Projeto e Análise de Algoritmos
Trabalho 2

1 Descrição

Este trabalho considera a análise de um algoritmo de aproximação para o problema do *Caixeiro Viajante*.

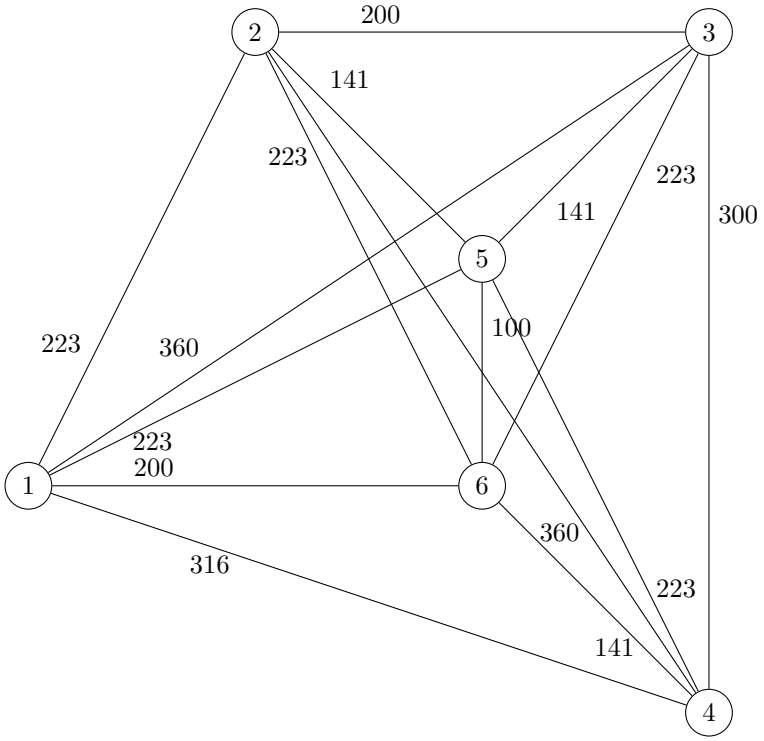
Cada trabalho submetido deverá ser desenvolvido por no máximo três estudantes. Os códigos devem conter cabeçalhos discriminando cada um dos estudantes que desenvolveu o trabalho submetido, com nome completo e registo acadêmico.

A avaliação do trabalho irá considerar os códigos e a execução das implementações. Todo o conteúdo submetido pelos estudantes deve ser de autoria deles.

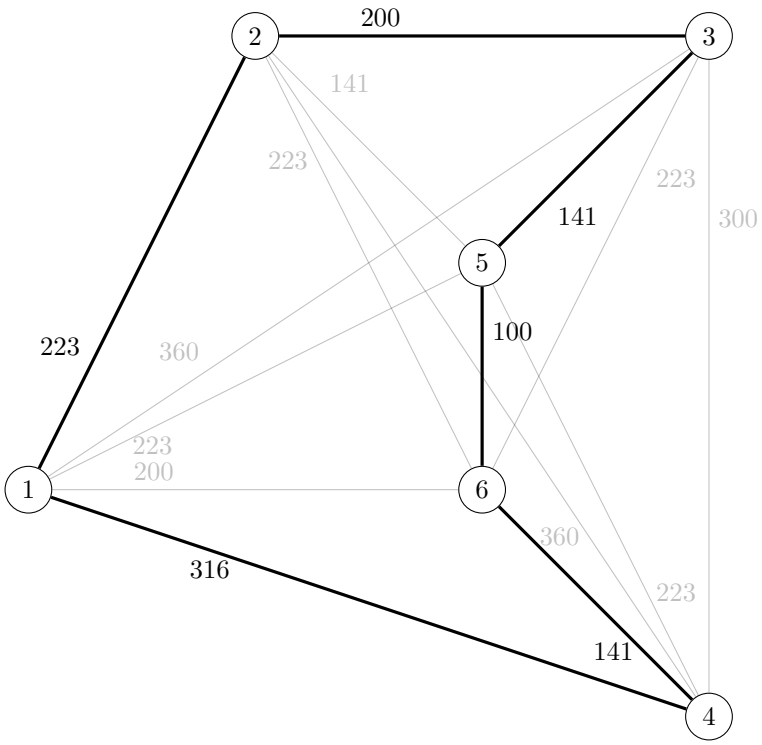
Na sequência deste documento serão descritos os detalhes sobre o trabalho. Leia com atenção. Trabalhos que não respeitarem esta descrição serão penalizados.

2 Problema do Caixeiro Viajante

Um ciclo hamiltoniano em um grafo $G = (V, E)$ é um ciclo que passa por todos os vértices de G sem repetir vértices. Vamos definir o problema do caixeiro viajante como o problema de computar um ciclo hamiltoniano de custo mínimo em um grafo $G = (V, E)$ com pesos positivos. O custo do ciclo é dado pela soma dos pesos de suas aresta. No exemplo abaixo, temos um grafo completo com o peso de cada aresta representando a distância Euclidiana entre seus vértices.



Um ciclo mínimo para o grafo, dado por $1, 2, 3, 5, 6, 4, 1$, é representado em destaque no grafo abaixo. O custo do ciclo é $223 + 200 + 141 + 100 + 141 + 316 = 1121$.



O problema pode ser resolvido computando todas as permutação de vértices, mas isso leva a um algoritmo $O(n!)$. Uma solução conhecida, que usa programação dinâmica, leva tempo $O(n^2 2^n)$, o que ainda torna o algoritmo impraticável mesmo para entradas não muito grandes.

3 Algoritmo de Aproximação

Neste trabalho deverá ser implementado um algoritmo de aproximação para solução do problema. O algoritmo não necessariamente computa a solução ótima mas a solução pode ser computada rapidamente, mesmo para entradas bastante grandes.

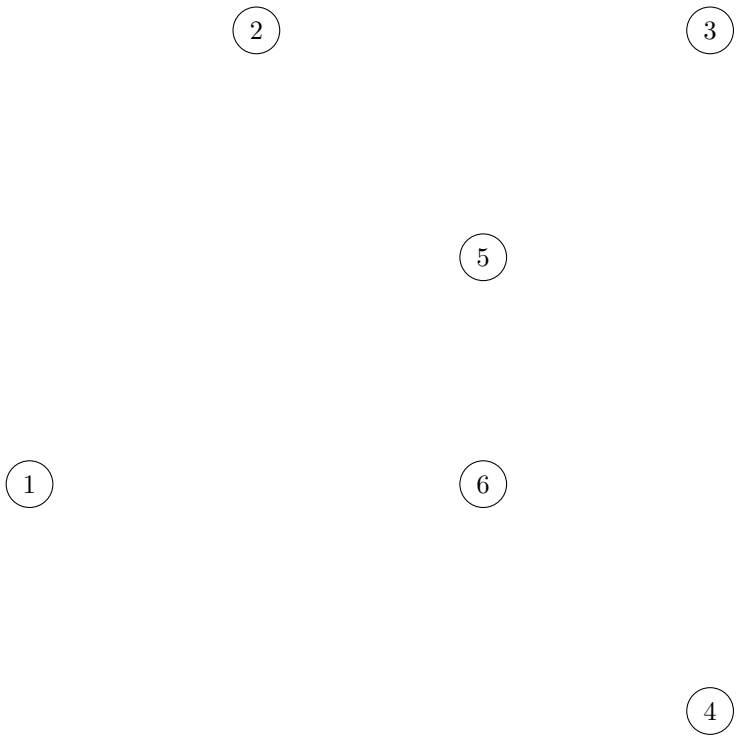
A entrada para o algoritmo será um conjunto de n pontos $P = \{p_1, p_2, \ldots, p_n\}$ e, a partir desses pontos, a solução deve ser computada seguindo os seguintes passos:

1. computação de uma grafo completo $G = (V = P, E)$, tal que o custo de uma aresta $(u, v) \in E$ é dado pela distância Euclidiana entre os vértices $u, v \in V$;

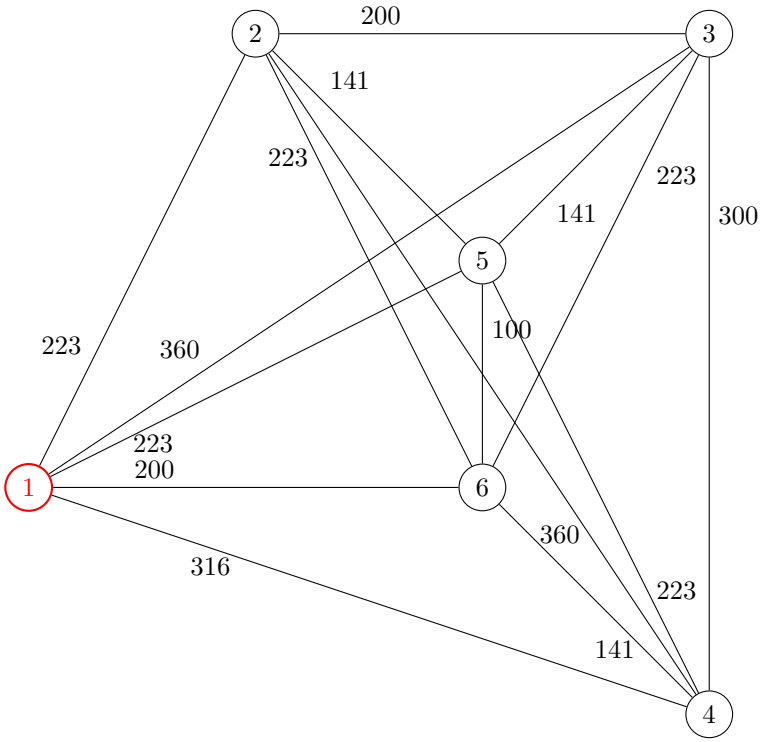
2. computação de uma Árvore Gerado Mínima (AGM) $T = (V, E' \subset E)$, para o grafo G, usando o algoritmo de Prim iniciando em um vértice $s \in V$;

3. computar uma Busca em Profundidade (BP) na árvore T, a partir do mesmo vértice s usado para iniciar o algoritmo de Prim, para criar a sequência de vértices $s, v_1, v_2, \ldots, v_{|V|-1}, s$, $v_i \neq s$ para $i = 1, 2, \ldots, |V| - 1$, que será a solução para o problema do caixeiro viajante (note que o primeiro vértice é repetido no final da sequência para formar o ciclo.).

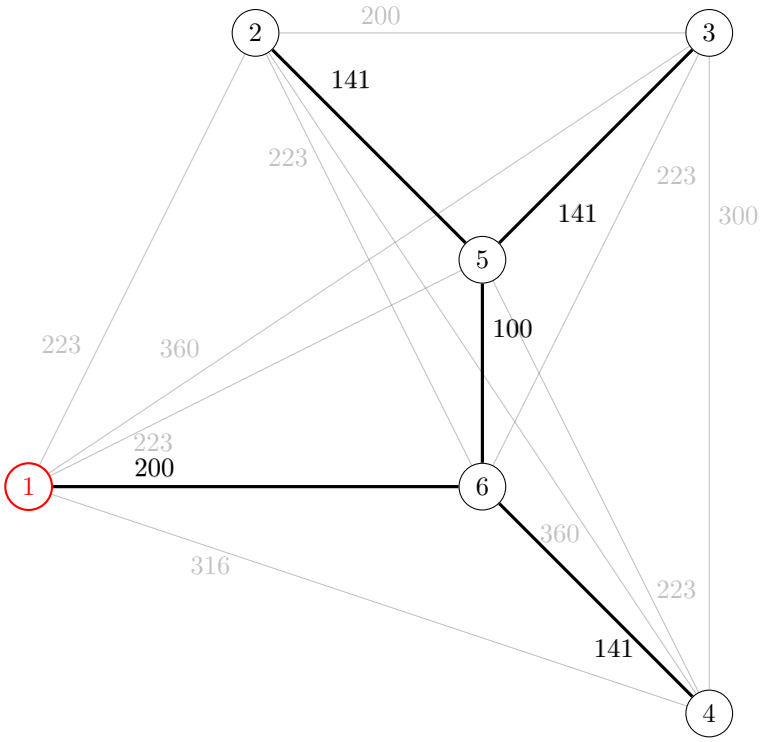
Considerando o exemplo da seção anterior, o conjunto de pontos de entrada forma o conjunto V de vértices do grafo G.



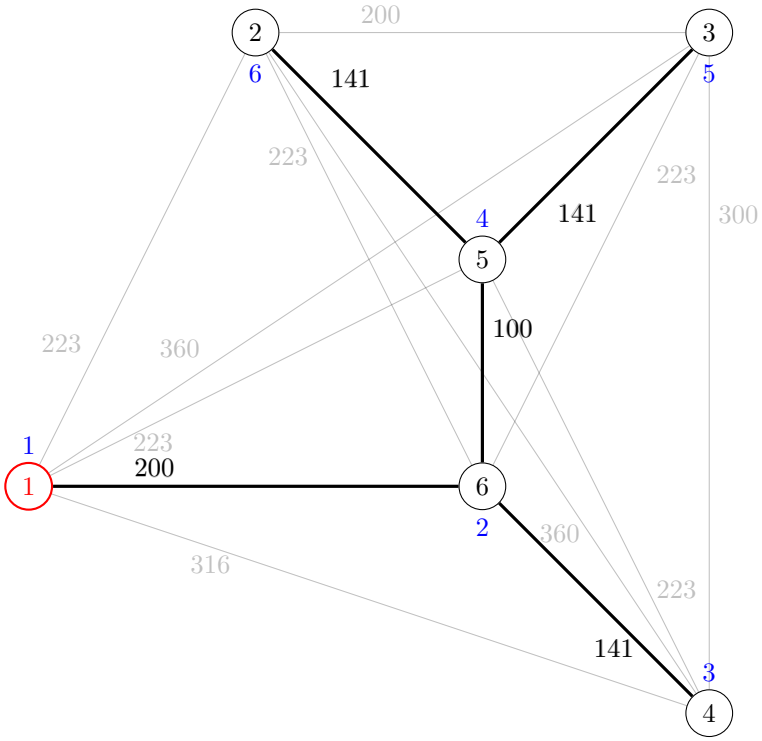
A partir dos pontos é possível computar as arestas E do grafo G e atribuir a cada aresta custo igual à distância Euclidiana entre seus vértices. Realizando, desta forma, o passo 1 do algoritmo.



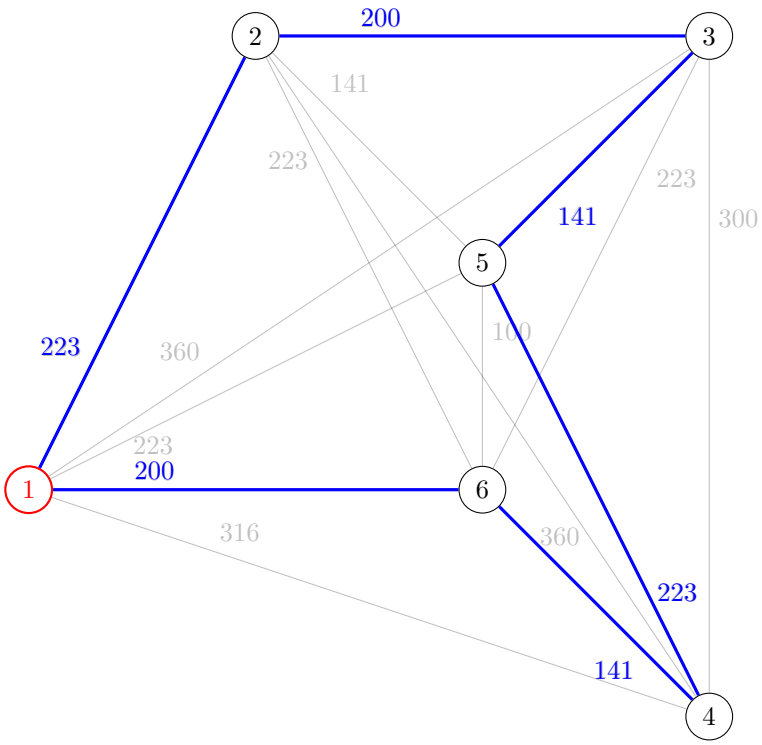
No passo 2 do algoritmo, computamos a AGM pelo algoritmo de Prim, escolhendo um vértice de início, por exemplo, o vértice 1 (destacado na figura acima). A AGM resultante T seria como na figura abaixo.



Por fim, ao fazer a BP do passo 3, a ordem de visita (destacada em azul no grafo abaixo) irá produzir o ciclo $1, 6, 4, 5, 3, 2, 1$.



O custo do ciclo $1, 6, 4, 5, 3, 2, 1$, dado pela soma dos custos (distâncias Euclidianas) das arestas que o compõem, é $200 + 141 + 223 + 141 + 200 + 223 = 1128$. Logo, o ciclo mostrado na figura abaixo não é ótimo, mas pode ser computado rapidamente.



4 Implementação e Execução

As implementações devem ser feitas em linguagem C ou C++ e compilar, respectivamente, usando os compiladores gcc e g++, em sistema operacional Linux.

Um arquivo Makefile deve ser disponibilizado para compilar e gerar o arquivo executável. A compilação deve ser feita usando o comando *make* em um terminal de comandos do Linux, conforme o exemplo abaixo. O símbolo $ representa o prompt de comando do terminal.

```
$ make
```

O programa gerado pelo comando *make* deve ter o nome "tsp" e receber como argumento um arquivo TXT com os pontos de entrada (o formato do arquivo será descrito na sequência). A execução deve ser iniciada em um terminal de comandos do Linux, como no exemplo abaixo.

```
$ ./tsp input.txt
```

O programa "tsp" deve computar o ciclo conforme o algoritmo descrito na seção anterior. Como saída, o programa deve:

- apresentar o tempo de execução e custo do ciclo computado, na saída padrão do terminal Linux;
- gravar um arquivo contendo as arestas da AGM computada;
- gravar um arquivo contendo os pontos computados para o ciclo.

Os detalhes de formatação das saídas são descritos na sequência.

4.1 Arquivo de Entrada

O arquivo de entrada deve ser formatado da seguinte maneira:

- a primeira linha contém um inteiro n com o número de pontos no arquivo a serem lidos como entrada para o programa;
- as n linhas que seguem contêm as coordenadas dos pontos em valores inteiros. Cada linha representa um ponto e contém o seu valor de x e depois de y, separados por um espaço.

Um exemplo de arquivo de entrada com 6 pontos é mostrado abaixo.

```
6
0 1
1 3
3 3
3 0
2 2
2 1
```

No Moodle da disciplina foi disponibilizado um código para gerar um arquivo de pontos aleatórios neste formado. O código tem nome "genpoints.c". Para compilar basta executar o make que acompanha o código.

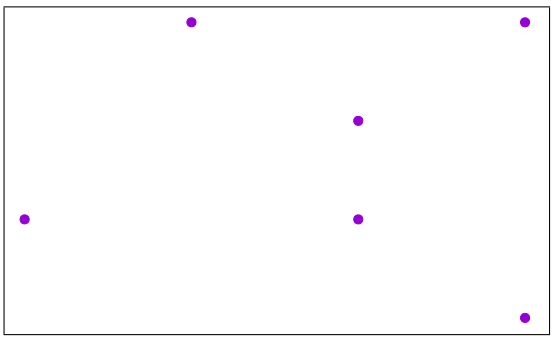
```
$ make
```

O programa recebe como argumento o número de pontos desejados, conforme o exemplo abaixo,

```
./genpoints 523
```

e gera um arquivo chamado "input.txt" no formato da entrada.

Caso queira visualizar os pontos do arquivo usando o programa *gnuplot*, está disponível no Moodle o script *points.plot*, que gera um arquivo chamado *pontos.pdf* com os pontos. A figura abaixo mostra a saída do *gnuplot* para o arquivo de entrada do exemplo.



Para criar o arquivo basta executar a seguinte linha no terminal[1].

```
$ npts=$(head -1 input.txt); tail -$npts input.txt > input2.txt; gnuplot points.plot
```

[1]Para execução em Linux e você deve ter o *gnulot instalado.*

4.2 Formatação da Saída no Terminal

O programa "tsp" deve imprimir no terminal uma linha com o tempo de execução do algoritmo, em segundos, e o custo do ciclo computado. Os valores devem ser separados por espaço e formatos em ponto flutuante com seis casas decimais. Veja o exemplo abaixo. Você não deve acrescentar nada mais à saída, apenas os valores, exatamente como no exemplo.

```
$ ./tsp input.txt
0.000001 1128.000000
$
```

4.3 Formatação do Arquivo de Saída para a AGM

O arquivo de saída para a AGM computada deve ter o nome *tree.txt*. Cada aresta da árvore deve ser representada por duas linhas subsequentes no arquivo, cada uma representando um dos vértices da aresta. Um vértice deve ser representado por duas colunas, separadas por espaço, contendo respectivamente os valores de x e y para o vértice.

Uma exemplo de arquivo para o nosso exemplo é mostrado abaixo.

```
0 1
2 1
2 1
3 0
2 1
2 2
2 2
3 3
2 2
1 3
```

Caso queira visualizar a árvore do arquivo usando o *gnuplot*, está disponível no Moodle o script *tree.plot* que gera um arquivo chamado *arvore.pdf*. A figura abaixo mostra a saída para o exemplo.

Para que a sua árvore apareça correntamente no gnuplot, você deve mudar o arquivo "tree.txt" para separar as arestas no arquivo usando uma linha. Como no exemplo abaixo. IMPORTANTE: A saída do seu programa submetido deve ser sem linhas separando as arestas, coloque-as somente para seus testes e se for usar o gnuplot.

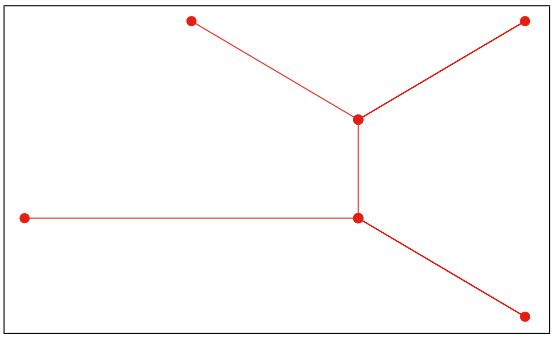
```
0 1
2 1

2 1
3 0

2 1
2 2
```

```
2 2
3 3

2 2
1 3
```



Para criar o arquivo basta executar a seguinte linha no terminal.

```
$ gnuplot tree.plot
```

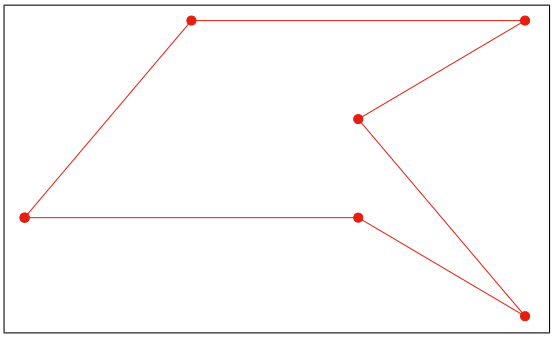
4.4 Formatação do Arquivo de Saída para o Ciclo

O arquivo de saída para o ciclo computado deve ter o nome *cycle.txt*. As coordenadas x e y de um vértice do ciclo devem aparecer em uma linha, separadas por espaço. A primeira e a última linhas do arquivo devem corresponder ao mesmo vértice para fechar o ciclo. A ordem das linhas no arquivo deve seguir a ordem dos vértices no ciclo.

Uma exemplo de arquivo para o nosso exemplo é mostrado abaixo.

```
0 1
2 1
3 0
2 2
3 3
1 3
0 1
```

Caso queira visualizar o ciclo do arquivo usando o *gnuplot*, está disponível no Moodle o script *cycle.plot* que gera um arquivo chamado *ciclo.pdf*. A figura abaixo mostra a saída para o exemplo.



Para criar o arquivo basta executar a seguinte linha no terminal.

```
$ gnuplot cycle.plot
```

5 Documentação do código

Seu código deve estar bem comentado, descrevendo de forma clara as principais linhas do código. Também deverá haver comentários descrevendo de forma clara e completa os seguintes itens:

- estruturas de dados criadas: o que armazenam e como, onde são usadas, etc.;
- funções auxiliares: suas entradas, saídas, como computam as saídas, etc.;
- funções principais (para os passo 1 a 3 do algoritmo): suas entradas, saídas, descrição dos algoritmos, considerações importantes, etc.

6 Entrega

As entregas deverão ser realizadas usando o Moodle da disciplina dentro do prazo limite estabelecido no mesmo. Não serão aceitas entregas fora do prazo.

Os autores do trabalho devem reunir os códigos-fonte em um diretório nomeado com os números dos seus registros acadêmicos (RAs) separados por '_'. Por exemplo, se dois estudantes desenvolveram o trabalho e têm RAs 123456 e 789012, então o diretório deve ter o nome 123456_789012.

Dentro do diretório deverão estar os códigos-fonte e o arquivo Makefile. Por exemplo, o diretório poderia conter os arquivos listados abaixo (não coloque acentos ou espaços em nomes de arquivos).

```
123456_789012/tsp.c
123456_789012/Makefile
```

O arquivo de entrega deve ser um ZIP do diretório. O arquivo ZIP deve inclusive conter o diretório e ser nomeado da mesma maneira que o diretório. No nosso exemplo: 123456_789012.zip.