**ALGORITMO DE ORDENAÇÃO EXTERNA**

**NATAN LUIZ PAETZHOLD BERWALDT**

**DENES VARGAS TEIXEIRA**

nlberwaldt@inf.ufsm.br e dvteixeira@inf.ufsm.br

# Introdução

O presente trabalho tem por objetivo implementar um algoritmo de ordenação externa de textos grandes na linguagem de programação C++, utilizando o método QuickSort para a ordenação interna dos itens com tamanho de memória definidos, e a Intercalação Balanceada de vários conjuntos ordenados de dados em arquivos para trabalhar com a relação de memórias interna e externa, gerando um arquivo de saída com as palavras ordenadas após todo o processo de ordenação.

# Algoritmo

**2.1 - Funcionamento**

Para o funcionamento do programa é necessário que o arquivo com os dados a serem ordenados estejam no mesmo diretório do programa com o nome ‘entrada’ e no tipo ‘txt’.

A princípio temos um arquivo de texto muito grande que não cabe em memória interna para a execução de algoritmos de ordenação, sabendo disso o principio do algoritmo baseia-se em ler sequencialmente o arquivo por pequenos conjuntos e ordenando cada uma dessas partes isoladamente, definidas por um tamanho fixo no programa.

Após as ordenações, o conteúdo estará dividido entre os arquivos temporários e inicia-se o processo de intercalação entre os arquivos ‘temp’ e ‘out’, na primeira execução teremos pequenos conjuntos ordenados divididos nos arquivos ‘temp’ e se intercala os elementos de cada conjunto correspondente de cada arquivo ‘temp’ para um único arquivo ‘out’, assim esses conjuntos de cada arquivo se juntam e formam um conjunto maior de dados ordenados. Entretanto, cada arquivo ‘temp’ possui vários conjuntos ordenados e cada um desses novos conjuntos maiores gerados das intercalações vão ser distribuídos dos arquivos ‘temp’ para um mesmo número de arquivos ‘out’.

Após todos os conjuntos serem intercalados e divididos entre os novos arquivos, os arquivos ‘temp’ serão sobrescritos com a nova intercalação dos arquivos ‘out’ anteriores, e assim gerando conjuntos internos ordenados ainda maiores. Esse processo de intercalação entre esses dois grupos de arquivos temporários é recursivo, até que o conjunto gerado seja do mesmo tamanho do arquivo de entrada e fique em um único arquivo de saída.

Após o arquivo de saída ser gerado, todos os arquivos temporários usado no processo de intercalação são excluídos.

O número de ciclos de intercalação necessárias entre esses arquivos temporários é calculado previamente pelo programa com base no número de conjuntos ordenados em memória interna, para assim ter melhor controle das execuções.

**2.2 – Implementação**

Logo no início do programa temos dois Defines que controlam o número de arquivos usados entre a intercalação, onde cada ciclo de intercalação usará o dobro de arquivos ali definidos, pois um é o arquivo onde estão os dados temporariamente, e o outro servirá de destino da ordenação parcial.

O segundo Define é o número de Bytes máximo usados pelos conjuntos iniciais de dados no processo de ordenação interna.

O arquivo de entrada é lido no inicio do programa e ordenados os pequenos conjuntos de palavras (respeitando o número de Bytes máximo) por um QuickSort interno.

Cada conjunto ao ser ordenado é enviado para a função ‘grava’, que coloca os conjuntos de dados ordenados nos arquivos, intercalando os arquivos. Após isso, o numero de intercalações é definido pelo calculo baseado no numero de arquivos e pelo número de conjuntos ordenados.

Em seguida, ele inicia a ordenação externa com a função recursiva ‘intercalacao’ que usa dois vetores de arquivos, sendo um onde estão os dados e o outro o destino dos dados da intercalação.

Dependendo do ciclo de execução ele verifica onde estão os dados e para quais conjuntos de arquivos eles irão no ciclo atual da recursão e coloca as primeiras palavras de cada conjunto ordenado em um vetor onde o índice é correspondente ao arquivo temporário no vetor de arquivos.

Procura-se o menor valor entre os arquivos e o escreve em um outro arquivo temporário, quando acaba-se o conjunto atual de cada arquivo, ele vai para o próximo conjunto de cada arquivo novamente, mas alterando o arquivo de destino para distribuir os novos conjuntos.

Ao terminar todos os conjuntos de um grupo de arquivos ele recalcula o número de execuções restantes para sobrar apenas um arquivo e quando o número de execuções previsto chega ao máximo, o ciclo de intercalação dos arquivos acaba chamando a função ‘encerra’.

A função ‘encerra’ é responsável por reescrever todo o código ordenado para um arquivo final de saída. Ao sair dela o programa deleta todos os arquivos temporários e termina a execução.

# Aplicações

As aplicações do algoritmo de ordenação são muito variadas em especial a de palavras, cujo relacionada ao presente trabalho, pode servir na criação de dicionários em um banco de dados, os quais permitem uma busca mais rápida por um nome específico, podendo o algoritmo acelerar consideravelmente o processo de busca através da utilização de métodos para dados ordenados.

Mas o uso de ordenação externa se torna um pouco restrito ao fato de o acesso em memória secundário se torna muito mais caro em memoria secundária do que em primária, ou seja, o tempo de execução é penalizado. Além disso o acesso é feito sequencialmente na memória.

**Referências**

https://pt.wikipedia.org/wiki/Quicksort

https://en.wikipedia.org/wiki/External\_sorting

http://wiki.icmc.usp.br/images/1/1e/SCC0203-1o-2012-15.OrdenacaoExterna.pdf

https://www.youtube.com/watch?v=ATK74YSzwxg

http://www.decom.ufop.br/guilherme/BCC203/geral/ed2\_ordenacao-externa.pdf

http://www.showme.com/sh/?h=0a0OGGW