graph.ixi

Samuel Rufinatscha

February 2019

1 Abstract

By nature, IOTA transactions reference only two other transcations [1] - which may not suffice in all cases. There are scenarios, for example like in qubic or when determining timestamps for transactions, where arbitrary data pieces need to be linked with arbitrary number of other data pieces [2][3]. As specified by Paul Handy, an alternative, uniform solution for organizing data is required.

graph.ixi is an IXI (IOTA eXtension Interface) module for the Iota Controlled agenT (Ict). It extends the core client with the functionality to link arbitrary data with any number of other data pieces. As a result, it circumvents the usual limit of transactions that can be referenced.

2 Graph

The following graph, represents how data relationships could look like. Every vertex points to one data element and to all the other vertices that are to be referenced. This way, m:n relationships between data elements can be realized. Moreover, this kind of structure could also be used to create cyclic data relationships.

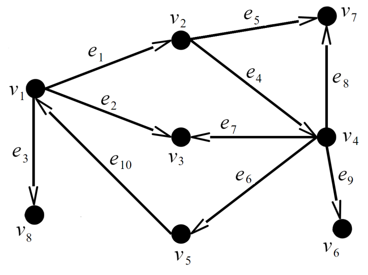

Figure 1: example of m:n data relationships where v=vertex and e=edge.

In order to avoid confusion, the term bundle fragment should be clarified: A bundle fragment, as the name suggests, is a subset of a bundle. This subset is nothing more than an ordered list of transactions through the trunk. The first transaction of this bundle fragment which points in trunk direction to all other transactions, could be called bundle fragment tail. The last transaction of this bundle fragment could be referred as bundle fragment head.

3 Data structure

In general, a vertex is a bundle fragment and therefore a list of transactions ordered through the trunk. However, we have to differentiate between following two vertex structures:

- serialized vertex
- deserialized vertex

3.1 Serialized vertex

A serialized vertex is a vertex ready to be attached to the native Tangle. As already mentioned, a vertex points to one data element and to all the other vertices that are to be referenced. The reference to the data is stored in the data digist field of the tail transaction. The signature or message fragment of the rest of the bundle fragment stores the references (edges) to all the other vertices that are to be referenced. A total of 2187 trytes per signature or message fragment are available; by considering the length of a hash is 81 trytes, a maximum of 27 edges can be stored per field.

Note that a serialized vertex is a subset of a bundle. Therefore, a bundle could contain multiple vertices. This implies that start and end flags must be encoded in order to distinguish the fragments. The third trit of the tag of each transaction indicates whether it's the first transaction of the fragment (fragment tail). The second trit of the tag of each transaction indicates if it's the last transaction of the fragment (fragment head). Figure 2 illustrates this more clearly. In the case a vertex consists only of one transaction, both flags are set.

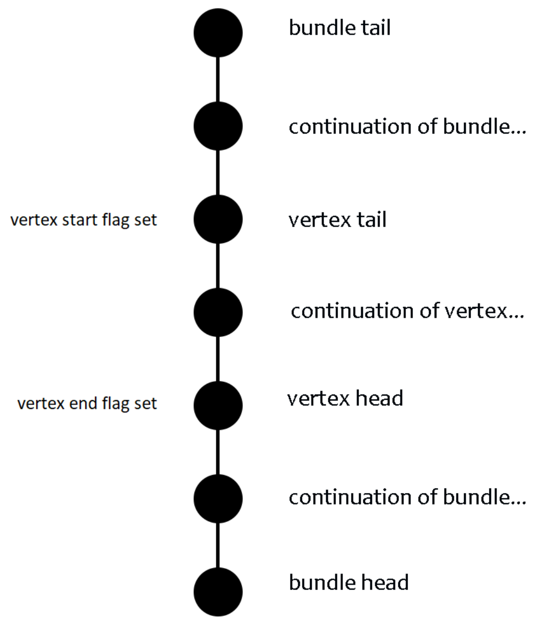

Figure 2: illustration of a serialized vertex inside a bundle

3.2 Deserialized vertex

A deserialized vertex is a vertex that is part of the local graph. To attach a deserialized vertex to the native Tangle, it must get serialized first. The deserialized vertex differs from the serialized vertex regarding the internal organization of data and its edges: the head of the deserialized vertex points in trunk direction to the data element and in branch direction to the first vertex that is to be referenced. For all other edges, a seperate transaction on top of it is appended. This second layer structure serves for code reuse and more easy manipulation of the graph.

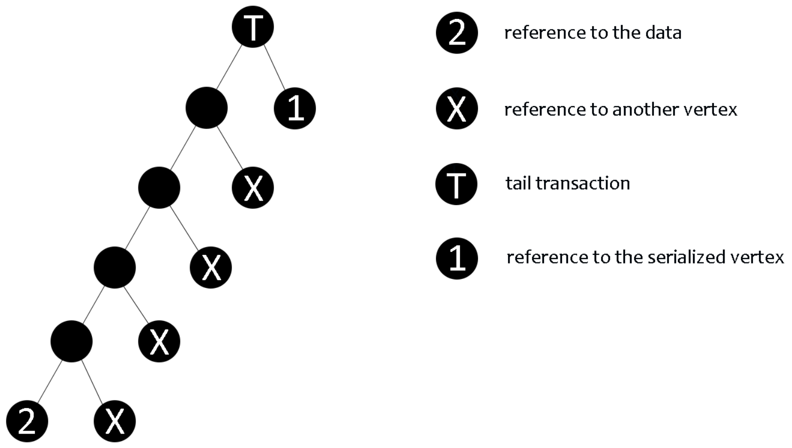
T
1
X
X
X
2
X
2 reference to the data
X reference to another vertex
T tail transaction
1 reference to the serialized vertex

Figure 3: example of a deserialized vertex

3.3 Implementation

For the sake of code quality, the logic of the graph model was strictly separated from the logic of the module [4]. The graph model represents the local graph, and provides various methods for its manipulation. It could be considered as module dependecy. In contrast to that, the DefaultGraphModule class has access to IXI and is therefore able to interact with Ict. The advantage of this architecture is that code can be reused and maintained very easily. Moreover, there exist gossip listeners in the module class that notify when a vertex is received. Following methods can be used to manipulate the graph model:

Creates a vertex with trunk pointing to the data and branch pointing to the outgoing vertex tails that are to be referenced.
createVertex(String data, String[] edges)

Starts a new vertex with trunk pointing to the data and branch pointing to the first vertex that is to be referenced.
startVertex(String data, String edge)

Continues a vertex with branch pointing to the next vertex that is to be referenced.
addEdge(String midVertexHash, String edge)

Continues a vertex with the branches pointing to the vertices that are to be referenced.

addEdges(String midVertexHash, String[] edges)

Finalizes a vertex ready to put into a bundle.
finalizeVertex(String reflectedTail)

Serializes bundle fragments ready to attach to the Tangle.
serialize(List<TransactionBuilder> ... vertices)

Deserializes and adds all vertices of a bundle to the graph.
deserializeAndStore(Bundle bundle)

Deserializes and adds all vertices of a bundle fragment to the graph.
deserializeAndStore(List<Transaction> transactions)

Returns the data bundle fragment tail from the vertex.
getData(String vertex)

Returns all outgoing edges of a vertex.
getEdges(String vertex)

Returns the next outgoing edge of a vertex.
getNextEdge(String vertex, String previousEdge)

Returns all vertices which point to a specific data bundle fragment.
getCompoundVertex(String data)

Returns all vertices which point to given vertex.
getReferencingVertices(String vertex)

Checks if a vertex fragment contains a specific trunk or branch.
isDescendant(String vertex, String descendant)

Returns next vertex which points to a specific data bundle fragment.
getNextCompoundVertex(String data, String previousVertex)

Returns the next vertex which points to given vertex.
getNextReferencingVertex(String vertex, String previousVertex)

Returns all transactions of the graph.
getTransactionsByHash()

4 References

[1] The Tangle, by Serguei Popov
`https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a21`
`iota1_4_3.pdf`

[2] Specification of graph.ixi, by Paul Handy
`https://github.com/iotaledger/omega-docs/blob/master/ixi/graph/Spec.`
`md`

[3] Specification of timestamping.ixi, by Paul Handy
`https://github.com/iotaledger/omega-docs/blob/master/ixi/timestamping/`
`Spec.md`

[4] graph.ixi repository
`https://github.com/iotaledger/graph.ixi`