VF2算法实验报告

海量图数据的管理和挖掘

2015 年 12 月 4 日

作者：马凌霄

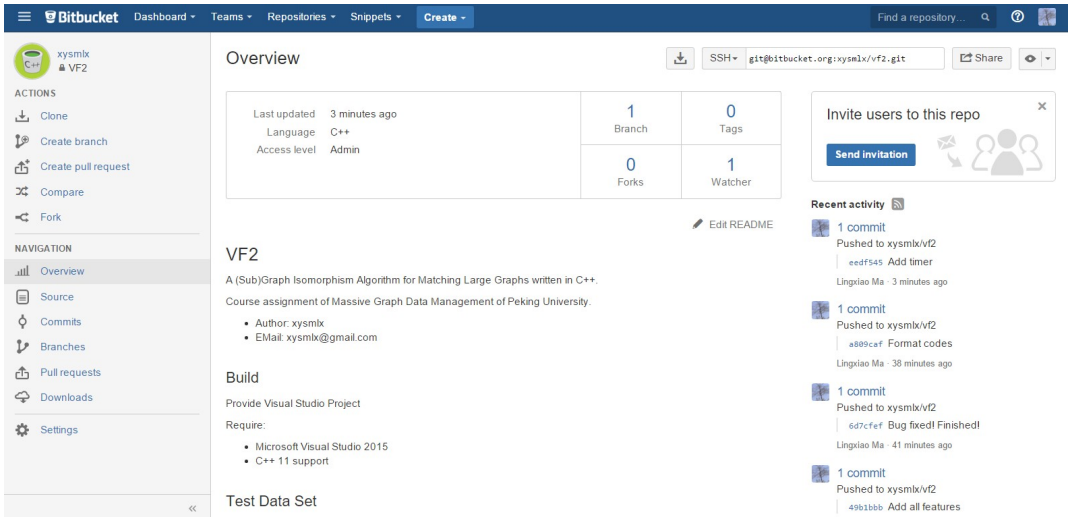
学号：1501111302

院系：信息科学技术学院

EMail：xysmlx@gmail.com

Repository (Private): https://bitbucket.org/xysmlx/vf2

Git (Remote): https://xysmlx@bitbucket.org/xysmlx/vf2.git



目　　录

§ 1 VF2算法简介

§ 2 程序实现

2.1 图-类：class Graph

2.1.1 定义代码

```
1  struct Vertex
2  {
3      int id;
4      int label;
5      int seq;
6      bool del;
7
8      Vertex(int _id = 0, int _label = 0) : id(_id), label(_label), seq(-1),
           del(0) {}
9      ~Vertex() {}
10 };
11
12 struct Edge
13 {
14     int u;
15     int v;
16     int label;
17     int next;
18     bool del;
19
20     Edge(int _u = 0, int _v = 0, int _label = 0, int _next = -1) : u(_u),
           v(_v), label(_label), next(_next), del(0) {}
21     ~Edge() {}
22
23     bool operator == (const Edge &o) const
24     {
25         return u == o.u&&v == o.v&&label == o.label;
26     }
27 };
28
29 class Graph
30 {
31 public:
32     Graph()
33     {
34         memset(head, -1, sizeof(head));
```

```
35          vn = 0;
36          en = 0;
37      }
38      ~Graph() {}
39
40      void init();
41      void addv(int id, int label);
42      void addse(int u, int v, int label);
43      void adde(int u, int v, int label);
44      void delse(int u, int v, int label);
45      void dele(int u, int v, int label);
46
47  public:
48      const static int maxv = 250;
49      const static int maxe = 510;
50
51  public:
52      int head[maxv];
53      int vn;
54      int en;
55      Vertex vtx[maxv]; // 0 to vn-1
56      Edge edge[maxe]; // 0 to en-1
57  };
```

2.1.2 设计

图的存储使用链式前向星来存储。链式前向星的效率高于使用*vector*写的邻接表。

链式前向星的标准设计是：

- *head*[]数组：大小为顶点数，存这个点的对应的第一条边在*edge*[]数组的下标
- *edge*[]数组：用数组存储边
- *Edge*边的结构：边的节点u, v，边的标号*label*，删除标记*del*，下一个访问的边的下标*next*
- 添加边：

```
void Graph::addv(int id, int label)
{
    vtx[id] = Vertex(id, label);
```

```
    vn++;
}
void Graph::addse(int u, int v, int label)
{
    edge[en] = Edge(u, v, label, head[u]);
    head[u] = en++;
}
```

- 访问一个定点的所有边

```
for (int i = head[u]; ~i; i = edge[i].next)
{
    Edge e = edge[i];
    // Solve this edge
}
```

- 删除边：令边的$del = 1$，由于边i和边i^1互为反向边，所以直接遍历一次即可。

```
void Graph::dele(int u, int v, int label)
{
    for (int i = head[u];~i;i = edge[i].next)
        if (edge[i].u == u&&edge[i].v == v&&edge[i].label == label)
        {
            edge[i].del = 1;
            edge[i ^ 1].del = 1;
            return;
        }
}
```

图里面用数组存储节点的访问顺序和点的标号，用链式前向星存储图的结构。

2.2 VF2算法-类：VF2

2.2.1 定义代码

```
1  class VF2
2  {
3  public:
4      struct State // State of dfs matching
5      {
6          vector<prii> s; // Store matched pairs
7          // Same means with the paper
8          int core1[Graph::maxv]; // core1[n] = m or -1
9          int core2[Graph::maxv]; // core2[m] = n or -1
10         bool in1[Graph::maxv];
11         bool in2[Graph::maxv];
12         bool out1[Graph::maxv];
13         bool out2[Graph::maxv];
14         State()
15         {
16             s.clear();
17             memset(core1, -1, sizeof(core1));
18             memset(core2, -1, sizeof(core2));
19             memset(in1, 0, sizeof(in1));
20             memset(in2, 0, sizeof(in2));
21             memset(out1, 0, sizeof(out1));
22             memset(out2, 0, sizeof(out2));
23         }
24     };
25
26     struct Match // Store match
27     {
28         vector<prii> s; // match
29         int id;         // graphDB id
30         Match() {}
31         Match(vector<prii> _s, int _id) : s(_s), id(_id) {}
32     };
33
34 public:
35     VF2() {}
36     void init(const vector<Graph> &db);          // Init the VF2 class
37     int vf2(const Graph &QG, const int &QID);    // Run VF2 on
           QueryGraph & DBGraph (engine)
38
39 private:
40     void GenRevGraph(const Graph &src, Graph &dst); // Generate
           reversed graph
41     bool CheckPrev(const State &s, int a, int b);   // Prev
42     bool CheckSucc(const State &s, int a, int b);   // Succ
43     bool CheckIn(const State &s);                   // In
```

```
44      bool CheckOut(const State &s);                  // Out
45      bool CheckNew(const State &s);                  // New
46      void CalDFSVec(const State &s);                 // Cal all temp vec per
            dfs
47      void CalCheckVec(const State &s, int a, int b); // Cal all temp vec per
            check
48      bool check(const State &s, int a, int b);       // Check feasibility
49      void GenPairs(const State &s);                  // Generate all pairs
            to var allPairs
50      void CheckPairs(const State &s);                // Check allPairs,
            return candiPairs
51      void UpdateState(State &s, int a, int b);       // Update state ns
            with pair(a,b)
52      bool FinalCheck(const State &s);                // Final check for
            answer
53      bool dfs(const State &s);                       // VF2 dfs (recursive)
54      bool query();                                   // Run VF2 on pat &
            g (main procedure)
55
56  public:
57      vector<Graph> DBGraph; // Graph database
58      Graph QueryGraph;      // Graph for query, pattern
59      int QueryID;           // Store the Query ID
60
61      vector<Match> match; // Store the matched graph info
62
63  private:
64      // Temp var
65      vector<prii> tlist;             // Store temp list of match
66      vector<prii> allPairs;          // Generated pairs
67      vector<prii> candiPairs;        // Candidate pairs pass check()
68      bool flagIn, flagOut, flagAll; // Flags
69      vector<int> pred1, pred2;       // Pred of a in pat & b in g
70      vector<int> succ1, succ2;       // Succ of a in pat & b in g
71      vector<int> m1, m2;             // M_1, M_2
72      vector<int> tin1, tin2;         // Tin_1,Tin_2
73      vector<int> tout1, tout2;       // Tout_1,Tout_2
74      vector<int> n1, n2;             // N_1, N_2
75      vector<int> ns1, ns2;           // Point set of pat & g
76      vector<int> t1, t2;             // tin+tout
77
78      Graph pat, g;         // Pattern & DBGraph
79      Graph revpat, revg; // Reversed Pattern & DBGraph
80  };
```

2.2.2 设计

变量 变量命名按照论文[1]的变量名称进行命名。为了使得程序更快，本文选

择用开全局变量的方式代替传递参数的方式，所以VF2类中有大量的临时变量，临时变量就不介绍了，这里仅介绍有意义的主要变量。

- $DBGraph$：图的数据库，使用链式前向星存储
- $QueryGraph$：当前进行query的图模式
- $QueryID$：当前进行query的图模式的编号
- $match$：query的匹配结果（包括匹配的数量，以及点-点映射的结果）
- pat：模式图
- g：目标图
- $revpat$：模式图的反向图
- $revg$：目标图的反向图

方法

- $init$：
- $vf2$：
- $GenRevGraph$：
- $CheckPrev$：
- $CheckSucc$：
- $CheckIn$：
- $CheckOut$：
- $CheckNew$：
- $CalDFSVec$：
- $CalCheckVec$：
- $check$：
- $GenPairs$：
- $CheckPairs$：
- $UpdateState$：

- $FinalCheck$：
- dfs：
- $query$：

2.3 程序主驱动-类：Solver

2.3.1 定义代码

```
1  class Solver
2  {
3  public:
4      void init(); // init
5      void input(); // input
6      void solve(); // solve
7      void output(); // output
8
9  private:
10     void ReadFile(string path, vector<Graph> &vec); // Read file from
           path, write graph data to vec
11     void ReadDB(string path); // Read DB file
12     void ReadQuery(string path); // Read Query file
13
14     void PrintQueryAns(int id, int cnt); // Pring query ans, id = query file
           id, cnt = match num
15
16 private:
17     VF2 vf2; // VF2 main component
18     string dbPath; // DB file path
19     vector<string> queryPath; // Query file path vector
20     vector<string> outputPath; // Output ans file path
21
22     vector<Graph> DBGraph; // Store DB
23     vector<Graph> QueryGraph; // Store Query
24 };
```

2.3.2 设计

Solver类是整个程序的主驱动，包含了读取输入文件，运行VF2，输出运行时间和运行结果。从Data文件夹读取输入数据，将匹配结果输出到Output文件夹中，对每个query文件的匹配结果输出一个文件。并将运行时间输出至time.txt中。

变量

- $vf2$：VF2类的对象，VF2算法
- $dbPath$：图的数据库文件路径
- $queryPath$：query的模式图的文件路径集合
- $outputPath$：query输出的文件路径集合
- $DBGraph$：读入的图的数据库，使用链式前向星存储
- $QueryGraph$：读入的一个query的模式图

方法

- $init$：
- $input$：
- $solve$：
- $output$：
- $ReadFile$：
- $ReadDB$：
- $ReadQuery$：
- $PrintQueryAns$：

§ 3 实验

3.1 测试数据

实验选择graphDB数据集进行运行时间测试，选择graphDB数据集的一个子集进行正确性测试。

graphDB数据集包括大小为10000个图的图数据库文件”mygraphdb.data”，以及6个query图数据文件”Q4.my”，”Q8.my”，”Q12.my”，”Q16.my”，”Q20.my”，”Q24.my”。

3.2 测试环境

使用的计算机硬件和软件配置如表1所示。

表 1 测试环境（计算机配置和编译环境）

项目	详细信息
CPU	Core i7-2630QM (2.0GHz, 4 Cores, 6MB L3 Cache)
内存	16GB DDR3 1600MHz
测试所用磁盘	480GB Sandisk Extreme Pro SSD (Read: 550MB/s)
操作系统	Windows 10 Professional x64 Build 10240
C/C++编译器	Microsoft Visual Studio 2015
C++版本	需支持C++11

该代码已上传至BitBucket的私有仓库，课程结束后会开源。

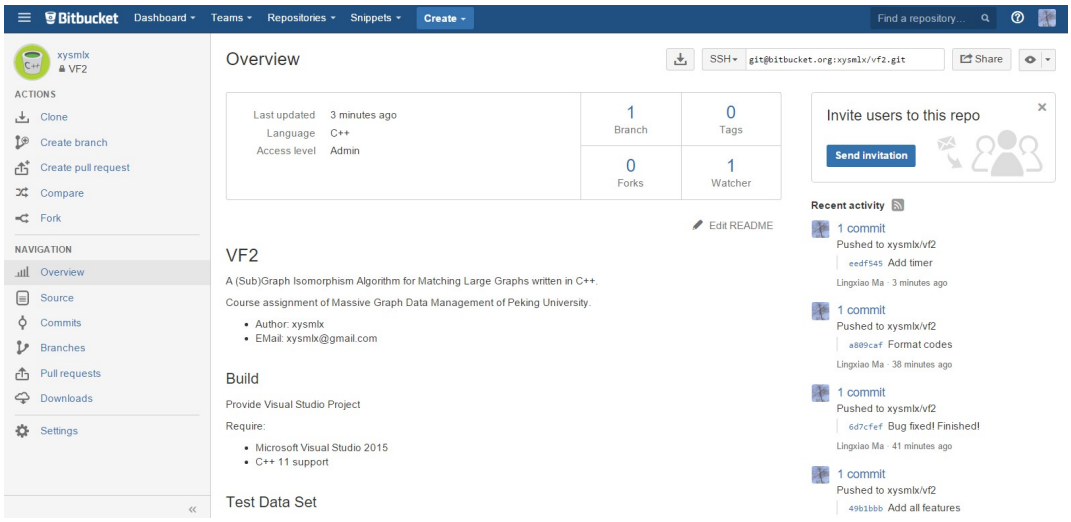


图 1 BitBucket私有仓库

3.3 实验结果

3.3.1 正确性测试

3.3.2 运行速度测试

§ 4 总结

参考文献

[1] Cordella L P, Foggia P, Sansone C, et al. A (sub) graph isomorphism algorithm for matching large graphs[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2004, 26(10): 1367-1372.