**Виконав:** Герман І.В.

**Група:** КН-404

**Кафедра:** САПР

**Дисципліна:** Дискретні моделі в САПР

# Варіант: 6

**Перевірив:** канд. техн. наук , доц. Кривий Р.З.

**Дата:** 18.02.2021

**GitHub:** https://github.com/Gera-byte/DM\_CAD/tree/main/Lab3

# ЗВІТ

до лабораторної роботи №3 на тему:

АЛГОРИТМ РІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА

# МЕТА РОБОТИ

Метою даної лабораторної роботи є вивчення і дослідження алгоритмів рішення задачі комівояжера.

**2. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

Зада́ча комівояже́ра (комівояжер — бродячий торговець; англ. Travelling Salesman Problem, TSP; нім. Problem des Handlungsreisenden) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.

Існує маса різновидів узагальненої постановки задачі, зокрема геометрична задача комівояжера (коли матриця відстаней відображає відстані між точками на площині), трикутна задача комівояжера (коли на матриці вартостей виконується нерівність трикутника), симетрична та асиметрична задачі комівояжера.

Прості методи розв'язання задачі комівояжера: повний лексичний перебір, жадібні алгоритми (метод найближчого сусіда), метод включення найближчого міста, метод найдешевшого включення, метод мінімального кістяка дерева. На практиці застосовують різні модифікації ефективніших методів: метод гілок і меж і метод генетичних алгоритмів, а так само алгоритм мурашиної колонії.

Всі ефективні (такі, що скорочують повний перебір) методи розв'язання задачі комівояжера — евристичні. У більшості евристичних методів знаходиться не найефективніший маршрут, а наближений розв'язок. Користуються популярністю так звані any-time алгоритми, тобто алгоритми, що поступово покращують деякий поточний наближений розв'язок.

Задача комівояжера — NP-повна. Часто на ній проводять випробування нових підходів до евристичного скорочення повного перебору.

**3. ЗАВДАННЯ**

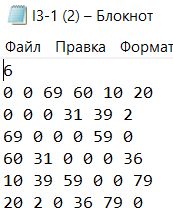
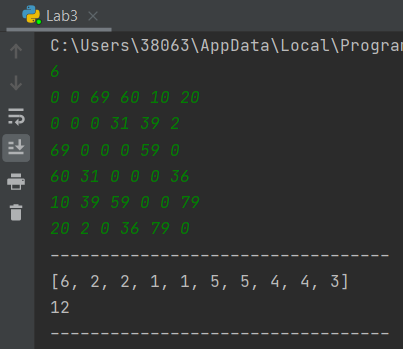
Задача комівояжера. Реалізувати метод гілок і меж  


Рис. 1. Індивідуальне завдання

# 4. Код програми

def Min(lst,myindex):  
 return min(x for idx, x in enumerate(lst) if idx != myindex)  
  
def Delete(matrix,index1,index2):  
 del matrix[index1]  
 for i in matrix:  
 del i[index2]  
 return matrix  
  
def PrintMatrix(matrix):  
 print("---------------")  
 for i in range(len(matrix)):  
 print(matrix[i])  
 print("---------------")  
  
n=int(input())  
matrix=[]  
H=0  
PathLenght=0  
Str=[]  
Stb=[]  
res=[]  
result=[]  
StartMatrix=[]  
  
for i in range(n):  
 Str.append(i)  
 Stb.append(i)  
  
for i in range(n): matrix.append(list(map(int, input().split())))  
for i in range(n):StartMatrix.append(matrix[i].copy())  
  
for i in range(n): matrix[i][i]=float('inf')  
  
while True:  
 for i in range(len(matrix)):  
 temp=min(matrix[i])  
 H+=temp  
 for j in range(len(matrix)):  
 matrix[i][j]-=temp  
  
 for i in range(len(matrix)):  
 temp = min(row[i] for row in matrix)  
 H+=temp  
 for j in range(len(matrix)):  
 matrix[j][i]-=temp  
  
 NullMax=0  
 index1=0  
 index2=0  
 tmp=0  
 for i in range(len(matrix)):  
 for j in range(len(matrix)):  
 if matrix[i][j]==0:  
 tmp=Min(matrix[i],j)+Min((row[j] for row in matrix),i)  
 if tmp>=NullMax:  
 NullMax=tmp  
 index1=i  
 index2=j  
 res.append(Str[index1]+1)  
 res.append(Stb[index2]+1)  
 oldIndex1=Str[index1]  
 oldIndex2=Stb[index2]  
 if oldIndex2 in Str and oldIndex1 in Stb:  
 NewIndex1=Str.index(oldIndex2)  
 NewIndex2=Stb.index(oldIndex1)  
 matrix[NewIndex1][NewIndex2]=float('inf')  
 del Str[index1]  
 del Stb[index2]  
 matrix=Delete(matrix,index1,index2)  
 if len(matrix)==1:break  
  
for i in range(0,len(res)-1,2):  
 if res.count(res[i])<2:  
 result.append(res[i])  
 result.append(res[i+1])  
for i in range(0,len(res)-1,2):  
 for j in range(0,len(res)-1,2):  
 if result[len(result)-1]==res[j]:  
 result.append(res[j])  
 result.append(res[j+1])  
print("----------------------------------")  
print(result)  
  
for i in range(0,len(result)-1,2):  
 if i==len(result)-2:  
 PathLenght+=StartMatrix[result[i]-1][result[i+1]-1]  
 PathLenght+=StartMatrix[result[i+1]-1][result[0]-1]  
 else: PathLenght+=StartMatrix[result[i]-1][result[i+1]-1]  
print(PathLenght)  
print("----------------------------------")  
input()

  
Рис. 2. Результат виконання програми

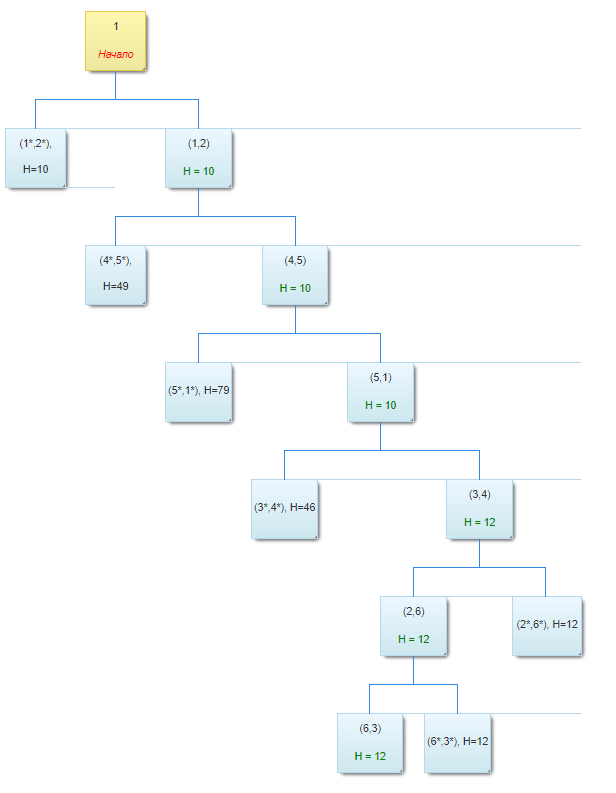


Рис. 3. Дерево рішень задачі

## Висновок

На лабораторній роботі вивчено і досліджено алгоритм рішення задачі комівояжера.