**《数据结构与算法》实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | 校园地图导航 | | | | |
| **姓名** | 叶鹏 | **学号** | 20020007095 | **日期** | 2022/5/13 |
| **实验内容** | **校园地图导航**  \* 地图不低于五个点  \* 可以在代码里设置好点名，路径，路径权重  \* 程序运行，输入两个点，输出最短距离及最短路径  \* 加分项:可视化地图QT | | | | |
| **实验目的** | 掌握图论算法 | | | | |
| **实验步骤** | | 1. 建立QT项目，准备好资源，规划窗口部件    1. 新建QT窗口项目，构建项目     Mainwindow为主窗口，dijkstra为算法类   * 1. 在mainwindow中声明窗口所用到的部件      * 1. 制作一张校园地图，上标识路径点，编号，以及路径权重     蓝色线表示路径，白色数字表示路径权重   * 1. 做好窗口布局     左边的地图，右边是输入以及输出，可以选择起点与终点，点击“点击计算”进行计算，  右边会显示最短距离以及最短路径   1. 准备好地图数据    1. 在Dijkstra构造函数中写入地图数据     其中每三个数据u, v, w表示u到v的路径权重为w   1. 编写Dijkstra算法    1. 先将矩阵中的数据读入存入图(graph)中      * 1. 用一个最小堆来构造dijkstra算法中每次寻找距离最短的点的过程      * 1. 构建distance数组，储存每个结点的距离，初始化起始点距离为0，从起点开始优化其能到达的所有点的位置，再从新的位置不断优化，直到最小堆中没有结点为止，最后返回从起点到终点的最短距离      * 1. 考虑到在优化的过程中还要保存路径，我们可以让每一个结点保存他的前驱结点，这样最后我们从终点开始回溯，就能输出一条从起点到终点的路径。  1. 窗口部件逻辑功能实现    1. 其实只有一个Button按钮有功能实现，为其创建点击事件即可     当按钮被点击，首先获取两个下拉框选项的数据，分别作为起点和终点参数传入Dijkstra类中进行计算，获取返回值为distance，同时传入的参数还有pre数组，其储存每一个结点的前驱结点，为输出路径做准备。  因为我们是倒序储存路径的(终点->起点)，因此我们使用栈结构反向输出路径(起点->终点)   * 1. 将计算结果填入相应的框中      1. 运行    1. 南区宿舍 -> 北区宿舍      * 1. 南区宿舍 -> 东区宿舍      * 1. 东区宿舍 -> 图书馆      * 1. 图书馆 -> 北操      1. 源代码：    1. Dijkstra.h   #ifndef DIJKSTRA\_H  #define DIJKSTRA\_H  #include <iostream>  #include <vector>  #include <queue>  #include <QPair>  #include <stack>  using namespace std;  class **Dijkstra**  {  private:  vector<vector<pair<int, int>>> graph;  vector<vector<int>> adjMatrix;  const int inf = INT\_MAX/2;  // enum location {南区宿舍, 图书馆, 信南, 信北, 行远楼, 五子顶, 二五广场, 北操, 体育馆, 北区宿舍, 东操, 东区宿舍};  public:  **Dijkstra**();  int n; // num of nodes  int **dij**(int start, int end, vector<int>& pre);  };  #endif // DIJKSTRA\_H   * 1. Dijkstra.cpp   #include "dijkstra.h"  Dijkstra::**Dijkstra**()  {  this->n = 12;  this->adjMatrix = {  {0, 1, 3},  {0, 4, 3},  {4, 2, 4},  {4, 3, 2},  {4, 6, 2},  {1, 2, 1},  {2, 3, 1},  {3, 6, 4},  {2, 5, 7},  {3, 5, 3},  {6, 8 ,2},  {8, 7, 3},  {7, 9 ,2},  {9, 10, 2},  {8, 10, 6},  {5, 10 ,7},  {5, 11, 8},  {10, 11, 3}  };  }  int Dijkstra::**dij**(int start, int end, vector<int> &pre) {  graph.resize(n);  for(auto mat: adjMatrix){  int from = mat[0], to = mat[1], dist = mat[2];  graph[from].push\_back(pair<int, int>(*dist*, *to*));  graph[to].emplace\_back(pair<int, int>(*dist*, *from*));  }  priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,int>>> q;  vector<int> dis(n, inf);  dis[start] = 0;  q.emplace(0, *start*);  pre[start] = start;  while(!q.empty()){  auto cur = q.top();  q.pop();  int x = cur.second, dist = cur.first;  // if(dist > dis[x])break;  for(auto to: graph[x]){  int y = to.second, d = to.first + dist;  if(d < dis[y]){  dis[y] = d;  q.emplace(*d*, *y*);  pre[y] = x;  }  }  }  return dis[end];  }   * 1. Mainwindow.h   #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QMainWindow>  #include <QComboBox>  #include <QStringList>  #include <QPushButton>  #include <QLabel>  #include <QFont>  #include <QDebug>  #include <QPen>  #include <QMouseEvent>  #include <QPainter>  #include "dijkstra.h"  #include "mylabel.h"  QT\_BEGIN\_NAMESPACE  namespace **Ui** { class **MainWindow**; }  QT\_END\_NAMESPACE  class **MainWindow** : public QMainWindow  {  Q\_OBJECT  public:  **MainWindow**(QWidget \*parent = nullptr);  ~***MainWindow***();  // 部件  myLabel\* picBox;  QLabel\* title, \*sel\_title\_1, \*sel\_title\_2, \*sel\_title\_3, \*sel\_title\_4;  QLabel\* distanceBox, \*pathBox;  QComboBox\* comboBox\_1, \*comboBox\_2;  QStringList locateList = {"南区宿舍", "图书馆", "信南", "信北", "行远楼",  "五子顶", "二五广场", "北操", "体育馆", "北区宿舍",  "东操", "东区宿舍"};  vector<vector<int>> locatePosition;  QPushButton\* calcBtn;  void ***mouseMoveEvent***(QMouseEvent \*event) override{  qDebug() << event->pos();  }  // bool eventFilter(QObject \*watched, QEvent \*event) override; //事件滤波器  // void paintEvent(QPaintEvent \*event) override;  void **Painter**(); //画图  private:  Ui::MainWindow \*ui;  stack<int> pathStk;  private slots:  void **calcBtnClicked**();  };  #endif // MAINWINDOW\_H   * 1. Mainwindow.cpp   #include "mainwindow.h"  #include "ui\_mainwindow.h"  MainWindow::**MainWindow**(QWidget \*parent)  : QMainWindow(parent)  , ui(new Ui::MainWindow)  {  ui->setupUi(this);  // set location position  this->locatePosition = {  {330, 640},  {340, 480},  {290, 430},  {260, 380},  {210, 510},  {370, 310},  {160, 300},  {120, 170},  {120, 250},  {240, 130},  {380, 150},  {550, 140}  };  // set size  this->setFixedSize(1280, 720);  // set pic&others  picBox = new myLabel(this);  picBox->setGeometry(30, 10, 620, 700);  picBox->setStyleSheet("background-image: url(:/map\_1.jpg);"  "border-radius: 10px;");  // picBox->installEventFilter(this);  // QPainter painter(picBox);  // QPen P;  // P.setWidth(6); //设置画笔宽度  // P.setColor(Qt::red); //设置画笔颜色  // P.setStyle(Qt::DashLine); //设置画笔风格  // painter.setPen(P); //调用画笔  // picBox->Paint();  title = new QLabel(this);  title->setGeometry(740, 40, 440, 200);  title->setStyleSheet("background-image: url(:/title.jpg);"  " border-radius: 10px; ");  sel\_title\_1 = new QLabel(this);  sel\_title\_1->setGeometry(740, 280, 150, 50);  sel\_title\_1->setStyleSheet("background-image: url(:/title\_1.jpg);"  " border-radius: 10px; ");  sel\_title\_2 = new QLabel(this);  sel\_title\_2->setGeometry(1030, 280, 150, 50);  sel\_title\_2->setStyleSheet("background-image: url(:/title\_2.jpg);"  " border-radius: 10px; ");  comboBox\_1 = new QComboBox(this);  comboBox\_1->addItems(locateList);  comboBox\_1->setGeometry(740, 360, 150, 50);  comboBox\_2 = new QComboBox(this);  comboBox\_2->addItems(locateList);  comboBox\_2->setGeometry(1030, 360, 150, 50);  sel\_title\_3 = new QLabel(this);  sel\_title\_3->setGeometry(740, 440, 150, 50);  sel\_title\_3->setStyleSheet("background-image: url(:/title\_3.jpg);"  " border-radius: 10px; ");  distanceBox = new QLabel(this);  distanceBox->setGeometry(1030, 440, 150, 50);  distanceBox->setStyleSheet("background-color: white;"  " border-radius: 10px; ");  QFont font = distanceBox->font();  font.setPointSize(18);  font.setBold(true);  distanceBox->setFont(font);  distanceBox->setText("0");  distanceBox->setAlignment(Qt::AlignCenter);  sel\_title\_4 = new QLabel(this);  sel\_title\_4->setGeometry(740, 520, 150, 50);  sel\_title\_4->setStyleSheet("background-image: url(:/title\_4.jpg);"  " border-radius: 10px; ");  calcBtn = new QPushButton(this);  calcBtn->setGeometry(1030, 520, 150, 50);  calcBtn->setFont(font);  calcBtn->setText("点击计算");  connect(calcBtn, &QPushButton::clicked, this, &MainWindow::calcBtnClicked);  pathBox = new QLabel(this);  pathBox->setGeometry(740, 600, 440, 100);  pathBox->setStyleSheet("background-color: white;"  " border-radius: 10px; ");  pathBox->setWordWrap(true);  QFont pathFont = pathBox->font();  pathFont.setFamily("SimHei");  pathFont.setPointSize(12);  pathBox->setFont(pathFont);  // setMouseTracking(true);  }  MainWindow::~***MainWindow***()  {  delete ui;  }  void MainWindow::**calcBtnClicked**(){  vector<int> pre;  stack<int> stk;  pre.resize(Dijkstra().n);  int start = comboBox\_1->currentIndex(), end = comboBox\_2->currentIndex();  int distance = Dijkstra().dij(start, end, *pre*);  QString path = "";  while (pre[end] != end) {  stk.emplace(*end*);  end = pre[end];  }  stk.emplace(*end*);  pathStk = stk;  while(!stk.empty()){  path += locateList[stk.top()];  if(stk.size() != 1){  path += + " -> ";  }  stk.pop();  }  // qDebug() << "start: " << comboBox\_1->currentText();  // qDebug() << "startIndex: " << comboBox\_1->currentIndex();  // qDebug() << "end: " << comboBox\_2->currentText();  // qDebug() << "endIndex: " << comboBox\_2->currentIndex();  // qDebug() << "distance: " << distance;  // qDebug() << "path: " << path;  distanceBox->setText(QString::number(distance));  pathBox->setText(path);  // QPaintEvent\* evt = new QPaintEvent(QRect(50, 50, 90, 90));  // picBox->paintEvent(evt);  }  //事件过滤器  //bool MainWindow::eventFilter(QObject \*watched, QEvent \*event){  // if(watched == picBox && event->type() == QEvent::Paint){//在frame上画图  // Painter();  // }else{}  // return QWidget::eventFilter(watched,event); //将事件传递给父类  //}  //void MainWindow::paintEvent(QPaintEvent \*event){  // QPainter p(picBox); //在frame(框架上画图)  // QPen P;  // P.setWidth(6); //设置画笔宽度  // P.setColor(Qt::red); //设置画笔颜色  // P.setStyle(Qt::DashLine); //设置画笔风格  // p.setPen(P); //调用画笔  //// qDebug() << event->rect();  // p.drawLine(0, 0, 20, 20);  //}  //绘图  //void MainWindow::Painter(){  // QPainter p(picBox); //在frame(框架上画图)  // QPen P;  // P.setWidth(2); //设置画笔宽度  // P.setColor(Qt::black); //设置画笔颜色  // P.setStyle(Qt::DashLine); //设置画笔风格  // p.setPen(P); //调用画笔  // p.drawEllipse(QPoint(0,0),20,20); //画目标圆  // p.drawLine(200,0,200,400); //画横线  // p.drawLine(0,200,400,200); //画竖线  //// QPainter p(picBox); //在frame(框架上画图)  //// QPen P;  //// P.setWidth(6); //设置画笔宽度  //// P.setColor(Qt::red); //设置画笔颜色  //// P.setStyle(Qt::DashLine); //设置画笔风格  //// p.setPen(P); //调用画笔  //// int x0 = 0, x1 = 0, y0 = 0, y1 = 0;  //// while(!pathStk.empty()){  //// x0 = locatePosition[pathStk.top()][0];  //// y0 = locatePosition[pathStk.top()][1];  //// x0 -= 30, x1 -= 30;  //// pathStk.pop();  //// if(!pathStk.empty()){  //// x1 = locatePosition[pathStk.top()][0];  //// y1 = locatePosition[pathStk.top()][1];  //// y0 -= 10, y1 -= 10;  //// p.drawLine(x0, y0, x1, y1); //画横线  //// }  //// }  //}   * 1. myLabel.h   #ifndef MYLABEL\_H  #define MYLABEL\_H  #include <QLabel>  class **myLabel** : public QLabel  {  Q\_OBJECT  public:  explicit **myLabel**(QWidget \*parent = 0);  void **Paint**();  void ***paintEvent***(QPaintEvent \*); // 重写绘图事件  };  #endif // MYLABEL\_H   * 1. myLabel.cpp   #include "mylabel.h"  #include <QPainter>  myLabel::**myLabel**(QWidget \*parent) : QLabel(parent)  {  }  // 在控件发⽣重绘时触发的事件  void myLabel::***paintEvent***(QPaintEvent \*)  {  // 创建⼀个绘图对象，指定绘图设备为 QLabel  QPainter painter(this);  // 绘制⼀个图像  painter.drawLine(0, 0, 100, 100);  painter.end();  }  void myLabel::**Paint**(){  QPainter p(this); //在frame(框架上画图)  QPen P;  P.setWidth(2); //设置画笔宽度  P.setColor(Qt::black); //设置画笔颜色  P.setStyle(Qt::DashLine); //设置画笔风格  p.setPen(P); //调用画笔  p.drawEllipse(QPoint(0,0),20,20); //画目标圆  p.drawLine(200,0,200,400); //画横线  p.drawLine(0,200,400,200); //画竖线  }   * 1. main.cpp   #include "mainwindow.h"  #include <QApplication>  int main(int argc, char \*argv[])  {  QApplication a(*argc*, argv);  MainWindow w;  w.show();  return a.exec();  } | | | | |
| **实验总结** | 我原本设计了一个功能，在点击计算按钮的同时在左边图片上绘制路径线路，采用QT的绘图工具QPainter，但是遇到了诸多的问题，首先是绘制出的路径在图片的下方被图片遮挡，查阅相关资料，解决办法是为自定义新的Label类继承QT原本的QLabel，在新的Label类中重写绘图事件，我试了，但是发现绘图事件是在类被实例化的时候执行，有点类似构造函数的概念，但是使用button点击事件调用之后发现行不通，似乎是不能再次调用，我又采取安装事件过滤器的办法，针对绘图事件重新写函数，但是问题似乎仍然没有解决，至此在这方面花费的时间已经远超过之前的部分，我决定放弃，希望能在之后的学习中觉得现在遇到的问题。  除此之外本次实验还是学到了很多的东西，QT作为C++的GUI框架在之前的实践课之后很少用过了，这次重新拾起，发现因为加深了对C++知识的掌握，之前不懂的一些程序写法现在能理解了，于此同时又加深了图论算法的熟练度，受益匪浅。 | | | | |