Graph algorithms library

PV264 project

Vojtěch Kaňa, Matej Hulín

The main goal of the proposed project is, as its name suggets, to create C++ library, which should provide various graph traversal and shortest-paths algorithms. For implementation, we have chosen following algorithm categories:

Graph traversal

```
• Depth-first search (DFS) with function prototype:
 template<typename Graph,
           typename = std::enable_if_t<Graph::traversableTag>>
 void dfs(Graph& graph);
• Breadth-first search (BFS) with function prototype:
 template<typename Graph,
           typename = std::enable_if_t<Graph::traversableTag>>
 void bfs(Graph& graph,
           const typename graph_traits<Graph>::node_handle &source);
• Iterative deepening depth-first search (IDDFS) with function prototype:
  template<typename Graph,
           std::enable_if_t<Graph::traversableTag>>
 bool iddfs(Graph& graph,
             const typename graph_traits<Graph>::node_handle &root_nh,
             const typename graph_traits<Graph>::node_handle &goal_nh,
             std::size_t max_depth);
• A* with function prototype:
 template<typename Graph,
           typename Heuristic,
           typename PriorityQueue = BinaryHeap<typename Graph::node_handle,</pre>
                                                  CustomComparator<Graph>>>
 void AStar(Graph& graph,
              const typename graph_traits<Graph>::node_handle &source,
              const typename graph_traits<Graph>::node_handle &target,
              const Heuristic& heuristic);
 A* algorithm traverses graph from source using heuristic until it finds node target.
```

Single source shortest-paths

```
• Dijkstra algorithm (using Heap and Fibonacci heap) with function prototype:
     template<typename Graph,
              typename PriorityQueue = BinaryHeap<Graph>,
              typename = std::enable_if_t<Graph::weightedTag &&</pre>
                                           Graph::pathTag>>
    bool dijkstra(Graph& graph,
               const typename graph_traits<Graph>::node_handle &source);
   • Bellman-Ford algorithm with function prototypes:
    template<typename Graph,
              typename = std::enable_if_t<Graph::directedTag &&</pre>
                                           Graph::weightedTag &&
                                           Graph::pathTag>>
    bool bellmanFord(Graph& graph,
                      const typename graph_traits<Graph>::node_handle &source);
    template<typename Graph,
              typename = std::enable_if_t<!Graph::directedTag &&</pre>
                                           Graph::weightedTag &&
                                           Graph::pathTag>>
    bool bellmanFord(Graph& graph,
                      const typename graph_traits<Graph>::node_handle &source);
   • DAG shortest-paths (Topological sort and DFS) with function prototype:
    template<typename Graph,
              typename = std::enable_if_t<Graph::directedTag &&
                                           Graph::weightedTag &&
                                           Graph::pathTag>>
    bool dag(Graph& graph,
              const typename graph_traits<Graph>::node_handle &source);
All-pairs shortest-paths
   • Floyd-Warshall algorithm with function prototype:
     template<typename Graph,
              typename = std::enable_if_t<Graph::directedTag &&
                                           Graph::weightedTag &&
                                           Graph::pathTag>>
    Matrix<typename graph_traits<Graph>::distance_type floydWarshall(const Graph& graph);
   • Johnson's algorithm with function prototype
     template<typename Graph,
              typename = std::enable_if_t<Graph::directedTag &&
                                           Graph::weightedTag &&
                                           Graph::pathTag>>
    Matrix<typename graph_traits<Graph>::distance_type johnson(Graph& graph);
```