CSCI 3613 | Artificial Intelligence

2018 Fall

Nijat Mursali, Musadig Aliyev

Final Report

General information

A* (called as “A star”) search algorithm is one of the best algorithms for pathfinding and graph traversal. A* search algorithm is used to find the path from the starting node to the goal node with having as small cost as possible which means the distance should be minimum while using this algorithm. A* search algorithm uses two functions f(n) and g(n) which are the cost of distance and heuristic function. In our project we added approximately 30 cities into the graph/map to check the distance from one node/region to another.

What was attempted

As we mentioned in our initial report, we have used A* search algorithm in order to find the path from one node to another. First of all, g(n) should be equal to zero because no path is going through yet. If there’s no second node and if the first node is equal to second node the it should give zero because no path is detected. Then, we have to find the cost of A* by implementing g(n) + h(n) and add it to the queue.

AIProject.py

```
85      def yoluTap(self):
86          return self.yol
87
88
89      # Esas A star algoritmasi
90      def Astarfunksiyasi(self, bashlangic_noqte, son_noqte):
91          if not self.edgeolanlar:
92              print('Error: graph-da edge yoxdur!')
93          else:
94              # Her iki nodun olub olmamasin yoxlayir
95              if bashlangic_noqte in self.nodlar and son_noqte in self.nodlar:
96                  if bashlangic_noqte == son_noqte: # Eger eynidirse
97                      return 0
98
99                  queue = Stackadielementler() # queue duzeldir
100
101                 # "mesafe_vectoru" ve "pathquran" yolu yeniden qurmaq uchun ishlenir
102                 mesafe_vectoru, pathquran = {}, {}
103                 for node in self.nodlar:
104                     mesafe_vectoru[node.Nodaget()] = None # none ile initialize olun
105                     pathquran[node.Nodaget()] = None
106                 mesafe_vectoru[bashlangic_noqte.Nodaget()] = 0
107
108                 # ne qeder cost var onu hesableyir : g(n) + h(n)
109                 g_cost, h_cost = 0, bashlangic_noqte.HeuristikCostTap()
110                 Astarcost = g_cost + h_cost
111                 queue.elaveet((bashlangic_noqte, g_cost, h_cost), Astarcost)
112                 umumi_cost = None
113
114                 while True:
115
```

Then we added all the nodes for heuristic function which are as follows:

```
    # tests ...
    #Azerbaijani map
    #Naxchivan
    nodeSadarak = Node('Sadarak', 190)
    nodeSharur = Node('Sharur', 180)
    nodeBabak = Node('Babak', 170)
    nodeCulfa = Node('Culfa', 160)
    nodeOrdubad = Node('Ordubad', 130)
    nodeShahbuz = Node('Shahbuz', 135)

    nodeQazax = Node('Qazax', 454)
    nodeAgstafa = Node('Agstafa', 442)
    nodeTovuz = Node('Tovuz', 424)
    nodeShamkir = Node('Shamkir', 385)
    nodeGanja = Node('Ganja', 348)
    nodeYevlax = Node('Yevlax', 280)
    nodeAgdash = Node('Agdash', 265)
    nodeBarda = Node('Barda', 305)
    nodeTartar = Node('Tartar', 324)
    nodeGoychay = Node('Goycay', 253)
    nodeUcar = Node('Ucar', 235)
    nodeZardab = Node('Zardab', 266)
```

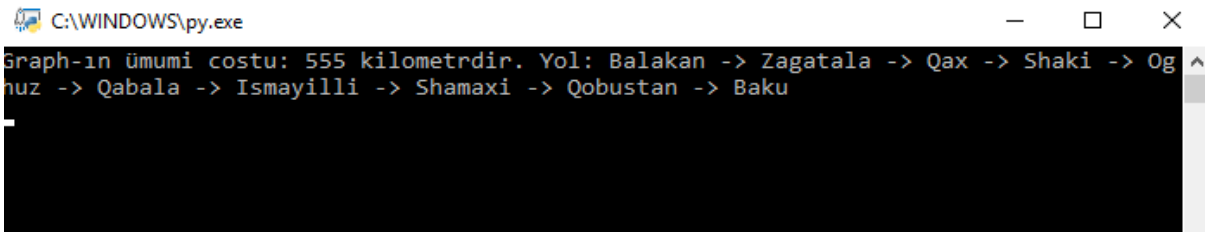
The string variable shows the name of region and integer variable is used for heuristic function. What “Node” function does is that it just selects the heuristic cost of the graph. Then using the graph, we added 55 nodes to the graph by using “EdgeElaveEt” function.

```
    graph.EdgeElaveEt(nodeSadarak, nodeSharur, 34)
    graph.EdgeElaveEt(nodeSharur, nodeBabak, 71)
    graph.EdgeElaveEt(nodeBabak, nodeShahbuz, 50)
    graph.EdgeElaveEt(nodeBabak, nodeCulfa, 32)
    graph.EdgeElaveEt(nodeCulfa, nodeOrdubad, 65)
```

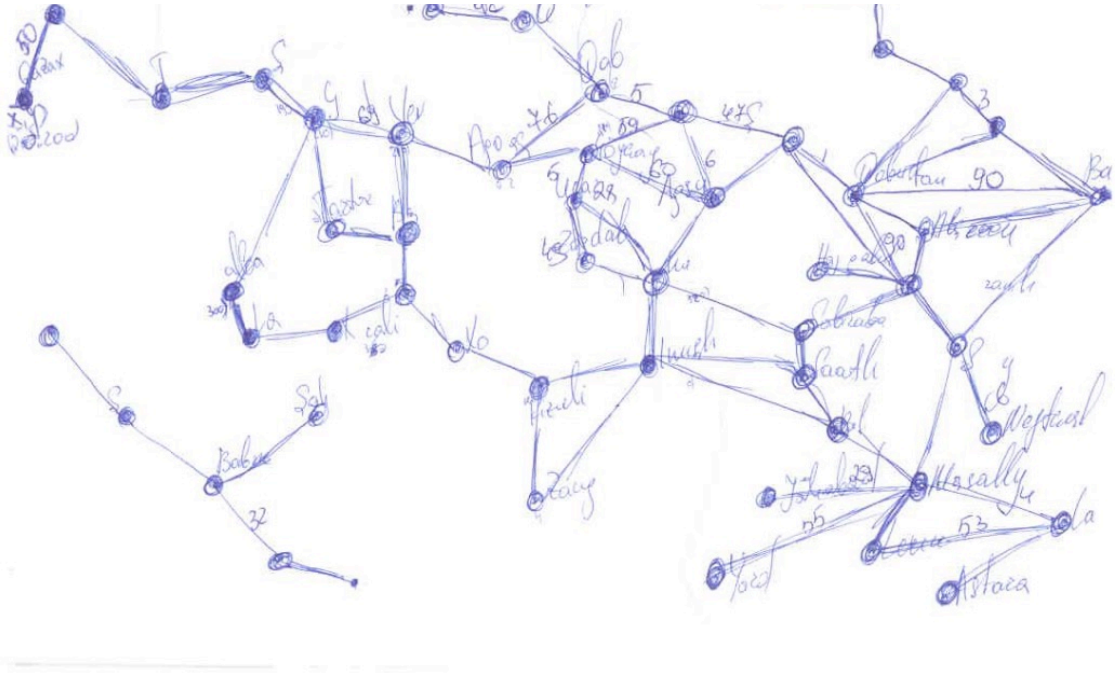
What was accomplished

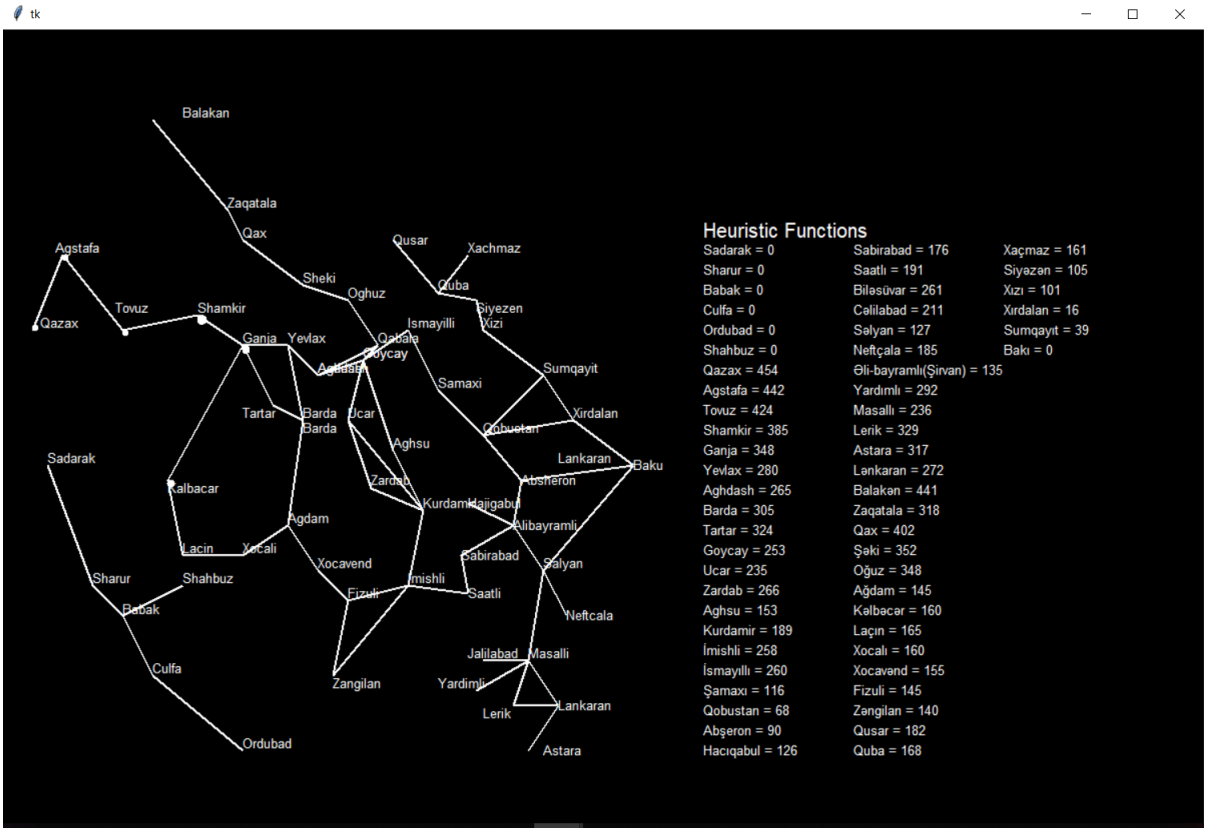
Initially, we created real-world application to apply A* search algorithm to Azerbaijan map. We have used A* search algorithm in order to find the shortest path between our Cities/Towns/Villages. Application display total cost of distance between cities and path

between initial and final regions in terminal. Input regions have to given via program. Fortunately, we found time to make program take cities as input from user.



We created GUI which includes Azerbaijan map and heuristic function:





Regions added to GUI using source and destination axis:

```
#Baku-Salyan
w.create_line(740,270,700,300,fill="white",width=2)
#Baku-Neftcala
w.create_line(700,300,730,340,fill="white",width=2)
#Salyan-Bilesuvar
w.create_line(700,300,670,380,fill="white",width=2)
#Bilesuvar-Celilabad
w.create_line(670,380,650,420,fill="white",width=2)
#Celilabad-Masalli
w.create_line(650,420,680,460,fill="white",width=2)
#Masalli-Lenkeran
w.create_line(680,460,710,500,fill="white",width=2)
#Lenkeran-Astara
w.create_line(710,500,720,570,fill="white",width=2)
```

What was learned

Initially we have learned from our research where A* search algorithm can be applied and which real-world application built using A* search algorithm. Then we learnt what exactly A* search algorithm is and how implement it in program. In addition to that, we learnt some Tkinter functionality to make GUI of our application. We are planning to add all cities/regions and improve functionality of program in the future phase of project.

Reference:

https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/

https://en.wikipedia.org/wiki/A*_search_algorithm

https://tkdocs.com/tutorial/