目录

Bstr

目前还未写　在 Btree.h 中也有该类

Bstr

```
class Bstr{
private:
    char* str;
    int len;
public:
    Bstr(const char* _str, const int _len);
    bool operator > (const Bstr& _b_str);
    bool operator < (const Bstr& _b_str);
    bool operator == (const Bstr& _b_str);
    bool read(FILE* _fp);
    int write(FILE* _fp);
};
```

Database

Database

six tuples: <sub pre obj sid pid oid>

Binary file 保存签名结果

Database::Database(std::string _name)

构造函数

赋值 name signature_binary_file six_tuples_file kvstore

vstree emcode_mode sub_bum pre_num
literal_num fp_debug
Database::~Database()

析构函数 关闭文件

```
bool Database::load()
    (this->kvstore)->open();
    (this->vstree)->loadTree();
bool Database::unload()
    delete this->vstree;
```

bool Database::query(const string _query, ResultSet& _result_set)

在 result_set 中查询，并返回各步骤用时

步骤包括：parse endcode retrieve join final

关键函数：

_parser.sparqlParser(_query, _sparql_q);//转换为 sparql 的查询

_sparql_q.encodeQuery(this->kvstore)//编码为基础块查询

(this->vstree)->retrieve(_sparql_q);//检索可能的结果

this->join(_sparql_q);//将检索到的结果加入到候选列表中~ //

this->getFinalResult(_sparql_q, _result_set);//将结果写入到 result_set 中

bool Database::join(SPARQLquery& _sparql_query)

对于 union 中的每一块 先按规则过滤 this filter_before_join

再加入基本块中 bool Database::join_basic

bool Database::insert(const Triple& _triple)
bool Database::remove(const Triple& _triple)
bool Database::build(const string& _rdf_file)

string Database::getSixTuplesFile()
string Database::getSignatureBFile()
string Database::getStorePath()
void Database::buildSparqlSignature(SPARQLquery & _sparql_q) //编码查询图的相关签名数据

```
    _basic_q->encodeBasicQuery(this->kvstore, _sparql_q.getQueryVar());
```

bool Database::calculateEntityBitSet(int _sub_id, EntityBitSet & _bitset)

//计算对应实体的签名集合

```
    this->encodeTriple2EntityBitSet(_bitset, &_triple);
```

bool Database::encodeTriple2EntityBitSet(EntityBitSet& _bitset, const Triple* _p_triple)

//编码三元组到签名单元中

```
/* check whether the relative 3-tuples exist
 * usually, through sp2olist */
```

bool Database::exist_triple(int _sub_id, int _pre_id, int _obj_id)

```
/*
 * _rdf_file denotes the path of the RDF file, where stores the rdf data
 * there are many step will be finished in this function:
 * 1. assign tuples of RDF data with id, and store the map into KVstore
 * 2. build signature of each entity
 *
 * multi-thread implementation may save lots of time
 */
```

bool Database::encodeRDF(const string _rdf_file)

```
{
    Database::log("In encodeRDF");
    int ** _p_id_tuples = NULL;
    int _id_tuples_max = 0;

    /* map sub2id and pre2id, storing in kvstore */
    this->sub2id_pre2id(_rdf_file, _p_id_tuples, _id_tuples_max);

    /* map literal2id, and encode RDF data into signature in the meantime */
    this->literal2id_RDFintoSignature(_rdf_file, _p_id_tuples, _id_tuples_max);

/* map subid 2 objid_list  &subIDpreID 2 objid_list &subID 2 <preIDobjID>_list */
```

```
    this->s2o_sp2o_s2po(_p_id_tuples, _id_tuples_max);

/* map objid 2 subid_list &objIDpreID2subid_list&*objID 2 <preIDsubID>_list */
    this->o2s_op2s_o2ps(_p_id_tuples, _id_tuples_max);

    Database::log("finish encodeRDF");

    return true;
*/
```

Example

KVstore

Bstree

```
class Btree{
public:
    Btree(const string& _store_path, const string& _file_name, const char* _mode);
    ~Btree();
    bool insert(const bstr& _key, const bstr& _val);
    bool insert(const char* _key, int _klen, const char* _val, int _vlen);
    bool search(const char* _key, int _klen, char*& _val, int & _vlen)const;
    bool search(const char* _key, int _klen);
    const bstr* getValueTransfer();
    bool remove(const bstr& _key);
    bool remove(const char* _key, int _klen);
    bool close();
    bool flush();
    bool release();
private:
    string getBtreeFilePath(){
        return storePath+"/"+fileName;
    }
    bool deleteifExistPath(const string& _path);
    bool openRead(const string& _btree_path);
    bool openWrite(const string& _btree_path);
```

```
    bool openReadWrite(const string& _btree_path);
private:
    BPlusTree* btree;
    string storePath;
    string fileName;
    string mode;
    bstr* value_transfer;
    int value_transfer_max_len;
};
```

Cbtreefunc

多个类 比较混乱

KeyType
mBlockLink
mValue
mQueue
mNode
mitnldata
mItnlNode
mleafdata
mLeafNode
BPlusTree

Kvstore

计算出度 入度等 以及对列表的增删改查等

12 个 B 树存储的列表

Entity2id
id2entity
Predicate2id
Id2predicate
Literal2id
Id2literal
subID2objIDlist
objid2subidlist
subIDpreID2objIDlist
objidpreid2subidlist
subID2preIDobjIDlist
objid2preidsubidlist

包括对这些文件的打开、读取、设置、关闭、删除、清理缓存等

Main

Build_signature

Encode_test

Gload

Gquery

Gstore

Join_test

Testparser

Vstree_test

Parser

Dbparser

解析用　关键

只有两个属性 line_buf 和 buf_line　用于读文件

函数

ParseNode
SpiralParser
rdfParser

```
class DBparser{
private:
    int parseNode(pANTLR3_BASE_TREE node, SPARQLquery& query,ParseNodeStruct* par);

public:
    /* how many triples at most will be parsed out when call rdfParser() once
     * when -1, it means parse all triples in the file into the triples vector
     * */
    static const int TRIPLE_NUM_PER_GROUP = 10*1000*1000;

    DBparser();

    /* input sparql query string and parse query into SPARQLquery
     * the returned string is set for log when error happen */
    string sparqlParser(const string& _sparql, SPARQLquery& _sparql_query);

    /* file stream _fin points to rdfFile
     * that was opened previously in Database::encodeRDF
     * rdfParser() will be called many times until all triples in the rdfFile is parsed
     * and after each call, a group of triples will be parsed into the vector;
     * the returned string is set for log when error happen;
     */
    string rdfParser(ifstream& _fin, Triple* _triple_array, int& _triple_num);

private:
    /*
     * used in readline of FILE, avoiding new memory each time
     */
    static char* line_buf;
    static int buf_len;
};
```

parseNodeStruct

四个属性 以及对这几个属性的获取与设置函数

```
    int block;
```

```
int type;
int depth;
Triple triple;;
```

SparqlLexer

词法分析

SparqlParser

语法分析

Query

BasicQuery

基本查询用 重要 细看代码

```
private:
    vector<string> option_vs;
    vector<Triple> triple_vt;
    map<std::string, int> var_str2id;
    map<std::string, int> var_not_in_select;
    int select_var_num;

    /* var_num is different from   that in SPARQLquery
     * because there are some variable not in select */
    int        graph_var_num;
    string* var_name;
    IDList* candidate_list;
    vector<int*> result_list;
    int*       var_degree;
    char encode_method;

    /* edge_id[var_id][i] : the line id of the i-th edge of the var */
    int**      edge_id;

    /* edge_id[var_id][i] : the neighbor id of the i-th edge of the var */
    int**      edge_nei_id;

    /* edge_pre_id[var_id][i] : the preID of the i-th edge of the var */
```

```
    int**      edge_pre_id;

    /* denote the type of edge, assigned with
     * BasicQuery::IN or BasicQuery::OUT
     * edge_type[var_id][i] */
    char**     edge_type;

    EntityBitSet* var_sig;

    /* edge_sig[sub_id][obj_id] */
    EdgeBitSet**   edge_sig;

    void addInVarNotInSelect();
    void findVarNotInSelect();
    void initial();

public:
    static const char EDGE_IN = 'i';
    static const char EDGE_OUT= 'o';
    static const int MAX_VAR_NUM = 10;
    static const char NOT_JUST_SELECT = 'a';
    static const char SELECT_VAR = 's';

    /* _query is a SPARQL query string */
    BasicQuery(const string _query);
    ~BasicQuery();
    void clear();
    std::string to_str();
//获取各种参数
    int getVarNum();
    std::string getVarName(int _var);
    int getTripleNum();
    const Triple& getTriple(int _i_th_triple);
    int getEdgeID(int _var, int _i_th_edge);
    int getEdgeNeiID(int _var, int _i_th_edge);
    int getEdgePreID(int _var, int _i_th_edge);
    char getEdgeType(int _var, int _i_th_edge);
    int getVarDegree(int _var);
    const EntityBitSet& getVarBitSet(int _i)const;
    IDList& getCandidateList(int _var);
    int getCandidateSize(int _var);
    vector<int*>& getResultList();
    const EntityBitSet& getEntitySignature(int _var);
```

```
    /* check whether the i-th edge of _var is IN edge */
    bool isInEdge(int _var, int _i_th_edge)const;

    /* check whether the i-th edge of _var is OUT edge */
    bool isOutEdge(int _var, int _i_th_edge)const;

private:
    void updateSubSig(int _sub_id, int _pre_id, int _obj_id, std::string
_obj, int _line_id);
    void updateObjSig(int _obj_id, int _pre_id, int _sub_id, std::string
_sub, int _line_id);

public:

    /* encode relative signature data of the query graph */
    void encodeBasicQuery(KVstore* _p_kvstore, const
std::vector<std::string>& _query_var);

    /* add triple */
    void addTriple(const Triple& _triple);

    /* print whole Basic query */
    void print(ostream& _out_stream);
    /*   */

    int getVarID_MinCandidateList();
    int getVarID_MaxCandidateList();

    static int cmp_result(const void* _a, const void* _b);
    bool dupRemoval_invalidRemoval();


    std::string candidate_str();
    std::string result_str();
    std::string triple_str();
};
```

IDList

处理 id_list 数组

```
std::vector<int> id_list;
```

增加了求交集函数

ResultSet

处理结果用的　较简单

```
class ResultSet{
public:
    int select_var_num;
    int ansNum;
    string*   var_name;
    string** answer;

    ResultSet();
    ~ResultSet();
    ResultSet(int _v_num, const string* _v_names);

    /* convert to binary string */
    Bstr* to_bstr();

    /* convert to usual string */
    string to_str();

    /*   */
    void setVar(const std::vector<string> & _var_names);
};
```

SPARQLquery

查询用 重要　细看！ 代码中注释得较好

```
class SPARQLquery{
private:

    vector<BasicQuery*> query_union; //基本查询块的集合

    vector<string>    query_var; //查询的变量

public:
    SPARQLquery(const string& _query);
    SPARQLquery();
    ~SPARQLquery();
    void addQueryVar(const string& _var);
```

```
    void addTriple(const Triple& _triple);
    void addBasicQuery(BasicQuery* _basic_q);
    void addBasicQuery();
    const int getBasicQueryNum();
    BasicQuery& getBasicQuery(int _basic_query_id);
    const int getQueryVarNum();
    const vector<string>& getQueryVar()const;
    const string& getQueryVar(int _id);
    void encodeQuery(KVstore* _p_kv_store);
    vector<BasicQuery*>& getBasicQueryVec();
    void print(ostream& _out_stream);
    std::string triple_str();
    std::string candidate_str();
    std::string result_str();
    std::string to_str();
};
```

Signature

SigEntry

```
    EntitySig sig;
    int entity_id;
```

Signature

Bitset 点集

std::bitset<n> bs　　表示 bs 有 n 位

函数 set（pos） 把 pos 处的二进制位置 1

Reset（pos）　把 pos 处的二进制位置 0

Test（pos）　判断 pos 处是否为 1

Flip（pos） 取反

Hash 方法：BKDR simple RS JS PJW ELF SDB DJB AP

构造函数

EdgeSig：

EdgeSig();
EdgeSig(const EdgeSig* _p_sig);
EdgeSig(const EdgeSig& _sig);
EdgeSig(const EdgeBitSet& _bitset);

重载运算符 |=

EdgeSig& operator|=(const EdgeSig& _sig);

Entity 类似　额外重载了==　！=　= getBitset　encode

Triple

Triple

Triple::Triple(const string _s, const string _p, const string _o)

构造函数

Triple::Triple(string _line)

从字符串中读取出 RDF 三元组　S P O 分别以“/t .”区分

Triple::Triple()

析构函数

Triple::Triple(const Triple& _triple)
Triple& Triple::operator=(const Triple& _triple)
void Triple::setSubject(const string& _s)
void Triple::setPredicate(const string& _p)
void Triple::setObject(const string& _o)
const string& Triple::getSubject()const
const string& Triple::getPredicate()
const string& Triple::getObject()

const string Triple::toString() 将三元组转换为字符串

Util

Util

共用函数

int util::cmp_int(const void* _i1, const void* _i2)

比较

void util::sort(int*& _id_list, int _list_len)

排序

int util::bsearch_int_uporder(int _key,int* _array,int _array_num)

二分查找某特定值的次序号

int util::bsearch_vec_uporder(int _key, const std::vector<int>& _vec)

二分查找某特定变量的次序号

std::string util::result_id_str(std::vector<int*>& _v, int _var_num)

返回[]的字符串

bool util::dir_exist(const std::string _dir)

返回某文件是否存在

bool util::create_dir(const std:: string _dir)

创建文件

long util::get_cur_time()

利用 gettimeofday 返回当前时间

Vstree

这几个类建树用的 整体思路清晰 但有些细节不太明白

EntryBuffer

```
class EntryBuffer
{
private:
```

```
    int capacity;
    int num;
    SigEntry* elems;
public:

    static int DEFAULT_CAPACITY;

    EntryBuffer(int _capacity=200000); // to be determine the default
capacity.
    ~EntryBuffer();
    int getCapacity()const;
    int getNum()const;
    bool isEmpty()const;
    bool isFull()const;
    SigEntry* getElem(int _i);
    bool insert(const SigEntry& _entry);
    int fillElemsFromFile(FILE* _p_file); // fill this buffer with SigEntry
from _p_file, until the buffer is full or meeting EOF.
    void clear();

};
```

LRUCache

```
// before using the cache, you must loadCache or createCache.
class LRUCache
{
public:

   static int DEFAULT_CAPACITY;

    LRUCache(int _capacity=-1);
    ~LRUCache();
    /* load cache's elements from an exist data file. */
    bool loadCache(std::string _filePath="./tree_file");
    /* create a new empty data file, the original one will be overwrite.
*/
    bool createCache(std::string _filePath="./tree_file");
    /* get the value(node's pointer) by key(node's file line). */
    VNode* get(int _key);
    /* set the key(node's file line) and value(node's pointer). if the key
exists now, the value of this key will be overwritten. */
    bool set(int _key, VNode * _value);
    /* update the _key's mapping _value. if the key do not exist, this
```

```
operation will fail and return false. */
    bool update(int _key, VNode* _value);
    /* write out all the elements to hard disk. */
    bool flush();
    int getCapacity();
    int getRestAmount();
    void showAmount();
    bool isFull();
private:
    int capacity;
    int size;
    int* next;
    int* prev;
    int* keys;
    VNode** values;
    std::map<int,int> key2pos; // mapping from key to pos.
    std::string dataFilePath;
    static const int DEFAULT_NUM = 2;
    static const int START_INDEX = 0;
    static const int END_INDEX = 1;
    static const int NULL_INDEX = -1;
    static const int EOF_FLAG = -1;
    /* put the new visited one to the tail */
    void refresh(int _pos);
    /* free the memory of the _pos element in cache. */
    void freeElem(int _pos);
    /* set the memory of the _pos element in cache */
    void setElem(int _pos, int _key, VNode* _value);
    /* just write the values[_pos] to the hard disk, the VNode in
memory will not be free. */
    bool writeOut(int _pos, int _fileLine=-1);
    /* read the value from hard disk, and put it to the values[_pos].
     * before use it, you must make sure that the _pos element in
cache is free(unoccupied).*/
    bool readIn(int _pos, int _fileLine);
};
```

VNode

```
class VNode{
public:
    VNode();
    bool isLeaf()const;
    bool isRoot()const;
```

```
    bool isFull()const;
    void setAsLeaf(bool _isLeaf);
    void setAsRoot(bool _isRoot);
    int getChildNum()const;
    int getFileLine()const;
    int getFatherFileLine()const;
    int getChildFileLine(int _i)const;
    void setChildNum(int _num);
     void setFileLine(int _line);
     void setFatherFileLine(int _line);
     void setChildFileLine(int _i, int _line);
    const SigEntry& getEntry()const;
    const SigEntry& getChildEntry(int _i)const;
     void setEntry(const SigEntry _entry);
     void setChildEntry(int _i, const SigEntry _entry);
    VNode* getFather(LRUCache& _nodeBuffer)const;
// get the father node's pointer.
    VNode* getChild(int _i, LRUCache& _nodeBuffer)const;
// get the _i-th child node's pointer.
    /* add one child node to this node. when splitting this node, can
add a new child to it. */
    bool addChildNode(VNode* _p_child_node, bool _is_splitting =
false);
    /* add one child entry to this node. when splitting this node, can
add a new entry to it. */
    bool addChildEntry(const SigEntry _entry, bool _is_splitting =
false);
    bool removeChild(int _i);
    int getIndexInFatherNode(LRUCache& _nodeBuffer);
    void refreshSignature(); // just refresh itself signature.
    void refreshAncestorSignature(LRUCache& _nodeBuffer); //
refresh self and its ancestor's signature.
    /* used by internal Node */
    bool retrieveChild(std::vector<VNode*>& _child_vec, const
EntitySig _filter_sig, LRUCache& _nodeBuffer);
    /* only used by leaf Node */
    bool retrieveEntry(std::vector<SigEntry>& _entry_vec, const
EntitySig _filter_sig, LRUCache& _nodeBuffer);
    std::string to_str();
private:
     bool is_leaf;
     bool is_root;
     int child_num;
     int self_file_line;
```

```
    int father_file_line;
    SigEntry entry;
    SigEntry child_entries[VNode::MAX_CHILD_NUM];
    int child_file_lines[VNode::MAX_CHILD_NUM];
};
```

VSTree

Remove 中未处理树为空的情况

```
class VSTree{
    friend class VNode;
public:
    VSTree(std::string _store_path);
    ~VSTree();
    int getHeight()const;
    /* build the VSTree from the _entity_signature_file. */
    bool buildTree(std::string _entity_signature_file);
    bool deleteTree();

    /* Incrementally update bitset of _entity_id
     * conduct OR operation on Entry(_entity_id)'s EntityBitSet with _bitset
     * Entry of _entity_id must exists      */
    bool updateEntry(int _entity_id, const EntityBitSet& _bitset);

    /* Replace the Entry(_enitty_id)'s EntityBitSet with _bitset
     * Entry of _entity_id must exists      */
    bool replaceEntry(int _entity_id, const EntityBitSet& _bitset);

    /* insert an new Entry, whose entity doesn't exist before */
    bool insertEntry(const SigEntry& _entry);

    /* remove an existed Entry(_entity_id) from VSTree */
    bool removeEntry(int _entity_id);

    /* save the tree information to tree_info_file_path, and flush the tree nodes in memory to tree_node_file_path. */
    bool saveTree();

    /* load tree from tree_info_file_path and tree_node_file_path files. */
    bool loadTree();

    /* get the tree's root node pointer. */
```

```
    VNode* getRoot();

    /* get the node pointer by its file line. */
    VNode* getNode(int _line);

    /* retrieve candidate result set by the var_sig in the _query. */
    void retrieve(SPARQLquery& _query);

private:
    int root_file_line;
    int node_num;
    int entry_num;
    int height;
    LRUCache* node_buffer;
    EntryBuffer* entry_buffer;
    map<int, int> entityID2FileLineMap;
// record the mapping from entityID to their node's file line.
    static std::string tree_file_foler_path;
    static std::string tree_node_file_path;
    static std::string tree_info_file_path;

    /* choose the best leaf node to insert the _entry, return the
choosed leaf node's pointer. */
    VNode* chooseNode(VNode* _p_node, const SigEntry& _entry);

    /* split the _p_full_node to two new node when it is full.
      * the parameter _insert_entry and _p_insert_node are the
entry/node
     * need to be insert to the _p_full_node.
     */
    void split(VNode* _p_full_node, const SigEntry& _insert_entry,
VNode* _p_insert_node);

    /* create a new node when one node need splitting. */
    VNode* createNode();

    /* swap two nodes' file line, their related nodes(father and
children nodes) will also be updated. */
    void swapNodeFileLine(VNode* _p_node_a, VNode* _p_node_b);

    /* save VSTree's information to tree_info_file_path, such as
node_num, entry_num, height, etc. */
    bool saveTreeInfo();
```

```
    /* load VSTree's information from tree_info_file_path. */
    bool loadTreeInfo();

    /* traverse the tree_node_file_path file, load the mapping from entity id to file line. */
    bool loadEntityID2FileLineMap();

    /* update the entityID2FileLineMap with the _p_node's child entries, the _p_node should be leaf node. */
    void updateEntityID2FileLineMap(VNode* _p_node);

    /* get the leaf node pointer by the given _entityID */
    VNode* getLeafNodeByEntityID(int _entityID);

    /* retrieve the candidate entity ID which signature can cover the_entity_bit_set, and add them to the  _p_id_list. */
    void retrieveEntity(const EntityBitSet& _entity_bit_set, IDList* _p_id_list);

    std::string to_str();
};
```