Object oriented software engineering: Spatial Algorithms

Lecture and Workshop 5

Gary Watmough, Peter Alexander, Guillaume Goodwin

Feedback

- You all have feedback now for each learning diary entry
- Use the feedback already recieved to help with the assignment 2
- Moderation of learning diaries is this week
- Final marks/grades delivered Friday or early next week.

Assignment 2

- Testing Coding Ability
- Testing problem solving and spatial analytical skills
- I cannot answer specific questions about what should be done as it is for you to decide what should be done
- If you are confused or stuck:
  - Sketch out the problem on paper
  - Test the existing code to see what it is doing (later tasks build on previous tasks)
  - Pseudo-code and english descriptions of the problem and data may help
  - If you are still stuck take a break!

Week by week guide

1. ~~Handling spatial data:~~
   a) ~~Simple geometric calculations, distance and bearing, range searching and data sorting.~~
2. ~~Divide and Conquer~~
   a) ~~Binary searching, recursion and line generalisation~~
3. ~~Grid data and arrays~~
   a) ~~Handling, traversing and searching raster data. Point and focal functions.~~
4. ~~Raster Analysis and Problem Solving~~
   a) ~~DEM and Flow, integrating vector and raster data and concepts~~
5. **Spatial analysis packages**
   1. **Nearest neighbour, KdTree, Gdal**
   2. **Coursework Help session**

Any problems

- Office hours:
  - Wednesday 09:00 – 11:00
- Help Session
  - Monday April 2nd 9:00 – 11:00 in here??????
- Outside these hours: contact me by email gary.watmough@ed.ac.uk

Nearest Neighbour Searching in point field

- There are lots of situations we need to search for nearest point neighbours. E.g.
  - Generating TINs
  - Point filtering
  - A ‘travelling salesman’ algorithm
- Brute force methods are okay for small datasets but can be a major limitation once datasets get larger
- Are there better ways of doing this?

How to code distance?

def distance(self, other_point):

xd= x_i - x_t

yd= y_i – y_t

return math.sqrt((xd*xd)+(yd*yd))

Where;

- xd/yd are the differences in the x/y coordinates of two points
- x_i/y_i the x/y coordinate of the *ith* point in a list and x_t/y_t is another point location we are interested in.

Example: distance

PointField

x	Y
4	3
10	3
3	6
5	7
8	6
1	7
4	10

Other Point

x	y
4	7

Differences

xd	Yd
0	-4
6	-4
-1	-1
1	0
4	-1
-3	0
0	3

$$\sqrt{(0 * 0) + (-4 * -4)}$$

$$\sqrt{(6 * 6) + (-4 * -4)}$$

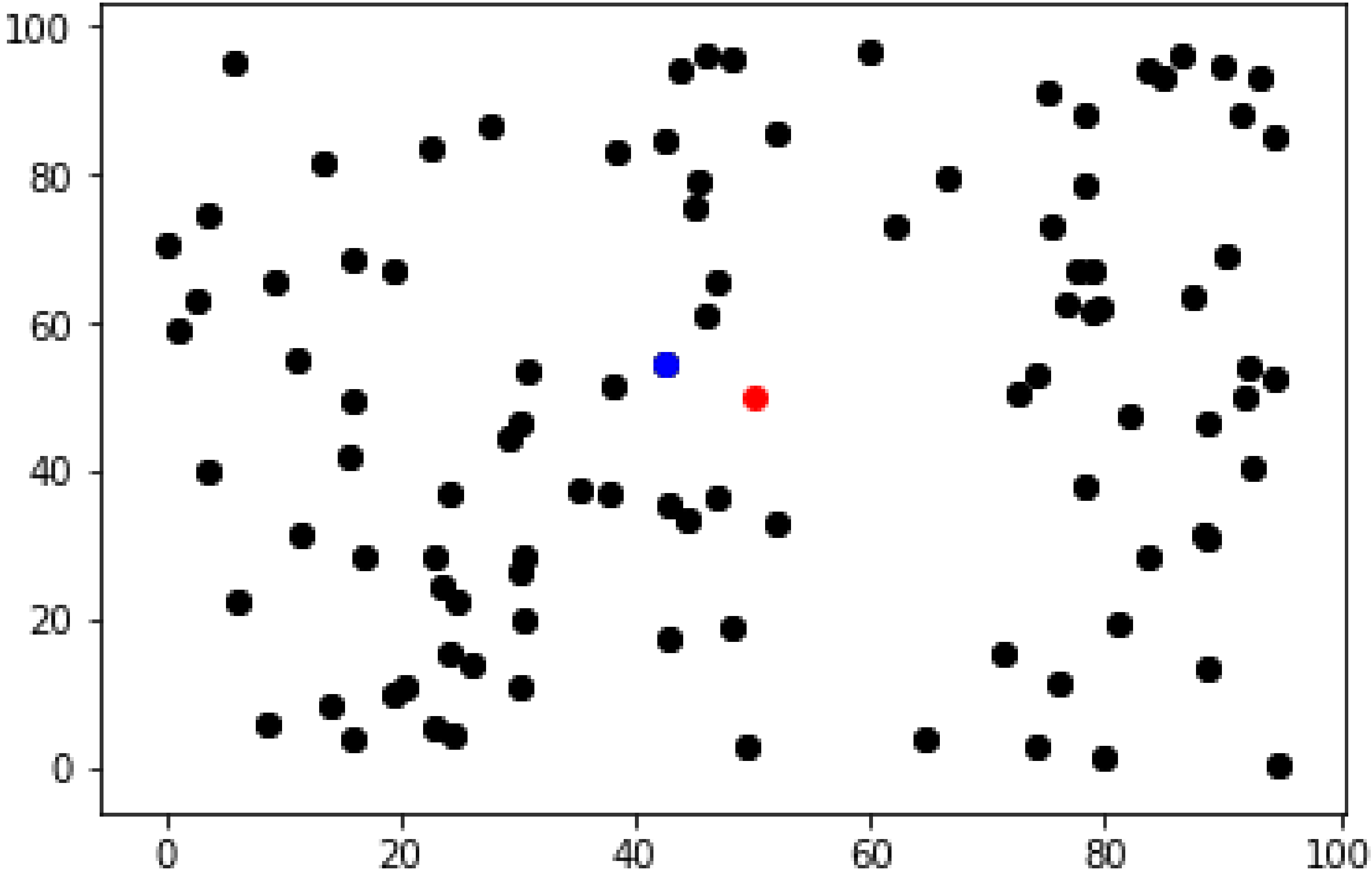
$$\sqrt{(0 * 0) + (3 * 3)}$$

Distance
4
7.2
1.4
1
4.1
3
3

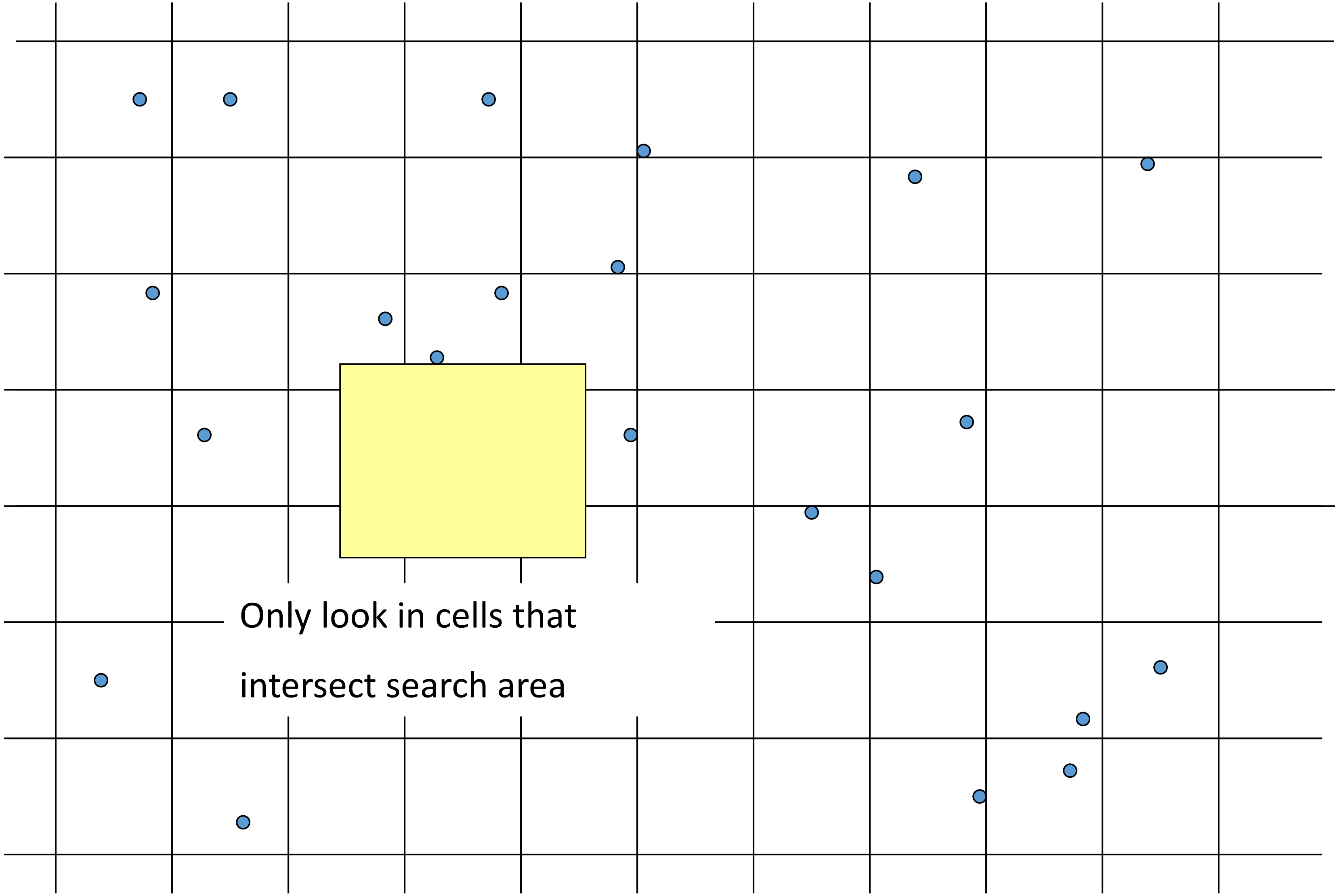
Task: How to code nearest neighbour?

- Need to hold onto the *ith* point with the lowest distance to the target in x and y.
- Open points.py
  - The nearestPoint method requires the distance to be calculated
  - The distance method name is provided in line 47
  - Currently we pass on the method.
  - Can you write a method that calculates a simple distance between a target point in X and Y and the pointfield – exactly as we just did in the previous slide.
  - Once written, open the driver NN Driver final and run the analysis

Example output

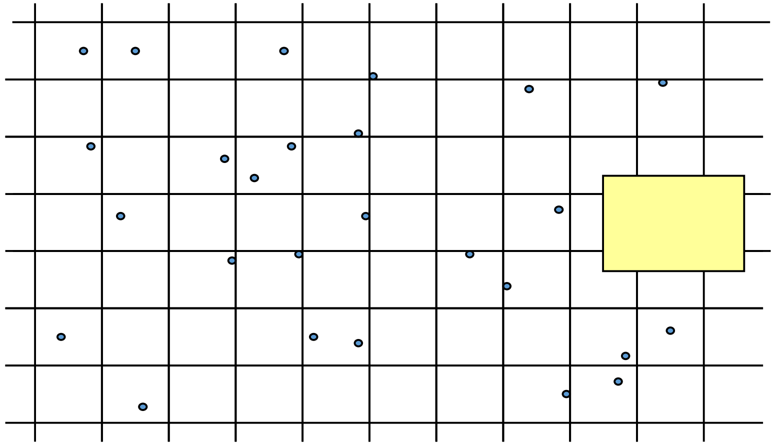


Grid Methods

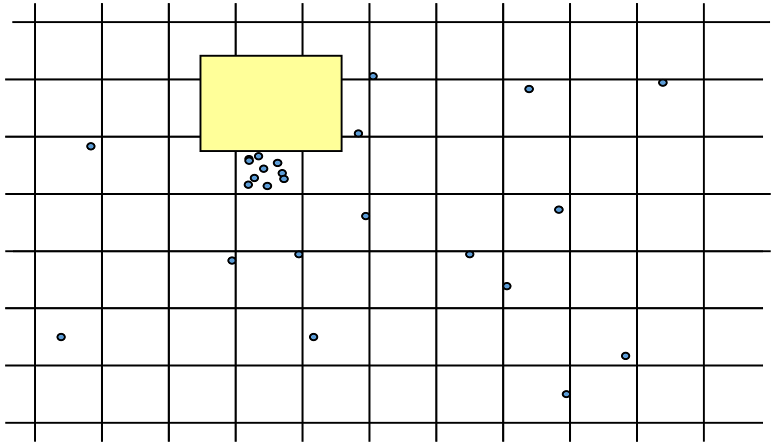


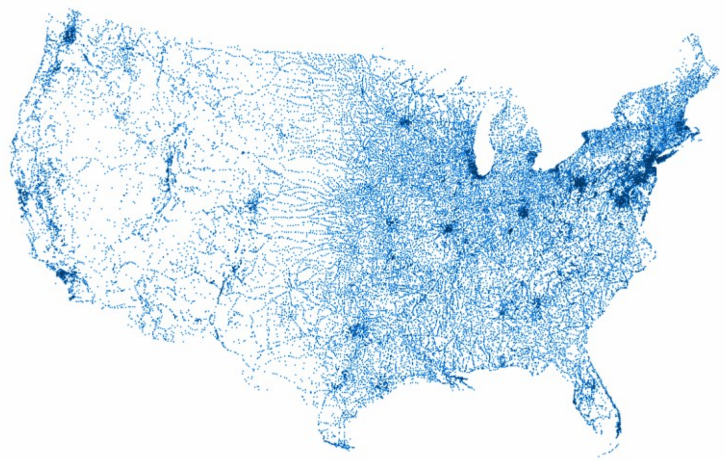
- Identify all points in each grid cell, hold in a list
- Only work on points within the cells that intersect our range.
- Grid cell size is important, too big and it includes lots of points and doesn't speed things up
- Too small – lots of objects created

Grid Method



- Works on evenly distributed points
- Not so good on uneven distributions
- If all points lie in one cell, you are not improving much
- Real data is more likely clustered





Source: https://simplemaps.com/data/us-cities

If place 1000 grid cells over the 13000 cities

Half of the cells would be empty

Half of the cities are in 10% of the cells.

Other options?

- So we need a data structure that can adapt to unevenly distributed data points
- kd tree – recursively divides space into two half planes
- Quadtree – recursively divides space into four quadrants
- Many others…

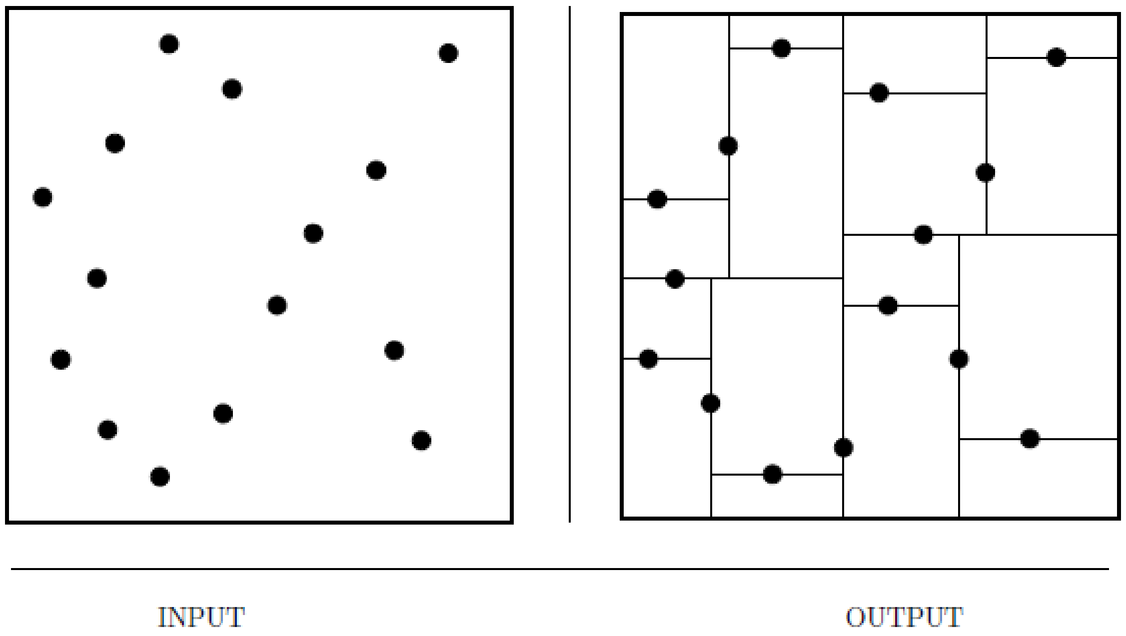
- 2d-Tree

K(2)-dimensional trees (Kd-Tree)

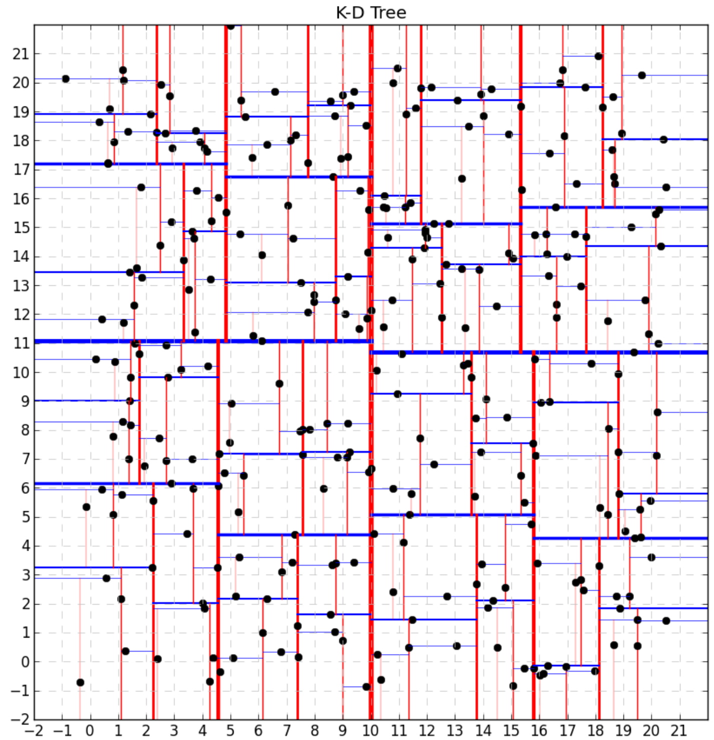
- Generalisation of a binary search tree
- Useful for range and NN searches
- Common operation in:
  - Computer vision
  - Computational geometry
  - Data mining
  - Machine learning
- Good for queries such as:
  - What is close by
  - Which is the nearest point

Kd-tree

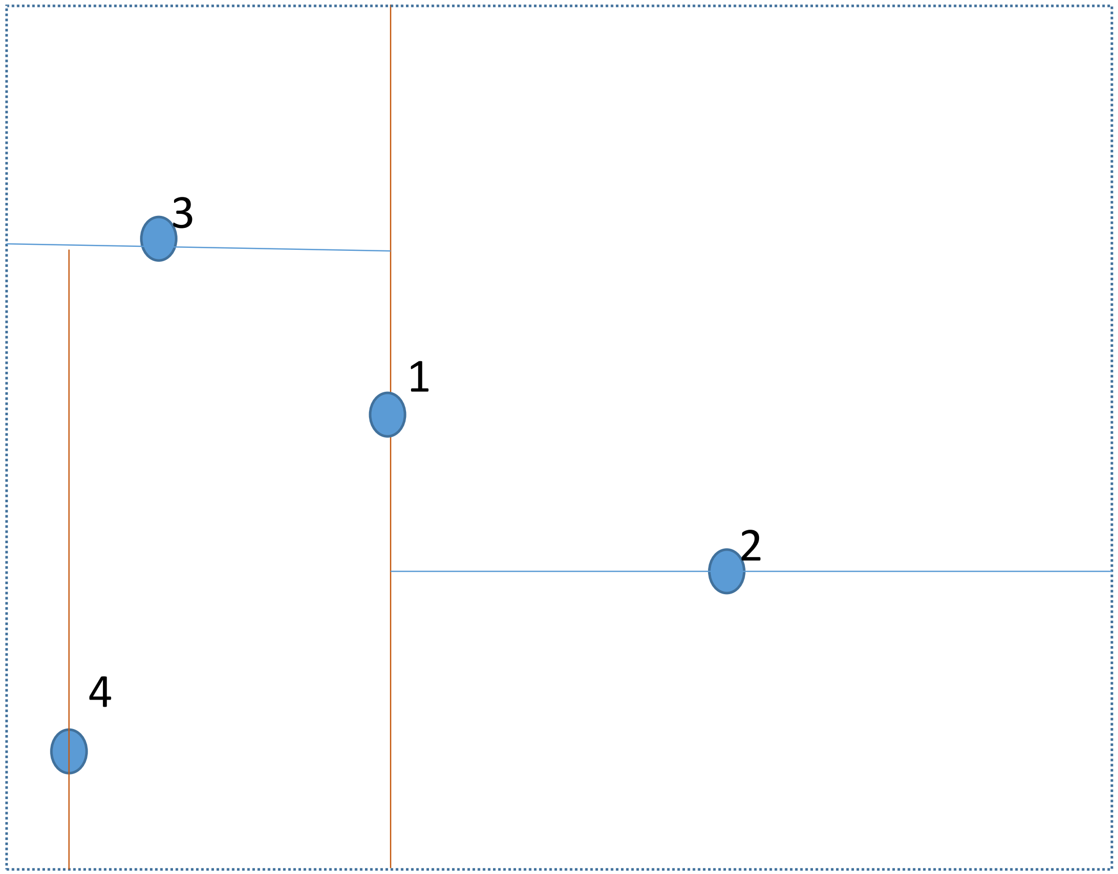
- Partition space by half-planes such that each object is contained in its own region
- Hierarchically decompose space into small number of cells each containing a few points
- Provides a fast way to access objects by position

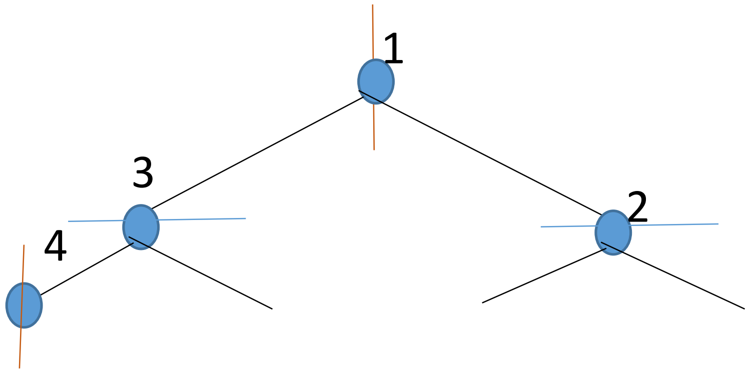


Skiena (2011) page 389.



Construction





- Switch the key (x/y coordinate) each time
- On a vertical split all points on the left of the line appear to the left of the tree node
- On horizontal split the left sub-trees are below the line and right subtrees are above.

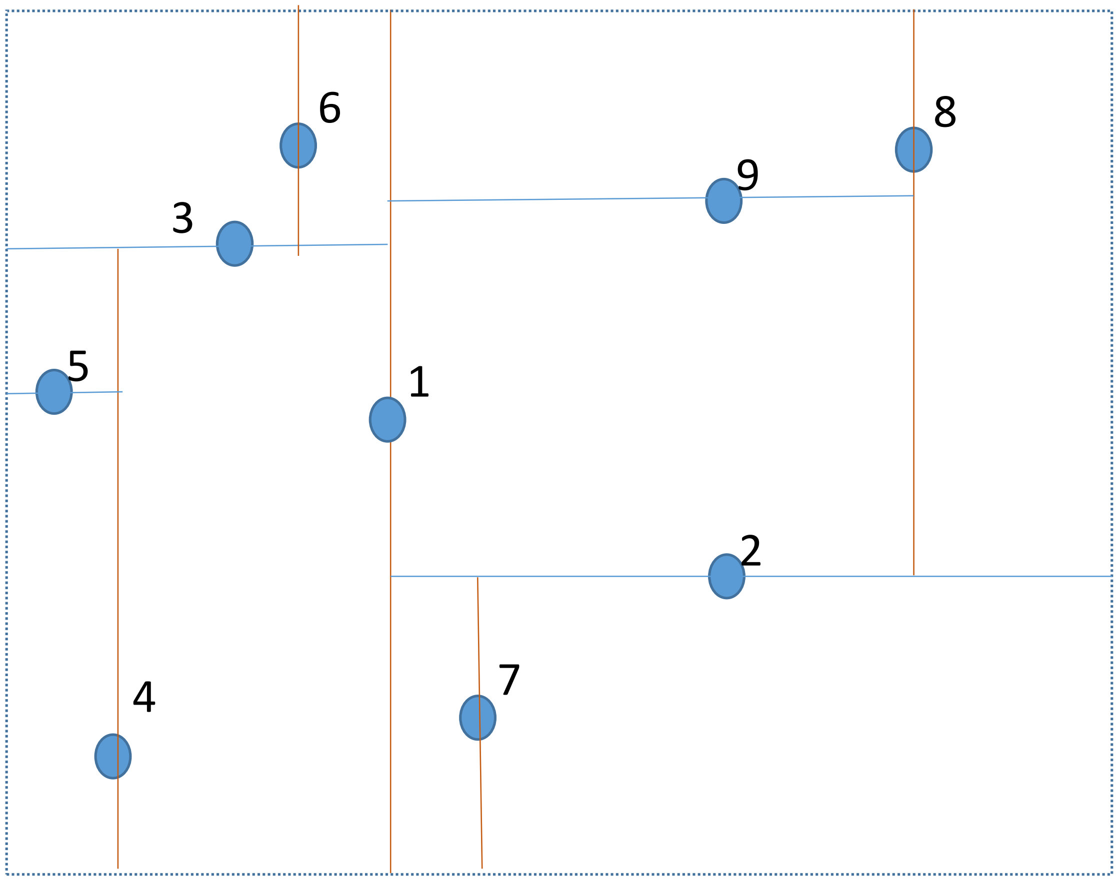
2-dimensional tree example

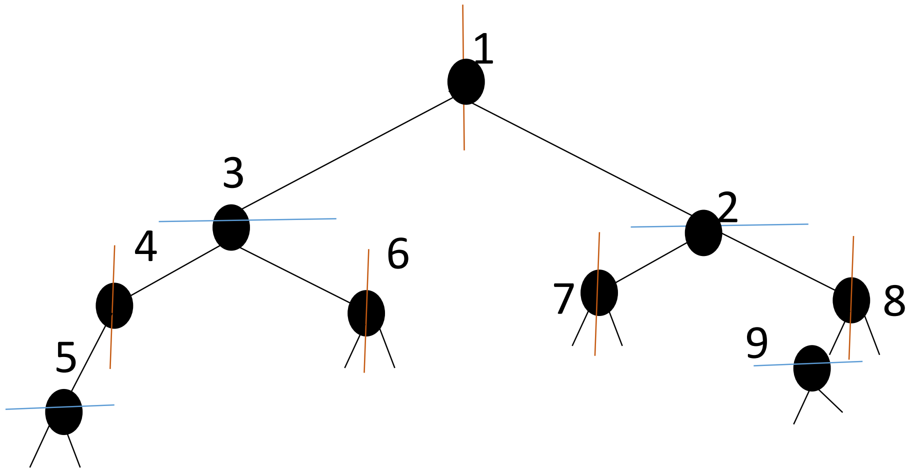
- At root of tree all data points are split based on 1st dimension
- Split by hyperplane perpendicular to the corresponding axis
- If the 1st dimension coordinate (say x) is <root it is in the left subtree
- If coordinate is >root it is in the right subtree
- At each level the tree divides on the next dimension
- Returns to the first dimension once all dimensions considered

How to build the tree (partitioning data)

- Use a partitioning method such as QuickSort
  - This places the median point at the root
  - Everything smaller to left
  - Everything larger to right
  - Repeat on left and right subtrees
  - Continue until last to be partitioned are composed of only 1 element (leafs).
- But there are other ways of constructing the tree (Skiena 2011).

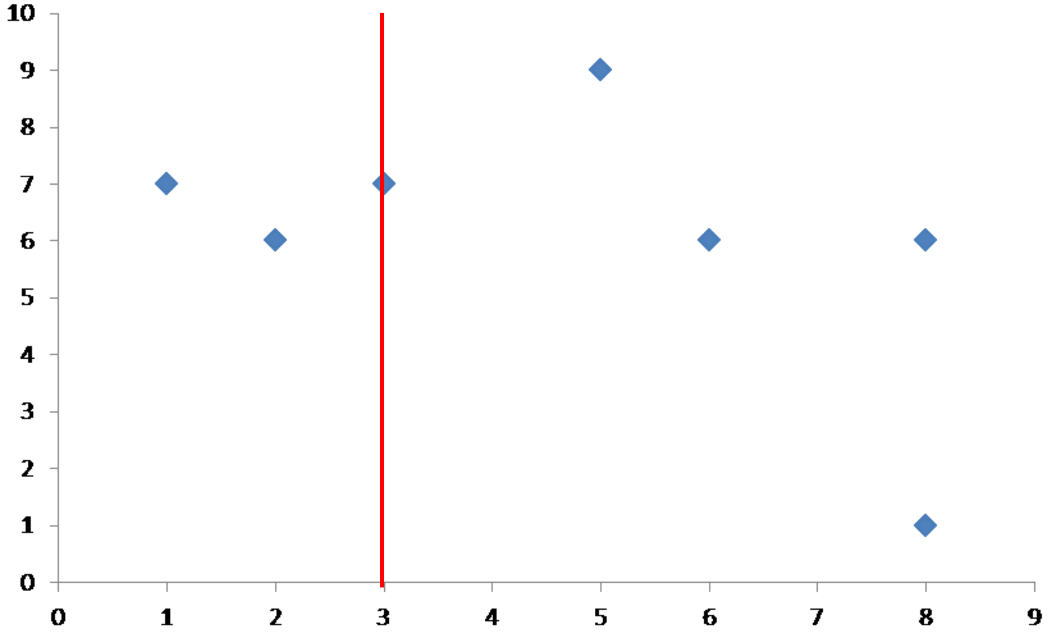
Tree construction

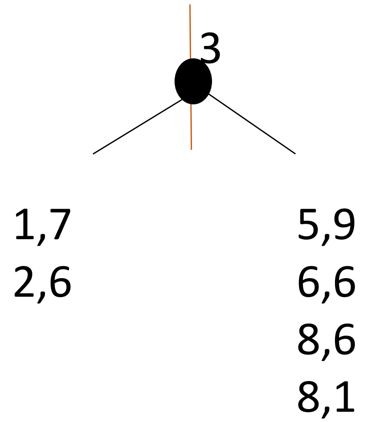




Construction: Consider a 2-d array of x,y points

X	Y
3	7
8	1
6	6
2	6
1	7
8	6
5	9

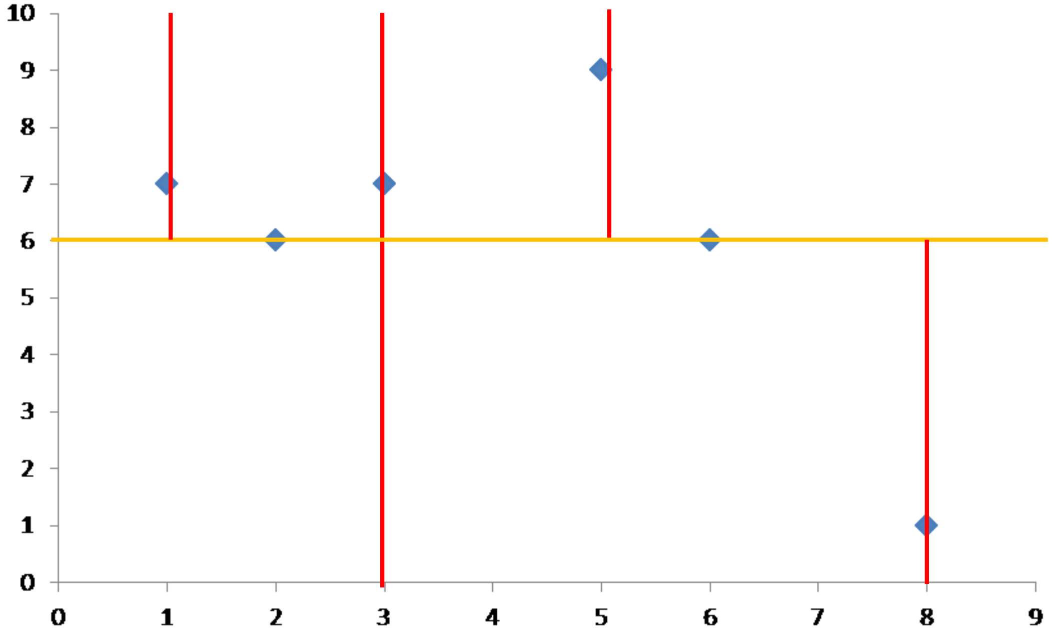


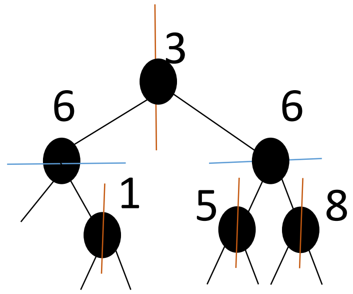


1. Root median (x): median = 3 (3,4,6,2,1,8,5))
2. Left subtree (y): median = 6 (6, 7) – floor division arbitrarily select either
3. Right subtree (y): median = 6 (9,6,6,1)
4. X coordinate = 1 (complete)
5. X coordinate = 8 (complete)
6. X coordinate = 5 (complete)

Construction: Consider a 2-d array of x,y points

X	Y
3	7
8	1
2	6
1	7
6	6
5	9





1. Root median (x): median = 3 (3,4,6,2,1,8,5))
2. Left subtree (y): median = 6 (6, 7) – floor division arbitrarily select either
3. Right subtree (y): median = 6 (9,6,6,1)
4. X coordinate: median = 1 (1,7)
5. X coordinate: median = 8 (8,1)
6. X coordinate: median = 5 (5,9)

Using the tree: traversing

- Traverse down the tree until we find the smallest cell containing our object
- Then scan through the objects in this cell to identify the right one

Nearest neighbour search

- Find point in S closest to query point q
- Perform point location to find cell c containing q (as above)
- Since c is bordered by some point p we can compute the distance $d(p,q)$ from p to q
- Point p is likely close to q
- But it might not be the single closest neighbour (if q is close to a boundary of a cell q's nearest neighbour might lie in another cell
- Therefore, we must traverse all cells that lie within a distance of $d(p,q)$ of cell c and check none contain closer points

Example: Nearest Neighbour

X	Y
3	7
8	1
2	6
1	7
6	6
5	9

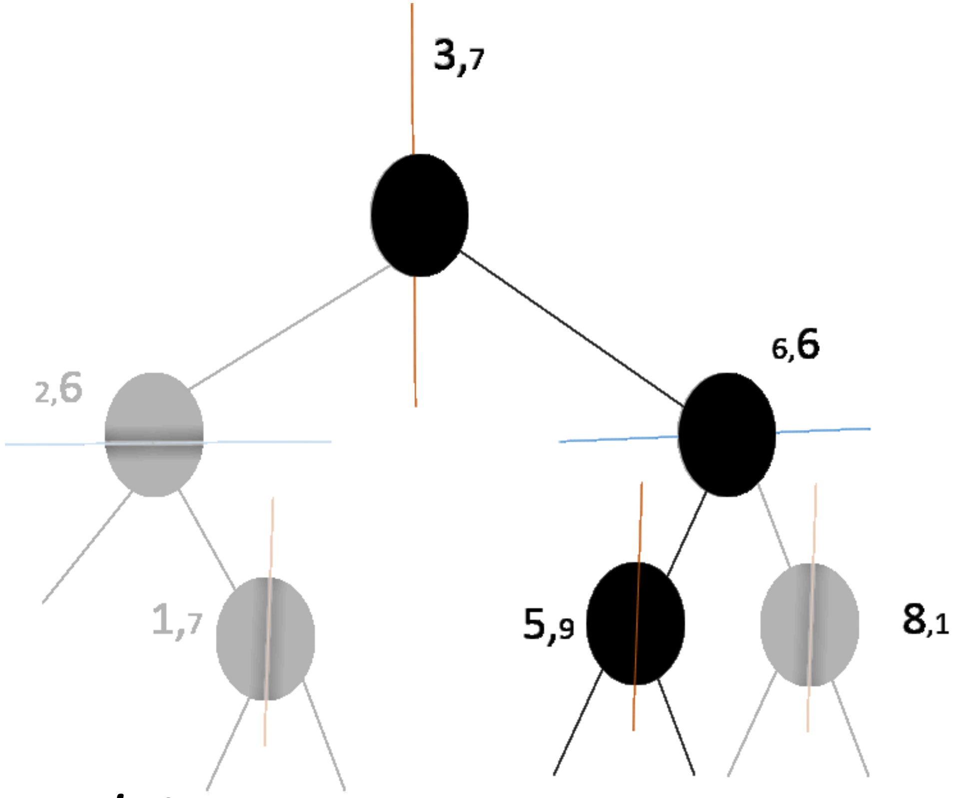
$$distance = \sqrt{(x_i - x_t)^2 + (y_i - y_t)^2}$$

Target (q) = 5,7
S = [3,7; 8,1; 2,6;1,7; 6,6; 5 9]

X domain: 5 > 3 – move to right
Y domain: 7 > 6 – move to left

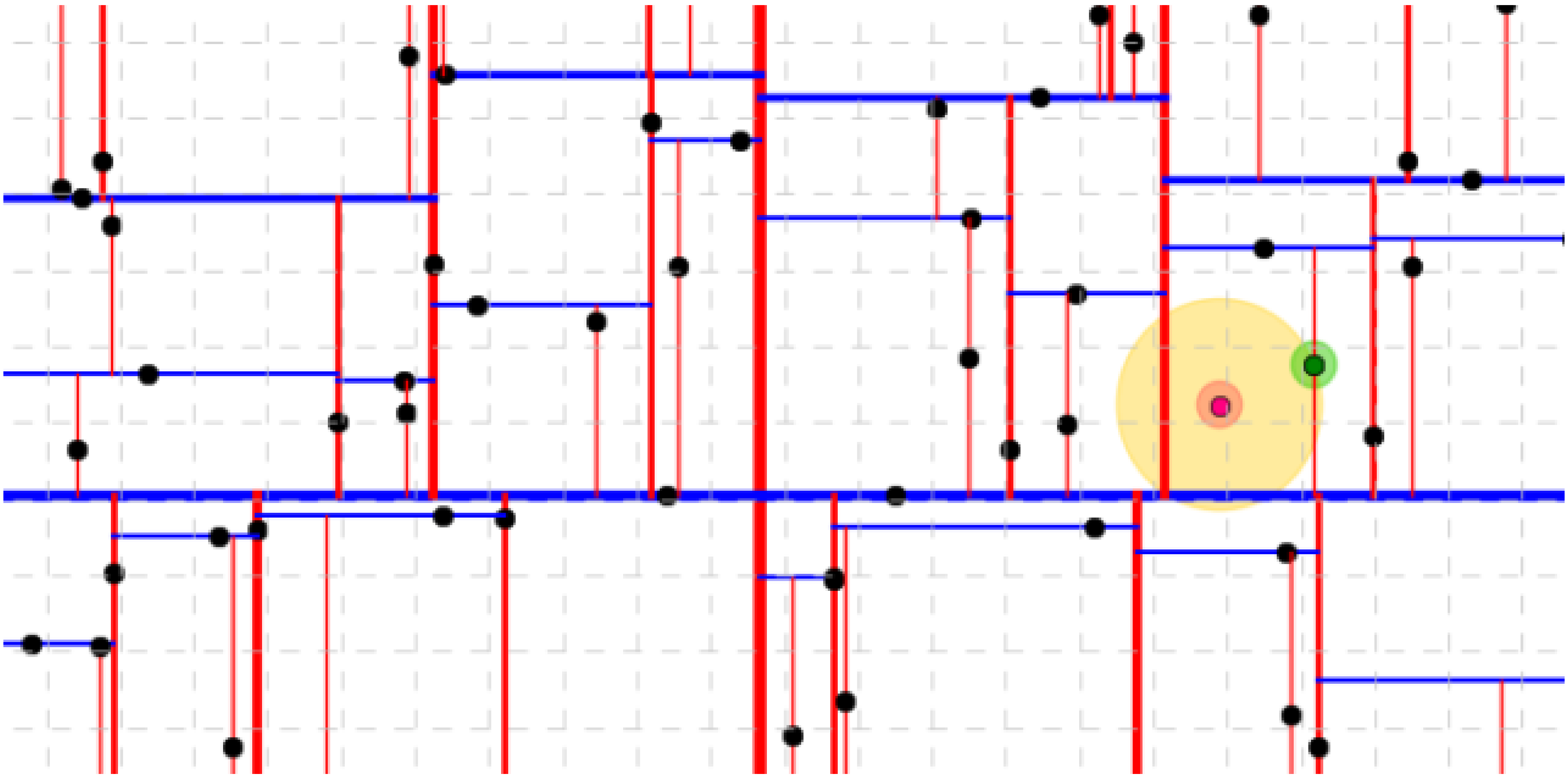
Left with: 5,9; 6,6

Trace back and test the distance's of these points



In Python

- Traversing the tree the algorithm saves the node with the shortest distance to our target as the current best
- Once the algorithm reaches the leaf node (end) it unwinds the recursion using the following steps:
  - If the current node is closer than the current best it becomes current best
  - Algorithm checks if there could be any points on the other side of the splitting plane that are closer than the current best
  - Does this by intersecting the splitting hyperplane with a hypersphere around the target point
  - This sphere has a radius equal to the current nearest distance
  - If the sphere crosses a plane there could be a point on the other side that is nearer
  - So the algorithm must also move down the other branch of the tree from the current node to check



Fortunately…. This is already part of SciPy….

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | SciPy v0.14.0 Reference Guide | Spatial algorithms and data structures (`scipy.spatial`)

index | modules | next | previous

scipy.spatial.KDTree

class **scipy.spatial.KDTree**(*data*, *leafsize=10*) [source]

kd-tree for quick nearest-neighbor lookup

This class provides an index into a set of k-dimensional points which can be used to rapidly look up the nearest neighbors of any point.

Parameters: **data** : *(N,K) array_like*

The data points to be indexed. This array is not copied, and so modifying this data will result in bogus results.

leafsize : *int, optional*

The number of points at which the algorithm switches over to brute-force. Has to be positive.

Raises: **RuntimeError**

The maximum recursion limit can be exceeded for large data sets. If this happens, either increase the value for the *leafsize* parameter or increase the recursion limit by:

Previous topic

scipy.spatial.distance.yule

Next topic

scipy.spatial.KDTree.count_neig

https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.spatial.KDTree.query.html

This has a number of methods that exploit the tree structure….

Methods

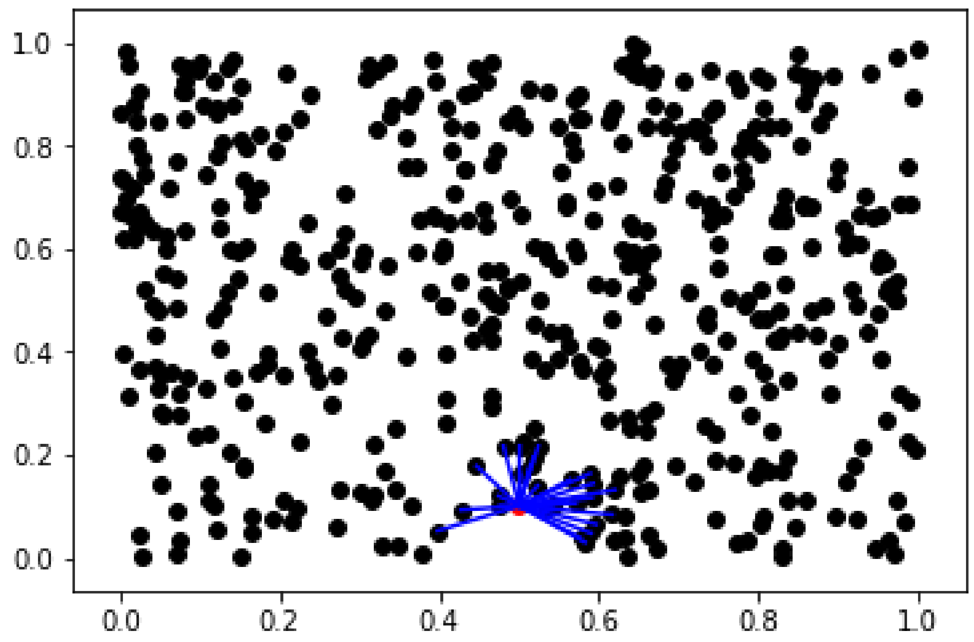
count_neighbors(other, r[, p])	Count how many nearby pairs can be formed.
innernode	
leafnode	
node	
query(x[, k, eps, p, distance_upper_bound])	Query the kd-tree for nearest neighbors
query_ball_point(x, r[, p, eps])	Find all points within distance r of point(s) x.
query_ball_tree(other, r[, p, eps])	Find all pairs of points whose distance is at most r
query_pairs(r[, p, eps])	Find all pairs of points within a distance.
sparse_distance_matrix(other, max_distance)	Compute a sparse distance matrix

The main issue is getting the data in the right format to make a tree: It's expecting a list (or array) of tuples:
$[(x_1,y_1), (x_2,y_2), (x_2,y_3), (x_4,y_4)\ldots\ldots (x_n,y_n)]$

One way is to use the zip function (see the Kdtree driver.py)

Identify nearest neighbours: **tree.query**

```
import matplotlib.pyplot as mp
import scipy.spatial as spatial
import random

x=[]
y=[]
for i in range(500):
    x.append(random.uniform(0,1))
    y.append(random.uniform(0,1))

points=zip(x,y) #converts two lists in to a tuple
print(type(points))

pointsList=list(points)
print(pointsList)

for x, y in pointsList:
    print ("x={}, y={}".format(x, y))
    mp.scatter(x,y,color="black")

myTree=spatial.KDTree(pointsList)


pointsToCheck = (0.5, 0.1)

# query for one nearest neighbour
#res = myTree.query(pointsToCheck)
#print ("Nearest point is index {}, distance {}, coordinate {}".format(res[1], res[0], pointsList[res[1]]))

# query for multiple nearest neighbour
dist, ind = myTree.query(pointsToCheck, 25)
```



Kdtree

- Useful for small to moderate number of dimensions
- Can lose effectiveness as the dimensionality increases
- Try to reduce number of dimensions to more manageable size before proceeding (dimension reduction techniques)
- More information:
  - Nice worked exampled on this blog: https://salzis.wordpress.com/2014/06/28/kd-tree-and-nearest-neighbor-nn-search-2d-case/
  - Skiena (2011) Algorithm design manual

Break Time

Next: Spatial analysis packages

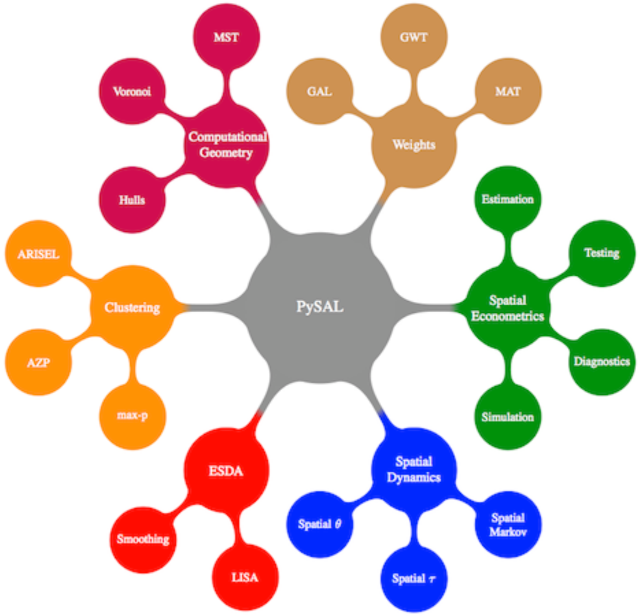
Other spatial packages

- Numpy
- Scipy
- PySal – Python Spatial Analysis Library
- Pandas – Python data analysis library
- Shapely – Computational Geometry package
- Fiona – Reading and writing geospatial data files
- Six – Python 2 and 3 compatibility library
- Gdal – Geospatial Data Abstraction Library

- Some packages are not included in different installs (anaconda)

Data handling	Analysis	Plotting data
Shapely	Shapely	Matplotlib
GDAL	Numpy, scipy	Prettyplotlib – improvements to matplotlib – no longer supported.
pyQGIS – python plugin to QGIS	Pandas, geopandas- extends datatypes in pandas to allow spatial operations	Decartes -
Pyshp – reading ESRI shapefiles	PySal	cartopy
Pyproj – converting between projections	Rasterio – read in GeoTif and other formats and store as gridded raster	
Fiona: reading and writing GIS formats	Rtree: NN search and others	
	Statsmodels: statistical modelling can it be as good as R?	

PySal: Vector data

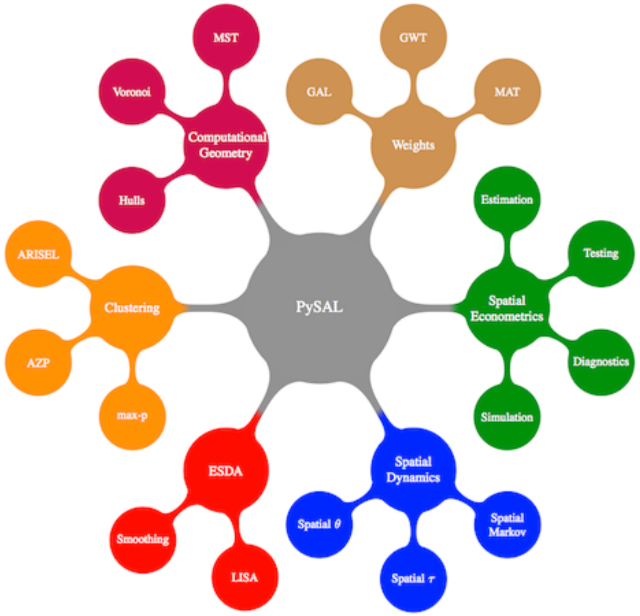


http://darribas.org/gds_scipy16/

- Not deterministic – overlays etc
- Focus on spatial statistical analysis
- Spatial Weights – express spatial relationships
  - Geographical relationships
- Computational geometry
  - Need this to format data for other analytical processes
- Clustering –
  - Finding neighbourhoods that are homogeneous and contiguous
- ESDA – exploratory spatial data analysis (autocorrelation)
  - *Is the spatial distribution of the attribute random?*

PySal

- Spatial Dynamics
  - Adding in time components as well to clustering problems for example
- Spatial econometrics
  - Spatial regression techniques



http://darribas.org/gds_scipy16/

Rasterio: raster geoprocessing and data analysis

- Raster Manipulation
  - Stacking and merging bands
  - Calculations across bands
    - Vegetation indices
  - Conversions from different types of raster file types
- It does require several other libraries/packages including gdal.

GeoPandas

- Vector geoprocessing
  - Buffer,
  - Intersect
  - Union
  - Difference
- Requires other python packages
  - Numpy, pandas, shapely, Fiona, six
- Can be difficult to install with some versions or setups.

Packages and Libraries

- Often find others have already had similar questions/problems
- Worth searching online for pre-existing algorithms or approaches before you begin coding something new

Coursework help session

We cannot answer all of the questions