

## PROJETO 2018

O projeto a ser implementado durante o ano letivo é um sistema de informação geográfica simplificado, como definido abaixo.

*A geographic information system (GIS) is a computer system for capturing, storing, checking, and displaying data related to positions on Earth's surface. GIS can show many different kinds of data on one map. This enables people to more easily see, analyze, and understand patterns and relationships.*

Fonte: <http://www.nationalgeographic.org/encyclopedia/geographic-information-system-gis/>

O sistema será implementado incrementalmente e em fases. É importante enfatizar que em cada fase o sistema evolui, isto é, novas funcionalidades são acrescentadas, requisitos de implementação podem ser mudados, porém, as funcionalidades existentes **devem continuar funcionais**.

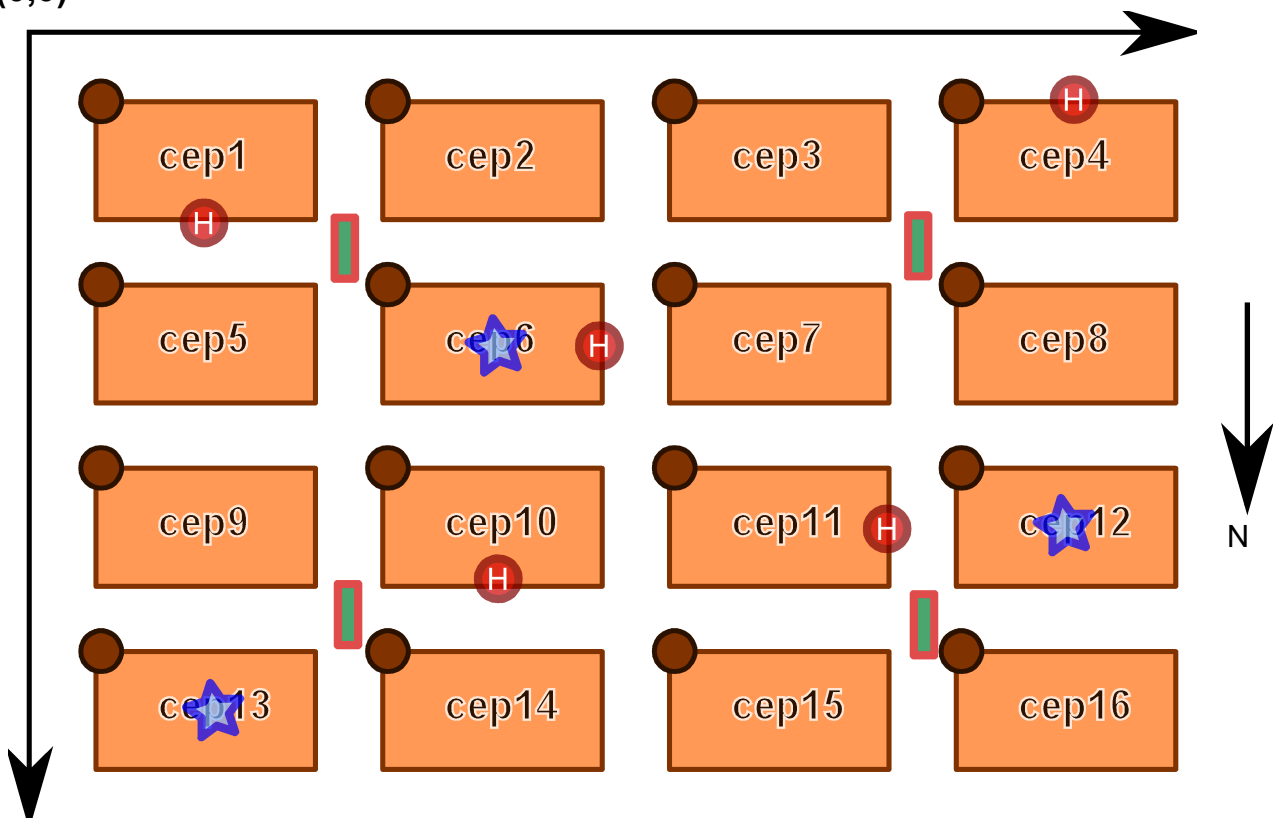
### DESCRIÇÃO

Um Sistema de Informações Geográficas (SIG), para a nossa finalidade, é um sistema que contém (não exclusivamente) dados geo-referenciados, isto é, dados com algum atributo de localização espacial (uma coordenada).

O sistema manipulará o mapa de uma cidade e algumas informações relacionadas.

O mapa de uma cidade é composto por um conjunto de retângulos que representam as quadras; e, um conjunto de equipamentos urbanos (hidrantes, semáforos, torres de celular, pontos de ônibus, etc). Cada equipamento urbano é localizado no mapa por um único ponto, conforme o exemplo abaixo.

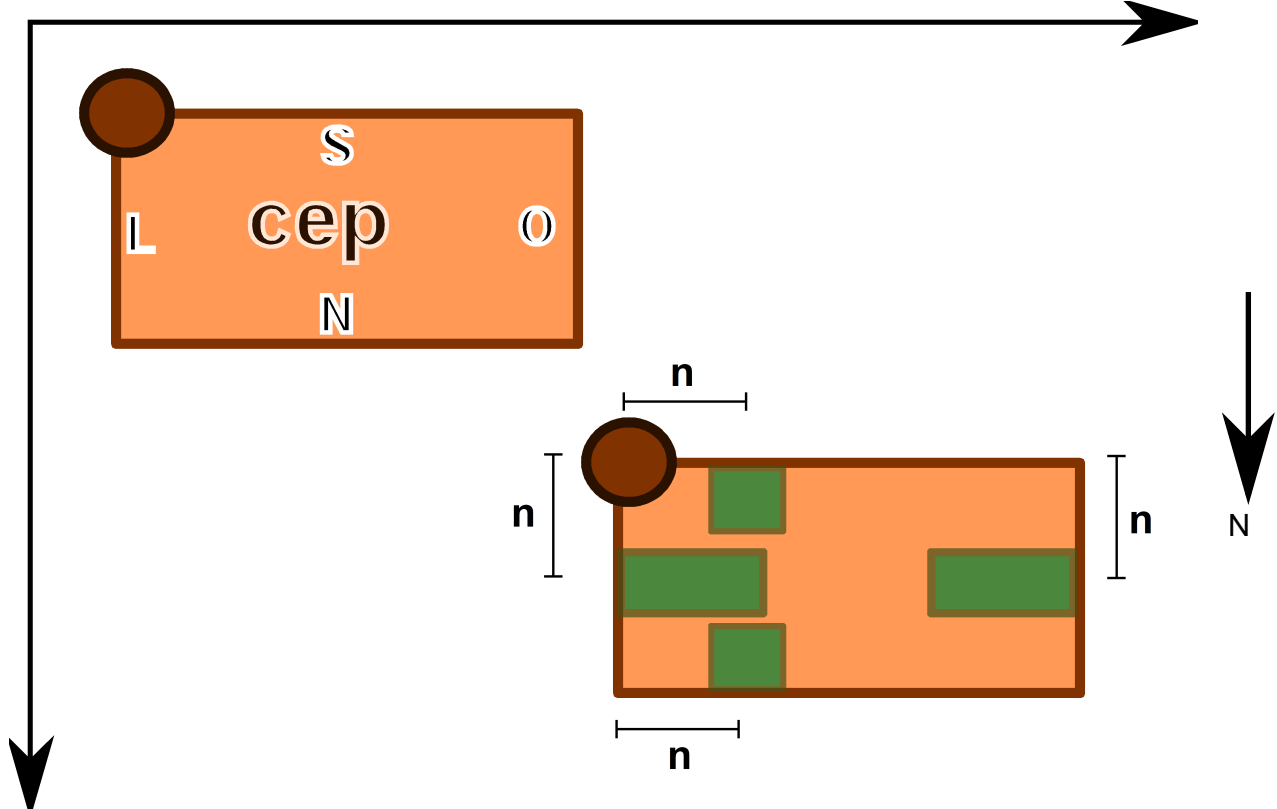
(0,0)



A cidade exemplificada acima chama-se **Bitnópolis** e possui 16 quadras. O

sistema de endereçamento de Bitnópolis é inspirado no de nossa capital federal. Cada **quadra** possui 4 **faces** (N,S,L,O) e é identificada por um **CEP** alfanumérico. O número de uma casa ou estabelecimento comercial é a **distância** da frente da casa até uma projeção do ponto de ancoragem do retângulo que representa a quadra (veja figura abaixo). Assim, um endereço é da forma CEP/Face/número, por exemplo, cep15/S/45. O ponto de ancoragem do retângulo é o canto sudeste da quadra.

(0,0)



### ENTRADA DE DADOS

A entrada de dados, via de regra, ocorrerá por meio de um ou mais arquivos. Estes arquivos estarão sob um diretório, referenciado por **BED** neste texto.<sup>1</sup>

### SAIDA DE DADOS

O dados produzidos serão mostrados na saída padrão e/ou em diversos arquivos-texto. Alguns resultados serão gráficos no formato SVG. Os arquivos de saída serão colocados sob um diretório, referenciado por **BSD** neste texto.<sup>2</sup>

### ORGANIZAÇÃO DA ENTREGA

O trabalho deve ser submetido no formato **ZIP**, cujo nome deve ser curto, mas suficiente para identificar o aluno ou a equipe.<sup>3</sup> Este arquivo deve estar organizado como descrito à frente.

<sup>1</sup> Indicado pela opção -e.

<sup>2</sup> Indicado pela opção -o.

<sup>3</sup> Por exemplo, jrsilva.zip (se aluno se chamar José Roberto da Silva), jrsilva-mrcarneiro.zip (para uma equipe com dois alunos. Evite usar maiúsculas, caracteres acentuados ou especiais).

## PROCESSO DE COMPILAÇÃO E TESTES DO TRABALHO

### Organização do ZIP a ser entregue

A organização do zip a ser entregue pelo aluno deve ser a seguinte:

#### [abreviatura-nome]

LEIA-ME.txt

*Por exemplo, jrsilva.*

*colocar matrícula e o nome do aluno. Atenção: O número da matrícula de estar no início da primeira linha do arquivo. Só colocar os números; não colocar qualquer pontuação.*

\*

*Outros arquivos podem ser solicitados a cada fase.*

**/src**

*(arquivos-fonte)*

makefile

*deve ter target para a geração do arquivo objeto de cada módulo e o target **siguel** que produzirá o executável de mesmo nome dentro do mesmo diretório **src**. Os fontes devem ser compilados com as opções **-pedantic -ansi**.*

\*.h e \*.c

***Atenção:** não devem existir outros arquivos além dos arquivos fontes e do makefile*

### Organização do diretório para a compilação e correção dos trabalhos (no computador do professor):

#### [HOME DIR]

\*.py

*scripts para compilar e executar*

\t

*diretório contendo os arquivos de testes*

\*.geo

*arquivos de teste e, talvez, alguns outros sub-diretórios*

\alunos

*(contém um diretório para cada aluno)*

\abrnome

*diretório pela expansão do arquivo submetido (p.e., jrsilva)*

*outros subdiretórios para os arquivos de saída informados na opção -o*

Os passos para correção serão os seguintes:

1. O arquivo .zip será descomprimido dentro do diretório alunos, conforme mostrado acima
2. O makefile provido pelo aluno será usado para compilar os módulos e produzir o executável. Os fontes serão compilados com o compilador gcc em um máquina virtual Linux. Os executáveis devem ser produzidos no mesmo diretório dos arquivos fontes O professor usará o GNU Make. Serão executadas (a partir dos scripts) o seguinte comando:
  - **make siguel**
3. O programa será executado automaticamente várias vezes: uma vez para cada arquivo de testes e o resultado produzido será inspecionado visualmente pelo

professor. Cada execução produzirá (pelo menos) um arquivo `.svg` diferente dentro do diretório informado na opção `-o`. Possivelmente serão produzidos outros arquivos `.svg` e `.txt`.

APENDICE

<https://www.gnu.org/software/make/manual/make.html>

<http://opensourceforu.com/2012/06/gnu-make-in-detail-for-beginners/>

# FASE I

## A Entrada

A entrada do algoritmo será basicamente um conjunto de retângulos e círculos dispostos numa região do plano cartesiano e, possivelmente, algumas consultas, por exemplo, que indagam se duas das formas geométricas se sobrepõem.

Considere a Ilustração 1. Cada forma geométrica é definida por uma coordenada âncora e por suas dimensões. A coordenada âncora do círculo é o seu centro e sua dimensão é definida por seu raio. A coordenada âncora do retângulo é seu canto inferior esquerdo<sup>4</sup> e suas dimensões são sua largura e sua altura. As coordenadas que posicionam as formas geométricas são valores reais e estão contidas dentro da região delimitada pelos cantos  $(0,0)$  e  $(x_{\max}, y_{\max})$ . Cada forma geométrica é indentificada por um número inteiro.

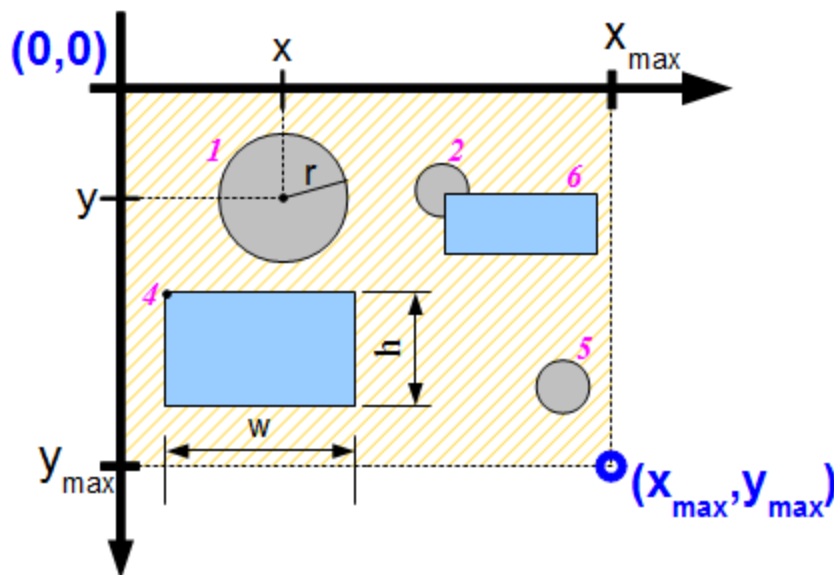


Ilustração 1

A tabela abaixo mostra o formato do arquivo de entrada, composto, basicamente, por conjunto de comandos (uma operação por linha), a saber: **c** (desenhe um círculo), **r** (desenhe um retângulo), **o** (consulta sobreposição), **i** (ponto é interno a figura?), **d** (calcule a distância entre duas figuras) e **a** (desenhe apenas as âncoras das figuras). A última linha possui a marca de final de arquivo (#).

Cada comando tem um certo número de parâmetros. Os parâmetros mais comuns são:

- $i, j, k$ : número inteiro, maior ou igual a 1. Identificador de uma forma geométrica criada pelos comandos **c** ou **r**.
- $r$ : número real. Raio do círculo.
- $x, y$ : números reais. Coordenada  $(x,y)$ .
- **cor**: string. Cor válida dentro do padrão SVG.<sup>5</sup>

<sup>4</sup> Note que o plano cartesiano está desenhado "de ponta-cabeça" em relação à representação usual.

<sup>5</sup> <http://www.december.com/html/spec/colorsvg.html>.

comando	parâmetros	descrição
<b>nx</b>	i	<i>Altera o número máximo de círculos e retângulos (i.e., <math>c + r</math>) criados no arquivo. O valor default é 1000.</i>
<b>c</b>	i cor1 cor2 r x y	<i>desenhar círculo. cor1 é a cor da borda e cor2 é a cor do preenchimento</i>
<b>r</b>	i cor1 cor2 w h x y	<i>desenhar retângulo: w é a largura do retângulo e h, a altura. cor1 é a cor da borda e cor2, a do preenchimento</i>
<b>o</b>	j k	<i>As formas geométricas cujos identificadores são j e k se sobrepõem?<sup>6</sup></i>
<b>i</b>	j x y	<i>O ponto (x,y) é interno à j-ésima forma geométrica?<sup>7</sup></i>
<b>d</b>	j k	<i>Qual é a distância entre os centro de massa das formas geométricas i e j?</i>
<b>a</b>	i sufixo	<i>Cria um arquivo svg contendo os círculos e retângulos referentes aos comandos c e r processados até o momento. Além dos círculos e retângulos, devem ser traçadas linhas a partir do centro de massa da figura de identificador i até o centro de massa de todas as outras figuras. O nome do arquivo gerado deve ser nomebase-sufixo.svg. As linhas devem ser desenhadas usando a cor da borda da figura i. Próximo a cada linha deve ser escrito o seu comprimento.</i>
<b>#</b>		<i>Marca o final do arquivo</i>

A figura abaixo mostra um exemplo de um arquivo de entrada (consistente com a Ilustração 1). Note que a extensão do arquivo é **.geo**. As primeiras operações desenhavam círculos e retângulos. Na parte final do arquivo, estão colocadas duas consultas de sobreposição. A primeira pergunta se o círculo 2 e o retângulo 6 se sobrepõem (de fato, sim) e, a segunda, pergunta se os círculos 1 e 5 se sobrepõem (de fato, não). O último comando solicita que seja produzido um outro arquivo **.svg** (a01-lnhs2.svg) mostrando linhas originárias do centro de massa da figura 2 até cada uma das outras figuras, anotando na respectiva linha o seu comprimento.

<https://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html>

6 A borda da figura pertence à figura. Assim, as figuras que coincidem apenas nas bordas também se sobrepõem.

7 Um ponto na borda da figura pertence à figura, **mas não é interno** à figura.

```

c 1 grey magenta 50.00 50.0 30.00
r 6 cyan yellow 121.0 46.0 100.0 30.0
c 2 grey magenta 120.0 45.0 15.0
r 4 cyan yellow 10.0 150.0 90.0 40.0
c 5 grey magenta 230.0 180.0 13.0
o 2 6
o 1 5
a 2 lnhs2
#

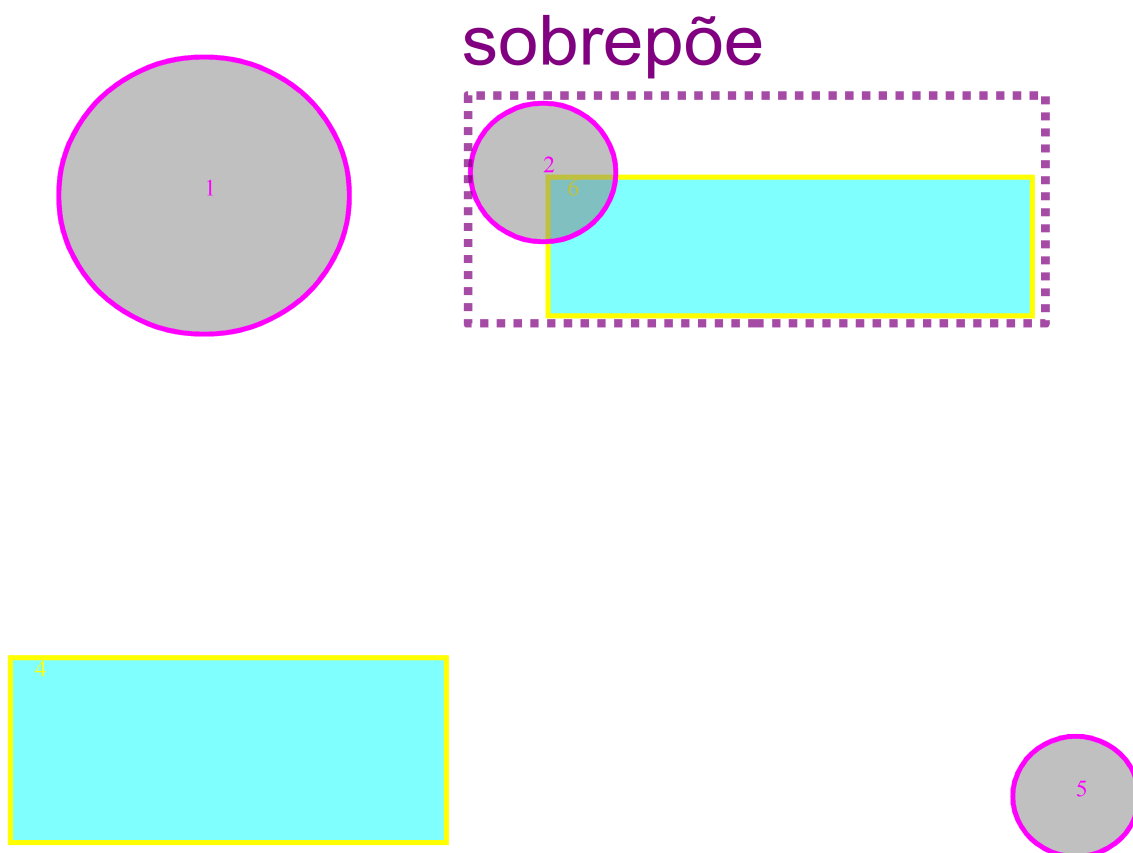
```

*a01.geo*

## A Saída

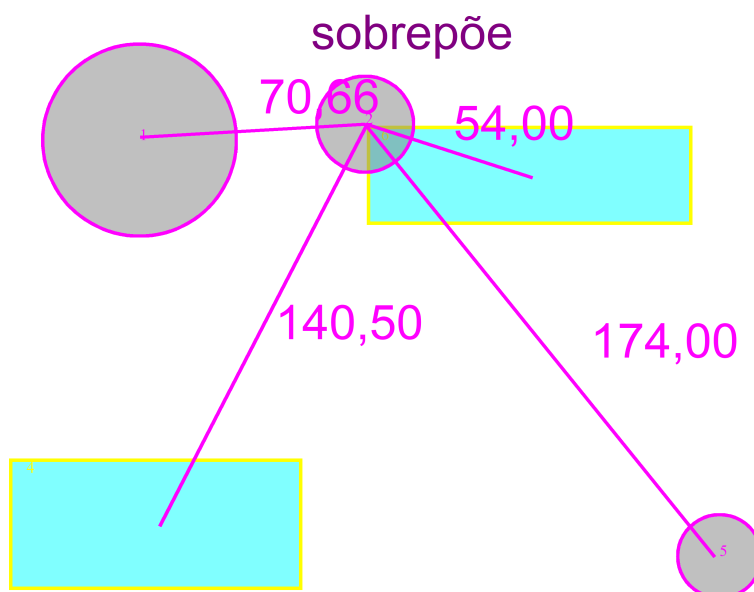
O programa deverá produzir um arquivo **.svg** e um arquivo **.txt** ambos com o mesmo nome base do arquivo **.geo**.

O arquivo **.svg** produzido deve mostrar as formas geométricas. Além disso, para o comando **o**, caso as figuras identificadas se sobreponham, elas devem ser envolvidas por um retângulo tracejado contendo a palavra "sobrepo". Existem várias ferramentas que renderizam arquivos **.svg**. As figuras abaixo mostram um exemplo de arquivo **.svg** e sua respectiva renderização.



*Illustration 2: Arquivo a01.svg*

A operação **a** deve produzir um outro arquivo .svg, com nome *nomebase-sufixo.svg*, (no exemplo, *a01-lnhs2.svg*), como exemplificado na figura abaixo.



*Illustration 3: Arquivo a01-lnhs2.svg*

O processamento de um arquivo .geo deve também produzir um arquivo-texto contendo o resultado das consultas *i*, *d*, *o*. Este arquivo-texto deve ser nomeado *nome-base.txt*. Neste arquivo deve ser copiado em uma linha o texto da consulta e, na linha seguinte, o seu resultado.

o	2	6
SIM		
o	1	5
NAO		
Arquivo a01.txt		

## O Programa

O nome do programa deve ser **siguel** e aceitar três parâmetros:

```
siguel [-e path] -f arq.geo -o dir
```

O primeiro parâmetro (**-e**) indica o diretório base de entrada. É opcional. Caso não seja informado, o diretório de entrada é o diretório corrente da aplicação. O segundo parâmetro (**-f**) especifica o nome do arquivo de entrada que deve ser encontrado sob o diretório informado pelo primeiro parâmetro. O último parâmetro (**-o**) indica o diretório onde os arquivos de saída (**\*.svg** e **\*.txt**) deve ser colocados. Note que o nome do arquivo pode ser precedido por um caminho relativo; ***dir*** e ***path*** é um caminho absoluto ou relativo (ao diretório corrente).

A seguir, alguns exemplos de possíveis invocações de **siguel**:



- `siguel -e /home/ed/testes/ -f t001.geo -o /home/ed/alunos/aluno1/o/`
- `siguel -e /home/ed -f ts/t001.geo -o /home/ed/alunos/al1/o`
- `siguel -f ./tsts/t001.geo -e /home/ed -o /home/ed/alunos/aluno1/o/`
- `siguel -o ./alunos/aluno1/o -f ./testes/t001.geo`