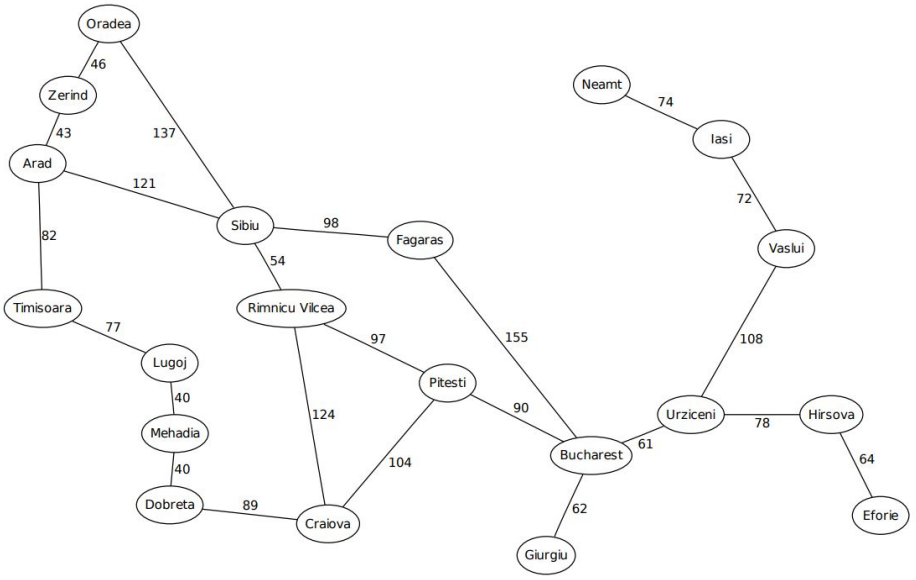
Графаар хайлт

Румины замын зураглалын дасгал

Румин улсын замын зураг графаар өгөгдсөн байя. Нэг хотоос нөгөө хот хүрэх маршрутад хайлт хийж үзье. Оролт нь хотын замын зураглал болон эхлэх хот, төгсгөл хот байна. Гаралт нь зорилго хотруу хүрэхэд дамжин өнгөрөх хотуудын дараалал мөн нийт туулсан замын уртын утга байна. Румин хотын замын зураглалыг графаар үзүүлэв.



Румин хотын замын зураглал

```
class GraphProblem(Problem):

    def __init__(self, initial, goal, graph):
        super().__init__(initial, goal)
        self.graph = graph

    def actions(self, A):
        return list(self.graph.get(A).keys())

    def result(self, state, action):
```

```
        return action

    def path_cost(self, cost_so_far, A, action, B):
        return cost_so_far + (self.graph.get(A, B) or np.inf)

    def find_min_edge(self):
        m = np.inf
        for d in self.graph.graph_dict.values():
            local_min = min(d.values())
            m = min(m, local_min)

        return m

    def h(self, node):
        locs = getattr(self.graph, 'locations', None)
        if locs:
            if type(node) is str:
                return int(distance(locs[node], locs[self.goal]))

            return int(distance(locs[node.state],
locs[self.goal]))
        else:
            return np.inf


class GraphProblemStochastic(GraphProblem):

    def result(self, state, action):
        return self.graph.get(state, action)

    def path_cost(self):
        raise NotImplementedError
```

```
class Graph:
```

```
    def __init__(self, graph_dict=None, directed=True):
        self.graph_dict = graph_dict or {}
        self.directed = directed
        if not directed:
            self.make_undirected()

    def make_undirected(self):
        for a in list(self.graph_dict.keys()):
            for (b, dist) in self.graph_dict[a].items():
                self.connect1(b, a, dist)

    def connect(self, A, B, distance=1):
        self.connect1(A, B, distance)
        if not self.directed:
            self.connect1(B, A, distance)

    def connect1(self, A, B, distance):
        self.graph_dict.setdefault(A, {})[B] = distance

    def get(self, a, b=None):
        links = self.graph_dict.setdefault(a, {})
        if b is None:
            return links
        else:
            return links.get(b)

    def nodes(self):
        s1 = set([k for k in self.graph_dict.keys()])
        s2 = set([k2 for v in self.graph_dict.values() for k2, v2
in v.items()])
        nodes = s1.union(s2)
        return list(nodes)
```

Өргөнөөр хайлтыг графаар хэрэгжүүлвэл дараах байдлаар хэрэгжүүлнэ.

```
def breadth_first_graph_search(problem):
  #Эхлээд хайлтын модны хамгийн гүехэн зангилааг хайх
```

```
    node = Node(problem.initial)
    if problem.goal_test(node.state):
        return node
    frontier = deque([node])
    explored = set()
    while frontier:
        node = frontier.popleft()
        explored.add(node.state)
        for child in node.expand(problem):
            if child.state not in explored and child not in frontier:
                if problem.goal_test(child.state):
                    return child
                frontier.append(child)
    return None
```

Гүнээр хайлтыг графаар хэрэгжүүлвэл дараах байдлаар хэрэгжүүлнэ.

```
def depth_first_graph_search(problem):
    #Эхлээд хайлтын модны хамгийн гүнзгий зангилааг хайх
    frontier = [(Node(problem.initial))]  # Stack

    explored = set()
    while frontier:
        node = frontier.pop()
        if problem.goal_test(node.state):
            return node
        explored.add(node.state)
        frontier.extend(child for child in node.expand(problem)
                        if child.state not in explored and child not in frontier)
    return None
```

Хотууд нь чиглэлгүй холбоосоор холбогдсон бөгөөд хот хоорондын замын уртыг оруулсан графыг дараах байдлаар кодлов. Графын ирмэг бүр хоёр тийшээ чиглэлтэй эсвэл чиглэлгүй байх граф байгуулна.

```
import numpy as np

def UndirectedGraph(graph_dict=None):
    return Graph(graph_dict=graph_dict, directed=False)

romania_map = UndirectedGraph(dict(
    Arad=dict(Zerind=75, Sibiu=140, Timisoara=118),
    Bucharest=dict(Urziceni=85, Pitesti=101, Giurgiu=90,
Fagaras=211),
    Craiova=dict(Drobeta=120, Rimnicu=146, Pitesti=138),
    Drobeta=dict(Mehadia=75),
    Eforie=dict(Hirsova=86),
    Fagaras=dict(Sibiu=99),
    Hirsova=dict(Urziceni=98),
    Iasi=dict(Vaslui=92, Neamt=87),
    Lugoj=dict(Timisoara=111, Mehadia=70),
    Oradea=dict(Zerind=71, Sibiu=151),
    Pitesti=dict(Rimnicu=97),
    Rimnicu=dict(Sibiu=80),
    Urziceni=dict(Vaslui=142)))
```

Графт суурилсан түвшинээр хайлтын аргыг ашиглан Румын замын зураглалаас Arad-aas Buharest хүртэл маршрутыг хайж олж замын өртгийг тооцоолох кодын загварчлалыг үзүүлэе.

```
def GraphSearch():
    problem = GraphProblem('Arad', 'Bucharest', romania_map)
    searcher = breadth_first_graph_search

    def do(searcher, problem):
        goal_node =searcher(problem)
        print('Search algorithm ', ‘breadth_first_graph_search’)
```

```
        print('List of nodes visited:', Node.solution(goal_node),
' path cost:', goal_node.path_cost)

    do(searcher, problem)

GraphSearch()
```

Румын замын зураглалаас Arad-aas Buharest хүртэл маршрут нь Arad->Sibiu->Fagaras->Buharest байна. Нийт замын өртөг нь 450 гарч байна.

```
Search algorithm  breadth_first_graph_search
List of nodes visited: ['Sibiu', 'Fagaras', 'Bucharest']  path
cost: 450
```

Дээрх маршрут хайлтыг графт суурилсан гүнээр хайх аргаар туршиж үзвэл Arad-aas Buharest хүртэл маршрут нь Arad->Timisoara>Lugoj>Mehadia->Drobeta->Craiova->Pitesti->Bucharest байна. Нийт замын өртөг нь 733 гарч байна.

Лабораторийн ажил 4

Өгөгдсөн үгийн нэг үсгийг сольж үг үүсгэх замаар зорилгын үгийг үүсгэнэ. Үүсэх үг нь толь бичигт байдаг буюу оршдог үг байх ёстой. Жишээ нь fool гэсэн үгийг sage гэдэг үг болтол хувиргах замыг графт суурилсан хайлтаар хэрэгжүүлнэ. Зорилгын үг хүрэх хамгийн богино замыг үзүүлнэ мөн графыг хэвлэж үзүүлнэ үү. Жишээ нь оюуна гэдэг үгээс ухаан гэдэг үгийг үүсгээрэй. Хугацаа 3 долоо хоног.