Trabalho prático – Programação e desenvolvimento de software

Felipe Dias de Souza Martins – 2023002073

Introdução:

O trabalho prático consiste em criar um código capaz de atender o que foi pedido. Nessa atividade, levamos em conta uma cidade, que possui coordenadas, ruas e restaurantes, na segunda dimensão. Para isso, foram dado 3 arquivos, um com as coordenadas em x,y e um outro número binário para indicar se tal coordenada era pavimentada ou não, o outro era sobre restaurantes, o qual cada um tinha uma localização em x e y, um nome, um indicador se o restaurante era caro ou barato, a velocidade que o entregador viria até uma certa localização. O desafio era colocar isso tudo dentro de um código, e implementar um método que só permitisse que eu “percorresse” caminhos e coordenadas pavimentadas.

Proposta de solução:

Para solucionar o problema mencionado anteriormente, foi usado o método BFS, o qual faz uma simulação a partir de grafos, contendo vértices e arestas, onde tenhamos um grafo inicial – o qual poderia ser atribuído à localização que os restaurantes fariam a entrega –, e a partir do primeiro grafo, os que estavam nas proximidades tinham sua distancia calculada. Falando de uma forma mais técnica e de como essa estrutura me ajudou a solucionar o problema, no primeiro momento todas as coordenadas eram dadas como “falsas”, e a partir disso começava uma fila, que começava com a coordenada “casa” e finalizava vazia, a partir disso, criei um método para a locomoção de coordenadas adjacentes, o qual era marcada como “verdadeiras” e sua distancia até o ponto que deu origem era calculada, logo depois, essas adjacentes às primeiras coordenadas era postas na fila, e aí o algoritmo seguia para as demais levando em conta somente as coordenadas pavimentadas, já que só era permitido caminhar por elas.Com esse método era possível fazer a primeira questão do trabalho, que pedia justamente para calcular a distância entre a localização do restaurante e a localização “casa” fornecida pelo usuário.

A partir disso, para elaborar uma solução para a segunda questão, que pedia para que o usuário escolhesse entre restaurantes caros ou baratos, e em seguida, o programa retornasse uma lista com os que atendem ao pedido por ordem de rapidez de entrega. Para resolver esse problema, depois da escolha da preferencia do usuário, o programa separa os que atendem à escolha de preço e armazena em uma lista, depois disso, foi necessário implementar um método que organiza cada um dos restaurantes pela distância e os divide pela

respectiva velocidade de entrega, os que tinham as maiores distâncias geralmente iam para o final da fila, enquanto os mais pertos iam para o começo, e após isso, era mostrado ao usuário a lista com esses restaurantes, mostrando o nome e o tempo.

Para resolver a última questão, que requisitava ao usuário um tempo de espera, e a partir daí era criado uma lista somente com os restaurantes que chegariam até esse tempo. O processo era feito utilizando a resolução das outras duas questões, e partindo da distância e o tempo de demora que cada restaurante teria da coordenada “casa”, partindo disso, era só implementar um código que filtrasse para mostrar somente os que estavam iguais ou abaixo do tempo requisitado.

Além do método BFS e do método de organização, foi-se usado estrutura de tipagem, para criar os tipos “coordenadas”, “restaurantes” e “ruas”, também foi utilizado vetores, principalmente para armazenar cada coordenada e cada restaurante em uma variável diferente, além de outras ferramentas mais simples, como matrizes, abrir e ler arquivos, entre outros.

Apresentação de resultados:

Segue abaixo o código utilizado com comentários e com um exemplo.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>

//criação das estruturas

struct Coordenadas {
    int x;
    int y;
};

//verificação de pavimentação

struct Rua {
    struct Coordenadas coordenada;
    bool paviment;
    void pavimento(int a);
};

void Rua::pavimento(int a) {
    if (a == 1) {
        this->paviment = true;
    } else {
        this->paviment = false;
```

```
    }
}

struct Restaurante {
    int x;
    int y;
    char nome[100];
    char preco[10];
    int velocidade;
    int distancia(int x, int y, struct Rua cidade[23][39]);
};

//método BFS adaptado para o código

int Restaurante::distancia(int x, int y, struct Rua cidade[23][39]) {
    int zambs = 0, cont = 0, marcador = 0;
    struct Coordenadas ruaatual, novarua;
    x--;
    y--;
    bool verificados[23][39];
    for (int i = 0; i < 23; i++) {
        for (int j = 0; j < 39; j++) {
            verificados[i][j] = false;
        }
    }
    int vizvert[] = {0, 0, -1, 1}; //método de locomoção entre
coordenadas adjacentes verticais
    int vizhor[] = {-1, 1, 0, 0}; //método de locomoção entre coordenadas
adjacentes horizontais
    struct Coordenadas fila[23 * 39]; //criação da fila
    int frente = 0;
    int tras = 1;
    verificados[y][x] = true;
    fila[0].x = x;
    fila[0].y = y;
    if (x == this->x && y == this->y) {
        return 0;
    }
    while (frente != tras) {
        ruaatual = fila[frente];
        int atualX = ruaatual.x;
        int atualY = ruaatual.y;
        for (int i = 0; i < 4; i++) {
            novarua.x = atualX + vizhor[i];
            novarua.y = atualY + vizvert[i];
            if (novarua.x < 39 && novarua.x >= 0 && novarua.y < 23 &&
novarua.y >= 0) {
                if (cidade[novarua.y][novarua.x].paviment &&
!verificados[novarua.y][novarua.x]) {
```

```
                    verificados[novarua.y][novarua.x] = true;
                    fila[tras].x = novarua.x;
                    fila[tras].y = novarua.y;
                    tras++;
                    cont++;
                    if (novarua.y == this->y && novarua.x == this->x) {
                        return zambs + 1;
                    }
                }
            }
        }
        if (frente == marcador) {
            zambs++;
            marcador = frente + cont;
            cont = 0;
        }
        frente++;
    }
};

//fim do método de busca

int main() {
    //implementação de variáveis importantes
    int casax, casay;
    int x, y, z;
    int limite;
    struct Rua rua[23][39];
    struct Restaurante locais[99];
    struct Restaurante lista[23];
    int preferencia;

//leitura do arquivo
    FILE* ruas = fopen("ruas.txt", "r");
    if (ruas == NULL) {
        printf("Nao foi possivel abrir o arquivo com os dados das
coordenadas das ruas\n");
        return 1;  // Sair do programa com código de erro
    }

//escaneamento das ruas
    while (!feof(ruas)) {
        fscanf(ruas, "%i", &x);
        fscanf(ruas, "%i", &y);
        fscanf(ruas, "%i", &z);
        rua[y - 1][x - 1].pavimento(z);
    }
    fclose(ruas);
    bool coordenadaValida = false;
```

```c
//método que usa repetição para receber as coordenadas da casa e
verificar se elas são pavimentadas
//se a coordenada não for pavimentada, ela é dada como falsa e só poderá
sair do escopo se for verdadeira
    while (!coordenadaValida) {
        printf("Digite as coordenadas da casa: ");
        scanf("%i", &casax);
        scanf("%i", &casay);
        if (rua[casay - 1][casax - 1].paviment == false) {
            while (rua[casay - 1][casax - 1].paviment == false) {
                printf("Essa coordenada nao esta pavimentada ou esta fora
da cidade\n");
                printf("Digite outra coordenada: ");
                scanf("%i %i", &casax, &casay);
            }
        }
        coordenadaValida = true;
    }

//analisa a prefeência por restaurantes caros ou baratos
    printf("Digite 1 para restaurantes caros ou 0 para baratos: ");
    scanf("%i", &preferencia);
    while (preferencia != 0 && preferencia != 1) {
        printf("Digite um valor valido, 1 ou 0: ");
        scanf("%i", &preferencia);
    }

//leitura dos arquivos contendo os dados dos restaurantes
    FILE* restaurantes_arquivo = fopen("restaurantes.txt", "r");
    if (restaurantes_arquivo == NULL) {
        printf("Nao foi possivel abrir o arquivo que contem os dados
sobre os restaurantes\n");
        return 1;  // Sair do programa com código de erro
    }

//repetição que analiza os dados de toods os restaurantes e armazena-os
em seus devidos locais
    for(int k = 0;!feof(restaurantes_arquivo); k++){
    fscanf(restaurantes_arquivo, "%i", &x); //salvo na matriz
    fscanf(restaurantes_arquivo,"%i", &y); //salvo na matriz
    fscanf(restaurantes_arquivo, "%s", &locais[k].nome);//salvo em um
char dos nomes
    fscanf(restaurantes_arquivo,"%s", &locais[k].preco); // salvo em um
char com os preços
    fscanf(restaurantes_arquivo,"%i", &z); //armazenado como inteiro
    locais[k].x = x-1; locais[k].y = y-1;
    locais[k].velocidade = z;}
```

```
//distingue os caros dos baratos e retorna o requisitado
    int k = 0;
    for (int i = 0; i < 23; i++) {
        if (preferencia == 1) {
            if (strcmp(locais[i].preco, "Caro") == 0) {
                lista[k] = locais[i];
                k++;
            }
        }
        if (preferencia == 0) {
            if (strcmp(locais[i].preco, "Barato") == 0) {
                lista[k] = locais[i];
                k++;
            }
        }
    }

    for(int inicio = 0 ;inicio < k ;inicio++){
    for(int verificador = inicio + 1; verificador < k;verificador++){
    if(((lista[verificador].distancia(casax,casay,rua)/1.0)
/lista[verificador].velocidade) <
((lista[inicio].distancia(casax,casay,rua)/1.0) /
lista[inicio].velocidade)){
    Restaurante memory = lista[inicio];
    lista[inicio] = lista[verificador];
    lista[verificador] = memory;
}}
}

    printf("Qual o limite de tempo de espera ?\n");
    scanf("%i", &limite);

    printf("Segue a lista ordenada dos restaurantes com menor tempo de
entrega para os maiores\n\n");
    for(int i = 0; i<k;i++){
    printf("Nome do restaurante: %s \nTempo de espera %.2f minutos\n\n",
lista[i].nome, ((lista[i].distancia(casax,casay,rua)/1.0) /
lista[i].velocidade));}

    printf("Lista de restaurantes dentro do limite de tempo e com a
preferencia correta:\n\n");
    for(int i = 0; i<k;i++){
    if((lista[i].distancia(casax,casay,rua) / lista[i].velocidade) <=
limite){
    printf("%s\n", lista[i].nome);
}
}
    return 0;
}
```

Resultado caso a coordenada seja (1,1), a preferência seja por restaurantes caros e o tempo de espera seja de até 30 minutos:

```
Digite as coordenadas da casa: 1
1
Digite 1 para restaurantes caros ou 0 para baratos: 1
Qual o limite de tempo de espera ?
30
Segue a lista ordenada dos restaurantes com menor tempo de entrega para
os maiores

Nome do restaurante: Cantina_da_Carol
Tempo de espera 0.00 minutos

Nome do restaurante: Churrasco_de_gato
Tempo de espera 6.50 minutos

Nome do restaurante: Taste_Vin
Tempo de espera 7.60 minutos

Nome do restaurante: Espeto_do_Chico
Tempo de espera 9.50 minutos

Nome do restaurante: Voador
Tempo de espera 9.80 minutos

Nome do restaurante: Las_pombas
Tempo de espera 10.50 minutos

Nome do restaurante: Xucesso_da_Cida
Tempo de espera 15.60 minutos

Nome do restaurante: Caro_e_ruim
Tempo de espera 30.00 minutos

Nome do restaurante: Comida_de_buteco
Tempo de espera 58.00 minutos

Nome do restaurante: Moto_velha
Tempo de espera 63.00 minutos

Nome do restaurante: Macarrao_na_chapa
Tempo de espera 116.00 minutos

Lista de restaurantes dentro do limite de tempo e com a preferencia
correta:

Cantina_da_Carol
Churrasco_de_gato
```

```
Taste_Vin
Espeto_do_Chico
Voador
Las_pombas
Xucesso_da_Cida
Caro_e_ruim
```