Lab 1 报告

设计思路

1.对子串进行封装

```
class SubStringInsert
{
public:
  SubStringInsert(uint64_t index,const std::string& data)
  :_index(index),_data(data),_length(data.size()){}
  const uint64_t& getIndex()const{return _index;}
  const std::string& getString()const{return _data;}
  const uint64_t& getLength()const{return _length;}
  void setIndex(uint64_t index){_index = index;}
  void setData(std::string data){_data = data;}
  void updateLength(){_length = _data.size();}
  bool operator< (const SubStringInsert& s)const{return _index<s._index;}
private:
  uint64_t _index;
  std::string _data;
  uint64_t _length;
};
```

子串有三个成员函数：index data 和length，同时给出了读和写的接口。

2.对Reassembler进行划分

```
uint64_t _first_unpopped_index;
uint64_t _first_unassembled_index;
uint64_t _first_unacceptable_index;
```

我们把Reassembler进行一个划分，分别划分成

0 - first_upoped_index

表示已经被上层应用读取的部分，也就是Reader已经读取的部分。

first_upoped_index - first_unassenbled_index

表示Writer已经写入，但是上层应用（Reader）还未读取的部分。
我们保证这部分的数据是连续的、并且没有缝隙的。

first_unassenbled_index - first_unacceptable

这部分表示剩下的容量，这里面包含的index可能已经送达，也可能没有送达。

3 对还没放入缓冲区的数据进行组织

```
std::set<SubStringInsert> _subStringSet;
```

我们使用一个set来维护已经送达的，但是目前还不能插入缓冲区的子串。我们维护它们有以下的性质。

1.它们的index是递增排序的。

这点很好保证，set的有序性可以保证这一点。

2.这些子串是不重叠的。

我们在后面会有函数对此进行操作。当我们插入新的子串的时候，会和set中的子串进行对比，如果出现set中的某个子串与新子串有重叠的部分，我们会把两者合并在一起。

3.所有的子串都在first_upoped_index - first_unacceptable这个范围之内。

如果超过了这个范围，我们会立刻把它切除掉。

4.Index最小的子串的Index也大于first_unassenbled_index。

一旦Index最小的子串的Index等于first_unassenbled_index，我们会立刻把它加入到缓冲区当中。同时从set移除它们。

4 Insert函数

```cpp
void Reassembler::insert( uint64_t first_index, const string& data, bool is_last_substring )
{
    SubStringInsert current_substring(first_index,data);
    _first_unpopped_index = output_.reader().bytes_popped();
    _first_unacceptable_index = _first_unpopped_index + output_.writer().available_capacity() + output_.reader().bytes_buffered();
    //std::cerr<<"now insert is "<<data<<" and unpoped_index unacceptable_index is"<<_first_unpopped_index<<' '<<_first_unacceptable_index<<std::endl;
    cut_off_substring(current_substring,is_last_substring);
    //std::cerr<<"now the substring become " <<current_substring.getString()<<std::endl;
    update_assembly(current_substring);
    push_assembly();
    if(_subStringSet.empty()&&_is_last_input)[[unlikely]]
    {

     output_.writer().close();
    }
}
```

1.准备工作

构造新的子串对象，并且更新_first_unppoed_index,first_unacceptable_index.

2.切除子串

这是为了维护性质3.

```cpp
void Reassembler::cut_off_substring(SubStringInsert& new_substring,bool is_last_substring)
{
  if(new_substring.getIndex()>=_first_unacceptable_index)[[unlikely]]
  {
    new_substring.setData("");
    new_substring.updateLength();
    _is_last_input = _is_last_input||is_last_substring;
    return;
  }
  int64_t bytes_overflow = new_substring.getIndex()+new_substring.getLength()-_first_unacceptable_index;
  if(bytes_overflow>0)[[unlikely]]
  {
    int64_t new_length = new_substring.getLength() - bytes_overflow;
    if(new_length<=0)[[unlikely]]
    {
      new_substring.setData("");
      new_substring.updateLength();
      is_last_substring = false;
    }
    else[[likely]]
    {
      new_substring.setData(new_substring.getString().substr(0,new_length));
      new_substring.updateLength();
      is_last_substring = false;
    }
  }
  if(new_substring.getIndex()<_first_unassembled_index)[[unlikely]]
  {
    int64_t new_length = new_substring.getIndex()+new_substring.getLength() - _first_unassembled_index;
    if(new_length<=0)[[unlikely]]
    {
      new_substring.setData("");
      new_substring.updateLength();
      new_substring.setIndex(_first_unassembled_index);
    }
    else[[likely]]
    {
      new_substring.setData(new_substring.getString().substr(_first_unassembled_index-new_substring.getIndex(),new_length));
      new_substring.updateLength();
      new_substring.setIndex(_first_unassembled_index);
    }
  }
  _is_last_input = _is_last_input||is_last_substring;
}
```

主要就三大点进行讨论：

1.是否整个子串都超出了可容纳的范围

2.是否子串有部分超过了容纳的范围。

如果有超过了范围的部分，那么就切除掉。这个时候，即使is_last_substring = true,我们也需要把它置为false。

3.是否有子串已经有部分/全部在缓冲区当中了

如果有，那么就需要切除。如果全部都在缓冲区了，那么就应该置为空串。

3. 合并子串集合

这是为了维护性质2.

```cpp
void Reassembler::update_assembly(SubStringInsert& new_substring)
{
  int64_t bytes_changed = 0;
  //std::cerr<<"i will update :"<<new_substring.getString()<<std::endl;
  for(auto it = _subStringSet.begin();it!=_subStringSet.end();)
  {

    if(new_substring.getIndex()>=it->getIndex()&&new_substring.getIndex()<it->getIndex()+it->getLength())[[unlikely]]
    {
      if(new_substring.getIndex()+new_substring.getLength()<=it->getIndex()+it->getLength())
      {
        new_substring.setIndex(it->getIndex());
        new_substring.setData(it->getString());
        new_substring.updateLength();
      }
      else
      {
        new_substring.setData(it->getString().substr(0,new_substring.getIndex()-it->getIndex())+new_substring.getString());
        new_substring.setIndex(it->getIndex());
        new_substring.updateLength();
       // std::cerr<<"it is "<<it->getString()<<" and "<<new_substring.getIndex() <<' '<<it->getIndex()<<std::endl;
       // std::cerr<<"substring is"<<new_substring.getString()<<std::endl;
        //std::cerr<<"it->getString().substr ->"<<it->getString().substr(0,new_substring.getIndex()-it->getIndex())<<std::endl;
      }
     // bytes_changed+=new_substring.getLength();
      bytes_changed-=it->getLength();
       it = _subStringSet.erase(it);
     // std::cerr<<"now,substring become: "<<new_substring.getString()<<std::endl;
    }
    else if(new_substring.getIndex()<=it->getIndex()&&it->getIndex()<new_substring.getIndex()+new_substring.getLength())[[unlikely]]
    {
      if(new_substring.getIndex()+new_substring.getLength()<=it->getIndex()+it->getLength())
      {
        uint64_t begin_sub_index = new_substring.getIndex()+new_substring.getLength()-it->getIndex();
        uint64_t end_index = new_substring.getIndex()+new_substring.getLength()-(it->getIndex()+it->getLength());
        new_substring.setData(new_substring.getString()+it->getString().substr(begin_sub_index,end_index));
        new_substring.updateLength();
      }
     // bytes_changed+=new_substring.getLength();
      bytes_changed-=it->getLength();
       it = _subStringSet.erase(it);
      //std::cerr<<"now,substring become: "<<new_substring.getString()<<std::endl;
    }
    else
    {
      ++it;
    }
  }
  bytes_changed+=new_substring.getLength();
  _bytes_pending=static_cast<int64_t>(_bytes_pending)+bytes_changed;
  _subStringSet.insert(new_substring);
  //std::cerr<<"now,the _bytes_pending ->"<<_bytes_pending<<std::endl;
}
```

讨论主要集中在两个大分支上:

1.新子串的index落在老串上.

2.老串的index落在新子串上.

4.更新缓冲区

具体来说,就是看set中有无子串可以放进缓冲区了.

```cpp
void Reassembler::push_assembly()
{
  while(!_subStringSet.empty()&&_subStringSet.begin()->getIndex()==_first_unassembled_index)
  {
    output_.writer().push(_subStringSet.begin()->getString());
    //std::cerr<<"i have write "<<_subStringSet.begin()->getString()<<" now ,bytes_push is"<<output_.writer().bytes_pushed()<<std::endl;
    _first_unassembled_index+=_subStringSet.begin()->getLength();
   // std::cerr<<"now,the _bytes_pending ->"<<_bytes_pending<<std::endl;
    _bytes_pending-=_subStringSet.begin()->getLength();
    _subStringSet.erase(_subStringSet.begin());
  }
}
```

5.检查是否需要关闭输出.

我们通过成员变量_is_last_input维护是否收到了最后一个子串.如果set中是空,并且之前收到过最后一个子串,那么我们就关闭写端.

5 bytes_pending函数

我们用_bytes_pending成员变量来维护.当我们更新set中的串的时候,如果出现了合并串,那么更新bytes_pending;另外,如果进入了缓冲区,我们也要更新.

出现的问题与解决（均已解决）

1.在设计读的接口时，发现最优的方法是getString()类似的函数应该声明为const std::string& getString()const

2.所有的成员变量最好都走初始化列表.

3.末位的子串如果越界被切除之后，islastsubstring也应该被置为false.

4.发现如果新子串如果全部是在缓冲区当中,会出现问题.最终通过加入特判解决.

```cpp
if(new_substring.getIndex()<_first_unassembled_index)[[unlikely]]
{
  int64_t new_length = new_substring.getIndex()+new_substring.getLength() - _first_unassembled_index;
  if(new_length<=0)[[unlikely]]
  {
    new_substring.setData("");
    new_substring.updateLength();
    new_substring.setIndex(_first_unassembled_index);
  }
  else[[likely]]
  {
    new_substring.setData(new_substring.getString().substr(_first_unassembled_index-new_substring.getIndex(),ne
    new_substring.updateLength();
    new_substring.setIndex(_first_unassembled_index);
  }
}
```

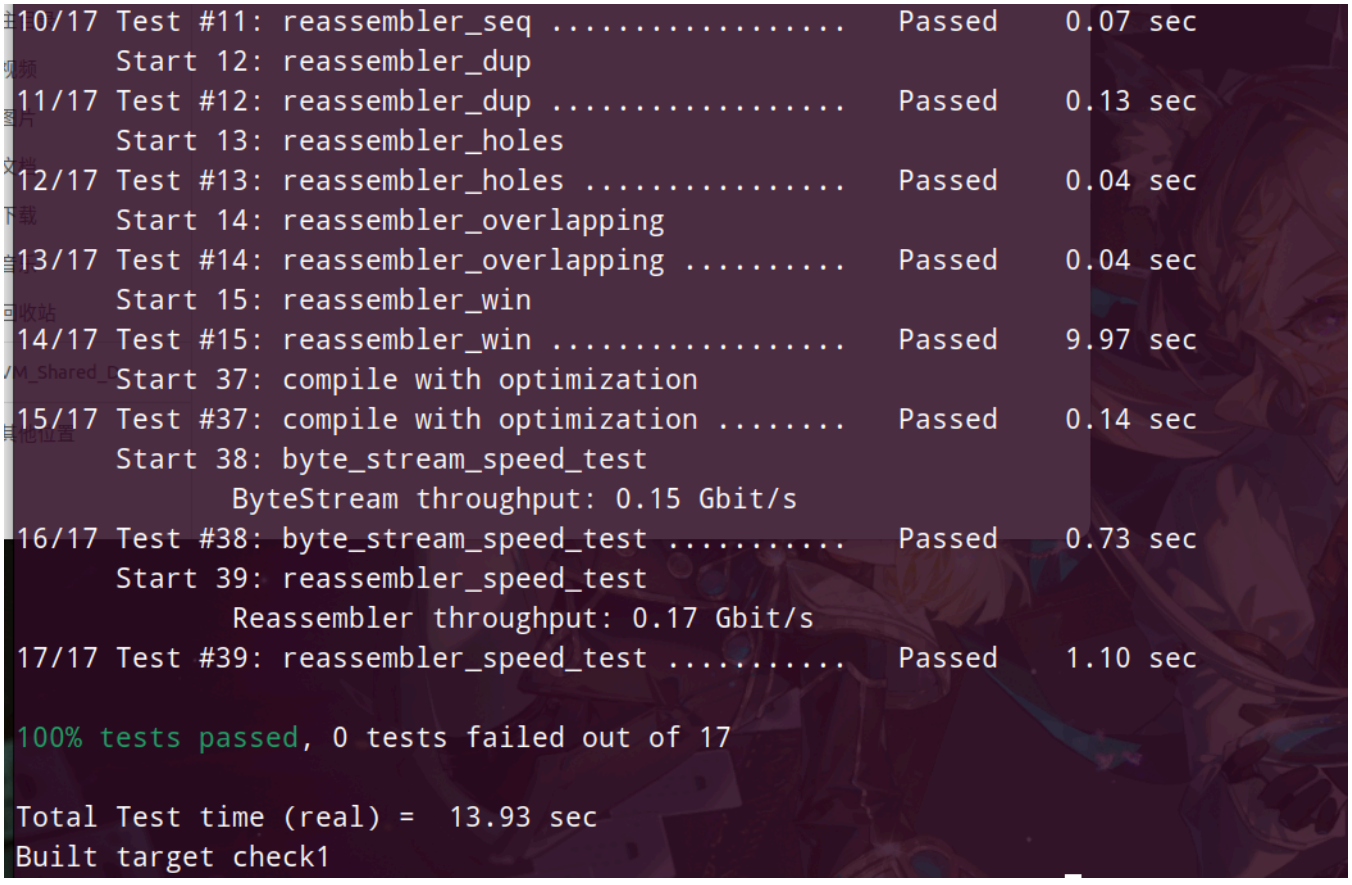
5.发现一开始迭代器的写法会出现bug.

原本我的写法是:

```
for(auto it = _subStringSet.begin();it!=_subStringSet.end();it++)
{
    auto past_it = --it;
    it++;
    if(...)
    {
        it = past_it;
    }
}
```

后来发现这样写有问题.因为一旦it是begin的时候,begin--会出现问题.故改成现在的写法.

测试结果



这个速度应该还可以提升.但是根据lab0的结果来看,可能是lab0写的不够好.如果后面速度不够,我可能会修改一下lab0.