# TRABALHO PRÁTICO 2

**André Costa Werneck**
**Matrícula : 2017088140**
**AEDS II - TP2**
**Turma TN2**

## INTRODUÇÃO

A finalidade desse trabalho é desenvolver um sistema de auto completação de teclado. Além disso, é esperado o desenvolvimento da prática de modelagem em tipos abstratos de dados (TADs) e da prática de programação, em particular para algoritmos de busca e ordenação.

## DESCRIÇÃO

Dado um dicionário de palavras, o sistema deverá criar uma tabela hash para indexar todas as palavras do dicionário de acordo com seus prefixos. Após a indexação de todas as palavras, o sistema deverá atender a requisições do usuário. Quando o usuário digitar uma letra, o programa deverá procurar na tabela hash por palavras que contenham aquela letra inicial e exibi-las na tela. Quando o usuário digitar a segunda letra, o programa deverá pesquisar na tabela por palavras que iniciem com as duas letras passadas. A tabela hash deverá ser criada para até n letras iniciais. O programa deverá mostrar todas as sugestões de palavras que contenham o prefixo passado em ordem alfabética. Caso não existam sugestões, o sistema deverá informar isso ao usuário através de uma mensagem.

## EXECUÇÃO

Fiz um TAD denominado, *hash* com as seguintes funções :

```
typedef struct hash_t* hash;
typedef char* string;

hash criaTabela(int m); // criar a tabela hash
```

- cria a tabela hash e coloca todas as posições como vazias (NULL)

```
unsigned int hashing (string n,unsigned int m);
```

- funcao de hash - indexa a string na tabela hash

```
int numeriza(char t);
```

- converte um char num inteiro

unsigned int buscanatab(string w,hash t, int tam);
- busca na tabela pela string indexada e retorna o índice da string, caso ela seja encontrada, ou uma mensagem caso ela não seja encontrada na tabela.

void insere(hash table, string w, int tam);
- insere os elementos na tabela hash e , caso ocorra colisão, realizada o tratamento por encadeamento.

void imprime(hash table, string w, int tam);
- imprime os elementos da tabela e da lista relacionada a posição acessada.

Além dele, criei um TAD *lista*, para inserir elementos na tabela utilizando do tratamento de colisões por encadeamento.

```
typedef struct ListNode{
   string word;
   struct ListNode *next;
} ListNode;
// definição do meu tad list

typedef struct ListNodeH{
   struct ListNode *first;
   struct ListNode *last;
} ListSentinel;
// definição do sentila, indicando a última posição da lista.

ListSentinel* createList();
// cria a lista relacionada a respectiva posição na tabela

void addNodeList(ListSentinel *head, string word);
// adiciona um no na lista

void printList(ListSentinel *head);
// imprime um nó na lista

void clearlist(ListSentinel *head);
// apaga a lista encadeada.
```

Com ambos os TAD's sendo usados em conjunto, foi possível desenvolver o trabalho indexando as palavras e distribuindo-as na tabela, de acordo com a função

de hash. Além disso, caso uma palavra recebesse o índice para uma posição já ocupada, o que significa que ocorreu uma colisão, era criada ou aproveitada uma lista encadeada para aquela posição. Desse modo, foi possível realizar o tratamento das colisões e a estratégia escolhida foi o encadeamento.

*Entretanto, por falta de tempo,devido a um final de semestre muito apertado, não consegui terminar o trabalho, faltou a parte de ordenar para a impressão, constando, portanto, no meu trabalho, apenas as strings desordenadas. Fiz o que foi possível com meu conhecimento e com o tempo que tive.*

## PERGUNTAS

### 1. Qual o custo de montar a tabela hash?

A função criaTabela segue abaixo :

```
hash criaTabela(int m){
   int i;
   hash tabela = malloc (pow(26,m) * sizeof(struct hash_t));
   for(i = 0; i < pow(26,m) ; i++){
      tabela[i].lista = NULL;
   }
   return tabela;
}
```

E, seu custo é exponencial, uma vez que cria-se a tabela de tamanho 26^m , o que significa uma baixa taxa de otimização do meu projeto.

### 2. Qual o custo de buscar uma palavra?

O custo de buscar uma palavra também é, no pior caso, exponencial, uma vez que busca na tabela de tamanho 26^m.

### 3. Qual o custo para ordenar as sugestões?

Teria usado o selection sort, com custo fixo de $O(n^2)$ comparações para a ordenação e com O(n) movimentações.

## 4. Qual o custo de inserir uma nova palavra?

```
void insere(hash t,string w,int tam){
   unsigned int indice = hashing(w,tam);

   if (t[indice].lista == NULL){

    t[indice].lista = createList();

    addNodeList(t[indice].lista, w);

  }
   else{
      addNodeList(t[indice].lista, w);
   }
}
```

O custo seria, no pior caso O(n), uma vez que foi usado o tratamento de colisões por encadeamento.

## 5. Qual o custo de remover uma palavra?

O custo seria, no pior caso O(n), uma vez que foi usado o tratamento de colisões por encadeamento.

## 6. Qual o custo total do seu código?

O custo total do meu código, que está incompleto e não otimizado é o custo exponencial, uma vez que para criar e buscar na tabela, as funções criadas por mim possuem o custo em O(26^m).

## 7. Qual função hash você escolheu?

Escolhi usar o hashing linear , no qual a indexação é feita através da operação :
hashing = (N mod M), onde N é o tamanho da string e M é o tamanho da tabela.

## 8. Como seu sistema se comporta no caso de haver colisões?

Meu sistema se comporta jogando as strings em listas encadeadas relacionadas à cada posição da tabela. Obviamente, só é criada uma lista para posições a serem ocupadas. Ou seja, um índice que não recebe nenhuma string também não possui nenhuma lista associada.