

```

01 //
02 // Created by along on 17-11-26.
03 //
04
05 #ifndef PROJECT_GRAPH_H
06 #define PROJECT_GRAPH_H
07
08 #include <vector>
09 #include <forward_list>
10 #include <functional>
11 #include <stack>
12 #include <queue>
13 #include <iostream>
14
15 template<typename Info>
16 class InfoGraph {
17 public:
18     /**
19      * 产生一个顶点为 0 -- n-1 的图
20      * @param n
21      */
22     explicit InfoGraph(unsigned long n) : vexNum(n), edgeNum(0) {};
23
24
25     /**
26      * 使用已有的图构造一个新图
27      * @param rhs
28      */
29     InfoGraph(const InfoGraph &rhs) : vexNum(rhs.vexNum), edgeNum(0) {
30         this->clone(rhs);
31     }
32 }
33

```

```

34     /**
35      * 析构函数，避免内存泄漏
36      */
37     virtual ~InfoGraph() { clear(); };
38
39     /**
40      * 拷贝赋值函数
41      * @param rhs
42      * @return
43      */
44     InfoGraph &operator=(const InfoGraph &rhs) {
45         this->clone(rhs);
46         return *this;
47     }
48
49     /**
50      * 添加一条边
51      * @param source
52      * @param sink
53      */
54     virtual void addEdge(unsigned long source, unsigned long
55     sink, Info info) { ++edgeNum; }
56
57     /**
58      * 删除一条边
59      * @param source
60      * @param sink
61      */
62     virtual void delEdge(unsigned long source, unsigned long
63     sink) { --edgeNum; }
64

```

```

65     /**
66      * 返回顶点个数
67      * @return
68      */
69     virtual unsigned long vexCount() const { return vexNum; }
70     ;
71
72     /**
73      * 边的个数
74      * @return
75      */
76     virtual unsigned long edgeCount() const { return edgeNum; }
77     };
78
79     /**
80      * 顶点的出度
81      * @param source
82      * @return
83      */
84     virtual unsigned long outDegree(unsigned long source)
85     const {
86         unsigned long outDegree = 0;
87         std::function<bool(unsigned long, unsigned long,
88 Info)> func = [&](unsigned long src, unsigned long
89 dst, Info) {
90             ++outDegree;
91             return true;
92         };
93         foreach(source, func);
94         return outDegree;
95     }
96
97     /**

```

```

98      * 顶点的入度
99      * @param source
100     * @return
101     */
102     virtual unsigned long inDegree(unsigned long source) {
103     const {
104         unsigned long inDegree = 0;
105         std::function<bool(unsigned long, unsigned long, {
106             Info)> func = [&](unsigned long src, unsigned long
107             dst, Info) {
108                 if (dst == source)
109                     ++inDegree;
110                 return true;
111             };
112             for (unsigned long i = 0; i != vexNum; ++i)
113                 foreach(i, func);
114             return inDegree;
115         };
116
117         /**
118         * 两个顶点之间是否有边
119         * @param source
120         * @param sink
121         * @return
122         */
123         virtual bool hasEdge(unsigned long source, unsigned long
124         sink) const {
125             bool hasEdge = false;
126             std::function<bool(unsigned long, unsigned long, {
127                 Info info)>
128                 func = [&](unsigned long src, unsigned long dst,

```

```

2
129         Info) {
130             if (dst == sink) {
131                 hasEdge = true;
132                 return false;
133             }
134             return true;
135         };
136         foreach(source, func);
137         return hasEdge;
138     }
139
140     /**
141     * 遍历与某个顶点相临接的所有顶点
142     * 当 func 返回值为 false 的时候可以停止访问
143     * @param source
144     * @param func
145     */
146     virtual void foreach(unsigned long source, std::function<
147     bool(unsigned long, unsigned long, Info)> &func) const
= 2
148     0;
149
150     /**
151     * 重置
152     */
153     virtual void reset() {
154         reset(vexNum);
155     }
156
157     /**
158     * 重置
159     * @param vexNum

```

```

160     */
161     virtual void reset(unsigned long vexNum) {
162         clear();
163         this->vexNum = vexNum;
164         edgeNum = 0;
165     };
166
167     /**
168     * 将 dot 图打印到流
169     * @param out
170     */
171     void printDot(std::ostream &out) {
172         out << "digraph mGraph{" << std::endl;
173         for (unsigned long i = 0; i != vexNum; ++i)
174             out << "Node" << i << "[label = \"" << i <<
175             "\"\"];<br>
176             " << std::endl;
177
178         std::function<bool(unsigned long, unsigned long, <br>
179         Info info)>
180             func = [&](unsigned long src, unsigned long dst,
181             Info info) {
182                 out << "Node" << src << " -> " << "Node" <<
183                 dst <<
184                 < "[label=" << info << "];" << std::endl;
185                 return true;
186             };
187         for (unsigned long i = 0; i != vexNum; ++i)
188             foreach(i, func);
189
190         out << "}" << std::endl;
191     }

```

```

190
191     /**
192     * 从文件构造一个图
193     * @param filename
194     */
195     void resetFromStream(std::istream &theStream) {
196         unsigned long vexNum;
197         theStream >> vexNum;
198         reset(vexNum);
199         Info info;
200         unsigned long src, dst;
201         while (theStream >> src >> dst >> info)
202             addEdge(src, dst, info);
203     }
204
205     /**
206     * 获取某个弧的信息
207     * @param src
208     * @param dst
209     * @return
210     */
211     virtual Info getArcInfo(const unsigned long src, const unsigned long dst) const =0;
212
213
214     /**
215     * 用于修改相关的值
216     * @param src
217     * @param dst
218     * @return
219     */
220     virtual Info &operator()(const unsigned long src, const unsigned long dst) const =0;
221     unsigned long dst)=0;

```

```

222 protected:
223     /**
224      * 对数据进行清空
225      */
226     virtual void clear() {
227         vexNum = 0;
228         edgeNum = 0;
229     }
230 private:
231
232     /**
233      * 克隆一个图
234      * @param graph
235      */
236     void clone(const InfoGraph &graph) {
237         reset(graph.vexCount());
238
239         std::function<bool(unsigned long, unsigned long, 2
240 Info)>
241         func = [&](unsigned long source, unsigned long 2
242 sink, Info info) {
243             this->addEdge(source, sink, info);
244             return true;
245         };
246
247         unsigned long verNum = graph.vexCount();
248         for (unsigned long ver = 0; ver != verNum; ++ver) {
249             graph.foreach(ver, func);
250         }
251     };
252
253     unsigned long vexNum;
254     unsigned long edgeNum;

```



```

255 };
256
257 /**
258  * 图的邻接表实现
259  * T:Table
260  */
261 template<typename Info>
262 class InfoGraphT : public InfoGraph<Info> {
263 public:
264     explicit InfoGraphT(unsigned long n) : InfoGraph<Info>(n) {
265     {
266         for (int i = 0; i != n; ++i) {
267             vexes.emplace_back(0, 0);
268         }
269     }
270     explicit InfoGraphT(const InfoGraph<Info> &rhs) : InfoGraph<Info>(0) {
271         InfoGraph<Info>(0) {
272             *(dynamic_cast<InfoGraph<Info> *>(this)) = rhs;
273         }
274         void addEdge(unsigned long source, unsigned long sink, Info info) override {
275             Info info;
276             if (!hasEdge(source, sink)) {
277                 InfoGraph<Info>::addEdge(source, sink, info);
278                 InfoGraphT<Info>::VexNode &src = vexes[source];
279                 InfoGraphT<Info>::VexNode &dst = vexes[sink];
280
281                 src.adjVex.emplace_front(sink, info);
282
283                 ++src.out;
284                 ++dst.in;
285             }
286         }
287         void delEdge(unsigned long source, unsigned long sink) {

```

```

288     override {
289         if (hasEdge(source, sink)) {
290             InfoGraph<Info>::delEdge(source, sink);
291             InfoGraphT<Info>::VexNode &src = vexes[source];
292             InfoGraphT<Info>::VexNode &dst = vexes[sink];
293
294             src.adjVex.remove({sink, Info()});
295
296             --src.out;
297             --dst.in;
298         }
299     };
300     inline unsigned long vexCount() const override {
301         return vexes.size();
302     }
303     unsigned long edgeCount() const override {
304         return InfoGraph<Info>::edgeCount();
305     }
306     unsigned long outDegree(unsigned long source) const &
307     override {
308         return vexes[source].out;
309     }
310     unsigned long inDegree(unsigned long source) const &
311     override {
312         return vexes[source].in;
313     }
314     bool hasEdge(unsigned long source, unsigned long sink) &
315     const override {
316         for (Arc adjVex:vexes[source].adjVex) {
317             if (adjVex.vex == sink)
318                 return true;
319         }
320         return false;

```

```

321     }
322     Info getArcInfo(const unsigned long src, const unsigned
323     )
324     long dst) const override {
325         for (const Arc &adjVex:vexes[src].adjVex) {
326             if (adjVex.vex == dst)
327                 return adjVex.info;
328         }
329         return Info();
330     }
331     void foreach(unsigned long source, std::function<bool(
332     unsigned long, unsigned long, Info)> &func) const {
333         override {
334             for (Arc sink:vexes[source].adjVex) {
335                 if (!func(source, sink.vex, sink.info))
336                     break;
337             }
338         };
339     void reset() override {
340         InfoGraph<Info>::reset();
341     }
342     void reset(unsigned long vexNum) override {
343         InfoGraph<Info>::reset(vexNum);
344         for (int i = 0; i != vexNum; ++i) {
345             vexes.emplace_back(0, 0);
346         }
347     };
348     Info &operator()(const unsigned long src, const unsigned
349     )
350     long dst) override {
351         for (Arc &adjVex:vexes[src].adjVex) {
352             if (adjVex.vex == dst)
353                 return adjVex.info;

```

```

352     }
353 }
354 private:
355     void clear() override {
356         InfoGraph<Info>::clear();
357         vexes.clear();
358     }
359     /** 弧及其信息 */
360     typedef struct Arc {
361         unsigned long vex;
362         Info info;
363         Arc(unsigned long vex, Info info) : vex(vex), info(info) {}
364         bool operator==(const Arc &rhs) const {
365             return vex == rhs.vex;
366         }
367     } Arc;
368     /** 顶点表的数据结构 */
369     typedef struct VexNode {
370         unsigned long in;
371         unsigned long out;
372         std::forward_list<Arc> adjVex;
373         VexNode(unsigned long in, unsigned long out, const std::forward_list<Arc> &adjVex)
374             : in(in), out(out), adjVex(adjVex) {}
375         VexNode(unsigned long in, unsigned long out) : in(in), out(out) {}
376     } VexNode;
377     /** 邻接表顶点 */
378     std::vector<VexNode> vexes;
379 };
380 /**

```

```

385 * 图的邻接矩阵实现
386 * M:Matrix
387 */
388 template<typename Info>
389 class InfoGraphM : public InfoGraph<Info> {
390 public:
391     explicit InfoGraphM(unsigned long n) : InfoGraph<Info>(n) {
392     {
393         for (unsigned long i = 0; i != n; ++i) {
394             vexes.emplace_back(n, false);
395         }
396     }
397     explicit InfoGraphM(const InfoGraph<Info> &rhs) : InfoGraph<Info>(0) {
398         *(dynamic_cast<InfoGraph<Info>*>(this)) = rhs;
399     }
400 }
401 void addEdge(unsigned long source, unsigned long sink, Info info) override {
402     InfoGraph<Info>::addEdge(source, sink, info);
403     vexes[source][sink].link = true;
404     vexes[source][sink].info = info;
405 }
406 void delEdge(unsigned long source, unsigned long sink) override {
407     InfoGraph<Info>::delEdge(source, sink);
408     vexes[source][sink].link = false;
409 }
410 inline unsigned long vexCount() const override {
411     return InfoGraph<Info>::vexCount();
412 }
413 unsigned long edgeCount() const override {
414     return InfoGraph<Info>::edgeCount();
415 }

```

```

418     unsigned long outDegree(unsigned long source) const {
419     override {
420         unsigned long count = 0;
421         for (VexNode out:vexes[source])
422             count += out.link;
423         return count;
424     }
425     unsigned long inDegree(unsigned long source) const {
426     override {
427         unsigned long count = 0;
428         for (unsigned long i = 0; i != vexCount(); ++i)
429             count += vexes[i][source].link;
430         return count;
431     }
432     bool hasEdge(unsigned long source, unsigned long sink) {
433     const override {
434         return vexes[source][sink].link;
435     }
436     Info getArcInfo(const unsigned long src, const unsigned
437     long dst) const override {
438         return vexes[src][dst].info;
439     }
440     void foreach(unsigned long source, std::function<bool(
441     unsigned long, unsigned long, Info)> &func) const {
442     override {
443         for (unsigned long adjVec = 0; adjVec != vexCount();
444         ++adjVec) {
445             VexNode vex = vexes[source][adjVec];
446             if (vex.link)
447                 func(source, adjVec, vex.info);
448         }

```

```

449
450     }
451     void reset() override {
452         InfoGraph<Info>::reset();
453     }
454     void reset(unsigned long vexNum) override {
455         InfoGraph<Info>::reset(vexNum);
456         for (unsigned long i = 0; i != vexNum; ++i) {
457             vexes.emplace_back(vexNum, false);
458         }
459     }
460     Info &operator()(const unsigned long src, const unsigned
461     long dst) override {
462         return vexes[src][dst].info;
463     }
464 private:
465     void clear() override {
466         InfoGraph<Info>::clear();
467         vexes.clear();
468     }
469     typedef struct VexNode {
470         bool link;
471         Info info;
472         VexNode(bool link) : link(link), info(Info()) {}
473         VexNode(bool link, Info info) : link(link), info(
474         info) {}
475     } VexNode;
476     std::vector<std::vector<VexNode>> vexes;
477 };
478
479 /**
480  * 图的十字链表实现

```

```

481  * L:List
482  */
483  template<typename Info>
484  class InfoGraphL : public InfoGraph<Info> {
485  public:
486      explicit InfoGraphL(unsigned long n) : InfoGraph<Info>(n) {
487          {
488              for (unsigned long i = 0; i != n; ++i)
489                  vexes.emplace_back(VexNode());
490          }
491      explicit InfoGraphL(const InfoGraph<Info> &rhs) : InfoGraph<Info>(0) {
492          InfoGraph<Info>(0) {
493              *(dynamic_cast<InfoGraph<Info> *>(this)) = rhs;
494          }
495      ~InfoGraphL() override { clear(); };
496      void addEdge(unsigned long source, unsigned long sink, Info info) override {
497          if (!hasEdge(source, sink)) {
498              InfoGraph<Info>::addEdge(source, sink, info);
499              auto &src = vexes[source];
500              auto &dst = vexes[sink];
501
502              ++src.out;
503              ++dst.in;
504
505              auto *newArc = new ArcBox(source, sink, nullptr,
506              nullptr, info);
507              if (src.firstOut == nullptr)
508                  src.firstOut = newArc;
509              else {
510                  ArcBox **findArc = &src.firstOut;
511                  while ((*findArc) != nullptr && (*findArc)->

```



```

513         tailVex < sink)
514             findArc = &(*findArc)->hLink;
515         newArc->hLink = *findArc;
516         *findArc = newArc;
517     }
518
519     if (dst.firstIn == nullptr)
520         dst.firstIn = newArc;
521     else {
522         ArcBox **findArc = &dst.firstIn;
523         while ((*findArc) != nullptr && (*findArc)->
524             headVex < source)
525             findArc = &(*findArc)->tLink;
526         newArc->tLink = *findArc;
527         *findArc = newArc;
528     }
529 }
530 }
531 void delEdge(unsigned long source, unsigned long sink) {
532     override {
533         if (hasEdge(source, sink)) {
534             InfoGraph<Info>::delEdge(source, sink);
535             auto &src = vexes[source];
536             auto &dst = vexes[sink];
537
538             --src.out;
539             --dst.in;
540
541             ArcBox **head = &src.firstOut, **tail = &dst.
542             firstIn;
543             while ((*tail)->headVex != source)
544                 tail = &(*tail)->tLink;
545             *tail = (*tail)->tLink;

```

```

546
547         while ((*head)->tailVex != sink)
548             head = &(*head)->hLink;
549         // 2
550         废物利用，暂时保存一下要释放的内 2
551         ηŸ 指针
552         *tail = *head;
553         *head = (*head)->hLink;
554         delete *tail;
555     }
556 }
557 unsigned long vexCount() const override {
558     return InfoGraph<Info>::vexCount();
559 }
560 unsigned long edgeCount() const override {
561     return InfoGraph<Info>::edgeCount();
562 }
563 unsigned long outDegree(unsigned long source) const 2
564 override {
565     return vexes[source].out;
566 }
567 unsigned long inDegree(unsigned long source) const 2
568 override {
569     return vexes[source].in;
570 }
571 bool hasEdge(unsigned long source, unsigned long sink) 2
572 const override {
573     for (ArcBox *arc = vexes[source].firstOut; arc != 2
574         nullptr; arc = arc->hLink)
575         if (arc->tailVex == sink)
576             return true;
577     return false;
578 }

```

```

579 Info getArcInfo(const unsigned long src, const unsigned
580 long dst) const override {
581     for (ArcBox *arc = vexes[src].firstOut; arc != nullptr; arc = arc->hLink)
582         if (arc->tailVex == dst)
583             return arc->info;
584     return Info();
585 }
586 void foreach(unsigned long source, std::function<bool(
587 unsigned long, unsigned long, Info)> &func) const override {
588     Info info;
589     for (const ArcBox *arc = vexes[source].firstOut; arc
590         != nullptr; arc = arc->hLink)
591         if (!func(source, arc->tailVex, info))
592             break;
593 }
594 void foreachIn(unsigned long dst, std::function<bool(
595 unsigned long, unsigned long, Info)> &func) const {
596     for (const ArcBox *arc = vexes[dst].firstIn; arc !=
597         nullptr; arc = arc->tLink)
598         if (!func(arc->headVex, dst, arc->info))
599             break;
600 }
601 void reset() override {
602     InfoGraph<Info>::reset();
603 }
604 void reset(unsigned long vexNum) override {
605     InfoGraph<Info>::reset(vexNum);
606     for (unsigned long i = 0; i != vexNum; ++i)

```

```

609         vexes.emplace_back(VexNode());
610     }
611     Info &operator()(const unsigned long src, const unsigned
612     2
613     long dst) override {
614         for (ArcBox *arc = vexes[src].firstOut; arc != 2
615         nullptr; arc = arc->hLink)
616             if (arc->tailVex == dst)
617                 return arc->info;
618     }
619 private:
620     typedef struct ArcBox {
621         unsigned long headVex, tailVex;
622         ArcBox *hLink, *tLink;
623         Info info;
624         ArcBox(unsigned long head, unsigned long tail, 2
625         ArcBox *headLink, ArcBox *tailLink, Info theInfo) :
626             headVex(head), tailVex(tail), hLink(headLink), 2
627             tLink(tailLink), info(theInfo) {}
628     } ArcBox;
629     typedef struct VexNode {
630         unsigned long in, out;
631         ArcBox *firstIn, *firstOut;
632         VexNode() : in(0), out(0), firstIn(nullptr), 2
633         firstOut(nullptr) {};
634     } VexNode;
635     void clear() override {
636         InfoGraph<Info>::clear();
637         for (VexNode &vex:vexes) {
638             while (vex.firstOut != nullptr) {
639                 ArcBox *curr = vex.firstOut;
640                 vex.firstOut = curr->tLink;
641                 delete curr;

```

```
641         }
642     }
643     vexes.clear();
644 }
645     std::vector<VexNode> vexes;
646 };
647
648 #endif //PROJECT_GRAPH_H
```