Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
DCC205 - Estrutura de Dados

Trabalho Prático 2

Prazo de Entrega: 08/11/2022
Valor: 15 pontos

1. Objetivo

Este trabalho consiste em analisar o desempenho de diferentes algoritmos de ordenação em diferentes cenários, descritos a seguir. Esta análise consiste em comparar os algoritmos considerando três métricas de desempenho: número de comparações de chaves, o número de cópias de registros realizadas, e o tempo total gasto para ordenação (tempo de processamento e não o tempo de relógio). As entradas são conjuntos de elementos com chaves aleatoriamente geradas. Para obter o tempo de processamento na linguagem C, você pode utilizar o comando `getrusage`, lembrando de somar os tempos gastos em modo de usuário (`utime` ou user time) e em modo de sistema (`stime` ou system time) (vide exemplo no final do enunciado). O trabalho é dividido em duas partes, indicadas a seguir.

2. Descrição

2.1 Parte 1: Impacto de variações do Quicksort

Neste cenário, você deverá comparar o desempenho de diferentes variações do Quicksort para ordenar um conjunto de N registros armazenados em um vetor. Cada registro contém:

- Um inteiro, que é a chave para ordenação
- Quinze cadeias de caracteres (strings), cada uma com 200 caracteres
- 10 números reais

As variações do Quicksort a serem implementadas e avaliadas são:

1. Quicksort Recursivo: este é o Quicksort recursivo apresentado em sala de aula.
2. Quicksort Mediana(k): esta variação do Quicksort recursivo escolhe o pivô para partição como sendo a mediana de k elementos do vetor, aleatoriamente escolhidos. Experimente com k = 3, k = 5, e k = 7.
3. Quicksort Seleção(m): esta variação modifica o Quicksort Recursivo para utilizar o algoritmo de Seleção para ordenar partições (isto é, pedaços do vetor) com tamanho menor ou igual a m. Experimente com m = 10 e m = 100.
4. Quicksort não Recursivo: esta variação escolhe o pivô como o elemento do meio (como apresentado em sala de aula), mas não é recursiva, utilizando uma pilha para armazenar partições a serem processadas posteriormente.

5. Quicksort Empilha Inteligente: esta variação do Quicksort processa primeiro a menor partição. Você deve aplicar esta otimização à versão não recursiva do Quicksort.

Você deverá instrumentar os algoritmos para contabilizar o número de comparações de chaves, o número de cópias de registros e o tempo total (tempo de processamento) gasto na ordenação. Estes números deverão ser impressos ao final de cada ordenação para posterior análise. Você ainda deverá implementar funções para criação dos conjuntos de elementos aleatórios. Estas funções devem ser chamadas uma vez para cada um dos N elementos a serem ordenados. Note que dois elementos podem ter a mesma chave.

Cada variação do algoritmo Quicksort deverá ser aplicada a entradas aleatórias com diferentes tamanhos (parâmetro N). Experimente, no mínimo, com valores de N = 1000, 5000, 10000, 50000, 100000, 500000 e 1000000. Para cada valor de N, você deverá gerar 5 (cinco) conjuntos de elementos diferentes, utilizando sementes diferentes para o gerador de números aleatórios. Faz parte do trabalho descobrir como gerar números aleatórios a partir de uma semente.

Os algoritmos serão avaliados comparando os valores médios das 5 execuções (um conjunto diferente por execução) para cada uma das três métricas consideradas (tempo de execução, número de comparações de chaves e número de cópias de registros) para cada valor de N testado.

O seu programa principal deve ser organizado da seguinte maneira:

1. Recebe a semente do gerador de números aleatórios bem como os demais parâmetros, incluindo os nomes dos arquivos de entrada e de saída. Estes parâmetros devem ser passados pela linha de comando (argc e argv em C). Assim:
    a. Quicksort Recursivo: quicksort -v 1 -s 10 -i entrada.txt -o saida10.txt
    b. Quicksort Mediana (k): quicksort -v 2 -k 3 -s 10 -i entrada.txt -o saida10.txt
    c. Quicksort Seleção (m): quicksort -v 3 -m 100 -s 10 -i entrada.txt -o saida10.txt
    d. Quicksort não Recursivo: quicksort -v 4 -s 10 -i entrada.txt -o saida10.txt
    e. Quicksort Empilha Inteligente: quicksort -v 5 -s 10 -i entrada.txt -o saida10.txt

    Em todos os casos, entrada.txt tem o formato:

    7 -> número de valores de N que se seguem, um por linha
    1000
    5000
    10000
    50000
    100000
    500000
    1000000

2. Para cada valor de N, lido do arquivo de entrada:

a. Gera um conjunto de N elementos aleatoriamente (utilizando a semente passada, no caso 10).
b. Ordena o conjunto, contabilizando as estatísticas de desempenho
c. Armazena estatísticas de desempenho em arquivo de saída

3. Ao final, basta processar os arquivos de saída referentes a cada uma das sementes, calculando as médias de cada estatística, para cada valor de N e algoritmo considerado.

Avaliação Experimental

Com relação ao desempenho computacional, apresente gráficos e/ou tabelas para as três métricas pedidas, número de comparações, número de cópias e tempo de execução (tempo de processamento), comparando o desempenho médio de cada variação do Quicksort para diferentes valores de N . Discuta os resultados e conclusões obtidos. Qual variação tem melhor desempenho, considerando as diferentes métricas. Por quê? Qual o impacto das variações nos valores de k e de m nas versões Quicksort Mediana(k) e Quicksort Selecao(m)?

Com relação a localidade de referência, caracterize e compare os padrões de acesso à memória das 5 variações de algoritmos para um tamanho pequeno, mas suficiente para entender as diferenças entre eles.

2.2 Parte 2: Quicksort X Mergesort X Heapsort

Na segunda parte do trabalho, você vai comparar a melhor variação do Quicksort (justificada pelos resultados do Cenário 2) com o Mergesort e o Heapsort, para ordenar um conjunto de N registros (conforme definido acima), aleatoriamente gerados, armazenados em um vetor.

Realize experimentos considerando vetores aleatoriamente gerados com tamanho N =1000, 5000, 10000, 50000, 100000, 500000, 1000000, no mínimo. Para cada valor de N, realize experimentos com 5 sementes diferentes. Para esta comparação, avalie os valores médios do tempo de execução, do número de comparações de chaves e do número de cópias de registros. Apresente gráficos e/ou tabelas com os resultados obtidos. Discuta os resultados e conclusões obtidas. Qual algoritmo tem melhor desempenho, considerando as diferentes métricas? Por quê?

Note que o grande desafio deste trabalho está na avaliação dos vários algoritmos nos diferentes cenários, e não na implementação de código.

Uma boa documentação deverá apresentar não somente resultados brutos, mas também uma discussão dos mesmos, levando a conclusões sobre a superioridade de um ou outro algoritmo em cada cenário considerado, para cada métrica avaliada.

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 de dados, 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.
Estratégias de Robustez: Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
Análise Experimental: Apresenta os experimentos realizados em termos de desempenho computacional e localidade de referência, assim como as análises dos resultados.
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).

***Número máximo de páginas:** 20

A documentação deve conter a descrição do seu trabalho em termos funcionais, dando foco nos algoritmos, estruturas de dados e decisões de implementação importantes durante o desenvolvimento.

Evite a descrição literal do código-fonte na documentação do trabalho.

Dica: Sua documentação deve ser clara o suficiente para que uma pessoa (da área de Computação ou não) consiga ler, entender o problema tratado e como foi feita a solução.

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 - (30% da nota total)
- Apresentação da análise de complexidade das implementações - (10% da nota total)
- Estrutura e conteúdo exigidos para a documentação - (45% da nota total)
    - Observe que as análises realizadas a partir da execução dos casos de teste, ilustradas por gráficos e tabelas e discutidas no texto de acordo com a análise de complexidade realizada, são cruciais neste trabalho.
- Indentação, comentários do código fonte e uso de boas práticas - (5% da nota total)
- Cumprimento total da especificação - (10% da nota total)

Se o programa submetido não compilar[1], 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.

[1] 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.

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 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.

Anexo: Medindo o tempo de execução de uma função em C

O comando `getrusage()` é parte da biblioteca padrão de C da maioria dos sistemas Unix. Ele retorna os recursos correntemente utilizados pelo processo, em particular os tempos de processamento (tempo de CPU) em modo de usuário e em modo sistema, fornecendo valores com granularidades de segundos e microsegundos. Um exemplo que calcula o tempo total gasto na execução de uma tarefa é mostrado abaixo:

```
#include <stdio.h>
#include <sys/resource.h>
void main () {
  struct rusage resources;
  int rc;
  double utime, stime, total_time;
  /* do some work here */
  if((rc = getrusage(RUSAGE_SELF, &resources)) != 0)
    perror("getrusage failed");
  utime = (double) resources.ru_utime.tv_sec
          + 1.e-6 * (double) resources.ru_utime.tv_usec;
  stime = (double) resources.ru_stime.tv_sec
          + 1.e-6 * (double) resources.ru_stime.tv_usec;
  total_time = utime+stime;
  printf("User time %.3f, System time %.3f, Total Time %.3f\n",
         utime, stime, total_time);
}
```