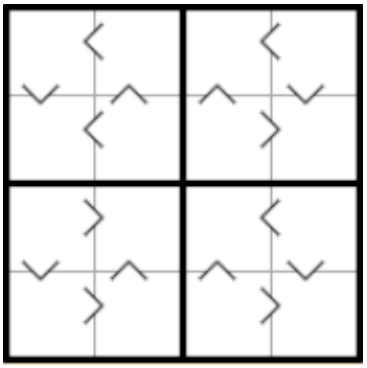
Matheus Paulon Novais - 20203078

Trabalho I: Haskell

1. Descrição

Entre as três opções disponíveis para o trabalho foi escolhida a do *Vergleichssudoku*, também conhecido como *Inequality Sudoku*(Sudoku comparativo) ou *Greater Than Sudoku*(Sudoku maior que, nome dado pelos símbolos presentes no tabuleiro). Esse jogo é uma vertente muito parecida com o sudoku comum pelos espaçamentos do tabuleiro, podendo ser 4x4(com 4 casas de 2x2), 6x6(com quatro casas de 3x2) e 9x9(com quatro casas de 3x3).

O jogo inicia com um tabuleiro de sudoku zerado, e a forma de resolvê-lo é identificar os símbolos de maior-que(“>”) e menor-que(“<”) presentes no tabuleiro, causando uma comparação direta entre as casas. Assim como aprendemos no começo da nossa caminhada no mundo matemático, a boca aberta do símbolo fica para o lado que é maior que o outro, devemos então completar os números que vão de 1 à *N* (*N* = tamanho de células em cada casa) em uma casa do tabuleiro a partir dessas noções.



Exemplo número 1: Vergleichssudoku

2. Desenvolvimento

Após compreender o funcionamento do jogo e também como jogar, o último passo era escolher uma estratégia para resolver o problema. Vendo semelhança

com o sudoku comum, a estratégia foi desenvolver um resolvedor de Sudoku normal.

A partir desse momento surgiram duas vertentes da ideia, a primeira diminuindo a quantidade de opções possíveis para cada célula a partir de combinações específicas de símbolos, ou a cada elemento testar a possibilidade do número com os números vizinhos e os símbolos apontados para a célula atual. Junto com a opinião do professor foi escolhida a segunda opção.

Com a estratégia escolhida, agora era escolher como desenvolver ela. Iniciamos com uma forma de pegar os dados da matriz de entrada, que será explicada em sequência, e comparar sequencialmente com os dados desenvolvidos pelo algoritmo.

3. Entrada e Saída de dados

A entrada é dada por uma matriz com listas que descrevem os quatros lados de cada casa, com quatro valores para indicar se contém algum sinal, e caso tenha, qual sinal é.

Cada lista segue o exemplo de [esquerda, direita, cima, baixo], e os valores possíveis para cada um deles são:

-3 -> Não contém sinal na parede indicada

-2 -> Símbolo de maior que “>”

-1 -> Símbolo de menor que “<”

Exemplo segundo a imagem do primeiro tópico:

[-3, -1, -3, -2]	[-2, -3, -3, -1]	[-3, -1, -3, -1]	[-2, -3, -3, -2]
[-3, -1, -1, -3]	[-2, -3, -2, -3]	[-3, -2, -2, -3]	[-1, -3, -1, -3]
[-3, -2, -3, -2]	[-1, -3, -3, -1]	[-3, -1, -3, -1]	[-2, -3, -3, -2]
[-3, -2, -1, -3]	[-1, -3, -2, -3]	[-3, -2, -2, -3]	[-1, -3, -1, -3]

O resultado é dado diretamente na saída do console após a execução do programa.

4. Resolução do problema

Para resolver o problema, iniciamos com o resolvedor de sudoku comum, e a partir dele foi criado funções para poder processar os dados da matriz de entrada e da matriz feita pelo programa, e futuramente essas funções foram utilizadas para criar uma validação de cada elemento que é processado e adicionado a matriz final.

A ideia consiste em dividir em três partes o teste em cima dos dados passados, primeiro checamos a matriz de entrada com os símbolos, em uma segunda função fazemos uma comparação com os dados da matriz do tabuleiro para que em uma última função façamos os testes para todos os elementos, retornando um booleano que permitirá o *backtracking* continuar caso verdadeiro e o mandará um passo para trás caso seja falso.

Esses testes e condições são auxiliados por funções nativas da biblioteca *Prelude* e da biblioteca *Data.List*.

5. Dificuldades

Quanto às dificuldades encontradas no trabalho, a primeira e maior delas foi compreender como implementar a solução esperada e escolher como fazer o processamento desses dados. A partir desse momento, a cada passo foi se encontrando um novo tipo de problema a ser resolvido, como quando se foi escolhida a forma de processamento, e testar ela em uma linguagem em qual se tinha mais afinidade, se provou mais complicado do que o esperado e custando mais tempo do que deveria, dificultando e impedindo um foco geral na última parte que seria passar esse código para uma linguagem com menos conhecimento, a qual no primeiro trabalho é Haskell.

Trabalhar em Haskell sem ter as formas com as quais estamos acostumados em linguagens mais novas e ter que pensar em novos jeitos de processar elementos complexos como matrizes, foi provavelmente o mais desafiador do trabalho. O último problema, que se estendeu pelo trabalho inteiro, foi a falta de compartilhamento de ideias com membros de grupos, resultando em momentos em que em caso de obstrução de ideias era mais difícil continuar.