HamaWhite 原创，转载请注明出处！欢迎大家加入**Giraph 技术交流群**： 228591158

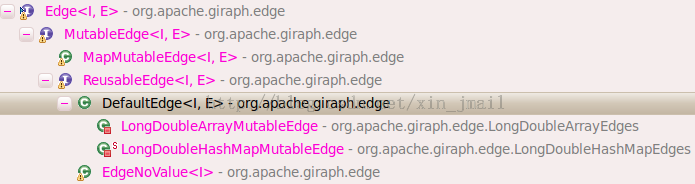
欢迎访问： 西北工业大学 - 大数据与知识管理研究室 （Northwestern Polytechnical University - BigData and Knowledge Management Lab），链接：<http://wowbigdata.cn/>，<http://wowbigdata.net.cn/>，[http://wowbigdata.com.cn](http://wowbigdata.com.cn/)。

1. 在Vertex类中，顶点的存储方式采用邻接表形式。每个顶点有 VertexId、VertexValue、OutgoingEdges和Halt，boolean型的halt变量用于记录顶点的状态，false时表示active，true表示inactive状态。 片段代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/24241951)[在CODE上查看代码片](https://code.csdn.net/snippets/304382)

1. /\*\* Vertex id. \*/
2. **private** I id;
3. /\*\* Vertex value. \*/
4. **private** V value;
5. /\*\* Outgoing edges. \*/
6. **private** OutEdges<I, E> edges;
7. /\*\* If true, do not do anymore computation on this vertex. \*/
8. **private** **boolean** halt;
9. /\*\* Global graph state \*\*/
10. **private** GraphState<I, V, E, M> graphState;

2 org.apache.giraph.edge.Edge 接口，用于存储顶点的边，每条边包含targetVertexId和edgeValue两个属性。 类关系图如下：

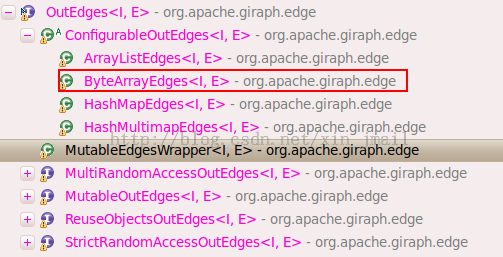


     Giraph默认使用DefaultEdge类存储边，该类中有两个变量： I targetVertexId和 E value，I为顶点ID的类型，E为边的类型。注意，DefaultEdge类同时继承ReusableEdge<I,E>接口，在ReusableEdge<I,E>类的定义中，有如下说明文字：

A complete edge, the target vertex and the edge value. Can only be one edge with a destination vertex id per edge map. This edge can be reused, that is you can set it's target vertex ID and edge value. Note: this class is useful for certain optimizations, but it's not meant to be exposed to the user. Look at MutableEdge instead.

    从上述说明文字可知，edge可以被重用，只需要修改targetVertexId和value的值就行。即**每个Vertex若有多条出边，只会创建一个DefaultEdge对象来存储边**。

3. org.apache.giraph.edge.OutEdges<I,E> 用于存储每个顶点的out-edges。从Vertex类的定义可知，顶点的每条边都被存储在OutEdges<I,E>类型的edge对象中，OutEdges<I,E>接口的关系图如下：



    Giraph默认的使用ByteArrayEdges<I,E>，每个顶点的所有边都被存储在byte[ ]中。当顶点向它的出边发送消息时，需要遍历Vertex类中的edges对象。示例代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/24241951)[在CODE上查看代码片](https://code.csdn.net/snippets/304382)

1. //遍历所有的边。getEdges()返回的是Vertex中的edges对象，
2. //那么该for循环会调用edges对象的iterator()方法，即调用ByteArrayEdges类中的iterator方法。
3. **for** (Edge<LongWritable, FloatWritable> edge : getEdges()) {
4. //edge对象表示每条边，默认为DefaultEdge类型。
5. **double** distance = minDist + edge.getValue().get();
6. sendMessage(edge.getTargetVertexId(), **new** DoubleWritable(distance));
7. }

**注意：由DefaultEdge的定义可知，遍历getEdges时，返回的Edge对象时同一个对象，只是该对象中值改变了**。下面继续查看代码来证明此观点。

查看ByteArrayEdges类的iterator()方法，如下。

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/24241951)[在CODE上查看代码片](https://code.csdn.net/snippets/304382)

1. @Override
2. **public** Iterator<Edge<I, E>> iterator() {
3. **return** **new** ByteArrayEdgeIterator();
4. }

返回的是内部类ByteArrayEdgeIterator对象，定义如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/24241951)[在CODE上查看代码片](https://code.csdn.net/snippets/304382)

1. /\*\*
2. \* Iterator that reuses the same Edge object.
3. \*/
4. **private** **class** ByteArrayEdgeIterator
5. **extends** UnmodifiableIterator<Edge<I, E>> {
6. //extendedDataInput存储所有Edge边对应的字节
7. /\*\* Input for processing the bytes \*/
8. **private** ExtendedDataInput extendedDataInput =
9. getConf().createExtendedDataInput(
10. serializedEdges, 0, serializedEdgesBytesUsed);
11. //创建一个Edge对象，默认返回的是DefaultEdge对象。
12. /\*\* Representative edge object. \*/
13. **private** ReusableEdge<I, E> representativeEdge =
14. getConf().createReusableEdge();
16. @Override
17. **public** **boolean** hasNext() {
18. **return** serializedEdges != **null** && extendedDataInput.available() > 0;
19. }
21. @Override
22. **public** Edge<I, E> next() {
23. **try** {
24. //核心：此处遍历每条Edge时，都是从extendedDataInput读入每天边的数据存储在representativeEdge对象中。
25. //从此处就可知，每个顶点的所有出边只有一个Edge对象， 遍历时修改每条边的数据的即可
26. WritableUtils.readEdge(extendedDataInput, representativeEdge);
27. } **catch** (IOException e) {
28. **throw** **new** IllegalStateException("next: Failed on pos " +
29. extendedDataInput.getPos() + " edge " + representativeEdge);
30. }
31. **return** representativeEdg
32. }
33. }

总结：当顶点的出度很大时，此优化甚好，能很好的节约内存。如UK-2005数据中，顶点的最大出度为 5213。

假设顶点**1**的出度顶点有<2 , 0.4>，<3 , 7.8> ，<5 , 6.4> 。如下代码：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/24241951)[在CODE上查看代码片](https://code.csdn.net/snippets/304382)

1. //定义list列表用于存储出度顶点的Id。
2. List<LongWritable> list=**new** ArrayList<LongWritable>();
3. **for** (Edge<LongWritable, FloatWritable> edge : getEdges()) {
4. list.add(edge.getTargetVertexId());
5. System.out.println(list);
6. }

输出结果为：

[ 2 ]

[ 3 ,  3 ]

[ 5 , 5 , 5 ]

并非是希望的 [ 2 , 3 , 5 ]

完！

本人原创，转载请注明出处！ 本人QQ：530422429，欢迎大家指正、讨论。

版权声明：本文为博主原创文章，未经博主允许不得转载。