ДОМАШНА РАБОТА №1

Използван метод за решаване на задачата.

Избраният метод за решаване на задачата е чрез генетичен алгоритъм. Доста лесно се представят индивидите от популациите, като низ от символи – реда в който търговския пътник минава през градовете. А пък фитнес функцията може директно да бъде взета от дадения граф на пътищата между всички градове. Тъй като между всеки два града има път, то и създаването на нови индивиди ще бъде сравнително безпроблемно.

Описание на реализацията, чрез псевдокод.

```
GeneticAlgorithm(population_size, fitness_fn) → individual:
  where individual is a an array of 6 characters
        population_size is the number of individuals in each population
        fitness_fn is a function that measures the fitness of an individual

  population ← a set of randomly generated individuals (paths)
  bests      ← a queue of size 6 filled with 0's
  avg        ← 0.0
  last_avg   ← 1.0
  while the fittest individuals of the last 10 generations have an identical
        average fitness score (avg ≠ last_avg)
  do
    last_avg ← avg
    new_pop  ← an empty set

    repeat size of population times
      par_i  ← randomly selected individual from population
      par_j  ← randomly selected individual from population
      (child1, child2) ← Reproduce(par_i, par_j)
      if 5% probability then child1 ← Mutate(child1)
      if 5% probability then child2 ← Mutate(child2)
      add child1 to new_pop
      add child2 to new_pop
    end

    population ← new_pop
    fittest    ← fittest indivudual from population according to fitness_fn
    remove first from bests
    add fittest to bests
    avg        ← average of bests
  end
  return the fittest individual in population, according to fitness_fn

Reproduce(parent1, parent2) → a pair of individuals:
  pos    ← a random integer from 2 to 6
  child1 ← Append(Substring(parent1, 1, pos-1), Substring(parent2, pos, 6))
  child2 ← Append(Substring(parent2, 1, pos-1), Substring(parent1, pos, 6))
  child1 ← MakeUnique(child1)
  child2 ← MakeUnique(child2)
  return (child1, child2)

Mutate(individual) → a mutated indivudual:
  pos_i ← a random integer from 1 to 6
  pos_j ← a random integer from 1 to 6
  Swap(individual[pos_i], individual[pos_j])
  return individual

MakeUnique(individual) → a valid individual:
  for each character in individual
    if character occurs more than once in individual then
      replace character with a character from [A, B, C, D, E, F]
      that doesn't occur in individual
  return individual
```

Инструкции за компилиране на програмата.

Програмата е написана на езика Rust. Реализацията се намира във файла *main.rs* в папката *tsp-ai/src*. Можете да инсталирате компилатор за Rust код от тук.
Компилирането и стартирането на програмата става, чрез *cargo* по следния начин:

1. Отворете конзола (терминал) в папката *tsp-ai*
2. Изпълнете `cargo run`

Изпълнението на Rust код също може да стане и онлайн тук.

Резултати от изпълнението на програмата.

Изпълняването на генетичния алгоритъм с аргументи фитнес функция връщаща времето за което търговския пътник изминава пътя зададен от индивида. И с размер на популацията:

- 5
  Изход:
  ```
  Genetic algorithm stopped after 12 generations.
  [229, 229, 229, 244, 244, 247, 247, 247, 247, 251]
  ['E', 'D', 'C', 'B', 'A', 'F'] with road length: 229
  ```

- 10
  Изход:
  ```
  Genetic algorithm stopped after 9 generations.
  [229, 229, 229, 235, 235, 235, 244, 244, 244, 244]
  ['A', 'B', 'C', 'D', 'E', 'F'] with road length: 229
  ```

- 100
  Изход:
  ```
  Genetic algorithm stopped after 7 generations.
  [229, 229, 229, 229, 229, 235, 235, 235, 235, 235]
  ['A', 'B', 'C', 'D', 'E', 'F'] with road length: 229
  ```

- 1000
  Изход:
  ```
  Genetic algorithm stopped after 7 generations.
  [229, 229, 229, 229, 229, 235, 235, 235, 235, 235]
  ['A', 'B', 'C', 'D', 'E', 'F'] with road length: 229
  ```

Разглеждайки изходните данни за популация с размер 5 забелязваме, че алгоритъма приключва след 12 генерации откривайки път EDCBAF с дължина 229. Също така забелязваме, че има и още 2 индивида от тази популация с такава дължина на пътя. Тъй като търсим Хамилтонов цикъл, очевидно индивидът EDCBAF удовлетворява изискванията.
Ако увеличим размера на популацията до 10, то алгоритъма приключва още на 9-тата генерация със същата дължина на пътя, но с друг път. Можем да забележим и, че индивидите в тази популация са със сравнително по-нисък резултат от фитнес функцията. Тази тенденция продължава и за размер на популацията 100.
За по-големи популации, не забелязваме подобрение на бързодействието или пък на индивидите, тъй като входните данни на задачата са с много малък размер.