Universidade Federal de Ouro Preto – UFOP
Departamento de Computação e Sistemas – DECSI
Disciplina: Algoritmos e Estrutura de Dados I – CSI 488
Professor: Bruno Hott (brhott@ufop.edu.br)

Valor: 1,0 ponto (10% da nota total) Data de entrega: 29/05/2025
Documentação em LaTeX: +0,2 ponto
Impressão frente-verso: +0,1 ponto (ecologicamente correto)

# Trabalho Prático 1: Caça Palavras

## Objetivos

O objetivo desse trabalho é rever conceitos básicos de programação bem como explorar os conceitos de tipos abstratos de dados (TADs), alocação dinâmica de memória e análise de complexidade.

## Descrição

Um passatempo bastante comum são os "caça-palavras". Neste tipo de problema existem uma grade repleta de letras distribuída de maneira aparentemente sem sentido e uma lista de palavras sobre algum tema. Por exemplo:

| | | | | | |
|---|---|---|---|---|---|
| s | c | c | e | l | r |
| u | e | h | a | y | r |
| g | e | a | i | m | c |
| c | a | v | r | e | a |
| c | x | e | p | l | a |

- chave
- mel
- erva
- cama
- veu

Resolver o problema consiste em encontrar as palavras indicadas na lista. Neste trabalho você deve projetar um programa que receba como entrada um problema do tipo "caça-palavras" e deve retornar as coordenadas de início e término de cada palavra fornecida na lista de palavras.

A solução implementada por você deverá utilizar de uma estrutura conhecida como *Matriz de Ocorrências*. A ideia consiste em tentar otimizar a busca de palavras armazenando as ocorrências de cada letra da matriz de entrada em uma tabela de ocorrências. Na tabela podemos registrar não apenas quantas vezes cada letra ocorre na matriz mas também as coordenadas de todas estas ocorrências. Dessa forma, o programa não precisa percorrer toda a matriz de letras sempre que for procurar por uma palavra nova.

Valerão pontos: a correta escolha da estrutura de dados, o bom uso de funções e procedimentos, a modularidade, a possibilidade de generalização do problema, por exemplo, se você tiver um dia que modificar o seu programa para um caça-palavras tridimensional, seu programa deve ser facilmente adaptável. Note que as palavras estão em qualquer direção, horizontal, vertical, nas diagonais e que podem ser escritas normalmente ou de maneira inversa. Preocupe-se inicialmente em detalhar a estrutura do programa, isto já valerá pontos. Mas observe que valerá a totalidade dos pontos apenas se todo o código for detalhado de maneira clara e suficiente.

## Entrada

Escreva um programa que implemente o jogo "caça-palavras". O seu programa deve ler da entrada padrão seguindo a seguinte especificação: A primeira linha contém o número de linhas ($n$) e colunas ($m$) da matriz de letras. As próximas $n$ linhas da entrada contém a matriz de letras. Em seguida, tem-se o número de palavras ($k$) a serem buscadas. As próximas $k$ linhas da entrada contém a lista de palavras.

## Saída

O programa deve procurar cada palavra da lista na matriz. Para cada palavra a ser procurada devem ser impressas as coordenadas da matriz onde a palavra se inicia e onde termina. Ou as coordenadas $(0, 0)$ caso a palavra não esteja presente no caça-palavras. Confira o exemplo de entrada e saída dados.

### Exemplo de entrada

```
5 6
sccelr
uehayr
geaimc
cavrea
cxepla
6
chave
mel
erva
cama
veu
uva
```

### Exemplo de saída

```
0 2 4 2 chave
2 4 4 4 mel
3 4 3 1 erva
0 2 3 5 cama
3 2 1 0 veu
0 0 0 0 uva
```

## Tipo abstrato de dados (TAD)

Para o bom sucesso do trabalho, você deverá estruturar seu programa criando tipos abstratos de dados (TADs) que melhor representam o problema. Os seguintes TADs são aconselháveis, porém você está livre para modificá-los ou acrescentar novos.

Implemente o TAD `TCoordenada` para representar uma posição na matriz de letras. O TAD `TPalavra` para representar uma palavra a ser buscada, seu tipo de dado deverá armazenar a palavra em si, sua posição (inicial e final) na matriz de letras e um valor booleano representando se a palavra foi encontrada.

Implemente também o TAD `TMatriz` para representar um caça-palavras, seu tipo de dado deverá armazenar a matriz em si e suas dimensões. Considere que o tamanho máximo de um caça-palavras é limitado pela memória principal (você deverá usar alocação dinâmica). Use arranjos bidimensionais (matrizes) para implementar o TAD. O TAD `TOcorrencias` para representar uma matriz de ocorrências (como explicado anteriormente), seu tipo de dado deverá armazenar o número de vezes que cada letra ocorre na matriz e as coordenadas de cada ocorrência.

As operações que devem ser realizadas por seus TADs são:

1. Criar uma nova coordenada.

   ```
   TCoordenada coordenada_criar(int i, int j);
   ```

2. Verificar se coordenada está dentro dos limites da matriz, retornando 1 em caso positivo e 0 caso contrário.

   ```
   int coordenada_verificar(TCoorenada coordenada, TMatriz* matriz);
   ```

3. Criar uma nova palavra.

   ```
   TPalavra palavra_criar(char* palavra);
   ```

4. Buscar palavra em uma coordenada específica do caça-palavras, retornando coordenada do final da palavra caso exista, ou coordenada (-1,-1) caso contrário.

   ```
   TCoordenada palavra_buscar_pos(TPalavra palavra, TMatriz* matriz,
                                  TCoordenada pos);
   ```

5. Buscar palavra no caça-palavras, retornando coordenada do final da palavra caso exista, ou coordenada (-1,-1) caso contrário.

   ```
   TCoordenada palavra_buscar(TPalavra palavra, TMatriz* matriz,
                              TOcorrencias ocorr);
   ```

6. Buscar uma palavra no caça-palavras, retornando 1 caso encontre e 0 caso contrário.

```
int palavra_buscar(TPalavra* palavra, TMatriz* matriz,
                   TOcorrencias ocorr);
```

7. Adicionar uma nova coordenada no vetor de ocorrências de letras de um caça-palavras, retornando 1 em caso de sucesso e 0 em caso de falha.

   ```
   int ocorrencias_adicionar(TOcorrencias* ocorr, TCoordenada coor);
   ```

8. retornar um vetor de ocorrencias da primeira letra de uma palavra

   ```
   TOcorrencias ocorrencias_buscar_palavra(TOcorrencias*, TPalavra);
   ```

9. Calcular ocorrências de letras de um caça-palavras, retornando 1 em caso de sucesso e 0 em caso de falha.

   ```
   int ocorrencias_calcular(TOcorrencias* ocorr, TMatriz* matriz);
   ```

10. Inicializar um caça-palavras vazio de dimensões $n \times m$, retornando 1 se possível e 0 se não.

    ```
    TMatriz* matriz_criar(int linhas, int colunas);
    ```

11. Preencher um caça-palavras a partir dos dados recebidos da entrada padrão, retornando 1 se possível e 0 se não.

    ```
    int matriz_preencher(TMatriz* matriz, int linhas, int colunas);
    ```

12. Imprimir um caça-palavras.

    ```
    void matriz_imprimir(TMatriz* matriz);
    ```

13. Adicionar uma palavra ao vetor de palavras, retornando o tamanho do vetor ou -1, caso não seja possível;

    ```
    int palavras_add(TPalavra* palavras, TPalavra palavra);
    ```

14. Preencher caça-palavras a partir dos dados recebidos da entrada padrão, retornando 1 se possível e 0 se não.

    ```
    int palavras_preencher(TPalavra* palavras, int num_palavras);
    ```

15. Resolver um caça-palavras (como explicado anteriormente), retornando 1 se possível e 0 se não.

    ```
    int matriz_solucionar(TMatriz* matriz, TOcorrencias ocorr,
                          TPalavra* palavras, int num_palavras);
    ```

16. Imprimir solução do caça-palavras.

    ```
    void palavras_imprimir_solucao(TPalavra* palavras);
    ```

17. Desalocar memória do vetor de palavras, retornando 1 se possível e 0 caso contrário.

    ```
    int palavras_apagar(TPalavra* palavras);
    ```

18. Desalocar memória da matriz de ocorrências, retornando 1 se possível e 0 caso contrário.

    ```
    int ocorrencias_apagar(TOcorrencias* ocorrencias);
    ```

19. Desalocar memória da matriz caça-palavras, retornando 1 se possível e 0 caso contrário.

    ```
    int matriz_apagar(TMatriz* matriz);
    ```

Implemente os seus TADs em arquivos separados do programa principal (`<nome_do_tad>.h` e `nome_do_tad.c`) Se necessário, você poderá criar outras funções auxiliares em seu TAD. Suas funções devem executar testes de consistência (por exemplo, não se pode inserir uma palavra em uma posição inválida).

Uma vez criado os seus TADs, você deverá utilizá-lo em um programa que recebe o caça-palavras (pela entrada padrão) e imprime (também pela entrada padrão) sua solução (como mostrado anteriormente). Procure testar o programa com várias configurações de caça-palavras. Crie-os usando sua imaginação.

Note que o seu programa principal não poderá acessar diretamente a estrutura interna do TAD. Se necessário, acrescente novas funções ao seu TAD detalhando-as na documentação do trabalho. A sua aplicação deve ser criada para ser "portátil", isto é, ela deverá funcionar com diversos sistemas operacionais (Windows, Linux, macOS, etc.) Desse modo, você não poderá utilizar métodos da linguagem C que são dependentes do sistema, como a função:

```
int system(const char *command);
```

## O que deve ser entregue

Código fonte do programa em C (bem identado e comentado) e documentação do trabalho. Entre outras coisas, a documentação deve conter:

1. **Introdução**: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
2. **Implementação**: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. *Muito importante*: os códigos utilizados nas implementações devem ser inseridos na documentação.
3. **Estudo de complexidade**: estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O).
4. **Listagem de testes executados**: os testes executados devem ser apresentados, analisados e discutidos.
5. **Conclusão:** comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
6. **Bibliografia**: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso. Uma referência bibliográfica deve ser citada no texto quando da sua utilização
7. **Em Latex**: caso o trabalho seja elaborado/escrito em latex, ganha-se 0,2 ponto.
8. **Formato**: mandatoriamente em PDF.

Obs1: Consulte as dicas do Prof. Nívio Ziviani de como deve ser feita uma boa implementação e documentação de um trabalho prático: https://homepages.dcc.ufmg.br/ nivio/cursos/aed2/roteiro/.

Obs2: Veja modelo de como fazer o trabalho em latex: https://bit.ly/4jMoUQh

## Como deve ser feita a entrega

A entrega DEVE ser feita pelo Moodle (https://moodlepresencial.ufop.br) na forma de um único arquivo zipado, contendo o código, os arquivos e a documentação. Também deve ser entregue a documentação impressa na próxima aula (teórica ou prática) após a data de entrega do trabalho.

## Comentários gerais

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- Faça o trabalho de maneira incremental, comece com os métodos mais simples, sempre testando exaustivamente antes de passar para a próxima etapa;

- Clareza, identação e comentários no programa também serão avaliados;
- O trabalho é individual (grupo de UM aluno);
- Trabalhos copiados (e FONTE) terão nota zero; Devido a recorrentes problemas com cópias de trabalhos (plágios), os autores de trabalhos copiados também terão todos os demais trabalhos zerados, como forma de punição e coação ao plágio acadêmico;
- Trabalhos entregue em atraso serão aceitos, todavia a nota atribuída ao trabalho será zero;
- Evite discussões inócuas com o professor em tentar postergar a data de entrega do referido trabalho.