Tiago Cahú Lacerda do Nascimento RA: 103272

Vinícius Cerqueira Silva RA: 111494

Marcos Deilson da Silva Freitas RA: 080242

Jhonathas Neves dos Santos RA: 101715

Professora Josiane Rodrigues

Análise de Algoritmos

5/5/16

Ordenação Externa Utilizando Merge-Sort

Objetivo

Este relatório tem como objetivo demonstrar as análises dos resultados obtidos após o desenvolvimento e aplicação do método Merge-Sort. Pode-se citar ainda objetivos específicos:

- Conceito de Ordenação externa;
- Entender interações do método Merge-Sort multivias;
- Desenvolvimento de algoritmo utilizando os conceitos anteriores para ordenar um arquivo de dados;
- Verificar eficácia do algoritmo desenvolvido.

Implementação

Para a implementação do algoritmo de ordenação foi utilizada a linguagem C. O trabalho prático pode ser encontrado para a verificação de sua validade em: https://github.com/vinics12/Trabalho1AnaliseProfJosiane.

Desenvolvimento Prático

Questão 1.

Para um melhor entendimento do algoritmo, vamos descreve-lo em passos:

Passo 1

Receber os parâmetros na seguinte ordem:

1-Arquivo de entrada(.txt);

2-Nome que será utilizado no arquivo de saída(.txt);

3-M(megabytes) quantidade máxima de memória que o programa poderá usar 4-K(2~9) quantidade de arquivos que serão combinados ao mesmo tempo.

Passo 2

Chamar a função PARTICIONAR que além de dividir em partes de tamanho máximo M os arquivo de entrada, irá converter o valor de Megabytes para bytes, ordenar (utilizando o algoritmo mergesort) as partes que geradas de tamanho máximo M e retornar o número de partes necessárias para dividir o arquivo de entrada em partes de tamanho máximo M. Infelizmente é necessário ler o arquivo de entrada duas vezes, a primeira para saber quantas posições deveremos alocar e a segunda para inserirmos os valores lidos no vetor alocado.

Considerando que cada caracter necessita de 1 byte, por exemplo o numero 1000 precisa de 4 bytes, e os valores gerados no arquivo de entrada são de 0 até 99.999 em média teríamos 3 bytes por valor mas usaremos 4 para garantir que o a memória alocada não ultrapasse o valor M. Após saber o valor total de M em bytes e levar em conta o paragrafo anterior, vamos dividir nosso M(já em bytes) por 4 para obter o numero máximo de posições que o vetor pode ter sem ultrapassar o valor M. Como vamos utilizar K vetores, vamos ter que dividir esse vetor de (M/4) por K vetores para todos terem a mesma capacidade obedecendo a regra do limite.

Resumindo, é criada uma matriz de K linhas por J colunas onde J = (M/4)/K.

Passo 3

Executar um laço enquanto a quantidade de partes geradas seja maior que 1, pois devemos gerar um arquivo de saída com o nome do parâmetro 2 com os mesmos valor do arquivos de entrada porém ordenados.

Considerando que cada laço seja um nivel, exemplo... 1,2,3,4,5,6(nível 0) ; 1-2,3-4,5-6(nível 1) ; 1-4,5-8(nível 2) ; parâmetro 2 (nível 3). Agora vamos chamar a função MergeMultiNivel que trabalha em conjunto com a função COMBINA que utiliza as funções RECARREGA e MENOR, em seguida veremos em partes. A função MergeMultiNivel só acaba quando termina de executar a combinção de todas as partes daquele nível e também controla os k arquivos que serão lidos por vez. Ela preenche os K vetores a primeira vez e envia para a função COMBINA que é responsavel por combinar os K arquivos da vez, com auxílio da função MENOR que retorna o menor valor da vez entre os K vetores, ela vai escrevendo o novo arquivo que é a combinação dos K arquivos assim gerando o próximo “nível”.Porém se o programa não pode alocar todos os valor, o que acontece quando o vetor chegar ao fim? quando o vetor tem seu último valor retornado pela função MENOR, é executada uma verificação da ultima posição, se o vetor foi até a ultima posição e não encontrou o valor ‘-1' é porque ainda tem valores no arquivo correspondente aquele vetor, então é chamada a função RECARREGA que vai preencher o vetor com o seu arquivo correspondente até que o ultimo valor seja inserido no novo arquivo que a função COMBINA está escrevendo. Nesse intervalo de leitura e escrita é necessário controlar os nomes para não deixar de ler um arquivo ou acabar lendo um arquivo errado. Para esse controle de nomeclatura temos as funções INCREMENTA, proximoNomeL,proximoNomeE e nivelUP.

Como MergeMultiNivel é uma função ela gera um retorno e esse retorno é a quantidade de partes geradas pela combinação do "nível" anterior e ela vai fazendo isso enquanto retornar mais de uma parte;

Passo 4

Para saber o tempo de execução apenas do laço que executa o MergeMultiNivel usamos a função CLOCK que retorna um "relogio" do processador antes e depois do laço subtraímos o valor do clock final pelo inicial e depois dividimos por CLOCKS_PER_SEC/1000 como o nome da constante já diz, clocks por segundo, para sabermos o valor em segundos,

mas, como solicitado o tempo em milisegundos também dividimos por 1000.

Questão 2.

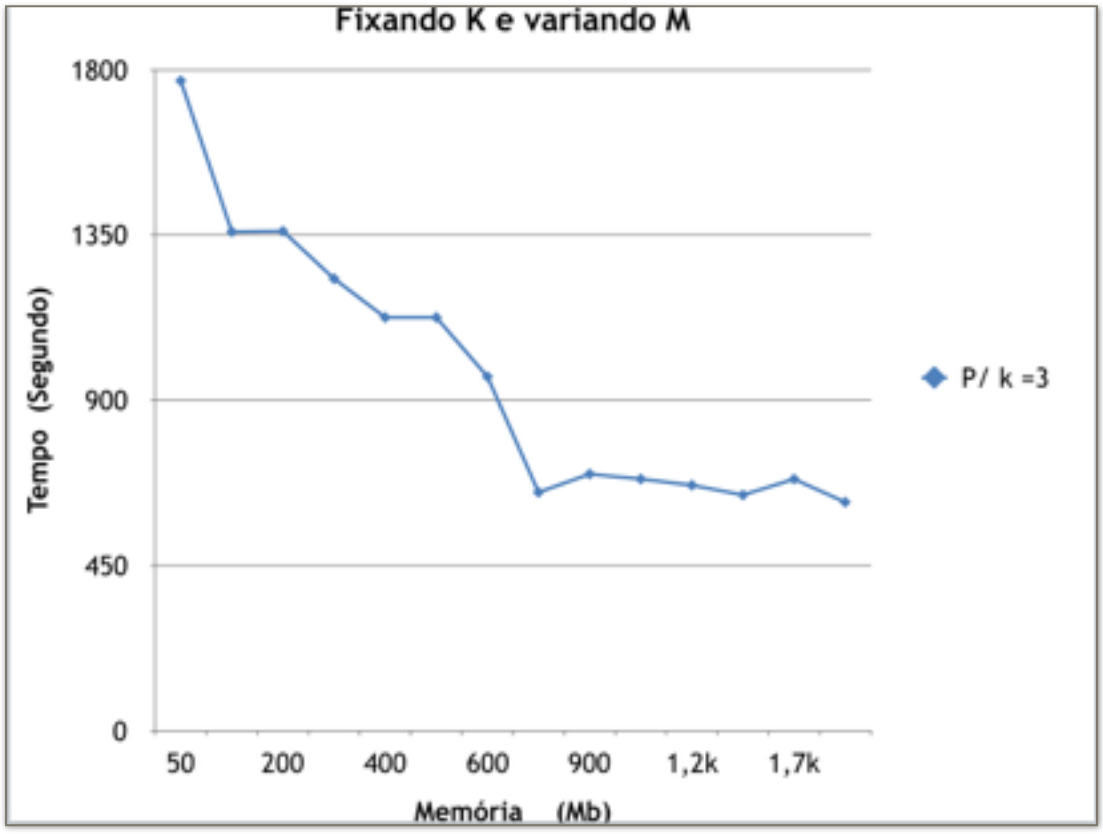
O programa encontra-se no endereço:
https://github.com/vinics12/Trabalho1AnaliseProfJosiane

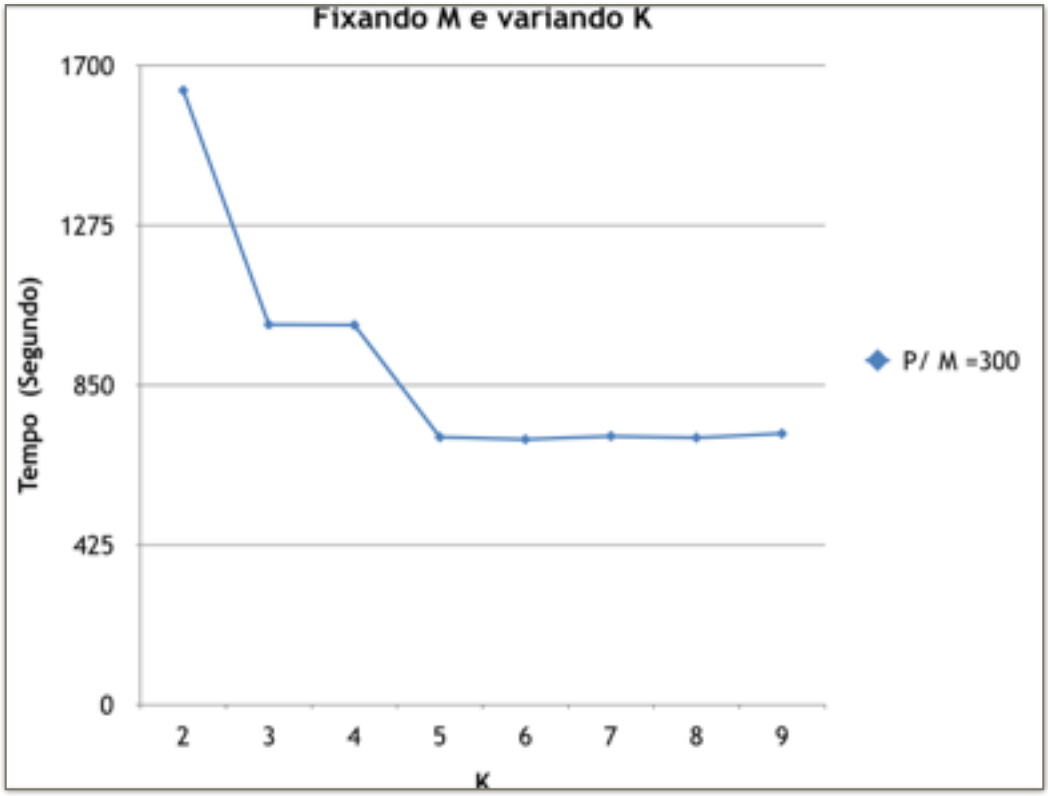
Questão 3.

A complexidade do programa em questão será O(log n) justamente pelo seu mecanismo de particionamento ser similar a altura de uma árvore.

Resultados

Questão 4.





Analisando e comparando os gráficos, é possível perceber que tanto no crescimento de K, quanto no de M, o tempo e execução tendem a cair respectivamente.