UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados II

Trabalho Prático 2 - Ordenação

Descrição

Neste laboratório devemos implementar e avaliar algoritmos de ordenação.

Objetivo

O objetivo deste trabalho é implementar os seguintes algoritmos de ordenação: inserção, bolha, seleção, mergesort e quicksort. Além disso, utilizando um programa onde seja possível escolher o algoritmo utilizado para ordenar uma sequência de números, poderemos avaliar o tempo de computação necessário para ordenar a mesma sequência utilizando diferentes algoritmos.

Especificação

Nesta aula prática você deve implementar em C as seguintes funções:

- `void insertionSort(int n, int v[])`: a ordenação por inserção constrói o arranjo final ordenado como se inserisse um elemento do arranjo original por vez.

  Entrada:
  `n` – tamanho do arranjo `v`.
  `v` – arranjo de entrada desordenado.

  Saída:
  arranjo `v` ordenado.

- `void bubbleSort(int n, int v[])`: a ordenação por bolha percorre o arranjo diversas vezes, fazendo flutuar para o final o maior elemento da sequência.

  Entrada:
  `n` – tamanho do arranjo `v`.
  `v` – arranjo de entrada desordenado.

  Saída:
  arranjo `v` ordenado.

- `void selectionSort(int n, int v[])`: é um algoritmo de ordenação baseado em se passar sempre o menor valor do arranjo para a primeira posição, depois o de segundo menor valor para a segunda posição, e assim sucessivamente.

  Entrada:
  `n` – tamanho do arranjo `v`.
  `v` – arranjo de entrada desordenado.

Saída:
arranjo `v` ordenado.

- `void mergeSort(int c, int f, int v[])`: algoritmo de ordenação que utiliza a técnica *Dividir para Conquistar*. A ideia básica consiste em dividir o problema em sub-problemas, resolver os sub-problemas através da recursividade e conquistar (após todos os sub-problemas terem sido resolvidos ocorre a conquista, que é a união das soluções dos sub-problemas). A função `mergeSort` é a função de divisão em sub-problemas e a função `merge` é a função de conquistar, que faz a combinação das duas sub-soluções.

  `void mergeSort(int c, int f, int v[])`
  Entrada:
  `c` – índice do começo do (sub)arranjo no arranjo `v`.
  `f` – índice do fim do (sub)arranjo no arranjo `v`.
  `v` – arranjo de entrada desordenado entre os índices `c` e `f`.

  Saída:
  arranjo `v` ordenado entre os índices `c` e `f`.

  `void merge(int c, int m, int f, int v[])`
  Entrada: dois arranjos ordenados separadamente e que devem ser combinados em um maior arranjo ordenado.
  `c` – índice do começo do primeiro (sub)arranjo no arranjo `v`.
  `m` – índice do fim do primeiro (sub)arranjo no arranjo `v`.
  `f` – índice do fim do segundo (sub)arranjo no arranjo `v`.
  `v` – dois arranjos de entrada ordenados. O primeiro está entre os índices `c` e `m` e o segundo entre `m+1` e `f`.

  Saída:
  arranjo `v` ordenado de `c` a `f`.

- `void quickSort(int c, int f, int v[])`: algoritmo de ordenação que também utiliza a técnica *Dividir para Conquistar*. A estratégia consiste em rearranjar os elementos de modo que os menores precedam os maiores. Em seguida os sub-arranjos são ordenados com elementos menores e maiores recursivamente até que o arranjo completo esteja ordenado. Um ponto chave para o algoritmo é o procedimento `partition`, que reorganiza o sub-arranjo `v[c..f]` localmente, colocando à direita do pivô os elementos maiores a ele e à sua esquerda os elementos maiores que ele, retornando o pivô ao final.

  `int partition(int c, int f, int v[])`
  Entrada:
  `c` – índice do começo do (sub)arranjo no arranjo `v`.
  `f` – índice do fim do (sub)arranjo no arranjo `v`.
  `v` – arranjo de entrada desordenado entre os índices `c` e `f`.

  Saída:
  retorna o pivô e, além disso, modifica o (sub)arranjo `v` de tal forma que os elemenos à direita do pivô são maiores que ele e os elementos à esquerda do pivô são menores que ele.

`void quickSort(int c, int f, int v[])`
Entrada:.
`c` – índice do começo do primeiro (sub)arranjo no arranjo `v`.
`f` – índice do fim do (sub)arranjo no arranjo `v`.
`v` – arranjo de entrada ordenado de `c` a `f`.

Saída:
arranjo `v` ordenado de `c` a `f`.

A ordenação deve ser crescente para todos os algoritmos.

Material

No arquivo compactado **TP2.zip** são fornecidos:

- **main.c**: implementação completa da função `main` e funções auxiliares para controlar a entrada dos dados, a chamada do algoritmo de ordenação baseado no desejo do usuário e a impressão da saída. Com o programa compilado, devemos ser capazes de avaliar o desempenho de cada um dos algoritmos implementados.
- **ordenacao.h**: *header* base para a implementação de seus algoritmos de ordenação.
- **ordenacao.c**: arquivo com o esqueleto que deve ser completado pelo aluno. **Esse é o único arquivo que será entregue na sua submissão**
- **Makefile**: o `Makefile` é um arquivo que auxilia a compilação do seu programa (em ambientes Linux). Basta colocá-lo na mesma pasta do arquivo `main.c` e usar o comando `make` que o executável `main` será gerado.
- **testes**: diretório contendo casos de teste. Os arquivos test#.in podem ser utilizados para simular uma entrada via teclado utilizando-se o caracter $<$ [1]. Já os arquivos test#.res são as respectivas saídas esperadas para a execução de cada caso de teste. Dessa forma você pode utilizar os arquivos de testes para verificar se seu programa responde da mesma forma esperada pelo software de correção.

Entrada

Como mencionado anteriormente, o código para ler a entrada do usuário, chamar a rotina de ordenação adequada e produzir a saída do seu programa é inteiramente fornecido no arquivo **main.c**. O programa lê da entrada padrão um `char` que indica qual algoritmo de ordenação deve ser utilizado: **i**nserção, **b**olha, **s**eleção, **m**ergesort ou **q**uicksort; seguido pelo tamanho n do arranjo de entrada com $0 < n < 100000$; e, finalmente, pelos elementos do arranjo, separados por um espaço em branco, obedecendo, ainda, a restrição de que $-100000 < v_i < 100000$.

Saída

A saída deve conter os n valores da sequência separados por um espaço, na ordem em que foram inseridos. Após o último elemento deve-se pular uma linha. Em seguida deve-se imprimir o nome do algoritmo de ordenação utilizado seguido por uma quebra de linha e, finalmente, a sequência ordenada, com elementos separados por um espaço em branco e terminada por uma quebra de linha. A Figura 1 mostra um exemplo completo, desde a compilação à execução do programa utilizando dois arquivos de teste e usando entrada manual.

Para verificar se a saída do seu programa está exatamente igual ao caso de teste enviado, você pode utilizar o utilitário `diff`. Por exemplo, caso de sucesso:

[1] Para mais informações acesse `http://linuxcommand.org/lc3_lts0070.php`

Figura 1: Exemplo de execução do programa em uma máquina virtual Linux.

```
$ ./main < testes/test10i.in | diff -E -Z -b -B testes/test10i.res -
```

e caso de erro:

```
$ ./main < testes/test10i.in | diff -E -Z -b -B testes/test10q.res -
2c2
< QUICK
---
> INSERTION
```

Entrega

Você deve entregar um arquivo **zip** nomeado com sua matrícula e contendo os arquivos fonte para execução do seu programa. No caso desta aula prática, **apenas o arquivo ordenacao.c**. Por exemplo, o aluno cuja de matrícula é 201700000001, deve enviar o arquivo *201700000001.zip*, contendo seu arquivo solução **ordenacao.c**.
Seu programa deverá ser possível de ser compilado no Linux, portanto ele não deverá conter nenhuma biblioteca que seja específica do sistema operacional Windows.
Além disso, você deve utilizar o endereço indicado na descrição do trabalho prático no Moodle para acessar e preencher um formulário com questões sobre o trabalho. Relacionado às questões, uma dica para avaliar o tempo de execução de um programa é o comando `time` do Linux, onde você deve atentar para o valor *user* informado, conforme mostrado na Figura 2 (para melhorar a visualização omitiu-se a saída impressa do programa, redirecionando-a para `/dev/null` e manteve-se apenas a saída do comando `time`).

Outras Instruções

A data para conclusão da aula prática está na descrição da aula no Moodle. Eventuais dúvidas deverão ser postadas no fórum do Moodle aberto para esta tarefa e serão respondidas por lá, ou nos horários de atendimento de monitoria.

Os exercícios das aulas práticas serão corrigidos de forma automática, tenha certeza de que a saída produzida por seu programa seja a mesma esperada pelo software de correção. Você pode se basear nos casos de teste enviados para comparar sua resposta com a resposta esperada. Além dos casos de testes enviados, serão executados outros testes.

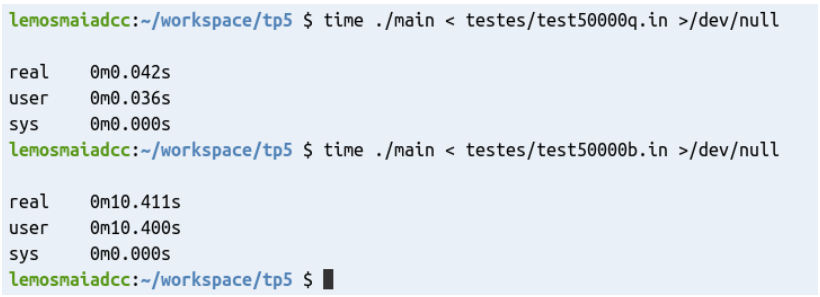

Figura 2: Exemplo de avaliação de tempo de execução usando o comando `time`.

É altamente aconselhável que os alunos troquem experiências e conversem sobre os exercícios práticos, contudo as atividades, a menos que informado explicitamente, são individuais e eventuais casos de plágio serão punidos. Para tal, o software de correção utilizado é equipado com algoritmos de detecção de plágio, portanto, implementem suas próprias soluções.

Bom Trabalho!