**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 4 з дисципліни

«Проектування алгоритмів»

„**Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Гуськов Кирило Михайлович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 11](#_Toc51260925)

[Висновок 12](#_Toc51260926)

[Критерії оцінювання 13](#_Toc51260927)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою.

# Завдання

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Задача і алгоритм** |
| 1 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 2 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 3 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 4 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 5 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 6 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 7 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 8 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 9 | Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники). |
| 10 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 11 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 12 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники). |
| 13 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 14 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 15 | Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники). |
| 16 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 17 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 18 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники). |
| 19 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 20 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 21 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники). |
| 22 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 23 | Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм (α = 3, β = 2, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 24 | Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники). |
| 25 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 26 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 27 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 28 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 29 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 30 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 31 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 32 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 33 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 34 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 35 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

PA’Lab4.cpp

#include<iostream>

#include<ctime>

#include"Header.h"

usingnamespacestd;

intmain()

{

srand(time(NULL));

intgraph[200][200];

doublepheromoneGraph[200][200];

doublepheromoneSumGraph[200][200];

BuildGraph(graph);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

TSPAlgorithm(graph,pheromoneGraph,pheromoneSumGraph);

}

voidBuildGraph(intgraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

if(i==j)

{

graph[i][j]=0;

}

else

{

graph[i][j]=rand()%51;

}

}

}

}

voidBuildPheromoneGraph(doublepheromoneGraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

if(i==j)

{

pheromoneGraph[i][j]=0;

}

else

{

pheromoneGraph[i][j]=rand()%9+1;

pheromoneGraph[i][j]=pheromoneGraph[i][j]/10;

}

}

}

}

voidBuildPheromoneSumGraph(doublepheromoneSumGraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

pheromoneSumGraph[i][j]=0;

}

}

}

voidBuildVisibilityGraph(doublevisibilityGraph[200][200],intgraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

visibilityGraph[i][j]=1/graph[i][j];

}

}

}

Header.H

#pragmaonce

#include<iostream>

#include<vector>

usingnamespacestd;

structProbability

{

doublechance;

intvertix;

};

voidBuildGraph(intgraph[200][200]);

voidBuildPheromoneGraph(doublepheromoneGraph[200][200]);

voidBuildPheromoneSumGraph(doublepheromoneSumGraph[200][200]);

voidBuildVisibilityGraph(doublevisibilityGraph[200][200],intgraph[200][200]);

voidTSPAlgorithm(intgraph[200][200],doublepheromoneGraph[200][200],doublepheromoneSumGraph[200][200]);

intGenerateStartingPoint(intUsedStartingPoints[45],intM);

boolCheckForStartingPoint(intUsedStartingPoints[45],intStartingPoint,intM);

doubleMoveProbability(intcurVertix,intdestination,doublepheromone,intalpha,intbeta,intgraph[200][200],doubleStepTwo);

doubleMoveProbabilityStepOne(intcurVertix,intdestination,doublepheromone,intalpha,intbeta,intgraph[200][200]);

doubleGetLmin(intgraph[200][200]);

intGetMin(intgraph[200][200],intcurrVertix);

boolCheckVisited(intvisited[200]);

voidprintArray(intarr[200]);

Ant.H

#pragmaonce

#include"Header.h"

classAnt

{

public:

intStartingPoint;

vector<int>Path;

vector<int>UnvisitedVertices;

Ant()

{

this->Path.push\_back(1);

this->UnvisitedVertices.push\_back(1);

this->Path.clear();

this->UnvisitedVertices.clear();

}

Ant(intStartingPoint);

voidReset();

voidMoveToVertix(intdestination,intdestinationIdx);

voidChooseDestination(vector<Probability>Probabilities);

doublegetL(intgraph[200][200]);

};

Ant.cpp

#include"Ant.h"

Ant::Ant(intStartingPoint)

{

this->StartingPoint=StartingPoint;

this->Path.push\_back(StartingPoint);

for(inti=0;i<200;i++)

{

if(i!=StartingPoint)

{

this->UnvisitedVertices.push\_back(i);

}

}

}

voidAnt::Reset()

{

this->Path.clear();

this->Path.push\_back(this->StartingPoint);

this->UnvisitedVertices.clear();

for(inti=0;i<200;i++)

{

if(i!=this->Path.at(0))

{

this->UnvisitedVertices.push\_back(i);

}

}

}

voidAnt::MoveToVertix(intdestination,intdestinationIdx)

{

this->Path.push\_back(destination);

this->UnvisitedVertices.erase(this->UnvisitedVertices.begin()+destinationIdx);

}

voidAnt::ChooseDestination(vector<Probability>Probabilities)

{

doublelowerBound=0;

doublerandomValue=((double)rand()/(RAND\_MAX));

intdestination=0;

intdestinationIdx=0;

if(randomValue==1)

{

destination=UnvisitedVertices.at(UnvisitedVertices.size()-1);

destinationIdx=UnvisitedVertices.size()-1;

}

elseif(randomValue==0)

{

destination=UnvisitedVertices.at(0);

destinationIdx=0;

}

elseif(UnvisitedVertices.size()==1)

{

destination=UnvisitedVertices.at(0);

destinationIdx=0;

}

else

{

for(intj=0;j<Probabilities.size();j++)

{

if((Probabilities.at(j).chance>-1)&&(Probabilities.at(j).vertix==this->UnvisitedVertices.at(j)))

{

if((lowerBound<randomValue)&&(randomValue<=lowerBound+Probabilities.at(j).chance))

{

destination=UnvisitedVertices.at(j);

destinationIdx=j;

break;

}

else

{

lowerBound+=Probabilities.at(j).chance;

}

}

}

}

MoveToVertix(destination,destinationIdx);

}

doubleAnt::getL(intgraph[200][200])

{

intL=0;

for(inti=0;i<this->Path.size()-1;i++)

{

L+=graph[this->Path.at(i)][this->Path.at(i+1)];

}

returnL;

}

TSP.cpp

#include"Header.h"

#include"Ant.h"

voidprintList(AntAnt);

boolCheckOrig(vector<int>Path)

{

for(inti=0;i<Path.size()-1;i++)

{

for(intj=i+1;j<Path.size()-1;j++)

{

if(Path.at(i)==Path.at(j))

{

returnfalse;

}

}

}

returntrue;

}

voidTSPAlgorithm(intgraph[200][200],doublepheromoneGraph[200][200],doublepheromoneSumGraph[200][200])

{

intAlpha=3;

intBeta=2;

doubleRo=0.3;

doubleLmin=GetLmin(graph);

vector<Ant>Ants;

vector<Probability>Probabilities;

intUsedStartingPoints[45];

for(inti=0;i<45;i++)

{

UsedStartingPoints[i]=-1;

}

for(inti=0;i<45;i++)

{

intStartingPoint=GenerateStartingPoint(UsedStartingPoints,i);

Ants.push\_back(Ant(StartingPoint));

}

intiteration=0;

intcount=0;

while(iteration<20)

{

for(inti=0;i<Ants.size();i++)

{

Ants.at(i).Reset();

}

for(inti=0;i<Ants.size();i++)

{

while(Ants.at(i).UnvisitedVertices.size()!=0)

{

Probabilities.clear();

doubleStepTwo=0;

for(intp=0;p<Ants.at(i).UnvisitedVertices.size();p++)

{

StepTwo+=MoveProbabilityStepOne(Ants.at(i).Path.at(Ants.at(i).Path.size()-1),Ants.at(i).UnvisitedVertices.at(p),pheromoneGraph[Ants.at(i).Path.at(Ants.at(i).Path.size()-1)][Ants.at(i).UnvisitedVertices.at(p)],Alpha,Beta,graph);

}

for(intj=0;j<Ants.at(i).UnvisitedVertices.size();j++)

{

if(graph[Ants.at(i).Path.at(Ants.at(i).Path.size()-1)][Ants.at(i).UnvisitedVertices.at(j)]!=0)

{

Probabilityprob;

prob.chance=MoveProbability(Ants.at(i).Path.at(Ants.at(i).Path.size()-1),Ants.at(i).UnvisitedVertices.at(j),pheromoneGraph[Ants.at(i).Path.at(Ants.at(i).Path.size()-1)][Ants.at(i).UnvisitedVertices.at(j)],Alpha,Beta,graph,StepTwo);

prob.vertix=Ants.at(i).UnvisitedVertices.at(j);

Probabilities.push\_back(prob);

}

else

{

Probabilityprob;

prob.chance=-1;

prob.vertix=Ants.at(i).UnvisitedVertices.at(j);

Probabilities.push\_back(prob);

}

}

Ants.at(i).ChooseDestination(Probabilities);

pheromoneSumGraph[Ants.at(i).Path.at(Ants.at(i).Path.size()-2)][Ants.at(i).Path.at(Ants.at(i).Path.size()-1)]+=Lmin/Ants.at(i).getL(graph);

count++;

}

}

doubletemp;

for(intm=0;m<200;m++)

{

for(intn=0;n<200;n++)

{

if(m!=n)

{

temp=pheromoneGraph[m][n];

pheromoneGraph[m][n]=(1-Ro)\*temp+pheromoneSumGraph[m][n];

}

}

}

iteration++;

}

intminIdx=0;

intmin=Ants.at(0).getL(graph);

for(inti=1;i<Ants.size();i++)

{

if(Ants.at(i).getL(graph)<min)

{

min=Ants.at(i).getL(graph);

minIdx=i;

}

}

printList(Ants.at(minIdx));

cout<<"L min: "<<Lmin<<endl;

cout<<"Path L: "<<Ants.at(minIdx).getL(graph);

}

intGenerateStartingPoint(intUsedStartingPoints[45],intM)

{

intStartingPoint=rand()%200;

if(!CheckForStartingPoint(UsedStartingPoints,StartingPoint,M))

{

StartingPoint=GenerateStartingPoint(UsedStartingPoints,M);

}

returnStartingPoint;

}

boolCheckForStartingPoint(intUsedStartingPoints[45],intStartingPoint,intM)

{

for(inti=0;i<M;i++)

{

if(StartingPoint==UsedStartingPoints[i])

{

returnfalse;

}

}

returntrue;

}

doubleMoveProbability(intcurVertix,intdestination,doublepheromone,intalpha,intbeta,intgraph[200][200],doubleStepTwo)

{

doubleStepOne=MoveProbabilityStepOne(curVertix,destination,pheromone,alpha,beta,graph);

doubleprobability=StepOne/StepTwo;

returnprobability;

}

doubleMoveProbabilityStepOne(intcurVertix,intdestination,doublepheromone,intalpha,intbeta,intgraph[200][200])

{

if(graph[curVertix][destination]!=0)

{

doubleedge=graph[curVertix][destination];

doublevisibility=1/edge;

doubleStepOne=(pow(pheromone,alpha)\*pow(visibility,beta));

returnStepOne;

}

else

{

return0;

}

}

doubleGetLmin(intgraph[200][200])

{

vector<int>Path;

intUnvisitedVertices[200];

for(inti=0;i<200;i++)

{

UnvisitedVertices[i]=-1;

}

doubleLmin=0;

intmin=graph[0][1];

intnewVertix=1;

intoldVertix=0;

UnvisitedVertices[oldVertix]=1;

UnvisitedVertices[newVertix]=1;

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

if((graph[i][j]<min)&&(graph[i][j]!=0))

{

min=graph[i][j];

UnvisitedVertices[oldVertix]=-1;

UnvisitedVertices[newVertix]=-1;

oldVertix=i;

newVertix=j;

UnvisitedVertices[oldVertix]=1;

UnvisitedVertices[newVertix]=1;

}

}

}

intfirst=oldVertix;

Path.push\_back(oldVertix);

Path.push\_back(newVertix);

while(Path.size()!=200)

{

intnext=GetMin(graph,UnvisitedVertices,Path.back());

UnvisitedVertices[next]=1;

Path.push\_back(next);

}

Path.push\_back(first);

boolcheck=CheckOrig(Path);

for(inti=0;i<Path.size()-1;i++)

{

Lmin+=graph[Path.at(i)][Path.at(i+1)];

}

returnLmin;

}

intGetMin(intgraph[200][200],intUnvisitedVertices[200],intcurrVertix)

{

intmin=0;

intnextVertix=0;

for(inti=0;i<200;i++)

{

if((graph[currVertix][i]!=0)&&(UnvisitedVertices[i]==-1))

{

min=graph[currVertix][i];

nextVertix=i;

UnvisitedVertices[i]=1;

break;

}

}

for(inti=0;i<200;i++)

{

if((graph[currVertix][i]!=0)&&(graph[currVertix][i]<min)&&(UnvisitedVertices[i]==-1))

{

min=graph[currVertix][i];

UnvisitedVertices[nextVertix]=-1;

nextVertix=i;

UnvisitedVertices[i]==1;

}

}

returnnextVertix;

}

voidprintList(AntAnt)

{

cout<<Ant.Path.size()<<endl;

for(inti=0;i<200;i++)

{

cout<<Ant.Path.at(i)<<"->";

}

cout<<Ant.Path.at(0);

cout<<endl;

cout<<endl;

}

voidprintArray(intarr[200])

{

for(inti=0;i<200;i++)

{

cout<<arr[i]<<"->";

}

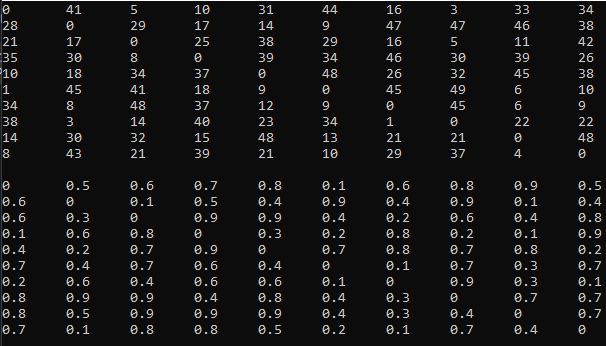
cout<<endl;

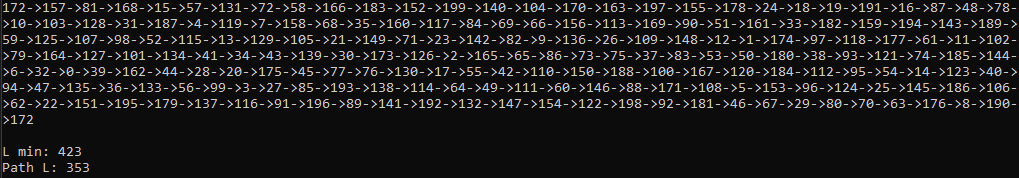
cout<<endl;

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 – Частина початкових параметрів: відстані між вершинами та феромон на ребрах них

Рисунок 3.2 – Кінцевий шлях після 20 ітерацій, отримане Lmin та довжина шляху L

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

|  |  |
| --- | --- |
| Кількість ітерацій | Значення цільової функції |
| 20 | 353 |
| 70 | 333 |
| 120 | 333 |
| 170 | 77 |
| 220 | 356 |
| 270 | 84 |
| 320 | 338 |
| 370 | 120 |
| 420 | 328 |
| 470 | 120 |
| 520 | 120 |
| 570 | 148 |
| 620 | 148 |
| 670 | 148 |
| 720 | 149 |
| 770 | 149 |
| 820 | 160 |
| 870 | 160 |
| 920 | 172 |
| 970 | 172 |
| 1000 |  |

### Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи…

Критерії оцінювання

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* програмна реалізація алгоритму – 75%;
* тестування алгоритму– 20%;
* висновок – 5%.