GUNDAM::SmallGraph说明文档

概述

本文档是针对GUNDAM::SmallGraph的说明文档，主要内容包括参数配置，接口说明，使用方法以及使用建议

参数配置

GUNDAM::SmallGraph参数配置如下所示：

```
template <class VertexIDType, class VertexLabelType, class EdgeIDType,
          class EdgeLabelType>
class SmallGraph;
```

VertexIDType:点的ID的数据类型
VertexLabelType:点的label的数据类型
Edge部分同理，不再赘述

结构说明

整个Graph结构可以分为两个部分，分别是点(Vertex)和边（Edge）

Vertex

在GUNDAM::SmallGraph中，无法访问到一个具体的Vertex实例，只能通过指针(VertexPtr,VertexConstPtr)来访问点以及相关内容

Edge

在GUNDAM::SmallGraph中，无法访问到一个具体的Edge实例，只能通过指针(EdgePtr,EdgeConstPtr)来访问点以及相关内容

接口说明

本部分的接口说明主要针对以下三个部分进行说明：Graph能使用的接口，VertexPtr/VertexConstPtr可以使用的接口以及EdgePtr/EdgeConstPtr使用的接口。

Graph接口

std::pair<VertexPtr, bool> AddVertex(const typename VertexType::IDType &id,const typename VertexType::LabelType &label)

往Graph中添加一个ID为id，Label为Label的点。若当前Graph中已有ID为id的vertex，则返回的pair为：<指向已有节点的指针, false>；否则返回<指向新加入节点的指针, true>。

std::pair<EdgePtr, bool> AddEdge(const typename VertexType::IDType &src,const typename VertexType::IDType &dst,const typename EdgeType::LabelType &label,const typename EdgeType::IDType &id)

往Graph中加入一条从src到dst,Label为label,ID为id的有向边。若当前Graph中已有相同ID的edge，则返回的pair为：<指向已有边的指针, false>；否则返回<指向新加入边的指针, true>；若Graph中不存在ID为src的vertex、或Graph中不存在ID为dst的vertex，则返回<nullptr, false>。

size_t CountEdge()

得到这个图的边数

size_t CountVertex()

得到这个图的点数

EdgeIterator EdgeBegin()

得到这个图的第一条（不是指输入的第一条，而是内部存储的第一条）边的迭代器

EdgeConstIterator EdgeCBegin()

得到这个图的第一条（不是指输入的第一条，而是内部存储的第一条）边的常量迭代器

VertexIterator VertexBegin()

得到这个图的第一个（不是指输入的第一个，而是内部存储的第一个）点的迭代器

VertexConstIterator VertexCBegin()

得到这个图的第一个（不是指输入的第一个，而是内部存储的第一个）点的常量迭代器

VertexPtr FindVertex(const typename VertexType::IDType &id)

返回点ID为id的VertexPtr，如果不存在则返回nullptr

VertexConstPtr FindConstVertex(const typename VertexType::IDType &id)

返回点ID为id的VertexConstPtr，如果不存在则返回nullptr

EdgePtr FindEdge(const typename EdgeType::IDType &id)

返回边ID为id的EdgePtr，如果不存在则返回nullptr

EdgeConstPtr FindConstEdge(const typename EdgeType::IDType &id)

返回边ID为id的EdgeConstPtr，如果不存在则返回nullptr

bool EraseVertex(const typename VertexType::IDType &id)

删除点ID为id的点以及所有与他相连的边，如果原图中没有这个点，则返回false，否则返回true

bool EraseEdge(const typename EdgeType::IDType &id)

删除边ID为id的边，如果原图中没有这个边，则返回false，否则返回true

void Clear()

清空这个图

Graph接口使用说明

```
//图相关的using定义
using Pattern = GUNDAM::SmallGraph<uint32_t, uint32_t, uint32_t, uint32_t>;
using VertexType = typename Pattern::VertexType;
using VertexIDType = typename VertexType::IDType;
using VertexLabelType = typename VertexType::LabelType;
using EdgeType = typename Pattern::EdgeType;
using EdgeIDType = typename EdgeType::IDType;
using EdgeLabelType = typename EdgeType::LabelType;
using VertexPtr = typename Pattern::VertexPtr;
using VertexConstPtr = typename Pattern::VertexConstPtr;
using EdgePtr = typename Pattern::EdgePtr;
using EdgeConstPtr = typename Pattern::EdgeConstPtr;
using VertexSizeType = size_t;
Pattern g;  //定义一个图
for (VertexIDType vertex_id = 1;vertex_id<=10;vertex_id++){
  g.AddVertex(vertex_id,1);  //加点
}
//加边
g.AddEdge(1,2,3,1);
g.AddEdge(1,3,4,2);
for (auto vertex_it = g.VertexCBegin();!vertex_it.IsDone();vertex_it++){
```

```
    //用迭代器访问全部的点
    VertexConstPtr vertex_ptr = vertex_it;  //可直接将迭代器转化为指针
}
VertexPtr vertex_ptr = g.FindVertex(1);
for (auto edge_it = g.EdgeCBegin();!edge_it.IsDone();edge_it++){
    //用迭代器访问全部的边
    EdgeConstPtr edge_ptr = edge_it; //可直接将迭代器转化为指针
}
EdgePtr edge_ptr = g.FindEdge(1);
g.Clear();  //清空这个图
```

Vertex接口

const IDType &id()

返回这个点的id

const VertexLabelType &label()

返回这个点的label

size_t CountOutEdge()

返回这个点的出边数

size_t CountOutEdge(const EdgeLabelType &edge_label)

返回这个点出边Label为edge_label的边数

size_t CountInEdge()

返回这个点的入边数

size_t CountInEdge(const EdgeLabelType &edge_label)

返回这个点入边Label为edge_label的边数

size_t CountInVertex()

返回这个点入边的起点数（去重）

size_t CountInVertex(const EdgeLabelType &edge_label)

返回这个点入边Label为edge_label的起点数（去重）

size_t CountOutVertex()

返回这个点出边的终点数（去重）

size_t CountOutVertex(const EdgeLabelType &edge_label)

返回这个点出边Label为edge_label的终点数（去重）

EdgeIterator OutEdgeBegin()

返回第一条出边的迭代器

EdgeConstIterator OutEdgeCBegin()

返回第一条出边的常量迭代器

EdgeIterator InEdgeBegin()

返回第一条入边的迭代器

EdgeConstIterator InEdgeCBegin()

返回第一条入边的常量迭代器

Vertex接口使用说明

```
//图相关的using定义
using Pattern = GUNDAM::SmallGraph<uint32_t, uint32_t, uint32_t, uint32_t>;
using VertexType = typename Pattern::VertexType;
using VertexIDType = typename VertexType::IDType;
using VertexLabelType = typename VertexType::LabelType;
using EdgeType = typename Pattern::EdgeType;
using EdgeIDType = typename EdgeType::IDType;
using EdgeLabelType = typename EdgeType::LabelType;
using VertexPtr = typename Pattern::VertexPtr;
using VertexConstPtr = typename Pattern::VertexConstPtr;
using EdgePtr = typename Pattern::EdgePtr;
using EdgeConstPtr = typename Pattern::EdgeConstPtr;
using VertexSizeType = size_t;
Pattern g;  //定义一个图
for (auto vertex_it = g.VertexCBegin();!vertex_it.IsDone();vertex_it++){
   //用迭代器访问全部的点
   VertexConstPtr vertex_ptr = vertex_it;  //可直接将迭代器转化为指针
   //输出ID跟Label，上下完全等价
   std::cout<<vertex_it->id()<<" "<<vertex_it->label()<<std::endl;
```

```
    std::cout<<vertex_ptr->id()<<" "<<vertex_ptr->label()<<std::endl;
    //访问这个点所有的出边，上下完全等价
    for (auto edge_it = vertex_it->OutEdgeCBegin();!edge_it.IsDone();edge_it++){
    }
    for (auto edge_it = vertex_ptr->OutEdgeCBegin();!edge_it.IsDone();edge_it++){
    }
}
```

Edge接口

属性部分与Vertex一模一样 不再说明

const EdgeIDType &id()

得到这条边的id

const EdgeLabelType &label()

得到这条边的label

const VertexIDType &src_id()

得到这条边的起点的id

const VertexIDType &dst_id()

得到这条边的终点的id

VertexData *src_ptr()

得到这条边起点的指针

VertexData *dst_ptr()

得到这条边终点的指针

const VertexData *const_src_ptr()

得到这条边起点的常量指针

const VertexData *const_dst_ptr()

得到这条边终点的常量指针

Edge接口使用说明

```
  //图相关的using定义
  using Pattern = GUNDAM::SmallGraph<uint32_t, uint32_t, uint32_t, uint32_t>;
  using VertexType = typename Pattern::VertexType;
  using VertexIDType = typename VertexType::IDType;
  using VertexLabelType = typename VertexType::LabelType;
  using EdgeType = typename Pattern::EdgeType;
  using EdgeIDType = typename EdgeType::IDType;
  using EdgeLabelType = typename EdgeType::LabelType;
  using VertexPtr = typename Pattern::VertexPtr;
  using VertexConstPtr = typename Pattern::VertexConstPtr;
  using EdgePtr = typename Pattern::EdgePtr;
  using EdgeConstPtr = typename Pattern::EdgeConstPtr;
  using VertexSizeType = size_t;
  Pattern g;  //定义一个图
  for (auto vertex_it = g.VertexCBegin();!vertex_it.IsDone();vertex_it++){
     //用迭代器访问全部的点
     VertexConstPtr vertex_ptr = vertex_it;  //可直接将迭代器转化为指针
     for (auto edge_it = vertex_it->OutEdgeCBegin();!edge_it.IsDone();edge_it++){
     EdgeConstPtr edge_ptr = edge_it;
     //得到这条边的相关信息，上下完全等价
       std::cout<<edge_it->id()<<" "<<edge_it->label()<<" "<<edge_it->src_id()<<" "<<edg
e_it->dst_id()<<std::endl;
        std::cout<<edge_ptr->id()<<" "<<edge_ptr->label()<<" "<<edge_ptr->src_id()<<" "<
<edge_ptr->dst_id()<<std::endl;
     }
  }
```

使用建议

Pattern建议使用SmallGraph，请不要在DataGraph使用SmallGraph