Relatório - Projeto de C204 L4

Equipe: Juliely Barbosa de Lima
Luiz Augusto Moreira Barbosa
Vinícius Santos Araújo

Na primeira entrega apresentamos um código que calculava o melhor caminho para buscar passageiros e levar estes ao seu destino final, mas esse primeiro código também considerava viagens que não utilizavam o serviço Pool.
Após a primeira entrega, realizamos algumas mudanças no código de maneira que apenas o serviço Pool fosse possível. Com as alterações, o carro identifica os 3 passageiros mais próximos e através do método guloso ele busca os passageiros da melhor maneira. Quando os passageiros são selecionados damos entrada nas coordenadas de seu destino, aplicamos novamente o método guloso para decidir a melhor rota para levar os 3 passageiros até seus destinos.

Algoritmo utilizado

```
#include <bits/stdc++.h>
using namespace std;

// struct usada para declarar os pontos para formação dos vetores
struct ponto{
	float x;
	float y;
};
typedef ponto vetor;

// struct usada para criar as retas com os pontos
struct reta{
	ponto A;
	ponto B;
};
typedef reta segmento;

// função usada para criar o vetor
vetor cria_vetor(ponto A, ponto B){
	vetor AB;

	AB.x = B.x - A.x;
	AB.y = B.y - A.y;

	return AB;
}

// função que retorna o comprimento do vetor
float distancia_Vetor(vetor A){
	float comprimento_Vetor;
	float x;

	x = pow(A.x, 2) + pow(A.y, 2);
	comprimento_Vetor = sqrt(x);

	return comprimento_Vetor;
```

```
}

// função que retorna o ponto com a menor distância de outro ponto origem
int menor_distancia(ponto origem, ponto destinos[], int numpassageiros)
{
	float menordistancia; // guarda o valor da menor distância
	int menor; // guarda o ponto com a menor distância
	int i = 0; // variável auxiliar
	vetor distancias; // guarda os vetores que serão criados para encontrar a menor distância
	float comprimento; // guarda o comprimento dos vetores

	// atribuindo valores iniciais para as variáveis
	distancias = cria_vetor(origem, destinos[i]);
	comprimento = distancia_Vetor(distancias);
	menordistancia = comprimento;
	menor = i;

	// percorrendo o vetor de destinos
	for(int i = 1; i < numpassageiros; i++){
		distancias = cria_vetor(origem, destinos[i]);
		comprimento = distancia_Vetor(distancias);

		// comparando as distâncias para encontrar a menor
		if(comprimento < menordistancia){
			menordistancia = comprimento;
			menor = i;
		}
	}

	// retornando o ponto com a menor distância
	return menor + 1;
}

int main()
{
	ponto origem; // ponto que armazena as posições do motorista ao longo do percurso
	int num_passageiros; // armazena o número de possíveis passageiros na região
	int aux; // variável auxiliar
	int npassageiro[3]; // armazena a ordem de busca dos passageiros

	// entrando com a posição inicial do motorista
	cout << "Entre com a posicao do motorista: " << endl;
	cin >> origem.x >> origem.y;

	// entrando com o número de possíveis passageiros na região
	cout << "Entre com o numero de possiveis passageiros proximos (minimo 3): " << endl;
	cin >> num_passageiros;

	// while para aceitar apenas um valor igual ou maior que 3
	while(num_passageiros < 3){
		cout << "numero invalido !" << endl << "entre com um novo valor: " << endl;
		cin >> num_passageiros;
	}

	ponto coordenadas_passageiros[num_passageiros]; // armazena a posição dos possíveis
passageiros
	ponto coordenadaspassageiros2[3]; // armazena a posição dos passageiros selecionados para a
viagem
	ponto coordenadas_destinos[3]; // armazena a posição dos destinos dos passageiros selecionados
```

```
	// entrando com a posição dos possíveis passageiros
	for(int i = 0; i < num_passageiros; i++){
		cout << "Entre com a posicao do passageiro " << i + 1 << ": " << endl;
		cin >> coordenadas_passageiros[i].x >> coordenadas_passageiros[i].y;
	}

	// calculando quais os 3 passageiros mais próximos do motorista e mostrando quais são eles
	cout << endl << "passageiros selecionados: ";
	for(int z = 0; z < 3; z++){
		// encontrando o passageiro mais próximo e mostrando ele
		aux = menor_distancia(origem, coordenadas_passageiros, num_passageiros);
		cout << aux << " ";

		// guardando a posição do passageiro
		coordenadaspassageiros2[z].x = coordenadas_passageiros[aux - 1].x;
		coordenadaspassageiros2[z].y = coordenadas_passageiros[aux - 1].y;

		// atribuindo um valor alto para a posição do passageiro encontrado para ele não ser selecionado novamente na função
		coordenadas_passageiros[aux - 1].x = 10000;
		coordenadas_passageiros[aux - 1].y = 10000;

		// guardando quais os passageiros selecionados
		npassageiro[z] = aux;
	}

	cout << endl << endl;

	// entrando com o destino dos passageiros selecionados
	for(int v = 0; v < 3; v++){
		cout << "Entre com o destino do passageiro " << npassageiro[v] << ": " << endl;
		cin >> coordenadas_destinos[v].x >> coordenadas_destinos[v].y;
	}

	// calculando o melhor trajeto para pegar os passageiros e mostrando ele
	cout << endl << "motorista " << endl;
	for(int f = 0; f < 3; f++){
		// achando o passageiro com a menor distância da origem
		aux = menor_distancia(origem, coordenadaspassageiros2, 3);

		// mostrando qual é o passageiro
		cout << " --> passageiro " << npassageiro[aux - 1] << endl;

		// definindo a posição do passageiro como origem
		origem = coordenadaspassageiros2[aux - 1];

		// atribuindo um valor alto para a posição do passageiro para ele não ser selecionado novamente na função
		coordenadaspassageiros2[aux - 1].x = 10000;
		coordenadaspassageiros2[aux - 1].y = 10000;
	}

	// calculando o melhor trajeto para deixar os passageiros nos destinos e mostrando ele
	cout << endl << "indo para os destinos" << endl << endl;
	for(int j = 0; j < 3; j++){
		// achando o destino com a menor distância da origem
		aux = menor_distancia(origem, coordenadas_destinos, 3);

		// mostrando qual é o destino
		cout << " --> destino do passageiro " << npassageiro[aux - 1] << endl;
```

```
            // definindo a posição do destino como origem
            origem = coordenadas_destinos[aux - 1];

            // atribuindo um valor alto para a posição do destino para ele não ser selecionado
novamente na função
            coordenadas_destinos[aux - 1].x = 10000;
            coordenadas_destinos[aux - 1].y = 10000;
        }

        return 0;
}
```

Complexidade: O (n^2)

Casos de testes

```
caso teste 1:

posição inicial: 0 0

número de passageiros: 7

localização dos passageiros:

1 2
5 6
3 7
4 9
4 6
2 8
3 9

localização dos destinos:

34 57
36 51
37 55
```

```
Entre com a posicao do passageiro 5:
4 6
Entre com a posicao do passageiro 6:
2 8
Entre com a posicao do passageiro 7:
3 9

passageiros selecionados: 1 5 3

Entre com o destino do passageiro 1:
34 57
Entre com o destino do passageiro 5:
36 51
Entre com o destino do passageiro 3:
37 55

motorista
 --> passageiro 1
 --> passageiro 5
 --> passageiro 3

indo para os destinos

 --> destino do passageiro 5
 --> destino do passageiro 3
 --> destino do passageiro 1

O Processo retornou 0   tempo de execução : 43.586 s
Pressione uma tecla para continuar...
```

caso teste 2:

posição inicial: 0 0

número de passageiros: 4

localização dos passageiros:

7 8
6 5
5 7
3 2

localização dos destinos:

28 90
21 89
25 93

```
Entre com a posicao do passageiro 2:
6 5
Entre com a posicao do passageiro 3:
5 7
Entre com a posicao do passageiro 4:
3 2

passageiros selecionados: 4 2 3

Entre com o destino do passageiro 4:
28 90
Entre com o destino do passageiro 2:
21 89
Entre com o destino do passageiro 3:
25 93

motorista
 --> passageiro 4
 --> passageiro 2
 --> passageiro 3

indo para os destinos

 --> destino do passageiro 2
 --> destino do passageiro 3
 --> destino do passageiro 4

O Processo retornou 0   tempo de execução : 35.712 s
Pressione uma tecla para continuar...
```

caso teste 3:

posição inicial: 0 0

número de passageiros: 6

localização dos passageiros:

10 12
13 15
12 16
11 19
15 14
14 11

localização dos destinos:

56 62
52 67
55 64

```
Entre com a posicao do passageiro 4:
11 19
Entre com a posicao do passageiro 5:
15 14
Entre com a posicao do passageiro 6:
14 11

passageiros selecionados: 1 6 2

Entre com o destino do passageiro 1:
56 62
Entre com o destino do passageiro 6:
52 67
Entre com o destino do passageiro 2:
55 64

motorista
 --> passageiro 1
 --> passageiro 6
 --> passageiro 2

indo para os destinos

 --> destino do passageiro 1
 --> destino do passageiro 2
 --> destino do passageiro 6

O Processo retornou 0   tempo de execução : 39.628 s
Pressione uma tecla para continuar...
```

Conclusão

Nosso algoritmo implementa o serviço Pool, tornando possivel que o carro busque 3 pessoas em uma mesma viagem, e leve cada uma ao seu destino desejado utilizando a melhor rota possivel. Para isso foi necessário utilizar o método guloso em dois momentos, para buscar os passageiros e para levá-los ao destino final. Nosso código permite a entrada de mais de 3 passageiros, mas seleciona apenas os 3 mais próximos do carro para realizar o serviço.