Good first issues in retworkx

Soham Pal
Mentor: Matthew Treinish
git: https://github.com/Qiskit/retworkx/

Project summary:

- retworkx is a high performance general purpose graph library written in Rust for Python 3.
- It was originally developed to replace the usage of NetworkX, the gold standard Python graph library, in Qiskit.
- While featureful, NetworkX is slow, because it is implemented using Python. Hence the need for something like retworkx.
- This project is aimed at fixing some of the “good first issues” in retworkx.

Qiskit

Why and What I did

I wanted to join this project because

1. I wanted to learn more about graph theory, and
2. I wanted to get some real-world experience with Rust.

I mostly worked on two issues that are about bridging the gap, in terms of functionalities offered, between NetworkX, and retworkx:

1. Graph generators (Issue #150)
2. Graph operations (Issue #440)

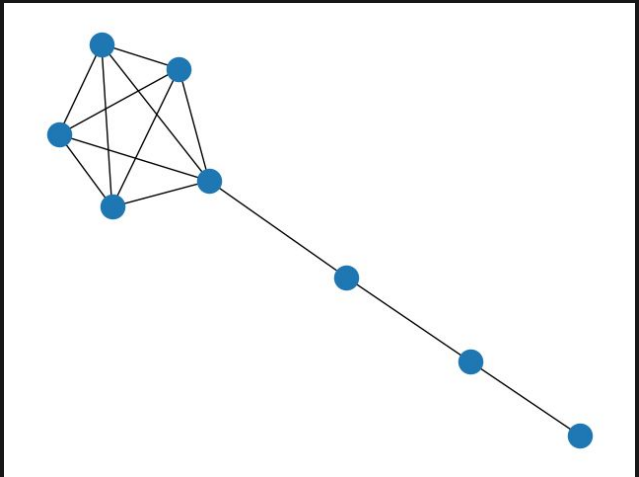
I have submitted two PRs (which have been merged):

1. PR#454 added a lollipop graph generator.
2. PR#471 added a barbell graph generator

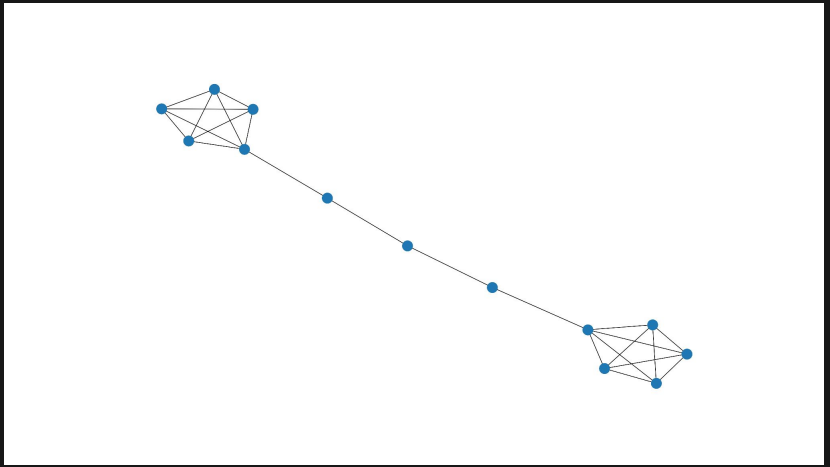
Currently working on adding “Symmetric Difference” of graphs which pertain to Issue#440.

Submitted PRs

A (5, 3) lollipop graph.



A (5, 3) barbell graph.



Qiskit

Work in Progress (Symmetric Diff)

Symmetric difference of two graphs G, H, which have the same nodes is a graph F which has the same nodes as G and H, but has edges that exist either in G or in H, but not in both.

Rust is still complaining about types and illegal moves when it comes to compiling this code. 😩

Hope to fix this soon and submit it as a PR.

```rust
fn symmetric_difference<Ty: EdgeType>(
    py: Python,
    first: &StablePyGraph<Ty>,
    second: &StablePyGraph<Ty>,
) -> PyResult<StablePyGraph<Ty>> {
    let mut first_nodes_set = first.node_references().cloned();
    let mut second_nodes_set: HashSet<_> = second.node_references().cloned();
    let nodes_symm_diff: HashSet<_> = first_nodes_set
        .symmetric_difference(&second_nodes_set)
        .collect();
    if !nodes_symm_diff.is_empty() {
        // return Err(PyIndexError::new_err(
        //     "The two graphs do not have the same nodes.",
        // ));
    }

    let mut final_graph = StablePyGraph::<Ty>::with_capacity(
        first.node_count(),
        first.edge_count() + second.edge_count(),
    );
    let mut node_map: HashMap<NodeIndex, NodeIndex> =
        HashMap::with_capacity(first.node_count());
    for node_index in first.node_indices() {
        let node = first[node_index].clone_ref(py);
        let new_index = final_graph.add_node(node);
        node_map.insert(node_index, new_index);
    }

    let mut first_edges_set: HashSet<_> =
        first.edge_references().cloned().collect();
    let mut second_edges_set: HashSet<_> =
        second.edge_references().cloned().collect();

    for edge in first_edges_set.symmetric_difference(&second_edges_set) {
        let &source = node_map.get(&edge.source()).unwrap();
        let &target = node_map.get(&edge.target()).unwrap();
        let weight = edge.weight();
        final_graph.add_edge(source, target, weight.clone_ref(py));
    }

    Ok(final_graph)
}
```

Qiskit