```cpp
//
// Created by along on 17-11-26.
//

#ifndef PROJECT_GRAPH_H
#define PROJECT_GRAPH_H

#include <vector>
#include <forward_list>
#include <functional>
#include <stack>
#include <queue>
#include <iostream>

template<typename Info>
class InfoGraph {
public:
    /**
     * 产生一个顶点为 0 -- n-1 的图
     * @param n
     */
    explicit InfoGraph(unsigned long n) : vexNum(n), edgeNum(0) {};

    /**
     * 使用已有的图构造一个新图
     * @param rhs
     */
    InfoGraph(const InfoGraph &rhs) : vexNum(rhs.vexNum), edgeNum(0) {
        this->clone(rhs);
    }

    /**
     * 析构函数，避免内存泄漏
     */
    virtual ~InfoGraph() { clear(); };

    /**
     * 拷贝赋值函数
     * @param rhs
     * @return
     */
    InfoGraph &operator=(const InfoGraph &rhs) {
        this->clone(rhs);
        return *this;
    }

    /**
     * 添加一条边
     * @param source
     * @param sink
     */
    virtual void addEdge(unsigned long source, unsigned long sink, Info info) { ++edgeNum; }

    /**
     * 删除一条边
     * @param source
     * @param sink
     */
    virtual void delEdge(unsigned long source, unsigned long sink) { --edgeNum; }

```

```cpp
/**
 * 返回顶点个数
 * @return
 */
virtual unsigned long vexCount() const { return vexNum; };

/**
 * 边的个数
 * @return
 */
virtual unsigned long edgeCount() const { return edgeNum; };

/**
 * 顶点的出度
 * @param source
 * @return
 */
virtual unsigned long outDegree(unsigned long source) const {
    unsigned long outDegree = 0;
    std::function<bool(unsigned long, unsigned long, Info)> func = [&](unsigned long src, unsigned long dst, Info) {
        ++outDegree;
        return true;
    };
    foreach(source, func);
    return outDegree;
}

/**
```

```
* 顶点的入度
* @param source
 * @return
 */
virtual unsigned long inDegree(unsigned long source) const {
    unsigned long inDegree = 0;
    std::function<bool(unsigned long, unsigned long, Info)> func = [&](unsigned long src, unsigned long dst, Info) {
        if (dst == source)
            ++inDegree;
        return true;
    };
    for (unsigned long i = 0; i != vexNum; ++i)
        foreach(i, func);
    return inDegree;
};

/**
 * 两个顶点之间是否有边
 * @param source
 * @param sink
 * @return
 */
virtual bool hasEdge(unsigned long source, unsigned long sink) const {
    bool hasEdge = false;
    std::function<bool(unsigned long, unsigned long, Info info)>
        func = [&](unsigned long src, unsigned long dst,
⤸
        Info) {
        if (dst == sink) {
            hasEdge = true;
            return false;
        }
        return true;
    };
    foreach(source, func);
    return hasEdge;
}

/**
 * 遍历与某个顶点相临接的所有顶点
 * 当 func 返回值为 false 的时候可以停止访问
 * @param source
 * @param func
 */
virtual void foreach(unsigned long source, std::function<⤸
bool(unsigned long, unsigned long, Info)> &func) const
= ⤸
0;

/**
 * 重置
 */
virtual void reset() {
    reset(vexNum);
}

/**
 * 重置
 * @param vexNum
 */
virtual void reset(unsigned long vexNum) {
    clear();
    this->vexNum = vexNum;
    edgeNum = 0;
};

/**
 * 将 dot 图打印到流
 * @param out
 */
void printDot(std::ostream &out) {
    out << "digraph mGraph{" << std::endl;
    for (unsigned long i = 0; i != vexNum; ++i)
        out << "Node" << i << "[label = \"" << i << "\"];
        " << std::endl;

    std::function<bool(unsigned long, unsigned long,
    Info info)>
        func = [&](unsigned long src, unsigned long dst,
        Info info) {
        out << "Node" << src << " -> " << "Node" << dst <
        < "[label=" << info << "];" << std::endl;
        return true;
    };
    for (unsigned long i = 0; i != vexNum; ++i)
        foreach(i, func);

    out << "}" << std::endl;
}
```

```

/**
 * 从文件构造一个图
 * @param filename
 */
void resetFromStream(std::istream &theStream) {
    unsigned long vexNum;
    theStream >> vexNum;
    reset(vexNum);
    Info info;
    unsigned long src, dst;
    while (theStream >> src >> dst >> info)
        addEdge(src, dst, info);
}

/**
 * 获取某个弧的信息
 * @param src
 * @param dst
 * @return
 */
virtual Info getArcInfo(const unsigned long src, const ⤸
unsigned long dst) const =0;

/**
 * 用于修改相关的值
 * @param src
 * @param dst
 * @return
 */
virtual Info &operator()(const unsigned long src, const
⤸

unsigned long dst)=0;
```

```
protected:
    /**
     * 对数据进行清空
     */
    virtual void clear() {
        vexNum = 0;
        edgeNum = 0;
    }
private:

    /**
     * 克隆一个图
     * @param graph
     */
    void clone(const InfoGraph &graph) {
        reset(graph.vexCount());

        std::function<bool(unsigned long, unsigned long, ⤸
        Info)>
            func = [&](unsigned long source, unsigned long ⤸
            sink, Info info) {
            this->addEdge(source, sink, info);
            return true;
        };

        unsigned long verNum = graph.vexCount();
        for (unsigned long ver = 0; ver != verNum; ++ver) {
            graph.foreach(ver, func);
        }
    };

    unsigned long vexNum;
    unsigned long edgeNum;
```

```
};

/**
 * 图的邻接表实现
 * T:Table
 */
template<typename Info>
class InfoGraphT : public InfoGraph<Info> {
public:
    explicit InfoGraphT(unsigned long n) : InfoGraph<Info>(n)⤸
    {
        for (int i = 0; i != n; ++i) {
            vexes.emplace_back(0, 0);
        }
    }
    explicit InfoGraphT(const InfoGraph<Info> &rhs) : ⤸
    InfoGraph<Info>(0) {
        *(dynamic_cast<InfoGraph<Info> *>(this)) = rhs;
    }
    void addEdge(unsigned long source, unsigned long sink, ⤸
    Info info) override {
        if (!hasEdge(source, sink)) {
            InfoGraph<Info>::addEdge(source, sink, info);
            InfoGraphT<Info>::VexNode &src = vexes[source];
            InfoGraphT<Info>::VexNode &dst = vexes[sink];

            src.adjVex.emplace_front(sink, info);

            ++src.out;
            ++dst.in;
        }
    }
    void delEdge(unsigned long source, unsigned long sink) ⤸
```

```
override {
    if (hasEdge(source, sink)) {
        InfoGraph<Info>::delEdge(source, sink);
        InfoGraphT<Info>::VexNode &src = vexes[source];
        InfoGraphT<Info>::VexNode &dst = vexes[sink];

        src.adjVex.remove({sink, Info()});

        --src.out;
        --dst.in;
    }
};
inline unsigned long vexCount() const override {
    return vexes.size();
}
unsigned long edgeCount() const override {
    return InfoGraph<Info>::edgeCount();
}
unsigned long outDegree(unsigned long source) const ⤸
override {
    return vexes[source].out;
}
unsigned long inDegree(unsigned long source) const ⤸
override {
    return vexes[source].in;
}
bool hasEdge(unsigned long source, unsigned long sink) ⤸
const override {
    for (Arc adjVex:vexes[source].adjVex) {
        if (adjVex.vex == sink)
            return true;
    }
    return false;
```

```
}
Info getArcInfo(const unsigned long src, const unsigned
↩
long dst) const override {
    for (const Arc &adjVex:vexes[src].adjVex) {
        if (adjVex.vex == dst)
            return adjVex.info;
    }
    return Info();
}
void foreach(unsigned long source, std::function<bool(↩
unsigned long, unsigned long, Info)> &func) const ↩
override {
    for (Arc sink:vexes[source].adjVex) {
        if (!func(source, sink.vex, sink.info))
            break;
    }
};
void reset() override {
    InfoGraph<Info>::reset();
}
void reset(unsigned long vexNum) override {
    InfoGraph<Info>::reset(vexNum);
    for (int i = 0; i != vexNum; ++i) {
        vexes.emplace_back(0, 0);
    }
};
Info &operator()(const unsigned long src, const unsigned
↩
long dst) override {
    for (Arc &adjVex:vexes[src].adjVex) {
        if (adjVex.vex == dst)
            return adjVex.info;
        }
    }
private:
    void clear() override {
        InfoGraph<Info>::clear();
        vexes.clear();
    }
    /** 弧及其信息 */
    typedef struct Arc {
        unsigned long vex;
        Info info;
        Arc(unsigned long vex, Info info) : vex(vex), info(
        info) {}
        bool operator==(const Arc &rhs) const {
            return vex == rhs.vex;
        }
    } Arc;
    /** 顶点表的数据结构 */
    typedef struct VexNode {
        unsigned long in;
        unsigned long out;
        std::forward_list<Arc> adjVex;
        VexNode(unsigned long in, unsigned long out, const 
        std::forward_list<Arc> &adjVex)
            : in(in), out(out), adjVex(adjVex) {}
        VexNode(unsigned long in, unsigned long out) : in(in)
        , out(out) {}
    } VexNode;
    /** 邻接表顶点 */
    std::vector<VexNode> vexes;
};

/**
```

```
 * 图的邻接矩阵实现
 * M:Matrix
 */
template<typename Info>
class InfoGraphM : public InfoGraph<Info> {
public:
    explicit InfoGraphM(unsigned long n) : InfoGraph<Info>(n)⤸
    {
        for (unsigned long i = 0; i != n; ++i) {
            vexes.emplace_back(n, false);
        }
    }
    explicit InfoGraphM(const InfoGraph<Info> &rhs) : ⤸
    InfoGraph<Info>(0) {
        *(dynamic_cast<InfoGraph<Info> *>(this)) = rhs;
    }
    void addEdge(unsigned long source, unsigned long sink, ⤸
    Info info) override {
        InfoGraph<Info>::addEdge(source, sink, info);
        vexes[source][sink].link = true;
        vexes[source][sink].info = info;
    }
    void delEdge(unsigned long source, unsigned long sink) ⤸
    override {
        InfoGraph<Info>::delEdge(source, sink);
        vexes[source][sink].link = false;
    }
    inline unsigned long vexCount() const override {
        return InfoGraph<Info>::vexCount();
    }
    unsigned long edgeCount() const override {
        return InfoGraph<Info>::edgeCount();
    }
```

```
unsigned long outDegree(unsigned long source) const ⤸
override {
    unsigned long count = 0;
    for (VexNode out:vexes[source])
        count += out.link;
    return count;
}
unsigned long inDegree(unsigned long source) const ⤸
override {
    unsigned long count = 0;
    for (unsigned long i = 0; i != vexCount(); ++i)
        count += vexes[i][source].link;
    return count;
}
bool hasEdge(unsigned long source, unsigned long sink) ⤸
const override {
    return vexes[source][sink].link;
}
Info getArcInfo(const unsigned long src, const unsigned
⤸
long dst) const override {
    return vexes[src][dst].info;
}
void foreach(unsigned long source, std::function<bool(⤸
unsigned long, unsigned long, Info)> &func) const ⤸
override {
    for (unsigned long adjVec = 0; adjVec != vexCount();
⤸
    ++adjVec) {
        VexNode vex = vexes[source][adjVec];
        if (vex.link)
            func(source, adjVec, vex.info);
    }

    }
    void reset() override {
        InfoGraph<Info>::reset();
    }
    void reset(unsigned long vexNum) override {
        InfoGraph<Info>::reset(vexNum);
        for (unsigned long i = 0; i != vexNum; ++i) {
            vexes.emplace_back(vexNum, false);
        }
    }
    Info &operator()(const unsigned long src, const unsigned long dst) override {
        return vexes[src][dst].info;
    }
private:
    void clear() override {
        InfoGraph<Info>::clear();
        vexes.clear();
    }
    typedef struct VexNode {
        bool link;
        Info info;
        VexNode(bool link) : link(link), info(Info()) {}
        VexNode(bool link, Info info) : link(link), info(info) {}
    } VexNode;
    std::vector<std::vector<VexNode>> vexes;
};

/**
 * 图的十字链表实现
 * L:List
 */
template<typename Info>
class InfoGraphL : public InfoGraph<Info> {
public:
    explicit InfoGraphL(unsigned long n) : InfoGraph<Info>(n)⤸
    {
        for (unsigned long i = 0; i != n; ++i)
            vexes.emplace_back(VexNode());
    }
    explicit InfoGraphL(const InfoGraph<Info> &rhs) : ⤸
    InfoGraph<Info>(0) {
        *(dynamic_cast<InfoGraph<Info> *>(this)) = rhs;
    }
    ~InfoGraphL() override { clear(); };
    void addEdge(unsigned long source, unsigned long sink, ⤸
    Info info) override {
        if (!hasEdge(source, sink)) {
            InfoGraph<Info>::addEdge(source, sink, info);
            auto &src = vexes[source];
            auto &dst = vexes[sink];

            ++src.out;
            ++dst.in;

            auto *newArc = new ArcBox(source, sink, nullptr,
⤸
            nullptr, info);
            if (src.firstOut == nullptr)
                src.firstOut = newArc;
            else {
                ArcBox **findArc = &src.firstOut;
                while ((*findArc) != nullptr && (*findArc)->⤸
                tailVex < sink)
                    findArc = &(*findArc)->hLink;
                newArc->hLink = *findArc;
                *findArc = newArc;
            }

            if (dst.firstIn == nullptr)
                dst.firstIn = newArc;
            else {
                ArcBox **findArc = &dst.firstIn;
                while ((*findArc) != nullptr && (*findArc)->headVex < source)
                    findArc = &(*findArc)->tLink;
                newArc->tLink = *findArc;
                *findArc = newArc;
            }
        }
    }
    void delEdge(unsigned long source, unsigned long sink) override {
        if (hasEdge(source, sink)) {
            InfoGraph<Info>::delEdge(source, sink);
            auto &src = vexes[source];
            auto &dst = vexes[sink];

            --src.out;
            --dst.in;

            ArcBox **head = &src.firstOut, **tail = &dst.firstIn;
            while ((*tail)->headVex != source)
                tail = &(*tail)->tLink;
            *tail = (*tail)->tLink;

            while ((*head)->tailVex != sink)
                head = &(*head)->hLink;
            // ⤸
            废物利用，暂时保存一下要释放的内 ⤸
            ŋŸ 指针
            *tail = *head;
            *head = (*head)->hLink;
            delete *tail;
        }
    }
    unsigned long vexCount() const override {
        return InfoGraph<Info>::vexCount();
    }
    unsigned long edgeCount() const override {
        return InfoGraph<Info>::edgeCount();
    }
    unsigned long outDegree(unsigned long source) const ⤸
    override {
        return vexes[source].out;
    }
    unsigned long inDegree(unsigned long source) const ⤸
    override {
        return vexes[source].in;
    }
    bool hasEdge(unsigned long source, unsigned long sink) ⤸
    const override {
        for (ArcBox *arc = vexes[source].firstOut; arc != ⤸
        nullptr; arc = arc->hLink)
            if (arc->tailVex == sink)
                return true;
        return false;
    }
    Info getArcInfo(const unsigned long src, const unsigned long dst) const override {
        for (ArcBox *arc = vexes[src].firstOut; arc != nullptr; arc = arc->hLink)
            if (arc->tailVex == dst)
                return arc->info;
        return Info();
    }
    void foreach(unsigned long source, std::function<bool(unsigned long, unsigned long, Info)> &func) const override {
        Info info;
        for (const ArcBox *arc = vexes[source].firstOut; arc != nullptr; arc = arc->hLink)
            if (!func(source, arc->tailVex, info))
                break;
    }
    void foreachIn(unsigned long dst, std::function<bool(unsigned long, unsigned long, Info)> &func) const {
        for (const ArcBox *arc = vexes[dst].firstIn; arc != nullptr; arc = arc->tLink)
            if (!func(arc->headVex, dst, arc->info))
                break;
    }
    void reset() override {
        InfoGraph<Info>::reset();
    }
    void reset(unsigned long vexNum) override {
        InfoGraph<Info>::reset(vexNum);
        for (unsigned long i = 0; i != vexNum; ++i)
```

```cpp
            vexes.emplace_back(VexNode());
    }
    Info &operator()(const unsigned long src, const unsigned long dst) override {
        for (ArcBox *arc = vexes[src].firstOut; arc != nullptr; arc = arc->hLink)
            if (arc->tailVex == dst)
                return arc->info;
    }
private:
    typedef struct ArcBox {
        unsigned long headVex, tailVex;
        ArcBox *hLink, *tLink;
        Info info;
        ArcBox(unsigned long head, unsigned long tail, ArcBox *headLink, ArcBox *tailLink, Info theInfo) :
            headVex(head), tailVex(tail), hLink(headLink), tLink(tailLink), info(theInfo) {}
    } ArcBox;
    typedef struct VexNode {
        unsigned long in, out;
        ArcBox *firstIn, *firstOut;
        VexNode() : in(0), out(0), firstIn(nullptr), firstOut(nullptr) {};
    } VexNode;
    void clear() override {
        InfoGraph<Info>::clear();
        for (VexNode &vex:vexes) {
            while (vex.firstOut != nullptr) {
                ArcBox *curr = vex.firstOut;
                vex.firstOut = curr->tLink;
                delete curr;
            }
        }
        vexes.clear();
    }
    std::vector<VexNode> vexes;
};

#endif //PROJECT_GRAPH_H
```