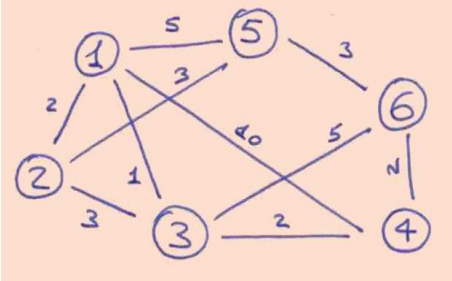
DIU-EIL Computer Networks **Univ. Rennes 1**

Paths on Computer Networks

The main aim of this assignment is to explore the different possible ways for a message, sent by a computer device belonging to a particular network, to find its best route among the other computers of the network. We won't be talking too much about *computers* and *networks*, but we will rather focus on the *graph* representation of the network.

All the exercises below will be based on the following graph G (the curious student may want to try, subsequently, to apply them to other graphs).



Shortest paths on graphs

First we'll explore some algorithms for shortest paths on graphs.

- Find the shortest path on the graph G between the vertex 1 and the vertex 6. Just look at the graph structure, it is not necessary to apply a specific algorithm.
- Your shortest path should not contain the vertex 5. Define and draw the subgraph H containing only the vertices of the shortest paths identified above, plus the vertex 5 (an edge of G is in the subgraph if and only if, by definition, the two corresponding vertices are also in the subgraph).
- Apply the Bellman-Ford algorithm to rediscover the shortest path between the vertex 1 and the vertex 6 on the subgraph H. You can "execute" the algorithm in a graphical way, by keeping track of the label values on a drawing of the graph. Did you perform all the iterations of the algorithm's main loop? Do you think we can reduce the number of iterations while keeping it effective for all kinds of graphs?
- Now apply Dijkstra algorithm for the identification of the shortest path on H. Again, you can run the algorithm in a graphical way. When does the algorithm *realize* that the longest path is not the good one?
- Modify the graph H so that the edge between the vertex 5 and the vertex 6 has now weight -4. Run again, and compare, the two algorithms.

Routing tables

Routing tables are used in computer networks to keep information about the shortest paths that they can be identified in the network. On every graph vertex, they do not contain the information

about the *entire* path towards a destination vertex, but rather they store information about the next vertex to be visited in order to reach a given destination.

- Compute the routing tables for all vertices of our graph G. For every destination vertex, the information about the *next vertex* should be provided.
- By exploring the result obtained above, we may wonder whether it is actually necessary to have a dedicated line in such tables for every distinct destination vertex. What is your opinion?

Flooding on graphs

The routing tables allow us to predefine the shortest path for a message to travel through the vertices of a graph so that a vertex can "communicate" with another. This approach allows us to compute the shortest paths only once, and to keep this information in the routing tables as far as the graph does not change its structure.

However, this approach does not take into consideration that the network is a dynamical entity: one of its vertices may be overcharged, or even currently not functional. If a message attempts taking a path that is supposed to pass through one of these vertices, the path that will be chosen will not be the most efficient ...

Flooding is therefore an alternative approach to finding paths on computer networks, where the best route is identified dynamically, on the basis of the current status of the network.

- Suppose that vertex 1 needs to send a message to vertex 6 over our graph G. To this aim, the vertex 1 sends this message to all its neighboring vertices; such neighboring vertices will then be in change to send the message to their own neighboring vertices, until the destination vertex is reached. If the sent message contains the information about the sending and destination vertex (that the other vertices will exploit to decide whether to forward the message or not), how many messages are sent in total over the graph?
- In order to reduce the total number of unnecessary messages that are sent over the graph, we decide to introduce another information in the messages: the *hop limit*, which establishes the maximum number of times that the message can be sent forward by the receiving vertices. Compute the total number of messages with hop limit equal to 2 and 1. What can you remark?
- Let's verify in more details what happens in vertex 3. Are there chances that vertex 3 forwards more than once the same message to another vertex? If yes, how to avoid this problem?

Network fault

Suppose that vertex 1 still wants to send a message to vertex 6, but vertex 5 is currently not working properly.

- What approach will be able to better deal with this problem? Routing tables or network flooding? And how?
- Suppose that the vertex is still working but it is overcharged. In this case vertex 5 can send a message to its neighbors and tell them to avoid sending more messages. However, can you see in what situation this approach is not optimal?