Shortest-path grocery shopping

What's the shortest path thru the grocery store to get my items?

Justin Pearson

Apr 19, 2020

In[1]:=
```
Clear["Global`*"]
SetDirectory[NotebookDirectory[]];
```

Summary

Given an image of the store's aisles, we mesh it and generate a graph from the mesh. Then we use various shortest-path algorithms on the graph to find the shortest tour that visits all the items.

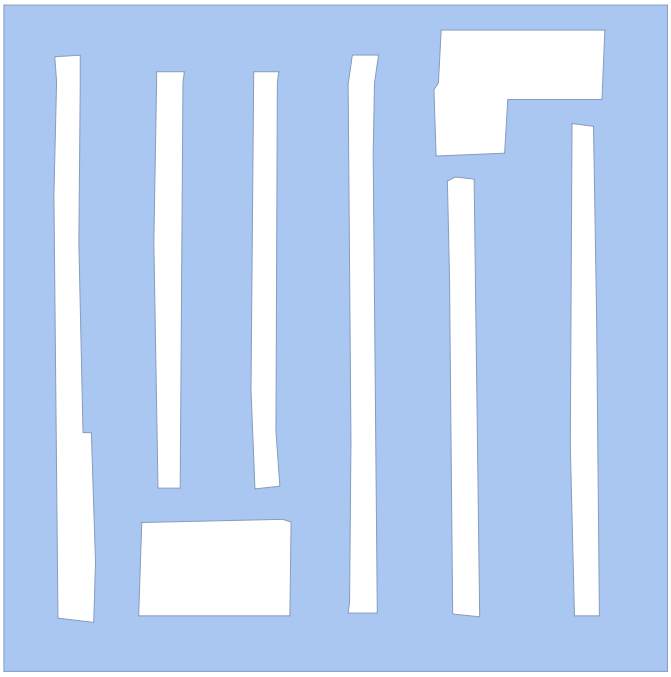
We are careful to build a graph whose edge-weights equal the euclidean distance between adjacent vertices, so that that shortest-path is shortest with respect to actual space, not just "number of hops on the graph".

In[3]:=
```
i =
```

```
;
```

In[4]:=
```
m = ImageMesh[i]
```

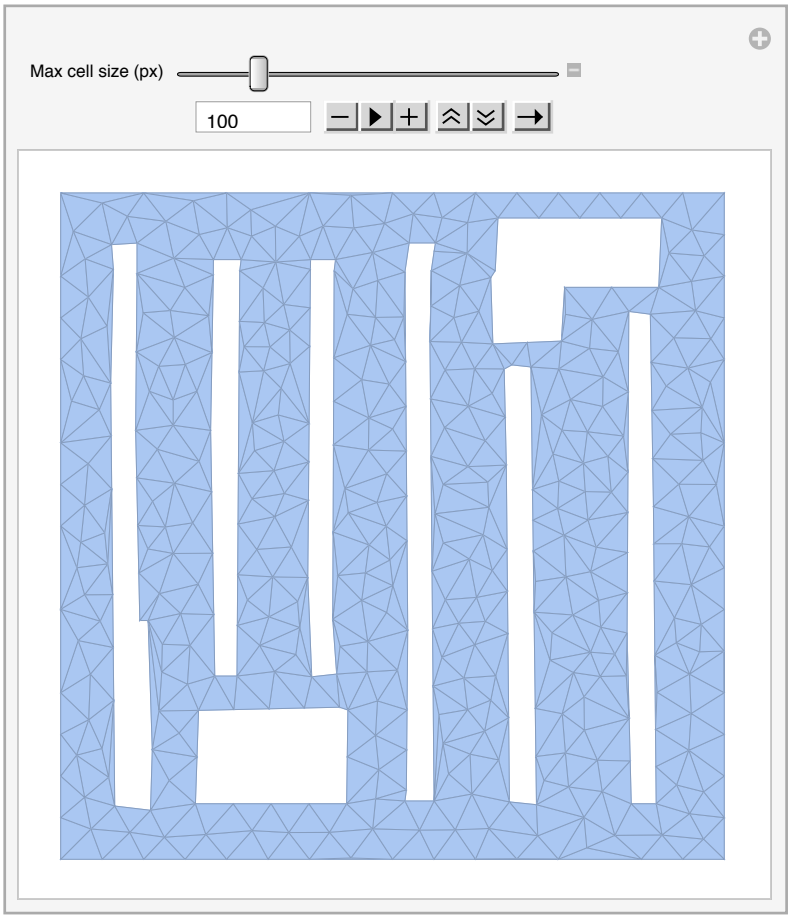
Out[4]=



Triangularize the mesh. Pick a small “max cell size” to get small triangles.

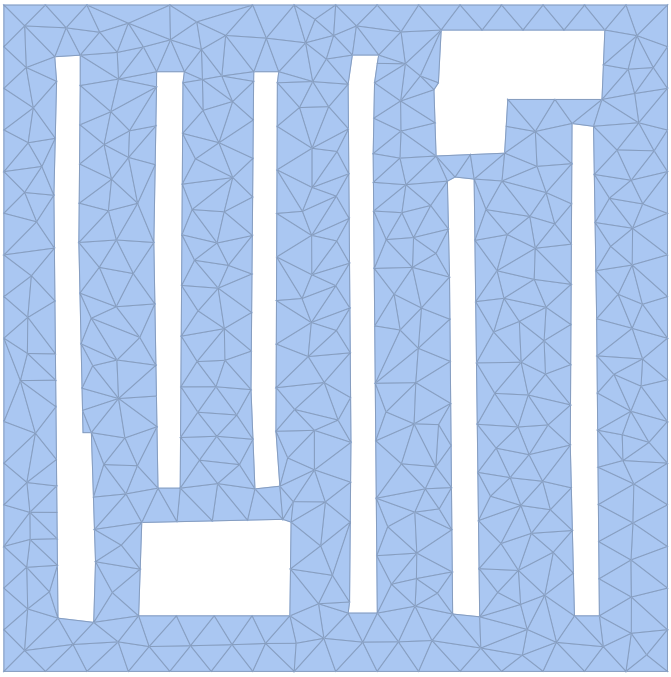
In[5]:=
```
Manipulate[
 TriangulateMesh[m, MaxCellMeasure → c],
 {{c, 100, "Max cell size (px)"}, 10, 500, 1, Appearance → "Open"}
]
```

Out[5]=



In[6]:=
```
t = TriangulateMesh[m, MaxCellMeasure → 100]
```

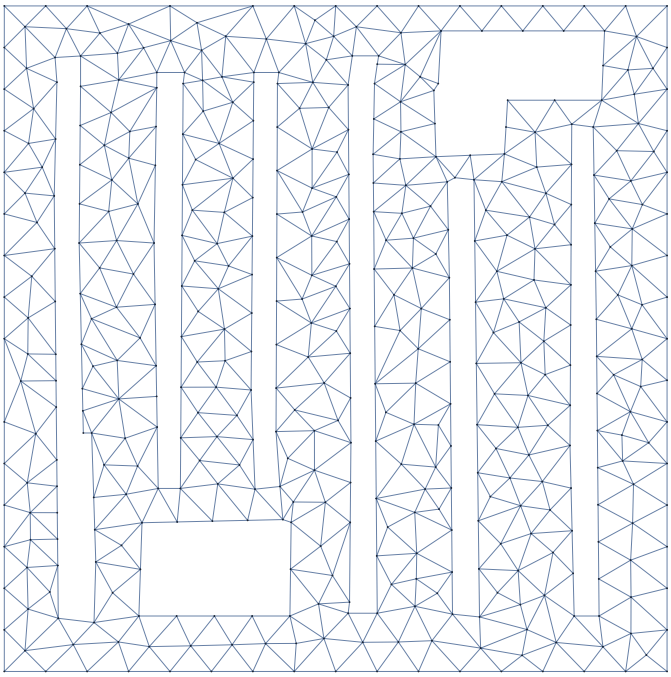
Out[6]=



Convert the triangularized-mesh to a graph, using the degree-0 elements of the mesh -- the mesh points -- as vertices. (Degree-1 is the triangle edges, degree-2 is the triangles themselves.)

In[7]:=
```
g = MeshConnectivityGraph[t, 0]
```

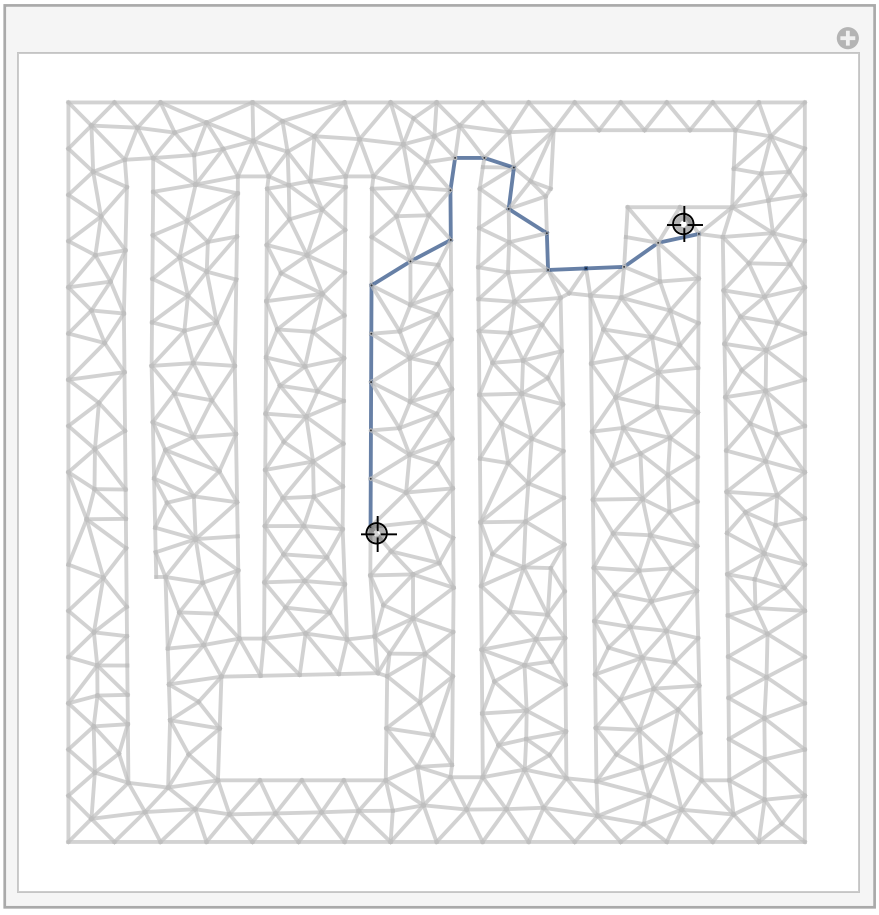
Out[7]=



A slick UI for showing shortest-paths between any two vertices.

In[8]:=
```
Manipulate[
 GraphPlot[
  HighlightGraph[
   g,
   PathGraph@
    FindShortestPath[
     g,
     NearestMeshCells[{t, 0}, u[[1]], 1] // First,
     NearestMeshCells[{t, 0}, u[[2]], 1] // First
    ],
   GraphHighlightStyle → "DehighlightGray"],
  EdgeStyle → Thickness[0.005],
  ImageSize → 400
 ],
 {{u, {{100, 100}, {200, 200}}}, Locator}
]
```

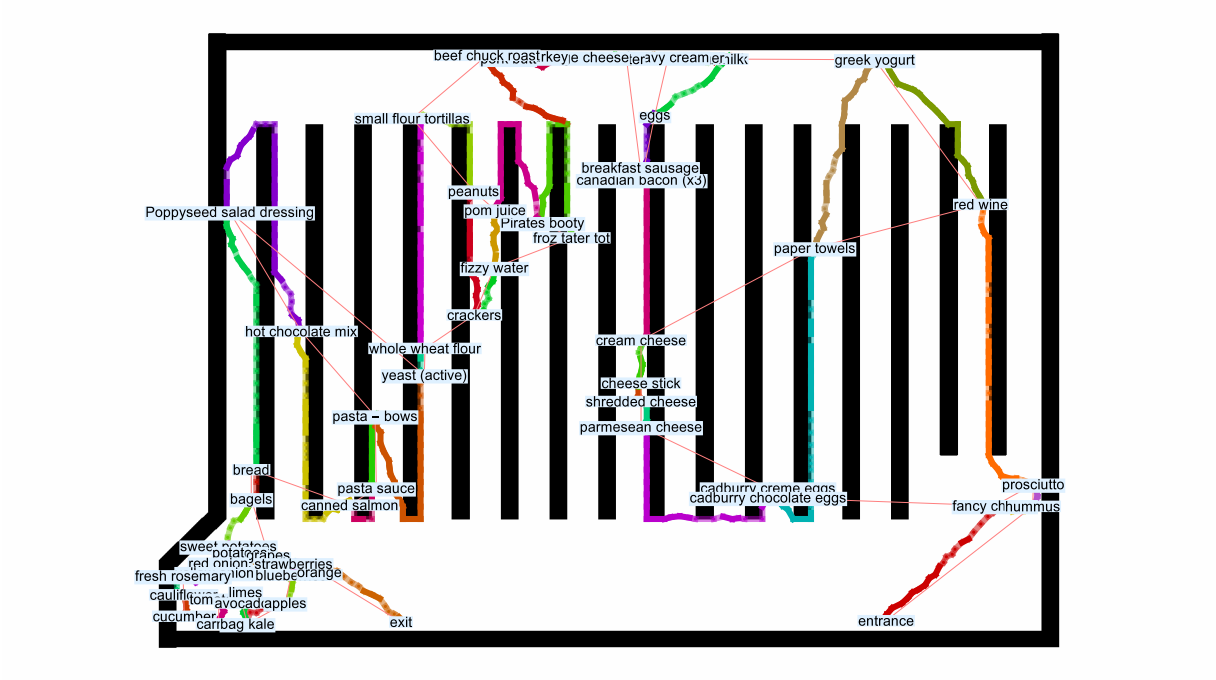
Out[8]=



We extend this idea to build a shortest-path thru the store that visits all your items:

In[9]:=
```
Show[First@Import["map-1200px.pdf"], ImageSize → Full]
```

Out[9]=

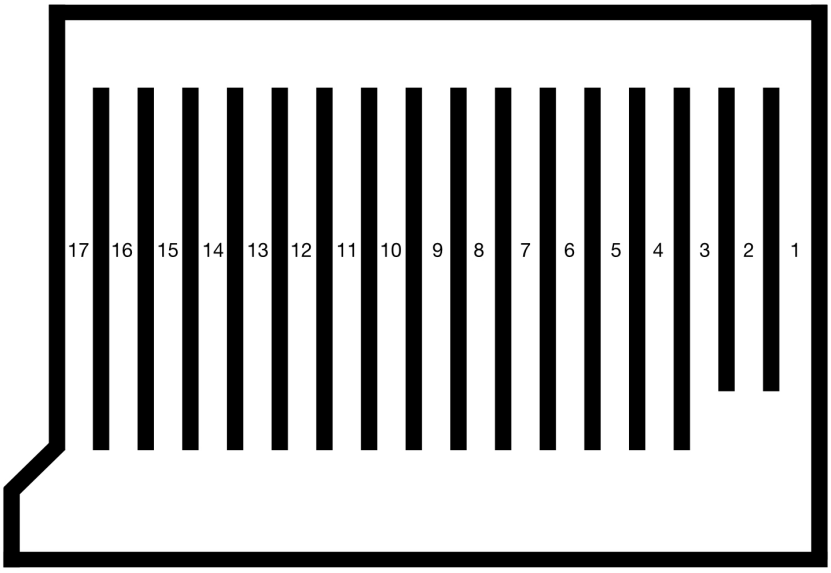


- This is a real map of my local Albertsons grocery store.
- The thin pink line is the shortest tour, ignoring occlusions like the (black) aisles.
- The colored paths visit the items in the order of the shortest tour.
- To find the shortest tour thru the items along the triangulated mesh, I first found the shortest pairwise distance between every pair of items on the triangulated-mesh graph. Then I made a new complete graph with only 40 vertices -- the items -- whose edge weights were the pairwise distances. Then I asked MMA to find the shortest tour of those 40 vertices. (It would have been simpler to ask MMA to find the shortest path on the triangulated-mesh graph from vertex 1 to vertex 40 that visits all 40 vertices, but I didn't know how to do that.)

Build graph of the grocery store

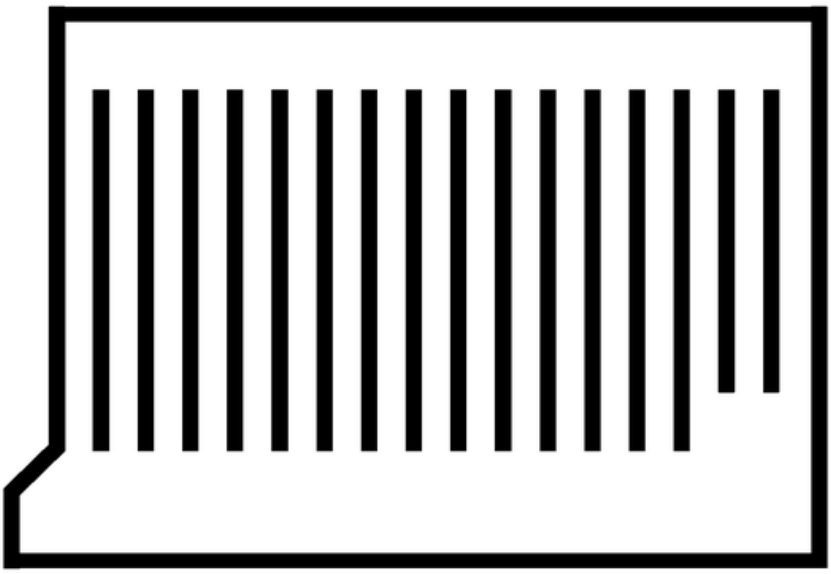
In[10]:=
```
albertsons = Import["albertsons-map/albertsons-map.001.jpeg"]
```

Out[10]=



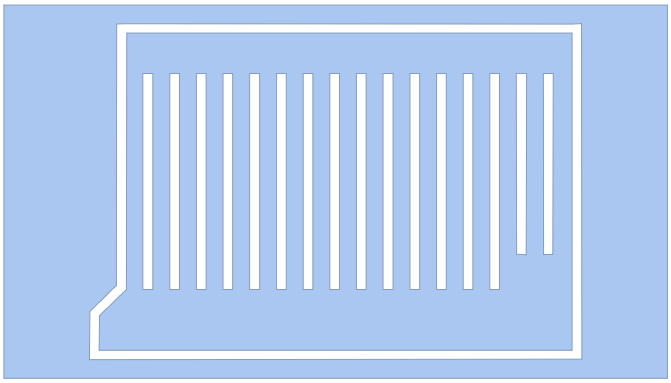
In[11]:=
```
im = albertsons // Binarize // ColorNegate //
     DeleteSmallComponents[#, 700] & // ColorNegate;
Thumbnail[
 im,
 700]
```

Out[12]=



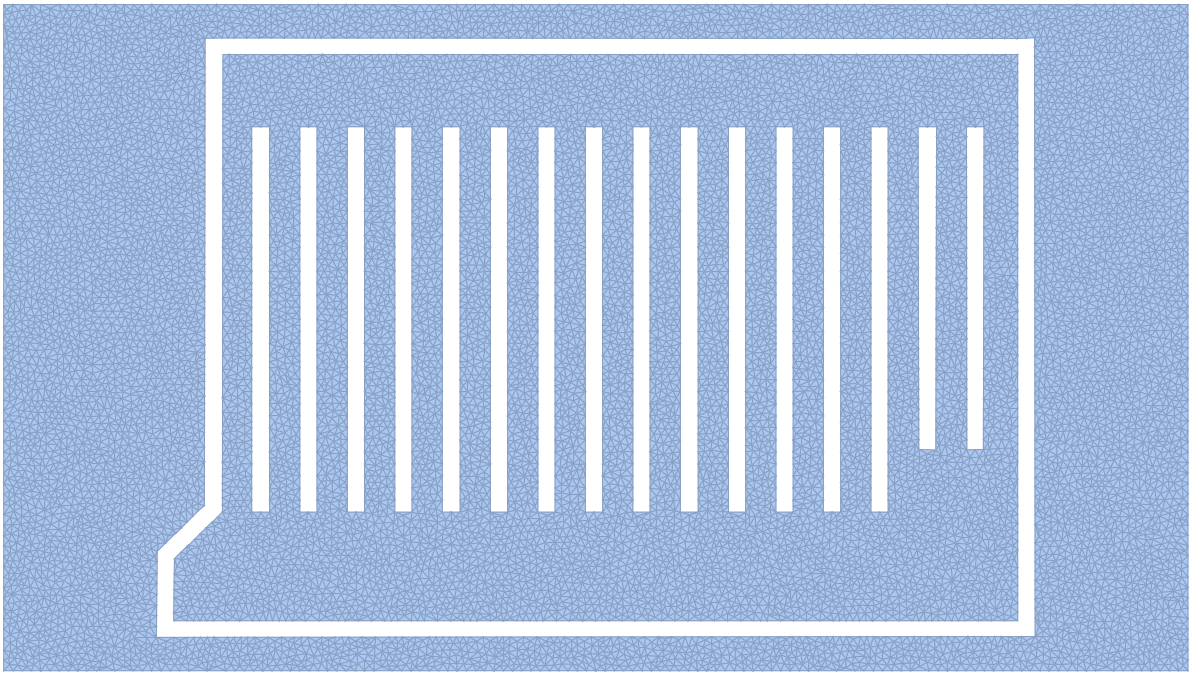
In[13]:=
```
mesh = ImageMesh[im]
```

Out[13]=



In[14]:=
```
tmesh = TriangulateMesh[mesh, MaxCellMeasure → 100];
Show[tmesh, ImageSize → Full]
```

Out[15]=

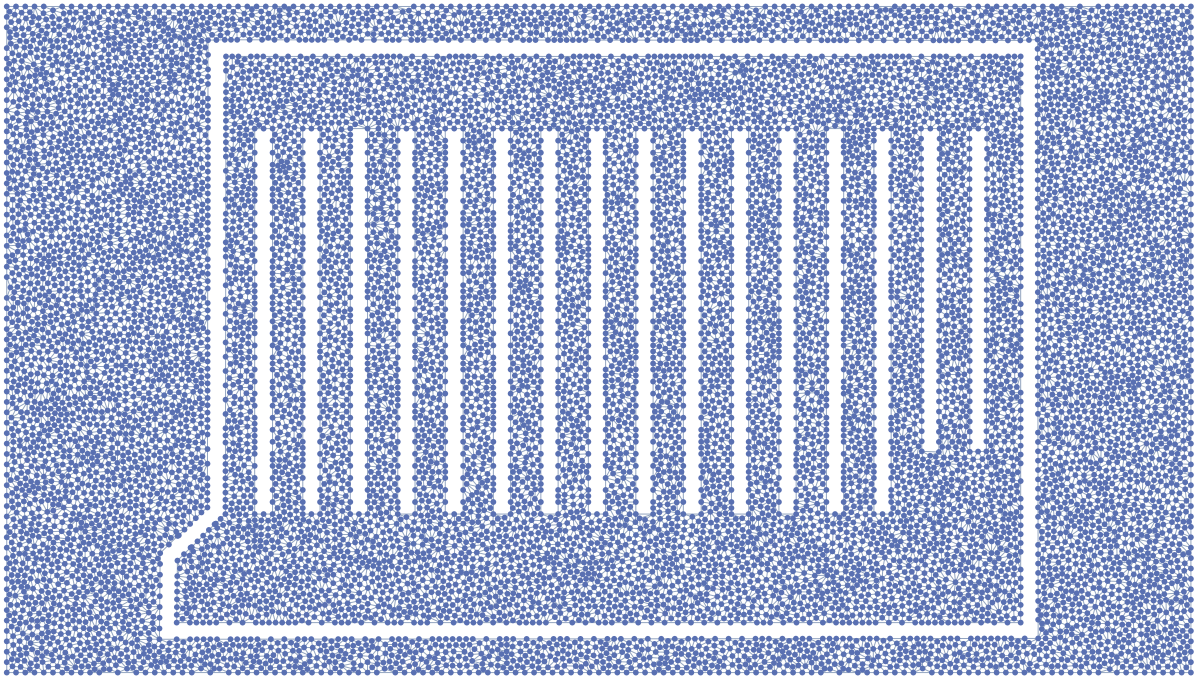


In[16]:=
```
meshGraph = MeshConnectivityGraph[tmesh, 0]
```

Out[16]=
```
Graph[ ⊞  Vertex count: 14150
Edge count: 40299 ]
```

In[17]:=
```
GraphPlot[meshGraph, ImageSize → Full]
```

Out[17]=



Get the graph vertices.
Notice that the vertices are not integers, but rather of the form {0,3}. Read this as “Point number 3”. (Because vertices correspond to points on the mesh, not lines or faces, and points are “degree 0”.)

In[18]:=
```
vlist = VertexList[meshGraph];
Length@vlist
vlist // Short
```

Out[19]=
```
14 150
```

Out[20]//Short=
```
{{0, 1}, {0, 2}, {0, 3}, {0, 4}, {0, 5},
  <<14 141>>, {0, 14 147}, {0, 14 148}, {0, 14 149}, {0, 14 150}}
```

In[21]:=
```
elist = EdgeList[meshGraph];
Length@elist
elist // Short
```

Out[22]=
```
40 299
```

Out[23]//Short=
```
{{0, 1} ⟷ {0, 13 897}, {0, 1} ⟷ {0, 13 899},
  <<40 295>>, {0, 14 145} ⟷ {0, 14 146}, {0, 14 145} ⟷ {0, 14 147}}
```

Here are the euclidean coordinates of each vertex.

In[24]:=
```
vcoords = GraphEmbedding[meshGraph];
vcoords // Short
```

Out[25]//Short=
```
{{1920., 1080.}, {0., 1080.}, <<14 147>>, {784.269, 266.766}}
```

Map vertices to their coordinates.

In[26]:=
```
v2coord = Association @@ MapThread[Rule, {vlist, vcoords}]
```

Out[26]=
```
⟨| {0, 1} → {1920., 1080.}, {0, 2} → {0., 1080.},
 {0, 3} → {0., 0.}, {0, 4} → {1920., 0.}, ··· 14 142 ···,
 {0, 14 147} → {745.668, 305.828}, {0, 14 148} → {770.708, 307.776},
 {0, 14 149} → {748.463, 280.563}, {0, 14 150} → {784.269, 266.766} |⟩
```

large output | show less | show more | show all | set size limit...

Build a new graph with the same vertices and edges, but whose edges are weighted by the euclidean distance between the vertices.

In[27]:=
```
AbsoluteTiming[
 weightedMeshGraph = Graph[
    vlist,
    elist,
    EdgeWeight → {u_ ⟷ v_ ↦ EuclideanDistance[v2coord[u], v2coord[v]]},
    VertexCoordinates → vcoords
   ];
]
```

Out[27]=
```
{90.4457, Null}
```

The weighted graph has edges that are weighted by the euclidean distance between the vertices. This is important for distance-finding. Here are two adjacent vertices:

In[28]:=
```
{v1, v2} = List @@ First@elist
```

Out[28]=
```
{{0, 1}, {0, 13 897}}
```

In the raw mesh graph, they are only a distance “1” apart:

In[29]:=
```
GraphDistance[meshGraph, v1, v2]
```

Out[29]=
```
1
```

In the weighted graph, their graph distance is the euclidean distance:

In[30]:=
```
GraphDistance[weightedMeshGraph, v1, v2]
```

Out[30]=
```
16.875
```

In[31]:=
```
v2coord /@ {v1, v2}
```

Out[31]=
```
{{1920., 1080.}, {1920., 1063.13}}
```

In[32]:=
```
EuclideanDistance @@ %
```

Out[32]=
```
16.875
```

Shopping trip

In[33]:=
```
header = Import["Grocery list - items.csv"] // First
```

Out[33]=
```
{Name, Category, Location in Kitchen, aisle, front/mid/back/backwall/endcap,
 left/right, hi/med/lo, Notes, x-coordinate, y-coordinate, 4/18/2020 - need it?}
```

In[34]:=
```
rows = Import["Grocery list - items.csv"] // Cases[{___, _Integer, _Integer, ___}];
```

In[35]:=
```
numAllItems = Length@rows
Short[rows, 10]
```

Out[35]=
```
89
```

Out[36]//Short=
```
{{entrance, n/a, n/a, 1, F, , , , 1399, 96, },
   {red wine, booze, over-toaster-cupboard, 2, M, R, M, , 1549, 755, y},
   {greek yogurt, dairy, fridge, 4, BW, , , whole milk, large, 1381, 984, y},
   {paper towels, paper products, coat closet, 5, B, L, , , 1286, 686, y},
   {cadburry chocolate eggs, candy, pantry, 6, F, L, , , 1212, 291, y},
   <<80>> , {tomatoes, produce, fridge, produce, , , , , 343, 130, y},
   {yellow onion, produce, counter, produce, , , , , 336, 172, y},
   {apples, produce, fridge, produce, , , , , 448, 124, y},
   {avocado, produce, counter, produce, , , , , 379, 124, y}}
```

In[37]:=
```
entranceCoord =
  rows // FirstCase[{"entrance", ___, x_Integer, y_Integer, ___} :> {x, y}];
exitCoord = rows // FirstCase[{"exit", ___, x_Integer, y_Integer, ___} :> {x, y}];
```

In[39]:=
```
{iName, iXcoord, iYcoord, iNeedIt} = Table[
   First@FirstPosition[header, patt, Heads → False], {patt,
    {"Name", "x-coordinate", "y-coordinate", _?(StringContainsQ["need it?"])}}
  ];
allItemNames = rows[[All, iName]];
allItemCoords = N@rows[[All, {iXcoord, iYcoord}]];
shoppingList = Pick[allItemNames, rows[[All, iNeedIt]], "y"];
PrependTo[shoppingList, "entrance"];
AppendTo[shoppingList, "exit"];
```

In[45]:=
```
item2coord = Association@MapThread[Rule, {allItemNames, allItemCoords}];
```

In[46]:=
```
item2coord["avocado"]
```

Out[46]=
```
{379., 124.}
```

In[47]:=
```
item2coord["milk"]
```

Out[47]=
```
{1146., 985.}
```

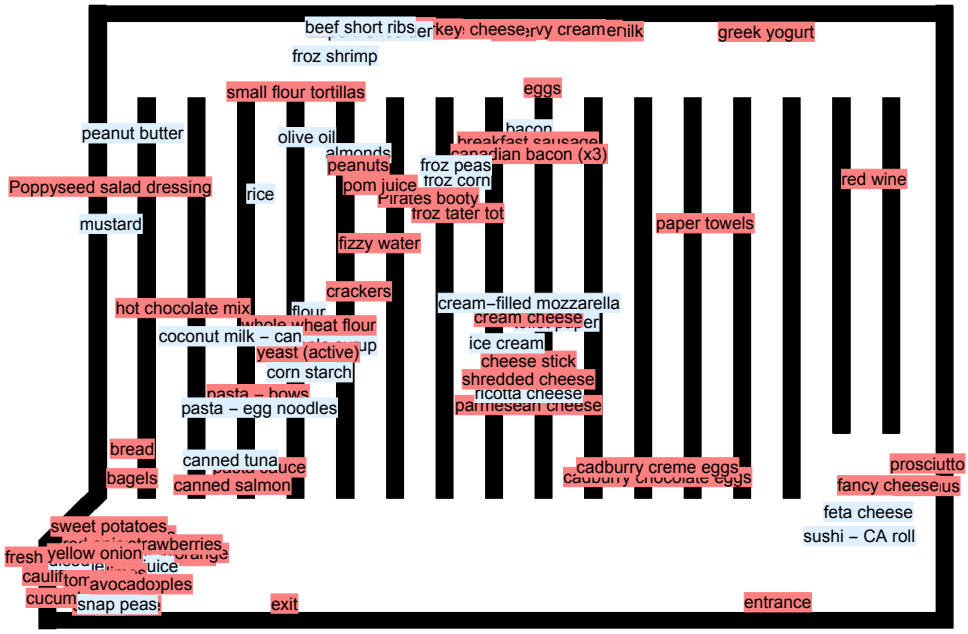
In[48]:=
```
item2coord["entrance"]
```

Out[48]=
```
{1399., 96.}
```

In[49]:=
```
gItemNames = Graphics@Table[
     Text[
      s,
      item2coord[s],
      Background → If[
        MemberQ[shoppingList, s],
        Pink,
        LightBlue
       ]
     ], {s, allItemNames}];
```

In[50]:=
```
Show[im, gItemNames, ImageSize → Full]
```

Out[50]=



In[51]:=
```
itemCoords = item2coord /@ shoppingList
```

Out[51]=
```
{{1399., 96.}, {1549., 755.}, {1381., 984.}, {1286., 686.}, {1212., 291.},
 {1212., 306.}, {1146., 985.}, {1113., 985.}, {1127., 985.}, {1067., 987.},
 {1054., 988.}, {1033., 897.}, {1010., 813.}, {1012., 794.}, {988., 985.},
 {1011., 403.}, {1011., 443.}, {1011., 472.}, {1011., 540.}, {901., 702.},
 {915., 988.}, {767., 991.}, {841., 988.}, {801., 986.}, {826., 989.},
 {855., 726.}, {746., 776.}, {747., 580.}, {779., 655.}, {780., 746.},
 {649., 891.}, {669., 527.}, {668., 485.}, {592., 306.}, {590., 420.},
 {550., 279.}, {473., 554.}, {394., 289.}, {394., 335.}, {360., 743.},
 {1631., 277.}, {1571., 278.}, {1632., 312.}, {388., 91.}, {457., 168.},
 {342., 91.}, {287., 137.}, {288., 103.}, {286., 167.}, {368., 178.}, {421., 200.},
 {385., 141.}, {502., 173.}, {376., 204.}, {342., 187.}, {354., 175.}, {461., 186.},
 {358., 214.}, {343., 130.}, {336., 172.}, {448., 124.}, {379., 124.}, {631., 94.}}
```

In[52]:=
```
itemVertices = First@NearestMeshCells[{tmesh, 0}, #, 1] & /@ itemCoords
```

Out[52]=
```
{{0, 4861}, {0, 6989}, {0, 9420}, {0, 8101}, {0, 7142}, {0, 7147}, {0, 8868},
 {0, 7834}, {0, 7831}, {0, 6206}, {0, 6200}, {0, 7222}, {0, 10152}, {0, 10140},
 {0, 6131}, {0, 7115}, {0, 7134}, {0, 7500}, {0, 9873}, {0, 7742}, {0, 6138},
 {0, 6169}, {0, 5996}, {0, 6162}, {0, 6156}, {0, 5874}, {0, 11070}, {0, 14004},
 {0, 13980}, {0, 11086}, {0, 6279}, {0, 4890}, {0, 4884}, {0, 7408}, {0, 11568},
 {0, 6510}, {0, 5628}, {0, 7047}, {0, 194}, {0, 8043}, {0, 5115}, {0, 5139},
 {0, 5118}, {0, 344}, {0, 429}, {0, 364}, {0, 5709}, {0, 5723}, {0, 373},
 {0, 184}, {0, 3215}, {0, 304}, {0, 452}, {0, 173}, {0, 204}, {0, 272},
 {0, 432}, {0, 205}, {0, 148}, {0, 188}, {0, 418}, {0, 335}, {0, 3747}}
```

Example

Find a path from the first item (“entrance”)to the last item (“exit”):

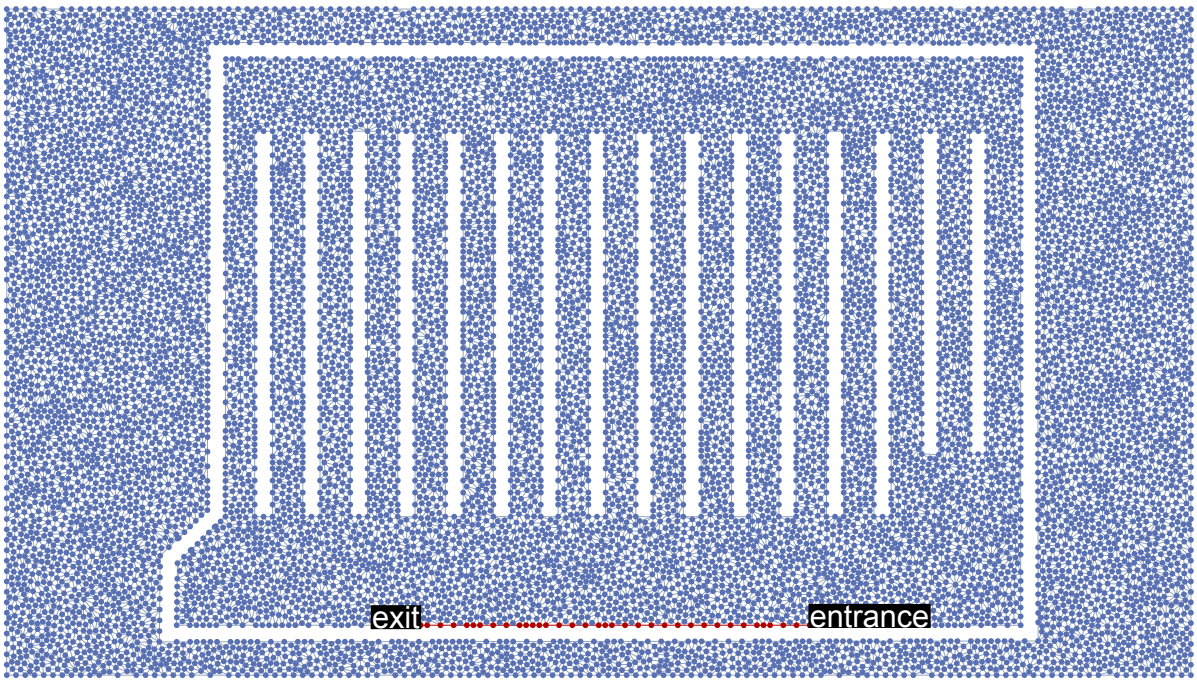
In[53]:=
```
p = FindShortestPath[
  weightedMeshGraph,
  itemVertices[[1]],
  itemVertices[[-1]]
 ]
```

Out[53]=
```
{{0, 4861}, {0, 4751}, {0, 4758}, {0, 4768}, {0, 4757}, {0, 4399}, {0, 4398}, {0, 3790},
 {0, 4731}, {0, 4219}, {0, 4221}, {0, 4226}, {0, 4191}, {0, 4204}, {0, 4203}, {0, 4122},
 {0, 3803}, {0, 4069}, {0, 3892}, {0, 4051}, {0, 3711}, {0, 3878}, {0, 3792}, {0, 3891},
 {0, 3881}, {0, 96}, {0, 3567}, {0, 3699}, {0, 3714}, {0, 3481}, {0, 5743}, {0, 3719},
 {0, 5744}, {0, 3488}, {0, 3651}, {0, 3342}, {0, 3648}, {0, 3772}, {0, 3421},
 {0, 3771}, {0, 3359}, {0, 3750}, {0, 3755}, {0, 3636}, {0, 3748}, {0, 3747}}
```

In[54]:=
```
txt = Graphics@Table[
     Text[
      Style[shoppingList[[i]], 16, White, Background → Black],
      itemCoords[[i]]
     ], {i, {1, -1}}];
```

In[55]:=
```
Show[GraphPlot@HighlightGraph[
   weightedMeshGraph,
   PathGraph[p]
  ], txt, ImageSize → Full]
```

Out[55]=



Find best order to visit the items

Idea: make a NEW graph, (a complete graph): vertices are the items, edges are weighted by shortest-dist from the mesh graph.
Then find a tour of this new graph, and map it back to the mesh graph.

In[56]:=
```
AbsoluteTiming[
 distMatrix = Table[GraphDistance[weightedMeshGraph, s, t],
    {s, itemVertices}, {t, itemVertices}];
]
```

Out[56]=
```
{34.4538, Null}
```

In[57]:=
```
itemGraph = WeightedAdjacencyGraph[distMatrix];
```

Here's the shortest path thru all the items:

In[58]:=
```
{len, path} = FindShortestTour[itemGraph, 1, Length@shoppingList]
```

Out[58]=
```
{8567.37, {1, 42, 41, 43, 2, 3, 4, 6, 5, 16, 17, 18, 19, 14, 13, 12, 7, 9, 8, 10, 11, 15, 21,
  23, 25, 24, 22, 20, 26, 30, 29, 28, 27, 31, 32, 33, 35, 34, 36, 37, 40, 39, 38,
  58, 54, 51, 50, 56, 55, 60, 49, 47, 48, 46, 59, 52, 62, 44, 61, 45, 57, 53, 63}}
```

In[59]:=
```
shoppingList[[path]]
```

Out[59]=
```
{entrance, fancy cheese, hummus, prosciutto, red wine, greek yogurt, paper towels,
 cadburry creme eggs, cadburry chocolate eggs, parmesean cheese, shredded cheese,
 cheese stick, cream cheese, canadian bacon (x3), breakfast sausage,
 eggs, milk, whole milk, almond milk, coffee creamer, heavy cream, butter,
 cottage cheese, chicken, ground turkey, pork butt, beef chuck roast,
 froz tater tot, Pirates booty, pom juice, fizzy water, crackers, peanuts,
 small flour tortillas, whole wheat flour, yeast (active), pasta - bows,
 pasta sauce, canned salmon, hot chocolate mix, Poppyseed salad dressing,
 bread, bagels, sweet potatoes, potatoes, grapes, garlic, shallot, red onion,
 yellow onion, fresh rosemary, cauliflower, cucumber, carrots, tomatoes,
 limes, avocado, bag kale, apples, blueberries, strawberries, orange, exit}
```

This should be the same distance as traversing associated vertices on the weighted mesh graph:

In[60]:=
```
GraphDistance[itemGraph, path[[1]], path[[2]]]
```

Out[60]=
```
262.221
```

In[61]:=
```
GraphDistance[weightedMeshGraph, itemVertices[[path[[1]]], itemVertices[[path[[2]]]]
```

Out[61]=
```
262.221
```

In[62]:=
```
GraphDistance[itemGraph, #1, #2] &@@@
  Partition[path, 2, 1] // Total
```

Out[62]=
```
8567.37
```

In[63]:=
```
GraphDistance[weightedMeshGraph, itemVertices[[#1]], itemVertices[[#2]]] &@@@
  Partition[path, 2, 1] // Total
```

Out[63]=
```
8567.37
```

Here's the shortest path thru the items, if you get to move through the walls:

In[64]:=
```
{lenIgnoringBoundaries, pathIgnoringBoundaries} =
 FindShortestTour[itemCoords, 1, Length@shoppingList]
```

Out[64]=
```
{5818.61, {1, 41, 43, 42, 5, 6, 16, 17, 18, 19, 4, 2, 3, 7, 9, 8, 10, 11, 12, 14, 13, 15, 21,
  23, 25, 24, 22, 31, 27, 30, 26, 20, 29, 28, 32, 33, 40, 37, 35, 34, 36, 39, 38,
  51, 54, 58, 50, 56, 55, 60, 49, 47, 48, 46, 59, 52, 62, 44, 61, 45, 57, 53, 63}}
```

However, if you account for the boundaries, then visiting the items in that order is longer than the shortest path:

In[65]:=
```
GraphDistance[weightedMeshGraph, itemVertices[[#1]], itemVertices[[#2]]] & @@@
  Partition[pathIgnoringBoundaries, 2, 1] // Total
```

Out[65]=
```
9807.88
```

(The optimal one is less:)

In[66]:=
```
len
```

Out[66]=
```
8567.37
```

Show the optimal path thru the items:

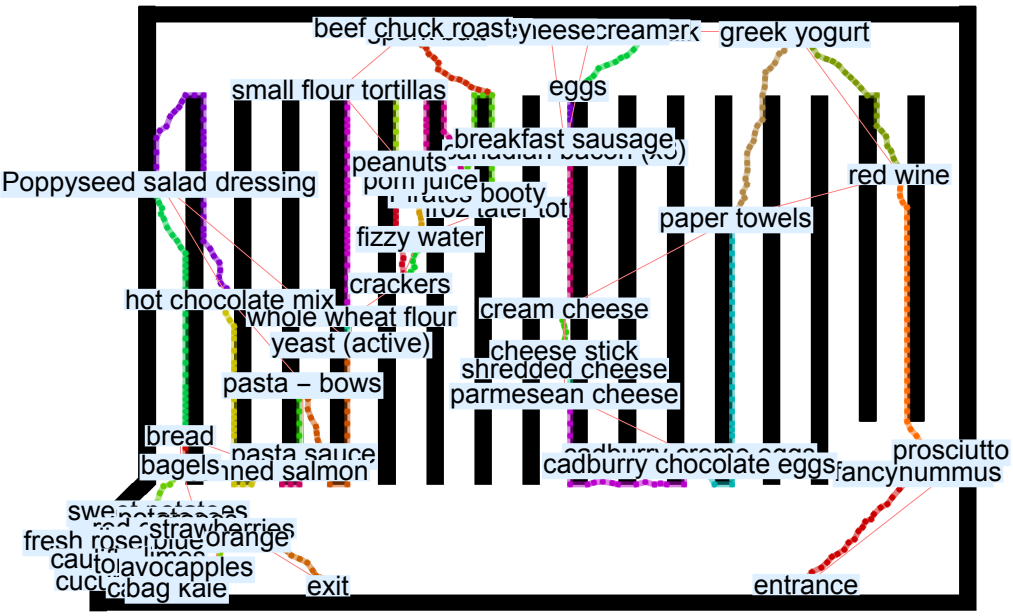
In[67]:=
```
pathGraphs = FindShortestPath[weightedMeshGraph, #1, #2] & @@@
    Partition[itemVertices[[path]], 2, 1] // Map[PathGraph];
```

In[68]:=
```
highlight2 = GraphPlot[
   HighlightGraph[weightedMeshGraph,
    pathGraphs, GraphHighlightStyle → "DehighlightHide"],
   EdgeStyle → Thickness[0.005], ImageSize → Full];
```

In[69]:=
```
gItems = Graphics@Table[
    With[{pt = itemCoords[[path]][[i]]}, {
      Text[
       Style[shoppingList[[path]][[i]], Black, 14, Plain, Background → LightBlue], pt]
     }], {i, Length@path}];
```

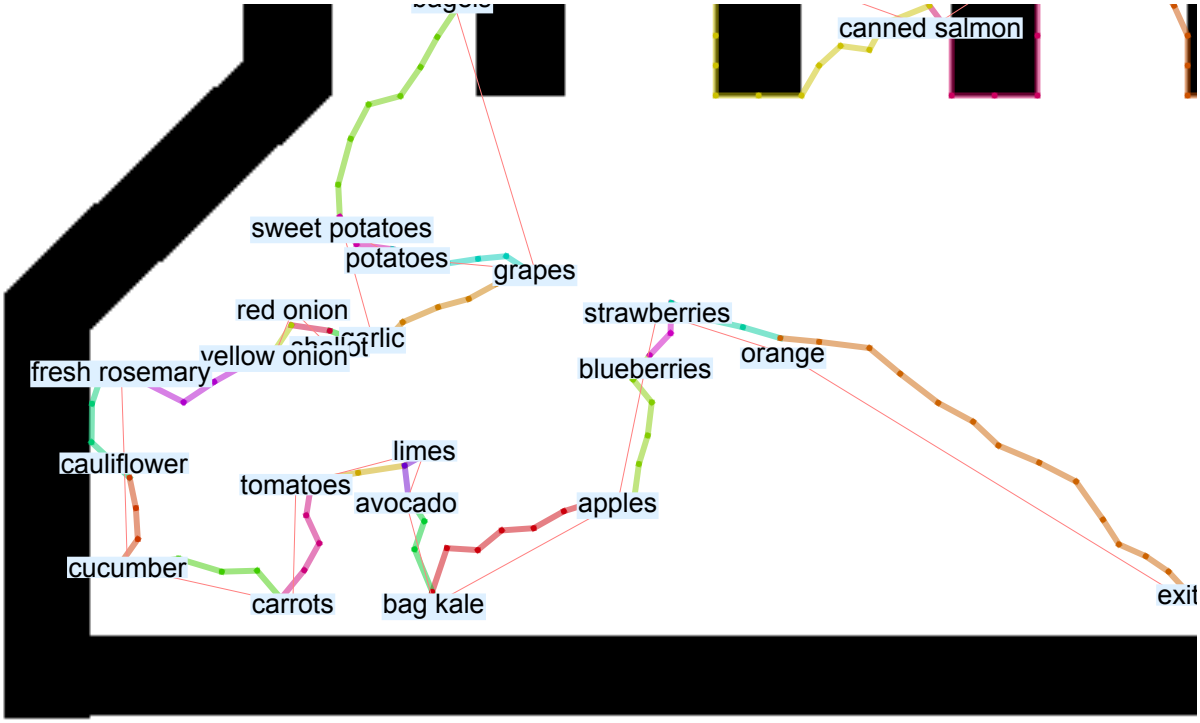
In[70]:=
```
map = Show[im,
  highlight2,
  Graphics@{Pink, Line[itemCoords[[pathIgnoringBoundaries]]]},
  gItems,
  ImageSize → Full,
  ImagePadding → 10]
```

Out[70]=



In[71]:=
```
Show[map, PlotRange → {{240, 630}, {50, 280}}]
```

Out[71]=



In[72]:=
```
Export["map-1200px.pdf", map, ImageSize → 1200]
Speak["Done"]
```

Out[72]=
```
map-1200px.pdf
```