Graph and Search Library Tutorial

Andrew Grant, Anton Igorevich, Somya Vasudevan

1 How to Use the Library

- The most important thing is for the user to define his/her vertex and edge data types. The only requirements are that the vertex/edge are comparable (aka must implement operator==) and hashable (aka implement struct hash ...).
- Then the user should select one of `graph_dg`, `graph_dag`, `graph_dt`, `matrix_graph` and provide the struct with two template parameters that specify the vertex and edge types (as mentioned above the lib provides vertex and edge for this but the user can use his/her own data types) e.g. `dag_graph¡my_vertex_1`, `my_edge_1¿ my_graph`; e.g. `dt_graph¡`vertex, edge`¿ my_graph`; e.g. `dg_graph¡my_vertex_2`, `my_edge_2¿ my_graph`;
- At this point any/all of the functions can be used. Note that all functions require pointers as inputs (more specifically `shared_ptrs`); this is to avoid the cost of copying large graphs/vertices/edges see examples/ directory for some examples
- Note that the same function name is used for all graph types, vertex types and edge types. This is another benefit of concepts; that is, concepts are used to make sure the right function is called using overloading

2 Examples

2.1 Creating a graph

Create a matrix graph, where Vertex type is "city", and Edge type is "road". "city" and "road" are both user defined classes.

```
shared_ptr<matrix_graph<city , road>> my_graph = make_shared<matrix_graph<city , road>>(10);
```

2.2 Creating a Vertex using helper function

Create a Vertex using helper function. A unique id is automatically assigned to v1

```
auto v1 = create_vertex(my_graph);
```

2.3 Setting a value to a Vertex

Note, the user defined Vertex does not need to know anything about Value

```
set_value(my_graph, v0, Value {"A", 1990});
```

2.4 Creating Edge

```
auto e1 = create_edge(my_graph, v0, v2);
```

2.5 Removing a Vertex

```
remove(my_graph, v11);
```

2.6 Finding path between Vertices

First we get a struct that contains a bunch of data about the path. The second line we get a vector of the vertices along the path.

```
path_v0_v1 = find_path_ucs(my_graph, v0, v1);
auto vector_of_vertices = path_v0_v1->path_v;
```