ORGANIZAÇÃO DO PROJETO

Serão desenvolvidos alguns projetos durante o semestre. Todos serão organizados como descrito a seguir.

Basicamente, os programas lerão 2 arquivos: um arquivo que descreve um conjunto de dados a serem armazenados em alguma estrutura de dados e um arquivo descrevendo operações sobre o primeiro conjunto. As operações podem causar a delição, modificação, inserção de dados ao primeiro conjunto.
O resultado final do processamento de cada conjunto é gravado em um ou mais arquivos de saída.

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.[1]

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.[2]

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.[3] Este arquivo deve estar organizado como descrito à frente.

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]	*Por exemplo, josers.*
LEIA-ME.txt	*colocar matrícula e o nome do aluno. Atenção: O número da matricula 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* **t2** *que produzirá o executável de mesmo nome dentro do mesmo diretório* **src**. *Os fontes devem ser compilados com a opção -fstack-protector-all.* ** adotamos o padrão C99. Usar a opção -std=c99.*

1 Indicado pela opção -e.
2 Indicado pela opção -o.
3 Por exemplo, josers.zip (se aluno se chamar José Roberto da Silva), josers-mariabc.zip (para uma equipe com dois alunos. Evite usar maiúsculas, caracteres acentuados ou especiais.

`*.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 *.qry`	*arquivos de consultas, talvez, distribuídos em alguns outros sub-diretórios*
`\alunos`	*(contém um diretório para cada aluno)*
`\abrnome`	*diretório pela expansão do arquivo submetido (p.e., josers)*
	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 t2`**
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/

EXAME

O exame é uma variação do trabalho 2. **Muita atenção nas especificações.**

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. Os comandos estão contidos num arquivo .geo e as consultas num arquivo .qry.

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[4] e suas dimensões são sua largura e sua altura. As coordenadas que posicionam as formas geométricas são valores reais. Cada forma geométrica é identificada por um número inteiro.

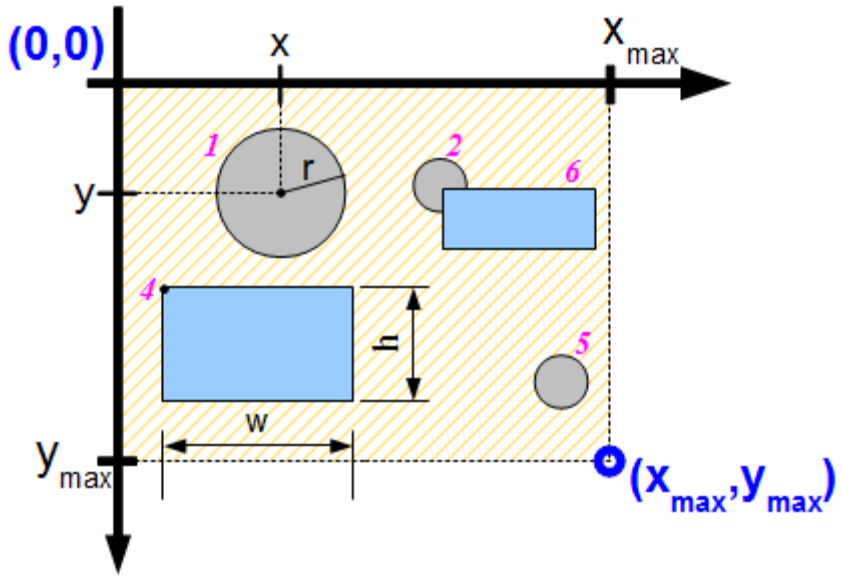


Ilustração 1

As tabelas abaixo mostram os formatos dos arquivos de entrada (.geo e .qry). Os arquivos de entrada são compostos, basicamente, por conjunto de comandos (um por linha), a saber: **`c`** (desenhe um círculo), **`r`** (desenhe um retâgulo), **`t`** (escreva um texto), etc.

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.

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

- `x,y`: números reais. Coordenada (x,y).
- w e h: números reais. Largura e altura de uma região retangular.
- `cor`: string. Cor válida dentro do padrão SVG.[5]

Alguns comandos utilizam memória auxiliar para armazenar identificadores

comando	parâmetros	descrição
c	`i r x y corb corp`	*desenhar círculo.* corb *é a cor da borda e* corp *é a cor do preenchimento*
r	`i w h x y corb corp`	*desenhar retângulo: w é a largura do retângulo e h, a altura.* corb *é a cor da borda e* corp *é a cor do preenchimento*
l	`i x1 y1 x2 y2 cor`	*Desenhar linha com extremidades nos pontos (x1,y1) e (x2,y2), com a cor especifiicada.*
t	`i x y corb corp txto`	*desenha o texto* txto *nas coordenadas (x,y) e com a cores indicadas.* corb *é a cor da borda e* corp *é a cor do preenchimento*
comandos .geo		

Algumas consultas "extraem" pontos "de fixação" das figuras e textos. Um retângulo produz 4 pontos, um círculo produz 3 pontos, a linha produz 2 pontos e um texto produz um único ponto (veja Ilustração 2).

A cada elemento pode ser associado um determinado nível de energia (valor real de 0.0 a 100.0). No momento da atribuição, este nível de energia é dividido igualmente entre os pontos de fixação. Por exemplo, na Ilustração 3, ao retângulo foi atribuído o nível de energia 80.0. Como o retângulo possui 4 pontos de fixação, cada ponto recebeu 20.0 de energia. Similarmente, o círculo recebeu 75.0 de energia e, portanto, 25.0 de energia para cada ponto de fixação. No momento da atribuição inicial, a energia é distribuída igualmente entre os pontos de fixação. Porém, no decorrer das consultas, os níveis de energia de cada ponto podem ser alterados individualmente.

5 http://www.december.com/html/spec/colorsvg.html.
https://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html

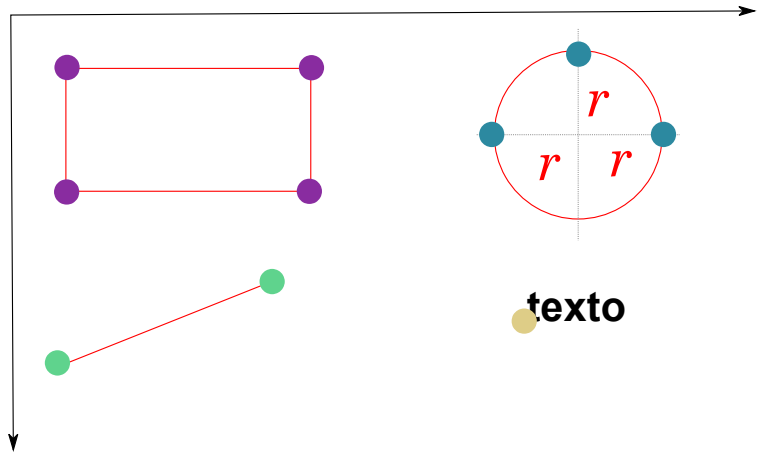


Ilustração 2: Pontos de "fixação" das fiiguras e textos.

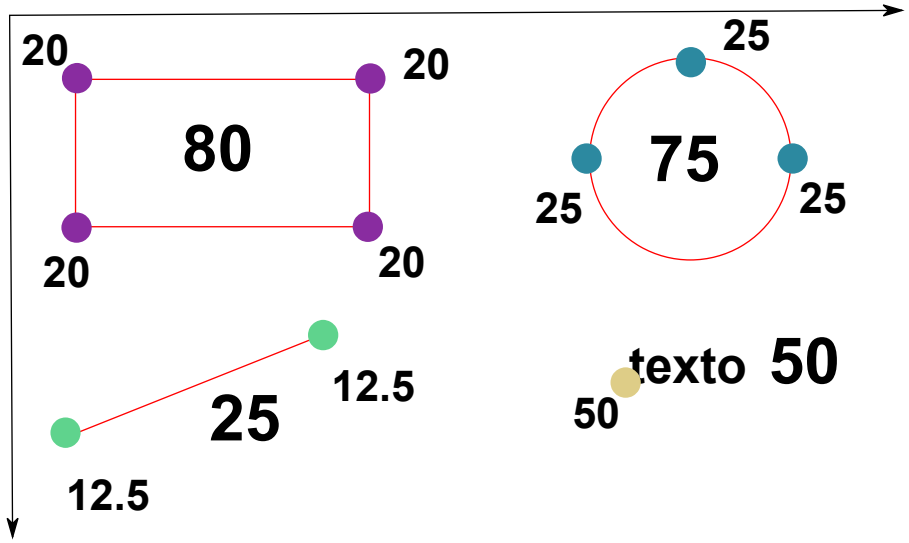


Ilustração 3: Figuras com níveis de energia distribuídas entre seus pontos de fixação

Algumas operações podem provocar o decréscimo de energia de um ponto de fixação. Quando o nível de fixação de um ponto atinge 0.0, significa que aquele ponto não contribui para a fixação da respectiva figura/texto (dizemos que aquela fixação foi removida). Quando todos os pontos de fixação de uma determinada figura tenham sido “removidos”, a respectiva figura é removida do conjunto de dados.

O decréscimo de energia, se dá pela “interação” do par de pontos mais próximos. Cada ponto do par mais próximo “debita” de seu parceiro o seu nível de energia. Por exemplo, suponha que em um dado momento o par mais próximo sejam os pontos p1 (energia 30.5) e p2 (energia 10.3). A energia resultante de p1 será 20.2 (30.5 – 10.3) e de p2 será 0.0 (10.3 – 30.5 < 0). Assim, na prática, cada interação remove o ponto de menor energia e decrementa o de maior energia.

comando	parâmetros	descrição
ef	(id | *) v	*Atribui o valor de energia **v** à figura/texto com identificador id (ou a todos, caso *). **Obs**.: caso uma figuras não atribuídas, o nível de energia é 0.0.*
sf	(* | x y w h)	*Seleciona as figuras/textos que estiverem contidos na região especificada; ou todas as figuras/textos, caso *. SVG: envolver os pontos selecionados com um retângulo, sem preenchimento e com borda TXT: reportar id dos elementos selecionados.*
ep		*Extrai os pontos de fixação das figuras contidas na última seleção (ver **sf**). Descarta pontos por ventura extraídos anteriormente. **TXT**: reporta os dados sobre as figuras selecionadas e respectivos pontos de fixação. SVG: Os pontos extraídos e não removidos devem aparecer na saída.*
ip	n	*Calcula o par de pontos mais próximos e promove a interação entre eles. Pontos "sem energia" não devem ser considerados; **n** é o número de interações a promovidas. Note que a cada iteração, necessariamente um ponto será desenergizado. Se todos os pontos de fixação de uma figura forem desenergizados, a figura deve ser removida. **TXT**: reportar o par mais próximo em cada interação e seus níveis de energia; as respectivas figuras/textos; dados do elemento removido (caso algum tenha sido removido) **SVG**: Figuras removidas não devem aparecer no svg final*
xf	(id | *) d	*Semelhante ao **ef**. Aumenta o nível de energia da figura em d. **TXT**: reportar o id da figura envolvida, o tipo da figura, o nível de energia anterior e o novo nível de energia.*

comando	parâmetros	descrição
sp	x y v	*Semelhante a* **ip**, *mas determina o ponto de fixação mais próximo do ponto (x,y). Debita a energia* **v** *do ponto encontrado. Remove figura respectiva, caso todos pontos de fixação tenham sido desenergizados.* ***SVG**: plotar o ponto (x,y) em vermelho. Envolver o ponto encontrado com uma circunferência também vermelha. Traçar uma linha vermelha ligando o ponto (x,y) ao ponto encontrado. Figuras eventualmente removidas não devem aparecer no resultado final.* ***TXT**: semelhante a* ip.
q?	id	***TXT:** Reporta os dados relativos à figura/texto* **id**. *Incluindo (se existentes) os pontos de fixação (necessário ter havido ep anterior) e os respectivos níveis de energia.*
qr?	x y w h	*Semelhate a* **q?**. *Reporta os dados das figuras/texto contidas na região especificada.*
nf		*Normaliza o nível de energia de todas a figuras/textos existentes. O nível de energia da figura é a soma do nível de energia de seus pontos de fixação e este novo nível de energia é redistribuído igualmente pelos pontos de fixação (incluindo os "zerados").* ***TXT**: reportar níveis de energia das figuras e seus pontos de fixação antes e depois da operação.*
Comandos .qry		

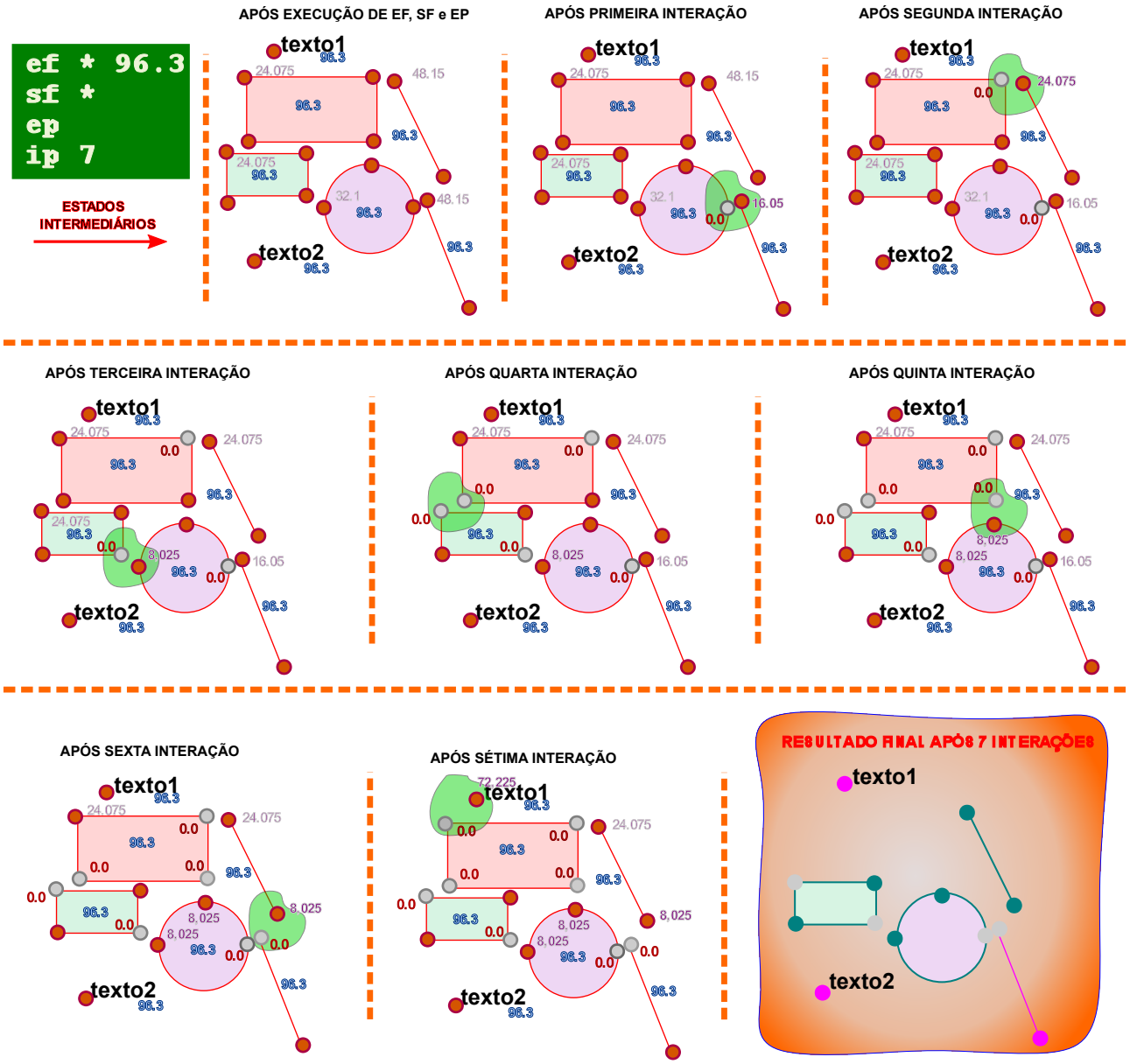
Ao final do processamento do arquivo .qry deve ser produzido um arquivo svg, conforme descrito na próxima seção.

A SAÍDA

Após o processamento das consultas, figuras/textos (não removidas) devem aparecer no refletindo o último estado do elemento. Estes elemento devem estar sobrepostos por seus respectivos pontos de fixação (caso tenha havido ep). A cor de preenchimento dos pontos de fixação de uma figura/texto e a cor da borda da respectiva figura dependem o número de pontos de fixação não zerados da figura/texto (veja tabela abaixo). Pontos de fixação zerados devem ser pintados cinza claro. Também colocar próximo à ancora da figura um texto pequeno indicando o nível de energia total da figura.

ATENÇÃO: no arquivo txt indicar claramente a qual comando o resultado se refere. Copiar o texto do comando precedido pelo texto **[*OP*]**.

#pontos	Cor (borda da forma, preenchimento do ponto)	Espessura borda da forma
4	#0000FF	5px
3	#FF0000	4px
2	#008080	3px
1	#FF00FF	2px



IMPLEMENTAÇÃO

As figuras/textos devem ser armazenadas (**e recuperadas, atualizadas, removidas**) em **kd-trees.** É **terminantemente proibido** usar força bruta para calcular o par mais próximo. Usar o algoritmo mostrado em aula. Voce **deve** usar a função **`qsort`** da biblioteca do C para fazer as ordenações necessárias. A busca por região **deve** evitar sub-árvores que certamente não contenham os dados buscados. O cálculo do vizinho mais próximo **deve** ser feito sobre a árvore Kd. Também é **terminantemente proibido** calculá-lo por força bruta (fazer a **poda** da árvore). **Atenção**: é **imprescindível** que as operações sobre a árvore estejam bem e corretamente implementadas.

É **terminantemente proibido** declarar structs nos arquivos de cabeçalho (.h).

O programa deve estar bem modularizado (arquivos .h e .c). Cada estrutura de dados deve estar em um módulo separado. O arquivo .h **deve** estar muito bem documentado (lembre-se que é um "contrato").[6]

ATENÇÃO: Leia atentamente, com muito cuidado todas as especificações.

ATENÇÃO: Não adianta produzir uma saída (aparentemente correta), sem implementar corretamente a kd-tree, suas operações (incluindo remoção), fazendo as podas adequadas.

ATENÇÃO: teste o makefile antes se submeter o trabalho.

KD-TREE

*Uma **árvore k-d** (abreviação para a árvore k-dimensional) é uma estrutura de dados de particionamento do espaço para a organização de pontos em um k-dimensional espaço. Árvores k-d são estruturas úteis para uma série de aplicações, tais como pesquisas envolvendo pesquisa multidimensional de chaves (e.g. busca de abrangência e busca do vizinho mais próximo). Árvores k-d são um caso especial de árvores de particionamento binário de espaço.*

Uma árvore k-d é uma árvore binária em que cada nó é um ponto k-dimensional. Cada nó não-folha pode ser considerado implicitamente como um gerador de um hiperplano que divide o espaço em duas partes, conhecido como semiespaço. Os pontos à esquerda do hiperplano são representados pela subárvore esquerda desse nó e pontos à direita do hiperplano são representados pela subárvore direita. A direção do hiperplano é escolhida da seguinte maneira: cada nó na árvore é associado a uma das k-dimensões, com o hiperplano perpendicular a esse eixo dimensional. Assim, por exemplo, se para uma determinada operação de split o eixo "x" é escolhido, todos os pontos da subárvore com um valor "x" menor que o nó irão aparecer na subárvore esquerda e todos os pontos com um valor "x" maior vão estar na subárvore direita. Nesse caso, o hiperplano seria definido pelo valor de x do ponto, e o seu normal seria a unidade do eixo x.

Copiado da Wikipedia: https://pt.wikipedia.org/wiki/%C3%81rvore_k-d

AVALIAÇÃO

A avaliação consistirá da execução dos testes e da inspeção de código.

O PROGRAMA

6 Um texto descrevendo o funcionamento geral do módulo e um texto para cada operação descrevendo os parâmetros (especialmente restrições) e o efeito produzido.

O nome do programa deve ser **t2** e aceitar quatro parâmetros:

t2 [-e *path*] -f *arq*.geo [-q consulta.qry] -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 terceiro parâmetro (-q) é um arquivo de consultas. 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**:

- **t2** -e /home/ed/testes/ -f t001.geo -o /home/ed/alunos/aluno1/o/
- **t2** -e /home/ed -f ts/t001.geo -o /home/ed/alunos/al1/o
- **t2** -f ./tsts/t001.geo -e /home/ed -o /home/ed/alunos/aluno1/o/
- **t2** -o ./alunos/aluno1/o -f ./testes/t001.geo
- **t2** -o ./alunos/aluno1/o -f ./testes/t001.geo -q ./t001/q1.qry
- **t2** -e ./testes -f t001.geo -o ./alunos/aluno1/o/ -q ./q1.qry

O Que Entregar

Submeter no Classroom o arquivo .zip com os fontes , conforme descrito anteriormente.

RESUMO DOS PARÂMETROS DO PROGRAMA SIGUEL

Parâmetro / argumento	Opcional	Descrição
`-e` ***`path`***	S	Diretório-base de entrada (**BED**)
`-f` ***`arq`*`.geo`**	N	Arquivo com a descrição da cidade. Este arquivo deve estar sob o diretório **BED**.
`-o` ***`path`***	N	Diretório-base de saída (**BSD**)
`-q` ***`arqcons`*`.qry`**	S	Arquivo com consultas. Este arquivo deve estar sob o diretório **BED**.

RESUMO DOS ARQUIVOS PRODUZIDOS

-f	-q	comando com sufixo	arquivos
arq.geo			arq.svg
arq.geo	***arqcons***.qry		arq.svg arq-arqcons.svg arq-arqcons.txt

ATENÇÃO:
* os fontes devem ser compilados com a opção `-fstack-protector-all`.
* adotamos o padrão C99. Usar a opção `-std=c99`.