SCC0220 - Laboratório Introdução à Ciência da Computação II

Relatório de execução do trabalho prático 3

Alunos	**NUSP**
Alec Campos Aoki	15436800
Juan Henrique Passos	15464826

Trabalho 3 – Seleção

Selection Sort

▯ Comentário

A tarefa consistia de ordenar uma lista de jogadores por suas notas e, em caso de empate, sua ordem lexicográfica, usando o método de seleção e um TAD pilha. O TAD pilha consistia de uma fila encadeada e dinâmica de política FILO (*First In Last Out*), de forma que o primeiro item empilhado será o último a ser desempilhafo, e o último item empilhado (no topo da pilha) será o primeiro a ser desempilhado. O TAD pilha utilizou um TAD item (composto de um ponteiro *void* e uma chave) para tratar as informações a serem empilhadas; as informações dos jogadores foram armazenadas em uma struct JOGADOR, que guardava seus nomes e notas. Um vetor de jogadores foi alocado dinâmicamente.

Como a saída exigia os jogadores em ordem decrescente de nota, foi preciso empilhar eles de forma crescente (começando pelo de menor nota e terminando pelo de maior). Para isso, foi usado o método de seleção (*Selection Sort*), que consiste de varrer um vetor de *n* elementos (index 0 até *n-1*) para achar o menor elemento; uma vez encontrado, ele troca de lugar com o elemento na posição 0. Na próxima iteração, o vetor é varrido de 1 até *n*-1 (ou seja, *n-1* elementos foram varridos), e o menor elemento é trocado pela posição 1, e assim por diante. Seu pseudocódigo seria algo similar a:

```
Início
i <- 0
j <- 0
para i de 0 até (não inclusivo) n, incremento i:
        para j de i até (não inclusivo) n, incremento j:
                se vetor[j] < menorNota:
                        menorNota = vetor[j]
Fim.
```

Nota-se que há dois laços *for* aninhandos, o que nos dá uma complexibilidade de **O(n²)**. Para o melhor caso, a complexidade é **O(n)**.

▯ Código

```c
//main.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>

#include "./pilha/item.h"
#include "./pilha/pilhaEncadeada.h"

typedef struct jogador_ JOGADOR;
struct jogador_{
   char nome[51];
   int nota;
};

typedef struct{
   clock_t start;
   clock_t end;
}Timer;
void start_timer(Timer *timer);
double stop_timer(Timer *timer);

PILHA *algoritmo_selecao(JOGADOR *vetJogadores, int quantJogadores);

/*Main*/
int main(void){
   Timer tempoTimer;
   double tempoExec;

   int quantJogadores;
   scanf(" %d", &quantJogadores);

   JOGADOR *vetJogadores = (JOGADOR *)malloc(quantJogadores*sizeof(JOGADOR));
   if(vetJogadores == NULL){
       printf("Erro ao alocar vetJogadores\n");
       return 1;
   }
```

```
    char nomeAux[51];

    for(int i=0; i<quantJogadores; i++){
        scanf("%s %d", nomeAux, &(vetJogadores[i].nota));
        strcpy(vetJogadores[i].nome, nomeAux);
    }

    start_timer(&tempoTimer);
    PILHA *pilha = algoritmo_selecao(vetJogadores, quantJogadores);
    tempoExec = stop_timer(&tempoTimer);

    JOGADOR *jogadorAux;

    printf("\n");

    for(int i=0; i<quantJogadores; i++){
        jogadorAux = (JOGADOR *)item_getDado(pilha_desempilhar(pilha));
        printf("%s %d\n", jogadorAux->nome, jogadorAux->nota);
    }

    printf("\n%lf\n", tempoExec);

    pilha_apagar(&pilha);

    return 0;
}

/*Funções do timer*/
void start_timer(Timer *timer){
    timer->start = clock();
    return;
}
double stop_timer(Timer *timer){
    timer->end = clock();
    return((double)(timer->end - timer->start)) / CLOCKS_PER_SEC;
}
```

```c
/*Função Selection Sort*/
PILHA *algoritmo_selecao(JOGADOR *vetJogadores, int quantJogadores){
   PILHA *pilha = pilha_criar();
   ITEM *vetItens[quantJogadores];

   JOGADOR jogadorMenorNota, jogadorAux;
   jogadorMenorNota.nota = 121;
   int index_jogadorMenorNota=-1;

   int i=0, j=0;
   for(; i<quantJogadores; i++){

       /*loop para achar a menor nota entre os jogadores*/
       for(j=i; j<quantJogadores; j++){

           if(vetJogadores[j].nota < jogadorMenorNota.nota){
               jogadorMenorNota = vetJogadores[j];
               index_jogadorMenorNota = j;
           }
           else if(vetJogadores[j].nota == jogadorMenorNota.nota){
               /*dois jogadores têm notas iguas, desempate por ordem lexicográfica*/
               if(strcmp(jogadorMenorNota.nome, vetJogadores[j].nome) < 0){
                   jogadorMenorNota = vetJogadores[j];
                   index_jogadorMenorNota = j;
               }
           }

       }

       /*swap jogador menor nota e posição i*/
       jogadorAux = vetJogadores[i];
       vetJogadores[i] = vetJogadores[index_jogadorMenorNota];
       vetJogadores[index_jogadorMenorNota] = jogadorAux;

       vetItens[i] = item_criar(&vetJogadores[i], i); //jogadorMenorNota, depois do swap,
está em vetJogadores[i]
```

```
        pilha_empilhar(pilha, vetItens[i]);

        jogadorMenorNota.nota = 121;
        index_jogadorMenorNota = -1;

    }


    return pilha;
}
```

```
//TAD PILHA
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#include "item.h"
#include "pilhaEncadeada.h"

typedef struct no_ NO;
typedef struct pilha_ PILHA;

struct no_{
    ITEM *pont_item;
    NO *proximoNo;
};

struct pilha_{
    NO *pont_noTopo;
    int topo;
};

PILHA *pilha_criar(void){
    PILHA *pilha = (PILHA *)malloc(sizeof(PILHA));
    if(pilha != NULL){
        pilha->pont_noTopo = NULL;
        pilha->topo = 0;
```

```
    }

    return pilha;
}

void pilha_apagar(PILHA **pilha){
    if(*pilha == NULL) return;

    while((*pilha)->topo > 0){
        ITEM *itemTemp = pilha_desempilhar(*pilha);
        item_apagar(&itemTemp);
    }

    free(*pilha);
    *pilha = NULL;

    return;
}

bool pilha_empilhar(PILHA *pilha, ITEM *item){
    if(pilha == NULL || item == NULL) return false;
    if(pilha_cheia(pilha)) return false;

    NO *noNovo = (NO *)malloc(sizeof(NO));
    if(noNovo == NULL) return false;

    noNovo->pont_item = item;
    noNovo->proximoNo = pilha->pont_noTopo;

    pilha->pont_noTopo = noNovo;
    (pilha->topo)++;

    return true;
}

ITEM *pilha_desempilhar(PILHA *pilha){
    if(pilha == NULL) return NULL;
```

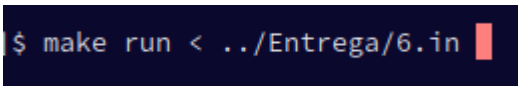
```
    NO *noAux = pilha->pont_noTopo;
    ITEM *itemAux = noAux->pont_item;

    pilha->pont_noTopo = noAux->proximoNo;

    free(noAux);
    noAux->pont_item = NULL;
    noAux->proximoNo = NULL;

    (pilha->topo)--;

    return itemAux;
}

ITEM *pilha_topo(PILHA *pilha){
    if(pilha == NULL) return NULL;

    return ((pilha->pont_noTopo)->pont_item);
}

int pilha_tamanho(PILHA *pilha){
    if(pilha == NULL) return -1;

    return pilha->topo;
}

bool pilha_vazia(PILHA *pilha){
    if(pilha == NULL) return true;

    if(pilha->topo == 0) return true;

    return false;
}

bool pilha_cheia(PILHA *pilha){
    if(pilha == NULL) return false;

    NO *noTeste = (NO *)malloc(sizeof(NO));
```

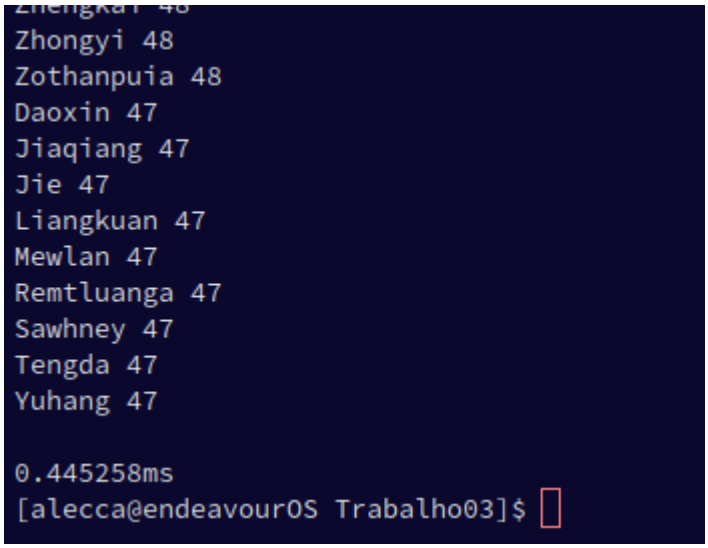
```
    if(noTeste != NULL){
        free(noTeste);
        noTeste = NULL;
        return false;
    }

    return true;
}
```

```
//TAD ITEM
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#include "item.h"

typedef struct item_ ITEM;

struct item_{
    void *dado;
    int index;
};

ITEM *item_criar(void *dado, int index){
    ITEM *item = (ITEM *)malloc(sizeof(ITEM));
    if(item != NULL){
        item->dado = dado;
        item->index = index;
    }

    return item;
}

bool item_apagar(ITEM **item){
    if(*item != NULL){
        free(*item);
        *item = NULL;
```

```
        return true;
    }
    return false;
}

void *item_getDado(ITEM *item){
    if(item == NULL) return NULL;

    return item->dado;
}

int item_getChave(ITEM *item){
    if(item == NULL) return -1;

    return item->index;
}
```

Saída





Shell Sort

Comentário

O algortimo de ordenação *shellshort* consiste na divisão do vetor em janelas formando subvetores com determinados elementos (escolhidos da forma *i+k*janela*, com *i* representando a quantidade de subvetores, e *k* iterações, que garante que cada vetor tenha apenas 1 elemento da janela. Tais subvetores são ordenados 2 a 2, e no final do ciclo dessa janela, inicialmente valendo tamanho do vetor dividido por 2, garante que os valores estarão em posições mais próximas da ordenação. Esse ciclo se repete até a janela possuir tamanho 1, no qual ocorre a última ordenação caso necessária. No pior dos casos a complexidade é **O(n²)**. No entanto, se o vetor vier ordenado, a janela irá se dividir *log n* vezes, e cada janela chegará ao vetor apenas uma vez. Logo no melhor dos casos a complexidade é **Ω(n logn)**. Portanto, é valido afirmarmos que o *shellsort* será, em média, mais eficiente que o *selection sort*. Interessante notar também que o fato de essa implementação não utilizar um TAD também contribui para que ela seja mais rápida.

Código

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

typedef struct jogo{
    char *nome;
    int rank;
}jogador;

jogador *preencher_jogadores(int n);
int maior(jogador a, jogador b);
void shellsort(jogador *selecao, int n);

int main(){ ···

jogador *preencher_jogadores(int n){ ···

int maior(jogador a, jogador b){ ···

void shellsort(jogador *selecao, int n){ ···
```


ICMC USP
SÃO CARLOS

```c
int main(){
    int n;
    jogador *selecao;

    scanf("%d", &n);

    selecao = preencher_jogadores(n);

    //  // Marca o início do tempo
    // clock_t start = clock();

    shellsort(selecao, n);
    // // Marca o fim do tempo
    // clock_t end = clock();
    for(int i = 0; i < n; i++){
        printf("%s %d\n", selecao[i].nome, selecao[i].rank);
    }
    // // Calcula e imprime o tempo de execução em milissegundos
    // double time_spent = (double)(end - start) / CLOCKS_PER_SEC * 1000;
    // printf("Tempo de execução: %f ms\n", time_spent);

    for(int i = 0; i < n; i++){
        free(selecao[i].nome);
        selecao[i].nome = NULL;
    }
    free(selecao); selecao = NULL;

    return 0;
}
```

```c
jogador *preencher_jogadores(int n){
    jogador *selecao;
    char nome[52];

    selecao = (jogador*) malloc(n * sizeof(jogador));

    for(int i = 0; i < n; i++){
        int rank;

        scanf("%s %d", nome, &rank);

        selecao[i].nome = (char*) malloc((strlen(nome)+1)*sizeof(char));

        strcpy(selecao[i].nome, nome);

        selecao[i].rank = rank;
    }

    return selecao;
}
```

```c
int maior(jogador a, jogador b){
    if((a.rank > b.rank) || (a.rank == b.rank && strcmp(a.nome, b.nome) < 0)){
        return 1;
    }
    return 0;
}

void shellsort(jogador *selecao, int n){
    //Dividir o vetor em janelas.
    for(int janela = n/2; janela > 0; janela /= 2){
        //Formar grupos de i até i+janela, sempre 2 a 2.
        for(int i = janela; i < n ; i++){
            //Variavel para realizar o swap caso necessario.
            jogador troca = selecao[i];
            //Index da posição final do jogador[i] apos a troca.
            int j;
            //Troca elementos 2 a 2, até estar ordenado.
            for(j = i; j >= janela && maior(troca, selecao[j-janela]); j -= janela){
                selecao[j] = selecao[j - janela];
            }
            //Caso j tenha sido alterado, houve trocas. Sendo selecao[j] nova posicao do jogador troca.
            selecao[j] = troca;

        }
    }
}
```

Saída

```
t]$ gcc shellsort.c -o shellsort -Wall
t]$ ./shellsort < ../Entrega/6.in
```

```
Jie 47
Liangkuan 47
Mewlan 47
Remtluanga 47
Sawhney 47
Tengda 47
Yuhang 47

0.008599ms
```