Simple Graph: Connectivity and Induced Subgraph

Some of the previous proofs are now simplified and included in the lean file. All works from line 90 are new (more-or-less)[1].

Definitions, Corollaries and Theorems

Subgraph Obtained from a Walk

A subgraph G of G obtained from walk $p : G.\text{walk}\ \_\ \_$ is a subgraph with vertex set $\text{verts} = \{v \in p.\text{support}\}$ and edge set $\text{adj} = \{e \in p.\text{edges}\}$. Since p is a walk in G, obviously, we have $\text{verts} \subseteq G.\text{verts}$[2] and $\text{adj} \subseteq G.\text{adj}$, thus G' is indeed a subgraph of G.

Hence, we have a Lean definition as:

```
def subgraph.from_walk {u v : V} (p : G.walk u v) : subgraph G :=
  {
    verts := {w : V | w ∈ p.support},
    adj := λ w x, ⟦(w, x)⟧ ∈ p.edges,
    adj_sub := _
    edge_vert := _
    symm := _
  }
```

Helper Functions

1. Coercion of Vertices
2. Coercion of `adj` adjacent relation
3. Coercion of the whole walk p that the subgraph G' Is obtained from (not done yet)
    - Unable to prove that the recursion is well-founded. A decreasing, well-formed variable is found (`p.length`), yet unable to get `using_well_founded` work.

Euler Circuit

Definition. Euler Circuit (`is_euler_circuit`).

Sometimes written as "Euler Cycle". For consistency with mathlib, "circuit" is preferred. A circuit p in a graph G is a **Euler circuit** of graph G if this circuit contains all edges in this graph. Formally,

$$\forall e, e \in G.\text{edge\_set} \implies e \in p.\text{edges} \tag{1}$$

(`is_eulerian`). It is also said that a graph G is **Eulerian** if it contains a Euler circuit. i.e., there is a p walk in G s.t. p is a Euler circuit, and all vertices in G are visited by such circuit. Thus,

```
def is_euler_circuit {v : V} (p : G.walk v v) : Prop :=
  p.is_circuit ∧ (∀ ⦃e : sym2 V⦄, e ∈ G.edge_set → (e ∈ p.edges)) ∧ ∀ ⦃u : V⦄, u ∈ p.support
```

All Supports on a Walk are Reachable to Each Other

For any two vertices v w In graph G, v is reachable to w if there is a walk passing through both of them. Formally, we say

```
theorem reachable_if_support {u v w x : V} (p : G.walk u x) (h1 : v ∈ p.support) (h2 : w ∈ p.support) [decidable_eq V]:
  G.reachable v w
```

Example on Euler Circuit

A minimum example which mimic (poorly) the *Seven Bridges of Königsberg* is included and fully proven. Due to the restriction of `simple_graph`, the resultant graph is a cycle with 4 vertices, and thus is actually a Eulerian graph.

Construction of Example

Type

An inductive type `sector_` is defined as the `Type u` required to define the `simple_graph`. Both `decidable_eq` and `fintype` instances are derived as necessary for further proofs.

Though `fin_enum` instance is easy to define for it (and actually being defined if you uncomment it), it is not used in proof and thus being eliminated to accelerate evaluation.

In the proof, helper variables and Props are used to simplify the wording of statements, by renaming `sector_.v1` into `v1` and show their equity, etc.

Relation (adj)

Edges (the only 4 non-duplicate bridges) are defined as a relation. It need not to be symmetric by itself as the factory function to construct the graph `simple_graph.from_rel` will take care of this. It is being defined with minimal statements required to simplify the proof.

Listing true conditions does help then negating false conditions. Further, it is easier to work with `or` rather than pattern matching branches in the proof.

Circuit

The circuit is constructed on-site in the proof, by defining the edges with `graph_2.adj` first then chain them up.

Proof

Vertices Set is Complete

trivial

Edge set is Complete

This is done by enumerating the edges and proof the exhaustance.

Reflection

I am very sorry of trying to prove a false statement in coursework 2 (and continued here which wasted me a day).

Working with recursions is hard in Lean, and more design should be put in before actual working.

---

1. diff shows 357 lines of additions and about 175 deletions. ↩

2. Note that when G is a `simple_graph`, $G.\mathrm{verts}$ is not defined in mathlib, but should be referred as `V : Type u` given `G : simple_graph V`. ↩