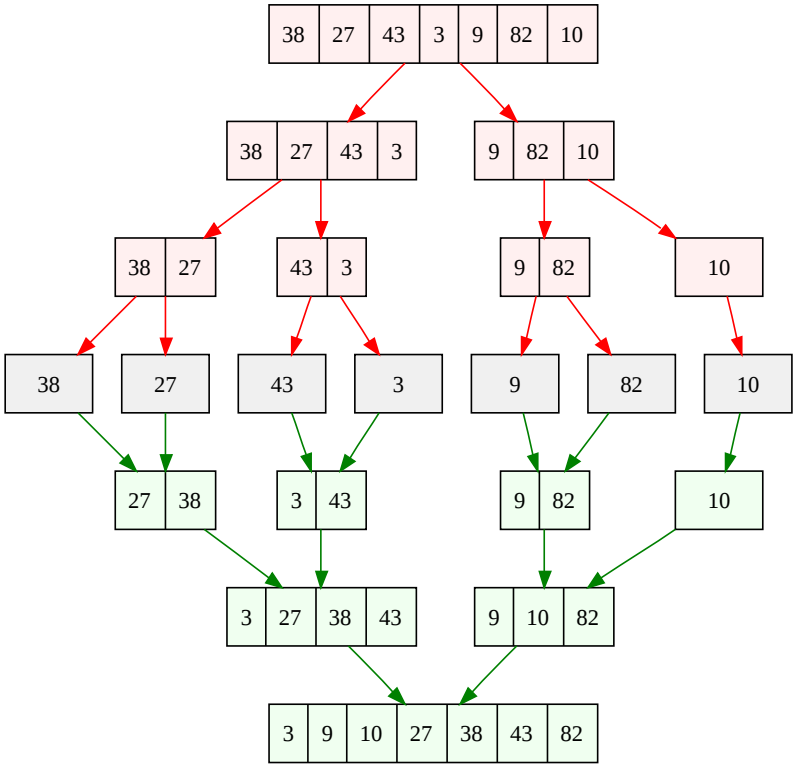
Atividade Prática - Threads em Java

O programa em Java fornecido ordena uma lista de inteiros usando o algoritmo *MergeSort*. Já existe uma implementação recursiva, mas serial na classe **`MergeSortSerial`**.

A lógica do MergeSort, mostrada na figura abaixo é baseada na ideia de dividir e conquistar. Para ordenar uma lista, basta **dividir a lista na metade** em duas listas. Supondo que cada uma das metades esteja ordenada, uma operação *merge* **entrelaça as duas listas em uma nova lista ordenada** (essa é a parte "conquistar"). No programa em questão, a operação *merge* está implementada em **`MergeSortHelper.merge(List<T> left, List<T> right)`** e não deve ser alterada. Para garantir a suposição de que cada metade da lista está ordenada, o próprio algoritmo MergeSort pode ser aplicado. Quando uma sub-lista tem um ou zero elementos, não é possível mais dividir, mas tal lista já **está ordenada** e não há nada que precise ser feito.



Você deve completar o programa com duas implementações paralelas do *MergeSort*. Siga esses passos:

Passo 1

Implemente o método **`MergeSortThread.sort()`** para que as duas sub-listas sejam ordenadas em paralelo, cada uma por uma thread. Crie as threads usando a classe java.lang.Thread, como mostrado na aula teórica. Lembre de fazer um **`join()`** antes de chamar `MergeSortHelper.merge()` com o resultado de cada thread.

Dica: para obter o resultado produzido por uma thread, faça com que a thread escreva seu resultado em um`ArrayList<ArrayList<Integer>>` **no escopo do método que criou a thread**

```
final ArrayList<ArrayList<T>> results = new ArrayList<>();
results.add(null);
//....
results.set(0, MergeSortThread.this.sort(list.subList(0, mid)))
```

Passo 2

Implemente o método **`MergeSortExecutor.sort()`** usando um **`ExecutorService`** **com a política cached**. Ou seja, um executor que **tenta** reutilizar threads, mas que na ausência de threads livres criará uma nova thread, **sem limite máximo de threads ativas**. A estratégia é parecida com a usada em `MergeSortThread`, mas ao invés de interagir com uma thread, você deverá interagir com um **`Future`**. Leia a documentação dessa classe e descubra como ler o resultado que foi computado assincronamente.

Dica 1: Você deverá usar uma implementação da interface Callable<ArrayList<T>> com o Executor

Dica 2: Como implementar um Callable em uma classe anônima:

```
new Callable<ArrayList<T>>() {
    @Override
    public ArrayList<T> call() {
        return MinhaClasse.this.meuMetodo(arg1, arg2);
    }
}
```

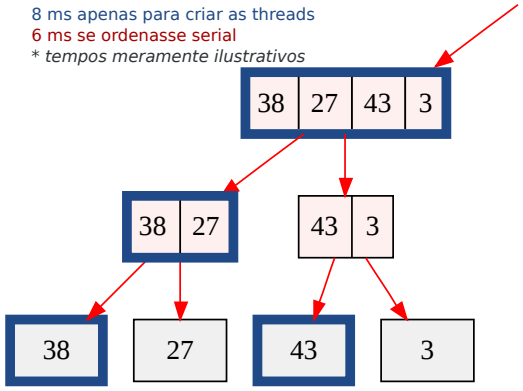
Passo 3

Use a classe Main para executar suas implementações a partir da linha de comando:

```
./mvnw -DskipTests=true package
java -jar target/threads-1.0-SNAPSHOT.jar -i THREADS 65536
```

A opção -i indica a implementação. Valores possíveis são `SERIAL`, `THREADS` e `EXECUTOR`. Observe os tempos com as três implementações. Muito provavelmente as implementações paralelas estão mais lentas que a serial.

Para entender o motivo desse comportamento, concentre-se no sub-vetor da figura abaixo. Os subvetores com borda azul são os que tiveram uma thread criada para os ordenar.



Se a sua implementação paralela segue fielmente o código serial, para ordenar o sub-vetor [38,27], você irá criar 1 thread para ordenar o vetor [38]. Acontece que o **tempo gasto para criar uma thread que ordena o sub-vetor [38] é maior do que o tempo necessário para ordenar o vetor [38]** (que já está ordenado)! Esse mesmo efeito acontece para o sub-vetor **[38,27,43,3]**: ordenar esse vetor serialmente é mais rápido do que criar as 3 threads necessárias para ordenar os sub-vetores. Sabendo disso, **altere as implementações paralelas de modo que elas sejam mais rápidas que as versões seriais**.

Passo 4

Estude para atividade de reforço da P1, que será realizada na aula do dia 24 de Outubro.

Correção automática

O script de correção (**grade-threads.sh**) está dentro do esqueleto e deve ser executado na própria pasta onde está. Você deve implementar as classes já criadas. Não crie suas soluções em novas classes ou o script corretor não as irá encontrar. Os testes estão visíveis na classe MergeSortTest, para os curiosos. O arquivo de testes será substituído durante a avaliação.

Caso queira usar features do Java 8+ altere o pom.xml.

Entrega do Exercício

Submeta um arquivo **.tar.gz (ou zip)** na **mesma estrutura do esqueleto**dado como ponto inicial nessa tarefa. O uso do esqueleto fornecido é **obrigatório**. Utilize **make submission** para garantir que será gerado um arquivo conforme solicitado.

O prazo para entrega é **21 de Outubro às 23h55**.

Maven - Linha de Comando, Eclipse ou IDEA

No Eclipse

1. File -- Import... -- Maven -- Existing Maven Projects -- Next
2. Selecione a pasta atividade_8
3. Finish

Para compilar, testar e empacotar:

1. Botão direito no projeto -- Run As -- maven build...
2. Digite package como goal
3. Run

Para Rodar os testes de dentro da IDE:

1. Instale o plugin do TestNG: https://marketplace.eclipse.org/content/testng-eclipse
2. Botão direito no projeto -- Run As -- TestNG Test

No IntelliJ IDEA

1. Import project
2. Escolha a pasta contendo o arquivo pom.xml
3. Next -- Next -- ... -- Finish

Para Compilar, rodar os testes e empacotar: Shift Shift > maven goal > package

Para rodar os teste de dentro da IDE:

1. Vá até o MergeSortTests.java
2. Posicione o cursor fora de um método
3. Aperte Alt+Shift+F10

No NetBeans

Abra o projeto Maven com o menu Arquivo > Abrir Projeto... ou com o botão Abrir Projeto...

Para rodar os teste de dentro da IDE, clique no Projeto com o botão direito do mouse e selecione a opção 'Testar'.

Esqueleto

(Atualizado em 22/10/2018 - 20:14)

- Makefile (estava faltando)
- Adicionadas instruções sobre maven para quem não conhecer
- MergeSortTest.java: Teste mais bondoso em máquinas com pouca memória
- Main.java: agora retorna o tempo médio de 10 execuções, ao invés de só uma.
- MergeSortTest.java: re-escrito em JUnit devido a alguns **ubuntus mal-comportados**
- Criado wrappers `mvnw` (linux) e `mvnw.cmd` (windows) que **dispensam instalação do maven** (necessário baixar .tar.gz inteiro)
- Makefile: `make` compila, `make verify` testa e `make submission`roda testes antes de empacotar (mas empacota mesmo que testes falhem)

(Atualizado em (27/10/2018 - 13:08)

- `grade-threads.sh` mostra a saída do `mvn verify`

atividade_8.tar.gz
grade-threads.sh
Main.java
Makefile
MergeSortTest.java