Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências de Computação

Profs. Gisele L Pappa e Wagner Meira Jr.
Estrutura de Dados 2021.02

Trabalho Prático 2 - Ordenação em Memória Externa

Valor: 10 pontos
Data da entrega: 20/01/2021

Introdução

Devido a grande quantidade de dados gerados todos os dias, está cada vez mais difícil processar e extrair informações relevantes deste grande volume de dados. Em uma plataforma de Web Analytics[1], por exemplo, uma das tarefas é detectar quais os sites mais populares e ordená-los. Um arquivo contendo URLs e o número de visitas que cada uma recebe pode ter diversos Gb de tamanho, excedendo facilmente a memória principal (RAM) de um computador pessoal. Neste trabalho prático cabe a você resolver este problema implementando um algoritmo de ordenação que utilize não só a memória principal mas também a memória secundária (disco, fita, pendrive, holograma).

Descrição do Algoritmo

O algoritmo para ordenação será dividido em duas etapas: na primeira o algoritmo deverá ler do arquivo de entrada um determinado número de entidades (que no nosso caso são páginas Web e seus respectivos números de visitas) que possa ser ordenado em memória principal. Esse conjunto deverá ser ordenado e escrito em um arquivo de nome “rodada-**n**.txt” onde **n** é o número do arquivo gerado. Por questões históricas, esses arquivos serão chamados de fitas[2]. Na segunda etapa os **n** arquivos gerados serão intercalados, até que um arquivo final seja gerado contendo todo o conteúdo ordenado. A seguir o pseudo-algoritmo em alto nível:

ORDENA(arquivo):
1) rodadas = gera_rodadas_ordenadas(arquivo)

[1] *Web analytics* desenvolve ferramentas capazes de medir, coletar e analisar dados para ajudar um negócio que funciona online (que pode ser uma loja ou uma plataforma social). Essas ferramentas podem responder como: "qual o comportamento de um usuário quando ele acessa a página do negócio?", ou mostrar como é a navegação das pessoas dentro de um conjunto de páginas.
[2] http://pt.wikipedia.org/wiki/Fita_magn%C3%A9tica

2) saida = intercala_rodadas(rodadas)

Gerando as rodadas

Para gerar as rodadas, o arquivo de entrada deve ser lido enquanto as entidades couberem na memória principal. No caso deste trabalho prático, o **número de entidades a ser lido** nesta etapa **será um parâmetro** do programa. Após a leitura das entidades, esse conjunto deve ser ordenado com um algoritmo de ordenação e o resultado deve ser escrito no arquivo relativo à rodada. O pseudo-algoritmo para essa etapa é apresentado a seguir:

GERA_RODADA(n):
1) entidades = le_entidades(NUM_ENTIDADES)
2) ordena(entidades)
3) escreve(entidades, “rodada-”+n+”.txt”)

Obs. O algoritmo de ordenação utilizado nesta fase deve ser o Quicksort. Na documentação deve ser discutida a escolha do pivô.

Intercalando n-rodadas (ou fitas)

Após as **n** rodadas terem sido escritas em arquivos separados, inicia-se o processo da intercalação. Os **n** arquivos são lidos, uma entidade por vez, e inseridos em um heap. A maior entidade (no nosso caso, a página web com o maior número de visitas) é então escrita na saída e uma nova entidade do arquivo do qual a entidade saiu é lida e inserida no heap. Esse processo é repetido até que o heap fique vazio.

No pseudocódigo abaixo, cada arquivo é chamado de “fita”, em homenagem aos pioneiros da computação que criaram esse algoritmo para funcionar com fitas magnéticas para cumprir o papel de arquivos, em uma época que as unidades de disco eram raras e caríssimas.

INTERCALA(fitas):
1) heap = Heap()
4) // le a primeira entidade de cada fita
5) for fita in fitas:
6) heap.add(fita.read())
7) while !heap.empty():
8) entidade = heap.pop()
9) escreve entidade no arquivo de saida
10) origin_run = checa de qual arquivo entidade veio
11) if !eof(fitas[orig_run]):
12) heap.add(fita.read())

Um exemplo de execução para 4 fitas, considerando o parâmetro de entrada do algoritmo ("tamanho da memória") como sendo no máximo 6:

F1: 8 7 4 3 2 1
F2: 16 12 10 9 8 4
F3: 14 12 9 6 4 1
F4: 29 18 14 12 9 8
Heap:
FSaida:

Ao inserir os primeiros elementos no heap:
F1: 7 4 3 2 1
F2: 12 10 9 8 4
F3: 12 9 6 4 1
F4: 18 14 12 9 8
Heap: 29 16 14 8
FSaida:

O maior número é inserido na saída. Como ele veio da fita F4, um novo número é lido da mesma e colocado no heap.
F1: 7 4 3 2 1
F2: 12 10 9 8 4
F3: 12 9 6 4 1
F4: 14 12 9 8
Heap: 18 16 14 8
FSaida: 29

O maior número é lido do heap e escrito na saída. Como ele veio da fita F4 um novo número é lido da mesma e inserido no heap.
F1: 7 4 3 2 1
F2: 12 10 9 8 4
F3: 12 9 6 4 1
F4: 14 12 9 8
Heap: 16 14 14 8
FSaida: 29 18

Esse processo é repetido até que o heap fique vazio, ou seja, todas as entidades foram lidas de todos os arquivos.

Entrada

O seu programa deve receber como parâmetros o **nome do arquivo de entrada** a ser ordenado, o **nome do arquivo em que a saída deve ser escrita** e o **número de entidades que podem ser carregadas**

para a memória primária (NUM_ENTIDADES). Cada entidade corresponde a um par que contém uma URL e o número de vezes em que ela foi visualizada. O programa deve receber pelo menos os seguintes parâmetros de execução:

- Arquivo de entrada
- Arquivo de saida
- Número de entidades por rodada
- Número de fitas

O arquivo de entrada a ser ordenado consiste em uma lista de entidades (uma por linha). Cada linha traz no nome da página web seguido de um espaço em branco e do número de vezes que a página foi visitada, conforme ilustrado abaixo.

Exemplo de Entrada

```
http://www.google.com 3
http://www.uol.com.br 5
http://aeds.dcc.ufmg.br 6
http://www.dcc.ufmg.br 1
http://www.dcc.ufmg.br/cursos 2
```

Saída

O arquivo de saída apresenta exatamente o mesmo formato do arquivo de entrada. Porém, as entidades devem estar ordenadas em ordem decrescente do número de visitas (em caso de empate, a ordenação deve ser feita em ordem alfabética de acordo com o nome da página). Note que toda linha termina com uma quebra de linha (\n) e que não há espaços extras antes ou depois de cada entidade.

Exemplo de Saída

```
http://aeds.dcc.ufmg.br 6
http://www.uol.com.br 5
http://www.google.com 3
http://www.dcc.ufmg.br/cursos 2
http://www.dcc.ufmg.br 1
```

Desafios (2 pontos extras)

1. Implemente os algoritmos mergesort e heapsort para a geração das rodadas e analise comparativamente o desempenho computacional e a localidade de referência dos três algoritmos. (1 ponto extra)
2. Implemente versões não recursivas dos algoritmos quicksort e mergesort para a geração de rodadas e analise comparativamente o desempenho computacional e a localidade de referência das duas versões dos dois algoritmos. (1 ponto extra)

3. Entregáveis

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é **terminantemente vetado**. Caso seja necessário, use as estruturas que **você** implementou nos Trabalhos Práticos anteriores para criar **suas próprias implementações** para todas as classes, estruturas, e algoritmos.

Você **DEVE utilizar** a estrutura de projeto abaixo junto ao *Makefile* :

```
- TP
   |- src
   |- bin
   |- obj
   |- include
   Makefile
```

A pasta **TP** é a raiz do projeto; a pasta **bin** deve estar vazia; src deve armazenar arquivos de código (*.c, *.cpp ou *.cc); a pasta include, os cabeçalhos (*headers*) do projeto, com extensão *.h, por fim a pasta **obj** deve estar vazia. O **Makefile** deve estar na **raiz do projeto**. A execução do **Makefile** deve gerar os códigos objeto *.o no diretório **obj,** e o executável do TP no diretório **bin**.

3. 1 Documentação

A documentação do trabalho deve ser entregue em formato **pdf**. A documentação deve conter os itens descritos a seguir :

Título, nome, e matrícula.
Introdução: Contém a apresentação do contexto, problema, e qual solução será empregada.

Método: Descrição da implementação, detalhando as estruturas de dados, tipos abstratos de dados (ou classes) e funções (ou métodos) implementados.

Análise de Complexidade: Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.

Análise Experimental: A análise experimental deve considerar separadamente e em conjunto as fases do algoritmo (geração de rodadas e intercalação). Em cada caso, a análise experimental pode ser dividida em duas partes.

- **Desempenho Computacional**

  Nesta parte você deve realizar uma avaliação de desempenho computacional do seu algoritmo de ordenação. Escolha ou gere pelo menos 10 arquivos de tamanho variado e monte uma tabela com o tempo necessário para a ordenação. Além disso, para cada um destes arquivos apresente o tempo que foi gasto nas operações e a configuração da máquina usada. Preferencialmente apresente a medida de tempo como uma média de várias (pelo menos 10) execuções para o mesmo arquivo para que este seja um valor mais confiável. Compare os resultados obtidos com a análise de complexidade teórica (o que acontece com o tempo quando a entrada dobra de tamanho?). A sua análise deve distinguir entre as duas fases do algoritmo de ordenação. Uma boa análise de resultados deve cobrir no mínimo os seguintes pontos:
  - Descrição e justificativa das métricas relevantes para a avaliação de seu algoritmo (no caso de ordenação, comparações e movimentações).
  - Descrição dos parâmetros relevantes para a execução do seu algoritmo. Neste ponto é importante também avaliar a significância dos dados usados como entrada para seu algoritmo (se a métrica pode variar a cada execução para o mesmo conjunto de parâmetros é bom usar média de repetições). No caso de ordenação, os parâmetros de análise são o número de elementos a serem ordenados, a organização inicial do vetor (já ordenado, aleatório, inversamente ordenado, etc), o número de registros por rodada e número de fitas na intercalação.
  - Gráficos e/ou tabelas que apresentem o resultado de seu algoritmo tendo em vista as métricas definidas anteriormente. Por exemplo, no caso de ordenação, vamos considerar o que acontece em vetores aleatoriamente ordenados quando o número de elementos do vetor aumenta. Se utilizarmos o método bolha, a curva gerada no gráfico representará uma função quadrática, uma vez que a ordem de complexidade do algoritmo é quadrática. Assim, uma análise de resultados bem feita pode confirmar sua análise de complexidade.
  - Avaliação e explicação dos resultados obtidos, explicando possíveis desvios, evidenciando padrões e outras características relevantes ao problema.

- **Análise de Padrão de Acesso à Memória e Localidade de Referência**
  Avalie o padrão de acesso à memória em cada fase do algoritmo através de gráficos de dispersão (acesso X endereço de memória), relacionando as características do algoritmo ao padrão observado. Avalie a distância de pilha exibida nas fases do algoritmo através de histogramas de distância de pilha e explique os fenômenos observados.

Conclusões: A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.

Bibliografia: Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.

Instruções para compilação e execução: Esta seção deve ser colocada em um apêndice ao fim do documento e em uma página exclusiva (não divide página com outras seções).

3.2 Submissão

Todos os arquivos relacionados ao trabalho devem ser submetidos na atividade designada para tal no Moodle. A entrega deve ser feita **em um único arquivo** com extensão **.zip**, com nomenclatura nome_sobrenome_matricula.zip}, onde nome, sobrenome e matrícula devem ser substituídos por suas informações pessoais. O arquivo **.zip** deve conter a sua documentação no formato **.pdf** e a estrutura de projeto descrita no início da Seção 3.

4. Avaliação

O trabalho será avaliado de acordo com:

- A Corretude na execução dos casos de teste - (50% da nota total)
- Apresentação da análise de complexidade das implementações - (15% da nota total)
- Estrutura e conteúdo exigidos para a documentação - (20% da nota total)
- Indentação, comentários do código fonte e uso de boas práticas - (10% da nota total)
- Cumprimento total da especificação - (5% da nota total)

Se o programa submetido não compilar[3], seu trabalho não será avaliado e sua nota será 0. Trabalhos entregues com atrasos sofrerão penalização de 2^{d-1} pontos, com d = dias de atraso.

5. Considerações Finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega está tão distante quanto jamais estará.
2. Leia **atentamente** o documento de especificação, pois o descumprimento de quaisquer requisitos

[3] Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.

obrigatórios aqui descritos causará penalizações na nota final.

3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
4. Plágio é CRIME. Trabalho onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas (em relação a todos os envolvidos). Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na sessão de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos**.

FaQ (Frequently asked Questions)

1. **Posso utilizar alguma estrutura de dados do tipo Queue, Stack, Vector, List, e etc..., do C++? NÃO**
2. Posso utilizar o tipo String? SIM.
3. Posso utilizar o tipo String para simular minhas estruturas de dados? NÃO
4. Posso utilizar alguma biblioteca para tratar exceções? SIM.
5. Posso utilizar alguma biblioteca para gerenciar memória? SIM.
6. As análises e apresentação dos resultados são importantes na documentação? SIM.
7. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? NÃO
8. Posso fazer o trabalho em dupla ou em grupo? NÃO
9. Posso trocar informações com os colegas sobre a teoria? SIM.
10. Posso fazer o trabalho no Windows, Linux, ou MacOS? SIM.
11. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? SIM.

Referências

Cormen , T., Leiserson, C., Rivest, R., Stein, C. Introduction to Algorithms. MIT Press, 2001.