**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчет по лабораторной работе №6**

**по курсу «Структуры и алгоритмы обработки данных»**

**на тему**

**«Решение комбинаторных задач с помощью эвристических алгоритмов»**

**Вариант №8**

Выполнил: студент

группы КТбо2-1: Самардак А.В

Проверил:

Доцент каф. МОП ЭВМ Дроздов С.Н

# **8 ВАРИАНТ ЗАДАНИЯ**

**2.1 Общие требования**

Во всех вариантах задания требуется написать и отладить программу на произвольном языке программирования, решающую методом перебора комбинаторную задачу, выбранную в соответствии с вариантом задания. Желательно учесть все соображения, позволяющие сократить перебор.

Программа должна быть испытана как на конкретных примерах с заранее известным ответом (чтобы убедиться в правильности работы), так и на достаточном количестве случайных задач, позволяющем сделать выводы о среднем времени решения и его зависимости от размерности задачи. Если есть такая возможность, желательно сравнить различные модификации алгоритма, исследовать его работу на разных подклассах задач.

Все числовые параметры всех задач можно считать целочисленными.

Во всех вариантах заданий предложены такие задачи, для которых неизвестны алгоритмы решения, принципиально более быстрые, чем перебор. Не рекомендуется пытаться найти такой алгоритм в процессе выполнения лабораторной работы, иначе вы зря потеряете время.

Отчет о работе должен включать вариант задания, текст программы и таблицу результатов работы программы..

**ЦЕЛЬ РАБОТЫ**

Цель настоящей работы – изучение эвристических алгоритмов и способов их разработки, а также сравнительная оценка использования эвристических и переборных алгоритмов для решения комбинаторных задач.

**2.2 Индивидуальное задание, вариант № 8**

Даны N деталей, каждая из которых должна быть обработана на станке A, затем на станке B, затем на станке C. Каждый станок может в данный момент времени обрабатывать только одну деталь. Если нужный станок занят, то другие детали могут ожидать его освобождения. Для каждой детали известны длительности ее обработки на каждом станке: TA i , TB i , TC i . Требуется найти такой порядок запуска деталей на обработку (т.е. найти такую перестановку номеров деталей), при котором длительность обработки всего комплекта деталей минимальна.

**Таблица**

В следующей таблице приведены сравнительные данные о решении серии задач с помощью эвристического алгоритма и алгоритма перебора. Симметричные матрицы расстояний заполнялись случайными целыми значениями.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Эвристика | | Перебор | | Оценка эвристики | |
| N | Среднее время | Кратч.путь | Среднее время | Кратч.путь (средний) | % совпадений | Качество планов |
| 3 | 0.0 | 263 | 0.0 | 270 | 102 | 1 |
| 4 | 1.9 | 330 | 2.4 | 325 | 98 | 0.98 |
| 5 | 18.5 | 420 | 17.6 | 395 | 94 | 0.94 |
| 6 | 123 | 433 | 126 | 464 | 107 | 1.07 |
| 7 | 1091 | 712 | 1119 | 508 | 71 | 0.71 |
| 8 | 10295 | 841 | 10362 | 545 | 64 | 0.64 |
| 9 | 108173 | 862 | 108373 | 594 | 68 | 0.68 |

Под процентом совпадений в этой таблице понимается процент задач данной размерности, для которых эвристический алгоритм нашел кратчайший путь такой же длины, как и перебор. Под качеством планов понимается отношение средней длины кратчайшего пути, найденного перебором, к длине, полученной эвристикой.

**Листинг программы на языке С++**

**Main.h**

#include <iostream>

#include "Console.h"

int main()

{

setlocale(LC\_ALL, "Russian");

Console().start();

return 0;

}

**Console.h**

#pragma once

#include <iostream>

#include <vector>

class Console

{

public:

void start();

private:

long long get\_data();

void push\_vector(int count);

long long computation(int count);

void print(int result, int time) const;

std::vector<long long> \_machine\_a;

std::vector<long long> \_machine\_b;

std::vector<long long> \_machine\_c;

std::vector<long long> \_remaining\_time = { 0, 0, 0 };

};

**Console.cpp**

#include "Console.h"

void Console::start()

{

long long count = get\_data();

push\_vector(count);

unsigned int start = clock();

long long result = computation(count);

unsigned int stop = clock();

print(result, stop - start);

}

long long Console::get\_data() const

{

long long data;

std::cin >> data;

return data;

}

void Console::push\_vector(int count)

{

for (long long i = 0; i < count; i++)

{

\_machine\_a.push\_back(get\_data());

\_machine\_c.push\_back(get\_data());

\_machine\_b.push\_back(get\_data());

}

}

long long Console::computation(int count)

{

long long result = 0;

for (long long i = 0; i < count; i++)

{

if (\_remaining\_time[0] != 0 || \_remaining\_time[1] != 0 || \_remaining\_time[2] != 0)

{

long long A = \_remaining\_time[0] - \_machine\_a[i];

long long B = \_remaining\_time[1] - \_machine\_b[i];

long long C = \_remaining\_time[2] - \_machine\_c[i];

if (A > B && B > C)

{

if (\_remaining\_time[0] >= \_machine\_a[i])

{

\_remaining\_time[0] -= \_machine\_a[i];

}

else

{

\_machine\_a[i] -= \_remaining\_time[0];

\_remaining\_time[0] = 0;

result += \_machine\_a[i];

\_remaining\_time[1] += \_machine\_a[i];

\_remaining\_time[2] += \_machine\_a[i];

}

}

else if (B > A && A > C)

{

if (\_remaining\_time[1] >= \_machine\_b[i])

{

\_remaining\_time[1] -= \_machine\_b[i];

}

else

{

\_machine\_b[i] -= \_remaining\_time[1];

\_remaining\_time[1] = 0;

result += \_machine\_b[i];

\_remaining\_time[0] += \_machine\_b[i];

\_remaining\_time[2] += \_machine\_b[i];

}

}

else if (C > B && B > A)

{

if (\_remaining\_time[2] >= \_machine\_c[i])

{

\_remaining\_time[2] -= \_machine\_c[i];

}

else

{

\_machine\_c[i] -= \_remaining\_time[2];

\_remaining\_time[2] = 0;

result += \_machine\_c[i];

\_remaining\_time[1] += \_machine\_c[i];

\_remaining\_time[0] += \_machine\_c[i];

}

}

}

else

{

if (\_machine\_a[i] < \_machine\_b[i] && \_machine\_b[i] < \_machine\_c[i])

{

result += \_machine\_a[i];

\_remaining\_time[1] += \_machine\_a[i];

\_remaining\_time[2] += \_machine\_a[i];

}

else if (\_machine\_b[i] < \_machine\_a[i] && \_machine\_a[i] < \_machine\_c[i])

{

result += \_machine\_b[i];

\_remaining\_time[0] += \_machine\_b[i];

\_remaining\_time[2] += \_machine\_b[i];

}

else if (\_machine\_c[i] < \_machine\_b[i] && \_machine\_b[i] < \_machine\_a[i])

{

result += \_machine\_c[i];

\_remaining\_time[1] += \_machine\_c[i];

\_remaining\_time[0] += \_machine\_c[i];

}

}

}

return result;

}

void Console::print(int result, int time) const

{

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

std::cout << "Время: " << time << "мс" << std::endl;

}