Министерство образования и науки Российской Федерации

Государственное образовательное учреждение высшего профессионального образования «Нижегородский государственный университет   
им. Н.И. Лобачевского»

**Институт информационных технологий, математики и механики**

**Кафедра: программная инженерия**

Специальность (направление): Программная инженерия

**Отчет**

по самостоятельной работе

по дисциплине «Параллельное программирование»

тема:

**«Поиск кратчайших путей из одной вершины (алгоритм Мура)»**

**Выполнил:** студент группы 381508

Яшков Владислав

Нижний Новгород  
2018

Оглавление

[**Введение** 3](#_Toc514689437)

[**Постановка задачи** 4](#_Toc514689438)

[**Вспомогательные элементы** 5](#_Toc514689439)

[Typer 5](#_Toc514689440)

[Viewer 5](#_Toc514689441)

[Generator 5](#_Toc514689442)

[Checker 5](#_Toc514689443)

[**Метод решения** 6](#_Toc514689444)

[Последовательный алгоритм 6](#_Toc514689445)

[Параллельный алгоритм 7](#_Toc514689446)

[**Результаты экспериментов** 8](#_Toc514689447)

[**Вывод** 9](#_Toc514689448)

# **Введение**

Стандартная математическая задача поиска кратчайших путей из одной вершины формируется таким образом. Пусть дан ориентированный взвешенный граф G с n вершинами и m рёбрами, и указана некоторая вершина v. Требуется найти **длины кратчайших путей** от вершины v до всех остальных вершин.

В данной работе для решения поставленной задачи применяется алгоритм Мура. Он был независимо предложен Р.Беллманом и Э.Муром, в силу чего некоторые авторы называют его алгоритмом Беллмана-Мура-Форда.

В отличие от алгоритма Дейкстры, этот алгоритм применим также и к графам, содержащим рёбра отрицательного веса. Впрочем, если граф содержит отрицательный цикл, то, понятно, кратчайшего пути до некоторых вершин может не существовать (по причине того, что вес кратчайшего пути должен быть равен минус бесконечности); впрочем, этот алгоритм можно модифицировать, чтобы он сигнализировал о наличии цикла отрицательного веса, или даже выводил сам этот цикл.

Постановка задачи

В данной работе необходимо реализовать последовательный алгоритм и параллельные алгоритмы с использованием библиотек OpenMP и TBB, позволяющие найти кратчайшие пути из одной вершины в графе.

**Для выполнения задачи необходимо**:

* Разработать тестовую версию, включающую в себя:
  + solver - решение задачи последовательным алгоритмом, по возможности, максимально простым;
  + generator - программа для генерации набора тестовых данных и их сохранения в бинарные файлы;
  + checker - программа для проверки корректности параллельных версий;
  + набор тестов (не менее десяти, из которых не менее пяти "большие") для проверки корректности параллельных версий.
* Разработать OpenMP версию
* Разработать TBB версию

Вспомогательные элементы

### **Typer**

Вспомогательная программа, служит для перевода текстового файла, содержащего массив, в бинарный

### **Viewer**

Вспомогательная программа, служит для перевода бинарного файла, содержащего массив, в текстовый

### **Generator**

Генератор тестов. Создает два бинарных файла:

1. с исходными данными графа
2. с набором кратчайших путей к каждой вершины

Возможные входные параметры:

<номер теста (1-20)> <имя бинарного файла c данными графа> <имя бинарного файла с ответами>

Количество вершин в графе зависит от номера теста.

Генерация графа начинается с центральной нулевой вершины, для которой и необходимо выполнить поиск кратчайших путей по условию задачи. Для неё сразу задаётся кратчайший путь, равный нулю. Затем с помощью двойного цикла между всеми вершинами графа случайным образом генерируются рёбра и их веса, рассчитываются кратчайшие пути.

### **Checker**

Проверяет корректность работы алгоритма, сравнивая результаты работы алгоритма на тестовых данных с готовыми ответами и выдает вердикт:

* AC = Accepted = Решение выдаёт корректный результат на данном тесте
* WA = Wrong Answer = Решение выдаёт некорректный результат на данном тесте
* PE = Presentation Error = Ошибка формата выходных данных

Метод решения

Мы считаем, что граф не содержит цикла отрицательного веса. Заведём массив расстояний d[0..n-1], который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: d[0]=0, а все остальные элементы d[] равны бесконечности infty.

Сам алгоритм представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести **релаксацию** (relax, ослабление) вдоль каждого ребра (a, b) стоимости c. Релаксация вдоль ребра — это попытка улучшить значение d[b] значением d[a]+c. Фактически это значит, что мы пытаемся улучшить ответ для вершины b, пользуясь ребром (a, b) и текущим ответом для вершины a.

Утверждается, что достаточно n-1 фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (повторимся, мы считаем, что циклы отрицательного веса отсутствуют). Для недостижимых вершин расстояние d[] останется равным бесконечности infty.

Для алгоритма Форда-Беллмана, в отличие от многих других графовых алгоритмов, более удобно представлять граф в виде одного списка всех рёбер (а не n списков рёбер — рёбер из каждой вершины). В приведённой реализации заводится структура данных edge для ребра. Входными данными для алгоритма являются квадратная матрица графа размером n, где каждый элемент матрицы является весом ребра; массив вершин, для каждой из которых нужно указать длину её кратчайшего пути и массив вершин, составляющих этот путь; число вершин n. Все номера вершин нумеруются с 0 по n-1.

Константа INF обозначает число "бесконечность" — её надо подобрать таким образом, чтобы она заведомо превосходила все возможные длины путей.

Зачастую ответ находится уже за несколько фаз, а оставшиеся фазы никакой полезной работы не происходит, лишь впустую просматриваются все рёбра. Поэтому будем хранить флаг того, изменилось что-то на текущей фазе или нет, и если на какой-то фазе ничего не произошло, то алгоритм можно останавливать. (Эта оптимизация не улучшает асимптотику, т.е. на некоторых графах по-прежнему будут нужны все n-1 фаза, но значительно ускоряет поведение алгоритма "в среднем", т.е. на случайных графах.)

Для восстановления путей заведём ещё один массив p[0…n-1], в котором для каждой вершины будем хранить её "предка", т.е. предпоследнюю вершину в кратчайшем пути, ведущем в неё. В самом деле, кратчайший путь до какой-то вершины a является кратчайшим путём до какой-то вершины p[a], к которому приписали в конец вершину a.

Заметим, что алгоритм Форда-Беллмана работает по такой же логике: он, предполагая, что кратчайшее расстояние до одной вершины уже посчитано, пытается улучшить кратчайшее расстояние до другой вершины. Следовательно, в момент улучшения нам надо просто запоминать в p[a], из какой вершины это улучшение произошло.

|  |  |  |
| --- | --- | --- |
| void solve(int \*\* graph, Vertex \* vertexes, int n) { | | |
|  | const int INF = 1000000000; |
|  | vector<int> d(n, INF); |
|  | d[0] = 0; |
|  | vector<edge> e; |
|  | for (int i = 0; i < n-1; i++) { |
|  | for (int j = i + 1; j < n; j++) { |
|  | if (graph[i][j] != -1 && i != j) { |
|  | edge \_e; |
|  | \_e.a = i; |
|  | \_e.b = j; |
|  | \_e.cost = graph[i][j]; |
|  | e.push\_back(\_e); |
|  | } |
|  | } |
|  | } |
|  | int m = e.size(); |
|  | vector<int> p(n, -1); |
|  | for (;;) { |
|  | bool any = false; |
|  | for (int j = 0; j < m; ++j) { |
|  | if (d[e[j].a] < INF) |
|  | if (d[e[j].b] > d[e[j].a] + e[j].cost) { |
|  | d[e[j].b] = d[e[j].a] + e[j].cost; |
|  | p[e[j].b] = e[j].a; |
|  | any = true; |
|  | } |
|  | if (d[e[j].b] < INF) |
|  | if (d[e[j].a] > d[e[j].b] + e[j].cost) { |
|  | d[e[j].a] = d[e[j].b] + e[j].cost; |
|  | p[e[j].a] = e[j].b; |
|  | any = true; |
|  | } |
|  | } |
|  | if (!any) break; |
|  | } |
|  | for (int i = 0; i < n; i++) { |
|  | vector<int> path; |
|  | for (int cur = i; cur != -1; cur = p[cur]) |
|  | path.push\_back(cur); |
|  | reverse(path.begin(), path.end()); |
|  | vertexes[i].min\_path\_length = d[i]; |
|  | vertexes[i].min\_path\_vertex\_size = path.size() - 1; |
|  | if (vertexes[i].min\_path\_vertex\_size > 0) { |
|  | vertexes[i].min\_path = new int[vertexes[i].min\_path\_vertex\_size]; |
|  | for (int j = 0; j < path.size() - 1; j++){ |
|  | vertexes[i].min\_path[j] = path[j]; |
|  | } |
|  | } |
|  | } |
|  | } |

### **Последовательный алгоритм**

Алгоритм последовательно просматривает все рёбра графа и пытается произвести **релаксацию** (relax, ослабление) вдоль каждого ребра (a, b) стоимости c до тех пор, пока не будут найдены кратчайшие пути до всех вершин. Релаксация вдоль ребра — это попытка улучшить значение d[b] значением d[a]+c. Фактически это значит, что мы пытаемся улучшить ответ для вершины b, пользуясь ребром (a, b) и текущим ответом для вершины a.

Для недостижимых вершин расстояние d[] останется равным бесконечности.

Затем происходит восстановление кратчайших путей последовательно для каждой вершины.

### **Параллельный алгоритм**

Исходный и результирующий массивы являются общими для всех потоков. Каждый поток обрабатывает свою часть массива, используя общие исходные данные и записывает результат в общий результирующий массив. Это необходимо, потому что для вычисления результата нужны значения соседних вершин, не все из которых являются соседними в массиве.

Результаты экспериментов

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 поток, сек. | Библиотека | Параллельный алгоритм | | | | | | | |
| 2 потока | | 4 потока | | 8 потоков | | 16 потоков | |
| T, сек | ускорение | T, сек | ускорение | T, сек | ускорение | T, сек | ускорение |
| 40.9129 | OpenMP | 32.6802 | 1.2519 | 32.5884 | 1.2554 | 24.2138 | 1.6896 | 24.1284 | 1.6956 |
|  | TBB | 32.5912 | 1.2553 | 32.5901 | 1.2553 | 24.169 | 1.6927 | 24.169 | 1.6927 |

Для вычисления ускорения параллельных алгоритмов был использован тестовый граф из теста номер 20 с 3000 вершин. Из полученных данных можно заключить, что параллельные алгоритмы работают гораздо быстрее последовательной версии. Наилучшее время работы достигается при распараллеливании на 16-ти потоках с помощью TBB.

Вывод

В результате работы была написана программа, в которой была реализована как последовательная, так и параллельная версия поиска кратчайших путей в графе из одной вершины.

В результате замера времени работы различных версий алгоритма было выяснено, что параллельные версии с использованием библиотек OpenMP и TBB работают быстрее последовательной версии во всех случаях.