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

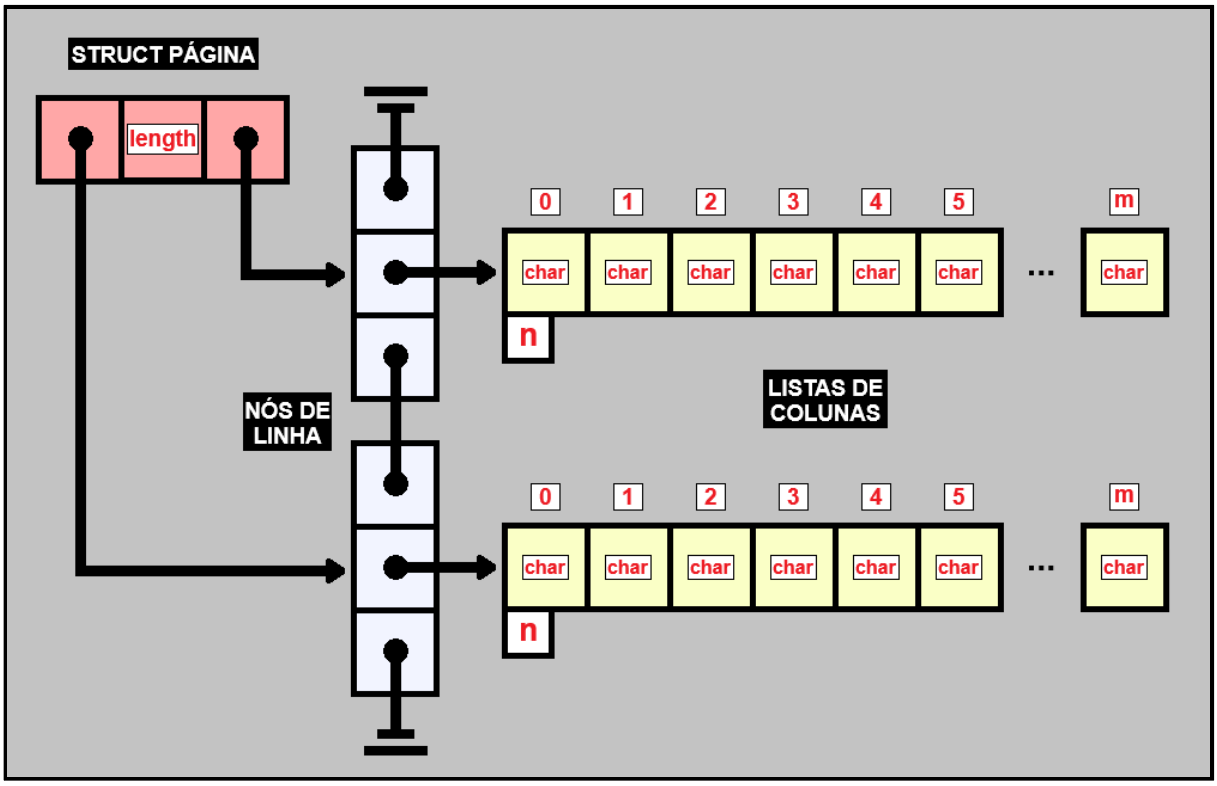
PRESIDENTE EPITÁCIO – SP
2023

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 uma lista duplamente encadeada, uma lista estática adaptada e uma *struct*, dispostas no diagrama a seguir:

Figura 1 – Diagrama de Estruturas



Fonte: Próprio Autor.

Cada linha se trata de um nó de uma lista duplamente encadeada – *ROW_NODE* – cujo *buffer* que armazena os caracteres é representado por uma lista

estática adaptada – *ROW_BUFFER* – de tamanho *m*, em que *m* é definido como a largura da janela do console, medida em caracteres. Por último, a página é responsável por encapsular a lista de linhas. A *struct sPage* contém ponteiros para o primeiro e último nós de linha da lista, como também o comprimento da página (quantidade atual de linhas).

Abaixo estão descritas as devidas estruturas, em linguagem C:

```
typedef struct sRowBuffer
{
	char *buffer;
	int max_length;		// definido na inicialização (m)
	int n;

} ROW_BUFFER;

typedef struct sRowNode
{
	ROW_BUFFER *row_buffer;
	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 em linguagem C, duas abordagens foram consideradas: o uso de vetores 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, cuja interface gráfica é baseada em DOS, o primeiro foco foi acerca das dimensões da janela do console exibida. Para cumprir com o comportamento esperado desse tipo de aplicação, o usuário deve ser capaz de redigir textos expansíveis, sem comprometer sua legibilidade. Em termos de implementação, as linhas da página devem ser limitadas à largura da janela, enquanto a página em si deve ser dinamicamente expansível em altura, tendo a memória como o seu único limite.

Por isso, listas estáticas seriam mais convenientes para abstrair as linhas, ao tempo que uma estrutura dinâmica supriria as necessidades da página expansível.

A preferência pela lista encadeada para representar a página deu-se pela análise de outro ponto. Em um vetor dinâmico, haveria a necessidade de expansão, sempre que o número de linhas atingisse o seu limite. Para expandir o vetor, um novo e maior vetor deveria ser instanciado, todas as linhas transferidas a 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. A lista encadeada, por outro lado, é naturalmente expansível, utiliza somente a memória necessária e possui apenas o *overhead* da alocação de memória dinâmica para o novo item.

O ponto fraco da lista encadeada, no entanto, estaria nas operações de busca, quando um nó de linha distante do primeiro nó tivesse de ser acessado – 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 da lista, o endereço de memória do último nó de linha será armazenado, junto à quantidade de linhas atual. Com essa estratégia, será

possível varrer a lista com mais agilidade, a partir de ambas as extremidades, de acordo com a posição da linha acessada na tela. Por exemplo, se a página em um determinado momento possui 100 linhas redigidas e o usuário posiciona o cursor na 70º linha da tela, a varredura na lista será feita a partir do último nó, retrocedendo até o nó chave ser alcançado. Ademais, seria conveniente poder percorrer as linhas para cima e para baixo, a partir do posicionamento do cursor. Esse detalhe consolidou a escolha da lista duplamente encadeada.

Como resultado da análise, obteve-se um agrupamento de estruturas que abstrai uma página de texto, encapsulando seus elementos internos: a página se trata de uma lista duplamente encadeada de linhas, contendo informações relevantes para a manipulação delas, como a sua quantidade atual e os endereços de memória da primeira e última linhas. Cada linha, por sua vez, é um nó da lista e possui um *buffer* de caracteres representado por uma lista estática, cujo tamanho físico é definido no início da aplicação e permanece estático até o seu encerramento.

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.