1.BoW2项目中在TemplatedVocabulary.h文件中添加两个函数后重新编译BoW2，更新lib文件：

```
// ------------------------loadFromBinaryFile----------------------
----------------
template<class TDescriptor, class F>
bool TemplatedVocabulary<TDescriptor,F>::loadFromBinaryFile(const s
td::string &filename) {
  fstream f;
  f.open(filename.c_str(), ios_base::in|ios::binary);
  unsigned int nb_nodes, size_node;
  f.read((char*)&nb_nodes, sizeof(nb_nodes));
  f.read((char*)&size_node, sizeof(size_node));
  f.read((char*)&m_k, sizeof(m_k));
  f.read((char*)&m_L, sizeof(m_L));
  f.read((char*)&m_scoring, sizeof(m_scoring));
  f.read((char*)&m_weighting, sizeof(m_weighting));
  createScoringObject();

  m_words.clear();
  m_words.reserve(pow((double)m_k, (double)m_L + 1));
  m_nodes.clear();
  m_nodes.resize(nb_nodes+1);
  m_nodes[0].id = 0;
  char buf[41]; int nid = 1;
  while (!f.eof()) {
    f.read(buf, size_node);
    m_nodes[nid].id = nid;
    // FIXME
    const int* ptr=(int*)buf;
    m_nodes[nid].parent = *ptr;
    //m_nodes[nid].parent = *(const int*)buf;
    m_nodes[m_nodes[nid].parent].children.push_back(nid);
    m_nodes[nid].descriptor = cv::Mat(1, F::L, CV_8U);
    memcpy(m_nodes[nid].descriptor.data, buf+4, F::L);
    m_nodes[nid].weight = *(float*)(buf+4+F::L);
    if (buf[8+F::L]) { // is leaf
      int wid = m_words.size();
      m_words.resize(wid+1);
      m_nodes[nid].word_id = wid;
      m_words[wid] = &m_nodes[nid];
    }
    else
      m_nodes[nid].children.reserve(m_k);
    nid+=1;
  }
  f.close();
  return true;
}
```

```
// ------------------------saveToBinaryFile-----------------------
--------------

template<class TDescriptor, class F>
void TemplatedVocabulary<TDescriptor,F>::saveToBinaryFile(const st
d::string &filename) const {
  fstream f;
  f.open(filename.c_str(), ios_base::out|ios::binary);
  unsigned int nb_nodes = m_nodes.size();
  float _weight;
  unsigned int size_node = sizeof(m_nodes[0].parent) + F::L*sizeo
f(char) + sizeof(_weight) + sizeof(bool);
  printf("size_node %d \n", size_node);
  f.write((char*)&nb_nodes, sizeof(nb_nodes));
  f.write((char*)&size_node, sizeof(size_node));
  f.write((char*)&m_k, sizeof(m_k));
  f.write((char*)&m_L, sizeof(m_L));
  f.write((char*)&m_scoring, sizeof(m_scoring));
  f.write((char*)&m_weighting, sizeof(m_weighting));
  for(size_t i=1; i<nb_nodes;i++) {
    const Node& node = m_nodes[i];
    f.write((char*)&node.parent, sizeof(node.parent));
    f.write((char*)node.descriptor.data, F::L);
    _weight = node.weight; f.write((char*)&_weight, sizeof(_weigh
t));
    bool is_leaf = node.isLeaf(); f.write((char*)&is_leaf, sizeof(i
s_leaf)); // i put this one at the end for alignement....
  }
  f.close();
}
```

2.创建新项目，包含BoW2的头文件和库文件,创建main.cpp, 运行以下代码

```
const std::string strVocFile= "ORBvoc.txt";
mpVocabulary = new ORBVocabulary();
bool bVocLoad = false; // chose loading method based on file extens
ion
mpVocabulary->loadFromTextFile(strVocFile);
mpVocabulary->saveToBinaryFile(strVocFile);
```

这样在相同文件夹下面生成ORBvoc.bin.

3.将ORB-SLAM2的依赖库BoW2更新，且在工作路径下添加ORBvoc.bin
将system.cc构造函数中这句话

```
    bVocLoad = mpVocabulary->loadFromTextFile(strVocFile);
```

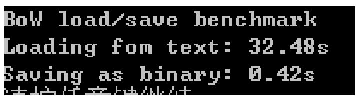
修改为下面这块代码，并且重新编译新的ORB_SLAM2.lib

```
if (has_suffix(strVocFile, ".txt"))
    bVocLoad = mpVocabulary->loadFromTextFile(strVocFile);
else
    bVocLoad = mpVocabulary->loadFromBinaryFile(strVocFile);
```

4.在新工程下运行测试时间

```
const std::string vocabularyPath = "ORBvoc.bin";
const std::string settingPath = "vidoo_mono.yaml";
// Create SLAM system. It initializes all system threads and gets r
eady to process frames.
static ORB_SLAM2::System SLAM(vocabularyPath, settingPath, ORB_SLAM
2::System::MONOCULAR, true);
SLAM.TrackMonocular(rectifyImageLROI, 0);
```

对比读取字典时间，可以看到加载时间是明显降低的



```
BoW load/save benchmark
Loading fom text: 32.48s
Saving as binary: 0.42s
请按任意键继续
```