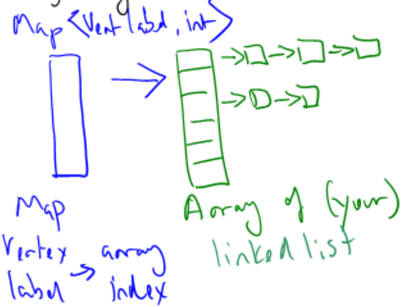
Assignment 1 Discussion

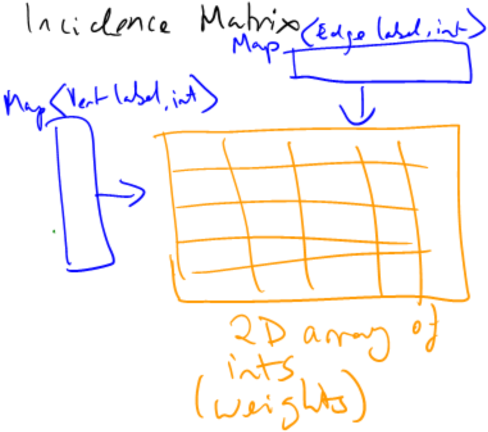
- Maps and where it should be used.
- inNearestNeighbour() and outNearestNeighbour().
- Running python code.
- Generating Data.
- Report.

Maps and where it should be used

Adjacency list



Incidence Matrix



inNearestNeighbour() and outNearestNeighbour()

- List<MyPair> inNearestNeighbour(...)
- You can use any List (including ArrayList) to return for inNearestNeighbour.
- NO Need to write your own list.

Running Python code

Generating Data

Aim: to generate different data distributions to test your data structures.
Two parts:

- Generate graphs to do scenario operations on.
    - Option 1: Use the assocGraph.csv file we gave you (next slide talk about structure).
    - Option 2: Generate your own graph using a random graph generator (we have suggested some possibilities in specs).
- For each scenario, generate a number of operations (e.g., weight updates for scenario 3), for each data structure, to evaluate performance. Test on each graph density, L, M, H.

Generating Data (Graphs of various densities)

Example:

- I choose to use existing graph, and load it using '-f' option.
- I start with scenario 3.
- I treat initial density as "L", then generate "M" by adding a number of random (directed) edges. Do the same for "L", with "L" having more edges.
- Do this over a number of graphs, say 3 per density.

Generating Data (Operations for scenarios)

Example:

- For each graph generated in previous step (9 say), I random pick an existing edge in that graph and generate an operation to update that weight. (U command) Repeat this for a largish number of operations.
- Time updates over all operations and average.
- Report this average time.

Task B – Report

graph:

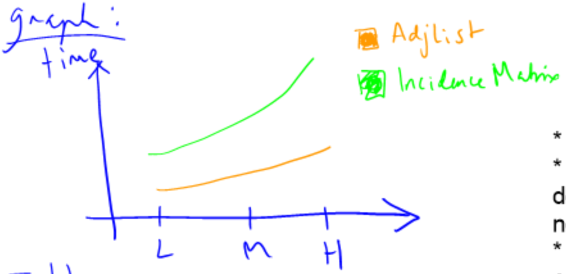


Table:

	L	m	H
Adj list	t_1	t_2	t_3
IncMat	t_4	t_5	t_6

* Compare data structures
* Compare trends as change density (or k in nearest neighbour)
* Relate to what you know about the data structures and your implementation
* Relate to the theoretical complexities
* Draw conclustions