**实验报告**

姓名：曾颖

学号：1501214439

## 实验环境

|  |  |
| --- | --- |
| 操作系统 | OS X EI Capitan |
| 处理器 | 2.9 GHz Intel Core i5 |
| 内存 | 8 GB 1867 MHz DDR3 |

## 编译运行

g++ VF2.cpp –o VF2 –std=c++11 –O2

./VF2

## 算法框架

## 数据结构和算法实现

## 类型名和常量

|  |  |
| --- | --- |
| typedef int VIndex | VIndex，顶点的编号 |
| typedef int EIndex | EIndex，边的编号 |
| typedef int VLabel | VLabel，顶点的属性 |
| typedef int ELabel | ELabel，边的属性 |
| const VIndex NULL\_VIndex = -1 | 因为点的标号从0开始，所以用-1表示该点不存在，用于初始化和表示图中的点是否找到了匹配 |
| const VIndex NULL\_EIndex = -1 | 因为边的标号从0开始，所以用-1表示该边不存在，用于初始化 |
| const set<VIndex> NULL\_VERTEX\_SET = {} | 空点集，用于初始化 |

## 类

|  |  |  |
| --- | --- | --- |
| Edge | | |
| 类属性 | 数据类型 | 含义 |
| u | int | 边的起点 |
| v | int | 边的终点 |
| label | int | 边的标签，属性 |
| next | int | 边在邻接表中的下一条边的编号 |
| prev | int | 边在邻接表中的前一条边的编号 |
| Graph | | |
| 类属性 | 数据类型 | 含义 |
| vertex\_count | int | 图中点的总数 |
| edge\_count | int | 图中边的总数 |
| vertex | vector<VLabel> | 每个点的标签，属性 |
| edge | vector<Edge> | 存储图中的边 |
| head\_edge | vector<EIndex> | 邻接表的顶点数组 |
| rev\_head\_edge | vector<EIndex> | 邻接表的反向顶点数组，用于寻找所有以某一点为终点的边 |
| pred | vector<set<VIndex>> | 存储每个点的前继节点 |
| succ | vector<set<VIndex>> | 存储每个点的后继节点 |
| 类方法 | | 含义 |
| void addVertex(int label) | | 向图中增加一个点 |
| void addEdge(int u, int v, int label) | | 向图中添加一条从u到v，标号为label的边 |
| void initial() | | 初始化，清空一个图 |
| void printGraphInfo() const | | 输出图的结构信息 |
| State | | |
| 类属性 | 数据类型 | 含义 |
| vertex\_count | int | 查询图的点数 |
| subisomorphism | bool | 当前查询为子图同构还是图同构 |
| M1 | set<VIndex> | G1（或G2）中已成功匹配的点 |
| M2 |
| core\_1 | vector<VIndex> | G1（或G2）中每个点在G2（或G1）所匹配的点，如果当前还没有匹配，设成NULL\_VIndex |
| core\_2 |
| in\_1 | set<VIndex> | 能直接连接M1（或M2）中的点，且不属于M1（或M2）的点的集合 |
| in\_2 |
| out\_1 | set<VIndex> | 能被M1（或M2）中的点直接连接，且不属于M1（或M2）的点的集合 |
| out\_2 |
| 类方法 | | 含义 |
| vector<pair<VIndex, VIndex>> genCandiPairSet() | | 生成当前状态下所有可能的匹配节点对 |
| void addNewPair(VIndex n, VIndex m…) | | 添加G1中的点n到G2中的点m的匹配到当前状态 |
| bool checkPredRule(const Graph &G1, const Graph &G2, VIndex n, VIndex m) | | 类似功能的类方法还有：checkSuccRule，checkInRule, checkOutRule，checkNewRule， 分别检查是否满足论文中提到的五项语法条件 |
| set<VIndex> genComplementary(…) | | 检查规则的辅助函数，用于计算两个集合在第三个集合的补集 |
| int set\_intersection\_size(const set<VIndex> &a, const set<VIndex> &b) | | 检查规则的辅助函数，用于计算两个集合交集的大小 |
| bool checkSynRules(const Graph &G1, const Graph &G2, VIndex n, VIndex m) | | 论文中提及的“语法”规则，即检查候选的映射(n->m)是否破坏了两图结构的匹配 |
| bool checkSemRules(const Graph &G1, const Graph &G2, VIndex n, VIndex m) | | 论文中提及的“语义”规则，即检查候选的映射(n->m)是否破坏了两图性质的匹配 |

## 全局函数

|  |  |
| --- | --- |
| void readGraph(vector<Graph> &G, int total) | 读取文件中的前total个图 |
| bool solve(const Graph &G1, const Graph &G2, State &state) | 当前匹配状态为state时，判断G1和G2是否同构，实现了第三节的算法框架 |
| bool isomorphism(const Graph &G1, const Graph &G2) | 判断G1和G2是否同构 |
| bool subisomorphism(const Graph &G1, const Graph &G2) | 判断G1是否是G2的子图同构 |

## 实验结果

实验中使用了mygraphdb.data作为图数据库，mygraphdb.data中有10000个图，平均有个25点，27条边。Q\*.my中的图为查询图，每个.my文件中有1000个图。

## 图同构

因为没有给出答案，所以没有计算匹配的正确率。

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 平均点数 | 平均边数 | 运行时间 |
| Q4.my | 5 | 4 | 73s |
| Q8.my | 9 | 8 | 74s |
| Q12.my | 13 | 12 | 75s |
| Q16.my | 16 | 16 | 77s |
| Q20.my | 20 | 20 | 78s |
| Q24.my | 23 | 24 | 82s |

## 子图同构

实验过程中因为原始数据量太大，运行程序时会出现的内存方面的错误。所以我使用了mygraphdb.data中的前5000个图，每个查询文件也只用了前500个图。

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 平均点数 | 平均边数 | 运行时间 |
| Q4.my | 5 | 4 | 70s |
| Q8.my | 9 | 9 | 57s |
| Q12.my | 13 | 12 | 91s |
| Q16.my | 16 | 16 | 116s |
| Q20.my | 19 | 19 | 125s |
| Q24.my | 23 | 23 | 128s |