

Trabalho Prático

Programação e Desenvolvimento de Software I

1. Introdução

O trabalho prático da disciplina de Programação e Desenvolvimento de Software I (PDS1) consiste em resolver problemas que ocorrem em uma cidade. A cidade é representada por uma matriz e possui ruas, restaurantes e uma casa como características. A unidade de medida utilizada para indicar distância é "zambs". As ruas podem ser pavimentadas ou não pavimentadas, sendo que não é possível trafegar pelas que não possuem pavimento. Cada restaurante possui seu nome, seu custo (podendo ser caro ou barato), e um entregador que tem uma moto com velocidade característica. Através de arquivos, são informados tais dados citados anteriormente e, além disso, são dadas as coordenadas de cada rua e de cada restaurante:

- | | |
|--|--|
| - Arquivo ruas.txt :
(x, y, pavimento) | - Arquivo restaurantes.txt :
(x, y, nome, custo, velocidade) |
|--|--|

Em relação a casa, o usuário necessita de informar as coordenadas através de um teclado. É importante ressaltar que a casa deve estar em um trecho pavimentado da cidade.

Com todos estes dados, o trabalho propõe os seguintes problemas que devem ser resolvidos através de um algoritmo feito pelo aluno:

- 1- Calcular a distância da casa até todos os restaurantes;
- 2- Ler a preferência de custo do usuário e retornar todos os restaurantes que respeitem sua preferência em uma ordem de rapidez de entrega;
- 3- Ler a preferência de custo e de tempo de espera do usuário e retornar todos os restaurantes que respeitem ambas as preferências em ordem de rapidez de entrega.

2. Proposta de solução

O algoritmo utilizado se inicia com a definição dos Tipos Abstratos de Dados (TAD) que serão usados ao longo do código. É notório que os dados citados na introdução estão todos dentro de suas estruturas, porém é importante destacar alguns que não foram mencionados:

Restaurante.distancia → Distância do respectivo restaurante até a casa do usuário.

Esquina → TAD usado futuramente para a definição das rotas/distâncias.

Além disso, todos os métodos definidos nos TADs serão explicados ao decorrer deste documento.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  struct Casa{
6      int x;
7      int y;
8  };
9
10 struct Restaurante{
11     int x;
12     int y;
13     char nome[20];
14     char custo[6];
15     float velocidade;
16     int distancia;
17 };
18
19 struct Rua{
20     int x;
21     int y;
22     int pavimento;
23 };
24
25 struct Esquina{
26     int x;
27     int y;
28     int distancia;
29 };
30
31 struct Cidade{
32     Casa casa;
33     Rua caminhos[1000];
34     Restaurante rest[100];
35
36     void preencher_ruas(int n, int a, int b, int c);
37     void preencher_restaurantes(int n, int a, int b, float v);
38     void rotas(int a, int b, int max_x, int max_y, int n_rua, int n_rest);
39 };
40
```

Agora que temos todos estes novos tipos definidos, podemos preenchê-los com os dados disponibilizados nos documentos ruas.txt e restaurantes.txt.

Nota-se que, por definição, a cidade é formada por arranjos de ruas e restaurantes, e assim vamos preenchê-los com os métodos criados na própria estrutura:

```
40
41 void Cidade::preencher_ruas(int n, int a, int b, int c){
42     this->caminhos[n].x = a;
43     this->caminhos[n].y = b;
44     this->caminhos[n].pavimento = c;
45 }
46
47 void Cidade::preencher_restaurantes(int n, int a, int b, float v){
48     this->rest[n].x = a;
49     this->rest[n].y = b;
50     this->rest[n].velocidade = v;
51 }
52
```

Ao chamar tais métodos na função main, é possível atribuir os valores fornecidos no arquivo ao arranjo de ruas e restaurantes. Note que ambos os métodos possuem como parâmetro uma variável “n”, que controla o índice do arranjo que está sendo preenchido. Na função main, o método é chamado da seguinte forma:

```
136
137 int main(){
138     Cidade bh;
139     FILE *ruas, *restaurantes;
140     int x, y, pav;
141     float velocidade;
142     int n_ruas = 0, n_rest = 0;
143
144     // Lendo arquivos:
145     ruas = fopen("ruas.txt", "r+t");
146     restaurantes = fopen("restaurantes.txt", "r+t");
147     if (ruas == NULL || restaurantes == NULL){
148         printf("Algum arquivo nao foi encontrado.\n");
149         return 0;
150     }
151
152     while(!feof(ruas)){
153         fscanf(ruas, "%i %i %i", &x, &y, &pav);
154         bh.preencher_ruas(n_ruas, x, y, pav);
155         n_ruas++;
156     }
157     while(!feof(restaurantes)){
158         fscanf(restaurantes, "%i %i %s %s %f", &x, &y, bh.rest[n_rest].nome, bh.rest[n_rest].custo, &velocidade);
159         bh.preencher_restaurantes(n_rest, x, y, velocidade);
160         n_rest++;
161     }
162 }
```

Como observado na imagem acima, é feita uma verificação de existência dos arquivos e, caso eles não existam na pasta do programa, ocorre o encerramento do mesmo. Além disso, ao mesmo tempo que há o preenchimento dos arranjos da cidade “bh”, é feita uma contagem para saber o número total de restaurantes e ruas. Estes dados serão usados no futuro.

Após isso, na função main, são definidas duas variáveis que terão como conteúdo o maior valor de x e de y de toda a cidade, ou seja, terão o comprimento e a largura total da cidade. Essas variáveis serão usadas no método para calcular as distâncias dos

restaurantes até a casa, bem como para uma funcionalidade extra deste código: a representação gráfica do mapa da cidade. Ambos serão explicados mais à frente.

```
163     int maior_x = 0, maior_y = 0;
164     for(int i = 0; i<n_ruas; i++){
165         if (bh.caminhos[i].x > maior_x){
166             maior_x = bh.caminhos[i].x;
167         }
168         if (bh.caminhos[i].y > maior_y){
169             maior_y = bh.caminhos[i].y;
170         }
171     }
172     for(int i = 0; i<n_rest; i++){
173         if (bh.rest[i].x > maior_x){
174             maior_x = bh.rest[i].x;
175         }
176         if (bh.rest[i].y > maior_y){
177             maior_y = bh.rest[i].y;
178         }
179     }
180     int m[maior_y+1][maior_x+1];
181     for (int i = 0; i<=maior_y; i++){
182         for (int j = 0; j<=maior_x; j++){
183             m[i][j] = 0;
184         }
185     }
186     for(int i = 0; i<n_ruas; i++){
187         if (bh.caminhos[i].pavimento == 1){
188             m[bh.caminhos[i].y][bh.caminhos[i].x] = 1;
189         }
190     }
191     for(int i = 0; i<n_rest; i++){
192         m[bh.rest[i].y][bh.rest[i].x] = 2;
193     }
```

Também é criada uma matriz, que será usada na criação do mapa e para fazer a verificação da posição da casa. Por falar na casa, a próxima etapa do código consiste em obter as coordenadas da residência. O usuário deve informar através do teclado, como mostrado no código abaixo:

```

194 // Coordenadas da casa:
195 printf("Digite as coordenadas de sua casa: ");
196 while(1){
197     scanf("%i %i", &bh.casa.x, &bh.casa.y);
198     if(m[bh.casa.y][bh.casa.x] != 1){
199         printf("Coordenada da casa invalida.\n");
200         printf("Voce posicionou ela emm uma rua nao pavimentada ou no lugar de um restaurante.\n");
201         printf("Digite outras coordenadas: ");
202     } else{
203         break;
204     }
205 }
206 system("cls");
207 m[bh.casa.y][bh.casa.x] = 3;
208 // Calculando Distancias:
209 bh.rotas(bh.casa.x, bh.casa.y, maior_x, maior_y, n_ruas, n_rest);

```

Como mencionado anteriormente, a casa só pode ser posicionada em uma rua pavimentada. Dessa maneira o usuário tem que informar uma coordenada que possua pavimento, caso contrário, terá que digitar novamente até que a condição seja atendida. Após isso se inicia o processo do cálculo das distâncias de todos os restaurantes até a casa. O código do método chamado no fim da figura acima será dividido em partes, sendo que haverá explicação para cada uma.

```

53 void Cidade::rotas(int a, int b, int max_x, int max_y, int n_rua, int n_rest){
54     int x = a, y = b;
55     int esq = 0, dir = 0, cima = 0, baixo = 0, ver_esquina;
56     int ind = 0, dist = 0;
57     int ind_rua;
58     Esquina esquina[50];
59     while(ind >= 0){
60         for(int i = 0; i < n_rest; i++){
61             if(x == this->rest[i].x && y == this->rest[i].y){
62                 this->rest[i].distancia = dist;
63             }
64         }

```

Antes de iniciar a explicação da lógica aplicada no método, é importante mencionar como a struct "Esquina" funciona. Este TAD guarda três variáveis: as posições x e y da esquina e a distância percorrida da casa até determinada esquina, ou seja, funciona como se fosse uma memória.

A partir disso, observe que o método possui um arranjo de esquinas, pois assim é possível guardar todas as esquinas que a cidade possui. O método inteiro funciona sob uma condição de "ind >= 0". Esta variável "ind" tem duas funções nesta parte do código. A primeira é controlar o método e, ao mesmo tempo, ela indica o índice do arranjo de esquinas. Assim, se esta variável possui valor -1, significa que todas as esquinas da cidade já foram percorridas (caso a cidade não possua esquinas, funcionaria da mesma forma) e que todas as distâncias já foram calculadas.

Esta imagem mostra apenas uma parte do método, mais para frente deve ficar mais claro como esta variável é alterada.

Agora, analogamente, podemos fazer uma comparação entre este método e uma pessoa caminhando. A posição inicial dessa pessoa seria a coordenada da casa e, em cada loop, ela avança uma posição. Assim, sempre que o loop "while" é iniciado, é feita uma

verificação para ver se a coordenada da "pessoa" é igual à de um restaurante. Em caso afirmativo, a variável "distância" do respectivo restaurante é atualizada com a distância percorrida entre a casa e o restaurante em questão. Esta analogia entre o método e a pessoa será usada durante a documentação para o melhor entendimento, assim, esta pessoa, a partir de agora, é um personagem fictício da cidade.

```
65     for(int i = 0; i<n_rua; i++){
66         if (x == this->caminhos[i].x && y == this->caminhos[i].y){
67             ind_rua = i;
68         }
69         if (x == max_x){
70             dir = 0;
71         } else if (x+1 == this->caminhos[i].x && y == this->caminhos[i].y){
72             dir = this->caminhos[i].pavimento;
73         }
74         if (x == 1){
75             esq = 0;
76         } else if (x-1 == this->caminhos[i].x && y == this->caminhos[i].y){
77             esq = this->caminhos[i].pavimento;
78         }
79         if (y == max_y){
80             baixo = 0;
81         } else if (x == this->caminhos[i].x && y+1 == this->caminhos[i].y){
82             baixo = this->caminhos[i].pavimento;
83         }
84         if (y == 1){
85             cima = 0;
86         } else if (x == this->caminhos[i].x && y-1 == this->caminhos[i].y){
87             cima = this->caminhos[i].pavimento;
88         }
89     }
90     ver_esquina = dir+esq+cima+baixo;
91     if(ver_esquina > 1){
92         esquina[ind].x = x;
93         esquina[ind].y = y;
94         esquina[ind].distancia = dist;
95         ind++;
96     }
```

Nesta parte, é feita uma verificação das possíveis direções em que podemos caminhar a partir da nossa coordenada atual. Para isso, foi criado um loop "for" que percorre todas as ruas da cidade. A primeira condição do laço tem a função de obter o índice exato da rua em que estamos posicionados no momento. Essa variável será utilizada posteriormente no código. Em seguida, é feita a verificação da disponibilidade das ruas em todas as quatro direções. Se a nossa coordenada atual corresponder a uma extremidade da cidade, a variável da respectiva direção é automaticamente definida como 0. Caso contrário, é feita a verificação do pavimento da rua na direção específica. Se a rua estiver pavimentada, a variável correspondente receberá o valor 1, caso contrário, receberá o valor 0.

Após isso, é feita a verificação se a posição atual é uma esquina. Para saber isso, basta somar todas as direções verificadas anteriormente. Caso essa soma dê um número superior a 1, isso significa que mais de uma direção pode ser tomada, o que significa que

estamos em uma esquina. Dessa maneira, o arranjo de esquinas é atualizado e o seu índice aumentado em um.

```
97         if(dir){
98             this->caminhos[ind_rua].pavimento = 0;
99             x++;
100            dist++;
101        } else if(esq){
102            this->caminhos[ind_rua].pavimento = 0;
103            x--;
104            dist++;
105        } else if(cima){
106            this->caminhos[ind_rua].pavimento = 0;
107            y--;
108            dist++;
109        } else if(baixo){
110            this->caminhos[ind_rua].pavimento = 0;
111            y++;
112            dist++;
113        } else{
114            this->caminhos[ind_rua].pavimento = 0;
115            x = esquina[ind-1].x;
116            y = esquina[ind-1].y;
117            dist = esquina[ind-1].distancia;
118            ind--;
119        }
120    }
121 }
```

Por fim, vamos mover nosso personagem fictício. Isso é feito de maneira simples, basta movê-lo para uma posição disponível. Ao mesmo tempo em que nos movemos, vamos alterar o valor do pavimento para 0. Isso indica que já passamos por essa posição e evita erros nas verificações mencionadas anteriormente. Por exemplo, se não alterássemos o valor dessa variável, o código consideraria todas as ruas como esquinas, pois consideraria o retorno à posição anterior como um caminho possível. Isso resultaria em um loop infinito, impedindo o progresso.

Além disso, vamos incrementar a distância percorrida à medida que nos movemos. Caso não haja mais direções disponíveis, o que significa que estamos em uma rua sem saída, vamos retornar para a última esquina percorrida. Dessa forma, poderemos explorar a direção que ignoramos anteriormente. Se a última esquina percorrida também não tiver mais direções disponíveis, ela será considerada uma rua sem saída e iremos retornar para a esquina anterior. Esse processo será repetido até chegarmos à primeira esquina encontrada. Quando a primeira esquina não tiver mais direções disponíveis, a variável "ind" será definida como -1, encerrando o método, conforme mencionado no início da explicação.

Dessa forma, vamos percorrer o mapa todo e calcular a distância de todos os restaurantes até a casa do usuário.

Antes de voltarmos ao código da função main, vou citar o algoritmo de ordenação implementado, chamado de "Insert Sort":

```
123 void insert_sort(float *vet, int *ind, int n){
124     int i, j, aux_i;
125     float aux;
126     for(i = 1; i<n; i++){
127         aux = vet[i]; aux_i = ind[i];
128         for (j = i; j > 0 && aux < vet[j-1]; j--){
129             vet[j] = vet[j-1];
130             ind[j] = ind[j-1];
131         }
132         vet[j] = aux;
133         ind[j] = aux_i;
134     }
135 }
136
```

A partir desta função, ordenamos todos os valores de um vetor. O algoritmo funciona como se você estivesse jogando um jogo de cartas, um exemplo seria o jogo chamado buraco. À medida que vai recebendo as cartas, você irá trocar as posições da nova carta com a carta na posição anterior, até achar a posição correta. Note que a função recebe como parâmetro dois vetores. Normalmente usamos só um arranjo nesta função, mas neste caso optei por usar dois, pois vamos ordenar os índices de acordo com o vetor principal. Para ficar mais claro, vamos voltar a função main:

```
210 float distancias[n_rest];
211 int indices_dist[n_rest];
212 for (int i = 0; i<n_rest; i++){
213     distancias[i] = (float)bh.rest[i].distancia;
214     indices_dist[i] = i;
215 }
216 insert_sort(distancias, indices_dist, n_rest);
```

Antes de chamarmos o Insert Sort, criamos e preenchemos dois vetores: um com todas as distâncias calculadas e outro com os índices dos restaurantes. Em seguida aplicamos o algoritmo de ordenação. O vetor de índices será ordenado de acordo com as distâncias de seus restaurantes, para que retornemos ao usuário o nome do restaurante e a sua distância de maneira correta.

Usando a mesma lógica, vamos calcular e ordenar os restaurantes de acordo com sua rapidez:


```

217 // Calculando Rapidez:
218 float rapidez[n_rest];
219 int indices_rap[n_rest];
220 for (int i = 0; i < n_rest; i++){
221     rapidez[i] = bh.rest[i].distancia / bh.rest[i].velocidade;
222     indices_rap[i] = i;
223 }
224 insert_sort(rapidez, indices_rap, n_rest);

```

Para calcular a rapidez de entrega, basta dividir a distância do restaurante pela velocidade do entregador. Em seguida, realizamos a ordenação da mesma forma que foi feita com as distâncias.

Desta maneira, já temos todas as distâncias e rapidez de entrega de cada restaurante.

Com isso, introduzir mais uma funcionalidade do código: Um menu para o usuário:

```

225 char preferencia[6];
226
227 int opcao;
228 do{
229     printf("===== MENU =====\n");
230     printf("(0) Mostrar Mapa\n");
231     printf("(1) Distancias dos restaurantes\n");
232     printf("(2) Preferencia + Rapidez\n");
233     printf("(3) Preferencia + Rapidez + Tempo Maximo\n");
234     printf("(4) Fechar Programa\n");
235     printf("===== \n\n");
236     printf("Digite a opcao desejada: ");
237     scanf("%i", &opcao);
238     system("cls");
239     switch(opcao){

```

Usaremos a estrutura do/while para repetir o menu até que o usuário opte por encerrar o programa. Além disso, a condicional switch/case foi implementada para o código retornar apenas o que o usuário deseja.

Como citado anteriormente, o código possui a função extra de mostrar o mapa da cidade ao usuário, que pode ser requisitado ao selecionar a opção 0 do menu. O mapa possui uma legenda e pode ser usado para fazer algumas verificações. Na seção de discussão de resultados será mostrado um exemplo.

```

240         case 0:
241             printf(" Mapa:\n\n");
242             for (int i = 1; i<=maior_y; i++){
243                 for (int j = 1; j<=maior_x; j++){
244                     if(m[i][j] == 0){
245                         printf(". ");
246                     } else if (m[i][j] == 1){
247                         printf("o ");
248                     } else if (m[i][j] == 2){
249                         printf("M ");
250                     } else if (m[i][j] == 3){
251                         printf("H ");
252                     }
253                 }
254                 printf("\n");
255             }
256             printf("\nLegenda:\n");
257             printf("o -> Rua pavimentada\t\tM -> Restaurante\n");
258             printf(". -> Rua nao pavimentada\tH -> Casa\n");
259             printf("\n\n");
260             break;
261

```

A opção 1 do menu retorna a distância de todos os restaurantes até a casa do usuário. Como foi tudo calculado previamente, basta imprimir na tela o vetor ordenado:

```

262         case 1:
263             printf("Distancias da casa ao restaurante:\n\n");
264             for(int i = 0; i<n_rest; i++){
265                 printf("%s : %.0f zambs\n", bh.rest[indices_dist[i]].nome, distancias[i]);
266             }
267             printf("\n\n");
268             break;
269

```

A opção 2 deve imprimir na tela os restaurantes em ordem de rapidez de entrega, de acordo com a preferência de preço do usuário. Para isso basta obtermos a preferência do mesmo, pois já calculamos e ordenamos os restaurantes de acordo com a rapidez. Foi criada uma variável antes do início do menu que será usado para obter este dado, como mostra a figura abaixo:

```

270 case 2:
271     printf("Digite a preferencia de custo (Caro ou Barato): ");
272     scanf("%s", preferencia);
273     while(strcmp(preferencia, "Caro") && strcmp(preferencia, "Barato")){
274         printf("Preferencia inserida invalida.\n");
275         printf("Digite a preferencia de custo (Caro ou Barato): ");
276         scanf("%s", preferencia);
277     }
278     system("cls");
279     printf("Restaurantes %ss em ordem de rapidez de entrega:\n\n", preferencia);
280     for(int i = 0; i<n_rest; i++){
281         if (!(strcmp(bh.rest[indices_rap[i]].custo, preferencia))){
282             printf("%s : %.1f minutos\n", bh.rest[indices_rap[i]].nome, rapidez[i]);
283         }
284     }
285     printf("\n\n");
286     break;
287 }

```

Para a verificação da preferência, foi usada a função “strcmp”, que é disponibilizada pela biblioteca string.h.

Já a opção 3 do menu requer obter além da preferência, do tempo máximo que o usuário está disposto a esperar. Isso é resolvido apenas inserindo uma nova variável para obter este tempo e adicionando esta nova condição a estrutura if demonstrada na imagem acima.

```

287 case 3:
288     float temp_max;
289     printf("Digite a preferencia de custo (Caro ou Barato): ");
290     scanf("%s", preferencia);
291     while(strcmp(preferencia, "Caro") && strcmp(preferencia, "Barato")){
292         printf("Preferencia inserida invalida.\n");
293         printf("Digite a preferencia de custo (Caro ou Barato): ");
294         scanf("%s", preferencia);
295     }
296     printf("Digite o tempo maximo de espera: ");
297     scanf("%f", &temp_max);
298     system("cls");
299     printf("Restaurantes %ss e com tempo de entrega <= %.2f minutos:\n\n", preferencia, temp_max);
300     for (int i = 0; i<n_rest; i++){
301         if (!(strcmp(preferencia, bh.rest[indices_rap[i]].custo)) && rapidez[i]<=temp_max){
302             printf("%s : %.1f minutos\n", bh.rest[indices_rap[i]].nome, rapidez[i]);
303         }
304     }
305     printf("\n\n");
306     break;
307
308 case 4:
309     printf("Fechando...\n");
310     break;
311
312 default:
313     printf("Opcao Invalida\n\n");
314 }
315 }while(opcao != 4);
316 return 0;
317 }
318 }

```

Caso a opção 4 seja selecionada, o programa se encerra, e caso uma opção inserida seja inválida, o menu é iniciado novamente. Com isso, o código chega ao fim.

3. Discussão dos Resultados

Retomando os problemas propostos no trabalho, temos:

- 1- Calcular a distância da casa até todos os restaurantes;
- 2- Ler a preferência de custo do usuário e retornar todos os restaurantes que respeitem sua preferência em uma ordem de rapidez de entrega;
- 3- Ler a preferência de custo e de tempo de espera do usuário e retornar todos os restaurantes que respeitem ambas as preferências em ordem de rapidez de entrega.

Para a apresentação dos resultados, iremos utilizar o arquivo contendo as informações das ruas e restaurantes fornecido pelo professor, juntamente com um exemplo em que as coordenadas da casa são (2, 2).

Primeiramente, antes de mostrar o resultado dos problemas enunciados, veremos que o mapa da cidade está sendo gerado corretamente, quando o usuário escolhe a opção 0 do menu:

```
Mapa:
M . . . . . M
o H o o o o o o o o o o o o o o o o o o M . . . . . o . . . . .
o . . . . . . . . . . . . . . . . . . . . M . . . . . o . . . . .
o . . . . . . . M . . . . . M o o o o o o o o o o o o o o o o o o
o . . . . . o . . . . . o . . . . . . . . . . o . . . . . o . . . . .
o . . M o o o o o o o o o o o . o . . . . . . . o o o o o o o o M . . o
o . . . o . . . . . . . o . . . . . M . o . . . . . . . . . . o
o . . . o . . . . . M . o . . . . . o . o . . . . . . . . . . o
o . . . o . . . . . o . o . . . o M o o o o o o . o . . . . . . . o
o . . . o . . . . . M o o o o . o . . . . . o . o . . . . . . . o
o o o o o . . . . . o . . . . . o . . . . . o o o o o o M . . . . . o
o . . . o . . . . . o . . . . . o . o . . . o . o . . . . . . . o
o . . . o o o o o o o o o o o o o . . . . . o . o . . . o . . . . . o
o . . . o . . . . . o . . . . . o . o . . . o . o . . . . . M o
o . . M . . . . . . . . . o . M o o o o o o o . o . . . . . . . o
o . . . o . . . . . . . . . o . . . . . . . . . . M . . . . . o
o . . . o . . . . . o . . . . . o . . . . . . . . . . . . . . o
o o M . o . . . . M o o o o o o o o o o o o o o M . . . . . . . o
. . . . o . . . . . . . . . . . . . . . . . . . . . . . . . . o
. . . . o o o o o o o o o o o o M . . . . . . . . . . . . . . o
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . o
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . M

Legenda:
o -> Rua pavimentada      M -> Restaurante
. -> Rua nao pavimentada  H -> Casa
```

Agora partimos para o problema número 1 do trabalho: Calcular as distâncias da casa até todos os restaurantes:

```
Distancias da casa ao restaurante:
Cantina_da_Carol : 2 zambs
Vila_Matriz : 18 zambs
Espeto_do_Chico : 19 zambs
Espolex : 20 zambs
Quibao : 21 zambs
Churrasco_de_gato : 26 zambs
Macarrao_na_chapa : 35 zambs
Xulambs : 37 zambs
Sai_de_baixo : 37 zambs
Taste_Vin : 38 zambs
Burger_queen : 41 zambs
Las_pombas : 42 zambs
Voador : 49 zambs
Ta_danado : 56 zambs
Comida_de_buteco : 58 zambs
Caro_e_ruim : 60 zambs
Moto_velha : 63 zambs
Xapurex : 66 zambs
Dogao_da_esquina : 69 zambs
Chega_frio : 70 zambs
Emporio_zambs : 73 zambs
Xucesso_da_Cida : 78 zambs
Cometa : 84 zambs
```

A partir dos dados utilizados neste exemplo, as distâncias obtidas foram as mostradas na imagem. Todas as distâncias foram verificadas utilizando o mapa como referência. Além disso, foram testados outros exemplos que também foram concluídos com sucesso. Portanto, o problema número 1 foi resolvido com êxito.

É importante observar que as distâncias estão sendo impressas em ordem crescente, demonstrando que a ordenação também está funcionando corretamente.

No exercício número 2, de acordo com a preferência de preço do usuário, os restaurantes são impressos em ordem crescente de rapidez. As duas imagens a seguir mostram os resultados de acordo com as duas preferências possíveis:

Restaurantes Caros em ordem de rapidez de entrega:

```
Cantina_da_Carol : 0.4 minutos
Churrasco_de_gato : 6.5 minutos
Taste_Vin : 7.6 minutos
Espeto_do_Chico : 9.5 minutos
Voador : 9.8 minutos
Las_pombas : 10.5 minutos
Xucesso_da_Cida : 15.6 minutos
Macarrao_na_chapa : 17.5 minutos
Caro_e_ruim : 30.0 minutos
Comida_de_buteco : 58.0 minutos
Moto_velha : 63.0 minutos
```

Restaurantes Baratos em ordem de rapidez de entrega:

```
Vila_Matriz : 6.0 minutos
Burger_queen : 10.3 minutos
Emporio_zambs : 14.6 minutos
Cometa : 16.8 minutos
Sai_de_baixo : 18.5 minutos
Ta_danado : 18.7 minutos
Espolex : 20.0 minutos
Quibao : 21.0 minutos
Dogao_da_esquina : 23.0 minutos
Chega_frio : 35.0 minutos
Xulambs : 37.0 minutos
Xapurex : 66.0 minutos
```

Com isso, podemos considerar o problema número 2 como resolvido.

O problema número 3 também foi solucionado com sucesso, como pode ser observado nas próximas imagens. A única diferença entre as questões 2 e 3 é que na terceira questão temos uma restrição de tempo. Para observar essa restrição, é possível comparar os resultados obtidos nas duas questões.

Restaurantes Caros e com tempo de entrega ≤ 20.00 minutos:

```
Cantina_da_Carol : 0.4 minutos
Churrasco_de_gato : 6.5 minutos
Taste_Vin : 7.6 minutos
Espeto_do_Chico : 9.5 minutos
Voador : 9.8 minutos
Las_pombas : 10.5 minutos
Xucesso_da_Cida : 15.6 minutos
Macarrao_na_chapa : 17.5 minutos
```

```
Restaurantes Baratos e com tempo de entrega <= 20.00 minutos:  
Vila_Matriz : 6.0 minutos  
Burger_queen : 10.3 minutos  
Emporio_zambs : 14.6 minutos  
Cometa : 16.8 minutos  
Sai_de_baixo : 18.5 minutos  
Ta_danado : 18.7 minutos  
Espolex : 20.0 minutos
```

Com isso concluímos que todo o trabalho foi solucionado com sucesso. Para qualquer outro teste, o código fonte estará disponível junto com este pdf.