succinct graphs

challenge

vg's current graph memory model is weak and extremely bloated. It relies on fixed-width 64-bit integer ids and large hash tables mapping these to other entities. This makes it difficult to store in memory, and a general-purpose key-value store (rocksdb) is used to allow low-memory access to the entire graph. Although this design has some advantages, querying the graph requires costly IO operations, and thus use must be managed carefully when developing high-performance applications.

Fully-indexed graphs should be cheap to store and hold in memory, but it doesn't seem there is a standard approach that can be used just for high-performance access to the sequence and identifier space of the graph. Most work has gone into improving performance for querying the text of such a graph GCSA or generating one out of sequencing reads (assemblers such as SGA or fermi2).

The basic requirement is a system that a minimal amount of memory to store the sequence of the graph, its edges, and paths in the graph, but still allows constant-time access to the essential features of the graph. The system should support accessing:

- the node's label (a DNA sequence, for instance, or URL)
- the node's neighbors (inbound and outbound edges)
- the node's region in the graph (ranges of node id space that are within some distance of the node)
- node locations relative to stored paths in the graph
- node and edge path membership

sketch

In theory we could construct a mutable system based on wavelet tries, but research in this area is very new, and I have not found readily-available code for working with these systems. It should be possible to construct mutable wavelet tries using sdsl-lite as a basis, but at present this may be too complex an objective. An immutable system seems like a straightforward thing to do.

First some definitions. We have a graph $G = N, E, P$ with nodes $N = n_1, \ldots, n_{|N|}$, directed edges $E = e_1, \ldots, e_{|E|}$, and paths $P = p_1, \ldots, p_{|P|}$. Nodes match labels l_{n_i} to ranks i in the collection of node labels: $n_i = l_{n_i}, i$. Edges go from one node to another $e_j = n_x, n_y$. Paths match labels l_{p_k} to sets of nodes and edges $p_k = l_{p_k}, \{n_1, e_3, n_4, e_5, \ldots\}$.

We first store the concatenated sequences of all elements, $S = l_{n_1} l_{n_2} l_{n_3} \ldots l_{n_{|N|}}$, in the graph in a compressed integer vector, S_{iv}. A second compressed bitvector,

$S_{bv} : |S_{iv}| = |S_{bv}|$, flags node starts, providing a system of node identifiers. We can apply $rank_1(S_{bv}, x)$ to determine the node rank/id at a given position in S_{iv}, and we can use $select_1(S_{bv}, x)$ to find the positions in S_{iv} corresponding to node with rank/id x, thus allowing basic navigation of the nodes and their labels.

To store edges we keep compressed integer vectors of node ids for the forward F_{iv} and reverse T_{iv} link directions, where $F_{iv} = f_1, \ldots, f_{|N|}$ and $f_i = i, to_{i_1}, \ldots, to_{i_{|to_i|}}$. T_{iv} inverts this relationship, providing $T_{iv} = t_1, \ldots, t_{|N|}$ and $t_i = i, from_{i_1}, \ldots, from_{i_{|from_i|}}$. Recall that i is the rank of the node. Using another bitvector $F_{bv} : |F_{bv}| = |F_{iv}|$ and $T_{bv} : |T_{bv}| = |T_{iv}|$ for we record the first position of each node's entries in F_{iv} and T_{iv}. This first position simply records the rank i in S_{iv}. The rest of the positions in the node's range record the ranks/ids of the nodes on the other end of the edge— on the "to" end in the F_{iv} and the "from" end in T_{iv}. If a node has no edges either coming from or going to it, it will only be represented by reference to its own rank in the corresponding edge integer vector.

We can represent the path space of the graph using a bitvector marking which entities in the edge-from integer vector F_{iv} lie in a path. For each traversed node or edge, we mark a 1 in a new bitvector $P_{ibv} : |P_{i_{bv}}| = |F_{iv}|$. We mark contained entries with 1 and set the un-traversed nodes and edges to 0. Each path thus maps a label to a list of nodes and edges.