

Міністерство освіти і науки України  
Національний авіаційний університет  
Факультет кібербезпеки, комп'ютерної та програмної інженерії  
Кафедра комп'ютеризованих систем управління

Лабораторна робота № 1.4  
з дисципліни «Системи штучного інтелекту»  
на тему «Методи пошуку в просторі станів»

Виконала:  
студентка ФККПІ  
групи СП-425  
Ульич І. Г.  
Перевірила:  
Росінська Г. П.

Київ 2019

## 1. ЗАВДАННЯ РОБОТИ

Ознайомитись з методами пошуку розв'язків інтелектуальних задач у просторі станів. Вивчити блок-схеми алгоритмів для всіх методів пошуку.

## 2. ХІД РОБОТИ

Щоб виконати лабораторну роботу, необхідно розв'язати задачу комівояжера для графа, заданого за варіантом (рис. 1), тобто знайти такий найкоротший шлях, який відвідає усі точки графа і повернеться у початкову точку.

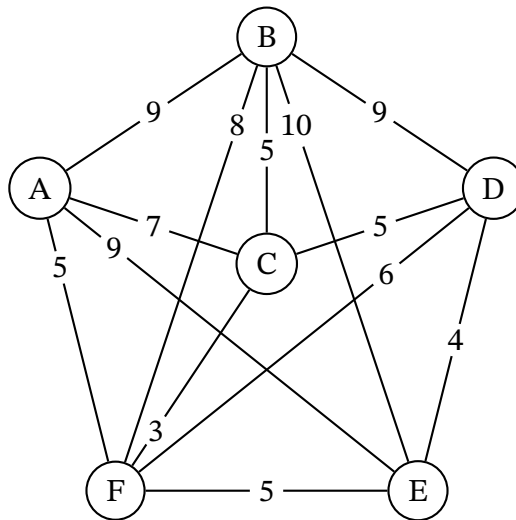


Рис. 1: Граф, який описує умову задачі комівояжера

Щоб розв'язати задачу, розроблюємо програму, яка шукатиме розв'язок задачі за допомогою методу повного перебору можливих розв'язків у просторі станів. Необхідно розв'язати задачу двома способами: перебор в ширину та в глибину. Розробивши програму (лістинг А.1), запускаємо її і спостерігаємо за результатом (рис. 2). Як бачимо, програма знайшла такий найкоротший шлях:

$$A \xrightarrow{9} B \xrightarrow{5} C \xrightarrow{5} D \xrightarrow{4} E \xrightarrow{5} F \xrightarrow{5} A = 33,$$

перебравши можливі розв'язки у просторі станів двома обраними способами (рис. 3, 4).

## 3. ВИСНОВОК

Виконуючи дану лабораторну роботу, ми дослідили процес породження простору станів і ознайомились із прикладами задач представлення у просторі станів,

```

4. bash
bash %1 bash %2
01-solution-ulchich $ python3 solution.py dfs
Solving using method: dfs
Shortest path: ('A', 'B', 'C', 'D', 'E', 'F', 'A') (33)
01-solution-ulchich $

```

а) Пошуквшир

```

4. bash
bash %1 bash %2
01-solution-ulchich $ python3 solution.py
usage: Solve a TSP problem using either DFS or BFS on solution tree.
[-h] [-p] method
Solve a TSP problem using either DFS or BFS on solution tree.: error: the follow
ing arguments are required: method
01-solution-ulchich $ python3 solution.py bfs
Solving using method: bfs
Shortest path: ('A', 'B', 'C', 'D', 'E', 'F', 'A') (33)
01-solution-ulchich $

```

б) Пошуквглиб

Рис. 2: Результат розв'язання задачі програмою

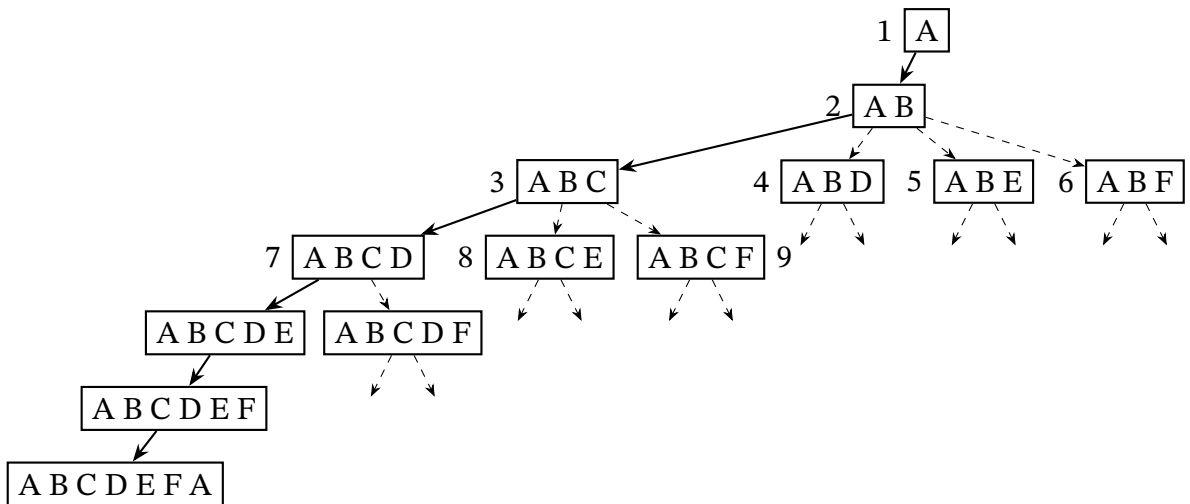


Рис. 3: Частина графа пошуку розв'язкувшир

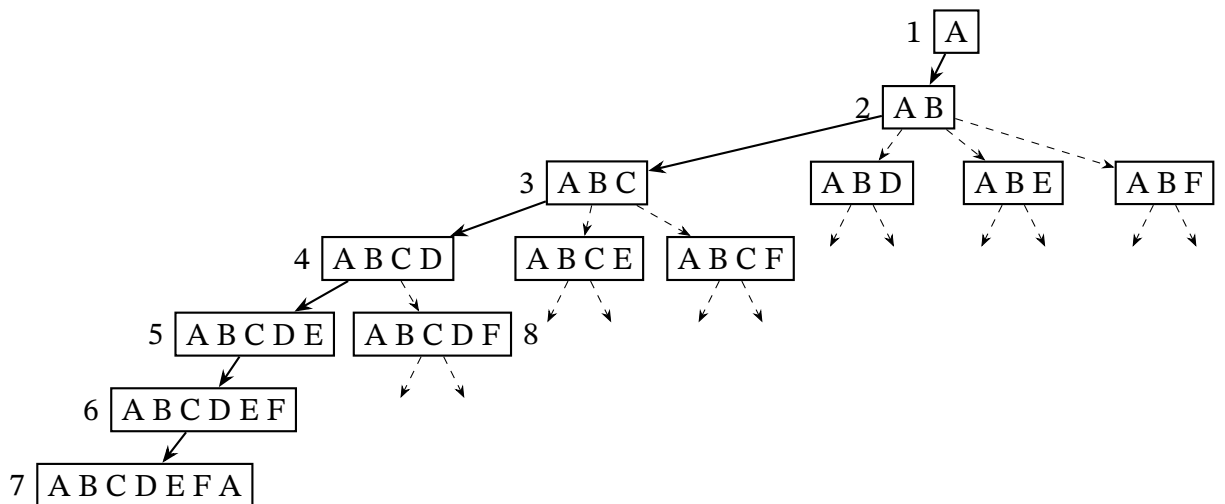


Рис. 4: Частина графа пошуку розв'язкувглиб

а також розробили програму для розв’язання задачі комівояжера методом повного перебору розв’язків у просторі станів.

## А. ПРОГРАМА ДЛЯ РОЗВ’ЯЗКУ ПОСТАВЛЕНОЇ ЗАДАЧІ

---

Лістинг А.1: Початковий код програмного модуля для розв’язання задачі комівояжера

---

```
1  import argparse
2  import itertools
3  import networkx as nx
4  import matplotlib.pyplot as plt
5
6  INF = 9999
7
8
9  class Path(object):
10
11      def __init__(self, path):
12          self.path = path
13
14      def __str__(self):
15          s = "".join(str(self.path))
16          return s
17
18      def __add__(self, other):
19          if not isinstance(other, Path):
20              raise TypeError(
21                  "Cannot add {} to Path"
22                  .format(type(other))
23              )
24
25          res = Path(
26              self.path + other.path
27          )
28          return res
29
30      def __hash__(self):
31          return hash(self.path)
32
33      def __eq__(self, other):
34          return self.path == other.path
35
36      def __len__(self):
37          return len(self.path)
38
```

```

39     @property
40     def name(self):
41         name = "".join(self.path)
42         return name
43
44     @property
45     def subpaths(self):
46         for idx in range(1, len(self.path) + 1):
47             yield Path(self.path[:idx])
48
49     @property
50     def edges(self):
51         subpaths = list(self.subpaths)
52         src_edge = subpaths[0]
53         for dst_edge in subpaths[1:]:
54             yield (src_edge, dst_edge)
55             src_edge = dst_edge
56
57     def calculate_cost(self, distance_matrix):
58         distance = 0
59         src_idx = self.path[0]
60         for dst_idx in self.path[1:]:
61             distance += distance_matrix[src_idx][dst_idx]
62             src_idx = dst_idx
63
64         return distance
65
66
67 class SearchTree(nx.Graph):
68
69     SOLUTION_METHODS = {
70         "bfs": nx.bfs_tree,
71         "dfs": nx.dfs_tree,
72     }
73
74     def __init__(self, distance_matrix, *args, **kwargs):
75
76         self.distance_matrix = distance_matrix
77         self.shortest_path_length = len(distance_matrix) + 1
78
79         super().__init__(*args, **kwargs)
80
81         self._build()
82
83     def _build(self):
84         nodes = list(self.distance_matrix.keys())
85

```

```

86     starting_vertex = Path((nodes[0],))
87     nodes = nodes[1:]
88     for permutation in itertools.permutations(nodes):
89         cur_path = starting_vertex + Path(permutation) +
            ↪ starting_vertex
90         self.add_path(cur_path)
91
92     def add_path(self, path):
93         for sp in path.subpaths:
94             self.add_node(sp)
95
96         for src, dst in path.edges:
97             self.add_edge(src, dst)
98
99     def show(self, *args, **kwargs):
100         nx.draw(self, *args, **kwargs)
101         plt.show()
102
103     def solve_tsp(self, starting_v, method="bfs", print_search=False):
104         print("Solving using method: {}".format(method))
105         try:
106             method = self.SOLUTION_METHODS[method]
107         except KeyError:
108             raise ValueError(
109                 "I don't know the solution method <{}>"
110                 .format(method)
111             )
112
113         if isinstance(starting_v, str):
114             starting_v = Path((starting_v,))
115
116         cur_min = INF
117
118         # Traverse the tree using the selected method
119         for n in method(self, starting_v):
120             # If the current path length is less than the required
121             ↪ minimum,
122             # continue the iteration. Used to skip short, unsuitable paths
123             if len(n) != self.shortest_path_length:
124                 continue
125
126             cur_cost = n.calculate_cost(self.distance_matrix)
127             if print_search:
128                 print(
129                     "Checking path: {} (Cost: {})"
130                     .format(

```

```

131         cur_cost
132     )
133 )
134     if cur_cost < cur_min:
135         cur_min = cur_cost
136         min_path = n
137
138     return min_path
139
140
141 def main(method, print_search=False, *args, **kwargs):
142     distances = {
143         "A": {"A": INF, "B": 4, "C": 3, "D": 7, "E": 4, "F": 2},
144         "B": {"A": 4, "B": INF, "C": 1, "D": 5, "E": 3, "F": 6},
145         "C": {"A": 3, "B": 1, "C": INF, "D": 3, "E": 1, "F": 4},
146         "D": {"A": 7, "B": 5, "C": 3, "D": INF, "E": 2, "F": 4},
147         "E": {"A": 4, "B": 3, "C": 1, "D": 2, "E": INF, "F": 2},
148         "F": {"A": 2, "B": 6, "C": 4, "D": 4, "E": 2, "F": INF},
149     }
150
151     search_tree = SearchTree(distances)
152
153     minpath = search_tree.solve_tsp(
154         starting_v="A",
155         method=method,
156         print_search=print_search,
157     )
158     minpathcost = minpath.calculate_cost(distances)
159
160     print(
161         "Shortest path: {} ({})"
162         .format(
163             minpath,
164             minpathcost,
165         )
166     )
167
168
169 if __name__ == "__main__":
170     parser = argparse.ArgumentParser(
171         "Solve a TSP problem using either DFS or BFS on solution tree."
172     )
173     parser.add_argument(
174         "method",
175         type=str,
176         default="bfs"
177     )

```

```
178     parser.add_argument(  
179         "-p",  
180         "--print-search",  
181         dest="print_search",  
182         action="store_true",  
183         help=(  
184             "Print the search process."  
185         )  
186     )  
187     parser.set_defaults(print_search=False)  
188  
189     args = parser.parse_args()  
190     method = args.method  
191     print_search = args.print_search  
192     main(  
193         method=method,  
194         print_search=print_search  
195     )
```

---