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

Trabalho Prático 2

BCC202 - Estrutura de Dados

Caio Lucas Pereira da Silva, Vinicius Nunes dos Anjos, Lucca Sales de Souza Teodoro

Professor: Pedro Henrique Lopes Silva

Ouro Preto
20 de fevereiro de 2023

Sumário

Lista de Códigos Fonte

1 Introdução

Este trabalho prático tem como objetivo a ordenação de objetos móveis com base na análise de trajetórias, que é um conjunto de pontos que representam as posições ocupadas pelo objeto durante o seu movimento. A partir da análise exploratória desses dados é possível obter informações relevantes como o deslocamento e a distância percorrida pelo objeto, o que pode ser utilizado em diferentes domínios e aplicações, como análise de movimento de animais, detecção de movimento em multidões e rastreio de objetos em operações militares. O programa a ser desenvolvido deve receber um conjunto de trajetórias e listar os objetos móveis em ordem decrescente da distância percorrida, e caso haja empate, ordená-los com base no deslocamento e, se necessário, no identificador do objeto móvel. A ordenação de objetos móveis com base na trajetória é uma técnica importante para a análise de dados de movimento e pode ser utilizada em diversas áreas para obter informações relevantes sobre o comportamento dos objetos em movimento.

1.1 Especificações do problema

O problema consiste em construir um programa em C que receba como entrada trajetórias de objetos móveis, representadas por pontos em um plano, e retorne a lista dos objetos móveis ordenados por distância percorrida e deslocamento. Para implementar o programa, é necessário criar um Tipo Abstrato de Dados (TAD) Ponto e implementar as operações de alocação e desalocação de pontos, cálculo de distância e deslocamento, ordenação e impressão. O código-fonte deve ser modularizado em três arquivos, e o programa não pode ter memory leaks. A entrada é composta pelo número de objetos móveis, o número de pontos em cada trajetória e as coordenadas dos pontos, enquanto a saída deve apresentar o nome do objeto, a distância percorrida e o deslocamento, ordenados conforme os critérios especificados.

1.2 Pontos fortes

- O código é bem modularizado, com a divisão de funções em arquivos separados para lidar com objetos, pontos e ordenação.
- As funções possuem nomes claros e intuitivos, que facilitam a leitura e compreensão do código.
- As funções são bem definidas e focadas em apenas uma tarefa, o que as torna reutilizáveis em outros contextos.
- Há comentários que ajudam a entender o objetivo de cada função.
- O código faz uso de ponteiros para evitar a cópia desnecessária de estruturas e dados.

1.3 Pontos fracos

- Não há tratamento de erros em relação à alocação dinâmica de memória, o que pode resultar em erros de execução caso a alocação falhe.

1.4 Ferramentas utilizadas

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]

1.5 Ferramentas adicionais

Algumas ferramentas foram utilizadas para auxiliar no desenvolvimento, como:

- *Live Share*: ferramenta usada para *pair programming* à distância.
- *Valgrind*: ferramentas de análise dinâmica do código.

1.6 Especificações da máquina

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

- Processador: AMD Ryzen 5-5500U.
- Memória RAM: 8GB.
- Sistema Operacional: Linux Pop_OS.

1.7 Instruções de compilação e execução

Para a compilação do projeto, basta digitar o seguinte comando para rodar o arquivo *Makefile* disponível:

Compilando o projeto

```
make
```

Usou-se para a compilação as seguintes opções:

- *-o*: para definir o arquivo de saída.
- *-g*: para compilar com informação de depuração e ser usado pelo Valgrind.
- *-Wall*: para mostrar todos os possível *warnings* do código.
- *-c*: para compilação do código e geração dos arquivos objetos.

Para a execução do programa basta digitar :

```
./exe < arquivo.in
```

Onde o arquivo de entrada contém o tamanho da matriz e seus valores.

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

2 Desenvolvimento

O desenvolvimento foi realizado utilizando da técnica de pair programming, onde todos os integrantes do grupo programaram e participaram ativamente do código ao mesmo tempo. O uso das ferramentas Live Share para o compartilhamento de código e do Discord para a comunicação em equipe foi o que viabilizou o uso da técnica citada.

2.1 Codificação

Para a codificação foram utilizados sete arquivos (4 ".c"e 3 ".h"), os quais foram preechidos corretamente para o funcionamento.

2.2 Main code

```
1
2  #include <stdio.h>
3  #include "sort.h"
4  #include "point.h"
5  #include "object.h"
6
7  int main() {
8    int n;
9    int pointsLength;
10
11   scanf("%d %d", &n, &pointsLength);
12
13   Object *objects = allocateObjects(n, pointsLength);
14   readObjects(objects, n);
15
16   for (int i = 0; i < n; i++)
17   {
18     objects[i].distance = calcDistances(&objects[i]);
19     objects[i].displacement = calcDisplacement(&objects[i]);
20   }
21
22   shellSort(objects, n);
23   printObjects(objects, n);
24   freeObjects(&objects, n);
25 }
```

Código 1: Main

O programa recebe dois números inteiros, 'n' e 'pointsLength', que determinam, respectivamente, a quantidade de objetos a serem lidos e a quantidade de pontos que cada objeto contém. Em seguida, é alocado espaço na memória para os objetos usando a função 'allocateObjects'.

Os objetos são lidos com a função 'readObjects', e então é feita uma iteração sobre os objetos para calcular a distância e deslocamento de cada objeto, atribuindo os resultados aos campos correspondentes na struct 'Object'.

Em seguida, o algoritmo 'shellSort' é aplicado aos objetos usando a função 'shellSort', que está implementada no arquivo 'sort.h' incluído no início do código. Depois de ordenar os objetos, a função 'printObjects' é chamada para imprimir os objetos ordenados. Finalmente, a memória alocada para os objetos é liberada usando a função 'freeObjects'.

2.3 TAD utilizada

```c
typedef struct
{
  char id[5];
  int pointsLength;
  Point *points;
  double distance;
  double displacement;
} Object;
```

Código 2: TAD

Cada objeto contém um identificador de 5 caracteres (id), a quantidade de pontos que ele contém (pointsLength), um ponteiro para um array de Point (points), e dois valores de ponto flutuante (distance e displacement) que armazenam informações calculadas posteriormente.

A estrutura de dados 'Point' contém coordenadas (por exemplo, x e y), que são usadas para calcular a distância e deslocamento entre os pontos.

A variável 'pointsLength' armazena o número de pontos que cada objeto contém e o ponteiro points aponta para um array de 'Point', onde cada elemento desse array contém as coordenadas de um ponto do objeto.

As variáveis 'distance' e 'displacement' armazenam informações calculadas posteriormente. A variável 'distance' é usada para armazenar a distância total entre os pontos do objeto, enquanto a variável 'displacement' armazena a distância percorrida pelo objeto.

2.4 Funções de cálculos

```c
double calcDistanceBetweenPoints(Point *point1, Point *point2)
{
  double distance = 0;
  distance += (point2->x - point1->x) * (point2->x - point1->x);
  distance += (point2->y - point1->y) * (point2->y - point1->y);
  distance = sqrt(distance);
  return distance;
}
```

Código 3: calcDistanceBetweenPoints

Função de cálculo comum da distância de 2 pontos quaisquer, através do teorema de pitágoras, retornando o resultado desse calculo.

```c
double calcDistances(Object *object)
{
  double distance = 0;
  for (int i = 0; i < object->pointsLength - 1; i++)
  {
    distance += calcDistanceBetweenPoints(&object->points[i], &object->points[
        i + 1]);
  }
  return distance;
}
```

Código 4: calcDistances

Essa é uma função que calcula a distância percorrida de um objeto, dado um ponteiro para o objeto Object. A distância percorrida é calculada como a soma das distâncias entre cada par de pontos adjacentes no objeto.

```
double calcDisplacement(Object *object)
{
  return calcDistanceBetweenPoints(&object->points[0], &object->points[object
      ->pointsLength - 1]);
}
```

Código 5: calcDisplacement

Essa é uma função que calcula a distância entre o primeiro ponto e o ultimo ponto, que o resultado é o deslocamento.

2.5 Ordenação

Para a ordenação correta pedida pelo professor, seguindo os seus critérios, foram feitas 2 funções, uma de de ShellSort e outra de comparação.

```
void shellSort(Object *objects, int n)
{
  Object v;

  int i, j;
  int h = 1;
  while (h < n)
    h = 3 * h + 1;
  for (; h > 0; h /= 3)
    for (i = h; i < n; i++)
    {
      v = objects[i];
      j = i - h;
      while (j >= 0 && objectComp(&objects[j], &v))
      {
        objects[j + h] = objects[j];
        j -= h;
      }
      objects[j + h] = v;
    }
}
```

Código 6: shellSort

```
bool objectComp(Object *object1, Object *object2)
{
  if (object1->distance < object2->distance)
    return true;
  else if (object1->distance > object2->distance)
    return false;
  else
  {
    if (object1.displacement > object2.displacement)
      return true;
    else if (object1.displacement < object2.displacement)
      return false;
    else
    {
      if (strcmp(object1->id, object2->id) < 0)
```

```
        return true;
      else
        return false;
    }
  }

  return false;
}
```

Código 7: objectComp

A função 'shellSort' usa o algoritmo de ordenação ShellSort para ordenar o array de objetos 'Object' de acordo com as regras definidas pela função 'objectComp', que é uma função de comparação. O algoritmo ShellSort usa uma abordagem de dividir para conquistar, onde o array é dividido em subarrays menores e ordenado individualmente, em seguida, os subarrays são combinados para produzir um array ordenado.

Na implementação, a função começa inicializando a variável 'h' como 1 e, em seguida, usa um loop while para calcular o valor inicial de 'h', que é o maior número inteiro menor que n/3. Isso é feito para definir um valor de intervalo para o algoritmo ShellSort, que começa comparando e trocando elementos que estão 'h' posições de distância um do outro. Em seguida, a função usa dois loops for alinhados, um para percorrer os valores de 'h' de 'h' até 1 e outro para percorrer os elementos do array a partir da posição 'i = h'. Dentro do loop externo, a função faz um loop interno que começa a partir da posição j = i - h e, em seguida, percorre as posições anteriores do array, comparando o elemento na posição 'j' com o elemento na posição 'i'. Se o elemento na posição 'j' for maior que o elemento na posição 'i', então os elementos são trocados de posição. Esse processo é repetido enquanto 'j' for maior ou igual a zero e enquanto a função de comparação 'objectComp' retornar true. O objetivo é garantir que o elemento 'i' esteja na posição correta no subarray ordenado.

2.6 Resultados

```
15 3
B000 1 2 3 2 2 2
K001 0 1 2 2 3 3
T002 0 1 0 1 3 3
U003 1 1 1 2 1 3
N004 1 0 3 0 1 2
Z005 0 1 3 2 0 1
J006 3 3 0 3 3 3
H007 3 2 1 3 1 3
H008 1 0 1 3 2 2
M009 2 0 3 0 1 1
Q010 1 1 3 0 3 3
Y011 3 0 0 3 2 2
M012 1 2 2 3 3 0
H013 0 0 1 0 1 2
E014 1 3 3 3 3 3
```

Código 8: Arquivo de entrada

```
Y011 6.48 2.24
Z005 6.32 0.00
J006 6.00 0.00
Q010 5.24 2.83
N004 4.83 2.00
M012 4.58 2.83
H008 4.41 2.24
K001 3.65 3.61
T002 3.61 3.61
M009 3.24 1.41
H013 3.00 2.24
B000 3.00 1.00
H007 2.24 2.24
U003 2.00 2.00
E014 2.00 2.00
```

Código 9: Arquivo de saída

Foisse feito todos os calculos de deslocamento e distância e assim ordenado, como pedido pelo professor.

3 Conclusão

3.1 Considerações finais

Diante da análise do PDF e da implementação das funções nos arquivos .c correspondentes, o grupo conseguiu desenvolver as funcionalidades requeridas no projeto. Iniciando pelas funções mais simples e seguindo para as mais complexas, foi possível consolidar a compreensão do problema e garantir a correta integração das funcionalidades no código final. Além disso, a organização dos arquivos permitiu uma melhor estruturação do projeto e uma leitura mais clara do código. Ao final do projeto, os resultados esperados foram alcançados e o grupo pôde comprovar que a solução desenvolvida é capaz de lidar com a ordenação de objetos móveis com base na análise de trajetórias. Dessa forma, concluímos que o projeto foi bem sucedido e que a colaboração do grupo foi fundamental para atingirmos os objetivos propostos.