Estrutura de Dados

Prof. Frederico S. Oliveira - Prazo final 11/03/2016

Segundo Semestre de 2016

Segundo Trabalho Prático

1 Introdução

Este trabalho consiste em analisar o desempenho de diferentes algoritmos de ordenação em diferentes cenários, descritos a seguir. Esta análise consistirá 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 deverão ser conjuntos de elementos com chaves aleatoriamente geradas. Para obter o tempo de processamento na linguagem C, você pode utilizar o comando clock(), conforme apresentado em sala de aula.

2 Cenário 1: Impacto de diferentes estruturas de dados

Neste cenário, você deverá avaliar o desempenho dos métodos de ordenação por comparação apresentados em sala, considerando as seguintes entradas:

- Os elementos a serem ordenados são inteiros armazenados em um vetor de tamanho n.
- Os elementos a serem ordenados são inteiros armazenados em uma lista duplamente encadeada com n elementos.
- Os elementos a serem ordenados são registros armazenados em um vetor de tamanho n. Cada registro contém:
  - Um inteiro, a chave para ordenação.
  - Dez cadeias de caracteres (strings), cada uma como 200 caracteres.
  - 1 booleano
  - 4 números reais.

Para cada tipo de dado, você deverá implementar os algoritmos de ordenação por comparação (apresentados em sala de aula) que recebe, como entrada, o conjunto de elementos a serem ordenados (uma Lista ou um vetor) e o número de elementos a serem ordenados. Você deverá instrumentar os algoritmos para contabilizar o número de comparações de chaves, o número de cópias de registros e o tempo total gasto na ordenação. Estes números deverão ser impressos ao final de cada ordenação para posterior análise.

Os algoritmos por comparação a serem analisados são: *bubblesort, insertionsort, selectionsort, shellsort, mergesort, heapsort e quicksort.*

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. Para o caso dos elementos

serem registros, a função de criação deve inicializar todos os campos do registro com valores aleatórios. Note que dois elementos podem ter a mesma chave.

Caso o teste de algum dos métodos de ordenação torne-se inviável, por algum dado motivo qualquer, como por exemplo o valor de n ser muito grande, ou ser impossível aplicar o algoritmo na estrutura de dados, informe nos resultados o motivo da inviabilidade.

2.1 Análise

Os algoritmos deverão ser aplicados a entradas aleatórias com diferentes tamanhos (parâmetro n). Para cada valor de n, você deve gerar 5 (cinco) conjuntos de elementos diferentes, utilizando sementes diferentes para o gerador de números aleatórios. Experimente, no mínimo, com valores de n = 1000, 5000, 10000, 50000, 100000, 500000 e 1000000. Os algoritmos serão avaliados comparando os valores médios das 5 execuções para cada valor de n testado. O seu programa principal deve ser organizado da seguinte maneira:

- Recebe os nomes dos arquivos de entrada e de saída. Estes parâmetros devem ser passados pela linha de comando (argc e argv em C). Por exemplo:

  ```
  quicksort  entrada.txt saida10.txt
  ```

  onde entrada.txt tem o formato:

  ```
  7
  1000
  5000
  10000
  50000
  100000
  500000
  1000000
  ```

  em que, o primeiro valor (7 no exemplo), indica o número de valores de n que seguem, um por linha.

- Para cada valor de n, lido do arquivo de entrada:
  - Gera cada um dos conjuntos de elementos, ordena, contabiliza estatísticas de desempenho
  - Armazena estatísticas de desempenho em arquivo de saída

2.2 Resultados

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 dos algoritmos de ordenação por comparação para os três tipos de estruturas de dados e diferentes valores de n. Discuta seus resultados. Quais são as conclusões em relação as análises de desempenho observadas?

3 Cenário 2: Impacto de variações do Quicksort

Neste cenário, você deverá comparar o desempenho de diferentes variações do Quicksort para ordenar um conjunto de n inteiros armazenados em um vetor. As variações do Quicksort a serem implementadas e avaliadas são:

i) Quicksort Recursivo: este é o Quicksort recursivo apresentado em sala de aula.

ii) Quicksort Mediana: esta variação do Quicksort recursivo (i) escolhe o pivô para partição como sendo a mediana de k elementos do vetor, aleatoriamente escolhidos. Experimente com $k = 3$ e $k = 5$.

iii) Quicksort Insercao: esta variação modifica o Quicksort Recursivo (i) para utilizar o algoritmo de Inserção para ordenar partições (isto é, pedaços do vetor) com tamanho menor ou igual a m. Experimente com $m = 10$ e $m = 100$.

iv) Quicksort Empilha Inteligente: esta variação otimizada do Quicksort Recursivo (i) processa primeiro o lado menor da partição.

v) Quicksort Iterativo: esta variação escolhe o pivô como o elemento do meio (como apresentado em sala de aula), mas não é recursiva. Em outras palavras, esta é uma versão iterativa do Quicksort Recursivo(i), apresentado em sala de aula.

vi) Quicksort Empilha Inteligente: esta variação otimizada do Quicksort Iterativo (v) processa primeiro o lado menor da partição.

Realize experimentos com as cinco variações considerando vetores aleatoriamente gerados com tamanho n = 1000, 5000, 10000, 50000, 100000, 500000 e 1000000, no mínimo. Para cada valor de n, realize experimentos com 5 sementes diferentes e avalie os valores médios do tempo de execução, do número de comparações de chaves e do número de cópias de registros. Estruture o seu programa principal como sugerido acima para facilitar a coleta e posterior análise das estatísticas de desempenho.

Apresente gráficos e/ou tabelas com os resultados obtidos. 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 (ii) e Quicksort Insercao (iii)?

4 Cenário 3: Quicksort X Mergesort X Heapsort X Countingsort X Meu-sort

Neste cenário, você comparará a melhor variação do Quicksort (justificada pelos resultados do Cenário 2) com o Mergesort, o Heapsort, o Countingsort, e um outro algoritmo de ordenação de sua escolha (não pode ser nenhum algoritmo dado em sala de aula) para ordenar um conjunto de n inteiros positivos, aleatoriamente gerados, armazenados em um vetor. Todos os algoritmos implementados devem constar no relatório, detalhando-os por meio de código ou pseudocódigo. Você deverá também apresentar no seu relatório o quinto algoritmo escolhido, explicitando sua fonte.

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 a comparação do Quicksort, Mergesort, Heapsort e Meusort, 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. Logo, na divisão de pontos, a documentação receberá, no mínimo, 50% dos pontos totais. 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.

5 Documentação

Obrigatoriamente a documentação deve conter:

1. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.

2. Desenvolvimento: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (preferencialmente com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que por ventura estejam omissos no enunciado.
3. Compilação e Execução: descrição de como como compilar o código-fonte, bem como a sua execução.
4. Testes: descrição de quais testes foram executados e as saídas obtidas.
5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo livros e/ou sites da internet se for o caso, seguindo padrão ABNT.

6 Material a ser Entregue

Os seguintes itens devem ser enviados via Ambiente Virtual de Aprendizagem (AVA), em um único arquivo compactado, com extensão ".tar.gz":

- Código fonte do programa em C, bem identado e comentado. O código-fonte, bem como os testes de sua execução, valem 50% da nota.
- Documentação do trabalho, que deve ser entregue mandatoriamente no formato PDF. A documentação vale 50% da nota.
- Caso também seja entregue a documentação em LaTex, ganha-se 1 ponto, em um total de 10 pontos, não acumulativo.

7 Conclusão

Para cálculo da nota, serão realizados testes, e a nota será calculada com base nestes testes. Será atribuído zero aos seguintes casos:

- Código-fonte não compilar no Linux.
- Nenhum teste for executado com sucesso.
- Não apresentar documentação.
- Plágio.

Este trabalho deve ser desenvolvido de forma individual ou em dupla. Não serão realizadas exceções. Bom trabalho.