Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Кафедра «Информационные технологии и автоматизированные системы»

Творческая работа

«Разработка калькулятора» и «Решение задачи Коммивояжера»

Вариант 19

Выполнил студент группы ИВТ-20-1б

Михайлов Михаил Александрович

Пермь 2021

Разработка калькулятора

**Постановка задачи:**

1. Разработать алгоритм калькулятора в соответствии с заданием.

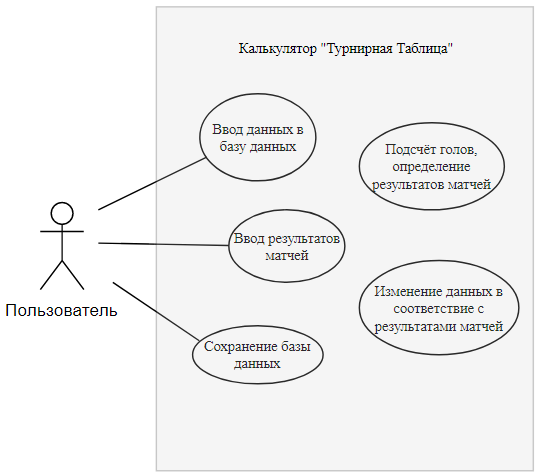
2. Реализовать алгоритм в виде программы на алгоритмическом языке С++.

3. Разработать интерфейс средствами Qt.

Необходимо ввести турнирную таблицу, в которой вводятся названия команд, результаты матчей, выполняется проверка -- встречались ли указанные команды между собой, корректируются результатов матчей, считается количество выигранных и проигранных матчей, количество ничьих, забитых и пропущенных мячей, количество набранных очков. По окончании турнира команды должны быть расположены в соответствие с набранными очками от максимума к минимуму.

На панели калькулятора должны отображаться исходные данные и результаты, операции и служебные клавиши. Калькулятор должен позволять корректировать последнее введённое число и выдавать сообщения об ошибках.

**Диаграмма вариантов:**

****

**Ввод данных в базу данных:** Ввод названий команды производится через окно ввода команды в приложении. После добавления команды возможно ручное изменение рядов таблицы, а также удаление команд из калькулятора через окно удаления команды.

|  |
| --- |
| void Add::on\_buttonBox\_accepted() |
|  | { |
|  | QString team\_name = ui->lineEdit->text(); |
|  | int row\_count = 0; |
|  | QSqlQuery \* find\_row = new QSqlQuery; |
|  | find\_row->exec("SELECT \* FROM teams WHERE No=(SELECT max(No) FROM teams)"); |
|  | find\_row->next(); |
|  | row\_count = find\_row->value(0).toInt(); |
|  | row\_count++; |
|  | qDebug() << "Last team No. is" << row\_count; |
|  | QSqlQuery \* qry = new QSqlQuery; |
|  | qry->prepare("INSERT INTO teams VALUES (:No, :team, 0, 0, 0, 0, 0, 0)"); |
|  | qry->bindValue(":No", row\_count); |
|  | qry->bindValue(":team", team\_name); |
|  | qry->exec(); |
|  | } |

**Ввод результатов матчей:** В окне «Сыграть в игру» пользователь вводит названия играющих друг с другом команд, и может изменять счёт кнопками «+» и «-». После нажатия на кнопку ввода данных, будет произведён подсчёт голов и определятся результаты матча.

|  |
| --- |
| void Match::on\_buttonBox\_accepted() |
|  | { |
|  | int row\_count = 0; |
|  | int team\_count = 0; |
|  | QString Team1 = ui->lineTeam1->text(); |
|  | QString Team2 = ui->lineTeam2->text(); |
|  | qDebug() << Team1 << Team2; |

|  |  |
| --- | --- |
| void Match::on\_goal1\_clicked() | |
|  | | { | | |
|  | | if (score1 <= 999) { | | |
|  | | score1++; | | |
|  | | ui->label\_Score1->setText(QString::number(score1)); | | |
|  | | } | | |
|  | | } | | |
| void Match::on\_goal2\_clicked() | |
|  | | { | | |
|  | | if (score2 <= 999) { | | |
|  | | score2++; | | |
|  | | ui->label\_Score2->setText(QString::number(score2)); | | |
|  | | } | | |
|  | | } | | |
|  |
| void Match::on\_retract1\_clicked() |
|  | { | |
|  | if (score1 > 0) { | |
|  | score1--; | |
|  | ui->label\_Score1->setText(QString::number(score1)); | |
|  | } | |
|  | } | |
|  |  | |
| void Match::on\_retract2\_clicked() |
|  | { | | |
|  | if (score2 > 0) { | | |
|  | score2--; | | |
|  | ui->label\_Score2->setText(QString::number(score2)); | | |
|  | } | | |
|  | } | | |

**Подсчёт голов, определение результатов матчей:** После ввода результатов программа произведёт поиск команд по названию в базе данных, затем считает данные этих команд, определит победителя, проигравшего или ничью.

|  |
| --- |
| QSqlQuery qry2; |
|  | if((qry2.exec("SELECT \* FROM teams WHERE team='"+Team1+"'")) && qry2.exec("SELECT \* FROM teams WHERE team='"+Team2+"'")) //Наличие команд? |
|  | { |
|  | while (qry2.next()) { |
|  | team\_count++; |
|  | } |
|  | if (team\_count==1) { |
|  | QSqlQuery \* find\_row = new QSqlQuery; |
|  | find\_row->exec("SELECT No FROM teams WHERE team='"+Team1+"'"); |
|  | find\_row->next(); |
|  | row\_count = find\_row->value(0).toInt(); |
|  | qDebug() << "Team found at" << row\_count; |
|  | } |
|  | if (team\_count < 1) |
|  | qDebug() << "Unable to find team"; |
|  | if (team\_count > 1) |
|  | qDebug() << "Duplicate teams found"; |
|  | } |
|  |  |
|  | int wins1, wins2, losses1, losses2, draws1, draws2, shot1, shot2, got\_shot1, got\_shot2, pts1, pts2; |
|  | QSqlQuery \* match\_logic1 = new QSqlQuery; |
|  | QSqlQuery \* match\_logic2 = new QSqlQuery; |
|  | match\_logic1->exec("SELECT \* FROM teams WHERE team='"+Team1+"'"); |
|  | match\_logic2->exec("SELECT \* FROM teams WHERE team='"+Team2+"'"); //Поиск рядов по командам |
|  | match\_logic1->next(); |
|  | match\_logic2->next(); |
|  |  |
|  | //Чтение всех данных из рядов |
|  | wins1 = match\_logic1->value(2).toInt(); |
|  | wins2 = match\_logic2->value(2).toInt(); |
|  | losses1 = match\_logic1->value(3).toInt(); |
|  | losses2 = match\_logic2->value(3).toInt(); |
|  | draws1 = match\_logic1->value(4).toInt(); |
|  | draws2 = match\_logic2->value(4).toInt(); |
|  | shot1 = match\_logic1->value(5).toInt(); |
|  | shot2 = match\_logic2->value(5).toInt(); |
|  | got\_shot1 = match\_logic1->value(6).toInt(); |
|  | got\_shot2 = match\_logic2->value(6).toInt(); |
|  | pts1 = match\_logic1->value(7).toInt(); |
|  | pts2 = match\_logic2->value(7).toInt(); |
|  | shot1 += score1; got\_shot1 += score2; shot2 += score2; got\_shot2 += score1; |
|  |  |
|  | if (score1 > score2) { |
|  | wins1++; losses2++; |
|  | pts1 += 3; |
|  | } |
|  | if (score1 < score2) { |
|  | wins2++; losses1++; |
|  | pts2 += 3; |
|  | } |
|  | if (score1 == score2) { |
|  | draws1++; draws2++; |
|  | pts1++; pts2++; |
|  | } |

**Изменение данных в соответствие с результатом матчей:** Программа обновит данные, увеличив кол-во побед и очков у победителя, кол-во поражений у проигравшего и прибавив забитые за матч мячи к общему числу забитых и пропущенных мячей. Вид базы данных упорядочивается по количеству очков, по убыванию.

|  |
| --- |
|  |
| match\_logic1->prepare("UPDATE teams SET wins = :w1, losses = :l1, draws = :d1, balls\_shot = :s1, balls\_lost = :gs1, points = :p1 WHERE team = '"+Team1+"'"); |
|  | match\_logic1->bindValue(":w1", wins1); |
|  | match\_logic1->bindValue(":l1", losses1); |
|  | match\_logic1->bindValue(":d1", draws1); |
|  | match\_logic1->bindValue(":s1", shot1); |
|  | match\_logic1->bindValue(":gs1", got\_shot1); |
|  | match\_logic1->bindValue(":p1", pts1); |
|  | match\_logic1->exec(); |
|  |  |
|  | match\_logic2->prepare("UPDATE teams SET wins = :w2, losses = :l2, draws = :d2, balls\_shot = :s2, balls\_lost = :gs2, points = :p2 WHERE team = '"+Team2+"'"); |
|  | match\_logic2->bindValue(":w2", wins2); |
|  | match\_logic2->bindValue(":l2", losses2); |
|  | match\_logic2->bindValue(":d2", draws2); |
|  | match\_logic2->bindValue(":s2", shot2); |
|  | match\_logic2->bindValue(":gs2", got\_shot2); |
|  | match\_logic2->bindValue(":p2", pts2); |
|  | match\_logic2->exec(); |
|  |  |

**Сохранение базы данных:** Сохранение изменений производится при нажатии на кнопку «сохранить изменения/обновить таблицу». Сама база данных автоматически сохраняется в папке с программой.

|  |
| --- |
| #include <main\_menu.h> |
|  | #include "set\_team.h" | |
|  | #include "ui\_set\_team.h" | |
|  | #include "QTableWidget" | |
|  | #include "match.h" | |
|  | #include "remove.h" | |
|  | #include "add.h" | |
|  |  | |
|  | Set\_team::Set\_team(QWidget \*parent) : | |
|  | QDialog(parent), | |
|  | ui(new Ui::Set\_team) | |
|  | { | |
|  | ui->setupUi(this); | |
|  |  | |
|  | this->setWindowTitle("Edit Teams"); | |
|  |  | |
|  | // this->setCentralWidget(ui->tableView); | |
|  |  | |
|  | db2=QSqlDatabase::addDatabase("QSQLITE"); | |
|  | db2.setDatabaseName("./data.db"); | |
|  | db2.open(); | |
|  |  | |
|  | QSqlQuery qry; | |
|  | qry.exec("CREATE TABLE IF NOT EXISTS teams (No int primary key, team text, wins int, losses int, draws int, balls\_shot int, balls\_lost int, points int)"); | |
|  |  | |
|  | model = new QSqlTableModel(ui->tableView); | |
|  | model->setTable("teams"); | |
|  | model->setEditStrategy(QSqlTableModel::OnManualSubmit); | |
|  | model->select(); | |
|  | model->setSort(7,Qt::DescendingOrder); | |
|  |  | |
|  | ui->tableView->setModel(model); | |
|  | ui->tableView->setColumnHidden(0, true); | |
|  | ui->tableView->resizeColumnsToContents(); | |
|  | ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection); | |
|  | ui->tableView->horizontalHeader()->setStretchLastSection(true); | |
|  |  | |
| void Set\_team::on\_submit\_clicked() | |
|  | | { |
|  | | model->submitAll(); |
|  | | } |
|  | |  |
|  | | void Set\_team::on\_undo\_clicked() |
|  | | { |
|  | | model->revertAll(); |
|  | | } |

Решение задачи Коммивояжера

**Постановка задачи:**

Задача заключается в поиске оптимального маршрута либо минимального времени, либо с наименьшим расстоянием, по которому движется Коммивояжер от пункта А с возвратом в этот же пункт. Причем, каждый город, встречающийся на пути Коммивояжера, должен быть посещен хотя бы один раз.

В задании указано использования метода ветвей и границ, чтобы найти оптимальный маршрут коммивояжёра данных методом, необходимо выполнить следующие шаги:

1. Все нули таблицы представить в самое большое число;
2. Найти минимумы строк в таблицы, и выписать в отдельный столбец;
3. Из значений ячеек каждой строки вычесть соответствующий минимум;
4. Найти минимумы столбцов в таблице и выписать в отдельную строку;
5. Из значений ячеек каждого столбца вычесть соответствующий минимум;
6. Выбрать нулевую клетку (ячейку с числом 0) с максимальной оценкой (если таких ячеек несколько, выбирается любая);
7. Получаем пару ветвей;
8. Вычислить нижнюю границу первой ветви: найти минимумы по строкам, произвести редукцию строк, то же со столбцами, вычислить нижнюю границу, как сумму предыдущей локальной нижней границы и минимумов;
9. Вычислить по принципу выше нижнюю границу второй ветви;
10. Если полный маршрут не найден, продолжить решение, перейти к шагу 8;
11. Построить полный маршрут и определить его длину:

const int N = 6;

const int INF = INT\_MAX;

int adjacensyMatrix[N][N] =

{

{ INF, 28, 13, 15, INF, 18 },

{ 28, INF, INF, 20, 21, INF },

{ 13, INF, INF, INF, 30, INF },

{ 15, 20, INF, INF, 39, 31},

{ 30, 21, INF, 39, INF, INF },

{ 18, INF, INF, 31, INF, INF }

};

int final\_min;

QString final\_path;

bool was\_drawn;

class Logic

{

public:

vector<pair<int, int>> path;

int matrix\_reduced[N][N];

int cost;

int vertex;

int level;

};

void ReduceRow(int matrix\_reduced[N][N], int row[N])

{

fill\_n(row, N, INF);

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (matrix\_reduced[i][j] < row[i])

row[i] = matrix\_reduced[i][j];

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (matrix\_reduced[i][j] != INF && row[i] != INF)

matrix\_reduced[i][j] -= row[i];

}

}

}

void ReduceColumn(int matrix\_reduced[N][N], int column[N])

{

fill\_n(column, N, INF);

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (matrix\_reduced[i][j] < column[j])

column[j] = matrix\_reduced[i][j];

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (matrix\_reduced[i][j] != INF && column[j] != INF)

matrix\_reduced[i][j] -= column[j];

}

}

}

int CostCalc(int matrix\_reduced[N][N])

{

int cost = 0;

int row[N];

ReduceRow(matrix\_reduced, row);

int column[N];

ReduceColumn(matrix\_reduced, column);

for (int i = 0; i < N; i++)

cost += (row[i] != INT\_MAX) ? row[i] : 0,

cost += (column[i] != INT\_MAX) ? column[i] : 0;

return cost;

}

Logic\* NewNode(int matrix\_parent[N][N], vector<pair<int, int>> const &path,int level, int i, int j)

{

Logic\* node = new Logic;

node->path = path;

if (level != 0)

node->path.push\_back(make\_pair(i, j));

memcpy(node->matrix\_reduced, matrix\_parent,

sizeof node->matrix\_reduced);

for (int k = 0; level != 0 && k < N; k++)

{

node->matrix\_reduced[i][k] = INF;

node->matrix\_reduced[k][j] = INF;

}

node->matrix\_reduced[j][0] = INF;

node->level = level;

node->vertex = j;

return node;

}

void PrintPath(vector<pair<int, int>> const &list)

{

for (int i = 0; i < list.size(); i++) {

// qInfo() << list[i].first + 1 << "->"

// << list[i].second + 1;

if (final\_path.isEmpty()) {

// path.append(QString::number(list[i].first+1));

final\_path.append(QString::number(list[i].first+1));

}

// path.append(QString::number(list[i].second+1));

final\_path.append("->");

final\_path.append(QString::number(list[i].second+1));

}

}

class comp {

public:

bool operator()(const Logic\* lhs, const Logic\* rhs) const

{

return lhs->cost > rhs->cost;

}

};

int solve(int adjacensyMatrix[N][N])

{

priority\_queue<Logic\*, vector<Logic\*>, comp> pq;

vector<pair<int, int>> Pair;

Logic\* root = NewNode(adjacensyMatrix, Pair, 0, -1, 0);

root->cost = CostCalc(root->matrix\_reduced);

pq.push(root);

while (!pq.empty())

{

Logic\* min = pq.top();

pq.pop();

int i = min->vertex;

if (min->level == N - 1)

{

min->path.push\_back(make\_pair(i, 0));

PrintPath(min->path);

final\_min = min->cost;

return min->cost;

}

for (int j = 0; j < N; j++)

{

if (min->matrix\_reduced[i][j] != INF)

{

Logic\* child = NewNode(min->matrix\_reduced, min->path,

min->level + 1, i, j);

child->cost = min->cost + min->matrix\_reduced[i][j]

+ CostCalc(child->matrix\_reduced);

pq.push(child);

}

}

delete min;

}

}

Таким образом, на выходе получится число, длина маршрута коммивояжёра и вектор чисел – список вершин, через которые он проходит. Это и есть ответ на задачу о коммивояжёре. Однако помимо числового ответа, необходимо реализовать графическое представление решения данной задачи. Для этого будут применены технологии OpenGL внутри Qt.

MyPoint::MyPoint(QObject \*parent)

: QObject(parent), QGraphicsItem()

{

}

MyPoint::~MyPoint()

{

}

QRectF MyPoint::boundingRect() const

{

return QRectF(0,0,100,100);

}

void MyPoint::paint(QPainter \*draw, const QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

int x1 = 20, y1 = 160,

x2 = -40, y2 = -230,

x3 = 60, y3 = 0,

x4 = -200, y4 = -50,

x5 = 170, y5 = -170,

x6 = -260, y6 = 210,

d = 75;

draw->setBrush(Qt::green);

draw->setPen(Qt::black);

draw->drawLine(x4+d/2-5, y4+d/2-5, x5+d/2-5, y5+d/2-5);

draw->drawLine(x1+d/2-5, y1+d/2-5, x4+d/2-5, y4+d/2-5);

draw->drawLine(x2+d/2-5, y2+d/2-5, x1+d/2-5, y1+d/2-5);

draw->drawLine(x1+d/2-5, y1+d/2-5, x3+d/2-5, y3+d/2-5);

draw->drawLine(x3+d/2-5, y3+d/2-5, x5+d/2-5, y5+d/2-5);

draw->drawLine(x5+d/2-5, y5+d/2-5, x2+d/2-5, y2+d/2-5);

draw->drawLine(x2+d/2-5, y2+d/2-5, x4+d/2-5, y4+d/2-5);

draw->drawLine(x4+d/2-5, y4+d/2-5, x6+d/2-5, y6+d/2-5);

draw->drawLine(x6+d/2-5, y6+d/2-5, x1+d/2-5, y1+d/2-5);

if (was\_drawn==true) {

draw->setPen(Qt::red);

draw->drawLine(x1+d/2-5, y1+d/2-5, x3+d/2-5, y3+d/2-5);

draw->drawLine(x3+d/2-5, y3+d/2-5, x5+d/2-5, y5+d/2-5);

draw->drawLine(x5+d/2-5, y5+d/2-5, x2+d/2-5, y2+d/2-5);

draw->drawLine(x2+d/2-5, y2+d/2-5, x4+d/2-5, y4+d/2-5);

draw->drawLine(x4+d/2-5, y4+d/2-5, x6+d/2-5, y6+d/2-5);

draw->drawLine(x6+d/2-5, y6+d/2-5, x1+d/2-5, y1+d/2-5);

}

draw->setPen(Qt::black);

draw->drawEllipse(QRectF(x1, y1, d, d));

draw->drawEllipse(QRectF(x2, y2, d, d));

draw->drawEllipse(QRectF(x3, y3, d, d));

draw->drawEllipse(QRectF(x4, y4, d, d));

draw->drawEllipse(QRectF(x5, y5, d, d));

draw->drawEllipse(QRectF(x6, y6, d, d));

draw->drawText(x1+d/2-2, y1+d/2+5, "1");

draw->drawText(x2+d/2-2, y2+d/2+5, "2");

draw->drawText(x3+d/2-2, y3+d/2+5, "3");

draw->drawText(x4+d/2-2, y4+d/2+5, "4");

draw->drawText(x5+d/2-2, y5+d/2+5, "5");

draw->drawText(x6+d/2-2, y6+d/2+5, "6");

Q\_UNUSED(option);

Q\_UNUSED(widget);

}