RGFA/RGFATools v.1.2 Cheatsheet (1/2)

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

RGFA/RGFATools v.1.2 Cheatsheet (2/2)

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