

NAMA : IRMA ERYANTI PUTRI

NIM : 1800018272

## Parcel knapsack.php

```
1 <?php
2
3 class Parameters
4 {
5     public $file_name;
6     public $indexes;
7     public $columns;
8     public $population_size;
9     public $fitness;
10    public $max_generation;
11    public $budget;
12    public $crossover_rate;
13
14    public function __construct($parameters)
15    {
16        $this->file_name = $parameters['file_name'];
17        $this->indexes = $parameters['indexes'];
18        $this->columns = $parameters['columns'];
19        $this->population_size = $parameters['population_size'];
20        $this->fitness = $parameters['fitness'];
21        $this->max_generation = $parameters['max_generation'];
22        $this->budget = $parameters['budget'];
23        $this->crossover_rate = $parameters['crossover_rate'];
24    }
25 }
26
27 class Products
28 {
29     public static function catalogue($parameters)
30     {
31         $raw_data = file($parameters->file_name);
32         foreach ($raw_data as $val) {
33             $data[] = explode(",", $val);
34         }
35     }
36 }
```

```
35
36     foreach ($data as $key => $val) {
37         foreach (array_keys($val) as $subkey) {
38             if ($subkey == $parameters->indexes[$subkey]) {
39                 $data[$key][$parameters->columns[$subkey]] = $data[$key][$subkey];
40                 unset($data[$key][$subkey]);
41             }
42         }
43     }
44     return [
45         'dataset' => $data,
46         'gen_length' => count($data)
47     ];
48 }
49 }
50
51 class Generate
52 {
53     public $population_size;
54
55     function __construct($parameters)
56     {
57         $this->population_size = $parameters->population_size;
58     }
59
60     public function parcel($parameters)
61     {
62         $ret = [];
63         $catalogue = Products::catalogue($parameters);
64         for ($i = 0; $i <= $catalogue['gen_length'] - 1; $i++) {
65             $ret[] = rand(0, 1);
66         }
67         return $ret;
68     }
69 }
```

```
69
70 public function initialPopulation($parameters)
71 {
72     $ret = [];
73     for ($i = 0; $i <= $this->population_size - 1; $i++) {
74         $ret[] = $this->parcel($parameters);
75     }
76     return $ret;
77 }
78
79 public static function randomZeroToOne()
80 {
81     return (float) rand() / (float) getrandmax();
82 }
83
84 public static function randomGenLength($parameters)
85 {
86     $catalogue = Products::catalogue($parameters);
87     return rand(1, $catalogue['gen_length'] - 1);
88 }
89
90
91 class Pairing
92 {
93     public static function binaryWithProduct($populations, $parameters)
94     {
95         $ret = [];
96         foreach ($populations as $id => $parcel) {
97             foreach ($parcel as $index => $binary) {
98                 if ($binary == 1) {
99                     $selected_product = Products::catalogue($parameters)['dataset'][$index]['item'];
100                     $price = Products::catalogue($parameters)['dataset'][$index]['price'];
101                     $ret[$id][] = [
102                         'product' => $selected_product,
103                         'price' => $price
104                     ];
105                 }
106             }
107         }
108         return $ret;
109     }
110
111     public static function binaryWithProductFound($parcels, $parameters)
112     {
113         $ret = [];
114         foreach ($parcels as $index => $binary) {
115             if ($binary == 1) {
116                 $selected_product = Products::catalogue($parameters)['dataset'][$index]['item'];
117                 $price = Products::catalogue($parameters)['dataset'][$index]['price'];
118                 $ret[] = [
119                     'product' => $selected_product,
120                     'price' => $price
121                 ];
122             }
123         }
124         return $ret;
125     }
126 }
127
128
129 class Calculate
130 {
131     public static function parcelPrice($selected_product)
132     {
133         $ret = [];
134         foreach ($selected_product as $products) {
135             // $ret[] = array_sum( array_column($products, 'price'));
136         }
137         return $ret;
138     }
139 }
140
```

```
105
106     }
107 }
108 return $ret;
109 }
110
111 public static function binaryWithProductFound($parcels, $parameters)
112 {
113     $ret = [];
114     foreach ($parcels as $index => $binary) {
115         if ($binary == 1) {
116             $selected_product = Products::catalogue($parameters)['dataset'][$index]['item'];
117             $price = Products::catalogue($parameters)['dataset'][$index]['price'];
118             $ret[] = [
119                 'product' => $selected_product,
120                 'price' => $price
121             ];
122         }
123     }
124     return $ret;
125 }
126
127
128
129 class Calculate
130 {
131     public static function parcelPrice($selected_product)
132     {
133         $ret = [];
134         foreach ($selected_product as $products) {
135             // $ret[] = array_sum( array_column($products, 'price'));
136         }
137         return $ret;
138     }
139 }
140
```

```
AnalyticsTest.php x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x
140
141 class Optimized
142 {
143     public static function parcelFound($bests)
144     {
145         $max_price = max(array_column($bests, 'price'));
146         $index = array_search($max_price, array_column($bests, 'price'));
147         return $bests[$index];
148     }
149 }
150
151 class Fitness
152 {
153     public static function evaluation($parcelPrice, $budget)
154     {
155         $ret = [];
156         $negative_residu = 2;
157         $positive_residu = 1;
158         foreach ($parcelPrice as $total) {
159             $residu = $budget - $total;
160             if ($residu < 0) {
161                 $ret[] = 1 / (1 + $negative_residu);
162             }
163             if ($residu > 0) {
164                 $ret[] = 1 / (1 + $positive_residu);
165             }
166             if ($residu === 0) {
167                 $ret[] = 1;
168             }
169         }
170         return $ret;
171     }
172
173     public static function hasOne($parcel_fitness, $parcel_population)
174     {
```

```
AnalyticsTest.php x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x
172
173     public static function hasOne($parcel_fitness, $parcel_population)
174     {
175         $index = array_search(1, $parcel_fitness);
176         if ($index) {
177             return $parcel_population[$index];
178         }
179     }
180
181     public static function hasHalf($parcel_population, $parcel_fitness, $parcel_price)
182     {
183         $max_value = 1 / 2;
184         $indexes = [];
185         $optimized = [];
186         foreach ($parcel_fitness as $key => $fitness_value) {
187             if ($max_value === $fitness_value) {
188                 $indexes[] = $key;
189             }
190         }
191         foreach ($parcel_price as $key => $val) {
192             foreach ($indexes as $ind) {
193                 if ($key === $ind) {
194                     $optimized[] = [
195                         'index' => $key,
196                         'price' => $val
197                     ];
198                     break;
199                 }
200             }
201         }
202         if (!$optimized) {
203             ## TODO refactor it!
204             //throw new exception(' ada unoptimized yang seluruh fitness 0.333');
205         }
206         $price = max(array_column($optimized, 'price'));
207         $id = array_search($price, array_column($optimized, 'price'));
```

```
AnalyticsTest.php x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x
206 $price = max(array_column($optimized, 'price'));
207 $id = array_search($price, array_column($optimized, 'price'));
208 $products = $parcel_population[$optimized[$id]['index']];
209 return [
210     'price' => $price,
211     'parcel' => $products
212 ];
213 }
214 }
215
216 class RandomNumbers
217 {
218     public static function zeroToOne($population_size)
219     {
220         for ($i = 0; $i <= $population_size - 1; $i++) {
221             $ret[] = Generate::randomZeroToOne();
222         }
223         return $ret;
224     }
225 }
226
227 class RouletteWheelSelection
228 {
229     public $population_size;
230
231     function __construct($population_size)
232     {
233         $this->population_size = $population_size;
234     }
235
236     function probability($parcel_fitness)
237     {
238         foreach ($parcel_fitness as $fitness) {
239             $ret[] = $fitness / array_sum($parcel_fitness);
240         }
241     }
242 }
```

Spaces: 4 PHP

```
AnalyticsTest.php x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x
241     return $ret;
242 }
243
244 function cumulative($parcel_fitness)
245 {
246     foreach ($this->probability($parcel_fitness) as $key => $probability) {
247         if ($key === 0) {
248             $ret[$key] = $probability;
249         } else {
250             $ret[$key] = $probability + $ret[$key - 1];
251         }
252     }
253     return $ret;
254 }
255
256 function selection($parcel_fitness, $parcel_populations)
257 {
258     $randomZeroToOne = RandomNumbers::zeroToOne($this->population_size);
259     foreach ($randomZeroToOne as $key => $random) {
260         foreach ($this->cumulative($parcel_fitness) as $subkey => $roulette) {
261             if ($random < $roulette) {
262                 $ret[$key] = $parcel_populations[$subkey];
263                 break;
264             }
265         }
266     }
267     return $ret;
268 }
269 }
270
271 class CrossOver
272 {
273     public $population_size;
274     public $crossover_rate;
275 }
```

Spaces: 4 PHP

```
AnalyticsTest.php x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x x
271 class Crossover
272 {
273     public $population_size;
274     public $crossover_rate;
275
276     function __construct($population_size, $crossover_rate)
277     {
278         $this->population_size = $population_size;
279         $this->crossover_rate = $crossover_rate;
280     }
281
282     function selectedParcel()
283     {
284         $randomZeroToOne = RandomNumbers::zeroToOne($this->population_size);
285         foreach ($randomZeroToOne as $key => $val) {
286             if ($val < $this->crossover_rate) {
287                 $ret[] = $key;
288             }
289         }
290         return $ret;
291     }
292
293     function generateCombination($selected_parcel)
294     {
295         $ret = [];
296         foreach ($selected_parcel as $val) {
297             $acak = $selected_parcel[array_rand($selected_parcel)];
298             $ret[] = [$val, $acak];
299         }
300         return $ret;
301     }
302
303     function combination($selected_parcel)
304     {
305         $counter = 0;
```

```
AnalyticsTest.php x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x x
303     function combination($selected_parcel)
304     {
305         $counter = 0;
306         while ($counter < 1) {
307             $combination = $this->generateCombination($selected_parcel);
308             foreach ($combination as $val) {
309                 $sum[] = count(array_unique($val));
310             }
311             if (array_sum($sum) === count($combination) * 2) {
312                 return $combination;
313             }
314             $counter = 0;
315             $sum = [];
316         }
317     }
318
319     function cutPositions($parameters, $number_of_parents)
320     {
321         for ($i = 0; $i <= $number_of_parents - 1; $i++) {
322             $ret[] = Generate::randomGenLength($parameters);
323         }
324         return $ret;
325     }
326
327     function newParcel($selected_product, $selected_individuals, $cut_positions, $combinations, $parameters)
328     {
329         $catalogue = Products::catalogue($parameters);
330         $offspring = [];
331
332         foreach ($combinations as $key => $combination) {
333             $individu1 = $selected_product[$combination[0]];
334             $individu2 = $selected_product[$combination[1]];
335
336             for ($i = 0; $i <= $catalogue['gen_length'] - 1; $i++) {
337                 if ($i < $cut_positions[$key]) {
```

```
336         }
337         if ($i < $cut_positions[$key]) {
338             $offspring[$key][] = $individu1[$i];
339         } else {
340             $offspring[$key][] = $individu2[$i];
341         }
342     }
343     $ret[] = [
344         'index' => $selected_individuals[$key],
345         'offspring' => $offspring[$key]
346     ];
347 }
348 return $ret;
349 }
350
351 function newParcelPopulation($selected_parcel_ids, $new_parcel)
352 {
353     for ($i = 0; $i <= count($selected_parcel_ids) - 1; $i++) {
354         foreach ($new_parcel as $key => $individual) {
355             if ($i == $individual['index']) {
356                 $ret[$i][$key] = [
357                     'a' => 'new',
358                     'b' => $i
359                 ];
360             } else {
361                 $ret[$i][$key] = [
362                     'a' => 'old',
363                     'b' => $i
364                 ];
365             }
366         }
367     }
368
369     foreach ($ret as $key => $z) {
370         $index = array_search('new', array_column($z, 'a'));
371         if ($index !== false) {
372             $result[$key] = $new_parcel[$index]['offspring'];
373         } else {
374             $result[$key] = $selected_parcel_ids[$key];
375         }
376     }
377     return $result;
378 }
379
380 }
381
382 $parameters = [
383     'file_name' => 'product.txt',
384     'indexes' => [0, 1],
385     'columns' => ['item', 'price'],
386     'population_size' => 30,
387     'fitness' => 1000,
388     'max_generation' => 50,
389     'crossover_rate' => 0.75,
390     'budget' => 350000
391 ];
392
393 $parameters = new Parameters($parameters);
394 $parcel = new Generate($parameters);
395
396 $generation = 0;
397 $parcel_population = $parcel->initialPopulation($parameters);
398
399 while ($generation < $parameters->max_generation) {
400     $selected_products = Pairing::binaryWithProduct($parcel_population, $parameters);
401     $parcel_prices = Calculate::parcelPrice($selected_products);
402     $parcel_fitness = Fitness::evaluation($parcel_prices, $parameters->budget);
403     $optimized_parcel = Fitness::hasOne($parcel_fitness, $parcel_population);
404 }
```

```
369     foreach ($ret as $key => $z) {
370         $index = array_search('new', array_column($z, 'a'));
371         if ($index || $index == 0) {
372             $index_new = array_search($z[$index]['b'], array_column($new_parcel, 'index'));
373             $result[] = $new_parcel[$index_new]['offspring'];
374         } else {
375             $result[] = $selected_parcel_ids[$key];
376         }
377     }
378     return $result;
379 }
380 }
381
382 $parameters = [
383     'file_name' => 'product.txt',
384     'indexes' => [0, 1],
385     'columns' => ['item', 'price'],
386     'population_size' => 30,
387     'fitness' => 1000,
388     'max_generation' => 50,
389     'crossover_rate' => 0.75,
390     'budget' => 350000
391 ];
392
393 $parameters = new Parameters($parameters);
394 $parcel = new Generate($parameters);
395
396 $generation = 0;
397 $parcel_population = $parcel->initialPopulation($parameters);
398
399 while ($generation < $parameters->max_generation) {
400     $selected_products = Pairing::binaryWithProduct($parcel_population, $parameters);
401     $parcel_prices = Calculate::parcelPrice($selected_products);
402     $parcel_fitness = Fitness::evaluation($parcel_prices, $parameters->budget);
403     $optimized_parcel = Fitness::hasOne($parcel_fitness, $parcel_population);
404 }
```

```
AnalyticsTest.php x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x x
400 $selected_products = Pairing::binaryWithProduct($parcel_population, $parameters);
401 $parcel_prices = Calculate::parcelPrice($selected_products);
402 $parcel_fitness = Fitness::evaluation($parcel_prices, $parameters->budget);
403 $optimized_parcel = Fitness::hasOne($parcel_fitness, $parcel_population);
404 if (is_array($optimized_parcel)) {
405     print_r($optimized_parcel);
406     exit();
407 }
408 $unoptimized_parcel = Fitness::hasHalf($parcel_population, $parcel_fitness, $parcel_prices);
409 $parcel_selection = new RouletteWheelSelection($parameters->population_size);
410 $selected_parcel_ids = $parcel_selection->selection($parcel_fitness, $parcel_population);
411
412 $crossover = new Crossover($parameters->population_size, $parameters->crossover_rate);
413 $selected_parcel = $crossover->selectedParcel();
414 $combination = $crossover->combination($selected_parcel);
415 $cut_positions = $crossover->cutPositions($parameters, count($selected_parcel));
416 $new_parcel = $crossover->newParcel($selected_parcel_ids, $selected_parcel, $cut_positions, $combination, $parameters);
417 $parcel_population = $crossover->newParcelPopulation($selected_parcel_ids, $new_parcel);
418 $bests[] = $unoptimized_parcel;
419
420 $generation++;
421 }
422
423 echo '<p>';
424 $optimized = Optimized::parcelFound($bests);
425 $yourparcel = Pairing::binaryWithProductFound($optimized['parcel'], $parameters);
426 echo "Your parcel: " . count($yourparcel) . " items <br>";
427 print_r($optimized);
428 echo '<br>';
429 foreach ($yourparcel as $item) {
430     print_r($item);
431     echo '<br>';
432 }
```

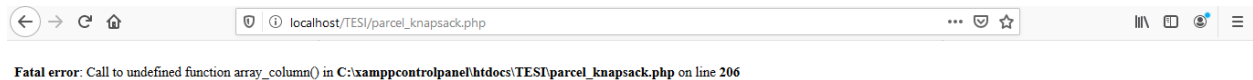
Spaces: 4 PHP

## Products.txt

```
Analytics x MutationTest.php x QuotationTest.php x SelectionTest.php x MiscellaneousTest.php x parcel_knapsack.php x ChromosomeTest.php x product.txt x
1 Chitato 68 gr, 8900
2 Teh Sosro Kotak, 6900
3 Botan Mackarel 425 gr, 28900
4 Keju Cheddar 180 gr, 15250
5 Palm Fruit Kurma 500 gr, 56900
6 Marjan Syrup 460 ml, 17900
7 365 Wafer Stick 500 gr, 32390
8 Nissin Biscuit Lemonia Twist 340 gr, 22500
9 Kokola Wafer Cream 252 gr, 18500
10 Sari Kacang Hijau 150 ml, 7790
11 Pop Mie Pake Nasi 75 gr, 8000
12 Teriyaki Saori 275 ml, 17900
13 Dua Belibis Sambal 135/340/535 gr, 8650
14 Teh Hijau Kepala Djenggot 60 gr, 11900
15 Madunasa 150 gr, 16900
16 Makaroniku 200 gr, 14900
17 Ultra Low Fat Susu UHT 1000 ml, 19900
18 Khong Guan Malkist Salut Cokelat 120 gr, 4990
19 Kopi susu ABC, 3000
20 Coca cola, 15000
21 Khong guan, 113000
22 Danis biscuit, 46900
23
```

Tab Size: 4 Plain Text

Output:



Your parcel: 11 items

Array ( [price] => 348330 [parcel] => Array ( [0] => 1 [1] => 0 [2] => 0 [3] :  
[16] => 0 [17] => 0 [18] => 0 [19] => 0 [20] => 1 [21] => 1 ) )

Array ( [product] => Chitato 68 gr [price] => 8900 )

Array ( [product] => Palm Fruit Kurma 500 gr [price] => 56900 )

Array ( [product] => 365 Wafer Stick 500 gr [price] => 32390 )

Array ( [product] => Nissin Biscuit Lemonia Twist 340 gr [price] => 22500 )

Array ( [product] => Kokola Wafer Cream 252 gr [price] => 18500 )

Array ( [product] => Sari Kacang Hijau 150 ml [price] => 7790 )

Array ( [product] => Teriyaki Saori 275 ml [price] => 17900 )

Array ( [product] => Dua Belibis Sambal 135/340/535 gr [price] => 8650 )

Array ( [product] => Makaroniku 200 gr [price] => 14900 )

Array ( [product] => Khong guan [price] => 113000 )

Array ( [product] => Danis biscuit [price] => 46900 )