Jose Diego - 1761, Vitor - 3049

Trabalho de Projeto e Análise de Algoritmos

Brasil

25 de Outubro de 2018

Jose Diego - 1761, Vitor - 3049

Trabalho de Projeto e Análise de Algoritmos

Trabalho referente a implementação de algoritmos para resolução de problemas baseado em técnicas de tentativa e erro, Bracktracking, para a disciplina de Projeto e Analise de Algoritmos.

Universidade Federal de Viçosa

Campos Florestal

Ciência da Computação

Brasil

25 de Outubro de 2018

Resumo

Em ciência da computação vários são os problemas enfrentados na implantação de uma solução baseado em um contexto da vida real, abstrair uma solução viável é sempre um desafio. Técnicas como Backtracking vem ajudar os profissionais nessas áreas neste momento fornecendo meios pelo quais problemas podem ser divididos em sub-problemas parciais, e que estes podem serem explorados exaustivamente .

Palavras-chaves: algoritmos, Backtracking, sudoku, matriz.

Lista de ilustrações

Sumário

1 Introdução

Como já foi dito em Ciência da computação vários são os problemas enfrentados, e a divisão desses problemas em subproblemas aparentementes mais simples tornou se uma solução, técnicas como Backtracking tornaram-se largamente usadas. Os algoritmos de tentativa e erro compõem as seguintes características, primeiro passos em direção a solução final são avaliados e registrados, caso eles não levem a uma solução final devem serem excluídos dos resultados. Neste trabalho veremos essas técnicas sendo utilizadas em duas tarefas ao longo do desenvolvimento.

2 Metodologia

A metodologia adotada consiste primeiramente, no estudo teórico da técnica de tentativa e erro ,Backtracking, para a solução de problemas, em seguida a implementação destes na linguagem C conforme a descrição das tarefas contidas no relatório do trabalho prático.

Todas os exemplos e medições foram feitas em um computador com um processador Intel i5 (2.4GHz), 6GB de memória RAM, em uma distribuição linux. E o relatório foi elaborado em cima da ferramenta overleaf em látex, o que possibilitou padronização, além de desenvolvimento compartilhado.

3 Desenvolvimento

3.1 Tarefa A

3.1.1 Descrição da Tarefa

A tarefa A tem como objetico implementar em C, utilizando o método do backtracking para resolver palavras cruzadas.Tendo esse problema em mente devemos deinir algumas regras a este:

1. As palavras sempre comecaram de baixo para cima ou da esquerda para direita.
2. No maximo haverá 2 letras em sequencia em qualquer direção.

3.1.2 Implementaçao

Com acordo a proposta do trabalho, o algoritimo foi implementado em C, utilizando as bibliotecas *stdio.h* e *stdlib.h*. O grande diferencial foi a implentação da logica de backtracking para resolver o dado problema, para isso foi se utilizado 2 matrizes de char uma contendo as letras lida de uma arquivo e outra para guardar as posiçoes das ocorrencias das palavras. Na figura 5 pode se observar parte do codigo usado na implementação do backtracking.

Figura 1: Imagem do codigo de backtracking

```
int backtracking(Matriz *letras, char *palavra,int analise) {
    int i, j;
    int contrecusi=0;
    int ocorrencia=0;
    for (i = 0; i < letras->linha; i++) {
        for (j = 0; j < letras->coluna; j++) {
            if (letras->palavraCruzadas[i][j] == palavra[0])
                if (j > 0) {
                    contrecusi++;
                    backtrackingOp(letras, palavra, i, j, 'b',&contrecusi,&ocorrencia);
                } else {
                    contrecusi=contrecusi+2;
                    backtrackingOp(letras, palavra, i, j, 'b',&contrecusi,&ocorrencia);
                    backtrackingOp(letras, palavra, i, j, 'd',&contrecusi,&ocorrencia);
                }
        }
    }
    printf("Foram achadas %d ocorrencias",ocorrencia);
    if(analise)
     printf("Foram feitas %d chamadas recusivas",contrecusi);
}
```

Essa função teve como objetivo ler toda matriz, afim de achar alguma letra que

corresponda a primeira letra da palavra para que assim fosse chamado o método *backtrackingOp*. A baixo segue a figura 6, com o codigo usado no *backtrackingOp*.

Figura 2: Imagem do codigo de backtrackingop

```
int backtrackingOp(Matriz *letras, char *palavra, int i, int j, char direcao,int *contrecusi,int *ocorrencia) {
    if (letras->palavraCruzadas[i][j] == palavra[0]) letras->final[i][j] = '*';
    if(letras->palavraCruzadas[i][j] != palavra[0]) return 0;
    palavra++;
    if (direcao == 'b') {
        i++;
        if (letras->palavraCruzadas[i][j] == palavra[0] && palavra[1] == '\0') {
            (*ocorrencia)++;
            letras->final[i][j] = '*';
            return 1;
        } else {
            if (j < letras->coluna) {
                (*contrecusi)++;
                backtrackingOp(letras, palavra, i, j, 'd',contrecusi,ocorrencia);
            }
            if (j > 0) {
                (*contrecusi)++;
                backtrackingOp(letras, palavra, i, j, 'e',contrecusi,ocorrencia);
            }
        }
    } else if (direcao == 'e' || direcao == 'd') {
        if (direcao == 'e') j--;
        else j++;
        if (letras->palavraCruzadas[i][j] == palavra[0] && palavra[1] == '\0') {
            (*ocorrencia)++;
            letras->final[i][j] = '*';
            return 1;
        } else {
            if (i < letras->linha) {
                (*contrecusi)++;
                backtrackingOp(letras, palavra, i, j, 'b', contrecusi,ocorrencia);
            }
        }
    }
}
```

Seguindo as especificações para a tarefa A passadas na descriçao do TP, essa função pecore a matriz de letras em busca da dada palavra.

Como fonte de pesquisa para concretizar essa logica foi utilizado como suporte os slides passados em sala e alguns sites, assim como a ajuda de alguns colegas.

3.1.3 Execução e Resultados

Para executar o progama bastar entrar com o arquivo desejado e em seguida escolher entre verificar a martiz de letras ou já selecionar uma palavra a qual será procurada na matriz.

Na figura vemos melhor como esta a interface.

Na proxima figura vemos o progama no modo analise, que pode ser mudado simplesmente muadando o *define modoanalise* para 1, nesse modo o progama nos informa quantas recussividades foram chamadas.

Figura 3: Imagem da interface

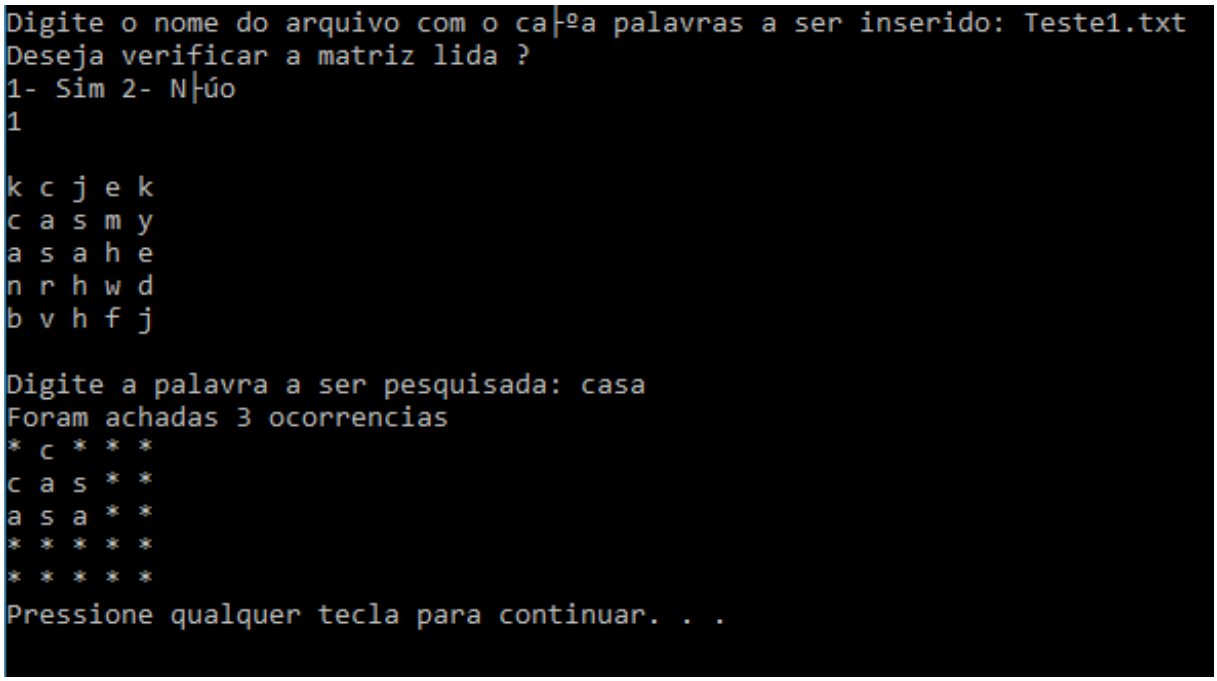
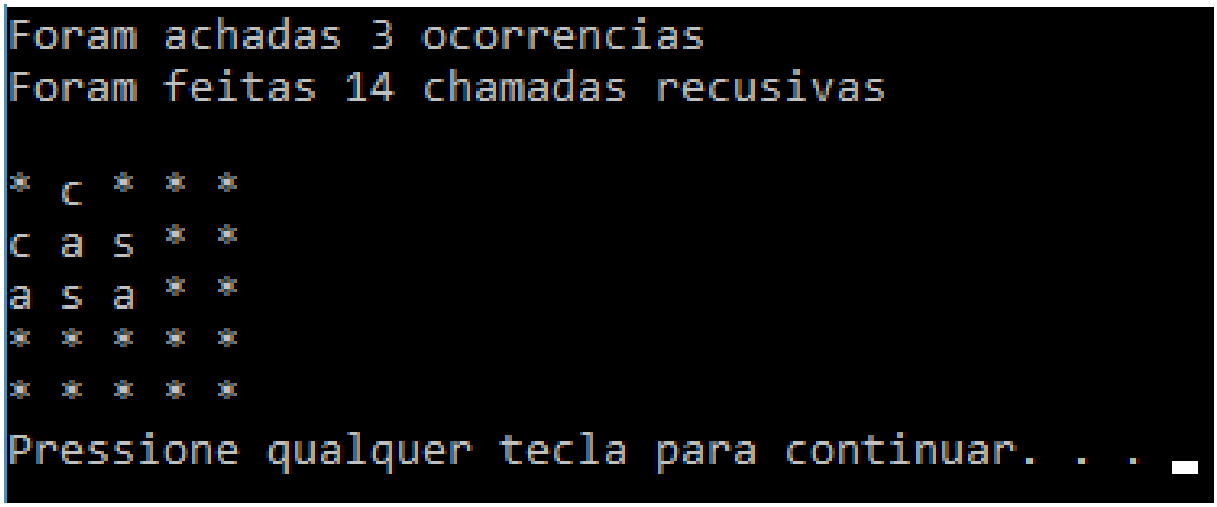


Figura 4: Imagem da interface



3.2 Tarefa B

3.2.1 Descrição da Tarefa B

A tarefa B tem como objetivo implementar em C um algoritmo com backtracking para resolver puzzles sudoku. Para isso devemos ter em mente como funciona o puzzles sudoku, ele consiste de uma serie de regras que são:

1. Não devera haver numeros iguais em uma mesma linha
2. Como também não devera haver numeros iguais em uma mesma coluna.
3. Além de não podera haver numeros iguais em uma mesma grade de 3x3

Logo abaixo podemos ver uma imagem representando o jogo puzzles sudoku

Figura 5: Imagem do puzzles Sudoku

Logo na figura acima podemos ver de vermelho a representação visual da primeira regra, em azul a restrição da segunda regra, como também uma grade de 3x3 em verde representando nossa terceira restrição

3.2.2 Implementação

A implementação se deu na linguagem c ,seguindo boas praticas de estruturas de dados como debatidas em Ziviani (2010).

Logo na figura 2 abaixo podemos ver uma parte do código contendo a função resolve Sudoku(Sudoku *sudoku), que é chamada inicialmente no case 1 e caso 2 do menu da implementação. Esta mesma função é chamada recursivamente em seu escopo e ao final retorna se houve ou não uma solução para a tabela sudoku passada para a análise. Dentro do escopo da função resolveSudoku podemos ver também outras funções auxiliares como posicaoVaziaSudoku que é responsável por verifica se não tem mais posições vazias na tabela, além também da posicaoSeguraSudoku que analise se a posição a ser inserida é segura para o novo número, logo ela deve respeitar as regras estabelecidas pelo jogo, que já foram abordadas neste texto anteriormente em Descição da Tarefa B.

Figura 6: Função ResolveSudoku

```
196  int resolveSudoku(Sudoku *sudoku) {
197
198      int i;
199      int linhaVazia, colunaVazia;
200
201      if (posicaoVaziaSudoku(*sudoku, &linhaVazia, &colunaVazia) == 0)
202          return 1;
203
204      for (i = 1; i <= 9; i++) {
205          if (posicaoSeguraSudoku(sudoku, linhaVazia, colunaVazia, i)) {
206              (*sudoku).tentativas++;
207              (*sudoku).tabuleiro[linhaVazia][colunaVazia] = i;
208
209              if (resolveSudoku(sudoku))
210                  return 1;
211
212              (*sudoku).tabuleiro[linhaVazia][colunaVazia] = 0;
213          }
214      }
215      return 0;
216  }
```

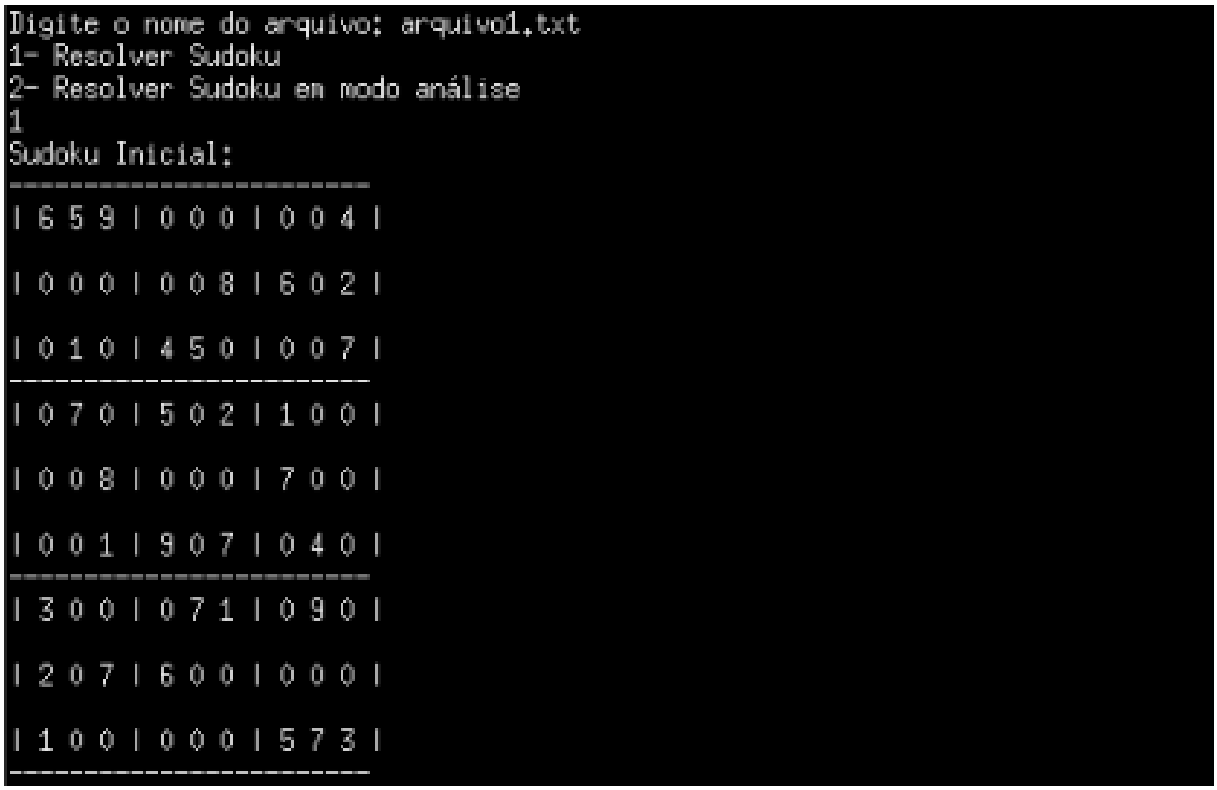
3.2.3 Execução e Resultados

A execução do programa desenvolvido é bem simples, como descrito em metodologia, a implementação se deu em um ambiente linux, o programa é composto de 3 arquivos main.c, Sudoku.c, Sudoku.h os mesmo podem serem executados em um amiente linux.

Após a execução o seguinte tela sera aberta como mostra a figura 3, a seguir.

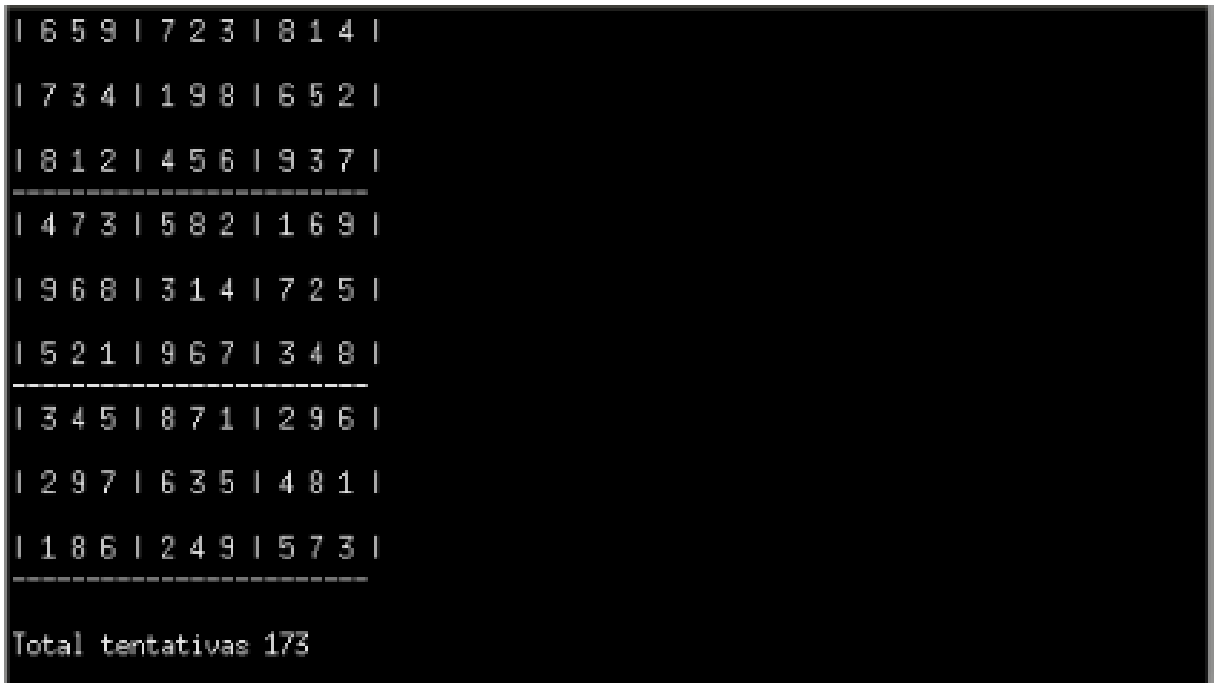
Na figura podemos ver que inicialmente é pedido que o usuário entre com o nome do arquivo a ser lido contendo os valores para a tabela sudoku, os valores devem estarem disposto no arquivo como é exibido na imagem, na tela do terminal as informações foram redesenhadas para melhor visualização.

Figura 7: Tela inicial do programa



Podemos observar também na imagem que a dois modos de exibição, para o segundo modo de resolução a seguinte resposta é obtida na figura 4, para a primeira opção o mesmo resultado é mostrado, mas ignorando o número de tentativas, esta são apenas mostradas no modo análise.

Figura 8: Informações do resutado



Conclusão

A partir desse trabalho foi possível verificar como técnicas de tentativa e erro, backtracking, são utilizadas e sua importância para a literatura. Isto foi possível graças a realizações das duas tarefas como descritas no relatório do trabalho prático , e que neste texto estão contidas na seção de desenvolvimento, nelas abordamos o backtracking e duas implementações, trazendo uma solução para um problema inicial sob o ponto de vista de vários sub-problemas, o que tornou sua solução de mais fácil compreensão.

Referências

ZIVIANI, N. *Projeto de Algoritmos*. 3 edição. ed. [S.l.]: Cengage Learning, 2010. Citado na página 10.