1 Phase King algorithm

The algorithm runs for $f+1$ phases, where f is the number of malicious processes. We need $f < \texttt{ceil}(n/4)$. In phase $1 \leq 1 \leq f+1$, the process with process id p_i is the "king", which is used to break ties.

1.0.1 Example run with 2 faulty, 7 processes

This algorithm cannot work in this case. We need $f < \texttt{ceil}(n/4)$, or $n > 4f$. However:

$$(n = 7) \underset{?}{>} (4f = 4 * 2 = 8) = \texttt{false}$$

It is easy to observe why. Assume we have this allocation of 0/1 values to the processes, where a M above a process means it is malicious.

Id	1	2	3	4	5	6	7
Malicious?						M	M
Value	0	0	1	1	1	0	0

In this case, in the non-malicious userspace, we have only 2 0s, and three 1s. Thus, there is not enough information

1.0.2 Preferred over oral message

Message complexity: For this algorithm, we only need polynomial number of messages. We have $f+1 \simeq O(n)$ phases, and in each phase we send $O(n^2)$ messages. So in total, we need $O(n^3)$ messages.

This is in contrast to the oral message algorithm where we need to send $O(n^f)$ messages, which is not polynomial in n (it is pseudopolynomial).

2 Distributed MST: Gallager-Humblet-Spira

2.0.1 Complexity

2.0.2 Making it work with duplicate edge weights, unique node IDs

We can augment the edge weight information used for naming fragments with (edge-weight, source-node-id, dest-node-id). As long as the node IDs can be totally ordered, the full triple will be totally ordered, and will give us unique names for edges. This will ensure that we receive unique names for fragment combinations.

2.0.3 Example

2.0.4 Does the algorithm work with non-unique edge ids, node weights?

We can make it work by slightly jittering the edge weights. For each edge $e \in E$ make a new edge $e' \in E'$ as $e' \equiv e + \epsilon_e$, where ϵ_e is some small, random weight that is added to the edge e. This ensures that our edge weights will be distinct.

Alternatively, we can have nodes choose random IDs. This has very small probability of failing; Once this is done, we can then run the algorithm as explain with "duplicate edge weights, unique node IDs".

If we do not accept small chances of error, then this is impossible. Consider 3 nodes (which non-unique IDs) in a triangle where all edges have weight 1. No node can distinguish itself from its neighbours. Hence, if the code for one node does something, it will do the same thing for all of them.

For any node, its outgoing edges form a minimum spanning tree. Let us say some node $n \in \{1, 2, 3\}$ chooses to make a spanning tree. The code which is the same on the other two nodes, and has no way of distinguishing n from n' will also execute the same code. Thus, the nodes cannot "agree" on a spanning tree.

3 True/False Question: Distributed MST

3.0.1 Question:

True/False: Let G be a graph and let T be its MST. The minimum distance between any two nodes $u, v \in V(G)$ as measured in the minimum spanning tree $d_T(u, v)$ gives a good (non-trivial) upper bound on the minimum distance between u and v in the original graph, $d_G(u, v)$.

Alternatively, is the relation $d_G(u, v) < d_T(u, v)$ useful for all choices of G, u, v, T, where T is guaranteed to be a minimum spanning tree of G.

3.0.2 Answer:

No it does not; consider G to be N nodes arranged in a ring. Label the vertices $u_0, u_2, \ldots u_{N-1}$. We can make a spanning tree T for this topology by disconnecting any two nodes, say (u_0, u_1).

Now, the min distance between u_0, u_1 in the spanning tree is $N - 1$. Their min distance in the original graph is 1.

Thus, in general, a spanning tree is useless for estimating distances in the original graph. In this case, the bound gives us:

$$1 = d_G(u_0, u_1) < d_T(u_0, u_1) = N - 1 \simeq \texttt{diameter}(G)$$

Which is useless. We already know by the worst case that $d_G(u_0, u_1) \leq \texttt{diameter}(G)$.

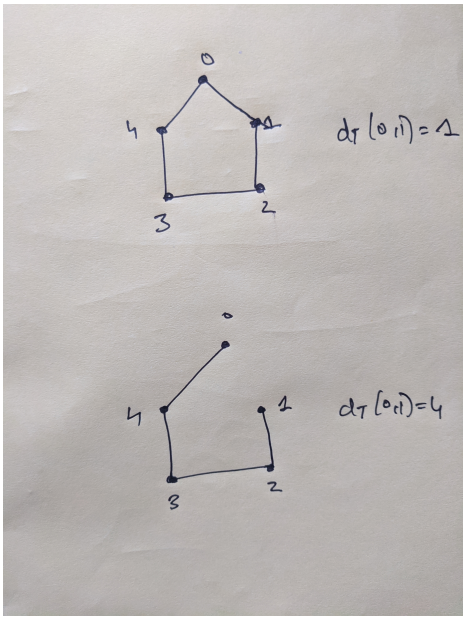

Figure 1: Example of ring with 5 nodes; relationship b/w spanning tree distance and graph distance