Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №4
по дисциплине "Анализ Алгоритмов"

Тема Параллельные вычисления

Студент Светличная А.А.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л., Строганов Ю.В.

Москва — 2022 г.

Оглавление

Введение

Одной из главных задач программирования является ускорение решения вычислительных задач, которая может быть решена использованием параллельных вычислений.

В последовательном алгоритме решения какой-либо задачи есть операции, которые может выполнять только один процесс, например, операции ввода и вывода. Кроме того, в алгоритме могут быть операции, которые могут выполняться параллельно разными процессами. Способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой, называют **многопоточностью** [1].

Процессом является программа в ходе своего выполнения. Каждый процесс состоит из одного или нескольких потоков - исполняемых сущностей, которые выполняют задачи, стоящие перед исполняемым приложением. После окончания выполнения всех потоков завершается процесс.

Современные процессоры могут выполнять две задачи на одном ядре при помощи дополнительного виртуального ядра. Такие процессоры называются многоядерными. Каждое ядро может выполнять только один поток за единицу времени. Если потоки выполняются последовательно, то их выполняет только одно ядро процессора, другие ядра не задействуются. Если независимые вычислительные задачи будут выполняться несколькими потоками параллельно, то будет задействовано несколько ядер процессора и решение задач ускорится.

1 Аналитическая часть

1.1 Цель и задачи

Целью данной лабораторной работы является изучение многопоточности на основе алгоритма Дейкстры (поиск кратчайших расстояний от одной вершины до всех остальных).

Для достижения поставленной цели требуется выполнить следующие задачи:

- изучить понятие параллельных вычилений;
- изучить последовательный и параллельный варианты алгоритма поиска кратчайших расстояний Дейкстры;
- реализовать изученные алгоритмы;
- провести сравнительный анализ по времени реализованных алгоритма;
- подготовить отчет о выполненной лабораторной работе.

1.2 Алгоритм Дейкстры

Алгоритм Дейкстры — это последовательность действий, позволяющих оптимально найти на графе кратчайший путь от некоторой его вершины до всех остальных [2].

Основная идея данного алгоритма заключается в том, что на каждом шаге помечается определённым образом выбранная вершина, а далее просматриваются все последующие (ещё не отмеченные) вершины графа и вычисляется длина пути до каждой из них от начальной точки. Помеченная на каждом этапе новая вершина (та, до которой путь оказался кратчайшим) становится определяющей для следующего шага. И этот построенный до неё кратчайший путь закрашивается. Повторяя данные действия, последовательно просматривая и поочерёдно отмечая вершины графа, в

итоге алгоритм добирается до конечного пункта. Те рёбра графа, которые в результате оказались закрашенными (вместе с отмеченными их крайними точками), образуют ориентированное дерево кратчайших путей (с корнем в исходной вершине графа). Далее остаётся выбрать на нём путь, который соединяет начальную точку с конечной. Это и будет искомый путь.

Вывод

В данном разделе был изучен и рассмотрен алгоритм Дейкстры, хорошо подхлдящий для параллельных вычислений за счет того, что относительно каждого узла расматривает всех соседей независимо друг от друга.

2 Конструкторская часть

Описания алгоритмов

На рисунке 2.1 представлены схема алгоритмов Дейкстры.

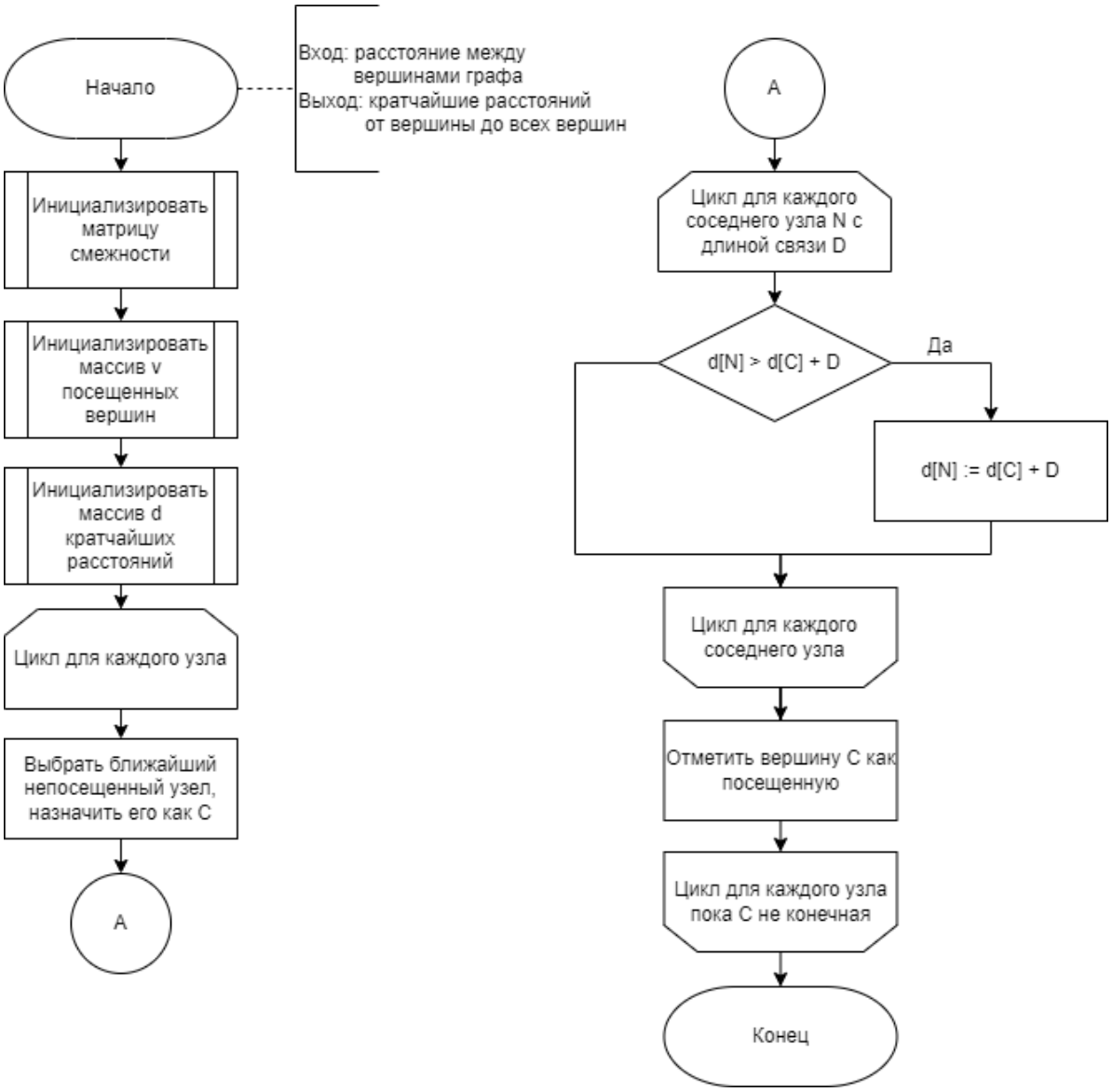

Рисунок 2.1 – Схема алгоритма Дейкстры

Вывод

В данном разделе на основе теоретических данных были построены схема требуемого алгоритма.

3 Технологическая часть

3.1 Требования к программному обеспечению

В программе должна быть возможность:

1) генирировать входные данные;
2) находить кратчайшие расстояния от вершины одинм из указанных алгоритмов;
3) замерять процессорное время выполнения реализаций алгоритмов.

3.2 Выбор языка программирования

Для реализации алгоритмов поиска редакционных расстояний был выбран язык программирования C в силу наличия точных библиотек и быстродействености языка.

3.3 Выбор библиотеки и способа для замера времени

Для замера процессорного времени выполнения реализаций агоритмов была выбрана не стандартная функция библиотеки <time.h> языка C — clock(), которая недостаточно четко работает при замерах небольших промежутков времени, а QueryPerformanceCounter - API-интерфейс, использующийся для получения меток времени с высоким разрешением или измерения интервалов времени.

Для облегчения работы с данным инструментом были самостоятельно написаны обертки-функции, представленные на листинге 3.1.

Листинг 3.1 – Функции замеров процессорного времени

```
1     LARGE_INTEGER frequency;
2     LARGE_INTEGER t1,t2;
3     double elapsedTime;
4
5     void TIMER_INIT()
6     {
7         QueryPerformanceFrequency(&frequency);
8     }
9
10    void TIMER_START()
11    {
12      QueryPerformanceCounter(&t1);
13    }
14
15    double TIMER_STOP()
16    {
17        QueryPerformanceCounter(&t2);
18        elapsedTime=(float)(t2.QuadPart-t1.QuadPart)/frequency.
              QuadPart/COUNT*MICRO;
19
20      return elapsedTime;
21    }
```

В силу существования явления вытеснения процессов из ядра, квантования процессорного времени все процессорное время не отдается какой-либо одной задаче, поэтому для получения точных результатов необходимо усреднить результаты вычислений: замерить совокупное время выполнения реализации алгоритма N раз и вычислить среднее время выполнения.

3.4 Реализации алгоритмов

В листингах 3.2, 3.3, приведены реализации алгоритмов Дейкстры без использования параллельных вычислений и с использованием соответственно.

Листинг 3.2 – Алгоритм Дейкстры

```
1     void dijkstra(int a[SIZE][SIZE], int *d, int *v)
2     {
3         int temp, minindex, min;
4         int begin_index = 0;
5
6         d[begin_index] = 0;
7
8         do {
9             minindex = min = INT_MAX;
10            for (int i = 0; i < SIZE; i++)
11            {
12                if ((v[i] == 1) && (d[i] < min))
13                {
14                    min = d[i];
15                    minindex = i;
16                }
17            }
18
19            if (minindex != INT_MAX)
20            {
21                for (int i = 0; i < SIZE; i++)
22                if (a[minindex][i] > 0)
23                    d[i] = get_min(2, d[i], min + a[minindex][i]);
24                v[minindex] = 0;
25            }
26        } while (minindex < INT_MAX);
27    }
```

Листинг 3.3 – Алгоритм Дейкстры с использованием параллельных вычислений

```
1     typedef struct{
2         int (*a)[SIZE];
3         int* d;
4         int min;
5         int minindex;
6         int i;
7     } pthrData;
8
9
10    void* threadFunc(void *thread_data)
11    {
12        pthrData *data = (pthrData*) thread_data;
13
14        if (data->a[data->minindex][data->i] > 0)
15            data->d[data->i] = get_min(2, data->d[data->i],
16            data->min + data->a[data->minindex][data->i]);
17
18        return NULL;
19    }
20
21    void dijkstra_multi(int a[SIZE][SIZE], int *d, int *v)
22    {
23        pthread_t* threads = (pthread_t*) malloc(COUNT_THR *
              sizeof(pthread_t));
24        pthrData* threadData = (pthrData*) malloc(COUNT_THR *
              sizeof(pthrData));
25
26
27        int temp, minindex, min;
28        int begin_index = 0;
29
30        d[begin_index] = 0;
31
32        do {
33            minindex = min = INT_MAX;
34            for (int i = 0; i < SIZE; i++)
35            {
36                if ((v[i] == 1) && (d[i] < min))
37                {
```

```
38                      min = d[i];
39                      minindex = i;
40                  }
41              }
42
43              if (minindex != INT_MAX)
44              {
45                  for (int i = 0; i < SIZE; i += COUNT_THR)
46                  {
47                      for (int j = 0; j < COUNT_THR; j++)
48                      {
49                          threadData[j].a = a;
50                          threadData[j].d = d;
51                          threadData[j].min = min;
52                          threadData[j].minindex = minindex;
53                          threadData[j].i = i + j;
54
55                          pthread_create(&(threads[j]), NULL,
                               threadFunc, &threadData[j]);
56                      }
57                      for(int j = 0; j < COUNT_THR; j++)
58                          pthread_join(threads[j], NULL);
59                  }
60                  v[minindex] = 0;
61              }
62          } while (minindex < INT_MAX);
63      }
```

3.5 Тестирование алгоритмов

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритм Дейкстры. Все тесты пройдены успешно как алгоритмом без использования параллельных вычислений, так и алгоритмом с использованием.

Таблица 3.1 – Функциональные тесты

Матрица A	Ожидаемый результат
$\begin{pmatrix}0\end{pmatrix}$	0
$\begin{pmatrix}0 & 37\\37 & 0\end{pmatrix}$	0, 37
$\begin{pmatrix}0 & 59 & 13 & 14 & 85\\59 & 0 & 98 & 25 & 80\\13 & 98 & 0 & 8 & 92\\14 & 25 & 8 & 0 & 99\\85 & 80 & 92 & 99 & 0\end{pmatrix}$	0, 39, 13, 14, 85

Вывод

В данном разделе были реализованы последовательный и параллельный алгоритма Дейкстры, а также проведено функциональное тестирование указанных алгоритмов.

4 Экспериментальная часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование программного обеспечения:

1) операционная система Windows-10, 64-bit;
2) оперативная память 8 ГБ;
3) процессор Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz, 2304 МГц, ядер 2, логических процессоров 4.

4.2 Замеры времени

В таблице 4.1 приведены результаты замеров в микросекундах времени работы алгоритмов для входных графоф с разным количеством узлов.

Таблица 4.1 – Замеры времени выполнения алгоритмов на произвольных массивах разной длины

Кол-во узлов	10	20	50
1	2.98	8.67	48.36
2	15475.22	59728.75	315804.90
3	16409.54	58712.59	337089.45
4	15970.94	51750.29	337464.52
5	12853.70	51316.59	338854.04

Из таблицы 4.1 видно, что последовательный алгоритм работает намного быстрее, по этой причине стоит строить графики алгоритмов отдельно.

Зависимость времени работы последовательного алгоритма Дейкстры от количества узлов графа представлена на рисунке 4.2.

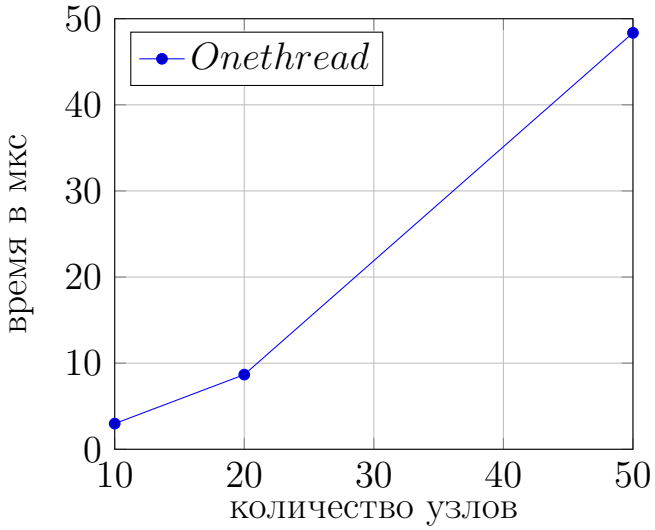

Рисунок 4.1 – Зависимость времени работы последовательного алгоритма Дейкстры от количества узлов графа

Из таблицы 4.1 видно, что многопоточный алгоритм с разным количеством потоков работает за приблизительно одинаковое время, по этой причине строить все графики потоков нецелесообразно.

Зависимость времени работы многопоточного алгоритма Дейкстры от количества узлов графа представлена на рисунке 4.2.

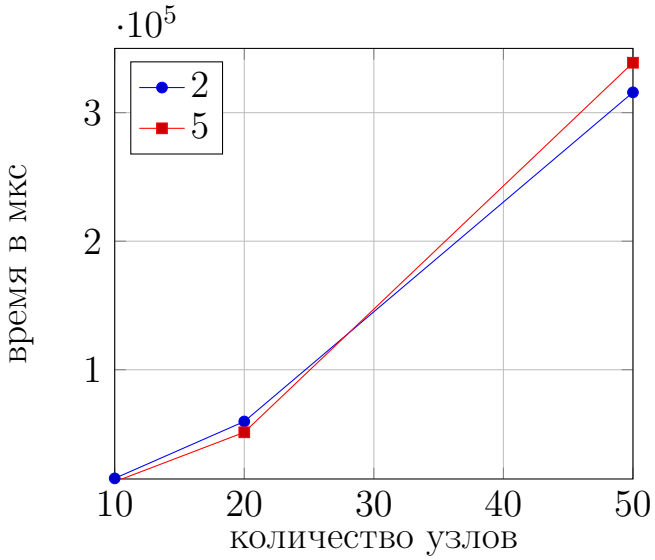

Рисунок 4.2 – Зависимость времени работы многопоточного алгоритма Дейкстры от количества узлов графа

Вывод

В данном разделе былпа проведена оценка эффективности реализаций последовательного и многопоточного алгоритма Дейкстры, которая показала, что последовательный алгоритм выигрывает в скорости, это происходит по причине того, что выбранная для распараллеливания операция является слишком маленькой относительно всего алгоритма, а для выполнения каждой такой операции в многопоточном режиме необходимо сохранить большое количество данных, что и увеличивает время работы данной реализации.

Заключение

Цель данной лабораторной работы достигнута — изучена многопоточность на основе алгоритма Дейкстры.

Все **задачи** выполнены:

- изучено понятие параллельных вычилений;
- изучены последовательный и параллельный варианты алгоритма поиска кратчайших расстояний Дейкстры;
- реализованы изученные алгоритмы;
- проведен сравнительный анализ по времени реализованных алгоритма;
- подготовлен отчет о выполненной лабораторной работе.

На основании проведенных экспериментов было определено, что быстродействие работы последователього алгоритма Дейкстры много больше параллельного варианта данного алгоритма, что происходит по причине незначительности параллельной операции в соотношении с необходимыми для многопоточности действиями.

Список использованных источников

[1] Гладышев Е.И., Мурыгин А.В. Многопоточность в приложениях // Актуальные проблемы авиации и космонавтики. 2012. №8. URL: https://cyberleninka.ru/article/n/mnogopotochnost-v-prilozheniyah (дата обращения: 06.12.2022).

[2] Лебедев Сергей Сергеевич, Новиков Федор Александрович Необходимое и достаточное условие применимости алгоритма Дейкстры // КИО. 2017. №4. URL: https://cyberleninka.ru/article/n/neobhodimoe-i-dostatochnoe-uslovie-primenimosti-algoritma-deykstry (дата обращения: 06.12.2022).