**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Бинарные деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Виноградов К.А. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2017

**Цель работы.**

Знакомство с такими структурами данных, как бинарные деревья, их реализация и построение и изучение связанных с ними алгоритмов.

**Текст задания.**

Задано бинарное дерево b типа ВТ с типом элементов Elem. Для введенной пользователем величины Е (var E: Elem) найти в дереве b длину пути (число ветвей) от корня до ближайшего узла с элементом Е (если Е не входит в b, за ответ принять -1).

**Понятие бинарного дерева.**

Бинарное дерево – конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом. Реализация стека и функций, связанных с ним представлена в приложении А.

Определим скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= Λ,

< непустое БД > ::= ( < корень > < БД > < БД > ).

Здесь пустое дерево имеет специальное обозначение − Λ.

В программе бинарное дерево реализовано как двусвязный список структур с полем данных типа elem. Реализация и функции для работы с деревом представлены в приложении А.

**Ход работы.**

В данном отчете представлены лишь функции, написанные для решения задачи. Все иные, автоматически сгенерированные файлы, в отчет не включены.

В программе реализован доступ к Проводнику Windows. в нем происходит выбор файла для считывания данных. Из функции чтения из файла вызывается функция построения бинарного дерева. Код функций представлен в приложении А.

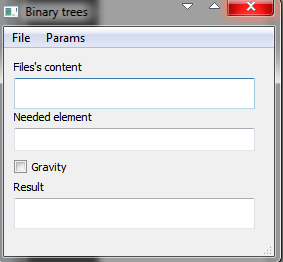
После этого есть возможность построить визуальное отображение дерева. Для этого были написаны функции реализующие подготовку полотна, создание узлов-объектов, соединение узлов, размещение их на полотне. Код функций представлен в приложении Б.

Также реализована функция нахождения минимального расстояния до вхождения объекта(-ов) в дереве, основании работы которой в поле «textEdit» выводится результат, после чего программа ожидает повторного запроса или завершения. Код функции представлен в приложении А

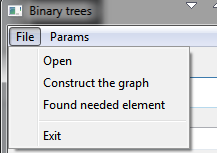
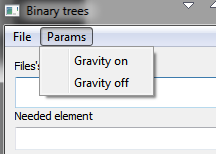
**Пользовательский интерфейс.**

Для более удобной работы программы был создан пользовательский интерфейс:

* основное окно программы



* дополнительные функции на основном окне

**Тестирование программы.**

Таблица 1. Результаты тестирования программы

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| F c: (a(b(d(m(n)(o)))(f))(c(a)(b(E))))  N el: abd | 0 | 1 | 2 | |
| F c:(a(b(d(m(n)(o)(f)(H)))(f))(c(a)(b(E)))) | ERROR, You entered wrong tree! |
| F c: ( )  N el: abd | -1 | -1 | -1 | |
| Empty ex(nope)  N el: sbc | -1 | 1 | 2 | |

**Вывод.**

В результате выполнения данной лабораторной работы были построены функции для создания бинарного дерева, а также созданапрограмма, использующая данные функции.

**ПРИЛОЖЕНИЕ А**

**Реализация дерева и функций для работы с ним**

BT\* **RootBT**(BT\* tree){

return tree->Root;

}

BT\* **LeftBT**(BT\* tree){

return tree->Left;

}

BT\* **RightBT**(BT\* tree){

return tree->Right;

}

bool **NullBT**(BT\* tree){

if (tree == nullptr){

return 1;

}

return 0;

}

BT\* **ConsBT**(elem info, BT\* Left, BT\* Right, BT\* Root){

BT\* tree = new BT;

tree->info = info;

tree->Root = Root;

tree->Right = Right;

tree->Left = Left;

return tree;

}

int **WayLength**(BT\* tree, int counter, elem point){

int l=-1, r=-1, min=-1;

if(tree->info==point)

{

counter++;

return counter;

}

if(tree->info!=point)

{

counter++;

if(tree->Left!=nullptr)

{

l=WayLength(tree->Left, counter, point);

if((min==-1||min>l)&&(l!=-1))

min=l;

}

if(tree->Right!=nullptr)

{

r=WayLength(tree->Right, counter, point);

if((min==-1||min>r)&&(r!=-1))

min=r;

}

if((tree->Left==nullptr)&&(tree->Right==nullptr)&&(tree->Root==nullptr))

{

return -1;

}

}

if(l!=-1||r!=-1)

{

if(l!=-1&&l<r)

min=l;

else if(r!=-1)

min=r;

}

return min;

}

BT\* **Read**(QString str, BT\* TRoot, int\* i){

if (str[\*i] == 'E'){

(\*i)++;

return nullptr;

}

if(str[\*i] == ' ')

(\*i)++;

if (str[\*i] == '('){

(\*i)++;

BT\* tree = ConsBT(str[(\*i)++], nullptr, nullptr, TRoot);

if(str[\*i] == ' ')

(\*i)++;

if (str[\*i] == '(')

tree->Left = Read(str, tree, i);

if (str[\*i] == 'E')

(\*i)++;

if(str[\*i] == ' ')

(\*i)++;

if (str[\*i] == '(')

tree->Right = Read(str, tree, i);

if (str[\*i] == ')'){

(\*i)++;

return tree;

}

return tree;

}

return nullptr;

}

BT\* **ReadFile**(QString fileName, QTextEdit\* te){

QFile file1(fileName);

file1.*open*(QIODevice::ReadOnly);

if (!file1.isOpen()){

QMessageBox msg;

msg.setText("File isn't open!");

msg.setWindowTitle("ERROR");

msg.*exec*();

return nullptr;

}

QTextStream file1s(&file1);

QString temp = file1s.readAll();

te->setPlainText(temp);

int i = 0;

return Read(temp, nullptr, &i);

}

**ПРИЛОЖЕНИЕ Б**

**Визуальное построение дерева**

Файл edge.cpp:

static const double Pi = 3.14159265358979323846264338327950288419717;

static double TwoPi = 2.0 \* Pi;

//! [0]

Edge::**Edge**(Node \*sourceNode, Node \*destNode, bool vis)

: arrowSize(10)

{

setAcceptedMouseButtons(0);

source = sourceNode;

dest = destNode;

visible = vis;

source->addEdge(this);

dest->addEdge(this);

adjust();

}

//! [0]

//! [1]

Node \*Edge::**sourceNode**() const

{

return source;

}

Node \*Edge::**destNode**() const

{

return dest;

}

//! [1]

//! [2]

void Edge::**adjust**()

{

if (!source || !dest)

return;

QLineF line(mapFromItem(source, 0, 0), mapFromItem(dest, 0, 0));

qreal length = line.length();

prepareGeometryChange();

if (length > qreal(20.)) {

QPointF edgeOffset((line.dx() \* 10) / length, (line.dy() \* 10) / length);

sourcePoint = line.p1() + edgeOffset;

destPoint = line.p2() - edgeOffset;

} else {

sourcePoint = destPoint = line.p1();

}

}

//! [2]

//! [3]

QRectF Edge::***boundingRect***() const

{

if (!source || !dest)

return QRectF();

qreal penWidth = 1;

qreal extra = (penWidth + arrowSize) / 2.0;

return QRectF(sourcePoint, QSizeF(destPoint.x() - sourcePoint.x(),

destPoint.y() - sourcePoint.y()))

.normalized()

.adjusted(-extra, -extra, extra, extra);

}

//! [3]

//! [4]

void Edge::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*, QWidget \*)

{

if (!source || !dest)

return;

QLineF line(sourcePoint, destPoint);

if (qFuzzyCompare(line.length(), qreal(0.)))

return;

//! [4]

//! [5]

if (visible){

// Draw the line itself

painter->setPen(QPen(Qt::red, 1, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter->drawLine(line);

//! [5]

//! [6]

// Draw the arrows

double angle = ::acos(line.dx() / line.length());

if (line.dy() >= 0)

angle = TwoPi - angle;

QPointF sourceArrowP1 = sourcePoint + QPointF(sin(angle + Pi / 3) \* arrowSize,

cos(angle + Pi / 3) \* arrowSize);

QPointF sourceArrowP2 = sourcePoint + QPointF(sin(angle + Pi - Pi / 3) \* arrowSize,

cos(angle + Pi - Pi / 3) \* arrowSize);

painter->setBrush(Qt::black);

painter->drawPolygon(QPolygonF() << line.p1() << sourceArrowP1 << sourceArrowP2);

// painter->drawPolygon(QPolygonF() << line.p2() << destArrowP1 << destArrowP2);

}

}

//! [6]

Файл graphwidget.cpp:

//! [0]

int GraphWidget::**Build**(QWidget\* parent, BT \*tree, qreal x, qreal y, QGraphicsScene \*scene, bool\* gr){

tree->node = new Node(this, gr);

scene->addItem(tree->node);

tree->node->setPos(x, y);

tree->node->text = tree->info;

if (RootBT(tree)!=nullptr){

scene->addItem(new Edge(tree->node, tree->Root->node, true));

}

qreal delta;

if (x == 0)

delta = -100;

else

delta = (y+51)/2;

if (LeftBT(tree)!=nullptr)

Build(parent, LeftBT(tree), x+delta, y+50, scene, gr);

if (RightBT(tree)!=nullptr)

Build(parent, RightBT(tree), x-delta, y+50, scene, gr);

if ((LeftBT(tree)!=nullptr) && (RightBT(tree)!=nullptr)){

scene->addItem(new Edge(LeftBT(tree)->node, RightBT(tree)->node, false));

if ((RightBT(LeftBT(tree))!=nullptr) && ((LeftBT(RightBT(tree)))!=nullptr)){

scene->addItem(new Edge(RightBT(LeftBT(tree))->node, LeftBT(RightBT(tree))->node, false));

if ((RightBT(LeftBT(LeftBT(tree)))!=nullptr) && ((LeftBT(RightBT(RightBT(tree))))!=nullptr))

scene->addItem(new Edge(RightBT(RightBT(LeftBT(tree)))->node, LeftBT(LeftBT(RightBT(tree)))->node, false));

}

}

return 0;

}

GraphWidget::**GraphWidget**(QWidget \*parent, BT\* tree, bool\* gr)

: QGraphicsView(parent), timerId(0)

{

QGraphicsScene \*scene = new QGraphicsScene(this);

scene->setItemIndexMethod(QGraphicsScene::NoIndex);

scene->setSceneRect(-375, -275, 750, 550);

setScene(scene);

setCacheMode(CacheBackground);

setViewportUpdateMode(BoundingRectViewportUpdate);

setRenderHint(QPainter::Antialiasing);

setTransformationAnchor(AnchorUnderMouse);

scale(qreal(0.98), qreal(0.98));

setMinimumSize(750, 550);

setWindowTitle(tr("Elastic Nodes"));

//! [0]

//! [1]

Build(this, tree, 0, -200, scene, gr);

}

//! [1]

//! [2]

void GraphWidget::**itemMoved**()

{

if (!timerId)

timerId = startTimer(1000 / 25);

}

void GraphWidget::***timerEvent***(QTimerEvent \*event)

{

Q\_UNUSED(event);

QList<Node \*> nodes;

foreach (QGraphicsItem \*item, scene()->items()) {

if (Node \*node = qgraphicsitem\_cast<Node \*>(item))

nodes << node;

}

foreach (Node \*node, nodes)

node->calculateForces();

bool itemsMoved = false;

foreach (Node \*node, nodes) {

if (node->advance())

itemsMoved = true;

}

if (!itemsMoved) {

killTimer(timerId);

timerId = 0;

}

}

//! [4]

//! [6]

void GraphWidget::***drawBackground***(QPainter \*painter, const QRectF &rect)

{

Q\_UNUSED(rect);

// Shadow

QRectF sceneRect = this->sceneRect();

QRectF rightShadow(sceneRect.right(), sceneRect.top() + 5, 5, sceneRect.height());

QRectF bottomShadow(sceneRect.left() + 5, sceneRect.bottom(), sceneRect.width(), 5);

if (rightShadow.intersects(rect) || rightShadow.contains(rect))

painter->fillRect(rightShadow, Qt::darkGray);

if (bottomShadow.intersects(rect) || bottomShadow.contains(rect))

painter->fillRect(bottomShadow, Qt::darkGray);

// Fill

/\*QLinearGradient gradient(sceneRect.topLeft(), sceneRect.bottomRight());

gradient.setColorAt(0, Qt::white);

gradient.setColorAt(1, Qt::green);

painter->fillRect(rect.intersected(sceneRect), gradient);

painter->setBrush(Qt::NoBrush); \*/

painter->drawRect(sceneRect);

// Text

QRectF textRect(sceneRect.left() + 4, sceneRect.top() + 4,

sceneRect.width() - 4, sceneRect.height() - 4);

QString message("Graph output");

QFont font = painter->font();

font.setBold(true);

font.setPointSize(14);

painter->setFont(font);

painter->setPen(Qt::lightGray);

painter->drawText(textRect.translated(2, 2), message);

painter->setPen(Qt::black);

painter->drawText(textRect, message);

}

//! [6]

Файл node.cpp:

Node::**Node**(GraphWidget \*graphWidget, bool\* gr)

: graph(graphWidget)

{

setFlag(ItemIsMovable);

setFlag(ItemSendsGeometryChanges);

setCacheMode(DeviceCoordinateCache);

setZValue(-1);

gravity = gr;

}

//! [0]

//! [1]

void Node::**addEdge**(Edge \*edge)

{

edgeList << edge;

edge->adjust();

}

QList<Edge \*> Node::**edges**() const

{

return edgeList;

}

//! [1]

//! [2]

void Node::**calculateForces**()

{

if (!scene() || scene()->mouseGrabberItem() == this) {

newPos = pos();

return;

}

//! [2]

//! [3]

// Sum up all forces pushing this item away

qreal xvel = 0;

qreal yvel = 0;

foreach (QGraphicsItem \*item, scene()->items()) {

Node \*node = qgraphicsitem\_cast<Node \*>(item);

if (!node)

continue;

QPointF vec = mapToItem(node, 0, 0);

qreal dx = vec.x();

qreal dy = vec.y();

double l = 2 \* (dx \* dx + dy \* dy);

if (l > 0) {

xvel += (dx \* 150.0) / l;

yvel += (dy \* 150.0) / l;

}

}

//! [3]

//! [4]

// Now subtract all forces pulling items together

double weight = (edgeList.size() + 1) \* 10;

foreach (Edge \*edge, edgeList) {

QPointF vec;

if (edge->sourceNode() == this)

vec = mapToItem(edge->destNode(), 0, 0);

else

vec = mapToItem(edge->sourceNode(), 0, 0);

xvel -= vec.x() / weight;

yvel -= vec.y() / weight;

}

//! [4]

//! [5]

qreal lim = 0.1;

if (\*gravity == false){

lim = 10;

}

if (qAbs(xvel) < lim && qAbs(yvel) < lim)

xvel = yvel = 0;

//! [5]

//! [6]

QRectF sceneRect = scene()->sceneRect();

newPos = pos() + QPointF(xvel, yvel);

newPos.setX(qMin(qMax(newPos.x(), sceneRect.left() + 10), sceneRect.right() - 10));

newPos.setY(qMin(qMax(newPos.y(), sceneRect.top() + 10), sceneRect.bottom() - 10));

}

//! [6]

//! [7]

bool Node::**advance**()

{

if (newPos == pos())

return false;

setPos(newPos);

return true;

}

//! [7]

//! [8]

QRectF Node::***boundingRect***() const

{

qreal adjust = 2;

return QRectF( -10 - adjust, -10 - adjust, 23 + adjust, 23 + adjust);

}

//! [8]

//! [9]

QPainterPath Node::***shape***() const

{

QPainterPath path;

path.addEllipse(-10, -10, 20, 20);

return path;

}

//! [9]

//! [10]

void Node::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*)

{

painter->setPen(Qt::NoPen);

painter->setBrush(Qt::darkGray);

painter->drawEllipse(-7, -7, 20, 20);

QRadialGradient gradient(-3, -3, 10);

if (option->state & QStyle::State\_Sunken) {

gradient.setCenter(3, 3);

gradient.setFocalPoint(3, 3);

gradient.setColorAt(1, QColor(Qt::yellow).light(120));

gradient.setColorAt(0, QColor(Qt::darkYellow).light(120));

} else {

gradient.setColorAt(0, Qt::yellow);

gradient.setColorAt(1, Qt::darkYellow);

}

painter->setBrush(gradient);

painter->setPen(QPen(Qt::black, 0));

painter->drawEllipse(-10, -10, 20, 20);

QFont font = painter->font();

font.setBold(true);

font.setPointSize(10);

painter->setFont(font);

painter->drawText(-2, 5, text);

}

//! [10]

//! [11]

QVariant Node::***itemChange***(GraphicsItemChange change, const QVariant &value)

{

switch (change) {

case ItemPositionHasChanged:

foreach (Edge \*edge, edgeList)

edge->adjust();

graph->itemMoved();

break;

default:

break;

};

return QGraphicsItem::*itemChange*(change, value);

}

//! [11]

//! [12]

void Node::***mousePressEvent***(QGraphicsSceneMouseEvent \*event)

{

update();

QGraphicsItem::*mousePressEvent*(event);

}

void Node::***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event)

{

update();

QGraphicsItem::*mouseReleaseEvent*(event);

}

//! [12]