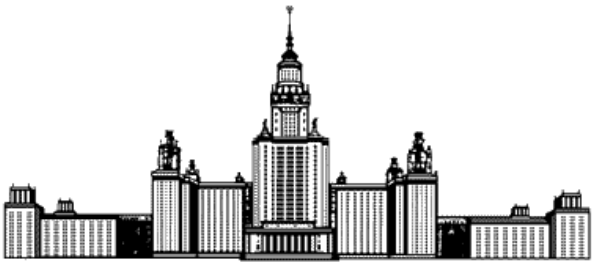

Московский государственный университет имени М.В. Ломоносова

Факультет Вычислительной математики и кибернетики

Задание 1. Расписание сети сортировки

Выполнил:

студент 524 группы

I курс магистратуры

Петухов Иван Андреевич

Дата подачи:

17/10/2017

Москва, 2017

Описание условия

Разработать последовательную программу вычисления расписания сети сортировки, числа, использованных компараторов и числа тактов, необходимых для её срабатывания при выполнении на n процессорах. Число тактов сортировки при параллельной обработке не должно превышать числа тактов, затрачиваемых четно-нечетной сортировкой Бетчера.

Параметр командной строки запуска: n.

n>=1 – количество элементов в упорядочиваемом массиве, элементы которого расположены на строках с номерами [0…n-1]

Формат команды запуска:
bsort n

Требуется:

1. вывести в файл стандартного вывода расписание и его характеристики в представленном далее формате;
2. обеспечить возможность вычисления сети сортировки для числа элементов 1<=n<=10000;
3. предусмотреть полную проверку правильности сети сортировки для значений числа сортируемых элементов 1<=n<=24;

Формат файла результата:

Начало файла результата

n 0 0
cu_0 cd_0
cu_1 cd_1
…
cu_{n_comp-1} cd_{n_comp-1}
n_comp
n_tact

Конец файла ре результата

Здесь:
n 0 0 – число сортируемых элементов, ноль, ноль.
cu_i cd_i – номера строк, соединяемых i-м компаратором сравнения перестановки.
n_comp – число компараторов
n_tact – число тактов сети сортировки

Описание метода решения

Сети Бэтчера – наиболее быстродействующие из масштабируемых сетей. Для построения сети использовался следующий рекурсивный алгоритм.

Для сортировки массива из p элементов с номерами [0, …, p - 1] нужно разделить его на 2 части. В первой части оставить [p/2] элементов с номерами [p/2 - 1]

При сортировке массива из p элементов с номерами ***[1, … , p]*** следует разделить его на две части: в первой оставить $\boldsymbol{n} = [\frac{\boldsymbol{p}}{\boldsymbol{2}}]$ элементов с номерами ***[1, …, n]***, а во второй ***m = p - n*** элементов с номерами ***[n + 1, …, p]***. Далее следует отсортировать каждую из частей (функция ***sort***) и объединить результаты сортировки (функция ***join***).

Рассмотрим данные функции подробнее.

sort – функция рекурсивного построения сети сортировки группы линий. Рекурсивно делит массив на два подмассива из ***n*** и ***m*** элементов соответственно, после чего вызывает функцию слияния ***join*** для этих подмассивов.

join – функция рекурсивного слияния двух групп линий. В сети нечетно-четного слияния отдельно объединяются элементы массивов с нечетными номерами и отдельно с четными, после чего с помощью заключительной группы компараторов обрабатываются пары соседних элементов с номерами ***(i, i + 1)***, где ***i*** – натуральные числа от ***1*** до ***p - 1***.

Описание метода проверки

Метод проверки основан на принципе нулей и единиц. Осуществляется перебор всевозможных перестановок из нулей и единиц в массиве заданной длины. Такая проверка осуществляется функцией *test*. Выполняется для заданного ***n*** от 1 до 24 (только для алгоритма сортировки, а не слияния).

Чтобы провести проверку, требуется раскомментировать вызов функции *test(n)* в *main()*. Результат проверки (true или false) выводится после результатов работы программы.

Приложение.

Исходный текст программы на C++.

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

struct comparator
{
	unsigned long int first;
	unsigned long int second;
	comparator *next;
};

int comparators = 0;
vector<int> tacts;
comparator *firstComp = NULL;
comparator *currentComp = NULL;

void push(int i1, int i2)
{
	cout << i1 << " " << i2 << endl;
	if (firstComp == NULL) {
		firstComp = new comparator;
		firstComp->first = i1;
		firstComp->second = i2;
		firstComp->next = NULL;
		currentComp = firstComp;
	}
	else {
		currentComp->next = new comparator;
		currentComp = currentComp->next;
		currentComp->first = i1;
		currentComp->second = i2;
		currentComp->next = NULL;
	}
	comparators++;
}

void join(int start1, int start2, int step, int n, int m)
{
	int tactCount0 = 0, tactCount1 = 0;
	int i, n1, m1;
	if (n * m < 1){
		return;
	}
	if (n == 1 && m == 1) {
		push(start1, start2);
		tacts[start1] = tacts[start2] = max(tacts[start1], tacts[start2]) + 1;
		return;
	}
	n1 = n - n / 2;
	m1 = m - m / 2;
	join(start1, start2, step * 2, n1, m1);
	join(start1 + step, start2 + step, step * 2, n - n1, m - m1);
	for (i = 1; i < n - 1; i += 2) {
		push(start1 + step * i, start1 + step * (i + 1));
	    tacts[start1 + step * i] = tacts[start1 + step * (i + 1)] = max(tacts[start1 +
step * i], tacts[start1 + step * (i + 1)]) + 1;
	}
```

```
	if (n % 2 == 0){
		push(start1 + step * (n - 1), start2);
		tacts[start1 + step * (n - 1)] = tacts[start2] = max(tacts[start1 + step *
(n - 1)], tacts[start2]) + 1;
	}
	for (i = (n % 2 == 0) ? 1 : 0; i < m - 1; i += 2) {
		push(start2 + step * i, start2 + step * (i + 1));
	    tacts[start2 + step * i] = tacts[start2 + step * (i + 1)] = max(tacts[start2 +
step * i], tacts[start2 + step * (i + 1)]) + 1;
	}
	return;
}

void sort(int start, int step, int n)
{
	int tactCount0, tactCount1, tactCountJoin;
	if (n < 2)
		return;
	int half = n / 2;
	sort(start, step, half);
	sort(start + step * half, step, n - half);
	join(start, start + step * half, step, half, n - half);
	return;
}

void test(int n){
	bool testArray[24];
	unsigned int mask;
	bool isCorrect = true;
	for (unsigned int i = 0; i < (int)pow(2.0, n); i++) {
		mask = 1;
		for (int j = n - 1; j >= 0 ; j--) {
			testArray[j] = i & mask > 0 ? true : false;
			mask = mask << 1;
		}
		currentComp = firstComp;
		while (currentComp != NULL)
		{
			if (testArray[currentComp->first] != testArray[currentComp->second]
&& testArray[currentComp->first]) {
				testArray[currentComp->first] = false;
				testArray[currentComp->second] = true;
			}
			currentComp = currentComp->next;
		}
		for (int j = 0; j < n - 1; j++) {
			if (testArray[j] != testArray[j + 1] && testArray[j]) {
				isCorrect = false;
				break;
			}
		}
		if (!isCorrect) break;
	}
	if (n <= 24) printf(isCorrect ? "true\n" : "false\n");
}

int main(int argc, char* argv[])
{
	int n = 0, start = 0, tactMax = 0, step = 1;
	if (argc >= 2) {
		n = atoi(argv[1]);
	} else {
		cout << "Not arguments" << endl;
		return -1;
	}
```

```
        tacts = vector<int>(n, 0);
        cout << n << " " << 0 << " " << 0 << endl;
        sort(start, step, n);
        vector<int>::iterator it = tacts.begin();
        tactMax = *it;
        for (; it < tacts.end(); it++){
               if(tactMax < *it)
                      tactMax = *it;
        }
        cout << comparators << endl << tactMax << endl;
        //test(n);
    return 0;
}
```