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

Лабораторна робота № 1.3
з дисципліни «Системи штучного інтелекту»
на тему «Представлення просторів станів за допомогою недетермінованих програм»

Виконав:
студент ФКСПІ
групи СП-425
Клокун В. Д.
Перевірила:
Росінська Г. П.

Київ 2019

1. Завдання роботи

Дослідити процес породження простору станів. Ознайомитись із прикладами задач представлення у просторі станів.

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.

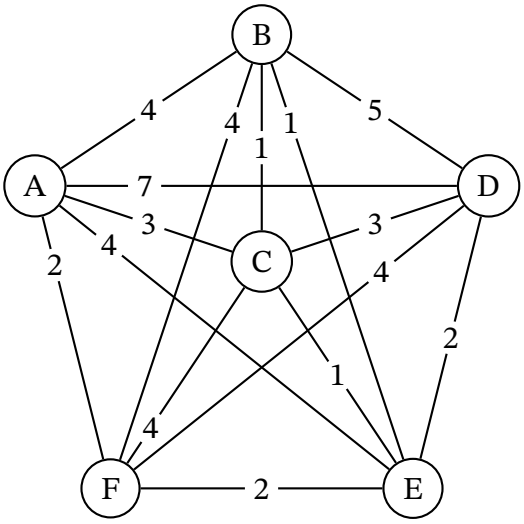

Рис. 1: Граф, який описує умову задачі комівояжера

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

$$A \overset{4}{\to} B \overset{1}{\to} C \overset{3}{\to} D \overset{2}{\to} E \overset{2}{\to} F \overset{2}{\to} A = 14,$$

перебравши можливі розв'язки у просторі станів (рис. 3).

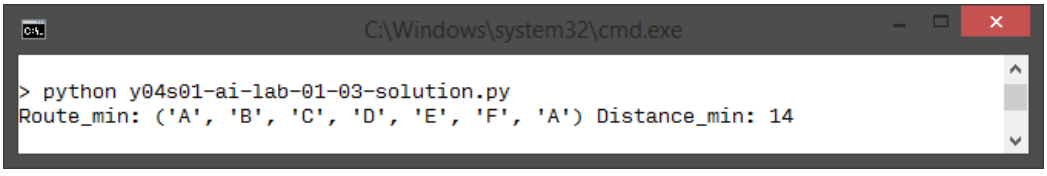

Рис. 2: Результат розв'язання задачі програмою

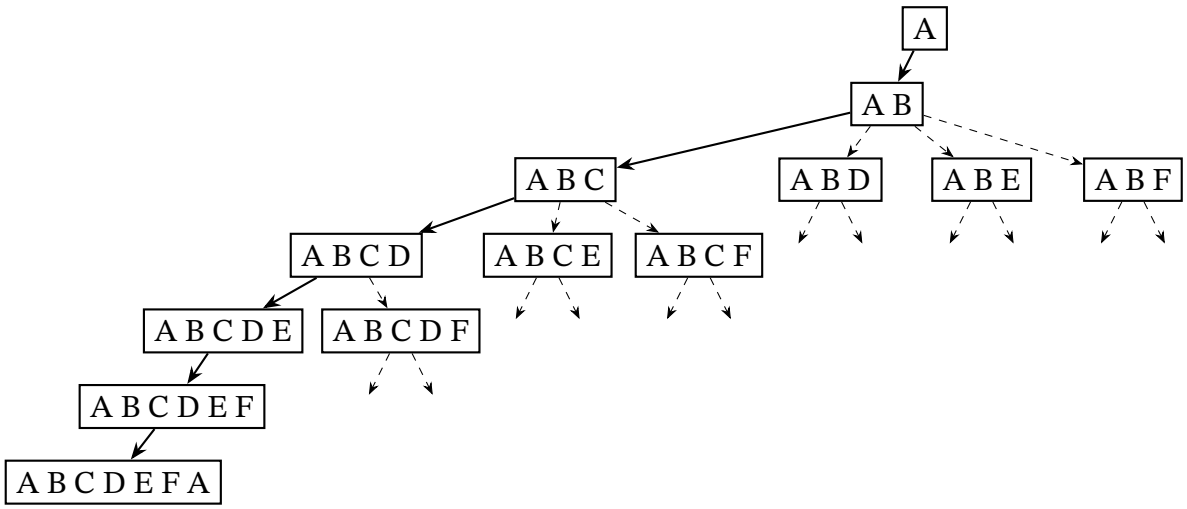

Рис. 3: Частина графа пошуку розв'язку для поставленої задачі комівояжера

3. Висновок

Виконуючи дану лабораторну роботу, ми дослідили процес породження простору станів і ознайомились із прикладами задач представлення у просторі станів, 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 itertools
2
3  INF = 9999
4
5
6  def calc_distance(route, distance_matrix):
7      distance = 0
8      src_idx = route[0]
9      for dst_idx in route[1:]:
10         distance += distance_matrix[src_idx][dst_idx]
11         src_idx = dst_idx
12
13     return distance
14
15 def main():
```

```
    distances = {
        "A": {"A": INF, "B":   4, "C":   3, "D":   7, "E":   4, "F":   2},
        "B": {"A":   4, "B": INF, "C":   1, "D":   5, "E":   3, "F":   6},
        "C": {"A":   3, "B":   1, "C": INF, "D":   3, "E":   1, "F":   4},
        "D": {"A":   7, "B":   5, "C":   3, "D": INF, "E":   2, "F":   4},
        "E": {"A":   4, "B":   3, "C":   1, "D":   2, "E": INF, "F":   2},
        "F": {"A":   2, "B":   6, "C":   4, "D":   4, "E":   2, "F": INF},
    }
    nodes = {**distances}
    nodes.pop("A")
    nodes = nodes.keys()

    distance_min = INF
    for permutation in itertools.permutations(nodes):
        # Always start and finish at node "A"
        route_cur = ("A", ) + permutation + ("A",)

        distance_cur = calc_distance(route_cur, distances)
        """
        print(
            "Route: {} Distance {}"
            .format(route_cur, distance_cur)
        )
        """
        if distance_cur < distance_min:
            distance_min = distance_cur
            route_min = route_cur

    print(
        "Route_min: {} Distance_min: {}"
        .format(route_min, distance_min)
    )

if __name__ == "__main__":
    main()
```