Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: Projeto e Análise de Algoritmos
Código da Disciplina: 117536

Trabalho 2

O objetivo deste trabalho é implementar em C alguns algoritmos de ordenação.

1) Toda a interação com os programas deve ser feita através da entrada e saída padrão (stdin, stdout). Quando a entrada for feita através de arquivos, deve ser utilizado o redirecionamento.

Ex:

entrada.txt arquivo que contem as strings de entrada
saida.txt arquivo que contem as strings ordenadas

Utilização:

```
programa.exe  < entrada.txt > saida.txt
```

Neste caso, programa.exe recebe entrada.txt na entrada padrão através do '<' e a saída padrão de programa.exe é colocada em saida.txt através do '>'

O arquivo entrada.txt deve ser um arquivo texto composto de strings. Ele contem as strings a serem ordenadas.

Uma string é composta de um ou mais caracteres alfabéticos e espaço. Cada string pode ter no máximo 100 caracteres. Todos os caracteres são minúsculos. Existe apenas uma string por linha. Linhas sem pelo menos um caractere alfabético são ignoradas.

Ex:

abc
acb
baa

O arquivo saída.txt deve ser um arquivo texto composto de strings (mesmo formato de entrada.txt). Contem a saída do algoritmo, ou seja, as mesmas strings arquivo entrada.txt

ordenadas, uma por linha. A ordenação deve ser a ordem alfabética de acordo com a tabela ASCII.

1.1) Deve ser feito um programa para geração de strings (que depois vão ser ordenadas).

Deve ter as seguintes opções

gera_strings N ordem > entrada.txt

onde

N é o número de strings que serão geradas
Ordem pode ter três valores **norm, inver e aleat**.

Se for **norm** então as strings estarão em ordem crescente (sem repetições).
Se for **inver** as strings estarão em ordem decrescente (sem repetições).
Se for **aleat** as strings são geradas em ordem aleatória (sem repetições).

entrada.txt é o arquivo com as strings que será utilizado como entrada para os algoritmos de ordenação

Exemplos: gera_strings 5 norm > entrada.txt

entrada.txt:

aaa
aab
aac
aad
aae

gera_strings 5 inver > entrada.txt

entrada.txt:

aae
aad
aac
aab
aaa

gera_strings 5 aleat > entrada.txt

entrada.txt:

aad
aaa
aac
aae
aab

2) Implementar o algoritmo de ordenação Selection sort

O algoritmo deve ter uma linha de comando

Selection_sort < entrada.txt > saída.txt

Onde entrada.txt e saída.txt são do mesmo formato do item 1)

3) Algoritmo de ordenação Shellsort

O algoritmo deve ter uma linha de comando

Shell_sort < entrada.txt > saída.txt

Onde entrada.txt e saída.txt são do mesmo formato do item 1)

4) Algoritmo de ordenação Combsort

O algoritmo deve ter uma linha de comando

Comb_sort < entrada.txt > saída.txt

Onde entrada.txt e saída.txt são do mesmo formato do item 1)

5) Algoritmo de ordenação Merge_sort

Deve ter alinha de comando

Merge_sort < entrada.txt > saída.txt

6) Algoritmo de ordenação Insertion_sort

O algoritmo deve ter uma linha de comando

insertion_sort < entrada.txt > saída.txt

Onde entrada.txt e saída.txt são do mesmo formato do item 1)

6.1) Um programa separado do insertionsort, chamado conta_inversão, deve gravar para o vetor de strings que está sendo ordenado o número de inversões que o vetor contém. Uma inversão é se i > j e A[i] < A [j] (A[i] acontece antes de A[j] na ordem alfabética). O programa deve contar todos os pares (i,j) que são inversões.

7) Deve ser feito um programa com o algoritmo Bubblesort

O algoritmo deve ter uma linha de comando

bubble_sort < entrada.txt > saída.txt

Onde entrada.txt e saída.txt são do mesmo formato do item 1)

8) Deve ser feito um programa merge_insert que é uma mistura entre o mergesort e o insertionsort. O algoritmo executa como o mergesort. A diferença é que se o tamanho s do subvetor for s<=k então o subvetor é ordenado usando o insertionsort. O valor k é um parâmetro do algoritmo.

O algoritmo deve ter uma linha de comando

merge_insert < entrada.txt > saída.txt

Onde entrada.txt e saída.txt são do mesmo formato do item 1)

9) A seguir deve ser feita uma análise comparativa de todos dos algoritmos de ordenação. Cada um deles deve ter uma média de 10 execuções.

(Usar um script com o comando time do Unix para pegar o tempo real e fazer *append* em um arquivo de log pode ajudar bastante)

Para o caso do merge_insert, todos os casos devem ser rodados para k=4, k=8, k=32 e k=64.

10) Alguns algoritmos exigem que você escolha alguns parâmetros adicionais. Neste caso, você é livre para escolher parâmetros os necessários.

A seguir faça os seguintes itens:

a) Os algoritmos devem ter a média de seus tempos comparados na ordem normal para entradas de:

1-10
1-20
1-40
1-80
1-100
1-200
1-400
1-800
1-1600
1-3200
1-6400
1-12800
1-25600
1-51200
1-102400
1-204800
1-409600
1-819200
1-1638400
1-3276800

(Se o algoritmo for quadrático ele pode ser interrompido se demorar mais de 2 horas no total das execuções)

Deve ser gerada uma tabela com as médias e um gráfico (com as médias) comparando os algoritmos. O eixo **x** deve ser o tamanho da entrada e o **y** o tempo.

b) Os algoritmos devem ter a média de seus tempos comparados na ordem inversa para as mesmas entradas do item a) e gerados a tabela e o gráfico correspondente.

c) Os algoritmos devem ter a média de seus tempos comparados na ordem aleatória para as mesmas entradas do item a) e gerados a tabela e o gráfico correspondente.

d) Qual a relação entre o tempo de execução dos algoritmos e o número de inversões no arquivo a ser ordenado? Justifique sua resposta. Qual a complexidade do programa conta_inversão? Justifique sua resposta. Usando os tempos de execução e o número de inversões, faça um gráfico.

e) Compare o impacto dos valores de k para o merge_insert. Faça um gráfico que mostre a influencia de k no tempo total. Quais valores de k obtiveram os melhores resultados? Justifique.

f) Para os casos de algoritmos (**norm, inver e aleat**) que não puderam rodar devido ao estouro do tempo, estime baseado na complexidade do algoritmo e nos tempos que foram medidos qual seria o tempo necessário para ordenar um vetor de tamanho n=3276800.

g) Para cada um dos casos (**norm, inver e aleat**) e para cada um dos tamanhos de n (10 - 3276800) faça uma tabela que indique qual algoritmo foi mais rápido. Os resultados estão de acordo com a complexidade do algoritmo? Justifique.

Importante:

a) **Deve ser enviado também um arquivo texto dizendo como o programa deve ser compilado no GCC versão 3.4.4 ou superior com um exemplo de compilação utilizando os arquivos enviados bem como um exemplo da compilação e da execução do programa com eventuais parâmetros de linha de comando**
b) **Os programas devem ser implementados de acordo com a especificação, sem modificações**

O(s) arquivo(s) deve(m) ter o seguinte nome: programa_matricula_primeironome.c (Ex: insertion_sort_06_12345_Jose.c) e um 06_12345_Jose.txt. Os arquivos devem ser enviados compactados (.zip ex. 06_12345_Jose.zip).

Data de entrega:

21/ 04/15

Pela tarefa na página da disciplina no ead.unb.br