Editor de Texto - Documentação

Gustavo Pauzner Mezzovilla
n^{o}USP:10844320

2º Semestre de 2022

Resumo

Este é a documentação de um projeto de avaliação da disciplina MAC0122: Princípios de Desenvolvimentos de Algoritmos, ministrada pelo Professor Dr. Denis Deratani Mauá, docente do Departamento de Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP). O presente projeto encontra-se disponível no repositório do GitHub, no endereço https://github.com/M3zzinho/editor.

Is	Insere a string s na posição atual do texto
As	Carrega o conteúdo do arquivo de texto s no editor
Es	(Sobre)escreve o conteúdo do editor no arquivo de texto s
F	Move o cursor um carácter à frente
T	Move o cursor um carácter para trás
0	Move o cursor para o início da linha atual
P	Move cursor para início da próxima palavra (dentro da mesma linha)
Q	Move cursor para início da palavra atual
$	Move o cursor para o fim da linha atual
J	Ir para próxima linha (mantém a mesma coluna, ou a mais próxima dela)
H	Ir para a linha anterior (manter a mesma coluna, ou a mais próxima dela)
:x	Move o cursor para o início da linha x
:F	Move o cursor para a última linha do arquivo
D	Apaga o carácter da posição atual
DW	Apaga a palavra em que o cursor se encontra
DL	Deleta a linha atual
C	Copia valor da célula na posição atual
V	Cola o valor da célula copiada na posição atual
X	Recorta o valor da célula copiada na posição atual
Bs	Busca pela próxima ocorrência do padrão s no texto
Ss/r	Substitui toda ocorrência de s por r no texto a partir da posição atual
N	Separa linha atual na posição do cursor
U	Unir linha atual e a próxima
!	Encerra o programa

Resumo das funções implementadas

Sumário

1 Estruturas

O presente editor de texto considera as seguintes estruturas de dados:

- `console`: estrutura responsável por capturar a entrada do usuário, que consistirá de funções e strings.

```
typedef struct CONSOLE {
 char *letras;
 int tamanho;
} console;
```

- `celula`: estrutura responsável por armazenar um carácter e ponteiro para as células adjacentes.

```
typedef struct CELULA {
 char val;
 struct CELULA *next;
 struct CELULA *prev;
} celula;
```

- `linha`: estrutura responsável por armazenar uma linha de texto, que consiste de uma lista duplamente encadeada de células, além de contemplar informações do início e do final da linha, bem como o tamanho da mesma.

```
typedef struct LINHA {
 int tamanho;
 struct CELULA *head;
 struct CELULA *tail;
 struct LINHA *up;
 struct LINHA *down;
} linha;
```

- `ponto`: estrutura da variável global cursor, que consiste de um ponteiro para a célula atual, além de os inteiros especificando sua linha e sua coluna.

```
typedef struct PONTO {
 int linha;
 int coluna;
 struct CELULA *cel;
} ponto;
```

As funções implementadas se dividem nas seguintes categorias: As que simplesmente realizam uma ação (como mover o cursor, apagar uma linha, etc.) sem nenhum parâmetro a mais, e as que dependem de um *prompt* de texto como entrada. Portanto, é possível compor as funções que não

dependem de entradas de texto, e.g., mover o cursor 3 unidades pra trás, deletar um caracter e inserir outro, como ilustrado abaixo (1).

Listing 1: Exemplo de composição de funções

```
1 ------------------------
2 >0|vasa
3       ^
4 ------------------------
5 (0,4): T3DIc
6 ------------------------
7 >0|casa
8    ^
9 ------------------------
```

A utilização de listas duplamente encadeadas foi considerada para a implementação do editor de texto, pois a inserção e remoção de elementos em uma lista duplamente encadeada é feita em tempo constante, enquanto que em uma lista simplesmente encadeada, a inserção e remoção de elementos é feita em tempo linear.

Utilizar estruturas diferentes para as células e para listas também foi importante visando a navegabilidade do texto, em conjunto com a performance. O único problema dessa abordagem é no quesito complexidade de código. Dessa forma, a implementação da ferramenta de busca foi mais simplória, donde optou-se por verificar desde o início do documento até o final linearmente. Para auxiliar essa abordagem, foi considerado o algoritmo de Knuth-Morris e Pratt, que consiste em um algoritmo de busca de padrões em strings, que possui complexidade de tempo linear no número de caracteres no pior caso.

Esse algoritmo foi desenvolvido para rodar em qualquer sistema operacional, contudo, para utilizar acentos gráficos e caracteres especiais, é necessário que o sistema operacional seja o Windows, de modo que algumas bibliotecas específicas foram consideradas. Contudo, a adaptação do mesmo não deve ser de difícil implementação para outros sistemas operacionais.

Alguns problemas surgiram durante a implementação desse projeto, como a tentativa de utilizar arquivos de cabeçalho para a implementação das funções, o que não foi possível devido a limitações do compilador utilizado. Priorizando simplicidade, optou-se por implementar todas as funções em um único arquivo, o que não é uma boa prática de programação, mas foi a melhor solução para o problema.

2 Descrição das funções

Inserção/Remoção de texto s

Chamada	Input	Complexidade
`I`	s: String	$O(\|s\|)$
`D`	-	$O(1)$
`DL`	-	$O(1)$
`DW`	-	$O(\|\text{palavra}\|)$

A inserção de texto ocorre através da chamada no console do comando `I`s, onde s é uma *string* digitada pelo usuário. A inserção de cada carácter ocorre em tempo constante através da função `insert_char_a_direita`, que recebe um `char`, aloca uma nova célula com seu valor atribuído, e atualiza os ponteiros dos elementos adjacentes.

Listing 2: Exemplo de Inserção de funções

```
1 ------------------------
2 >0|hello
3        ^
4 ------------------------
5 (0,5): I world
6 ------------------------
7 >0|hello world
8              ^
9 ------------------------
```

Mudar estilo do cursor

Dado que os cursores de texto mais usuais são comumente indicados por `|` no meio do texto. Para emular esse comportamento e evitar confusão, ao pressionar `G`, a representação de $\hat{}$ passa a ser /\. Ao pressionar novamente, o cursor retorna ao ser estado anterior.

Listing 3: Mudança do estilo do cursor

```
------------------------
>0|hello world
             ^
------------------------
(0,11): G
------------------------
>0|hello world
             /\
------------------------
```

2.1 Escreve/Sobrescreve/Carrega arquivo

Chamada	Input	Complexidade
`E/A`	s: Nome de um arquivo	$O(\|\text{letras de } s\|)$

Realiza uma das opções acima num arquivo de texto s (`txt, tex, py, c,...`) na pasta em que o arquivo `main.c` se encontra.

Caso seja a função de abrir outro documento (`A`), isso ocasionará no questionamento do sistema se há desejo em salvar o trabalho anterior, caso haja, o arquivo será salvo. Após isso, o programa apagará o conteúdo de todas as linhas e o arquivo será carregado no lugar.

2.2 Movimentação direcional do cursor

Seja $(i, j) \in \mathbb{N}^2$ a posição atual do cursor em coordenadas, e $n_{\texttt{up}}, n_{\texttt{down}} \in \mathbb{N}$ os tamanhos das linhas de cima e de baixo respectivamente.

Chamada	Direção	Complexidade
`T`	Esquerda	$O(1)$
`F`	Direita	$O(1)$
`J`	Baixo	$\leqslant O(\min(j, \|n_{\texttt{down}} - j\|))$
`H`	Cima	$\leqslant O(\min(j, \|n_{\texttt{up}} - j\|))$
`0`	Início da linha	$O(1)$
`$`	Final da linha	$O(1)$

As funções de movimentação vertical tentam preservar a coluna (se existir) em que o cursor se encontra, pois as funções engatilhadas por `J` e `H` apontam inicialmente o cursor para o início da linha desejada, e assim, navegam na linha de modo a tentar encontrar a coluna mais próxima.

2.3 Movimentação do cursor para uma linha específica

Seja $(i, j) \in \mathbb{N}^2$ a posição atual do cursor em coordenadas e considere n sendo o número total de linhas.

Chamada	Descrição	Input	Complexidade
`:0`	Cursor para 1ª linha	-	$O(1)$
`:`x	Cursor para linha x	$x \in \mathbb{N}_{\neq 0}$	$O(\min(x, \lvert x - i\rvert, \lvert x - n\rvert))$
`:F`	Cursor para última linha	-	$O(1)$

Cada documento possui sempre três referências alocadas: A linha inicial, a linha final e a linha atual. A linha atual é atualizada sempre que o cursor é movido para uma nova linha, e a linha final é atualizada sempre que uma nova linha é inserida no documento. Dessa forma, fez-se a otimização de andar a menor quantidade de passos individuais, identificando qual referência das mencionadas está mais próxima da linha x.

2.4 Copiar/Colar/Recortar

Chamada	Descrição	Complexidade
`C`	Copia célula que o cursor aponta	$O(1)$
`V`	Insere valor da célula copiada na posição do cursor	$O(1)$
`X`	Análoga a `V` porém remove o valor da célula copiada anteriormente	$O(1)$

2.5 Busca/Substituição de padrões

Seja n o número de linhas total do documento e denote por ℓ_i a i-ésima linha do documento, de modo que o mesmo seja composto de $(\ell_0, \ell_1, \ldots, \ell_{n-1})$.

Chamada	Descrição	Input	Complexidade
`B`	Busca primeira aparição de s no documento	s: String	$O\Big(\sum_{i=0}^{n-1} \lvert\text{letras de } \ell_i\rvert\Big)$
`S`	Análoga a `B`, porém questiona o usuário se ele deseja substituir um dado padrão encontrado	s: String	$O\Big(\sum_{i=0}^{n-1} \lvert\text{letras de } \ell_i\rvert\Big)$

O algoritmo de busca implementado é o Knuth-Morris-Pratt (KMP), que procura ocorrências de uma palavra s dentro de uma cadeia de texto principal, empregando a observação de que, quando ocorre uma incompatibilidade,

a própria palavra incorpora informações suficientes para determinar onde a próxima correspondência pode começar, evitando algumas comparações das correspondências anteriores desnecessárias a partir daquele momento. Por conta desse mecanismo, o algoritmo KMP possui uma complexidade de tempo linear, ou seja, $O(N)$, sendo N o tamanho do texto principal.

No entanto, o algoritmo foi adaptado para que a busca seja feita em todo o documento, e não apenas em uma linha. Dessa forma, a complexidade do algoritmo de busca é $O\Big(\sum_{i=0}^{n-1} |\text{letras de } \ell_i|\Big)$. Além disso, a implementação do mesmo movimenta o cursor até a primeira instância da busca, caso ela seja encontrada. A partir de uma dada ocorrência, o programa pergunta ao usuário se ele deseja substituir o padrão encontrado e inicia sua busca a partir da próxima célula da linha atual caso o mesmo decida continuar buscando.

A implementação da substituição adiciona uma única condição na função anterior, questionando se o usuário deseja ou não substituir uma dada ocorrência.

Listing 4: Exemplo de Substituição

```
------------------------
 0|Larga a tia, largatixa!
 1|Lagartixa, larga a tia!
                         ^
------------------------
(2,23): Slargatixa
------------------------
 >0|Larga a tia, largatixa!
                  ^
 1|Lagartixa, larga a tia![s/n]
------------------------
(0,14): Deseja substituir? [s/n] s
Substituir por: lagartixa

------------------------
>0|Larga a tia, lagartixa!
                        ^
 1|Lagartixa, larga a tia!
------------------------
(0,22): Deseja continuar a busca? [s/n] n
```

2.6 Separar/Unir linhas

Chamada	Descrição	Complexidade
`N`	Separa linha atual na posição do cursor. Cria linha nova vazia se não tiver a conteúdo depois do cursor.	$O(1)$
`U`	Unir linha atual e a próxima	$O(1)$

Listing 5: Quebra de linha

```
------------------------
>0|hello world
         ^
------------------------
(0,6): N
------------------------
>0|hello
         ^
 1|world
------------------------
```

Listing 6: União de linha

```
------------------------
>0|hello
      ^
 1|world
------------------------
(0,3): U
------------------------
>0|hello world
      ^
------------------------
```