Estrutura de dados I
Kelvin Lazzaris

Neste relatório eu irei descrever as metodologias usadas na implementação de uma lista encadeada dupla em C. O objetivo do projeto foi de criar uma lista encadeada capaz de armazenar nomes e implementar operações básicas, que seriam a inserção, remoção e busca.

Inicialmente, eu defini o tamanho 53 da tabela Hash conforme solicitado, depois foi necessário definir a estrutura de dados que representaria os nós da lista encadeada, eu optei por criar uma struct chamada Nodo que possui três campos: nome, proximo e anterior, o campo nome é responsável por armazenar o nome a ser inserido na lista, enquanto os campos proximo e anterior são ponteiros para os nós a serem trabalhados na lista.

Em seguida, foi definida a struct Lista que representa a própria lista encadeada,essa struct possui os campos inicio, fim e tamanho.

***inicio* =** é um ponteiro para o primeiro nó da lista;
***fim* =** é um ponteiro para o último nó da lista;
***tamanho* =** armazena o número de elementos na lista;

Essa struct é responsável por manter as informações sobre a lista e auxiliar nas operações de manipulação, segue a ilustração das struct abaixo:

```
 9  typedef struct nodo {
10      struct nodo *proximo;
11      struct nodo *anterior;
12      char nome[50];
13  } Nodo;
14
15  typedef struct sLista {
16      struct nodo *inicio;
17      struct nodo *final;
18      int tam;
19  } Lista;
```

Uma parte importante a ser mencionada é como foi feita a inserção de nomes, a inserção de nomes no código é feita através da função adicionar, que recebe como parâmetros a tabela hash ‘t[]’ e o nome a ser inserido ‘nome[]’.

Essa função é chamada dentro do bloco while no main, após a leitura de cada linha do arquivo "nomes.txt", como podemos ver na ilustração abaixo:

```c
while (fgets(linha, sizeof(linha), arquivo) != NULL) {
    linha[strcspn(linha, "\n")] = '\0';
    adicionar(tabela, linha);
}
```

Logo após, comecei a trabalhar nas operações básicas, que como citado anteriormente, foi busca, inserção e remoção, onde foram implementados os conceitos de ponteiros e alocação dinâmica de memória. Para inserir um novo nome na lista, um novo nó é criado e os ponteiros são ajustados corretamente. A remoção de um nó envolve a atualização dos ponteiros dos nós adjacentes. A busca por um nome percorre a lista, comparando o nome buscado com o nome em cada nó.

Depois foi feita a parte de validação e testes, após a implementação das operações, foram realizados testes para verificar a correta funcionalidade da lista encadeada. Foram criados cenários de teste que abrangiam diferentes casos, como inserção no início, no fim e no meio da lista, remoção de nós, busca por nomes existentes e não existentes, entre outros. Um exemplo de validação de testes é chamar a função “inserirNomeLista” com diferentes nomes e verificar se os nomes são inseridos corretamente na lista vazia, verificar também se os ponteiros inicio e final são ajustados corretamente e se o tamanho da lista é atualizado corretamente.

```c
void inserirNomeLista(Lista *lista, char nome[]) {
    Nodo *nodo = malloc(sizeof(Nodo));

    if (nodo) {
        strcpy(nodo->nome, nome);
        nodo->proximo = NULL;
        nodo->anterior = NULL;

        if (lista->inicio == NULL) {
            lista->inicio = nodo;
            lista->final = nodo;
        } else {
            nodo->proximo = lista->inicio;
            lista->inicio->anterior = nodo;
            lista->inicio = nodo;
        }
        lista->tam++;
```

Sobre a função Hash, foi usado a fórmula repassada em sala de aula pelo próprio professor, que consiste em: índice (hash) da string (nome) com base em uma tabela de tamanho fixo. O objetivo é distribuir os nomes de forma uniforme pela tabela para melhorar o desempenho das operações de inserção, busca e remoção.

```
h = (31 * h + nome[i]) % TAM;
```

Também está presente no código a função do histograma, que é responsável por exibir graficamente a quantidade de elementos em cada lista da tabela hash.Esse histograma funciona da seguinte maneira:

1 - Encontra o maior número de elementos em uma lista da tabela hash. É necessário para definir a altura máxima do histograma.

Ilustração do passo 1;

```
for (i = 0; i < TAM; i++) {
    if (t[i].tam > maxCount) {
        maxCount = t[i].tam;
    }
}
```

2 - Cria o histograma de cima para baixo, mostrando a quantidade de elementos em cada lista em intervalos de 10.

Y = quantidade de elementos;

X = posições da tabela hash;

Ilustração do passo 2;

```
for (i = maxCount; i > 0; i -= 10) {
    printf("%2d   ", i);

    for (j = 0; j < setNum; j++) {
        if (t[j].tam >= i) {
            printf(".");
        } else {
            printf(" ");
        }
    }
    printf("\n");
}
```

3 - O valor de ‘setNum’ é usado para definir a quantidade de posições mostradas no eixo X (por padrão, são mostradas 10 posições).

4 - Para cada intervalo de 10 elementos no eixo Y, a função percorre cada lista da tabela e verifica se a quantidade de elementos é maior ou igual ao valor do intervalo. Se for, um ponto é exibido no histograma para indicar que aquela lista contém elementos naquele intervalo.

Conclusão: As principais funções implementadas são responsáveis por inicializar a lista e a tabela, inserir e remover elementos, buscar elementos na tabela, imprimir a listagem dos elementos, ordenar as listas, exibir um histograma com a quantidade de elementos em cada lista, exportar a tabela para um arquivo CSV e percorrer as letras iniciais dos nomes. No geral, a metodologia utilizada permite a organização eficiente dos nomes em listas, facilitando as operações de inserção, remoção, busca e ordenação. Além disso, o uso da tabela de hash distribui os elementos de forma equilibrada, evitando colisões e proporcionando um acesso rápido aos elementos.