Assignment 1

- Topic: Implementing data structures for directed, weighted graphs and performance evaluation
- Assessment - correct implementation, code commenting (task A) and evaluation (task B)
- Skeleton code, testing script and example test instances provided.
- work in pairs.

Outline

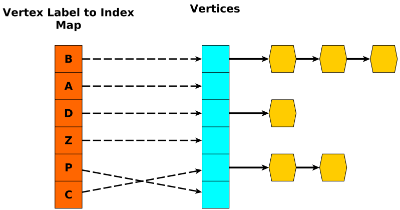
Focus on Task A this week, talk about task B next week.

- Discuss implementations
- Discuss how to run the script and the interactive mode

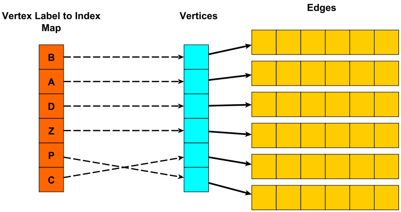
Assignment 1 – Implementation

- Implement two graph representations, *Adjacency list* and *Incidence matrix*.
- Implement k-nearest neighbour operation

Assignment 2 – Data structures

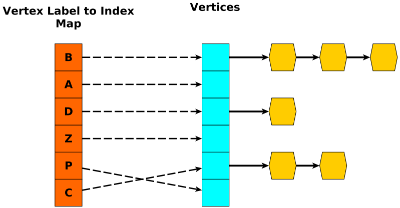


(a) Adjacency list.

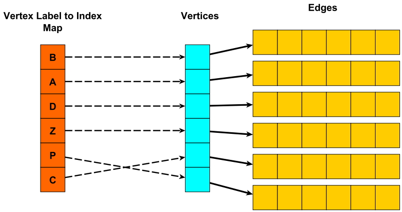


(b) Incidence Matrix.

Assignment 2 – Data structures



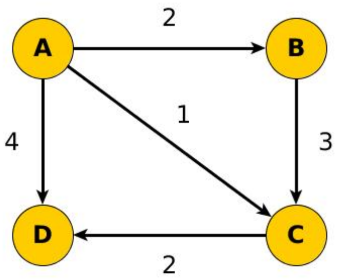
(c) Adjacency list.



(d) Incidence Matrix.

- **Do not** use java.util.LinkedList or java.util.ArrayList etc for implementing the array or linkedlist.
- Instead use the primitive array types (X[])
- For the vertex label map, you can use java.util.Map and children.

Assignment 2 – K-nearest Neighbour



- All out-neighbours of A?
- 2 nearest out-neighbours of A?
- All in-neighbours of D?
- 1 nearest in-neighbour of D?

Assignment 2 – Skeleton Code

Compile code first: javac -cp .:jopt-simple-5.0.2.jar *.java

Now to run?

- $< \mathit{impl} >= [\mathit{adjlist}|\mathit{incmat}]$
- Iteractive mode:
  ```
  > java -cp .:jopt-simple-5.0.2.jar GraphEval <impl>
  ```
- "Non-Iterative" mode (output saved to files):
  ```
  > java -cp .:jopt-simple-5.0.2.jar GraphEval <impl> vert.out edge.out neigh.out misc.out
  ```
- Non-Iterative mode (output saved to files, input from test input):
  ```
  > java -cp .:jopt-simple-5.0.2.jar GraphEval <impl> vert.out edge.out neigh.out misc.out < test1.in
  ```
- Load a file beforehand:
  ```
  > java -cp .:jopt-simple-5.0.2.jar:sample.jar GraphEval -f assocGraph.csv <impl>
  ```

Assignment 2 – Skeleton Code

Compile code first: javac -cp .:jopt-simple-5.0.2.jar *.java
Now to run?

- $< impl >= [adjlist|incmat]$
- Iteractive mode:

  ```
  > java -cp .:jopt-simple-5.0.2.jar GraphEval <impl>
  ```

- “Non-Iterative” mode (output saved to files):

  ```
  > java -cp .:jopt-simple-5.0.2.jar GraphEval <impl> vert.out edge.out neigh.out misc.out
  ```

- Non-Iterative mode (output saved to files, input from test input):

  ```
  > java -cp .:jopt-simple-5.0.2.jar GraphEval <impl> vert.out edge.out neigh.out misc.out < test1.in
  ```

- Load a file beforehand:

  ```
  > java -cp .:jopt-simple-5.0.2.jar:sample.jar GraphEval -f assocGraph.csv <impl>
  ```

On Windows system, you may need to change the ‘:’ to ‘;’ in the classpath.

Assignment 2 – Python script

Now to run?

- Basic:
  > python assign1TestScript.py -v Assign1-s1234 <impl> tests/test1.in
- Use the given association data as initial graph:
  > python assign1TestScript.py -v -f (absolute path)/assocGraph.csv Assign1-s1234 <impl> tests/test2.in (this won’t pass tests but this might be useful mode to use for task B)

Assignment 2 – Python script

Now to run?

- Basic:

  ```
  > python assign1TestScript.py -v Assign1-s1234 <impl> tests/test1.in
  ```

- Use the given association data as initial graph:

  ```
  > python assign1TestScript.py -v -f (absolute path)/assocGraph.csv Assign1-s1234 <impl> tests/test2.in
  ```

  (this won’t pass tests but this might be useful mode to use for task B)
- The script can test more than one test input at a time.

Assignment 2 – Python script

Now to run?

- Basic:
  > python assign1TestScript.py -v Assign1-s1234 <impl> tests/test1.in
- Use the given association data as initial graph:
  > python assign1TestScript.py -v -f (absolute path)/assocGraph.csv Assign1-s1234 <impl> tests/test2.in (this won't pass tests but this might be useful mode to use for task B)
- The script can test more than one test input at a time.
- The script will compare the output (test1.vert.out, test1.edge.out, test1.neigh.out and test1.misc.out) with the expected output (test1.vert.exp, test1.edge.exp, test1.neigh.exp and test1.misc.exp)