HW3 Report

By Clarence Jiang (uni: yj2737)

This assignment focuses on converting each building on Columbia Campus into a human description that should effectively help a visitor identify any of their interested buildings.

Step1:

In the first step, the inputs include a table that encodes each building with a number and a PGM campus image that stores the location of each building in a 2d map using the encoded number. That is to say, buildings like Pupin could be encoded as the number 9, so in the 2d map where Pupin stays, all pixels in the PGM campus will be written as 9. To read the encoded table, I used the typical "with" keyword in Python and stored the encoded relationship as a dictionary, called "encoded_dict". The encoded building number will serve as the key and the building name serves as the value. To read the PGM image, I used the "imageio" python package since it works well with the PGM format, and the 2d map is represented as a 2d numpy array, called "labeled_map". These 2 variables will serve as the main inputs for the following exploration.

Also, I will print out some important properties for each building. First, the building number and name could be accessed through the key-value pair of "encoded_dict". Second, the center of mass could be calculated by finding the indices of this building in the "labeled_map" through the "np. where(labeled_map == building number)" function and then taking the np. mean of row indices and column indices. The average of row indices and column indices serve as the y and x coordinates of the center mass respectively. Third, the total area in pixels could be extracted using the size of the result returned by "np. where" since this function will return a tuple of x and y indices. Fourth, to find the upper left and lower right corner of the Minimum Bounding Rectangle (MBR), I applied np. min and np. max on the row and column indices. For instance, the upper left coordinate is essentially (np. min(column indices), np. min(row indices)). Fifth, the diagonal length is calculated by the Euclidean distance between the upper left and lower right points using "np. sqrt()". Finally, before calculating the overlap situation of each building, I stored all the building properties in a big dictionary for further usage, called "building_num_collective_info_dict". The key is building numbers, and the values are the tuple of (building name, building area in pixels, diagonal length, (upper left, lower right), center of mass coordinate). I printed out the overlap information separately from the previous 5 properties because the overlap results are based on the previous 5 properties. Two buildings overlap if their minimum bounding rectangles overlap. I wrote a "cal_overlap()" function, where I essentially used a nested for loop to check every pair of buildings. There is an overlap between 2 buildings if the lower right corner of the MBR of the first building is greater than or equal to the upper left of the second building and the upper left corner is less than or equal to the lower right corner of the second building, or if it is the other way around switching building 1 and 2. Remember the upper left and lower right points are stored in my "building_num_collective_info_dict". I found "Uris" overlaps with "Mudd"; "Schermerhorn" overlaps with "Mudd"; "EarlHall" overlaps with

“Lewisohn”. Again, throughout this assignment, I realized a lot of information obtained in previous steps might be helpful for exploration in later steps, so I created tons of variables to store all intermediate results I found.

Building number: 9, building name: Pupin
Center of mass: (76.15304878048781, 14.976219512195122)
Total area: 1640
Upper left point coordinates: (39, 3)
Lower right point coordinates: (115, 27)
Diagonal length: 79.6994353806851

Building number: 19, building name: SchapiroCEPSR
Center of mass: (143.0, 20.0)
Total area: 1435
Upper left point coordinates: (123, 3)
Lower right point coordinates: (163, 37)
Diagonal length: 52.49761899362675

Building number: 28, building name: Mudd&EngTerrace&Fairchild&CS
Center of mass: (223.24181101011834, 35.2795403875836)
Total area: 5831
Upper left point coordinates: (166, 3)
Lower right point coordinates: (272, 86)
Diagonal length: 134.6291201783626

Building number: 38, building name: NorthwestCorner
Center of mass: (16.0, 40.5)
Total area: 1998
Upper left point coordinates: (3, 4)
Lower right point coordinates: (29, 77)
Diagonal length: 77.4919350642375

Building number: 47, building name: Uris
Center of mass: (142.52563879714933, 99.46532244046584)
Total area: 5753
Upper left point coordinates: (110, 48)
Lower right point coordinates: (175, 147)
Diagonal length: 118.43141475132347

Building number: 57, building name: Schermerhorn
Center of mass: (233.4469445154692, 120.51009971874201)
Total area: 3911
Upper left point coordinates: (181, 77)
Lower right point coordinates: (273, 147)
Diagonal length: 115.6027681329474

Building number: 66, building name: Chandler&Havemeyer
Center of mass: (37.53086078051481, 119.54248546913922)
Total area: 3613
Upper left point coordinates: (3, 81)
Lower right point coordinates: (80, 147)

Diagonal length: 101.41498903022176

Building number: 76, building name: OldComputerCenter
Center of mass: (96.5, 136.0)
Total area: 322
Upper left point coordinates: (90, 125)
Lower right point coordinates: (103, 147)
Diagonal length: 25.553864678361276

Building number: 85, building name: Avery
Center of mass: (204.04896907216494, 175.9819587628866)
Total area: 1164
Upper left point coordinates: (191, 151)
Lower right point coordinates: (215, 201)
Diagonal length: 55.46169849544819

Building number: 94, building name: Fayerweather
Center of mass: (259.6150592216582, 176.0)
Total area: 1182
Upper left point coordinates: (247, 151)
Lower right point coordinates: (272, 201)
Diagonal length: 55.90169943749474

Building number: 104, building name: Mathematics
Center of mass: (17.0, 182.0)
Total area: 1191
Upper left point coordinates: (3, 158)
Lower right point coordinates: (31, 206)
Diagonal length: 55.569775957799216

Building number: 113, building name: LowLibrary
Center of mass: (135.0, 221.5)
Total area: 3898
Upper left point coordinates: (101, 187)
Lower right point coordinates: (169, 256)
Diagonal length: 96.87620966986684

Building number: 123, building name: StPaulChapel
Center of mass: (226.5538178472861, 222.02115915363385)
Total area: 1087
Upper left point coordinates: (201, 210)
Lower right point coordinates: (251, 234)
Diagonal length: 55.46169849544819

Building number: 132, building name: EarlHall
Center of mass: (49.599472990777336, 221.86297760210803)
Total area: 759
Upper left point coordinates: (31, 211)
Lower right point coordinates: (68, 233)
Diagonal length: 43.04648650006177

Building number: 142, building name: Lewisohn9
Center of mass: (17.0, 259.0)
Total area: 1307

Upper left point coordinates: (3, 233)
Lower right point coordinates: (31, 285)
Diagonal length: 59.0592922409336

Building number: 151, building name: Philosophy
Center of mass: (258.3354838709677, 263.0)
Total area: 1085
Upper left point coordinates: (245, 240)
Lower right point coordinates: (272, 286)
Diagonal length: 53.33854141237835

Building number: 161, building name: Buell
Center of mass: (208.0, 253.5)
Total area: 340
Upper left point coordinates: (196, 246)
Lower right point coordinates: (220, 261)
Diagonal length: 28.30194339616981

Building number: 170, building name: AlmaMater
Center of mass: (136.0, 276.0)
Total area: 225
Upper left point coordinates: (129, 269)
Lower right point coordinates: (143, 283)
Diagonal length: 19.79898987322333

Building number: 179, building name: Dodge
Center of mass: (41.5, 301.09056603773587)
Total area: 1590
Upper left point coordinates: (3, 289)
Lower right point coordinates: (80, 311)
Diagonal length: 80.0812087820857

Building number: 189, building name: Kent
Center of mass: (233.0, 300.9)
Total area: 1470
Upper left point coordinates: (194, 290)
Lower right point coordinates: (272, 310)
Diagonal length: 80.52328855678958

Building number: 198, building name: CollegeWalk
Center of mass: (137.0, 322.5)
Total area: 4950
Upper left point coordinates: (0, 314)
Lower right point coordinates: (274, 331)
Diagonal length: 274.5268657162719

Building number: 208, building name: Journalism&Furnald
Center of mass: (30.58699808795411, 363.9441682600382)
Total area: 2615
Upper left point coordinates: (4, 338)
Lower right point coordinates: (81, 414)
Diagonal length: 108.18964830333815

Building number: 217, building name: Hamilton&Hartley&Wallach&JohnJay

Center of mass: (240.33885567890692, 416.9472245943638)
Total area: 5855
Upper left point coordinates: (191, 338)
Lower right point coordinates: (270, 490)
Diagonal length: 171.30382365843442

Building number: 236, building name: Lerner
Center of mass: (38.5, 446.5)
Total area: 2940
Upper left point coordinates: (4, 426)
Lower right point coordinates: (73, 467)
Diagonal length: 80.26207074328447

Building number: 246, building name: ButlerLibrary
Center of mass: (132.0, 460.4146156758803)
Total area: 5282
Upper left point coordinates: (85, 431)
Lower right point coordinates: (179, 490)
Diagonal length: 110.98198051936178

Building number: 255, building name: Carman
Center of mass: (38.5, 479.5)
Total area: 1540
Upper left point coordinates: (4, 469)
Lower right point coordinates: (73, 490)
Diagonal length: 72.12489168102785

Here is the overlap information:
Building Mudd&EngTerrace&Fairchild&CS has overlapped building ['Uris', 'Schermerhorn']
Building Uris has overlapped building ['Mudd&EngTerrace&Fairchild&CS']
Building Schermerhorn has overlapped building ['Mudd&EngTerrace&Fairchild&CS']
Building EarlHall has overlapped building ['Lewisohn9']
Building Lewisohn9 has overlapped building ['EarlHall']

Step 2:

In Step 2, my system outputs a “what” description that contains 3 aspects of information: size, aspect ratio, and geometry. Size includes smallest/small/medium-size/large/largest; aspect ratio includes narrow/medium-width/wide, and geometry includes square/rectangular/I-shaped/C-shaped/L-shaped/asymmetric.

Again, retrieving the shape description only requires the same inputs in Step 1. I used “area in pixels” as the metric to measure size. For each building, I calculated size using 2 threshold values 1000 and 2000. I got these 2 threshold values by observing the area distribution of all buildings. Remember that I have a dictionary that stores collective information for each building and “area” is included in it. The whole list is contained as a comment in my code, but it is clear that there are 2 places of big value jump: from 759 to 1085 and from 1998 to 2615. In other words, the distribution reveals that areas mostly fall in the range of 1000 to 2000, so it is

natural that I call these buildings “medium” size, areas less than 1000 “small”, and areas greater than 2000 “big”. I think my approach is reasonable since normally “medium-size” should be the category that appears the most, and “small/big” is more like the 2 extremes. Thus, finding the 2 threshold ends of the “medium-size” naturally makes everything into any of the 3 categories.

As for the aspect ratio, I applied a similar approach. I first printed out all aspect ratios. The aspect ratio is evaluated by the ratio of the smaller one over the greater one between height and width. If the aspect ratio is close to 1, it means width and height are somewhat similar, so it is a “wide” building. If it’s close to 0, it means one of the “height/width” is significantly smaller than the other. I calculated height and width using the upper left and lower right coordinates stored in my “collective_info” dictionary. Again, I printed out all aspect ratios and looked through the distribution. I first used 0.5 as a threshold; if the aspect ratio is less than or equal to 0.5, I defined the building as “narrow”. This value means that one of the sides is at least two times longer than the other side, and I define this as sufficiently “narrow”. Then I used “SchapiroCEPSR” and “LowLibrary” as 2 examples of wide buildings, which have 0.85 and 0.98 aspect ratios. However, I thought they are too perfect as examples of wide buildings, so I wanted to leave some room for the definition of “wide”, which drives me to choose 0.7 as another threshold. Then the medium-width building is defined with an aspect ratio between 0.5 and 0.7.

Finally, for geometry, my methodology is to check geometry in this order: Square, Rectangular, I-shaped, L-shaped, C-shaped, and asymmetric. In other words, if any building passes a test, then there is no need to go to the following stages. For example, if the system found a building to be Square, it will not go through the checking process of rectangular. First, for a “square/rectangular/I-shaped” building, its “area in pixels” will be really close to the area of the MBR. By contrast, an “L-shaped/C-shaped/asymmetric” building will have a lot of little “holes” in the MBR. Thus, I calculated this ratio of “area in pixel/ area of MBR” using the area, upper left, and lower right points in my collective dictionary. Then I performed a similar process of printing out the distribution of this ratio and find most buildings that look “square/rectangular/I-shaped” indeed have a high ratio above 0.83, such as “Schapiro” and “Uris”, so I used 0.83 as a threshold.

Then, to further distinguish “I-shaped”, since “I-shaped” is more different than the other two, I designed a complicated algorithm. “I” has two horizontal bars at the top and bottom and a skinny vertical bar in the middle. Imagine the skinny vertical bar of the letter “I” as a rectangle. I will find the upper left and lower right points of this rectangle. Then the difference between the x-coordinates of these 2 points is the width of this skinny body of “I”. Then, I get the x-coordinate of the upper left point and I will check if the x-coordinates are the same for every point with indices “upper left + k * width (where k from 1 to the height of the skinny body)”. Also, I will do a similar thing for the lower right point. If there is even one point that has a different x-coordinate, I strictly do not call it an "I", but it works well for Columbia Campus. Essentially, I am testing if the skinny vertical bar structure of “the letter” exists and if there is only one. This whole testing function is called “isIShape()” in my system.

Then, to distinguish a square and a rectangular, I will simply use the aspect ratio obtained previously. If a building is “wide”, it is called a square; otherwise, it is a rectangular building. Thus, the overall logic would be “area pixel/area MBR > 0.83 AND aspect ratio is not wide AND is not I-shape → Rectangular”, “area pixel/area MBR > 0.83 AND aspect ratio is wide AND is not I-shape → Square”, and finally “area pixel/area MBR > 0.83 AND is I-shape → I-shaped”.

If neither of the 3 conditions is satisfied, the system will run into a function called “isL_or_C_or_Asymmetric_Shape()”. L-shaped buildings look the same if reflected according to one of the 2 diagonals. One is the diagonal from upper left to lower right, which could be obtained through np. transpose(), while the other is from upper right to lower left, tested by “np.rot90(np.fliplr(sub_array), -1)”. After getting the reflected building shape, I will compare it with the original building shape and count the overlapped pixels. If the ratio of this count and the total number of pixels. called the “reflection score”, is greater than 0.8, this building is L-shaped. I got this number by checking the distribution of all buildings. In fact, the “Journalism” building is almost a perfect “L” and it has a high score above 0.9, but the other 2 buildings look L-shaped, “Chandler” and “Schermohorn” has an even lower score than “Mudd”. Thus, I know having a low ratio does not mean it is not “L-shaped”. Thus, I further calculated the max length of the continuous horizontal black gap (black color is the area without building pixels, represented as 0 in the 2d numpy array). If the ratio between this length and the width of the MBR is greater than 0.6 but less than or equal to 0.75, it is also an “L-shaped”. It makes sense because the upper right area of the letter “L” will have some long continuous horizontal black parts compared to an asymmetric building like “Mudd”, but at the same time, it should not be too high (> 0.75) because a “C-shaped” building like “Hamilton” will also have a continuous black gap with long length. In this case, I distinguish both L-shaped and C-shaped. Either “reflection score > 0.8” or “reflection score < 0.8 and 0.75 >= black line width/width > 0.6” gives an “L-shaped”. Then “reflection score < 0.8 and black line width/width > 0.75” gives a “C-shaped”. The building left will be asymmetric. Again the 0.6, 0.8, and 0.75 are chosen as thresholds because of distribution observation since I want my system to successfully recognize “Chandler” and “Schermerhorn” as “L-shaped” but not misclassify the C-shaped “Hamilton” or the asymmetric “Mudd” as “L-shaped”. All these threshold values are well-chosen to make this work.

The confusion part is easy because I also stored the 3 aspects in a dictionary, so I could just run a nested for loop that checks if any pair has the same 3 aspects. In fact, there is a lot of confusion. For example, Pupin can be confused with ['NorthwestCorner', 'Avery', 'Fayerweather', 'StPaulChapel', 'Dodge', 'Kent', 'Carman']. For the minimization process, I followed 2 general principles. First, any of the 3 aspects could be dropped if dropping it will make the “what” description unique. For instance, there is only 1 “asymmetric” building, so dropping size and aspect ratio is acceptable, as this building cannot be confused with any other building. Second, dropping any of the 3 aspects makes the “what” description all the same. For instance, all the 3 “I-shaped” buildings have the same size and aspect ratio, so keeping them does not help us further identify buildings as I-shaped, and we could just call them “I-shaped” buildings.

In addition, I stored the minimized description of this step in a dictionary called "step2_result"; it has the building name as the key and the string description as values. This dictionary will be helpful in step 5 when we put all things together.

building name: Pupin, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
building name: SchapiroCEPSR, size: medium-size, aspect_ratio: wide, geometry: Square
building name: Mudd&EngTerrace&Fairchild&CS, size: large, aspect_ratio: wide, geometry: asymmetric
building name: NorthwestCorner, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
building name: Uris, size: large, aspect_ratio: medium-width, geometry: Rectangular
building name: Schermerhorn, size: large, aspect_ratio: wide, geometry: L-shaped
building name: Chandler&Havemeyer, size: large, aspect_ratio: wide, geometry: L-shaped
building name: OldComputerCenter, size: small, aspect_ratio: medium-width, geometry: Rectangular
building name: Avery, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
building name: Fayerweather, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
building name: Mathematics, size: medium-size, aspect_ratio: medium-width, geometry: I-shaped
building name: LowLibrary, size: large, aspect_ratio: wide, geometry: Square
building name: StPaulChapel, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
building name: EarlHall, size: small, aspect_ratio: medium-width, geometry: Rectangular
building name: Lewisohn9, size: medium-size, aspect_ratio: medium-width, geometry: I-shaped
building name: Philosophy, size: medium-size, aspect_ratio: medium-width, geometry: I-shaped
building name: Buell, size: small, aspect_ratio: medium-width, geometry: Rectangular
building name: AlmaMater, size: smallest, aspect_ratio: wide, geometry: Square
building name: Dodge, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
building name: Kent, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
building name: CollegeWalk, size: large, aspect_ratio: narrow, geometry: Rectangular
building name: Journalism&Furnald, size: large, aspect_ratio: wide, geometry: L-shaped
building name: Hamilton&Hartley&Wallach&JohnJay, size: largest, aspect_ratio: medium-width, geometry: C-shaped
building name: Lerner, size: large, aspect_ratio: medium-width, geometry: Rectangular
building name: ButlerLibrary, size: large, aspect_ratio: medium-width, geometry: Rectangular
building name: Carman, size: medium-size, aspect_ratio: narrow, geometry: Rectangular
---
Pupin can be confused with ['NorthwestCorner', 'Avery', 'Fayerweather', 'StPaulChapel', 'Dodge', 'Kent', 'Carman']
NorthwestCorner can be confused with ['Pupin', 'Avery', 'Fayerweather', 'StPaulChapel', 'Dodge', 'Kent', 'Carman']
Uris can be confused with ['Lerner', 'ButlerLibrary']
Schermerhorn can be confused with ['Chandler&Havemeyer', 'Journalism&Furnald']
Chandler&Havemeyer can be confused with ['Schermerhorn', 'Journalism&Furnald']
OldComputerCenter can be confused with ['EarlHall', 'Buell']
Avery can be confused with ['Pupin', 'NorthwestCorner', 'Fayerweather', 'StPaulChapel', 'Dodge', 'Kent', 'Carman']
Fayerweather can be confused with ['Pupin', 'NorthwestCorner', 'Avery', 'StPaulChapel', 'Dodge', 'Kent', 'Carman']
Mathematics can be confused with ['Lewisohn9', 'Philosophy']
StPaulChapel can be confused with ['Pupin', 'NorthwestCorner', 'Avery', 'Fayerweather', 'Dodge', 'Kent', 'Carman']
EarlHall can be confused with ['OldComputerCenter', 'Buell']
Lewisohn9 can be confused with ['Mathematics', 'Philosophy']
Philosophy can be confused with ['Mathematics', 'Lewisohn9']
Buell can be confused with ['OldComputerCenter', 'EarlHall']
Dodge can be confused with ['Pupin', 'NorthwestCorner', 'Avery', 'Fayerweather', 'StPaulChapel', 'Kent', 'Carman']
Kent can be confused with ['Pupin', 'NorthwestCorner', 'Avery', 'Fayerweather', 'StPaulChapel', 'Dodge', 'Carman']

Journalism&Furnald can be confused with ['Schermerhorn', 'Chandler&Havemeyer']
Lerner can be confused with ['Uris', 'ButlerLibrary']
ButlerLibrary can be confused with ['Uris', 'Lerner']
Carman can be confused with ['Pupin', 'NorthwestCorner', 'Avery', 'Fayerweather', 'StPaulChapel', 'Dodge', 'Kent']

Minimization description Step 2:
Pupin is a medium-size Rectangular building
SchapiroCEPSR is a medium-size Square building
Mudd&EngTerrace&Fairchild&CS is a large asymmetric building
NorthwestCorner is a medium-size Rectangular building
Uris is a large medium-width building
Schermerhorn is a L-shaped building
Chandler&Havemeyer is a L-shaped building
OldComputerCenter is a small building
Avery is a medium-size Rectangular building
Fayerweather is a medium-size Rectangular building
Mathematics is a I-shaped building
LowLibrary is a large Square building
StPaulChapel is a medium-size Rectangular building
EarlHall is a small building
Lewisohn9 is a I-shaped building
Philosophy is a I-shaped building
Buell is a small building
AlmaMater is a smallest Square building
Dodge is a medium-size Rectangular building
Kent is a medium-size Rectangular building
CollegeWalk is a large Rectangular building
Journalism&Furnald is a L-shaped building
Hamilton&Hartley&Wallach&JohnJay is a largest C-shaped building
Lerner is a large medium-width building
ButlerLibrary is a large medium-width building
Carman is a medium-size Rectangular building

Step 3:

In Step 3, I performed a similar process as in Step 2. I need to convert the image location on the labeled map into a "where" description that involves 3 aspects: verticality, horizontality, and orientation. Verticality involves 5 options: uppermost, upper, mid-height, lower, and lowermost. Horizontality includes leftmost, left, mid-width, right, and rightmost. Orientation contains vertically-oriented, non-oriented, and horizontally-oriented.

As for horizontality and verticality, I simply divided the whole 495 * 275 map into 5 * 5 areas. The width and height are just equally divided into 5 sections, which correspond to the labels from leftmost to rightmost or uppermost to lowermost correspondingly. I just accessed the center of mass coordinates in my collective information dictionary and used the x and y coordinates of the center of mass to assign horizontality and verticality labels. It worked pretty

well since Columbia Campus does not have too many clustered blocks. As for orientation, I first directly retrieved the aspect ratio of each building from my "step2_result" dictionary. If it's a wide building, it indicates it is likely a square, so it is not oriented. Then for those narrow buildings, I calculated the width and height, and if the height is greater than the width, it is a vertically-oriented building, and vice versa for horizontally-oriented buildings. Again, I stored this "where" information in a dictionary where the building name and a dictionary of {horizontality, verticality, orientation} serve as key and value respectively.

In Step 3, to find confusion, I applied the same methodology and just replaced the dictionary in Step 2 with my Step 3 dictionary. I discovered that "LowLibrary's location could be confused with "AlmaMater" and "Lerner" could be confused with "Carman". As for the minimization step, I just applied the same idea in Step 2. I discovered that the Campus is designed in a way to avoid clustered buildings. Thus, if imagining the Campus being divided into 5 * 5 little areas, there is likely only 1 building in each little area based on the center of mass, so you essentially can remove most of the orientation information. There are definitely exceptions like in the mid-height on the right, there are both "St.Paul" and "Avery" which can only be distinguished by their orientation. Thus, I carefully designed different cases with a lot of if-else statements (details included in my code implementation). Likewise, the minimized location description is stored in a "step3_result" dictionary.

building name: Pupin, horizontality: left, verticality: uppermost, orientation: horizontally-oriented
building name: SchapiroCEPSR, horizontality: mid-width, verticality: uppermost, orientation: non-oriented
building name: Mudd&EngTerrace&Fairchild&CS, horizontality: rightmost, verticality: uppermost, orientation: non-oriented
building name: NorthwestCorner, horizontality: leftmost, verticality: uppermost, orientation: vertically-oriented
building name: Uris, horizontality: mid-width, verticality: upper, orientation: vertically-oriented
building name: Schermerhorn, horizontality: rightmost, verticality: upper, orientation: non-oriented
building name: Chandler&Havemeyer, horizontality: leftmost, verticality: upper, orientation: non-oriented
building name: OldComputerCenter, horizontality: left, verticality: upper, orientation: vertically-oriented
building name: Avery, horizontality: right, verticality: upper, orientation: vertically-oriented
building name: Fayerweather, horizontality: rightmost, verticality: upper, orientation: vertically-oriented
building name: Mathematics, horizontality: leftmost, verticality: upper, orientation: vertically-oriented
building name: LowLibrary, horizontality: mid-width, verticality: mid-height, orientation: non-oriented
building name: StPaulChapel, horizontality: rightmost, verticality: mid-height, orientation: horizontally-oriented
building name: EarlHall, horizontality: leftmost, verticality: mid-height, orientation: horizontally-oriented
building name: Lewisohn9, horizontality: leftmost, verticality: mid-height, orientation: vertically-oriented
building name: Philosophy, horizontality: rightmost, verticality: mid-height, orientation: vertically-oriented
building name: Buell, horizontality: right, verticality: mid-height, orientation: horizontally-oriented
building name: AlmaMater, horizontality: mid-width, verticality: mid-height, orientation: non-oriented
building name: Dodge, horizontality: leftmost, verticality: lower, orientation: horizontally-oriented
building name: Kent, horizontality: rightmost, verticality: lower, orientation: horizontally-oriented
building name: CollegeWalk, horizontality: mid-width, verticality: lower, orientation: horizontally-oriented
building name: Journalism&Furnald, horizontality: leftmost, verticality: lower, orientation: non-oriented
building name: Hamilton&Hartley&Wallach&JohnJay, horizontality: rightmost, verticality: lowermost, orientation: vertically-oriented
building name: Lerner, horizontality: leftmost, verticality: lowermost, orientation: horizontally-oriented

building name: ButlerLibrary, horizontality: mid-width, verticality: lowermost, orientation: horizontally-oriented
building name: Carman, horizontality: leftmost, verticality: lowermost, orientation: horizontally-oriented
---
LowLibrary can be confused with ['AlmaMater']
AlmaMater can be confused with ['LowLibrary']
Lerner can be confused with ['Carman']
Carman can be confused with ['Lerner']

Minimization description Step 3:
Pupin is at uppermost and left in Columbia Campus
SchapiroCEPSR is at uppermost and mid-width in Columbia Campus
Mudd&EngTerrace&Fairchild&CS is at uppermost and rightmost in Columbia Campus
NorthwestCorner is at uppermost and leftmost in Columbia Campus
Uris is at mid-width and upper in Columbia Campus
Schermerhorn is a building at rightmost and upper and it's non-oriented in Columbia Campus
Chandler&Havemeyer is a building at leftmost and upper and it's non-oriented in Columbia Campus
OldComputerCenter is at left and it's vertically-oriented in Columbia Campus
Avery is at right and it's vertically-oriented in Columbia Campus
Fayerweather is a building at rightmost and upper and it's vertically-oriented in Columbia Campus
Mathematics is a building at leftmost and upper and it's vertically-oriented in Columbia Campus
LowLibrary is at mid-height and mid-width in Columbia Campus
StPaulChapel is a building at rightmost and mid-height and it's horizontally-oriented in Columbia Campus
EarlHall is a building at leftmost and mid-height and it's horizontally-oriented in Columbia Campus
Lewisohn9 is a building at leftmost and mid-height and it's vertically-oriented in Columbia Campus
Philosophy is a building at rightmost and mid-height and it's vertically-oriented in Columbia Campus
Buell is at right and it's horizontally-oriented in Columbia Campus
AlmaMater is at mid-height and mid-width in Columbia Campus
Dodge is a building at leftmost and lower and it's horizontally-oriented in Columbia Campus
Kent is a building at rightmost and lower and it's horizontally-oriented in Columbia Campus
CollegeWalk is at mid-width and lower in Columbia Campus
Journalism&Furnald is a building at leftmost and lower and it's non-oriented in Columbia Campus
Hamilton&Hartley&Wallach&JohnJay is at rightmost and lowermost in Columbia Campus
Lerner is at leftmost and lowermost in Columbia Campus
ButlerLibrary is at mid-width and lowermost in Columbia Campus
Carman is at leftmost and lowermost in Columbia Campus

Step 4:

In Step 4, I created a 26 by 26 2d numpy array. Each pair of buildings is represented as row and column indices; the row represents the target and the column represents the source. So position (0, 1) is interpreted as we are interested in the first building and plan to use the second building as a source to locate the first building. The value of this matrix could only be 0 or 1, and 1 means “near to”.

To evaluate if one building is near to the other, I first checked their horizontality and verticality. If both of horizontality and verticality of 2 buildings are off by two levels or more (for example, “left” and “leftmost” is off by 1 level, “leftmost” and “mid-width” are off by 2

levels), there are no need to do any calculation, and I will just assign a 0. I do not use an “or” but use an “and” relationship here, because using a less strict exclusion criterion could avoid accidentally removing some actual near cases in a diagonal direction. Then for the rest of the pair, I calculated the Euclidean distance between their center of masses and took the ratio of this distance over the smaller area of the pair, called “distance_size_ratio”, since dimension also plays a role. Also, my system does not make this symmetric. The fact that Building A is near Building B does not imply the reverse. For instance, you can say Almamater is near Low Library since Low Library is such a big building, so it makes sense. However, it will sound weird if you say Low Library is near Almamater because Almamater is such a small thing that does not serve well as a source. As a result, I designed my logic using 3 variables: the distance, the distance_size ratio, and the area of the pair of buildings. There are also 3 cases. First, if the area of the target building is less than ⅛ of that of the source building, the distance needs to be less than or equal to 80 for them to be called “near”. This is the case when the target building is really small. The threshold value needs to be smaller as well. This threshold of 80 is selected to help guarantee that small target buildings like “EarlHall” and “OldComputerCenter” have a source. Second, if the distance is less than or equal to 120 and the distance_size ratio is less than or equal to 0.06, this pair is also considered “near”. These 2 thresholds are based on the “Pupin” and “Schapiro” pair which represents the most standard “near” relationship. This is the case for normal building pairs. Third, if the area of the source building is greater than or equal to ⅛ of the target building, it means the source building is not that small, so the system will just use the distance as the sole metric. If it is greater than or equal to 60, it will assign a 1 to the building pair. This condition is to satisfy those cases that the distance_size_ratio is not small enough because the area is not big enough but not as small as typical small buildings like “OldComputerCenter”, such as “St.Paul”. The rest of the cases are not considered as near. Finally, we could easily use the binary matrix and iterate row by row to print the result for each building serving as the target and column by column for each building serving as the source.

To solve the confusion part of this step, I utilized the lambda function and max/min function to calculate the source/target element that has the most/least target/source buildings. For minimization, I iterated through each building and also utilized a similar lambda function and max() to find the source that appears the most in other buildings. Besides, I also used 2 dictionaries to store the source and target buildings for each building that acts as a target/source and another dictionary to store the landmark choice for each building.

Pupin as target, it is near to ['SchapiroCEPSR', 'NorthwestCorner']
SchapiroCEPSR as target, it is near to ['Pupin', 'Mudd&EngTerrace&Fairchild&CS', 'Uris']
Mudd&EngTerrace&Fairchild&CS as target, it is near to ['SchapiroCEPSR', 'Uris', 'Schermerhorn']
NorthwestCorner as target, it is near to ['Pupin', 'Chandler&Havemeyer']
Uris as target, it is near to ['SchapiroCEPSR', 'Mudd&EngTerrace&Fairchild&CS', 'Schermerhorn', 'Chandler&Havemeyer']
Schermerhorn as target, it is near to ['Mudd&EngTerrace&Fairchild&CS', 'Uris', 'Avery', 'Fayerweather']
Chandler&Havemeyer as target, it is near to ['NorthwestCorner', 'Uris', 'Mathematics']
OldComputerCenter as target, it is near to ['Uris', 'Chandler&Havemeyer']

Avery as target, it is near to ['Schermerhorn', 'Fayerweather', 'StPaulChapel']
Fayerweather as target, it is near to ['Schermerhorn', 'Avery', 'StPaulChapel']
Mathematics as target, it is near to ['Chandler&Havemeyer', 'EarlHall']
LowLibrary as target, it is near to ['CollegeWalk']
StPaulChapel as target, it is near to ['Avery', 'Fayerweather', 'Philosophy', 'Buell']
EarlHall as target, it is near to ['Mathematics', 'Lewisohn9']
Lewisohn9 as target, it is near to ['EarlHall', 'Dodge']
Philosophy as target, it is near to ['StPaulChapel', 'Buell', 'Kent']
Buell as target, it is near to ['LowLibrary', 'StPaulChapel', 'Philosophy', 'Kent']
AlmaMater as target, it is near to ['LowLibrary', 'CollegeWalk']
Dodge as target, it is near to ['Lewisohn9', 'Journalism&Furnald']
Kent as target, it is near to ['Philosophy', 'Buell']
CollegeWalk as target, it is near to ['LowLibrary', 'Journalism&Furnald']
Journalism&Furnald as target, it is near to ['Dodge', 'CollegeWalk', 'Lerner']
Hamilton&Hartley&Wallach&JohnJay as target, it is near to ['ButlerLibrary']
Lerner as target, it is near to ['Journalism&Furnald', 'ButlerLibrary', 'Carman']
ButlerLibrary as target, it is near to ['Hamilton&Hartley&Wallach&JohnJay', 'Lerner']
Carman as target, it is near to ['Lerner']

Pupin as source, source buildings close to ['SchapiroCEPSR', 'NorthwestCorner']
SchapiroCEPSR as source, source buildings close to ['Pupin', 'Mudd&EngTerrace&Fairchild&CS', 'Uris']
Mudd&EngTerrace&Fairchild&CS as source, source buildings close to ['SchapiroCEPSR', 'Uris', 'Schermerhorn']
NorthwestCorner as source, source buildings close to ['Pupin', 'Chandler&Havemeyer']
Uris as source, source buildings close to ['SchapiroCEPSR', 'Mudd&EngTerrace&Fairchild&CS', 'Schermerhorn', 'Chandler&Havemeyer', 'OldComputerCenter']
Schermerhorn as source, source buildings close to ['Mudd&EngTerrace&Fairchild&CS', 'Uris', 'Avery', 'Fayerweather']
Chandler&Havemeyer as source, source buildings close to ['NorthwestCorner', 'Uris', 'OldComputerCenter', 'Mathematics']
OldComputerCenter as source, source buildings close to []
Avery as source, source buildings close to ['Schermerhorn', 'Fayerweather', 'StPaulChapel']
Fayerweather as source, source buildings close to ['Schermerhorn', 'Avery', 'StPaulChapel']
Mathematics as source, source buildings close to ['Chandler&Havemeyer', 'EarlHall']
LowLibrary as source, source buildings close to ['Buell', 'AlmaMater', 'CollegeWalk']
StPaulChapel as source, source buildings close to ['Avery', 'Fayerweather', 'Philosophy', 'Buell']
EarlHall as source, source buildings close to ['Mathematics', 'Lewisohn9']
Lewisohn9 as source, source buildings close to ['EarlHall', 'Dodge']
Philosophy as source, source buildings close to ['StPaulChapel', 'Buell', 'Kent']
Buell as source, source buildings close to ['StPaulChapel', 'Philosophy', 'Kent']
AlmaMater as source, source buildings close to []
Dodge as source, source buildings close to ['Lewisohn9', 'Journalism&Furnald']
Kent as source, source buildings close to ['Philosophy', 'Buell']
CollegeWalk as source, source buildings close to ['LowLibrary', 'AlmaMater', 'Journalism&Furnald']
Journalism&Furnald as source, source buildings close to ['Dodge', 'CollegeWalk', 'Lerner']
Hamilton&Hartley&Wallach&JohnJay as source, source buildings close to ['ButlerLibrary']
Lerner as source, source buildings close to ['Journalism&Furnald', 'ButlerLibrary', 'Carman']
ButlerLibrary as source, source buildings close to ['Hamilton&Hartley&Wallach&JohnJay', 'Lerner']
Carman as source, source buildings close to ['Lerner']

Uris as source, has the most targets ['SchapiroCEPSR', 'Mudd&EngTerrace&Fairchild&CS', 'Schermerhorn', 'Chandler&Havemeyer', 'OldComputerCenter']
OldComputerCenter as source, has the least targets []
Uris as target, has the most sources ['SchapiroCEPSR', 'Mudd&EngTerrace&Fairchild&CS', 'Schermerhorn', 'Chandler&Havemeyer']
LowLibrary as target, has the least sources ['CollegeWalk']

Minimization Step 4
Pupin as a target, the most-often-used source is SchapiroCEPSR
SchapiroCEPSR as a target, the most-often-used source is Uris
Mudd&EngTerrace&Fairchild&CS as a target, the most-often-used source is Uris
NorthwestCorner as a target, the most-often-used source is Chandler&Havemeyer
Uris as a target, the most-often-used source is Schermerhorn
Schermerhorn as a target, the most-often-used source is Uris
Chandler&Havemeyer as a target, the most-often-used source is Uris
OldComputerCenter as a target, the most-often-used source is Uris
Avery as a target, the most-often-used source is Schermerhorn
Fayerweather as a target, the most-often-used source is Schermerhorn
Mathematics as a target, the most-often-used source is Chandler&Havemeyer
LowLibrary as a target, the most-often-used source is CollegeWalk
StPaulChapel as a target, the most-often-used source is Avery
EarlHall as a target, the most-often-used source is Mathematics
Lewisohn9 as a target, the most-often-used source is EarlHall
Philosophy as a target, the most-often-used source is StPaulChapel
Buell as a target, the most-often-used source is StPaulChapel
AlmaMater as a target, the most-often-used source is LowLibrary
Dodge as a target, the most-often-used source is Journalism&Furnald
Kent as a target, the most-often-used source is Philosophy
CollegeWalk as a target, the most-often-used source is LowLibrary
Journalism&Furnald as a target, the most-often-used source is CollegeWalk
Hamilton&Hartley&Wallach&JohnJay as a target, the most-often-used source is ButlerLibrary
Lerner as a target, the most-often-used source is Journalism&Furnald
ButlerLibrary as a target, the most-often-used source is Lerner
Carman as a target, the most-often-used source is Lerner

Step 5:

In Step 5, I try to group all results from previous steps to develop a final human description of a building. First, I will extract the landmark choice for each building from Step 4, then I will simply concatenate the “what_T, where_T, what_S, where_S” result from my “step2_result” and “step3_result” dictionaries.

I discovered that my system will not have any confusion when concatenating the “what” and “where” for a target building. This is intuitive. Remember that in Step 3, I have only 2 pairs of confusion cases if I just applied the “where” information and these 2 pairs of buildings exactly have different “what” information. So my system will have no confusion even without the source “nearTo” information.

However, there is definitely more space for minimization. To some extent, my target "what + where" is already enough, so I could reduce some less important features in the source building. I took 3 main principles. First, Geometry is the most unique "what" property of a building. If the source building is "I", "L", "C" or "Square", I will remove all the other information about the building. Again this is because my system has really strict algorithms to generate these 4 shapes due to the strict thresholds, so the source building, which theoretically is already extraneous, will be fine with just its geometry. The second principle is that there is only one asymmetric building, so if the target is asymmetric, I will just describe it as an asymmetric building. Finally, the last principle is for the last geometry: Rectangular. Unlike the other 5 shapes, which I have discussed in the first and second principles, “Rectangular” is not a geometry that has many characteristics. Thus, to further minimize description, I should not think from using the perspective of geometry anymore, so I just removed the location information of the source building (horizontality and vertically). It is not that useful because the source is always near the target, so visitors could just look around. That is to say, I already have a precise description of the “where” of my target information; recall that there are only 2 pairs of buildings that have confused “where”. Thus, removing the “where” of the source building will not affect much because of both the good accuracy of the “where” of the target building and my strict definition of “near”.

Finally, for finding the shortest and longest unambiguous description, I put all the keywords in a list and count the total occurrences. The longest description has 7 keywords for “StPaulChapel”, while the shortest has 5 for a total of 12 buildings (listed in code, e.g. Pupin and Uris). I think this is a good result because I intentionally set up my ideal threshold to be 5 as the lowest. As Professor points out, people like redundant information. I incorporated the concern that less information might negatively affect a person’s locating process. From a real-life perspective, I do not want to throw away too many keywords, and I believe giving more is always better than not giving enough.

Total description without minimization:
a medium-size Rectangular building at uppermost and left near a medium-size Square building at uppermost and mid-width
a medium-size Square building at uppermost and mid-width near a large medium-width building at mid-width and upper
a large asymmetric building at uppermost and rightmost near a large medium-width building at mid-width and upper
a medium-size Rectangular building at uppermost and leftmost near a L-shaped building non-oriented at leftmost and upper
a large medium-width building at mid-width and upper near a L-shaped building non-oriented at rightmost and upper
a L-shaped building non-oriented at rightmost and upper near a large medium-width building at mid-width and upper
a L-shaped building non-oriented at leftmost and upper near a large medium-width building at mid-width and upper
a small building vertically-oriented at left near a large medium-width building at mid-width and upper
a medium-size Rectangular building vertically-oriented at right near a L-shaped building non-oriented at rightmost and upper
a medium-size Rectangular building vertically-oriented at rightmost and upper near a L-shaped building non-oriented at rightmost and upper
a I-shaped building vertically-oriented at leftmost and upper near a L-shaped building non-oriented at leftmost and upper
a large Square building at mid-height and mid-width near a large Rectangular building at mid-width and lower
a medium-size Rectangular building horizontally-oriented at rightmost and mid-height near a medium-size Rectangular building vertically-oriented at right
a small building horizontally-oriented at leftmost and mid-height near a I-shaped building vertically-oriented at leftmost and upper

a I-shaped building vertically-oriented at leftmost and mid-height near a small building horizontally-oriented at leftmost and mid-height
a I-shaped building vertically-oriented at rightmost and mid-height near a medium-size Rectangular building horizontally-oriented at rightmost and mid-height
a small building horizontally-oriented at right near a medium-size Rectangular building horizontally-oriented at rightmost and mid-height
a smallest Square building at mid-height and mid-width near a large Square building at mid-height and mid-width
a medium-size Rectangular building horizontally-oriented at leftmost and lower near a L-shaped building non-oriented at leftmost and lower
a medium-size Rectangular building horizontally-oriented at rightmost and lower near a I-shaped building vertically-oriented at rightmost and mid-height
a large Rectangular building at mid-width and lower near a large Square building at mid-height and mid-width
a L-shaped building non-oriented at leftmost and lower near a large Rectangular building at mid-width and lower
a largest C-shaped building at rightmost and lowermost near a large medium-width building at mid-width and lowermost
a large medium-width building at leftmost and lowermost near a L-shaped building non-oriented at leftmost and lower
a large medium-width building at mid-width and lowermost near a large medium-width building at leftmost and lowermost
a medium-size Rectangular building at leftmost and lowermost near a large medium-width building at leftmost and lowermost
---
Confusion:
Ok, there is no confusion

Minimization Step 5:
a medium-size Rectangular building at uppermost and left near a Square building
a medium-size Square building at uppermost and mid-width near a large medium-width building
a asymmetric building near a large medium-width building at mid-width and upper
a medium-size Rectangular building at uppermost and leftmost near a L-shaped building
a large medium-width building at mid-width and upper near a L-shaped building
a L-shaped building non-oriented at rightmost and upper near a large medium-width building
a L-shaped building non-oriented at leftmost and upper near a large medium-width building
a small building vertically-oriented at left near a large medium-width building
a medium-size Rectangular building vertically-oriented at right near a L-shaped building
a medium-size Rectangular building vertically-oriented at rightmost and upper near a L-shaped building
a I-shaped building vertically-oriented at leftmost and upper near a L-shaped building
a large Square building at mid-height and mid-width near a large Rectangular building
a medium-size Rectangular building horizontally-oriented at rightmost and mid-height near a medium-size building vertically-oriented
a small building horizontally-oriented at leftmost and mid-height near a I-shaped building
a I-shaped building vertically-oriented at leftmost and mid-height near a small building horizontally-oriented
a I-shaped building vertically-oriented at rightmost and mid-height near a medium-size building horizontally-oriented
a small building horizontally-oriented at right near a medium-size building horizontally-oriented
a smallest Square building at mid-height and mid-width near a Square building
a medium-size Rectangular building horizontally-oriented at leftmost and lower near a L-shaped building
a medium-size Rectangular building horizontally-oriented at rightmost and lower near a I-shaped building
a large Rectangular building at mid-width and lower near a Square building
a L-shaped building non-oriented at leftmost and lower near a large Rectangular building
a largest C-shaped building at rightmost and lowermost near a large medium-width building
a large medium-width building at leftmost and lowermost near a L-shaped building
a large medium-width building at mid-width and lowermost near a large medium-width building
a medium-size Rectangular building at leftmost and lowermost near a large medium-width building
---
Longest unambiguous description has count 7
The building is StPaulChapel
Shortest unambiguous description has count 5
The building is Pupin

The building is Mudd&EngTerrace&Fairchild&CS
The building is NorthwestCorner
The building is Uris
The building is OldComputerCenter
The building is Avery
The building is Mathematics
The building is EarlHall
The building is Buell
The building is AlmaMater
The building is CollegeWalk
The building is Lerner

```
import imageio.v2 as imageio
import numpy as np
import math
import itertools
from collections import Counter

# dimensions of the Campus Map
height = 495
width = 275
# Used in step 1, key: coordinates of upper left and lower right as dictionary key,  value: building name
# this is used to find overlapped buildings in the "cal_overlap" function
building_coordinate_name_dict = {}

# Used in step 1, key: building name, value: a list of overlapped building name
overlap_dict = {}

# Used in almost all steps,  key: building number, value = (building name, area, diagonal length,
# (upper_left_coordinate, lower_right_coordinate), center of mass coordinate)
building_num_collective_info_dict = {}

# # Used in step2 , key: building name
# value: a dictionary that describes its size, aspect ratio, and geometry
building_name_shape_description = {}

# the confusion dictionary in step 2, key: building name,  value: a list of other building names
# that each building could be confused with from a what perspective
confusion_dictionary_what = {}

# this is the result in Step 3, key: building name, value: a dictionary that stores {size, aspect ratio, geometry}
building_name_location_description = {}

# the confusion dictionary in step 3, key: building name,  value: a list of other building names
# that each building could be confused with from a where perspective
confusion_dictionary_where = {}

# this is the result in Step 4, key: building name as target, value: a list of source building names
buildings_nearBy_target = {}

# this is the result in Step 4, key: building name as source, value: a list of target building names
buildings_nearBy_source = {}

# this is the final printed result for step 2, key: building name, value: the minimized description printed out
step2_result = {}

# this is the final printed result for step 3, key: building name, value: the minimized description printed out
step3_result = {}

# this is the final printed result for step 4, key: building name, value: the minimized description printed out
step4_result = {}

# this is the printed result for step 5 without minimization, key: building name, value: what_T + where_T + what_S +
# where_S based on the variables of step2_result, step3_result, and step4_result
step5_result = {}

# this is the minimized version of step 5 result, key: building name, value: final version of description
step5_result_minimized = {}

# the confusion dictionary in step 5, key: building name, value: a list of building name it could be confused with.
confusion_dict_step5 = {}


def read_image():
    # read in the pgm image as a 2d numpy array, which will be used as a main input for following steps
    image = imageio.imread("Labeled.pgm")
    return image


def read_table():
    # read the Table.txt into a dictionary form, where the building number serves as key and building name as value,
    # which will be used as a main input for following steps
    table_result = {}
    with open('Table.txt', 'r') as file:
        lines = file.readlines()

    for line in lines:
        line = line.rstrip().strip()
        a = line.split()
        table_result[a[0]] = a[1]
    return table_result


def cal_overlap():
    """
    If the lower right corner of the MBR of the first building is greater than or equal to the upper left of the second
    building and the upper left corner is less than or equal to the lower right corner of the second building,
    or if it is the other way around switching building 1 and 2, then there is an overlap between these two buildings.
    :return:
    """
    for t in building_coordinate_name_dict:
        upper_left, lower_right = t
        for compared_t in building_coordinate_name_dict:
            if compared_t != t:
                upper_left2, lower_right2 = compared_t
                if (lower_right[0] >= upper_left2 [0] and lower_right[1] >= upper_left2 [1]
                        and upper_left[0] <= lower_right2 [0] and upper_left[1] <= lower_right2 [1]) or \
                    (lower_right[0] <= upper_left2[0] and lower_right[1] <= upper_left2[1]
                    and upper_left[0] >= lower_right2[0] and upper_left[1] >= lower_right2[1]):
                        overlap_dict[building_coordinate_name_dict[t]].append(building_coordinate_name_dict[compared_t])


def step1(encode_dict, labeled_map):
    """
    Iterate through each building
    1. Print center of mass
    2. Print total area
    3. Print upper left and lower right coordinates
    4. Print diagonal length

    Then perform the cal_overlap() to fill in the overlap_dict
    Iterate through each building in overlap_dict
        print out the overlapped buildings for each building

    :param encode_dict:
    :param labeled_map:
    :return:
    """
    print("Step 1")
    for building in encode_dict:
```

```
        print(f"Building number: {building}, building name: {encode_dict[building]}")

        building_num = int(building)
        # retrieve a target area
        target_area_info = np.where(labeled_map == building_num)
        # calculate center of mass
        print(f"Center of mass: ( {np.mean(target_area_info[1])}, {np.mean(target_area_info[0])} )")
        # calculate the number of pixels as the size of the building
        area = target_area_info[0].size
        print(f"Total area: {area}")
        # calculate the upper left and lower right coordinates
        upper_left_coordinate = (np.min(target_area_info[1]), np.min(target_area_info[0]))
        lower_right_coordinate = (np.max(target_area_info[1]), np.max(target_area_info[0]))
        # a dictionary that uses the tuple of the upper left and lower right points as the key
        # and the building name as values, this is used for calculating overlap.
        building_coordinate_name_dict[(upper_left_coordinate, lower_right_coordinate)] = encode_dict[building]
        # initialize empty list for each building to store overlapped buildings
        overlap_dict[encode_dict[building]] = []
        print(f"Upper left point coordinates: ({upper_left_coordinate[0]}, {upper_left_coordinate[1]} )")
        print(f"Lower right point coordinates: ({lower_right_coordinate[0]}, {lower_right_coordinate[1]} )")
        # calculate the diagonal length
        diagonal_length = np.sqrt( (np.max(target_area_info[1]) - np.min(target_area_info[1])) *
                                   (np.max(target_area_info[1]) - np.min(target_area_info[1])) +
                                   (np.max(target_area_info[0]) - np.min(target_area_info[0])) *
                                   (np.max(target_area_info[0]) - np.min(target_area_info[0])) )
        print(f"Diagonal length: {diagonal_length}")
        # store all information in a dictionary, where building number is the key
        # and a bunch of collective information as the value, which will be used later
        building_num_collective_info_dict[building_num] = (encode_dict[building], area, diagonal_length,
                                                           (upper_left_coordinate, lower_right_coordinate),
                                                           (np.mean(target_area_info[1]), np.mean(target_area_info[0]))
                                                           )
        print("-"* 50)
    print()
    print(f"Here is the overlap information: ")
    # This is the function to calculate overlap, after running this function, a global variable called "overlap_dict"
    # will be filled in, and I will just print out information from that dictionary.
    cal_overlap()
    for key, val in overlap_dict.items():
        if val:
            print(f"Building {key} has overlapped building {val}")
    print("-" * 50)


def get_size(building):
    """

    :param building: a building number as input
    :return: the size label of this particular building

    My approaching is first printing out all the areas of all buildings and take a look at the distribution after
    sorting. The result is like this [225, 322, 340, 759, 1085, 1087, 1164, 1182, 1191, 1307, 1435, 1470, 1540, 1590,
    1640, 1998, 2615, 2940, 3613, 3898, 3911, 4950, 5282, 5753, 5831, 5855]. To distinguish what is defined as small
    or big, it is important to find a threshold area that distinguish two groups of areas, where each group has similar
    values. It is obvious there is a group of area less than 1000, which I defined as small, and it is obvious there is
    another group that has significantly larger area greater than 2000. In other words, I found the threshold values
    by looking at the difference in the sorted area. I see there is a big jump from 1998 to 2615 as compared to 2615 to
    2940 or 1640 to 1998, so this big jump point is a good place to separate 2 labels. This is the methodology I took.
    """
    area = building_num_collective_info_dict[building][1]
    area_list = [v[1] for v in building_num_collective_info_dict.values()]
    max_area = max(area_list)
    min_area = min(area_list)
    if area == min_area:
        return "smallest"
    if area == max_area:
        return "largest"

    if area < 1000:
        return "small"
    elif area < 2000:
        return "medium-size"
    else:
        return "large"

    # list.sort()
    # print(list)


def get_aspect_ratio(building):
    """
    Threshold <= 0.5 narrow, medium-width <= 0.7 wide > 0.7

    I first used 0.5 as a threshold; if the aspect ratio is less than or equal to 0.5,
    I defined the building as “narrow”. This value means that one of the sides is at least two times longer
    than the other side, and I think this is sufficiently “narrow”.
    Then I used “SchapiroCEPSR” and “LowLibrary” as 2 examples of wide buildings, which have 0.85 and 0.98 aspect ratios.
    However, I thought these 2 buildings are too perfect as examples of wide buildings in my mind, so I wanted to
    leave some room for the definition of “wide”, which drives me to choose 0.7 as another threshold.
    Then the medium-width building is defined with an aspect ratio between 0.5 and 0.7.


    :param building: a building number as input
    :return: the aspect_ratio label of this particular building

    [('Pupin', 0.3157894736842105), ('SchapiroCEPSR', 0.85), ('Mudd&EngTerrace&Fairchild&CS', 0.7830188679245284),
    ('NorthwestCorner', 0.3561643835616438), ('Uris', 0.6565656565656566), ('Schermerhorn', 0.7608695652173914),
    ('Chandler&Havemeyer', 0.8571428571428571), ('OldComputerCenter', 0.5909090909090909), ('Avery', 0.48),
    ('Fayerweather', 0.5), ('Mathematics', 0.5833333333333334), ('LowLibrary', 0.9855072463768116),
    ('StPaulChapel', 0.48), ('EarlHall', 0.5945945945945946), ('Lewisohn', 0.5384615384615384),
    ('Philosophy', 0.5869565217391305), ('Buell', 0.625), ('AlmaMater', 1.0), ('Dodge', 0.2857142857142857),
    ('Kent', 0.2564102564102564), ('CollegeWalk', 0.06204379562043796), ('Journalism&Furnald', 0.987012987012987),
    ('Hamilton&Hartley&Wallach&JohnJay', 0.5197368421052632), ('Lerner', 0.5942028985507246),
    ('ButlerLibrary', 0.6276595744680851), ('Carman', 0.30434782608695654)]

    """

    # aspect_ratio_list = [(v[0], min((v[3][1][0]-v[3][0][0]), (v[3][1][1]-v[3][0][1]))/
    #                      max((v[3][1][0]-v[3][0][0]), (v[3][1][1]-v[3][0][1]))) for v in
    #                      building_num_collective_info_dict.values()]
    # print(aspect_ratio_list)

    v = building_num_collective_info_dict[building]
    aspect_ratio = min((v[3][1][0] - v[3][0][0]), (v[3][1][1] - v[3][0][1])) / \
                   max((v[3][1][0] - v[3][0][0]), (v[3][1][1] - v[3][0][1]))
    if aspect_ratio <= 0.5:
        return "narrow"
    elif aspect_ratio <= 0.7:
```

```
        return "medium-width"
    else:
        return "wide"


def is_area_over_rectangle_ratio_big(building_num):
    """

    :param building_num: the encoded building number
    :return:

    [('Pupin', 0.8991228070175439), ('SchapiroCEPSR', 1.0551470588235294),
    ('Mudd&EngTerrace&Fairchild&CS', 0.6627642646055922), ('NorthwestCorner', 1.0526870389884089),
    ('Uris', 0.8940170940170941), ('Schermerhorn', 0.6072981366459628), ('Chandler&Havemeyer', 0.7109405745769383),
    ('OldComputerCenter', 1.1258741258741258), ('Avery', 0.97), ('Fayerweather', 0.9456),
    ('Mathematics', 0.8861607142857143), ('LowLibrary', 0.8307757885763001), ('StPaulChapel', 0.9058333333333334),
    ('EarlHall', 0.9324324324324325), ('Lewisohn', 0.8976648351648352), ('Philosophy', 0.8735909822866345),
    ('Buell', 0.9444444444444444), ('AlmaMater', 1.1479591836734695), ('Dodge', 0.9386068476977568),
    ('Kent', 0.9423076923076923), ('CollegeWalk', 1.0626878488621727), ('Journalism&Furnald', 0.44685577580314423),
    ('Hamilton&Hartley&Wallach&JohnJay', 0.48759160559626913), ('Lerner', 1.039236479321315),
    ('ButlerLibrary', 0.9523981247746124), ('Carman', 1.0628019323671498)]

    I printed out al the area_over_rectangle_ratio, because if a building shape is more close to square or rectangle,
    the area of pixel should be quite close to the area of the bounding rectangle. There should not be a lot of gap,
    like L shape and C shape has, that caused a big difference between the 2 area values.
    """
    v = building_num_collective_info_dict[building_num]
    area_over_rectangle_ratio_list = v[1] / ((v[3][1][0] - v[3][0][0]) * (v[3][1][1] - v[3][0][1]))
    if area_over_rectangle_ratio_list > 0.83:
        return True
    else:
        return False

    # area_over_rectangle_ratio_list = [(v[0], v[1]/((v[3][1][0]-v[3][0][0]) * (v[3][1][1] - v[3][0][1]))) for v in
    #                       building_num_collective_info_dict.values()]
    # print(area_over_rectangle_ratio_list)


def is_aspect_ratio_wide(building_num):
    """
    This function will be used continuously after we check if a building could be called a rectangle.
    When the building is recognized as Rectangular, we could use the aspect ratio of wide to tell
    if it is specifically a Square.

    :param building_num:
    :return:
    """
    if get_aspect_ratio(building_num) == "wide":
        return True
    else:
        return False


def isL_or_C_or_Asymmetric_Shape(building_num, target_area):

    """
    L-shaped buildings have a property that will look the same if reflected according to one of the 2 diagonals.
    There are two ways to fold, one is with respect to the upper left to lower right diagonal, which could be obtained
    through np. transpose(), while the other type of diagonal is from upper right to lower left, tested by
    “np.rot90(np.fliplr(sub_array), -1)”. After getting the reflected building shape, I will compare it with the
    original building shape and count the overlapped pixels.
    If the ratio of this count and the total number of pixels of this building. called the “reflection score”,
    is greater than 0.8, this building is L-shaped. I got this number by checking the distribution of all buildings.
    In fact, the “Journalism” building is almost a perfect “L” and it has a high score above 0.9,
    but the other 2 buildings look L-shaped, “Chandler” and “Schermohorn” has an even lower score than “Mudd”.
    Thus, I know having a low ratio does not mean it is “L-shaped”. Thus, I further calculated the max length of
    the continuous black gap (black color is the area without building pixels, represented as 0 in the 2d numpy array).
    If the ratio between this length and the width of the MBR is greater than 0.6 but less than or equal to 0.75,
    it is also an “L-shaped”. It makes sense because the upper right area of the letter “L” will have a long continuous
    black part compared to an asymmetric building like “Mudd”, but at the same time, it should not be too high (> 0.75)
    because a “C-shaped” building like “Hamilton” will also have a continuous black gap with long length.
    In this case, I distinguish both L-shaped and C-shaped.

    Either “reflection score > 0.8” or “reflection score < 0.8 and 0.75 >= black line width/width > 0.6” gives an
    “L-shaped”. Then “reflection score < 0.8 and black line width/width > 0.75” gives a “C-shaped”.
    The building left will be asymmetric. Again the 0.6, 0.8, and 0.75 are chosen as thresholds because of distribution
    observation since I want my system to successfully recognize “Chandler” and “Schermerhorn” as “L-shaped” but not
    misclassify the C-shaped “Hamilton” or the asymmetric “Mudd” as “L-shaped”.
    All these threshold values are well-chosen to make this work.

    :param building_num:
    :param target_area:
    :return:
    """

    arr2d = np.zeros((height, width))
    row_index_list = target_area[0]
    column_index_list = target_area[1]
    arr2d[row_index_list, column_index_list] = 1

    upper_left = building_num_collective_info_dict[building_num][3][0]
    lower_right = building_num_collective_info_dict[building_num][3][1]

    height_bounding_rectangle = lower_right[1] - upper_left[1]
    width_bounding_rectangle = lower_right[0]- upper_left[0]

    # fix the upper left corner position
    if height_bounding_rectangle > width_bounding_rectangle:
        sub_array = arr2d[upper_left[1]: lower_right[1]+1, lower_right[0]-height_bounding_rectangle:lower_right[0]+1]
    else:
        bound1 = (upper_left[1] + width_bounding_rectangle+1)
        sub_array = arr2d[upper_left[1]: bound1, upper_left[0]:lower_right[0]+1]

    """
    There are two ways to fold, one is respect to the upper left to lower right diagonal, which could be
    obtained through np.transpose, while the other type of diagonal is from upper right to lower left, which
    could be obtained through np.rot90(np.fliplr(sub_array), -1)
    """
    array_reflected_by_diagonal = np.transpose(sub_array)
    array_reflected_by_diagonal2 = np.rot90(np.fliplr(sub_array), -1)

    # Find the indices of the elements in A that are equal to 1
    indices = np.argwhere(sub_array == 1)

    # Find the corresponding elements in B
    matches1 = array_reflected_by_diagonal[indices[:, 0], indices[:, 1]] == 1
    matches2 = array_reflected_by_diagonal2[indices[:, 0], indices[:, 1]] == 1
```

```
    # Count the number of matches
    count1 = np.count_nonzero(matches1)
    count2 = np.count_nonzero(matches2)

    reflection_value = (max(count1, count2)/building_num_collective_info_dict[building_num][1])
    if reflection_value > 0.8:
        return "L-shaped"
    else:
        # it maybe an L-shaped, count the length of the black gap
        max_length = 0
        for i in range(upper_left[1], lower_right[1]):
            current_length = 0
            for j in range(upper_left[0], lower_right[0]):
                if arr2d[i,j] == 0:
                    current_length += 1
                    max_length = max(max_length, current_length)
                else:
                    current_length = 0
        black_line_width_ratio = max_length/(lower_right[0]-upper_left[0])
        if 0.6 < black_line_width_ratio <= 0.75:
            return "L-shaped"
        elif black_line_width_ratio > 0.75:
            return "C-shaped"
        else:
            return "asymmetric"



def isIShape(target_area):
    """
    “I” has two horizontal bars at the top and bottom and a skinny vertical bar.
    I will find the first and last point of the skinny vertical bar.
    Imagine the skinny vertical bar of the letter “I” as a rectangle, the first and last point is the upper left
    and lower right point of this rectangle. Then the difference between the x-coordinates of these 2 points is
    the width of this skinny body of “I”. Then, I get the x-coordinate of the first point and I will check
    if the x-coordinates are the same for every point with indices “first point + k * width (where k from 1 to the
    height of the skinny body)”. Also, I will do a similar thing for the last point. If there is even one point
    that has different x-coordinate, I strictly do not call it an "I", but it works well for Columbia Campus.
    Essentially, I am testing if the skinny vertical bar structure of “the letter” exists, and there is only one.

    :param target_area:
    :return:
    """
    row_index_list = target_area[0]
    column_index_list = target_area[1]

    leftmost_column = np.min(column_index_list)
    rightmost_column = np.max(column_index_list)

    if column_index_list[0] != leftmost_column or column_index_list[len(column_index_list)-1] != rightmost_column:
        return False

    breakpoint_upper_row_index = 0
    breakpoint_upper_row_index_in_list = 0
    breakpoint_lower_row_index = 0
    breakpoint_lower_row_index_in_list = 0
    for i in range(len(column_index_list)-1):
        if column_index_list[i] == rightmost_column and column_index_list[i+1] != leftmost_column:
            breakpoint_upper_row_index, breakpoint_upper_row_index_in_list = row_index_list[i+1], i+1
            break
    if breakpoint_upper_row_index_in_list == 0:
        return False

    for i in range(len(column_index_list)-1):
        if column_index_list[i] != rightmost_column and column_index_list[i+1] == leftmost_column:
            breakpoint_lower_row_index, breakpoint_lower_row_index_in_list = row_index_list[i], i
            break
    if breakpoint_lower_row_index_in_list == 0:
        return False

    breakpoint_leftmost_index = column_index_list[breakpoint_upper_row_index_in_list]
    breakpoint_rightmost_index = column_index_list[breakpoint_lower_row_index_in_list]

    for i in range(1, breakpoint_lower_row_index-breakpoint_upper_row_index):
        left_bound = column_index_list[breakpoint_upper_row_index_in_list + (breakpoint_rightmost_index-breakpoint_leftmost_index+1) * i]
        right_bound = column_index_list[breakpoint_lower_row_index_in_list - (breakpoint_rightmost_index-breakpoint_leftmost_index+1) * i]
        if (left_bound != breakpoint_leftmost_index) or (right_bound != breakpoint_rightmost_index):
            return False

    return True


def get_geometry(building_num, labeled_map):
    """
    The logic here is first we will get the indices of the target building, represented by the building number within
    the labeled map, and I call these index information "target_area_info".

    Then my logic is:
    if this building has big area/rectangular area ratio and is wide and is not I-shaped --> Square
    if this building has big area/rectangular area ratio and is not wide and is not I-shaped --> Rectangular
    if this building has big area/rectangular and is I-shaped --> I-shaped


    :param building_num:
    :param labeled_map:
    :return:
    """
    target_area_info = np.where(labeled_map == building_num)

    # # Just use number 142 to explore how to detect an I shaped
    # target_area_info = np.where(labeled_map == 142)
    # print(isIShape(target_area_info))

    # # Just use number 66 to explore how the pixels are arranged
    # target_area_info = np.where(labeled_map == 66)
    # isLShape(66, target_area_info)

    if is_area_over_rectangle_ratio_big(building_num) and is_aspect_ratio_wide(building_num) and not isIShape(target_area_info):
        return "Square"
    elif is_area_over_rectangle_ratio_big(building_num) and not is_aspect_ratio_wide(building_num) and not isIShape(target_area_info):
        return "Rectangular"
    elif is_area_over_rectangle_ratio_big(building_num) and isIShape(target_area_info):
        return "I-shaped"
    else:
        geometry = isL_or_C_or_Asymmetric_Shape(building_num, target_area_info)
        return geometry
```

```
def step2(encode_dict, labeled_map):
    """

    1. print building "what" description
    2. Find confusion building for each building
    3. Minimization of description

    :param encode_dict:
    :param labeled_map: shape (495, 275) 2d numpy array
    :return:
    """
    # print(f"labeled map size: {labeled_map.shape}")
    print("-" * 20)
    print("Step2")
    for building_num in encode_dict:
        building_num = int(building_num)
        # Get the size
        size = get_size(building_num)
        aspect_ratio = get_aspect_ratio(building_num)
        geometry = get_geometry(building_num, labeled_map)
        building_name_shape_description[building_num_collective_info_dict[building_num][0]] = {"size": size, "aspect_ratio": aspect_ratio, "geometry": geometry}
        print(f"building name: {building_num_collective_info_dict[building_num][0]}, size: {size}, aspect_ratio: {aspect_ratio}, geometry: {geometry}")

    print("-" * 20)

    # Check if there is any confusion
    for key, value in building_name_shape_description.items():
        for key2, value2 in building_name_shape_description.items():
            if key != key2 and value == value2:
                if key in confusion_dictionary_what:
                    confusion_dictionary_what[key].append(key2)
                else:
                    confusion_dictionary_what[key] = [key2]

    for key, value in confusion_dictionary_what.items():
        print(f"{key} can be confused with ", end="")
        print(value)

    # minimization
    print("-"*20)
    print("Minimization description Step 2:")
    """

    For the minimization process, I followed 2 general principles. First, any of the 3 aspects could be dropped if
    dropping it will make the "what" description unique. For instance, there is only 1 "asymmetric" building,
    so dropping size and aspect ratio is acceptable, as this building cannot be confused with any other building.
    Second, dropping any of the 3 aspects makes the "what" description all the same. For instance, all the 3
    "I-shaped" buildings have the same size and aspect ratio, so keeping them does not help us further identify
    buildings as I-shaped, and we could just call them "I-shaped" buildings.


    If this building is not confused with any other building, then it could be minimized. I found this
        already include the largest and smallest size description
    If this building is medium-size and its geometry is rectangular, I discover it is always narrow, so there is
        no need to describe its aspect ratio
    If this building is large and its aspect ratio is medium-width, I discover it is always rectangular, so there is
        no need to describe its geometry
    If this building is shaped, it is always large and wide, so just describe it as L-shaped
    If this building is small but not the smallest, the aspect_ratio will always be medium width and the geometry
        is rectangular.
    If this building is I-shaped, it is always medium-size and medium-width, so just describe it as I shaped


    """
    for key, value in building_name_shape_description.items():
        size = value["size"]
        aspect_ratio = value["aspect_ratio"]
        geometry = value["geometry"]
        if key not in confusion_dictionary_what:
            step2_result[key] = f"a {size} {geometry} building"
            print(f"{key} is a {size} {geometry} building")

        elif key in confusion_dictionary_what and size == "medium-size" and geometry == "Rectangular":
            step2_result[key] = f"a {size} {geometry} building"
            print(f"{key} is a {size} {geometry} building")

        elif key in confusion_dictionary_what and size == "large" and aspect_ratio == "medium-width":
            step2_result[key] = f"a {size} {aspect_ratio} building"
            print(f"{key} is a {size} {aspect_ratio} building")

        elif key in confusion_dictionary_what and geometry == "L-shaped":
            step2_result[key] = f"a {geometry} building"
            print(f"{key} is a {geometry} building")

        elif key in confusion_dictionary_what and size == "small":
            step2_result[key] = f"a {size} building"
            print(f"{key} is a {size} building")

        elif key in confusion_dictionary_what and geometry == "I-shaped":
            step2_result[key] = f"a {geometry} building"
            print(f"{key} is a {geometry} building")


def extract_verticality_horizontality(building_num):
    """
    As for horizontality and verticality, I simply divided the whole 495 * 275 map into 5 * 5 areas. The width and
    height are just equally divided into 5 sections, which correspond to the labels from leftmost to rightmost or
    uppermost to lowermost correspondingly. I just accessed the center of mass coordinates in my collective
    information dictionary and used the x and y coordinates of the center of mass to assign horizontality and
    verticality labels. It worked pretty well.

    :param building_num:
    :return:
    """
    unit_height = height / 5
    unit_width = width / 5

    center_mass = building_num_collective_info_dict[building_num][4]
    x, y = center_mass[0], center_mass[1]
    if x >= 0 and x < unit_width:
        horizontality = "leftmost"
    elif x >= unit_width and x < 2 * unit_width:
        horizontality = "left"
    elif x >= 2 * unit_width and x < 3 * unit_width:
        horizontality = "mid-width"
    elif x >= 3 * unit_width and x < 4 * unit_width:
        horizontality = "right"
```

```
    else:
        horizontality = "rightmost"

    if y >= 0 and y < unit_height:
        verticality = "uppermost"
    elif y >= unit_height and y < 2 * unit_height:
        verticality = "upper"
    elif y >= 2 * unit_height and y < 3 * unit_height:
        verticality = "mid-height"
    elif y >= 3 * unit_height and y < 4 * unit_height:
        verticality = "lower"
    else:
        verticality = "lowermost"

    return (verticality, horizontality)


def extract_orientation(building_num):
    """
    As for orientation, I first directly retrieved the aspect ratio of each building from my “step2_result”
    dictionary. If it’s a wide building, it indicates it is likely a square, so it is not oriented. Then for those
    narrow buildings, I calculated the width and height, and if the height is greater than the width,
    it is a vertically-oriented building, and vice versa for horizontally-oriented buildings.

    :param building_num:
    :return:
    """
    building_name = building_num_collective_info_dict[building_num][0]
    if building_name_shape_description[building_name]["aspect_ratio"] == "wide":
        return "non-oriented"
    else:
        upper_left = building_num_collective_info_dict[building_num][3][0]
        lower_right = building_num_collective_info_dict[building_num][3][1]
        height = lower_right[1] - upper_left[1]
        width = lower_right[0] - upper_left[0]

        if height > width:
            return "vertically-oriented"
        else:
            return "horizontally-oriented"


def step3():
    # Print out the horizontality, verticality, and orientation
    print("-" * 20)
    print("Step3")
    for building_num in encode_dict:
        building_num = int(building_num)
        verticality, horizontality = extract_verticality_horizontality(building_num)
        orientation = extract_orientation(building_num)
        building_name_location_description[building_num_collective_info_dict[building_num][0]] = {"verticality": verticality,
                                                                                                  "horizontality": horizontality,
                                                                                                  "orientation": orientation}
        print(f"building name: {building_num_collective_info_dict[building_num][0]}, horizontality: {horizontality}, "
              f"verticality: {verticality}, orientation: {orientation}")

    print("-" * 20)

    # Check if there is any confusion using the same methodology in step 2
    for key, value in building_name_location_description.items():
        for key2, value2 in building_name_location_description.items():
            if key != key2 and value == value2:
                if key in confusion_dictionary_where:
                    confusion_dictionary_where[key].append(key2)
                else:
                    confusion_dictionary_where[key] = [key2]

    for key, value in confusion_dictionary_where.items():
        print(f"{key} can be confused with ", end="")
        print(value)

    # Minimization

    """As for the minimization step, I just applied the same idea in Step 2. I discovered that the Campus is designed
    in a way to avoid clustered buildings. Thus, if imagining the Campus being divided into 5 * 5 little areas,
    there is likely only 1 building in each little area based on the center of mass, so you essentially can remove
    most of the orientation information. There are definitely exceptions like in the mid-height on the right,
    there are both “St.Paul” and “Avery” which can only be distinguished by their orientation. Thus, I carefully
    designed different cases with a lot of if-else statements. """

    print("-" * 20)
    print("Minimization description Step 3:")

    """
    If this building is at uppermost, then using only horizontality is enough to distinguish them
    If this building is mid-width and mid-height, then its orientation is non-oriented, so drop orientation
    If this building is mid-width and upper, then its orientation has to be vertically-oriented, since only Uris is
        at that position
    If this building is left and vertically-oriented, it has to be old computer center, that's the only one.
    If this building is right and vertically-oriented, it is Avery, if it's right and horizontally-oriented, it's Buell
    If this building is mid-width and mid-height, it will always be non-oriented-->low library and AlmaMater
    If this building is mid-width and lower, it has to be College walk
    If this building is mid-width and lowermost, it has to be Butler library
    If this building is rightmost and lowermost, it it Hamilton, since that's the only building at lower right corner
    If this building is leftmost and lowermost, having orientation does not help distinguish them, drop orientation

    """
    for key, value in building_name_location_description.items():
        verticality = value["verticality"]
        horizontality = value["horizontality"]
        orientation = value["orientation"]
        if verticality == "uppermost":
            step3_result[key] = f"at uppermost and {horizontality}"
            print(f"{key} is at uppermost and {horizontality} in Columbia Campus")
        elif verticality == "mid-height" and horizontality == "mid-width":
            step3_result[key] = f"at mid-height and mid-width"
            print(f"{key} is at mid-height and mid-width in Columbia Campus")
        elif horizontality == "mid-width" and verticality == "upper":
            step3_result[key] = f"at mid-width and upper"
            print(f"{key} is at mid-width and upper in Columbia Campus")
        elif horizontality == "left" and orientation == "vertically-oriented":
            step3_result[key] = f"vertically-oriented at left"
            print(f"{key} is at left and it's vertically-oriented in Columbia Campus")
        elif horizontality == "right" and orientation == "vertically-oriented":
            step3_result[key] = f"vertically-oriented at right"
            print(f"{key} is at right and it's vertically-oriented in Columbia Campus")
        elif horizontality == "right" and orientation == "horizontally-oriented":
            step3_result[key] = f"horizontally-oriented at right"
```

```
            print(f"{key} is at right and it's horizontally-oriented in Columbia Campus")
        elif horizontality == "mid-width" and verticality == "mid-height":
            step3_result[key] = f"at mid-width and mid-height"
            print(f"{key} is at mid-width and mid-height in Columbia Campus")
        elif horizontality == "mid-width" and verticality == "lower":
            step3_result[key] = f"at mid-width and lower"
            print(f"{key} is at mid-width and lower in Columbia Campus")
        elif horizontality == "mid-width" and verticality == "lowermost":
            step3_result[key] = f"at mid-width and lowermost"
            print(f"{key} is at mid-width and lowermost in Columbia Campus")
        elif horizontality == "rightmost" and verticality == "lowermost":
            step3_result[key] = f"at rightmost and lowermost"
            print(f"{key} is at rightmost and lowermost in Columbia Campus")
        elif horizontality == "leftmost" and verticality == "lowermost":
            step3_result[key] = f"at leftmost and lowermost"
            print(f"{key} is at leftmost and lowermost in Columbia Campus")
        else:
            step3_result[key] = f"{orientation} at {horizontality} and {verticality}"
            print(f"{key} is a building at {horizontality} and {verticality} and it's {orientation} in Columbia Campus")


def present_nearness_matrix():
    # Create the 26 * 26 2d numpy array
    building_num_list = list(building_num_collective_info_dict.keys())
    matrix = np.zeros((26, 26))

    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            # Skip a building that is compared to itself
            if i == j:
                continue
            else:
                """To evaluate if one building is near to the other, I first checked their horizontality and
                verticality. If both of horizontality and verticality of 2 buildings are off by two levels or more (
                for example, “left” and “leftmost” is off by 1 level, “leftmost” and “mid-width” are off by 2
                levels), there are no need to do any calculation, and I will just assign a 0. """
                building_num1 = building_num_list[i]
                building_num2 = building_num_list[j]
                building_name1 = building_num_collective_info_dict[building_num1][0]
                building_name2 = building_num_collective_info_dict[building_num2][0]
                building_horizontality1 = building_name_location_description[building_name1]["horizontality"]
                building_horizontality2 = building_name_location_description[building_name2]["horizontality"]
                building_verticality1 = building_name_location_description[building_name1]["verticality"]
                building_verticality2 = building_name_location_description[building_name2]["verticality"]
                horizontality_encoding_list = ["leftmost", "left", "mid-width", "right", "rightmost"]
                verticality_encoding_list = ["uppermost", "upper", "mid-height", "lower", "lowermost"]

                encoded_horizontality1 = horizontality_encoding_list.index(building_horizontality1)
                encoded_horizontality2 = horizontality_encoding_list.index(building_horizontality2)
                encoded_verticality1 = verticality_encoding_list.index(building_verticality1)
                encoded_verticality2 = verticality_encoding_list.index(building_verticality2)

                if abs(encoded_verticality1 - encoded_verticality2) > 1 and \
                        abs(encoded_horizontality2 - encoded_horizontality1) > 1:
                            continue

                """Then for the rest of the pair, I calculated the Euclidean distance between their center of masses
                and took the ratio of this distance over the smaller area of the pair, called “distance_size_ratio”

                The logic to evaluate if a pair if near to each other:

                1. if the area of the target building is less than ⅕ of that of the source building, the distance
                needs to be less than or equal to 80 for them to be called “near”. This is the case when the target
                building is really small.
                2. if the distance is less than or equal to 120 and the distance_size ratio
                is less than or equal to 0.06, this pair is also considered “near”. This is the case for normal
                building pairs
                3. if the area of the source building is greater than or equal to ⅕ of the target
                building, it means the source building is not that small, so the system will just use the distance as
                the sole metric. If it is greater than or equal to 60, it will assign a 1 to the building pair.

                More logic explanations in the report writeup.

                """

                center_mass_1 = building_num_collective_info_dict[building_num1][4]
                center_mass_2 = building_num_collective_info_dict[building_num2][4]

                macro_distance = math.sqrt((center_mass_1[0] - center_mass_2[0])**2 + (center_mass_1[1] - center_mass_2[1])**2)

                smaller_area = min(building_num_collective_info_dict[building_num1][1], building_num_collective_info_dict[building_num2][1])
                distance_size_ratio = macro_distance / smaller_area

                # target building is the small one
                if building_num_collective_info_dict[building_num1][1] < building_num_collective_info_dict[building_num2][1]/5:
                    if macro_distance <= 80:
                        matrix[i, j] = 1
                # target building is not that small, or it is the large one
                else:
                    # If the distance is small, plus the source building is not a small building
                    if macro_distance <= 60 and building_num_collective_info_dict[building_num2][1] >= building_num_collective_info_dict[building_num1][1]/5:
                        matrix[i,j] = 1
                    # normal building pair.
                    elif macro_distance <= 120 and distance_size_ratio <= 0.06:
                        matrix[i,j] = 1

    # Print out results of the source building list for each target building
    for i in range(matrix.shape[0]):
        building_name = building_num_collective_info_dict[building_num_list[i]][0]
        row = matrix[i]
        indices = np.where(row == 1)[0].tolist()
        near_building_num_list = [building_num_list[i] for i in indices]
        building_name_list = [building_num_collective_info_dict[num][0] for num in near_building_num_list]

        buildings_nearBy_target[building_name] = building_name_list
        print(f"{building_name} as target, it is near to ", end="")
        print(buildings_nearBy_target[building_name])

    row_indices = np.where(matrix == 1)[0]
    col_indices = np.where(matrix == 1)[1]

    print("-" * 20)

    # # Print out results of the target building list for each source building
    for j in range(matrix.shape[1]):
        building_name = building_num_collective_info_dict[building_num_list[j]][0]
        indices = row_indices[col_indices == j]
        print(f"{building_name} as source, source buildings close to ", end="")
```

```
        near_building_num_list = [building_num_list[i] for i in indices]

        building_name_list = [building_num_collective_info_dict[num][0] for num in near_building_num_list]

        buildings_nearBy_source[building_name] = building_name_list
        print(buildings_nearBy_source[building_name])


def step4():
    print("-"*40)
    print("Step 4:")
    # This function will print out all the near results for each building
    present_nearness_matrix()

    """Confusion part is solved using typical max and min fucntions of lambda function"""
    print("-" * 20)
    most_pair_source = max(buildings_nearBy_source.items(), key=lambda x: len(x[1]))
    max_source_building_name, max_target_building_list = most_pair_source
    print(f"{max_source_building_name} as source, has the most targets ", end="")
    print(max_target_building_list)

    # Retrieve the key-value pair with the least elements
    least_pair_source = min(buildings_nearBy_source.items(), key=lambda x: len(x[1]))
    least_source_building_name, least_target_building_list = least_pair_source
    print(f"{least_source_building_name} as source, has the least targets ", end="")
    print(least_target_building_list)

    most_pair_target = max(buildings_nearBy_target.items(), key=lambda x: len(x[1]))
    max_target_building_name, max_source_building_list = most_pair_target
    print(f"{max_target_building_name} as target, has the most sources ", end="")
    print(max_source_building_list)

    least_pair_target = min(buildings_nearBy_target.items(), key=lambda x: len(x[1]))
    least_target_building_name, least_source_building_list= least_pair_target
    print(f"{least_target_building_name} as target, has the least sources ", end="")
    print(least_source_building_list)

    # Minimization part also solved by typical lambda function that counts the number
    # of occurrence of each source building in other target building lists.
    print("-" * 20)
    print("Minimization Step 4")
    source_list = list(buildings_nearBy_target.values())
    flat_list = list(itertools.chain(*source_list))
    counts = Counter(flat_list)
    for key, value in buildings_nearBy_target.items():
        if not value:
            step4_result[key] = None
            print(f"There is no source, if {key} serves as a target")
        else:
            landmark_building = max(value, key=lambda x: counts.get(x, 0))
            step4_result[key] = landmark_building
            print(f"{key} as a target, the most-often-used source is {landmark_building}")


def step5(encoded_dict):
    print("-"*20)
    print("Step 5: ")
    target_what_where_dict = {}

    """
    The idea is to first extract the landmark choice obtained in step 4 for each building
    This will serve as the source building. Then for this pair of target and source building,
    I will just retrieve the what and where to concatenate them together.
    """
    for building_num in encoded_dict:
        building_num = int(building_num)
        building_name = building_num_collective_info_dict[building_num][0]
        step2_description = step2_result[building_name]
        step3_description = step3_result[building_name]
        source = step4_result[building_name]
        source_what = step2_result[source]
        source_where = step3_result[source]

        target_what_where_dict[building_name] = f"{step2_description} {step3_description}"
        step5_result[building_name] = f"{step2_description} {step3_description} near {source_what} {source_where}"
        print(f"{step2_description} {step3_description} near {source_what} {source_where}")

    # Check for confusion, I discovered that my system will not have any confusion when concatenating the “what” and
    # “where” for a target building.
    print("-"* 20)
    print("Confusion: ")
    for key, value in step5_result.items():
        for key2, value2 in step5_result.items():
            if key != key2 and value == value2:
                if key in confusion_dict_step5:
                    confusion_dict_step5[key].append(key2)
                else:
                    confusion_dict_step5[key] = [key2]

    for key, value in confusion_dict_step5.items():
        print(f"{key} can be confused with ", end="")
        print(value)
    print("Ok, there is no confusion")

    # Minimization:
    print("-" * 20)
    print("Minimization Step 5:")
    """
    I discovered that every "what + where" description is unique, so my description
    relies on the source landmark building, so to some extent, my target "what + where"
    is already enough, so I could reduce some less important feature in the source building.

    This section of commented code help confirm that there is no confusion of "what+where"
    just considering the target building, without considering the source.

    Principle: 1. Geometry is most unique "what" property of a building. If the source building is "I" "L",
    "C" "Square", remove all the other information of the source building
    2. There is only one asymmetric,
    so if the source is the asymmetric building ,just describe it as asymmetric building
    3. If the source building is
    the most usual geometry: Rectangular, remove the horizontality and verticality of the source """

    # # The confusion list that stores the pair of buildings that could be confused
    # # based on both what and where description is empty
    # confusion_target_list = []
    # for key, value in target_what_where_dict.items():
```

```
    #     for key2, value2 in target_what_where_dict.items():
    #         if key != key2 and value == value2:
    #             confusion_target_list.append((key, key2))
    # print(confusion_target_list)

    for key, value in step5_result.items():
        near_index = value.index("near")
        source_description = value[near_index+5:]
        if "Square" in source_description:
            step5_result_minimized[key] = f"{value[:near_index+5]}a Square building"
            print(f"{value[:near_index+5]}a Square building")
        elif "I-shaped" in source_description:
            step5_result_minimized[key] = f"{value[:near_index+5]}a I-shaped building"
            print(f"{value[:near_index+5]}a I-shaped building")
        elif "C-shaped" in source_description:
            step5_result_minimized[key] = f"{value[:near_index+5]}a C-shaped building"
            print(f"{value[:near_index+5]}a C-shaped building")
        elif "L-shaped" in source_description:
            step5_result_minimized[key] = f"{value[:near_index+5]}a L-shaped building"
            print(f"{value[:near_index+5]}a L-shaped building")
        elif "asymmetric" in value[: near_index+5]:
            step5_result_minimized[key] = f"a asymmetric building near {source_description}"
            print(f"{step5_result_minimized[key]}")
        # The only shape left in this area is Rectangular, which is not a geometry that has
        # much characteristics, to further minimize description, we could not think from using
        # the perspective of geometry any more. Then just remove the location information of
        # the source building. It is not that useful as source is always near to target, vistor
        # could just look around.
        else:
            at_index = value.rindex("at")
            # have a orientation itself indicates the rectangular shape
            if "Rectangular" and "oriented" in source_description:
                source_description = source_description.replace("Rectangular ", "")
                at_index = source_description.rindex("at")
                step5_result_minimized[key] = value[:near_index+5] + source_description[:at_index]
            else:
                step5_result_minimized[key] = value[:at_index]
            print(step5_result_minimized[key])

    print("-" * 20)

    # Find the shortest unambiguous description and longest by counting keywords
    key_word = ["smallest", "small", "medium-size", "large", "largest", "narrow", "medium-width",
                "wide", "Square", "Rectangular", "I-shaped", "C-shaped", "L-shaped", "asymmetric",
                "uppermost", "upper", "mid-height", "lower", "lowermost", "leftmost", "left", "mid-width",
                "right", "rightmost", "vertically-oriented", "horizontally-oriented", "non-oriented"]
    count_keyword_dict = {}
    for key, value in step5_result_minimized.items():
        value_list = value.split()
        count = 0
        for keyword in key_word:
            count += value_list.count(keyword)
        count_keyword_dict[key] = count

    max_keyword = max(list(count_keyword_dict.values()))
    min_keyword = min(list(count_keyword_dict.values()))

    print(f"Longest unambiguous description has count {max_keyword} ")
    for key, value in count_keyword_dict.items():
        if value == max_keyword:
            print(f"The building is {key}")
    print(f"Shortest unambiguous description has count {min_keyword} ")
    for key, value in count_keyword_dict.items():
        if value == min_keyword:
            print(f"The building is {key}")


if __name__ == '__main__':
    result = read_image()
    encode_dict = read_table()
    step1(encode_dict, result)
    step2(encode_dict, result)
    step3()
    step4()
    step5(encode_dict)
```