





پردیس دانشکده های فنی دانشگاه تهران

دانشکده‌ مهندسی نقشه برداری و اطلاعات مکانی

تمرین اول هندسه محاسباتی

کد نویسی A\*

استاد:

سرکار خانم دکتر زهرا بهرامیان

گروه هشتم:

کاروان جلالی ، فرشاد شمسی خانی

بهار 1404

**فهرست مطالب**

[**1) مقدمه** 4](#_Toc195879087)

[**2) هدف** 4](#_Toc195879088)

[**3) مراحل پیاده‌ سازی** 5](#_Toc195879089)

[**3- 1) وارد کردن کتابخانه ‌ها و خواندن داده ‌ها** 5](#_Toc195879090)

[**3- 2) ساخت گراف جاده ‌ها** 6](#_Toc195879091)

[**3- 3) دریافت ورودی از کاربر** 7](#_Toc195879092)

[**3- 4) پیاده‌ سازی الگوریتم A\*** 8](#_Toc195879093)

[**3- 5) ذخیره‌ سازی و ترسیم نتایج** 10](#_Toc195879094)

[**4) اجرای کد** 11](#_Toc195879095)

[**5) نتیجه‌ گیری** 12](#_Toc195879096)

[**6) منابع** 13](#_Toc195879097)

# **1) مقدمه**

الگوریتم A\* یکی از معروف‌ ترین و کاربردی ‌ترین الگوریتم ‌ها برای پیدا کردن کوتاه ‌ترین مسیر در گراف‌ ها است. این الگوریتم به طور گسترده در سیستم ‌های مسیریابی و جستجوی مسیر استفاده می‌ شود. هدف اصلی الگوریتم A\* پیدا کردن کوتاه‌ ترین مسیر از یک نقطه شروع به یک نقطه هدف در یک گراف است.

این الگوریتم به دو عامل مهم توجه می‌ کند:

1. هزینه واقعی (هزینه از نقطه شروع تا نقطه فعلی)، که با g(n) نمایان می ‌شود.
2. تخمین هزینه باقی ‌مانده تا مقصد، که با h(n) نشان داده می ‌شود.

ترکیب این دو عامل، یعنی f(n) = g(n) + h(n)، به الگوریتم کمک می‌ کند که در هر مرحله بهترین گزینه برای ادامه مسیر را انتخاب کند. این پروژه با استفاده از داده ‌های مکانی برای مدل ‌سازی شبکه جاده ‌ها و پیاده‌ سازی الگوریتم A\* انجام شده است تا کوتاه ‌ترین مسیر بین دو نقطه مشخص را محاسبه کند.

# **2) هدف**

هدف این پروژه، پیاده‌ سازی الگوریتم A\* برای مسیریابی در داده‌ های مکانی است. با استفاده از این الگوریتم، می ‌توان کوتاه‌ ترین مسیر بین دو نقطه را در شبکه جاده ‌ها پیدا کرد. این امر می ‌تواند در برنامه ‌های مسیریابی، تحلیل شبکه‌ های حمل ‌و نقل و برنامه‌ ریزی مسیرهای بهینه کاربرد داشته باشد.

# **3) مراحل پیاده‌ سازی**

در ادامه مراحل پیاده سازی به همراه کد ارائه خواهد گردید.

## **3- 1) وارد کردن کتابخانه ‌ها و خواندن داده ‌ها**

ابتدا، کتابخانه‌ های مورد نیاز مانند geopandas برای کار با داده ‌های مکانی، shapely برای کار با shapefile، matplotlib برای ترسیم نمودارها و networkx برای مدل ‌سازی گراف بارگذاری می ‌شوند. سپس، داده‌ های جاده‌ ها از فایل shapefile خوانده می ‌شوند.

|  |
| --- |
| *#------------------------Importing the libraries and reading the file----------------------*  *import* os  *import* geopandas *as* gpd  *#work with spatial data*  *from* shapely.geometry *import* Point, LineString   *#work with shapefiles*  *import* matplotlib  *#plot*  *import* matplotlib.pyplot *as* plt  *import* networkx *as* nx  *#creating graph*  matplotlib.use('TkAgg') *#matplot lib backend for showing plots*  *# reading shapefile using geopandas*  roads = gpd.read\_file("layers/roads.shp")  *# Check if the 'export' folder exists, if not, create it*  *if* not os.path.exists('Exports'):      os.makedirs('Exports') |

## **3- 2) ساخت گراف جاده ‌ها**

برای مدل کردن معابر به عنوان یک گراف، از کتابخانه networkx استفاده می ‌کنیم. در اینجا، برای هر خط در معابر (جاده ‌ها)، نود های گراف و یال ‌ها را ایجاد کرده و هزینه (یا وزن) هر یال را برابر با فاصله اقلیدسی بین دو نود می ‌گذاریم.

|  |
| --- |
| *#------------------------Creating graph--------------------------------*  *# creating empty graph using networkx*  roads\_graph = nx.Graph()  *# Loop over all rows of ((roads dataframe)) with iterrows from pandas for filling the graph*  *# each dataframe has indices and rows. rows has many columns include geometry*  *for* index, row *in* roads.iterrows():      line = row.geometry  *#creating linestring includes x,y coordinates as tuples for each line*      coords = list(line.coords) *#creating list of coordinates*  *for* j *in* range(len(coords) - 1):          u = coords[j]  *#first node of each line*          v = coords[j + 1] *#last node of each line*          length = Point(u).distance(Point(v)) *#calculating length using distance function from shaply library*          roads\_graph.add\_edge(u, v, weight=length) *#adding each edge to the graph* |

## **3- 3) دریافت ورودی از کاربر**

با استفاده از تعامل گرافیکی، دو نقطه توسط کاربر انتخاب می‌ شوند. این نقاط به نزدیک ‌ترین نودهای گراف نسبت داده می‌ شوند تا به‌ عنوان نقاط شروع و هدف برای الگوریتم A\* استفاده شوند.

|  |
| --- |
| *#-------------------getting the points from the user---------------------------*  *# function to find the nearest node of where the user clicked*  def find\_nearest\_node(point, nodes):      nearest\_node = None      min\_dist = float('inf')  *for* node *in* nodes: *#loop over all nodes to find the nearest node*          dist = Point(node).distance(point)  *if* dist < min\_dist:              min\_dist = dist              nearest\_node = node  *return* nearest\_node  *#empty list for saving clicks*  clicks = []  *#Function for Receiving two clicks from the user to select start and goal points*  def getpointbyclick(event):  *if* event.xdata is not None and event.ydata is not None:          clicks.append(Point(event.xdata, event.ydata))*#list of x,y tuples*          print(f"📍 selected point: ({event.xdata:.2f}, {event.ydata:.2f})")  *if* len(clicks) == 2:              plt.close()  fig, ax = plt.subplots(figsize=(8, 6))  roads.plot(ax=ax, color='#FFCC99')  plt.title("Specify Start and Goal points")  *#connecting click function to click event in figure window*  click\_event\_id = fig.canvas.mpl\_connect('button\_press\_event', getpointbyclick)  plt.show()  *if* len(clicks) < 2:      print("Two points not selected! The program stopped.")      exit()  *# Getting the closest nodes to selected points using find\_nearest\_node function*  start = find\_nearest\_node(clicks[0], roads\_graph.nodes)  goal = find\_nearest\_node(clicks[1], roads\_graph.nodes) |

## **3- 4) پیاده‌ سازی الگوریتم A\***

الگوریتم A\* با استفاده از دو تابع هزینه، g(n) که هزینه پیموده‌ شده تا نود n است و h(n) که برآوردی از هزینه باقی ‌مانده تا هدف است، اجرا می ‌شود. با استفاده از این توابع، کوتاه‌ترین مسیر بین نقطه شروع و هدف محاسبه می ‌شود.

|  |
| --- |
| *#------------------------Implementation of A\* algorithm---------------------------*  *#Definition of a heuristic function to calculate the Euclidean distance from the current node to the goal node*  def h(n):  *return* Point(n).distance(Point(goal))  def f(n):  *return* g[n] + h(n)  *#g[n] is the value for node n from the g dictionary*  *#A\* algorithm According to the PSEUDO CODE*  open\_list = [start] *#List of nodes that need to be evaluated*  closed\_list = [] *#LList of nodes that have already been evaluated*  g = {start: 0} *#Dictionary storing the cost to reach each node (g(n)). key:node value:cost*  previous\_nodes = {} *#dictionary contain nodes as keys parents as values*  *while* open\_list:  *# current node coords is node which f(n) is min*      current = min(open\_list, key=f)  *#apply f function on all nodes in open\_list*  *if* current == goal:          path = []  *#creating the path from end to begining*  *while* current in previous\_nodes:              path.append(current)              current = previous\_nodes[current]          path.append(start)          path.reverse()  *break*      open\_list.remove(current)      closed\_list.append(current)  *for* neighbor *in* roads\_graph.neighbors(current): *#loop over all neighbours of current neighbour using networkx.neighbors library*          cost = roads\_graph[current][neighbor]['weight'] *#returning the cost between current and all neighbours*          new\_g = g[current] + cost *#calculating new g(n)*  *# Get the cost of reaching the neighbor node, default to infinity if it's not found*          neighbor\_cost = g.get(neighbor, float('inf'))  *# Check if the neighbor is in open\_list and the new path is worse or equal*  *if* neighbor in open\_list and new\_g >= neighbor\_cost:  *continue*  *# Check if the neighbor is in closed\_list and the new path is worse or equal*  *if* neighbor in closed\_list and new\_g >= neighbor\_cost:  *continue*  *# If the neighbor is in open\_list, remove it for reevaluation*  *if* neighbor in open\_list:              open\_list.remove(neighbor)  *# If the neighbor is in closed\_list, remove it for reevaluation*  *if* neighbor in closed\_list:              closed\_list.remove(neighbor)          previous\_nodes[neighbor] = current *#Set the current node as the parent of the neighbor*          g[neighbor] = new\_g *# add the neighbor to the open\_list to evaluate it later*          open\_list.append(neighbor) *#Add the neighbor to the open\_list to evaluate it later*  *else*:      print("No path found.")      exit()  *#create linestrings from path(list of nodes)*  lines = []  *for* i *in* range(len(path) - 1):      lines.append(LineString([path[i], path[i + 1]]))  *#creating shape file*  shortest\_path = gpd.GeoDataFrame(geometry=lines, crs=roads.crs)  shortest\_path.to\_file("Exports/shortest\_path.shp")  *#calculating length*  total\_length = sum(line.length *for* line *in* lines)  print(f"Shortest path: {total\_length:.2f} meters") |

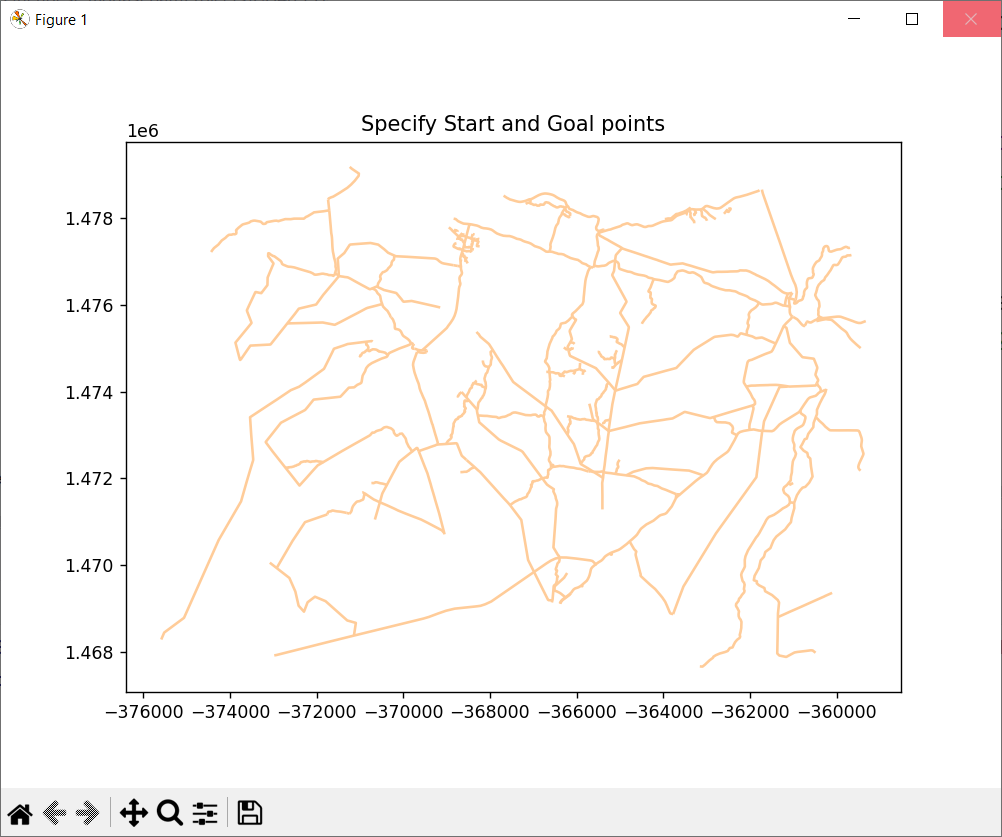
## **3- 5) ذخیره‌ سازی و ترسیم نتایج**

مسیر کوتاه‌ ترین به دست آمده در قالب یک shapefile ذخیره می ‌شود. همچنین، نقشه ‌ای از شبکه جاده ‌ها به همراه مسیر کوتاه ‌ترین و نقاط شروع و هدف ترسیم و ذخیره می ‌شود.

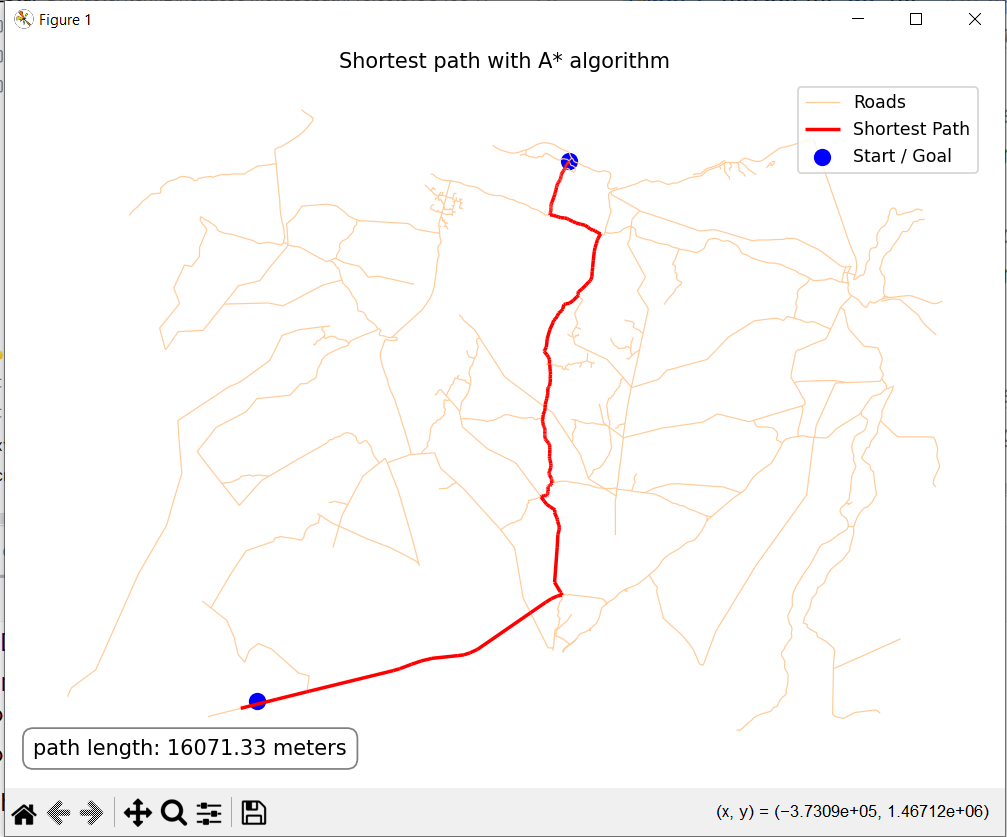
|  |
| --- |
| *#------------------------Plottings shapefiles----------------------------*  fig, ax = plt.subplots(figsize=(8, 6))  roads.plot(ax=ax, color='#FFCC99', linewidth=0.7, label='Roads')  shortest\_path.plot(ax=ax, color='red', linewidth=2, label='Shortest Path')  gpd.GeoSeries(clicks).plot(ax=ax, color='blue', markersize=80, label='Start / Goal')  plt.title("Shortest path with A\* algorithm")  plt.legend()  plt.text(      x=0.01, y=0.01,  *# Position the text at 1% from the left and 1% from the bottom of the axis*      s=f"path length: {total\_length:.2f} meters",  *# The text to display, including the formatted total path length*      transform=ax.transAxes,  *# Set the coordinate system to relative axis coordinates (0 to 1 range)*      fontsize=12,  *# Set the font size of the text*      color="black",  *# Set the text color to black*      bbox=dict(facecolor='white', edgecolor='gray', boxstyle='round,pad=0.5')  *# Add a white background box with rounded corners*  )  plt.axis('off')  plt.tight\_layout() *# Adjust the paddings*  plt.savefig("Exports/shortest\_path\_map.png", dpi=300)  plt.show() |

# **4) اجرای کد**

همان طور که مشاهده می کنید در ابتدا روی دو نقطه در دلخواه کلیک می کنیم، برنامه ابتدا نزدیک ترین نقاط از گره های معابر نسبت به نقاط وارد شده را یافته. نقاط جدید به عنوان نقاط شروع و پایان در نظر گرفته می شود و بهترین مسیر مطابق الگوریتم A\* پیدا می شود.



شکل 1. وارد کردن نقاط



شکل 2. یافتن مسیر به کمک الگوریتم A\*

# **5) نتیجه‌ گیری**

در این پروژه، الگوریتم A\* برای مسیریابی در داده‌ های مکانی پیاده‌ سازی شد. این الگوریتم با استفاده از داده‌ های شبکه جاده ‌ها و ورودی ‌های کاربر، کوتاه ‌ترین مسیر بین دو نقطه مشخص را محاسبه می ‌کند. نتایج به دست آمده می ‌تواند در برنامه‌ های مسیریابی، تحلیل شبکه‌ های حمل ‌و نقل و برنامه ‌ریزی مسیرهای بهینه کاربرد داشته باشد.

# **6) منابع**

* Wikipedia contributors. (2025). A\* search algorithm. Wikipedia. <https://en.wikipedia.org/wiki/A%2A_search_algorithm>
* Amit Patel. (2023). Introduction to A\* Pathfinding. Red Blob Games. <https://www.redblobgames.com/pathfinding/a-star/introduction.html>
* MDPI. (2015). Using the Hierarchical Pathfinding A\* Algorithm in GIS to Find Paths. ISPRS International Journal of Geo-Information, 2(4), 996-1015. https://www.mdpi.com/2220-9964/2/4/996