Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по предмету «Операционные системы»**

Студент: Корнев Максим Сергеевич

Группа: М8О–212Б–22

Вариант: 33

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

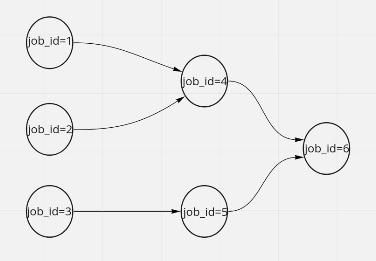
Подпись: \_\_\_\_\_\_\_\_\_\_

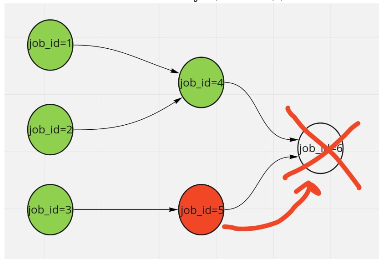
Москва, 2023.

**Постановка задачи**

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

1. По конфигурационному файлу в формате yaml, json или ini 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.

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



\* DAG - Directed acyclic graph. Направленный ациклический граф.

\*\* Джоб(Job) – процесс, который зависит от результата выполнения других процессов (если он не стартовый), которые исполняются до него в DAG, и который порождает данные от которых может быть зависят другие процессы, которые исполняются после него в DAG (если он не завершающий).

*Вариант 33:*

Формат конфигурационного файла — yaml.

**Общая информация о yaml**

**YAML** (рекурсивный акроним англ. *«****Y****AML* ***A****in't* ***M****arkup* ***L****anguage»* — «YAML — не язык разметки») — дружественный формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования.

Синтаксис YAML минималистичен, особенно по сравнению с XML-синтаксисом. В спецификации указывают, что большое влияние оказал стандарт RFC 822.

**Реализация**

**test1.yaml:**

jobs:

 job1:

   dependencies: []

 job2:

   dependencies: [job1]

 job3:

   dependencies: [job1]

 job4:

   dependencies: [job2]

 job5:

   dependencies: [job3]

 job6:

   dependencies: [job4, job5]

**test2.yaml:**

jobs:

 job1:

   dependencies: []

 job2:

   dependencies: []

 job3:

   dependencies: [job1, job2]

 job4:

   dependencies: [job3]

 job5:

   dependencies: [job3]

 job6:

   dependencies: [job4, job5]

 job6:

   dependencies: [job6]

**test3.yaml:**

jobs:

 job1:

   dependencies: [job2]

 job2:

   dependencies: [job3]

 job3:

   dependencies: [job4]

 job4:

   dependencies: [job5]

 job5:

   dependencies: [job6]

 job6:

   dependencies: []

**test4.yaml:**

jobs:

 job1:

   dependencies: []

 job2:

   dependencies: []

 job3:

   dependencies: []

 job4:

   dependencies: [job1, job2]

 job5:

   dependencies: [job3]

 job6:

   dependencies: [job4, job5]

**main.cpp**

#include <iostream>

#include <fstream>

#include <yaml-cpp/yaml.h>

#include <set>

#include <functional>

#include <vector>

#include <string>

#include <sys/wait.h>

#include <queue>

#include <unordered\_set>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

void processJob(const std::string& job, const std::unordered\_map<std::string, std::vector<std::string>>& dependencies,

                std::unordered\_set<std::string>& visitedJobs) {

    if (visitedJobs.size() == dependencies.size()) {

        return;

    }

    if (visitedJobs.count(job) == 0) {

        visitedJobs.insert(job);

    }

    if (dependencies.count(job) > 0) {

        const std::vector<std::string>& currentDependencies = dependencies.at(job);

        for (const auto& dependency : currentDependencies) {

            if (visitedJobs.count(dependency) == 0) {

                visitedJobs.insert(dependency);

                if (visitedJobs.size() == dependencies.size()) {

                    std::cout << "Job done: " << dependency << std::endl;

                    sleep(1);

                } else {

                    processJob(dependency, dependencies, visitedJobs);

                }

            }

        }

    }

    std::cout << "Job done: " << job << std::endl;

    sleep(1);

}

bool hasCycle(const YAML::Node& jobs, const std::string& currentJob, std::set<std::string>& visited,

              std::set<std::string>& recursionStack) {

    visited.insert(currentJob);

    recursionStack.insert(currentJob);

    const YAML::Node& dependencies = jobs[currentJob]["dependencies"];

    for (const auto& dependency : dependencies) {

        const std::string& dependencyJob = dependency.as<std::string>();

        if (recursionStack.count(dependencyJob)) {

            return true;

        }

        if (!visited.count(dependencyJob) && hasCycle(jobs, dependencyJob, visited, recursionStack)) {

            return true;

        }

    }

    recursionStack.erase(currentJob);

    return false;

}

bool isValidDAG(const YAML::Node& jobs) {

    std::set<std::string> visited;

    std::set<std::string> recursionStack;

    for (const auto& job : jobs) {

        if (!visited.count(job.first.as<std::string>()) &&

            hasCycle(jobs, job.first.as<std::string>(), visited, recursionStack)) {

            return false;

        }

    }

    return true;

}

bool hasOnlyOneComponent(const YAML::Node& jobs) {

    std::unordered\_map<std::string, std::vector<std::string>> adjacencyList;

    std::set<std::string> visited;

    // Строим список смежности

    for (const auto& job : jobs) {

        const std::string &jobName = job.first.as<std::string>();

        const YAML::Node &dependencies = job.second["dependencies"];

        for (const auto& dependency : dependencies) {

            adjacencyList[dependency.as<std::string>()].push\_back(jobName);

            adjacencyList[jobName].push\_back(dependency.as<std::string>());

        }

    }

    int componentCount = 0;

    // Обходим граф и считаем количество компонент связности

    for (const auto& job : jobs) {

        const std::string &jobName = job.first.as<std::string>();

        if (visited.count(jobName) == 0) {

            std::set<std::string> component;

            std::queue<std::string> q;

            q.push(jobName);

            visited.insert(jobName);

            while (!q.empty()) {

                std::string currentJob = q.front();

                q.pop();

                component.insert(currentJob);

                for (const auto& neighbor : adjacencyList[currentJob]) {

                    if (visited.count(neighbor) == 0) {

                        q.push(neighbor);

                        visited.insert(neighbor);

                    }

                }

            }

            componentCount++;

        }

    }

    return componentCount > 1;

}

bool hasStartAndEndJobs(const YAML::Node &jobs) {

    std::set<std::string> startJobs;

    std::set<std::string> endJobs;

    for (const auto& job : jobs) {

        const std::string &jobName = job.first.as<std::string>();

        const YAML::Node &dependencies = job.second["dependencies"];

        if (dependencies.IsNull() || dependencies.size() == 0) {

            startJobs.insert(jobName);

        }

        for (const auto& dependency : dependencies) {

            endJobs.erase(dependency.as<std::string>());

        }

        endJobs.insert(jobName);

    }

    return !startJobs.empty() && !endJobs.empty();

}

int main(int argc, char const \*argv[]) {

    std::string fileName = argv[1];

    std::ifstream file(fileName);

    YAML::Node data = YAML::Load(file);

    YAML::Node jobs = data["jobs"];

    if (!isValidDAG(jobs)) {

        throw std::runtime\_error("DAG contain cycle");

    }

    if (hasOnlyOneComponent(jobs)) {

        throw std::runtime\_error("DAG has more then one connectivity component");

    }

    if (!hasStartAndEndJobs(jobs)) {

        throw std::runtime\_error("DAG does not have a start and end job");

    }

    std::cout << "DAG is valid" << std::endl;

    sleep(1);

    int n = jobs.size();

    std::vector<std::string> jobNames;

    for (const auto& job : jobs) {

        jobNames.push\_back(job.first.as<std::string>());

    }

    std::unordered\_map<std::string, std::vector<std::string>> dependencies\_map;

    std::unordered\_set<std::string> visitedJobs;

    for (const auto& job : jobs) {

        const std::string &jobName = job.first.as<std::string>();

        std::vector<std::string> dependencies;

        for (const auto& dependency : job.second["dependencies"]) {

            dependencies.push\_back(dependency.as<std::string>());

        }

        dependencies\_map[jobName] = dependencies;

    }

    for (const std::string& job : jobNames) {

        processJob(job, dependencies\_map, visitedJobs);

    }

    std::cout << "All jobs done" << std::endl;

    return 0;

}

**Пример работы**

radioactive@DESKTOP-RNP2IGB:/mnt/d/labs/os\_CP$ ./main test1.yaml

DAG is valid

Job done: job1

Job done: job2

Job done: job3

Job done: job4

Job done: job5

Job done: job6

All jobs done

radioactive@DESKTOP-RNP2IGB:/mnt/d/labs/os\_CP$ ./main test2.yaml

DAG is valid

Job done: job1

Job done: job2

Job done: job3

Job done: job4

Job done: job5

Job done: job6

All jobs done

radioactive@DESKTOP-RNP2IGB:/mnt/d/labs/os\_CP$ ./main test3.yaml

DAG is valid

Job done: job6

Job done: job5

Job done: job4

Job done: job3

Job done: job2

Job done: job1

All jobs done

radioactive@DESKTOP-RNP2IGB:/mnt/d/labs/os\_CP$ ./main test4.yaml

DAG is valid

Job done: job1

Job done: job2

Job done: job3

Job done: job4

Job done: job5

Job done: job6

All jobs done

radioactive@DESKTOP-RNP2IGB:/mnt/d/labs/os\_CP$

**Вывод**

В ходе выполнения курсового проекта я познакомился с языком разметки yaml и библиотекой yamlcpp для языка cpp, с помощью которой смог реализовать программу, проверяющую на корректность DAG (направленный ациклический граф), поданный в формате yaml. В процессе обработки графа я уведомляю пользователя о допущенных ошибках, что позволяет достоверно проверить корректность DAG.