Universidade de São Paulo

Matheus Oliveira da Silva, Número USP: 13696262

Princípios de Desenvolvimento de Algoritmos

Editor de texto via comando na linguagem C

São Paulo – São Paulo
2022

Princípios de Desenvolvimento de Algoritmos

Editor de texto via comando na linguagem C

Relatório final, apresentado à Universidade de São Paulo, como parte da nota da matéria MAC0122 – Princípios de Desenvolvimento de Algoritmos.

São Paulo, vinte e dois de dezembro de 2022.

Prof. Denis Deratani Mauá

Introdução

Tem-se por objetivo, neste relatório, comentar e se aprofundar sobre o percurso de desenvolvimento do programa de editor de texto via comando na linguagem C.

O primeiro passo pensado e planejado para a execução do desenvolvimento foi a solução para o problema da abertura de arquivos, a estrutura de dados de texto que o editor iria trabalhar deveria ser compatível com a abertura de arquivos de texto. A partir desse pensamento decidi utilizar uma matriz para representar o texto sendo cada linha uma ‘string’ diferente e independente, vale a pena citar que durante o desenvolvimento percebi que tratar o texto inteiro como uma única ‘string’ facilitaria a implementação de diversas funções, porém, decidi manter na forma matricial pela objetividade na utilização dos índices e o tempo de execução de algumas funções ser dependente somente da linha.

Somente duas funções ficaram com complexidade de tempo quadráticas, ou seja, dependem do quadrado do tamanho do texto – a função de busca e a função substitui. Seria preferível a utilização do algoritmo de boyer-moore para essas duas funções, porém pela dificuldade de implementação optei pela implementação do algoritmo trivial visto em aula. O algoritmo de boyer-moore deixaria a complexidade das duas funções, substitui e busca, como O(n).

Foi utilizado somente um tipo de estrutura de dados abstrata, a pilha. A pilha foi implementada como uma única ‘string’, separando seus elementos por ‘\n’.

Funções

As funções estão listadas seguindo esta ordem abaixo.

Is	Insere a string s na posição atual do texto
An	Carrega o conteúdo do arquivo de texto de nome n no editor
En	(Sobre)escreve o conteúdo do editor no arquivo de texto de nome n
F	Move o cursor um caractere à frente
T	Move o cursor um caractere 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
:x	Move o cursor para o início da linha x
:F	Move o cursor para a última linha do arquivo
D	Apaga o caractere da posição atual
M	Marca (lembra) a posição atual do cursor
V	Desempilha e insere o conteúdo do topo pilha na posição atual
C	Empilha o texto entre a posição marcada e a posição atual (sem modificá-lo)
X	Empilha o texto entre a posição marcada e a posição atual e o deleta
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
J	Ir para próxima linha (manter a mesma coluna, se possível)
H	Ir para a linha anterior (manter a mesma coluna, se possível)
Z	Exibe a pilha de memória, começando pelo topo.

1. A função de inserção foi desenvolvida para copiar os caracteres que estão após o cursor em uma string de memória, em seguida inserir a string recebida no comando via teclado e finalizar inserindo a string de memória no final da linha. Um grande problema relacionado à maneira que escolhi fazer o texto, matricialmente, é que a linha possui um tamanho alocado de memória e portanto qualquer índice que esteja no tamanho da linha pode ser acessado e modificado, assim o índice texto[i] [j] com i sendo a linha e j o caractere da linha, com 0 <= j < tamanho da linha, suponhamos então que o caractere usado para simbolizar o fim do conteúdo da linha '\0' estivesse no índice texto[i] [j – 1], assim o índice texto[i] [j] ainda poderia ser acessado e modificado, porém na varredura da string as funções pararam no caractere '\0' inutilizando o caractere [j – 1]. Para resolver tal problema foi criada uma variável que carrega o índice do elemento '\0' e assim impede a inserção após esse elemento. A complexidade desse comando é dependente do tamanho da string a ser inserida e do tamanho da linha, sendo O(n).
2. A função para abrir arquivos de texto no editor foi uma das mais delicadas e complicadas de ser desenvolvida. É necessário que qualquer texto com qualquer tamanho seja aberto no editor, e, portanto, foi preciso fazer com que o

número de linhas no texto e o número de caracteres nas linhas fossem dinâmicos, e sempre que necessário o seu tamanho expandisse. Para isso foram criadas duas funções diferentes em que uma expande o tamanho da linha e outra expande o número de linhas do texto. Se desconsiderarmos a complexidade das funções auxiliares de expansão de texto e linha, a complexidade da função é O(n) onde n é o número de caracteres no arquivo.

- A função que expande a linha recebe uma variável com o tamanho da linha, cria uma nova linha alocando o dobro da capacidade da linha inicial e atribui todos os caracteres da linha inicial para a linha nova com o dobro de tamanho. A sua complexidade é de O(2n), sendo n o tamanho inicial da linha.
- A função que expande o texto recebe a quantidade inicial de linhas, cria uma nova matriz alocando o dobro da capacidade de linhas, porém ao invés de copiar todos os elementos de cada linha antiga na linha nova, a linha nova é liberada e o endereço da linha antiga é atribuído ao índice da linha nova. Assim, as linhas antigas são copiadas para o texto novo sem a necessidade de percorrer a linha inteira tornando essa função uma função com complexidade O(2n) onde n é o número inicial de linhas do texto.

3. A função escreve arquivo, diferentemente da função abre arquivo, é simples e somente navega pelo texto no editor e insere cada caractere e formata as linhas no arquivo a ser modificado. A sua complexidade de tempo é O(n) onde n é o número de caracteres no texto.
4. A função para avançar o cursor é trivial e somente aumenta em uma unidade a variável que carrega o índice do caractere. Tem complexidade de tempo constante.
5. A função para recuar o cursor também é trivial e diminui em uma unidade a variável que carrega o índice do caractere se e somente se ela for maior do que 0. Tem complexidade de tempo constante.
6. A função para levar o cursor ao início da linha simplesmente modifica a variável que carrega o índice do caractere para 0. Tem complexidade de tempo constante.
7. A função para levar o cursor ao início da próxima palavra navega na linha e assim que encontrar um espaço atribui o índice do próximo caractere à variável que carrega o índice do caractere. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
8. A função que leva o cursor ao início da palavra atual navega na linha em direção ao primeiro caractere e assim que encontrar um espaço, atribui o valor do índice depois do espaço à variável que carrega o índice do caractere. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
9. A função que leva o cursor ao fim da linha navega na linha até encontrar o símbolo ‘\0’ que indica o fim da string e atribui o índice à variável que carrega o índice do caractere. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
10. A função que leva o cursor até a última linha navega no texto até encontrar a linha que contém o caractere EOF, e atribui à

variável que carrega o índice da linha o valor da linha anterior. Tem complexidade de tempo O(n) onde n é o tamanho da linha.

11. A função que leva o cursor até o início da linha indicada via comando transforma os caracteres numéricos em valores inteiros e atribui o valor à variável que carrega o índice da linha. A minuciosidade nessa função foi transformar cada caractere em um valor inteiro correspondente à sua casa decimal, portanto se fez necessário ler o comando de trás para frente para assim atribuir os valores como unidade, dezena, centena e assim por diante. Tem complexidade de tempo O(n^2) onde n é o tamanho da linha.
12. A função que apaga o caractere no cursor é trivial, armazena os valores antes do cursor e depois do cursor em uma string de memória e depois atribui os valores da memória para a linha. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
13. A função para atribuir o valor do cursor ao marcador é trivial e somente passa o valor da variável que carrega o índice do caractere à variável que carrega o índice do caractere do marcador. Tem complexidade de tempo constante.
14. A função que desempilha foi feita armazenando os valores após o cursor na linha em uma string de memória, depois encontrando o índice do primeiro caractere que está na string que representa a pilha, cada elemento nessa string é separado por '\n', atribuindo o valor do último elemento da pilha na posição do cursor e em seguida inserindo o conteúdo da memória no final da linha. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
15. A função que empilha sem modificar a linha foi feita lendo os valores entre o marcador e o cursor e atribuindo-os ao topo da pilha. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
16. A função que empilha recortando os caracteres selecionados foi feita da mesma forma que a função 15, porém deixando apenas os caracteres que não foram selecionados pelo cursor e pelo marcador. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
17. O algoritmo de busca, como dito anteriormente, é trivial. Vale ressaltar novamente que o algoritmo de Boyer-Moore seria mais eficiente em questão de tempo. O algoritmo trivial foi feito comparando os valores da string a partir de seu último elemento, assim que todos os caracteres buscados fossem encontrados sequencialmente no texto, o índice do primeiro elemento e da linha em que se encontrava eram atribuídos às suas respectivas variáveis. Tem complexidade de tempo O(n^2) onde n é o tamanho da linha.
18. O algoritmo de substituição foi feito utilizando a lógica do algoritmo de busca, porém ele não se limitou à primeira ocorrência, mas a todas as ocorrências do texto. Assim que a palavra era encontrada ela era substituída por outra. A dificuldade encontrada foi alterar o tamanho da string quando havia mais de uma ocorrência da palavra na mesma linha. Tem complexidade de tempo O(n^2) onde n é o tamanho da linha.
19. O algoritmo de separar a linha verifica se há espaço na matriz para mais uma linha, caso não haja espaço ele aumenta o tamanho usando a função para expandir o

texto, se houver espaço ele reatribui o endereço de cada linha para a próxima linha, liberando assim uma linha vazia logo após a linha que queremos separar. Atribui os caracteres depois do cursor para a memória e os insere na linha livre, e em seguida retira os caracteres da linha atual. Tem complexidade de tempo O(n^2) onde n é o tamanho da linha.

20. O algoritmo que une as linhas faz o caminho contrário do algoritmo anterior, ele atribui os valores da próxima linha no final da linha atual e em seguida reatribui o endereço de cada linha para a linha anterior, liberando uma última linha no final do programa. Tem complexidade de tempo O(n) onde n é o tamanho da linha.
21. O comando que encerra o programa libera a memória do texto quebra o laço que verifica e chama as funções. Tem complexidade de tempo constante.
22. O comando para pular para a próxima linha é trivial e só aumenta o valor da variável que carrega o índice da linha, porém, caso o texto esteja em sua última linha ela expande o texto. Tem complexidade de tempo constante.
23. O algoritmo para recuar para a linha anterior somente diminui o valor da variável que carrega o índice da linha, porém, caso esse valor seja igual à 0, representando a primeira linha, o algoritmo não faz nada. Tem complexidade de tempo constante.
24. O algoritmo que exibe a pilha navega por ela usando a função strlen que verifica o tamanho da string e imprime ela, com cada elemento separado por ‘\n’. Tem complexidade de tempo O(n) onde n é o tamanho da pilha.
25. Criei um algoritmo extra que expande o texto somente. Com complexidade de tempo constante.

Conclusão

O editor de texto realiza todas as funções de forma eficiente, a maioria das complexidades de tempo são subquadráticas e o tamanho da linha e do texto é “infinito” garantindo que qualquer arquivo possa ser aberto e modificado, ou criado. Se tornando um editor de texto capaz de produzir um livro, caso seja a demanda.