Wydział Nauk Inżynieryjnych ANS w Nowym Sączu Optymalizacja dyskretna - projekt		
Temat: Problem komiwojażera - dołączanie najdalszego wierzchołka		Symbol: OD_P4
Nazwisko i imię: Ryczek Arkadiusz	Ocena sprawozdania:	Zaliczenie:
Data wykonania ćwiczenia:	Oceniane efekty uczenia się: EUU1=..... ,EUU2=..... ,EUU3=....., EUK1=.....	

Cel zajęć

Celem zajęć była implementacja algorytmu rozwiązującego problem komiwojażera (TSP - Travelling Salesman Problem) metodą dołączania najdalszego wierzchołka oraz zastosowanie tego algorytmu do optymalizacji pracy automatu wiertarskiego.

Problem komiwojażera - podstawy teoretyczne

Problem komiwojażera polega na znalezieniu najkrótszej trasy przechodzącej przez wszystkie wierzchołki grafu (miasta) dokładnie raz i powracającej do wierzchołka startowego. Formalnie, dany jest graf pełny z wagami na krawędziach (odległościami między miastami). Należy znaleźć cykl Hamiltona o minimalnej sumie wag.

Metoda dołączania najdalszego wierzchołka (farthest insertion) jest algorytmem zachłannym, który rozpoczyna od trasy zawierającej dwa najbliższe wierzchołki, a następnie iteracyjnie dołącza do trasy wierzchołek najbardziej oddalony od bieżącej trasy, wstawiając go w optymalne miejsce.

Algorytm dołączania najdalszego wierzchołka

Pseudokod

```
1. Wybierz wierzchołek startowy s
2. Inicjalizuj trasę T = [s], zbiór odwiedzonych V = {s}
3. Inicjalizuj tablicę d odległości od trasy:
   dla każdego wierzchołka i: d[i] = odległość od s do i
4. Dopóki |V| < n:
   a) Znajdź wierzchołek v nie należący do V o maksymalnej wartości d[v]
   b) Znajdź pozycję w trasie T, gdzie wstawienie v minimalizuje wzrost długości:
        delta = A[i,v] + A[v,j] - A[i,j]
   c) Wstaw v do T w znalezionej pozycji
   d) Zaktualizuj V = V u {v}
   e) Zaktualizuj tablicę d: dla każdego wierzchołka i nieodwiedzonego
        d[i] = min(d[i], A[v,i])
5. Dodaj s na koniec trasy aby zamknąć cykl
```

Przykład

Dana jest macierz odległości między 5 miastami:

$$\begin{bmatrix} 0 & 2 & 4 & 7 & 3 \\ 2 & 0 & 4 & 5 & 7 \\ 8 & 4 & 0 & 7 & 3 \\ 5 & 9 & 2 & 0 & 7 \\ 4 & 8 & 1 & 4 & 0 \end{bmatrix}$$

Algorytm wykonuje kolejne iteracje dołączając wierzchołki 4, 5, 2 i 3, uzyskując końcową trasę (1, 2, 3, 5, 4, 1) o długości 18.

Implementacja algorytmu dla problemu ogólnego

Poniżej przedstawiono implementację algorytmu dołączania najdalszego wierzchołka w języku C#.

```
using System;
using System.Collections.Generic;
using System.Linq;

class Program
{
    static void Main()
    {
        Console.WriteLine("=== PROBLEM KOMIWOJAŻERA - DOŁĄCZANIE NAJDALSZEGO
            WIERZCHOŁKA ===\n");

        Console.Write("Podaj liczbę miast: ");
        int n = int.Parse(Console.ReadLine() ?? "0");

        double[,] A = new double[n, n];
        Console.WriteLine("Podaj macierz odległości (n wierszy po n liczb,
            oddzielone spacją):");
        for (int i = 0; i < n; i++)
        {
            string[] row = Console.ReadLine().Split();
            for (int j = 0; j < n; j++)
                A[i, j] = double.Parse(row[j]);
        }

        int s = 0;
        List<int> trasa = new List<int> { s };
        bool[] odwiedzone = new bool[n];
        odwiedzone[s] = true;

        double[] d = new double[n];
        for (int i = 0; i < n; i++)
            d[i] = (i == s) ? double.PositiveInfinity : A[s, i];

        Console.WriteLine($"\nPoczątkowa tablica d = [{string.Join(", ", d.
            Select(x => x == double.PositiveInfinity ? "-" : x.ToString()))}]");

        double dlugosc = 0;

        while (trasa.Count < n)
        {
            int v = -1;
            double maxD = double.MinValue;
            for (int i = 0; i < n; i++)
            {
                if (!odwiedzone[i] && d[i] > maxD)
                {
                    maxD = d[i];
                    v = i;
                }
            }
```

```
49              if (v == -1) break;
50
51              Console.WriteLine($"\nNajdalszy wierzchołek: {v + 1} (d = {maxD})")
                    ;
52
53              double bestDelta = double.MaxValue;
54              int insertPos = 0;
55
56              if (!trasa.Contains(s))
57                  trasa.Add(s);
58
59              for (int k = 0; k < trasa.Count; k++)
60              {
61                  int i = trasa[k];
62                  int j = (k + 1 == trasa.Count) ? s : trasa[k + 1];
63
64                  double kij = A[i, v] + A[v, j] - A[i, j];
65                  if (kij < bestDelta)
66                  {
67                      bestDelta = kij;
68                      insertPos = k + 1;
69                  }
70              }
71
72              trasa.Insert(insertPos, v);
73              odwiedzone[v] = true;
74              dlugosc += bestDelta;
75
76              Console.WriteLine($"Dodano wierzchołek {v + 1} (delta = {bestDelta
                    }), nowa trasa: ({string.Join(", ", trasa.Select(x => x + 1))})"
                    );
77
78              for (int i = 0; i < n; i++)
79              {
80                  if (!odwiedzone[i])
81                      d[i] = Math.Min(d[i], A[v, i]);
82                  else
83                      d[i] = double.PositiveInfinity;
84              }
85
86              Console.WriteLine($"Nowa tablica d = [{string.Join(", ", d.Select(x
                    => x == double.PositiveInfinity ? "-" : x.ToString()))}]");
87          }
88
89          if (trasa.Last() != s)
90              trasa.Add(s);
91
92          double total = 0;
93          for (int i = 0; i < trasa.Count - 1; i++)
94              total += A[trasa[i], trasa[i + 1]];
95
96          Console.WriteLine("\n=== WYNIK KOŃCOWY ===");
97          Console.WriteLine($"Trasa: ({string.Join(", ", trasa.Select(x => x + 1)
                )})");
98          Console.WriteLine($"Całkowita długość trasy: {total}");
```

```
99        }
100   }
```

Listing 1: Implementacja algorytmu dołączania najdalszego wierzchołka dla problemu komiwojażera.

Opis implementacji dla problemu ogólnego

Program składa się z następujących kluczowych elementów:

- **Wczytywanie danych** - użytkownik podaje liczbę miast oraz macierz odległości między nimi
- **Inicjalizacja** - wybór wierzchołka startowego, utworzenie początkowej trasy i tablicy odległości
- **Główna pętla algorytmu** - iteracyjne dołączanie najdalszych wierzchołków:
  - Znajdowanie wierzchołka o maksymalnej odległości od bieżącej trasy
  - Wyznaczanie optymalnej pozycji wstawienia tego wierzchołka
  - Aktualizacja trasy i tablicy odległości
- **Prezentacja wyników** - wyświetlenie końcowej trasy z całkowitą długością
- **Szczegółowe logowanie** - program pokazuje każdy krok algorytmu wraz z wartościami tablicy odległości

Program umożliwia śledzenie procesu optymalizacji poprzez wyświetlanie kolejnych iteracji, co pozwala na zrozumienie działania algorytmu dołączania najdalszego wierzchołka Rys. 1 (s. 4).

```
Podaj liczbę miast: 5
Podaj macierz odległości (n wierszy po n liczb, oddzielone spacją)
0 2 4 7 3
2 0 4 5 7
8 4 0 7 3
5 9 2 0 7
4 8 1 4 0

Początkowa tablica d = [-, 2, 4, 7, 3]

Najdalszy wierzchołek: 4 (d = 7)
Dodano wierzchołek 4 (Δ = 12), nowa trasa: (1, 4)
Nowa tablica d = [-, 2, 2, -, 3]

Najdalszy wierzchołek: 5 (d = 3)
Dodano wierzchołek 5 (Δ = 0), nowa trasa: (1, 5, 4)
Nowa tablica d = [-, 2, 1, -, -]

Najdalszy wierzchołek: 2 (d = 2)
Dodano wierzchołek 2 (Δ = 6), nowa trasa: (1, 2, 5, 4)
Nowa tablica d = [-, -, 1, -, -]

Najdalszy wierzchołek: 3 (d = 1)
Dodano wierzchołek 3 (Δ = 0), nowa trasa: (1, 2, 3, 5, 4)
Nowa tablica d = [-, -, -, -, -]

=== WYNIK KOŃCOWY ===
Trasa: (1, 2, 3, 5, 4, 1)
Całkowita długość trasy: 18
```

Rysunek 1: Wynik algorytmu komiwojażera

Implementacja dla automatu wiertarskiego

Poniżej przedstawiono specjalizowaną implementację algorytmu dla problemu automatu wiertarskiego.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;

class Program
{
    class Punkt
    {
        public int Id { get; set; }
        public double X { get; set; }
        public double Y { get; set; }
        public string Nazwa { get; set; }

        public Punkt(int id, double x, double y, string nazwa = "")
        {
            Id = id;
            X = x;
            Y = y;
            Nazwa = nazwa;
        }
    }

    static void Main()
    {
        Console.WriteLine("=== AUTOMAT WIERTARSKI - OPTYMALIZACJA TRASY ===\n")
            ;

        List<Punkt> punkty = new List<Punkt>();
        Punkt startowy = null;

        Console.WriteLine("Wybierz metodę wczytywania danych:");
        Console.WriteLine("1 - Z klawiatury");
        Console.WriteLine("2 - Losowe współrzędne");
        Console.WriteLine("3 - Z pliku");
        Console.Write("Twój wybór: ");

        int wybor = int.Parse(Console.ReadLine() ?? "1");

        switch (wybor)
        {
            case 1:
                WczytajZKlawiatury(ref punkty, ref startowy);
                break;
            case 2:
                GenerujLosowePunkty(ref punkty, ref startowy);
                break;
            case 3:
                WczytajZPliku(ref punkty, ref startowy);
                break;
            default:
```

```
                WczytajZKlawiatury(ref punkty, ref startowy);
                break;
        }

        double[,] macierzOdleglosci = UtworzMacierzOdleglosci(punkty);

        Console.WriteLine("\n=== MACIERZ ODLEGŁOŚCI ===");
        WyswietlMacierz(macierzOdleglosci, punkty);

        List<int> optymalnaTrasa = RozwiazTSP(macierzOdleglosci, 0);

        Console.WriteLine("\n=== WYNIK OPTYMALIZACJI ===");
        WyswietlTrase(optymalnaTrasa, punkty, macierzOdleglosci);
    }

    static void WczytajZKlawiatury(ref List<Punkt> punkty, ref Punkt startowy)
    {
        Console.WriteLine("\n--- Wczytywanie z klawiatury ---");

        Console.Write("Podaj współrzędne punktu A (x0 y0): ");
        string[] a = Console.ReadLine().Split();
        double x0 = double.Parse(a[0]);
        double y0 = double.Parse(a[1]);

        Console.Write("Podaj współrzędne punktu B (xn yn): ");
        string[] b = Console.ReadLine().Split();
        double xn = double.Parse(b[0]);
        double yn = double.Parse(b[1]);

        startowy = new Punkt(0, x0, y0, "START");
        punkty.Add(startowy);

        Console.Write("Podaj liczbę otworów: ");
        int n = int.Parse(Console.ReadLine());

        for (int i = 0; i < n; i++)
        {
            Console.Write($"Podaj współrzędne otworu {i + 1} (x y): ");
            string[] otwor = Console.ReadLine().Split();
            double x = double.Parse(otwor[0]);
            double y = double.Parse(otwor[1]);
            punkty.Add(new Punkt(i + 1, x, y, $"OTWÓR_{i + 1}"));
        }
    }

    static void GenerujLosowePunkty(ref List<Punkt> punkty, ref Punkt startowy)
    {
        Console.WriteLine("\n--- Generowanie losowych punktów ---");

        Random rand = new Random();

        double x0 = rand.NextDouble() * 100;
        double y0 = rand.NextDouble() * 100;
        double xn = x0 + rand.NextDouble() * 50 + 10;
        double yn = y0 + rand.NextDouble() * 50 + 10;
```

```
107         startowy = new Punkt(0, x0, y0, "START");
108         punkty.Add(startowy);
109
110         Console.WriteLine($"Materiał: A({x0:F2}, {y0:F2}) - B({xn:F2}, {yn:F2})
                ");
111
112         int n = rand.Next(5, 15);
113         Console.WriteLine($"Liczba otworów: {n}");
114
115         for (int i = 0; i < n; i++)
116         {
117             double x = x0 + rand.NextDouble() * (xn - x0);
118             double y = y0 + rand.NextDouble() * (yn - y0);
119             punkty.Add(new Punkt(i + 1, x, y, $"OTWÓR_{i + 1}"));
120             Console.WriteLine($"Otwór {i + 1}: ({x:F2}, {y:F2})");
121         }
122     }
123
124     static void WczytajZPliku(ref List<Punkt> punkty, ref Punkt startowy)
125     {
126         Console.WriteLine("\n--- Wczytywanie z pliku ---");
127         Console.Write("Podaj ścieżkę do pliku: ");
128         string sciezka = Console.ReadLine();
129
130         try
131         {
132             string[] linie = File.ReadAllLines(sciezka);
133
134             string[] a = linie[0].Split();
135             double x0 = double.Parse(a[0]);
136             double y0 = double.Parse(a[1]);
137
138             string[] b = linie[1].Split();
139             double xn = double.Parse(b[0]);
140             double yn = double.Parse(b[1]);
141
142             startowy = new Punkt(0, x0, y0, "START");
143             punkty.Add(startowy);
144
145             for (int i = 2; i < linie.Length; i++)
146             {
147                 string[] otwor = linie[i].Split();
148                 double x = double.Parse(otwor[0]);
149                 double y = double.Parse(otwor[1]);
150                 punkty.Add(new Punkt(i - 1, x, y, $"OTWÓR_{i - 1}"));
151             }
152
153             Console.WriteLine($"Wczytano {punkty.Count - 1} otworów z pliku");
154         }
155         catch (Exception ex)
156         {
157             Console.WriteLine($"Błąd wczytywania pliku: {ex.Message}");
158             GenerujLosowePunkty(ref punkty, ref startowy);
159         }
```

```
160     }
162     static double[,] UtworzMacierzOdleglosci(List<Punkt> punkty)
163     {
164         int n = punkty.Count;
165         double[,] macierz = new double[n, n];
167         for (int i = 0; i < n; i++)
168         {
169             for (int j = 0; j < n; j++)
170             {
171                 if (i == j)
172                 {
173                     macierz[i, j] = 0;
174                 }
175                 else
176                 {
177                     double dx = punkty[i].X - punkty[j].X;
178                     double dy = punkty[i].Y - punkty[j].Y;
179                     macierz[i, j] = Math.Sqrt(dx * dx + dy * dy);
180                 }
181             }
182         }
184         return macierz;
185     }
187     static List<int> RozwiazTSP(double[,] A, int startIndex)
188     {
189         int n = A.GetLength(0);
190         List<int> trasa = new List<int> { startIndex };
191         bool[] odwiedzone = new bool[n];
192         odwiedzone[startIndex] = true;
194         double[] d = new double[n];
195         for (int i = 0; i < n; i++)
196             d[i] = (i == startIndex) ? double.PositiveInfinity : A[startIndex,
                    i];
198         while (trasa.Count < n)
199         {
200             int v = -1;
201             double maxD = double.MinValue;
202             for (int i = 0; i < n; i++)
203             {
204                 if (!odwiedzone[i] && d[i] > maxD)
205                 {
206                     maxD = d[i];
207                     v = i;
208                 }
209             }
211             if (v == -1) break;
213             double bestDelta = double.MaxValue;
```

```
214             int insertPos = 0;
215
216             for (int k = 0; k < trasa.Count; k++)
217             {
218                 int i = trasa[k];
219                 int j = (k + 1 == trasa.Count) ? trasa[0] : trasa[k + 1];
220
221                 double delta = A[i, v] + A[v, j] - A[i, j];
222                 if (delta < bestDelta)
223                 {
224                     bestDelta = delta;
225                     insertPos = k + 1;
226                 }
227             }
228
229             trasa.Insert(insertPos, v);
230             odwiedzone[v] = true;
231
232             for (int i = 0; i < n; i++)
233             {
234                 if (!odwiedzone[i])
235                     d[i] = Math.Min(d[i], A[v, i]);
236             }
237         }
238
239         trasa.Add(startIndex);
240         return trasa;
241     }
242
243     static void WyswietlMacierz(double[,] macierz, List<Punkt> punkty)
244     {
245         int n = macierz.GetLength(0);
246
247         Console.Write("       ");
248         for (int i = 0; i < n; i++)
249             Console.Write($"{punkty[i].Nazwa,-8}");
250         Console.WriteLine();
251
252         for (int i = 0; i < n; i++)
253         {
254             Console.Write($"{punkty[i].Nazwa,-6} ");
255             for (int j = 0; j < n; j++)
256             {
257                 Console.Write($"{macierz[i, j],6:F2} ");
258             }
259             Console.WriteLine();
260         }
261     }
262
263     static void WyswietlTrase(List<int> trasa, List<Punkt> punkty, double[,]
        macierz)
264     {
265         double totalDlugosc = 0;
266
267         Console.WriteLine("Optymalna kolejność wiercenia:");
```

```
268         for (int i = 0; i < trasa.Count - 1; i++)
269         {
270             int od = trasa[i];
271             int dokad = trasa[i + 1];
272             double odleglosc = macierz[od, dokad];
273             totalDlugosc += odleglosc;

275             Console.WriteLine($"{i + 1}. {punkty[od].Nazwa} -> {punkty[dokad].
                    Nazwa} ({odleglosc:F2})");
276         }

278         Console.WriteLine($"\nCałkowita długość trasy: {totalDlugosc:F2}");
279         Console.WriteLine($"Liczba punktów: {trasa.Count - 1}");
280     }
281 }
```

Listing 2: Implementacja algorytmu dla automatu wiertarskiego.

Opis implementacji dla automatu wiertarskiego

Program realizuje optymalizację trasy automatu wiertarskiego poprzez zastosowanie algorytmu dołączania najdalszego wierzchołka. Implementacja składa się z następujących elementów:

- **Struktura danych** - Klasa `Punkt` przechowuje współrzędne (X, Y), identyfikator oraz nazwę każdego punktu wiercenia
- **Wczytywanie danych** - Program oferuje trzy sposoby wprowadzania danych:
  - Wczytywanie z klawiatury - interaktywne wprowadzanie współrzędnych
  - Generowanie losowe - automatyczne tworzenie przykładowych danych testowych
  - Wczytywanie z pliku - import danych z pliku tekstowego
- **Obliczanie odległości** - Utworzenie macierzy odległości na podstawie wzoru euklidesowego:

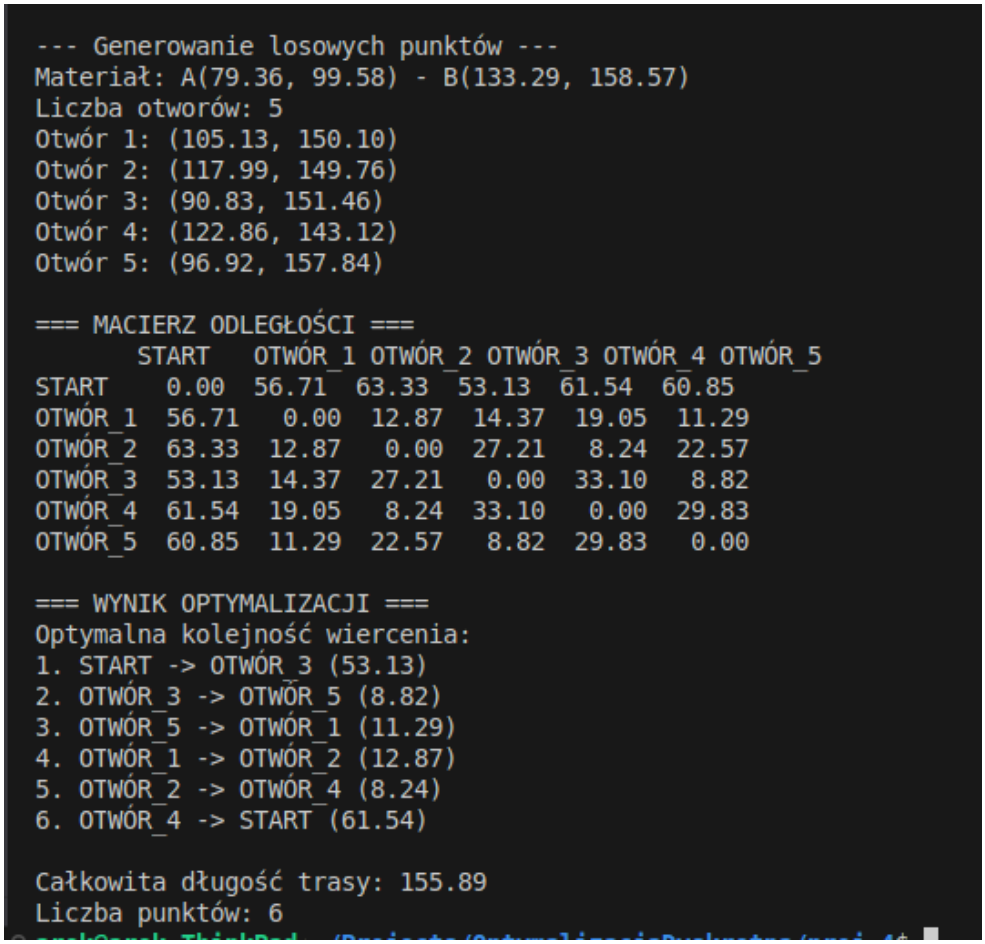
$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- **Algorytm optymalizacji** - Zastosowanie metody dołączania najdalszego wierzchołka z uwzględnieniem specyfiki problemu wiertarskiego
- **Punkt referencyjny** - Uwzględnienie punktu startowego A (x_0, y_0) jako początku i końca całej trasy wiercenia
- **Prezentacja wyników** - Wyświetlenie:
  - Optymalnej kolejności wiercenia otworów
  - Odległości między kolejnymi punktami
  - Całkowitej długości trasy
  - Liczby punktów do przetworzenia

Program umożliwia praktyczne zastosowanie algorytmu komiwojażera do optymalizacji pracy automatu wiertarskiego, minimalizując czas przejazdu narzędzia między otworami i zwiększając efektywność procesu produkcyjnego.

Wyniki

Dla przykładowych danych automatu wiertarskiego program wyznacza optymalną kolejność wiercenia otworów minimalizującą całkowitą drogę przejazdu narzędzia. Algorytm skutecznie rozwiązuje problem optymalizacji trasy dla automatu wiertarskiego, redukując czas wykonania operacji Rys. 2 (s. 11).



Rysunek 2: Wynik algorytmu dla automatu wiertarskiego