INSTITUTO FEDERAL DE SÃO PAULO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CRISTIAN TREVELIN ROEFERO

TRABALHO FINAL DE ESTRUTURAS DE DADOS 2:

Detalhamento e Justificativa da Estrutura Utilizada

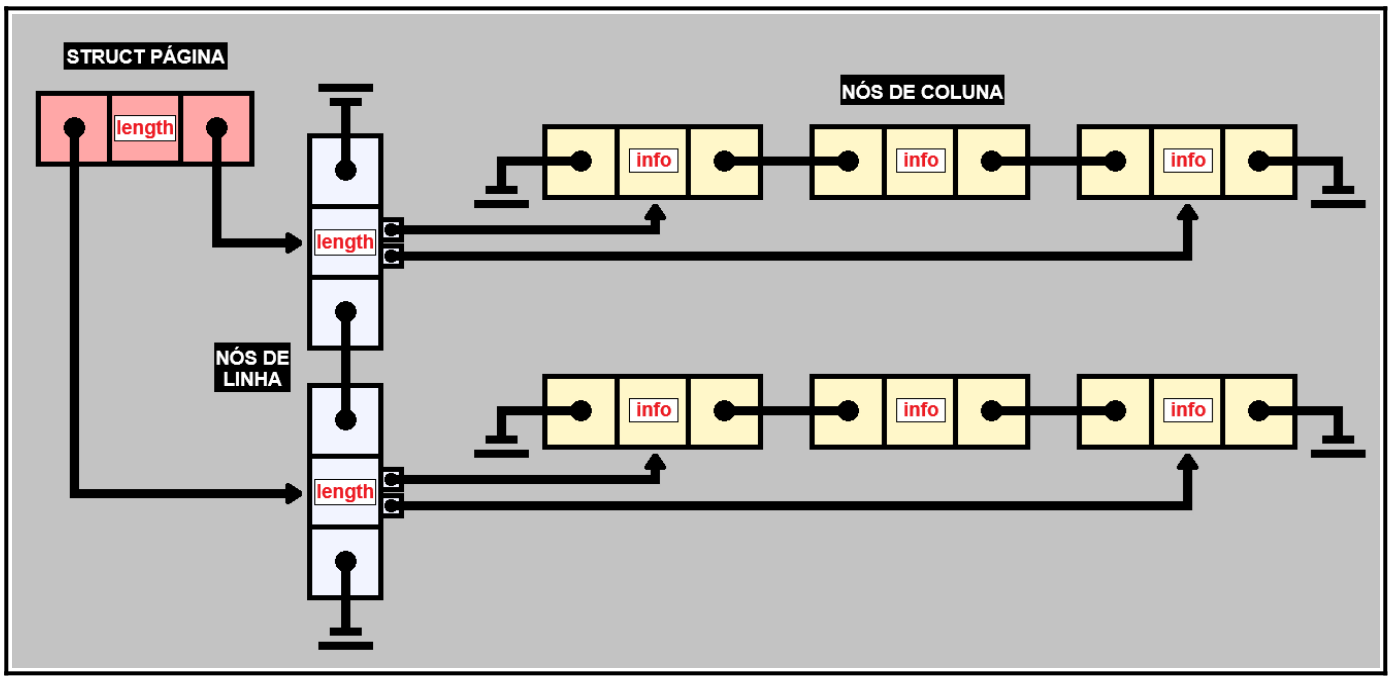
PRESIDENTE EPITÁCIO – SP
2023

1 DETALHAMENTO

O presente projeto foi realizado para atender às requisições do trabalho final da disciplina de Estruturas de Dados 2, do curso de Bacharelado em Ciência da Computação – IFSP, campus Presidente Epitácio. O objetivo é desenvolver uma aplicação de edição de texto plano em linguagem C, baseada em DOS, que seja capaz de simular algumas das funcionalidades mais populares desse tipo de aplicação, como navegar no texto pelas setas do teclado e salvar o conteúdo escrito em um arquivo.

Para alcançar o objetivo, foi necessária a escolha das estruturas de dados ideais, que pudessem armazenar o texto plano em linhas e colunas e atender às necessidades do problema. Três estruturas foram utilizadas, sendo duas variações de lista duplamente encadeada e uma *struct*, dispostas no diagrama a seguir:

Figura 1 – Diagrama de Estruturas



Fonte: Próprio Autor.

Cada coluna será representada por um nó da *struct sColumnNode*, que contém um caractere e ponteiros para o nó seguinte e anterior. Uma linha se trata de uma lista de colunas, portanto, a *struct sRowNode* terá ponteiros para a primeira e última colunas, além do comprimento atual da linha (quantidade de colunas) e

ponteiros para a próxima linha e para a anterior. Por fim, uma página pode ser definida como uma lista de linhas. A *struct sPage* contém ponteiros para a primeira e última linhas, como também o comprimento da página (quantidade de linhas).

Abaixo estão descritas as devidas estruturas, em linguagem C:

```
typedef struct sColumnNode
{
	char info;
	struct sColumnNode *next;
	struct sColumnNode *previous;

} COLUMN_NODE;

typedef struct sRowNode
{
	COLUMN_NODE *first_column;
	COLUMN_NODE *last_column;
	unsigned int length;
	struct sRowNode *next;
	struct sRowNode *previous;

} ROW_NODE;

typedef struct sPage
{
	ROW_NODE *first_row;
	ROW_NODE *last_row;
	unsigned int length;

} PAGE;
```

2 JUSTIFICATIVA

A princípio, para representar uma página de texto plano que pudessem se expandir à vontade do usuário, em linguagem C, duas abordagens foram consideradas: o uso de vetores dinâmicos e o uso de listas encadeadas. Outras estruturas como pilhas e filas foram descartadas, devido à necessidade de acessar frequentemente, elementos que estivessem em posições arbitrárias da estrutura. Entre as abordagens consideradas, uma análise comparativa foi feita, levando em conta alguns pontos.

Tendo em vista uma aplicação de edição de texto plano, é provável que as operações mais realizadas pelo sistema sejam de inserção e remoção de elementos, que ocorrem quando o usuário escreve e apaga o texto escrito, respectivamente. Neste ponto, listas encadeadas têm menor *overhead* em relação aos vetores dinâmicos, já que em um vetor, há a necessidade de deslocar elementos na memória, sempre que o elemento inserido ou removido não está na última posição.

Outro ponto considerado foi o fato da página possuir comprimento imprevisível e variável, de linhas e colunas, a cada uso da aplicação. Em vetores dinâmicos, haveria a necessidade de expansão, sempre que o texto escrito atingisse o limite da estrutura (da linha ou coluna). Para expandir o vetor, um novo e maior vetor deveria ser instanciado, todos os elementos transferidos para ele e todos os blocos do vetor antigo, desalocados da memória. Esse processo implicaria em um alto *overhead* – principalmente se tivesse que ser feito constantemente. As listas encadeadas, por outro lado, são naturalmente expansíveis, utilizam somente a memória necessária e possuem apenas o *overhead* da alocação de memória dinâmica, que não é tão alto para as colunas, já que os elementos em questão são caracteres (1 byte).

O ponto fraco das listas encadeadas estaria nas operações de busca, quando elementos distantes do primeiro nó tivessem de ser acessados – por exemplo, caso o usuário desejasse escrever ou apagar o texto do meio ou do final da página. Cada busca na lista encadeada requer uma varredura até que o nó chave seja encontrado, ao contrário dos vetores, cuja mesma operação tem complexidade de tempo de O(1).

Para mitigar o problema das listas, os endereços de memória dos últimos nós de linha e coluna serão armazenados, junto à quantidade de nós atual (de linhas e

colunas). Com essa estratégia, será possível varrer as listas com mais agilidade, de acordo com a posição do elemento buscado na tela. Por exemplo, se uma determinada linha possui 100 caracteres armazenados e o usuário posiciona o cursor no 70º caractere da tela, a varredura na lista de colunas será feita a partir do último nó, retrocedendo até o nó chave ser alcançado.

Por fim, o agrupamento das estruturas foi definido para simular uma página de texto, facilitando na manipulação dos elementos: colunas representam um único caractere e cada linha é uma lista de colunas, guardando as informações acerca delas (quantidade total de colunas e endereços do primeiro e último nós). A página, por sua vez, é uma lista de linhas e também armazena suas informações, sendo elas o total de linhas e endereços da primeira e última linha. Com essa estrutura, é possível mapear o *buffer* da tela do console, de modo com que toda ação realizada pelo usuário no console seja refletida na página e devidamente tratada.