Міністерство освіти і науки України

Черкаський державний технологічний університет

Кафедра програмного забезпечення автоматизованих систем

**ЗВІТ**

з лабораторної роботи №6

з предмету «Аналіз великих даних»

|  |  |
| --- | --- |
| Перевірив:  к. т. н., д. каф. ПЗАС  Рідкокаша А.А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2019 р. | Виконав:  студент 1-го курсу  групи МПЗ-1904  Гаврилюк В. Є. |

Черкаси 2019

**Лабораторна робота №6**

**Тема:** Генетичний алгоритм.

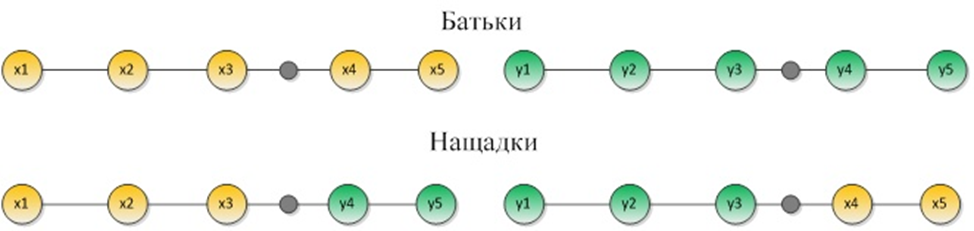
**Мета:** Вивчити основні методи генетичного пошуку. Навчитися використовувати генетичні методи для розв’язку оптимізаційних задач.

**Завдання:** Реалізувати генетичний алгоритм, застосувавши мови програмування.

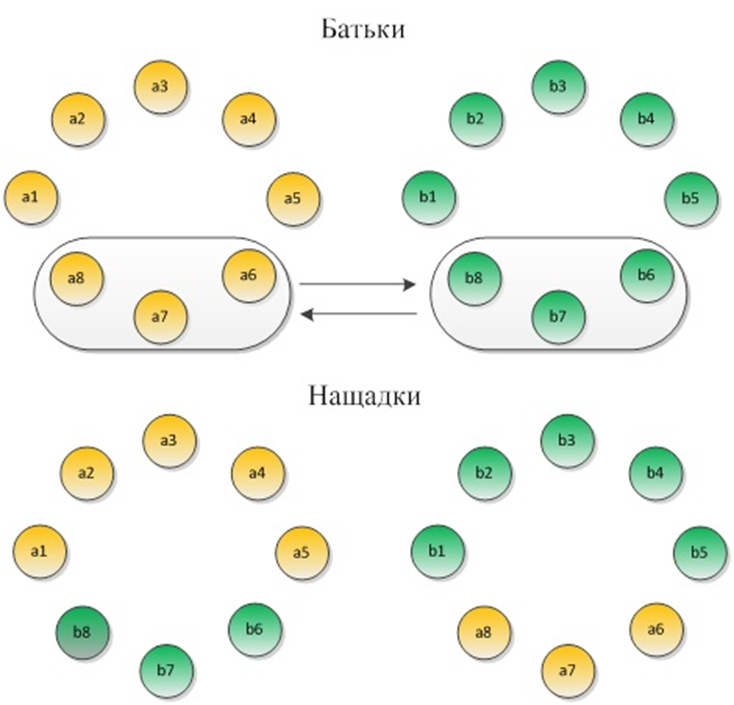
**Короткі теоретичні відомості**

Кросинговер - це один із видів оператора рекомбінації генетичного алгоритму. Застосовується на хромосомах з бінарними генами.

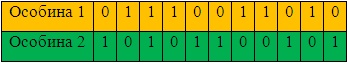
Одноточковий кросинговер (Single-point crossover) моделюється наступним чином. Нехай є дві батьківські особини з хромосомами і . Випадковим чином визначається точка всередині хромосоми (точка розриву), в якій обидві хромосоми діляться на дві частини і обмінюються ними. Такий тип кросинговеру називаються одноточковим, так як при ньому батьківські хромосоми розділяються тільки в одній випадкової точці.



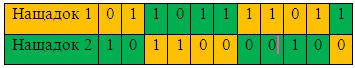
У двоточковому кросинговері (і в багатоточковому кросинговері також) хромосоми розглядаються як цикли, які формуються з'єднанням кінців лінійної хромосоми разом. Для заміни сегменту одного циклу сегментом іншого циклу потрібно вибрати дві точки розрізу. З цієї точки зору, одноточковий кросинговер може бути розглянутий як кросинговер з двома точками, але з однією точкою розриву, зафіксованій на початку рядка.



Для багатоточкового кросинговеру (Multi-point crossover), вибираємо точок розриву , , – кількість змінних (генів) у особині. Точки розриву вибираються випадково без повторень і сортуються в порядку зростання. При кросинговері походить обмін ділянками хромосом, обмеженими точками розрізу і таким чином отримують двох нащадків. Ділянка особини з першим геном до першої точки розрізу в обміні не бере участь:



Оберемо такі точки розриву кросинговеру: 2, 6 і 10. Отримаємо таких нащадків:



Хромосомні мутації — структурні зміни хромосом, хромосомні перебудови, які відбуваються синхронно в обох хроматидах. Перебудови можуть здійснюватися як в межах однієї хромосоми – внутрішньохромосомні мутації (делеція, інверсія, дуплікація), так і між хромосомами - міжхромосомні мутації (транслокація). Деякі типи хромосомних мутацій змінюють дію деяких генів і чинять на фенотип більш значний вплив, ніж генні мутації.

Основні типи хромосомних мутацій:

1. Делеції - втрата ділянки хромосоми або послідовність ДНК. При делеції виникає втрата генетичного матеріалу. При цьому може бути вилучена будь-яке число нуклеотидів, від одної основи до великого шматка хромосоми.
2. Дуплікація — подвоєння генів у певній ділянці хромосоми. Дуплікація генів може відбуватися в результаті помилки при гомологічній рекомбінації, в результаті ретротранспозиції або дуплікації всієї хромосоми.
3. Інверсія - хромосомна мутація, коли після двох розривів в одній хромосомі сегмент хромосоми, розташований між розривами, повертається на 180 ° і займає інвертоване положення.
4. Транслокація — перенесення частини хромосом на іншу негомологічну хромосому, як результат — зміна групи зчеплення генів. Загальне число генів не змінюється.

**Хід роботи**

Код програми, що реалізує генетичний алгоритм

ChildKindWrapperClass.ts – клас потомства.

import {ailab} from 'ark.ailab';

export class ChildKindWrapperClass {

individual: number;

geneticValue: number;

isMutant: boolean;

}

KindWrapperClass.ts – клас що уособлює вид.

import {ailab} from 'ark.ailab';

export class KindWrapperClass extends ChildKindWrapperClass {

qualityValue: number;

}

PopulationManagerClass.ts – допоміжний клас, що реалізує головну логіку генетичного алгоритму та контролює покоління особин.

import {ailab} from 'ark.ailab';

export class PopulationManagerClass {

const CROSSING\_POSSIBILITY: number = 0.75;

const MUTATON\_POSSIBILITY: number = 0.1;

private ArrayList<KindWrapperClass> population;

public PopulationManager(ArrayList<KindWrapperClass> population) {

this.population = population;

}

public PopulationManager(populationNum: number) {

this.population = new ArrayList<>();

for (let i = 0; i < populationNum; i++){

population.add( createKind() );

}

for (KindWrapperClass kind : population){

kind.qualityValue = calculateQualityValue(kind.geneticValue);

}

}

public ArrayList<KindWrapperClass> getPopulation() {

return population;

}

public calculateGeneticvalue(x: number): number {

return ( -1 \* Math.pow( (17\*Math.pow(x, 2) - 14\*x + 15), 2) ) / 256;

}

public calculateQualityValue(double individualGeneticValue): number {

return individualGeneticValue / calculatePopulationGeneticValue();

}

public KindWrapperClass createKind(){

minRange: number = 0;

maxRange: number = 255;

random: number = new Random();

return createKind( random.nextInt(maxRange - minRange + 1) + minRange );

}

public KindWrapperClass createKind(int individual){

KindWrapperClass kind = new KindWrapperClass();

kind.individual = individual;

kind.geneticValue = calculateGeneticvalue(kind.individual);

return kind;

}

public KindWrapperClass getKindByQualityRange(quality: number){

qualityRangeStepsValue: number = 0;

for (KindWrapperClass kind : population){

if (quality >= qualityRangeStepsValue && quality <= qualityRangeStepsValue + kind.qualityValue){

return kind;

}

qualityRangeStepsValue += kind.qualityValue;

}

return null;

}

public ChildKindWrapperClass speciesCrossesResult(KindWrapperClass parentOne, KindWrapperClass parentTwo){

if (Math.random() <= CROSSING\_POSSIBILITY){

int childIndividual = parentOne.individual | parentTwo.individual;

return kindMutation(childIndividual);

}else {

return null;

}

}

public ChildKindWrapperClass kindMutation(individual: number){

ChildKindWrapperClass child = new ChildKindWrapperClass();

if (Math.random() <= MUTATON\_POSSIBILITY){

child.individual = ~individual;

child.isMutant = true;

}else {

child.individual = individual;

child.isMutant = false;

}

child.geneticValue = calculateGeneticvalue(child.individual);

return child;

}

private calculatePopulationGeneticValue(): number {

populationGeneticValue: number = 0;

for (KindWrapperClass kind : population){

populationGeneticValue += kind.geneticValue;

}

return populationGeneticValue;

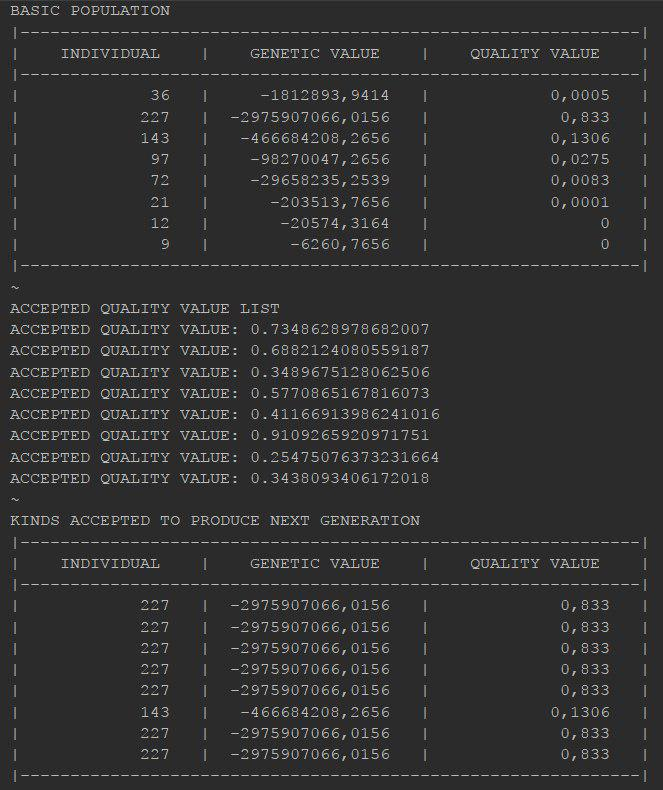
}

}

MainAppActivityClass.ts – головний клас, що дає початок процесу еволюції та виконує спостереження і логування.

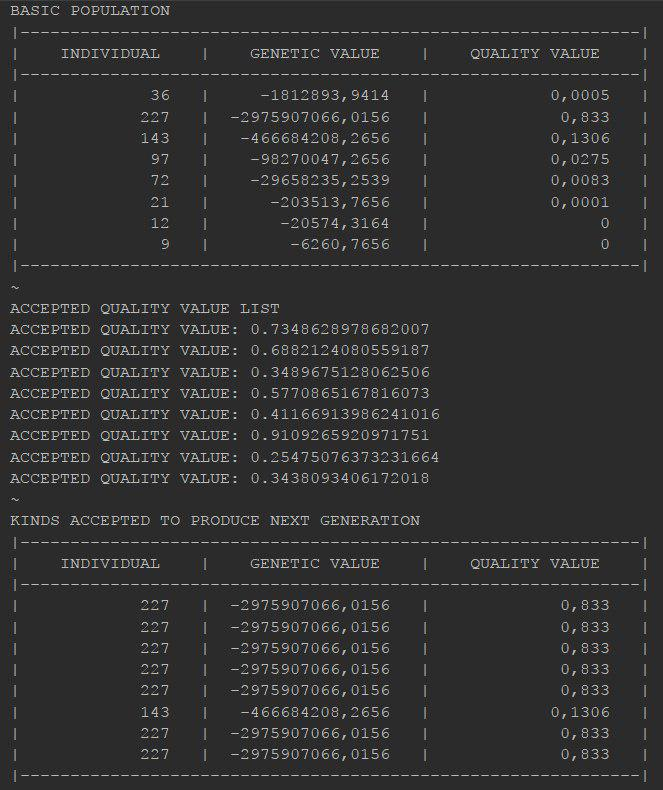
Результат роботи програми

Програма генерує початковий набір особин, що будуть давати потомство. Для даної лабораторної роботи початковий набір зображено на рисунку 1.



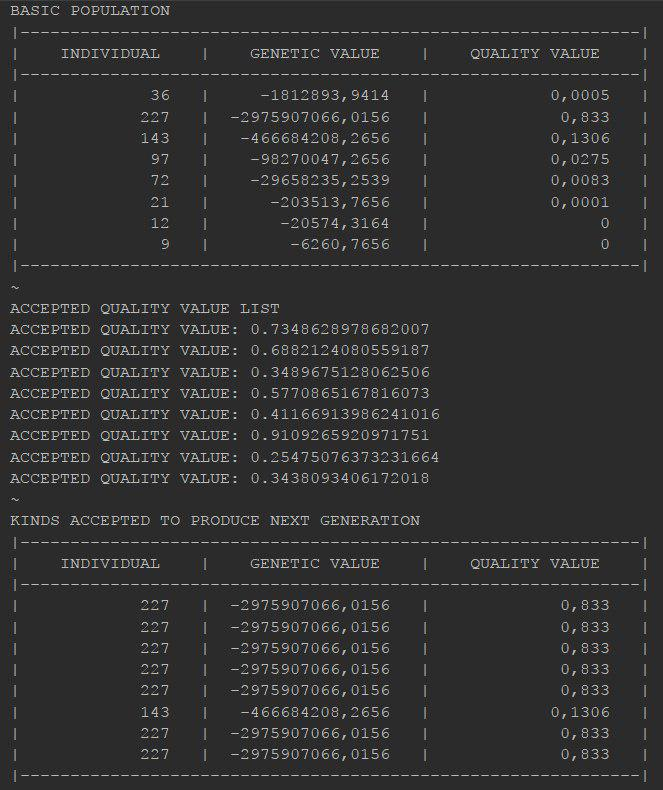
*Рис. 1 – Початковий набір особин популяції.*

Випадковим чином обираєтсья набір характеристик, що відбере особин здатних відтворити нащадків.



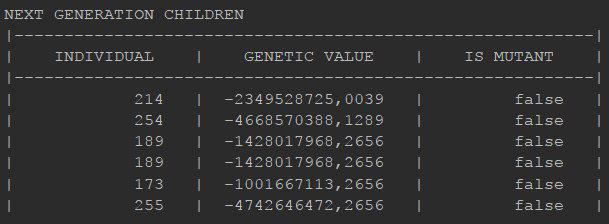
*Рис. 2 – Характеристики, яким мають відповідати особини, щоб дати нащадків.*

Потім серед усієї популяції відбираються особини, що підходять під визначені раніше характеристики (рис. 3).



*Рис. 3 – Особини, що здатні дати потомство.*

Особини, що здатні дати потомство групуються у пари. Отримані пари проходять перевірку чи дали воно потомство чи ні. Якщо пара створила нащадка, ця особина проходять перевірку на мутації. На рисунку 4 зображено список нащадків та інформація про мутації.



*Рис. 4 – Список нащадків та інформації про мутації нащадків.*

**Висновок:** На даній лабораторній роботі я ознайомився із генетичним алгоритмом. Вивчив основні методи генетичного пошуку. Навчився використовувати генетичні методи для розв’язку оптимізаційних задач. Реалізував генетичний алгоритм, застосувавши мову програмування Typescript.