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

Ordenação de objetos móveis com base na trajetória

BCC202 - Estrutura de Dados 1

Lucas Chagas, Nicolas Mendes, Pedro Morais
Professor: Pedro Henrique Lopes Silva

Ouro Preto
19 de fevereiro de 2023

Sumário

Lista de Figuras

1 Introdução

O objetivo deste trabalho pratico foi implementar um codigo que pertime indentificar e calcular as trajetorias entre diversos pontos dados em um plano.

1.1 Especificações do problema

O programa deverá calcular a trajetória de várias sequências de pontos, sendo a quantidade de sequências e pontos definidas por uma entrada do usuário. Os cálculos feitos no código são a trajetória e o deslocamento de uma sequência de pontos(Rotas). E por fim o vetor deve ser ordenado por meio de algum algoritmo de ordenação.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code. [1]
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [2]

1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *CLANG*: ferramentas de análise estática do código.
- *Valgrind*: ferramentas de análise dinâmica do código.

1.4 Especificações da máquina

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

- Processador: Ryzen 7 3700x.
- Memória RAM: 8 Gb.
- Sistema Operacional: Ubuntu.

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
gcc ordenacao.c tp.c -o exe -std=c99 -Wall -pg -g
```

Usou-se para a compilação as seguintes opções:

- *-std=99*: para usar-se o padrão ANSI C 99, conforme exigido.
- *-g*: para compilar com informação de depuração e ser usado pelo Valgrind.
- *-Wall*: para mostrar todos os possível *warnings* do código.
- *-pg*: para gerar o arquivo para fazer-se o profiling para identificar gargalos no programa.

[1] Vscode está disponível em https://code.visualstudio.com/
[2] Disponível em https://www.overleaf.com/

Para a execução do programa basta digitar:

```
./exe caminho_até_o_arquivo_de_entrada opcao
```

Onde “opcao” pode ser: “p” para fazer a, “d” para fazer b e “n” para fazer c.

2 Desenvolvimento

As funções auxiliares implementadas no código(localizadas no arquivo ”ordenacao.c”), consistem em criar vetores, desalocar vetores, ler vetores e imprimir vetores do tipo abstrato ”Rota”e ”Ponto”, sendo elas:”*alocaPonto”, ”desalocaPontos”, ”*alocaRota,”desalocaRota”,”lerRotas”e ”imprime”.

As funções Principais do codigo sao responsaveis pelos calculos matematicos realizados nas rotas e nos pontos e pelos criterios de ordenaçao, sendo elas: ”calcularDistancia”, ”calcularDeslocamento”, ”ordenaDistancia”, ”ordenaDeslocamento”, ”ordenaNome”e ”ordena”.

2.1 Funções principais

- ”calcularDistancia”: esta função recebe um vetor do tipo ”Rota”,a quantidade de rotas e a quantidade de pontos em cada rota. Com essas informações, a função irá percorrer o vetor ”Rota”e o vetor de pontos contido em cada posição do vetor ”Rota”e irá realizar a equaç ao de distância entre dois pontos com as coordenadas de dois pontos consecutivos e irá somar todas as distâncias encontradas em uma variável soma. Após isso a função irá armazenar o somatório total das distância na variável ”distância”na respectiva posição do vetor ”Rota”.

- ”calcularDeslocamento”: esta função recebe uma um vetor do tipo ”Rota”, a quantidade de rotas e a quantidade de pontos em cada rota. Com essas informações, a função percorre o vetor de rotas e em cada rota ela irá aplicar a regra de distância entre pontos apenas no primeiro e último ponto, descobrindo o deslocamento da rota, e atribuindo a variável ”deslocamento”da respectiva posição no vetor ”Rota”.

- ”ordenarDistância”: esta função é o método quicksort adaptado para os TADS ”Rota”e ”Ponto”com o ”pivô centrado inicializado no início. a função recebe o vetor ”Rota”, à ”esquerda”do vetor e a ”direita”, a partir disso a função irá colocar o vetor em ordem decrescente com base na ”distância”de cada posição do vetor ”Rota”.

- ”ordenaDeslocamento”: Esta função é o método quick sort adaptado para os TADS ”Rota”e ”Ponto”com o pivo centrado no início.A função recebe um vetor ”Rota”,um início e um final,a partir disso,irá ordenar o vetor em forma crescente com base no deslocamento de cada item do vetor ”Rota”.

- ”ordenaNome”: esta função é o método quick sort adaptado para os TADS ”Rota”e ”Ponto”com o pivo centrado no início.A função recebe o vetor ”Rota”,um início e um final,a partir disso,irá ordenar o ID de cada Rota em ordem crecente de acordo com o valor de cada letra na tabela ASCII.

- ”ordena”: esta função pega todas as outras funções de ordenação e às bota na seguinte sequência de prioridade:”ordenaDistancia”,”ordenaDeslocamento”e ”ordenaNome”.Em primeiro lugar o vetor irá ser ordenado de maneira decrescente com base na distância, nos casos em que a distância é a mesma, ele irá ordenar estes casos de maneira crescente com base no deslocamento, e se ainda houver casos de igualdade entre o deslocamento, estes casos serão ordenados com base na ordem do identificador/nome do objeto móvel.

2.2 Funções auxiliares

- ”alocaPonto”: esta função cria um vetor do tipo TAD Ponto e retorna tal vetor inicializado com 0 devido a função ”calloc”.

- "alocaRota": esta função aloca um vetor do tipo TAD Rota e também chama a função "alocaPonto"para alocar todos os pontos desse vetor Rota.
- "desalocaRota"esta função recebe um vetor tipo Rota e desaloca cada ponto dessa rota e a rota usando a função "free".
- "lerRotas": Esta função lê o ID da rota,e seus pontos usando a função "scanf".
- "imprime": esta função recebe o vetor rotas e a quantidade de rotas para que cada TAD do tipo Rota seja impresso na tela com seu ID,distância e deslocamento.

2.3 Imagens

A seguir algumas imagens de alguns casos de teste:

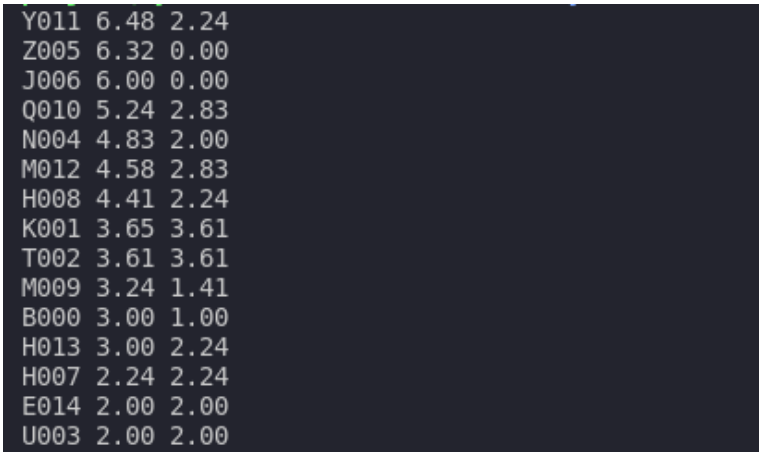

Figura 1: Primeiro caso de teste(output).

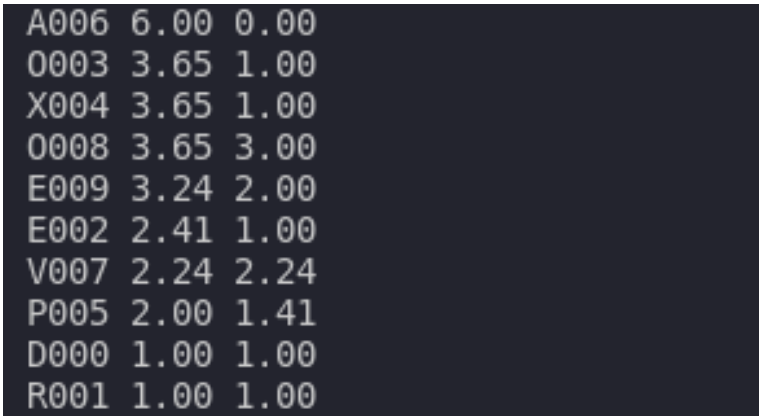

Figura 2: Terceiro caso de teste(output).

2.4 Trechos do código

A seguir estão as implementações das funções principais do código e a struct usada..

```
#include "ordenacao.h"
#define MAXTAM 1000
struct ponto
{
    int x;
```

```
6       int y;
7   };
8
9   struct rota
10  {
11      Ponto *pontos;
12      char id[5];
13      float distancia;
14      float deslocamento;
15  };
16
17
18
19  void calcularDistancia(Rota *rota, int qtdPontos, int qtdRotas)
20  {
21      int j = 0;
22      for (; j < qtdRotas; j++)
23      {
24          float soma = 0;
25          int i = 0;
26          for (; i < qtdPontos - 1; i++)
27          {
28              soma += (float)sqrt(pow(rota[j].pontos[i].x - rota[j].pontos[i +
                    1].x, 2) + (pow(rota[j].pontos[i].y - rota[j].pontos[i + 1].y,
                     2)));
29          }
30          soma = round(soma * 100) / 100;
31          rota[j].distancia = soma;
32      }
33  };
34
35  void calcularDeslocamento(Rota *rota, int qtdPontos, int qtdRotas)
36  {
37      int i = 0;
38      for (; i < qtdRotas; i++)
39      {
40          rota[i].deslocamento = sqrt(pow(rota[i].pontos[0].x - rota[i].pontos[
                qtdPontos - 1].x, 2) + pow(rota[i].pontos[0].y - rota[i].pontos[
                qtdPontos - 1].y, 2));
41      }
42  }
43
44
45
46  void ordenaDistancia(Rota *rotas, int inicio, int final)
47  {
48      int esquerda, direita, pivo;
49      Rota aux;
50      pivo = inicio;
51      esquerda = inicio;
52      direita = final;
53
54      while (esquerda <= direita)
55      {
56          while ((esquerda < final) && (rotas[esquerda].distancia > rotas[pivo].
                distancia))
57          {
58              esquerda++;
59          }
60
61          while ((direita > inicio) && (rotas[direita].distancia < rotas[pivo].
                distancia))
```

```
62          {
63              direita--;
64          }
65
66          if (esquerda <= direita)
67          {
68              aux = rotas[esquerda];
69              rotas[esquerda] = rotas[direita];
70              rotas[direita] = aux;
71              esquerda++;
72              direita--;
73          }
74      }
75      if (direita > inicio)
76      {
77          ordenaDistancia(rotas, inicio, direita);
78      }
79
80      if (esquerda < final)
81      {
82          ordenaDistancia(rotas, esquerda, final);
83      }
84  }
85
86  void ordenaDeslocamento(Rota *rotas, int inicio, int final)
87  {
88      int esquerda, direita, pivo;
89      Rota aux;
90      pivo = inicio;
91      esquerda = inicio;
92      direita = final;
93
94      while (esquerda <= direita)
95      {
96          while ((esquerda < final) && (rotas[esquerda].deslocamento < rotas[
                pivo].deslocamento))
97          {
98              esquerda++;
99          }
100
101         while ((direita > inicio) && (rotas[direita].deslocamento > rotas[pivo
                ].deslocamento))
102         {
103             direita--;
104         }
105
106         if (esquerda <= direita)
107         {
108             aux = rotas[esquerda];
109             rotas[esquerda] = rotas[direita];
110             rotas[direita] = aux;
111             esquerda++;
112             direita--;
113         }
114     }
115     if (direita > inicio)
116     {
117         ordenaDeslocamento(rotas, inicio, direita);
118     }
119     if (esquerda < final)
120     {
121         ordenaDeslocamento(rotas, esquerda, final);
```

```
122     }
123 }
124
125 void ordenaNome(Rota *rotas, int inicio, int final)
126 {
127     int esquerda, direita, pivo;
128     Rota aux;
129     pivo = inicio;
130     esquerda = inicio;
131     direita = final;
132
133     while (esquerda <= direita)
134     {
135         while ((esquerda < final) && (strcmp(rotas[esquerda].id, rotas[pivo].
                id) < 0))
136         {
137             esquerda++;
138         }
139
140         while ((direita > inicio) && (strcmp(rotas[direita].id, rotas[pivo].id
                ) > 0))
141         {
142             direita--;
143         }
144
145         if (esquerda <= direita)
146         {
147             aux = rotas[esquerda];
148             rotas[esquerda] = rotas[direita];
149             rotas[direita] = aux;
150
151             esquerda++;
152             direita--;
153         }
154     }
155
156     // Chamadas recursivas
157     if (direita > inicio)
158     {
159         ordenaNome(rotas, inicio, direita);
160     }
161     if (esquerda < final)
162     {
163         ordenaNome(rotas, esquerda, final);
164     }
165 }
166 void ordena(Rota *rotas, int qtdRotas)
167 {
168     int i;
169     ordenaDistancia(rotas, 0, qtdRotas - 1);
170
171
172
173     for (i = 0; i < qtdRotas - 1; i++)
174     {
175         if (rotas[i].distancia == rotas[i + 1].distancia)
176         {
177             int novoFinal = i;
178             while ((novoFinal < qtdRotas)&&(rotas[i].distancia == rotas[
                    novoFinal].distancia))
179             {
180                 novoFinal++;
```

```
181             }
182             novoFinal--;
183             ordenaDeslocamento(rotas, i, novoFinal);
184             i = novoFinal;
185         }
186     }
187
188
189     for (i = 0; i < qtdRotas - 1; i++)
190     {
191         if ((rotas[i].distancia == rotas[i + 1].distancia) && (rotas[i].
                deslocamento == rotas[i + 1].deslocamento))
192         {
193             int novoFinal = i;
194             while ((novoFinal < qtdRotas) && (rotas[i].distancia == rotas[
                    novoFinal].distancia) && (rotas[i].deslocamento == rotas[
                    novoFinal].deslocamento))
195             {
196                 novoFinal++;
197             }
198             novoFinal--;
199             ordenaNome(rotas, i, novoFinal);
200             i = novoFinal;
201         }
202     }
203
204 }
```

3 Experimetos e Resultados

Foram realizados vários testes com diversos tipos de input, com isso conseguimos corrigir alguns erros ao longo do processo,tendo entraves principalmente na escolha de um método de ordenação que fosse eficiente para a atividade proposta e para os casos de testes propostos,portanto,foi decidido implementar o método de ordenação "Quick Sort"devido a ser simples de ser implementado e também por sua complexidade ser O(n * logn), o que mostra ser extremamente eficiente e com um menor custo de tempo do que outros métodos.

4 Considerações Finais

O código foi capaz de lidar com todos os casos de teste propostos no "runcodes"e não apresentou nenhum erro na verificação do valgrind. O código ficou suficientemente intuitivo, sendo comentado e indentado em toda sua constituição.