DATA ANALYSIS II

Data Structures for Network Representation

2021-22

Sources

- Newman, M. (2010). *Networks: an introduction*. Oxford University Press. [275-298]

Issues

- The first task of most programs that work with network data is to read the data, usually from a computer file, and store it in some form in the memory of the computer.

- The way the data are stored in the computer memory after they are read from the file can make a substantial difference to both the speed of a program and the amount of memory it uses.

Storing Network Data

- To label the nodes so that each can be uniquely identified by a numeric label, usually an integer.
- Nodes could also have properties like age, capacity, or weight represented by other numbers, integer or not.
  - All of these other notations and values can be stored straightforwardly in the memory…
- We need a way to represent the edges in the network. This is where things get more complicated.

Computational Complexity

- The computational complexity of an algorithm is an indication of how the algorithm's running time scales with the size of its input.
- *Time Complexity*: How long does this algorithm run?
- *Space Complexity*: How many memory cells this algorithm needs?
- How to store a network? We have to balance time and space complexity…

Adjacency Matrix

- Do we need all cells?
  - Directed and undirected networks…
- How effective?
  - Weighted and unweighted networks…
- What complexity?
  - *O(1)*? *O(n)*?

Sparse Networks

- A disadvantage of the adjacency matrix representation is that for sparse graphs it makes inefficient use of computer memory.
  - Sparse matrix implementation.
  - *DoK* (Dictionary of Keys) representation.
- What is maximum size of such adjacency matrix?
- Most current research on networks however is focused on larger data sets, and for these another representation is needed.

Problem: Neighbors

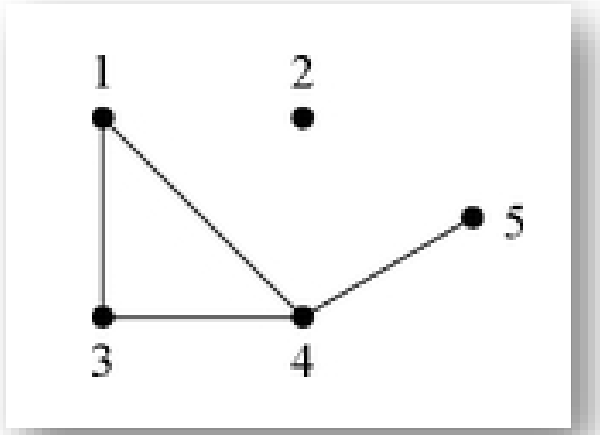
- Many network algorithms require to find all neighbors of a node, often repeatedly.

- For such algorithms the adjacency matrix is not an ideal tool.

Adjacency List

- Probably the most widely used network representation for computer algorithms.

- A set of lists, one for each node. How many neighbors?



Vertex	Neighbors
1	3, 4
2	
3	4, 1
4	5, 1, 3
5	4

Representation

- An adjacency list can be stored in a series of integer arrays or as a two-dimensional array with one row for each node.
- It is common to also store the degree of each node, so that we know how many entries there are in the list of neighbors for each node.
- There is no requirement that the neighbors of a node in the adjacency list appear in numerical order.
- Time complexity of “find” operation is $O(m/n)$.

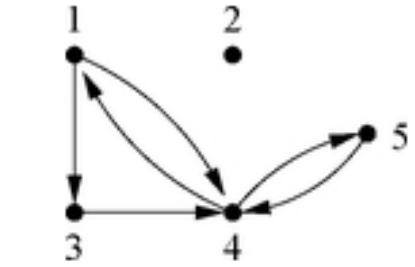
Memory

- Each edge appears twice…
- To represent m edges, therefore, we need to store $2m$ integers.
- However, the double storage turns out to have other advantages, making our algorithms substantially faster and easier.
- An adjacency list can store networks with multi-edges or self-edges.

Directed Networks

- Adjacency lists can be used with directed networks as well.

- This approach is guaranteed to capture every edge in the network. Each edge now appears only once in the adjacency list, not twice as in the undirected case.



Vertex	Incoming edges
1	4
2	
3	1
4	3, 1, 5
5	4

Vertex	Outgoing edges
1	3, 4
2	
3	4
4	5, 1
5	4

Weighted networks and other attributes?

- Both, nodes and edges can be weighted…
- In adjacency list representation, weights (and *other attributes*) can be stored in extra structures.
- *Edge list*: A list of the labels of pairs of vertices that are connected by edges.
- For instance, the edge list representation would be *(1, 3), (4, 1), (4, 3), (4, 5)*. The order of the edges is usually not important nor is the order of the vertices in the node pairs.
- It is worthwhile to create an additional data structure that stores the properties separately (in memory, file or database).

Problem: Concrete Neighbor

- Finding of a concrete neighbor has $O(m/n)$ complexity.
- Compared with the adjacency matrix with $O(1)$ complexity.
- Algorithms such as calculating the clustering coefficient require this operations and can be slowed by the use of an adjacency list.

Adjacency Tree

- A data structure that has most of the advantages of the adjacency list, while being considerably faster in many situations.

- An *adjacency tree* is identical to an adjacency list except that each "row" - the set of neighbors of each node – is stored in a tree.

Tree of Neighbors

- The goal is to use trees to store the lists of neighbors of each node in a network.
- Four basic network operations - addition, removal, and finding of edges, and enumeration of the complete set of a node's neighbors - can be performed very quickly on average.
- *Binary Search Tree*: Time complexity of "find" operation is $O(log(m/n))$.
- The neighbor tree should be balanced.

Summary: Time Complexity

Operation	Adjacency matrix	Adjacency list	Adjacency tree
Insert	$O(1)$	$O(1)$	$O(\log(m/n))$
Delete	$O(1)$	$O(m/n)$	$O(\log(m/n))$
Find	$O(1)$	$O(m/n)$	$O(\log(m/n))$
Enumerate	$O(n)$	$O(m/n)$	$O(m/n)$

- Dynamic approaches can be used (objects, hash sets, array lists,…). Is there a problem or not?

Hybrid Representation

- Hybrid matrix/list is a representation that consists of an adjacency matrix and an adjacency list.
- Non-zero elements in the adjacency matrix, those corresponding to edges, are accompanied by pointers that point to the corresponding elements in the adjacency list.
- Why to use *Hybrid Representation*? Graph algorithms.
- It is suitable only for relatively small networks. Why?

Large-scale Graph Data Processing Platforms

- Pregel (Google)
  - https://www.researchgate.net/profile/James_Dehnert/publication/221257383_Pregel_A_system_for_large-scale_graph_processing/links/00b7d537c615821fa4000000.pdf
- Apache Giraph (Facebook)
  - https://giraph.apache.org/
- Cassovary (Twitter)
  - https://blog.twitter.com/engineering/en_us/a/2012/cassovary-a-big-graph-processing-library.html