Ullmann 算法

2015-11-18

main.c

```
#include <string>
#include "sys/time.h"
#include "ullman.h"
#include "ullman_utils.h"
#include "common.h"

int main(int argc, char *argv[])
{
	ullman::parameters_t parameters;

	//./ullman -database ../data/ -query ../data/
	size_t nargh = ullman::get_parameters(&parameters, argc, argv);

	//[parameters] - [number of default parameters]
	if (nargh != ullman::get_narg() - 1) {
		exit(ULLMAN_ERROR);
	}

	std::string seperator;
	ullman::get_seperator(&parameters, seperator);
	ullman::print_parameters(&parameters);

	timeval t1, t2;
	double elapsed_time = 0.0f;
	gettimeofday(&t1, NULL);

	ullman::Isomorphism isomorphism(parameters.database, parameters.query,
seperator.c_str());
	if (ULLMAN_SUCCESS != isomorphism.execute()) {
		fprintf(stderr, "not successful execution!");
		exit(ULLMAN_ERROR);
	}

	gettimeofday(&t2, NULL);
	elapsed_time = (t2.tv_sec - t1.tv_sec) * 1000.0;
	elapsed_time += (t2.tv_usec - t1.tv_usec) / 1000.0;
	printf("elapsed time %f\n", elapsed_time);

	return 0;
}
```

graph.cpp

```
#include "graph.h"

namespace ullman {
    const std::map<int32_t, std::vector<size_t> > Graph::get_vertex_label_map()
    {
        if (!_m_vertex_label_map.size()) {
            construct_vertex_label_map();
        }

        return _m_vertex_label_map;
    }

    void Graph::construct_vertex_label_map()
    {
        for (size_t i = 0; i < _m_vertice.size(); ++i) {

    _m_vertex_label_map[_m_vertice[i].label].push_back(_m_vertice[i].id);
        }
    }

    const std::map<struct edge_label_list_t, std::vector<struct edge_t *> >&
        Graph::get_edge_label_map()
    {
        if (!_m_edge_label_map.size()) {
            construct_edge_label_map();
        }

        return _m_edge_label_map;
    }

    //Todo
    void Graph::construct_edge_label_map()
    {
        for (size_t i = 0; i < _m_vertice.size(); ++i) {
            for (size_t j = 0; j < _m_vertice[i].edges.size(); ++j) {
                struct edge_t *edge = &_m_vertice[i].edges[j];

                struct edge_label_list_t edge_label_list;
                edge_label_list.from_label = _m_vertice[edge->from].label;
                edge_label_list.edge_label = edge->label;
                edge_label_list.to_label = _m_vertice[edge->to].label;

                _m_edge_label_map[edge_label_list].push_back(edge);
            }
        }
    }
}//namespace ullman

```

Graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

#include <map>
#include <algorithm>
#include "common.h"

namespace ullman {
    struct edge_t {
        size_t from;
        int32_t label;
        size_t to;
        size_t id;
    };

    struct vertex_t {
        size_t id;
        int32_t label;
        std::vector<struct edge_t> edges;
    };
    typedef std::vector<struct vertex_t> Vertice;

    struct edge_label_list_t {
        size_t from_label;
        int32_t edge_label;
        size_t to_label;

        bool operator < (const struct edge_label_list_t& edge_label_list) const
        {
            if (from_label != edge_label_list.from_label) {
                return from_label < edge_label_list.from_label;
            } else {
                if (edge_label != edge_label_list.edge_label) {
                    return edge_label < edge_label_list.edge_label;
                } else {
                    return to_label < edge_label_list.to_label;
                }
            }
        }
    };

    class Graph {
        public:
            explicit Graph() : id(0), _m_nedges(0) {};

            explicit Graph(size_t size) : id(0), _m_nedges(0), _m_vertice(size) {};

            size_t size() const
            {
                return _m_vertice.size();
```

```
1                   }
2
3                   void resize(size_t s)
4                   {
5                       _m_vertice.resize(s);
6                   }
7
8                   void set_id(size_t id)
9                   {
10                      this->id = id;
11                  }
12
13                  size_t get_id() const
14                  {
15                      return id;
16                  }
17
18                  void set_nedges(size_t size)
19                  {
20                      _m_nedges = size;
21                  }
22
23                  size_t get_nedges() const
24                  {
25                      return _m_nedges;
26                  }
27
28                  void set_vertice(const Vertice& vertice)
29                  {
30                      this->_m_vertice = vertice;
31                  }
32
33                  struct vertex_t& get_vertex(size_t idx) {
34                      return _m_vertice[idx];
35                  }
36
37                  const struct vertex_t& get_vertex(size_t idx) const {
38                      return _m_vertice[idx];
39                  }
40
41                  const std::map<struct edge_label_list_t, std::vector<struct edge_t
42  *> >& get_edge_label_map();
43
44                  const std::map<int32_t, std::vector<size_t> >
45  get_vertex_label_map();
46
47                  void clear()
48                  {
49                      id = 0;
50                      _m_vertice.clear();
51                  }
52
```

```
            void sort_vertex_by_degree() {
                std::sort(_m_vertice.begin(), _m_vertice.end(), sort_vertex);
            }

        private:
            struct sort_vertex_t {
                bool operator () (const struct vertex_t& vertex_a,
                        const struct vertex_t& vertex_b)
                {
                    return vertex_a.edges.size() > vertex_b.edges.size();
                }
            } sort_vertex;

            void construct_edge_label_map();

            void construct_vertex_label_map();

        private:
            size_t _m_nedges;
            size_t id;
            Vertice _m_vertice;
            std::map<struct edge_label_list_t, std::vector<struct edge_t *> >
_m_edge_label_map;
            std::map<int32_t, std::vector<size_t> > _m_vertex_label_map;
    };
}//namespace ullman

#endif
```

matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#include <vector>
#include <map>
#include <string>
#include <sstream>

namespace ullman {

    //efficient matrix
    template<typename T>
        class Matrix {
            public:
                Matrix<T>(size_t nrows, size_t ncolumn) :
                    _m_nrows(nrows), _m_ncolumns(ncolumn), _m_change(true),
                    _m_value(nrows, std::vector<T>(ncolumn, 0)) {};

                Matrix<T>() : _m_nrows(0), _m_ncolumns(0), _m_change(true) {};
```

```
1
2                   inline Matrix<T> operator * (const Matrix<T>& other) const;
3
4                   inline Matrix<T> transposition() const;
5
6                   inline void set(size_t i, size_t j, T v);
7
8                   inline T get(size_t i, size_t j) const;
9
10                  inline void fill(T v);
11
12                  inline void fill(size_t row, T v);
13
14                  void resize(size_t size_i, size_t size_j);
15
16                  inline void clear();
17
18                  inline const char * c_str();
19
20                  //{nrows, ncolumns}
21                  std::pair<size_t, size_t> size() const
22                  {
23                      return std::make_pair(_m_nrows, _m_ncolumns);
24                  }
25
26              private:
27                  size_t _m_ncolumns, _m_nrows;
28                  std::string _m_str;
29                  bool _m_change;
30                  std::vector<std::vector<T> > _m_value;
31          };
32
33      template <typename T>
34          inline Matrix<T> Matrix<T>::transposition() const
35          {
36              Matrix<T> matrix(_m_ncolumns, _m_nrows);
37
38              for (size_t i = 0; i < _m_nrows; ++i) {
39                  for (size_t j = 0; j < _m_ncolumns; ++j) {
40                      matrix.set(j, i, this->get(i, j));
41                  }
42              }
43
44              return matrix;
45          }
46
47      template <typename T>
48          inline Matrix<T> Matrix<T>::operator * (const Matrix<T>& other) const
49          {
50              std::pair<size_t, size_t> other_size = other.size();
51
52              if (other_size.first != _m_ncolumns)
```

```
                return Matrix<T>();

            Matrix<T> matrix(_m_nrows, other_size.second);
            for (size_t i = 0; i < _m_nrows; ++i) {
                for (size_t j = 0; j < other_size.second; ++j) {
                    T tmp = 0;
                    for (size_t k = 0; k < _m_ncolumns; ++k) {
                        tmp += this->get(i, k) * other.get(k, j);
                    }
                    matrix.set(i, j, tmp);
                }
            }

            return matrix;
        }


    template <typename T>
        inline void Matrix<T>::set(size_t i, size_t j, T v)
        {
            if (i < _m_nrows && i >= 0 && j < _m_ncolumns && j >= 0) {
                _m_value[i][j] = v;
                _m_change = true;
            }
        }


    template <typename T>
        inline T Matrix<T>::get(size_t i, size_t j) const
        {
            if (i < _m_nrows && i >= 0 && j < _m_ncolumns && j >= 0) {
                return _m_value[i][j];
            }
            return 0;
        }

    template <typename T>
        inline void Matrix<T>::fill(T v)
        {
            for (size_t i = 0; i < _m_nrows; ++i) {
                _m_value[i].assign(_m_ncolumns, v);
            }
            _m_change = true;
        }

    template <typename T>
        inline void Matrix<T>::fill(size_t nrow, T v)
        {
            if (nrow < _m_nrows) {
                _m_value[nrow].assign(_m_ncolumns, v);
            }
        }
```

```
1
2         template <typename T>
3             void Matrix<T>::resize(size_t size_i, size_t size_j)
4             {
5                 if (_m_value.size() < size_i) {
6                     _m_value.resize(size_i, std::vector<T>(_m_ncolumns, 0));
7                 }
8
9                 _m_nrows = size_i;
10
11                if (_m_value[0].size() < size_j) {
12                    for (size_t i = 0; i < _m_nrows; ++i) {
13                        _m_value[i].resize(size_j, 0);
14                    }
15                }
16
17                _m_ncolumns = size_j;
18
19                _m_change = true;
20            }
21
22        template <typename T>
23            inline void Matrix<T>::clear()
24            {
25                for (size_t i = 0; i < _m_nrows; ++i) {
26                    _m_value[i].clear();
27                }
28                _m_ncolumns = 0;
29                _m_nrows = 0;
30                _m_change = true;
31            }
32
33        template <typename T>
34            inline const char * Matrix<T>::c_str()
35            {
36                if (_m_change) {
37                    std::stringstream ss;
38                    for (size_t i = 0; i < _m_nrows; ++i) {
39                        for (size_t j = 0; j < _m_ncolumns; ++j) {
40                            ss << _m_value[i][j] << " ";
41                        }
42                        ss << "\n";
43                    }
44
45                    _m_str = ss.str();
46                }
47
48                _m_change = false;
49                return _m_str.c_str();
50            }
51    }//namespace ullman
52    #endif
53
```

Seperater.cpp

```
#include "seperator.h"

namespace ullman {
    static const uint32_t MAX_LENGTH = 1024;

    uint32_t Seperator::seperate(const char* file_path, Buffer& stream) {
        char line[MAX_LENGTH];

        FILE *fp = fopen(file_path, "r+");

        if (fp == NULL) {
            fprintf(stderr, "error occurs when reading file %s\n", file_path);
            exit(ULLMAN_ERROR);
        }

        uint32_t ncount = 0;

        while (fgets(line, MAX_LENGTH - 1, fp) != NULL) {
            stream.resize(ncount + 1);
            char *pch = NULL;
            pch = strtok(line, _m_token);
            while (pch != NULL) {
                stream[ncount].push_back(std::string(pch));
                pch = strtok(NULL, _m_token);
            }
            ncount++;
        }

        fclose(fp);

        return ncount;
    };
}//namespace ullman


```

Seperater.h

```
#ifndef SEPERATOR_H
#define SEPERATOR_H

#include "common.h"

namespace ullman {

class Seperator {
    public:
        Seperator(const char* token): _m_token(token) {
        };

        uint32_t seperate(const char *file_path, Buffer& stream);

    private:
        const char* _m_token;
};

}//namespace ullman

#endif //SEPERATOR_H
```

Ullman.h

```
#ifndef ULLMAN_H
#define ULLMAN_H
#include <set>
#include "graph.h"
#include "matrix.h"
#include "seperator.h"
#include "common.h"

namespace ullman {
    class Graph;

    class Database {
        public:
            void push_graph(const Graph& graph)
            {
                graphs.push_back(graph);
            }

            const Graph& get_graph(size_t id) const
            {
                return graphs[id];
            }

            Graph& get_graph(size_t id)
            {
                return graphs[id];
            }

            size_t size() const
            {
                return graphs.size();
            }

            void sort()
            {
                for (size_t i = 0; i < graphs.size(); ++i)
                    graphs[i].sort_vertex_by_degree();
            }

        private:
            std::vector<Graph> graphs;
    };

    class Isomorphism {
        public:
            explicit Isomorphism(const char *database, const char *query, const
char *sep_type) :
                    _m_file_data(database), _m_file_query(query) ,
_m_seperator(sep_type) {};
```

```
1
2                   UllmanReturnCode execute();
3
4           private:
5                   UllmanReturnCode read_input(const Buffer& input, Database&
6   database);
7
8                   UllmanReturnCode query();
9
10                  bool construct_match(Graph& query_graph, Graph& entry_graph);
11
12                  void dfs_search(size_t idx, Matrix<bool> matrix);
13
14                  void build_matrix(Matrix<bool>& matrix, size_t nrows, size_t
15  ncolumns);
16
17                  void build_matrix(Matrix<bool>& matrix, const Graph& graph);
18
19                  bool judge(const Matrix<bool>& matrix);
20
21                  void refine(Matrix<bool>& matrix, size_t start);
22
23                  UllmanReturnCode output();
24
25          private:
26                  const char *_m_file_data;
27                  const char *_m_file_query;
28                  Seperator _m_seperator;
29
30                  Database _m_database;
31                  Database _m_query;
32                  //whether current columns are used
33                  Matrix<bool> _m_columns;
34                  //whether all the possible olumns
35                  std::vector<int32_t> _m_columns_used;
36                  //corresponding matrix b
37                  Matrix<bool> matrix_b;
38                  //corresponding matrix a
39                  Matrix<bool> matrix_a;
40
41                  std::vector<std::vector<size_t> > _m_output;
42
43                  size_t _m_cur_graph_id;
44                  size_t _m_cur_query_id;
45                  bool _m_cur_find;
46      };
47  }//namespace ullman
48
49  #endif
50
```

Ullman_build.cpp

```
#include "ullman.h"

namespace ullman {
    //Todo
    void Isomorphism::build_matrix(Matrix<bool>& matrix, size_t nrows, size_t
ncolumns)
    {
        matrix.clear();
        matrix.resize(nrows, ncolumns);

        for (size_t i = 0; i < _m_columns_used.size(); ++i) {
            if (_m_columns_used[i] != -1)
                matrix.set(_m_columns_used[i], i, 1);
        }
    }

    //Todo
    void Isomorphism::build_matrix(Matrix<bool>& matrix, const Graph& graph)
    {
        matrix.clear();
        matrix.resize(graph.size(), graph.size());

        for (size_t i = 0; i < graph.size(); ++i) {
            const struct vertex_t& vertex = graph.get_vertex(i);
            size_t from_id = vertex.id;

            for (size_t j = 0; j < vertex.edges.size(); ++j) {
                size_t to_id = vertex.edges[j].to;
                matrix.set(from_id, to_id, 1);
            }
        }
    }
}//namespace ullman


```

Ullman_init.cpp

```
#include "sys/time.h"
#include "ullman.h"

namespace ullman {
    UllmanReturnCode Isomorphism::execute()
    {
        Buffer ullman_database;
        Buffer ullman_query;

        _m_seperator.seperate(_m_file_data, ullman_database);
        _m_seperator.seperate(_m_file_query, ullman_query);

        if (ULLMAN_SUCCESS != read_input(ullman_database, _m_database)) {
            fprintf(stderr, "read input database error!\n");
            return ULLMAN_ERROR;
        }

        //To-do: sort the vertices

        if (ULLMAN_SUCCESS != read_input(ullman_query, _m_query)) {
            fprintf(stderr, "read input query error!\n");
            return ULLMAN_ERROR;
        }

        timeval t1, t2;
        double elapsed_time = 0.0f;
        gettimeofday(&t1, NULL);

        if (ULLMAN_SUCCESS != query()) {
            fprintf(stderr, "find isomorphism error!\n");
            return ULLMAN_ERROR;
        }

        gettimeofday(&t2, NULL);
        elapsed_time = (t2.tv_sec - t1.tv_sec) * 1000.0;
        elapsed_time += (t2.tv_usec - t1.tv_usec) / 1000.0;
        //printf("elapsed time->execute %f\n", elapsed_time);

        if (ULLMAN_SUCCESS != output()) {
            fprintf(stderr, "output error!\n");
            return ULLMAN_ERROR;
        }

        return ULLMAN_SUCCESS;
    }

    UllmanReturnCode Isomorphism::read_input(const Buffer& buffer, Database&
database)
    {
```

```
        Graph graph;
        Vertice vertice;

        size_t graph_idx = 0;
        size_t edge_id = 0;
        for (size_t i = 0; i < buffer.size(); ++i) {
            if (buffer[i][0] == "t") {
                if (i != 0) {
                    graph.set_nedges(edge_id);
                    graph.set_vertice(vertice);
                    edge_id = 0;
                    database.push_graph(graph);
                    graph.clear();
                    vertice.clear();
                }

                char indicator, seperator;
                size_t idx;
                indicator = buffer[i][0][0];
                seperator = buffer[i][1][0];
                sscanf(buffer[i][2].c_str(), "%zu", &idx);

                if (graph_idx != idx) {
                    fprintf(stderr, "reading buffer warning! %zu %zu\n", graph_idx,
idx);
                    return ULLMAN_WARNING;
                }
                //debug
                //printf("t # %zu\n", idx);

                graph.set_id(idx);
                ++graph_idx;
            } else if (buffer[i][0] == "v") {
                char indicator;
                size_t id;
                int32_t label;
                indicator = buffer[i][0][0];
                sscanf(buffer[i][1].c_str(), "%zu", &id);
                sscanf(buffer[i][2].c_str(), "%d", &label);
                //debug
                //printf("v %zu %d\n", id, label);

                struct vertex_t vertex;
                vertex.id = id;
                vertex.label = label;

                vertice.push_back(vertex);
            } else if (buffer[i][0] == "e") {
                char indicator;
                size_t from, to;
                int32_t label;
                indicator = buffer[i][0][0];
```

```
                    sscanf(buffer[i][1].c_str(), "%zu", &from);
                    sscanf(buffer[i][2].c_str(), "%zu", &to);
                    sscanf(buffer[i][3].c_str(), "%d", &label);
                    //debug
                    //printf("e %zu %zu %d\n", from, to, label);

                    struct edge_t edge;
                    edge.from = from;
                    edge.to = to;
                    edge.label = label;
                    edge.id = edge_id;
                    ++edge_id;

                    //first edge
                    vertice[from].edges.push_back(edge);

                    //second edge
                    edge.from = to;
                    edge.to = from;
                    vertice[to].edges.push_back(edge);
                } else {
                    fprintf(stderr, "reading buffer warning!\n");
                }
            }

            graph.set_vertice(vertice);
            database.push_graph(graph);

            return ULLMAN_SUCCESS;
        }

        UllmanReturnCode Isomorphism::output()
        {
            size_t sum = 0;
            for (size_t i = 0; i < _m_output.size(); ++i) {
                printf("t # %zu : %zu\n", i, _m_output[i].size());
                sum += _m_output[i].size();
                for (size_t j = 0; j < _m_output[i].size(); ++j) {
                    printf("%zu ", _m_output[i][j]);
                }
                printf("\n\n");
            }
            printf("\nsum: %zu\n", sum);

            return ULLMAN_SUCCESS;
        }

    }//namespace ullman


```

Ullman_query.cpp

```
#include "ullman.h"

namespace ullman {
    UllmanReturnCode Isomorphism::query()
    {
        _m_output.resize(_m_query.size());

        for (size_t i = 0; i < _m_query.size(); ++i) {
            Graph& query_graph = _m_query.get_graph(i);
            _m_cur_query_id = query_graph.get_id();
            build_matrix(matrix_a, query_graph);

            for (size_t j = 0; j < _m_database.size(); ++j) {
                Graph& entry_graph = _m_database.get_graph(j);
                _m_cur_graph_id = entry_graph.get_id();
#ifdef DEBUG
        printf("query_id %zu, graph_id %zu\n", _m_cur_query_id,
_m_cur_graph_id);
        //printf("query_graph.size() %zu, entry_graph.size() %zu\n",
query_graph.size(), entry_graph.size());
#endif
                _m_columns.resize(query_graph.size(), entry_graph.size());
                _m_columns_used.resize(entry_graph.size(), -1);

                build_matrix(matrix_b, entry_graph);

                if (!construct_match(query_graph, entry_graph)) {
                    _m_columns.clear();
                    continue;
                }

#ifdef DEBUG
//          printf("%s\n", _m_columns.c_str());
    //    getchar();
    //    getchar();
#endif
                _m_cur_find = false;
                for (size_t k = 0; k < _m_columns.size().second; ++k) {
                    if (!_m_columns.get(0, k))
                        continue;

                    Matrix<bool> matrix = _m_columns;
                    matrix.fill(0, 0);
                    matrix.set(0, k, 1);

                    _m_columns_used[k] = 0;

                    dfs_search(1, matrix);

                    _m_columns_used[k] = -1;
                }
```

```
1
2                       _m_columns_used.clear();
3                       _m_columns.clear();
4                   }
5               }
6
7               return ULLMAN_SUCCESS;
8           }
9
10          bool Isomorphism::construct_match(Graph& query_graph, Graph&
11      entry_graph)
12          {
13              //prune : the size
14              if (query_graph.size() > entry_graph.size())
15                  return false;
16
17              const std::map<struct edge_label_list_t, std::vector<struct edge_t *> >&
18      query_list =
19                  query_graph.get_edge_label_map();
20              std::map<struct edge_label_list_t, std::vector<struct edge_t
21      *> >::const_iterator query_list_it =
22                  query_list.begin();
23
24              const std::map<struct edge_label_list_t, std::vector<struct edge_t *> >&
25      entry_list =
26                  entry_graph.get_edge_label_map();
27              std::map<struct edge_label_list_t, std::vector<struct edge_t
28      *> >::const_iterator entry_list_it =
29                  entry_list.begin();
30
31              while (query_list_it != query_list.end() && entry_list_it != entry_list.end())
32      {
33                  if (query_list_it->first < entry_list_it->first) {
34                      ++query_list_it;
35                  } else if (entry_list_it->first < query_list_it->first) {
36                      ++entry_list_it;
37                  } else {
38                      for (size_t i = 0; i < (query_list_it->second).size(); ++i) {
39                          struct edge_t *query_edge = (query_list_it->second)[i];
40                          size_t query_from = query_edge->from;
41                          size_t query_to = query_edge->to;
42
43                          for (size_t j = 0; j < (entry_list_it->second).size(); ++j) {
44                              struct edge_t *entry_edge = (entry_list_it->second)[j];
45                              size_t entry_from = entry_edge->from;
46                              size_t entry_to = entry_edge->to;
47
48                              _m_columns.set(query_from, entry_from, 1);
49                              _m_columns.set(query_to, entry_to, 1);
50                          }
51
52                      }
```

```
1
2                       ++entry_list_it;
3                       ++query_list_it;
4                   }
5               }
6
7               //prune : degree
8               for (size_t i = 0; i < _m_columns.size().first; ++i) {
9                   for (size_t j = 0; j < _m_columns.size().second; ++j) {
10                      if (_m_columns.get(i, j) == 1) {
11                          size_t degree_from = query_graph.get_vertex(i).edges.size();
12                          size_t degree_to = entry_graph.get_vertex(j).edges.size();
13                          if (degree_to < degree_from)
14                              _m_columns.set(i, j, 0);
15                      }
16                  }
17              }
18              //prune : refine
19              refine(_m_columns, 0);
20
21  #ifdef DEBUG
22              //printf("after refine\n%s\n", _m_columns.c_str());
23  #endif
24
25              //prune : the mapping of zeros
26              for (size_t i = 0; i < _m_columns.size().first; ++i) {
27                  bool find = false;
28                  for (size_t j = 0; j < _m_columns.size().second; ++j) {
29                      if (_m_columns.get(i, j) == 1) {
30                          find = true;
31                          break;
32                      }
33                  }
34                  if (!find) {
35                      return false;
36                  }
37              }
38              return true;
39          }
40
41          void Isomorphism::dfs_search(size_t idx, Matrix<bool> matrix)
42          {
43              //prune 3: if only one of the mapping has found
44              if (_m_cur_find)
45                  return;
46  #ifdef DEBUG
47  //              printf("idx %zu, size %zu\n", idx, _m_columns.size().first);
48  //              printf("query_id %zu, graph_id %zu\n", _m_cur_query_id,
49  _m_cur_graph_id);
50              //printf("%s\n", matrix.c_str());
51  #endif
52              if (idx == matrix.size().first) {
```

```
                Matrix<bool> matrix_c = matrix * ((matrix *
matrix_b).transposition());

#ifdef DEBUG
//          printf("idx %zu, size %zu\n", idx, _m_columns.size().first);
//          printf("query_id %zu, graph_id %zu\n", _m_cur_query_id,
_m_cur_graph_id);
      //    printf("m\n%s\n", _m_columns.c_str());
      //    printf("b\n%s\n", matrix_b.c_str());
      //    printf("m * b\n%s\n", (_m_columns * matrix_b).c_str());
      //    printf("m * b.trans\n%s\n", ((_m_columns *
matrix_b).transposition()).c_str());
      //    printf("m * m * b.trans\n%s\n", (_m_columns * ((_m_columns *
matrix_b).transposition()))).c_str());
//          printf("c\n%s\n", matrix_c.c_str());
//          printf("a\n%s\n", matrix_a.c_str());
#endif

                if (judge(matrix_c)) {
                    _m_cur_find = true;
                    _m_output[_m_cur_query_id].push_back(_m_cur_graph_id);
                }

                return ;
            }

            refine(matrix, idx);
#ifdef DEBUG
            //printf("after refine\n%s\n", matrix.c_str());
#endif

            for (size_t i = 0; i < matrix.size().second; ++i) {
                if (_m_columns_used[i] != -1 || !matrix.get(idx, i))
                    continue;

                Matrix<bool> next_matrix = matrix;
                next_matrix.fill(idx, 0);
                next_matrix.set(idx, i, 1);

                _m_columns_used[i] = idx;

                dfs_search(idx + 1, next_matrix);

                _m_columns_used[i] = -1;
            }
        }

        bool Isomorphism::judge(const Matrix<bool>& matrix_c)
        {
            for (size_t i = 0; i < matrix_a.size().first; ++i) {
                for (size_t j = 0; j < matrix_a.size().second; ++j) {
                    if (matrix_a.get(i, j) == 0)
```

```
                    continue;

                if (matrix_c.get(i, j) == 0)
                    return false;
            }
        }

        return true;
    }

    void Isomorphism::refine(Matrix<bool>& matrix, size_t start)
    {
        const Graph& query_graph = _m_query.get_graph(_m_cur_query_id);

        //Todo: optimize refine
        while (true) {
            bool change = false;
            for (size_t i = start; i < matrix.size().first; ++i) {
                for (size_t j = 0; j < matrix.size().second; ++j) {
                    if (matrix.get(i, j) == 0)
                        continue;

                    const struct vertex_t& vertex = query_graph.get_vertex(i);
                    bool find = true;
                    for (size_t k = 0; k < vertex.edges.size(); ++k) {
                        size_t x = vertex.edges[k].to;

                        bool non_zero = false;
                        for (size_t y = 0; y < matrix.size().second; ++y) {
                            if (matrix.get(x, y) && matrix_b.get(y, j)) {
                                non_zero = true;
                                break;
                            }
                        }
                        if (!non_zero) {
                            find = false;
                            break;
                        }
                    }
                    if (!find) {
                        change = true;
                        matrix.set(i, j, 0);
                    }
                }
            }
            if (!change)
                break;
        }
    }

}//namespace ullman
```

Ullman_utils.cpp

```
#include "ullman_utils.h"

namespace ullman {
    size_t get_parameters(struct parameters_t *p_parameters, int argc, char
*argv[])
    {
        char *database = NULL;
        char *query = NULL;
        int sep_type = 0;

        size_t n_argh = 0;
        for (size_t i_argv = 1; i_argv < argc - 1; i_argv += 2)
        {
            for (size_t i_argh = 0; i_argh < N_ARG; i_argh++)
            {
                if (strcmp(argv[i_argv], ARGH[i_argh]) != 0)
                {
                    continue;
                }
                switch (i_argh)
                {
                    case 0: database = argv[i_argv + 1];
                            n_argh++;
                            break;
                    case 1: query = argv[i_argv + 1];
                            n_argh++;
                            break;
                    case 2: sep_type = atoi(argv[i_argv + 1]);
                            if (sep_type < 0 || sep_type >= SEP_TYPE_NCOUNT) {
                                usage();
                                exit(-1);
                            }
                            break;
                    default:
                            usage();
                            exit(-1);
                }
                break;
            }
        }

        p_parameters->database = database;
        p_parameters->query = query;
        p_parameters->sep_type = sep_type;
        return n_argh;
    }

    void get_seperator(const struct parameters_t *p_parameters,
            std::string& seperator)
    {
        if (p_parameters->sep_type == 0) {
```

```
                //To-do: seperator factory
                seperator = " ";
            } else {
                //do nothing
            }
        }

        void print_parameters(const struct parameters_t *p_parameters)
        {
            printf("-database: %s\n", p_parameters->database);
            printf("-query: %s\n", p_parameters->query);
            printf("-sep: %d\n", p_parameters->sep_type);
        }

        size_t get_narg()
        {
            return N_ARG;
        }

        void usage()
        {
            printf("usage!\n");
        }
    }//namespace ullman


```

Ullman_utils.h

```
#ifndef ULLMAN_UTILS_H
#define ULLMAN_UTILS_H

#include "common.h"

namespace ullman {
    const static size_t N_ARG = 3;
    const static char *ARGH[N_ARG] = { "-database", "-query", "-sep"};

    struct parameters_t {
        char *database;
        char *query;
        int sep_type;
    };

    enum SEP_TYPE {
        DEFAULT,
        SEP_TYPE_NCOUNT
    };

    size_t get_parameters(struct parameters_t *p_parameters, int argc, char
*argv[]);

    void print_parameters(const struct parameters_t *p_parameters);

    void get_seperator(const struct parameters_t *p_parameters, std::string&
seperator);

    size_t get_narg();

    void usage();

}//namespace ullman

#endif
```

//此程序共 1100 行左右.