STUD - Marek Magula

S1

Obsah

[Generátory 2](#_Toc192692901)

[ContinuousGenerator 2](#_Toc192692902)

[DiscreteGenerator 4](#_Toc192692903)

[Simulácia 5](#_Toc192692904)

[Úloha 1 6](#_Toc192692905)

[Úloha 2 6](#_Toc192692906)

# Generátory

Pre potreby semestrálnej práce bolo potrebné implementovať generátory pre spojité a diskrétne rozdelenie(rovnomerné a empirické).

Implementovali sme abstraktnú triedu AbstractGenerator, ktorá obsahuje seedGenerátor, ktorý generuje násady pre všetky využité generátorov náhodných čísel.

Ďalej obsahuje probabilityGenerator ,ktorý sa využíva na výber intervalu z ktorého bude číslo generované a zoznam generátorov pre každý zadaný interval.

Rovnako obsahuje metódu getSample, ktorá vracia vygenerovanú hodnotu.

*abstract class AbstractGenerator<T> {*

*protected final Random seedGenerator;*

*protected final Random probabilityGenerator;*

*protected final List<Random> intervalGenerators;*

*public AbstractGenerator(Random seedGenerator) {*

*this.seedGenerator = seedGenerator;*

*this.probabilityGenerator = new Random(seedGenerator.nextLong());*

*this.intervalGenerators = new ArrayList<>();*

*}*

*public abstract T getSample();*

*}*

Následne sme implementovali triedy ContinuousGenerator pre generovanie spojitého rovnomerného a empirického rozdelenia a triedu DiscreteGenerator pre generovanie diskrétneho rovnomerného a empirického rozdelenia. Intervaly zadané v parametri týchto tried sú polouzavreté sprava.

## ContinuousGenerator

V konštruktore sa vytvorí generátor pre všetky zadané intervaly. V metóde getSample sa vygeneruje za použitia probabilityGenerator náhodné číslo, ktoré následne rozhodne o výbere intervalu z ktorého bude generované číslo, následne sa vygeneruje za použitia generátora pre daný zvolený interval náhodné číslo z tohto intervalu.

V prípade rovnomerného rozdelenia je potrebné do konštruktora poslať jeden interval(horná hranica intervalu nie je súčasťou generovaných čísel) a jednu pravdepodobnosť s hodnotou 1.

*public class ContinuousGenerator extends AbstractGenerator<Double> {*

*private final List<double[]> intervals;*

*private final List<Double> probabilities;*

*public ContinuousGenerator(Random seedGenerator, List<double[]> intervals, List<Double> probabilities) {*

*super(seedGenerator);*

*this.intervals = intervals;*

*this.probabilities = probabilities;*

*for (int i = 0; i < intervals.size(); i++) {*

*intervalGenerators.add(new Random(seedGenerator.nextLong()));*

*}*

*}*

*@Override*

*public Double getSample() {*

*double rand = probabilityGenerator.nextDouble();*

*double cumulative = 0.0;*

*for (int i = 0; i < intervals.size(); i++) {*

*cumulative += probabilities.get(i);*

*if (rand < cumulative + Constants.epsilon) {*

*double min = intervals.get(i)[0];*

*double max = intervals.get(i)[1];*

*return min + (intervalGenerators.get(i).nextDouble() \* (max - min));*

*}*

*}*

*return null;*

*}*

*}*

## DiscreteGenerator

V konštruktore sa vytvorí generátor pre všetky zadané intervaly. V metóde getSample sa vygeneruje za použitia probabilityGenerator náhodné číslo, ktoré následne rozhodne o výbere intervalu z ktorého bude generované číslo, následne sa vygeneruje za použitia generátora pre daný zvolený interval náhodné číslo z tohto intervalu.

V prípade rovnomerného rozdelenia je potrebné do konštruktora poslať jeden interval(horná hranica intervalu nie je súčasťou generovaných čísel) a jednu pravdepodobnosť s hodnotou 1.

*public class DiscreteGenerator extends AbstractGenerator<Integer> {*

*private final List<int[]> intervals;*

*private final List<Double> probabilities;*

*public DiscreteGenerator(Random seedGenerator, List<int[]> intervals, List<Double> probabilities) {*

*super(seedGenerator);*

*this.intervals = intervals;*

*this.probabilities = probabilities;*

*for (int i = 0; i < intervals.size(); i++) {*

*intervalGenerators.add(new Random(seedGenerator.nextLong()));*

*}*

*}*

*@Override*

*public Integer getSample() {*

*double rand = probabilityGenerator.nextDouble();*

*double cumulative = 0.0;*

*for (int i = 0; i < intervals.size(); i++) {*

*cumulative += probabilities.get(i);*

*if (rand < cumulative + Constants.epsilon) {*

*int min = intervals.get(i)[0];*

*int max = intervals.get(i)[1];*

*return  min + intervalGenerators.get(i).nextInt(max - min);*

*}*

*}*

*return null;*

*}*

*}*

# Simulácia

Implementovali sme simulačné jadro od ktorého sme následne odvodili jednotlivé simulované stratégie.

Následne sme implementovali triedu SimulationStrategy, ktorá združuje spoločné atribúty a metódy našich stratégií. Jednotlivé stratégie sme implementovali a ako potomkov triedy SimulationStrategy.

*public abstract class MonteCarloCore {*

*private boolean stop = false;*

*public void setStop(boolean stop) {*

*this.stop = stop;*

*}*

*public final void runSimulation(int numberOfReplications) {*

*for (int i = 0; i < numberOfReplications; i++) {*

*if (this.stop) {*

*break;*

*}*

*beforeSimRun();*

*executeSimRun();*

*afterSimRun();*

*}*

*afterSimulation();*

*}*

*protected void afterSimulation(){*

*}*

*protected void beforeSimRun() {*

*}*

*protected void afterSimRun() {*

*}*

*protected abstract void executeSimRun();*

*}*

# Úloha 1

Ktorú stratégiu má Ján použiť, ak začína s prázdnym skladom a vždy objednáva rovnaké počty súčiastok?

Vykonali sme simuláciu jednotlivých stratégii a na základe výsledkov týchto simulácii zobrazených v tabuľke by sme zvolili stratégiu B.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Počet replikácií | A | B | C | D | E | G | H |
| 100 000 | 23099 | 11612 | 15968 | 15659 | 17873 | 9351 | 4913 |
| 1 000 000 | 23093 | 11616 | 15948 | 15688 | 17858 | 9349 | 4917 |
| 10 000 000 | 23086 | 11626 | 15951 | 15683 | 17867 | 9347 | 4915 |

Table 1 Porovnanie celkových nákladov simulovaných stratégií

# Úloha 2

Simulovali sme stratégiu E, pri ktorej sme prvých 15 týždňov boli zásobovaný dodávateľom 2 a zvyšné týždne zásoboval dodávateľ 1 pričom sme nemenili počet dodávaných súčiastok v porovnaní so stratégiami A,B,C,D.

Prvých 11 týždňov je Dodávateľ 1 spoľahlivejší ako Dodávateľ 2, pretože pravdepodobnosť , že dodávateľ 1 dodá súčiastky s pravdepodobnosťou v intervale <10, 70) je 100% a pri dodávateľovi 2 je to len 50%. Dodávateľ 2 síce môže dodať súčiastky aj s väčšou pravdepodobnosťou ako 70 avšak šanca, že tento jav nastane je len 10%. Od 11 týždňa sa spoľahlivosť dodávateľa 1 ešte zvyšuje nakoľko sa jeho interval mení na <30, 95). Ostáva teda spoľahlivejší aj po zvyšné týždne, čomu napomáha aj fakt, že pri dodávateľovi 1 sa jedná o rovnomerné rozdelenie, kde každá hodnota z intervalu má rovnakú pravdepodobnosť výberu, zatiaľ čo pri dodávateľovi 2 sa jedná o empirické rozdelenie a teda vyššie hodnoty ako napríklad 80, 90 atď majú nižšiu pravdepodobnosť výberu ako pri dodávateľovi 1.

Zvolili by sme pre celú periódu 30 týždňov dodávateľa 1. V aplikácii sme implementovali stratégiu F, ktorá sa za pomoci nami implementovaných generátorov snaží každý týždeň predikovať dopyt zákazníka vygenerovaním náhodnej hodnoty. Na základe tejto predikcie a aktuálne dostupných skladových zásob následne odhadne vhodné množstvo súčiastok, ktoré je daný týždeň potrebné objednať od dodávateľa a toto množstvo súčiastok sa objedná. Stratégia si ukladala štatistiky z týchto predikcií a na záver vypísala informáciu o priemernom počte dodávaných súčiastok v každej replikácii. Na základe výstupov z simulovania tejto stratégie sme vytvorili stratégiu G, v ktorej využívame len dodávateľa 1 a každý týždeň objednávame 75 tlmičov, 155 brzdových doštičiek a 91 svetlometov. Táto stratégia sa ukázala byť značne lepšia ako predošlé testované stratégie.

Následne sme stratégiu F upravili tak, aby štatistiky zberala pre jednotlivé týždne simulácie a vo výstupe dala informáciu o priemernom počte objednávaných súčiastok v každom z 30 týždňov. Na základe tohto výstupu sme testovali stratégiu H v ktorej sme opäť využívali iba dodávateľa 1 a každý týždeň sme objednávali 37 tlmičov, 77 brzdových doštičiek a 45 svetlometov. Táto stratégia sa ukázala byť spomedzi nami testovaných stratégií najlepšia vo výške celkových nákladov v snahe uspokojiť aj zákazníka. Najnižšie celkové náklady mala stratégia, ktorá neobjednávala žiadne súčiastky a platili sa len pokuty zákazníka, to však ale nie je želaný stav, nakoľko chceme dosiahnuť aj určitú mieru spokojnosti zákazníka.

Na základe našich zistení sme vytvorili tabuľku objednávok pre 30 týždňov. Obsah tejto tabuľky aplikácia načítava z konfiguračného csv súboru, ktorý sa nachádza pri zdrojových kódoch v src\main\java\com\example\Config\CustomStrategy.csv. Štruktúra csv súboru je nasledovná tlmiče, brzdové doštičky, svetlomety, dodávateľ. V aplikácii je následne túto konfiguráciu simulovať výberom CustomStrategy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Týždeň | Tlmiče | Brzdové Doštičky | Svetlomety | Dodávateľ |
| 1 | 37 | 77 | 45 | 1 |
| 2 | 37 | 77 | 45 | 1 |
| 3 | 37 | 77 | 45 | 1 |
| 4 | 37 | 77 | 45 | 1 |
| 5 | 37 | 77 | 45 | 1 |
| 6 | 37 | 77 | 45 | 1 |
| 7 | 37 | 77 | 45 | 1 |
| 8 | 37 | 77 | 45 | 1 |
| 9 | 37 | 77 | 45 | 1 |
| 10 | 37 | 77 | 45 | 1 |
| 11 | 37 | 77 | 45 | 1 |
| 12 | 37 | 77 | 45 | 1 |
| 13 | 37 | 77 | 45 | 1 |
| 14 | 37 | 77 | 45 | 1 |
| 15 | 37 | 77 | 45 | 1 |
| 16 | 37 | 77 | 45 | 1 |
| 17 | 37 | 77 | 45 | 1 |
| 18 | 37 | 77 | 45 | 1 |
| 19 | 37 | 77 | 45 | 1 |
| 20 | 37 | 77 | 45 | 1 |
| 21 | 37 | 77 | 45 | 1 |
| 22 | 37 | 77 | 45 | 1 |
| 23 | 37 | 77 | 45 | 1 |
| 24 | 37 | 77 | 45 | 1 |
| 25 | 37 | 77 | 45 | 1 |
| 26 | 37 | 77 | 45 | 1 |
| 27 | 37 | 77 | 45 | 1 |
| 28 | 37 | 77 | 45 | 1 |
| 29 | 37 | 77 | 45 | 1 |
| 30 | 37 | 77 | 45 | 1 |

Table 2 Objednávky