Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Курсовой проект по курсу**

**«Операционные системы»**

Группа: М80-210Б-22

Студент: Сисенов М.М.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 04.03.2024

Москва, 2024

**Постановка задачи**

**Вариант на оценку 3.**

**Формат конфигурационного файла 3 ( xml)**

На языке C\C++ написать программу, которая:

По конфигурационному файлу в формате xml принимает спроектированный DAG джобов и проверяет на корректность: отсутствие циклов, наличие только одной компоненты связанности, наличие стартовых и завершающих джоб. Структура описания джоб и их связей произвольная.

При завершении джобы с ошибкой, необходимо прервать выполнение всего DAG’а и всех запущенных джоб.

**Общий метод и алгоритм решения**

В начале обрабатывает конфигурационный файл, открываем его, считываем информацию и создаем DAG. Также получаем информацию о стартовых и конечных узлах. Затем проверяем на корректность DAG, с помощью модифицированного dfs проверяем на циклы. Далее проверяем граф на связанность( и то что компонента связанности одна), для этого нам необходимо убедиться, что наш граф слабо-связан. Затем проверяем наличие start/end узлов. После проверки добавляем джобы( относительно данных из файла), запускаем джобы. Запуск происходит путем обхода графа и функций из специального map, хранящего указатели на функции, соответсвующие именам джобов. При ошибки в узле, процесс выполнения завершается для всего Dag.

**Код программы**

**“node.h”**

#pragma once

#include <string>

#include <vector>

struct Node {

std::string job;

std::vector<int> connections;

};

**“parser.h”**

#pragma once

#include "node.h"

#include <iostream>

#include <fstream>

#include <sstream>

#include <cstring>

#include <algorithm>

#include <map>

std::string openXml(const std::string& xmlFilePath);

void parseGraph(const std::string& xmlContent, std::map<int, Node>& graph);

void parseNodes(const std::string& xmlContent, std::vector<int>& startNodes, std::vector<int>& endNodes);

**“parser.cpp”**

#include "parser.h"

std::string openXml(const std::string& xmlFilePath){

std::ifstream file(xmlFilePath);

if (!file.is\_open()) {

std::cout << "Failed to open XML file." << std::endl;

exit(-1);

}

std::stringstream buffer;

buffer << file.rdbuf();

file.close();

std::string xmlContent = buffer.str();

return xmlContent;

}

void parseGraph(const std::string& xmlContent, std::map<int, Node>& graph) {

if(xmlContent == "") return;

size\_t pos = 0;

while ((pos = xmlContent.find("<node", pos)) != std::string::npos) {

size\_t idStart = xmlContent.find("id=", pos) + strlen("id=\"");

size\_t idEnd = xmlContent.find("\"", idStart);

int nodeId = std::stoi(xmlContent.substr(idStart, idEnd - idStart));

size\_t jobStart = xmlContent.find(">", idEnd) + 1;

size\_t jobEnd = xmlContent.find("<", jobStart);

std::string job = xmlContent.substr(jobStart, jobEnd - jobStart);

graph[nodeId] = {job, std::vector<int>()};

pos = jobEnd;

}

pos = 0;

while ((pos = xmlContent.find("<connection", pos)) != std::string::npos) {

size\_t fromStart = xmlContent.find("from=", pos) + strlen("from=\"");

size\_t fromEnd = xmlContent.find("\"", fromStart);

int fromNodeId = std::stoi(xmlContent.substr(fromStart, fromEnd - fromStart));

size\_t toStart = xmlContent.find("to=", pos) + strlen("to=\"");

size\_t toEnd = xmlContent.find("\"", toStart);

int toNodeId = std::stoi(xmlContent.substr(toStart, toEnd - toStart));

graph[fromNodeId].connections.push\_back(toNodeId);

pos = toEnd;

}

}

void parseNodes(const std::string& xmlContent, std::vector<int>& startNodes, std::vector<int>& endNodes) {

if(xmlContent == "") return;

size\_t pos = 0;

while((pos = xmlContent.find("<start\_node>", pos)) != std::string::npos) {

size\_t idStart = xmlContent.find(">", pos) + 1;

size\_t idEnd = xmlContent.find("<", idStart);

int nodeId = std::stoi(xmlContent.substr(idStart, idEnd - idStart));

startNodes.push\_back(nodeId);

pos = idEnd;

}

pos = 0;

while((pos = xmlContent.find("<end\_node>", pos)) != std::string::npos) {

size\_t idStart = xmlContent.find(">", pos) + 1;

size\_t idEnd = xmlContent.find("<", idStart);

int nodeId = std::stoi(xmlContent.substr(idStart, idEnd - idStart));

endNodes.push\_back(nodeId);

pos = idEnd;

}

}

**“jobs.h”**

#pragma once

#include <iostream>

#include <functional>

#include <map>

void AAA();

void BBB();

void CCC();

void DDD();

void EEE();

void FFF();

void GGG();

void HHH();

typedef std::function<void()> jobFunc;

extern const std::map<std::string, jobFunc> functionMap;

**“jobs.cpp”**

#include "jobs.h"

const std::map<std::string, jobFunc> functionMap = {

{"AAA", AAA},

{"BBB", BBB},

{"CCC", CCC},

{"DDD", DDD},

{"EEE", EEE},

{"FFF", FFF},

{"GGG", GGG},

{"HHH", HHH}

};

void AAA() {

std::cout << "Привет, я джоб AAA!" << std::endl;

std::cout << "Я работаю!" << std::endl;

}

void BBB() {

std::cout << "Привет, я джоб BBB!" << std::endl;

int number;

std::cout << "Введите число: ";

std::cin >> number;

std::cout << "Вы ввели число: " << number << std::endl;

}

void CCC() {

std::cout << "Привет, я джоб CCC!" << std::endl;

srand(time(NULL));

int randomNum = rand() % 10 + 1;

std::cout << "Случайное число: " << randomNum << std::endl;

}

void DDD() {

std::cout << "Привет, я джоб DDD!" << std::endl;

throw std::runtime\_error("Ошибка в джобе DDD");

}

void EEE() {

std::cout << "Привет, я джоб EEE!" << std::endl;

double num1, num2;

std::cout << "Введите первое число: ";

std::cin >> num1;

std::cout << "Введите второе число: ";

std::cin >> num2;

if (num2 == 0) {

throw std::runtime\_error("Ошибка: деление на ноль!");

}

double result = num1 / num2;

std::cout << "Результат деления: " << result << std::endl;

}

void FFF() {

std::cout << "Привет, я джоб FFF!" << std::endl;

std::string input;

std::cout << "Введите строку: ";

std::cin >> input;

std::string reversed;

for (int i = input.length() - 1; i >= 0; i--) {

reversed += input[i];

}

std::cout << "Строка в обратном порядке: " << reversed << std::endl;

}

void GGG() {

std::cout << "Привет, я джоб GGG!" << std::endl;

for (int i = 0; i < 5; i++) {

std::cout << "Привет, мир!" << std::endl;

}

}

void HHH() {

std::cout << "Привет, я джоб HHH!" << std::endl;

int number;

std::cout << "Введите число: ";

std::cin >> number;

int factorial = 1;

for (int i = 1; i <= number; i++) {

factorial \*= i;

}

std::cout << "Факториал числа " << number << " равен " << factorial << std::endl;

}

**“main.cpp”**

#include "node.h"

#include "parser.h"

#include "jobs.h"

#include <iostream>

#include <algorithm>

#include <map>

#include <set>

#include <queue>

#include <stack>

void jobsAdd(std::map<std::string, jobFunc>& jobs, const std::map<int, Node>& graph) {

for (const auto& [\_,node] : graph) {

// Ищите функцию в functionMap по имени и добавляйте ее в jobs map

if (functionMap.find(node.job) != functionMap.end()) {

jobs[node.job] = functionMap.at(node.job);

}

}

}

bool hasCyclesDFS(const std::map<int, Node>& graph, int node\_id, std::set<int>& visited, std::set<int>& curPath) {

visited.insert(node\_id);

curPath.insert(node\_id);

const Node& current\_node = graph.at(node\_id);

for (int connection : current\_node.connections) {

if (visited.find(connection) == visited.end()) {

if (hasCyclesDFS(graph, connection, visited, curPath)) {

return true;

}

} else if (curPath.find(connection) != curPath.end()) {

return true;

}

}

curPath.erase(node\_id);

return false;

}

bool hasCycles(const std::map<int, Node>& graph) {

std::set<int> visited;

std::set<int> curPath;

for (const auto& [node\_id, \_] : graph) {

if (hasCyclesDFS(graph, node\_id, visited, curPath)) {

return true;

}

}

return false;

}

bool hasStartAndEndNodes(const std::vector<int>& startNodes, const std::vector<int>& endNodes) {

return !startNodes.empty() && !endNodes.empty();

}

// void printVisited(const std::set<int>& visited) {

// std::cout << "Visited nodes: ";

// for (int node\_id : visited) {

// std::cout << node\_id << " ";

// }

// std::cout << std::endl;

// }

size\_t isWeaklyConnected(const std::map<int, Node>& graph) {

if (graph.empty()) {

return true;

}

std::set<int> visited;

std::stack<int> stack;

stack.push(graph.begin()->first);

while (!stack.empty()) {

int node\_id = stack.top();

stack.pop();

if (visited.find(node\_id) == visited.end()) {

visited.insert(node\_id);

const Node& current\_node = graph.at(node\_id);

for (int connection : current\_node.connections) {

stack.push(connection);

}

}

}

return visited.size();

}

bool isGraphConnected(const std::map<int, Node>& graph) {

std::map<int, Node> undirected\_graph;

for (const auto& [node\_id, node] : graph) {

for (int connection : node.connections) {

undirected\_graph[connection].connections.push\_back(node\_id);

undirected\_graph[node\_id].connections.push\_back(connection);

}

}

return isWeaklyConnected(undirected\_graph) == graph.size();

}

void jobsRun(const std::map<int, Node>& graph, const std::vector<int>& startNodes, const std::map <std::string, jobFunc>& jobs) {

std::set<int> visited;

std::queue<int> queue;

for (int start\_node : startNodes) {

queue.push(start\_node);

visited.insert(start\_node);

}

while (!queue.empty()) {

int node\_id = queue.front();

queue.pop();

const Node& current\_node = graph.at(node\_id);

std::string job\_name = current\_node.job;

try {

jobs.at(job\_name)();

} catch (const std::exception& e) {

std::cerr << e.what() << std::endl;

return;

}

for (int connection : current\_node.connections) {

if (visited.find(connection) == visited.end()) {

queue.push(connection);

visited.insert(connection);

}

}

}

}

int main() {

std::map<std::string, jobFunc> jobs;

std::map<int, Node> graph;

std::vector<int> startNodes;

std::vector<int> endNodes;

std::string dagXml = openXml("../dags/dag.xml");

parseGraph(dagXml, graph);

parseNodes(dagXml, startNodes, endNodes);

bool has\_cycles = hasCycles(graph);

bool has\_connection = isGraphConnected(graph);

bool has\_start\_and\_end\_nodes = hasStartAndEndNodes(startNodes, endNodes);

if (has\_cycles) {

std::cout << "Has cycles: Yes" << std::endl;

return 1;

}

else{

std::cout << "Has cycles: No" << std::endl;

}

if (!has\_start\_and\_end\_nodes) {

std::cout << "Has start and end nodes: No" << std::endl;

return 1;

}

else{

std::cout << "Has start and end nodes: Yes" << std::endl;

}

if (!has\_connection) {

std::cout << "Has only 1 connection: No" << std::endl;

return 1;

}

else{

std::cout << "Has only 1 connection: Yes" << std::endl;

}

jobsAdd(jobs, graph);

jobsRun(graph, startNodes, jobs);

// for (const auto& [node\_id, node\_data] : graph) {

// std::cout << "Node ID: " << node\_id << ", Job: " << node\_data.job << ", Connections: ";

// for (int connection : node\_data.connections) {

// std::cout << connection << " ";

// }

// std::cout << std::endl;

// }

// for (int start\_node : startNodes) {

// std::cout << "Start node: " << start\_node << std::endl;

// }

// for (int end\_node : endNodes) {

// std::cout << "End node: " << end\_node << std::endl;

// }

return 0;

}

**“dag.xml”**

<dag>

<nodes>

<node id="1">AAA</node>

<node id="2">BBB</node>

<node id="3">CCC</node>

<node id="4">DDD</node>

<node id="5">EEE</node>

<node id="6">FFF</node>

</nodes>

<connections>

<connection from="1" to="4"></connection>

<connection from="2" to="4"></connection>

<connection from="3" to="5"></connection>

<connection from="4" to="6"></connection>

<connection from="5" to="6"></connection>

</connections>

<start\_nodes>

<start\_node>1</start\_node>

<start\_node>2</start\_node>

<start\_node>3</start\_node>

</start\_nodes>

<end\_nodes>

<end\_node>6</end\_node>

</end\_nodes>

</dag>

**Протокол работы программы**

**Тестирование:**

marat@marat-RedmiBook:~/VUZ/OS/OS\_lab2/build$ ./main

Has cycles: No

Has start and end nodes: Yes

Has only 1 connection: Yes

Привет, я джоб AAA!

Я работаю!

Привет, я джоб BBB!

Введите число: 20

Вы ввели число: 20

Привет, я джоб CCC!

Случайное число: 7

Привет, я джоб DDD!

Ошибка в джобе DDD

**Вывод**

Во время выполнения лабораторной работы смог разобраться в составление и парсинге xml файла. Также научился создавать на его основе DAG джобов, обрабатывать его и запускать.