3ª Lista de Exercícios de Paradigmas de Linguagens Computacionais

Professor: Fernando Castor

Monitores:

Ciência da computação: Eric Lucena Palmeira Silva (elps), João Gabriel Santiago Maurício de Abreu (jgsma), Marcos Paulo Barros Barreto (mpbb), Tulio Paulo Lages da Silva (tpls),Victor Félix Pimenta (vfp).

CIn-UFPE – 2014.1
Disponível desde: 13/06/2014
Entrega: 27/06/2014

*A lista deverá ser respondida em dupla. A falha em entregar a lista até a data estipulada implicará na **perda de 0,25 ponto** na média da disciplina para os membros da dupla. Considera-se que uma lista na qual **menos que 4** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 6** das questões corretamente respondidas implica em um **acréscimo de 0,125 ponto** na média da disciplina para os membros da dupla. Se **qualquer situação de cópia de respostas for identificada**, os membros de todas as duplas envolvidas **perderão 0,5 ponto** na média da disciplina. O mesmo vale para respostas obtidas a partir da Internet. As respostas deverão ser entregues **exclusivamente em formato texto ASCII** (nada de .pdf, .doc, .docx ou .odt) e deverão ser enviadas para o monitor responsável por sua dupla, **sem** cópia para o professor. Devem ser organizadas em arquivos separados, um por questão, entregues em um único arquivo compactado, ou seja, um único arquivo .zip contendo as respostas para todas as questões.*

1) Quase todo aluno de Ciência da Computação tem sérios problemas em pronunciar Fila e Pilha em sequência. Pimenta, que é um ótimo aluno, resolveu acabar com esse problema criando uma nova estrutura de dados, a **PilaFilha**.

A **PilaFilha** consiste de uma lista encadeada que permite as operações padrão de Filas e Pilhas: queuePush(A junção de queue e push), dequeue, pop.

QueuePush: Coloca um elemento na cauda da lista.
Dequeue: Retira o elemento que se encontra na cabeça da lista.
Pop: Retira um elemento da cauda da lista.

Vendo que a implementação estava muito simples, Pimenta resolveu dificultar um pouco o seu trabalho, e decidiu que a classe **PilaFilha** deveria ser thread safe (Ele não pagou PLC, mas achou o termo bonito). Assim, sobrou pra você fazer essa implementação. Implemente duas versões da estrutura **PilaFilha**, uma utilizando **apenas** blocos sincronizados, e outra utilizando **apenas** travas explícitas.

OBS:

A estrutura deve permitir operações simultâneas **sempre** que possível. Descreva em forma de comentário as situações onde é impossível realizar operações simultâneas, e descreva brevemente o porquê das suas implementações serem realmente thread safe.

2) Em cenários de fantasia medieval, brigas em tavernas ocorrem com bastante frequência (para a infelicidade dos taverneiros), geralmente envolvendo várias pessoas. Em uma taverna com M (M >= 4) pessoas e K cadeiras (K >= M), caso uma briga comece, N (2 <= N <= M, com N aleatório) pessoas decidem participar. Cada pessoa possui H pontos de vida (10 <= H <= 20) e está próxima a I cadeiras (1 <= I <= 3) e J copos (1 <= J <= 4), onde H, I e J são números calculados aleatoriamente para cada pessoa, e já incluem o copo e a cadeira que a pessoa usava antes do começo da briga. Cada pessoa pode, enquanto a briga não houver acabado:

1) Golpear alguém com uma cadeira, causando 5 pontos de dano.
2) Atirar um copo em alguém, causando 3 pontos de dano.
3) Esmurrar alguém, causando 2 pontos de dano.

Sempre que puder agir, cada pessoa escolhe aleatoriamente uma ação e um alvo. Além disso, sempre que não for possível realizar a ação (por falta de cadeiras ou copos), a escolha é refeita. Ao atingir 0 ou menos pontos de vida, o participante fica inconsciente e está efetivamente fora da briga, não podendo mais ser alvo de golpes. Quando houver somente uma pessoa consciente, a briga é considerada finalizada.

Implemente um simulador de brigas de taverna, que siga essas instruções. Seu programa deve implementar cada participante como uma thread e ao final imprimir no console quem foi o vencedor, além das quantidades de cadeiras e copos que foram quebrados. Adicionalmente, sempre que alguém for golpeado, imprima uma mensagem no console informando o atacante, o alvo, o dano sofrido e a quantidade de pontos de vida restantes do alvo (e.g. "A pessoa X acertou a pessoa Y com uma cadeira (dano 5)! Y agora tem HP Z"). Sempre que alguém ficar inconsciente, imprima também no console uma mensagem informando (e.g. "Pessoa X está inconsciente!"). As quantidades de cadeiras e pessoas na taverna, K e M, respectivamente, são escolhidas pelo usuário.

Observações:

1) Todos os participantes são fortes o suficiente para quebrar cadeiras e copos após um golpe, então eles não podem ser reusados.
2) Cada pessoa pode utilizar somente as cadeiras e copos que estavam inicialmente próximas a ela.
3) Qualquer quantidade de pessoas pode estar próxima a uma mesma cadeira ou copo, então uma vez usado por alguém, ninguém mais pode utilizar o mesmo objeto.
4) A quantidade inicial de copos é a quantidade de pessoas participando da briga (N).
5) Para garantir a integridade e corretude dos dados, devem ser usados somente métodos ou blocos sincronizados, ou travas explícitas.

3) Faça uma segunda implementação da questão anterior, desta vez sem usar métodos ou blocos sincronizados, nem travas explícitas. A integridade e corretude dos dados deve ser garantida através de estruturas de alto nível contidas na biblioteca java.util.concurrent.

4) Faça um programa em Java que receba dois números (n e m) e monte uma lista com todos os números menores do que m que sejam divisíveis por n. Desenvolva 3 implementações: uma sequencial, uma paralela com divisão de trabalho estática e por fim uma paralela com divisão dinâmica.

Implemente e utilize sua própria lista segura para inserções para montar o resultado.

a) Teste com pelo menos 6 entradas diferentes (algumas com valores menores outras com valores maiores) e compare o tempo de execução. Comente os resultados obtidos e diga de maneira geral qual foi a melhor opção entre as implementações nos diferentes cenários.

*O número de threads deve ser o máximo suportado pelo seu computador, deixe esse valor explicito em comentários.

b) Use a mesma estrutura das 3 implementações, mas passe a devolver uma lista dos co-primos a n. (primos entre si). Teste com os mesmos valores da letra a). Comente os resultados obtidos e diga de maneira geral qual foi a melhor opção entre as implementações nos diferentes cenários.

*Não deve se dividir o trabalho entre threads para se definir se um número é co-primo de n, cada thread faz uma (ou várias) dessas verificações por completo e insere ou não na lista de resultado.

5) Faça uma estrutura de árvore em Java, onde cada nó só possa ser visitado por uma thread exploradora a cada momento. (zona de exclusão mútua). As threads essencialmente fazem uma busca em profundidade e marcam os nós que já foram visitados, caso elas visitem algum nó ainda inexplorado elas acrescentam 1 a um contador geral. Quando o contador chegar ao número de nós na árvore, não existem mais locais inexplorados, dessa forma todas as threads devem parar sua exploração. Se uma thread tentar visitar um nó que está ocupado, ela desiste e tenta visitar um nó vizinho. Se conseguir algum vizinho ela despreza os galhos anteriores e continua com sua busca em profundidade, mas se não conseguir nenhum vizinho ela volta a tentar o nó inicial nessa profundidade, recomeçando o ciclo de tentativas.
(considere que as threads exploradoras passam 0.01 ms “apreciando” cada nó inexplorado)

Monte uma estrutura com 50 nós e 10 threads exploradoras. Faça os exploradores guardarem seu caminho e imprimirem no final.
(Nenhum explorador deve ficar travado. Sempre que executado, todos os nós devem ser explorados)

6) Sabendo que apenas 10 alunos do CIn que estivessem conectados à rede do centro (tanto em um computador de um dos laboratórios, como na wi-fi ou pela VPN) seriam selecionados para entrar no concurso Geração Brasil, Roberto Noveleiro não poderia deixar ninguém atrapalhar seu objetivo de se tornar o próximo Jonas Marra. Para isso, ele teria de usar suas super-habilidades hacker para impedir o acesso à rede do CIn durante o concurso, utilizando o seu incrível Marra Phone. Com aplicativo Marra Tools, ele pode monitorar toda a rede do CIn e desconectar em X segundos (determinados pelo usuário) qualquer pessoa logada na rede e desabilitar seu login durante os próximos 10 segundos. Existem 5 laboratórios com N (definido pelo usuário) computadores cada e várias pessoas tentando se conectar usando à rede wi-fi do CIn ou VPN. Considere as seguintes informações:

- Cada usuário tentando logar na rede é uma thread distinta. Cada um terá um ID

único, além do progresso de sua inscrição de (0% a 100%) e caso tenha sido deslogado pelo Marra Tools, o tempo restante em que ele poderá logar novamente.

- Em cada laboratório, um usuário tenta logar (ou seja, uma thread é criada) a cada 5~10 segundos. O laboratório para de criar threads quando sua capacidade física N for alcançada, ou seja, os 5 laboratórios são fontes limitadas de threads.
- Usuários vindos (threads criadas) do wi-fi ou VPN (considerados uma só fonte) chegam a cada 5~15 segundos para tentar logar. Esta é uma fonte ilimitada de threads.
- Para Roberto Noveleiro manter o controle, ele tem uma fila de prioridades em forma de Heap (seguro para threads) que você deverá implementar. Usuários que logam na rede do CIn entram nessa fila, indicando qual o usuário que deve mais rapidamente ter o cadastro atrasado. Imprima sempre uma mensagem assim que alguém entrar na fila, especificando quem é.
- Usuários desconectados pelo Marra Tools demoram 10 segundos para conseguir relogar e perdem entre 5% e 50% do progresso de sua inscrição. Quando passarem os 10 segundos, eles tentam logar novamente (entrar na fila). Imprima uma mensagem sempre que alguém for deslogado e especificando quem foi.
- A cada 10s, um usuário logado completa 5% de seu cadastro.
- Sempre que ou um usuário tenta logar, o Marra Tools avalia-o para colocá-lo no lugar correto da fila de prioridades. A prioridade será para o usuário com o cadastro com a porcentagem mais alta. Caso haja empate, quem logou antes terá prioridade.
- Caso um usuário complete seu cadastro, exiba uma imagem sobre o fato. Este usuário sairá da fila e não será mais considerado pelo programa.
- O programa deve parar ou se 10 usuários (imprimindo que Roberto falhou) completarem seu cadastro ou após um tempo determinado pelo usuário (imprimindo congratulações para Roberto, que está um passo mais próximo de ser o próximo Jonas Marra).

Escreva também as propriedades de segurança e vivacidade do seu programa em forma de comentário no seu código, utilizando um argumento formal para provar sua corretude.

7) Implemente um sistema de gerenciamento de dúvidas em sala de aula. Para tal considere a presença de uma lista de alunos, cada qual com uma lista de perguntas a serem respondidas, e um professor que é o único capaz de respondê-las. Apenas um aluno pode abordar o professor por vez e dizer sua pergunta e o professor não pode ser abordado por alunos enquanto resolve uma pergunta. Cada pergunta tem um tempo específico para ser respondida e o professor deve passar esse tempo respondendo sem ser atrapalhado. O professor pode memorizar uma quantidade de até N perguntas (onde N é a quantidade de alunos na sala). Outro fato é que o professor precisa beber água sempre que vai responder uma pergunta, e a cada duas perguntas respondidas sua água acaba, de forma que ele precisa deixar a sala para encher sua caneca. Enquanto isso os alunos não podem efetuar perguntas, pois o professor está ausente. O usuário determina o tempo e duração necessários para cada ação.

8) Sua missão é desenvolver uma versão caseira do Twitch Plays Pokemon. Nesse caso você deve considerar um mapa de tamanho NxN de uma caverna gerado de forma

aleatória em que há espaços vazios (representados por '_'), espaços com zubats (representados por 'Z'), um espaço em que Red está (representado por 'R') e a saída da caverna (representada por 'X'). Red começa o jogo com 3 pokemons, cada qual com valores de HP entre 30 e 50. A missão de Red é deixar a caverna sem perder todos os seus pokemons. Um pokemon é perdido quando a vida deste chega a 0. Quando Red se move para uma área com zubats ele escolhe aleatoriamente um dos pokemons e este perde entre 5 e 10 pontos de HP antes de derrotar o zubat.

É claro que para que o jogo seja compartilhado no twitch você deve ter uma solução em que vários jogadores possam efetuar as 6 funções básicas do jogo ao mesmo tempo (mover red para cima, baixo, esquerda, direita, votar no modo anarquia ou no modo democracia, que são atingidos quando algum dos dois modos tem 100 votos).

Detalhes: No modo anarquia, cada jogador (thread) escolhe aleatoriamente uma das 4 funções de movimento e esta é executada caso o sistema esteja esperando um novo input (já tenha acabado de efetuar alguma ação), caso contrário o sistema ignora o input. No modo democracia, cada input de movimento dos jogadores é contabilizado, o primeiro comando de movimento a atingir 100 votos é executado e os quatro são zerados em seguida (comandos de voto não seguem essa regra e são computados normalmente). Os comandos de votos são contabilizados independentemente do modo em que se está jogando, e cada vez que um deles chegar a 100, o modo ganhador é ativado e os buffers dos 6 comandos são zerados.