Betweenness Centrality

SD : Data Structures

Graphs are everywhere ! Transportation maps, biological processes and interactions, computer networks, the Internet, social networks, are all represented (should I say abstracted ?) as graphs.

Graphs are mathematical objects. A graph $G = (V, E)$ usually defined as a set of vertices $v \in V$ and a set of edges $e \in E$ connecting pairs of vertices.

Many types of graphs exist. They can be directed or undirected ; edges can be weighted or not ; edges and vertices can be static or dynamic over the lifetime of the graph ; some graphs do support loops or self-loops, other do not (trees), …Different models are suited for different application domains.

Graphs have properties. They can be perfect, planar, bipartites, …and a multitude of metrics can be computed over them (density,order, size, …).

In this unit we are interested in identifying particular nodes in undirected graphs. In computer networks, for example, some nodes are more important than others (as it is in social networks or any type of network).

To define what importance means, several metrics can be used to define and locate important nodes. These metrics are usually defined ingraph theory. One such concept is called **Centrality**. Several types of centrality exist in graphs (Katz, Pagerank, Closeness, …) all covering a different way to define what a central node actually is.

In this unit, we are interested in **betweenness centrality**, a centrality that allows to identify the importance of a node in a graph from its impact on the possible flow of information among all nodes in the network that follow a shortest path communication.

The exercises defined below aims at building a program to compute the betweenness centrality of all nodes in a graph.

The betweenness centrality of a vertice v is defined as follows :

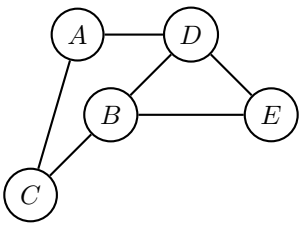
$$c_b(v) = \sum_{s,t} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s, t)-paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s, t.

In the first part, we will do some background exercises on graphs, their representation and algorithms that exploit them. During the lab, students are then asked to do the implementation. In this unit, you will :

- design and implement algorithms to build graphs,
- write a set of operations on those graphs and analyse their complexity (time and space),
- implement algorithms in the C language,
- understand and assess the importance of representation and the combination of multiple data structures to reach your objectives.

This is a typical graph :



This graph will be used throughout the lab to illustrate the algorithms.

★ **Exercice 1 :** Graph Representation

▷ **Question 1 :** Provide a graphic representation of the given graph using :

- an **adjacency matrix**,
- an **adjacency list**,
- an **incidence matrix**.

▷ **Question 2 :** Provide the Abstract Data Type of a graph data structure.

★ **Exercice 2 :** Graph building

In many problems, graphs need to be synthetically built. One way of doing it is to do it in a recursive way through dichotomy.

▷ **Question 1 :** Write a recursive algorithm that, given a target graph of size $n = 2^p$ number of vertices, does the generation in a recursive way. To simplify the algorithm, we consider that we connected two sub-graphs only through one edge.

▷ **Question 2 :** Does the resulting graph have a special property ?

▷ **Question 3 :** Propose an extension of the algorithm to explore alternative graphs.

▷ **Question 4 :** The Erdöz - Rényie binomial random graph

In 1959, Paul Erdöz and Alfréd Rényie worked on different models of random graphs. One of them, is called the **binomial random graph** $G(n, p)$, defined as follows :

A $G(n, p)$ graph is a graph holding n vertices and a maximum of $n(n-1)/2$ undirected edges (an edge between A and B and between B and A is counted only once). In such a graph, each edge exists with a probability of p.

Write a short algorithm that builds a $G(n, p)$ graph using an adjacency matrix (you suppose you have such a adjacency[n][n] structure on hand).

★ **Exercice 3 :** Graph exploration

▷ **Question 1 :** Provide the order of exploration of vertices in the graph given on page 1 when :
- — starting the exploration from vertice A,
- — doing a Breadth First traversal.

▷ **Question 2 :** Write an iterative algorithm to explore the graph following a BFS policy.

▷ **Question 3 :** Write a recursive algorithm to explore the graph following a DFS policy.

▷ **Question 4 :** Write an algorithm that does a search from a source vertice to a destination vertice following a BFS policy

★ **Exercice 4 :** BFS-based shortest path

▷ **Question 1 :** Write an algorithm that computes the shortest path between a source and a destination following a BFS policy traversal.

▷ **Question 2 :** Extend the algorithm to collect all shortest paths between a source and a destination.

★ **Exercice 5 :** Betweenness centrality

With the previous questions answered, you now have all the algorithmic material to implement the betweenness centrality computing in C.

▷ **Question 1 :** Choose your graph representation structure between adjacency matrix and adjacency list and build the associated C representation (sizes are dynamic....)

▷ **Question 2 :** Write a function that computes the betweenness centrality of a given node

▷ **Question 3 :** Write a function that locates the node with the highest betweenness centrality

★ **Exercice 6 :** For the braves …

▷ **Question 1 :** Test your code on real size graphs

▷ **Question 2 :** Start from scratch and compute the closeness centrality and/or the degree centrality (some are easy to compute) …