 **地球科学学院实验报告**

专业： 地理信息科学

姓名： 秦卫付

学号： 3170106249

成绩：

教师签名： 陶煜波

课程名称： 地理空间数据库 实验名称： 实习3空间索引编程 指导老师： 陶煜波

实验人（组）： 秦卫付 日期： 2019年4月14日

1. 实验目的和要求（必填）

|  |
| --- |
| 了解空间数据类型层次结构，熟悉包围盒Envelope在空间查询中作用，熟悉常见的空间计算方法，掌握Z-Curve、Hilbert Curve和四叉树的创建，掌握四叉树的区域查询和最邻近查询方法，理解空间数据查询的过滤和精炼步。 |

1. 实验内容和原理（必填）

|  |
| --- |
| 1. 实现Z-curve曲线的二维坐标与Z值的相互转换  基于bit-shuffling思想，实现二维坐标(coor)与Z值(value)的相互转换。order为Z-Curve的阶数，order为4时，平面网格大小为(2^4, 2^4)，即16x16。  2. 实现Hilbert Curve的二维坐标与H值的相互转换  基于linear-quadtrees递归填充思想，实现二维坐标(coor)到H值(value)的相互转换。order为Hilbert Curve的阶数，order为4时，平面网格大小为(2^4, 2^4)，即16x16。  3. 包围盒空间关系判断  空间索引是基于几何特征的包围盒创建，首先通过判断几何特征的包围盒是否和查询区域相交，降低复杂的几何特征空间关系计算的次数。contain含义和PostGIS中的contain不同，可以是相同的包围盒。  4. Point到LineString和Polygon距离计算  Point到LineString的距离计算分解为Point到每个线段的距离计算，即点P(x, y)到线段[P1(x1, y1), P2(y2, y2)]在二维笛卡尔空间的最短距离。基本思路是通过(P2-P1)归一化向量和(P-P1)向量的内积，计算P在直线上的投影点，判断该投影点是否在线段上，如果在线段上，计算投影点到P的距离，不在线段上，计算P到线段端点距离的最小值。  Point到Polygon的距离计算关键是判断Point是否在Polygon内部，可以通过射线法判断，参考<https://www.cnblogs.com/luxiaoxun/p/3722358.html>，如果Point在Polygon内部，距离为0，否则计算Point与Polygon边界距离。Polygon定义与PostGIS相同，外环第一个点和最后一个点为同一个点。  5. 四叉树构建  四叉树创建输入一组几何特征，将节点分裂为四个子节点，每个特征加到包围盒重叠的子节点中（即一个特征可能在多个节点中），删除当前节点的几何特征记录（即所有特征只存储在叶节点中），如果子节点的几何特征个数大于capacity，递归分裂子节点。  6. 基于四叉树的区域查询  区域查询输入区域rect，查询与区域rect相交的几何特征，存储在features。区域rect如果与当前节点的包围盒bbox相交，递归遍历四叉树，查询哪些几何特征的包围盒和查询区域相交（filter）；再获得可能和查询区域相交的候选几何特征后，精确判断几何特征是否与查询区域相交（refine）。  通过鼠标选择查询区域，在站点和道路数据上，验证区域查询。  7. 基于四叉树的最邻近几何特征查询  最邻近几何特征查询(K-NN)输入查询点(x, y)，返回与该点最邻近的几何特征，存储在feature。首先，通过pointInLeafNode查询点(x, y)所在的叶节点，计算查询点(x, y)与该叶节点内的几何特征包围盒的最大距离的最小值minDist，即通过包围盒而非原始几何加速最小距离计算；然后，构造查询区域 (x – minDist, x + minDist, y – minDist, y + minDist)，查询几何特征的包围盒与该区域相交的几何特征（filter），再查询与查询点(x, y)距离最近的几何特征（refine）。  通过鼠标移动选择离鼠标最近的几何特征（站点和道路），验证最邻近几何特征查询。  [选做题部分]  9  10  11 |

1. 主要仪器设备（必填）

|  |
| --- |
| 编程语言：C++  编程工具：Visual Studio 2012/2015  仪器：电脑 |

1. 实验步骤和记录

|  |
| --- |
| 1. 实现Z-curve曲线的二维坐标与Z值的相互转换   [代码]  void zorder(int order, int& value, int coor[2])  {  // Calculate the z-order by bit shuffling  value = 0;  for (int i = 0; i < 2; ++i) {  for (int j = 0; j < order; ++j) {  // Task 1.1 zorder，修改以下代码  int mask = 1 << j;  // Check whether the value in the position is 1  if (coor[i] & mask)  // Do bit shuffling  value |= 1 << (2 \* j + 1 - i);  }  }  }  void izorder(int order, int value, int coor[2])  {  // Initialize the coordinate to zeros  for (int i = 0; i < 2; ++i)  coor[i] = 0;  // Task 1.2 izoder  // Write your code here  for (int i = 0; i < 2; ++i)  {  for (int j = 0; j < order; ++j) {  int mask = 1 << (2 \* j + 1 - i);  if (value&mask)  coor[i] |= 1 << j;  }  }  }  2. 实现Hilbert Curve的二维坐标与H值的相互转换  [代码]  void horder(int order, int& value, int coor[2])  {  // order = 1  int num = int(pow(2, 1));  int \* hcurve = new int[num \* num];  hcurve[0] = 1;  hcurve[1] = 2;  hcurve[2] = 0;  hcurve[3] = 3;  for (int i = 2; i <= order; ++i) {  int add = (int)pow(2, 2 \* i - 2); // the number of values in order - 1  int blockLen = (int)pow(2, i - 1);  int \* temp = hcurve;  num = int(pow(2, i));  hcurve = new int[num \* num];  for (int j = 0; j < blockLen; ++j) {  for (int k = 0; k < blockLen; ++k) {  // Task 2.1 horder，修改以下四行代码  hcurve[k\*num + j] = temp[k\*blockLen + j] + add;  hcurve[k\*num + (j + blockLen)] = temp[k\*blockLen + j] + 2 \* add;  hcurve[(k + blockLen)\*num + j] = temp[(blockLen - 1 - j)\*blockLen + (blockLen - 1 - k)];  hcurve[(k + blockLen)\*num + (j + blockLen)] = temp[j\*blockLen + k] + 3 \* add;  }  }    delete temp;  }  // Task 2.1 horder，修改以下一行代码  value = hcurve[(num - 1 - coor[1])\*num + coor[0]];  delete hcurve;  }  void ihorder(int order, int value, int coor[])  {  // order = 1  int num = int(pow(2, 1));  int \* hcurve = new int[num \* num];  hcurve[0] = 1;  hcurve[1] = 2;  hcurve[2] = 0;  hcurve[3] = 3;  // Task 2.2 ihorder  // Write your code here  for (int i = 2; i <= order; ++i) {  int add = (int)pow(2, 2 \* i - 2); // the number of values in order - 1  int blockLen = (int)pow(2, i - 1);  int \* temp = hcurve;  num = int(pow(2, i));  hcurve = new int[num \* num];  for (int j = 0; j < blockLen; ++j) {  for (int k = 0; k < blockLen; ++k) {  hcurve[k\*num + j] = temp[k\*blockLen + j] + add;  if (i == order && hcurve[k\*num + j] == value)  {  coor[0] = j;  coor[1] = num - 1 - k;  return;  }  hcurve[k\*num + (j + blockLen)] = temp[k\*blockLen + j] + 2 \* add;  if (i == order && hcurve[k\*num + (j + blockLen)] == value)  {  coor[0] = j + blockLen;  coor[1] = num - 1 - k;  return;  }  hcurve[(k + blockLen)\*num + j] = temp[(blockLen - 1 - j)\*blockLen + (blockLen - 1 - k)];  if (i == order && hcurve[(k + blockLen)\*num + j] == value)  {  coor[0] = j;  coor[1] = num - 1 - (k + blockLen);  return;  }  hcurve[(k + blockLen)\*num + (j + blockLen)] = temp[j\*blockLen + k] + 3 \* add;  if (i == order && hcurve[(k + blockLen)\*num + (j + blockLen)] == value)  {  coor[0] = j + blockLen;  coor[1] = num - 1 - (k + blockLen);  return;  }  }  }  delete temp;  }  delete hcurve;  }  3. 包围盒空间关系判断  [代码]  bool Envelope::contain(const Envelope& envelope) const  {  // Task 3.1 测试Envelope是否包含关系  // Write your code here  if (getMaxX() >= envelope.getMaxX() && getMinX() <= envelope.getMinX() && getMaxY() >= envelope.getMaxY() && getMinY() <= envelope.getMinY())  return true;  else  return false;  }  bool Envelope::intersect(const Envelope& envelope) const  {  // Task 3.2 测试Envelope是否相交  // Write your code here  if (getMaxX() < envelope.getMinX() || getMinX() > envelope.getMaxX() || getMaxY() < envelope.getMinY() || getMinY() > envelope.getMaxY())  return false;  else  return true;  }  Envelope Envelope::unionEnvelope(const Envelope& envelope) const  {  // Task 3.3 合并两个Envelope生成一个新的Envelope  // Write your code here  double maxX1;  maxX1 = (maxX >= envelope.getMaxX()) ? maxX: envelope.getMaxX();  double minX1;  minX1 = (minX <= envelope.getMinX()) ? minX : envelope.getMinX();  double maxY1;  maxY1= (maxY >= envelope.getMaxY()) ? maxY : envelope.getMaxY();  double minY1;  minY1 = (minY <= envelope.getMinY()) ? minY : envelope.getMinY();  //cout << minX1 << " " << maxX1 << " " << minY1 << " " << maxY1 << endl;  Envelope envelope1(minX1, maxX1, minY1, maxY1);  return envelope1;  }  4. Point到LineString和Polygon距离计算  [代码]  double Point::distance(const Point\* point) const  {  return sqrt((x - point->x) \* (x - point->x) + (y - point->y) \* (y - point->y));  }  double Point::distance(const LineString\* line) const  {  double mindist = line->getPointN(0).distance(this);  for (size\_t i = 0; i < line->numPoints() - 1; ++i) {  double dist = 0;  double x1 = line->getPointN(i).getX();  double y1 = line->getPointN(i).getY();  double x2 = line->getPointN(i + 1).getX();  double y2 = line->getPointN(i + 1).getY();  // Task 4.1 calculate the distance between Point P(x, y) and Line [P1(x1, y1), P2(x2, y2)] (less than 10 lines)  // Write your code here  if (x1 == x2)  {  if ((y >= y1 && y <= y2) || (y <= y1 && y >= y2))  dist = ((x - x1) > 0) ? x - x1 : x1 - x;  else  {  Point p1(x1, y1), p2(x2, y2);  dist = (distance(&p1) < distance(&p2)) ? distance(&p1) : distance(&p2);  }  }  else if (y1 == y2)  {  if ((x >= x1 && x <= x2) || (x <= x1 && x >= x2))  dist = ((y - y1) >= 0) ? y - y1 : y1 - y;  else  {  Point p1(x1, y1), p2(x2, y2);  dist = (distance(&p1) < distance(&p2)) ? distance(&p1) : distance(&p2);  }  }  else  {  double Y = y2 - y1, X = x2 - x1;  double y0 = (y1\*X\*X+Y\*Y\*y+X\*Y\*x-X\*Y\*x1) / (X\*X + Y\*Y);  double x0 = (X/ Y \* (y0 - y1) + x1);  Point p(x0, y0);  Point p1(x1, y1), p2(x2, y2);  Envelope e(x1 < x2 ? x1 : x2, x1 < x2 ? x2 : x1, y1 < y2 ? y1 : y2, y1 < y2 ? y2 : y1);  if (e.contain(x0, y0))  dist = distance(&p);  else  dist= (distance(&p1) < distance(&p2)) ? distance(&p1) : distance(&p2);  }  if (dist < mindist)  mindist = dist;  }  return mindist;  }  double Point::distance(const Polygon\* polygon) const  {  LineString line = polygon->getExteriorRing();  size\_t n = line.numPoints();  bool inPolygon = false;  // Task 4.2 whether Point P(x, y) is within Polygon (less than 15 lines)  // write your code here  int crossings = 0;  for (size\_t i = 0; i < line.numPoints() - 1; i++) {  double x1 = line.getPointN(i).getX();  double y1 = line.getPointN(i).getY();  double x2 = line.getPointN(i + 1).getX();  double y2 = line.getPointN(i + 1).getY();  double slope = (y2 - y1) / (x2 - x1);  bool cond1 = (x1 <= x) && (x < x2);  bool cond2 = (x2 <= x) && (x < x1);  bool above = (y < slope\*(x - x1) + y1);  if ((cond1 || cond2) && above)  crossings++;  }  if (crossings % 2 != 0)  inPolygon = true;  if (crossings == 0)  inPolygon = false;  double mindist = 0;  if (!inPolygon)  mindist = this->distance(&line);  return mindist;  }  5. 四叉树构建  [代码]  bool QuadTree::constructQuadTree(vector<Feature>& features)  {  if (features.empty())  return false;  // Task 5.1 construction  // Write your code here  Envelope e = features.at(0).getEnvelope();  int i;  for (i = 1; i < features.size(); i++)  {  e = e.unionEnvelope(features.at(i).getEnvelope());  }  root = new QuadNode(e);  //root = new QuadNode(Envelope(-74.1, -73.8, 40.6, 40.8).unionEnvelope(e));//root节点初始化  root->add(features);  if (features.size() <= this->capacity)//个数少于capacity时,不需要分割  {  //return true;  Envelope e1 = features.at(0).getEnvelope();  for (int i = 1; i < features.size(); i++)  {  Envelope e2 = features.at(i).getEnvelope();  e1 = e1.unionEnvelope(e2);  }  bbox = e1;  }  else  {  root->split(capacity);  }  bbox = e;//Envelope(-74.1, -73.8, 40.6, 40.8).unionEnvelope(e);//Envelope(-74.1, -73.8, 40.6, 40.8);  // 注意此行代码需要更新为features的包围盒，或根节点的包围盒  return true;  }  void QuadNode::split(size\_t capacity)  {  for (int i = 0; i < 4; ++i) {  delete[]nodes[i];  nodes[i] = NULL;  }  //获取当前包围盒  double X = bbox.getMaxX(), Y = bbox.getMaxY(), x = bbox.getMinX(), y = bbox.getMinY();  nodes[0] = new QuadNode(Envelope(x, (x + X) / 2, (y + Y) / 2, Y));  nodes[1] = new QuadNode(Envelope((x + X) / 2, X, (y + Y) / 2, Y));  nodes[2] = new QuadNode(Envelope((x + X) / 2, X, y, (y + Y) / 2));  nodes[3] = new QuadNode(Envelope(x, (x + X) / 2, y, (y + Y) / 2));  for (int j = 0; j < 4; j++) {  vector<Feature>::iterator p = features.begin();  while (p != features.end())//为node添加元素  {  if (nodes[j]->bbox.intersect((\*p).getEnvelope()) || nodes[j]->bbox.contain((\*p).getEnvelope()))  {  nodes[j]->add((\*p));  }  p++;  }  }    features.clear();  for (int i = 0; i < 4; i++)  {  if (nodes[i] != NULL)  {  if (nodes[i]->getFeatureNum() > capacity)  {  nodes[i]->split(capacity);  }  }  }  }  6. 基于四叉树的区域查询  [代码]  void QuadNode::rangeQuery(Envelope& rect, vector<Feature>& features)  {  if (!bbox.intersect(rect))  return;  // Task 6.2 range query  // Write your code here  bool isLeaf = true;  for (int i = 0; i < 4; i++)//判断是不是叶节点  if (nodes[i] != NULL)  {  isLeaf = false;  break;  }  if (isLeaf)//是叶节点如进入向量  {  vector<Feature>::iterator p = this->features.begin();  while (p != this->features.end())  {  if (rect.intersect((\*p).getEnvelope()))  features.push\_back(\*p);  p++;  }  }  else//不是叶节点，遍历其子节点  for (int j = 0; j < 4; j++)  {  nodes[j]->rangeQuery(rect, features);  }  }  void QuadTree::rangeQuery(Envelope& rect, vector<Feature>& features)  {  features.clear();  // Task 6.1 range query  // Write your code here  // filter step (选择查询区域与几何对象包围盒相交的几何对象)  root->rangeQuery(rect, features);  // refine step (精确判断时，需要去重，避免查询区域和几何对象的重复计算)  //去重  vector<Feature>::iterator p = features.begin(), q;  while (p != features.end())  {  q = p + 1;  while (q != features.end())  {  if ((\*p).getGeom() == (\*q).getGeom())  q = features.erase(q);  else q++;  }  p++;  }  //jisuan  p = features.begin();  while (p != features.end())  {  if (rect.contain((\*p).getEnvelope()))  p++;  else p = features.erase(p);  }  }  7. 基于四叉树的最邻近几何特征查询  [代码]  bool QuadTree::NNQuery(double x, double y, Feature& feature)  {  if (!root || !(root->getEnvelope().contain(x, y)))  return false;  // Task 7.1 NN query  // Write your code here  // filter step (使用maxDistance2Envelope函数，获得查询点到几何对象包围盒的最短的最大距离，然后区域查询获得候选集)  QuadNode \*qd = root->pointInLeafNode(x, y);    const Envelope& envelope = qd->getEnvelope();  double minDist = max(envelope.getWidth(), envelope.getHeight());  Envelope E(x - minDist, x + minDist, y - minDist, y + minDist);  vector<Feature> ff;  if (qd != NULL)  {  this->rangeQuery(E, ff);    }  // refine step (精确计算查询点与几何对象的距离)  Point point(x, y);  vector<Feature> ::iterator p = ff.begin();  Feature temp;  while (p != ff.end())  {  if ((\*p).distance(x, y) < minDist) {  minDist = (\*p).distance(x, y);  temp = \*p;  }  p++;  }  feature = temp;  ff.clear();  return true;  }  [选做题代码]  9  void QuadTreeAnalysis()  {  vector<Feature> features;  vector<Geometry \*> geom = readGeom(".//data/taxi");  vector<string> name = readName(".//data/taxi");  features.clear();  features.reserve(geom.size());  for (size\_t i = 0; i < geom.size(); ++i)  features.push\_back(Feature(name[i], geom[i]));  cout << "taxi number: " << geom.size() << endl;  srand(time(NULL));  for (int cap = 70; cap <= 200; cap += 10) {  QuadTree \*qtree = new QuadTree();  // Task 9 构造四叉树，输出四叉树的节点数目和高度  // Write your code here  qtree->setCapacity(cap);    clock\_t start\_time = clock();  // Write your code here  qtree->constructQuadTree(features);  clock\_t end\_time = clock();  int height = 0, interiorNum = 0, leafNum = 0;  // Write your code here  qtree->countHeight(height);  qtree->countQuadNode(interiorNum, leafNum);  cout << "Capacity " << cap << "\n";  cout << "Height: " << height << " \tInterior node number: " << interiorNum << " \tLeaf node number: " << leafNum << "\n";  cout << "Construction time: " << (end\_time - start\_time) / 1000.0 << "s" << endl;  double x, y;  Feature f;  start\_time = clock();  for (int i = 0; i < 100000; ++i) {  x = -((rand() % 225) / 10000.0 + 73.9812);  y = (rand() % 239) / 10000.0 + 40.7247;  qtree->NNQuery(x, y, f);  }  end\_time = clock();  cout << "NNQuery time: " << (end\_time - start\_time) / 1000.0 << "s" << endl << endl;  delete qtree;  }  }  10  double LineString::distance(const LineString\* line) const  {  //cout << "to be implemented: LineString::distance(const LineString\* line)\n";  int numberOfPoint1, numberOfPoint2;  numberOfPoint1 = this->numPoints();  numberOfPoint2 = line->numPoints();  //判断是否相交  double x1, y1, x2, y2, x3, y3, x4, y4;  bool ISInterset = false;  for (int i = 0; i < numberOfPoint1 - 1; i++)  {  x1 = this->getPointN(i).getX(), y1 = this->getPointN(i).getY();  x2 = this->getPointN(i + 1).getX(), y2 = this->getPointN(i + 1).getY();  for (int j = 0; j < numberOfPoint2 - 1; j++)  {  x3 = line->getPointN(j).getX(), y3 = line->getPointN(j).getY();  x4 = line->getPointN(j + 1).getY(), y4 = line->getPointN(j + 1).getY();  if (!(min(x1, x2) <= max(x3, x4) && min(y3, y4) <= max(y1, y2) && min(x3, x4) <= max(x1, x2) && min(y1, y2) <= max(y3, y4)))  ISInterset = false;  double fc = (y3 - y1) \* (x2 - x1) - (x3 - x1) \*(y2 - y1);  double fd = (y4 - y1) \* (x2 - x1) - (x4 - x1) \*(y2 - y1);  if (fc \* fd > 0)  ISInterset = false;  else  ISInterset = true;  if (ISInterset == true)  break;  }  if (ISInterset == true)  break;  }  double distance, mindist;  if (ISInterset)//如果相交距离为0  mindist = 0;  else  {  mindist = this->getPointN(0).distance(line);  for (int i = 1; i < numberOfPoint1; i++)  {  distance = getPointN(i).distance(line);  if (distance < mindist)  mindist = distance;  }  for (int i = 0; i < numberOfPoint2; i++)  {  distance = line->getPointN(i).distance(this);  if (distance < mindist)  mindist = distance;  }  }  return mindist;  }  double LineString::distance(const Polygon\* polygon) const  {  //cout << "to be implemented: LineString::distance(const Polygon\* polygon)\n";  //判断线是不是在多边形内  bool inPolygon = false;  int number = this->numPoints();  LineString line = polygon->getExteriorRing();  for (int i = 0; i < number; i++)  {  double x = this->getPointN(i).getX(), y = this->getPointN(i).getY();  int crossings = 0;  for (size\_t i = 0; i <line.numPoints() - 1; i++) {  double x1 = line.getPointN(i).getX();  double y1 = line.getPointN(i).getY();  double x2 = line.getPointN(i + 1).getX();  double y2 = line.getPointN(i + 1).getY();  double slope = (y2 - y1) / (x2 - x1);  bool cond1 = (x1 <= x) && (x < x2);  bool cond2 = (x2 <= x) && (x < x1);  bool above = (y < slope\*(x - x1) + y1);  if ((cond1 || cond2) && above)  crossings++;  }  if (crossings % 2 != 0)  inPolygon = true;  if (crossings == 0)  inPolygon = false;  if (inPolygon == true)  break;  }  double dist;  if (inPolygon)//在多边形内部，距离为0  dist = 0;  else  dist = this->distance(&line);  return dist;  }  11  bool Polygon::intersects(const Envelope& rect) const  {  //cout << "to be implemented: Polygon::intersects(const Envelope& box)\n";  bool IsInterset = false;  LineString polyLine = this->getExteriorRing();  for (int i = 0; i < polyLine.numPoints(); i++)  {  if(rect.intersect(polyLine.getPointN(i).getEnvelope()))  IsInterset=true;  }  return IsInterset;  } |

1. 实验结果与分析（必填）

|  |
| --- |
| 1. 实现Z-curve曲线的二维坐标与Z值的相互转换  [TEST1测试结果截图]    2. 实现Hilbert Curve的二维坐标与H值的相互转换  [TEST2测试结果截图]    3. 包围盒空间关系判断  [TEST3测试结果截图]    4. Point到LineString和Polygon距离计算  [TEST4测试结果截图]    [TEST5测试结果截图]    5. 四叉树构建  [TEST6测试结果截图]    [站点四叉树截图]    [道路四叉树截图]    6. 基于四叉树的区域查询  [站点区域查询截图2张]      [道路区域查询截图2张]      7. 基于四叉树的最邻近几何特征查询  [站点最邻近查询截图2张]      [道路最邻近查询截图2张]      [选做题实习结果]  9、      结论：当每个四叉树的容量越大时，建立四叉树的时间越短，但是与此同时其查询的时间就会增加，所以应该适当选择容量的大小来提高查询性能。  10、11简单测试的结果和postgre测试的结果一致 |

1. 实习体会

|  |
| --- |
| 本次作业花了多长时间完成，哪些题目太简单或太难，有什么收获，其他建议？  花了两天完成，第二个曲线的绘制和四叉树的建立太难了；收获是提高了编程能力；  建议是给出第二章曲线绘制的算法思路。 |

1. 评语

|  |
| --- |
| 通过此次实习，充分认识到了图形化展示地理数据的重要性，在实习过程中收获了许多的乐趣；也在解决问  题的过程中收获了自信。 |