**Федеральное государственное автономное образовательное** **учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»**

**Факультет компьютерных наук**

**Департамент программной инженерии**

**Дисциплина:** Архитектура Вычислительных Систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ПРОГРАММЕ**

**Вариант 24**

**Группа БПИ191**

**Студент:** Удачин Данил Андреевич

**Преподаватель:** Легалов Александр Иванович

**Москва 2020**

**Содержание**

[**Постановка задачи и условие** 3](#_Toc56629839)

[**Решение задачи** 3](#_Toc56629840)

[**Текст программы** 4](#_Toc56629841)

[**Тестирование программы** 6](#_Toc56629842)

[**Список использованных источников** 8](#_Toc56629843)

# **Постановка задачи и условие**

Задача о Пути Кулака. На седых склонах Гималаев стоят два древних буддистских монастыря: Гуань-Инь и Гуань-Янь. Каждый год в день сошествия на землю боддисатвы Араватти монахи обоих монастырей собираются на совместное празднество и показывают свое совершенствование на Пути Кулака. Всех соревнующихся монахов разбивают на пары, победители пар бьются затем между собой и так далее, до финального поединка. Монастырь, монах которого победил в финальном бою, забирает себе на хранение статую боддисатвы. Реализовать многопоточное приложение, определяющего победителя. В качестве входных данных используется массив, в котором хранится количество энергии Ци каждого монаха. При решении использовать принцип дихотомии.

Разработать программу с применением OpenMP.

# **Решение задачи**

Информация подаётся на вход из файла **input.txt**, лежащего рядом с исполняемым файлом.

Формат входных данных:

\*\*\*

**a b**

**vector<a>**

**vector<b>**

\*\*\*

Где,

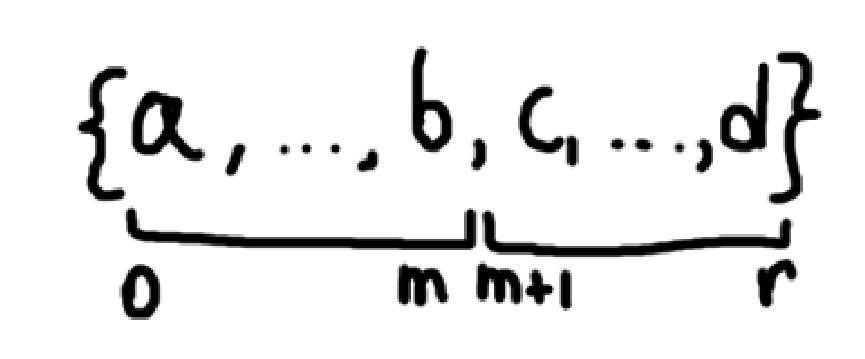
\*\*\* - начало и конец файла;

**a** и **b** – целочисленные значения, количество монахов в каждой команде;

**vector<(a|b)>** - вектор длины **a** или **b**, где каждый элемент вектора – вещественное число (уровень энергии Ци каждого бойца), разделённых пробелом.

На вход подаются входные данные и записываются в одномерную матрицу. Каждый элемент матрицы – класс **Monk** с полями **id**, **power** и **team**.

Алгоритм при каждой итерации перемешивает весь массив данных для выбора случайного противника. Алгоритм продолжает работу до тех пор, пока во всём массиве не останется 1 единственный элемент.



*Рисунок 1 – иллюстрация разбиения массива данных на 2 части.*

В основе работы алгоритма лежит принцип дихотомии с использованием 2 потоков. Массив данных разбивается на 2 части. Каждый поток обрабатывает свою часть, определяя победителя на данном отрезке. Функция выполняется рекурсивно для соблюдения принципа дихотомии. Проигравшие бойцы удаляются из набора данных. В конце работы алгоритма обе части соединяются воедино и возвращаются из функции.

# **Текст программы**

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <string>

#include <fstream>

#include <map>

#include <algorithm>

#include <unordered\_set>

#include <omp.h>

// Assuming Yin team: 0, Yang team: 1.

struct Monk {

bool team{ false };

double power{ 0 };

int id{ 0 };

bool operator == (const Monk& o) const {

return team == o.team &&

power == o.power &&

id == o.id;

}

};

struct Hash {

size\_t operator() (const Monk& o) const {

return o.id;

}

};

#define TeamMatrix std::vector<Monk>

void getInput(std::istream& inp, TeamMatrix& matrix) {

std::size\_t sizeYin, sizeYang;

inp >> sizeYin >> sizeYang;

matrix.resize(sizeYin + sizeYang);

for (auto i = 0; i < sizeYin; i++) {

matrix[i].team = false;

matrix[i].id = i;

inp >> matrix[i].power;

}

for (auto i = 0; i < sizeYang; i++) {

matrix[sizeYin + i].team = true;

matrix[i].id = sizeYin + i;

inp >> matrix[sizeYin + i].power;

}

}

TeamMatrix merge(TeamMatrix& leftHalf, TeamMatrix& rightHalf) {

std::unordered\_set<Monk, Hash> result;

for (const auto& leftOne : leftHalf)

result.insert(leftOne);

for (const auto& rightOne : rightHalf)

result.insert(rightOne);

TeamMatrix answer;

for (const auto& a : result) {

answer.push\_back(a);

}

return answer;

}

// Alternative implementation

//std::recursive\_mutex mutex;

struct Context {

int l, r;

TeamMatrix& matrix;

std::map<int, int>& enemies;

TeamMatrix result;

};

TeamMatrix eliminate(int l, int r, TeamMatrix& matrix, std::map<int, int>& enemies);

void wrapper(Context& context) {

context.result = eliminate(context.l, context.r, context.matrix, context.enemies);

}

TeamMatrix eliminate(int l, int r, TeamMatrix& matrix, std::map<int, int>& enemies) {

if (l == r) {

// Alternative implementation

//mutex.lock();

//auto self = matrix[l];

//auto enemy = matrix[enemies[l]];

//mutex.unlock();

Monk self;

Monk enemy;

#pragma omp critical

{

self = matrix[l];

enemy = matrix[enemies[l]];

}

if (self.power == enemy.power)

return l < enemies[l] ? TeamMatrix{ self } : TeamMatrix{ enemy };

else

return self.power > enemy.power ? TeamMatrix{ self } : TeamMatrix{ enemy };

}

int mid = (l + r) / 2;

Context a{ l, mid, matrix, enemies, {} },

b{ mid + 1, r, matrix, enemies, {} };

// Alternative implementation

//std::thread t1(wrapper, std::ref(a)),

// t2(wrapper, std::ref(b));

//t1.join();

//t2.join();

#pragma omp parallel num\_threads(2)

{

if (omp\_get\_thread\_num() == 0) {

wrapper(std::ref(a));

} else {

wrapper(std::ref(b));

}

}

return merge(a.result, b.result);

}

int main() {

TeamMatrix matrix;

std::fstream fin("input.txt");

getInput(fin, matrix);

std::cout << "The data has been received. Let's start the tournament ..." << std::endl;

while (matrix.size() != 1) {

std::map<int, int> enemies;

std::random\_shuffle(matrix.begin(), matrix.end());

for (int i = 0; i < matrix.size(); i += 2) {

enemies[i] = i + 1;

enemies[i + 1] = i;

}

Monk\* last = nullptr;

if (matrix.size() % 2 != 0) {

last = &matrix.back();

matrix.pop\_back();

}

auto result = eliminate(0, matrix.size() - 1, matrix, enemies);

if (last != nullptr) {

result.push\_back(\*last);

}

matrix = result;

}

auto result = matrix.back();

if (result.team)

std::cout << "Team Kuan Yang wins and claims the Bodhisattva statue!" << std::endl;

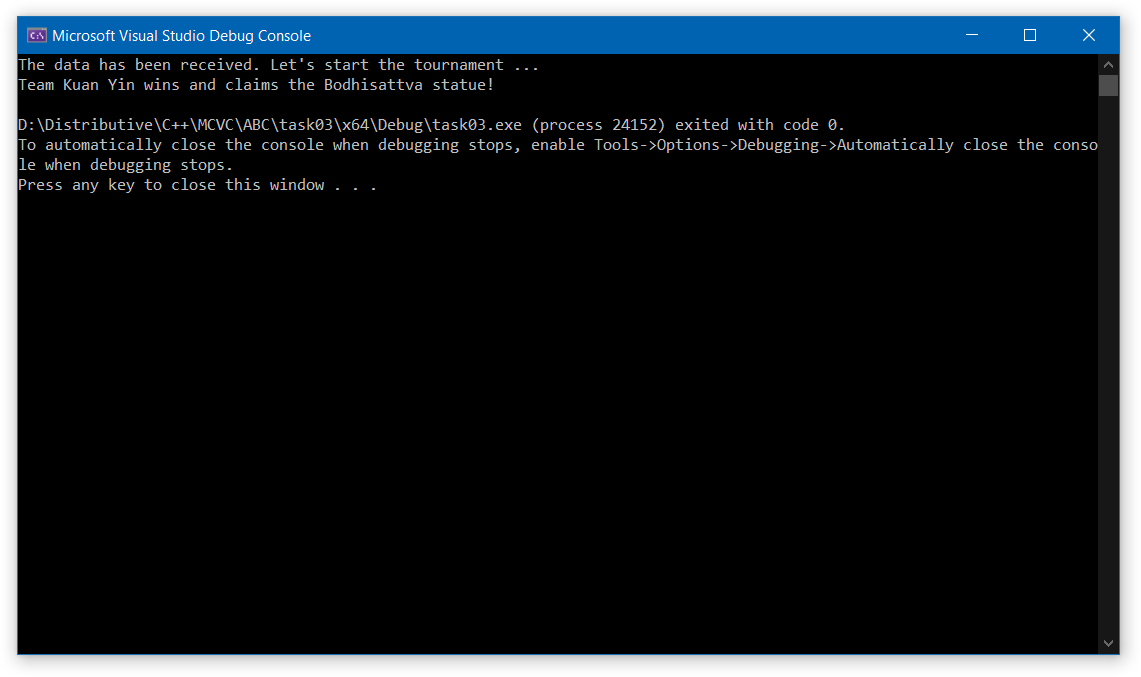
else

std::cout << "Team Kuan Yin wins and claims the Bodhisattva statue!" << std::endl;

return 0;

}

# **Тестирование программы**



*Рисунок 2 – результат работы программы с выводом в консоль.*

Работа программы продемонстрирована на рис. 2. Входные данные всегда корректны при соблюдении условий формата входных данных.

# **Список использованных источников**

1. Киберленинка, «Принцип дихотомии в программировании» // URL: https://cyberleninka.ru/article/n/metod-dihotomicheskogo-programmirovaniya (дата обращения 16.11.2020)
2. С++ Documentation // URL: https://en.cppreference.com/w/ (дата обращения 16.11.2020)
3. OpenMP Documentation // URL: https://www.openmp.org// (дата обращения 02.12.2020)