In [1]:
```
# Indicate where the projection database is prior to importing fiona
import os
os.environ['PROJ_LIB'] = r'C:\Users\osori050\AppData\Local\ESRI\conda\envs\arc
gispro-py3-clone\Library\share\proj'
```

In [2]:
```
import arcpy
from arcpy.sa import *
import requests
import zipfile
import io
import json
from shapely import geometry
from fiona.crs import from_epsg
import fiona
import glob
```

In [3]:
```
# Set workspace
arcpy.env.workspace = r'E:\ArcGIS_1\Lab3\Updated_Dory'
workspace = arcpy.env.workspace
```

In [4]:
```
# Retrieve the Location of the North Picnic Area park from Google Places
north_picnic_link = r'https://maps.googleapis.com/maps/api/place/findplacefrom
text/json?input=North%20Picnic%20area%20St%20Charles%20Minnesota&inputtype=tex
tquery&fields=formatted_address%2Cname%2Crating%2Copening_hours%2Cgeometry&key
=YOUR_API_KEY'
north_picnic = requests.get(north_picnic_link)
north_picnic_dic = json.loads(north_picnic.text)
coords = north_picnic_dic['candidates'][0]['geometry']['location']
north_picnic_location = [float(coords['lng']), float(coords['lat'])]

# Dory's house
house = [-92.148796, 44.127985]
```

In [5]:
```
# Create shapefiles with the coordinates of the start and end points

dory_schema =  {'geometry': 'Point', 'properties': {'location': 'str'}}

with fiona.open("points.shp", 'w', crs = from_epsg(4326), driver = 'ESRI Shape
file', schema = dory_schema) as output:
    points = [geometry.Point(house[0], house[1]), geometry.Point(north_picnic_
location[0], north_picnic_location[1])]
    location = ['Start point', 'End point']
    for i in range(2):
        prop = {'location': location[i]} # Attributes
        output.write({'geometry': geometry.mapping(points[i]), 'properties': p
rop})
```

In [6]:
```
# Project to NAD83 UTM Zone 15N, create a bounding box around the start and en
d points,
# and create an 8-km buffer to consider land beyond the bounding box in the an
alysis (AOI)
arcpy.management.Project("points.shp", "points_Project.shp", 'PROJCS["NAD_1983
_UTM_Zone_15N",GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",
SPHEROID["GRS_1980",6378137.0,298.257222101]],PRIMEM["Greenwich",0.0],UNIT["De
gree",0.0174532925199433]],PROJECTION["Transverse_Mercator"],PARAMETER["False_
Easting",500000.0],PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridia
n",-93.0],PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_Of_Origin",0.
0],UNIT["Meter",1.0]]', "WGS_1984_(ITRF00)_To_NAD_1983", 'GEOGCS["GCS_WGS_198
4",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["Gr
eenwich",0.0],UNIT["Degree",0.0174532925199433]]', "NO_PRESERVE_SHAPE", None,
"NO_VERTICAL")
arcpy.management.MinimumBoundingGeometry("points_Project.shp", "polygon.shp",
"ENVELOPE", "ALL", None, "NO_MBG_FIELDS")
arcpy.analysis.Buffer("polygon.shp", "AOI.shp", "8 Kilometers", "FULL", "ROUN
D", "ALL", None, "PLANAR")
```

Out[6]:

Messages

Slope

https://gisdata.mn.gov/dataset/elev-30m-digital-elevation-model (https://gisdata.mn.gov/dataset/elev-30m-digital-elevation-model)

In [7]:
```
# Retrieve DEM from MGC
dem_output = requests.post(r'https://resources.gisdata.mn.gov/pub/gdrs/data/
pub/us_mn_state_dnr/elev_30m_digital_elevation_model/fgdb_elev_30m_digital_e
levation_model.zip')
zipfile.ZipFile(io.BytesIO(dem_output.content)).extractall(workspace)
```

In [8]:
```
# Clip DEM to AOI, create slope raster, and reclassify using geometric inter
val methods
# as increments in higher slopes are penalized more severely
arcpy.management.Clip(r"elev_30m_digital_elevation_model.gdb\digital_elevati
on_model_30m", "560098.327934821 4870356.13572893 584510.985228164 4894439.3
4794601", r"dem.tif", r"AOI.shp", "32767", "NONE", "NO_MAINTAIN_EXTENT")
out_raster = arcpy.sa.Slope(r"dem.tif", "DEGREE", 1, "PLANAR", "METER"); out
_raster.save(r"slope.tif")
out_raster = arcpy.sa.Reclassify(r"slope.tif", "VALUE", "0 3.078800 1;3.0788
00 10.647242 2;10.647242 29.252321 3;29.252321 74.988144 4", "DATA"); out_ra
ster.save(r"Reclass_slope.tif")
```

Farm fields

https://gisdata.mn.gov/dataset/agri-cropland-data-layer-2021 (https://gisdata.mn.gov/dataset/agri-cropland-data-layer-2021)

In [9]:
```
# Retrieve farm field information from the Agricultural cropland data layer
 2021 from MGC
farm_fields = requests.post(r'https://resources.gisdata.mn.gov/pub/gdrs/dat
a/pub/us_mn_state_mda/agri_cropland_data_layer_2021/fgdb_agri_cropland_data_
layer_2021.zip')
zipfile.ZipFile(io.BytesIO(farm_fields.content)).extractall(workspace)
```

In [10]:
```
# Clip to AOI and reclassify
arcpy.management.Clip(r"agri_cropland_data_layer_2021.gdb\agri_cropland_data
_layer_2021", "560098.327934821 4870356.13572893 584510.985228164 4894439.34
794601", r"farm_fields.tif", r"AOI.shp", "32767", "NONE", "NO_MAINTAIN_EXTEN
T")
out_raster = arcpy.sa.Reclassify(r"farm_fields.tif", "CLASS_NAME", "Corn 4;S
orghum 4;Soybeans 4;'Sweet Corn' 4;Barley 4;'Spring Wheat' 4;'Winter Wheat'
 4;Rye 4;Oats 4;Alfalfa 4;'Other Hay/Non Alfalfa' 4;Sugarbeets 4;'Dry Beans'
4;Potatoes 4;Peas 4;Clover/Wildflowers 3;'Sod/Grass Seed' 4;Switchgrass 3;'F
allow/Idle Cropland' 4;Apples 4;'Open Water' 4;'Developed/Open Space' 1;'Dev
eloped/Low Intensity' 1;'Developed/Med Intensity' 1;'Developed/High Intensit
y' 1;Barren 1;'Deciduous Forest' 2;'Evergreen Forest' 2;'Mixed Forest' 2;Shr
ubland 2;Grassland/Pasture 3;'Woody Wetlands' 4;'Herbaceous Wetlands' 4", "D
ATA"); out_raster.save(r"Reclass_farm_fields.tif")
```

Water

[https://gisdata.mn.gov/dataset/water-mn-public-waters (https://gisdata.mn.gov/dataset/water-mn-public-waters)](https://gisdata.mn.gov/dataset/water-mn-public-waters)

In [11]:
```
# Retrieve watercourse layers from MGC
watercourses = requests.post(r'https://resources.gisdata.mn.gov/pub/gdrs/dat
a/pub/us_mn_state_dnr/water_mn_public_waters/shp_water_mn_public_waters.zip'
)
zipfile.ZipFile(io.BytesIO(watercourses.content)).extractall(workspace)
```

In [12]:
```
# Clip to AOI
arcpy.analysis.Clip(r"public_waters_watercourses_delineations.shp", r"AOI.sh
p", r"watercourse_Clip.shp", None)

# Polyline to raster
arcpy.conversion.PolylineToRaster(r"watercourse_Clip.shp", "FID", r"watercou
rse_Raster.tif", "MAXIMUM_LENGTH", "NONE", 30, "BUILD")

# Reclassifies river
out_raster = arcpy.sa.Reclassify(r"watercourse_Raster.tif", "Value", "0 115
 4;NODATA 0", "DATA"); out_raster.save(r"Reclass_water.tif")
```

Bridges

[https://gisdata.mn.gov/dataset/trans-bridges (https://gisdata.mn.gov/dataset/trans-bridges)](https://gisdata.mn.gov/dataset/trans-bridges)

In [13]:
```
# Retrieve bridges layers from MGC
bridges = requests.post(r'https://resources.gisdata.mn.gov/pub/gdrs/data/pu
b/us_mn_state_dot/trans_bridges/shp_trans_bridges.zip')
zipfile.ZipFile(io.BytesIO(bridges.content)).extractall(workspace)
```

In [14]:
```
# Buffer to only include bridges within 30 meters of the watercourses
arcpy.analysis.Buffer(r"watercourse_Clip.shp", r"watercourse_Buffer.shp", "3
0 Meters", "FULL", "ROUND", "ALL", None, "PLANAR")
arcpy.analysis.Clip(r"Bridge_locations_in_Minnesota.shp", r"watercourse_Buff
er.shp", r"bridges_clip.shp", None)

# As the shapefile is a point vector layer, a Snap is required to place brid
ges precisely on top of watercourses
arcpy.edit.Snap(r"bridges_clip.shp", "watercourse_Clip.shp EDGE '50 Meters'"
)

# Point to raster
arcpy.conversion.PointToRaster(r"bridges_clip.shp", "FID", r"bridges__Raste
r.tif", "MOST_FREQUENT", "NONE", 30, "BUILD")

# Reclassify bridges with the same value as watercourses
out_raster = arcpy.sa.Reclassify(r"bridges__Raster.tif", "Value", "0 109 4;N
ODATA 0", "DATA"); out_raster.save(r"Reclass_bridge.tif")
```

Cost surface

This block of code creates different cost surface rasters based on the weight factors inputted by the user and saves the datasets to disk. It stops when the user specifies they do not want to create a new cost surface by typing "no" in the input box.

In [15]:
```
# Directories
arcpy.CreateFolder_management(workspace, 'cost_surfaces')
arcpy.CreateFolder_management(workspace, 'cost_distance')
arcpy.CreateFolder_management(workspace, 'cost_paths')
```

Out[15]:

Messages

In [16]:
```
# Keep track of the cost surfaces created by the user
counter = 0

while counter < 3:
    try:
        w = float(input('Enter weighting factor for farm fields'))
        x = float(input('Enter weighting factor for water'))
        y = float(input('Enter weighting factor for bridge. It can be negat
ive if you want to counterbalance the cost of crossing water bodies'))
        z = float(input('Enter weighting factor for slope'))
    except:
        raise Exception ('Please enter numeric input')

    # Change the name of the output in each cycle not to overwrite the data
sets
    counter += 1
    output_name = workspace + r"\cost_surfaces\cost_surface_" + str(counter
) + ".tif"

    # Map algebra
    algebra = w*Raster("Reclass_farm_fields.tif") + x*Raster("Reclass_wate
r.tif") + y*Raster("Reclass_bridge.tif") + z*Raster("Reclass_slope.tif")
    algebra.save(output_name)
```

```
Enter weighting factor for farm fields1
Enter weighting factor for water1
Enter weighting factor for bridge. It can be negative if you want to count
erbalance the cost of crossing water bodies-1
Enter weighting factor for slope1
Enter weighting factor for farm fields3
Enter weighting factor for water4
Enter weighting factor for bridge. It can be negative if you want to count
erbalance the cost of crossing water bodies0
Enter weighting factor for slope2
Enter weighting factor for farm fields2
Enter weighting factor for water2
Enter weighting factor for bridge. It can be negative if you want to count
erbalance the cost of crossing water bodies-2
Enter weighting factor for slope1
```

Optimal route

In [17]:
```
# Start point selection
field = arcpy.AddFieldDelimiters('points_Project.shp', 'location')
selection_1 = "{field} = '{val}'".format(field='location', val='Start poin
t')
start_point = arcpy.management.SelectLayerByAttribute('points_Project.shp',
"NEW_SELECTION", selection_1)

# End point selection
selection_2 = "{field} = '{val}'".format(field='location', val='End point')
end_point = arcpy.management.SelectLayerByAttribute('points_Project.shp',
"NEW_SELECTION", selection_2)
```

In [18]:
```
directory = workspace + '\cost_surfaces'

# Keep track of the loops
counter = 1

while counter <= 3:
    for file in glob.iglob(f'{directory}/*'):
        # Only consider tif files in the directory
        if file[-3::] == 'tif':
            # Path of the cost-distance outputs
            output_distance = workspace + '\cost_distance\cost_distance_' +
str(counter) + ".tif"
            output_direction = workspace + '\cost_distance\direction_' + st
r(counter) + ".tif"

            out_distance_raster = arcpy.sa.CostDistance(start_point, file,
None, output_direction, None, None, None, None, ''); out_distance_raster.sa
ve(output_distance)

            # Path of the optimal route
            output_path = workspace + '\cost_paths\cost_path' + str(counter
) + ".tif"

            out_raster = arcpy.sa.CostPath(end_point, output_distance, outp
ut_direction, "EACH_CELL", "FID", "INPUT_RANGE"); out_raster.save(output_pa
th)
            counter += 1
```