QAMP Checkpoint 2

Work done after Checkpoint 1:

1. Practice how to write and form programs in Rust

Doing this was easy since I already had a good hand over C/C++. However, studying advanced syntax methods/techniques required at the level of the repository were a bit hard to catch up at first. It took me quite some time to get used to writing a program in Rust.

2. Construct program for implementing Triangular Lattice Graph (graph generator)

The overall code segment for implementing a graph generator is completed. Although there are still some errors and failing tests in the prepared code, I am working on rectifying them. The next step would be to push the code onto the repository and perform further tests to finally merge.

3. Understand basic GitHub commands and automated Python testing

Grasped a few new techniques and libraries that are, for example, used to automate Python testing (tox) and building Rust programs (cargo build). These commands are used quite frequently and overall helped me understand how open-source development is done.

4. Working on a few more graph generators

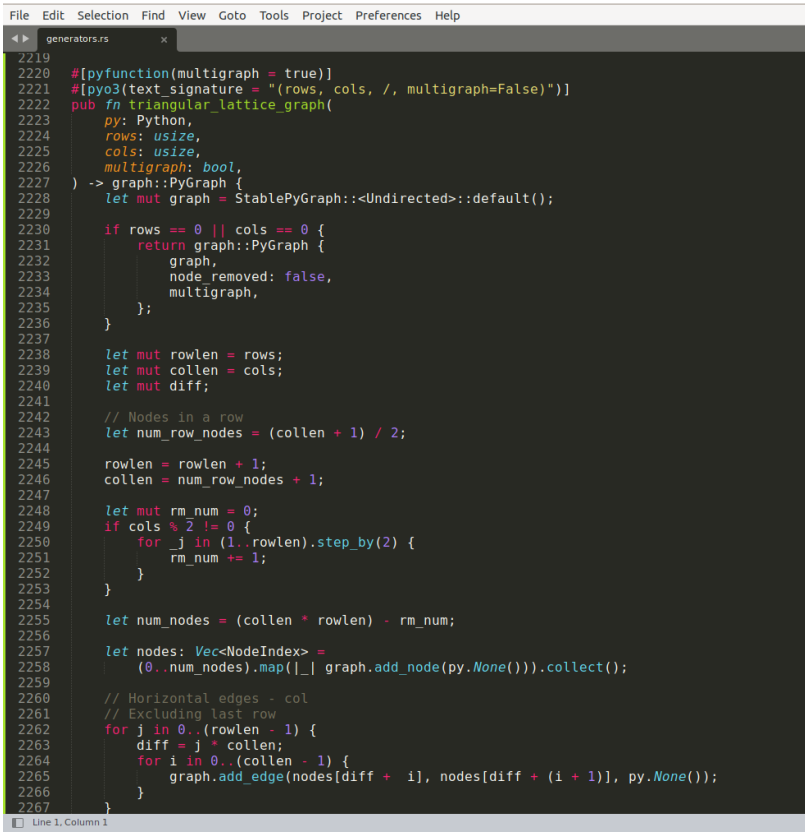
I am simultaneously trying to resolve issues with the triangular lattice program and also code other generators like a few random graphs.

Any difficulties faced?

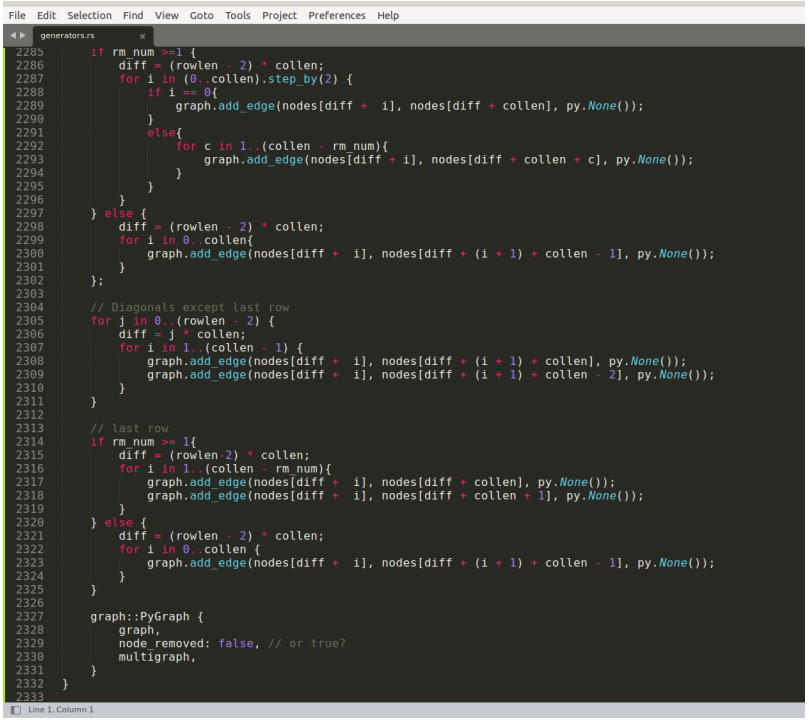
Not much but it is sometimes hard to figure out why tests are failing and the rectification process consumes a lot of time.

Required Visual Representation:

1. Code building



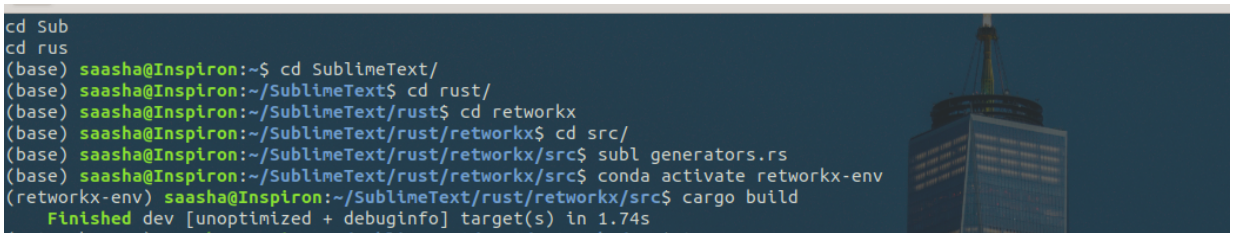
```
File  Edit  Selection  Find  View  Goto  Tools  Project  Preferences  Help
generators.rs
2219
2220  #[pyfunction(multigraph = true)]
2221  #[pyo3(text_signature = "(rows, cols, /, multigraph=False)")]
2222  pub fn triangular_lattice_graph(
2223      py: Python,
2224      rows: usize,
2225      cols: usize,
2226      multigraph: bool,
2227  ) -> graph::PyGraph {
2228      let mut graph = StablePyGraph::<Undirected>::default();
2229
2230      if rows == 0 || cols == 0 {
2231          return graph::PyGraph {
2232              graph,
2233              node_removed: false,
2234              multigraph,
2235          };
2236      }
2237
2238      let mut rowlen = rows;
2239      let mut collen = cols;
2240      let mut diff;
2241
2242      // Nodes in a row
2243      let num_row_nodes = (collen + 1) / 2;
2244
2245      rowlen = rowlen + 1;
2246      collen = num_row_nodes + 1;
2247
2248      let mut rm_num = 0;
2249      if cols % 2 != 0 {
2250          for _j in (1..rowlen).step_by(2) {
2251              rm_num += 1;
2252          }
2253      }
2254
2255      let num_nodes = (collen * rowlen) - rm_num;
2256
2257      let nodes: Vec<NodeIndex> =
2258          (0..num_nodes).map(|_| graph.add_node(py.None())).collect();
2259
2260      // Horizontal edges - col
2261      // Excluding last row
2262      for j in 0..(rowlen - 1) {
2263          diff = j * collen;
2264          for i in 0..(collen - 1) {
2265              graph.add_edge(nodes[diff +  i], nodes[diff + (i + 1)], py.None());
2266          }
2267      }
Line 1, Column 1
```



```
File  Edit  Selection  Find  View  Goto  Tools  Project  Preferences  Help
generators.rs
2285      if rm_num >=1 {
2286          diff = (rowlen - 2) * collen;
2287          for i in (0..collen).step_by(2) {
2288              if i == 0{
2289                  graph.add_edge(nodes[diff +  i], nodes[diff + collen], py.None());
2290              }
2291              else{
2292                  for c in 1..(collen - rm_num){
2293                      graph.add_edge(nodes[diff + i], nodes[diff + collen + c], py.None());
2294                  }
2295              }
2296          }
2297      } else {
2298          diff = (rowlen - 2) * collen;
2299          for i in 0..collen{
2300              graph.add_edge(nodes[diff +  i], nodes[diff + (i + 1) + collen - 1], py.None());
2301          }
2302      };
2303
2304      // Diagonals except last row
2305      for j in 0..(rowlen - 2) {
2306          diff = j * collen;
2307          for i in 1..(collen - 1) {
2308              graph.add_edge(nodes[diff +  i], nodes[diff + (i + 1) + collen], py.None());
2309              graph.add_edge(nodes[diff +  i], nodes[diff + (i + 1) + collen - 2], py.None());
2310          }
2311      }
2312
2313      // last row
2314      if rm_num >= 1{
2315          diff = (rowlen-2) * collen;
2316          for i in 1..(collen - rm_num){
2317              graph.add_edge(nodes[diff +  i], nodes[diff + collen], py.None());
2318              graph.add_edge(nodes[diff +  i], nodes[diff + collen + 1], py.None());
2319          }
2320      } else {
2321          diff = (rowlen - 2) * collen;
2322          for i in 0..collen {
2323              graph.add_edge(nodes[diff +  i], nodes[diff + (i + 1) + collen - 1], py.None());
2324          }
2325      }
2326
2327      graph::PyGraph {
2328          graph,
2329          node_removed: false, // or true?
2330          multigraph,
2331      }
2332  }
2333
Line 1, Column 1
```

2. Creating Tests for the above code

3. Building the Rust code - using cargo build



4. Running the tests - using tox -epy