC) e Minimum Remaining Values (MRV). O código-fonte submetido na plataforma Tidia-Ae também encontra-se disponível no GitHub[1].

O objetivo deste documento é explicar a estratégia de implementação utilizada e discutir brevemente seus resultados. Para isso, a Seção 2 apresenta maiores detalhes sobre os elementos da solução desenvolvida. A Seção 3 expõem uma análise de performance, sumarizando os resultados obtidos. Por fim, na Seção 4 é apresentada a conclusão deste trabalho, bem como suas limitações e trabalhos futuros.

2. Implementação

A solução implementada utiliza um algoritmo de Backtracking recursivo construído na linguagem Java em sua versão 8. A implementação conta com algumas classes responsáveis por domínios específicos do problema em questão. A seguir, cada uma dela é explanada de maneira sucinta.

2.1. br.usp.falvojr.sudoku.Main

A classe Main é a classe principal da implementação. Nela os argumentos de execução são processados com o objetivo de configurar os arquivos de entrada e as flags de ativação das heurísticas e validação, conforme a tabela abaixo:

Tabela 1: Argumentos de Execução

Argumento	Descrição	Sintaxe
-d [path]	Define um diretório para a varredura e identificação de um ou mais arquivos de entrada.	$ java -jar Project1.jar **-d ~/scc5900/Project1/src/main/resources/**
-fc	Ativa a heurística de FC.	$ java -jar Project1.jar -d [path] **-fc**
-mrv	Ativa as heurísticas de MRV e FC (conforme o enúnciado).	$ java -jar Project1.jar -d [path] **-mrv**

-f	Força a execução completa, ignorando o número máximo de tentativas (10^6).	$ java -jar Project1.jar -d [path] **-f**
		$ java -jar Project1.jar -d [path] -fc **-f**
		$ java -jar Project1.jar -d [path] -mrv **-f**

[1] Disponível em: https://github.com/falvojr-phd/scc5900/tree/master/Project1

A partir dos argumentos em questão, a classe Main percorre todos os arquivos do diretório especificado através do método nativo Files.*walk* (Java 8 ou superior). Com isso, a classe identifica os arquivos que possuem os pré-requisitos de um arquivo de entrada e processa cada um deles, estruturando e solucionando seus respectivos SuDokus através da classe Backtracking, descrita a seguir.

2.2. br.usp.falvojr.sudoku.algorithm.Backtracking

Classe responsável pela implementação do algoritmo que lhe dá nome. Essa estrutura armazena uma lista de SuDokus (representados por uma matriz de inteiros), um contador de tentativas e uma instância opcional de cada heurística.

Com base nas heurísticas atribuídas, o método *solve* aciona o método recursivo *isSolved* para cada SuDoku. O método *isSolved*, por sua vez, utilizará a estratégia adequada para a resolução do problema, dentro das seguintes possibilidades: Backtracking sem poda, Backtracking + FC ou Backtracking + FC + MRV.

2.3. br.usp.falvojr.sudoku.heuristic.ForwardCheking

A classe ForwardCheking disponibiliza um mapa global (obtido através de um Singleton) de possibilidades considerando cada coordenada do SuDoku. Para isso, a implementação de um HashMap ou LinkedHashMap (caso a heurística de MRV esteja ativa) com chaves e valores do tipo String foi utilizada.

O mapa foi idealizado dessa forma porque uma coordenada quando transformada em String se torna uma chave única (“00”, “01”, “02” etc). Além disso, seu conjunto de possibilidades também pode ser expresso da mesma forma, já que não existem caracteres que se repetem (“00”=“1237”, “01”=“267” etc).

Além disso, a classe conta com alguns métodos de apoio para a utilização desta heurística, são eles:

- ***init***: inicializa o mapa de possibilidades (atributo *domains*) com base em um SuDoku. Nesse sentido, para reduzir o esforço de busca no mapa, chaves sem possibilidades foram omitidas;
- ***syncDomains***: sincroniza as possibilidades afetadas por uma atribuição prévia. Para isso, os valores possíveis das linhas, colunas e quadrantes são atualizados;
- ***syncDomainKey***: sincroniza uma única chave, removendo de suas possibilidades um valor específico.

2.4. br.usp.falvojr.sudoku.heuristic.MinimumRemainingValues

Classe que implementa a heurística de MRV através da ordenação das chaves do mapa de possibilidades provido pela heurística de FC. Para isso, a classe possui um Comparator que é utilizado para organizar as chaves de acordo com o menor número de possibilidades. Além disso, para que a heurística possuísse controle sobre as chaves já utilizadas, um ArrayList foi criado para manter esses valores. Com isso, foi possível expor alguns métodos para que a MRV fosse utilizada pela classe de Backtraking:

- ***init***: recebe um mapa de possibilidade no padrão da heurística de FC, ordenando-o e retornando uma nova instância do tipo LinkedHashMap (ordenado através do Comparator supracitado). Além disso, esse método limpa o ArrayList com as chaves já utilizadas, caracterizando o início da heurística;
- ***sortByDomains***: ordena um mapa de possibilidades considerando o padrão da heurística de FC, organizando suas chaves em ordem crescente com base no número de possibilidades;
- ***getNextKey***: recupera a próxima chave com o menor número de possibilidades, considerando o ArryList de chaves já utilizadas;
- ***burnKey***: adiciona a chave em questão no ArrayList de chaves utilizadas.

2.5. br.usp.falvojr.sudoku.util.SuDokus

Classe responsável por disponibilizar de modo estático algumas constantes e métodos úteis a diferentes contextos, os mais relevantes são descritos abaixo:

- ***isLegal***: método responsável por verificar se um determinado valor é válido para uma linha e coluna específicas de um SuDoku. Ele percorre todas as linhas, colunas e quadrantes para identificação de possíveis violações às regras do jogo;
- ***write***: método responsável por escrever os SuDokus de acordo com os padrão de saída estabelecido no enunciado deste trabalho;
- ***generateKey***: método que cria uma chave única do tipo String considerando uma linha e coluna parametrizadas;
- ***sortByValue***: método que ordena através de um Comparator os valores de um mapa. No contexto deste trabalho, ele é usado pela heurística de MRV para a organização das chaves do mapa de acordo com os mínimos valores remanescentes de suas possibilidades.

3. Performance

Para a análise de performance duas variáveis foram mensuradas: tempo e tentativas . Para calcular o tempo foi utilizada a diferença entre duas chamadas da função System.*nanoTime* do Java. Os tempos apresentados nesta seção são as médias de 10 execuções realizadas. Por outro lado, a variável tentativas é inicializada com zero e é incrementada cada vez que um valor é verificado para atribuição. Os testes foram realizados usando o SO Ubuntu 14.04 LTS x64 em um notebook com processador 2.00 GHz Intel Core i7-2630QM e 8 GB de RAM.

As tabelas a seguir sumarizam os tempos de execução e tentativas para os casos de teste propostos para este projeto, um com 95 e outro com 10 SuDokus. A Tabela 2[2] apresenta os resultados obtidos com o limite de tentativas desativado, já na Tabela 3[2] essa restrição foi ativada.

Tabela 2: Resultados obtidos com o limite de tentativas desativado

Testes	Abordagem	Tentativas	Tempo (seg.)	Tempo Médio
95	BT sem poda	1.69E+09	21.622	0.228
	BT + FC	1.88E+08	630.269	6.634
	BT + FC + MRV	7.45E+05	9.423	0.099
10	BT sem poda	6.25E+08	8.074	0.807
	BT + FC	6.94E+07	240.325	24.032
	BT + FC + MRV	2.79E+05	3.341	0.334

Tabela 3: Resultados obtidos com o limite de tentativas ativado

Testes	Abordagem	Tentativas	Tempo (seg.)	Tempo Médio	Taxa de Sucesso
95	BT sem poda	6.98E+07	1.136	0.012	40/95
	BT + FC	3.65E+07	122.954	1.294	74/95
	BT + FC + MRV	7.45E+05	9.313	0.098	95/95
10	BT sem poda	8.70E+06	0.125	0.012	2/10
	BT + FC	5.90E+06	21.221	2.122	6/10
	BT + FC + MRV	2.79E+05	3.321	0.332	10/10

[2] A planilha com os detalhes dos dados coletados está disponível em: https://goo.gl/DKroSd.

Com base nos resultados demonstrados, apesar do FC apresentar um menor número de tentativas, o Backtracking sozinho apresentou um tempo consideravelmente menor. O provável motivo seria que ao realizar a poda e, consequentemente, reduzir o número de tentativas, o FC realiza uma quantidade alta de comparações para a sincronização de seu mapa de possibilidades, prejudicando a performance do algoritmo.

Com o objetivo de minimizar essa discrepância, a biblioteca Apache Commons Lang foi adicionada ao projeto por lidar com Strings de maneira mais performática que as implementações nativas do Java, que em grande parte utilizam expressões regulares para esse fim. Entretanto, os ganhos não foram suficientes para concluir que a manipulação de Strings é realmente o “gargalo” da solução.

Com isso, talvez a estrutura de dados definida para a heurística de FC possa ter influenciado negativamente em sua performance, tendo em vista que seria possível explorar operações de mais baixo nível como Bitwise e Bit Shift. Apesar da suspeita não foram realizados testes para comprovar tal reflexão.

Entretanto, quando a heurística de MRV foi aplicada junto da FC, os resultados foram consideravelmente melhores, tendo em vista uma redução brusca no número de tentativas. Com isso, o algoritmo se mostrou mais assertivo em suas recursões, reduzindo drasticamente seu tempo de execução.

4. Conclusão e Trabalhos Futuros

Através dos resultados obtidos foi possível concluir que a utilização de heurísticas pode reduzir significativamente o número de tentativas para um algoritmo de Backtracking. Entretanto, é necessário implementá-las de modo cuidadoso para que o ganho obtido pela poda justifique o processamento adicional para manutenção de suas estruturas auxiliares.

Como trabalhos futuros, espera-se otimizar as heurísticas de FC e MRV para a obtenção de melhores resultados. Para isso, talvez seja necessária uma refatoração analisando opções mais eficientes considerando as estruturas de dados utilizadas. Nesse sentido, um primeiro passo seria explorar operações de Bitwise e Bit Shift para o gerenciamento do mapa de possibilidades da heurística de FC, fazendo com que os esforços necessário para as atribuições e comparações fossem reduzidos significativamente.