# Specifikace pro zápočtový program
# Artem Kopyl
# 14.05.2023

# Téma programu

Hlavním cílem zápočtového projektu je naučit se a implementovat evoluční (genetické) algoritmy pro řešení vyhledávacích problémů. Výběr algoritmu je podpořen skutečností, že algoritmus je nejlépe implementován ve stylu OOP, což mi umožní ukázat znalosti získané během semestru. Tento algoritmus budu demonstrovat na řešení klasického Salesmanova problému (TSP). Je dána množina měst a vzdálenost mezi jednotlivými dvojicemi měst, problém spočívá v nalezení nejkratší možné trasy, která navštíví každé město přesně jednou a vrátí se do výchozího bodu.

# Teorie

**Definice pojmů:**
Generace - množina vygenerovaných řešení
Chromozom - instance možného řešení (generační prvky)
Crossover (operator) - genetický mechanismus používaný ke kombinaci genetické informace dvou rodičů za účelem vytvoření nového potomka.
Mutation (operator) - genetický mechanismus k vytvoření rozmanitosti budoucích generací
Selection - proces výběru rodičů, kteří se páří a rekombinují, aby vytvořili potomky pro další generaci

**Základ genetických algoritmů**
Genetické algoritmy (GA) jsou adaptivní heuristické vyhledávací algoritmy, které patří do větší části evolučních algoritmů. Genetické algoritmy vycházejí z myšlenek přírodního výběru a genetiky. Jedná se o inteligentní využití náhodného hledání opatřeného historickými daty k nasměrování hledání do oblasti lepšího výkonu v prostoru řešení. Běžně se používají ke generování kvalitních řešení optimalizačních problémů a problémů vyhledávání.

Genetické algoritmy simulují proces přírodního výběru, což znamená, že ty druhy, které se dokáží přizpůsobit změnám v prostředí, jsou schopny přežít a rozmnožit se a přejít do další generace. Zjednodušeně řečeno, simulují "přežití nejsilnějších" mezi jedinci po sobě jdoucích generací pro řešení problému. Každá generace se skládá z populace jedinců a každý jedinec představuje bod v prostoru hledání a možné řešení. Tímto způsobem, genetické algoritmy posouvají nejlepší nalezené řešení směrem ke skutečné (hledané) hodnotě v rámci každé vypočtené generace.

Genetické algoritmy jsou založeny na analogii s genetickou strukturou a chováním chromozomů populace.

Na této analogii je založen základ GA -

- Jedinci v populaci soutěží o zdroje a páření
- Ti jedinci, kteří jsou úspěšní (nejschopnější), se pak páří, aby vytvořili více potomků než ostatní
- Geny (některé vlastnosti řešení, které jednotlivec reprezentuje) od "nejschopnějších" rodičů se šíří v celé generaci, to znamená, že někdy rodiče vytvoří potomstvo, které je lepší než kterýkoli z rodičů.
- Každá následující generace je tak vhodnější pro své prostředí.

# Podrobnosti o realizací a rozhraní programu

**I/O:** Program obdrží sadu měst, která jsou zadána svými názvy a souřadnicemi na mapě. Po zpracování dat a spuštění algoritmu se uživateli zobrazí nejkratší cesta a její délka.

- Input - posloupnost měst a jejich souřadnic
- Output - posloupnost měst v nalezeném nejlepším řešení

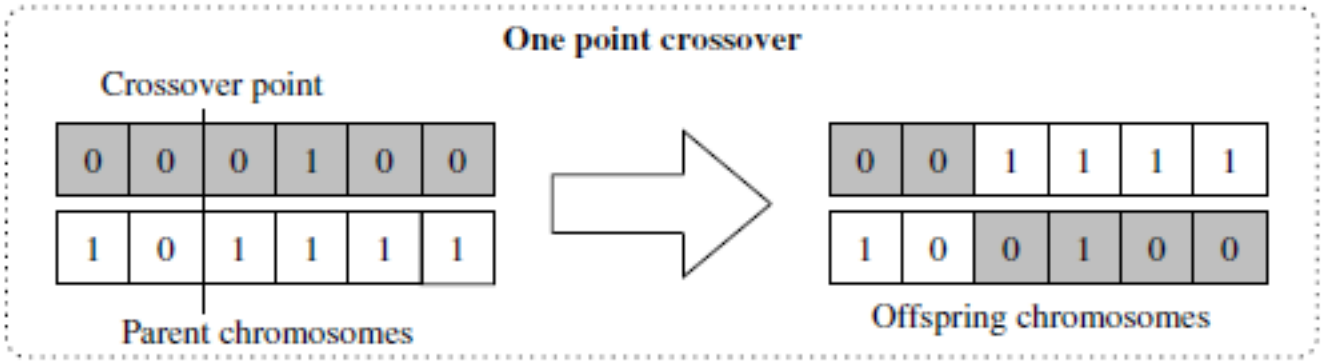
Software také podporuje testovací režim, ve kterém jsou data o městech generována nezávisle (uživatel musí pouze zadat počet měst, která chce otestovat).

K tomu musí být program spuštěn s argumentem v příkazovém řádku (ve složce projektu):

dotnet run (# měst)

**Kódování chromozomu:** *permutace měst na cestě*

**Crossover:** upravený *One Point Crossover*
Vyberou se dva chromozomy z aktuální generace. V jejich sadě genů je vybrán bod křížení, podle kterého se části těchto chromozomů přeskupí.
Protože chromozomy v našem případě představují permutace posloupnosti indexů měst, může se stát, že se některý index města bude opakovat (což není povoleno).




Proto algoritmus opravíme:
1) řekněme, že máme dva chromozomy Táta a Máma.
2) Vybereme bod křížení (část chromozomu před tímto bodem je hlava chromozomu a zbytek je ocas).
První syn - hlava otce + geny od matky, které nebyly v hlavě otce (jejich pořadí je zachováno).

PŘÍKLAD

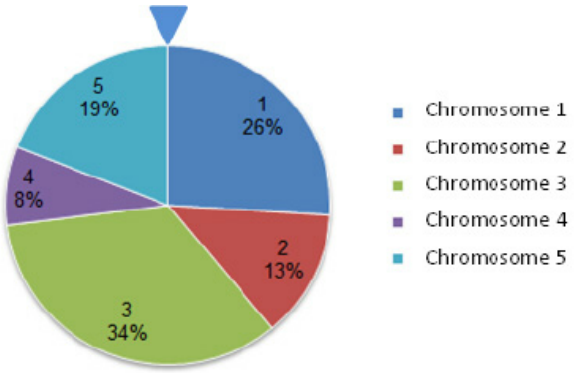
Father = [ 1, 3, 5, 4, 6, 7, 0, 2, 9, 8]
Mother = [ 1, 5, 6, 7, 2, 4, 8, 9, 3, 0] (geny v otce, zbytek)

První syn = [ 1, 3, 5, 4, 6, 7, 2, 8, 9, 0]
Druhý syn - symetricky

**Selection:** *Roulette Wheel Selection (schematický popis principu)*
Vezměme si kruhové kolo. Kolo je rozděleno na n částí, kde n je počet jedinců v populaci. Každý jedinec dostane část kruhu, která je úměrná jeho hodnotě fitness. Na obvodu kola je zvolen pevný bod, jak je znázorněno na obrázku, a kolo se otáčí.

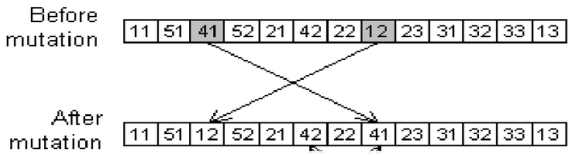
Oblast kola, která se nachází před pevným bodem, je vybrána jako rodičovská. Pro druhého rodiče se opakuje stejný postup. Je zřejmé, že zdatnější jedinec má na kole větší část, a tudíž větší šanci, že se při otáčení kola ocitne před pevným bodem. Pravděpodobnost výběru jedince tedy přímo závisí na jeho zdatnosti.

Při implementaci používáme následující kroky:
- Vypočítejte S - součet fitnessů
- Vypočítejte pravděpodobnost výběru jedince (fitness jedince / S).
- Normalizujte pravděpodobnosti
- Poté můžeme vybírat jedince k chovu podle jejich pravděpodobností

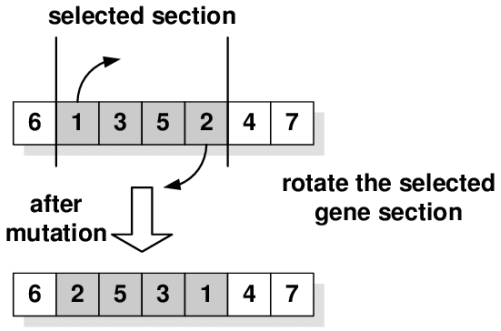



**Mutation:** *Swap Mutation & Inversion Mutation*
Při psaní programu jsem zjistil, že kombinace několika mutačních operátorů může vyhledávání urychlit, na rozdíl od situace, kdy jsem použil pouze jeden z nich. Jedná se o kombinaci Swap Mutation a Inversion Mutation.

*Swap Mutation* - Při výměnné mutaci vybereme náhodně dvě pozice na chromozomu a vyměníme jejich hodnoty.




*Inversion Mutation* - Z celého chromozomu je vybrána podmnožina genů, která je invertována.




(Pravděpodobnost mutace v programu byla 5 %.)

**Podmínka ukončení**
Program byl doplněn o kontrolu maximálního počtu vypočtených generací a maximálního počtu generací bez změny nejlepšího chromozomu.

# Testování projektu

Pro účely testování jsem použil [NEOS Concorde TSP Solver](NEOS Concorde TSP Solver).

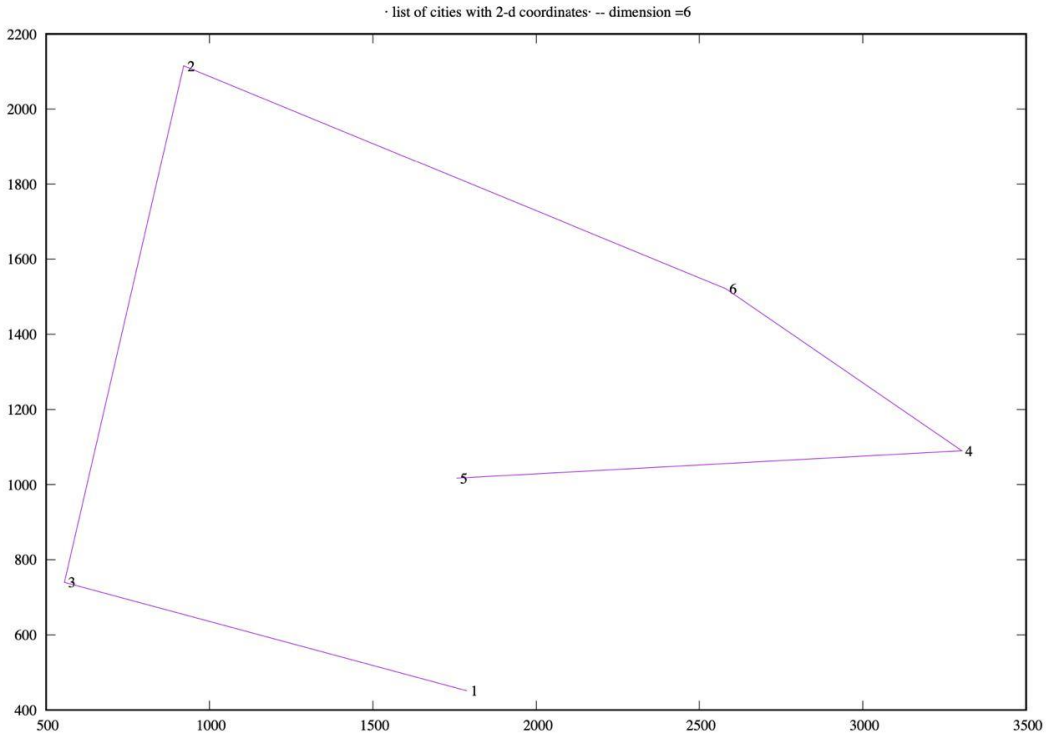
## Test #1

```
[who@mac tsp-genetics]$ dotnet run 6
City 0 | (1788 451)
City 1 | (921 2115)
City 2 | (556 740)
City 3 | (3303 1090)
City 4 | (1756 1017)
City 5 | (2581 1521)
GENERATION #1 - 7519
GENERATION #2 - 7519
GENERATION #3 - 7408
GENERATION #4 - 7408
GENERATION #5 - 7408
GENERATION #6 - 7408
GENERATION #7 - 7408
GENERATION #8 - 7408
GENERATION #9 - 7408
GENERATION #10 - 7408
GENERATION #11 - 7408
GENERATION #12 - 7408
GENERATION #13 - 7408
GENERATION #14 - 7408
GENERATION #15 - 7408
GENERATION #16 - 7408
GENERATION #17 - 7408
GENERATION #18 - 7408
GENERATION #19 - 7408
GENERATION #20 - 7408
GENERATION #21 - 7408
GENERATION #22 - 7408
GENERATION #23 - 7408

Found solution - 7407,624111956945
4->0->2->1->5->3
[who@mac tsp-genetics]$
```

```
class Config{
    3 references
    public static int POPULATION_COUNT = 50;
    1 reference
    public static int GENERATIONS_COUNT = 200;
    2 references
    public static double MUTATION_CHANCE = 0.05;
    1 reference
    public static int NO_IMPROVEMENT_GEN = 20;
    1 reference
    public static int MAP_WIDTH = 4000;
    1 reference
    public static int MAP_HEIGHT = 2200;
}
```

```
************************************************************
/home/neos5/bin/concorde.cplex -s 99 -f -N 2 sample.dat2
Host: neos  Current process id: 64237
Using random seed 99
nnodes = 6
Optimal Solution: 7408.00
Total Running Time: 0.00 (seconds)


***  ***


*** You chose the Concorde(CPLEX) solver ***


*** Cities are numbered 0..n-1 and each line shows a leg from one city to the next
 followed by the distance rounded to integers***

6 6
0 2 1265
2 1 1423
1 5 1763
5 3 841
3 4 1549
4 0 567

***PDF of optimal tour***unzip this file***
```

Additional Output:
[13353787-EaczbHhy-solver-output.zip]



Test #2

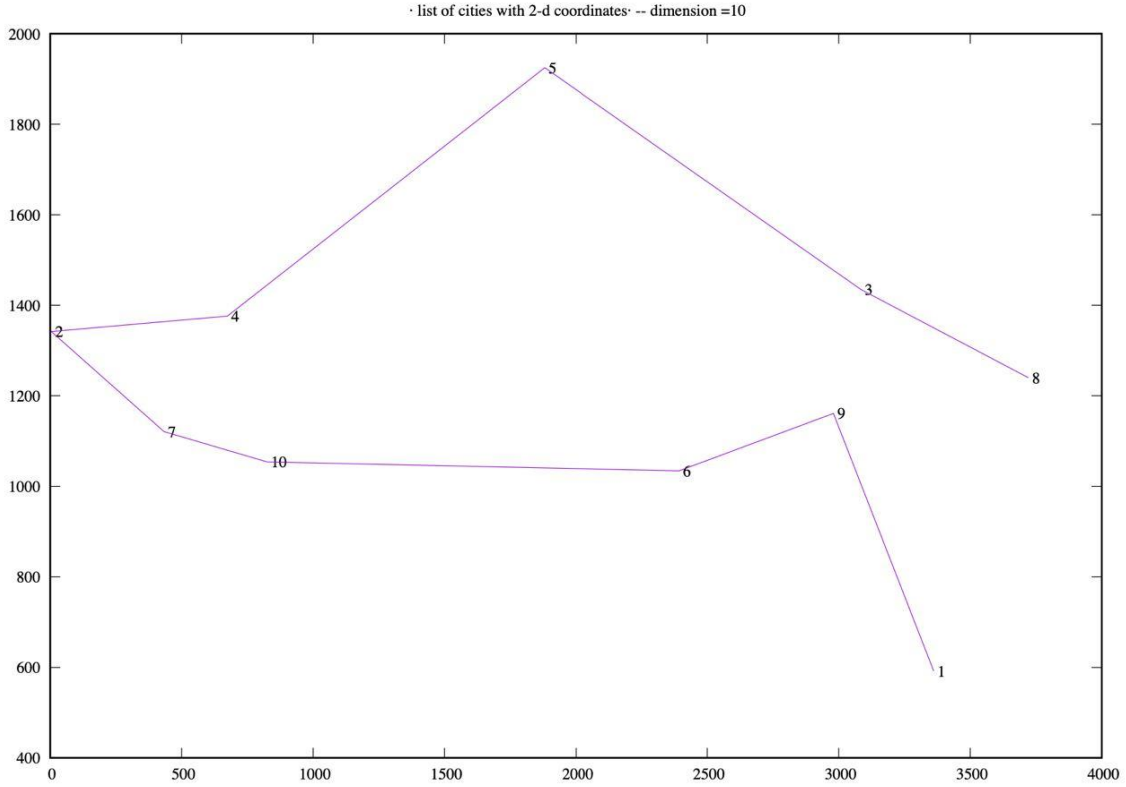
```
City 0 | (3361 592)
City 1 | (4 1342)
City 2 | (3084 1435)
City 3 | (674 1376)
City 4 | (1881 1925)
City 5 | (2393 1034)
City 6 | (433 1121)
City 7 | (3721 1240)
City 8 | (2980 1161)
City 9 | (825 1054)
GENERATION #1 - 10829
GENERATION #2 - 9871
GENERATION #3 - 9871
GENERATION #4 - 9777
GENERATION #5 - 9427
GENERATION #6 - 8963
GENERATION #7 - 8837
GENERATION #8 - 8832
GENERATION #9 - 8832
GENERATION #10 - 8832
GENERATION #11 - 8664
GENERATION #12 - 8664
GENERATION #13 - 8664
GENERATION #14 - 8664
GENERATION #15 - 8664
GENERATION #16 - 8664
GENERATION #17 - 8664
GENERATION #18 - 8664
GENERATION #19 - 8664
GENERATION #20 - 8664
GENERATION #21 - 8664
GENERATION #22 - 8664
GENERATION #23 - 8664
GENERATION #24 - 8664
GENERATION #25 - 8664
GENERATION #26 - 8664
GENERATION #27 - 8664
GENERATION #28 - 8664
GENERATION #29 - 8664
```

```
GENERATION #38 - 8589
GENERATION #39 - 8437
GENERATION #40 - 8437
GENERATION #41 - 8437
GENERATION #42 - 8437
GENERATION #43 - 8437
GENERATION #44 - 8437
GENERATION #45 - 8437
GENERATION #46 - 8437
GENERATION #47 - 8437
GENERATION #48 - 8437
GENERATION #49 - 8437
GENERATION #50 - 8437
GENERATION #51 - 8437
GENERATION #52 - 8437
GENERATION #53 - 8437
GENERATION #54 - 8437
GENERATION #55 - 8437
GENERATION #56 - 8437
GENERATION #57 - 8437
GENERATION #58 - 8437
GENERATION #59 - 8437

Found solution - 8437,031707065067
8->5->9->6->1->3->4->2->7->0
```

```java
class Config{
    3 references
    public static int POPULATION_COUNT = 50;
    1 reference
    public static int GENERATIONS_COUNT = 200;
    2 references
    public static double MUTATION_CHANCE = 0.05;
    1 reference
    public static int NO_IMPROVEMENT_GEN = 20;
    1 reference
    public static int MAP_WIDTH = 4000;
    1 reference
    public static int MAP_HEIGHT = 2200;
}
```

```
************************************************************
/home/neos5/bin/concorde.cplex -s 99 -f -N 2 sample.dat2
Host: athene  Current process id: 45351
Using random seed 99
nnodes = 10
Set initial upperbound to 8438 (from tour)
Basic dual change required, but no candidate edges
  LP Value  1: 6928.000000  (0.00 seconds)
  LP Value  2: 8438.000000  (0.00 seconds)
New lower bound: 8438.000000
Final lower bound 8438.000000, upper bound 8438.000000
Exact lower bound: 8438.000000
DIFF: 0.000000
Final LP has 14 rows, 28 columns, 84 nonzeros
Optimal Solution: 8438.00
Number of bbnodes: 1
Total Running Time: 0.01 (seconds)


***  ***


*** You chose the Concorde(CPLEX) solver ***


*** Cities are numbered 0..n-1 and each line shows a leg from one city to the next
 followed by the distance rounded to integers***

10 10
0 8 685
8 5 601
5 9 1568
9 6 398
6 1 483
1 3 671
3 4 1326
4 2 1299
2 7 666
7 0 741

***PDF of optimal tour***unzip this file***
```

Additional Output:
13353806-HXgMetQP-solver-output.zip



Test #3

```
[who@mac tsp-genetics]$ dotnet run
0 1541 1748
1 3963 1805
2 1386 1925
3 196 411
4 1747 2085
5 1992 1147
6 2772 1340
7 2077 1981
8 2924 1632
9 1548 68
10 1713 102
11 3520 599
12 1413 389
13 2063 1344
14 196 329
15 663 1168
16 3818 958
17 2765 191
18 2341 428
19 2706 2166
GENERATION #1 - 25348
GENERATION #2 - 24981
GENERATION #3 - 24981
GENERATION #4 - 22607
GENERATION #5 - 22288
GENERATION #6 - 21640
GENERATION #7 - 20435
GENERATION #8 - 20435
GENERATION #9 - 19840
GENERATION #10 - 19268
GENERATION #11 - 18659
GENERATION #12 - 17507
GENERATION #13 - 17507
GENERATION #14 - 16600
GENERATION #15 - 16600
GENERATION #16 - 16600
GENERATION #17 - 14189
GENERATION #18 - 14189
GENERATION #19 - 14189
GENERATION #20 - 14189
GENERATION #21 - 13693
GENERATION #22 - 13693
GENERATION #23 - 13398
```

```
GENERATION #151 - 12099
GENERATION #152 - 12099
GENERATION #153 - 12099
GENERATION #154 - 12099
GENERATION #155 - 12099
GENERATION #156 - 12099
GENERATION #157 - 12099
GENERATION #158 - 12099
GENERATION #159 - 12099
GENERATION #160 - 12099
GENERATION #161 - 12099
GENERATION #162 - 12099
GENERATION #163 - 12099

Found solution - 12098,993540765441
8->19->7->4->2->0->13->5->15->3->14->12->9->10->18->17->11->16->1->6
```

```
class Config{
    3 references
    public static int POPULATION_COUNT = 120;
    1 reference
    public static int GENERATIONS_COUNT = 2000;
    2 references
    public static double MUTATION_CHANCE = 0.4;
    1 reference
    public static int NO_IMPROVEMENT_GEN = 300;
    1 reference
    public static int MAP_WIDTH = 4000;
    1 reference
    public static int MAP_HEIGHT = 2200;
}
```

```
************************************************************
/home/neos5/bin/concorde.cplex -s 99 -f -N 2 sample.dat2
Host: athene  Current process id: 45395
Using random seed 99
nnodes = 20
Set initial upperbound to 11970 (from tour)
  LP Value  1: 10995.500000  (0.00 seconds)
  LP Value  2: 11970.000000  (0.00 seconds)
New lower bound: 11970.000000
Final lower bound 11970.000000, upper bound 11970.000000
Exact lower bound: 11970.000000
DIFF: 0.000000
Final LP has 23 rows, 81 columns, 187 nonzeros
Optimal Solution: 11970.00
Number of bbnodes: 1
Total Running Time: 0.01 (seconds)


***  ***



*** You chose the Concorde(CPLEX) solver ***



*** Cities are numbered 0..n-1 and each line shows a leg from one city to the next
 followed by the distance rounded to integers***

20 20
0 4 395
4 7 346
7 13 637
13 5 209
5 6 804
6 8 329
8 19 577
19 1 1308
1 16 859
16 11 467
11 17 858
17 18 486
18 10 708
10 9 168
9 12 348
12 14 1218
14 3 82
3 15 889
15 2 1047
2 0 235


***PDF of optimal tour***unzip this file***
```

Additional Output:
[13353862-KSlJMyIY-solver-output.zip]

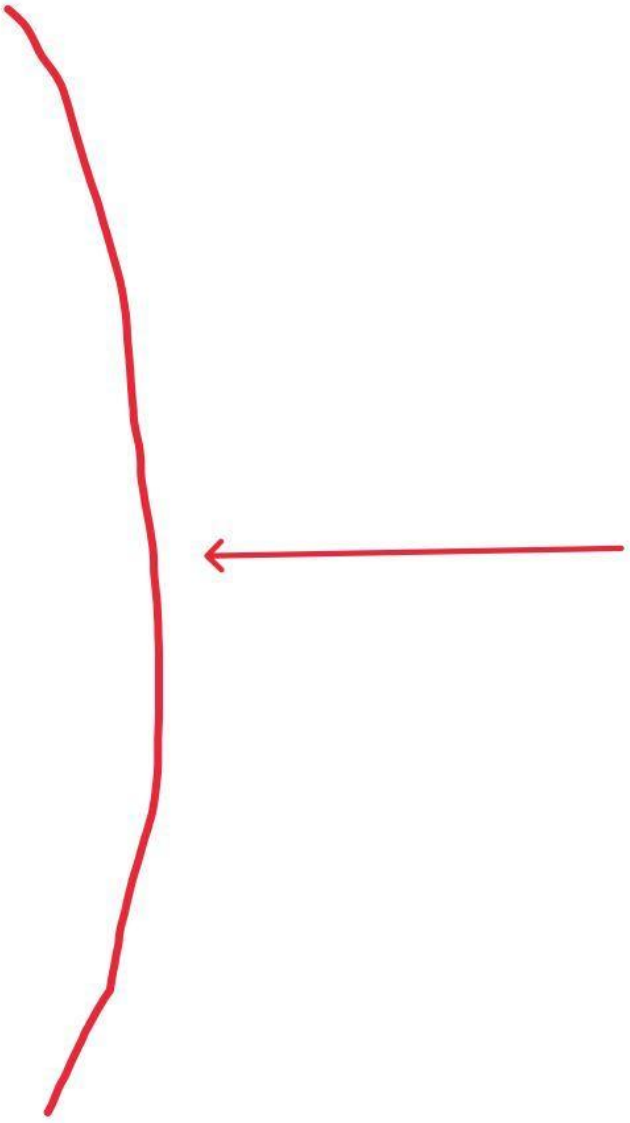
# Test #4

```
[who@mac tsp-genetics]$ dotnet run 40
/Users/who/programming/tsp-genetics/Entities.cs(4,11)
rride Object.GetHashCode() [/Users/who/programming/ts
2460 1825
672 2131
3316 308
1844 766
1971 971
2774 1094
1962 58
2307 2125
2584 999
685 2068
2568 1110
2652 1892
1720 1291
2561 327
548 1181
539 807
2495 379
2500 1660
2202 743
1227 2002
1193 1171
1415 1951
2689 317
202 7
3942 1272
3441 1050
1005 718
1063 1379
2595 828
2800 311
753 1309
535 1333
799 835
3232 1245
3662 1446
627 2109
2828 1571
1929 1344
866 1336
503 1023
GENERATION #1 - 47958
GENERATION #2 - 43299
GENERATION #3 - 43299
GENERATION #4 - 41692
GENERATION #5 - 40870
GENERATION #6 - 40654
GENERATION #7 - 39896
GENERATION #8 - 36414
GENERATION #9 - 36055
```

```
GENERATION #440 - 16092
GENERATION #441 - 16092
GENERATION #442 - 16092
GENERATION #443 - 16092
GENERATION #444 - 16092
GENERATION #445 - 16092
GENERATION #446 - 16092
GENERATION #447 - 16092
GENERATION #448 - 16092
GENERATION #449 - 16092
GENERATION #450 - 16092
GENERATION #451 - 16092
GENERATION #452 - 16092
GENERATION #453 - 16092
GENERATION #454 - 16092
GENERATION #455 - 16092
GENERATION #456 - 16092
GENERATION #457 - 16092
GENERATION #458 - 16092
GENERATION #459 - 16092
GENERATION #460 - 16092
GENERATION #461 - 16092
GENERATION #462 - 16092
GENERATION #463 - 16092

Found solution - 16091,78952655917
39->30->38->27->20->26->32->15->23->6->18->16->13->22->29->2->25->24->3
4->33->5->28->8->10->36->11->7->0->17->37->4->3->12->21->19->9->1->35->
31->14
```

```
class Config{
    3 references
    public static int POPULATION_COUNT = 120;
    1 reference
    public static int GENERATIONS_COUNT = 2000;
    2 references
    public static double MUTATION_CHANCE = 0.4;
    1 reference
    public static int NO_IMPROVEMENT_GEN = 300;
    1 reference
    public static int MAP_WIDTH = 4000;
    1 reference
    public static int MAP_HEIGHT = 2200;
}
```

```
New lower bound: 15037.000000
Final lower bound 15037.000000, upper bound 15037.000000
Exact lower bound: 15037.000000
DIFF: 0.000000
Final LP has 74 rows, 157 columns, 1013 nonzeros
Optimal Solution: 15037.00
Number of bbnodes: 1
Total Running Time: 0.04 (seconds)


***  ***


*** You chose the Concorde(CPLEX) solver ***


*** Cities are numbered 0..n-1 and each line shows a leg from one city to the next
 followed by the distance rounded to integers***

40 40
0 11 203
11 7 416
7 21 909
21 19 195
19 1 570
1 35 50
35 9 71
9 27 786
27 38 202
38 30 116
30 31 219
31 14 153
14 39 164
39 15 219
15 23 868
23 32 1021
32 26 237
26 20 490
20 12 540
12 37 216
37 4 375
4 3 241
3 18 359
18 6 726
6 16 622
16 13 84
13 22 128
22 29 111
29 2 516
2 25 752
25 24 548
24 34 330
34 33 475
33 5 482
5 28 321
28 8 171
8 10 112
10 36 529
36 17 340
17 0 170
```



Na testovacích případech můžeme vidět, jak genetický algoritmus dokáže dosáhnout správných řešení pro data v "různých dimenzích". Ačkoli, jak jsem se již zmínil,

genetické algoritmy NEBYLY vyvinuty pro hledání přesných řešení. Určité přesnosti můžeme dosáhnout v rámci ladění parametrů (konstanty v souboru Config.cs), abychom výsledek přiblížili skutečné hodnotě . Vložil jsem zde nejlepší hodnoty těchto konstant, které se mi podařilo najít.

# Testovací data

Pokud si chcete můj GA řešič otestovat sami, je zde adresář *testing*, který obsahuje 13 testovacích souborů. Byly vybrány tak, aby plně pokryly možné varianty dat. Soubory byly rozděleny do skupin podle velikosti testovacích dat:

test1 - test3 -> malé
test4 - test10 -> střední
test11 - test13 -> velké

Řešitel můžete spustit s kterýmkoli z těchto souborů pomocí příkazového řádku (v kořenovém adresáři projektu):

**dotnet run < testing/test#**

Pro kontrolu správnosti výsledků jsem uvedl také řešení z řešiče třetí strany, které jsem zmínil. Snímky obrazovky jsou umístěny v adresáři *testing/NEOS_testing/res#*.

**DOPORUČENÁ KONFIGURACE PRO SPUŠTĚNÍ TESTŮ (Config.cs):**

Test1 - Test10:

```
class Config{
    3 references
    public static int POPULATION_COUNT = 100;
    1 reference
    public static int GENERATIONS_COUNT = 500;
    2 references
    public static double MUTATION_CHANCE = 0.1;
    1 reference
    public static int NO_IMPROVEMENT_GEN = 40;
    1 reference
    public static int MAP_WIDTH = 4000;
    1 reference
    public static int MAP_HEIGHT = 2200;
}
```

Test11 - Test13:

```
class Config{
    3 references
    public static int POPULATION_COUNT = 150;
    1 reference
    public static int GENERATIONS_COUNT = 500;
    2 references
    public static double MUTATION_CHANCE = 0.4;
    1 reference
    public static int NO_IMPROVEMENT_GEN = 100;
    1 reference
    public static int MAP_WIDTH = 4000;
    1 reference
    public static int MAP_HEIGHT = 2200;
}
```

# Struktura kodu

- Program.cs
  - class Program - je hlavní třída projektu. Obsahuje hlavní genetický algoritmus a pomocné funkce pro něj.
    - **current_generation -** aktuální generace pro zpracování
    - **rnd -** generátor náhodných čísel
    - **gen_counter -** počet zpracovaných generací
    - **no_impr_counter -** počítadlo zpracovaných generací bez zlepšení hodnoty *best_fitness*
    - **previous_best -** dříve nejlepší dosažená hodnota *best_fitness*
    - **generate_chromosome()** - generace chromozomů v generaci
    - **spawn_generation()** - generování POPULATION_COUNT chromozomů pro počáteční generaci
    - **get_best_chromosome(** generation **)** - najít chromozom s nejlepším fitness skóre v průběžném generation
    - **wheel_selection(** generation **)** - funkce implementace Roulette Wheel Selection pro výběr "rodičů" k reprodukci v průběžném generation
    - **crossover(** chrom_a, chrom_b **)** - křížení genů chromozomů chrom_a a chrom_b za účelem získání syna
    - **swap_mutation(** original **)** - použití Swap Mutace na chromozomu original
    - **rotate_mutation(** original **)** - použití Inverzní Mutace na chromozomu original
    - **mutate(** chrom **)** - výběr typu mutace (na chromozomu chrom) na základě náhodného generování
    - **breed()** - generování nové generace na základě probíhajícího
    - **Main()** - funkce spuštění algoritmu, zpracování parametrů z příkazového řádku a zobrazení výsledků

- Config.cs
  - class Config - třída, která se skládá ze sady projektových konstant.
    - **POPULATION_COUNT** - počet chromozomů v jedné generaci
    - **GENERATIONS_COUNT** - maximální počet generací, které algoritmus zpracuje při hledání řešení
    - **MUTATION_CHANCE** - pravděpodobnost mutace pro chromozom
    - **NO_IMPROVEMENT_GEN** - maximální počet generací beze změny nejlepší hodnoty fitness, které algoritmus zpracuje před vynuceným zastavením

        - **MAP_WIDTH / MAP_HEIGHT** - velikosti mapy pro *testovací režim*
- Entities.cs
    - class Cromosome - datová struktura pro popis objektu chromozomu
        - **perm** - List zobrazení cesty řešení (permutace měst)
        - **get_fitness()** - výpočet hodnoty fitness chromozomu (délka cesty řešení)
        - **OVERRIDE ToString() -** překlad řešení do podoby vhodné pro zobrazení
        - **OVERRIDE Equals(** obj **)** - mechanismus pro porovnávání s chromozomem obj z hlediska identity
    - class Map - datová struktura pro reprezentaci mapy měst úlohy
        - **cities** - slovníková kolekce párových hodnot název města <-> souřadnice města
        - **add_city(** name, x, y**)** - přidat město z nazvem name a se souřadnicemi (x, y) do slovníku *cities*
        - **get_coords(** city_index **)** - získat souřadnice města podle jeho city_index
        - **cities_count** - počet přidaných měst ve struktuře Map
        - **city_name(** index **) -** získat název města podle jeho idnex

- Tools.cs
    - class Helper - třída s pomocnými funkcemi pro zpracování dat
        - **get_dist(** from, to **)** - výpočet vzdálenosti (L2) mezi body from a to
        - **shuffle(** list **)** - zamíchat list "náhodně"
        - **exclude(** list, target **)** - vyloučit target z list a vrátit zbytek
        - **random_indices(** list, n **)** - vybrat n náhodných indexů v list
        - **decode_path(** seq **)** - převést posloupnost indexů seq měst na posloupnost názvů měst
        - **input_generator(** city_count **)** - "náhodně" vygenerovat sadu city_count měst pro testovací režim
        - **generate_dist_matrix()** - generování (matice vzdáleností)* pro kontrolu správnosti řešení

*(MATICE VZDÁLENOSTÍ)** - zobrazuje vzdálenost mezi jednotlivými dvojicemi měst