Trabalho Prático 03 – AEDS 1 – Em TRIO

Professora: Thais R. M. Braga Silva

Valor: 10 pontos

Data de Entrega: 06/11/18

Forma de Entrega: PVANet (formato .zip ou .tar.gz)

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 movimentações de itens, e o tempo total gasto para ordenação (veja como ao final).

Para desenvolver este trabalho, você precisará da implementação feita para o Trabalho Prático 01 (TP1). Utilizando a implementação do TAD MATRIZ_VOOS, crie os cenários apresentados a seguir e, ordene o vetor criado em cada um deles utilizando todos os algoritmos vistos em sala de aula: Bolha, Seleção, Inserção, Shellsort, Quicksort e Heapsort. Você deverá fazer uma pequena alteração no TAD MATRIZ_VOOS, adicionando um campo de identificador. Este campo será utilizado como chave de ordenação.

Cada item do vetor é uma MATRIZ_VOOS. Para cada cenário deve ser possível: (i) escolher aleatoriamente as matrizes a serem preenchidas, bem como preencher os voos com dados aleatórios; (ii) ler de um arquivo de entrada os índices das matrizes a serem preenchidas, bem como os dados dos voos.

- **Cenário 01:** criar um vetor de MATRIZ_VOOS de tamanho 365, preenchendo apenas 20% das matrizes criadas, cada uma delas com 10 voos cada;
- **Cenário 02:** criar um vetor de MATRIZ_VOOS de tamanho 365, preenchendo 100% das matrizes criadas, cada uma delas com 10 voos cada;
- **Cenário 03:** criar um vetor de MATRIZ_VOOS de tamanho 365, preenchendo apenas 20% das matrizes criadas, cada uma delas com 100 voos cada;
- **Cenário 04:** criar um vetor de MATRIZ_VOOS de tamanho 365, preenchendo 100% das matrizes criadas, cada uma delas com 100 voos cada;
- **Cenário 05:** criar um vetor de MATRIZ_VOOS de tamanho 3650, preenchendo apenas 20% das matrizes criadas, cada uma delas com 10 voos cada;

- **Cenário 06:** criar um vetor de MATRIZ_VOOS de tamanho 3650, preenchendo 100% das matrizes criadas, cada uma delas com 10 voos cada;
- **Cenário 07:** criar um vetor de MATRIZ_VOOS de tamanho 3650, preenchendo apenas 20% das matrizes criadas, cada uma delas com 100 voos cada;
- **Cenário 08:** criar um vetor de MATRIZ_VOOS de tamanho 3650, preenchendo 100% das matrizes criadas, cada uma delas com 100 voos cada;
- **Cenário 09:** criar um vetor de MATRIZ_VOOS de tamanho 36500, preenchendo apenas 20% das matrizes criadas, cada uma delas com 10 voos cada;
- **Cenário 10:** criar um vetor de MATRIZ_VOOS de tamanho 36500, preenchendo 100% das matrizes criadas, cada uma delas com 10 voos cada;
- **Cenário 11:** criar um vetor de MATRIZ_VOOS de tamanho 36500, preenchendo apenas 20% das matrizes criadas, cada uma delas com 100 voos cada;
- **Cenário 12:** criar um vetor de MATRIZ_VOOS de tamanho 36500, preenchendo 100% das matrizes criadas, cada uma delas com 100 voos cada;

Após executar os 6 algoritmos de ordenação para um dado cenário, você deverá construir um gráfico de barras para cada métrica solicitada (comparações, movimentações e tempo total), comparando os resultados obtidos. Dessa forma, por exemplo, para o Cenário 1, haverão 3 gráficos, cada um deles mostrando os resultados obtidos para uma determinada métrica quando os 6 algoritmos de ordenação foram aplicados. Assim, será possível visualizar quantas comparações foram realizadas para o vetor deste cenário, quando foram utilizados os algoritmos Bolha, Seleção, Inserção, Shellsort, Quicksort e Heapsort. O mesmo será possível para o número de movimentações e para o tempo total gasto para a ordenação.

Observe que, o mesmo vetor inicial de cada cenário deve ser passado aos 6 algoritmos de ordenação, de modo que a comparação entre as métricas seja justa. Dessa forma, atente-se ao fato de que você não deve passar o vetor já ordenado pelo primeiro algoritmo para os demais, bem como não devem ser vetores diferentes.

O seu programa deve estar preparado para oferecer ao usuário a escolha entre fazer o preenchimento aletatório dos cenários ou utilizar um arquivo para ler os dados de entrada. Um arquivo de entrada de exemplo pode ser encontrado junto com a descrição deste TP. Seu programa deve seguir rigorosamente o formato de entrada proposto.

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 microssegundos. 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);

}
```