Documentação - Trabalho Pratico 1 - AEDS III

Aluno – Jesimon Barreto Santos (2016070093)

1 Introdução

Mathias, como todo estudante, é fascinado pela vida e por sua aleatoriedade. Porém, ele é um estudante esforçado e, por mais improvável e louca a ideia, ele não mede esforços para realizar seus desejos. Certa vez, ouviu falar do "Infinite Monkey Theorem". Ninguém duvidava de Mathias. Muito menos depois de conseguir alguns macacos para lhe ajudar (nenhum animal foi machucado para o enunciado deste trabalho). Embora tenha conseguido os macacos, que digitam linhas de tamanho fixo (sim, eles conseguem fazer isso sem problemas), Mathias está com um problema! Ele quer ordenar essas linhas digitadas em um computador ancião que ele tem em casa. O problema é que de vez em quando, um pente de memória RAM falha, fazendo com que Mathias não tenha mais espaço o suficiente para armazenar todas as linhas nela.Por isso, ele quer sua ajuda! Você é um amigo de Mathias meio preguiçoso (e que, por questões óbvias, tenta não se envolver muito com Mathias). Mathias tem quase todo o código pronto, ele só precisa de uma parte em específico: ordenar as linhas em uma quantidade pequena de memória. Você, apesar de não ser o único capaz de ajudar Mathias, foi a única pessoa que ele conseguiu encontrar (você não viu ele se aproximando...).

Felizmente, Mathias já tem quase tudo pronto. Ele até criou um sistema para verificar quanta memória virtual (é só com isso que precisamos nos preocupar). Então, você deve fazer o download do código dele para se livrar logo desta tarefa. Após o download, o único arquivo a ser editado é: sort.c. Nele, você encontrará duas funções com uma breve descrição delas:

1. **a_menor_que_b**: você deve implementar esta função de forma que ela retorne 1 se a cadeia de caracteres a é menor que a cadeia de caracteres b (as duas com tamanho *len*);
2. **external_sort**: essa é a função principal! Você deve ler o arquivo *input_file*, ordená-lo e escrever as linhas ordenadas no arquivo *output_file*, usando apenas *memory* KB de memória virtual.

2 Abordagem do Problema

Com base no problema apresentado, foi induzido uma abordagem de ordenação em memoria externa, visto que o dado não cabe em toda a memória principal (RAM). Com isso, o custo de acesso ao dado na memória externa é o fator de maior lentidão, então o objetivo é reduzi-lo ao máximo. Nesse momento, foi pesquisando usando o referencial teórico para implementação do método, inicialmente é importante salientar os passos de forma genérica de um métodos de ordenação externa.

i. Dividir o arquivo em blocos do tamanho da memória interna disponível.
ii. Ordenar cada bloco na memória interna
iii. Intercale os blocos ordenados, fazendo várias passadas sobre arquivo

Usando o pressuposto que os blocos intercalados anteriormente estão ordenados, ao fim o arquivo completo estará ordenado. Um métodos dessa classe é chamado de ordenação por intercalação balanceada de vários caminhos, o algoritmo geral é descrito abaixo:

1. Leitura do primeiro registro de cada fita.

2. Retirada do registro contendo a menor chave, armazenando-o em uma fita de saída.

3. Leitura de um novo registro da fita de onde o registro retirado é proveniente.

Ao ler o terceiro registro de um dos blocos, a fita correspondente fica inativa.

A fita é reativada quando o terceiro registro das outras fitas forem lidos.

Neste momento, um bloco de nove registros ordenados foi formado na fita de saída.

4. Repetição do processo para os blocos restantes.

Seguindo esse algoritmo o problema foi resolvido, a explicação da implementação relacionada a esse algoritmo:

Nessa explicação, registro se refere a uma linha completa do arquivo de entrada.

Inicialmente o programa aqui descrito, aloca dois ponteiros (*line* e *strG,* respectivamente) do tipo char, *line* é alocado tamanho de caracteres por linha e *strG* com tamanho *memory* (memoria disponível) menos a quantidade de caracteres por linha, ou, o restante da memória passado como parâmetro.

Inicia lendo o arquivo de entrada, usando *line*, guarda até encher a *strG*, depois disso, é aplicado o *bubble sort*, depois esse bloco é salvo em fita1 ou fita2, alternadamente. Quando as linhas do arquivo de entrada acabam, tem a possibilidade de *strG* estar com todas as linhas nele, o que significa que a memória disponível coube todas as linhas, então depois da ordenação interna, já é salvo diretamente no arquivo de saída e a execução é finalizada. Terminada a primeira fase, que consistiu no passo *i* dos algoritmos de ordenação externa, teremos dois arquivo, fita1 e fita2 com blocos ordenados de linhas de tamanho *strG* (*memory* – quantidade de caracteres por linha), distribuídos alternadamente. A partir disso, agora está pronto para começar a intercalação propriamente dita.

Agora é iniciado a intercalação, Nesse ponto a memória é liberada e tanto *line* quanto *strG* receber tamanho da quantidade de caracteres por linha. Assim, *line* irá ler o primeiro registro do bloco 1 da fita1, do mesmo modo que, *strG* irá ler o primeiro registro do bloco 1 da fita2, próximo passo a comparação e o menor é salvo em um novo bloco no arquivo fita3 ou fita4, alternadamente, caso o menor seja o registro de *line*, então *line* é atualizado e irá apontar para o registro 2 do bloco 1 da fita1, do mesmo modo se *strG* estiver apontando para o menor Registro, se um bloco não chegar ao final, então o outro é escrito para completar o próximo bloco, esse novo bloco é escrito em fita3 ou fita4, seguindo a ideia alternada. Ao final dessa fase teremos em fita3 e fita4 blocos com o dobro do tamanho que tinham em fita1 e fita2 e ordenados.

Se esse processo for repetido entre a fita1 e fita2; e fita3 e fita4, o bloco resultante da comparação dos primeiros blocos são escritos na fita1 (quando está intercalando entre fita3 e fita4) ou fita3(quanto está intercalando entre fita1 e fita2), desse modo, o final acontecerá quando o único bloco gerado conterá todos os registros e estará na fita1 ou fita3, com base nessa análise, o *looping* é feito até fita2 ou fita4 estiverem nulas.

O processo é usado com 4 fitas, cujo 2 são usadas simultaneamente, de tal forma a criar a intercalação balanceada de 2 caminhos.

3 Complexidade

Como já descrito anteriormente, o que prevalece em termos de tempo é a quantidade de acessos a memória secundaria, contabilizada por passadas no arquivo.

Para contabilizar essas passados temos a formula dada por:

$$P(n) = \log_f (n/m) + 1$$

Onde:

- n -> o número de registros do arquivo.
- m-> é o tamanho da memória interna disponível
- f -> o número de fitas utilizadas em cada passada.

Dessa maneira, encontramos P(n) que é o número de passadas na fase de intercalação, somamos 1 pois refere-se a primeira passada que é feita antes para primeira ordenação interna, divisão em blocos e serem colocados na fita1 e fita2.

Se *memory* >= 2 * chave, TamanhoBlocoInicial = ([*memory*/chave] – 1) * chave;
Senão se m < 2 * chave, então não existe.

Chave -> número de caracteres por linha
Memory -> memória disponível passada como parâmetro

Isso acontece porque uma quantidade *chave* é usada para pegar linha a linha cada registro. Para esse programa então o P(n) é:

$$P(n) = \log_2 (n/ \text{TamanhoBlocoInicial}) + 1$$

Onde:

- n -> o número de linhas que existem no arquivo de entrada
- m -> é dado pela função calculada anteriormente, pois, lembramos que um parte da memória total disponibilizada é usada para leitura linha a linha.
- f -> é sempre igual a 2, pois todo o processo é feita intercalação por 2 caminhos.

Dessa maneira, faz sentido, pois o número de passadas depende diretamente da quantidade de linhas lidas e do número de memorias disponíveis.

4 Experimentos

Os testes foram feitos em um computador com processador i5, 2.6 GHz e memória de 1 GB, no sistema operacional baseado em Linux.

Os casos de teste disponibilizados foram executados com sucesso, seguem o comportamento expresso como a seguir:

Tamanho de memória fixado em 160 bytes:

- Entrada: testes/test_010.010_1.txt Tempo: 4.69000 milisegundos
- Entrada: testes/test_010.040_1.txt Tempo: 26.5710 milisegundos
- Entrada: testes/test_010.080_1.txt Tempo: 38.8990 milisegundos
- Entrada: testes/test_010.320_1.txt Tempo: 24.4250 milisegundos
- Entrada: testes/test_010.640_1.txt Tempo: 58.9090 milisegundos
- Entrada: testes/test_040.010_1.txt Tempo: 14.2420 milisegundos
- Entrada: testes/test_020.010_1.txt Tempo: 4.31900 milisegundos
- Entrada: testes/test_020.040_1.txt Tempo: 12.2330 milisegundos
- Entrada: testes/test_020.080_1.txt Tempo: 16.1200 milisegundos
- Entrada: testes/test_020.160_1.txt Tempo: 34.1390 milisegundos
- Entrada: testes/test_020.320_1.txt Tempo: 39.4450 milisegundos
- Entrada: testes/test_080.010_1.txt Tempo: 16.0420 milisegundos
- Entrada: testes/test_080.040_1.txt Tempo: 25.1070 milisegundos
- Entrada: testes/test_080.080_1.txt Tempo: 22.7960 milisegundos
- Entrada: testes/test_080.160_1.txt Tempo: 20.2010 milisegundos
- Entrada: testes/test_080.320_1.txt Tempo: 24.3470 milisegundos

Tamanho de memória que caibam apenas 2 linhas de cada arquivo:

- Entrada: testes/test_010.010_1.txt Tempo: 16.6830 milisegundos
- Entrada: testes/test_010.040_1.txt Tempo: 23.8340 milisegundos
- Entrada: testes/test_010.080_1.txt Tempo: 34.4830 milisegundos
- Entrada: testes/test_010.160_1.txt Tempo: 24.9320 milisegundos
- Entrada: testes/test_010.320_1.txt Tempo: 35.0040 milisegundos
- Entrada: testes/test_010.640_1.txt Tempo: 27.2200 milisegundos
- Entrada: testes/test_020.010_1.txt Tempo: 11.9660 milisegundos
- Entrada: testes/test_020.040_1.txt Tempo: 21.9600 milisegundos
- Entrada: testes/test_020.080_1.txt Tempo: 23.4080 milisegundos
- Entrada: testes/test_020.160_1.txt Tempo: 32.4190 milisegundos
- Entrada: testes/test_020.320_1.txt Tempo: 66.4890 milisegundos
- Entrada: testes/test_040.010_1.txt Tempo: 10.7130 milisegundos
- Entrada: testes/test_080.010_1.txt Tempo: 38.1520 milisegundos
- Entrada: testes/test_080.040_1.txt Tempo: 21.6330 milisegundos
- Entrada: testes/test_080.080_1.txt Tempo: 64.1190 milisegundos
- Entrada: testes/test_080.160_1.txt Tempo: 558.401 milisegundos
- Entrada: testes/test_080.320_1.txt Tempo: 35.5590 milisegundos

5 Conclusão

Esse programa foi criado para resolver um problema de ordenação de arquivos que são muito grandes para serem carregados todo na memória RAM de uma vez, gerando um problema de ordenação em memoria secundaria. Para resolver foi aplicada a intercalação balanceada com 2 caminhos.

6 Referencial Teórico

Ziviani, N. *Projeto de Algoritmos com Implementações em Pascal e C*, São Paulo, Brazil, Cengage Learning, ISBN 13 978-85-221-1050-6, 2010, third edition reviewed and extended, 659 pages (in Portuguese).

http://www.decom.ufop.br/guilherme/BCC203/geral/ed2_ordenacao-externa.pdf. *Arquivo material de aula do professor Guilherme Tavares de Assis(UFOP)*.