Desenvolvimento do editor de texto

MAC0122 - Princípios de Desenvolvimento de Algoritmos

Marcos Vinicius Rodrigues Ribeiro - (12558954)

22 de dezembro de 2022

Instituto de Física
Universidade de São Paulo

Sumário

1 Estrutura de Dados utiliza

A estrutura de dados utilizada foi a lista encadeada com cabeça. Motivada pela ideia de que a leitura e manipulação do texto será, majoritariamente, sequencial, ou seja linha por linha, essa escolha foi fundamentada na ideia de manter todas as linhas conectadas de modo que a navegação entre elas fosse viabilizada de forma simples. Cada célula da lista encadeada armazena: um número inteiro que informa a posição da linha em relação as demais, um vetor de texto com capacidade definida de 1024, considerando que a produção de um arquivo de texto que será lido posteriormente, deve levar em conta um limite comum para todas as linhas, de modo que a leitura por alguém seja de maior qualidade se as linhas tiverem tamanhos iguais. E, o último elemento é um ponteiro para a próxima linha (célula).

A exibição da linha em que o cursor se encontra foi feita utilizando um ponteiro auxiliar que percorre a lista até a posição em que o usuário instruiu o programa a estar. Em outros momentos do algoritmo é necessário percorrer novamente a lista para garantir que o auxiliar esteja apontando para a linha que o cursor estiver marcando.

2 Inserção de String, comando Is

A função de inserir string's pressupõe, como foi elucidado no enunciado, que tudo que vier após ela será string, logo, foi utilizado um "for"para percorrer o que foi digitado pelo usuário e armazenar em um vetor de string alocado através de "malloc". O local onde string será salva é definido após duas verificações, uma para analisar se já há texto armazenado na lista, nesse caso é feita a chamada da função "InsereNaLinha"e a string é copiada elemento por elemento na posição em que o cursor se encontra. Nessa função foi utilizada o "for", que percorre a linha atual da posição em que está o cursor, copiando a nova string e o que vier após.

Analisando a complexidade de tempo desse processo, supondo tamanho n para nova string e m para o tamanho da linha, seria aproximadamente $O(n + m)$ no pior caso, em que a string está sendo inserida no início da linha e $O(n)$ no melhor caso, em que a string vai no final.

3 Carregamento e salvamento de arquivo ".txt"

3.1 Comando As.txt

A abertura de arquivos é feita extraindo a string que vem depois do comando "A"e abrindo o arquivo com um nome compatível com a string extraída no modo de leitura. E cada linha do texto é copiada para uma célula da lista encadeada junto do número da sua posição. Para isso é chamada a função "insereLinhaNaLista"que verifica se há dados iniciados na célula cabeça, caso não houver, adiciona a linha à primeira célula utilizando a função "strcpy()". Quando já houver uma linha copiada, a linha é inserida no final da lista encadeada,através da atualização do ponteiro que aponta para a cauda da lista.

Analisando a complexidade de tempo desse processo, uma nova linha é sempre inserida diretamente no final da lista encadeada graças ao ponteiro que aponta para a cauda da lista, logo o tempo de inserção da linha é constante, a única variável que interfere no tempo é o tamanho "n"da linha, logo, a complexidade é sempre $O(n)$.

3.2 Comando En.txt

Essa função foi implementada de forma simples, devido ao fato de que toda a informação digitada unicamente no editor é armazenada na primeira linha. Foi feito o processo de extrair a string depois do comando "E", e então é aberto um arquivo-texto no modo de escrita, e a linha é copiada utilizando a função "fputs()". A complexidade de tempo será dependente apenas do tamanho n da linha escrita, logo, $O(n)$.

4 Navegação do Cursor

O cursor foi implementado como uma string composta apenas de espaço em branco e do caractere ' ^ ' (acento circunflexo), onde são adicionado um números de espaços equivalentes a posição do cursor na linha (que está gravada em um inteiro) onde o usuário se encontra. É usado um "for"para produzir esses espaços, logo o pior caso é quando o cursor se encontrar no final da linha, a complexidade de tempo será $O(n)$ para gerar os espaços necessários, onde n é o tamanho da linha.

4.1 Comando F, T, O e $

O comando "F"soma um ao inteiro que armazena a posição do cursor se a posição atual não for maior que o tamanho da linha, o comando "T"subtrai um do inteiro se a posição atual for maior que zero. E o comando "O"iguala o inteiro que armazena a posição do cursor a zero. E o comando "$ "iguala o inteiro da posição do cursor com o tamanho da linha, levando o cursor pro final. Nesse último caso o ponteiro auxiliar é chamado para percorrer a lista até a linha que estava sendo exibida, adicionando uma complexidade de tempo m, para uma lista de m linhas (pior caso), com cunho de propiciar um algoritmo mais coeso e seguro, em comandos que é necessário acessar a linha em que o usuário se encontra, é utilizado esse ponteiro auxiliar igualando-o à cabeça e percorrendo até a linha correta, para garantir que o tamanho da linha medido seja o correto.

4.2 Comando P e Q

No comando "P", para buscar o próximo espaço foi implementado um "while"que percorre a linha atual a partir da posição do cursor e compara cada letra da linha com o char ' ' (espaço). Cada repetição incrementa a posição do cursor, o ciclo só se quebra quando o cursor chega no fim da linha ou o espaço é encontrado. O comando Q foi implementado de forma similar, executa a

mesma função que o comando P mas no sentido oposto, em direção ao início a linha e reduzindo a posição do cursor.

A complexidade de tempo dessa funções é imediata, depende apenas da posição do próximo espaço. Sendo seu pior caso quando o espaço estiver no final de uma linha de tamanho n e o cursor na posição zero, no caso do comando "P". E o pior caso do comando "Q"é o espaço no início da linha e o cursor no final.

4.3 Comando :x e F

Para o comando ":x", a entrada do usuário é armazenada, e dela é retirado o número em formato de 'char', que para convertê-lo em inteiro foi subtraído do caractere o char '0', como mostrado em aula. Em seguida a posição no texto é atualizada com o inteiro obtido. E para o comando F foi só adicionada uma condição em que caso o caractere da entrada seja um 'F', a posição do texto irá para a última célula, que o ponteiro cauda está endereçado.

4.4 Comando D

Novamente, utilizou-se o vetor auxiliar para garantir que a linha sendo manipulada seja a que foi exibida ao usuário. Utilizando o inteiro que armazena a posição do cursor, o caractere escolhido foi apagado movimentando todos os caracteres da string para a esquerda através de um "for". A complexidade de tempo nesse caso dependerá da posição em que o caractere se encontrava, sendo seu pior caso quando o caractere está no começo, com complexidade $O(n)$, onde n é o tamanho da linha. O melhor caso é quando o caractere se encontra no final, onde a complexidade de tempo é constante.

5 Empilhamento e Desempilhamento de texto

Para executar as funções envolvendo empilhamento foi necessário munir o algoritmo de uma nova struct chamada "No", que exerce a mesma função que a primeira struct citada anteriormente mas que armazena dados diferentes. É uma estrutura de dados implementada como lista encadeada onde os valores no topo são acessados primeiro e a struct de cada nó dessa pilha possui dois elementos: uma string que armazena um fragmento do texto e um ponteiro para o próximo nó da pilha. Como a própria funcionalidade clama, a melhor estrutura de dados para esses casos é uma pilha, pois quando os fragmentos são armazenados, eles são extraídos na ordem inversa que foram empilhados. Mas não foi possível implementar os comandos relacionados a pilha de modo que fragmentos de linhas distintas possam ser armazenados como fragmentos, apenas de mesma linha.

5.1 Comando M e C

O comando "M"marca posição atual do cursor, essa funcionalidade foi implementada em conjunto com o cursor, visto que onde não está o cursor são apenas espaços. Quando o usuário executa esse comando, dois inteiros armazenam a posição na linha e em qual linha foi executado e um caractere 'M' é posicionado no vetor de string onde está o cursor (na posição marcada). Ele inicializa no ¡0,0¿.

5.2 Comando V, X e Z

Essa funcionalidade foi implementada utilizando a função "strcpy", onde uma string auxiliar recebe o conteúdo recebido pela "desempilha", essa função retorna o Nó no topo da pilha e o fragmento armazenado. E após o fragmento ser copiado é feita uma chamada da função "InsereNaLinha", (mesma utilizada no comando Is) e o fragmento é copiado na posição do cursor.

A complexidade de tempo nesse caso depende de dois fatores, o tamanho do fragmento e o tamanho da linha, onde cada caractere será deslocado uma vez. Logo, a complexidade de tempo será $O(n + m)$, onde n é o tamanho do fragmento e m é o tamanho da linha.

O comando X foi implementado da mesma forma que o V, contudo, foi adicionado um "for"para recuar os caracteres depois da posição do cursor até a posição marcada, fator que adiciona uma complexidade $O(m)$ em sua execução.

O comando Z foi implementado com uma função "ImprimePilha", que através de um ponteiro recebido, acessa o topo da pilha e devolve o fragmento de texto armazenado nesse nó. Dentro da função, o fragmento é imprimido e o contador passa para o próximo nó. Esse comando depende do tamanho da pilha, tendo complexidade $O(m)$ sendo m o número de nós na pilha.

6 Busca de ocorrência - Comando Bs

Essa funcionalidade foi implementada como a Busca Trivial apresentada em sala de aula. Primeiro é feita a chamada do vetor auxiliar para certificação da posição no texto. Para comandos como esse, tudo que vier após o caractere que 'B' é considerado a string que será buscada, logo, o primeiro processo consiste em armazenar essa string através de um "while"que percorre a entrada do usuário, depois do B. Depois de armazenada, é chama a função "OndeOcorre"que retorna a posição de onde a ocorrência acaba, ou -1 caso não tenha localizado o padrão da armazenado na linha atual. Para verificar se o padrão foi encontrado, foi posicionado um "if"para verificar se o valor que a função retorno é mais que zero, caso for, define a posição do cursor como o valor retornado pela função, caso não for, roda um "while"buscando nas linhas sucessoras. Depois do "while"foi posicionado outro "if"para verificar o valor de retorno da função, caso for mais que zero, atualiza a posição do cursor no texto e na linha para a posição de onde acaba a ocorrência.

A complexidade de tempo nesse caso, depende do tamanho do padrão buscado e do tamanho do

texto, sendo o produto das duas no pior caso, $O(mn)$ sendo m o tamanho do texto e n o tamanho do padrão buscado. Foi considerado razoável utilizar esse método de busca no editor de texto, tendo em mente que o usuário possivelmente buscará palavras muito menores que o texto em geral. Logo, o desenvolvimento de um preprocessamento para utilização de métodos como Sufixo Bom ou Caractere Ruim não pareceram tão frutíferos nesse caso. E também por motivos de limitação de tempo, o algoritmo trivial foi o mais palpável para implementação.

7 Substiuição de string - Comando Ss/r

Essa funcionalidade, com o mesmo príncipio de tratar o que vem após o caractere de comando como string de interesse, armazena tudo que vem depois do caractere 'S' em uma string, que é dividida utilizando a função "strtok". E após separar a entrada em: padrão e substituto, o algoritmo entra em um "while"que percorre todo o texto a partir da posição atual e chama a função "substituir", que foi desenvolvida em um dos EP's entregues, para todas as linhas. Cada linha alterada é armazenada numa string auxiliar, que após passar pela função de susbtituição, é copiada para linha através do "strcpy". O ciclo se repete até a última linha.

No caso desse comando, a complexidade de tempo provavelmente é uma das maiores, pois percorre todas as linhas do texto, ou seja, tem uma dependência com o tamanho do texto e com o tamanho das linhas, já que todas as linhas são reescritas com o "strcpy"mesmo se não houver ocorrências, logo, há uma complexidade próxima de $O(nm)$ no caso médio, sendo n o número de linhas e m o tamanho das linhas. No pior caso em que há ocorrência em todas elas, a complexidade salta para $O(n^2 * m)$, já que todas as linhas serão reescritas mais de uma vez.

E nesse caso, também há uma complexidade de espaço, visto que a implementação da função "substituir"foi feita de forma recursiva, de modo que o espaço consumido depende do número de ocorrências de uma palavra numa mesma linha. Sendo no pior caso, onde há uma linha só de ocorrências do padrão, a complexidade de espaço ser $O(m)$, onde m é o tamanho da linha.

8 Manipulação e Navegação nas linhas

8.1 Comando N

Esse comando foi implementado também com o uso do vetor auxiliar para acessar a linha escolhida pelo usuário. Dois vetores de strings foram alocados com tamanho máximo, onde cada um recebe uma metade da linha, tendo o ponto de cissão definido pela posição do cursor. Em ambos os casos foi adicionado um caractere ' 0' ao fim da string, pois houve recorrência de lixo de memória quando as metades da linha eram copiadas. Após a linha onde está o cursor receber a primeira metade através do "strcpy", é alocado uma nova célula que recebe a segunda metade e integra a lista encadeada qeu está armazenando o texto. Depois desse processo, é feita um ajuste nos indíces das linhas (inteiro que armazena a posição).

8.2 Comando U

Nesse comando foi utilizado o vetor auxiliar novamente, para garantir com asserção a linha que será alterada. E para unir as duas linhas foi usada a função "strcpy"para armazenar umas das linhas em um vetor com memória alocada pelo "malloc"(tamanho medido com "strlen") e "strcat"para concatenar a string alocada com uma das linhas, nesse comando o racíocinio utilizado foi de que a linha onde o cursor está sempre recebe a sua sucessora. A complexidade geral desse processo será $O(a + b)$, sendo a e b o tamanho das duas linhas, pois as três funções utilizadas no comando tem complexidade $O(a + b)$.

8.3 Comando J e H

Em ambos comandos o vetor auxiliar foi chamado, percorrendo a lista até chegar na posição solicitada pelo usuário, que é definida pela posição atual acrescida ou subtraída por 1. No caso do comando "J", o método de só igualar o vetor auxiliar ao ponteiro que aponta para a próxima célula não foi usada pela mesma razão citada anteriormente, da possibilidade de não acessar a linha correta. Então, sacrificando um pouco de tempo por estabilidade, a lista é percorrida novamente (formas de não ter que percorres a lista novamente foram testadas, mas com resultados não satisfatórios. Logo a complexidade desse comando será $O(m)$, onde m é o número de linhas do texto, com queda de perfomance conforme a linha requisitada for mais próxima do fim da lista.