

RGFA 1.3 Cheatsheet (1/2)

Create graph ...from GFA file ...from string ...from string array (e.g.)	<code>RGFA.new</code> <code>RGFA.from_file("filename")</code> <code>string.to_rgfa</code> <code>["H\tVN:i:1.0", "S\tA\t*\tLN:i:1200"].to_rgfa</code>
Write GFA to file Write GFA to standard output Create deep copy Output statistics (normal/compact)	<code>gfa.to_file(filename)</code> <code>puts gfa</code> <code>gfa.clone</code> <code>puts gfa.info; puts gfa.info(true)</code>
Turn off validations Validate line references Enable progress logging	<code>gfa.turn_off_validations</code> <code>gfa.validate!</code> <code>gfa.enable_progress_logging</code>
Name of all segments Name of all paths All segments, links, paths, etc Iterate over segments, links, etc	<code>gfa.segment_names</code> <code>gfa.path_names</code> <code>gfa.segments; gfa.links; gfa.paths; ...</code> <code>gfa.segments.each { s ...}</code>
Find segment ...exception if does not exist	<code>gfa.segment(segment_name)</code> <code>gfa.segment!(segment_name)</code>
Find path All paths through segment	<code>gfa.path(path_name) (or: path!)</code> <code>gfa.paths_with(segment_name)</code>
Find link (or, if multiple may exist) All links of segment end (also segment instead of name) Target of all links	<code>gfa.link([:S1,:E],[S2,:B]) (or: link!)</code> <code>gfa.links_between([:S1,:E],[S2,:B])</code> <code>gfa.links_of([:S1,:E])</code> <code>gfa.links_of([segment!(S1),:E])</code> <code>gfa.neighbours([:S1,:E])</code>
Find containment (or, if multiple may exist) All containments for a segment	<code>gfa.containment(container, contained)</code> <code>gfa.containment!(container, contained)</code> <code>gfa.containments_between(c_ner, c_ned)</code> <code>gfa.containing(contained)</code> <code>gfa.contained_in(container)</code>
Add line (examples) Rename segment or path	<code>gfa << "H\tVZ:i:1.0"</code> <code>gfa << "S\tA\t*\tLN:i:1200"</code> <code>gfa.rename(:old, :new)</code>
Segment coverage Segment coverage (more accurate) Segment K-mer coverage Segment length Other end of a link Other end of other end of link	<code>s.coverage</code> <code>s.coverage(unit_length: avreadlen)</code> <code>s.coverage(count_tag: :KC)</code> <code>s.length</code> <code>link.other_end([s1,:E])</code> <code>link.other_end([s1,:E])</code> <code>.revert_end_type</code>
Read req.field/tag value ...raise if tag not available ...tag string	<code>segment.from; segment.LN</code> <code>segment.LN!</code> <code>segment.field_to_s(:LN)</code>
Set/create custom tag (ab, Z type) ...of i or B/i type ...of f or B/f type ...of J type (hash/array)	<code>segment.ab = "value"</code> <code>s.ab = 12; s.ab = [1,2,3]</code> <code>s.ab = 12.0; s.ab = [1.2,2.3,3.0]</code> <code>s.ab = {"a" => 12}; s.ab = ["a","b",1]</code>

RGFA 1.3 Cheatsheet (2/2)

Delete segment (and its links, etc)	<code>gfa.rm("a")</code>
Delete path	<code>gfa.rm("path1")</code>
Delete link/containment	<code>gfa.rm(gfa.link(...))</code>
Delete all headers	<code>gfa.rm(:headers)</code>
Delete sequences (set all to *)	<code>gfa.rm(:sequences)</code>
<i>(rm with a method)</i>	
Delete links of segment end	<code>gfa.rm(:links_of, [:S1, "E"])</code>
Delete link targets	<code>gfa.rm(:neighbours, [:S1, "E"])</code>
Delete paths of segment	<code>gfa.rm(:paths_with, :S1)</code>
Delete segments contained in s	<code>gfa.rm(:contained_in, :s)</code>
Delete s1-E links except to s2-B	<code>gfa.delete_other_links([s1, :E], [s2, :B])</code>
Access headers field	<code>gfa.header.xx</code>
Add new header field	<code>gfa.header.add(:xx, 12)</code>
Sum of read counts	<code>gfa.segments.map(&:RC).inject(:+)</code>
Highest coverage	<code>gfa.segments.map(&:coverage).max</code>
Delete low coverage segments	<code>gfa.rm(gfa.segments.select { s s.coverage < mincov })</code>
Delete isolated segments	<code>gfa.rm(gfa.segments.select { s gfa.connectivity(s) == [0,0] })</code>
Multiply segment	<code>gfa.multiply("A", 4)</code>
Detect linear paths	<code>gfa.linear_paths</code>
Detect and merge linear paths	<code>gfa.merge_linear_paths</code>
Compute connected components	<code>gfa.connected_components</code>
Component of a segment	<code>gfa.segment_connected_component(s)</code>
Split components	<code>gfa.split_connected_components</code>
Number of dead ends	<code>gfa.n_dead_ends</code>
<i>(require "rgfatools")</i>	
Multiply segment, distribute links	<code>gfa.multiply("A", 4)</code>
Compute copy numbers	<code>gfa.compute_copy_numbers</code>
Apply copy numbers	<code>gfa.apply_copy_numbers</code>
Orient invertible segments	<code>gfa.randomly_orient_invertibles</code>
Enforce mandatory links	<code>gfa.enforce_mandatory_links</code>
Remove p-bubbles	<code>gfa.remove_p_bubbles</code>
Remove small components	<code>gfa.remove_small_components(minlen)</code>
<i>(Command line tools)</i>	
Compare two GFA files	<code>gfadiff 1.gfa 2.gfa</code>
...only segments and links	<code>gfadiff -s -l 1.gfa 2.gfa</code>
...output as ruby script	<code>gfadiff -script 1.gfa 2.gfa</code>
Merge linear paths in graph	<code>rgfa-mergelinear 2.gfa > 3.gfa</code>
<i>(Case studies CLI tools)</i>	
Simulate de Bruijn graph	<code>rgfa-simdebruijn 27 gnm.fas > 1.gfa</code>
...and find CRISPRs candidates	<code>rgfa-findcrisprs 1.gfa</code>