Listagem do código-fonte

Projeto II – Projeto e Análise de Algoritmos I

Aluno: Mihael Rommel Barbosa Xavier RA: 10239617

Link do video do youtube: https://youtu.be/MYLvyHzl-98

Projeto I:

Este programa é uma ferramenta que gera um arquivo de entrada com compromissos aleatórios e os ordena a partir do Insertion e Merge Sort, assim gerando um arquivo de saida e informando ao usuário o tempo de execução e quantos passos foram necessarios . O programa segue as instruções especificadas pelo projeto.

Estrutura de Dados:

```
typedef struct {
int ano;
int mes;
int dia;
int hora;
int minuto;
float duracao;
char nome[50];
} Compromisso;
```

Estrutura para armazenar os compromissos

```
int rand_int(int min, int max) {
return min + rand() % (max - min + 1);
}
```

Função para gerar um número aleatório em um

```
void gerar_entrada_aleatoria(int quantidade) {
FILE *file = fopen("entrada.csv", "w");
if (file == NULL) {
```

```
printf("Erro ao abrir o arquivo entrada.csv\n");
return;
}

const char *compromissos[] = {"Aula", "TCC", "Webinar", "Natacao", "Reuniao"};
for (int i = 0; i < quantidade; i++) {
int ano = rand_int(2010, 2023);
int mes = rand_int(1, 12);
int dia = rand_int(1, 31);
int hora = rand_int(0, 23);
int minuto = rand_int(0, 59);
float duracao = (rand_int(1, 40) / 10.0);
const char *nome = compromissos[rand_int(0, 4)];

fprintf(file, "%d;%d;%d;%d;%d;%.1f;%s\n", ano, mes, dia, hora, minuto, duracao, nome);
}

fclose(file);
printf("Arquivo entrada.csv gerado com sucesso!\n");
}
```

Utiliza da funcao rand_int para gerar os dados do arquivo de entrada aleatoriamente.

```
Compromisso* ler_entrada(int *tamanho) {
FILE *file = fopen("entrada.csv", "r");
if (file == NULL) {
printf("Erro ao abrir o arquivo entrada.csv\n");
return NULL;
}

Compromisso *compromissos = NULL;
*tamanho = 0;
char linha[256];
while (fgets(linha, sizeof(linha), file)) {
compromissos = realloc(compromissos, (*tamanho + 1) * sizeof(Compromisso));
if (sscanf(linha, "%d;%d;%d;%d;%d;%f;%49[^\n]",
```

```
&compromissos[*tamanho].ano, &compromissos[*tamanho].mes,
&compromissos[*tamanho].dia, &compromissos[*tamanho].hora,
&compromissos[*tamanho].minuto, &compromissos[*tamanho].duracao,
compromissos[*tamanho].nome) == 7) {
(*tamanho)++;
} else {
compromissos = realloc(compromissos, (*tamanho) * sizeof(Compromisso));
}
}

fclose(file);
return compromissos;
}
```

Tive problemas em ler grandes quantidades de dados do arquivo. Utilizei alocação dinâmica de memória e a função sscanf para solucionar, que lê e analisa os dados de cada linha. Assim temos uma leitura correta dos dados dos compromissos

```
void escrever_saida(Compromisso *compromissos, int tamanho) {
FILE *file = fopen("saida.csv", "w");
if (file == NULL) {
printf("Erro ao abrir o arquivo saida.csv\n");
return;
}

for (int i = 0; i < tamanho; i++) {
fprintf(file, "%d;%d;%d;%d;%d;%.1f;%s\n",
compromissos[i].ano, compromissos[i].mes,
compromissos[i].dia, compromissos[i].hora,
compromissos[i].minuto, compromissos[i].duracao,
compromissos[i].nome);
}

fclose(file);
printf("Arquivo saida.csv gerado com sucesso!\n");
```

```
}
```

Função para escrever compromissos no arquivo saida.csv

```
void insertion_sort(Compromisso *compromissos, int tamanho, long *comparacoes) {
for (int i = 1; i < tamanho; i++) {
Compromisso chave = compromissos[i];
int j = i - 1;
while (j >= 0 && (compromissos[j].ano > chave.ano ||
(compromissos[j].ano == chave.ano && compromissos[j].mes > chave.mes) ||
(compromissos[j].ano == chave.ano && compromissos[j].mes == chave.mes &&
compromissos[j].dia > chave.dia))) {
compromissos[j + 1] = compromissos[j];
j--;
(*comparacoes)++;
}
compromissos[j + 1] = chave;
}
}
```

Algoritmo de ordenação Insertion Sort.

```
void merge(Compromisso *compromissos, int l, int m, int r, long *comparacoes) {
int i, j, k;
int n1 = m - l + 1;
int n2 = r - m;

Compromisso *L = malloc(n1 * sizeof(Compromisso));
Compromisso *R = malloc(n2 * sizeof(Compromisso));

for (i = 0; i < n1; i++)
L[i] = compromissos[l + i];
for (j = 0; j < n2; j++)
R[j] = compromissos[m + 1 + j];
```

```
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2) {
if (L[i].ano < R[j].ano || (L[i].ano == R[j].ano && L[i].mes < R[j].mes) ||
(L[i].ano == R[j].ano && L[i].mes == R[j].mes && L[i].dia <= R[j].dia)) {
compromissos[k] = L[i];
i++;
} else {
compromissos[k] = R[j];
j++;
}
k++;
(*comparacoes)++;
}

while (i < n1) {
compromissos[k] = L[i];
i++;
k++;
}

while (j < n2) {
compromissos[k] = R[j];
j++;
k++;
}

free(L);
free(R);
}

void merge_sort(Compromisso *compromissos, int l, int r, long *comparacoes) {
if (l < r) {
int m = l + (r - l) / 2;
```

```
merge_sort(compromissos, l, m, comparacoes);
merge_sort(compromissos, m + 1, r, comparacoes);

merge(compromissos, l, m, r, comparacoes);
}
}
```

Algoritmo de ordenação Merge Sort e Merge, para se funcionamento.

```
double get_time_in_seconds(struct timespec start, struct timespec end) {
return (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1000000000.0;
}
```

Com clock_gettime, temos uma maneira precisa de medir a duração das operações de ordenação, em segundos e nano segundos. Uma parte trabalhosa mas achei a solução.

```
int main() {
srand(time(NULL));
int opcao;
int tamanho = 0;
long comparacoes = 0;
struct timespec inicio, fim;
double tempo_execucao;

do {
printf("Menu:\n");
printf("1. Gerar entrada aleatória (entrada.csv)\n");
printf("2. Ordenar por ano (Insertion Sort)\n");
printf("3. Ordenar por ano (Merge Sort)\n");
printf("4. Sair\n");
printf("Escolha uma opcao: ");
scanf("%d", &opcao);
```

```
switch (opcao) {
case 1:
printf("Digite a quantidade de compromissos: ");
scanf("%d", &tamanho);
gerar_entrada_aleatoria(tamanho);
break;
case 2: {
Compromisso *compromissos = ler_entrada(&tamanho);
if (compromissos == NULL) break;

comparacoes = 0;
clock_gettime(CLOCK_MONOTONIC, &inicio);
insertion_sort(compromissos, tamanho, &comparacoes);
clock_gettime(CLOCK_MONOTONIC, &fim);
tempo_execucao = get_time_in_seconds(inicio, fim);

escrever_saida(compromissos, tamanho);
printf("Algoritmo: Insertion Sort\n");
printf("Tamanho Entrada: %d\n", tamanho);
printf("Tempo execucao: %.9f segundos\n", tempo_execucao);
printf("Comparacoes (passos): %ld\n", comparacoes);

free(compromissos);
break;
}
case 3: {
Compromisso *compromissos = ler_entrada(&tamanho);
if (compromissos == NULL) break;

comparacoes = 0;
clock_gettime(CLOCK_MONOTONIC, &inicio);
merge_sort(compromissos, 0, tamanho - 1, &comparacoes);
clock_gettime(CLOCK_MONOTONIC, &fim);
tempo_execucao = get_time_in_seconds(inicio, fim);

escrever_saida(compromissos, tamanho);
printf("Algoritmo: Merge Sort\n");
```

```
printf("Tamanho Entrada: %d\n", tamanho);
printf("Tempo execucao: %.9f segundos\n", tempo_execucao);
printf("Comparacoes (passos): %ld\n", comparacoes);

free(compromissos);
break;
}
case 4:
printf("Saindo...\n");
break;
default:
printf("Opcao invalida!\n");
}
} while (opcao != 4);

return 0;
}
```

A funcao main(), que chama todas as outras para a funcionalidade do programa.