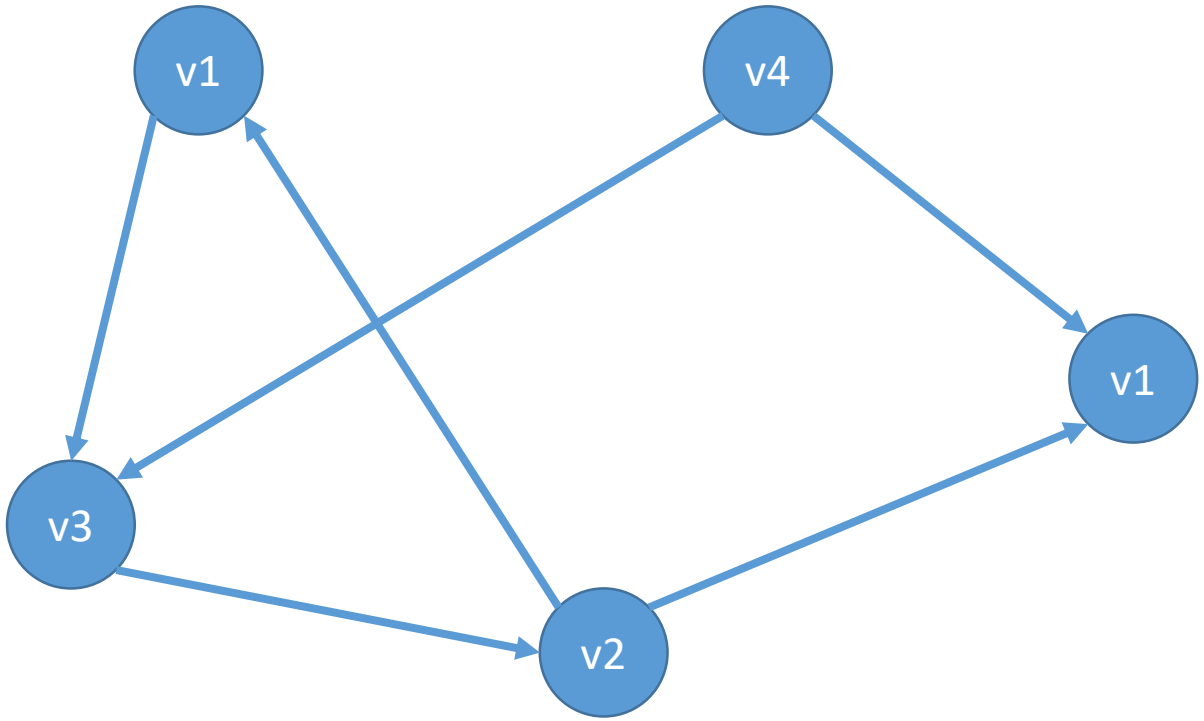
Graph Database

NPRG051 - Advanced Programming in C++

Assignment #2

faltin@ksi.mff.cuni.cz

Graph



- https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)

Requirements

- Columnar
- Static schema (passed as a template parameter)
- Dynamic workload

Row storage

```
struct point_3d {
   int x;
   int y;
   int z;
};
std::vector<point_3d> points;
```

std::vector<int [3]>
{1, 2, 3}
{4, 5, 6}
{7, 8, 9}
{10, 11, 12}
{13, 14, 15}

Memory

Columnar storage

```
struct points_3d {
  std::vector<int> xs;
  std::vector<int> ys;
  std::vector<int> zs;
};
points_3d points;
```

std::vectors xs[3]		
xs	ys	zs
[1, 4, 7, 10, 13]	[2, 5, 8, 11, 14]	[3, 6, 9, 12, 15]

Memory

1	4	7	10	13				
2	5	8	11	14				
3	6	9	12	15				

Columnar storage

- A struct of arrays vs. An array of structs
  - https://en.wikipedia.org/wiki/AoS_and_SoA
- Columns optimized for reading of single properties
- Easier for SIMD instructions

API: graph schema

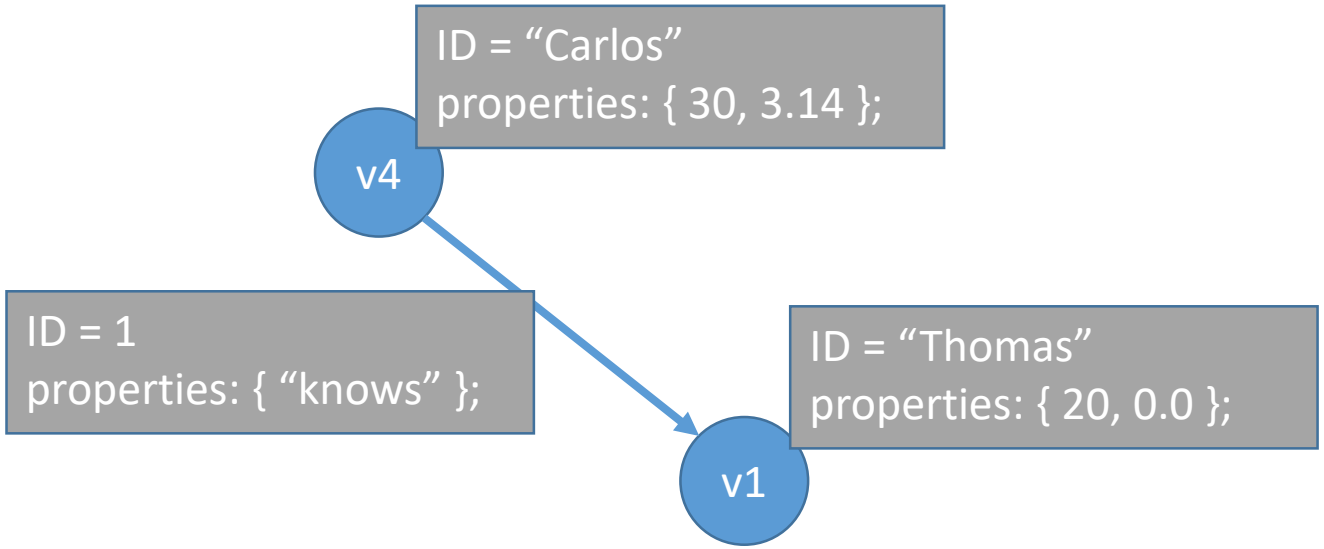
```
struct graph_schema {
  using vertex_user_id_t = // vertex id type
  using vertex_property_t = // vertex property type

  using edge_user_id_t = // edge id type
  using edge_property_t = // edge property type
};
```

API: graph schema example

```
struct graph_schema {
  using vertex_user_id_t = std::string;
  using vertex_property_t = std::tuple<int, double>

  using edge_user_id_t = int;
  using edge_property_t = std::tuple<std::string>;
};
```



API: the class `graph_db`

- Header: `graph_db.hpp`
  - Documented

```
template<class GraphSchema>
class graph_db {
  // types
  // functions for vertexes
  // functions for edges
};
```

API: functions for vertexes

```
// Add a new vertex into the DB with default properties
vertex_t add_vertex(GraphSchema::vertex_user_id_t &&);

// Add a new vertex into the DB with given properties
vertex_t add_vertex(GraphSchema::vertex_user_id_t &&,
                    Props &&...);

// Return [begin(),end()] iterators to all vertexes in DB
std::pair<vertex_it_t, vertex_it_t> get_vertexes();
```

API: functions for edges

```
// Add a new edge between 2 vertexes into the DB with default
//  property values
edge_t add_edge(GraphSchema::edge_user_id_t &&,
                const vertex_t &, const vertex_t &);

// Add a new edge between 2 vertexes into the DB with given
//  property values
edge_t add_edge(GraphSchema::edge_user_id_t &&,
                vertex_t, vertex_t, Props &&...);

// Return [begin(),end()] iterators to all edges in DB
std::pair<edge_it_t, edge_it_t> get_edges();
```

API: types

```
using vertex_t = // The vertex type
using edge_t = // The edge type

using vertex_it_t = // The vertex iterator type
using edge_it_t = // The edge iterator type

using neighbor_it_t = // The neighbor iterator type
```

API: the vertex class

```
// Returns id of the vertex
GraphSchema::vertex_user_id_t id();

// Returns all properties of vertex
GraphSchema::vertex_property_t get_properties();

// Returns a single property value on I-th place
auto get_property<I>();

// Set all values of properties
void set_properties(PropsType &&...);

// Set a single property value on I-th place
void set_property<I>(PropType);

// A iterator type that traverses outgoing edges
using neighbor_it_t =

// Return [begin(),end()] iterators to the neighbors
std::pair<neighbor_it_t, neighbor_it_t> edges();
```

API: the edge class

```
// Returns id of the edge
GraphSchema::edge_user_id_t id();

// Returns all properties of edge
GraphSchema::vertex_property_t get_properties();

// Returns a single property value on I-th place
auto get_property<I>();

// Set all values of properties
void set_properties(PropsType &&...);

// Set a single property value on I-th place
void set_property<I>(PropType);

// Returns the source vertex
graph_db::vertex_t src();

// Returns the destination vertex
graph_db::vertex_t dst();
```

Evaluation

- Upload `graph_db.hpp` with the correct API into Recodex
  - You can include also your own files
- Testing suite is available with the example test
  - If it compiles & runs on your machine, it should compile & run in Recodex too
- Resulting points based on the manual evaluation
- 10 points in total
  - Functionality (major)
  - Code culture (minor)
    - Readability, no warnings, no memory-leaks, const-correctness, no necessary copies (rvalues, references, …), ...

Hints

- Use proxy pattern for vertex/edge classes
  - https://en.wikipedia.org/wiki/Proxy_pattern
- Output iterators
  - https://en.cppreference.com/w/cpp/named_req/OutputIterator
- `std::vector<bool>` is specialized