UNIVERSIDADE ESTADUAL DE MARINGÁ

DEPARTAMENTO DE INFORMÁTICA

CIÊNCIA DA COMPUTAÇÃO

ORGANIZAÇÃO E RECUPERAÇÃO DE DADOS

PROFª.: VALÉRIA DELISANDRA FELTRIM

DESENVOLVIMENTO DO ALGORITMO DE ORDENAÇÃO "MERGE SORT EXTERNO"

JOÃO OTÁVIO BIONDO RA: 81746

MARCO AURÉLIO DEOLDOTO PAULINO RA: 82473

MARINGÁ

14 DE SETEMBRO DE 2015

1. Passo a passo para a compilação e execução de programa

Para compilar o programa:

- Ir para a pasta do código fonte do programa pela linha de comando.
- Digitar **gcc main.c –o main**

Para executar o programa:

- Ir para a pasta do código fonte do programa pela linha de comando.
- Digitar **main** e apertar ENTER.

O programa foi testado apenas no Windows 8.1 e Window 10.

Para que o programa funcione é necessário que os arquivos a serem convertidos estejam na mesma pasta do executável gerado.

2. Decisões de Projeto

Organização do programa:

- **main.c:** O programa está contido neste arquivo e apresentam as seguintes funções:
  - **abreArquivo:** Função para a abertura do programa;
  - **recebParamentros:** Recebe o tamanho os buffers e calcula a quantidade de Registros, tamanho mínimo do Buffer e o total de corridas;
  - **divisaoArquivo:** Faz a leitura para o Buffer de entrada, chama a função para a ordenação e salva os registros ordenados de cada corrida;
  - **radixsort:** Função de ordenação;
  - **merging:** Função para realizar a junção das corridas;
  - **menorChave:** Função que devolve o registro com menor chave entre as corridas;
  - **lerBloco:** Função que devolve o número de registros lidos.

Algoritmos utilizados:

Para a ordenação dos registros foi escolhidos a utilização do algoritmo *Radix Sort*, devido a sua rapidez e estabilidade, com complexidade $\theta(nk)$, sendo n: número de chaves e k: comprimento médio das chaves, como no caso deste trabalho, temos que o comprimento de chaves é fixo, 4, podemos definir a complexidade do algoritmo em $\theta(4n)$, ou ainda como $\theta(n)$, visto que com o número de chaves a ser ordenado crescer tendendo ao infinito, qualquer constante, se torna desprezível em relação a n.

Estruturas utilizadas:

Ao longo do projeto foram utilizadas algumas estruturas de dados para ao alcance dos objetivos propostos, tais estruturas serão explicadas a seguir:

Registro: Nesta estrutura é utilizada para armazenar a chave e descrição dos Registros;

- **regLidos** (em divisaoArquivo): Vetor de "Registro" para armazenar os Buffer Lidos em cada Corrida, tem a dimensão do tamanho do Buffer de Entrada;
- **regLidos** (em merging): Matriz de "Registro" com a dimensão de total de Corridas pela quantidade de Registros por Corridas;
- **regSaida**: Vetor de "Registro" para armazenar os Registros do Buffer de Saída e tem a dimensão do tamanho do Buffer de Saida

Arquivos usados e gerados:

Os arquivos de entrada que serão usados para serem convertidos precisam estar na mesma pasta do executável. Na execução do programa é pedido os nomes dos dois arquivos para importar.

Os arquivos de saída gerados serão:

- **corrida(0~totalCorridas).txt:** Arquivos que armazenam cada corrida ordenada, ao todo contém o número de Registros em cada arquivo equivale ao total de registros que o Buffer de Entrada pode armazenar, exceção pode ocorrer no último arquivo de Corrida, que pode armazenar os registros restantes, que pode ser menor que o total de registros que o Buffer de Entrada pode armazenar, por exemplo, em um arquivo de 500 registros com buffer de entrada que armazena no máximo 24 corridas, no último arquivo de corridas, teria 20 registros ao invés de 24 registros.

- **Saída-ordenado.txt:** Arquivo que armazena todos os registros de forma ordenada.