KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

# P170B400 Algoritmų sudarymas ir analizė

(P170B400)

Laboratorinių darbų ataskaita

Atliko:

IFF-1/9 gr. studentas

**Nedas Liaudanskis**

2023 m. gegužės 24 d.

Priėmė:

**Doc. Pilkauskas Vytautas**

**Lekt. Kraujalis Tadas**

**Doc. Čalnerytė Dalia**

**Lekt. Makackas Dalius**

KAUNAS 2023

# UŽDUOTYS

Faile places\_data.xlsx pateikta informacija apie miestus (įskaitant jų gerumo įvertį) ir kelius tarp jų (kelionės laikas ir kaina)  (2, 3 lentelė).

Tikslas: kaip galima geresnio (vertinama pagal aplankytų miestų gerumo įverčius) maršruto sudarymas kai:

* kelionės pradžios ir pabaigos vieta sutampa (su grįžimu atgal);
* bendras kelionės laikas negali viršyti 48 val.;
* tą patį miestą galima aplankyti kelis kartus, tačiau jo gerumo įvertis sumuojamas tik vieną kartą.

## 1 Dalis:

(~3 balai). Realizuoti programą, kuri individualiai problemai pateiktų optimalų sprendinį. Nustatyti, prie kokios duomenų apimties sprendinį pavyksta rasti jei programos vykdymo laikas negali būti ilgesnis nei 10 sek.

## 2 Dalis:

(~2 balai). Realizuoti programą, kuri individualios problemos rezultatą pateiktų priimant „lokaliai  geriausią“ sprendinį (pvz. esant keliems pasirinkimams keliauti į skirtingas viršūnes, visuomet renkamasi: pigiausias ar trumpiausias ar  ... kelias)

## 3 Dalis:

(~5 balai). Realizuoti programą, kuri pateiktų sprendinį taikant Genetinio Optimizavimo metodą. Programa pateikti rezultatą turi ne ilgiau, nei per 60 sek.

**Bendra:**

* Darbo ataskaitoje turi būti pateikta:
  + apskaičiuotas asimptotinis programos vykdymo laiko sudėtingumas (esant rekursiniams metodams, turi būti suformuojama rekurentinė lygtis);
  + realizuotų programų (sudaryto algoritmo konkrečiam uždaviniui) abstraktus aprašas („pseudo“ kodas ar „workflow“ diagrama);
  + skirtingų metodų rezultatų analizė uždavinio gerumo ir vykdymo laiko prasme esant kelioms skirtingoms pradinėms sąlygoms.
  + pateikti rekomenduojamus genetinio optimizavimo parametrus užduoties sprendimui (populiacijos dydis, elito kiekis, ...). Pateikti analizės rezultatus, kaip buvo parinkti rekomendaciniai parametrai, t. y. analizės rezultatuose turi pasimatyti, jog per maksimalų programos vykdymo laiką, tikėtina, jog tikslo funkcija įgaus didžiausią reikšmę.
  + marštutai, sudaryti analizuojant atksirus atvejus atvaizduojami grafiškai.
* Kelionės pradžią turi būti galima nurodyti bet kurią vietą ar miestą (priklausomai nuo individualios sąlygos) iš pateiktų duomenų.
* Programavimo kalba: bet kuri, išskyrus Python.

# Duomenų aprašymas

Norint išspręsti ši uždavinį reikia perskaityti duomenis iš Excel failo(places\_data.xlsx);

|  |  |  |  |
| --- | --- | --- | --- |
| **2 Lentelė** |  |  |  |
| **Miesto pav. (unikalus)** | **Gerumo įvertis** | **X** | **Y** |
| Alicante | 0,60238837 | -54342,92488 | 4628084,51 |
| Amsterdam | 0,726813257 | 544753,5166 | 6867807,879 |
| Ancona | 0,598182262 | 1504716,489 | 5406364,202 |
| Antalya | 0,598324817 | 3417005,715 | 4425311,436 |
| Athens | 0,732722808 | 2641422,852 | 4577157,491 |
| Barcelona | 0,743564997 | 239625,0407 | 5070618,459 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Miestas (iš)** | **Miestas (į)** | **Laikas (s)** | **Kaina** |
| Alicante | Amsterdam | 10500 | 28,99 |
| Alicante | Bari | 8400 | 133,99 |
| Alicante | Belfast | 11400 | 99,99 |
| Alicante | Birmingham | 9900 | 58,99 |
| Alicante | Bordeaux | 5700 | 70,99 |
| Alicante | Bremen | 10800 | 314,21 |
| Alicante | Bristol | 9000 | 130,99 |
| Alicante | Brussels | 9000 | 54,74 |
| Alicante | Bucharest | 12780 | 129,99 |
| Alicante | Budapest | 10920 | 64,99 |
|  |  |  |  |

Parodytos lentelės yra tik pavyzdys, duomenų faile duomenų yra daug daugiau.

Savo programoje aš duomenis skirstau į 2 tipus, Miestus ir Kelius. O atsakymo ieškau Kelio, tarp miestų.

**City** klase, kuri naudojama saugoti miestų duomenys:

|  |
| --- |
| public class City  {  public string Name { get; set; }  public double Rating { get; set; }  public double X { get; set; }  public double Y { get; set; }  public List<Road> Roads { get; set; }  public City(string name, double rating, double x, double y)  {  Name = name;  Rating = rating;  X = x;  Y = y;  Roads = new List<Road>();  }  public City()  {  Name = "";  Rating = 0;  X = 0;  Y = 0;  Roads = new List<Road>();  }  public void AddRoad(Road r)  {  this.Roads.Add((Road)r);  }  } |

**Road** klase, kuri naudojama saugoti kelių duomenys:

|  |
| --- |
| public class Road  {  public City From { get; set; }  public City To { get; set; }  public int Time { get; set; }  public double Cost { get; set; }  public Road(City from, City to, int time, double cost)  {  From = from;  To = to;  Time = time;  Cost = cost;  }  } |

**Path** klase, kuri naudojama saugoti kiekvieno maršruto duomenys(Naudojama atsakymuose):

|  |
| --- |
| public class Path  {  public double Rating { get; set; }  public int Time { get; set; }  public List<City> Roads { get; set; }  public Path()  {  this.Roads = new List<City>();  this.Rating = 0;  this.Time = 0;  }  public Path(Path p)  {  this.Roads = new List<City>(p.Roads);  this.Rating = p.Rating;  this.Time = p.Time;  }  public bool Contains(City c)  {  for (int i = 0; i < Roads.Count(); i++)  {  if(c.Name == Roads[i].Name)  {  return true;  }  }  return false;  }  public void Add(double rating, int time, City city)  {  this.Rating = rating;  this.Time = time;  this.Roads.Add(city);  }  public bool Repeat(City current, City next)  {  for (int i = 0; i < Roads.Count(); i++)  {  if (current.Name == Roads[i].Name)  {  if (i + 1 < Roads.Count() && next.Name == Roads[i + 1].Name)  {  return false;  }  }  }  return true;  }  } |

# Pirma Dalis.

Šis uždavinys yra, keliaujančio pirklio uždavinys, su keliomis papildomais sąlygomis ir apribojimais, kurie šį uždavinį pasunkina. Šio uždavinio tipas yra NP (reiškia nedeterministinį daugianario laiką ), tai problemų, kurių sprendimus galima patikrinti daugianario laiku, rinkinys. Tokias problemas norint išspręsti, dažniausiai užtrunka eksponentinį laiką.

Uždavinys susideda iš n miestų. Iš kiekvieno miesto galima patekti į kitą miestą, per tam tikrą laiką (kelią), kurį pavadinkime m. Tai reiškiasi iš kiekvieno miesto galima patekti į kitą miestą n, tai sudaro labai didelius begalinius kelius. Tačiau mes ieškome geriausio įverčio ir keliaujant į tą patį miestą kelis kartus, nėra naudinga nes įvertis neprisideda. Todėl optimaliausiam sprendiniui nenorime daryti daug grįžimų, nes tada yra švaistomas laikas nereikalingoms kelionėms, nes turime tik 48H. Norint surasti geriausius kelius, maršrutus, turinčius didžiausią reitingą, mums reikia rintis tokius kelius, kurie yra trumpiausi. Taip galėsime, užtikrinti didesnį kelionių skaičių, kuris mums duos didesnį įvertį.

Paveikslėlis, kuriame yra eskizas, piešimas, apskritimas

Automatiškai sugeneruotas aprašymas

pav. Miestai

Pagrindinė uždavinio formulė, yra paprasta, nes mes ieškome tiesiog viso kelio, sudarytos įverčio sumos. Kadangi kelias nebus didesni nei n, tai **mūsų medžio gylis bus n-1**.

**Kodui einant 10 sec, vidutiniškai randama tinkamų kelių: 240**

## Programinio kodo analizė

|  |
| --- |
| void FindAllPathsForced(City StartingCity, List<Path> list, int Timerr, double Ratings, City current, Path path, Stopwatch timer)  {  if(timer.Elapsed.TotalSeconds > 10) //c1 | 1  {  return; //c2 | 1  }  Path newPath = new Path(path); //c3 | 1  int Time = Timerr; //c4 | 1  double Rating = Ratings; //c5 | 1  if(Timerr/3600 > 48)  {  return; //c2 | 1  }  newPath.Add(Rating, Time, current); //c6 | 1      /\*foreach(City c in path.Roads)  {  Console.Write(c.Name + "->");  }\*/  //Console.Write("\n");  if (current.Name == StartingCity.Name && Time != 0) //c7 | 1  {  list.Add(newPath); //c8 | 1  return; //c2 | 1  }  for (int i = 0; i < current.Roads.Count(); i++) //c9 | n +1  {  //Console.WriteLine(newPath.Repeat(current, current.Roads[i].To));  if(newPath.Repeat(current, current.Roads[i].To)) //c10 | n  {  if(!newPath.Contains(current.Roads[i].To)) //c11 | n  {  FindAllPathsForced(StartingCity, list, Time + current.Roads[i].Time, Rating, current.Roads[i].To, newPath, timer); //T(n - 1) | n  }  else  {  FindAllPathsForced(StartingCity, list, Time + current.Roads[i].Time, Rating + current.Roads[i].To.Rating, current.Roads[i].To, newPath, timer); //T(n - 1) | n  }    }    }    } |

## Asimptotinio sudėtingumo apskaičiavimas

Šios programos sudėtingumas priklauso nuo daugelio faktorių: Miestų, skaičius, Miesto kelių skaičiaus, Kelionės laiko.

T(n) = c1 + 3\*c2 + c3 + c4 + c5 + c6 + c7 + c8 + c9\*n + c9 + c10 \*n + c11 \*n + T(n-1)\*n

T(n) = c + T(n-1)\*n;

T(n-1) – tai suma nuo n, n-1 ..... 0

**Sudarome medį:**

Paveikslėlis, kuriame yra diagrama, eskizas, piešimas, linija

Automatiškai sugeneruotas aprašymas

Iš gauto medžio lengva pastebėti, jog gauname, **T(n) = n!**, sudėtingumą.

Taigi, šio metodo sudėtingumas yra **O(T(n)) = n!.**

Turėdami 10 sekundžių apribojimą programos veikimui ir žinodami, jog kodo sudėtingumas yra faktorialinis, galime nujausti, jog didelių kiekių duomenų programa nesugebės apdoroti per numatytą laiką.

## Eksperimentinis tyrimas, rezultatai

Rezultatai, rodoma 10 miestų su geriausiu įverčiu.

Kadangi duomenų kiekis labai didelis, duomenų pereiti nepavyksta todėl, atsakymai yra labai panašūs.

Rezultatai:

|  |
| --- |
| 452  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Amsterdam -> Belfast -> London -> Paris -> Prague -> Kosice -> 171600 8,347759109968527  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Amsterdam -> Belfast -> London -> Copenhagen -> Prague -> Kosice -> 172200 8,235952776721673  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Alicante -> Geneva -> Paris -> Prague -> Kosice -> 172500 8,207133146837359  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Alicante -> Marseille -> Paris -> Prague -> Kosice -> 172200 8,198353242100461  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Amsterdam -> Belfast -> Edinburgh -> Paris -> Prague -> Kosice -> 172500 8,193680605408321  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Amsterdam -> Belfast -> Edinburgh -> London -> Prague -> Kosice -> 171000 8,192753195473184  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Amsterdam -> Belfast -> Liverpool -> Paris -> Prague -> Kosice -> 170400 8,187727362630374  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Alicante -> Toulouse -> Paris -> Prague -> Kosice -> 170400 8,185190575194168  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Alicante -> Milan -> Budapest -> Prague -> Kosice -> 171900 8,165175219667638  Kosice -> Dublin -> Alicante -> Amsterdam -> Alicante -> Bari -> Alicante -> Belfast -> Alicante -> Birmingham -> Alicante -> Bordeaux -> Alicante -> Bremen -> Alicante -> Bristol -> Alicante -> Marseille -> Rome -> Prague -> Kosice -> 172200 8,156253788610547 |

# Antra Dalis.

## Užduotis

(~2 balai). Realizuoti programą, kuri individualios problemos rezultatą pateiktų priimant „lokaliai  geriausią“ sprendinį (pvz. esant keliems pasirinkimams keliauti į skirtingas viršūnes, visuomet renkamasi: pigiausias ar trumpiausias ar  ... kelias)

Norint pateikti lokaliai geriausią, sprendinį, turime suoptimizuoti jau turimą kodą iš pirmos dalies. Kadangi Miestų reitingų sumuoti negalime jeigu aplankome tą patį miestą, tai aplankyti jau aplankytus miestus yra labai neoptimalu. Todėl sukuriame Dictionary objektą, kuris laikys duomenis apie miestų apsilankymus. True – miestas aplankytas, False – miestas neaplankytas. Tokiu būdu, negausime pasikartojančių, miestų, kurie labai sumažina mūsų įverčio sumą.

Kadangi įverčiai, tar kiekvieno miesto nelabai skiriasi, reikia atsižvelgti tik į laiką, nes kelionės laikas, nuo vieno miesto į kitą yra labai skirtingas. Todėl keliaujant per miestus, pasirinksime kitą miestą pagal trumpiausią atstumą iki jo. Tai leis, mums apeiti didesnį kiekį miestu ir surinkti didesnį kelionės įverčio sumą.

Kai renkame, trumpiausią kelia, ne visada garantuojame, jog grįšime į pradinį miestą, kai praėjo pusė kelionės laiko, pradedame ieškoti kelio iki pradinio miesto, jeigu toks egzistuoja, keliaujame į jį ir užrašome tai, kaip vieną kelią.

Paveikslėlis, kuriame yra apskritimas, diagrama, linija

Automatiškai sugeneruotas aprašymas

pav. Geriausio kelio radimas

## Programinio kodo analizė

|  |
| --- |
| void FindAllPaths(City StartingCity, List<Path> list, int Timerr, double Ratings, City current, Dictionary<City, bool> visited, Path path)  {  Dictionary<City, bool> dic = new Dictionary<City, bool>(visited); // c1 | 1  Path newPath = new Path(path); // c2 | 1  int Time = Timerr; // c3 | 1  double Rating = Ratings; // c4 | 1  void FindAllPaths(City StartingCity, List<Path> list, int Timerr, double Ratings, City current, Dictionary<City, bool> visited, Path path)  {  Dictionary<City, bool> dic = new Dictionary<City, bool>(visited); // c1 | 1  Path newPath = new Path(path); // c2 | 1  int Time = Timerr; // c3 | 1  double Rating = Ratings; // c4 | 1  for (int r = 0; r < current.Roads.Count(); r++) // c5 | n +1  {  if (dic[current] == false) // c6 | n  {  dic[current] = true; //c7 | n  }  else  {  if (current.Name != StartingCity.Name) //c8 | n  {  return; //c9 | n  }  if (Time / 3600 > 48) //c10 | n  {  return; //c11 | n  }  }    int Timer = 1000000; // c12 | n  City next = current; // c13 | n  //Console.WriteLine(Time);  if (StartingCity.Name == current.Name) // c14 | n  {  if (r == 0) // c15 | n  {  newPath.Add(Rating, Time, current); // c16 | n  }  }  else  {  newPath.Add(Rating, Time, current); // c17 | n  }  if (Time / 3600 > 20) // c18 | n  {  if (current.Name == StartingCity.Name && Time != 0) // c19 | n  {  list.Add(newPath); // c20 | n  return; // c21 | n  }  //Console.WriteLine(current.Roads.Count());  for (int i = 0; i < current.Roads.Count(); i++) // c22 | n\*(n+1)  {  if (StartingCity.Name == current.Roads[i].To.Name) //c23 | n\*n  {  FindAllPaths(StartingCity, list, Time + current.Roads[i].Time, Rating + current.Roads[i].To.Rating, current.Roads[i].To, dic, newPath); //T(n-1) \* n \* n  }  if (Timer > current.Roads[i].Time && (dic[current.Roads[i].To] == false || current.Roads[i].To.Name == StartingCity.Name)) //c24 | n\*n  {  Timer = current.Roads[i].Time; //c25 | n\*n  next = current.Roads[i].To; //c26 | n\*n  }  }  //Console.WriteLine(current.Name + " -> " + next.Name);  FindAllPaths(StartingCity, list, Time + Timer, Rating + next.Rating, next, dic, newPath); //T(n-1)\*n  dic[next] = true; //c28 | n  }  else  {  if (current.Name == StartingCity.Name && Time != 0) //c29 | n  {  list.Add(newPath); //c30 | n  return; //c31 | n  }  //Console.WriteLine(current.Roads.Count());  for (int i = 0; i < current.Roads.Count(); i++) //c32 | n\*(n + 1)  {  if (Timer > current.Roads[i].Time && (dic[current.Roads[i].To] == false || current.Roads[i].To.Name == StartingCity.Name)) //c33 | n  {  Timer = current.Roads[i].Time; //c34 | n\*n  next = current.Roads[i].To; //c35 | n\*n  }  }  //Console.WriteLine(current.Name + " -> " + next.Name);  FindAllPaths(StartingCity, list, Time + Timer, Rating + next.Rating, next, dic, newPath); //T(n-1)\*n  dic[next] = true; //c37 | n  }  }  } |

## Asimptotinio sudėtingumo radimas

Žiūrint į metodo kodą, gauname tokią lygtį.

Kai miestas neturi kelių: c1 + c2 + c3 + c4

Sudėtingumas, kai einantis miestas viršija, laiko limitą arba yra pradinis miestas. Tada programa baigia darbą, ir grįžta.

C1 + c2 + c3 + c4+ c5\*n + c6 + (c8\*n + c9\*n) | (c10\*n + c11\*n)

Kai praeina pirmąjį patikrinimą, programa, patikrina dar kartą, jog kelionė išnaudojo bent pusė, suteikto laiko. Ir tada patikrina, ar dabartinis miestas, kuriame esame nėra pradinis, miestas. Jeigu yra reikia šią kelionę įrašyti į kelionių masyvą ir nutraukti darbą(grįžti į ankstesnę iteracija).

c1 + c2 + c4 + c3 + c5 + n\*(c5 + c6 + c7 + c8 + c9 + c10 + c11 + c12+ c13 + c14 + c15 + c16 + c17 + c18 +c19 + c20+ c21)

Kai praeina pirmąjį patikrinimą, programa, patikrina dar kartą, jog kelionė neišnaudojo pusė, suteikto laiko. įrašyti į kelionių masyvą ir nutraukti darbą(grįžti į ankstesnę iteracija).

c1 + c2 + c4 + c3 + c5 + n\*(c5 + c6 + c7 + c8 + c9 + c10 + c11 + c12+ c13 + c14 + c15 + c16 + c17 + c18 +c29 + c30+ c31)

Šios lygtys parodo, visos programos kelio, sudėtingumą išanalizavus kodą. Šis kelias gaunasi, kai ieškome sekančio miesto, su trumpiausiu keliu iki jo ir dar tuo pačiu tikriname ar galime rasti pradinį miestą.

c1 + c2 + c4 + c3 + c5 + n\*(c5 + c6 + c7 + c8 + c9 + c10 + c11 + c12+ c13 + c14 + c15 + c16 + c17 + c18 + c22\*n + c23\*n + c24\*n +c25\*n+ c26\*n +c27 + c28 + n\*T(n-1) + T(n-1));

Šios lygtys parodo, visos programos kelio, sudėtingumą išanalizavus kodą. Šis kelias gaunasi, kai ieškome sekančio miesto, su trumpiausiu keliu iki jo. Kai tas miestas randama, programa eina toliau pro sekančias iteracijas.

c1 + c2 + c4 + c3 + c5 +n\*( c5 + c6\*n + c7\*n + c8\*n + c9\*n + c10\*n + c11\*n + c12+ c13 + c14 + c15 + c16 + c17 + c18 + c32\*n + c33\*n + c34\*n + c35\*n + c37 + T(n-1))

Iš viršuje pateikto kodo, galime rasti trumpiausią ir ilgiausią, kelio formule. Kai kelias, neturi jokių, kitų kelių, galime teigti jog sudėtingumas **geriausiu atveju** tada yra. **Ω(T(n)) = c.**

Norint rasti blogiausią, atveja reikia išspręsti sutrumpintą, lygtį, kai ieškome kelio atgal praėjus, puse kelionės laiko. T(n) = n\*nT(n-1) + T(n-1)\*n + n\*n + n + c;

Sutrumpinus lygtį gauname: T(n) = n^2\*T(n-1);

Perėjus keletą iteracijų, galime pamatyti, jog

T(n) = n^2 \* T(n-1)

= n^2 \* (n-1)^2 \* T(n-2)

= n^2 \* (n-1)^2 \* (n-2)^2 \* T(n-3)

= ...

= n^2 \* (n-1)^2 \* (n-2)^2 \* ... \* 2^2 \* T(1) = n^2!\*T(1);

Tai blogiausiu atveju **O(T(n)) = n^2!;**

## Eksperimentinis tyrimas

Rezultatai, rodoma 10 miestų su geriausiu įverčiu.

Rezultatai:

|  |
| --- |
| 6489  Barcelona -> Malaga -> Rabat -> Sevilla -> Lisbon -> Madrid -> Porto -> Valencia -> Toulouse -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Tours -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Barcelona -> 172500 26,760099982050832  Copenhagen -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Zagreb -> Bratislava -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Copenhagen -> 170400 26,686288696758993  Sofia -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> Venice -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Brussels -> Bordeaux -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> 171600 26,674282432254923  Gothenborg -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Zagreb -> Bratislava -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Gothenborg -> 171000 26,643879251844805  Copenhagen -> Kaunas -> Warsaw -> Szczecin -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Zagreb -> Bratislava -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Amsterdam -> Copenhagen -> 167700 26,59554633453375  Marseille -> Rome -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Pisa -> Barcelona -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> Venice -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Brussels -> Turin -> Valencia -> Marseille -> 171900 26,049924507706272  Catania -> Cagliari -> Palermo -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> Venice -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Brussels -> Bordeaux -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Catania -> 171000 26,047742001615934  Hamburg -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Nantes -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Hamburg -> 172740 26,034200909023657  Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Toulouse -> Lyon -> Bordeaux -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Venice -> Nuernberg -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Paris -> 168600 26,00555984735378  Sofia -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> Venice -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Brussels -> Bordeaux -> Marseille -> Milan -> Zagreb -> Sofia -> 167100 26,00156046144172 |

## 

# Trečia Dalis:

## Užduotis

(~5 balai). Realizuoti programą, kuri pateiktų sprendinį taikant Genetinio Optimizavimo metodą. Programa pateikti rezultatą turi ne ilgiau, nei per 60 sek.

Genetinis optimizavimas yra evoliucinės algoritmo klasė, naudojama spręsti optimizavimo problemas. Šis metodas yra paremtas biologinės evoliucijos principais ir daro prielaidą, kad galima rasti geriausią sprendimą, atliekant iteratyvų sprendimų paieškos procesą.

Genetinio optimizavimo metu kuriamas populiacija, kurią sudaro individualūs sprendimų kandidatai(Paths), vadinami chromosomomis arba genotipais. Kiekviena chromosoma koduoja vieną potencialų sprendimą. Sprendimų kokybė vertinama pagal objektyvią funkciją, vadinamą tinkamumo funkcija (fitness function).

Procesas prasideda nuo pradinės populiacijos sukūrimo, kuri dažnai yra atsitiktinė. Tada kartojasi trys pagrindiniai etapai: selekcija, rekombinacija ir mutacija.

Selekcijos etape parenkami geriausieji individualai, kurie turi didžiausią tinkamumo funkcijos reikšmę. Jie tampa tėvais, kurie dalyvauja rekombinacijoje.

Rekombinacijos etape atsitiktinai parenkami du tėviniai individualai, iš kurių sukuriama nauja karta palikuonių. Tai gali būti padaryta perkrėtimu (crossover) ar kitais metodais, priklausomai nuo konkrečios genetinio algoritmo implementacijos.

Mutacija yra mažų atsitiktinių pakeitimų įvedimas į palikuonius. Tai padeda išvengti atrėžimo lokaliam maksimumui ir padidinti sprendimų erdvės tyrinėjimo galimybes.

Kartojant selekcijos, rekombinacijos ir mutacijos žingsnius per kelias kartas, galiausiai gali būti rastas arba artimas geriausias sprendimas, pagal nustatytus optimizavimo tikslus.

Genetinis optimizavimas plačiai naudojamas sprendžiant įvairias optimizavimo problemas, tokius kaip parametrų derinimas, inžinerinės konstrukcijos, planavimo uždaviniai ir kt. Jis yra efektyvus tuo atveju, kai sprendimų erdvė yra didelė ir sunku rasti optimalų sprendimą naudojant tradicinius metodus.

## Programos kodas

|  |
| --- |
| public class GeneticOptimization  {  private const int Elitism = 5;  private const double Mutation = 1.5;  private const int TournamentSize = 3;  private static Path SelectViaTournament(Population Population, List<Road> roads)  {  Population Tournament = new Population(TournamentSize);  Random Randomizer = new Random();  for (int i = 0; i < TournamentSize; i++)  Tournament.PathAdd(Population.Paths[Randomizer.Next(0, Population.Paths.Count)]);  return Tournament.GetFittest();  }  private static Path CrossoverAndMutate(Path first, Path second, List<Road> roads)  {  Random randomizer = new Random();  // Create a new path object for the offspring  Path offspring = new Path();  // Randomly select a crossover point  int crossoverPoint = randomizer.Next(1, first.Roads.Count - 1);  // Add cities from the first parent up to the crossover point  for (int i = 0; i < crossoverPoint; i++)  {  City city = first.Roads[i];  offspring.Roads.Add(city);  offspring.Rating += city.Rating;    }  //offspring.Time = Utils.Calculatetime(offspring.Roads, roads);  // Add cities from the second parent after the crossover point  for (int i = crossoverPoint; i < second.Roads.Count - 1; i++)  {  City city = second.Roads[i];  // Check if the city is already present in the offspring  if (!offspring.Roads.Contains(city))  {  offspring.Roads.Add(city);  offspring.Rating += city.Rating;  //offspring.Time += city.Roads.Find(r => r.To == second.Roads[i + 1]).Time;  }  }  //offspring.Time = Utils.Calculatetime(offspring.Roads, roads);  // Add the starting city at the end to represent the round trip  offspring.Roads.Add(offspring.Roads[0]);  //offspring.Time += offspring.Roads.Last().Roads.Find(r => r.To == offspring.Roads[0]).Time;  // Perform mutation  if (randomizer.NextDouble() < Mutation)  {  int mutationPoint1 = randomizer.Next(1, offspring.Roads.Count - 1);  int mutationPoint2 = randomizer.Next(1, offspring.Roads.Count - 1);  // Swap the cities at the mutation points  City temp = offspring.Roads[mutationPoint1];  offspring.Roads[mutationPoint1] = offspring.Roads[mutationPoint2];  offspring.Roads[mutationPoint2] = temp;  // Update the time for the mutated path  offspring.Time = Utils.Calculatetime(offspring.Roads, roads);  }  //offspring.Time = Utils.Calculatetime(offspring.Roads, roads);  return offspring;  }  public static Population EvolvePopulation(Population population, List<Road> roads)  {  Population newPopulation = new Population(population.PopulationSize);  // Add the fittest individuals from the previous population (elitism)  Path bestPath = Utils.BestPath(population.Paths);  newPopulation.PathAdd(bestPath);  for (int i = 1; i < population.PopulationSize; i++)  {  // Select parents via tournament selection  Path parent1 = SelectViaTournament(population, roads);  Path parent2 = SelectViaTournament(population, roads);  // Perform crossover and mutation to create offspring  Path offspring = CrossoverAndMutate(parent1, parent2, roads);  // Add the offspring to the new population  newPopulation.PathAdd(offspring);  }  newPopulation.Paths = newPopulation.Paths.OrderBy(path => path.Rating).ToList();  return newPopulation;  }  }    public class Population  {  public int PopulationSize;  public List<Path> Paths;  public Population(List<Path> Paths, int PopulationSize, bool Initialize)  {  this.Paths = new List<Path>();  this.PopulationSize = PopulationSize;  for(int i = 0; i < PopulationSize; i++)  {  Random rnd = new Random();  int number = rnd.Next(0, Paths.Count);  if(Paths[number].Time / 3600 < 48)  {  this.Paths.Add(Paths[number]);  }  else  {  i--;  continue;  }    }    }  public Population(int PopulationSize)  {  this.Paths = new List<Path>();  this.PopulationSize = PopulationSize;  }  public void PathAdd(Path path)  {  this.Paths.Add((Path)path);  }  public Path GetFittest()  {  //Console.WriteLine(Utils.BestPath(Paths).Time+ " "+ Utils.BestPath(Paths).Rating);  return Utils.BestPath(Paths);  }  public void printer()  {  InOutUtils.PrintPaths(this.Paths);  }    } |

## Programos rezultatai

Genetinis metodas naudoja 50 atsitiktinių maršrutų, sudaryti pradinę Populiaciją. Tada pagal generacijų skaičių, atlieka Genetinį optimizavimą ir po skaičiavimų, randa geriausią kelią.

|  |
| --- |
| int populationSize = 50;  int numGenerations = 100;  Population population = new Population(distinctPaths, populationSize, true);  population.printer();  Console.WriteLine("--------------------------Genetic Best Path-------------------------------------");  // Iterate for the desired number of generations  for (int generation = 0; generation < numGenerations; generation++)  {  // Evolve the population  population = GeneticOptimization.EvolvePopulation(population, roads);  }  // Get the best path from the final population  Path bestPath = population.GetFittest();  // Output the best path, its rating, and time  Console.WriteLine("Best Path: " + string.Join(" -> ", bestPath.Roads.Select(city => city.Name)));  Console.WriteLine("Rating: " + bestPath.Rating);  Console.WriteLine("Time: " + bestPath.Time); |

Rezultatai:

|  |
| --- |
| -----------------------------------------------------------------------------------  --------------------------Genetic Paths-------------------------------------  Warsaw -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Eindhoven -> London -> Warsaw -> 89280 13,631544421038162  Krakow -> Cagliari -> Catania -> Perugia -> Barcelona -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nantes -> London -> Cologne -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Krakow -> 86400 12,087767916536055  Brussels -> Marseille -> Tours -> London -> Eindhoven -> Manchester -> Belfast -> Newcastle -> Bristol -> Amsterdam -> Birmingham -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Naples -> Palermo -> Bari -> Rome -> Cagliari -> Pisa -> Barcelona -> Nice -> Toulouse -> Lyon -> Nantes -> Porto -> Brussels -> 132900 18,770520207610172  Warsaw -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Barcelona -> Pisa -> Cagliari -> Palermo -> Naples -> Catania -> Bari -> Rome -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Krakow -> Prague -> Kosice -> London -> Warsaw -> 119100 17,071524281391174  Madrid -> Lisbon -> Sevilla -> Rabat -> Malaga -> Valencia -> Toulouse -> Lyon -> Nantes -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Tours -> London -> Madrid -> 105480 16,377972052425662  Athens -> Vienna -> Dubrovnik -> Geneva -> Nice -> Toulouse -> Lyon -> Bordeaux -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Venice -> Bari -> Palermo -> Cagliari -> Pisa -> Barcelona -> Poitiers -> London -> Athens -> 117900 15,59424149462266  Manchester -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Toulouse -> Lyon -> Venice -> Nuernberg -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Zagreb -> Bratislava -> London -> Luxemburg -> Barcelona -> Pisa -> Vienna -> Krakow -> Gdansk -> Manchester -> 127200 17,83573633188772  Manchester -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Tours -> London -> Luxemburg -> Barcelona -> Pisa -> Vienna -> Krakow -> Prague -> Budapest -> Poznan -> Stockholm -> Tallinn -> Nuernberg -> Bologna -> Timisoara -> Karlsruhe -> Valencia -> Manchester -> 169620 23,351912643430776  Riga -> Warsaw -> Kaunas -> Copenhagen -> Gdansk -> Lublin -> Eindhoven -> London -> Cologne -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Riga -> 106740 15,690165734600814  Paris -> Edinburgh -> Belfast -> Newcastle -> Gdansk -> Lublin -> Eindhoven -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Naples -> Palermo -> Bari -> Rome -> Cagliari -> Catania -> Perugia -> Vienna -> Paris -> 98220 13,395052056368401  Valencia -> Bordeaux -> Lyon -> Nantes -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Tours -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> Cluj-napoca -> Valencia -> 153060 22,52882658957264  Lisbon -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Budapest -> Poznan -> Eindhoven -> London -> Cologne -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Lisbon -> 99180 13,644135661780254  London -> Katowice -> Malmoe -> Stockholm -> Tallinn -> Vienna -> Krakow -> Riga -> Helsinki -> Kaunas -> Warsaw -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Naples -> Palermo -> Bari -> Rome -> Cagliari -> Pisa -> Catania -> Perugia -> Brussels -> Liverpool -> Nice -> Venice -> London -> 139440 18,34780324604978  Gothenborg -> Warsaw -> Szczecin -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Zagreb -> Bratislava -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Manchester -> Gothenborg -> 161160 25,192727498388265  Porto -> Barcelona -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Tours -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Porto -> 138000 20,464255505302017  Krakow -> Gothenborg -> Stockholm -> Tallinn -> Paris -> Geneva -> Nice -> Toulouse -> Lyon -> Bordeaux -> Marseille -> Tours -> London -> Cologne -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Krakow -> 91500 13,09440978488551  Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> Venice -> Naples -> Trieste -> Brussels -> Turin -> Barcelona -> Pisa -> Eindhoven -> Rome -> 144000 21,417684851155062  Malaga -> Porto -> Sevilla -> Alicante -> Toulouse -> Lyon -> Bordeaux -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Budapest -> Poznan -> Eindhoven -> London -> Malaga -> 86580 12,108389984192058  Malaga -> Cologne -> Copenhagen -> Gdansk -> Lublin -> Dublin -> Liverpool -> Belfast -> Edinburgh -> Birmingham -> Amsterdam -> Manchester -> Nantes -> Toulouse -> Lyon -> Rennes -> Nice -> Lille -> Krakow -> Prague -> Budapest -> Poznan -> Stockholm -> Malaga -> 131640 15,639403939653437  Bristol -> Copenhagen -> Gdansk -> Lublin -> Eindhoven -> London -> Luxemburg -> Milan -> Marseille -> Tours -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Cagliari -> Palermo -> Naples -> Catania -> Bari -> Rome -> Nice -> Venice -> Bristol -> 119280 16,9262213355247  Manchester -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Toulouse -> Lyon -> Venice -> Nuernberg -> Budapest -> Poznan -> Stockholm -> Tallinn -> London -> Luxemburg -> Milan -> Marseille -> Tours -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Pisa -> Manchester -> 162240 22,690917429238073  Paris -> Geneva -> Nice -> Toulouse -> Lyon -> Bordeaux -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Budapest -> Poznan -> Eindhoven -> London -> Cologne -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Barcelona -> Paris -> 116880 17,890871017019368  Alicante -> Sevilla -> Lisbon -> Madrid -> Porto -> Malaga -> Rabat -> Barcelona -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Tours -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Alicante -> 165540 25,259938357362866  Geneva -> Palermo -> Bari -> Catania -> Perugia -> Vienna -> Krakow -> Gdansk -> Lublin -> Eindhoven -> Milan -> Marseille -> Tours -> Dublin -> Liverpool -> Belfast -> Edinburgh -> Gothenborg -> Stockholm -> Tallinn -> Nuernberg -> Venice -> Lyon -> Copenhagen -> Cologne -> Bologna -> Timisoara -> Karlsruhe -> Zagreb -> Bratislava -> Sofia -> Geneva -> 162720 20,724767157382573  Manchester -> Cologne -> London -> Luxemburg -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Manchester -> 106200 15,74670066877456  Lisbon -> Nantes -> Lyon -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> London -> Lisbon -> 124200 18,629796859892963  Brussels -> Krakow -> Gdansk -> Lublin -> Eindhoven -> London -> Cologne -> Bristol -> Newcastle -> Belfast -> Manchester -> Amsterdam -> Birmingham -> Barcelona -> Lyon -> Nantes -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Cagliari -> Palermo -> Naples -> Catania -> Bari -> Rome -> Thessaloniki -> Bucharest -> Sofia -> Varna -> Nuernberg -> Budapest -> Brussels -> 170400 23,506110420329577  Milan -> Nantes -> Lyon -> Venice -> Nuernberg -> Bologna -> Catania -> Rome -> Nice -> Geneva -> Lille -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Pisa -> Bucharest -> Sofia -> Varna -> Karlsruhe -> London -> Dortmund -> Riga -> Helsinki -> Kaunas -> Warsaw -> Szczecin -> Liverpool -> Belfast -> Edinburgh -> Birmingham -> Milan -> 167880 22,945754117169383  Madrid -> Cagliari -> Palermo -> Naples -> Catania -> Bari -> Rome -> Nice -> Toulouse -> Lyon -> Nantes -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Nuernberg -> Venice -> Timisoara -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Warsaw -> Kaunas -> Madrid -> 167520 23,85455278004576  Valencia -> Bordeaux -> Lyon -> Nantes -> Geneva -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Marseille -> Tours -> London -> Eindhoven -> Manchester -> Valencia -> 96300 14,245549976523202  Madrid -> Cluj-napoca -> Nuernberg -> Barcelona -> Nice -> Toulouse -> Lyon -> Nantes -> Valencia -> Malaga -> Karlsruhe -> Zagreb -> Bratislava -> Thessaloniki -> Bucharest -> Madrid -> 97380 10,137906408046188  Barcelona -> Lyon -> Toulouse -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Barcelona -> 98700 15,817638061238952  Malaga -> Marseille -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Budapest -> Poznan -> Eindhoven -> London -> Cologne -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Malaga -> 117780 16,993893088710383  Bari -> Vienna -> Krakow -> Prague -> Budapest -> Poznan -> Eindhoven -> London -> Cologne -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Barcelona -> Poitiers -> Edinburgh -> Bari -> 94380 13,194027985896803  Warsaw -> Riga -> Dortmund -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Warsaw -> 101400 15,751091072138443  Catania -> Trieste -> Palermo -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Barcelona -> Poitiers -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Catania -> 112200 16,44010802423217  Lisbon -> Milan -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Budapest -> Poznan -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Amsterdam -> Birmingham -> Edinburgh -> Gothenborg -> Stockholm -> Tallinn -> Vienna -> Krakow -> Lisbon -> 114180 14,380532277464024  Catania -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Barcelona -> Poitiers -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Dubrovnik -> Amsterdam -> Bristol -> Catania -> 123900 17,106133308181555  Sevilla -> Turin -> Brussels -> Liverpool -> Belfast -> Edinburgh -> Birmingham -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Sevilla -> 93300 12,942468030766191  Venice -> Copenhagen -> Gdansk -> Lublin -> Dublin -> Liverpool -> Belfast -> Edinburgh -> Birmingham -> Geneva -> Lille -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Nantes -> London -> Cologne -> Milan -> Zagreb -> Bratislava -> Rome -> Prague -> Venice -> 130980 17,970381950351094  Dublin -> Bydgoszcz -> Dublin -> 19200 1,312712672747768  Paris -> Nice -> Venice -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Budapest -> Poznan -> Eindhoven -> Bologna -> Nuernberg -> Krakow -> Paris -> 95580 13,537776015678551  Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Kosice -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Edinburgh -> Brussels -> Bordeaux -> Lyon -> Toulouse -> Nantes -> Marseille -> Milan -> Vienna -> 154500 23,254567782141542  Krakow -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Zagreb -> Bratislava -> Milan -> Marseille -> Bordeaux -> Lyon -> Toulouse -> Nantes -> London -> Eindhoven -> Krakow -> 139800 21,694817570300522  Alicante -> Porto -> Madrid -> Lisbon -> Malaga -> Rabat -> Barcelona -> Nice -> Geneva -> Paris -> Bristol -> Newcastle -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Prague -> Krakow -> Vienna -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Cagliari -> Rome -> Zagreb -> Bratislava -> Milan -> Luxemburg -> London -> Eindhoven -> Manchester -> Amsterdam -> Birmingham -> Alicante -> 170580 25,455702855967512  Rome -> Athens -> Naples -> Trieste -> Brussels -> Liverpool -> Belfast -> Edinburgh -> Birmingham -> Dublin -> Leeds -> Poznan -> Eindhoven -> London -> Cologne -> Milan -> Luxemburg -> Madrid -> Porto -> Sevilla -> Lisbon -> Malaga -> Rabat -> Paris -> Bristol -> Rome -> 133320 17,339310769349446  Brussels -> Krakow -> Gdansk -> Lublin -> Eindhoven -> London -> Cologne -> Bristol -> Newcastle -> Belfast -> Manchester -> Amsterdam -> Birmingham -> Barcelona -> Lyon -> Nantes -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Cagliari -> Brussels -> 120480 16,189015028232433  Krakow -> Riga -> Dortmund -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Sofia -> Bucharest -> Thessaloniki -> Naples -> Palermo -> Bari -> Rome -> Cagliari -> Pisa -> Barcelona -> Bordeaux -> Lyon -> Toulouse -> Geneva -> Nice -> Venice -> Marseille -> Milan -> Zagreb -> Bratislava -> Copenhagen -> Krakow -> 144120 22,45237528851904  Amsterdam -> Nice -> Toulouse -> Lyon -> Bordeaux -> Marseille -> Tours -> Porto -> Sevilla -> Lisbon -> Madrid -> Rabat -> Malaga -> Valencia -> Cagliari -> Palermo -> Naples -> Catania -> Bari -> Rome -> Zagreb -> Bratislava -> Sofia -> Bucharest -> Thessaloniki -> Budapest -> Poznan -> Eindhoven -> Bologna -> Nuernberg -> Krakow -> Vienna -> Dubrovnik -> Amsterdam -> 157320 22,132039305182303  Madrid -> Bologna -> Bari -> Palermo -> Naples -> Catania -> Rome -> Nice -> Toulouse -> Lyon -> Nantes -> London -> Eindhoven -> Manchester -> Belfast -> Liverpool -> Dublin -> Leeds -> Poznan -> Budapest -> Madrid -> 96900 13,662923402747943  --------------------------Genetic Best Path-------------------------------------  Best Path: Madrid -> Dublin -> Vienna -> Catania -> Bari -> Cagliari -> Malaga -> Lyon -> Zagreb -> Paris -> Rabat -> Newcastle -> Liverpool -> Leeds -> Geneva -> Krakow -> Poznan -> Birmingham -> Luxemburg -> Palermo -> Sevilla -> Bristol -> Toulouse -> Manchester -> Naples -> Bologna -> Valencia -> Rome -> London -> Lisbon -> Budapest -> Bratislava -> Milan -> Eindhoven -> Belfast -> Prague -> Amsterdam -> Madrid  Rating: 25,518354778586158  Time: 0 |

Apačioje randame Geriausią maršrutą :

Best Path: Madrid -> Dublin -> Vienna -> Catania -> Bari -> Cagliari -> Malaga -> Lyon -> Zagreb -> Paris -> Rabat -> Newcastle -> Liverpool -> Leeds -> Geneva -> Krakow -> Poznan -> Birmingham -> Luxemburg -> Palermo -> Sevilla -> Bristol -> Toulouse -> Manchester -> Naples -> Bologna -> Valencia -> Rome -> London -> Lisbon -> Budapest -> Bratislava -> Milan -> Eindhoven -> Belfast -> Prague -> Amsterdam -> Madrid

Rating: 25,518354778586158