Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Ordenação de Objetos Móveis

BCC202 - Estrutura de Dados I

Felipe Braz Marques
Matheus Peixoto Ribeiro Vieira
Pedro Henrique Rabelo Leão de Oliveira

Professor: Pedro Henrique Lopes Silva

Ouro Preto
4 de fevereiro de 2023

Sumário

1 Introdução

Para este trabalho é necessário entregar um código em C, que passado o x e o y de três coordenadas calcula o deslocamento e a distância percorrida.

1.1 Especificações do problema

Utilizando a liguagem de programação C, é necessário ler a entrada de n pontos: A(Xa, Ya), B(Xb, Yb), ..., N(Xn, Yn). E calcular a distância percorrida e o deslocamento. Depois é necessário imprimir em ordem decrescente da distância e, caso há empate, utilizar o deslocamento em ordem crescente para critério de decisão. Porém se continuar igual, deverá ordenar por ordem alfabetica do nome.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code.
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [1]

1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *GCC*: versão 11.3.0.
- *Valgrind*: versão 3.18.1.

1.4 Especificações da máquina

A máquina onde o desenvolvimento e os testes foram realizados possui a seguinte configuração:

- Processador: Intel i5-9300H.
- Memória RAM: 16Gb.
- Sistema Operacional: Ubuntu 22.04.1 LTS.

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
gcc -c tp.c -Wall
gcc -c ordenacao.c -Wall
gcc tp.o ordenacao.o -o exe
```

Usou-se para a compilação as seguintes opções:

- *-Wall*: para mostrar todos os possível *warnings* do código.

Para a execução do programa basta digitar:

```
./exe caminho_até_o_arquivo_de_entrada
```

[1] Disponível em https://www.overleaf.com/

2 Desenvolvimento

2.1 ordenacao.h

Primeiramente, o grupo começou a desenvolver a interface do TAD no arquivo ordenacao.h, a fim de ter, desde o princípio, bem estruturado as principais funções do programa, sendo elas "alocaPontos", "desalocaPontos", "desalocaTrajeto", "calcularDistancia", "calcularDeslocamento", "imprime", "alocaTrajetos", "lerTrajetos", "ordenaDistancia", "ordenaDeslocamento", "ordenaNome"e "ordenacao", além dos typedefs das structs.

2.2 ordenacao.c

Dentro do arquivo ordenacao.c foi desenvolvido o corpo das funções prototipadas no ordenacao.h

No início do arquivo temos duas structs. A primeira com nome **"ponto"**, para representar as posições da trajetória, e a segunda com o nome de **"trajeto"**, para representar o trajeto feito pelos objetos móveis, possuindo um vetor do tipo ponto dentro de sua estrutura.

A função **"alocaPontos"** recebe por parâmetro um inteiro n indicando a quantidade de Pontos que serão alocados. Depois eles são alocados e é retornando um vetor de Ponto.

A função **"desalocaTrajeto"** recebe um ponteiro de Ponto e desaloca seu espaço da memória.

A função **"desalocaTrajeto"** recebe como parametro um ponteiro de ponteiro da struct Trajeto, a quantidade de trajetos e pontos e realiza um for de 0 até a quantidade de trajetos decrescido de um para desalocar os trajetos.

A função **"calcularDeslocamentoParcial"** recebe um ponto de início e um ponto final, e calcula a distância pela fórmula da distância entre dois pontos no plano cartesiano. Vale ressaltar que essa função existe pois, o cálculo da distância é a soma da distância de todos os pontos, e o cálculo de deslocamento é feito pela distancia de somente dois pontos. Assim, essa função faz-se útil para evitar a repetição de código.

A função **"calcularDistancia"** recebe como parâmetro um vetor de Trajeto, um index, um qtdPontos. A partir de um for, chama a função calcularDeslocamentoParcial para calcular o deslocamento entre dois pontos, e esse valor é somado na distância final. Ao final da função, o valor da distancia é atribuído para a distanciaTotal do trajetos na posição index, utilizando a função roundf para arredondar o valor para duas casas decimais, evitando imprecisões naturais de pontos flutuantes, permitindo, assim, a comparação dos mesmos em um momento futuro.

A função **"calcularDeslocamento"** vai receber o index do trajeto atual para calcular o desolocamento a partir do primeiro e último ponto.

A função **"ordenacao"** recebe como parâmetro um vetor de Trajeto e a quantidade de trajetos. Assim, serão realizadas três ordenações utilizando o QuickSort, que possui custo nLog(n), escolhido por ter um gasto de memória menor que o MergeSort. A primeira ordenação será de todo o vetor com base na distância, para deixá-la em ordem decrescente.

Depois é feito um for que varre todo o vetor de trajetos à procura de duas distâncias iguais, a do contador do for e a do contador acrescido de um. Caso encontre, é feito um for para saber até onde irá ocorrer essa igualdade. Quando encontra todas as igualdades, é chamada a função de ordenaDeslocamento, passando o vetor de trajetos e os índices de começo e fim da igualdade, para ordenar de forma crescente.

Por fim, para finalizar a ordenação, é feito um outro for de todo o vetor procurando onde houve uma igualdade da distância e deslocameto, passando para a função ordenaNome o vetor de trajeto e os indices, possibilitando, assim, a ordenação.

A função **"imprime"** recebe um vetor de Trajeto e a quantidade total de trajetos como parâmetros. Utilizando um for, imprime o nome dos trajetos, juntamente de suas respectivas distancia total e deslocamento total.

A função **"alocaTrajetos"** recebe um inteiro n como parâmetro, e aloca dinamicamente um vetor de Trajeto de tamanho n*sizeof(trajeto). Esse vetor é retornado ao final da função.

A função **"lerTrajetos"** recebe como parâmetros a quantidade de trajetos, a quantidade de Pontos e o vetor de trajetos. A partir de um for, é lido o nome de cada trajeto. Posteriormente, é alocado dinamicamente um vetor de pontos. E, por último, é lido os pontos de cada trajeto.

2.3 tp.c

Já o arquivo tp.c contém a inclusão do automato.h e a main, na qual está organizado a execução do programa, utilizando das interfaces do TAD.

Vale ressaltar, que há a presença de um for que vai de zero até a quantidade de trajetos para calcular os deslocamentos e distância de todos os trajetos.

3 Análise dos Resultados

Com o fim do trabalho prático, conseguimos obter os resultados esperados com os casos de teste disponibilizados, mesmo tendo um problema na ordenação dos nomes, que foi resolvido a partir da identificação de um erro na comparação dos vetores, pois o indíce estava trocado, o que gerou complicações durante um período do desenvolvimento.

4 Impressões Gerais e Considerações Finais

Fazendo uma análise geral sobre o trabalho, foi possível compreender melhor o uso das structs com a separação da interface no ordenacao.h e a implementação da mesma no arquivo ordenacao.c. O que, no início do desenvolvimento do código gerou algumas dúvidas e dificuldades para implementar. Porém após entender que as manipulações deveriam ser feitas no ordenacao.c, o desenvolvimento ficou mais simples e dinâmico.

Outro ponto a se destacar durante o desenvolvimento, foi na construção da função ordenaNome, pois foi necessário entender como funcionava a função strcmp da biblioteca string.h que retorna um valor positivo ou negativo, utilizado no código para saber se a comparação já estava, ou não, em ordem alfabética.

Por fim, o desenvolvimento do trabalho nos desafiou em algumas partes, como no entendimento dos métodos de ordenação, desenvolvidos antes de serem vistos em sala, mas em geral foi bem tranquilo, pois as outras partes já haviam sido conversadas em sala de aula.