**山东大学计算机科学与技术学院  
《数据结构与算法》课程设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 上机学时：16 | | 日期： 2019.5.27 | |
| 课程设计题目：数据结构与算法的综合应用之后缀树的构造 | | | |
| 软件开发环境：  Windows 10 Professional 19H1  Microsoft Visual Studio 2019  Qt for C++ 5.12 | | | |
| 报告内容：  1.需求描述  1.1 问题描述  后缀树是一种数据结构，一个具有m个字符的字符串S的后缀树T，就是包含1个根节点的有向树，该树恰好带有m+1个叶子(包含空字符)，这些叶子被赋予从0到m的标号。每一个内部节点，除了根节点以外，都至少有两个子节点，而且每条边都用S的个子串来标识。出自同一节点的任意两条边的标识不会以相同的字符开始。  后缀树的关键特征是:对于任何叶子，从根节点到该叶子所经历的边的所有标识串联起来后恰好拼出S的从位置i开始的后缀，即S[i, …, m]  1.2 基本要求  (1)对任意给定的字符串S，建立其后级树  (2)查找一个字符串S是否包含子串T  (3)统计S中出现T的次数:  (4)找出S中最长的重复子串。所谓重复子串是指出现了两次以上的子串  (5)分析以上各个算法的时间复杂性。  (6)查找两个字符串Q和R中最长的共有子串。分析时间复杂性并通过实验  1.3 输入说明  无  1.4 输出说明  无  2.设计  2.1 系统结构设计  整个系统采用图形化界面交互    图形化界面设计由Qt实现。系统的GUI交互与内部处理分离，再建立一个类连接两个类进行交互响应。    2.2 设计思路  后台建立后缀树，前台使用GUI界面与用户交互。点击build按钮可以构造S串的后缀树；点击match按钮可以在S匹配模式串T并且找到T第一次出现的下标；点击count按钮可以统计字符串T在S中出现的次数；点击LRS按钮可以求出字符串S中最长的重复子串；点击LCS按钮可以求出字符串Q和P的最长公共子串。这些按钮分别与对应的函数建立槽信号，单击后便可执行对应的函数，函数体执行完便可以在下方的标签中输出结果，供用户查看。  2.3 数据及数据类(型)定义  (1) SuffixTreeNode类表示后缀树上的节点，记录了字符串的长度，它的后缀链接等信息    (2) SuffixTree类表示后缀树，引入SuffixTreeNode作为节点，构造整个树，其中有一个图结构用于后缀树的打印，ADT包括建树、模式串匹配、统计子串出现次数、求最长重复子串、求LCS等    2.4.算法设计及分析  (1) 构造  在线算法，可以逐个加入字符串中的每个字符，并且在每一步中对应地维护结构。  一开始只包含一个状态s0，编号为0其它状态的编号为1, 2, 3, ...。为了方便，对于状态0指定len=0、link=-1（-1表示虚拟状态）。现在，任务转化为实现给当前字符串添加一个字符的过程。算法流程如下：  1)令last为添加字符c之前，整个字符串对应的状态（一开始设last=0，算法的最后一步更新last）。  2)创建一个新的状态cur，并将len(cur)赋值为len(last)+1  3)现在按以下流程进行(从状态last开始)。如果还没有到字符c的转移，就添加一个到状态的转移，遍历后缀链接。如果在某个点已经存在到字符c的转移，就停下来，并将这个状态标记为p。  4)如果没有找到这样的状态p，就到达了虚拟状态-1，将link(cur)赋值为0并退出。  5)假设现在找到了一个状态，其可以通过字符c转移。将转移到的状态标记为q。  6)现在分类讨论两种状态，要么len(p)+1=len(q)，要么不是。  7)如果len(p)+1=len(q)，只要将link(cur)赋值为q并退出。  8)否则就会有些复杂。需要复制状态q：创建一个新的状态clone，复制q的除了len的值以外的所有信息（后缀链接和转移）。将len(clone)赋值为len(p)+1。  9)复制之后，将后缀链接从cur指向clone，也从q指向clone。  10)最终需要使用后缀链接从状态p返回，因为存在一条通过c到状态q的转移，并在此过程中重定向所有状态到状态clone。  以上三种情况，在完成这个过程之后，将last的值更新为状态cur。  在整个算法结束后便可以得到一颗后缀树link和一个有向无环图next。该算法的复杂度可以证明是O(n)。沿树边和图边遍历均可以得到S的一个子串。  (2) 查询S中是否存在模式串T  对于S在O(n)的时间内建立后缀树，在附带的有向无环图中从s0开始根据T的字符进行转移，如果在某个点无法转一下去，则T不是S的子串。如果能够处理完整个字符串T，那么模式串T在S中出现过。总的时间复杂度为O(|S|+|T|)。  (3) 统计S中子串T出现的次数  后缀树上叶子节点所表示的字符串在且仅在S中出现了一次，而内部节点所表示的字符串出现的为所有儿子节点的和，即f[u] = sum(f[v])(v使u的儿子)。对于整颗后缀树做树上的动态规划后可以得到每个点所表示的字符串在S中出现的次数，此时只需要在搜索得到T所对应的节点便可得出答案。复杂度为O(树上的节点个数)=O(n)  (4) 找出S中出现的最长重复子串  由于在后缀树中仅有叶子节点所表示的字符串在S中出现了1次，因此将非叶子节点按照长度排序贪心选择最长的节点便是答案，排序可以使用桶排序，复杂度为O(非叶子节点个数)=O(n)  (6) 查找两个字符串Q和R中最长的共有子串  对字符串S构造后缀树。  现在处理字符串T，对于每一个前缀，都在S中寻找这个前缀的最长后缀。换句话说，对于每个字符串T中的位置，要找到这个位置结束的S和T的最长公共子串的长度。  使用两个变量，当前状态v和当前长度l。这两个变量描述当前匹配的部分：它的长度和它们对应的状态。  一开始v=s0且l=0，即匹配为空串。现在添加一个字符 并为其重新计算答案：  如果在投有向无环图存在一个从v到字符T[i]的转移，我们只需要转移并让l自增一。  如果不存在这样的转移，我们需要缩短当前匹配的部分，这意味着我们需要在后缀树上沿后缀链接进行转移：v=link(v)  与此同时，需要缩短当前长度。需要将l赋值为len(v)，因为经过这个后缀链接后所到达的状态所对应的最长字符串是一个子串。  如果仍然没有使用这一字符的转移，我们继续重复经过后缀链接并减l，直到找到一个转移或到达虚拟状态-1（这意味着字符T[i]根本没有在S中出现过，所以设置v=l=0）。  问题的答案就是所有l的最大值。时间复杂度为O(|S|+|T|)  3. 测试结果  构造    模式串匹配 S=abcbc, T=bc    模式串匹配 S=abcbc，T=dfdfd    统计字符串出现次数 S=abcbc， T=bcb    统计字符串出现次数 S=abcbc T=cbcb    求最长重复子串 S=abcbc    求最长重复子串 S=abcdefg    求LCS Q=sdcgvhgj, P=hgjvsdcg    4. 分析与探讨  存在传统的O(n)的后缀树构造算法。首先，将每次插入后缀改变为每次向所有后缀树中的后缀末尾添加一个字符，由于叶子节点不会变成内部节点，所以可以将叶子节点的字符串末尾设为已插入的字符串的末尾，使得叶子节点自动更新。其次，每次插入新字符时，需要判断新字符是否需要单独成为叶子，否则沿后缀树向下移动，若向下移动，再插入新字符时，只需将原字符继续向下移动，因为新插入的字符一定是已有后缀的前缀。为了维护这样的一种位置，定义活跃点的概念，活跃点包括三个信息，分别是活跃点的节点，活跃边和活跃半径，分别表示当前走到的节点，走的节点的哪条边和这条边的哪个位置。初始时活跃点为根节点，没有活跃边，活跃半径为0。每次插入一个字符，若可以沿后缀树走，则更新活跃点，否则添加叶子节点和内部节点。当添加叶子时，由于之前插入的字符也需要建立叶子节点，所以需要添加很多叶子，所以需要定义后缀链表的概念，每个阶段，当我们建立新的内部节点并且不是该阶段第一次建立内部节点的时候，我们需要用指针从当前内部节点指向本阶段最近一次建立的内部节点。那么每次插入字符并不能向下移动的时候，就可以沿后缀链表添加内部节点和叶子节点。  两种算法复杂度都是O(n)的，但是传统的构造方法常数较大。经过测试可知，当生成一个长度为5000000的随机字符串，对这个字符串构造后缀树，前者花费的时间为    而后者是    可以看出，传统算法的数量级一致，常数大概在2-3倍左右。  5. 附录：实现源代码  /\*SuffixTreeNode.h\*/  #pragma once  #include <cstring>  #define sz(a) sizeof(a)  class SuffixTreeNode {  public:  //长度 后缀链接 第一次出现的位置 结尾集合  int len, link, firstpos, endpos;  int\* next;  //字符集大小  int alpha\_size;  SuffixTreeNode(const int M = 26) :alpha\_size(M) {  len = link = firstpos = endpos = 0;  }  SuffixTreeNode(const SuffixTreeNode& t) {  len = t.len;  link = t.link;  firstpos = t.firstpos;  endpos = t.endpos;  alpha\_size = t.alpha\_size;  allocatNext();  memcpy(next, t.next, sz(int) \* alpha\_size);  }  ~SuffixTreeNode() { delete[] next; }  void allocatNext() {  next = new int[alpha\_size];  memset(next, 0, sz(int) \* alpha\_size);  }  };  /\*SuffixTree.h\*/  #pragma once  #include <sstream>  #include "SuffixTreeNode.h"  class SuffixTree  {  protected:  SuffixTreeNode\* st;  int last, sz;//当前状态、状态数  int MAX\_SIZE;//最大状态数  const char\* s;  void add(int); //末尾追加字符  void init(); //后缀树初始化  void get\_endpos(); //统计endpos集合  int \_find(const char[]) const;  // 存放树形图  struct edge { int to, next; } \*e;  int cnt, \* head, \*deg;  void add\_edge(int, int);  void build(int, int);//打印树  public:  std::stringstream ssm;  SuffixTree(const char[], const int = 26);  SuffixTree(const SuffixTree&);  ~SuffixTree();  int find(const char[]) const; //模式串匹配  int count(const char[]) const;//模式串统计  int lcs(const char[], char[]) const;//最长公共子串  int LongestRepetitiveSubstring(char[]) const;//最长重复子串  int getSZ() const { return sz; }  };  /\*SuffixTree.cpp\*/  #include "SuffixTree.h"  #include <cstring>  SuffixTree::SuffixTree(const char str[], const int M) {  s = str;  int n = strlen(s);  MAX\_SIZE = n \* 2;  sz = last = 0;  // 数组初始化  st = new SuffixTreeNode[MAX\_SIZE];  for (int i = 0; i < MAX\_SIZE; ++i) st[i].allocatNext();  init();  //在线构造后缀树  for (int i = 0; i < n; ++i) add(s[i] - 'a');  get\_endpos();  }  SuffixTree::SuffixTree(const SuffixTree & t) {  MAX\_SIZE = t.MAX\_SIZE;  last = t.last;  sz = t.sz;  s = t.s;  st = new SuffixTreeNode[MAX\_SIZE];  for (int i = 0; i < sz; ++i) st[i] = SuffixTreeNode(t.st[i]);  }  SuffixTree::~SuffixTree() {  delete[] st;  delete[] e;  delete[] deg;  delete[] head;  }  void SuffixTree::init() {  st[0].link = -1;  st[0].len = 0;  memset(st[0].next, 0, sizeof(int) \* st[0].alpha\_size);  last = 0;  sz = 1;  }  void SuffixTree::add(int c) {  int cur = sz++;  st[cur].len = st[last].len + 1;  st[cur].firstpos = st[cur].len - 1;  int p = last;  //遍历后缀链接树  while (p != -1 && !st[p].next[c]) {  st[p].next[c] = cur;  p = st[p].link;  }  //到达虚拟状态  if (p == -1) {  st[cur].link = 0;  }  else {  int q = st[p].next[c];  if (st[p].len + 1 == st[q].len) {  st[cur].link = q;  }  else {  //克隆状态q  int clone = sz++;  st[clone].len = st[p].len + 1;  memcpy(st[clone].next, st[q].next, sizeof(int) \* st[clone].alpha\_size);  st[clone].link = st[q].link;  st[clone].firstpos = st[q].firstpos;  while (p != -1 && st[p].next[c] == q) {  st[p].next[c] = clone;  p = st[p].link;  }  st[q].link = st[cur].link = clone;  }  }  last = cur;  }  void SuffixTree::get\_endpos() {  //桶排序 按长度从大到小  int\* tmp = new int[sz + 1];  int\* ind = new int[sz + 1];  deg = new int[sz + 1];  head = new int[sz + 1];  e = new edge[MAX\_SIZE];  cnt = 0;  for (int i = 0; i < sz; ++i) head[i] = deg[i] = ind[i] = tmp[i] = 0;  for (int i = 1; i < sz; ++i) tmp[st[i].len]++;  for (int i = 1; i < sz; ++i) tmp[i] += tmp[i - 1];  for (int i = 1; i < sz; ++i) ind[tmp[st[i].len]--] = i;  int cur = 0;  int n = strlen(s);  //叶子节点初始化为1  for (int i = 0; i < n; ++i) {  cur = st[cur].next[s[i] - 'a'];  st[cur].endpos = 1;  }  //树上dp 建立树形图  for (int i = sz - 1; i > 0; --i) {  int cur = ind[i];  if (~st[cur].link) {  st[st[cur].link].endpos += st[cur].endpos;  add\_edge(st[i].link, i);  deg[st[i].link]++;  }  }  ssm << ".\n";  build(0, 0);  delete[] tmp;  delete[] ind;  }  void SuffixTree::add\_edge(int u, int v) {  e[++cnt] = { v, head[u] };  head[u] = cnt;  }  //打印后缀树  void SuffixTree::build(int u, int dep) {  if (st[u].len) {  for (int i = 0; i < dep - 1; ++i) ssm << "| ";  if (--deg[st[u].link])  ssm << "+---";  else  ssm << "+---";  for (int i = 0; i < st[u].len; ++i) ssm << \*(s + st[u].firstpos - st[u].len + 1 + i);  ssm << '\n';  }  for (int i = head[u]; i; i = e[i].next) {  int v = e[i].to;  build(v, dep + 1);  }  }  int SuffixTree::\_find(const char ptr[]) const {  int n = strlen(ptr);  int cur = 0;  //沿状态转移  for (int i = 0; i < n; ++i) {  if (st[cur].next[ptr[i] - 'a']) cur = st[cur].next[ptr[i] - 'a'];  else return -1;  }  return cur;  }  int SuffixTree::find(const char ptr[]) const {  int cur = \_find(ptr);  if (~cur) return st[cur].firstpos - strlen(ptr) + 1;  return -1;  }  int SuffixTree::count(const char ptr[]) const {  int cur = \_find(ptr);  if (cur == -1) return 0;  return st[cur].endpos;  }  int SuffixTree::lcs(const char ptr[], char res[] = nullptr) const {  int n = strlen(ptr);  int cur = 0, l = 0, best = 0, bestpos = 0;  for (int i = 0; i < n; ++i) {  //当前状态不存在ptr[i]的转移  while (cur && !st[cur].next[ptr[i] - 'a']) {  cur = st[cur].link;  l = st[cur].len;  }  if (st[cur].next[ptr[i] - 'a']) {  cur = st[cur].next[ptr[i] - 'a'];  l++;  }  //更新答案  if (best < l) {  best = l;  bestpos = i;  }  }  if (res) memcpy(res, ptr + bestpos - best + 1, best);  return best;  }  int SuffixTree::LongestRepetitiveSubstring(char res[]) const {  //桶排序 按长度从大到小  int\* tmp = new int[sz + 1];  int\* ind = new int[sz + 1];  for (int i = 0; i < sz; ++i) ind[i] = tmp[i] = 0;  for (int i = 1; i < sz; ++i) tmp[st[i].len]++;  for (int i = 1; i < sz; ++i) tmp[i] += tmp[i - 1];  for (int i = 1; i < sz; ++i) ind[tmp[st[i].len]--] = i;  //从大到小遍历节点  for (int i = sz - 1; i; --i) {  int cur = ind[i];  if (st[cur].endpos > 1) {  memcpy(res, s + st[cur].firstpos - st[cur].len + 1, st[cur].len);  return st[cur].len;  }  }  return -1;  }  /\*ui\_MainWindow.h\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* Form generated from reading UI file 'MainWindow.ui'  \*\*  \*\* Created by: Qt User Interface Compiler version 5.12.1  \*\*  \*\* WARNING! All changes made in this file will be lost when recompiling UI file!  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QtCore/QVariant>  #include <QtWidgets/QApplication>  #include <QtWidgets/QLabel>  #include <QtWidgets/QLineEdit>  #include <QtWidgets/QMainWindow>  #include <QtWidgets/QMenuBar>  #include <QtWidgets/QPushButton>  #include <QtWidgets/QWidget>  QT\_BEGIN\_NAMESPACE  class Ui\_MainWindowClass  {  public:  QWidget\* centralWidget;  QPushButton\* build; //构造后缀树  QLineEdit\* S;//原串S  QLineEdit\* T;//模式串T  QPushButton\* match;//模式串匹配  QPushButton\* count;//模式串计数  QPushButton\* lrs;//最长重复出现字串  // Q和P的最长公共子串  QLineEdit\* Q;  QLineEdit\* P;  QPushButton\* lcs;  QLabel\* label;  QLabel\* Str;  QLabel\* Ttr;  QLabel\* Qtr;  QLabel\* Ptr;  QLabel\* tree;  QMenuBar\* menuBar;  QMainWindow\* MainWindow;  void setupUi(QMainWindow\* MainWindowClass)  {  MainWindow = MainWindowClass;  if (MainWindowClass->objectName().isEmpty())  MainWindowClass->setObjectName(QString::fromUtf8("MainWindowClass"));  MainWindowClass->resize(520, 150);  //MainWindowClass->setMaximumSize(520, 150);  //MainWindowClass->setMinimumSize(520, 150);  centralWidget = new QWidget(MainWindowClass);  centralWidget->setObjectName(QString::fromUtf8("centralWidget"));  build = new QPushButton(centralWidget);  build->setObjectName(QString::fromUtf8("build"));  build->setGeometry(QRect(150, 20, 61, 21));  S = new QLineEdit(centralWidget);  S->setObjectName(QString::fromUtf8("S"));  S->setGeometry(QRect(30, 20, 113, 21));  T = new QLineEdit(centralWidget);  T->setObjectName(QString::fromUtf8("T"));  T->setGeometry(QRect(30, 50, 113, 21));  match = new QPushButton(centralWidget);  match->setObjectName(QString::fromUtf8("match"));  match->setGeometry(QRect(150, 50, 61, 21));  count = new QPushButton(centralWidget);  count->setObjectName(QString::fromUtf8("count"));  count->setGeometry(QRect(220, 20, 61, 21));  lrs = new QPushButton(centralWidget);  lrs->setObjectName(QString::fromUtf8("LRS"));  lrs->setGeometry(QRect(220, 50, 61, 21));  Q = new QLineEdit(centralWidget);  Q->setObjectName(QString::fromUtf8("Q"));  Q->setGeometry(QRect(320, 20, 113, 21));  lcs = new QPushButton(centralWidget);  lcs->setObjectName(QString::fromUtf8("lcs"));  lcs->setGeometry(QRect(440, 20, 61, 21));  P = new QLineEdit(centralWidget);  P->setObjectName(QString::fromUtf8("P"));  P->setGeometry(QRect(320, 50, 113, 21));  label = new QLabel(centralWidget);  label->setObjectName(QString::fromUtf8("label"));  label->setGeometry(QRect(30, 90, 450, 50));  tree = new QLabel(centralWidget);  tree->setObjectName(QString::fromUtf8("tree"));  Str = new QLabel(centralWidget);  Str->setObjectName(QString::fromUtf8("Str"));  Str->setGeometry(QRect(10, 22, 10, 15));  Ttr = new QLabel(centralWidget);  Ttr->setObjectName(QString::fromUtf8("Ttr"));  Ttr->setGeometry(QRect(10, 52, 10, 15));  Qtr = new QLabel(centralWidget);  Qtr->setObjectName(QString::fromUtf8("Qtr"));  Qtr->setGeometry(QRect(300, 22, 10, 15));  Ptr = new QLabel(centralWidget);  Ptr->setObjectName(QString::fromUtf8("Ptr"));  Ptr->setGeometry(QRect(300, 52, 10, 15));  MainWindowClass->setCentralWidget(centralWidget);  menuBar = new QMenuBar(MainWindowClass);  menuBar->setObjectName(QString::fromUtf8("menuBar"));  menuBar->setGeometry(QRect(0, 0, 600, 26));  MainWindowClass->setMenuBar(menuBar);  retranslateUi(MainWindowClass);  QMetaObject::connectSlotsByName(MainWindowClass);  } // setupUi  void retranslateUi(QMainWindow\* MainWindowClass)  {  MainWindowClass->setWindowTitle(QApplication::translate("MainWindowClass", "SuffixTree", nullptr));  build->setText(QApplication::translate("MainWindowClass", "build", nullptr));  match->setText(QApplication::translate("MainWindowClass", "match", nullptr));  lcs->setText(QApplication::translate("MainWindowClass", "lcs", nullptr));  count->setText(QApplication::translate("MainWindowClass", "count", nullptr));  lrs->setText(QApplication::translate("MainWindowClass", "LRS", nullptr));  label->setText(QString());  Str->setText(QString("S"));  Ttr->setText(QString("T"));  Qtr->setText(QString("Q"));  Ptr->setText(QString("P"));  } // retranslateUi  // 设置组件可用  void setStatus(bool status) {  T->setEnabled(status);  match->setEnabled(status);  count->setEnabled(status);  lrs->setEnabled(status);  }  };  namespace Ui {  class MainWindowClass : public Ui\_MainWindowClass {};  } // namespace Ui  QT\_END\_NAMESPACE  #endif // MAINWINDOW\_H  /\*MainWindow.h\*/  #pragma once  #include <QtWidgets/QMainWindow>  #include "ui\_MainWindow.h"  #include "SuffixTree.h"  class MainWindow : public QMainWindow  {  Q\_OBJECT  public:  MainWindow(QWidget\* parent = Q\_NULLPTR);  ~MainWindow();  private:  Ui::MainWindowClass ui;  SuffixTree\* s;  std::string str;  private slots:  void build();//建树  void match();//匹配  void lcs();//lcs  void count();//计数  void LongestRepeatString();//最长公共子串  };  /\*MainWindow.cpp\*/  #include "MainWindow.h"  #include <qmessagebox.h>  MainWindow::MainWindow(QWidget\* parent)  : QMainWindow(parent)  {  ui.setupUi(this);  s = nullptr;  // 建立信号槽  connect(ui.build, &QPushButton::clicked, this, &MainWindow::build);  connect(ui.match, &QPushButton::clicked, this, &MainWindow::match);  connect(ui.lcs, &QPushButton::clicked, this, &MainWindow::lcs);  connect(ui.count, &QPushButton::clicked, this, &MainWindow::count);  connect(ui.lrs, &QPushButton::clicked, this, &MainWindow::LongestRepeatString);  ui.setStatus(false);  ui.S->setFocus();  }  MainWindow::~MainWindow() { if (s) delete s; }  void MainWindow::build() {  QString S = ui.S->text();  if (s != nullptr) delete s;  str = std::string(S.toLocal8Bit().data());  for (char c : str) if (!(c >= 'a' && c <= 'z')) {  QMessageBox::information(NULL, "Error", "Only lowercase allowed!", QMessageBox::Yes);  return;  }  // 构造后缀树  s = new SuffixTree(&str[0]);  ui.label->setText(QString(str.c\_str()) + ", building done!");  ui.setStatus(true);  //显示打印的后缀树  ui.tree->setGeometry(QRect(30, 140, 450, 50 + 15\*s->getSZ()));  ui.MainWindow->resize(520, 200 + 15 \* s->getSZ());  ui.tree->setText(s->ssm.str().c\_str());  }  void MainWindow::match() {  if (s == nullptr) return;  // 获取字符串  QString T = ui.T->text();  std::string tmp(T.toLocal8Bit().data());  for (char c : tmp) if (!(c >= 'a' && c <= 'z')) {  QMessageBox::information(NULL, "Error", "Only lowercase allowed!", QMessageBox::Yes);  return;  }  int sz = s->find(tmp.c\_str());  if (~sz) ui.label->setText("Ok, position starts at " + QString(std::to\_string(sz).c\_str()));  else ui.label->setText("Not found");  }  void MainWindow::count() {  if (s == nullptr) return;  QString T = ui.T->text();  std::string tmp(T.toLocal8Bit().data());  for (char c : tmp) if (!(c >= 'a' && c <= 'z')) {  QMessageBox::information(NULL, "Error", "Only lowercase allowed!", QMessageBox::Yes);  return;  }  ui.label->setText(std::to\_string(s->count(tmp.c\_str())).c\_str());  }  void MainWindow::lcs() {  // 获取字符串  QString Q = ui.Q->text();  QString P = ui.P->text();  std::string QQ(Q.toLocal8Bit().data());  std::string PP(P.toLocal8Bit().data());  for (char c : QQ) if (!(c >= 'a' && c <= 'z')) {  QMessageBox::information(NULL, "Error", "Only lowercase allowed!", QMessageBox::Yes);  return;  }  for (char c : PP) if (!(c >= 'a' && c <= 'z')) {  QMessageBox::information(NULL, "Error", "Only lowercase allowed!", QMessageBox::Yes);  return;  }  SuffixTree tmp(QQ.c\_str());  std::string res;  res.resize(QQ.length());  tmp.lcs(PP.c\_str(), &res[0]);  ui.label->setText(res.c\_str());  }  void MainWindow::LongestRepeatString() {  if (s == nullptr) return;  std::string res;  res.resize(str.length());  int sz = s->LongestRepetitiveSubstring(&res[0]);  if (~sz) ui.label->setText(res.c\_str());  else ui.label->setText("No such the longest repatitive string");  }  /\*main.cpp\*/  #include "MainWindow.h"  #include <QtWidgets/QApplication>  int main(int argc, char \*argv[])  {  QApplication a(argc, argv);  MainWindow w;  w.show();  return a.exec();  } | | | |
|  | | | |