ORGANIZAÇÃO DOS PROJETOS

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 ser 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

Os 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 a 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 solicitado em cada trabalho (p.e.,* **`t1`**`)` *que produzirá o executável de mesmo nome dentro do mesmo diretório* **`src`**. *Os fontes devem ser compilados com as opções -fstack-protector-all. -fstack-check*

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.

	** adotamos o padrão C99. Usar a opção -std=c99.*
`*.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 t1`**
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

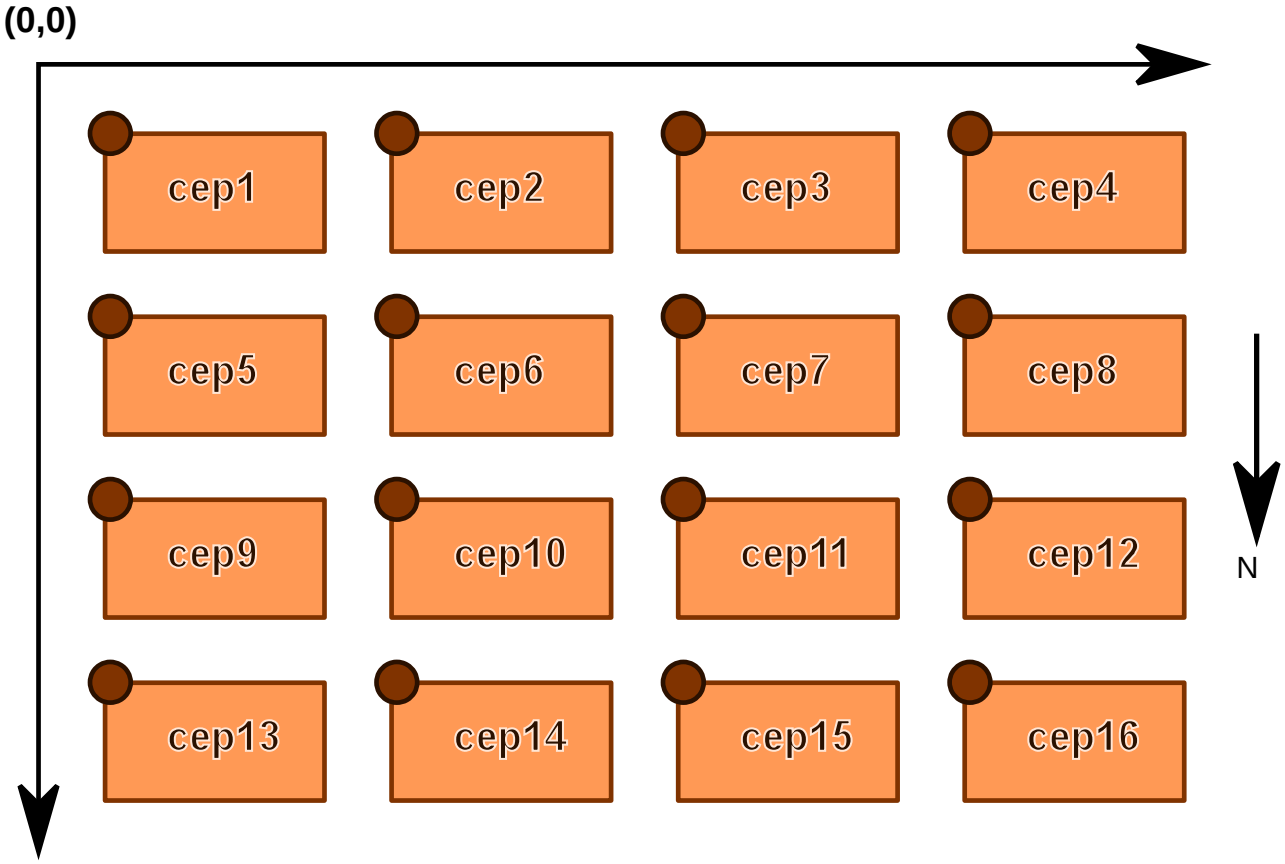
https://www.gnu.org/software/make/manual/make.html
http://opensourceforu.com/2012/06/gnu-make-in-detail-for-beginners/

TRABALHO 1

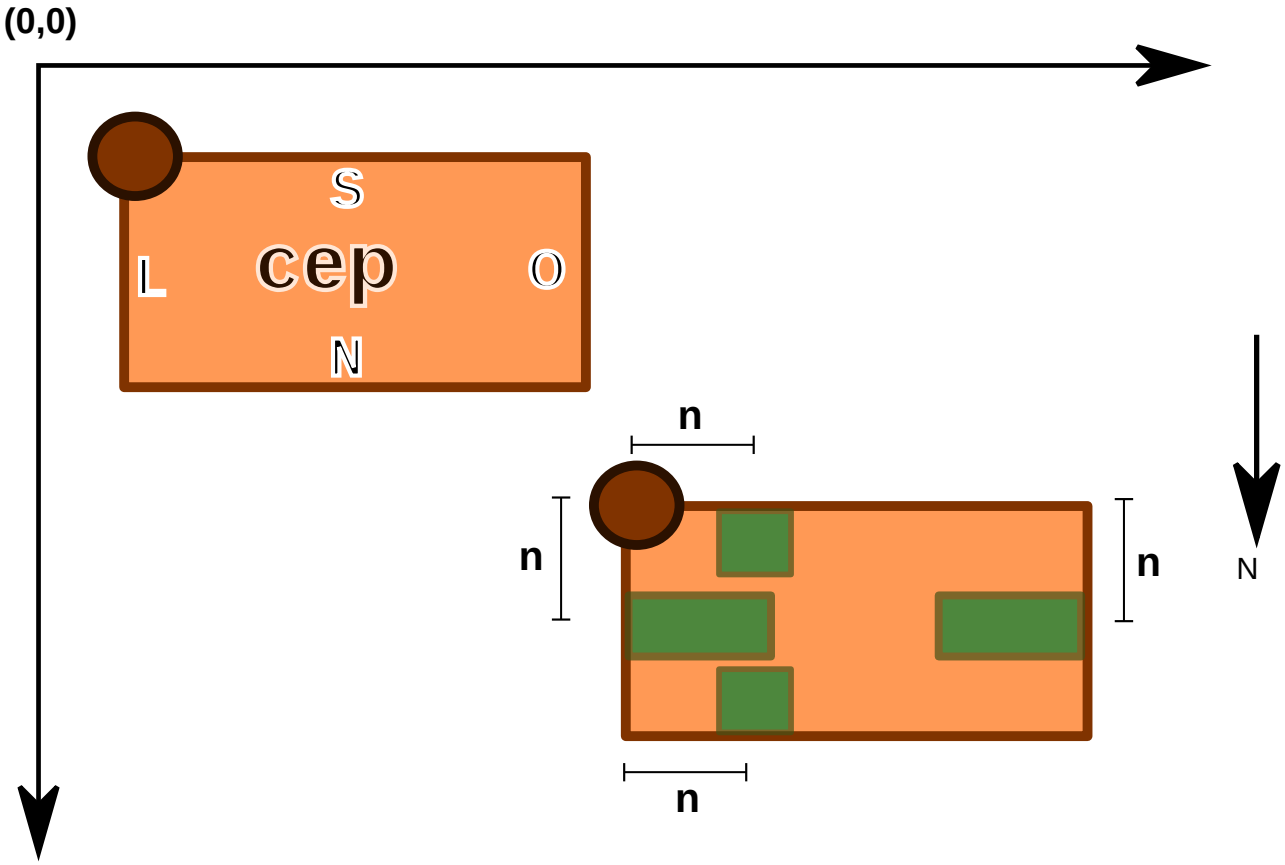
Um Sistema de Informações Geográficas (SIG), para a nossa finalidade, é um sistema que contém (não exclusivamente) dados geo-referenciados, isto é, dados com algum atributo de localização espacial (uma coordenada).

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

O mapa de uma cidade é, basicamente, composto por um conjunto de retângulos que representam as quadras.



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.



A Entrada

Bitnópolis tem quadras, moradores. Moradias podem ser alugadas.

Pessoas e moradores

comando	parâmetros	
p	`cpf nome sobrenome sexo nasc`	*Insere pessoa identificada por cpf, nomeada (nome,sobrenome), de um certo sexo (M,F), nascida numa determinada data (dd/mm/aaaa)*
m	`cpf cep face num compl`	*Informa que um dada pessoa (cpf) mora num dado endereço (cep,face,num,compl)*
Comandos do arquivo .pm (-pm)		

Cidade

A cidade é descrita pelo arquivo **`.geo`**:

comando	parâmetros	
nx	`n`	*Estimativa do número de pessoas*
q	`cep x y w h`	*Insere uma quadra (retângulo e cep)*
cq	`sw cfill cstrk`	*Cores do preenchimento (`cfill`) e da borda (`cstrk`) das quadras, espessura da borda (`sw`) (a partir deste comando)*
Novos comandos do arquivo .geo		

Consultas

comando	parâmetros	
del	`cep` DEL	*Remove a quadra `cep`, os moradores que nela residirem e as ofertas de locação que nela existirem.* ***No arquivo .svg**: quadra deve ser colocada uma linha vertical com início no centro do elemento removido até o topo do mapa. Também colocar (no topo) ao lado da linha vertical o cep.* ***No arquivo .txt**: reportar todos os dados relacionados aos elementos removidos.*
m?	`cep`	*Moradores da quadra cujo cep é `cep`.* ***TXT:** listar todos os dados dos moradores(nome, endereço,...). Reporta mensagem de erro se quadra não existir.*
dm?	`cpf`	*Imprime todos os dados do morador identificado pelo cpf.* ***TXT:** dados pessoais, seu endereço e se moradia é alugada.* ***SVG**: colocar uma linha vertical do endereço do morador até a margem superior do mapa. Anotar ao lado da linha o cpf, nome e endereço do morador*
mud	`cpf cep face num compl`	*A pessoa identificada por `cpf` muda-se para o endereço determinado pelos parâmetros.* ***TXT:** Mostrar os dados da pessoa (nome, etc), o endereço antigo e o novo endereço.* ***SVG**: Uma linha vermelha grossa do endereço antigo ao endereço novo. Um pequeno círculo vermelho no endereço antigo, outro círculo azul no novo endereço. Ambos círculos com borda branca grossa.*

oloc	`id cep face num cpl ar v`	*Oferta de locação. Imóvel do endereço especificado, de área* `ar` m^2 *é ofertado para locação por R$* `v` *mensais.*
oloc?	`x y w h`	*Quais são os imóveis disponíveis para locação na região especificada.* ***TXT:*** *relatar dados do imóvel e da locação* ***SVG:*** *marcar no svg a região (um retângulo com bordas tracejadas) e os imóveis disponíveis (com '*')*
loc	`id cpf`	*Pessoa identificada por* `cpf` *aluga o imóvel cuja oferta é identificada por* `id`*. O imóvel deixa de estar disponível para alocação. O indivíduo passa a ser morador do referido imóvel.* ***TXT****: reportar dados do imóvel, da locação e da pessoa.* ***SVG****: marcar endereço do imóvel, traçar uma reta vertical do imóvel até a borda superior do desenho e, ao lado, colocar informações sobre morador, imóvel e locação*
loc?	`id`	*Qual a situação da oferta de locação?* ***TXT:*** *reportar os dados da moradia, se ela está alocada, se está disponível para locação. Caso esteja locada, reportar os dados pessoais do morador.* ***SVG:*** *marcar o local da moradia com '*' se está locada; '$' se está disponível para locação; '#', caso o contrato de locação tenha sido encerrado (dloc).*
dloc	`id` DEL, NÃO USA TRAVERSE	*Locatário encerra contrato de locação. A oferta é removida.* ***TXT****: reportar dados pessoais e sobre a moradia.* ***SVG****: semelhante a* loc*.*
hom	`x y w h`	*Homens que moram dentro da região especificada.* ***TXT****: reportar dados da pessoa e da moradia* ***SVG****: marcar posição da moradia com uma pequena elipse azul.*
mul	`x y w h`	*Semelhante a* **hom***. Seleciona as mulheres. Pinta a elipse de rosa.*

dmpt	`sfx`	*Imprime o estado atual da árvore no arquivo* <arq>-sfx.*dot.* *Cada nó da árvore deve mostrar a coordenada X, o número de quadras "dentro" do nó, o cep de algumas (poucas) quadras, o x mínimo, o x máximo e o fator de balanceamento do nó. Opcionalmente: em vez de escrever o fator de balanceamento*
catac	`x y w h` **DEL**	*Remover as quadras, moradores que estejam inteiramente contidas na região delimitada pelo retângulo* `x,y,w,h`*. Ofertas de locação dentro da região também deve ser removidas.* ***TXT:** imprimir identificadores e dados associados a tudo o que foi removido.* ***SVG:** elementos removidos não devem aparecer. Desenhar sob o mapa o referido retângulo com cor de preenchimento #AB37C8, cor de borda #AA0044 e com 50% de transparência.*
Comandos do arquivo .qry		

A Saída

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

Se o arquivo .geo (a01.geo, no nosso exemplo) for processado em conjunto com um arquivo .qry (q.qry, no exemplo), além de a01.svg, deverá ser produzido o arquivo a01-q.svg, contendo as quadras acrescidas aos resultados da consulta.

Também produzir um arquivo-texto (a01-q.txt, no exemplo) contendo o resultado textual de todas as consultas. Neste arquivo deve ser copiado em uma linha o texto da consulta e, na linha seguinte, o seu resultado.

O comando **`dmpt`** produz um arquivo **`.dot`**., que será processado pelo programa **dot**. Veja https://graphviz.org/.

Implementação

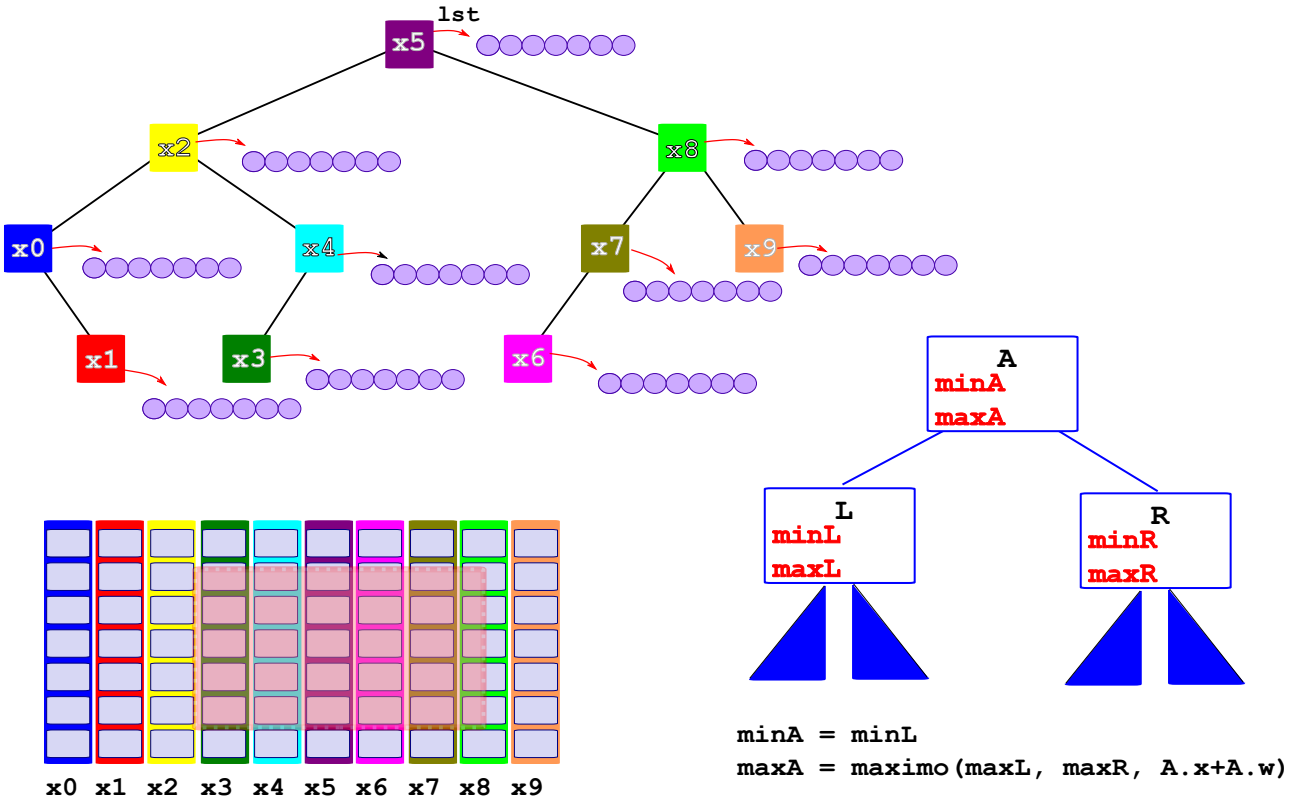
Os moradores e locações **devem** ser armazenados em tabelas de espalhamento. As quadras **devem** ser armazenadas em árvores AVL. As buscas espaciais **devem** "podar" ramos que, com certeza, não contém quadras elegíveis.

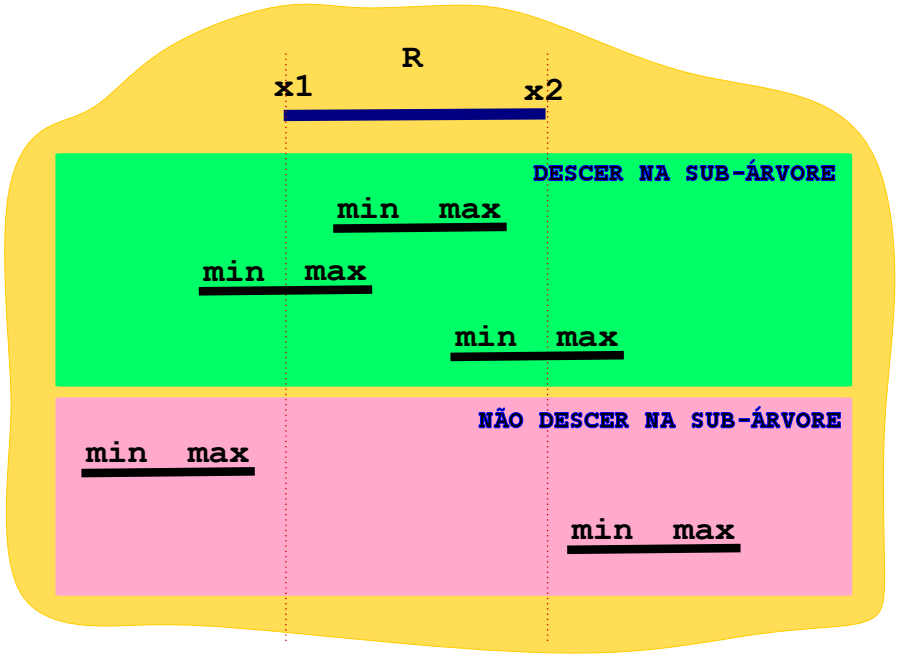
Árvore AVL

Lembrando que as buscas espaciais **devem** podar ramos não promissores, sugere-se **fortemente** que a árvore AVL seja implementada conforme descrito a seguir:

- Cada nó da árvore armazena como chave a coordenada X da âncora da quadra

- Como muitas quadras podem ter a mesma coordenada X, cada nó da árvore armazena uma lista das quadras cuja âncora possui aquela coordenada
- Para facilitar/possibilitar a poda, cada nó da árvore armazena os limites mínimo e máximo de X em suas sub-árvores.



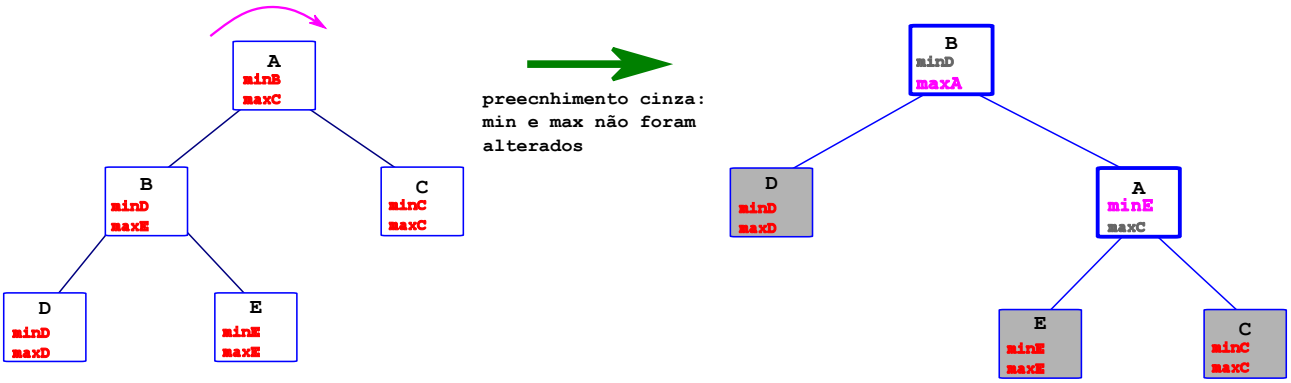



```
se dentroDe(t->x,r.x1, r.x2)
  então quadLst = quadrasDentro(t->lst,r)

se t->esq->max >= r.x1 AND t->esq->min <= r.x2
  então /* desce sub-arvore esq */

se t->dir->max >= r.x1 AND t->dir->min <= r.x2
  então /* desce sub-árvore dir */
```

Note, na figura abaixo, como as coordenadas mínima e máxima de um nó rotacionado devem ser alteradas:



AVALIAÇÃO

Espera-se uma atitude pró-ativa para a aquisição dos conhecimentos (i.e., estudo) para resolver o problema proposto.

A avaliação consistirá da execução dos testes e da inspeção de código. Além disso, deverá ser produzido um vídeo de, aproximadamente, 5 minutos no qual apresenta o sistema. O aluno deve ter por objetivo convencer o avaliador que: (a) o programa funciona; (b) o programa foi bem implementado. O vídeo deve ser colocado no Youtube e o seu link deve estar anotado no arquivo LEIA-ME.TXT.

O PROGRAMA

O nome do programa deve ser **t1** e aceitar até cinco parâmetros:[4]

t1 [-e *path*] -f *arq*.geo [-q consulta.qry] [-pm pessoas.pm] -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 parâmetro **-pm** informa o arquivo das pessoas e moradores.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 **t1**:

- **t1** -e /home/ed/testes/ -f t001.geo -o /home/ed/alunos/aluno1/o/
- **t1** -e /home/ed -f ts/t001.geo -o /home/ed/alunos/al1/o
- **t1** -f ./tsts/t001.geo -e /home/ed -o /home/ed/alunos/aluno1/o/
- **t1** -o ./alunos/aluno1/o -f ./testes/t001.geo
- **t1** -o ./alunos/aluno1/o -f ./testes/t001.geo -q ./t001/q1.qry
- **t1** -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.

4 Novos parâmetros são acrescentados no decorrer do ano

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**.
`-pm` ***`arq`*`.pm`**	S	Arquivo de pessoas. 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
arq.geo	***arqcons***.qry	***sufx***	arq.svg arq-arqcons.svg arq-arqcons.txt arq-arqcons-sufx.[svg|txt][5]

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

5 Podem ser produzidos os respectivos arquivos .svg e/ou .txt, dependendo da especificação do comando.