Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет

телекоммуникаций и информатики»

**ОТЧËТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

по дисциплине

«Теория функционирования распределённых вычислительных систем»

Выполнил:

Студент гр. ИС-641

Меняйлов В.О

Проверила:

Ассистент Кафедры ВС

Ткачева Т.А.

Новосибирск 2020

Оглавление

[**1.Цель роботы** 3](#_Toc37356802)

[**2.Теория** 4](#_Toc37356803)

[**2.1Определения основных параметров и основные формулы** 4](#_Toc37356804)

[**3.Ход работы** 6](#_Toc37356805)

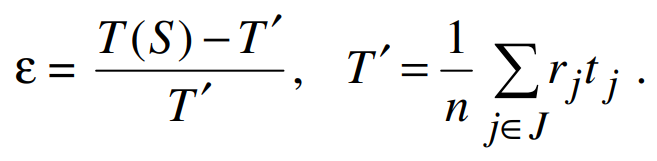
[**4.Заключение** 7](#_Toc37356806)

[**5. Листинг** 8](#_Toc37356807)

# **1.Цель роботы**

1. Написать программу, реализующую алгоритмы NFDH и FFDH [1, 2] для приближенного решения задачи (1) – (5).

В качестве входных параметров программа получает имя файла с набором задач, количество n ЭМ в системе и название алгоритма (NFDH или FFDH). Результат работы программы: расписание S решения задач, значение T(S) целевой функции, отклонение ε значения целевой функции от её нижней границы T′, время t выполнения алгоритма в секундах.



Упорядочивание задач в алгоритмах NFDH и FFDH выполнять сортировкой подсчетом (Counting Sort).

Для реализации алгоритма FFDH с вычислительной сложностью O(nlogn) рекомендуется использовать бинарное дерево турнира (tournament tree, max winner tree). Каждый лист дерева – это уровень упаковки, а значение листа – это количества свободных ЭМ на уровне. Каждый внутренний узел дерева содержит максимальное из значений левого и правого дочерних узлов. В таком дереве поиск первого подходящего уровня (First Fit) выполняется за время O(logn).

2. Исследовать время выполнения алгоритмов в зависимости от количества m задач в наборе.

Сформировать 10 наборов задач с m = 500, 1000, …, 5000; параметры задач генерировать как равномерно распределенные псевдослучайные числа rj ∈{1, 2, …, n}, tj ∈{1, 2, …, 100}. Рассмотреть случаи n = 1024, 4096.

Ответить на вопросы:

• Какова вычислительная сложность алгоритмов?

• Как зависит время работы алгоритмов от значения параметра m?

• Как зависит время работы алгоритмов от значения параметра n?

3. Провести сравнительный анализ значений целевой функции от расписаний, формируемых алгоритмами.

Сформировать 10 наборов задач (m = 500, 1000, …, 5000); параметры задач генерировать как равномерно распределенные псевдослучайные числа rj ∈{1, 2, …, n}, tj ∈{1, 2, …, 100}. Во всех 10 экспериментах n = 1024. По результатам экспериментов построить оценки математического ожидания и среднеквадратического отклонения случайной величины ε.

Ответить на вопрос: какой из алгоритмов, на рассмотренных наборах задач, формировал более точные расписания?

# **2.Теория**

## **2.1Определения основных параметров и основные формулы**

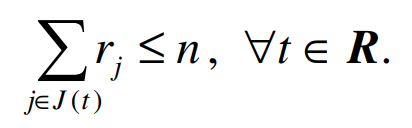
Имеется распределенная вычислительная система (ВС) укомплектованная n элементарными машинами (ЭМ). Задан набор из m параллельных задач. Каждая задача j ∈ J = {1, 2, …, m} характеризуется временем tj решения и количеством rj элементарных машин необходимых для неё.

Требуется построить расписание S решения параллельных задач на распределенной ВС. Для каждой задачи необходимо определить момент времени τj начала решения её ветвей и их распределение по элементарным машинам. Пусть xji ∈ C = {1, 2, …, n} – номер ЭМ, на которую распределена ветвь i ∈ {1, 2, …, rj} задачи j ∈ J.

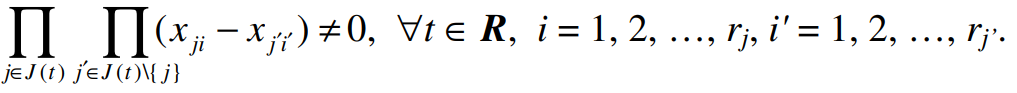
Обозначим через J(t) = {j ∈ J | τj ≤ t ≤ τj + tj} множество задач, решаемых на распределенной ВС в момент времени t.

Будем называть расписание S = (τ1, τ2, …, τm; x11, x12, …, 1 1 r x , …, xm1, xm2, …, mrm x ) допустимым, если оно удовлетворяет ограничениям.

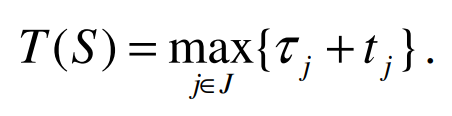
1. В любой момент времени на ресурсах распределенной ВС решается не более n ветвей параллельных задач:



1. Ветви параллельных задач решаются на разных элементарных машинах:



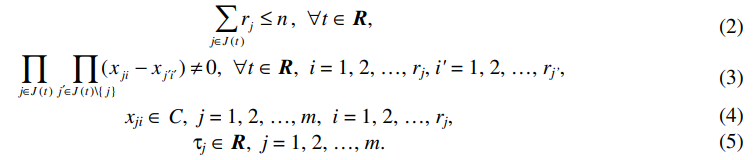
Обозначим через Ω множество допустимых расписаний. В качестве показателя оптимальности расписаний будем использовать время T(S) окончания решения последней задачи



Итак, требуется найти допустимое расписание S ∈ Ω, доставляющее минимум целевой функции T(S). Формально



При ограничениях:



Задача (1) − (5) относится к дискретной оптимизации и является трудноразрешимой.

Один из подходов к приближенному решению задачи (1) − (5) основан на её сведении к задаче двумерной упаковки прямоугольников в полуограниченную полосу (2D Strip Packing, 2DSP). Параллельная задача j ∈ J представляется в виде прямоугольника шириной rj и высотой tj условных единиц. Ширина полосы – n условных единиц, высота не ограничена. Требуется упаковать прямоугольники без их вращений и пересечений в полосу так, чтобы высота упаковки была минимальной. На рис. 1 приведен пример упаковки 5 прямоугольников (задач) в полосу шириной 10 условных единиц (элементарных машин).

Задача 1 запускается на решение в нулевой момент времени и использует элементарные машины 1 − 5, задача 4 начинает решаться в момент времени 2 и использует ЭМ 8, 9, 10. Значение целевой функции T(S) = 8.



# **3.Ход работы**

Была написана программа, которая реализует алгоритмы NFDH и FFDH. Так же были проведены эксперименты для сравнительного анализа и времени выполнения некоторой задачи. Задача подразумевает собой генерировать равномерно распределенные псевдослучайные числа. При разных n – 1024, 4096.

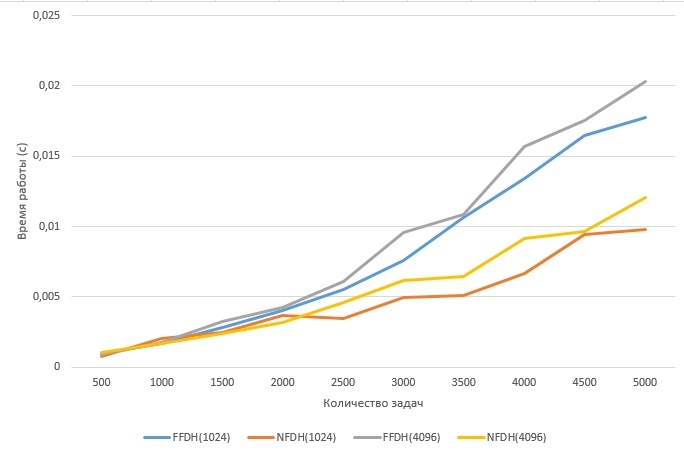


Рисунок 1График выполнения задач алгоритмами

# **4.Заключение**

В результате лабораторной работы были реализованы алгоритмы NFDH и FFDH. Так же проведены экспериментальные нагрузки на каждый из этих алгоритмов для выявления сравнительной оценки ( времени выполнения).

FFDH упаковывает элемент R ( в не увеличивающейся высоте ) на первом уровне , где R подходит. Если ни один уровень не может вместить R, создается новый уровень. Временная сложность FFDH O (n \* log n). Аппроксимация FFDH (I) <= (17/10) · OPT (I) +1; асимптотическая граница 17/10 является плотной.

NFDH упаковывает следующий элемент R ( в возрастающей высоте ) на текущий уровень, если R подходит. В противном случае текущий уровень закрыт и создается новый уровень. Сложность времени O (n \* log n). Отношения приближения NFDH (I) <= 2 · OPT (I) +1; асимптотическая оценка 2 плотна.

Из графика можно сделать вывод, что алгоритм FFDH эффективнее , чем NFDH. Алгоритм FFDH в равных условиях с NFDH выполнил задания быстрее , с увеличением количества входных данных время на выполнения задач растет пропорционально.

# **5. Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "../include/extralib.h"

#include "../include/color.h"

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

if (argc < 4) {

fprintf(stderr, "%s[ERROR]%s Not enough arguments!%s\n", RED, YELLOW, RESET);

exit(EXIT\_FAILURE);

}

char\* filename = argv[1];

char\* TDSP\_method = argv[2];

TT.cores = atoi(argv[3]);

TT.time = 0;

TT.occupancy = 0;

TT.curr\_core = 0;

int tasks\_amount = 0;

int \*list\_of\_tasks[2];

list\_of\_tasks[0] = (int\*)malloc(RANGE \* sizeof(int));

list\_of\_tasks[1] = (int\*)malloc(RANGE \* sizeof(int));

FILE\* fd\_in = fopen(filename, "r");

if (fd\_in) {

fscanf(fd\_in, "%d", &tasks\_amount);

if (tasks\_amount > RANGE) {

fprintf(stderr, "%s[ERROR]%s Amount of tasks is too large!%s\n", RED, YELLOW, RESET);

fclose(fd\_in);

exit(EXIT\_FAILURE);

}

for (int i = 0; i < tasks\_amount; ++i) {

fscanf(fd\_in, "%d", &list\_of\_tasks[0][i]);

fscanf(fd\_in, "%d", &list\_of\_tasks[1][i]);

}

fclose(fd\_in);

}

if (strcmp(TDSP\_method, "NFDH") == 0) {

printf("%s[SYSTEM]%s For %s%d%s tasks from %s%s%s and %s%d%s EM has been applied %s%s%s method...%s\n", YELLOW, WHITE, GREEN, tasks\_amount, WHITE, GREEN, filename, WHITE, GREEN, TT.cores, WHITE, GREEN, TDSP\_method, WHITE, RESET);

NFDH(list\_of\_tasks, tasks\_amount);

} else if (strcmp(TDSP\_method, "FFDH") == 0) {

printf("%s[SYSTEM]%s For %s%d%s tasks from %s%s%s and %s%d%s EM has been applied %s%s%s method...%s\n", YELLOW, WHITE, GREEN, tasks\_amount, WHITE, GREEN, filename, WHITE, GREEN, TT.cores, WHITE, GREEN, TDSP\_method, WHITE, RESET);

FFDH(list\_of\_tasks, tasks\_amount);

} else {

fprintf(stderr, "%s[ERROR]%s Unknown method for use!%s\n", RED, YELLOW, RESET);

exit(EXIT\_FAILURE);

}

return EXIT\_SUCCESS;

}

#include <iostream>

#include <fstream>

using namespace std;

void tasksGenerate(int n) {

int m = 500;

for (int i = 0; i < 10; i++) {

string str = "./result/task" + to\_string(i) + "n" + to\_string(n);

ofstream fout(str);

fout << n << " " << m << endl;

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

fout << 1 + rand() % 100 << " " << 1 + rand() % n << endl;

fout.close();

m += 500;

}

}

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

int n = 0;

cin >> n;

srand(time(0));

tasksGenerate(n);

return 0;

}

#include <iostream>

#include <fstream>

#include <utility>

#include <map>

#include <cmath>

#include <cstdlib>

#include <ctime>

using namespace std;

int n = 0; /\* Число ЭМ в системе \*/

int m = 0; /\* Число задач в наборе для обработки \*/

typedef struct TreeNode {

int gap\_size;/\* Вместимость уровня \*/

int start\_time; /\* Время запуска уровня \*/

int type;

struct TreeNode \*left;

struct TreeNode \*right;

struct TreeNode \*top;

} TreeNode;

typedef struct tasks {

int number, /\* Номер задачи \*/

rank, /\* Ранг задачи \*/

time, /\* Время выполнения задачи \*/

start\_time, /\* Время запуска задачи \*/

position; /\* Смещение размещения задачи на ЭМ \*/

struct tasks \*next; /\* Поле для сортировки подсчётом \*/

} tasks;

int NFDH\_sort(tasks \*root) {

int currentLevel = 0, nextLevel = 0, freeMachineNumber = 0;

tasks \*ptr = root;

currentLevel = 0;

freeMachineNumber = ptr->rank;

ptr->start\_time = currentLevel;

nextLevel = ptr->time;

ptr = ptr->next;

while (ptr->next != NULL)

{

if (ptr->rank + freeMachineNumber > n)

{

currentLevel = nextLevel;

nextLevel += ptr->time;

ptr->start\_time = currentLevel;

freeMachineNumber = ptr->rank;

} else {

ptr->start\_time = currentLevel;

freeMachineNumber += ptr->rank;

}

ptr = ptr->next;

}

return nextLevel;

}

TreeNode\* FFDH\_Create\_Node(TreeNode \*top, int i, int depth) {

TreeNode \*ptr = (TreeNode \*)malloc(sizeof(TreeNode));

if (i != 0)

ptr->top = top;

ptr->gap\_size = n;

if (i < depth) {

ptr->left = FFDH\_Create\_Node(ptr, i+1, depth);

ptr->right = FFDH\_Create\_Node(ptr, i+1, depth);

ptr->type = 0;

}

else {

ptr->start\_time = -1;

ptr->type = 1;

}

return ptr;

}

tasks\* createNode(int number, int rank, int time) {

tasks \*ptr = (tasks \*)malloc(sizeof(tasks));

ptr->number = number;

ptr->rank = rank;

ptr->time = time;

ptr->next = NULL;

return ptr;

}

TreeNode\* FFDH\_Find\_Node(TreeNode \*root, int gapsize, int depth) {

TreeNode \*ptr = root, \*b\_ptr;

do {

if (ptr->left->gap\_size >= gapsize)

ptr = ptr->left;

else

ptr = ptr->right;

} while (--depth);

ptr->gap\_size -= gapsize;

b\_ptr = ptr;

while (ptr != root) {

if (ptr->top->left->gap\_size >= ptr->top->right->gap\_size)

ptr->top->gap\_size = ptr->top->left->gap\_size;

else

ptr->top->gap\_size = ptr->top->right->gap\_size;

ptr = ptr->top;

}

return b\_ptr;

}

int FFDH\_Compute(tasks \*data)

{

int depth, time = 0;

TreeNode \*root, \*ptr;

tasks \*tptr = data;

depth = ceil(log(m) / log(2));

root = FFDH\_Create\_Node(NULL, 0, depth);

while (tptr->next != NULL) {

ptr = FFDH\_Find\_Node(root, tptr->rank, depth);

if (ptr->start\_time == -1) {

ptr->start\_time = time;

time += tptr->time;

}

tptr->start\_time = ptr->start\_time;

tptr->position = (n - ptr->gap\_size - tptr->rank);

tptr = tptr->next;

}

return time;

}

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

int a = 0, b = 0;

int TS = 0;

double Tsh = 0.0, e = 0;

string str;

if (argc < 2) {

cout << "Enter filename";

return 0;

}

ifstream fin(argv[1]);

fin >> n >> m;

int mass[n+1], mass2[n+1];

int myMap1[m], myMap2[m], mySortMap1[m], mySortMap2[m];

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

fin >> a >> b;

myMap1[i] = a;

myMap2[i] = b;

Tsh += a\*b;

}

Tsh /= n;

for (int i = 0; i < n+1; i++) {

mass[i] = 0;

mass2[i] = 0;

}

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

mass[myMap1[i]]++;

mass2[0] = mass[0];

for (int i = 1; i < n+1; i++)

mass2[i] = mass2[i-1] + mass[i];

mass[0] = 0;

for (int i = 1; i < n+1; i++)

mass[i] = mass2[i-1];

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

mySortMap1[mass[myMap1[i]]] = myMap1[i];

mySortMap2[mass[myMap1[i]]] = myMap2[i];

mass[myMap1[i]]++;

}

tasks \*root = createNode(0, mySortMap2[m-1], mySortMap1[m-1]);

tasks \*end = root;

for (int i = m-2, j = 1; i >= 0; i--, j++) {

end->next = createNode(j, mySortMap2[i], mySortMap1[i]);

end = end->next;

}

unsigned int startTime = clock();

TS = NFDH\_sort(root);

unsigned int endTime = clock() - startTime;

e = (TS - Tsh)/Tsh;

cout << m << " " << TS << endl;

startTime = clock();

TS = FFDH\_Compute(root);

endTime = clock() - startTime;

e = (TS - Tsh)/Tsh;

cout << m << " " << TS << endl;

fin.close();

return 0;

}