Relatório do Trabalho Prático

Filipe Mateus,
Evaldo Souza,
Bernardo Mota

11 de Dezembro de 2019

1 Introdução

O jogo Sudoku é uma espécie de quebra-cabeças bem popular. Sudoku em japonês é uma abreviação para uma frase que significa algo como "os números devem permanecer únicos". A ideia do jogo é extremamente simples, consiste numa tabela de 9×9 espaços, dividida em nove blocos de 3×3, assim como na Figura 1.

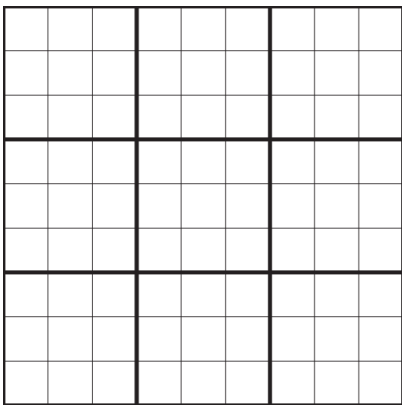

Figura 1: Sudoku em branco

Alguns desses espaços são preenchidos e fixados, dessa forma o objetivo do jogador é preencher a tabela colocando um número entre 1 e 9 em cada espaço vago, de maneira que cada linha, cada coluna e cada bloco 3×3 contenha cada dígito entre 1 e 9 exatamente uma vez. (FELGENHAUER e JARVIS, 2006)[1]

2 Desenvolvimento

Ao projetar um Sudoku tradicional o maior desafio é checar quando o quebra-cabeças foi resolvido. Duas ideias surgiram inicialmente:

- *A.* Verificar se há elementos repetidos em cada linha, coluna, ou bloco.

- B. Verificar se cada um dos nove dígitos possíveis aparecem em cada linha, coluna, ou bloco.

Entretanto é fácil observar que $A \iff B$, se um dos 9 dígitos entre 1 e 9 não foi utilizado então 9 espaços foram preenchidos com 8 dígitos diferentes, de onde pelo Princípio das Casas dos Pombos pode-se concluir que um dos dígitos ocorre mais de uma vez entre esses espaços, e vice-versa.

Para esse trabalho foi escolhida a segunda abordagem, para colocá-la em prática devemos saber se cada um dos 9 dígitos aparece respectivamente em cada uma das 9 linhas, 9 colunas, e 9 blocos, resultando em $3 \cdot (9 \cdot 9) = 243$ informações, além dessas é também necessário saber se cada uma das 81 células está preechida e se ela foi é uma das fixadas no início do jogo, o que totaliza $243 + 2 \cdot 81 = 405$ informações auxiliares.

Ao trabalhar com tantas informações é necessário organizá-las muito bem para que estejam acessíveis sempre que requisitadas. Escolhemos uma abordagem baseada em vetores bidimensionais.

Uma matriz 9×9 para as informações sobre as linhas, uma para as colunas e uma para os blocos, funcionando da seguinte maneira: se o elemento n está na linha i, coluna j e bloco k, onde $1 \leq i, j, k, n \leq 9$, então a matriz de linhas deve receber valor lógico $true$ na sua posição (i, n), a de colunas na posição (j, n) e a de blocos na posição (k, n).

Uma matriz em que cada elemento (i, j) assume $true$ se a respectiva posição está preenchida no jogo, e uma que assume $true$ caso a respectiva posição seja fixada, e $false$ caso contrário.

Entretanto movimentar entre funções essas 6 matrizes se mostrou trabalhoso e confuso, após a aula sobre estruturas de dados heterogêneas surgiu a ideia de usar apenas um vetor bidimensional de 9×9, onde cada elemento armazenasse cada uma destas 6 informações: um inteiro para o valor na célula, e 5 variáveis booleanas para respectivamente, linhas, colunas, blocos, elementos fixos e elementos preechidos. A estrutura utilizada foi a seguinte:

2.1 Estruturas e Funções

```
typedef struct{
  int value;
  bool row;
  bool column;
  bool box;
  bool full;
  bool fixed;
}element;
```

Outra estrutura utilizada foi a seguinte, que armazena um nome e um tempo:

```
typedef struct{
  char name[MAX_NAME];
  int time;
}high_score;
```

Para um trabalho bem feito foi necessário quebrar o problema em pequenas porções e criar funções para resolver cada uma, modularizando assim nosso

código. Criamos um $header file$ contendo as estruturas acima e as funções que serão explicadas abaixo:

```
void start( element sudoku[MAX_SIZE][MAX_SIZE], int level );
```

Essa função recebe como parâmetro uma matriz de elementos criados segundo a estrutura acima descrita, lê de um arquivo dado pelo nível de dificuldade em *level* um jogo aleatório entre os 15 disponíveis em cada arquivo e inicializa cada um desses elementos segundo o comportamento esperado para cada um. A variável aleatória usada é gerada usando como *seed* o tempo local dado pela função $time(0)$ da biblioteca $< time.h >$

```
bool check( element sudoku[MAX_SIZE][MAX_SIZE] );
```

Essa função recebe o jogo como parâmetro, e verifica se está completo, atualizando as informações em cada posição da matriz e verificando se todos os valores lógicos nas matrizes de linhas, colunas e blocos estão em $true$, nesse caso retorna $true$, caso contrário retorna $false$, o jogo não está resolvido.

```
void update(element sudoku[MAX_SIZE][MAX_SIZE], int i, int j, int input);
```

Recebe como parâmetros o jogo, inteiros i, j e $input$. Verifica se os valores inteiros são compatíveis e insere o valor de $input$ na posição (i, j) e faz as alterações necessárias nas informações auxiliares.

```
int box_number( int i, int j );
```

No contexto do jogo é muito fácil saber qual a linha e a coluna de um dado elemento, já que o vetor bidimensional é organizado em função de linhas e colunas, entretanto saber em qual bloco ele está não é tão intuitivo, trabalhamos então para encontrar uma relação entre o numero da linha, o numero da coluna e o numero do bloco. Esse é o objetivo da presente função, ela recebe inteiros i e j e retorna o número do bloco em que está o elemento da linha i e coluna j, contando de cima para baixo, da esquerda para a direita.

```
void get_highscore( char* buffer, char* name, int time, int level );
```

Recebe duas $strings$, $buffer$ e $name$, e um inteiro $time$. Primeiramente abre um arquivo de texto que contém os nomes e melhores tempos para a dificuldade selecionada por $level$, compara se a variável $time$, que contém o tempo entre o início e o fim de jogo é melhor do que o último tempo da lista, nesse caso o registro com o nome em $name$ e o tempo em $time$ é colocado no lugar dessa última posição e o vetor de 10 posições contendo essas estruturas de $highscore$ é ordenado colocando o resultado mais recente na posição adequada, o algoritmo de ordenação escolhido para essa função foi o $insertion\ sort$. Em seguida, usando a função $sprint()$ essas informações são armazenadas no vetor de caracteres $buffer$ e salva no arquivo.

2.2 Interface Gráfica

Uma interface gráfica foi criada usando a biblioteca GTK3 para tornar o jogo mais agradável para o usuário final, no arquivo *main.c* estão as funções relativas ao funcionamento da interface, porém a maioria das funções relevantes para a regra de jogo e coerentes com o conteúdo da disciplina de Algoritmos e Estruturas de Dados são essas que foram explicadas e fazem parte do arquivo de cabeçalho *sudoku.h*, e estão implementadas em *sudoku.c*.

Uma das funções que está em *main.c* por depender de estruturas do GTK mas merece ser citada aqui é:

```
void on_btn_help_clicked()
```

A função usa indicadores de $linha$ e $coluna$ para procurar quais números são elegíveis para figurar na posição dada por $(linha, coluna)$, ela percorre a linha $[linha]$ na matriz de linhas, a linha $[coluna]$ na matriz de colunas e a linha $[boxnumber(linha, coluna)]$ na matriz de blocos, buscando por ocorrências de valor lógico $false$ simultâneamente nos três casos, o que quer dizer que o elemento iterador i naquele instante tem um número que não aparece na linha, nem na coluna, nem no bloco, sendo assim elegível para ser colocado naquele espaço, então o respectivo botão é pintado colorido de azul. Para não ser injusto com os outros jogadores, é adicionado uma punição ao tempo computado no resultado final.

3 Considerações finais

O resultado final foi um Jogo de Sudoku que chamamos $< C > doku$, onde o jogador conta com uma interface gráfica, pode escolher entre três dificuldades diferentes, possui um sistema de dicas, sabe se o jogo foi resolvido corretamente e ainda é capaz de competir com outros jogadores pelo melhor tempo naquela respectiva dificuldade. Muitas dificuldades foram enfrentadas no planejamento e na execução desse trabalho, entre elas a de saber quando um sudoku realmente está resolvido, aprender a usar interfaces gráficas integradas com código em C, e lidar com arquivos para leitura e escrita de dados formatados importantes em posições específicas. Algumas outras funções poderiam ter sido implantadas caso houvesse mais tempo, como uma espécie de criptografia nos arquivos de texto para que não pudessem ser facilmente adulterados, ou a possibilidade de ler do usuário um jogo e salvar em arquivo para ser jogado depois, ainda assim foi um trabalho que trouxe muito aprendizado nos conceitos básicos de Algoritmos e Estrutura de Dados.

Referências

[1] Felgenhauer Bertram; Jarvis, Frazer. Mathmatics of sudoku i. mathematical spectrum, v. 39, n. 1, p. 15-22, 2006., 2006.